TITAN KIT

ESP32 learning kit with Wi-Fi and a robot car
Hands-on project manual

120 PROJECTS · BUILD · LEARN · INNOVATE

A Geekobot exclusive · Beginner to advanced · No soldering · geekobot.in

What is in the box

- ESP32 38-pin board
- ESP32 expansion board
- Micro-USB cable
- 830-point breadboard
- Jumper wires x120
- Red LEDs x10
- Green LEDs x10
- Yellow LEDs x10
- Blue LEDs x10
- RGB LEDs x10
- 220 ohm resistors x20
- 10k resistors x10
- 1k resistors x10
- 2k resistors x10
- Push buttons x4
- TTP223 touch modules x4
- Active buzzer
- Passive buzzer
- Sound sensor
- 1-digit 7-segment
- MAX7219 8x8 matrix
- 16x2 I2C LCD
- 0.96 inch OLED
- DHT11 sensor
- BMP280 pressure sensor
- 10k potentiometer
- LDR
- Rain sensor
- Soil moisture sensor
- 1-channel relay
- Pump with tube
- HC-SR04
- RC522 RFID, card and fob
- SG90 servo
- 4x4 keypad
- HC-05 Bluetooth
- MQ-2 gas sensor
- IR obstacle sensor
- MPU6050 motion sensor
- 28BYJ-48 stepper + ULN2003
- WS2812 8-LED ring
- IR receiver and remote
- 2WD robot chassis
- L298N motor driver
- 2 x 18650 holder
- 18650 cells x2

Seventeen levels

Level	Projects	What you do
Level 1 · Foundations	1-10	LEDs, buttons, touch, RGB colour, buzzers and a clap switch.
Level 2 · Displays and games	11-14	A 7-segment dice, an LED matrix and a countdown game.
Level 3 · LCD and environment	15-17	Text, temperature and humidity on a 16x2 LCD.
Level 4 · Light and sensing	18-21	Mood lamp, street light, rain and soil sensing.
Level 5 · Smart automation	22-24	Plant watering, tank filling and a temperature alarm.
Level 6 · Security and access	25-30	RFID, keypad, Bluetooth and touch locks, motion and smoke alarms.
Level 7 · Binary and code	31-35	Binary clocks and counters, hexadecimal and Morse code.
Level 8 · OLED graphics	36-40	Clock faces, a compass, gauges and animations on the OLED.
Level 9 · Environmental science	41-50	Heat index, dew point, comfort scores, trends and a greenhouse monitor.
Level 10 · Sound analysis	51-60	Sound level meters, noise monitors and sound-reactive light.
Level 11 · Motion and IMU	61-70	Spirit level, protractor, shake and fall detection with the MPU6050.
Level 12 · RFID systems	71-80	Library, attendance, checkout and entry systems with RFID cards.
Level 13 · Bluetooth	81-90	Phone dashboards, data logging and remote control over Bluetooth.

Level 14 · Distance	91-100	Tank level, height, speed and parking sensors with the HC-SR04.
Level 15 · Actuators	101-105	A servo meter and pointer, and a stepper conveyor with position memory.
Level 16 · Wi-Fi, IoT and new sensors	106-112	Wi-Fi web pages, an internet clock, the LED ring, IR remote and BMP280.
Level 17 · Robot car	113-120	The robot car: Wi-Fi, Bluetooth and IR driving, obstacle avoiding, follow-me and gyro steering.

The upload routine (every project)

1. Install the free Arduino IDE 2 from arduino.cc.
2. Add ESP32 support: File > Preferences > Additional boards manager URLs, add Espressif's package URL, then Tools > Board > Boards Manager and install esp32 by Espressif (3.0 or newer).
3. Seat the ESP32 on the expansion board and connect it with the micro-USB cable.
4. Build the circuit from the project's wiring table, with the USB cable unplugged.
5. Paste the code into a new sketch.
6. Tools > Board > ESP32 Arduino > ESP32 Dev Module, then Tools > Port and pick the COM port that appears (install the CP210x or CH340 driver if none does).
7. Click Upload. If it waits at "Connecting...", hold the BOOT button on the board until the upload starts.
8. Libraries: Tools > Manage Libraries, when the project lists one.

Safety and good practice

- Always use the stated resistor with every LED.
- The ESP32 is a 3.3V board: no GPIO may see more than 3.3V. The HC-SR04 ECHO and MQ-2 AO lines go through a 1k/2k divider.
- Never drive a motor, the pump or the stepper from a GPIO or the ESP32's own supply: use the L298N, the relay, the ULN2003 and the battery pack.
- GPIO34, 35, 36 and 39 are input-only; never use GPIO6 to GPIO11 (flash). GPIO12 must not be pulled high while the board starts.
- The RC522 RFID reader runs on 3.3V only. 5V destroys it.
- Relays: switch only low-voltage demo loads. Never wire mains in a classroom project.
- Check VCC, GND and signal before powering up; reversed power can damage a module.
- Unplug USB before rewiring.
- 18650 cells: insert them the right way round, never short the holder wires, switch the holder off after use.
- ESP32 expansion board: each GPIO has its own S (signal), V and G (ground) header, so a module plugs into one column with female jumpers.

Every project also has its own page with the same wiring and code, plus troubleshooting: geekobot.in/projects/best/geekobot-titan-kit-projects

CONTENTS

The 120 projects

#	Project	Level	Time
001	Blink One LED	Beginner	20 minutes
002	Alternate Two LEDs	Beginner	20 minutes
003	Traffic Signal System	Beginner	20 minutes
004	Push Button LED	Beginner	20 minutes
005	Touch Sensor Lamp	Beginner	30 minutes
006	RGB Color Lamp	Beginner	20 minutes
007	Emergency Light Effect	Beginner	20 minutes
008	Buzzer Beep Sound	Beginner	20 minutes
009	Musical Buzzer	Beginner	20 minutes
010	Clap Detection	Beginner	30 minutes
011	Digital Dice	Beginner	45 minutes
012	Scrolling Text Display	Beginner	30 minutes
013	Emoji Display on LED Matrix	Beginner	30 minutes
014	Countdown Timer Game	Beginner	30 minutes
015	LCD Welcome System	Beginner	20 minutes
016	LCD Temperature Display	Beginner	30 minutes
017	LCD Humidity Monitor	Beginner	30 minutes
018	RGB Mood Lamp	Beginner	30 minutes
019	Smart Street Light	Beginner	30 minutes
020	Smart Umbrella Reminder	Beginner	30 minutes
021	Soil Moisture Detector	Beginner	30 minutes
022	Automatic Plant Watering System	Intermediate	1 hour
023	Water Tank Automation	Intermediate	1 hour
024	Smart Temperature Alarm	Intermediate	45 minutes
025	RFID Door Lock	Intermediate	45 minutes
026	Password Door Lock	Intermediate	45 minutes
027	Bluetooth Door Lock	Intermediate	30 minutes
028	Motion Detection Alarm	Intermediate	30 minutes
029	Touch Password System	Intermediate	45 minutes
030	Smoke Detection Alarm	Intermediate	30 minutes
031	Binary Clock using LEDs	Intermediate	45 minutes
032	Binary Number Converter Display	Intermediate	45 minutes
033	Hexadecimal Counter Display	Intermediate	45 minutes
034	Morse Code Message Generator	Intermediate	20 minutes

035	Morse Code Decoder	Intermediate	30 minutes
036	OLED Analog Clock	Intermediate	30 minutes
037	OLED Compass Simulation	Intermediate	30 minutes
038	OLED Battery Level Indicator	Intermediate	30 minutes
039	OLED Sensor Dashboard	Advanced	30 minutes
040	OLED Animated Loading Screen	Intermediate	30 minutes
041	Heat Index Calculator	Intermediate	30 minutes
042	Dew Point Calculator	Intermediate	30 minutes
043	Weather Comfort Indicator	Intermediate	45 minutes
044	Temperature Trend Tracker	Intermediate	30 minutes
045	Humidity Trend Tracker	Intermediate	30 minutes
046	Mini Weather Logger	Advanced	30 minutes
047	Thermal Warning Indicator	Intermediate	45 minutes
048	Greenhouse Environment Monitor	Intermediate	45 minutes
049	Room Climate Score System	Advanced	30 minutes
050	Temperature-Based Color Visualizer	Intermediate	45 minutes
051	Sound Level Bar Graph	Intermediate	45 minutes
052	Noise Pollution Monitor	Intermediate	45 minutes
053	Voice Activity Detector	Intermediate	45 minutes
054	Sound Reactive RGB Lighting	Intermediate	30 minutes
055	Sound Frequency Visualizer	Intermediate	45 minutes
056	Smart Silence Detector	Intermediate	45 minutes
057	Meeting Room Noise Indicator	Intermediate	45 minutes
058	Audio Peak Detector	Intermediate	45 minutes
059	Sound Triggered Counter	Intermediate	45 minutes
060	Noise History Logger	Advanced	45 minutes
061	Digital Spirit Level	Intermediate	45 minutes
062	Tilt Angle Measurement System	Intermediate	45 minutes
063	Gesture Controlled LED Patterns	Intermediate	45 minutes
064	Motion Intensity Meter	Intermediate	45 minutes
065	Shake Detection System	Intermediate	45 minutes
066	Fall Detection Prototype	Advanced	45 minutes
067	Orientation Indicator	Intermediate	30 minutes
068	Digital Protractor	Intermediate	45 minutes
069	Motion Based Password	Intermediate	45 minutes
070	Gesture Controlled Menu Navigation	Advanced	45 minutes
071	RFID Based Library Management Demo	Intermediate	1 hour
072	RFID Product Tracking System	Intermediate	45 minutes

073	RFID Personal Profile Viewer	Intermediate	45 minutes
074	RFID Equipment Checkout System	Intermediate	1 hour
075	RFID Access Log Recorder	Intermediate	45 minutes
076	RFID Attendance Analytics	Advanced	45 minutes
077	RFID Multi-User Authentication	Advanced	1 hour
078	RFID Membership Verification	Intermediate	1 hour
079	RFID Based Task Assignment Demo	Intermediate	45 minutes
080	RFID Event Entry System	Advanced	1 hour
081	Bluetooth Sensor Monitoring Station	Intermediate	45 minutes
082	Bluetooth Data Logger	Intermediate	45 minutes
083	Bluetooth Wireless Terminal	Intermediate	45 minutes
084	Smartphone Controlled Servo Positioning	Intermediate	45 minutes
085	Bluetooth Environmental Dashboard	Advanced	45 minutes
086	Bluetooth Notification Display	Intermediate	45 minutes
087	Bluetooth Digital Name Board	Intermediate	30 minutes
088	Bluetooth Wireless Counter Reset	Intermediate	45 minutes
089	Bluetooth Home Appliance Simulator	Intermediate	45 minutes
090	Bluetooth Remote Configuration System	Advanced	45 minutes
091	Water Tank Level Monitor	Intermediate	45 minutes
092	Trash Bin Fill Level Indicator	Intermediate	45 minutes
093	Digital Height Measurement System	Intermediate	45 minutes
094	Queue Distance Monitor	Intermediate	45 minutes
095	Contactless Distance Switch	Intermediate	45 minutes
096	Safe Following Distance Indicator	Intermediate	45 minutes
097	Object Speed Estimator	Intermediate	45 minutes
098	Distance Based RGB Visualization	Intermediate	45 minutes
099	Smart Measuring Tape	Intermediate	45 minutes
100	Parking Slot Occupancy Detector	Intermediate	45 minutes
101	Servo Based Analog Meter	Intermediate	45 minutes
102	Servo Controlled Pointer Display	Intermediate	45 minutes
103	Stepper Motor Position Memory System	Advanced	1 hour
104	Stepper Motor Angle Preset Selector	Intermediate	1 hour
105	Automated Conveyor Simulation	Advanced	1 hour
106	Wi-Fi Light Switch	Intermediate	30 minutes
107	Wi-Fi Weather Station	Intermediate	45 minutes
108	Wi-Fi Colour Lamp	Intermediate	30 minutes
109	Internet Clock on OLED	Intermediate	30 minutes
110	IR Remote Light Ring	Intermediate	30 minutes

111	Wi-Fi Plant Monitor	Intermediate	30 minutes
112	Barometer and Altitude Meter	Intermediate	30 minutes
113	Robot Car First Drive	Intermediate	1 hour 30 minutes
114	Wi-Fi Robot Car	Intermediate	1 hour
115	Bluetooth Robot Car	Intermediate	1 hour
116	Obstacle Avoiding Robot Car	Intermediate	1 hour
117	IR Remote Robot Car	Intermediate	1 hour
118	Follow Me Robot Car	Intermediate	1 hour
119	Gyro Straight Line Robot Car	Advanced	1 hour 30 minutes
120	Auto Braking Smart Wi-Fi Robot Car	Advanced	2 hours

PROJECT 001 · LEVEL 1 · FOUNDATIONS

Blink One LED

Goal: Make a single LED switch on and off at a steady rate, the first program every maker writes.

Two instructions carry the whole sketch. pinMode() tells the ESP32 that GPIO33 is an output, a pin that pushes voltage out rather than listening for it. digitalWrite() then sets that pin to HIGH (3.3 volts, LED on) or LOW (0 volts, LED off). delay() simply stops everything for the number of milliseconds you give it, which is why the LED holds each state for a full second. Because loop() runs forever, the four instructions inside it repeat for as long as the board has power.

Online page with troubleshooting: geekobot.in/projects/titan-kit-001-blink-one-led

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- What an output pin is
- pinMode() and digitalWrite()
- How delay() controls timing
- Why loop() repeats forever

Wiring

Part	Pin	ESP32	Note
LED	LED long leg (anode)	**GPIO33 (S pin)**	Through a 220 ohm resistor
LED	LED short leg (cathode)	**GPIO33 (G pin)**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int led = 33;
void setup()
{
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  Serial.println("Blink started");
}
void loop()
{
  digitalWrite(led, HIGH); // Voltage on - LED lights
  Serial.println("ON");
  delay(1000); // Wait one second
  digitalWrite(led, LOW); // Voltage off - LED dark
  Serial.println("OFF");
  delay(1000);
}
```

What you should see: The LED lights for one second, then goes dark for one second. The pattern repeats forever. Serial Monitor prints ON and OFF in time with the LED.

Alternate Two LEDs

Goal: Run two LEDs so that one is always on and the other always off, swapping over at a fixed rate.

Two outputs now need managing at once, and the trick is that they are always opposites. Setting one HIGH in the same breath as setting the other LOW guarantees they can never both be lit. Try adding a second delay between the two digitalWrite() calls and you will see both LEDs briefly on together, a small experiment that shows exactly how literally the board follows the order you write.

Online page with troubleshooting: geekobot.in/projects/titan-kit-002-alternate-two-leds

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Controlling several outputs
- Keeping two states as exact opposites
- Using a variable to hold a timing value
- How instruction order affects what you see

Wiring

Part	Pin	ESP32	Note
LED	Red LED anode	**GPIO33 (S pin)**	Through a 220 ohm resistor
LED	Red LED cathode	**GPIO33 (G pin)**	
LED	Green LED anode	**GPIO32 (S pin)**	Through a 220 ohm resistor
LED	Green LED cathode	**GPIO32 (G pin)**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int redLed = 33;
int greenLed = 32;
int waitTime = 500;
void setup()
{
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  digitalWrite(redLed, HIGH);
  digitalWrite(greenLed, LOW);
  Serial.println("Red on, green off");
  delay(waitTime);
  digitalWrite(redLed, LOW);
  digitalWrite(greenLed, HIGH);
  Serial.println("Green on, red off");
  delay(waitTime);
}
```

What you should see: The red LED lights while the green one stays dark. After half a second they swap over. The alternation continues indefinitely. Serial Monitor names whichever LED is currently lit.

Traffic Signal System

Goal: Recreate a road traffic light with red, yellow and green LEDs following the correct real-world sequence.

A real signal is a fixed sequence of states, each held for a different length of time, and the timings are not arbitrary. Green and red are long because traffic needs time to move; yellow is short because it is only a warning. Writing each phase as its own block of code, with allOff() called first, means no phase can accidentally leave a previous lamp lit. That habit, clear everything, then set what you want, prevents a whole class of bugs later on.

Online page with troubleshooting: geekobot.in/projects/titan-kit-003-traffic-signal

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Real-life automation sequences
- Breaking a process into named phases
- Clearing all outputs before setting new ones
- Why different phases need different durations

Wiring

Part	Pin	ESP32	Note
LED	Red LED anode	**GPIO33 (S pin)**	Through a 220 ohm resistor
LED	Yellow LED anode	**GPIO32 (S pin)**	Through a 220 ohm resistor
LED	Green LED anode	**GPIO5 (S pin)**	Through a 220 ohm resistor
LED	All cathodes	**Any G pin**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int redLed = 33;
int yellowLed = 32;
int greenLed = 5;
void allOff()
{
  digitalWrite(redLed, LOW);
  digitalWrite(yellowLed, LOW);
  digitalWrite(greenLed, LOW);
}
void setup()
{
  pinMode(redLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  // STOP
  allOff();
  digitalWrite(redLed, HIGH);
  Serial.println("RED - stop");
  delay(5000);
  // GET READY
  allOff();
  digitalWrite(redLed, HIGH);
  digitalWrite(yellowLed, HIGH);
  Serial.println("RED + YELLOW - get ready");
  delay(2000);
  // GO
  allOff();
  digitalWrite(greenLed, HIGH);
  Serial.println("GREEN - go");
  delay(5000);
  // PREPARE TO STOP
  allOff();
  digitalWrite(yellowLed, HIGH);
  Serial.println("YELLOW - prepare to stop");
  delay(2000);
}
```

What you should see: Red holds for 5 seconds. Red and yellow light together for 2 seconds. Green holds for 5 seconds. Yellow alone for 2 seconds, then the cycle restarts.

Push Button LED

Goal: Read a button as an input and light an LED only while the button is held down.

An input pin left unconnected picks up electrical noise and reads randomly, so it needs pulling to a known voltage. INPUT_PULLUP switches on a resistor inside the ESP32 that holds the pin at 5 V, which is why an unpressed button reads HIGH. Pressing the button connects the pin to ground, so it reads LOW. That inversion catches everyone at first: pressed means LOW, not HIGH.

Online page with troubleshooting: geekobot.in/projects/titan-kit-004-push-button-led

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Reading a digital input
- Why INPUT_PULLUP is needed
- Pressed reads LOW, not HIGH
- Detecting the moment of a change rather than a held state

Wiring

Part	Pin	ESP32	Note
Button	Button one leg	**GPIO13 (S pin)**	
Button	Button other leg	**GPIO13 (G pin)**	
LED	LED anode	**GPIO33 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**GPIO33 (G pin)**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int button = 13;
int led = 33;
int presses = 0;
bool wasPressed = false;
void setup()
{
  pinMode(button, INPUT_PULLUP); // Internal resistor holds the pin HIGH
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  Serial.println("Press the button");
}
void loop()
{
  bool pressed = (digitalRead(button) == LOW); // LOW means pressed
  if (pressed)
  {
    digitalWrite(led, HIGH);
  }
  else
  {
    digitalWrite(led, LOW);
  }
  // Count only the moment of pressing, not the whole hold
  if (pressed && !wasPressed)
  {
    presses = presses + 1;
    Serial.print("Press number ");
    Serial.println(presses);
  }
  wasPressed = pressed;
  delay(20);
}
```

What you should see: The LED is dark while the button is untouched. Hold the button and the LED lights. Release it and the LED goes out immediately. Serial Monitor counts each press exactly once, however long you hold.

Touch Sensor Lamp

Goal: Use a capacitive touch pad as a switch, where one tap turns the light on and the next turns it off.

The TTP223 senses the tiny change in electrical capacitance your finger adds to its copper pad, no moving parts, and it works through several millimetres of plastic or wood. The important idea in the code is toggling: instead of the LED following the pad, one touch flips the state. The ! operator means "the opposite of", so lampOn = !lampOn turns true into false and back. Waiting for the finger to lift stops one touch from flipping the state dozens of times.

Online page with troubleshooting: geekobot.in/projects/titan-kit-005-touch-sensor-lamp

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Capacitive Touch Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Capacitive touch sensing basics
- Toggling a state with the ! operator
- Storing a state in a bool variable
- Waiting for release so one touch counts once

Wiring

Part	Pin	ESP32	Note
Touch module	VCC	**GPIO13 (V pin)**	
Touch module	GND	**GPIO13 (G pin)**	
Touch module	SIG	**GPIO13 (S pin)**	
LED	LED anode	**GPIO33 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**GPIO33 (G pin)**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int touchPin = 13;
int led = 33;
bool lampOn = false;
void setup()
{
  pinMode(touchPin, INPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  Serial.println("Touch the pad to switch the lamp");
}
void loop()
{
  if (digitalRead(touchPin) == HIGH)
  {
    lampOn = !lampOn; // Flip to the opposite state
    digitalWrite(led, lampOn);
    Serial.print("Lamp is now ");
    Serial.println(lampOn ? "ON" : "OFF");
    // Wait until the finger is lifted before accepting another touch
    while (digitalRead(touchPin) == HIGH)
    {
      delay(10);
    }
    delay(50);
  }
}
```

What you should see: Touch the pad once and the LED lights. Take your finger away, the LED stays on. Touch again and it goes out. Serial Monitor reports the new state after each touch.

RGB Color Lamp

Goal: Produce many different colours from a single RGB LED by mixing three brightness levels.

An RGB LED is three separate LEDs sharing one package. digitalWrite() could only ever give eight colours, because each channel would be fully on or fully off. analogWrite() instead uses PWM, it switches the pin on and off thousands of times a second, and the fraction of time it spends on decides the apparent brightness. A value of 0 is off, 255 is full, and everything between is a dimmer level, which is how three channels produce millions of colours.

Online page with troubleshooting: geekobot.in/projects/titan-kit-006-rgb-color-lamp

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Colour mixing from three channels
- PWM and analogWrite() for brightness
- Why 0 to 255 is the range
- Fading by stepping a value in a loop

Wiring

Part	Pin	ESP32	Note
RGB LED	Red leg	**GPIO32 (S pin)**	Through a 220 ohm resistor
RGB LED	Green leg	**GPIO5 (S pin)**	Through a 220 ohm resistor
RGB LED	Blue leg	**GPIO23 (S pin)**	Through a 220 ohm resistor
RGB LED	Common cathode (longest leg)	**Any G pin**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int redPin = 32;
int greenPin = 5;
int bluePin = 23;
void showColour(int r, int g, int b, String name)
{
  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);
  Serial.print(name);
  Serial.print(" R");
  Serial.print(r);
  Serial.print(" G");
  Serial.print(g);
  Serial.print(" B");
  Serial.println(b);
  delay(1500);
}
void setup()
{
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  showColour(255, 0, 0, "Red");
  showColour( 0, 255, 0, "Green");
  showColour( 0, 0, 255, "Blue");
  showColour(255, 255, 0, "Yellow");
  showColour( 0, 255, 255, "Cyan");
  showColour(255, 0, 255, "Magenta");
  showColour(255, 120, 0, "Orange");
  showColour(255, 255, 255, "White");
  // Smooth fade from red to green
  for (int i = 0; i <= 255; i = i + 5)
  {
    analogWrite(redPin, 255 - i);
    analogWrite(greenPin, i);
    analogWrite(bluePin, 0);
    delay(20);
  }
}
```

What you should see: The lamp steps through eight named colours, holding each for 1.5 seconds. Serial Monitor prints the name and the three channel values. After white it fades smoothly from red to green. The whole sequence then repeats.

PROJECT 007 · LEVEL 1 · FOUNDATIONS

Emergency Light Effect

Goal: Create a convincing emergency-vehicle flash using two LEDs and a repeating burst pattern.

A emergency light is not a simple alternation, it is a rapid burst on one side, a pause, then a burst on the other. That structure is what makes it read as urgent rather than decorative. A for loop produces each burst without repeating the same three lines four times over, and changing the loop's limit changes the number of flashes per burst. Short delays inside the burst and a longer one between them create the rhythm.

Online page with troubleshooting: geekobot.in/projects/titan-kit-007-emergency-light-effect

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Patterns and timing
- for loops to repeat an action
- Writing a reusable function with parameters
- How rhythm changes the character of a light

Wiring

Part	Pin	ESP32	Note
LED	Red LED anode	**GPIO33 (S pin)**	Through a 220 ohm resistor
LED	Blue LED anode	**GPIO32 (S pin)**	Through a 220 ohm resistor
LED	Both cathodes	**Any G pin**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int redLed = 33;
int blueLed = 32;
int flashesPerBurst = 4;
int flashTime = 60;
int gapBetweenBursts = 120;
void burst(int pin, int count)
{
  for (int i = 0; i < count; i++)
  {
    digitalWrite(pin, HIGH);
    delay(flashTime);
    digitalWrite(pin, LOW);
    delay(flashTime);
  }
}
void setup()
{
  pinMode(redLed, OUTPUT);
  pinMode(blueLed, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  Serial.println("Red burst");
  burst(redLed, flashesPerBurst);
  delay(gapBetweenBursts);
  Serial.println("Blue burst");
  burst(blueLed, flashesPerBurst);
  delay(gapBetweenBursts);
}
```

What you should see: The red LED flashes four times in rapid succession. A short pause follows. The blue LED flashes four times. The pattern repeats continuously with an urgent rhythm.

Buzzer Beep Sound

Goal: Drive a buzzer to make alarm sounds, and hear the difference between simple beeps and varied patterns.

An active buzzer contains its own oscillator, so digitalWrite() HIGH is enough to make it sound, it plays one fixed pitch and nothing else. What you can control is the pattern: how long each beep lasts and how long the silence between them is. Those two numbers alone carry a surprising amount of meaning, which is why a slow beep reads as a reminder and a fast one as an emergency.

Online page with troubleshooting: geekobot.in/projects/titan-kit-008-buzzer-beep-sound

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Sound output from a digital pin
- How an active buzzer differs from a passive one
- Building patterns from on-time and off-time
- Using repetition rate to signal urgency

Wiring

Part	Pin	ESP32	Note
Buzzer	Buzzer (+)	**GPIO33 (S pin)**	
Buzzer	Buzzer (−)	**GPIO33 (G pin)**	
LED	LED anode	**GPIO32 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**GPIO32 (G pin)**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int buzzer = 33;
int led = 32;
void beep(int onTime, int offTime, int count)
{
  for (int i = 0; i < count; i++)
  {
    digitalWrite(buzzer, HIGH);
    digitalWrite(led, HIGH);
    delay(onTime);
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(offTime);
  }
}
void setup()
{
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  Serial.println("Single long beep");
  beep(800, 400, 1);
  delay(800);
  Serial.println("Reminder - two slow beeps");
  beep(200, 300, 2);
  delay(1200);
  Serial.println("Warning - four medium beeps");
  beep(120, 120, 4);
  delay(1200);
  Serial.println("Emergency - rapid beeping");
  beep(50, 50, 10);
  delay(2000);
}
```

What you should see: One long beep, with the LED flashing in time. Two slower beeps as a gentle reminder. Four medium beeps as a warning. Ten rapid beeps that sound genuinely urgent.

Musical Buzzer

Goal: Play recognisable tunes on a passive buzzer by sending it specific frequencies.

Pitch is frequency measured in hertz, how many times per second the buzzer's disc is pushed and pulled. Middle C is 262 Hz, and doubling any frequency raises it exactly one octave, which is why 523 Hz is the C above. tone() generates that signal in the background so the sketch can carry on, and noTone() stops it. A tune is then just two arrays walked in step: one of pitches and one of how long each note lasts.

Online page with troubleshooting: geekobot.in/projects/titan-kit-009-musical-buzzer

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Musical notes as frequencies
- tone() and noTone() for pitch
- Parallel arrays for pitch and rhythm
- Why doubling a frequency gives the same note an octave higher

Wiring

Part	Pin	ESP32	Note
Passive buzzer	Buzzer (+)	**GPIO33 (S pin)**	
Passive buzzer	Buzzer (−)	**GPIO33 (G pin)**	
Button	Button one leg	**GPIO13 (S pin)**	
Button	Button other leg	**GPIO13 (G pin)**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int buzzer = 33;
int button = 13;
// Note frequencies in hertz
int NOTE_C4 = 262, NOTE_D4 = 294, NOTE_E4 = 330, NOTE_F4 = 349;
int NOTE_G4 = 392, NOTE_A4 = 440, NOTE_B4 = 494, NOTE_C5 = 523;
// Twinkle Twinkle Little Star
int melody[14] = {NOTE_C4, NOTE_C4, NOTE_G4, NOTE_G4, NOTE_A4, NOTE_A4, NOTE_G4, NOTE_F4, NOTE_F4,
NOTE_E4, NOTE_E4, NOTE_D4, NOTE_D4, NOTE_C4};
int beats[14] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2};
int beatLength = 400;
void playTune()
{
  for (int i = 0; i < 14; i++)
  {
    int duration = beats[i] * beatLength;
    tone(buzzer, melody[i], duration * 0.9);
    Serial.print("Note ");
    Serial.print(melody[i]);
    Serial.println(" Hz");
    delay(duration); // Full beat, including the short gap
  }
  noTone(buzzer);
}
void playScale()
{
  int scale[8] = {NOTE_C4, NOTE_D4, NOTE_E4, NOTE_F4, NOTE_G4, NOTE_A4, NOTE_B4, NOTE_C5};
  for (int i = 0; i < 8; i++)
  {
    tone(buzzer, scale[i], 250);
    delay(300);
  }
  noTone(buzzer);
}
void setup()
{
  pinMode(buzzer, OUTPUT);
  pinMode(button, INPUT_PULLUP);
  Serial.begin(9600);
  Serial.println("Press the button to play");
  playScale();
}
void loop()
{
  if (digitalRead(button) == LOW)
  {
    Serial.println("Playing the tune");
    playTune();
    while (digitalRead(button) == LOW)
    {
      delay(10);
    }
    delay(300);
  }
}
```

What you should see: A rising eight-note scale plays when the board starts. Press the button and Twinkle Twinkle Little Star plays. Serial Monitor shows the frequency of each note. Longer notes are held for two beats instead of one.

Clap Detection

Goal: Make an LED respond to a clap, using a sound sensor and a guard that ignores the echo.

The module's digital output pulses LOW whenever the sound crosses the level set by its onboard potentiometer. The complication is that one clap in a hard-walled room produces several pulses as it echoes off the surfaces. Recording the time of each accepted clap and refusing anything within 300 ms of it filters those echoes out. That guard is the difference between a light that toggles once and one that flickers unpredictably.

Online page with troubleshooting: geekobot.in/projects/titan-kit-010-clap-detection

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Sound sensing with a digital output
- Using millis() to record when something happened
- Guard windows that reject echoes
- Combining sensing with a toggled state
- Combine projects, put the Bluetooth dashboard on the smart home, or give the obstacle robot the RFID reader so only a registered card can start it.
- Replace every remaining delay() with millis() so your projects can do several things at once, as Experiment 51 does.
- Log your readings, stream them over Bluetooth, paste them into a spreadsheet and plot a day's temperature.
- Store settings in EEPROM so PINs, card UIDs and thresholds survive a power cut.
- Add a real-time clock module and let the smart home act on the time of day, not just on its sensors.
- Design your own project from scratch: write down the inputs, the outputs, and the decision in between, then wire and code it.

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**GPIO13 (V pin)**	
Sound sensor	GND	**GPIO13 (G pin)**	
Sound sensor	DO (digital out)	**GPIO13 (S pin)**	
LED	LED anode	**GPIO33 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**GPIO33 (G pin)**	
Buzzer	Buzzer (+)	**GPIO32 (S pin)**	
Buzzer	Buzzer (−)	**GPIO32 (G pin)**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int soundPin = 13;
int led = 33;
int buzzer = 32;
bool lightOn = false;
unsigned long lastClap = 0;
unsigned long deadTime = 300; // Ignore echoes for this long
int clapCount = 0;
void setup()
{
  pinMode(soundPin, INPUT);
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  Serial.println("Clap to switch the light");
}
void loop()
{
  if (digitalRead(soundPin) == LOW)
  {
    if (millis() - lastClap > deadTime)
    {
      lastClap = millis();
      clapCount = clapCount + 1;
      lightOn = !lightOn;
      digitalWrite(led, lightOn);
      digitalWrite(buzzer, HIGH);
      delay(40);
      digitalWrite(buzzer, LOW);
      Serial.print("Clap ");
      Serial.print(clapCount);
      Serial.print(" - light is ");
      Serial.println(lightOn ? "ON" : "OFF");
    }
  }
}
```

What you should see: Clap once and the LED lights with a short chirp. Clap again and it goes out. Echoes in a hard room do not cause extra switching. Serial Monitor counts every accepted clap.

Digital Dice

Goal: Roll an electronic dice: press a button and a random number from 1 to 6 appears on a 7-segment display, complete with a rolling animation.

Each digit is drawn by lighting a particular combination of the seven segments. Those combinations are stored in a two-dimensional array, so showing a number is simply a matter of walking through one row of that array. randomSeed(analogRead(A0)) reads electrical noise from an unconnected analog pin, which gives a different starting point every time the board is powered up, without it, the dice would roll the same sequence after every reset.

Online page with troubleshooting: geekobot.in/projects/titan-kit-011-digital-dice

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x 7-Segment LED Display 0.56"
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Random number generation with random() and randomSeed()
- 7-segment logic, which segments make which digit
- Two-dimensional arrays
- Reading a button with INPUT_PULLUP

Wiring

Part	Pin	ESP32	Note
7-segment	Segment a	**GPIO13**	Through a 220 ohm resistor
7-segment	Segment b	**GPIO12**	Through a 220 ohm resistor
7-segment	Segment c	**GPIO14**	Through a 220 ohm resistor
7-segment	Segment d	**GPIO27**	Through a 220 ohm resistor
7-segment	Segment e	**GPIO26**	Through a 220 ohm resistor
7-segment	Segment f	**GPIO25**	Through a 220 ohm resistor
7-segment	Segment g	**GPIO33**	Through a 220 ohm resistor
7-segment	Common (both COM pins)	**GND**	
Button	One leg	**GPIO32**	
Button	Other leg	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int segPins[7] = {13, 12, 14, 27, 26, 25, 33};
int buttonPin = 32;
// Segment order: a b c d e f g
byte digits[7][7] =
{
  {0, 0, 0, 0, 0, 0, 0}, // blank
  {0, 1, 1, 0, 0, 0, 0}, // 1
  {1, 1, 0, 1, 1, 0, 1}, // 2
  {1, 1, 1, 1, 0, 0, 1}, // 3
  {0, 1, 1, 0, 0, 1, 1}, // 4
  {1, 0, 1, 1, 0, 1, 1}, // 5
  {1, 0, 1, 1, 1, 1, 1} // 6
};
void showDigit(int n)
{
  for (int i = 0; i < 7; i++)
  {
    digitalWrite(segPins[i], digits[n][i]);
  }
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  for (int i = 0; i < 7; i++)
  {
    pinMode(segPins[i], OUTPUT);
  }
  pinMode(buttonPin, INPUT_PULLUP);
  randomSeed(analogRead(A0));
  showDigit(0);
}
void loop()
{
  if (digitalRead(buttonPin) == LOW)
  {
    // Rolling animation
    for (int i = 0; i < 15; i++)
    {
      showDigit(random(1, 7));
      delay(80);
    }
    int result = random(1, 7);
    showDigit(result);
    delay(600);
  }
}
```

What you should see: Display stays blank until the button is pressed. Numbers flicker rapidly for about one second. A single number between 1 and 6 stays on screen.

Scrolling Text Display

Goal: Scroll your own message across an 8×8 LED matrix, letter by letter, the way a railway station board does.

An 8×8 matrix has 64 LEDs but only 16 pins. The MAX7219 driver lights one row at a time, extremely fast, and your eye blends the rows into a single steady image, this is called multiplexing. The library holds the message as an array of characters and shifts the pixel columns one step to the left on every animation frame, which is what produces the scroll.

Online page with troubleshooting: geekobot.in/projects/titan-kit-012-scrolling-text

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x MAX7219 LED Display Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Matrix multiplexing, how 64 LEDs are driven from one chip
- Arrays of characters and pixel columns
- Using a display library
- Controlling scroll speed and brightness

Wiring

Part	Pin	ESP32	Note
Matrix	VCC	**5V**	
Matrix	GND	**GND**	
Matrix	DIN	**GPIO23**	
Matrix	CS	**GPIO5**	
Matrix	CLK	**GPIO18**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <SPI.h>
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES 1
#define CS_PIN 5
MD_Parola display = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);
void setup()
{
  display.begin();
  display.setIntensity(5);
  display.displayClear();
  display.displayText("HELLO STUDENTS",
  PA_CENTER,
  80,
  0,
  PA_SCROLL_LEFT,
  PA_SCROLL_LEFT);
}
void loop()
{
  if (display.displayAnimate())
  {
    display.displayReset();
  }
}
```

What you should see: The message HELLO STUDENTS scrolls right to left. The scroll repeats continuously. Change the text inside displayText() to show your own message.

Emoji Display on LED Matrix

Goal: Design your own pixel art, a smiley and a heart, and display them on the 8×8 LED matrix.

Each row of the matrix is one byte, eight bits, one per LED. Writing B00111100 turns on the four middle LEDs of that row and leaves the rest off. Stack eight of those bytes and you have drawn a picture. This is bitmapping, and it is exactly how fonts and icons are stored on every small display.

Online page with troubleshooting: geekobot.in/projects/titan-kit-013-emoji-led-matrix

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x MAX7219 LED Display Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Bitmapping, turning binary numbers into pictures
- Custom characters and icons
- Passing arrays to a function
- Row-by-row control of a matrix

Wiring

Part	Pin	ESP32	Note
Matrix	VCC	**5V**	
Matrix	GND	**GND**	
Matrix	DIN	**GPIO23**	
Matrix	CS	**GPIO5**	
Matrix	CLK	**GPIO18**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <LedControl.h>
// LedControl(DIN, CLK, CS, number of devices)
LedControl lc = LedControl(23, 18, 5, 1);
byte smiley[8] =
{
  0b00111100,
  0b01000010,
  0b10100101,
  0b10000001,
  0b10100101,
  0b10011001,
  0b01000010,
  0b00111100
};
byte heart[8] =
{
  0b00000000,
  0b01100110,
  0b11111111,
  0b11111111,
  0b11111111,
  0b01111110,
  0b00111100,
  0b00011000
};
void drawImage(byte image[])
{
  for (int row = 0; row < 8; row++)
  {
    lc.setRow(0, row, image[row]);
  }
}
void setup()
{
  lc.shutdown(0, false);
  lc.setIntensity(0, 5);
  lc.clearDisplay(0);
}
void loop()
{
  drawImage(smiley);
  delay(2000);
  drawImage(heart);
  delay(2000);
}
```

What you should see: A smiley face appears for 2 seconds. A heart appears for 2 seconds. The two images alternate continuously.

Countdown Timer Game

Goal: Build a defusal game: an LCD counts down from ten while the buzzer beeps, and the player must press the stop button before time runs out.

The sketch cannot simply call delay(1000) between ticks, during a delay the ESP32 ignores the button, so a player could press it and nothing would happen. Instead each second is split into ninety short 10 ms waits, and the button is checked between every one of them. A boolean variable remembers whether the player stopped the clock in time. Together, the counter and the flag are the state of the game.

Online page with troubleshooting: geekobot.in/projects/titan-kit-014-countdown-timer-game

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Complex timing without blocking the program
- State management with flag variables
- Splitting a long wait so inputs stay responsive
- Printing changing values to an LCD

Wiring

Part	Pin	ESP32	Note
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	
Buzzer	Buzzer (+)	**GPIO33**	
Buzzer	Buzzer (−)	**GND**	
Button	Button one leg	**GPIO25**	
Button	Button other leg	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
int buzzer = 33;
int button = 25;
int timeLeft = 10;
bool stopped = false;
void setup()
{
  lcd.init();
  lcd.backlight();
  pinMode(buzzer, OUTPUT);
  pinMode(button, INPUT_PULLUP);
  lcd.setCursor(0, 0);
  lcd.print("COUNTDOWN START");
  delay(1500);
  lcd.clear();
}
void loop()
{
  if (stopped)
  {
    return;
  }
  if (timeLeft > 0)
  {
    lcd.setCursor(0, 0);
    lcd.print("Time left: ");
    lcd.print(timeLeft);
    lcd.print(" ");
    digitalWrite(buzzer, HIGH);
    delay(100);
    digitalWrite(buzzer, LOW);
    // Wait the rest of the second, watching the button
    for (int i = 0; i < 90; i++)
    {
      if (digitalRead(button) == LOW)
      {
        stopped = true;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("YOU WIN!");
        return;
      }
      delay(10);
    }
    timeLeft = timeLeft - 1;
  }
  else
  {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("TIME UP!");
    digitalWrite(buzzer, HIGH);
    delay(2000);
    digitalWrite(buzzer, LOW);
    while (true)
    {
      // Game over - press reset to play again
    }
  }
}
```

What you should see: LCD shows COUNTDOWN START, then Time left: 10, 9, 8 …. The buzzer ticks once every second. Press the button in time and the LCD shows YOU WIN! Let it reach zero and the LCD shows TIME UP! with a long buzz.

LCD Welcome System

Goal: Display a personalised greeting on a 16×2 LCD, the foundation for every project in this manual that shows text.

An I2C LCD needs only two signal wires because the small board on its back converts the two-wire I2C messages into the sixteen signals the display itself expects. lcd.setCursor(column, row) chooses where the next text will start: columns are numbered 0 to 15 and rows 0 to 1.

Online page with troubleshooting: geekobot.in/projects/titan-kit-015-lcd-welcome

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x ESP32 38-Pin Expansion Board

What you learn

- LCD interfacing over I2C
- String printing and the String variable type
- Positioning text with setCursor()
- Clearing and redrawing a screen

Wiring

Part	Pin	ESP32	Note
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
String studentName = "STUDENT";
void setup()
{
  lcd.init();
  lcd.backlight();
}
void loop()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("WELCOME");
  lcd.setCursor(0, 1);
  lcd.print(studentName);
  delay(3000);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Have a great");
  lcd.setCursor(0, 1);
  lcd.print("day ahead!");
  delay(3000);
}
```

What you should see: WELCOME on the top line and the name below it. After 3 seconds the message changes to Have a great day ahead! The two screens alternate continuously.

LCD Temperature Display

Goal: Read the room temperature from a DHT11 sensor and show it live on the LCD instead of the Serial Monitor.

The DHT11 needs about two seconds between readings, so the sketch waits before asking again. Temperature comes back as a float, a number with a decimal point, and the degree symbol is written with lcd.write(223), because that character is not part of ordinary ASCII.

Online page with troubleshooting: geekobot.in/projects/titan-kit-016-lcd-temperature

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Sensor libraries and how they hide the hard work
- LCD formatting and special characters
- Working with float values
- Checking a reading with isnan() before trusting it

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup()
{
  lcd.init();
  lcd.backlight();
  dht.begin();
  lcd.setCursor(0, 0);
  lcd.print("Room Monitor");
  delay(2000);
  lcd.clear();
}
void loop()
{
  float temp = dht.readTemperature();
  if (isnan(temp))
  {
    lcd.setCursor(0, 0);
    lcd.print("Sensor error ");
    delay(2000);
    return;
  }
  lcd.setCursor(0, 0);
  lcd.print("Temperature: ");
  lcd.setCursor(0, 1);
  lcd.print(temp);
  lcd.write(223);
  lcd.print("C ");
  delay(2000);
}
```

What you should see: Temperature: on the top line. 29.00°C on the second line. The value refreshes every 2 seconds.

LCD Humidity Monitor

Goal: Track the moisture in the air continuously, showing the current reading together with the highest and lowest values seen so far.

A single reading tells you very little. By keeping two extra variables, one for the smallest value ever seen and one for the largest, the sketch turns a series of instant readings into a record of what happened while you were not watching. This is the essence of continuous monitoring, and the same three lines of logic appear in weather stations and data loggers.

Online page with troubleshooting: geekobot.in/projects/titan-kit-017-lcd-humidity-monitor

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Continuous sensor monitoring
- Keeping running minimum and maximum values
- Comparing floats with if statements
- Controlling decimal places when printing

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);
float minHum = 100.0;
float maxHum = 0.0;
void setup()
{
  lcd.init();
  lcd.backlight();
  dht.begin();
  lcd.setCursor(0, 0);
  lcd.print("Humidity Log");
  delay(2000);
  lcd.clear();
}
void loop()
{
  float hum = dht.readHumidity();
  if (isnan(hum))
  {
    lcd.setCursor(0, 0);
    lcd.print("Sensor error ");
    delay(2000);
    return;
  }
  if (hum < minHum)
  {
    minHum = hum;
  }
  if (hum > maxHum)
  {
    maxHum = hum;
  }
  lcd.setCursor(0, 0);
  lcd.print("Humidity: ");
  lcd.print(hum, 0);
  lcd.print("% ");
  lcd.setCursor(0, 1);
  lcd.print("L:");
  lcd.print(minHum, 0);
  lcd.print(" H:");
  lcd.print(maxHum, 0);
  lcd.print(" ");
  delay(2000);
}
```

What you should see: Humidity: 62% on the top line. L:58 H:67 on the second line. Both lines update every 2 seconds; breathe gently on the sensor and watch the high value climb.

RGB Mood Lamp

Goal: Turn a single knob and sweep smoothly through the whole colour spectrum on an RGB LED.

The potentiometer gives a number from 0 to 1023. That single number is stretched to a 0–764 colour wheel and then split into three stages: red fading into green, green into blue, and blue back into red. Each of the three channels is written with analogWrite(), which uses PWM, switching the LED on and off thousands of times a second so it appears dimmer or brighter rather than simply on or off.

Online page with troubleshooting: geekobot.in/projects/titan-kit-018-rgb-mood-lamp

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Analog input mapping to PWM output
- The map() function
- Mixing colours from three channels
- Why analogWrite() only works on PWM (~) pins

Wiring

Part	Pin	ESP32	Note
RGB LED	Red	**GPIO32**	Through a 220 ohm resistor
RGB LED	Green	**GPIO5**	Through a 220 ohm resistor
RGB LED	Blue	**GPIO23**	Through a 220 ohm resistor
RGB LED	Common cathode (−)	**GND**	
Potentiometer	Left pin	**GND**	
Potentiometer	Middle pin (wiper)	**GPIO34**	
Potentiometer	Right pin	**3V3**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int potPin = 34;
int redPin = 32;
int greenPin = 5;
int bluePin = 23;
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}
void loop()
{
  int potValue = analogRead(potPin);
  int wheel = map(potValue, 0, 1023, 0, 764);
  int r = 0;
  int g = 0;
  int b = 0;
  if (wheel < 255)
  {
    r = 255 - wheel;
    g = wheel;
    b = 0;
  }
  else if (wheel < 510)
  {
    r = 0;
    g = 510 - wheel;
    b = wheel - 255;
  }
  else
  {
    r = wheel - 510;
    g = 0;
    b = 764 - wheel;
  }
  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);
  delay(20);
}
```

What you should see: Knob fully left: pure red. Turning slowly: red → orange → green → cyan → blue → magenta. Knob fully right: back to red.

Smart Street Light

Goal: Light a row of LEDs automatically when the surroundings get dark, and switch them off again at dawn, the logic behind every street lamp in your town.

An LDR is simply a resistor whose value falls when light hits it. On its own the ESP32 cannot measure resistance, so the LDR is paired with a fixed 10kΩ resistor to form a voltage divider: as the light changes, the voltage at GPIO34 changes with it. The threshold in the sketch is the decision point between day and night, read your own room's values in the Serial Monitor first and set it about halfway between the lit and the covered reading.

Online page with troubleshooting: geekobot.in/projects/titan-kit-019-smart-street-light

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 4 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 10k resistor
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x ESP32 38-Pin Expansion Board

What you learn

- LDR thresholds and automation
- Voltage dividers, turning resistance into voltage
- Driving several outputs from one function
- Calibrating a sensor to its environment

Wiring

Part	Pin	ESP32	Note
LDR divider	LDR leg 1	**3V3**	
LDR divider	LDR leg 2	**GPIO34 and one end of 10kΩ resistor**	
LDR divider	Other end of 10kΩ resistor	**GND**	
LED	LED 1 anode	**GPIO13**	Through a 220 ohm resistor
LED	LED 2 anode	**GPIO12**	Through a 220 ohm resistor
LED	LED 3 anode	**GPIO14**	Through a 220 ohm resistor
LED	LED 4 anode	**GPIO27**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int ldrPin = 34;
int ledPins[4] = {13, 12, 14, 27};
int threshold = 400;
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  for (int i = 0; i < 4; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  Serial.begin(9600);
}
void setLamps(int state)
{
  for (int i = 0; i < 4; i++)
  {
    digitalWrite(ledPins[i], state);
  }
}
void loop()
{
  int lightLevel = analogRead(ldrPin);
  Serial.print("Light level: ");
  Serial.println(lightLevel);
  if (lightLevel < threshold)
  {
    setLamps(HIGH); // Dark - street lights ON
  }
  else
  {
    setLamps(LOW); // Bright - street lights OFF
  }
  delay(300);
}
```

What you should see: Serial Monitor prints Light level: 720 in a lit room. Cover the LDR with your hand and all four LEDs switch on together. Uncover it and they switch off again.

Smart Umbrella Reminder

Goal: Detect the first drops of rain and put a clear reminder on the LCD before anyone leaves the house.

The rain board's digital output is HIGH while the plate is dry and drops LOW the moment water bridges its tracks. The blue potentiometer on the comparator board sets how wet the plate must be before that happens, turn it until a single drop is enough. Notice that the sketch only redraws the LCD when the state actually changes; constantly clearing a display makes it flicker.

Online page with troubleshooting: geekobot.in/projects/titan-kit-020-smart-umbrella-reminder

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic Raindrops Detection Sensor Module – LM393 Comparator, Analog/Digital Output
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Conditional logic driving a display
- Weather sensing with a digital module
- Remembering the previous state to avoid flicker
- Adjusting a comparator's sensitivity

Wiring

Part	Pin	ESP32	Note
Rain sensor	VCC	**3V3**	
Rain sensor	GND	**GND**	
Rain sensor	DO (digital out)	**GPIO13**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int rainPin = 13;
LiquidCrystal_I2C lcd(0x27, 16, 2);
int lastState = -1;
void setup()
{
  pinMode(rainPin, INPUT);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Weather Check");
  delay(2000);
  lcd.clear();
}
void loop()
{
  int rainState = digitalRead(rainPin);
  if (rainState != lastState)
  {
    lcd.clear();
    if (rainState == LOW)
    {
      lcd.setCursor(0, 0);
      lcd.print("RAIN DETECTED!");
      lcd.setCursor(0, 1);
      lcd.print("Take umbrella");
    }
    else
    {
      lcd.setCursor(0, 0);
      lcd.print("Weather: Clear");
      lcd.setCursor(0, 1);
      lcd.print("Have a nice day");
    }
    lastState = rainState;
  }
  delay(200);
}
```

What you should see: Dry plate: Weather: Clear / Have a nice day. Sprinkle a few drops of water: RAIN DETECTED! / Take umbrella. Dry the plate and the message returns to clear.

Soil Moisture Detector

Goal: Measure how much water is in the soil of a plant pot and report it on the LCD as a percentage with a plain-English status.

Wet soil conducts electricity better than dry soil, so the analog reading falls as the soil gets wetter. That is backwards from what a person expects, so map() is used with its input range reversed, 1023 becomes 0% and 300 becomes 100%. constrain() then clips anything outside that band so the display never shows 104%.

Online page with troubleshooting: geekobot.in/projects/titan-kit-021-soil-moisture-detector

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Capacitive Soil Moisture Sensor – 3.3V/5V Analog Output, PH2.54-3P Interface
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Analog sensor reading for soil
- Reversing a range with map()
- Keeping values in bounds with constrain()
- Turning a number into a readable status

Wiring

Part	Pin	ESP32	Note
Soil sensor	VCC	**3V3**	
Soil sensor	GND	**GND**	
Soil sensor	AO (analog out)	**GPIO34**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int soilPin = 34;
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  lcd.init();
  lcd.backlight();
  Serial.begin(9600);
  lcd.setCursor(0, 0);
  lcd.print("Soil Monitor");
  delay(2000);
  lcd.clear();
}
void loop()
{
  int raw = analogRead(soilPin);
  // Wet soil gives a LOW reading, so the range is reversed
  int moisture = map(raw, 1023, 300, 0, 100);
  moisture = constrain(moisture, 0, 100);
  Serial.print("Raw: ");
  Serial.print(raw);
  Serial.print(" Moisture: ");
  Serial.println(moisture);
  lcd.setCursor(0, 0);
  lcd.print("Moisture: ");
  lcd.print(moisture);
  lcd.print("% ");
  lcd.setCursor(0, 1);
  if (moisture < 30)
  {
    lcd.print("Status: DRY ");
  }
  else if (moisture < 70)
  {
    lcd.print("Status: OK ");
  }
  else
  {
    lcd.print("Status: WET ");
  }
  delay(1000);
}
```

What you should see: Sensor in air: Moisture: 0% / Status: DRY. Sensor in damp soil: Moisture: 55% / Status: OK. Sensor in a glass of water: Moisture: 100% / Status: WET.

Automatic Plant Watering System

Goal: Combine the soil sensor with a relay and a pump so the plant waters itself whenever the soil dries out.

The relay is an electrically operated switch: a tiny current from GPIO25 closes a pair of heavy contacts that carry the pump's much larger current. The ESP32 and the pump therefore share no power at all, only a common ground reference. Most relay modules are active-LOW, meaning digitalWrite(relay, LOW) is what turns the pump on. Two thresholds are used rather than one: watering starts below 30% and stops above 60%, so the pump does not chatter on and off around a single boundary.

Online page with troubleshooting: geekobot.in/projects/titan-kit-022-automatic-plant-watering

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Capacitive Soil Moisture Sensor – 3.3V/5V Analog Output, PH2.54-3P Interface
- 1 x 5V Relay Module with Optocoupler
- 1 x Mini Submersible Water Pump 3–6V
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Relay control and threshold logic
- Hysteresis, two thresholds instead of one
- Powering a load from a separate supply
- Why a common ground is essential

Wiring

Part	Pin	ESP32	Note
Soil sensor	VCC	**3V3**	
Soil sensor	GND	**GND**	
Soil sensor	AO	**GPIO34**	
Relay	VCC	**5V**	
Relay	GND	**GND**	
Relay	IN	**GPIO25**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	The 5V jumper stays on: the board's regulator makes 5V
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
Relay	COM	**L298N +5V**	The pump never runs from the ESP32
Relay	NO	**Pump red wire (+)**	
Pump	Black wire (−)	**L298N GND**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

The pump runs at 5V from the L298N's on-board regulator, fed by the 2 x 18650 pack. The relay switches that 5V; the ESP32 only drives the relay.

The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Keep water away from the ESP32 and test the pump and tube over a tray first.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int soilPin = 34;
int relayPin = 25;
int dryLevel = 30; // Start watering below this
int wetLevel = 60; // Stop watering above this
bool pumpOn = false;
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH); // Relay is active LOW - pump OFF
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Auto Watering");
  delay(2000);
  lcd.clear();
}
void loop()
{
  int raw = analogRead(soilPin);
  int moisture = map(raw, 1023, 300, 0, 100);
  moisture = constrain(moisture, 0, 100);
  if (moisture < dryLevel)
  {
    pumpOn = true;
  }
  if (moisture > wetLevel)
  {
    pumpOn = false;
  }
  if (pumpOn)
  {
    digitalWrite(relayPin, LOW); // Pump ON
  }
  else
  {
    digitalWrite(relayPin, HIGH); // Pump OFF
  }
  lcd.setCursor(0, 0);
  lcd.print("Moisture: ");
  lcd.print(moisture);
  lcd.print("% ");
  lcd.setCursor(0, 1);
  if (pumpOn)
  {
    lcd.print("Pump: WATERING ");
  }
  else
  {
    lcd.print("Pump: IDLE ");
  }
  delay(1000);
}
```

What you should see: Damp soil: Moisture: 65% / Pump: IDLE. Soil dries below 30%: the relay clicks and Pump: WATERING appears. Once moisture passes 60% the relay clicks off again.

Water Tank Automation

Goal: Measure the water level in a tank with an ultrasonic sensor and switch a filling pump on and off automatically.

The sensor sends a burst of sound and measures how long the echo takes to return. Sound travels roughly 0.034 cm per microsecond, and the pulse makes the trip twice, so distance = duration × 0.034 ÷ 2. Mount the sensor facing down at the top of the tank: a large distance means the water is far away and the tank is empty, a small distance means it is nearly full.

Online page with troubleshooting: geekobot.in/projects/titan-kit-023-water-tank-automation

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x 5V Relay Module with Optocoupler
- 1 x Mini Submersible Water Pump 3–6V
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Ultrasonic distance measurement with pulseIn()
- Relay logic driven by a measured level
- Converting a distance into a percentage
- Writing your own function that returns a value

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
Relay	VCC	**5V**	
Relay	GND	**GND**	
Relay	IN	**GPIO25**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	The 5V jumper stays on: the board's regulator makes 5V
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
Relay	COM	**L298N +5V**	The pump never runs from the ESP32

Relay	NO	**Pump red wire (+)**	
Pump	Black wire (−)	**L298N GND**	

> The pump runs at 5V from the L298N's on-board regulator, fed by the 2 x 18650 pack. The relay switches that 5V; the ESP32 only drives the relay.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Keep water away from the ESP32 and test the pump and tube over a tray first.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int trigPin = 32;
int echoPin = 5;
int relayPin = 25;
int tankHeight = 30; // cm from sensor to the bottom
int fillBelow = 30; // Start pump below this percent
int stopAbove = 90; // Stop pump above this percent
bool pumpOn = false;
LiquidCrystal_I2C lcd(0x27, 16, 2);
long readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH);
  long distance = duration * 0.034 / 2;
  return distance;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH); // Pump OFF
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Tank Monitor");
  delay(2000);
  lcd.clear();
}
void loop()
{
  long distance = readDistance();
  int level = map(distance, tankHeight, 2, 0, 100);
  level = constrain(level, 0, 100);
  if (level < fillBelow)
  {
    pumpOn = true;
  }
  if (level > stopAbove)
  {
    pumpOn = false;
  }
  if (pumpOn)
  {
    digitalWrite(relayPin, LOW);
  }
  else
  {
    digitalWrite(relayPin, HIGH);
  }
  lcd.setCursor(0, 0);
  lcd.print("Level: ");
  lcd.print(level);
  lcd.print("% ");
  lcd.setCursor(0, 1);
  if (pumpOn)
  {
    lcd.print("Pump: FILLING ");
  }
  else
  {
    lcd.print("Pump: STOPPED ");
  }
  delay(500);
}
```

What you should see: Level: 85% / Pump: STOPPED with the tank nearly full. Move your hand away from the sensor to simulate a falling level. Below 30% the relay clicks and Pump: FILLING appears.

Smart Temperature Alarm

Goal: Watch the room temperature and sound a loud buzzer the moment it crosses a limit you set.

The alarm is a threshold comparison: read a value, compare it with a limit, act. Rather than holding the buzzer permanently on, the sketch pulses it, which is both more attention-grabbing and easier on the ear. Change alarmTemp to a value just above your current room temperature and you can trigger the alarm by cupping the sensor in your hand.

Online page with troubleshooting: geekobot.in/projects/titan-kit-024-smart-temperature-alarm

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Threshold-based environmental alarms
- Comparing float readings against a limit
- Pulsing an output instead of holding it on
- Combining a sensor, a display and two outputs

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
Buzzer	Buzzer (+)	**GPIO33**	
Buzzer	Buzzer (−)	**GND**	
LED	LED anode	**GPIO26**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);
int buzzer = 33;
int led = 26;
float alarmTemp = 32.0;
void setup()
{
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  dht.begin();
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Temp Alarm ON");
  delay(2000);
  lcd.clear();
}
void loop()
{
  float temp = dht.readTemperature();
  if (isnan(temp))
  {
    lcd.setCursor(0, 0);
    lcd.print("Sensor error ");
    delay(2000);
    return;
  }
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(temp, 1);
  lcd.write(223);
  lcd.print("C ");
  if (temp > alarmTemp)
  {
    lcd.setCursor(0, 1);
    lcd.print("** TOO HOT! ** ");
    for (int i = 0; i < 4; i++)
    {
      digitalWrite(buzzer, HIGH);
      digitalWrite(led, HIGH);
      delay(150);
      digitalWrite(buzzer, LOW);
      digitalWrite(led, LOW);
      delay(150);
    }
  }
  else
  {
    lcd.setCursor(0, 1);
    lcd.print("Status: Normal ");
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(1500);
  }
}
```

What you should see: Temp: 28.5°C / Status: Normal in an ordinary room. Hold the sensor in a closed hand for a few seconds. Above 32°C: ** TOO HOT! ** with a pulsing buzzer and flashing LED.

RFID Door Lock

Goal: Scan an RFID card and let a servo motor open the lock only for cards whose UID you have authorised.

The reader talks to the ESP32 over SPI, a four-wire bus that shares MOSI, MISO and SCK between devices and uses a separate SS pin to pick which one is listening. Every card carries a factory-programmed UID, a unique serial number. The sketch turns that UID into text and compares it with an allowed value; matching UIDs is the whole of the security here. Run the sketch once, scan your card, read the printed UID from the Serial Monitor, then paste it into allowedUID and upload again.

Online page with troubleshooting: geekobot.in/projects/titan-kit-025-rfid-door-lock

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- SPI communication and shared buses
- UID matching as a form of authentication
- Driving a servo to a fixed angle
- Building a system from three modules at once

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
Servo	Brown / black wire	**GND**	
Servo	Red wire	**5V**	
Servo	Orange / yellow wire	**GPIO26**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**GPIO21**	
LCD	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define SS_PIN 5
#define RST_PIN 32
MFRC522 rfid(SS_PIN, RST_PIN);
Servo lock;
LiquidCrystal_I2C lcd(0x27, 16, 2);
String allowedUID = "A1 B2 C3 D4"; // Replace with your own card
void setup()
{
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  lock.attach(26);
  lock.write(0); // Locked position
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Scan your card");
}
void loop()
{
  if (!rfid.PICC_IsNewCardPresent())
  {
    return;
  }
  if (!rfid.PICC_ReadCardSerial())
  {
    return;
  }
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10)
    {
      uid = uid + "0";
    }
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1)
    {
      uid = uid + " ";
    }
  }
  uid.toUpperCase();
  Serial.print("Card UID: ");
  Serial.println(uid);
  lcd.clear();
  if (uid == allowedUID)
  {
    lcd.setCursor(0, 0);
    lcd.print("ACCESS GRANTED");
    lock.write(90); // Unlocked
    delay(3000);
    lock.write(0); // Locked again
  }
  else
  {
    lcd.setCursor(0, 0);
    lcd.print("ACCESS DENIED");
    delay(2000);
  }
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Scan your card");
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: LCD shows Scan your card. Serial Monitor prints Card UID: A1 B2 C3 the second button whenever a card is tapped. Authorised card: ACCESS GRANTED and the servo turns to 90°,

returning after 3 seconds. Any other card: ACCESS DENIED and the servo does not move.

Password Door Lock

Goal: Enter a PIN on a 4×4 keypad and let a servo open the door only when the code is correct.

A matrix keypad has sixteen buttons but only eight wires. The library drives one row LOW at a time and reads all four columns, where a LOW appears, that button is down. This is keypad matrix scanning, the same trick the LED matrix used in reverse. Each key press is added to a String; pressing # compares that String with the stored password, and * clears it so a mistake can be retyped.

Online page with troubleshooting: geekobot.in/projects/titan-kit-026-password-door-lock

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x Membrane Matrix Keypad
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Keypad matrix scanning
- String comparison for authentication
- Masking an entry with * characters
- Using * and # as control keys

Wiring

Part	Pin	ESP32	Note
Keypad	Pin 1 (Row 1)	**GPIO32**	
Keypad	Pin 2 (Row 2)	**GPIO33**	
Keypad	Pin 3 (Row 3)	**GPIO25**	
Keypad	Pin 4 (Row 4)	**GPIO26**	
Keypad	Pin 5 (Col 1)	**GPIO27**	
Keypad	Pin 6 (Col 2)	**GPIO14**	
Keypad	Pin 7 (Col 3)	**GPIO12**	GPIO12 is a boot pin: if the board will not start, unplug this wire while it powers up
Keypad	Pin 8 (Col 4)	**GPIO13**	
Servo	Servo signal (orange)	**GPIO23**	
Servo	Servo red	**5V**	
Servo	Servo brown	**GND**	
LCD	LCD SDA	**GPIO21**	
LCD	LCD SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Keypad.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] =
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {32, 33, 25, 26};
byte colPins[COLS] = {27, 14, 12, 13};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
Servo lock;
LiquidCrystal_I2C lcd(0x27, 16, 2);
String password = "1234";
String entered = "";
void askForCode()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Enter PIN:");
  entered = "";
}
void setup()
{
  lock.attach(23);
  lock.write(0);
  lcd.init();
  lcd.backlight();
  askForCode();
}
void loop()
{
  char key = keypad.getKey();
  if (key == NO_KEY)
  {
    return;
  }
  if (key == '*')
  {
    askForCode();
    return;
  }
  if (key == '#')
  {
    lcd.clear();
    lcd.setCursor(0, 0);
    if (entered == password)
    {
      lcd.print("ACCESS GRANTED");
      lock.write(90);
      delay(3000);
      lock.write(0);
    }
    else
    {
      lcd.print("WRONG PIN");
      delay(2000);
    }
    askForCode();
    return;
  }
  entered = entered + key;
  lcd.setCursor(0, 1);
  for (int i = 0; i < entered.length(); i++)
  {
    lcd.print("*");
  }
}
```

What you should see: LCD shows Enter PIN:. Each key press adds a * on the second line. Press # after 1234: ACCESS GRANTED and the servo turns for 3 seconds. Any other code: WRONG PIN.

Bluetooth Door Lock

Goal: Unlock the servo latch from a phone by sending a single character over a Bluetooth link.

Bluetooth here is nothing more exotic than a wireless serial cable. The HC-05 receives characters from a phone app and hands them to the ESP32 over UART, the same one-byte-at-a-time protocol the Serial Monitor uses. The ESP32 has three hardware UARTs, so the sketch simply claims UART2 on GPIO16 and GPIO17 and leaves UART0 free for the USB Serial Monitor. That is a real peripheral rather than the bit-banged SoftwareSerial an Uno would need.

Online page with troubleshooting: geekobot.in/projects/titan-kit-027-bluetooth-door-lock

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- UART and Serial communication
- Hardware UART2 on GPIO16 and GPIO17
- Parsing single-character commands
- Two-way messaging with a phone

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
Servo	Servo signal	**GPIO26**	
Servo	Servo red	**5V**	
Servo	Servo brown	**GND**	
LCD	LCD SDA	**GPIO21**	
LCD	LCD SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17 // RX, TX
Servo lock;
LiquidCrystal_I2C lcd(0x27, 16, 2);
bool locked = true;
void setup()
{
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  lock.attach(26);
  lock.write(0);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Status: LOCKED");
}
void loop()
{
  if (!bt.available())
  {
    return;
  }
  char command = bt.read();
  Serial.println(command);
  if (command == 'U' || command == 'u')
  {
    lock.write(90);
    locked = false;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Status: UNLOCKED");
    bt.println("Door unlocked");
  }
  else if (command == 'L' || command == 'l')
  {
    lock.write(0);
    locked = true;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Status: LOCKED");
    bt.println("Door locked");
  }
}
```

What you should see: Pair with the HC-05 (default PIN 1234 or 0000) in any serial terminal app. Send U: the servo turns and the LCD reads Status: UNLOCKED. Send L: the servo returns and the LCD reads Status: LOCKED. The phone receives a confirmation message back.

Motion Detection Alarm

Goal: Use the ultrasonic sensor as a tripwire: if the distance in front of it changes suddenly, sound the alarm.

A plain distance threshold would fire at anything closer than, say, 50 cm, including a wall. Instead the sketch records a baseline when it starts and then watches for change. If a new reading differs from the baseline by more than the tolerance, something has moved into the beam. abs() is used so the alarm reacts whether the object came closer or moved away.

Online page with troubleshooting: geekobot.in/projects/titan-kit-028-motion-detection-alarm

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Ultrasonic distance thresholds
- Calibrating a baseline at startup
- Detecting change rather than absolute value
- Using abs() for two-sided comparison

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
Buzzer	Buzzer (+)	**GPIO33**	
Buzzer	Buzzer (−)	**GND**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int trigPin = 32;
int echoPin = 5;
int buzzer = 33;
int led = 25;
long baseline = 0;
int tolerance = 15; // cm of change that counts as movement
long readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH);
  return duration * 0.034 / 2;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  delay(2000); // Stand clear while it calibrates
  baseline = readDistance();
  Serial.print("Baseline distance: ");
  Serial.println(baseline);
}
void loop()
{
  long distance = readDistance();
  long change = abs(distance - baseline);
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.print(" Change: ");
  Serial.println(change);
  if (change > tolerance)
  {
    for (int i = 0; i < 5; i++)
    {
      digitalWrite(buzzer, HIGH);
      digitalWrite(led, HIGH);
      delay(120);
      digitalWrite(buzzer, LOW);
      digitalWrite(led, LOW);
      delay(120);
    }
  }
  delay(150);
}
```

What you should see: Serial Monitor prints the baseline once at startup. Distance and Change print continuously. Wave your hand across the beam: the buzzer pulses five times and the LED flashes.

Touch Password System

Goal: Authenticate with a secret sequence: touch three capacitive pads in the right order to unlock.

The three sensors are stored in an array, which lets one for loop check all of them however many there are. Each accepted touch is appended to a sequence array and compared against the secret. Because a finger rests on a pad for far longer than one loop pass, the sketch waits for the pad to be released before accepting the next touch, without that, a single tap would register dozens of times.

Online page with troubleshooting: geekobot.in/projects/titan-kit-029-touch-password

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 3 x Capacitive Touch Sensor Module
- 1 x PCB Mounted Active Buzzer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Capacitive touch arrays and logic
- Comparing two arrays element by element
- Waiting for release to debounce a touch
- Storing a secret as data rather than as code

Wiring

Part	Pin	ESP32	Note
Touch module	All VCC	**3V3**	
Touch module	All GND	**GND**	
Touch module	Sensor 1 SIG	**GPIO13**	
Touch module	Sensor 2 SIG	**GPIO12**	GPIO12 is a boot pin: if the board will not start, unplug this wire while it powers up
Touch module	Sensor 3 SIG	**GPIO14**	
LED	Green LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Red LED anode	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Buzzer	Buzzer (+)	**GPIO33**	
Buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int touchPins[3] = {13, 12, 14};
int greenLed = 27;
int redLed = 26;
int buzzer = 33;
int secret[3] = {1, 3, 2}; // Touch sensor 1, then 3, then 2
int entered[3];
int position = 0;
void beep(int ms)
{
  digitalWrite(buzzer, HIGH);
  delay(ms);
  digitalWrite(buzzer, LOW);
}
void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(touchPins[i], INPUT);
  }
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter the touch sequence");
}
void loop()
{
  for (int i = 0; i < 3; i++)
  {
    if (digitalRead(touchPins[i]) == HIGH)
    {
      entered[position] = i + 1;
      position = position + 1;
      Serial.print("Touched sensor ");
      Serial.println(i + 1);
      beep(60);
      // Wait until the finger is lifted
      while (digitalRead(touchPins[i]) == HIGH)
      {
        delay(10);
      }
    }
  }
  if (position == 3)
  {
    bool correct = true;
    for (int i = 0; i < 3; i++)
    {
      if (entered[i] != secret[i])
      {
        correct = false;
      }
    }
    if (correct)
    {
      Serial.println("ACCESS GRANTED");
      digitalWrite(greenLed, HIGH);
      beep(400);
      delay(2000);
      digitalWrite(greenLed, LOW);
    }
    else
    {
      Serial.println("ACCESS DENIED");
      digitalWrite(redLed, HIGH);
      beep(150);
      delay(150);
      beep(150);
      delay(1500);
      digitalWrite(redLed, LOW);
    }
    position = 0;
  }
}
```

What you should see: Serial Monitor prints Touched sensor 1 with a short beep on each pad. Correct order 1-3-2: ACCESS GRANTED, green LED and a long beep. Any wrong order: ACCESS DENIED, red LED and two short beeps.

Smoke Detection Alarm

Goal: Read an MQ2 gas sensor and raise an audible alarm when smoke or combustible gas appears.

Inside the MQ2 is a tin-dioxide element whose resistance falls when certain gases touch it. The module converts that into a voltage on AO, so the reading climbs as the air gets dirtier. The sensor's heater needs about twenty seconds to stabilise after power-up, which is why the sketch shows a warming-up message before it starts judging anything.

Online page with troubleshooting: geekobot.in/projects/titan-kit-030-smoke-detection-alarm

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic MQ-2 Smoke LPG Butane Hydrogen Gas Sensor Module – 5V Analog/Digital Output
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Gas sensor analog reading
- Why chemical sensors need a warm-up period
- Setting an alarm level from observed values
- Displaying a live number alongside a status

Wiring

Part	Pin	ESP32	Note
MQ-2	VCC	**5V**	
MQ-2	GND	**GND**	
MQ-2	AO (analog out)	**GPIO34**	Through a divider: 1k from AO to GPIO34 and 2k from GPIO34 to GND, because the MQ-2 runs on 5V
Buzzer	Buzzer (+)	**GPIO33**	
Buzzer	Buzzer (−)	**GND**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
LCD	LCD SDA	**GPIO21**	
LCD	LCD SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int gasPin = 34;
int buzzer = 33;
int led = 25;
int alarmLevel = 400;
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Warming up...");
  for (int i = 20; i > 0; i--)
  {
    lcd.setCursor(0, 1);
    lcd.print(i);
    lcd.print(" s ");
    delay(1000);
  }
  lcd.clear();
}
void loop()
{
  int gasLevel = analogRead(gasPin);
  Serial.print("Gas level: ");
  Serial.println(gasLevel);
  lcd.setCursor(0, 0);
  lcd.print("Gas: ");
  lcd.print(gasLevel);
  lcd.print(" ");
  lcd.setCursor(0, 1);
  if (gasLevel > alarmLevel)
  {
    lcd.print("!! SMOKE ALARM !!");
    digitalWrite(buzzer, HIGH);
    digitalWrite(led, HIGH);
    delay(200);
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(200);
  }
  else
  {
    lcd.print("Air: Clean ");
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(500);
  }
}
```

What you should see: A 20-second countdown while the sensor heats up. Gas: 180 / Air: Clean in normal room air. Hold a blown-out match near the sensor: the value jumps and !! SMOKE ALARM !! appears with a pulsing buzzer.

Binary Clock using LEDs

Goal: Build a clock that shows the time in binary: five LEDs for the hour and six for the minute, each one a single bit.

Every whole number can be written as a sum of powers of two, and that is all binary is. The hour 13 is 8 + 4 + 1, so the LEDs worth 8, 4 and 1 light and the rest stay dark. bitRead(value, position) pulls out a single bit for you, bitRead(13, 3) returns 1 because the value-8 bit is set. Five bits reach 31, enough for hours 0 to 23; six bits reach 63, enough for minutes 0 to 59.

Online page with troubleshooting: geekobot.in/projects/titan-kit-031-binary-clock-leds

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 11 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 2 x Tactile Push Button Switch
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Binary arithmetic and place value
- Bitwise operations with bitRead()
- Driving many outputs from an array and a loop
- Keeping time with millis() without drifting

Wiring

Part	Pin	ESP32	Note
LED	Hour bit 4 (value 16)	**GPIO13**	Through a 220 ohm resistor
LED	Hour bit 3 (value 8)	**GPIO12**	Through a 220 ohm resistor
LED	Hour bit 2 (value 4)	**GPIO14**	Through a 220 ohm resistor
LED	Hour bit 1 (value 2)	**GPIO27**	Through a 220 ohm resistor
LED	Hour bit 0 (value 1)	**GPIO26**	Through a 220 ohm resistor
LED	Minute bit 5 (value 32)	**GPIO25**	Through a 220 ohm resistor
LED	Minute bit 4 (value 16)	**GPIO33**	Through a 220 ohm resistor
LED	Minute bit 3 (value 8)	**GPIO32**	Through a 220 ohm resistor
LED	Minute bit 2 (value 4)	**GPIO5**	Through a 220 ohm resistor
LED	Minute bit 1 (value 2)	**GPIO23**	Through a 220 ohm resistor
LED	Minute bit 0 (value 1)	**GPIO19**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Button	Set hour	**GPIO16**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Set minute	**GPIO17**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int hourPins[5] = {13, 12, 14, 27, 26}; // bits 4,3,2,1,0
int minutePins[6] = {25, 33, 32, 5, 23, 19};
int setHourBtn = 16;
int setMinuteBtn = 17;
int hours = 12;
int minutes = 0;
int seconds = 0;
unsigned long lastTick = 0;
void showBinary(int value, int pins[], int bits)
{
  for (int i = 0; i < bits; i++)
  {
    int bitPosition = bits - 1 - i;
    digitalWrite(pins[i], bitRead(value, bitPosition));
  }
}
void setup()
{
  for (int i = 0; i < 5; i++)
  {
    pinMode(hourPins[i], OUTPUT);
  }
  for (int i = 0; i < 6; i++)
  {
    pinMode(minutePins[i], OUTPUT);
  }
  pinMode(setHourBtn, INPUT_PULLUP);
  pinMode(setMinuteBtn, INPUT_PULLUP);
  Serial.begin(9600);
  lastTick = millis();
}
void loop()
{
  if (millis() - lastTick >= 1000)
  {
    lastTick = lastTick + 1000;
    seconds = seconds + 1;
    if (seconds >= 60)
    {
      seconds = 0;
      minutes = minutes + 1;
    }
    if (minutes >= 60)
    {
      minutes = 0;
      hours = hours + 1;
    }
    if (hours >= 24)
    {
      hours = 0;
    }
    Serial.print(hours);
    Serial.print(":");
    Serial.println(minutes);
  }
  if (digitalRead(setHourBtn) == LOW)
  {
    hours = (hours + 1) % 24;
    delay(250);
  }
  if (digitalRead(setMinuteBtn) == LOW)
  {
    minutes = (minutes + 1) % 60;
    seconds = 0;
    delay(250);
  }
  showBinary(hours, hourPins, 5);
  showBinary(minutes, minutePins, 6);
}
```

What you should see: At 13:05 the hour LEDs read 01101 and the minute LEDs read 000101. The minute pattern changes once a minute. Press the set buttons to advance the hour or minute. Serial Monitor prints the time in decimal so you can check your reading.

Binary Number Converter Display

Goal: Turn a knob to choose any number from 0 to 255 and watch eight LEDs show its binary form instantly.

There are two ways to find the bits of a number and this sketch shows both. bitRead() is the quick way. The manual way, divide by two, keep the remainder, repeat, is printed alongside it in the Serial Monitor, because that is the algorithm you would use on paper and the one worth understanding. The two always agree, which is a good way to convince yourself the shortcut is doing what you think.

Online page with troubleshooting: geekobot.in/projects/titan-kit-032-binary-number-converter

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 8 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Decimal to binary conversion logic
- The divide-by-two remainder algorithm
- bitRead() as a shortcut for the same result
- Printing in a chosen number base with Serial.print(value, BIN)

Wiring

Part	Pin	ESP32	Note
LED	Bit 7 (value 128)	**GPIO13**	Through a 220 ohm resistor
LED	Bit 6 (value 64)	**GPIO12**	Through a 220 ohm resistor
LED	Bit 5 (value 32)	**GPIO14**	Through a 220 ohm resistor
LED	Bit 4 (value 16)	**GPIO27**	Through a 220 ohm resistor
LED	Bit 3 (value 8)	**GPIO26**	Through a 220 ohm resistor
LED	Bit 2 (value 4)	**GPIO25**	Through a 220 ohm resistor
LED	Bit 1 (value 2)	**GPIO33**	Through a 220 ohm resistor
LED	Bit 0 (value 1)	**GPIO32**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Potentiometer	Left pin	**GND**	
Potentiometer	Middle pin (wiper)	**GPIO34**	
Potentiometer	Right pin	**3V3**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int ledPins[8] = {13, 12, 14, 27, 26, 25, 33, 32};
int potPin = 34;
int lastValue = -1;
String toBinaryManually(int value)
{
  String result = "";
  for (int i = 0; i < 8; i++)
  {
    result = String(value % 2) + result; // Remainder is the next bit
    value = value / 2; // Then halve and repeat
  }
  return result;
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  for (int i = 0; i < 8; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  Serial.begin(9600);
}
void loop()
{
  int value = map(analogRead(potPin), 0, 1023, 0, 255);
  for (int i = 0; i < 8; i++)
  {
    int bitPosition = 7 - i;
    digitalWrite(ledPins[i], bitRead(value, bitPosition));
  }
  if (value != lastValue)
  {
    lastValue = value;
    Serial.print("Decimal ");
    Serial.print(value);
    Serial.print(" = binary ");
    Serial.print(toBinaryManually(value));
    Serial.print(" (library check: ");
    Serial.print(value, BIN);
    Serial.println(")");
  }
  delay(60);
}
```

What you should see: Turn the knob to 5 and only the value-4 and value-1 LEDs light. Turn it to 255 and every LED lights. Serial Monitor prints Decimal 5 = binary 00000101. The manual conversion and the library conversion always match.

Hexadecimal Counter Display

Goal: Count upwards and show each value in decimal, binary and hexadecimal at once, so the three number bases can be compared side by side.

Hexadecimal is base 16, so it needs sixteen digit symbols, 0 to 9 then A to F, where A is ten and F is fifteen. Its real appeal is that one hex digit is exactly four binary bits. Watch the LEDs as the count passes 0F to 10: the right-hand four LEDs roll over from 1111 to 0000 while the next group ticks up by one, which is exactly what the printed hex value does. That is why memory addresses and colour codes are always written in hex.

Online page with troubleshooting: geekobot.in/projects/titan-kit-033-hexadecimal-counter

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 8 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 2 x Tactile Push Button Switch
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Base-16 number systems
- Why one hex digit equals four binary bits
- Extracting nibbles with / 16 and % 16
- Indexing a character array to convert a value to a symbol

Wiring

Part	Pin	ESP32	Note
LED	Bit 7 (value 128)	**GPIO13**	Through a 220 ohm resistor
LED	Bit 6 (value 64)	**GPIO12**	Through a 220 ohm resistor
LED	Bit 5 (value 32)	**GPIO14**	Through a 220 ohm resistor
LED	Bit 4 (value 16)	**GPIO27**	Through a 220 ohm resistor
LED	Bit 3 (value 8)	**GPIO26**	Through a 220 ohm resistor
LED	Bit 2 (value 4)	**GPIO25**	Through a 220 ohm resistor
LED	Bit 1 (value 2)	**GPIO33**	Through a 220 ohm resistor
LED	Bit 0 (value 1)	**GPIO32**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Button	Step (count up)	**GPIO23**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Reset to zero	**GPIO19**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Hold step to auto-count	—	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int ledPins[8] = {13, 12, 14, 27, 26, 25, 33, 32};
int stepButton = 23;
int resetButton = 19;
int count = 0;
String toHex(int value)
{
  char digits[] = "0123456789ABCDEF";
  String result = "";
  result = result + digits[(value / 16) % 16]; // High nibble
  result = result + digits[value % 16]; // Low nibble
  return result;
}
String toBinary(int value)
{
  String result = "";
  for (int i = 7; i >= 0; i--)
  {
    result = result + String(bitRead(value, i));
    if (i == 4)
    {
      result = result + " "; // Space between the two nibbles
    }
  }
  return result;
}
void showCount()
{
  for (int i = 0; i < 8; i++)
  {
    digitalWrite(ledPins[i], bitRead(count, 7 - i));
  }
  Serial.print("DEC ");
  Serial.print(count);
  Serial.print("\tHEX 0x");
  Serial.print(toHex(count));
  Serial.print("\tBIN ");
  Serial.println(toBinary(count));
}
void setup()
{
  for (int i = 0; i < 8; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  pinMode(stepButton, INPUT_PULLUP);
  pinMode(resetButton, INPUT_PULLUP);
  Serial.begin(9600);
  showCount();
}
void loop()
{
  if (digitalRead(resetButton) == LOW)
  {
    count = 0;
    showCount();
    delay(300);
  }
  if (digitalRead(stepButton) == LOW)
  {
    count = (count + 1) % 256;
    showCount();
    delay(350); // First step is deliberate
    while (digitalRead(stepButton) == LOW)
    {
      count = (count + 1) % 256; // Held down - count quickly
      showCount();
      delay(120);
    }
  }
}
```

What you should see: Press step and the Serial Monitor prints DEC 1 HEX 0x01 BIN 0000 0001. Hold the button and the count runs upward. At 15 the display reads HEX 0x0F BIN 0000 1111. One more step

gives HEX 0x10 BIN 0001 0000, the low nibble rolls over.

Morse Code Message Generator

Goal: Type a message in the Serial Monitor and have the board flash and beep it out in correct International Morse code.

Morse timing is defined entirely in terms of one base unit. A dot is one unit, a dash is three, the gap inside a letter is one, between letters three, and between words seven. Because every duration is a multiple of that single unit, changing one variable changes the whole sending speed, that is why unit is the only timing number in the sketch. The letters themselves live in a lookup table indexed by position in the alphabet, so finding a character's pattern is simple subtraction.

Online page with troubleshooting: geekobot.in/projects/titan-kit-034-morse-code-message-generator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x ESP32 38-Pin Expansion Board

What you learn

- String to timing conversion
- Morse timing as multiples of one base unit
- Lookup tables indexed by character arithmetic
- Reading a whole line from Serial with readStringUntil()

Wiring

Part	Pin	ESP32	Note
LED	LED anode	**GPIO33**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO32**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int led = 33;
int buzzer = 32;
int unit = 120; // Length of one dot in milliseconds
int toneHz = 700;
// A-Z then 0-9
const char* morse[36] =
{
  ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....",
  "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.",
  "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-",
  "-.--", "--..",
  "-----", ".----", "..---", "...--", "....-",
  ".....", "-....", "--...", "---..", "----."
};
void signal(int lengthUnits)
{
  digitalWrite(led, HIGH);
  tone(buzzer, toneHz);
  delay(unit * lengthUnits);
  digitalWrite(led, LOW);
  noTone(buzzer);
  delay(unit); // Gap inside a letter
}
void sendCharacter(char c)
{
  c = toupper(c);
  if (c == ' ')
  {
    delay(unit * 7); // Word gap
    Serial.print(" / ");
    return;
  }
  int index = -1;
  if (c >= 'A' && c <= 'Z') index = c - 'A';
  else if (c >= '0' && c <= '9') index = 26 + (c - '0');
  if (index < 0)
  {
    return; // Ignore anything we cannot send
  }
  String pattern = morse[index];
  Serial.print(pattern);
  Serial.print(" ");
  for (int i = 0; i < pattern.length(); i++)
  {
    if (pattern.charAt(i) == '.')
    {
      signal(1);
    }
    else
    {
      signal(3);
    }
  }
  delay(unit * 2); // Letter gap (1 already spent above)
}
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  Serial.println("Type a message and press Enter");
}
void loop()
{
  if (!Serial.available())
  {
    return;
  }
  String message = Serial.readStringUntil('\n');
  message.trim();
  if (message.length() == 0)
  {
    return;
  }
```

```
  Serial.print("Sending: ");
  Serial.println(message);
  for (int i = 0; i < message.length(); i++)
  {
    sendCharacter(message.charAt(i));
  }
  Serial.println();
  Serial.println("Message sent");
}
```

What you should see: Type SOS and press Enter. The LED flashes three short, three long, three short with matching tones. Serial Monitor echoes... ---... as it sends. Spaces between words produce a longer silent gap.

Morse Code Decoder

Goal: Tap a button in Morse and watch the board work out which letters you sent, purely from how long you held it.

Decoding is the generator run backwards, and it is harder, because the board has to judge intent from timing alone. Two stopwatches run: one measures how long the key is held, the other how long it has been idle. A press shorter than the dot-dash boundary is a dot, longer is a dash. A silence longer than the letter gap means the symbol is finished, so the collected dots and dashes are looked up; a longer silence still means the word has ended. Everything hinges on choosing those two boundaries sensibly.

Online page with troubleshooting: geekobot.in/projects/titan-kit-035-morse-code-decoder

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Timing to string decoding
- Measuring press and release durations separately
- Choosing thresholds that separate intent
- Searching a lookup table for a match

Wiring

Part	Pin	ESP32	Note
Button	Button one leg	**GPIO13**	
Button	Button other leg	**GND**	
LED	LED anode	**GPIO33**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO32**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int keyPin = 13;
int led = 33;
int buzzer = 32;
int unit = 200; // Your dot length in ms
int dashThreshold = unit * 2; // Longer than this is a dash
int letterGap = unit * 3;
int wordGap = unit * 7;
const char* morse[36] =
{
  ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....",
  "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.",
  "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-",
  "-.--", "--..",
  "-----", ".----", "..---", "...--", "....-",
  ".....", "-....", "--...", "---..", "----."
};
const char letters[37] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
String symbol = "";
unsigned long pressStart = 0;
unsigned long releaseStart = 0;
bool wasDown = false;
bool wordEnded = true;
char decode(String pattern)
{
  for (int i = 0; i < 36; i++)
  {
    if (pattern == morse[i])
    {
      return letters[i];
    }
  }
  return '?';
}
void setup()
{
  pinMode(keyPin, INPUT_PULLUP);
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  Serial.println("Tap the key in Morse");
  releaseStart = millis();
}
void loop()
{
  bool isDown = (digitalRead(keyPin) == LOW);
  if (isDown && !wasDown)
  {
    pressStart = millis();
    digitalWrite(led, HIGH);
    tone(buzzer, 700);
    wasDown = true;
  }
  if (!isDown && wasDown)
  {
    unsigned long held = millis() - pressStart;
    digitalWrite(led, LOW);
    noTone(buzzer);
    if (held < dashThreshold)
    {
      symbol = symbol + ".";
    }
    else
    {
      symbol = symbol + "-";
    }
    releaseStart = millis();
    wasDown = false;
    wordEnded = false;
  }
  if (!isDown && symbol.length() > 0 && millis() - releaseStart > letterGap)
  {
    char decoded = decode(symbol);
    Serial.print(decoded);
    if (decoded == '?')
```

```
    {
      Serial.print("(");
      Serial.print(symbol);
      Serial.print(")");
    }
    symbol = "";
  }
  if (!isDown && !wordEnded && symbol.length() == 0 &&
  millis() - releaseStart > wordGap)
  {
    Serial.print(" ");
    wordEnded = true;
  }
}
```

What you should see: Tap short-short-short and pause: the Serial Monitor prints S. Tap long-long-long and pause: it prints O. A longer pause inserts a space between words. An unrecognised pattern prints ?(..--) so you can see what was captured.

OLED Analog Clock

Goal: Draw a traditional round clock face on the OLED, with hour, minute and second hands that sweep like a real watch.

A clock hand is a line from the centre outwards at a particular angle, so the whole face is trigonometry. Twelve hours cover 360 degrees, giving 30 degrees per hour and 6 degrees per minute. sin() and cos() convert that angle into an x and y offset, but they expect radians, hence the multiplication by PI / 180. The hour hand also advances a little within the hour, that is the minute / 2 term, which is what stops it jumping.

Online page with troubleshooting: geekobot.in/projects/titan-kit-036-oled-analog-clock

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- I2C graphics on an OLED display
- Trigonometry for clock hands with sin() and cos()
- Converting degrees to radians
- Redrawing a whole frame each tick with clearDisplay()

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Set hour	**GPIO13**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Set minute	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
int centreX = 64;
int centreY = 32;
int radius = 30;
int hours = 10;
int minutes = 10;
int seconds = 0;
unsigned long lastTick = 0;
int setHour = 13;
int setMinute = 12;
void drawHand(float degrees, int length, int thickness)
{
  float radians = (degrees - 90) * PI / 180.0;
  int x = centreX + cos(radians) * length;
  int y = centreY + sin(radians) * length;
  display.drawLine(centreX, centreY, x, y, SSD1306_WHITE);
  if (thickness > 1)
  {
    display.drawLine(centreX + 1, centreY, x, y, SSD1306_WHITE);
    display.drawLine(centreX, centreY + 1, x, y, SSD1306_WHITE);
  }
}
void drawFace()
{
  display.drawCircle(centreX, centreY, radius + 2, SSD1306_WHITE);
  for (int mark = 0; mark < 12; mark++)
  {
    float radians = (mark * 30 - 90) * PI / 180.0;
    int xOuter = centreX + cos(radians) * radius;
    int yOuter = centreY + sin(radians) * radius;
    int xInner = centreX + cos(radians) * (radius - 4);
    int yInner = centreY + sin(radians) * (radius - 4);
    display.drawLine(xInner, yInner, xOuter, yOuter, SSD1306_WHITE);
  }
}
void setup()
{
  pinMode(setHour, INPUT_PULLUP);
  pinMode(setMinute, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.clearDisplay();
  display.display();
  lastTick = millis();
}
void loop()
{
  if (millis() - lastTick >= 1000)
  {
    lastTick = lastTick + 1000;
    seconds = seconds + 1;
    if (seconds >= 60) { seconds = 0; minutes = minutes + 1; }
    if (minutes >= 60) { minutes = 0; hours = hours + 1; }
    if (hours >= 12) { hours = 0; }
  }
  if (digitalRead(setHour) == LOW) { hours = (hours + 1) % 12; delay(250); }
  if (digitalRead(setMinute) == LOW) { minutes = (minutes + 1) % 60; seconds = 0; delay(250); }
  display.clearDisplay();
  drawFace();
  drawHand(hours * 30 + minutes / 2, radius - 14, 2); // Hour hand
  drawHand(minutes * 6, radius - 7, 2); // Minute hand
  drawHand(seconds * 6, radius - 4, 1); // Second hand
  display.fillCircle(centreX, centreY, 2, SSD1306_WHITE);
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
```

```
  display.setCursor(0, 56);
  display.print(hours);
  display.print(":");
  if (minutes < 10) display.print("0");
  display.print(minutes);
  display.display();
  delay(50);
}
```

What you should see: A circular clock face appears with twelve tick marks. The second hand steps round once per second. Hour and minute hands sit at the correct angles. The digital time is printed in the bottom corner as a cross-check.

OLED Compass Simulation

Goal: Draw a compass dial that rotates smoothly, with the cardinal points moving around a fixed needle.

The potentiometer stands in for a magnetometer, which keeps the focus on the graphics: every point on the dial is rotated by the current heading before it is drawn. That is graphical rotation, and the two lines that do it, an angle plus an offset, fed through sin and cos, are the same two lines that rotate anything on any screen. A smoothing step blends each new heading with the last so the dial glides instead of jittering, exactly as real sensor fusion does.

Online page with troubleshooting: geekobot.in/projects/titan-kit-037-oled-compass-simulation

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Sensor fusion and graphical rotation
- Rotating a set of points around a centre
- Exponential smoothing of a noisy reading
- Converting a continuous angle into named sectors

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Potentiometer	Left pin	**GND**	
Potentiometer	Middle pin (wiper)	**GPIO34**	
Potentiometer	Right pin	**3V3**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
int centreX = 64;
int centreY = 32;
int radius = 26;
float heading = 0;
float smoothed = 0;
const char* points[4] = {"N", "E", "S", "W"};
void drawDial(float rotation)
{
  display.drawCircle(centreX, centreY, radius + 4, SSD1306_WHITE);
  for (int i = 0; i < 4; i++)
  {
    float angle = (i * 90 - rotation - 90) * PI / 180.0;
    int x = centreX + cos(angle) * radius - 2;
    int y = centreY + sin(angle) * radius - 3;
    display.setCursor(x, y);
    display.print(points[i]);
  }
  for (int i = 0; i < 12; i++)
  {
    float angle = (i * 30 - rotation - 90) * PI / 180.0;
    int xo = centreX + cos(angle) * (radius + 3);
    int yo = centreY + sin(angle) * (radius + 3);
    int xi = centreX + cos(angle) * (radius);
    int yi = centreY + sin(angle) * (radius);
    display.drawLine(xi, yi, xo, yo, SSD1306_WHITE);
  }
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  heading = map(analogRead(34), 0, 1023, 0, 359);
  // Smooth the reading, as a real compass would
  smoothed = smoothed * 0.85 + heading * 0.15;
  display.clearDisplay();
  drawDial(smoothed);
  // Fixed needle pointing up - the dial moves, not the needle
  display.drawLine(centreX, centreY, centreX, centreY - radius + 6, SSD1306_WHITE);
  display.fillTriangle(centreX - 3, centreY - radius + 10,
  centreX + 3, centreY - radius + 10,
  centreX, centreY - radius + 2, SSD1306_WHITE);
  display.fillCircle(centreX, centreY, 2, SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print((int)smoothed);
  display.print((char)247);
  display.print(" ");
  int sector = ((int)((smoothed + 22.5) / 45)) % 8;
  const char* names[8] = {"N", "NE", "E", "SE", "S", "SW", "W", "NW"};
  display.print(names[sector]);
  display.display();
  delay(40);
}
```

What you should see: A compass dial with N, E, S and W around the edge. Turning the knob rotates the whole dial while the needle stays fixed. The heading in degrees and the nearest compass point print in the corner. The dial glides rather than jumping, thanks to the smoothing.

OLED Battery Level Indicator

Goal: Draw a battery symbol that fills and empties, with an animated charging bolt and a low-battery warning.

A progress bar is two rectangles: an outline that never changes and a filled one whose width is the percentage scaled to the bar's inner width. That single map() call is the whole trick, and it works for battery meters, loading bars, volume sliders and level gauges alike. The charging animation is a second idea, a counter that advances every frame and wraps with %, so the fill repeatedly sweeps across regardless of the actual level.

Online page with troubleshooting: geekobot.in/projects/titan-kit-038-oled-battery-level-indicator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Graphical progress bars
- Mapping a percentage onto a pixel width
- Inverted text over a filled shape
- Simple frame-counter animation with %

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Potentiometer	Potentiometer wiper	**GPIO34**	
Potentiometer	Potentiometer ends	**3V3 and GND**	
Button	Charging button	**GPIO13**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
int chargeButton = 13;
int buzzer = 33;
bool charging = false;
int animation = 0;
unsigned long lastBeep = 0;
void drawBattery(int percent)
{
  int x = 24;
  int y = 18;
  int w = 80;
  int h = 30;
  display.drawRect(x, y, w, h, SSD1306_WHITE);
  display.fillRect(x + w, y + 9, 4, 12, SSD1306_WHITE); // Terminal nub
  int innerWidth = w - 6;
  int fill = map(percent, 0, 100, 0, innerWidth);
  if (charging)
  {
    // Sweep the fill across to show charging
    fill = map(animation, 0, 100, 0, innerWidth);
  }
  if (fill > 0)
  {
    display.fillRect(x + 3, y + 3, fill, h - 6, SSD1306_WHITE);
  }
  // Percentage text, inverted where it overlaps the fill
  display.setTextSize(2);
  display.setCursor(x + 18, y + 8);
  display.setTextColor(SSD1306_INVERSE);
  display.print(percent);
  display.print("%");
  display.setTextColor(SSD1306_WHITE);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(chargeButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
}
void loop()
{
  int percent = map(analogRead(34), 0, 1023, 0, 100);
  if (digitalRead(chargeButton) == LOW)
  {
    charging = !charging;
    delay(300);
  }
  animation = (animation + 4) % 101;
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  if (charging)
  {
    display.print("CHARGING");
  }
  else if (percent < 20)
  {
    display.print("LOW BATTERY");
  }
  else
  {
    display.print("BATTERY");
```

```
  }
  drawBattery(percent);
  display.setTextSize(1);
  display.setCursor(0, 56);
  if (percent < 20 && !charging)
  {
    display.print("Please charge now");
    if (millis() - lastBeep > 3000)
    {
      lastBeep = millis();
      tone(buzzer, 1500, 120);
    }
  }
  else
  {
    display.print("Status: OK");
  }
  display.display();
  delay(40);
}
```

What you should see: A battery outline fills as you turn the knob. The percentage sits inside the bar in inverted text. Below 20 percent the header reads LOW BATTERY and the buzzer chirps every 3 seconds. Press the button and the fill sweeps repeatedly to show charging.

OLED Sensor Dashboard

Goal: Fit several live readings onto one screen at once, using a divided layout rather than one value at a time.

The screen is 128 by 64 pixels, and the layout treats it as a title strip plus four quadrants of 64 by 24. Working out those coordinates once, as named constants, is what keeps the drawing code readable, every panel is the same function called with a different corner. The DHT11 is read only every two seconds while the screen redraws continuously, so the display stays smooth even though the sensor is slow.

Online page with troubleshooting: geekobot.in/projects/titan-kit-039-oled-sensor-dashboard

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 10k resistor
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Advanced screen layouts
- Dividing a display into coordinate regions
- Drawing repeated panels with one function
- Refreshing slow and fast sensors at different rates

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
DHT11	DHT11 DATA	**GPIO13**	
DHT11	DHT11 VCC / GND	**3V3 / GND**	
LDR divider	LDR divider output	**GPIO34**	
Potentiometer	Potentiometer wiper	**GPIO35**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
float temp = 0;
float hum = 0;
int light = 0;
int level = 0;
unsigned long lastRead = 0;
void panel(int x, int y, const char* label, String value, const char* unit)
{
  display.drawRect(x, y, 64, 24, SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(x + 4, y + 3);
  display.print(label);
  display.setTextSize(1);
  display.setCursor(x + 4, y + 14);
  display.print(value);
  display.print(unit);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  if (millis() - lastRead > 2000)
  {
    lastRead = millis();
    float t = dht.readTemperature();
    float h = dht.readHumidity();
    if (!isnan(t)) temp = t;
    if (!isnan(h)) hum = h;
    Serial.print("T=");
    Serial.print(temp);
    Serial.print(" H=");
    Serial.println(hum);
  }
  light = map(analogRead(34), 0, 1023, 0, 100);
  level = map(analogRead(35), 0, 1023, 0, 100);
  display.clearDisplay();
  // Title strip
  display.fillRect(0, 0, 128, 14, SSD1306_WHITE);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(28, 3);
  display.print("SENSOR PANEL");
  display.setTextColor(SSD1306_WHITE);
  panel(0, 16, "TEMP", String(temp, 1), "C");
  panel(64, 16, "HUMID", String(hum, 0), "%");
  panel(0, 40, "LIGHT", String(light), "%");
  panel(64, 40, "LEVEL", String(level), "%");
  display.display();
  delay(60);
}
```

What you should see: A white title strip reading SENSOR PANEL. Four bordered panels showing temperature, humidity, light and level. Temperature and humidity refresh every 2 seconds. Light and level follow the LDR and knob immediately.

OLED Animated Loading Screen

Goal: Build a set of loading animations, a spinner, a progress bar and moving dots, and cycle between them.

Animation is nothing but a frame counter. Each pass through loop() increases frame by one, and every shape on screen is drawn from that number, the spinner's angle, the bar's width, which dot is currently large. Because the counter wraps with %, the motion repeats forever without any special handling. Keep the frame delay near 40 ms and you get roughly 25 frames per second, which is where movement stops looking like stepping.

Online page with troubleshooting: geekobot.in/projects/titan-kit-040-oled-animated-loading-screen

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Frame-by-frame animation
- Driving every shape from one frame counter
- Wrapping motion with the % operator
- Choosing a frame rate that looks smooth

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	One leg	**GPIO13**	
Button	Other leg	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
int button = 13;
int style = 0;
int frame = 0;
void spinner()
{
  int cx = 64;
  int cy = 30;
  for (int i = 0; i < 8; i++)
  {
    float angle = (i * 45 - 90) * PI / 180.0;
    int x = cx + cos(angle) * 16;
    int y = cy + sin(angle) * 16;
    // The dot nearest the current frame is largest
    int distance = (i - (frame / 4) % 8 + 8) % 8;
    int size = 4 - distance / 2;
    if (size > 0)
    {
      display.fillCircle(x, y, size, SSD1306_WHITE);
    }
  }
}
void progressBar()
{
  int x = 14;
  int y = 26;
  int w = 100;
  int h = 12;
  display.drawRect(x, y, w, h, SSD1306_WHITE);
  int percent = frame % 101;
  int fill = map(percent, 0, 100, 0, w - 4);
  display.fillRect(x + 2, y + 2, fill, h - 4, SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(56, 44);
  display.print(percent);
  display.print("%");
}
void bouncingDots()
{
  int active = (frame / 6) % 4;
  for (int i = 0; i < 4; i++)
  {
    int x = 40 + i * 16;
    int y = 30;
    if (i == active)
    {
      display.fillCircle(x, y - 6, 5, SSD1306_WHITE);
    }
    else
    {
      display.fillCircle(x, y, 3, SSD1306_WHITE);
    }
  }
}
void setup()
{
  pinMode(button, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  if (digitalRead(button) == LOW)
  {
```

```
    style = (style + 1) % 3;
    frame = 0;
    delay(300);
  }
  frame = frame + 1;
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("LOADING");
  if (style == 0) spinner();
  else if (style == 1) progressBar();
  else bouncingDots();
  display.setCursor(0, 56);
  display.print("Style ");
  display.print(style + 1);
  display.print(" of 3");
  display.display();
  delay(40);
}
```

What you should see: A ring of dots rotates like a loading spinner. Press the button: a progress bar fills from 0 to 100 and repeats. Press again: four dots bounce in sequence. Press once more to return to the spinner.

Heat Index Calculator

Goal: Work out what the air actually feels like by combining temperature with humidity, using the official heat index formula.

Sweat cools you by evaporating, and humid air slows evaporation down, so 32°C at 80% humidity feels far hotter than 32°C at 30%. The Rothfusz regression captures that with nine terms fitted to real data. It is written in Fahrenheit, so the sketch converts in, computes, and converts back out, which is a useful lesson in itself: use a formula in the units it was designed for rather than trying to re-derive it.

Online page with troubleshooting: geekobot.in/projects/titan-kit-041-heat-index-calculator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Complex maths formulas in code
- Why humidity changes perceived temperature
- Converting units into and out of a formula
- Choosing between a simple and a full model by range

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
float toF(float c) { return c * 9.0 / 5.0 + 32.0; }
float toC(float f) { return (f - 32.0) * 5.0 / 9.0; }
float heatIndexC(float tempC, float humidity)
{
  float T = toF(tempC);
  float R = humidity;
  // Simple formula is accurate enough below about 80 F
  float simple = 0.5 * (T + 61.0 + ((T - 68.0) * 1.2) + (R * 0.094));
  if ((simple + T) / 2 < 80.0)
  {
    return toC(simple);
  }
  float hi = -42.379
  + 2.04901523 * T
  + 10.14333127 * R
  - 0.22475541 * T * R
  - 0.00683783 * T * T
  - 0.05481717 * R * R
  + 0.00122874 * T * T * R
  + 0.00085282 * T * R * R
  - 0.00000199 * T * T * R * R;
  return toC(hi);
}
void setup()
{
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (isnan(t) || isnan(h))
  {
    delay(2000);
    return;
  }
  float hi = heatIndexC(t, h);
  float difference = hi - t;
  Serial.print("Actual ");
  Serial.print(t);
  Serial.print("C Feels like ");
  Serial.println(hi);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("HEAT INDEX");
  display.setCursor(0, 14);
  display.print("Actual: ");
  display.print(t, 1);
  display.print("C ");
  display.print(h, 0);
  display.print("%");
  display.setTextSize(2);
  display.setCursor(0, 28);
  display.print(hi, 1);
  display.print((char)247);
  display.print("C");
  display.setTextSize(1);
  display.setCursor(0, 52);
```

```
  display.print("Feels ");
  if (difference > 0.5)
  {
    display.print(difference, 1);
    display.print("C hotter");
  }
  else
  {
    display.print("close to actual");
  }
  display.display();
  delay(2000);
}
```

What you should see: Actual: 30.0C 70% on the top line. A large 35.2°C reading below it. Feels 5.2C hotter at the bottom. Breathe on the sensor and the gap widens as humidity climbs.

Dew Point Calculator

Goal: Calculate the exact temperature at which the air would start to condense, and warn when surfaces are at risk.

Warm air holds more water vapour than cold air. Cool a parcel of air and eventually it cannot hold what it has, so water condenses, that temperature is the dew point. The Magnus formula computes it using log(), the natural logarithm, which is why this experiment needs floating-point maths rather than integers. The gap between the air temperature and the dew point is the useful number: a small gap means mist, mould and condensation on cold surfaces.

Online page with troubleshooting: geekobot.in/projects/titan-kit-042-dew-point-calculator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Advanced environmental algorithms
- The Magnus formula and natural logarithms
- Why the temperature–dew point spread matters
- Guarding a formula against invalid input

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#include <math.h>
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
float dewPoint(float tempC, float humidity)
{
  // Magnus formula
  float a = 17.27;
  float b = 237.7;
  float alpha = ((a * tempC) / (b + tempC)) + log(humidity / 100.0);
  return (b * alpha) / (a - alpha);
}
void setup()
{
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (isnan(t) || isnan(h) || h <= 0)
  {
    delay(2000);
    return;
  }
  float dp = dewPoint(t, h);
  float spread = t - dp;
  Serial.print("Temp ");
  Serial.print(t);
  Serial.print(" Dew point ");
  Serial.print(dp);
  Serial.print(" Spread ");
  Serial.println(spread);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("DEW POINT");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print(dp, 1);
  display.print((char)247);
  display.print("C");
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print("Air ");
  display.print(t, 1);
  display.print("C RH ");
  display.print(h, 0);
  display.print("%");
  display.setCursor(0, 46);
  display.print("Spread ");
  display.print(spread, 1);
  display.print("C");
  display.setCursor(0, 56);
  if (spread < 2.5)
  {
    display.print("CONDENSATION RISK");
  }
  else if (spread < 5)
  {
    display.print("Humid - watch it");
  }
```

```
  else
  {
    display.print("Comfortable air");
  }
  display.display();
  delay(2000);
}
```

What you should see: A large dew point figure such as 18.4°C. Air 25.0C RH 66% below it. Spread 6.6C and the verdict Comfortable air. Breathe steadily on the sensor: the spread shrinks and CONDENSATION RISK appears.

Weather Comfort Indicator

Goal: Sort the current conditions into named comfort zones by testing temperature and humidity together rather than separately.

A single threshold cannot answer this question: 28°C is pleasant at 40% humidity and unpleasant at 85%. Classification needs both variables at once, so the sketch scores each one separately and then combines the scores. Reaching a verdict by adding up partial judgements, rather than nesting if statements ten deep, keeps the logic readable and makes it easy to add a third variable later.

Online page with troubleshooting: geekobot.in/projects/titan-kit-043-weather-comfort-indicator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Multi-variable logical classification
- Scoring variables separately then combining
- Why single thresholds fail for comfort
- Giving specific advice rather than a bare verdict

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green (comfortable)	**GPIO27**	Through a 220 ohm resistor
LED	Yellow (borderline)	**GPIO26**	Through a 220 ohm resistor
LED	Red (harsh)	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
int greenLed = 27;
int yellowLed = 26;
int redLed = 25;
int scoreTemperature(float t)
{
  if (t >= 20 && t <= 26) return 0; // Ideal
  if (t >= 17 && t <= 29) return 1; // Acceptable
  return 2; // Harsh
}
int scoreHumidity(float h)
{
  if (h >= 40 && h <= 60) return 0;
  if (h >= 30 && h <= 70) return 1;
  return 2;
}
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (isnan(t) || isnan(h))
  {
    delay(2000);
    return;
  }
  int total = scoreTemperature(t) + scoreHumidity(h);
  String verdict;
  String advice;
  if (total == 0)
  {
    verdict = "COMFORTABLE";
    advice = "Ideal conditions";
    digitalWrite(greenLed, HIGH);
    digitalWrite(yellowLed, LOW);
    digitalWrite(redLed, LOW);
  }
  else if (total <= 2)
  {
    verdict = "ACCEPTABLE";
    advice = "Slightly off ideal";
    digitalWrite(greenLed, LOW);
    digitalWrite(yellowLed, HIGH);
    digitalWrite(redLed, LOW);
  }
  else
  {
    verdict = "HARSH";
    if (t > 29 && h > 70) advice = "Hot and humid";
    else if (t > 29) advice = "Too hot and dry";
    else if (t < 17) advice = "Too cold";
    else if (h > 70) advice = "Very humid";
    else advice = "Very dry air";
    digitalWrite(greenLed, LOW);
```

```
    digitalWrite(yellowLed, LOW);
    digitalWrite(redLed, HIGH);
  }
  Serial.print(verdict);
  Serial.print(" - ");
  Serial.println(advice);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print(t, 1);
  display.print("C ");
  display.print(h, 0);
  display.print("% RH");
  display.setTextSize(2);
  display.setCursor(0, 18);
  display.print(verdict);
  display.setTextSize(1);
  display.setCursor(0, 42);
  display.print(advice);
  display.setCursor(0, 54);
  display.print("Score: ");
  display.print(total);
  display.print(" of 4");
  display.display();
  delay(2000);
}
```

What you should see: 23.0C 50% RH with a large COMFORTABLE and the green LED lit. Warm the sensor by hand: ACCEPTABLE appears with the yellow LED. Breathe on it steadily: HARSH / Hot and humid with the red LED. The score line shows how the verdict was reached.

Temperature Trend Tracker

Goal: Store a rolling history of temperature readings and draw them as a live graph on the OLED.

The history array holds one reading per screen column. When it fills up, every value shifts one place left and the newest lands at the end, the oldest simply falls off. The graph then auto-scales: it finds the smallest and largest values currently stored and maps that range onto the plot height, so a half-degree wobble fills the screen just as a ten-degree swing would. Auto-scaling is what makes a small graph readable.

Online page with troubleshooting: geekobot.in/projects/titan-kit-044-temperature-trend-tracker

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Data logging into arrays
- Shifting an array to make a rolling window
- Auto-scaling a graph to its own data
- Detecting a trend by comparing across samples

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
#define POINTS 100
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
float history[POINTS];
int stored = 0;
unsigned long lastSample = 0;
unsigned long interval = 3000;
void addReading(float value)
{
  if (stored < POINTS)
  {
    history[stored] = value;
    stored = stored + 1;
    return;
  }
  for (int i = 0; i < POINTS - 1; i++) // Shift everything left
  {
    history[i] = history[i + 1];
  }
  history[POINTS - 1] = value;
}
void drawGraph()
{
  if (stored < 2)
  {
    return;
  }
  float lowest = history[0];
  float highest = history[0];
  for (int i = 1; i < stored; i++)
  {
    if (history[i] < lowest) lowest = history[i];
    if (history[i] > highest) highest = history[i];
  }
  if (highest - lowest < 1.0) // Never squash a flat line
  {
    highest = highest + 0.5;
    lowest = lowest - 0.5;
  }
  int plotTop = 20;
  int plotBottom = 60;
  display.drawLine(24, plotTop, 24, plotBottom, SSD1306_WHITE);
  display.drawLine(24, plotBottom, 127, plotBottom, SSD1306_WHITE);
  display.setCursor(0, plotTop - 3);
  display.print(highest, 1);
  display.setCursor(0, plotBottom - 6);
  display.print(lowest, 1);
  for (int i = 1; i < stored; i++)
  {
    int x1 = 25 + (i - 1);
    int x2 = 25 + i;
    int y1 = map(history[i - 1] * 10, lowest * 10, highest * 10, plotBottom, plotTop);
    int y2 = map(history[i] * 10, lowest * 10, highest * 10, plotBottom, plotTop);
    display.drawLine(x1, y1, x2, y2, SSD1306_WHITE);
  }
}
void setup()
{
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
```

```
void loop()
{
  if (millis() - lastSample > interval)
  {
    lastSample = millis();
    float t = dht.readTemperature();
    if (!isnan(t))
    {
      addReading(t);
      Serial.println(t);
    }
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("TEMP TREND");
  if (stored > 0)
  {
    display.setCursor(70, 0);
    display.print(history[stored - 1], 1);
    display.print("C");
    display.setCursor(0, 10);
    if (stored > 4)
    {
      float change = history[stored - 1] - history[stored - 5];
      if (change > 0.2) display.print("Rising");
      else if (change < -0.2) display.print("Falling");
      else display.print("Steady");
    }
    display.setCursor(70, 10);
    display.print(stored);
    display.print(" points");
  }
  drawGraph();
  display.display();
  delay(200);
}
```

What you should see: A line graph builds up from the left, one point every 3 seconds. The axis labels show the current highest and lowest values. The header reads Rising, Falling or Steady. After 100 points the graph scrolls, dropping the oldest reading.

Humidity Trend Tracker

Goal: Track humidity over time with the same rolling history, adding a running average and a record of the extremes.

Three different summaries of the same data appear here, and each answers a different question. The instant reading says what is happening now. The average over the stored window says what has been typical. The all-time minimum and maximum say what the extremes were, and they survive even after those readings have scrolled off the graph, which is why they are kept in separate variables rather than being recalculated from the array.

Online page with troubleshooting: geekobot.in/projects/titan-kit-045-humidity-trend-tracker

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Historical data tracking
- Running averages across an array
- Keeping records separate from the rolling window
- Bar charts as an alternative to line graphs

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	One leg	**GPIO12**	GPIO12 is a boot pin: if the board will not start, unplug this wire while it powers up
Button	Other leg	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
#define POINTS 100
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
int resetButton = 12;
float history[POINTS];
int stored = 0;
float recordLow = 100;
float recordHigh = 0;
unsigned long lastSample = 0;
void addReading(float value)
{
  if (value < recordLow) recordLow = value;
  if (value > recordHigh) recordHigh = value;
  if (stored < POINTS)
  {
    history[stored] = value;
    stored = stored + 1;
    return;
  }
  for (int i = 0; i < POINTS - 1; i++)
  {
    history[i] = history[i + 1];
  }
  history[POINTS - 1] = value;
}
float average()
{
  if (stored == 0) return 0;
  float total = 0;
  for (int i = 0; i < stored; i++)
  {
    total = total + history[i];
  }
  return total / stored;
}
void setup()
{
  pinMode(resetButton, INPUT_PULLUP);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  if (digitalRead(resetButton) == LOW)
  {
    stored = 0;
    recordLow = 100;
    recordHigh = 0;
    Serial.println("History cleared");
    delay(400);
  }
  if (millis() - lastSample > 3000)
  {
    lastSample = millis();
    float h = dht.readHumidity();
    if (!isnan(h))
    {
      addReading(h);
      Serial.println(h);
    }
  }
```

```
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("HUMIDITY LOG");
  if (stored > 0)
  {
    display.setCursor(80, 0);
    display.print(history[stored - 1], 0);
    display.print("%");
    display.setCursor(0, 12);
    display.print("Avg ");
    display.print(average(), 1);
    display.print("%");
    display.setCursor(64, 12);
    display.print("L");
    display.print(recordLow, 0);
    display.print(" H");
    display.print(recordHigh, 0);
    // Bar graph of the stored window
    for (int i = 0; i < stored; i++)
    {
      int x = 27 + i;
      int barHeight = map(history[i], 0, 100, 0, 34);
      display.drawLine(x, 60, x, 60 - barHeight, SSD1306_WHITE);
    }
    display.drawLine(26, 26, 26, 60, SSD1306_WHITE);
    display.drawLine(26, 60, 127, 60, SSD1306_WHITE);
    display.setCursor(0, 24);
    display.print("100");
    display.setCursor(6, 54);
    display.print("0");
  }
  else
  {
    display.setCursor(0, 30);
    display.print("Collecting data...");
  }
  display.display();
  delay(200);
}
```

What you should see: A bar chart fills from the left, one bar every 3 seconds. The header shows the latest humidity as a percentage. Avg 58.4% with L42 H71 for the all-time records. Press the button to clear both the history and the records.

Mini Weather Logger

Goal: Record temperature and humidity into EEPROM so the log survives a power cut, then play it back on demand.

RAM forgets everything the moment power is lost; EEPROM does not. The ESP32 emulates 512 bytes of EEPROM in flash of it, so each record here is squeezed into two bytes, one for temperature, one for humidity, giving room for hundreds of samples. EEPROM has a limited write endurance of about 100,000 cycles per cell, which is why the sample interval is measured in seconds rather than milliseconds, and why the write count is worth being deliberate about.

Online page with troubleshooting: geekobot.in/projects/titan-kit-046-mini-weather-logger

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Long-term state storage in EEPROM
- Packing records into fixed byte addresses
- Why EEPROM.update() is preferred over write()
- Detecting an unwritten EEPROM on first run

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Playback log	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Erase log	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#include <EEPROM.h>
#define DHTPIN 13
#define DHTTYPE DHT11
#define MAX_RECORDS 200
#define COUNT_ADDRESS 900
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
int playButton = 12;
int eraseButton = 14;
int recordCount = 0;
unsigned long lastSample = 0;
unsigned long interval = 10000; // One record every 10 s
void saveRecord(int temp, int hum)
{
  if (recordCount >= MAX_RECORDS)
  {
    return;
  }
  int address = recordCount * 2;
  EEPROM.update(address, temp);
  EEPROM.update(address + 1, hum);
  recordCount = recordCount + 1;
  EEPROM.update(COUNT_ADDRESS, recordCount);
  Serial.print("Saved record ");
  Serial.println(recordCount);
}
void playback()
{
  Serial.println("--- Stored log ---");
  for (int i = 0; i < recordCount; i++)
  {
    int temp = EEPROM.read(i * 2);
    int hum = EEPROM.read(i * 2 + 1);
    Serial.print(i + 1);
    Serial.print("\t");
    Serial.print(temp);
    Serial.print(" C\t");
    Serial.print(hum);
    Serial.println(" %");
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.print("PLAYBACK ");
    display.print(i + 1);
    display.print("/");
    display.print(recordCount);
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print(temp);
    display.print("C ");
    display.print(hum);
    display.print("%");
    display.display();
    delay(400);
  }
  Serial.println("--- End of log ---");
}
void setup()
{
  pinMode(playButton, INPUT_PULLUP);
  pinMode(eraseButton, INPUT_PULLUP);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  recordCount = EEPROM.read(COUNT_ADDRESS);
```

```
  if (recordCount > MAX_RECORDS)
  {
    recordCount = 0; // First run - EEPROM is unwritten
  }
  Serial.print("Existing records: ");
  Serial.println(recordCount);
}
void loop()
{
  if (digitalRead(playButton) == LOW)
  {
    playback();
    delay(400);
  }
  if (digitalRead(eraseButton) == LOW)
  {
    recordCount = 0;
    EEPROM.update(COUNT_ADDRESS, 0);
    Serial.println("Log erased");
    delay(400);
  }
  if (millis() - lastSample > interval)
  {
    lastSample = millis();
    float t = dht.readTemperature();
    float h = dht.readHumidity();
    if (!isnan(t) && !isnan(h))
    {
      saveRecord((int)t, (int)h);
    }
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("WEATHER LOGGER");
  display.setTextSize(2);
  display.setCursor(0, 18);
  display.print(recordCount);
  display.setTextSize(1);
  display.print(" / ");
  display.print(MAX_RECORDS);
  display.setCursor(0, 40);
  display.print("Next in ");
  display.print((interval - (millis() - lastSample)) / 1000);
  display.print("s");
  display.setCursor(0, 52);
  display.print("D3 play D4 erase");
  display.display();
  delay(200);
}
```

What you should see: The record count climbs by one every 10 seconds. Unplug and reconnect the board: the count is still there. Press the first button and each stored record shows in turn on the OLED and Serial Monitor. Press the second button to clear the log back to zero.

Thermal Warning Indicator

Goal: Build a tiered alert system where the severity of the response escalates with the temperature.

A single alarm level is a blunt instrument, it either shouts or says nothing. A tiered system gives graded warning, and each tier changes three things at once: the LED, the beep pattern and the wording on screen. Keeping the tier as a number and driving all three outputs from it means a new tier can be added by extending three short lookups rather than rewriting the logic.

Online page with troubleshooting: geekobot.in/projects/titan-kit-047-thermal-warning-indicator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Multi-tier alert systems
- Escalating several outputs from one tier value
- Using pitch and repetition to signal urgency
- Table-driven behaviour instead of nested ifs

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED	**GPIO27**	Through a 220 ohm resistor
LED	Yellow LED	**GPIO26**	Through a 220 ohm resistor
LED	Red LED	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
int ledPins[3] = {27, 26, 25}; // green, yellow, red
int buzzer = 33;
float warnAt = 28.0;
float alertAt = 32.0;
float dangerAt = 36.0;
String tierName[4] = {"NORMAL", "WARNING", "ALERT", "DANGER"};
int beepInterval[4] = {0, 4000, 1500, 500};
unsigned long lastBeep = 0;
int tierFor(float t)
{
  if (t >= dangerAt) return 3;
  if (t >= alertAt) return 2;
  if (t >= warnAt) return 1;
  return 0;
}
void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  float t = dht.readTemperature();
  if (isnan(t))
  {
    delay(2000);
    return;
  }
  int tier = tierFor(t);
  digitalWrite(ledPins[0], tier == 0);
  digitalWrite(ledPins[1], tier == 1);
  digitalWrite(ledPins[2], tier >= 2);
  if (tier > 0 && millis() - lastBeep > beepInterval[tier])
  {
    lastBeep = millis();
    tone(buzzer, 800 + tier * 400, 100 + tier * 50);
  }
  Serial.print(t);
  Serial.print(" C tier ");
  Serial.println(tierName[tier]);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("THERMAL MONITOR");
  display.setTextSize(3);
  display.setCursor(0, 14);
  display.print(t, 1);
  display.setTextSize(1);
  display.setCursor(0, 42);
  display.print("Level: ");
  display.print(tierName[tier]);
  display.setCursor(0, 54);
  if (tier == 0) display.print("All within limits");
  else if (tier == 1) display.print("Above ");
  else if (tier == 2) display.print("Above ");
```

```
  else display.print("Evacuate the area");
  if (tier == 1) { display.print(warnAt, 0); display.print("C"); }
  if (tier == 2) { display.print(alertAt, 0); display.print("C - act now"); }
  if (tier >= 2)
  {
    display.drawRect(0, 10, 128, 40, SSD1306_WHITE);
  }
  display.display();
  delay(500);
}
```

What you should see: Below 28°C: NORMAL with the green LED and no sound. 28–32°C: WARNING, yellow LED, a beep every 4 seconds. 32–36°C: ALERT, red LED, a higher beep every 1.5 seconds and a box around the reading. Above 36°C: DANGER with a rapid urgent beep.

Greenhouse Environment Monitor

Goal: Simulate a growing environment that tracks temperature, humidity and light against a target range and reports what needs adjusting.

This is a control loop with three parallel channels. Each one has an ideal band, and each reports one of three states: too low, in range, or too high. The overall verdict is simply how many channels are in range. Real greenhouse controllers work exactly this way before they act, they establish the error on every channel first, then decide which actuator to move, because heating the air also changes the humidity.

Online page with troubleshooting: geekobot.in/projects/titan-kit-048-greenhouse-environment-monitor

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 10k resistor
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Environmental control loops
- Three-state checks instead of simple booleans
- Prioritising one action from several faults
- Why channels interact in a real controller

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LDR divider	LDR divider output	**GPIO34**	
LED	Temp status LED	**GPIO27**	Through a 220 ohm resistor
LED	Humidity status LED	**GPIO26**	Through a 220 ohm resistor
LED	Light status LED	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
int statusLeds[3] = {27, 26, 25};
float tempMin = 20, tempMax = 28;
float humMin = 50, humMax = 75;
int lightMin = 40, lightMax = 90;
// -1 too low, 0 in range, 1 too high
int check(float value, float low, float high)
{
  if (value < low) return -1;
  if (value > high) return 1;
  return 0;
}
String describe(int state, String name, String lowWord, String highWord)
{
  if (state < 0) return name + " " + lowWord;
  if (state > 0) return name + " " + highWord;
  return "";
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  for (int i = 0; i < 3; i++)
  {
    pinMode(statusLeds[i], OUTPUT);
  }
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  int light = map(analogRead(34), 0, 1023, 0, 100);
  if (isnan(t) || isnan(h))
  {
    delay(2000);
    return;
  }
  int tState = check(t, tempMin, tempMax);
  int hState = check(h, humMin, humMax);
  int lState = check(light, lightMin, lightMax);
  digitalWrite(statusLeds[0], tState == 0);
  digitalWrite(statusLeds[1], hState == 0);
  digitalWrite(statusLeds[2], lState == 0);
  int inRange = (tState == 0) + (hState == 0) + (lState == 0);
  String action = "";
  if (action == "") action = describe(tState, "Heat:", "raise", "vent");
  if (action == "") action = describe(hState, "Mist:", "on", "dehumidify");
  if (action == "") action = describe(lState, "Lamp:", "on", "shade");
  if (action == "") action = "All conditions ideal";
  Serial.print(t); Serial.print(" ");
  Serial.print(h); Serial.print(" ");
  Serial.print(light); Serial.print(" in range: ");
  Serial.println(inRange);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("GREENHOUSE ");
  display.print(inRange);
  display.print("/3 OK");
  display.setCursor(0, 14);
```

```
  display.print("Temp ");
  display.print(t, 1);
  display.print("C ");
  display.print(tState == 0 ? "OK" : (tState < 0 ? "LOW" : "HIGH"));
  display.setCursor(0, 26);
  display.print("Humid ");
  display.print(h, 0);
  display.print("% ");
  display.print(hState == 0 ? "OK" : (hState < 0 ? "LOW" : "HIGH"));
  display.setCursor(0, 38);
  display.print("Light ");
  display.print(light);
  display.print("% ");
  display.print(lState == 0 ? "OK" : (lState < 0 ? "LOW" : "HIGH"));
  display.drawLine(0, 50, 127, 50, SSD1306_WHITE);
  display.setCursor(0, 54);
  display.print(action);
  display.display();
  delay(1000);
}
```

What you should see: Three rows showing temperature, humidity and light each marked OK, LOW or HIGH. The header counts how many are in range, such as 2/3 OK. One LED per channel lights when that channel is within its band. The bottom line names the single most important correction to make.

Room Climate Score System

Goal: Reduce several environmental readings to a single weighted score out of 100, so conditions can be compared at a glance.

Each reading is first scored on its own from 0 to 100, using distance from the ideal value: sitting exactly on target scores 100, and the score falls off as the reading drifts away. Those three scores are then blended with weights that add up to 1.0, temperature counts for half, humidity a third, light the rest. Changing a weight changes what the system considers important without touching any of the scoring code.

Online page with troubleshooting: geekobot.in/projects/titan-kit-049-room-climate-score

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 10k resistor
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Weighted data aggregation
- Scoring by distance from an ideal value
- Why weights must sum to one
- Turning several readings into one comparable figure

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LDR divider	LDR leg 1	**3V3**	
LDR divider	LDR leg 2	**GPIO34 and 10kΩ**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
float idealTemp = 23.0;
float idealHum = 50.0;
int idealLight = 65;
float weightTemp = 0.5;
float weightHum = 0.3;
float weightLight = 0.2;
// 100 at the ideal, falling away by "falloff" points per unit of error
float scoreValue(float value, float ideal, float falloff)
{
  float error = abs(value - ideal);
  float score = 100.0 - error * falloff;
  return constrain(score, 0, 100);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  int light = map(analogRead(34), 0, 1023, 0, 100);
  if (isnan(t) || isnan(h))
  {
    delay(2000);
    return;
  }
  float tScore = scoreValue(t, idealTemp, 8.0);
  float hScore = scoreValue(h, idealHum, 2.5);
  float lScore = scoreValue(light, idealLight, 1.5);
  float total = tScore * weightTemp + hScore * weightHum + lScore * weightLight;
  String grade;
  if (total >= 85) grade = "EXCELLENT";
  else if (total >= 70) grade = "GOOD";
  else if (total >= 50) grade = "FAIR";
  else grade = "POOR";
  Serial.print("T:"); Serial.print(tScore, 0);
  Serial.print(" H:"); Serial.print(hScore, 0);
  Serial.print(" L:"); Serial.print(lScore, 0);
  Serial.print(" Total:"); Serial.println(total, 0);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("CLIMATE SCORE");
  display.setTextSize(3);
  display.setCursor(0, 12);
  display.print((int)total);
  display.setTextSize(1);
  display.setCursor(56, 20);
  display.print("/100");
  display.setCursor(56, 30);
  display.print(grade);
  // Three small score bars
  int scores[3] = {(int)tScore, (int)hScore, (int)lScore};
  const char* labels[3] = {"T", "H", "L"};
  for (int i = 0; i < 3; i++)
  {
    int y = 42 + i * 7;
```

```
    display.setCursor(0, y);
    display.print(labels[i]);
    display.drawRect(10, y, 102, 6, SSD1306_WHITE);
    display.fillRect(11, y + 1, map(scores[i], 0, 100, 0, 100), 4, SSD1306_WHITE);
    display.setCursor(114, y);
    display.print(scores[i] / 10);
  }
  display.display();
  delay(1500);
}
```

What you should see: A large score such as 82 /100 with the grade GOOD. Three horizontal bars showing the temperature, humidity and light sub-scores. Cover the LDR and the L bar shrinks, pulling the total down a little. Breathe on the sensor and the H bar drops further, since humidity is weighted higher.

Temperature-Based Color Visualizer

Goal: Map temperature onto a colour gradient so an RGB LED glows deep blue when cold and red when hot.

The gradient runs blue → cyan → green → yellow → red across the working temperature range, and it is built by splitting that range into four bands and fading between two colours inside each. Because every band is a straight-line blend, the transitions join up smoothly. The same mapping-a-number-onto-a-colour-spectrum idea drives every thermal camera image and every heat map you have ever seen.

Online page with troubleshooting: geekobot.in/projects/titan-kit-050-temperature-color-visualizer

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Mapping variables to colour spectrums
- Building a gradient from linear bands
- Normalising a value to a 0.0–1.0 position
- How thermal images assign colour to data

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
RGB LED	Red	**GPIO32**	Through a 220 ohm resistor
RGB LED	Green	**GPIO5**	Through a 220 ohm resistor
RGB LED	Blue	**GPIO23**	Through a 220 ohm resistor
RGB LED	Common cathode	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 13
#define DHTTYPE DHT11
Adafruit_SSD1306 display(128, 64, &Wire, -1);
DHT dht(DHTPIN, DHTTYPE);
int redPin = 32;
int greenPin = 5;
int bluePin = 23;
float coldEnd = 15.0;
float hotEnd = 40.0;
int r, g, b;
void temperatureToColour(float t)
{
  float position = (t - coldEnd) / (hotEnd - coldEnd);
  position = constrain(position, 0.0, 1.0);
  float band = position * 4.0; // Four bands across the gradient
  if (band < 1) // Blue to cyan
  {
    r = 0;
    g = band * 255;
    b = 255;
  }
  else if (band < 2) // Cyan to green
  {
    r = 0;
    g = 255;
    b = (2 - band) * 255;
  }
  else if (band < 3) // Green to yellow
  {
    r = (band - 2) * 255;
    g = 255;
    b = 0;
  }
  else // Yellow to red
  {
    r = 255;
    g = (4 - band) * 255;
    b = 0;
  }
}
void setup()
{
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  float t = dht.readTemperature();
  if (isnan(t))
  {
    delay(2000);
    return;
  }
  temperatureToColour(t);
  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);
  Serial.print(t);
  Serial.print("C -> R");
  Serial.print(r);
  Serial.print(" G");
```

```
  Serial.print(g);
  Serial.print(" B");
  Serial.println(b);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("COLOUR VISUALISER");
  display.setTextSize(3);
  display.setCursor(0, 14);
  display.print(t, 1);
  display.setTextSize(1);
  display.setCursor(74, 22);
  display.print("deg C");
  display.setCursor(0, 40);
  display.print("R");
  display.print(r);
  display.print(" G");
  display.print(g);
  display.print(" B");
  display.print(b);
  // Position marker along a gradient scale
  display.drawRect(0, 52, 128, 10, SSD1306_WHITE);
  int marker = map(constrain(t * 10, coldEnd * 10, hotEnd * 10),
  coldEnd * 10, hotEnd * 10, 2, 124);
  display.fillRect(marker, 54, 3, 6, SSD1306_WHITE);
  display.setCursor(0, 44);
  display.display();
  delay(1000);
}
```

What you should see: At 18°C the LED glows blue-cyan. At 25°C it settles to green. Warm the sensor in your hand and it moves through yellow towards red. The OLED shows the exact RGB values and a marker sliding along the scale.

Sound Level Bar Graph

Goal: Turn ambient noise into a moving row of LEDs, so the room's volume becomes something you can see.

A microphone signal swings above and below a resting midpoint, so a single analogRead() tells you almost nothing, catch it mid-swing and a loud sound reads as silence. Instead the sketch samples repeatedly for 50 ms and keeps the largest and smallest values seen, then uses the difference between them. That peak-to-peak amplitude is a genuine measure of loudness, and it is the foundation of every sound experiment in this level.

Online page with troubleshooting: geekobot.in/projects/titan-kit-051-sound-level-bar-graph

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 10 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Analog mapping to array outputs
- Peak-to-peak sampling over a time window
- Why a single microphone reading is meaningless
- Calibrating quiet and loud limits for your room

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
LED	LED 1	**GPIO12**	Through a 220 ohm resistor
LED	LED 2	**GPIO14**	Through a 220 ohm resistor
LED	LED 3	**GPIO27**	Through a 220 ohm resistor
LED	LED 4	**GPIO26**	Through a 220 ohm resistor
LED	LED 5	**GPIO25**	Through a 220 ohm resistor
LED	LED 6	**GPIO33**	Through a 220 ohm resistor
LED	LED 7	**GPIO32**	Through a 220 ohm resistor
LED	LED 8	**GPIO5**	Through a 220 ohm resistor
LED	LED 9	**GPIO23**	Through a 220 ohm resistor
LED	LED 10	**GPIO19**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int micPin = 34;
int ledPins[10] = {12, 14, 27, 26, 25, 33, 32, 5, 23, 19};
int sampleWindow = 50; // ms of sampling per reading
int quietLevel = 20; // Peak-to-peak in a silent room
int loudLevel = 400; // Peak-to-peak when shouting
int readAmplitude()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < sampleWindow)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return highest - lowest;
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  for (int i = 0; i < 10; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  Serial.begin(9600);
}
void loop()
{
  int amplitude = readAmplitude();
  int level = map(amplitude, quietLevel, loudLevel, 0, 10);
  level = constrain(level, 0, 10);
  for (int i = 0; i < 10; i++)
  {
    digitalWrite(ledPins[i], i < level);
  }
  Serial.print("Amplitude: ");
  Serial.print(amplitude);
  Serial.print(" Bars: ");
  Serial.println(level);
}
```

What you should see: A quiet room lights no LEDs or one. Normal talking lights three to five. A clap briefly lights all ten. Serial Monitor prints the raw amplitude so you can calibrate.

Noise Pollution Monitor

Goal: Watch the room's noise level over time and raise an alert only when it stays too loud, not for a single shout.

One door slam is not noise pollution; ten minutes of shouting is. The difference is duration, so the sketch does not react to the instantaneous level at all, it reacts to a rolling average of the last twenty readings. A brief spike barely moves that average, while sustained noise drags it steadily upwards until the threshold is crossed. Averaging as a way of ignoring spikes is one of the most reusable ideas in sensing.

Online page with troubleshooting: geekobot.in/projects/titan-kit-052-noise-pollution-monitor

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Sustained threshold detection
- Rolling averages over a circular buffer
- Requiring a condition to persist before acting
- Drawing a threshold marker on a level bar

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Red LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SAMPLES 20
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int led = 25;
int buzzer = 33;
int history[SAMPLES];
int slot = 0;
bool filled = false;
int threshold = 55; // Percent
unsigned long overSince = 0;
bool alerting = false;
unsigned long alertHold = 5000; // Must stay loud this long
int readLevel()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 50)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  int amplitude = highest - lowest;
  return constrain(map(amplitude, 20, 400, 0, 100), 0, 100);
}
int rollingAverage()
{
  int count = filled ? SAMPLES : slot;
  if (count == 0) return 0;
  long total = 0;
  for (int i = 0; i < count; i++)
  {
    total = total + history[i];
  }
  return total / count;
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  int level = readLevel();
  history[slot] = level;
  slot = slot + 1;
  if (slot >= SAMPLES)
  {
    slot = 0;
    filled = true;
  }
  int average = rollingAverage();
  if (average > threshold)
  {
    if (overSince == 0)
    {
      overSince = millis();
    }
    else if (millis() - overSince > alertHold)
    {
      alerting = true;
    }
```

```
  }
  else
  {
    overSince = 0;
    alerting = false;
  }
  digitalWrite(led, alerting);
  if (alerting && (millis() / 500) % 2 == 0)
  {
    tone(buzzer, 1200, 100);
  }
  Serial.print("Now ");
  Serial.print(level);
  Serial.print(" Avg ");
  Serial.println(average);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("NOISE MONITOR");
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print(average);
  display.print("%");
  display.setTextSize(1);
  display.setCursor(70, 18);
  display.print("now ");
  display.print(level);
  // Threshold marker on a level bar
  display.drawRect(0, 34, 128, 12, SSD1306_WHITE);
  display.fillRect(1, 35, map(average, 0, 100, 0, 126), 10, SSD1306_WHITE);
  int mark = map(threshold, 0, 100, 0, 126);
  display.drawLine(mark, 32, mark, 48, SSD1306_WHITE);
  display.setCursor(0, 52);
  if (alerting)
  {
    display.print("TOO LOUD - QUIET PLEASE");
  }
  else if (overSince > 0)
  {
    display.print("Rising... ");
    display.print((alertHold - (millis() - overSince)) / 1000);
    display.print("s");
  }
  else
  {
    display.print("Acceptable level");
  }
  display.display();
}
```

What you should see: A quiet room reads around 10% with the verdict Acceptable level. A single clap raises the now value but barely moves the average. Sustained loud talking pushes the average past the marker and starts a countdown. After 5 seconds over threshold: TOO LOUD with the LED and buzzer.

Voice Activity Detector

Goal: Distinguish someone actually speaking from background hum, and time how long the speech lasts.

Speech is not a steady tone, it is a pattern of bursts and gaps, which is why the envelope of the signal matters more than the signal itself. The sketch measures the room's own background level for three seconds at startup, then treats anything well above that floor as voice. A short hang time keeps the detector latched during the natural pauses between words, so one sentence registers as one utterance rather than twenty.

Online page with troubleshooting: geekobot.in/projects/titan-kit-053-voice-activity-detector

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Audio envelope detection
- Calibrating against the room's own noise floor
- Hang time to bridge natural pauses
- Measuring event durations with millis()

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int led = 25;
int noiseFloor = 0;
int voiceMargin = 40;
unsigned long hangTime = 600; // Keep detecting through short pauses
bool speaking = false;
unsigned long lastVoice = 0;
unsigned long speechStart = 0;
unsigned long longestSpeech = 0;
int utterances = 0;
int amplitude()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 30)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return highest - lowest;
}
void calibrate()
{
  long total = 0;
  int count = 0;
  unsigned long start = millis();
  while (millis() - start < 3000)
  {
    total = total + amplitude();
    count = count + 1;
  }
  noiseFloor = total / count;
  Serial.print("Noise floor: ");
  Serial.println(noiseFloor);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.clearDisplay();
  display.setCursor(0, 24);
  display.print("Measuring silence...");
  display.print(" please wait");
  display.display();
  calibrate();
}
void loop()
{
  int level = amplitude();
  bool aboveFloor = (level > noiseFloor + voiceMargin);
  if (aboveFloor)
  {
    lastVoice = millis();
    if (!speaking)
    {
      speaking = true;
      speechStart = millis();
      utterances = utterances + 1;
      Serial.println("Voice started");
    }
```

```
  }
  if (speaking && millis() - lastVoice > hangTime)
  {
    unsigned long duration = lastVoice - speechStart;
    if (duration > longestSpeech)
    {
      longestSpeech = duration;
    }
    speaking = false;
    Serial.print("Voice ended after ");
    Serial.print(duration);
    Serial.println(" ms");
  }
  digitalWrite(led, speaking);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("VOICE DETECTOR");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print(speaking ? "SPEAKING" : "SILENT");
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print("Level ");
  display.print(level);
  display.print(" floor ");
  display.print(noiseFloor);
  display.setCursor(0, 46);
  display.print("Utterances: ");
  display.print(utterances);
  display.setCursor(0, 56);
  if (speaking)
  {
    display.print("Now: ");
    display.print((millis() - speechStart) / 100 / 10.0, 1);
    display.print("s");
  }
  else
  {
    display.print("Longest: ");
    display.print(longestSpeech / 100 / 10.0, 1);
    display.print("s");
  }
  display.display();
}
```

What you should see: Three seconds of Measuring silence... at startup. SILENT on screen with the LED off in a quiet room. Speak and SPEAKING appears with the LED lit and a running timer. Stop and the utterance count increases and the longest duration is recorded.

Sound Reactive RGB Lighting

Goal: Make an RGB LED pulse with the music: brightness follows the volume and the colour shifts as the beat changes.

Two competing needs meet here. The lights must jump instantly on a beat, but they must not flicker between beats, so the sketch rises fast and falls slowly. When the new level is higher than the current one it is adopted immediately; when it is lower, the current level decays gradually towards it. That asymmetric smoothing is exactly how a studio compressor behaves, and it is what makes reactive lighting look musical rather than twitchy.

Online page with troubleshooting: geekobot.in/projects/titan-kit-054-sound-reactive-rgb-lighting

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Fast analog reads and PWM scaling
- Asymmetric smoothing, fast attack, slow decay
- Converting a hue angle to RGB channels
- Passing values back through reference parameters

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
RGB LED	Red	**GPIO32**	Through a 220 ohm resistor
RGB LED	Green	**GPIO5**	Through a 220 ohm resistor
RGB LED	Blue	**GPIO23**	Through a 220 ohm resistor
RGB LED	Common cathode	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
int micPin = 34;
int redPin = 32;
int greenPin = 5;
int bluePin = 23;
float smoothLevel = 0;
float hue = 0;
int quietLevel = 20;
int loudLevel = 350;
int amplitude()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 20)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return highest - lowest;
}
void hueToRGB(float h, float brightness, int &r, int &g, int &b)
{
  float section = h / 60.0;
  if (section < 1) { r = 255; g = section * 255; b = 0; }
  else if (section < 2) { r = (2 - section) * 255; g = 255; b = 0; }
  else if (section < 3) { r = 0; g = 255; b = (section - 2) * 255; }
  else if (section < 4) { r = 0; g = (4 - section) * 255; b = 255; }
  else if (section < 5) { r = (section - 4) * 255; g = 0; b = 255; }
  else { r = 255; g = 0; b = (6 - section) * 255; }
  r = r * brightness;
  g = g * brightness;
  b = b * brightness;
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  int level = constrain(map(amplitude(), quietLevel, loudLevel, 0, 100), 0, 100);
  // Rise instantly, fall slowly
  if (level > smoothLevel)
  {
    smoothLevel = level;
  }
  else
  {
    smoothLevel = smoothLevel * 0.90 + level * 0.10;
  }
  // Loud sounds push the colour further round the wheel
  hue = hue + 0.2 + (smoothLevel / 25.0);
  if (hue >= 360) hue = hue - 360;
  float brightness = smoothLevel / 100.0;
  int r, g, b;
  hueToRGB(hue, brightness, r, g, b);
  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);
  Serial.print("Level ");
  Serial.print(smoothLevel);
  Serial.print(" Hue ");
  Serial.println(hue);
}
```

What you should see: Silence leaves the LED almost dark. Clap and it flashes brightly at once. Play music and it pulses in time, brightening on each beat. The colour drifts round the wheel, moving faster when the music is louder.

Sound Frequency Visualizer

Goal: Estimate the pitch of a sound by counting how often the waveform crosses its midpoint, and show which band it falls in.

A proper spectrum needs an FFT, which is more than we will attempt here. Zero-crossing counting is the honest approximation: count how many times the signal passes its midpoint in a fixed window, and half that count per second is roughly the dominant frequency. It works well for a whistle or a single tone and poorly for a chord, which is a genuinely useful thing for students to discover for themselves.

Online page with troubleshooting: geekobot.in/projects/titan-kit-055-sound-frequency-visualizer

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 5 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Basic spectrum analysis by zero crossing
- Finding a signal's resting midpoint
- Why an FFT would be needed for real spectra
- The limits of a simple approximation

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Band 1 (bass)	**GPIO12**	Through a 220 ohm resistor
LED	Band 2	**GPIO14**	Through a 220 ohm resistor
LED	Band 3 (mid)	**GPIO27**	Through a 220 ohm resistor
LED	Band 4	**GPIO26**	Through a 220 ohm resistor
LED	Band 5 (treble)	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int bandLeds[5] = {12, 14, 27, 26, 25};
int midpoint = 512;
int bandLimits[5] = {200, 500, 1000, 2000, 5000};
const char* bandNames[5] = {"BASS", "LOW MID", "MID", "HIGH MID", "TREBLE"};
int measureFrequency()
{
  int crossings = 0;
  bool wasAbove = false;
  int highest = 0;
  int lowest = 1023;
  unsigned long start = micros();
  unsigned long window = 100000; // 100 ms
  while (micros() - start < window)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
    bool isAbove = (sample > midpoint);
    if (isAbove && !wasAbove)
    {
      crossings = crossings + 1;
    }
    wasAbove = isAbove;
  }
  if (highest - lowest < 40)
  {
    return 0; // Too quiet to judge
  }
  return crossings * 10; // Crossings in 100 ms -> per second
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  for (int i = 0; i < 5; i++)
  {
    pinMode(bandLeds[i], OUTPUT);
  }
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  // Learn the resting midpoint of this module
  long total = 0;
  for (int i = 0; i < 200; i++)
  {
    total = total + analogRead(micPin);
    delay(2);
  }
  midpoint = total / 200;
  Serial.print("Midpoint: ");
  Serial.println(midpoint);
}
void loop()
{
  int frequency = measureFrequency();
  int band = -1;
  for (int i = 0; i < 5; i++)
  {
    if (frequency > 0 && frequency <= bandLimits[i])
    {
      band = i;
      break;
    }
  }
  if (frequency > bandLimits[4]) band = 4;
  for (int i = 0; i < 5; i++)
```

```
  {
    digitalWrite(bandLeds[i], i == band);
  }
  Serial.print("Frequency approx ");
  Serial.println(frequency);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("FREQUENCY BANDS");
  display.setTextSize(2);
  display.setCursor(0, 14);
  if (frequency > 0)
  {
    display.print(frequency);
    display.setTextSize(1);
    display.print(" Hz");
  }
  else
  {
    display.print("--");
  }
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print("Band: ");
  display.print(band >= 0 ? bandNames[band] : "silent");
  for (int i = 0; i < 5; i++)
  {
    int x = i * 25 + 3;
    display.drawRect(x, 46, 20, 14, SSD1306_WHITE);
    if (i == band)
    {
      display.fillRect(x + 2, 48, 16, 10, SSD1306_WHITE);
    }
  }
  display.display();
}
```

What you should see: Silence shows -- and no LED. Hum a low note: around 150 Hz with the BASS LED lit. Whistle: 1500 Hz or more with a higher band lit. Play a chord and the reading wanders, a single tone gives a far steadier number.

Smart Silence Detector

Goal: Trigger an action when a room becomes completely quiet, rather than when it becomes loud, inverted alarm logic.

Every alarm so far has fired on a rising value. Invert the comparison and the same machinery detects absence instead of presence, useful for a machine that has stopped, a fan that has failed, or a study room that has finally settled. The one addition is a required duration: silence only counts once it has lasted, because the gap between two words is also technically silence.

Online page with troubleshooting: geekobot.in/projects/titan-kit-056-smart-silence-detector

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Inverted threshold logic
- Detecting absence rather than presence
- Requiring a condition to persist before confirming
- Tracking best-case durations

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int led = 25;
int buzzer = 33;
int silenceBelow = 15; // Percent
unsigned long requiredQuiet = 5000; // Must be quiet this long
unsigned long quietSince = 0;
bool silenceConfirmed = false;
int silenceEvents = 0;
unsigned long longestSilence = 0;
int level()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 40)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return constrain(map(highest - lowest, 20, 400, 0, 100), 0, 100);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  quietSince = millis();
}
void loop()
{
  int now = level();
  bool isQuiet = (now < silenceBelow);
  if (isQuiet)
  {
    if (quietSince == 0)
    {
      quietSince = millis();
    }
    unsigned long quietFor = millis() - quietSince;
    if (quietFor > requiredQuiet && !silenceConfirmed)
    {
      silenceConfirmed = true;
      silenceEvents = silenceEvents + 1;
      tone(buzzer, 900, 200);
      Serial.println("Silence confirmed");
    }
    if (quietFor > longestSilence)
    {
      longestSilence = quietFor;
    }
  }
  else
  {
    if (silenceConfirmed)
    {
      Serial.println("Sound returned");
    }
    quietSince = 0;
    silenceConfirmed = false;
  }
  digitalWrite(led, silenceConfirmed);
  display.clearDisplay();
```

```
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("SILENCE DETECTOR");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print(silenceConfirmed ? "QUIET" : "ACTIVE");
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print("Level ");
  display.print(now);
  display.print("% limit ");
  display.print(silenceBelow);
  display.print("%");
  display.setCursor(0, 46);
  if (isQuiet && !silenceConfirmed)
  {
    display.print("Settling... ");
    display.print((requiredQuiet - (millis() - quietSince)) / 1000);
    display.print("s");
  }
  else if (silenceConfirmed)
  {
    display.print("Quiet for ");
    display.print((millis() - quietSince) / 1000);
    display.print("s");
  }
  else
  {
    display.print("Room is in use");
  }
  display.setCursor(0, 56);
  display.print("Events ");
  display.print(silenceEvents);
  display.print(" best ");
  display.print(longestSilence / 1000);
  display.print("s");
  display.display();
}
```

What you should see: Talking keeps the display on ACTIVE. Go quiet and a countdown appears: Settling... 4s. After 5 seconds of quiet: QUIET, the LED lights and a single beep sounds. The longest silence achieved is recorded at the bottom.

PROJECT 057 · LEVEL 10 · SOUND ANALYSIS

Meeting Room Noise Indicator

Goal: Judge whether a room is quiet enough for a meeting, sorting the noise level into named categories with clear advice.

Categorical mapping turns a continuous number into a small set of meaningful labels. The band boundaries are stored in one array and the labels in another, so finding the category is a single loop that stops at the first limit the value falls under. Adding a fifth category means adding one entry to each array, no new if statements. That is the practical advantage of table-driven code over branching.

Online page with troubleshooting: geekobot.in/projects/titan-kit-057-meeting-room-noise-indicator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Categorical data mapping
- Table-driven categories instead of branching
- Marking band boundaries on a level bar
- Smoothing before categorising to avoid flapping

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green (suitable)	**GPIO27**	Through a 220 ohm resistor
LED	Yellow (borderline)	**GPIO26**	Through a 220 ohm resistor
LED	Red (unsuitable)	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define CATEGORIES 4
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int ledPins[3] = {27, 26, 25};
int limits[CATEGORIES] = {20, 40, 65, 100};
const char* names[CATEGORIES] = {"LIBRARY", "SUITABLE", "BUSY", "TOO LOUD"};
const char* advice[CATEGORIES] =
{
  "Perfect for calls",
  "Fine for meetings",
  "Hard to hear speech",
  "Find another room"
};
int whichLed[CATEGORIES] = {0, 0, 1, 2};
float smoothed = 0;
int level()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 40)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return constrain(map(highest - lowest, 20, 400, 0, 100), 0, 100);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  for (int i = 0; i < 3; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  smoothed = smoothed * 0.8 + level() * 0.2;
  int value = (int)smoothed;
  int category = CATEGORIES - 1;
  for (int i = 0; i < CATEGORIES; i++)
  {
    if (value <= limits[i])
    {
      category = i;
      break;
    }
  }
  for (int i = 0; i < 3; i++)
  {
    digitalWrite(ledPins[i], i == whichLed[category]);
  }
  Serial.print(value);
  Serial.print("% -> ");
  Serial.println(names[category]);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("MEETING ROOM");
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print(names[category]);
  display.setTextSize(1);
```

```
  display.setCursor(0, 32);
  display.print(advice[category]);
  display.drawRect(0, 44, 128, 10, SSD1306_WHITE);
  display.fillRect(1, 45, map(value, 0, 100, 0, 126), 8, SSD1306_WHITE);
  // Mark the category boundaries on the bar
  for (int i = 0; i < CATEGORIES - 1; i++)
  {
    int x = map(limits[i], 0, 100, 0, 126);
    display.drawLine(x, 42, x, 56, SSD1306_WHITE);
  }
  display.setCursor(0, 56);
  display.print("Level ");
  display.print(value);
  display.print("%");
  display.display();
}
```

What you should see: A quiet room shows LIBRARY / Perfect for calls with the green LED. Normal conversation gives SUITABLE, still green. Several people talking: BUSY / Hard to hear speech with the yellow LED. Shouting reaches TOO LOUD with the red LED.

Audio Peak Detector

Goal: Capture the loudest sound in a session and hold it on screen, with a peak marker that decays slowly like a studio meter.

Two peaks are tracked at once and they behave differently. The session peak never falls, it is the loudest thing heard since the last reset, and it is what you would report. The floating peak marker holds for a moment and then slides down, which is what makes a studio meter readable: it shows you the recent maximum without hiding the current level. Max value retention is the whole idea, and the decay is what makes it usable.

Online page with troubleshooting: geekobot.in/projects/titan-kit-058-audio-peak-detector

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Max value retention algorithms
- Peak hold with timed decay
- Separating session maxima from floating peaks
- Drawing a vertical meter with a marker line

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	One leg	**GPIO12**	GPIO12 is a boot pin: if the board will not start, unplug this wire while it powers up
Button	Other leg	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int resetButton = 12;
int sessionPeak = 0;
float floatingPeak = 0;
unsigned long peakTime = 0;
unsigned long holdTime = 1200;
int peakCount = 0;
int reportAbove = 70;
int level()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 25)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return constrain(map(highest - lowest, 20, 400, 0, 100), 0, 100);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(resetButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  int now = level();
  if (digitalRead(resetButton) == LOW)
  {
    sessionPeak = 0;
    floatingPeak = 0;
    peakCount = 0;
    Serial.println("Peaks cleared");
    delay(300);
  }
  if (now > sessionPeak)
  {
    sessionPeak = now;
    if (now > reportAbove)
    {
      peakCount = peakCount + 1;
      Serial.print("New peak: ");
      Serial.println(now);
    }
  }
  if (now >= floatingPeak)
  {
    floatingPeak = now;
    peakTime = millis();
  }
  else if (millis() - peakTime > holdTime)
  {
    floatingPeak = floatingPeak - 0.8; // Slow decay
    if (floatingPeak < now) floatingPeak = now;
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("PEAK METER");
  display.setCursor(78, 0);
  display.print("Max ");
```

```
  display.print(sessionPeak);
  // Vertical meter
  int barBottom = 58;
  int barTop = 14;
  display.drawRect(6, barTop, 22, barBottom - barTop, SSD1306_WHITE);
  int fill = map(now, 0, 100, 0, barBottom - barTop - 2);
  display.fillRect(8, barBottom - 1 - fill, 18, fill, SSD1306_WHITE);
  int markerY = barBottom - 1 - map((int)floatingPeak, 0, 100, 0, barBottom - barTop - 2);
  display.drawLine(4, markerY, 30, markerY, SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(44, 20);
  display.print(now);
  display.setTextSize(1);
  display.print("%");
  display.setCursor(44, 40);
  display.print("Hold ");
  display.print((int)floatingPeak);
  display.print("%");
  display.setCursor(44, 50);
  display.print("Loud events ");
  display.print(peakCount);
  display.display();
}
```

What you should see: A vertical bar follows the sound level in real time. A horizontal line marks the recent peak and slides down after about a second. The session maximum in the corner never falls until reset. Clap sharply and the loud-events counter increases.

Sound Triggered Counter

Goal: Count sharp sounds like claps or drum beats, ignoring the echoes that follow each one.

A clap is a transient, a sudden rise followed by a rapid fall, and its echoes look like several smaller claps arriving a few tens of milliseconds later. Detecting the rise is easy; the real work is debouncing. After each accepted event the sketch refuses to count anything for 250 ms, which is longer than the echo but shorter than a human can deliberately clap twice. It also measures the interval between beats and converts it to beats per minute.

Online page with troubleshooting: geekobot.in/projects/titan-kit-059-sound-triggered-counter

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Transient peak counting and debouncing
- Dead time as an echo filter
- Converting an interval to beats per minute
- Rejecting intervals that are implausible

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Button	Reset button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int led = 25;
int resetButton = 12;
int triggerLevel = 55;
unsigned long deadTime = 250; // Ignore echoes for this long
int count = 0;
unsigned long lastTrigger = 0;
unsigned long lastInterval = 0;
int bpm = 0;
int level()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 12)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return constrain(map(highest - lowest, 20, 400, 0, 100), 0, 100);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(led, OUTPUT);
  pinMode(resetButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
}
void loop()
{
  int now = level();
  if (digitalRead(resetButton) == LOW)
  {
    count = 0;
    bpm = 0;
    lastInterval = 0;
    Serial.println("Counter reset");
    delay(300);
  }
  if (now > triggerLevel && millis() - lastTrigger > deadTime)
  {
    if (lastTrigger > 0)
    {
      lastInterval = millis() - lastTrigger;
      if (lastInterval > 200 && lastInterval < 3000)
      {
        bpm = 60000 / lastInterval;
      }
    }
    lastTrigger = millis();
    count = count + 1;
    digitalWrite(led, HIGH);
    Serial.print("Event ");
    Serial.print(count);
    Serial.print(" gap ");
    Serial.print(lastInterval);
    Serial.println(" ms");
  }
  if (millis() - lastTrigger > 100)
  {
    digitalWrite(led, LOW);
  }
  display.clearDisplay();
```

```
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("SOUND COUNTER");
  display.setTextSize(3);
  display.setCursor(0, 14);
  display.print(count);
  display.setTextSize(1);
  display.setCursor(64, 20);
  display.print("events");
  display.setCursor(0, 42);
  if (lastInterval > 0)
  {
    display.print("Gap ");
    display.print(lastInterval);
    display.print("ms");
  }
  else
  {
    display.print("Clap to start");
  }
  display.setCursor(0, 52);
  if (bpm > 0)
  {
    display.print("Tempo ");
    display.print(bpm);
    display.print(" BPM");
  }
  else
  {
    display.print("Level ");
    display.print(now);
    display.print("%");
  }
  display.display();
}
```

What you should see: Clap once and the counter reads 1 with a brief LED flash. Echoes in a hard-walled room do not add extra counts. Clap steadily and the tempo in BPM appears. Press the reset button to return to zero.

Noise History Logger

Goal: Record the average noise level for each of the last minutes and plot the result as a history graph.

Two timescales operate at once. Readings are taken constantly and accumulated into a running total; once a minute that total is divided by the number of samples to give one averaged data point, and the accumulator resets. Averaging before storing is what makes long-term logging possible on a board with 2 KB of RAM, a hundred stored minutes cost a hundred bytes, while storing every raw sample would exhaust memory in seconds.

Online page with troubleshooting: geekobot.in/projects/titan-kit-060-noise-history-logger

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Moving averages computation
- Accumulating samples before storing
- Why averaging is essential for long-term logs on small RAM
- Working with two different timescales in one sketch

Wiring

Part	Pin	ESP32	Note
Sound sensor	VCC	**3V3**	
Sound sensor	GND	**GND**	
Sound sensor	AO (analog out)	**GPIO34**	
Sound sensor	DO (digital out)	**GPIO13**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	One leg	**GPIO12**	GPIO12 is a boot pin: if the board will not start, unplug this wire while it powers up
Button	Other leg	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SLOTS 100
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int micPin = 34;
int clearButton = 12;
byte log_[SLOTS];
int stored = 0;
long accumulator = 0;
int samples = 0;
unsigned long slotStart = 0;
unsigned long slotLength = 60000; // One minute per slot
int level()
{
  unsigned long start = millis();
  int highest = 0;
  int lowest = 1023;
  while (millis() - start < 30)
  {
    int sample = analogRead(micPin);
    if (sample > highest) highest = sample;
    if (sample < lowest) lowest = sample;
  }
  return constrain(map(highest - lowest, 20, 400, 0, 100), 0, 100);
}
void storeSlot(byte value)
{
  if (stored < SLOTS)
  {
    log_[stored] = value;
    stored = stored + 1;
    return;
  }
  for (int i = 0; i < SLOTS - 1; i++)
  {
    log_[i] = log_[i + 1];
  }
  log_[SLOTS - 1] = value;
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(clearButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  slotStart = millis();
}
void loop()
{
  int now = level();
  accumulator = accumulator + now;
  samples = samples + 1;
  if (digitalRead(clearButton) == LOW)
  {
    stored = 0;
    accumulator = 0;
    samples = 0;
    slotStart = millis();
    Serial.println("Log cleared");
    delay(300);
  }
  if (millis() - slotStart > slotLength)
  {
    byte average = accumulator / samples;
    storeSlot(average);
    Serial.print("Minute ");
    Serial.print(stored);
```

```
    Serial.print(" average ");
    Serial.println(average);
    accumulator = 0;
    samples = 0;
    slotStart = millis();
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("NOISE HISTORY");
  display.setCursor(78, 0);
  display.print("now ");
  display.print(now);
  int runningAverage = samples > 0 ? accumulator / samples : 0;
  display.setCursor(0, 10);
  display.print("Minute avg ");
  display.print(runningAverage);
  display.print("% ");
  display.print((slotLength - (millis() - slotStart)) / 1000);
  display.print("s");
  // History graph
  display.drawLine(14, 22, 14, 58, SSD1306_WHITE);
  display.drawLine(14, 58, 127, 58, SSD1306_WHITE);
  display.setCursor(0, 20);
  display.print("100");
  display.setCursor(8, 52);
  display.print("0");
  for (int i = 0; i < stored; i++)
  {
    int x = 16 + i;
    if (x > 127) break;
    int h = map(log_[i], 0, 100, 0, 35);
    display.drawLine(x, 58, x, 58 - h, SSD1306_WHITE);
  }
  if (stored == 0)
  {
    display.setCursor(20, 36);
    display.print("First minute...");
  }
  display.display();
}
```

What you should see: The header shows the instantaneous level and the running average for this minute. A countdown shows when the next data point will be stored. After each minute a new bar appears in the history graph. Once 100 minutes are stored the graph scrolls, dropping the oldest.

Digital Spirit Level

Goal: Turn the board into a spirit level with a bubble that moves on screen and LEDs that confirm when a surface is flat.

An accelerometer at rest measures only gravity, and gravity always points down, so the way that 9.8 m/s² is shared between the X, Y and Z axes tells you how the board is tilted. atan2() converts those components into pitch and roll angles, and unlike plain atan() it gets the quadrant right, which matters once the board passes vertical. The bubble is simply those two angles drawn as an offset from the centre of a circle.

Online page with troubleshooting: geekobot.in/projects/titan-kit-061-digital-spirit-level

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Accelerometer pitch and roll
- Why atan2() is used instead of atan()
- Extracting orientation from the gravity vector
- Smoothing an IMU reading before displaying it

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Red LED anode	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int greenLed = 27;
int redLed = 26;
int buzzer = 33;
float levelTolerance = 1.5; // Degrees
float pitch = 0;
float roll = 0;
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    Serial.println("OLED not found");
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  // Gravity components become tilt angles
  float newPitch = atan2(a.acceleration.y,
  sqrt(a.acceleration.x * a.acceleration.x +
  a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
  float newRoll = atan2(-a.acceleration.x, a.acceleration.z) * 180.0 / PI;
  pitch = pitch * 0.8 + newPitch * 0.2;
  roll = roll * 0.8 + newRoll * 0.2;
  bool isLevel = (abs(pitch) < levelTolerance && abs(roll) < levelTolerance);
  digitalWrite(greenLed, isLevel);
  digitalWrite(redLed, !isLevel);
  Serial.print("Pitch ");
  Serial.print(pitch, 1);
  Serial.print(" Roll ");
  Serial.println(roll, 1);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("SPIRIT LEVEL");
  // Bubble vial
  int cx = 96;
  int cy = 32;
  display.drawCircle(cx, cy, 26, SSD1306_WHITE);
  display.drawCircle(cx, cy, 6, SSD1306_WHITE);
  int bx = cx + constrain((int)(roll * 1.2), -22, 22);
  int by = cy + constrain((int)(pitch * 1.2), -22, 22);
  display.fillCircle(bx, by, 4, SSD1306_WHITE);
  display.setCursor(0, 16);
  display.print("Pitch");
  display.setCursor(0, 26);
  display.print(pitch, 1);
  display.print((char)247);
  display.setCursor(0, 40);
  display.print("Roll");
  display.setCursor(0, 50);
```

```
  display.print(roll, 1);
  display.print((char)247);
  if (isLevel)
  {
    display.setCursor(0, 8);
    display.print("LEVEL");
    if ((millis() / 800) % 2 == 0)
    {
      tone(buzzer, 1500, 60);
    }
  }
  display.display();
  delay(40);
}
```

What you should see: The bubble sits off centre when the board is tilted. Level the board and the bubble drops into the middle circle. The green LED lights and a soft beep confirms level within 1.5 degrees. Pitch and roll are printed to a tenth of a degree.

Tilt Angle Measurement System

Goal: Measure and display precise inclination angles, with a calibration step that zeroes the sensor against a known flat surface.

No sensor is perfectly mounted, and no board is perfectly flat, so a raw reading of 1.8 degrees on a genuinely level table is normal. Calibration fixes that by recording the current angles as the new zero and subtracting them from every future reading. This offset-based approach is how almost all instruments are zeroed, from kitchen scales to surveying equipment, and it costs two variables.

Online page with troubleshooting: geekobot.in/projects/titan-kit-062-tilt-angle-measurement

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- IMU maths and calibration
- Zeroing an instrument with stored offsets
- Averaging many samples for a stable calibration
- Hold functions for taking a measurement away from the bench

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Calibrate (zero here)	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Hold reading	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int calibrateButton = 12;
int holdButton = 14;
float pitchOffset = 0;
float rollOffset = 0;
float pitch = 0;
float roll = 0;
bool holding = false;
float heldPitch = 0;
float heldRoll = 0;
void readAngles()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float p = atan2(a.acceleration.y,
  sqrt(a.acceleration.x * a.acceleration.x +
  a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
  float r = atan2(-a.acceleration.x, a.acceleration.z) * 180.0 / PI;
  pitch = pitch * 0.85 + (p - pitchOffset) * 0.15;
  roll = roll * 0.85 + (r - rollOffset) * 0.15;
}
void calibrate()
{
  display.clearDisplay();
  display.setCursor(0, 24);
  display.print("Calibrating...");
  display.setCursor(0, 36);
  display.print("Keep still");
  display.display();
  pitchOffset = 0;
  rollOffset = 0;
  float sumP = 0;
  float sumR = 0;
  for (int i = 0; i < 100; i++)
  {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    sumP += atan2(a.acceleration.y,
    sqrt(a.acceleration.x * a.acceleration.x +
    a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
    sumR += atan2(-a.acceleration.x, a.acceleration.z) * 180.0 / PI;
    delay(15);
  }
  pitchOffset = sumP / 100.0;
  rollOffset = sumR / 100.0;
  Serial.print("Offsets set to ");
  Serial.print(pitchOffset);
  Serial.print(" / ");
  Serial.println(rollOffset);
  pitch = 0;
  roll = 0;
}
void setup()
{
  pinMode(calibrateButton, INPUT_PULLUP);
  pinMode(holdButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
  {
    while (true) { }
  }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
```

```
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
  calibrate();
}
void loop()
{
  readAngles();
  if (digitalRead(calibrateButton) == LOW)
  {
    calibrate();
    delay(300);
  }
  if (digitalRead(holdButton) == LOW)
  {
    holding = !holding;
    if (holding)
    {
      heldPitch = pitch;
      heldRoll = roll;
    }
    delay(300);
  }
  float showPitch = holding ? heldPitch : pitch;
  float showRoll = holding ? heldRoll : roll;
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("TILT MEASURE");
  if (holding)
  {
    display.setCursor(86, 0);
    display.print("HELD");
  }
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print("X ");
  display.print(showPitch, 1);
  display.print((char)247);
  display.setCursor(0, 34);
  display.print("Y ");
  display.print(showRoll, 1);
  display.print((char)247);
  display.setTextSize(1);
  display.setCursor(0, 56);
  display.print("D3 zero D4 hold");
  display.display();
  delay(40);
}
```

What you should see: A short Calibrating... message at startup. On a flat table both axes read close to 0.0°. Tilt the board and the angles change to a tenth of a degree. Press the second button to freeze the reading so it can be written down.

Gesture Controlled LED Patterns

Goal: Change lighting effects by tilting the board, left, right, forward, back and flat each select a different pattern.

Mapping motion to action means turning a continuous angle into a discrete command, and the risk is chatter: hold the board near the boundary and the pattern would flip back and forth many times a second. Two defences are used. A gesture must exceed 25 degrees to register, but only drops below 15 degrees to clear, and once a pattern is chosen it is locked for 600 ms. Both are forms of hysteresis, and both are needed.

Online page with troubleshooting: geekobot.in/projects/titan-kit-063-gesture-controlled-led-patterns

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 6 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Motion-to-action mapping
- Hysteresis with separate enter and exit thresholds
- Lock periods to prevent rapid switching
- Driving patterns from a shared step counter

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED 1	**GPIO12**	Through a 220 ohm resistor
LED	LED 2	**GPIO14**	Through a 220 ohm resistor
LED	LED 3	**GPIO27**	Through a 220 ohm resistor
LED	LED 4	**GPIO26**	Through a 220 ohm resistor
LED	LED 5	**GPIO25**	Through a 220 ohm resistor
LED	LED 6	**GPIO33**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int ledPins[6] = {12, 14, 27, 26, 25, 33};
float pitch = 0;
float roll = 0;
int pattern = 0;
unsigned long lastChange = 0;
unsigned long lockTime = 600;
int step_ = 0;
const char* patternNames[5] =
{
  "ALL OFF", "CHASE", "BOUNCE", "BLINK ALL", "ALTERNATE"
};
void allOff()
{
  for (int i = 0; i < 6; i++)
  {
    digitalWrite(ledPins[i], LOW);
  }
}
void runPattern()
{
  step_ = step_ + 1;
  if (pattern == 0)
  {
    allOff();
  }
  else if (pattern == 1) // Chase
  {
    allOff();
    digitalWrite(ledPins[(step_ / 4) % 6], HIGH);
  }
  else if (pattern == 2) // Bounce
  {
    allOff();
    int position = (step_ / 4) % 10;
    if (position > 5) position = 10 - position;
    digitalWrite(ledPins[constrain(position, 0, 5)], HIGH);
  }
  else if (pattern == 3) // Blink all
  {
    bool on = ((step_ / 8) % 2 == 0);
    for (int i = 0; i < 6; i++)
    {
      digitalWrite(ledPins[i], on);
    }
  }
  else // Alternate
  {
    bool phase = ((step_ / 8) % 2 == 0);
    for (int i = 0; i < 6; i++)
    {
      digitalWrite(ledPins[i], (i % 2 == 0) == phase);
    }
  }
}
void setup()
{
  for (int i = 0; i < 6; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
```

```
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float p = atan2(a.acceleration.y,
  sqrt(a.acceleration.x * a.acceleration.x +
  a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
  float r = atan2(-a.acceleration.x, a.acceleration.z) * 180.0 / PI;
  pitch = pitch * 0.8 + p * 0.2;
  roll = roll * 0.8 + r * 0.2;
  if (millis() - lastChange > lockTime)
  {
    int newPattern = pattern;
    if (roll > 25) newPattern = 1; // Tilt right
    else if (roll < -25) newPattern = 2; // Tilt left
    else if (pitch > 25) newPattern = 3; // Tilt forward
    else if (pitch < -25) newPattern = 4; // Tilt back
    else if (abs(roll) < 15 && abs(pitch) < 15) newPattern = 0;
    if (newPattern != pattern)
    {
      pattern = newPattern;
      lastChange = millis();
      step_ = 0;
      Serial.print("Pattern -> ");
      Serial.println(patternNames[pattern]);
    }
  }
  runPattern();
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("GESTURE LIGHTS");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print(patternNames[pattern]);
  display.setTextSize(1);
  display.setCursor(0, 38);
  display.print("Pitch ");
  display.print(pitch, 0);
  display.print(" Roll ");
  display.print(roll, 0);
  display.setCursor(0, 52);
  display.print("Tilt to change");
  display.display();
  delay(25);
}
```

What you should see: Held flat the LEDs are off and the screen reads ALL OFF. Tilt right: a single light chases along the row. Tilt left: the light bounces back and forth. Tilt forward or back for blink-all and alternate patterns.

Motion Intensity Meter

Goal: Measure how hard the board is being moved by combining all three acceleration axes into a single magnitude.

Motion in three dimensions cannot be judged from one axis, because a shake along X and a shake along Z are equally vigorous. Pythagoras in three dimensions gives the answer: the magnitude is the square root of the sum of the squares. At rest that magnitude equals gravity, about 9.8 m/s², so subtracting 9.8 leaves the movement alone, which is why the meter reads zero on a still table regardless of which way up it is.

Online page with troubleshooting: geekobot.in/projects/titan-kit-064-motion-intensity-meter

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 5 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Vector magnitude calculation
- Removing gravity from a three-axis reading
- Why one axis is not enough for motion
- Attack and decay applied to a sensor signal

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Level 1	**GPIO12**	Through a 220 ohm resistor
LED	Level 2	**GPIO14**	Through a 220 ohm resistor
LED	Level 3	**GPIO27**	Through a 220 ohm resistor
LED	Level 4	**GPIO26**	Through a 220 ohm resistor
LED	Level 5	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int ledPins[5] = {12, 14, 27, 26, 25};
float intensity = 0;
float peak = 0;
unsigned long peakTime = 0;
void setup()
{
  for (int i = 0; i < 5; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setFilterBandwidth(MPU6050_BAND_44_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float x = a.acceleration.x;
  float y = a.acceleration.y;
  float z = a.acceleration.z;
  // 3D vector magnitude, then remove gravity
  float magnitude = sqrt(x * x + y * y + z * z);
  float movement = abs(magnitude - 9.81);
  // Fast attack, slow decay
  if (movement > intensity)
  {
    intensity = movement;
  }
  else
  {
    intensity = intensity * 0.9 + movement * 0.1;
  }
  if (intensity > peak)
  {
    peak = intensity;
    peakTime = millis();
  }
  else if (millis() - peakTime > 3000)
  {
    peak = peak * 0.98;
  }
  int level = constrain(map(intensity * 10, 0, 150, 0, 5), 0, 5);
  for (int i = 0; i < 5; i++)
  {
    digitalWrite(ledPins[i], i < level);
  }
  Serial.print("Movement ");
  Serial.print(movement, 2);
  Serial.print(" Intensity ");
  Serial.println(intensity, 2);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("MOTION INTENSITY");
  display.setTextSize(3);
  display.setCursor(0, 14);
```

```
  display.print(intensity, 1);
  display.setTextSize(1);
  display.setCursor(72, 24);
  display.print("m/s2");
  display.setCursor(0, 42);
  display.print("Peak ");
  display.print(peak, 1);
  display.setCursor(64, 42);
  if (intensity < 0.3) display.print("STILL");
  else if (intensity < 2) display.print("GENTLE");
  else if (intensity < 6) display.print("MODERATE");
  else display.print("VIGOROUS");
  display.drawRect(0, 52, 128, 10, SSD1306_WHITE);
  display.fillRect(1, 53, constrain((int)(intensity * 8), 0, 126), 8, SSD1306_WHITE);
  display.display();
  delay(20);
}
```

What you should see: A still board reads close to 0.0 with the verdict STILL. Gentle movement lights one or two LEDs and shows GENTLE. A firm shake lights all five and reads VIGOROUS. The peak value holds for 3 seconds and then decays.

Shake Detection System

Goal: Detect a deliberate shake, several strong movements in quick succession, and use it as a command.

A single jolt could be the board being set down; a shake is several jolts within a short window. The sketch counts spikes above a high-G threshold and requires three of them inside 1200 ms before it declares a shake. Requiring a pattern rather than a single event is what separates an intentional gesture from an accident, and it is why phones use exactly this technique for shake-to-undo.

Online page with troubleshooting: geekobot.in/projects/titan-kit-065-shake-detection

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- High-G acceleration spike detection
- Requiring a pattern of events, not a single one
- Time windows that expire
- Preventing one physical jolt from counting twice

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int led = 25;
int buzzer = 33;
float spikeThreshold = 14.0; // m/s2 above gravity
int spikesNeeded = 3;
unsigned long windowLength = 1200;
int spikeCount = 0;
unsigned long windowStart = 0;
unsigned long lastSpike = 0;
int shakeCount = 0;
bool ledState = false;
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_16_G);
  mpu.setFilterBandwidth(MPU6050_BAND_94_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float magnitude = sqrt(a.acceleration.x * a.acceleration.x +
  a.acceleration.y * a.acceleration.y +
  a.acceleration.z * a.acceleration.z);
  float movement = abs(magnitude - 9.81);
  // Count a spike, but not the same one twice
  if (movement > spikeThreshold && millis() - lastSpike > 90)
  {
    lastSpike = millis();
    if (spikeCount == 0 || millis() - windowStart > windowLength)
    {
      windowStart = millis();
      spikeCount = 1;
    }
    else
    {
      spikeCount = spikeCount + 1;
    }
    Serial.print("Spike ");
    Serial.print(spikeCount);
    Serial.print(" at ");
    Serial.println(movement, 1);
    if (spikeCount >= spikesNeeded)
    {
      shakeCount = shakeCount + 1;
      spikeCount = 0;
      ledState = !ledState; // The shake command
      digitalWrite(led, ledState);
      tone(buzzer, 1200, 150);
      Serial.println("SHAKE DETECTED");
    }
  }
  if (spikeCount > 0 && millis() - windowStart > windowLength)
  {
    spikeCount = 0; // Window expired
  }
```

```
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("SHAKE DETECTOR");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print("Shakes ");
  display.print(shakeCount);
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print("Force ");
  display.print(movement, 1);
  display.print(" m/s2");
  display.setCursor(0, 46);
  display.print("Spikes ");
  display.print(spikeCount);
  display.print("/");
  display.print(spikesNeeded);
  if (spikeCount > 0)
  {
    display.setCursor(64, 46);
    display.print((windowLength - (millis() - windowStart)));
    display.print("ms");
  }
  display.setCursor(0, 56);
  display.print("LED is ");
  display.print(ledState ? "ON" : "OFF");
  display.display();
}
```

What you should see: Gentle handling registers no spikes at all. One sharp jolt shows Spikes 1/3 with a countdown. Three jolts within 1.2 seconds: SHAKE DETECTED, the buzzer sounds and the LED toggles. Let the window expire and the spike count resets to zero.

Fall Detection Prototype

Goal: Recognise the signature of a fall, near weightlessness followed by a heavy impact, and raise an alarm that can be cancelled.

A falling object is in free fall, and an accelerometer in free fall reads close to zero, gravity is no longer being resisted. That brief near-zero reading, followed within half a second by a large impact spike, is a far more specific signature than impact alone, which is why simply dropping the board on a cushion does not trigger it. Real devices then wait before calling for help, so the user can cancel a false alarm.

Online page with troubleshooting: geekobot.in/projects/titan-kit-066-fall-detection-prototype

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Freefall detection algorithms
- Why free fall reads near zero on an accelerometer
- Requiring two events in sequence within a window
- Cancellable alarms with a grace period

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Red LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Passive buzzer	Piezo buzzer (−)	**GND**	
Button	Cancel button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int led = 25;
int buzzer = 33;
int cancelButton = 12;
float freeFallBelow = 3.0; // m/s2 - near weightless
float impactAbove = 22.0; // m/s2 - hard landing
unsigned long impactWindow = 600; // Impact must follow this soon
bool freeFalling = false;
unsigned long freeFallTime = 0;
bool alarm = false;
unsigned long alarmStart = 0;
unsigned long callDelay = 15000;
int fallsDetected = 0;
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(cancelButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_16_G);
  mpu.setFilterBandwidth(MPU6050_BAND_94_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float magnitude = sqrt(a.acceleration.x * a.acceleration.x +
  a.acceleration.y * a.acceleration.y +
  a.acceleration.z * a.acceleration.z);
  if (!alarm)
  {
    if (magnitude < freeFallBelow && !freeFalling)
    {
      freeFalling = true;
      freeFallTime = millis();
      Serial.println("Free fall detected");
    }
    if (freeFalling && millis() - freeFallTime > impactWindow)
    {
      freeFalling = false; // No impact followed - not a fall
    }
    if (freeFalling && magnitude > impactAbove)
    {
      alarm = true;
      alarmStart = millis();
      freeFalling = false;
      fallsDetected = fallsDetected + 1;
      Serial.println("FALL DETECTED");
    }
  }
  if (alarm)
  {
    digitalWrite(led, (millis() / 300) % 2);
    if ((millis() / 600) % 2 == 0)
    {
      tone(buzzer, 2000, 200);
    }
    if (digitalRead(cancelButton) == LOW)
```

```
    {
      alarm = false;
      noTone(buzzer);
      digitalWrite(led, LOW);
      Serial.println("Alarm cancelled by user");
      delay(300);
    }
    if (millis() - alarmStart > callDelay)
    {
      Serial.println("*** ALERTING EMERGENCY CONTACT ***");
      alarm = false;
      digitalWrite(led, LOW);
    }
  }
  else
  {
    digitalWrite(led, LOW);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("FALL DETECTION");
  if (alarm)
  {
    display.setTextSize(2);
    display.setCursor(0, 14);
    display.print("FALL!");
    display.setTextSize(1);
    display.setCursor(0, 36);
    display.print("Calling in ");
    display.print((callDelay - (millis() - alarmStart)) / 1000);
    display.print("s");
    display.setCursor(0, 48);
    display.print("Press D3 if you");
    display.setCursor(0, 56);
    display.print("are OK");
  }
  else
  {
    display.setTextSize(2);
    display.setCursor(0, 14);
    display.print(freeFalling ? "FALLING" : "MONITOR");
    display.setTextSize(1);
    display.setCursor(0, 38);
    display.print("Force ");
    display.print(magnitude, 1);
    display.print(" m/s2");
    display.setCursor(0, 50);
    display.print("Falls today: ");
    display.print(fallsDetected);
  }
  display.display();
}
```

What you should see: Normal handling shows MONITOR with the live force reading. Drop the board a short distance onto a thick cushion. Free fall then impact gives FALL! with a flashing LED and alarm. A 15-second countdown runs unless you press the cancel button.

Orientation Indicator

Goal: Work out which way up the board is, portrait, landscape, face up or face down, and show it on screen.

Six-axis gravity vectoring asks a simple question: which axis currently carries most of gravity, and in which direction? Whichever of X, Y or Z has the largest absolute value names the face that is pointing up or down, and its sign settles which of the two. A dominance margin stops the reading flickering when the board is held at 45 degrees, where two axes carry gravity almost equally.

Online page with troubleshooting: geekobot.in/projects/titan-kit-067-orientation-indicator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- 6-axis gravity vectoring
- Finding the dominant axis and using its sign
- Requiring a dominance margin before deciding
- Settle time before committing to a change

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
const char* orientations[7] =
{
  "UNCERTAIN", "PORTRAIT", "PORTRAIT UP-DOWN",
  "LANDSCAPE L", "LANDSCAPE R", "FACE UP", "FACE DOWN"
};
int current = 0;
int candidate = 0;
unsigned long candidateSince = 0;
unsigned long settleTime = 400;
int changes = 0;
int classify(float x, float y, float z)
{
  float ax = abs(x);
  float ay = abs(y);
  float az = abs(z);
  float dominant = max(ax, max(ay, az));
  float second = ax + ay + az - dominant - min(ax, min(ay, az));
  if (dominant < 7.0 || dominant - second < 3.0)
  {
    return 0; // Held at an angle - do not guess
  }
  if (az == dominant) return z > 0 ? 5 : 6;
  if (ay == dominant) return y > 0 ? 1 : 2;
  return x > 0 ? 3 : 4;
}
void setup()
{
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float x = a.acceleration.x;
  float y = a.acceleration.y;
  float z = a.acceleration.z;
  int detected = classify(x, y, z);
  if (detected != candidate)
  {
    candidate = detected;
    candidateSince = millis();
  }
  else if (candidate != current && millis() - candidateSince > settleTime)
  {
    current = candidate;
    changes = changes + 1;
    Serial.print("Orientation: ");
    Serial.println(orientations[current]);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("ORIENTATION");
  display.setTextSize(2);
  display.setCursor(0, 14);
```

```
  display.print(current == 0 ? "TILTED" : orientations[current]);
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print("X ");
  display.print(x, 1);
  display.print(" Y ");
  display.print(y, 1);
  display.print(" Z ");
  display.print(z, 1);
  display.setCursor(0, 48);
  display.print("Changes: ");
  display.print(changes);
  // Simple orientation icon
  if (current == 1 || current == 2) display.drawRect(100, 42, 14, 20, SSD1306_WHITE);
  else if (current == 3 || current == 4) display.drawRect(96, 46, 22, 14, SSD1306_WHITE);
  else if (current >= 5) display.fillRect(100, 46, 18, 14, SSD1306_WHITE);
  display.display();
  delay(50);
}
```

What you should see: Flat on the table: FACE UP with Z near 9.8. Stand it upright: PORTRAIT and the gravity moves to the Y axis. Turn it on its side: LANDSCAPE L or LANDSCAPE R. Hold it at 45 degrees and it honestly reports TILTED rather than guessing.

Digital Protractor

Goal: Measure the angle between two surfaces by taking a reference reading on the first and then reading the second.

Absolute angles are measured from the horizontal, but a carpenter usually wants the angle between two things, which is a relative measurement. Press the reference button against the first surface and its angle is stored; every reading after that is the difference from it. The arc drawn on screen makes the number tangible, and it is drawn with the same sin and cos work as the clock in Experiment 6.

Online page with troubleshooting: geekobot.in/projects/titan-kit-068-digital-protractor

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Relative angle computation
- Reference-based measurement
- Normalising an angle into a −180 to 180 range
- Drawing an angle as an arc

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Set reference	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Hold reading	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int referenceButton = 12;
int holdButton = 14;
float angle = 0;
float reference = 0;
bool hasReference = false;
bool holding = false;
float heldAngle = 0;
float readAngle()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  return atan2(a.acceleration.y, a.acceleration.z) * 180.0 / PI;
}
void setup()
{
  pinMode(referenceButton, INPUT_PULLUP);
  pinMode(holdButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}
void loop()
{
  angle = angle * 0.85 + readAngle() * 0.15;
  if (digitalRead(referenceButton) == LOW)
  {
    reference = angle;
    hasReference = true;
    Serial.print("Reference set at ");
    Serial.println(reference, 1);
    delay(400);
  }
  if (digitalRead(holdButton) == LOW)
  {
    holding = !holding;
    if (holding)
    {
      heldAngle = angle;
    }
    delay(400);
  }
  float shown = holding ? heldAngle : angle;
  float relative = shown - reference;
  // Keep the answer in -180..180
  while (relative > 180) relative = relative - 360;
  while (relative < -180) relative = relative + 360;
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("PROTRACTOR");
  if (holding)
  {
    display.setCursor(88, 0);
    display.print("HELD");
  }
  display.setTextSize(3);
  display.setCursor(0, 14);
```

```
  display.print(abs(relative), 0);
  display.setTextSize(1);
  display.print((char)247);
  display.setCursor(0, 42);
  if (hasReference)
  {
    display.print("Ref ");
    display.print(reference, 0);
    display.print((char)247);
    display.print(" abs ");
    display.print(shown, 0);
    display.print((char)247);
  }
  else
  {
    display.print("Press D3 on surface 1");
  }
  display.setCursor(0, 54);
  display.print("D3 ref D4 hold");
  // Draw the angle as an arc
  int cx = 100;
  int cy = 34;
  display.drawLine(cx - 22, cy, cx + 22, cy, SSD1306_WHITE);
  float radians = -relative * PI / 180.0;
  display.drawLine(cx, cy,
  cx + cos(radians) * 22,
  cy + sin(radians) * 22, SSD1306_WHITE);
  display.drawCircle(cx, cy, 2, SSD1306_WHITE);
  display.display();
  delay(40);
}
```

What you should see: Hold the board against one surface and press the first button to set the reference. Move it to the second surface: the angle between them appears in large digits. The arc on the right shows the angle graphically. Press the second button to freeze the reading before taking the board away.

Motion Based Password

Goal: Unlock the system with a secret sequence of tilts, a gesture password rather than a typed one.

This combines two earlier ideas: gesture recognition from Experiment 33 and sequence matching from the touch password. Each tilt is converted to a direction number and appended to an entered array, then compared with the secret. A gesture only counts once the board has returned to flat, which forces a deliberate rhythm, tilt, return, tilt, and prevents one long lean from filling the whole sequence.

Online page with troubleshooting: geekobot.in/projects/titan-kit-069-motion-password

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Sequence matching with IMUs
- Requiring a return to neutral between gestures
- Combining gesture recognition with authentication
- Masking an entry while it is being made

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Red LED anode	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	
Button	Clear button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#define LENGTH 4
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int greenLed = 27;
int redLed = 26;
int buzzer = 33;
int clearButton = 12;
// 1 up, 2 down, 3 left, 4 right
int secret[LENGTH] = {4, 3, 1, 2};
int entered[LENGTH];
int position = 0;
const char* names[5] = {"-", "UP", "DOWN", "LEFT", "RIGHT"};
bool atRest = true;
bool unlocked = false;
unsigned long unlockedAt = 0;
int readGesture(float pitch, float roll)
{
  if (pitch > 30) return 1;
  if (pitch < -30) return 2;
  if (roll < -30) return 3;
  if (roll > 30) return 4;
  return 0;
}
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(clearButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float pitch = atan2(a.acceleration.y,
  sqrt(a.acceleration.x * a.acceleration.x +
  a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
  float roll = atan2(-a.acceleration.x, a.acceleration.z) * 180.0 / PI;
  int gesture = readGesture(pitch, roll);
  if (digitalRead(clearButton) == LOW)
  {
    position = 0;
    unlocked = false;
    Serial.println("Entry cleared");
    delay(300);
  }
  // Must return to flat between gestures
  if (abs(pitch) < 12 && abs(roll) < 12)
  {
    atRest = true;
  }
  if (gesture > 0 && atRest && !unlocked)
  {
    atRest = false;
    entered[position] = gesture;
    position = position + 1;
```

```
    tone(buzzer, 900 + gesture * 100, 80);
    Serial.print("Gesture ");
    Serial.println(names[gesture]);
    if (position >= LENGTH)
    {
      bool correct = true;
      for (int i = 0; i < LENGTH; i++)
      {
        if (entered[i] != secret[i]) correct = false;
      }
      if (correct)
      {
        unlocked = true;
        unlockedAt = millis();
        digitalWrite(greenLed, HIGH);
        tone(buzzer, 1600, 400);
        Serial.println("UNLOCKED");
      }
      else
      {
        digitalWrite(redLed, HIGH);
        tone(buzzer, 200, 500);
        Serial.println("WRONG SEQUENCE");
        delay(900);
        digitalWrite(redLed, LOW);
      }
      position = 0;
    }
  }
  if (unlocked && millis() - unlockedAt > 5000)
  {
    unlocked = false;
    digitalWrite(greenLed, LOW);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("GESTURE LOCK");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print(unlocked ? "UNLOCKED" : "LOCKED");
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print("Entered: ");
  for (int i = 0; i < LENGTH; i++)
  {
    display.print(i < position ? "*" : ".");
    display.print(" ");
  }
  display.setCursor(0, 46);
  display.print("Now: ");
  display.print(names[gesture]);
  display.print(atRest ? " (ready)" : " (return flat)");
  display.setCursor(0, 56);
  display.print("D3 clears entry");
  display.display();
  delay(40);
}
```

What you should see: The screen shows LOCKED with four dots for the sequence. Tilt right, return flat, tilt left, return flat, and so on. Each accepted gesture adds a star and plays a different tone. The correct four-gesture sequence gives UNLOCKED with the green LED.

Gesture Controlled Menu Navigation

Goal: Scroll and select items in an OLED menu purely by tilting, with no buttons at all.

Tilting forward and back moves the highlight; holding a right tilt for half a second selects. That hold-to-confirm gesture solves the hardest problem in a button-free interface, telling a deliberate choice from a passing movement. The menu itself scrolls: only three of the five items fit on screen, so a window offset follows the selection, which is how every scrolling list works internally.

Online page with troubleshooting: geekobot.in/projects/titan-kit-070-gesture-controlled-menu-navigation

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- UI navigation via motion
- Hold-to-confirm as a selection gesture
- Scrolling windows over a longer list
- Highlighting with inverted text

Wiring

Part	Pin	ESP32	Note
MPU6050	VCC	**3V3 (module has a regulator)**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21 (shared with the OLED)**	
MPU6050	SCL	**GPIO22 (shared with the OLED)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED 1	**GPIO27**	Through a 220 ohm resistor
LED	LED 2	**GPIO26**	Through a 220 ohm resistor
LED	LED 3	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#define ITEMS 5
#define VISIBLE 3
Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_MPU6050 mpu;
int ledPins[3] = {27, 26, 25};
int buzzer = 33;
const char* menu[ITEMS] =
{
  "Light 1", "Light 2", "Light 3", "All On", "All Off"
};
int selected = 0;
int windowTop = 0;
bool atRest = true;
unsigned long selectStart = 0;
bool selecting = false;
unsigned long holdNeeded = 600;
String lastAction = "none";
void doAction(int index)
{
  if (index < 3)
  {
    digitalWrite(ledPins[index], !digitalRead(ledPins[index]));
    lastAction = String(menu[index]) + " toggled";
  }
  else if (index == 3)
  {
    for (int i = 0; i < 3; i++) digitalWrite(ledPins[i], HIGH);
    lastAction = "All lights on";
  }
  else
  {
    for (int i = 0; i < 3; i++) digitalWrite(ledPins[i], LOW);
    lastAction = "All lights off";
  }
  tone(buzzer, 1400, 150);
  Serial.println(lastAction);
}
void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  if (!mpu.begin())
  {
    display.clearDisplay();
    display.setCursor(0, 28);
    display.print("MPU6050 not found");
    display.display();
    while (true) { }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}
void loop()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float pitch = atan2(a.acceleration.y,
  sqrt(a.acceleration.x * a.acceleration.x +
  a.acceleration.z * a.acceleration.z)) * 180.0 / PI;
  float roll = atan2(-a.acceleration.x, a.acceleration.z) * 180.0 / PI;
  if (abs(pitch) < 12)
  {
```

```
    atRest = true;
  }
  if (atRest && pitch < -25)
  {
    selected = (selected + 1) % ITEMS;
    atRest = false;
    tone(buzzer, 800, 40);
  }
  else if (atRest && pitch > 25)
  {
    selected = (selected - 1 + ITEMS) % ITEMS;
    atRest = false;
    tone(buzzer, 800, 40);
  }
  // Keep the selection inside the visible window
  if (selected < windowTop) windowTop = selected;
  if (selected > windowTop + VISIBLE - 1) windowTop = selected - VISIBLE + 1;
  // Hold a right tilt to select
  if (roll > 30)
  {
    if (!selecting)
    {
      selecting = true;
      selectStart = millis();
    }
    else if (millis() - selectStart > holdNeeded)
    {
      doAction(selected);
      selecting = false;
      delay(500);
    }
  }
  else
  {
    selecting = false;
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("TILT MENU");
  for (int row = 0; row < VISIBLE; row++)
  {
    int item = windowTop + row;
    if (item >= ITEMS) break;
    int y = 14 + row * 12;
    if (item == selected)
    {
      display.fillRect(0, y - 2, 128, 12, SSD1306_WHITE);
      display.setTextColor(SSD1306_BLACK);
    }
    display.setCursor(4, y);
    display.print(item + 1);
    display.print(". ");
    display.print(menu[item]);
    display.setTextColor(SSD1306_WHITE);
  }
  // Scroll position indicator
  int barHeight = 36 * VISIBLE / ITEMS;
  int barY = 12 + (36 - barHeight) * windowTop / (ITEMS - VISIBLE);
  display.drawRect(124, 12, 4, 36, SSD1306_WHITE);
  display.fillRect(125, barY, 2, barHeight, SSD1306_WHITE);
  display.setCursor(0, 54);
  if (selecting)
  {
    display.print("Selecting... hold");
  }
  else
  {
    display.print(lastAction);
  }
  display.display();
  delay(40);
}
```

What you should see: A menu of five items with the current one highlighted. Tilt forward or back to move the highlight, one item per tilt. A scroll bar on the right shows your position in the list. Hold a right tilt for half a second to activate the selected item.

RFID Based Library Management Demo

Goal: Model a library counter: tap a book's tag to check it out, tap it again to return it, and keep a live record of what is on loan.

This is a database in miniature. Three parallel arrays act as the columns of a table, UID, title, and whether the item is out on loan, and the row index ties them together. Looking up a tag means searching the UID column for a match and remembering where it was found; everything else about that book is then one array access away. Understanding that a record is a row across several arrays is the step towards understanding real database tables.

Online page with troubleshooting: geekobot.in/projects/titan-kit-071-rfid-library-management

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Database structure simulation
- Parallel arrays as table columns
- Searching a column and using the row index
- Toggling a record's state on each scan

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Red LED anode	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define BOOKS 4
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int greenLed = 27;
int redLed = 26;
int buzzer = 33;
// The three columns of our table
String bookUID[BOOKS] = {"A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD", "12 34 56 78"};
String bookName[BOOKS] = {"Physics Vol 1", "World Atlas", "Short Stories", "Maths Guide"};
bool onLoan[BOOKS] = {false, false, false, false};
int loansOut = 0;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findBook(String uid)
{
  for (int i = 0; i < BOOKS; i++)
  {
    if (bookUID[i] == uid) return i;
  }
  return -1;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("LIBRARY DESK");
  display.setCursor(0, 14);
  display.print("Tap a book tag");
  display.setCursor(0, 30);
  display.print("On loan: ");
  display.print(loansOut);
  display.print(" of ");
  display.print(BOOKS);
  int y = 42;
  for (int i = 0; i < BOOKS; i++)
  {
    if (onLoan[i])
    {
      display.setCursor(0, y);
      display.print("OUT: ");
      display.print(bookName[i]);
      y = y + 10;
      if (y > 56) break;
    }
  }
  display.display();
}
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
```

```
  display.setTextColor(SSD1306_WHITE);
  showIdle();
  Serial.println("Library system ready");
}
void loop()
{
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int index = findBook(uid);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  if (index < 0)
  {
    display.print("UNKNOWN TAG");
    display.setCursor(0, 20);
    display.print(uid);
    display.setCursor(0, 40);
    display.print("Not in catalogue");
    digitalWrite(redLed, HIGH);
    tone(buzzer, 250, 400);
    Serial.print("Unknown tag: ");
    Serial.println(uid);
  }
  else
  {
    onLoan[index] = !onLoan[index];
    loansOut = loansOut + (onLoan[index] ? 1 : -1);
    display.print(onLoan[index] ? "CHECKED OUT" : "RETURNED");
    display.setTextSize(1);
    display.setCursor(0, 22);
    display.print(bookName[index]);
    display.setCursor(0, 40);
    display.print("Now on loan: ");
    display.print(loansOut);
    digitalWrite(greenLed, HIGH);
    tone(buzzer, 1400, 150);
    Serial.print(bookName[index]);
    Serial.println(onLoan[index] ? " checked out" : " returned");
  }
  display.display();
  delay(2000);
  digitalWrite(greenLed, LOW);
  digitalWrite(redLed, LOW);
  showIdle();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: The idle screen lists how many books are out and which ones. Tap a registered tag: CHECKED OUT with the book title. Tap the same tag again: RETURNED and the count drops. An unregistered tag shows UNKNOWN TAG with its UID so you can add it.

RFID Product Tracking System

Goal: Track stock through a warehouse by moving items between named states with a mode button.

Every item carries a state, and the scanner's current mode decides which state a scanned tag moves to. That separation, the reader does not decide, the operator's mode does, is how real warehouse scanners work, and it means one device handles receiving, storage, picking and dispatch. Counting how many items sit in each state then gives an instant stock summary without storing any history.

Online page with troubleshooting: geekobot.in/projects/titan-kit-072-rfid-product-tracking

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Asset state management
- Operator mode deciding the outcome of a scan
- Counting items by state
- Showing a before-and-after transition

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Mode button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define ITEMS 5
#define STATES 4
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int modeButton = 12;
int buzzer = 33;
String itemUID[ITEMS] = {"A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD",
  "12 34 56 78", "DE AD BE EF"};
String itemName[ITEMS] = {"Box A", "Box B", "Box C", "Box D", "Box E"};
int itemState[ITEMS] = {0, 0, 0, 0, 0};
const char* stateName[STATES] = {"RECEIVED", "IN STORE", "PICKED", "DISPATCHED"};
int mode = 0;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findItem(String uid)
{
  for (int i = 0; i < ITEMS; i++)
  {
    if (itemUID[i] == uid) return i;
  }
  return -1;
}
int countInState(int state)
{
  int total = 0;
  for (int i = 0; i < ITEMS; i++)
  {
    if (itemState[i] == state) total = total + 1;
  }
  return total;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("STOCK TRACKER");
  display.fillRect(0, 12, 128, 12, SSD1306_WHITE);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(4, 14);
  display.print("Mode: ");
  display.print(stateName[mode]);
  display.setTextColor(SSD1306_WHITE);
  for (int s = 0; s < STATES; s++)
  {
    display.setCursor(0, 28 + s * 9);
    display.print(stateName[s]);
    display.setCursor(96, 28 + s * 9);
    display.print(countInState(s));
  }
  display.display();
}
void setup()
{
  pinMode(modeButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
```

```
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showIdle();
}
void loop()
{
  if (digitalRead(modeButton) == LOW)
  {
    mode = (mode + 1) % STATES;
    tone(buzzer, 1000 + mode * 200, 80);
    Serial.print("Mode: ");
    Serial.println(stateName[mode]);
    showIdle();
    delay(300);
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int index = findItem(uid);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  if (index < 0)
  {
    display.print("UNKNOWN ITEM");
    display.setCursor(0, 24);
    display.print(uid);
    tone(buzzer, 250, 400);
  }
  else
  {
    int oldState = itemState[index];
    itemState[index] = mode;
    display.print("SCANNED");
    display.setTextSize(2);
    display.setCursor(0, 14);
    display.print(itemName[index]);
    display.setTextSize(1);
    display.setCursor(0, 36);
    display.print(stateName[oldState]);
    display.print(" ->");
    display.setCursor(0, 48);
    display.print(" ");
    display.print(stateName[mode]);
    tone(buzzer, 1400, 120);
    Serial.print(itemName[index]);
    Serial.print(": ");
    Serial.print(stateName[oldState]);
    Serial.print(" -> ");
    Serial.println(stateName[mode]);
  }
  display.display();
  delay(1800);
  showIdle();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: The idle screen shows the current mode and a count for each state. Press the button to cycle through RECEIVED, IN STORE, PICKED and DISPATCHED. Scan a tag and it shows the old state moving to the new one. The counts update immediately on the idle screen.

RFID Personal Profile Viewer

Goal: Show a person's full details on screen when their tag is scanned, cycling through more information than fits on one page.

A UID lookup table maps a meaningless string of hex bytes to a person. The tag itself stores no personal data at all, it only carries an identifier, and everything else lives in the system. That is a deliberate design choice with a real privacy benefit: losing the card reveals nothing. Because the profile is longer than the screen, the details are split into pages that the button steps through.

Online page with troubleshooting: geekobot.in/projects/titan-kit-073-rfid-personal-profile-viewer

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- UID lookup tables
- Why the tag stores an identifier, not personal data
- Paging content that exceeds the screen
- Timing out a display for privacy

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	One leg	**GPIO12**	GPIO12 is a boot pin: if the board will not start, unplug this wire while it powers up
Button	Other leg	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define PEOPLE 3
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int pageButton = 12;
String personUID[PEOPLE] = {"A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD"};
String name_[PEOPLE] = {"ANJALI R", "RAHUL K", "PRIYA S"};
String role[PEOPLE] = {"Class 10-A", "Class 10-B", "Class 9-C"};
String idNumber[PEOPLE] = {"STU-1042", "STU-1043", "STU-0987"};
String house[PEOPLE] = {"Blue House", "Red House", "Green House"};
String contact[PEOPLE] = {"Ext 210", "Ext 214", "Ext 118"};
int current = -1;
int page = 0;
unsigned long shownAt = 0;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findPerson(String uid)
{
  for (int i = 0; i < PEOPLE; i++)
  {
    if (personUID[i] == uid) return i;
  }
  return -1;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("PROFILE VIEWER");
  display.setTextSize(2);
  display.setCursor(0, 20);
  display.print("SCAN ID");
  display.setTextSize(1);
  display.setCursor(0, 48);
  display.print("Hold your card to");
  display.setCursor(0, 56);
  display.print("the reader");
  display.display();
}
void showProfile()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.fillRect(0, 0, 128, 12, SSD1306_WHITE);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(3, 2);
  display.print(name_[current]);
  display.setCursor(112, 2);
  display.print(page + 1);
  display.print("/2");
  display.setTextColor(SSD1306_WHITE);
  if (page == 0)
  {
    display.setCursor(0, 18);
    display.print("Class:");
    display.setCursor(48, 18);
    display.print(role[current]);
```

```
    display.setCursor(0, 30);
    display.print("ID no:");
    display.setCursor(48, 30);
    display.print(idNumber[current]);
    display.setCursor(0, 42);
    display.print("House:");
    display.setCursor(48, 42);
    display.print(house[current]);
  }
  else
  {
    display.setCursor(0, 18);
    display.print("Contact:");
    display.setCursor(0, 30);
    display.print(contact[current]);
    display.setCursor(0, 44);
    display.print("Card UID:");
    display.setCursor(0, 54);
    display.print(personUID[current]);
  }
  display.display();
}
void setup()
{
  pinMode(pageButton, INPUT_PULLUP);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showIdle();
}
void loop()
{
  if (current >= 0)
  {
    if (digitalRead(pageButton) == LOW)
    {
      page = (page + 1) % 2;
      shownAt = millis();
      showProfile();
      delay(300);
    }
    if (millis() - shownAt > 12000)
    {
      current = -1;
      showIdle();
    }
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int index = findPerson(uid);
  if (index < 0)
  {
    display.clearDisplay();
    display.setCursor(0, 0);
    display.print("CARD NOT REGISTERED");
    display.setCursor(0, 24);
    display.print(uid);
    display.setCursor(0, 48);
    display.print("Add this UID to");
    display.setCursor(0, 56);
    display.print("the personUID array");
    display.display();
    Serial.print("Unregistered: ");
    Serial.println(uid);
    delay(3000);
    showIdle();
  }
  else
  {
    current = index;
    page = 0;
    shownAt = millis();
```

```
    showProfile();
    Serial.print("Profile shown: ");
    Serial.println(name_[index]);
  }
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: SCAN ID on the idle screen. Tap a registered card: the name appears in a header bar with class, ID and house below. Press the button for page 2 showing contact details and the card UID. The profile clears itself after 12 seconds.

RFID Equipment Checkout System

Goal: Log exactly who borrowed which tool and when, by scanning a user card followed by an equipment tag.

Two scans make one transaction, so the sketch has to remember what it is waiting for. That is a state machine with two states, awaiting user, awaiting equipment, and it will not accept an equipment tag until a user has identified themselves. Each completed transaction records the borrower and the elapsed time since power-up, which is the simplest form of timestamp available without a clock module.

Online page with troubleshooting: geekobot.in/projects/titan-kit-074-rfid-equipment-checkout

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Timestamping and logging
- Two-scan transactions as a state machine
- Refusing an action until a prerequisite is met
- Session timeouts to avoid stale state

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Red LED anode	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define USERS 3
#define TOOLS 4
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int greenLed = 27;
int redLed = 26;
int buzzer = 33;
String userUID[USERS] = {"A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD"};
String userName[USERS] = {"ANJALI", "RAHUL", "PRIYA"};
String toolUID[TOOLS] = {"12 34 56 78", "DE AD BE EF", "CA FE 00 01", "BE EF 00 02"};
String toolName[TOOLS] = {"Multimeter", "Soldering Kit", "Oscilloscope", "Power Supply"};
int heldBy[TOOLS] = {-1, -1, -1, -1};
unsigned long takenAt[TOOLS] = {0, 0, 0, 0};
int waitingUser = -1;
unsigned long userScannedAt = 0;
unsigned long userTimeout = 15000;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findUser(String uid)
{
  for (int i = 0; i < USERS; i++) if (userUID[i] == uid) return i;
  return -1;
}
int findTool(String uid)
{
  for (int i = 0; i < TOOLS; i++) if (toolUID[i] == uid) return i;
  return -1;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("EQUIPMENT DESK");
  if (waitingUser >= 0)
  {
    display.setCursor(0, 14);
    display.print("User: ");
    display.print(userName[waitingUser]);
    display.setCursor(0, 26);
    display.print("Now scan a tool");
    display.setCursor(0, 38);
    display.print("Cancels in ");
    display.print((userTimeout - (millis() - userScannedAt)) / 1000);
    display.print("s");
  }
  else
  {
    display.setCursor(0, 14);
    display.print("Scan your ID card");
    int out = 0;
    for (int i = 0; i < TOOLS; i++) if (heldBy[i] >= 0) out = out + 1;
    display.setCursor(0, 30);
    display.print("Tools out: ");
    display.print(out);
    display.print(" of ");
    display.print(TOOLS);
```

```
    int y = 42;
    for (int i = 0; i < TOOLS && y <= 56; i++)
    {
      if (heldBy[i] >= 0)
      {
        display.setCursor(0, y);
        display.print(toolName[i].substring(0, 10));
        display.print(" - ");
        display.print(userName[heldBy[i]]);
        y = y + 10;
      }
    }
  }
  display.display();
}
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showIdle();
}
void loop()
{
  if (waitingUser >= 0)
  {
    if (millis() - userScannedAt > userTimeout)
    {
      waitingUser = -1;
      Serial.println("User session timed out");
    }
    showIdle();
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int user = findUser(uid);
  int tool = findTool(uid);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  if (user >= 0)
  {
    waitingUser = user;
    userScannedAt = millis();
    display.print("USER IDENTIFIED");
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print(userName[user]);
    display.setTextSize(1);
    display.setCursor(0, 48);
    display.print("Now scan a tool");
    digitalWrite(greenLed, HIGH);
    tone(buzzer, 1200, 100);
    Serial.print("User: ");
    Serial.println(userName[user]);
  }
  else if (tool >= 0 && waitingUser < 0)
  {
    display.print("SCAN ID FIRST");
    display.setCursor(0, 24);
    display.print(toolName[tool]);
    display.setCursor(0, 44);
    display.print("Who are you?");
    digitalWrite(redLed, HIGH);
    tone(buzzer, 300, 300);
  }
  else if (tool >= 0)
  {
    if (heldBy[tool] < 0)
```

```
    {
      heldBy[tool] = waitingUser;
      takenAt[tool] = millis();
      display.print("CHECKED OUT");
      display.setTextSize(1);
      display.setCursor(0, 20);
      display.print(toolName[tool]);
      display.setCursor(0, 34);
      display.print("to ");
      display.print(userName[waitingUser]);
      display.setCursor(0, 48);
      display.print("Time: ");
      display.print(millis() / 1000);
      display.print("s uptime");
      Serial.print(toolName[tool]);
      Serial.print(" taken by ");
      Serial.print(userName[waitingUser]);
      Serial.print(" at ");
      Serial.println(millis() / 1000);
    }
    else
    {
      unsigned long held = (millis() - takenAt[tool]) / 1000;
      display.print("RETURNED");
      display.setCursor(0, 20);
      display.print(toolName[tool]);
      display.setCursor(0, 34);
      display.print("was with ");
      display.print(userName[heldBy[tool]]);
      display.setCursor(0, 48);
      display.print("Held for ");
      display.print(held);
      display.print("s");
      Serial.print(toolName[tool]);
      Serial.print(" returned after ");
      Serial.print(held);
      Serial.println(" seconds");
      heldBy[tool] = -1;
    }
    digitalWrite(greenLed, HIGH);
    tone(buzzer, 1500, 150);
    waitingUser = -1;
  }
  else
  {
    display.print("UNKNOWN TAG");
    display.setCursor(0, 24);
    display.print(uid);
    digitalWrite(redLed, HIGH);
    tone(buzzer, 250, 400);
  }
  display.display();
  delay(2200);
  digitalWrite(greenLed, LOW);
  digitalWrite(redLed, LOW);
  showIdle();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: Scan an ID card: USER IDENTIFIED with the borrower's name. A 15-second countdown appears while it waits for a tool. Scan a tool tag: CHECKED OUT recording who took it. Scan the same tool again: RETURNED with how long it was held.

RFID Access Log Recorder

Goal: Keep a rolling log of the last ten tags that entered a secured area, newest first.

A ring buffer stores a fixed number of entries and overwrites the oldest when it is full, using a write position that wraps with %. Nothing is ever shifted, which matters when entries arrive quickly. Reading it back in newest-first order takes a little index arithmetic, start at the write position, step backwards, and wrap around, and that arithmetic is the whole reason ring buffers are worth learning properly.

Online page with troubleshooting: geekobot.in/projects/titan-kit-075-rfid-access-log-recorder

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Array ring buffers
- Wrapping a write position with %
- Reading a circular buffer in reverse order
- Paging through a log on a small screen

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Scroll button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define LOG_SIZE 10
#define KNOWN 3
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int scrollButton = 12;
int buzzer = 33;
String knownUID[KNOWN] = {"A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD"};
String knownName[KNOWN] = {"ANJALI", "RAHUL", "PRIYA"};
// Ring buffer
String logName[LOG_SIZE];
unsigned long logTime[LOG_SIZE];
int writePosition = 0;
int entries = 0;
int viewOffset = 0;
int totalScans = 0;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
String identify(String uid)
{
  for (int i = 0; i < KNOWN; i++)
  {
    if (knownUID[i] == uid) return knownName[i];
  }
  return "UNKNOWN";
}
void addEntry(String who)
{
  logName[writePosition] = who;
  logTime[writePosition] = millis() / 1000;
  writePosition = (writePosition + 1) % LOG_SIZE;
  if (entries < LOG_SIZE) entries = entries + 1;
  totalScans = totalScans + 1;
}
void showLog()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.fillRect(0, 0, 128, 11, SSD1306_WHITE);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(2, 2);
  display.print("ACCESS LOG total ");
  display.print(totalScans);
  display.setTextColor(SSD1306_WHITE);
  if (entries == 0)
  {
    display.setCursor(0, 28);
    display.print("No entries yet");
    display.display();
    return;
  }
  // Walk backwards from the newest
  for (int row = 0; row < 4; row++)
  {
    int which = viewOffset + row;
    if (which >= entries) break;
    int index = (writePosition - 1 - which + LOG_SIZE * 2) % LOG_SIZE;
    int y = 14 + row * 11;
```

```
    display.setCursor(0, y);
    display.print(which + 1);
    display.print(". ");
    display.print(logName[index]);
    display.setCursor(80, y);
    display.print(logTime[index]);
    display.print("s");
  }
  display.setCursor(0, 56);
  display.print("D3 scrolls ");
  display.print(viewOffset + 1);
  display.print("-");
  display.print(min(viewOffset + 4, entries));
  display.print(" of ");
  display.print(entries);
  display.display();
}
void setup()
{
  pinMode(scrollButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showLog();
}
void loop()
{
  if (digitalRead(scrollButton) == LOW)
  {
    viewOffset = viewOffset + 4;
    if (viewOffset >= entries) viewOffset = 0;
    showLog();
    delay(300);
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  String who = identify(uid);
  addEntry(who);
  viewOffset = 0;
  tone(buzzer, who == "UNKNOWN" ? 300 : 1400, 150);
  Serial.print("Logged: ");
  Serial.print(who);
  Serial.print(" at ");
  Serial.print(millis() / 1000);
  Serial.println("s");
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("ENTRY RECORDED");
  display.setTextSize(2);
  display.setCursor(0, 22);
  display.print(who);
  display.setTextSize(1);
  display.setCursor(0, 50);
  display.print("Entry ");
  display.print(totalScans);
  display.display();
  delay(1500);
  showLog();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: Each scan flashes ENTRY RECORDED with the person's name. The log lists the four most recent entries, newest at the top. Press the button to page back through older entries. After ten entries the oldest is overwritten automatically.

RFID Attendance Analytics

Goal: Register attendance so each person can only be counted once, and report who is present and who is missing.

A single boolean per person is the entire data structure, and it answers every question asked of it. Counting the trues gives the number present, counting the falses gives absentees, and the percentage is one division away. Checking that flag before registering is what makes the system idempotent, scanning twice has the same effect as scanning once, which is exactly what you want when a queue of students taps in quickly.

Online page with troubleshooting: geekobot.in/projects/titan-kit-076-rfid-attendance-analytics

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Boolean state toggling for check-ins
- Idempotent operations, repeating has no extra effect
- Deriving several statistics from one flag array
- Multiple views over the same data

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	View button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Reset button	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define STUDENTS 6
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int viewButton = 12;
int resetButton = 14;
int buzzer = 33;
String studentUID[STUDENTS] =
{
  "A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD",
  "12 34 56 78", "DE AD BE EF", "CA FE 00 01"
};
String studentName[STUDENTS] =
{
  "ANJALI", "RAHUL", "PRIYA", "AMIT", "SNEHA", "VIKAS"
};
bool present[STUDENTS];
unsigned long arrivedAt[STUDENTS];
int view = 0; // 0 summary, 1 present list, 2 absent list
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findStudent(String uid)
{
  for (int i = 0; i < STUDENTS; i++)
  {
    if (studentUID[i] == uid) return i;
  }
  return -1;
}
int countPresent()
{
  int total = 0;
  for (int i = 0; i < STUDENTS; i++) if (present[i]) total = total + 1;
  return total;
}
void showView()
{
  display.clearDisplay();
  display.setTextSize(1);
  int here = countPresent();
  display.fillRect(0, 0, 128, 11, SSD1306_WHITE);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(2, 2);
  display.print("ATTENDANCE ");
  display.print(here);
  display.print("/");
  display.print(STUDENTS);
  display.setTextColor(SSD1306_WHITE);
  if (view == 0)
  {
    int percent = here * 100 / STUDENTS;
    display.setTextSize(3);
    display.setCursor(0, 18);
    display.print(percent);
    display.setTextSize(1);
    display.print("%");
    display.setCursor(64, 22);
    display.print("Present ");
```

```
    display.print(here);
    display.setCursor(64, 32);
    display.print("Absent ");
    display.print(STUDENTS - here);
    display.drawRect(0, 48, 128, 10, SSD1306_WHITE);
    display.fillRect(1, 49, map(percent, 0, 100, 0, 126), 8, SSD1306_WHITE);
  }
  else
  {
    bool wanted = (view == 1);
    display.setCursor(0, 14);
    display.print(wanted ? "PRESENT:" : "ABSENT:");
    int y = 26;
    for (int i = 0; i < STUDENTS && y <= 56; i++)
    {
      if (present[i] == wanted)
      {
        display.setCursor(0, y);
        display.print(studentName[i]);
        if (wanted)
        {
          display.setCursor(80, y);
          display.print(arrivedAt[i]);
          display.print("s");
        }
        y = y + 10;
      }
    }
    if (y == 26)
    {
      display.setCursor(0, 30);
      display.print(wanted ? "Nobody yet" : "Everyone is here");
    }
  }
  display.display();
}
void setup()
{
  pinMode(viewButton, INPUT_PULLUP);
  pinMode(resetButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  for (int i = 0; i < STUDENTS; i++)
  {
    present[i] = false;
    arrivedAt[i] = 0;
  }
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showView();
}
void loop()
{
  if (digitalRead(viewButton) == LOW)
  {
    view = (view + 1) % 3;
    showView();
    delay(300);
  }
  if (digitalRead(resetButton) == LOW)
  {
    for (int i = 0; i < STUDENTS; i++) present[i] = false;
    Serial.println("New day - attendance cleared");
    tone(buzzer, 600, 300);
    showView();
    delay(400);
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int index = findStudent(uid);
  display.clearDisplay();
  display.setTextSize(1);
```

```
  display.setCursor(0, 0);
  if (index < 0)
  {
    display.print("CARD NOT ON ROLL");
    display.setCursor(0, 28);
    display.print(uid);
    tone(buzzer, 250, 400);
  }
  else if (present[index])
  {
    display.print("ALREADY MARKED");
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print(studentName[index]);
    display.setTextSize(1);
    display.setCursor(0, 48);
    display.print("Arrived at ");
    display.print(arrivedAt[index]);
    display.print("s");
    tone(buzzer, 700, 200);
    Serial.print(studentName[index]);
    Serial.println(" scanned again - ignored");
  }
  else
  {
    present[index] = true;
    arrivedAt[index] = millis() / 1000;
    display.print("WELCOME");
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print(studentName[index]);
    display.setTextSize(1);
    display.setCursor(0, 48);
    display.print("Present: ");
    display.print(countPresent());
    display.print("/");
    display.print(STUDENTS);
    tone(buzzer, 1500, 150);
    Serial.print(studentName[index]);
    Serial.println(" marked present");
  }
  display.display();
  delay(1800);
  showView();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: A summary screen showing the attendance percentage and a bar. Tap a card: WELCOME with the name and an updated count. Tap the same card again: ALREADY MARKED, and the count does not change. Press the view button to list who is present and who is absent.

RFID Multi-User Authentication

Goal: Require two different authorised cards within a short window before granting access, the two-person rule.

Requiring two authorisations is a genuine security pattern, used for bank vaults and launch systems, and it defends against a single lost card. Two conditions must both hold: the second card must be authorised, and it must be a different card from the first. Checking that the UIDs differ is what stops one person scanning their own card twice, an obvious attack that a careless implementation would allow.

Online page with troubleshooting: geekobot.in/projects/titan-kit-077-rfid-multi-user-authentication

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Multi-condition authorization
- The two-person rule as a security pattern
- Checking that two identifiers differ
- Windows that expire to limit an authorisation

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Servo	Servo signal	**GPIO26**	
Servo	Servo red / brown	**5V / GND**	
LED	Green LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Red LED anode	**GPIO14**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define AUTHORISED 4
MFRC522 rfid(SS_PIN, RST_PIN);
Servo lock;
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int greenLed = 27;
int redLed = 14;
int buzzer = 33;
String authUID[AUTHORISED] =
{
  "A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD", "12 34 56 78"
};
String authName[AUTHORISED] = {"ANJALI", "RAHUL", "PRIYA", "AMIT"};
String firstUID = "";
String firstName = "";
unsigned long firstScanAt = 0;
unsigned long pairWindow = 20000;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findAuth(String uid)
{
  for (int i = 0; i < AUTHORISED; i++)
  {
    if (authUID[i] == uid) return i;
  }
  return -1;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("DUAL AUTH LOCK");
  if (firstUID == "")
  {
    display.setTextSize(2);
    display.setCursor(0, 18);
    display.print("LOCKED");
    display.setTextSize(1);
    display.setCursor(0, 44);
    display.print("Two cards required");
    display.setCursor(0, 54);
    display.print("Scan the first");
  }
  else
  {
    display.setCursor(0, 14);
    display.print("1st: ");
    display.print(firstName);
    display.print(" OK");
    display.setTextSize(2);
    display.setCursor(0, 26);
    display.print("2ND CARD");
    display.setTextSize(1);
    display.setCursor(0, 50);
    display.print("Expires in ");
    display.print((pairWindow - (millis() - firstScanAt)) / 1000);
```

```
    display.print("s");
  }
  display.display();
}
void grantAccess(String secondName)
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("ACCESS GRANTED");
  display.setCursor(0, 20);
  display.print("1st: ");
  display.print(firstName);
  display.setCursor(0, 32);
  display.print("2nd: ");
  display.print(secondName);
  display.setCursor(0, 50);
  display.print("Opening for 5s");
  display.display();
  Serial.print("ACCESS GRANTED to ");
  Serial.print(firstName);
  Serial.print(" and ");
  Serial.println(secondName);
  digitalWrite(greenLed, HIGH);
  tone(buzzer, 1600, 300);
  lock.write(90);
  delay(5000);
  lock.write(0);
  digitalWrite(greenLed, LOW);
  firstUID = "";
}
void deny(String reason)
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("ACCESS DENIED");
  display.setTextSize(1);
  display.setCursor(0, 26);
  display.print(reason);
  display.display();
  Serial.print("DENIED: ");
  Serial.println(reason);
  digitalWrite(redLed, HIGH);
  tone(buzzer, 250, 500);
  delay(2000);
  digitalWrite(redLed, LOW);
}
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  lock.attach(26);
  lock.write(0);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showIdle();
}
void loop()
{
  if (firstUID != "" && millis() - firstScanAt > pairWindow)
  {
    firstUID = "";
    Serial.println("Pairing window expired");
  }
  showIdle();
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int index = findAuth(uid);
  if (index < 0)
```

```
  {
    deny("Card not authorised");
  }
  else if (firstUID == "")
  {
    firstUID = uid;
    firstName = authName[index];
    firstScanAt = millis();
    tone(buzzer, 1200, 120);
    Serial.print("First card: ");
    Serial.println(firstName);
  }
  else if (uid == firstUID)
  {
    deny("Same card twice - a second person is required");
  }
  else
  {
    grantAccess(authName[index]);
  }
  showIdle();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: LOCKED with the instruction that two cards are required. Scan one authorised card: it accepts and starts a 20-second countdown. Scan the same card again: ACCESS DENIED, a second person is required. Scan a different authorised card: ACCESS GRANTED and the servo opens.

RFID Membership Verification

Goal: Check whether a membership is still valid by comparing its expiry against a running day counter, and warn when it is nearly due.

Validity is a comparison between two numbers: today, and the day the membership runs out. Because the Uno has no real-time clock, a day counter stands in, the button advances it so the whole year can be demonstrated in a minute. Three outcomes rather than two is the important design point: valid, expiring soon, and expired. That middle state is what lets a business remind a member before they lapse.

Online page with troubleshooting: geekobot.in/projects/titan-kit-078-rfid-membership-verification

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Expiration and validity checks
- Simulating dates without a clock module
- Three-state validity instead of two
- Warning windows before a deadline

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED (valid)	**GPIO27**	Through a 220 ohm resistor
LED	Yellow LED (expiring)	**GPIO26**	Through a 220 ohm resistor
LED	Red LED (expired)	**GPIO14**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Button	Advance day button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define MEMBERS 4
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int greenLed = 27;
int yellowLed = 26;
int redLed = 14;
int buzzer = 33;
int dayButton = 12;
String memberUID[MEMBERS] =
{
  "A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD", "12 34 56 78"
};
String memberName[MEMBERS] = {"ANJALI", "RAHUL", "PRIYA", "AMIT"};
String plan[MEMBERS] = {"Annual", "Monthly", "Annual", "Trial"};
int expiryDay[MEMBERS] = {40, 12, 300, 5};
int today = 1;
int warnWithin = 7;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findMember(String uid)
{
  for (int i = 0; i < MEMBERS; i++)
  {
    if (memberUID[i] == uid) return i;
  }
  return -1;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("MEMBERSHIP CHECK");
  display.setTextSize(2);
  display.setCursor(0, 16);
  display.print("DAY ");
  display.print(today);
  display.setTextSize(1);
  display.setCursor(0, 40);
  display.print("Scan a member card");
  display.setCursor(0, 52);
  display.print("D3 advances the day");
  display.display();
}
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(dayButton, INPUT_PULLUP);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showIdle();
```

```
}
void loop()
{
  if (digitalRead(dayButton) == LOW)
  {
    today = today + 1;
    Serial.print("Day is now ");
    Serial.println(today);
    showIdle();
    delay(200);
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int index = findMember(uid);
  digitalWrite(greenLed, LOW);
  digitalWrite(yellowLed, LOW);
  digitalWrite(redLed, LOW);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  if (index < 0)
  {
    display.print("NOT A MEMBER");
    display.setCursor(0, 28);
    display.print(uid);
    digitalWrite(redLed, HIGH);
    tone(buzzer, 250, 400);
  }
  else
  {
    int daysLeft = expiryDay[index] - today;
    display.print(memberName[index]);
    display.print(" ");
    display.print(plan[index]);
    display.setTextSize(2);
    display.setCursor(0, 14);
    if (daysLeft < 0)
    {
      display.print("EXPIRED");
      display.setTextSize(1);
      display.setCursor(0, 36);
      display.print("Ran out ");
      display.print(-daysLeft);
      display.print(" days ago");
      display.setCursor(0, 48);
      display.print("Please renew today");
      digitalWrite(redLed, HIGH);
      tone(buzzer, 250, 500);
    }
    else if (daysLeft <= warnWithin)
    {
      display.print("EXPIRING");
      display.setTextSize(1);
      display.setCursor(0, 36);
      display.print(daysLeft);
      display.print(" days remaining");
      display.setCursor(0, 48);
      display.print("Renewal due soon");
      digitalWrite(yellowLed, HIGH);
      tone(buzzer, 900, 250);
    }
    else
    {
      display.print("VALID");
      display.setTextSize(1);
      display.setCursor(0, 36);
      display.print(daysLeft);
      display.print(" days remaining");
      display.setCursor(0, 48);
      display.print("Welcome back");
      digitalWrite(greenLed, HIGH);
      tone(buzzer, 1500, 150);
    }
    Serial.print(memberName[index]);
```

```
    Serial.print(": ");
    Serial.print(daysLeft);
    Serial.println(" days left");
  }
  display.display();
  delay(2500);
  digitalWrite(greenLed, LOW);
  digitalWrite(yellowLed, LOW);
  digitalWrite(redLed, LOW);
  showIdle();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: The idle screen shows the simulated day number. Scan a card with plenty of time left: VALID with the green LED. Press the day button repeatedly, then rescan: EXPIRING with the yellow LED. Push past the expiry: EXPIRED with the red LED and how long ago it lapsed.

RFID Based Task Assignment Demo

Goal: Give each person their task for the day when they scan in, choosing the task from their card's identity.

The card identifies the person; the person's index selects the task. A rotation offset shifts everyone's assignment by one, so the same four people and four tasks produce four different duty rosters across the week, and the modulo operator handles the wrap. This is switch-case matching in spirit: one input value chooses among several outcomes, and the array does the job a long switch statement would.

Online page with troubleshooting: geekobot.in/projects/titan-kit-079-rfid-task-assignment

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Switch-case matching via UID
- Rotating assignments with modulo arithmetic
- Marking who has already collected
- One input selecting among several outcomes

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Rotate button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define PEOPLE 4
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int rotateButton = 12;
int buzzer = 33;
String personUID[PEOPLE] =
{
  "A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD", "12 34 56 78"
};
String personName[PEOPLE] = {"ANJALI", "RAHUL", "PRIYA", "AMIT"};
String task[PEOPLE] =
{
  "Tidy the lab bench",
  "Check kit inventory",
  "Charge all batteries",
  "Update the log book"
};
String detail[PEOPLE] =
{
  "Return tools to trays",
  "Note any missing parts",
  "Label each pack",
  "Record today's tests"
};
bool collected[PEOPLE] = {false, false, false, false};
int rotation = 0;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findPerson(String uid)
{
  for (int i = 0; i < PEOPLE; i++)
  {
    if (personUID[i] == uid) return i;
  }
  return -1;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("TASK ASSIGNMENT");
  display.setCursor(0, 14);
  display.print("Rotation ");
  display.print(rotation + 1);
  display.print(" of ");
  display.print(PEOPLE);
  int done = 0;
  for (int i = 0; i < PEOPLE; i++) if (collected[i]) done = done + 1;
  display.setCursor(0, 26);
  display.print("Collected: ");
  display.print(done);
  display.print(" of ");
  display.print(PEOPLE);
  display.setCursor(0, 42);
  display.print("Scan your card for");
  display.setCursor(0, 52);
```

```
  display.print("today's task");
  display.display();
}
void setup()
{
  pinMode(rotateButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showIdle();
}
void loop()
{
  if (digitalRead(rotateButton) == LOW)
  {
    rotation = (rotation + 1) % PEOPLE;
    for (int i = 0; i < PEOPLE; i++) collected[i] = false;
    tone(buzzer, 1000, 120);
    Serial.print("Rotation now ");
    Serial.println(rotation + 1);
    showIdle();
    delay(350);
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int person = findPerson(uid);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  if (person < 0)
  {
    display.print("CARD NOT ON TEAM");
    display.setCursor(0, 28);
    display.print(uid);
    tone(buzzer, 250, 400);
  }
  else
  {
    int assignment = (person + rotation) % PEOPLE;
    display.fillRect(0, 0, 128, 12, SSD1306_WHITE);
    display.setTextColor(SSD1306_BLACK);
    display.setCursor(3, 2);
    display.print(personName[person]);
    display.setCursor(96, 2);
    display.print(collected[person] ? "AGAIN" : "NEW");
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 18);
    display.print("Your task:");
    display.setCursor(0, 32);
    display.print(task[assignment]);
    display.setCursor(0, 48);
    display.print(detail[assignment]);
    collected[person] = true;
    tone(buzzer, 1400, 150);
    Serial.print(personName[person]);
    Serial.print(" -> ");
    Serial.println(task[assignment]);
  }
  display.display();
  delay(3000);
  showIdle();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: The idle screen shows the current rotation and how many people have collected their task. Scan a card: the person's name in a header with their task below. Scan again and the header reads AGAIN rather than NEW. Press the rotate button and everyone gets a different task.

RFID Event Entry System

Goal: Run a door for a ticketed event: admit guests up to a capacity limit, reject duplicate tickets, and track live occupancy.

Three separate checks run in order and any one can refuse entry: is the ticket real, has it already been used, and is there room. Order matters, checking capacity first would waste a place on a forged ticket. Occupancy is a live figure that goes down as well as up, which is what a fire officer actually cares about, unlike the total admitted figure that only ever rises.

Online page with troubleshooting: geekobot.in/projects/titan-kit-080-rfid-event-entry

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Capacity counting and limits
- Ordering validation checks by priority
- Live occupancy versus cumulative totals
- Single-use tickets with a flag array

Wiring

Part	Pin	ESP32	Note
RC522	3.3V	**3V3**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**GPIO32**	
RC522	SDA / SS	**GPIO5**	
RC522	MOSI	**GPIO23**	
RC522	MISO	**GPIO19**	
RC522	SCK	**GPIO18**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Red LED anode	**GPIO14**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Button	Exit button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Power the RC522 from 3V3 only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 5
#define RST_PIN 32
#define TICKETS 6
MFRC522 rfid(SS_PIN, RST_PIN);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int greenLed = 27;
int redLed = 14;
int buzzer = 33;
int exitButton = 12;
String ticketUID[TICKETS] =
{
  "A1 B2 C3 D4", "11 22 33 44", "AA BB CC DD",
  "12 34 56 78", "DE AD BE EF", "CA FE 00 01"
};
String guestName[TICKETS] =
{
  "ANJALI", "RAHUL", "PRIYA", "AMIT", "SNEHA", "VIKAS"
};
bool used[TICKETS];
int capacity = 4;
int inside = 0;
int admitted = 0;
int rejected = 0;
String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10) uid = uid + "0";
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1) uid = uid + " ";
  }
  uid.toUpperCase();
  return uid;
}
int findTicket(String uid)
{
  for (int i = 0; i < TICKETS; i++)
  {
    if (ticketUID[i] == uid) return i;
  }
  return -1;
}
void showIdle()
{
  display.clearDisplay();
  display.setTextSize(1);
  display.fillRect(0, 0, 128, 11, SSD1306_WHITE);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(2, 2);
  display.print("EVENT ENTRY");
  display.setCursor(84, 2);
  display.print(inside);
  display.print("/");
  display.print(capacity);
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(2);
  display.setCursor(0, 16);
  display.print(inside >= capacity ? "FULL" : "OPEN");
  display.setTextSize(1);
  display.setCursor(64, 20);
  display.print("in ");
  display.print(admitted);
  display.setCursor(64, 30);
  display.print("no ");
  display.print(rejected);
  display.drawRect(0, 40, 128, 10, SSD1306_WHITE);
  display.fillRect(1, 41, map(inside, 0, capacity, 0, 126), 8, SSD1306_WHITE);
  display.setCursor(0, 54);
```

```
  display.print("Scan ticket D3 exit");
  display.display();
}
void setup()
{
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(exitButton, INPUT_PULLUP);
  for (int i = 0; i < TICKETS; i++) used[i] = false;
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  showIdle();
}
void loop()
{
  if (digitalRead(exitButton) == LOW)
  {
    if (inside > 0)
    {
      inside = inside - 1;
      tone(buzzer, 800, 100);
      Serial.print("Guest left. Inside: ");
      Serial.println(inside);
    }
    showIdle();
    delay(300);
  }
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;
  String uid = readUID();
  int index = findTicket(uid);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  if (index < 0)
  {
    rejected = rejected + 1;
    display.print("INVALID TICKET");
    display.setCursor(0, 24);
    display.print(uid);
    display.setCursor(0, 44);
    display.print("Not on the guest list");
    digitalWrite(redLed, HIGH);
    tone(buzzer, 250, 500);
    Serial.println("Invalid ticket");
  }
  else if (used[index])
  {
    rejected = rejected + 1;
    display.print("TICKET ALREADY USED");
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print(guestName[index]);
    display.setTextSize(1);
    display.setCursor(0, 48);
    display.print("One entry per ticket");
    digitalWrite(redLed, HIGH);
    tone(buzzer, 300, 400);
    Serial.print("Duplicate: ");
    Serial.println(guestName[index]);
  }
  else if (inside >= capacity)
  {
    rejected = rejected + 1;
    display.print("VENUE FULL");
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print("WAIT");
    display.setTextSize(1);
    display.setCursor(0, 48);
    display.print("Capacity reached");
```

```
    digitalWrite(redLed, HIGH);
    tone(buzzer, 400, 400);
    Serial.println("Turned away - at capacity");
  }
  else
  {
    used[index] = true;
    inside = inside + 1;
    admitted = admitted + 1;
    display.print("WELCOME");
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print(guestName[index]);
    display.setTextSize(1);
    display.setCursor(0, 48);
    display.print("Guest ");
    display.print(admitted);
    display.print(" of ");
    display.print(capacity);
    digitalWrite(greenLed, HIGH);
    tone(buzzer, 1500, 150);
    Serial.print("Admitted: ");
    Serial.println(guestName[index]);
  }
  display.display();
  delay(2000);
  digitalWrite(greenLed, LOW);
  digitalWrite(redLed, LOW);
  showIdle();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: The header shows live occupancy such as 2/4 and the door status OPEN. Scan a valid ticket: WELCOME with the guest name and the count rises. Scan the same ticket again: TICKET ALREADY USED. Once four guests are inside the door reads FULL until someone exits.

PROJECT 081 · LEVEL 13 · BLUETOOTH

Bluetooth Sensor Monitoring Station

Goal: Stream live sensor readings to a phone as they happen, with the phone able to start, stop and set the rate.

Telemetry means sending measurements as they happen, and the awkward part is that the board must keep measuring while it listens for commands. Nothing here blocks: the sensor is read on its own schedule, transmissions happen on theirs, and the incoming buffer is checked every pass. Sending a packet counter with each line lets the receiver notice if anything went missing, a small addition that turns a stream of numbers into a stream you can trust.

Online page with troubleshooting: geekobot.in/projects/titan-kit-081-bluetooth-sensor-monitoring-station

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 10k resistor
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Real-time telemetry
- Non-blocking send and receive together
- Packet counters for detecting loss
- Letting the receiver control the sample rate

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
DHT11	DHT11 DATA	**GPIO14**	
DHT11	DHT11 VCC / GND	**3.3v / GND**	
LDR divider	LDR divider output	**GPIO34**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 14
#define DHTTYPE DHT11
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
DHT dht(DHTPIN, DHTTYPE);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
float temp = 0, hum = 0;
int light = 0;
bool streaming = false;
unsigned long interval = 2000;
unsigned long lastSend = 0;
unsigned long lastRead = 0;
long packets = 0;
void readSensors()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (!isnan(t)) temp = t;
  if (!isnan(h)) hum = h;
  light = map(analogRead(34), 0, 1023, 0, 100);
}
void sendPacket()
{
  packets = packets + 1;
  bt.print("#");
  bt.print(packets);
  bt.print(",T=");
  bt.print(temp, 1);
  bt.print(",H=");
  bt.print(hum, 0);
  bt.print(",L=");
  bt.print(light);
  bt.print(",t=");
  bt.println(millis() / 1000);
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  readSensors();
  bt.println("READY. S=start X=stop F=fast N=normal R=read once");
}
void loop()
{
  if (millis() - lastRead > 2000)
  {
    lastRead = millis();
    readSensors();
  }
  if (bt.available())
  {
    char command = bt.read();
    Serial.println(command);
    if (command == 'S') { streaming = true; bt.println("Streaming ON"); }
    if (command == 'X') { streaming = false; bt.println("Streaming OFF"); }
    if (command == 'F') { interval = 500; bt.println("Rate 0.5s"); }
    if (command == 'N') { interval = 2000; bt.println("Rate 2s"); }
    if (command == 'R') { sendPacket(); }
  }
  if (streaming && millis() - lastSend > interval)
  {
    lastSend = millis();
    sendPacket();
  }
  display.clearDisplay();
```

```
  display.setCursor(0, 0);
  display.print("TELEMETRY STATION");
  display.setCursor(0, 14);
  display.print("T ");
  display.print(temp, 1);
  display.print("C H ");
  display.print(hum, 0);
  display.print("% L ");
  display.print(light);
  display.setCursor(0, 28);
  display.print("Link: ");
  display.print(streaming ? "STREAMING" : "IDLE");
  display.setCursor(0, 40);
  display.print("Rate: ");
  display.print(interval / 1000.0, 1);
  display.print("s");
  display.setCursor(0, 52);
  display.print("Packets sent: ");
  display.print(packets);
  display.display();
}
```

What you should see: Pair with the HC-05 and the phone receives a command list. Send S and lines like #1,T=28.4,H=61,L=74,t=12 arrive every 2 seconds. Send F to speed up to twice a second, N to return to normal. Send X to stop; the packet counter on the OLED shows the running total.

Bluetooth Data Logger

Goal: Record readings into memory while disconnected, then dump the whole log to a phone in one burst when it reconnects.

Live streaming needs a phone present the whole time, which is rarely practical. Store-and-forward solves it: the board logs quietly into RAM and only transmits when asked. Because a bulk dump is many lines sent back to back, a short pause after each one gives the receiver time to keep up, without it, phones with small serial buffers drop lines, which is a genuinely common and confusing bug.

Online page with troubleshooting: geekobot.in/projects/titan-kit-082-bluetooth-data-logger

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Bulk data transmission
- Store-and-forward instead of live streaming
- Pacing a bulk dump so the receiver keeps up
- CSV output that pastes straight into a spreadsheet

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
DHT11	DHT11 DATA	**GPIO14**	
DHT11	DHT11 VCC / GND	**3V3 / GND**	
Button	Clear button	**GPIO27**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 14
#define DHTTYPE DHT11
#define CAPACITY 60
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
DHT dht(DHTPIN, DHTTYPE);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int clearButton = 27;
byte logTemp[CAPACITY];
byte logHum[CAPACITY];
unsigned int logTime[CAPACITY];
int stored = 0;
unsigned long interval = 5000;
unsigned long lastLog = 0;
void addRecord(byte t, byte h)
{
  if (stored >= CAPACITY)
  {
    // Drop the oldest to make room
    for (int i = 0; i < CAPACITY - 1; i++)
    {
      logTemp[i] = logTemp[i + 1];
      logHum[i] = logHum[i + 1];
      logTime[i] = logTime[i + 1];
    }
    stored = CAPACITY - 1;
  }
  logTemp[stored] = t;
  logHum[stored] = h;
  logTime[stored] = millis() / 1000;
  stored = stored + 1;
}
void dumpLog()
{
  bt.println("=== LOG START ===");
  bt.print("Records: ");
  bt.println(stored);
  bt.println("index,seconds,tempC,humidity");
  for (int i = 0; i < stored; i++)
  {
    bt.print(i + 1);
    bt.print(",");
    bt.print(logTime[i]);
    bt.print(",");
    bt.print(logTemp[i]);
    bt.print(",");
    bt.println(logHum[i]);
    delay(25); // Let the receiver keep up
  }
  bt.println("=== LOG END ===");
  Serial.println("Log transmitted");
}
void setup()
{
  pinMode(clearButton, INPUT_PULLUP);
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  bt.println("D=dump C=clear I=info");
}
void loop()
{
  if (digitalRead(clearButton) == LOW)
  {
    stored = 0;
    Serial.println("Log cleared");
    delay(300);
```

```
  }
  if (bt.available())
  {
    char command = bt.read();
    if (command == 'D') dumpLog();
    if (command == 'C') { stored = 0; bt.println("Log cleared"); }
    if (command == 'I')
    {
      bt.print("Stored ");
      bt.print(stored);
      bt.print(" of ");
      bt.print(CAPACITY);
      bt.print(", interval ");
      bt.print(interval / 1000);
      bt.println("s");
    }
  }
  if (millis() - lastLog > interval)
  {
    lastLog = millis();
    float t = dht.readTemperature();
    float h = dht.readHumidity();
    if (!isnan(t) && !isnan(h))
    {
      addRecord((byte)t, (byte)h);
      Serial.print("Logged record ");
      Serial.println(stored);
    }
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("DATA LOGGER");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print(stored);
  display.setTextSize(1);
  display.print(" / ");
  display.print(CAPACITY);
  display.setCursor(0, 36);
  display.print("Next in ");
  display.print((interval - (millis() - lastLog)) / 1000);
  display.print("s");
  if (stored > 0)
  {
    display.setCursor(0, 46);
    display.print("Last: ");
    display.print(logTemp[stored - 1]);
    display.print("C ");
    display.print(logHum[stored - 1]);
    display.print("%");
  }
  display.setCursor(0, 56);
  display.print("Send D to download");
  display.display();
}
```

What you should see: The board logs one record every 5 seconds with no phone connected. Connect later and send D: the whole log arrives as comma-separated lines. The dump is bracketed by LOG START and LOG END markers. Send C or press the button to clear and start again.

Bluetooth Wireless Terminal

Goal: Hold a two-way text conversation between the phone and the board, with typed commands the board understands.

Reading whole lines rather than single characters is what makes a command interpreter possible. Characters are collected until a newline arrives, then the completed String is compared against known commands. Splitting a command from its argument at the first space, as in led on or say hello, is the same parsing pattern every shell uses, and it is worth writing once by hand.

Online page with troubleshooting: geekobot.in/projects/titan-kit-083-bluetooth-wireless-terminal

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Bidirectional Serial parsing
- Splitting a command from its argument
- Building a command interpreter
- Replying so the sender knows what happened

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int led = 25;
int buzzer = 33;
String input = "";
String lastCommand = "";
String lastReply = "";
int commandCount = 0;
void reply(String text)
{
  bt.println(text);
  lastReply = text;
  Serial.println(text);
}
void handle(String line)
{
  line.trim();
  if (line.length() == 0) return;
  lastCommand = line;
  commandCount = commandCount + 1;
  String command = line;
  String argument = "";
  int space = line.indexOf(' ');
  if (space > 0)
  {
    command = line.substring(0, space);
    argument = line.substring(space + 1);
  }
  command.toLowerCase();
  if (command == "help")
  {
    reply("Commands: help, led on|off, beep, say <text>, uptime, echo <text>");
  }
  else if (command == "led")
  {
    argument.toLowerCase();
    if (argument == "on") { digitalWrite(led, HIGH); reply("LED is on"); }
    else if (argument == "off") { digitalWrite(led, LOW); reply("LED is off"); }
    else { reply("Use: led on or led off"); }
  }
  else if (command == "beep")
  {
    tone(buzzer, 1400, 200);
    reply("Beeped");
  }
  else if (command == "say")
  {
    reply("Displayed: " + argument);
  }
  else if (command == "uptime")
  {
    reply("Up for " + String(millis() / 1000) + " seconds");
  }
  else if (command == "echo")
  {
    reply(argument);
  }
  else
  {
    reply("Unknown command. Type help");
  }
}
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
```

```
  display.setTextSize(1);
  reply("Terminal ready. Type help");
}
void loop()
{
  while (bt.available())
  {
    char c = bt.read();
    if (c == '\n')
    {
      handle(input);
      input = "";
    }
    else if (c != '\r')
    {
      input = input + c;
    }
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("WIRELESS TERMINAL");
  display.setCursor(0, 14);
  display.print("> ");
  display.print(lastCommand.substring(0, 18));
  display.setCursor(0, 28);
  display.print(lastReply.substring(0, 21));
  if (lastReply.length() > 21)
  {
    display.setCursor(0, 38);
    display.print(lastReply.substring(21, 42));
  }
  display.setCursor(0, 54);
  display.print("Commands: ");
  display.print(commandCount);
  display.display();
}
```

What you should see: Send help and the phone receives the full command list. Send led on and the LED lights with the reply LED is on. Send echo hello world and the same text comes back. The OLED shows the last command and the board's reply.

Smartphone Controlled Servo Positioning

Goal: Drive a servo to any angle from a phone, using either a slider value or named preset positions.

A slider in a phone app sends a number as text, so the board parses it and passes it to the servo, that is app-to-hardware PWM integration in one line. The addition worth studying is the sweep: rather than jumping to the new angle, the servo steps towards it a degree at a time. Non-blocking stepping keeps the Bluetooth link responsive during the move, so a new command can interrupt one already in progress.

Online page with troubleshooting: geekobot.in/projects/titan-kit-084-smartphone-controlled-servo-positioning

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- App-to-hardware PWM integration
- Parsing a numeric value from text
- Non-blocking stepped movement
- Presets alongside free numeric entry

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Servo	Signal (orange)	**GPIO26**	
Servo	Red	**5V**	
Servo	Brown / black	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
Servo arm;
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int currentAngle = 90;
int targetAngle = 90;
int stepDelay = 15;
unsigned long lastStep = 0;
String input = "";
String lastAction = "start";
void setTarget(int angle, String label)
{
  targetAngle = constrain(angle, 0, 180);
  lastAction = label;
  bt.print("Moving to ");
  bt.print(targetAngle);
  bt.println(" degrees");
  Serial.print("Target ");
  Serial.println(targetAngle);
}
void handle(String line)
{
  line.trim();
  if (line.length() == 0) return;
  String lower = line;
  lower.toLowerCase();
  if (lower == "min") setTarget(0, "preset min");
  else if (lower == "mid") setTarget(90, "preset mid");
  else if (lower == "max") setTarget(180, "preset max");
  else if (lower == "slow") { stepDelay = 30; bt.println("Slow sweep"); }
  else if (lower == "fast") { stepDelay = 5; bt.println("Fast sweep"); }
  else if (lower == "snap") { stepDelay = 0; bt.println("Instant move"); }
  else
  {
    int value = line.toInt();
    if (value > 0 || line == "0")
    {
      setTarget(value, "angle " + line);
    }
    else
    {
      bt.println("Send 0-180, or min mid max slow fast snap");
    }
  }
}
void setup()
{
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  arm.attach(26);
  arm.write(currentAngle);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  bt.println("Send an angle 0-180 or min mid max");
}
void loop()
{
  while (bt.available())
  {
    char c = bt.read();
    if (c == '\n') { handle(input); input = ""; }
    else if (c != '\r') { input = input + c; }
  }
  // Step towards the target without blocking
  if (currentAngle != targetAngle && millis() - lastStep >= stepDelay)
  {
    lastStep = millis();
    if (stepDelay == 0)
    {
```

```
      currentAngle = targetAngle;
    }
    else if (currentAngle < targetAngle)
    {
      currentAngle = currentAngle + 1;
    }
    else
    {
      currentAngle = currentAngle - 1;
    }
    arm.write(currentAngle);
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("SERVO CONTROL");
  display.setTextSize(3);
  display.setCursor(0, 12);
  display.print(currentAngle);
  display.setTextSize(1);
  display.print((char)247);
  display.setCursor(70, 20);
  display.print("target ");
  display.print(targetAngle);
  // Draw the arm position
  int cx = 100;
  int cy = 48;
  float radians = (180 - currentAngle) * PI / 180.0;
  display.drawLine(cx - 24, cy, cx + 24, cy, SSD1306_WHITE);
  display.drawLine(cx, cy, cx + cos(radians) * 22, cy - sin(radians) * 22, SSD1306_WHITE);
  display.setCursor(0, 40);
  display.print(lastAction);
  display.setCursor(0, 52);
  display.print(currentAngle == targetAngle ? "At position" : "Moving...");
  display.display();
}
```

What you should see: Send 45 and the servo sweeps smoothly to 45 degrees. Send max and it moves to 180. Send fast or slow to change the sweep speed, snap to jump instantly. The OLED draws the arm at its live angle as it moves.

Bluetooth Environmental Dashboard

Goal: Send readings in a structured format a custom app can parse, and let the app request exactly the fields it wants.

A remote user interface needs data the app can read reliably, so the board offers three formats of the same readings: human text, CSV for spreadsheets, and JSON-style key-value pairs for code. Choosing a format is a real engineering decision, human text is easy to read and awkward to parse, JSON is the reverse. Offering all three from one sketch makes the trade-off visible.

Online page with troubleshooting: geekobot.in/projects/titan-kit-085-bluetooth-environmental-dashboard

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 10k resistor
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Remote UI updating
- Choosing a data format for its consumer
- Text, CSV and JSON compared
- Escaping quote characters inside a string

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
DHT11	DHT11 DATA	**GPIO14**	
DHT11	DHT11 VCC / GND	**3V3 / GND**	
LDR divider	LDR divider output	**GPIO34**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 14
#define DHTTYPE DHT11
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
DHT dht(DHTPIN, DHTTYPE);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
float temp = 0, hum = 0;
int light = 0;
int format = 0; // 0 text, 1 CSV, 2 JSON
const char* formatName[3] = {"TEXT", "CSV", "JSON"};
bool autoSend = false;
unsigned long interval = 3000;
unsigned long lastSend = 0;
unsigned long lastRead = 0;
void readSensors()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (!isnan(t)) temp = t;
  if (!isnan(h)) hum = h;
  light = map(analogRead(34), 0, 1023, 0, 100);
}
void sendReadings()
{
  if (format == 0)
  {
    bt.print("Temperature: ");
    bt.print(temp, 1);
    bt.print(" C, Humidity: ");
    bt.print(hum, 0);
    bt.print(" %, Light: ");
    bt.print(light);
    bt.println(" %");
  }
  else if (format == 1)
  {
    bt.print(millis() / 1000);
    bt.print(",");
    bt.print(temp, 1);
    bt.print(",");
    bt.print(hum, 0);
    bt.print(",");
    bt.println(light);
  }
  else
  {
    bt.print("{\"t\":");
    bt.print(temp, 1);
    bt.print(",\"h\":");
    bt.print(hum, 0);
    bt.print(",\"l\":");
    bt.print(light);
    bt.print(",\"up\":");
    bt.print(millis() / 1000);
    bt.println("}");
  }
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  readSensors();
  bt.println("R=read A=auto X=stop 1=text 2=csv 3=json");
}
void loop()
```

```
{
  if (millis() - lastRead > 2000)
  {
    lastRead = millis();
    readSensors();
  }
  if (bt.available())
  {
    char command = bt.read();
    if (command == 'R') sendReadings();
    if (command == 'A') { autoSend = true; bt.println("Auto ON"); }
    if (command == 'X') { autoSend = false; bt.println("Auto OFF"); }
    if (command == '1') { format = 0; bt.println("Format: text"); }
    if (command == '2') { format = 1; bt.println("Format: csv (time,temp,hum,light)"); }
    if (command == '3') { format = 2; bt.println("Format: json"); }
  }
  if (autoSend && millis() - lastSend > interval)
  {
    lastSend = millis();
    sendReadings();
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("REMOTE DASHBOARD");
  display.drawRect(0, 12, 62, 22, SSD1306_WHITE);
  display.setCursor(4, 15);
  display.print("TEMP");
  display.setCursor(4, 24);
  display.print(temp, 1);
  display.print("C");
  display.drawRect(66, 12, 62, 22, SSD1306_WHITE);
  display.setCursor(70, 15);
  display.print("HUMID");
  display.setCursor(70, 24);
  display.print(hum, 0);
  display.print("%");
  display.drawRect(0, 36, 128, 14, SSD1306_WHITE);
  display.setCursor(4, 39);
  display.print("LIGHT ");
  display.print(light);
  display.print("%");
  display.fillRect(56, 38, map(light, 0, 100, 0, 68), 10, SSD1306_WHITE);
  display.setCursor(0, 54);
  display.print(formatName[format]);
  display.print(" ");
  display.print(autoSend ? "AUTO" : "MANUAL");
  display.display();
}
```

What you should see: Send R in text mode: Temperature: 28.4 C, Humidity: 61 %, Light: 74 %. Send 2 then R: 145,28.4,61,74 ready for a spreadsheet. Send 3 then R: a JSON object a phone app can parse directly. Send A to stream automatically in whichever format is selected.

Bluetooth Notification Display

Goal: Show notifications sent from a phone on the OLED, keeping the most recent few and marking them as read.

Handling strings on a board with 2 KB of RAM needs discipline. Incoming text is trimmed to a fixed maximum, only three notifications are kept, and the oldest is discarded when a fourth arrives. The unread flag drives the LED, which means the notification survives being missed, the light stays on until someone acknowledges it, which is the whole point of a notifier.

Online page with troubleshooting: geekobot.in/projects/titan-kit-086-bluetooth-notification

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- String handling over Bluetooth
- Managing limited RAM with fixed-length strings
- Unread flags that persist until acknowledged
- Wrapping long text across two lines

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Button	Mark read button	**GPIO27**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SLOTS 3
#define MAX_LENGTH 40
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int led = 25;
int buzzer = 33;
int readButton = 27;
String messages[SLOTS];
unsigned long arrived[SLOTS];
bool unread[SLOTS];
int count = 0;
String input = "";
int totalReceived = 0;
void addMessage(String text)
{
  if (text.length() > MAX_LENGTH)
  {
    text = text.substring(0, MAX_LENGTH);
  }
  if (count >= SLOTS)
  {
    for (int i = 0; i < SLOTS - 1; i++)
    {
      messages[i] = messages[i + 1];
      arrived[i] = arrived[i + 1];
      unread[i] = unread[i + 1];
    }
    count = SLOTS - 1;
  }
  messages[count] = text;
  arrived[count] = millis() / 1000;
  unread[count] = true;
  count = count + 1;
  totalReceived = totalReceived + 1;
  tone(buzzer, 1600, 120);
  delay(140);
  tone(buzzer, 2000, 120);
  bt.println("Notification displayed");
  Serial.println(text);
}
bool anyUnread()
{
  for (int i = 0; i < count; i++)
  {
    if (unread[i]) return true;
  }
  return false;
}
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(readButton, INPUT_PULLUP);
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  bt.println("Send any text to display it as a notification");
}
void loop()
{
  while (bt.available())
  {
    char c = bt.read();
    if (c == '\n')
    {
      input.trim();
      if (input.length() > 0) addMessage(input);
      input = "";
```

```
    }
    else if (c != '\r')
    {
      input = input + c;
    }
  }
  if (digitalRead(readButton) == LOW)
  {
    for (int i = 0; i < count; i++) unread[i] = false;
    Serial.println("All marked read");
    delay(300);
  }
  digitalWrite(led, anyUnread() && (millis() / 500) % 2);
  display.clearDisplay();
  display.fillRect(0, 0, 128, 11, SSD1306_WHITE);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(2, 2);
  display.print("NOTIFICATIONS ");
  display.print(totalReceived);
  display.setTextColor(SSD1306_WHITE);
  if (count == 0)
  {
    display.setCursor(0, 28);
    display.print("No messages yet");
  }
  else
  {
    for (int i = count - 1; i >= 0; i--)
    {
      int row = count - 1 - i;
      int y = 14 + row * 17;
      display.setCursor(0, y);
      display.print(unread[i] ? "* " : " ");
      display.print(messages[i].substring(0, 19));
      if (messages[i].length() > 19)
      {
        display.setCursor(12, y + 8);
        display.print(messages[i].substring(19, 38));
      }
      else
      {
        display.setCursor(12, y + 8);
        display.print(arrived[i]);
        display.print("s ago");
      }
    }
  }
  display.display();
}
```

What you should see: Send Meeting at 3pm from the phone and it appears at the top of the list. A two-tone chime sounds and the LED blinks while anything is unread. Send more messages and the oldest scrolls off after three. Press the button to mark everything read and stop the LED.

Bluetooth Digital Name Board

Goal: Update a scrolling marquee from a phone, changing both the message and how fast it moves.

Scrolling means drawing the same text at a decreasing x position each frame. The subtlety is knowing when to wrap: the text is fully off the left edge once x is less than minus its pixel width, and at size 2 each character is twelve pixels wide. Because the buffer holding the message can be replaced at any moment, the scroll position is reset whenever a new message arrives, otherwise a short message would start mid-air.

Online page with troubleshooting: geekobot.in/projects/titan-kit-087-bluetooth-digital-name-board

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Dynamic buffer updating
- Calculating text width in pixels
- Knowing when a scroll should wrap
- Changing display parameters at run time

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
Adafruit_SSD1306 display(128, 64, &Wire, -1);
String message = "WELCOME TO THE LAB";
String input = "";
int scrollX = 128;
int textSize = 2;
int speed = 2;
bool scrolling = true;
unsigned long lastFrame = 0;
void handle(String line)
{
  line.trim();
  if (line.length() == 0) return;
  String lower = line;
  lower.toLowerCase();
  if (lower == "stop") { scrolling = false; bt.println("Scrolling stopped"); }
  else if (lower == "start") { scrolling = true; bt.println("Scrolling started"); }
  else if (lower == "fast") { speed = 4; bt.println("Speed: fast"); }
  else if (lower == "slow") { speed = 1; bt.println("Speed: slow"); }
  else if (lower == "big") { textSize = 3; scrollX = 128; bt.println("Size: big"); }
  else if (lower == "normal") { textSize = 2; scrollX = 128; bt.println("Size: normal"); }
  else
  {
    message = line;
    scrollX = 128;
    bt.print("Now showing: ");
    bt.println(message);
    Serial.println(message);
  }
}
void setup()
{
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  bt.println("Send text to display, or stop start fast slow big normal");
}
void loop()
{
  while (bt.available())
  {
    char c = bt.read();
    if (c == '\n') { handle(input); input = ""; }
    else if (c != '\r') { input = input + c; }
  }
  if (millis() - lastFrame > 30)
  {
    lastFrame = millis();
    if (scrolling)
    {
      scrollX = scrollX - speed;
      // Each character is 6 pixels wide per size step
      int pixelWidth = message.length() * 6 * textSize;
      if (scrollX < -pixelWidth)
      {
        scrollX = 128;
      }
    }
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.print("NAME BOARD");
    display.setCursor(84, 0);
    display.print(scrolling ? "SCROLL" : "STATIC");
    display.setTextSize(textSize);
    display.setCursor(scrolling ? scrollX : 0, 26);
    display.print(message);
    display.drawLine(0, 10, 127, 10, SSD1306_WHITE);
    display.drawLine(0, 56, 127, 56, SSD1306_WHITE);
```

```
    display.setTextSize(1);
    display.setCursor(0, 58);
    display.print(message.length());
    display.print(" chars speed ");
    display.print(speed);
    display.display();
  }
}
```

What you should see: The default message scrolls right to left across the screen. Send any text from the phone and the board switches to it immediately. Send fast or slow to change the scroll speed. Send stop to freeze the text in place.

Bluetooth Wireless Counter Reset

Goal: Run a counter that can be reset remotely, with the reset protected by a confirmation step.

Anything destructive deserves a confirmation, and a remote command is the easiest thing in the world to send by accident. Sending RESET only arms the operation; a second CONFIRM within ten seconds carries it out, and the window closes on its own. The previous total is kept afterwards so a mistaken reset can still be reported, which costs one variable and has saved a great many people.

Online page with troubleshooting: geekobot.in/projects/titan-kit-088-bluetooth-wireless-counter-reset

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x Generic Infrared Obstacle Avoidance Sensor Module (Active Low) – LM393 Comparator, 2-30cm Range
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Remote interrupt handling
- Two-step confirmation for destructive commands
- Arming windows that expire safely
- Keeping the previous value after a reset

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
IR sensor	IR sensor OUT	**GPIO14**	
IR sensor	IR sensor VCC / GND	**3V3 / GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int sensorPin = 14;
int buzzer = 33;
long count = 0;
long previousTotal = 0;
int resets = 0;
bool lastState = HIGH;
unsigned long lastCount = 0;
bool resetArmed = false;
unsigned long armedAt = 0;
unsigned long armWindow = 10000;
String input = "";
void handle(String line)
{
  line.trim();
  line.toUpperCase();
  if (line.length() == 0) return;
  if (line == "RESET")
  {
    resetArmed = true;
    armedAt = millis();
    bt.print("Reset requested. Current count is ");
    bt.println(count);
    bt.println("Send CONFIRM within 10 seconds to proceed");
  }
  else if (line == "CONFIRM")
  {
    if (resetArmed && millis() - armedAt < armWindow)
    {
      previousTotal = count;
      count = 0;
      resets = resets + 1;
      resetArmed = false;
      tone(buzzer, 600, 400);
      bt.print("Counter reset. Previous total was ");
      bt.println(previousTotal);
      Serial.println("Counter reset");
    }
    else
    {
      bt.println("Nothing to confirm. Send RESET first");
    }
  }
  else if (line == "CANCEL")
  {
    resetArmed = false;
    bt.println("Reset cancelled");
  }
  else if (line == "COUNT")
  {
    bt.print("Count is ");
    bt.println(count);
  }
  else
  {
    bt.println("Commands: COUNT, RESET, CONFIRM, CANCEL");
  }
}
void setup()
{
  pinMode(sensorPin, INPUT);
  pinMode(buzzer, OUTPUT);
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  bt.println("Counter running. Commands: COUNT RESET CONFIRM CANCEL");
}
```

```
void loop()
{
  while (bt.available())
  {
    char c = bt.read();
    if (c == '\n') { handle(input); input = ""; }
    else if (c != '\r') { input = input + c; }
  }
  bool state = digitalRead(sensorPin);
  if (state == LOW && lastState == HIGH && millis() - lastCount > 200)
  {
    count = count + 1;
    lastCount = millis();
    tone(buzzer, 1500, 40);
    Serial.print("Count ");
    Serial.println(count);
  }
  lastState = state;
  if (resetArmed && millis() - armedAt > armWindow)
  {
    resetArmed = false;
    bt.println("Reset window expired");
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("REMOTE COUNTER");
  display.setTextSize(3);
  display.setCursor(0, 14);
  display.print(count);
  display.setTextSize(1);
  display.setCursor(0, 42);
  if (resetArmed)
  {
    display.print("RESET ARMED - ");
    display.print((armWindow - (millis() - armedAt)) / 1000);
    display.print("s");
    display.setCursor(0, 52);
    display.print("Send CONFIRM or CANCEL");
  }
  else
  {
    display.print("Previous total: ");
    display.print(previousTotal);
    display.setCursor(0, 52);
    display.print("Resets: ");
    display.print(resets);
  }
  display.display();
}
```

What you should see: Break the IR beam and the count rises by one with a short beep. Send RESET from the phone: the screen shows RESET ARMED with a countdown. Send CONFIRM and the counter zeroes, reporting the previous total. Let the window expire or send CANCEL and nothing is lost.

Bluetooth Home Appliance Simulator

Goal: Control a set of simulated appliances from a phone, each with its own state and a running total of what is switched on.

Remote GPIO control is the simple part; the interesting design work is in the command grammar. Named commands like light on read far better than single letters, and a scene command that sets several appliances at once is what turns a remote control into home automation. Each appliance carries a notional wattage, so the running total shows the consequence of a scene rather than just its state.

Online page with troubleshooting: geekobot.in/projects/titan-kit-089-bluetooth-home-appliance-simulator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 4 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 0.96" OLED Display 128x64
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Remote GPIO control
- Designing a readable command grammar
- Scenes that set several devices at once
- Deriving a total from device states

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Light	**GPIO14**	Through a 220 ohm resistor
LED	Fan	**GPIO27**	Through a 220 ohm resistor
LED	Heater	**GPIO26**	Through a 220 ohm resistor
LED	Kettle	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define DEVICES 4
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int pins[DEVICES] = {14, 27, 26, 25};
String names[DEVICES] = {"light", "fan", "heater", "kettle"};
int watts[DEVICES] = {12, 60, 1200, 2000};
bool state[DEVICES] = {false, false, false, false};
String input = "";
String lastAction = "ready";
int totalWatts()
{
  int total = 0;
  for (int i = 0; i < DEVICES; i++)
  {
    if (state[i]) total = total + watts[i];
  }
  return total;
}
void applyState()
{
  for (int i = 0; i < DEVICES; i++)
  {
    digitalWrite(pins[i], state[i]);
  }
}
void report()
{
  bt.println("--- Status ---");
  for (int i = 0; i < DEVICES; i++)
  {
    bt.print(names[i]);
    bt.print(": ");
    bt.print(state[i] ? "ON " : "OFF ");
    bt.print(watts[i]);
    bt.println("W");
  }
  bt.print("Total draw: ");
  bt.print(totalWatts());
  bt.println("W");
}
void handle(String line)
{
  line.trim();
  line.toLowerCase();
  if (line.length() == 0) return;
  String device = line;
  String action = "";
  int space = line.indexOf(' ');
  if (space > 0)
  {
    device = line.substring(0, space);
    action = line.substring(space + 1);
  }
  if (device == "status") { report(); return; }
  if (device == "all")
  {
    bool on = (action == "on");
    for (int i = 0; i < DEVICES; i++) state[i] = on;
    lastAction = on ? "all on" : "all off";
    applyState();
    bt.println(lastAction);
    return;
  }
  if (device == "scene")
  {
    if (action == "evening")
    {
      state[0] = true; state[1] = true; state[2] = false; state[3] = false;
      lastAction = "scene evening";
    }
```

```
    else if (action == "away")
    {
      for (int i = 0; i < DEVICES; i++) state[i] = false;
      lastAction = "scene away";
    }
    else if (action == "morning")
    {
      state[0] = true; state[1] = false; state[2] = false; state[3] = true;
      lastAction = "scene morning";
    }
    else
    {
      bt.println("Scenes: evening, morning, away");
      return;
    }
    applyState();
    bt.print("Applied ");
    bt.println(lastAction);
    return;
  }
  for (int i = 0; i < DEVICES; i++)
  {
    if (device == names[i])
    {
      if (action == "on") state[i] = true;
      else if (action == "off") state[i] = false;
      else state[i] = !state[i];
      applyState();
      lastAction = names[i] + " " + (state[i] ? "on" : "off");
      bt.print(names[i]);
      bt.println(state[i] ? " is on" : " is off");
      return;
    }
  }
  bt.println("Try: light on, fan off, all off, scene evening, status");
}
void setup()
{
  for (int i = 0; i < DEVICES; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  applyState();
  bt.println("Home simulator ready. Send status for a list");
}
void loop()
{
  while (bt.available())
  {
    char c = bt.read();
    if (c == '\n') { handle(input); input = ""; }
    else if (c != '\r') { input = input + c; }
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("HOME SIMULATOR");
  for (int i = 0; i < DEVICES; i++)
  {
    int y = 13 + i * 10;
    display.setCursor(0, y);
    display.print(names[i]);
    display.setCursor(60, y);
    display.print(state[i] ? "ON " : "OFF");
    display.setCursor(90, y);
    if (state[i])
    {
      display.print(watts[i]);
      display.print("W");
    }
  }
```

```
  display.drawLine(0, 52, 127, 52, SSD1306_WHITE);
  display.setCursor(0, 55);
  display.print("Total ");
  display.print(totalWatts());
  display.print("W ");
  display.print(lastAction);
  display.display();
}
```

What you should see: A list of four appliances with their state and power draw. Send light on and the first LED lights, adding 12W to the total. Send scene evening to set several at once. Send status and the phone receives the full list with the total draw.

Bluetooth Remote Configuration System

Goal: Change a running system's settings wirelessly, thresholds, intervals and modes, without touching the hardware or re-uploading.

Hard-coded settings mean a laptop and a cable every time a threshold needs adjusting. Sending set commands instead makes the device configurable in place, and saving those values to EEPROM makes them survive a power cut. The parsing pattern is worth noting: split at the first space to get the setting name, split again for the value, the same technique that scales all the way up to real configuration files.

Online page with troubleshooting: geekobot.in/projects/titan-kit-090-bluetooth-remote-configuration

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Wireless variable tuning
- Parsing set name value commands
- Persisting configuration to EEPROM
- Using a magic byte to detect unwritten EEPROM

Wiring

Part	Pin	ESP32	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
DHT11	DHT11 DATA	**GPIO14**	
DHT11	DHT11 VCC / GND	**3V3 / GND**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
Passive buzzer	Buzzer (+)	**GPIO33**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <HardwareSerial.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#include <EEPROM.h>
#define DHTPIN 14
#define DHTTYPE DHT11
#define MAGIC_ADDRESS 100
#define MAGIC_VALUE 42
HardwareSerial bt(2); // UART2: RX on GPIO16, TX on GPIO17
DHT dht(DHTPIN, DHTTYPE);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int led = 25;
int buzzer = 33;
// Configurable settings
int highLimit = 30;
int lowLimit = 18;
int checkSeconds = 3;
bool alarmEnabled = true;
float temp = 0;
bool alarmActive = false;
unsigned long lastCheck = 0;
String input = "";
void saveSettings()
{
  EEPROM.update(MAGIC_ADDRESS, MAGIC_VALUE);
  EEPROM.update(MAGIC_ADDRESS + 1, highLimit);
  EEPROM.update(MAGIC_ADDRESS + 2, lowLimit);
  EEPROM.update(MAGIC_ADDRESS + 3, checkSeconds);
  EEPROM.update(MAGIC_ADDRESS + 4, alarmEnabled ? 1 : 0);
  bt.println("Settings saved to EEPROM");
}
void loadSettings()
{
  if (EEPROM.read(MAGIC_ADDRESS) != MAGIC_VALUE)
  {
    return; // Never saved - keep the defaults
  }
  highLimit = EEPROM.read(MAGIC_ADDRESS + 1);
  lowLimit = EEPROM.read(MAGIC_ADDRESS + 2);
  checkSeconds = EEPROM.read(MAGIC_ADDRESS + 3);
  alarmEnabled = EEPROM.read(MAGIC_ADDRESS + 4) == 1;
  Serial.println("Settings loaded from EEPROM");
}
void showConfig()
{
  bt.println("--- Configuration ---");
  bt.print("high "); bt.println(highLimit);
  bt.print("low "); bt.println(lowLimit);
  bt.print("interval "); bt.println(checkSeconds);
  bt.print("alarm "); bt.println(alarmEnabled ? "on" : "off");
  bt.println("Use: set high 32");
}
void handle(String line)
{
  line.trim();
  line.toLowerCase();
  if (line.length() == 0) return;
  if (line == "config") { showConfig(); return; }
  if (line == "save") { saveSettings(); return; }
  if (line == "load") { loadSettings(); showConfig(); return; }
  if (line.startsWith("set "))
  {
    String rest = line.substring(4);
    rest.trim();
    int space = rest.indexOf(' ');
    if (space < 0)
    {
      bt.println("Use: set <name> <value>");
      return;
    }
    String name = rest.substring(0, space);
    String value = rest.substring(space + 1);
```

```
    value.trim();
    if (name == "high") { highLimit = value.toInt(); }
    else if (name == "low") { lowLimit = value.toInt(); }
    else if (name == "interval") { checkSeconds = constrain(value.toInt(), 1, 60); }
    else if (name == "alarm") { alarmEnabled = (value == "on"); }
    else
    {
      bt.println("Unknown setting. Send config");
      return;
    }
    bt.print(name);
    bt.print(" is now ");
    bt.println(value);
    Serial.print("Config change: ");
    Serial.println(line);
    return;
  }
  bt.println("Commands: config, set <name> <value>, save, load");
}
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  bt.begin(9600, SERIAL_8N1, 16, 17);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  loadSettings();
  bt.println("Remote config ready. Send config");
}
void loop()
{
  while (bt.available())
  {
    char c = bt.read();
    if (c == '\n') { handle(input); input = ""; }
    else if (c != '\r') { input = input + c; }
  }
  if (millis() - lastCheck > (unsigned long)checkSeconds * 1000)
  {
    lastCheck = millis();
    float t = dht.readTemperature();
    if (!isnan(t)) temp = t;
    alarmActive = alarmEnabled && (temp > highLimit || temp < lowLimit);
    if (alarmActive)
    {
      tone(buzzer, 1400, 150);
      bt.print("ALARM: ");
      bt.print(temp, 1);
      bt.println("C is outside limits");
    }
  }
  digitalWrite(led, alarmActive && (millis() / 400) % 2);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("REMOTE CONFIG");
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print(temp, 1);
  display.setTextSize(1);
  display.print(" C");
  display.setCursor(0, 34);
  display.print("Limits ");
  display.print(lowLimit);
  display.print(" to ");
  display.print(highLimit);
  display.print("C");
  display.setCursor(0, 44);
  display.print("Every ");
  display.print(checkSeconds);
  display.print("s alarm ");
  display.print(alarmEnabled ? "on" : "off");
  display.setCursor(0, 56);
```

```
  display.print(alarmActive ? "** OUT OF RANGE **" : "Within limits");
  display.display();
}
```

What you should see: Send config and the phone lists every current setting. Send set high 32 and the limit changes immediately, with the OLED updating. Send save and the settings survive a power cycle. Take the sensor outside the new limits and the alarm fires.

PROJECT 091 · LEVEL 14 · DISTANCE

Water Tank Level Monitor

Goal: Measure the empty space above the water and convert it into the volume remaining in the tank.

The sensor sits at the top of the tank looking down, so it measures air, not water, a full tank gives a short reading and an empty one a long reading. Subtracting that distance from the tank's height gives the depth of water, and multiplying by the cross-sectional area gives litres. Note the unit conversion: a cubic centimetre is a millilitre, so dividing by 1000 gives litres. Getting that step right is most of the work in any volume calculation.

Online page with troubleshooting: geekobot.in/projects/titan-kit-091-water-tank-level-monitor

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Distance to volume conversion
- Measuring air gap rather than the liquid
- Converting cubic centimetres to litres
- Smoothing a reading from a rippling surface

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED	**GPIO27**	Through a 220 ohm resistor
LED	Yellow LED	**GPIO26**	Through a 220 ohm resistor
LED	Red LED	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32;
int echoPin = 5;
int greenLed = 27, yellowLed = 26, redLed = 25, buzzer = 33;
// Tank dimensions in centimetres
float tankHeight = 30.0;
float tankLength = 20.0;
float tankWidth = 15.0;
float sensorGap = 2.0; // Sensor sits this far above full level
float smoothed = 0;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return tankHeight + sensorGap;
  return duration * 0.0343 / 2.0;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  smoothed = readDistance();
}
void loop()
{
  smoothed = smoothed * 0.7 + readDistance() * 0.3;
  float waterDepth = tankHeight + sensorGap - smoothed;
  waterDepth = constrain(waterDepth, 0, tankHeight);
  // cm3 to litres
  float litres = (waterDepth * tankLength * tankWidth) / 1000.0;
  float fullLitres = (tankHeight * tankLength * tankWidth) / 1000.0;
  int percent = (waterDepth / tankHeight) * 100;
  digitalWrite(greenLed, percent > 50);
  digitalWrite(yellowLed, percent > 20 && percent <= 50);
  digitalWrite(redLed, percent <= 20);
  if (percent <= 10 && (millis() / 2000) % 2 == 0)
  {
    tone(buzzer, 1200, 150);
  }
  Serial.print("Gap ");
  Serial.print(smoothed, 1);
  Serial.print("cm depth ");
  Serial.print(waterDepth, 1);
  Serial.print("cm ");
  Serial.print(litres, 2);
  Serial.println(" L");
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("WATER TANK");
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print(litres, 1);
  display.setTextSize(1);
  display.print(" L");
  display.setCursor(0, 32);
  display.print("of ");
  display.print(fullLitres, 0);
  display.print("L depth ");
  display.print(waterDepth, 1);
```

```
  display.print("cm");
  display.setCursor(0, 44);
  display.print(percent);
  display.print("% ");
  if (percent <= 10) display.print("REFILL NOW");
  else if (percent <= 20) display.print("Low");
  else if (percent >= 95) display.print("Full");
  else display.print("Normal");
  // Tank drawing
  display.drawRect(96, 10, 28, 44, SSD1306_WHITE);
  int fillHeight = map(percent, 0, 100, 0, 40);
  display.fillRect(98, 52 - fillHeight, 24, fillHeight, SSD1306_WHITE);
  display.display();
  delay(300);
}
```

What you should see: A full tank shows a short gap and a high litre figure. Lower a board towards the sensor to simulate the level falling. The tank graphic empties and the LEDs step from green to yellow to red. Below 10 percent the buzzer chirps and REFILL NOW appears.

PROJECT 092 · LEVEL 14 · DISTANCE

Trash Bin Fill Level Indicator

Goal: Detect how full a bin is, where a nearer reading means more rubbish, the inverse of the tank.

The tank and the bin use the same sensor and the same reading, but the meaning is reversed: a short distance means a full bin, not an empty one. That is why the map() call has its input range written backwards, mapping from binDepth down to zero rather than up. Reversing a map is a neat one-line trick and a common source of confusion, so it is worth reading that line carefully.

Online page with troubleshooting: geekobot.in/projects/titan-kit-092-trash-bin-fill-level-indicator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Inverse distance logic
- Reversing a map() by swapping its input range
- Edge-triggered alerts that fire once
- Counting service events

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED	**GPIO27**	Through a 220 ohm resistor
LED	Red LED	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Button	Emptied button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int greenLed = 27, redLed = 26, buzzer = 33;
int emptiedButton = 12;
float binDepth = 40.0; // cm from sensor to the bottom
int fullAbove = 80; // Percent that counts as full
float smoothed = 0;
int emptyings = 0;
bool wasFull = false;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return binDepth;
  return duration * 0.0343 / 2.0;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(emptiedButton, INPUT_PULLUP);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  smoothed = readDistance();
}
void loop()
{
  smoothed = smoothed * 0.7 + readDistance() * 0.3;
  // Reversed map: a SHORT distance means a FULL bin
  int fill = map(constrain((int)smoothed, 3, (int)binDepth),
  binDepth, 3, 0, 100);
  fill = constrain(fill, 0, 100);
  if (digitalRead(emptiedButton) == LOW)
  {
    emptyings = emptyings + 1;
    wasFull = false;
    tone(buzzer, 900, 200);
    Serial.println("Bin emptied");
    delay(400);
  }
  bool isFull = (fill >= fullAbove);
  digitalWrite(redLed, isFull);
  digitalWrite(greenLed, !isFull);
  if (isFull && !wasFull)
  {
    wasFull = true;
    tone(buzzer, 1400, 300);
    Serial.println("Bin needs emptying");
  }
  Serial.print("Gap ");
  Serial.print(smoothed, 1);
  Serial.print("cm fill ");
  Serial.print(fill);
  Serial.println("%");
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("BIN MONITOR");
  display.setTextSize(3);
  display.setCursor(0, 12);
  display.print(fill);
  display.setTextSize(1);
  display.print("%");
```

```
  display.setCursor(0, 40);
  if (fill >= 95) display.print("OVERFLOWING");
  else if (isFull) display.print("FULL - empty it");
  else if (fill >= 50) display.print("Half full");
  else display.print("Plenty of room");
  display.setCursor(0, 52);
  display.print("Gap ");
  display.print(smoothed, 0);
  display.print("cm emptied ");
  display.print(emptyings);
  // Bin drawing
  display.drawLine(92, 12, 96, 54, SSD1306_WHITE);
  display.drawLine(126, 12, 122, 54, SSD1306_WHITE);
  display.drawLine(96, 54, 122, 54, SSD1306_WHITE);
  int level = map(fill, 0, 100, 0, 40);
  display.fillRect(97, 53 - level, 24, level, SSD1306_WHITE);
  display.display();
  delay(300);
}
```

What you should see: An empty bin reads 0% with the green LED lit. Move your hand up towards the sensor and the percentage climbs. At 80% the red LED lights and the buzzer sounds once. Press the emptied button to log a collection and reset the alert.

Digital Height Measurement System

Goal: Measure a person's height by mounting the sensor above them and subtracting their head clearance from the mounting height.

Sound travels about 343 metres per second at room temperature, which is 0.0343 cm per microsecond, and the pulse makes a round trip, hence the division by two. That speed changes with temperature by roughly 0.6 m/s per degree, so the sketch includes the correction. Taking the median of nine readings rather than the average is deliberate: a single wild reflection would drag an average off, but it cannot move the middle value.

Online page with troubleshooting: geekobot.in/projects/titan-kit-093-digital-height-measurement

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Speed of sound mathematics
- Temperature correction for acoustic measurement
- Median filtering to reject wild readings
- Converting between metric and imperial units

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Measure button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int measureButton = 12;
int buzzer = 33;
float mountHeight = 200.0; // Sensor height above the floor in cm
float airTemperature = 25.0; // Used to correct the speed of sound
float lastHeight = 0;
float tallest = 0;
int measurements = 0;
float speedPerMicrosecond()
{
  // 331.3 + 0.606 * T metres per second, converted to cm per us
  float metresPerSecond = 331.3 + 0.606 * airTemperature;
  return metresPerSecond / 10000.0;
}
float singleReading()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return -1;
  return duration * speedPerMicrosecond() / 2.0;
}
float medianDistance()
{
  float samples[9];
  int valid = 0;
  for (int i = 0; i < 9; i++)
  {
    float d = singleReading();
    if (d > 0)
    {
      samples[valid] = d;
      valid = valid + 1;
    }
    delay(40);
  }
  if (valid == 0) return -1;
  // Sort, then take the middle value
  for (int i = 0; i < valid - 1; i++)
  {
    for (int j = 0; j < valid - 1 - i; j++)
    {
      if (samples[j] > samples[j + 1])
      {
        float swap = samples[j];
        samples[j] = samples[j + 1];
        samples[j + 1] = swap;
      }
    }
  }
  return samples[valid / 2];
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(measureButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  if (digitalRead(measureButton) == LOW)
  {
```

```
    display.clearDisplay();
    display.setCursor(0, 24);
    display.print("Stand still...");
    display.display();
    tone(buzzer, 900, 100);
    delay(600);
    float headGap = medianDistance();
    if (headGap > 0)
    {
      lastHeight = mountHeight - headGap;
      measurements = measurements + 1;
      if (lastHeight > tallest) tallest = lastHeight;
      tone(buzzer, 1600, 200);
      Serial.print("Head gap ");
      Serial.print(headGap, 1);
      Serial.print("cm height ");
      Serial.print(lastHeight, 1);
      Serial.println("cm");
    }
    delay(500);
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("HEIGHT MEASURE");
  if (lastHeight > 0)
  {
    display.setTextSize(3);
    display.setCursor(0, 14);
    display.print(lastHeight, 1);
    display.setTextSize(1);
    display.setCursor(96, 24);
    display.print("cm");
    int feet = (int)(lastHeight / 30.48);
    int inches = (int)((lastHeight / 2.54) - feet * 12);
    display.setCursor(0, 42);
    display.print("That is ");
    display.print(feet);
    display.print("ft ");
    display.print(inches);
    display.print("in");
  }
  else
  {
    display.setTextSize(2);
    display.setCursor(0, 20);
    display.print("READY");
  }
  display.setCursor(0, 54);
  display.print("N=");
  display.print(measurements);
  display.print(" tallest ");
  display.print(tallest, 0);
  display.print("cm");
  display.display();
  delay(150);
}
```

What you should see: Press the button and a beep tells the person to stand still. Nine readings are taken and the median used. The height appears in centimetres and also in feet and inches. The tallest measurement of the session is kept at the bottom.

Queue Distance Monitor

Goal: Check that people waiting in a line are keeping a safe distance apart, using zones rather than a single threshold.

Zoned tracking divides a continuous distance into named regions with fixed boundaries, and the useful addition here is time. Momentarily stepping close is not a problem; standing close for ten seconds is. The sketch therefore tracks how long the current zone has been occupied and only escalates once that time passes, which is the same sustained-condition idea used by the noise monitor, applied to space instead of sound.

Online page with troubleshooting: geekobot.in/projects/titan-kit-094-queue-distance-monitor

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Zoned distance tracking
- Timing how long a condition persists
- Drawing zone boundaries on a scale
- Counting violations rather than just flagging them

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED (safe)	**GPIO27**	Through a 220 ohm resistor
LED	Yellow LED (close)	**GPIO26**	Through a 220 ohm resistor
LED	Red LED (too close)	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define ZONES 3
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int ledPins[ZONES] = {27, 26, 25};
int buzzer = 33;
int zoneLimit[ZONES] = {60, 120, 400}; // cm
const char* zoneName[ZONES] = {"TOO CLOSE", "CLOSE", "SAFE"};
int currentZone = 2;
unsigned long zoneSince = 0;
unsigned long warnAfter = 5000;
int violations = 0;
float smoothed = 200;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return 400;
  return duration * 0.0343 / 2.0;
}
int zoneFor(float distance)
{
  for (int i = 0; i < ZONES; i++)
  {
    if (distance < zoneLimit[i]) return i;
  }
  return ZONES - 1;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  for (int i = 0; i < ZONES; i++) pinMode(ledPins[i], OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  zoneSince = millis();
}
void loop()
{
  smoothed = smoothed * 0.6 + readDistance() * 0.4;
  int zone = zoneFor(smoothed);
  if (zone != currentZone)
  {
    currentZone = zone;
    zoneSince = millis();
    Serial.print("Zone: ");
    Serial.println(zoneName[zone]);
  }
  unsigned long inZone = millis() - zoneSince;
  for (int i = 0; i < ZONES; i++)
  {
    digitalWrite(ledPins[i], i == zone);
  }
  bool warning = (zone == 0 && inZone > warnAfter);
  if (warning && (millis() / 700) % 2 == 0)
  {
    tone(buzzer, 1300, 180);
  }
  if (zone == 0 && inZone > warnAfter && inZone < warnAfter + 100)
  {
    violations = violations + 1;
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("QUEUE MONITOR");
```

```
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print((int)smoothed);
  display.setTextSize(1);
  display.print(" cm");
  display.setCursor(70, 18);
  display.print(zoneName[zone]);
  // Zone bar with boundaries
  display.drawRect(0, 34, 128, 10, SSD1306_WHITE);
  int marker = map(constrain((int)smoothed, 0, 200), 0, 200, 1, 125);
  display.fillRect(marker, 35, 3, 8, SSD1306_WHITE);
  for (int i = 0; i < ZONES - 1; i++)
  {
    int x = map(zoneLimit[i], 0, 200, 0, 126);
    display.drawLine(x, 32, x, 46, SSD1306_WHITE);
  }
  display.setCursor(0, 48);
  display.print("In zone ");
  display.print(inZone / 1000);
  display.print("s");
  display.setCursor(0, 56);
  if (warning) display.print("PLEASE STEP BACK");
  else if (zone == 0) display.print("Close - watch the time");
  else display.print("Violations: ");
  if (!warning && zone != 0)
  {
    display.print(violations);
  }
  display.display();
  delay(150);
}
```

What you should see: Standing well back shows SAFE with the green LED. Move to about a metre: CLOSE with the yellow LED. Move inside 60 cm: TOO CLOSE, red LED and a running timer. Stay there for 5 seconds and PLEASE STEP BACK appears with a beeping alert.

Contactless Distance Switch

Goal: Switch something on and off by hovering a hand at a set range, with hysteresis so it does not flicker at the boundary.

Proximity hysteresis means using two thresholds instead of one: the switch turns on inside 15 cm but only turns off beyond 25 cm. That gap is deliberate, with a single threshold, a hand hovering at exactly 20 cm would toggle the output many times a second as the reading wobbled. Add a short dwell requirement and the switch also ignores a hand that merely passes through on its way somewhere else.

Online page with troubleshooting: geekobot.in/projects/titan-kit-095-contactless-distance-switch

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Proximity hysteresis
- Separate activate and release thresholds
- Dwell time to ignore passing movement
- Preventing repeated toggles from one gesture

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	LED anode	**GPIO25**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
Passive buzzer	Piezo buzzer (+)	**GPIO33**	
Passive buzzer	Piezo buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int led = 25;
int buzzer = 33;
int onBelow = 15; // Activate inside this
int offAbove = 25; // Only release beyond this
unsigned long dwell = 400; // Hand must stay this long
bool switchOn = false;
bool handPresent = false;
unsigned long handSince = 0;
int activations = 0;
float smoothed = 100;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return 400;
  return duration * 0.0343 / 2.0;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  smoothed = smoothed * 0.5 + readDistance() * 0.5;
  bool nearNow = (smoothed < onBelow);
  bool farNow = (smoothed > offAbove);
  if (nearNow && !handPresent)
  {
    handPresent = true;
    handSince = millis();
  }
  if (farNow)
  {
    handPresent = false;
  }
  // Toggle once the hand has dwelled long enough
  if (handPresent && millis() - handSince > dwell && handSince != 0)
  {
    switchOn = !switchOn;
    activations = activations + 1;
    handSince = 0; // Only one toggle per approach
    tone(buzzer, switchOn ? 1600 : 800, 150);
    Serial.print("Switched ");
    Serial.println(switchOn ? "ON" : "OFF");
  }
  digitalWrite(led, switchOn);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("HOVER SWITCH");
  display.setTextSize(3);
  display.setCursor(0, 14);
  display.print(switchOn ? "ON" : "OFF");
  display.setTextSize(1);
  display.setCursor(70, 20);
  display.print((int)smoothed);
  display.print(" cm");
  display.setCursor(0, 42);
  display.print("On < ");
```

```
  display.print(onBelow);
  display.print("cm off > ");
  display.print(offAbove);
  display.print("cm");
  display.setCursor(0, 52);
  if (handPresent && handSince > 0)
  {
    display.print("Hold... ");
    display.print(dwell - (millis() - handSince));
    display.print("ms");
  }
  else
  {
    display.print("Uses: ");
    display.print(activations);
    display.print(" hover to switch");
  }
  display.display();
  delay(60);
}
```

What you should see: Hold your hand within 15 cm for 0.4 seconds and the LED turns on with a beep. Take your hand away past 25 cm to release the switch. Hover again to turn it off, one toggle per approach. Hold at exactly 20 cm and nothing flickers, thanks to the two thresholds.

Safe Following Distance Indicator

Goal: Simulate a car's radar warning, where the safe distance grows with speed rather than staying fixed.

A fixed warning distance is wrong at every speed but one. The two-second rule says the safe gap is however far you travel in two seconds, so it scales with speed, and because this is a desk model, real metres are scaled down to centimetres. The threshold being computed rather than stored is the whole idea: a dynamic threshold adapts to conditions, which is what makes an adaptive system adaptive.

Online page with troubleshooting: geekobot.in/projects/titan-kit-096-safe-following-distance-indicator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Dynamic distance thresholds
- The two-second rule as a formula
- Scaling a real-world model to a desk
- Thresholds computed from conditions rather than fixed

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Potentiometer	Potentiometer wiper	**GPIO34**	
Potentiometer	Potentiometer ends	**3V3 and GND**	
LED	Green LED	**GPIO27**	Through a 220 ohm resistor
LED	Yellow LED	**GPIO26**	Through a 220 ohm resistor
LED	Red LED	**GPIO25**	Through a 220 ohm resistor
Passive buzzer	Buzzer (+)	**GPIO33**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int greenLed = 27, yellowLed = 26, redLed = 25, buzzer = 33;
int speedPin = 34;
float modelScale = 0.02; // Scales real metres down to desk centimetres
float smoothed = 200;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return 400;
  return duration * 0.0343 / 2.0;
}
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  smoothed = smoothed * 0.6 + readDistance() * 0.4;
  int speedKmh = map(analogRead(speedPin), 0, 1023, 0, 120);
  // Two second rule: metres travelled in 2 s, scaled to the model
  float metresPerSecond = speedKmh / 3.6;
  float safeMetres = metresPerSecond * 2.0;
  float safeCm = safeMetres / modelScale / 100.0;
  safeCm = constrain(safeCm, 10, 300);
  float warnCm = safeCm * 0.7;
  digitalWrite(greenLed, LOW);
  digitalWrite(yellowLed, LOW);
  digitalWrite(redLed, LOW);
  String verdict;
  if (smoothed > safeCm)
  {
    verdict = "SAFE GAP";
    digitalWrite(greenLed, HIGH);
  }
  else if (smoothed > warnCm)
  {
    verdict = "CLOSING IN";
    digitalWrite(yellowLed, HIGH);
    if ((millis() / 900) % 2 == 0) tone(buzzer, 900, 100);
  }
  else
  {
    verdict = "BRAKE NOW";
    digitalWrite(redLed, HIGH);
    if ((millis() / 300) % 2 == 0) tone(buzzer, 1800, 120);
  }
  Serial.print(speedKmh);
  Serial.print(" km/h safe ");
  Serial.print(safeCm, 0);
  Serial.print("cm actual ");
  Serial.println(smoothed, 0);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("FOLLOWING DISTANCE");
  display.setTextSize(2);
```

```
  display.setCursor(0, 12);
  display.print(speedKmh);
  display.setTextSize(1);
  display.print(" km/h");
  display.setCursor(0, 30);
  display.print("Gap ");
  display.print((int)smoothed);
  display.print("cm need ");
  display.print((int)safeCm);
  display.print("cm");
  display.setTextSize(2);
  display.setCursor(0, 40);
  display.print(verdict);
  display.setTextSize(1);
  display.drawRect(0, 58, 128, 6, SSD1306_WHITE);
  int gapBar = map(constrain((int)smoothed, 0, 200), 0, 200, 0, 126);
  display.fillRect(1, 59, gapBar, 4, SSD1306_WHITE);
  int needMark = map(constrain((int)safeCm, 0, 200), 0, 200, 0, 126);
  display.drawLine(needMark, 56, needMark, 63, SSD1306_WHITE);
  display.display();
  delay(120);
}
```

What you should see: At 20 km/h a short gap is still reported as SAFE GAP. Turn the speed knob to 100 km/h and the same gap becomes BRAKE NOW. The required distance on screen grows as the speed rises. A marker on the bar shows where the safe gap currently sits.

Object Speed Estimator

Goal: Work out how fast something is moving by measuring how much the distance changed and how long that took.

Speed is change in distance divided by change in time, the definition of a rate, and the first derivative in disguise. Working in signed values keeps the direction: a negative speed means the object is approaching, a positive one that it is retreating. Because dividing two noisy numbers amplifies noise badly, the interval is kept at a full 100 ms and the result is heavily smoothed; a shorter interval gives a faster but far wilder reading.

Online page with troubleshooting: geekobot.in/projects/titan-kit-097-object-speed-estimator

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Delta distance over delta time, velocity
- Signed values to preserve direction
- Why dividing noisy readings needs heavy smoothing
- Tracking a peak approach speed

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED (moving away)	**GPIO27**	Through a 220 ohm resistor
LED	Red LED (approaching)	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int greenLed = 27, redLed = 26, buzzer = 33;
float lastDistance = 0;
unsigned long lastTime = 0;
float speedCmS = 0;
float fastestApproach = 0;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return -1;
  return duration * 0.0343 / 2.0;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  lastDistance = readDistance();
  lastTime = millis();
}
void loop()
{
  float distance = readDistance();
  if (distance < 0)
  {
    delay(80);
    return;
  }
  unsigned long now = millis();
  unsigned long elapsed = now - lastTime;
  if (elapsed >= 100)
  {
    float change = distance - lastDistance; // cm
    float seconds = elapsed / 1000.0;
    float instant = change / seconds; // cm per second
    // Heavy smoothing - dividing noisy numbers is noisy
    speedCmS = speedCmS * 0.7 + instant * 0.3;
    lastDistance = distance;
    lastTime = now;
    if (-speedCmS > fastestApproach)
    {
      fastestApproach = -speedCmS;
    }
    Serial.print("Distance ");
    Serial.print(distance, 1);
    Serial.print(" speed ");
    Serial.println(speedCmS, 1);
  }
  bool approaching = (speedCmS < -3);
  bool retreating = (speedCmS > 3);
  digitalWrite(redLed, approaching);
  digitalWrite(greenLed, retreating);
  if (approaching && abs(speedCmS) > 40 && (millis() / 400) % 2 == 0)
  {
    tone(buzzer, 1600, 120);
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("SPEED ESTIMATOR");
```

```
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print(abs(speedCmS), 0);
  display.setTextSize(1);
  display.print(" cm/s");
  display.setCursor(0, 32);
  display.print("= ");
  display.print(abs(speedCmS) * 0.036, 2);
  display.print(" km/h");
  display.setCursor(0, 42);
  display.print("Distance ");
  display.print(distance, 0);
  display.print("cm");
  display.setCursor(0, 52);
  if (approaching) display.print("<<< APPROACHING");
  else if (retreating) display.print("RETREATING >>>");
  else display.print("Stationary");
  display.setCursor(80, 52);
  display.print("pk ");
  display.print(fastestApproach, 0);
  display.display();
}
```

What you should see: A still object shows Stationary with a speed near zero. Move your hand towards the sensor: <<< APPROACHING with the red LED. Move it away: RETREATING >>> with the green LED. The speed is shown in cm/s and converted to km/h alongside.

Distance Based RGB Visualization

Goal: Map distance onto a smooth colour wheel so an RGB LED changes hue continuously as an object moves.

Distance is mapped to a hue angle from 0 to 360 degrees, and the hue-to-RGB conversion splits that circle into six 60-degree sectors where exactly one channel is ramping at a time. It is more code than a simple three-band gradient, but it is the standard conversion and it produces genuinely even colour transitions. Brightness is faded near the far limit so a distant object glows softly rather than cutting out abruptly.

Online page with troubleshooting: geekobot.in/projects/titan-kit-098-distance-rgb-visualization

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Distance mapping to a colour wheel
- The standard six-sector hue-to-RGB conversion
- Fading brightness at the edge of a range
- Reading a hue angle as a position on a circle

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
RGB LED	Red	**GPIO12**	Through a 220 ohm resistor
RGB LED	Green	**GPIO27**	Through a 220 ohm resistor
RGB LED	Blue	**GPIO26**	Through a 220 ohm resistor
RGB LED	Common cathode	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int redPin = 12, greenPin = 27, bluePin = 26;
int nearLimit = 5;
int farLimit = 120;
float smoothed = 60;
int r, g, b;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return farLimit + 20;
  return duration * 0.0343 / 2.0;
}
void hueToRGB(float hue, float brightness)
{
  float sector = hue / 60.0;
  if (sector < 1) { r = 255; g = sector * 255; b = 0; }
  else if (sector < 2) { r = (2 - sector) * 255; g = 255; b = 0; }
  else if (sector < 3) { r = 0; g = 255; b = (sector - 2) * 255; }
  else if (sector < 4) { r = 0; g = (4 - sector) * 255; b = 255; }
  else if (sector < 5) { r = (sector - 4) * 255; g = 0; b = 255; }
  else { r = 255; g = 0; b = (6 - sector) * 255; }
  r = r * brightness;
  g = g * brightness;
  b = b * brightness;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  smoothed = smoothed * 0.6 + readDistance() * 0.4;
  float clamped = constrain(smoothed, nearLimit, farLimit);
  float hue = map(clamped * 10, nearLimit * 10, farLimit * 10, 0, 3599) / 10.0;
  // Fade out as the object reaches the far limit
  float brightness = 1.0;
  if (smoothed > farLimit - 20)
  {
    brightness = constrain((farLimit - smoothed) / 20.0, 0.1, 1.0);
  }
  hueToRGB(hue, brightness);
  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);
  Serial.print(smoothed, 0);
  Serial.print("cm hue ");
  Serial.print(hue, 0);
  Serial.print(" RGB ");
  Serial.print(r); Serial.print(",");
  Serial.print(g); Serial.print(",");
  Serial.println(b);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("DISTANCE COLOUR");
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print((int)smoothed);
```

```
  display.setTextSize(1);
  display.print(" cm");
  display.setCursor(0, 32);
  display.print("Hue ");
  display.print(hue, 0);
  display.print((char)247);
  display.setCursor(64, 32);
  display.print("R");
  display.print(r / 10);
  display.print(" G");
  display.print(g / 10);
  display.print(" B");
  display.print(b / 10);
  // Position on the colour scale
  display.drawRect(0, 44, 128, 12, SSD1306_WHITE);
  int marker = map((int)hue, 0, 359, 2, 124);
  display.fillRect(marker, 46, 3, 8, SSD1306_WHITE);
  display.setCursor(0, 58);
  display.print("red");
  display.setCursor(52, 58);
  display.print("cyan");
  display.setCursor(104, 58);
  display.print("red");
  display.display();
  delay(40);
}
```

What you should see: An object at 5 cm makes the LED glow red. Move it back and the colour sweeps through yellow, green, cyan and blue. The OLED shows the hue angle and the three channel values. Beyond about 100 cm the LED dims smoothly rather than cutting out.

PROJECT 099 · LEVEL 14 · DISTANCE

Smart Measuring Tape

Goal: Build a precise digital tape measure with fast polling, unit switching and a memory of recent measurements.

Precision comes from polling quickly and reporting steadily. The sensor is read as fast as it allows, but the value shown is an average of the last eight readings, and a stability check compares the newest reading with that average, so the display can tell you when it is safe to record. Three unit systems are offered from one stored value in centimetres, because converting on display rather than on capture keeps the stored data clean.

Online page with troubleshooting: geekobot.in/projects/titan-kit-099-smart-measuring-tape

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- High-precision polling
- Averaging a sliding window of samples
- Detecting when a reading has settled
- Converting units at display time, not at capture

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Store measurement	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Change units	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define WINDOW 8
#define MEMORY 4
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int storeButton = 12, unitButton = 14, buzzer = 33;
float samples[WINDOW];
int sampleIndex = 0;
bool windowFull = false;
float stored_[MEMORY];
int storedCount = 0;
int units = 0; // 0 cm, 1 mm, 2 inches
const char* unitName[3] = {"cm", "mm", "in"};
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return -1;
  return duration * 0.0343 / 2.0;
}
float average()
{
  int count = windowFull ? WINDOW : sampleIndex;
  if (count == 0) return 0;
  float total = 0;
  for (int i = 0; i < count; i++) total = total + samples[i];
  return total / count;
}
float convert(float cm)
{
  if (units == 1) return cm * 10.0;
  if (units == 2) return cm / 2.54;
  return cm;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(storeButton, INPUT_PULLUP);
  pinMode(unitButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  float reading = readDistance();
  if (reading > 0)
  {
    samples[sampleIndex] = reading;
    sampleIndex = sampleIndex + 1;
    if (sampleIndex >= WINDOW)
    {
      sampleIndex = 0;
      windowFull = true;
    }
  }
  float value = average();
  bool steady = (reading > 0 && abs(reading - value) < 0.5);
  if (digitalRead(unitButton) == LOW)
  {
    units = (units + 1) % 3;
    tone(buzzer, 1000, 60);
    delay(250);
  }
  if (digitalRead(storeButton) == LOW)
```

```
  {
    if (storedCount >= MEMORY)
    {
      for (int i = 0; i < MEMORY - 1; i++) stored_[i] = stored_[i + 1];
      storedCount = MEMORY - 1;
    }
    stored_[storedCount] = value;
    storedCount = storedCount + 1;
    tone(buzzer, 1600, 150);
    Serial.print("Stored ");
    Serial.print(value, 2);
    Serial.println(" cm");
    delay(350);
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("MEASURING TAPE");
  display.setCursor(96, 0);
  display.print(steady ? "STEADY" : "...");
  display.setTextSize(3);
  display.setCursor(0, 12);
  display.print(convert(value), units == 1 ? 0 : 1);
  display.setTextSize(1);
  display.setCursor(96, 24);
  display.print(unitName[units]);
  display.drawLine(0, 38, 127, 38, SSD1306_WHITE);
  display.setCursor(0, 42);
  display.print("Memory:");
  for (int i = 0; i < storedCount; i++)
  {
    display.setCursor((i % 2) * 64, 52 + (i / 2) * 0);
    if (i < 2)
    {
      display.print(convert(stored_[i]), 1);
      display.print(unitName[units]);
      display.print(" ");
    }
  }
  if (storedCount == 0)
  {
    display.setCursor(44, 42);
    display.print("empty - D3 stores");
  }
  display.display();
}
```

What you should see: The measurement updates continuously with a STEADY flag when it settles. Press the second button to switch between centimetres, millimetres and inches. Press the first button to store the current value into memory. The last stored measurements are listed at the bottom in the current units.

Parking Slot Occupancy Detector

Goal: Report whether a parking space is occupied, requiring the vehicle to be present continuously before the state changes.

A person walking past a bay is not a parked car, so presence alone is not enough, the reading has to hold. Sustained presence detection requires the condition to be continuously true for three seconds before the state flips, and the same rule applies to leaving. Once the state does change, the sketch records when it happened, which turns a simple detector into something that can report how long the bay has been occupied.

Online page with troubleshooting: geekobot.in/projects/titan-kit-100-parking-slot-occupancy-detector

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Sustained presence detection
- Confirmation delays before changing state
- Timing how long a state has held
- Ignoring transient objects passing through

Wiring

Part	Pin	ESP32	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO32**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
LED	Green LED (free)	**GPIO27**	Through a 220 ohm resistor
LED	Red LED (occupied)	**GPIO26**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Passive buzzer	Buzzer (+)	**GPIO33**	
Passive buzzer	Buzzer (−)	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int trigPin = 32, echoPin = 5;
int greenLed = 27, redLed = 26, buzzer = 33;
int occupiedBelow = 40; // cm
unsigned long confirmTime = 3000;
bool occupied = false;
bool pendingState = false;
unsigned long pendingSince = 0;
unsigned long stateChangedAt = 0;
int parkEvents = 0;
unsigned long longestStay = 0;
float smoothed = 100;
float readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH, 30000);
  if (duration == 0) return 400;
  return duration * 0.0343 / 2.0;
}
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  stateChangedAt = millis();
}
void loop()
{
  smoothed = smoothed * 0.7 + readDistance() * 0.3;
  bool seesSomething = (smoothed < occupiedBelow);
  if (seesSomething != occupied)
  {
    if (seesSomething != pendingState)
    {
      pendingState = seesSomething;
      pendingSince = millis();
    }
    else if (millis() - pendingSince > confirmTime)
    {
      unsigned long previousStay = millis() - stateChangedAt;
      occupied = seesSomething;
      stateChangedAt = millis();
      if (occupied)
      {
        parkEvents = parkEvents + 1;
        tone(buzzer, 1200, 200);
        Serial.println("Vehicle parked");
      }
      else
      {
        if (previousStay > longestStay) longestStay = previousStay;
        tone(buzzer, 800, 200);
        Serial.print("Vehicle left after ");
        Serial.print(previousStay / 1000);
        Serial.println("s");
      }
    }
  }
  else
  {
    pendingState = occupied;
```

```
  }
  digitalWrite(redLed, occupied);
  digitalWrite(greenLed, !occupied);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("PARKING SLOT A1");
  display.setTextSize(3);
  display.setCursor(0, 14);
  display.print(occupied ? "FULL" : "FREE");
  display.setTextSize(1);
  display.setCursor(0, 40);
  display.print("Gap ");
  display.print((int)smoothed);
  display.print("cm ");
  display.print((millis() - stateChangedAt) / 1000);
  display.print("s");
  display.setCursor(0, 50);
  if (pendingState != occupied)
  {
    display.print(pendingState ? "Arriving... " : "Leaving... ");
    display.print((confirmTime - (millis() - pendingSince)) / 1000 + 1);
    display.print("s");
  }
  else
  {
    display.print("Parked today: ");
    display.print(parkEvents);
  }
  display.setCursor(0, 58);
  display.print("Longest stay ");
  display.print(longestStay / 1000);
  display.print("s");
  display.display();
  delay(150);
}
```

What you should see: An empty bay shows FREE with the green LED. Place an object within 40 cm: Arriving... 3s counts down. After 3 continuous seconds the state changes to FULL with a beep. Wave a hand past quickly and nothing changes at all.

Servo Based Analog Meter

Goal: Turn a servo into a physical needle gauge whose arm points to the current sensor value on a printed scale.

The whole experiment is one idea: any measured quantity can be turned into a physical angle. map() converts the sensor's range onto the servo's usable sweep, and the needle does the rest. Two details make it feel like a real instrument. The needle moves gradually towards its target rather than snapping, which imitates the inertia of a real moving-coil meter; and the sweep is kept inside 10 to 170 degrees, because SG90 servos buzz and strain at their mechanical limits.

Online page with troubleshooting: geekobot.in/projects/titan-kit-101-servo-analog-meter

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 10k resistor
- 1 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Variable to physical angle mapping
- Easing movement to imitate mechanical inertia
- Keeping a servo clear of its hard stops
- Driving a physical and an on-screen display together

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Servo	Signal (orange)	**GPIO26**	
Servo	Red	**5V**	
Servo	Brown / black	**GND**	
Potentiometer	Potentiometer wiper	**GPIO34**	
Potentiometer	Potentiometer ends	**3V3 and GND**	
LDR divider	LDR divider output	**GPIO35**	
Button	Source button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Servo needle;
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int sourceButton = 12;
int minAngle = 10; // Keep clear of the servo's hard stops
int maxAngle = 170;
float currentAngle = 90;
int targetAngle = 90;
int source = 0; // 0 potentiometer, 1 light level
const char* sourceName[2] = {"MANUAL", "LIGHT"};
const char* sourceUnit[2] = {"%", "lux%"};
int reading = 0;
void setup()
{
  analogReadResolution(10); // ESP32 is 12-bit; use 0-1023 like the examples
  pinMode(sourceButton, INPUT_PULLUP);
  needle.attach(26);
  needle.write((int)currentAngle);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  if (digitalRead(sourceButton) == LOW)
  {
    source = (source + 1) % 2;
    Serial.print("Source: ");
    Serial.println(sourceName[source]);
    delay(300);
  }
  if (source == 0)
  {
    reading = map(analogRead(34), 0, 1023, 0, 100);
  }
  else
  {
    reading = map(analogRead(35), 0, 1023, 0, 100);
  }
  // The one line that matters: value becomes a physical angle
  targetAngle = map(reading, 0, 100, minAngle, maxAngle);
  // Ease towards the target so the needle behaves like a real one
  currentAngle = currentAngle * 0.85 + targetAngle * 0.15;
  needle.write((int)currentAngle);
  Serial.print(sourceName[source]);
  Serial.print(" ");
  Serial.print(reading);
  Serial.print(" angle ");
  Serial.println((int)currentAngle);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("ANALOG METER");
  display.setCursor(88, 0);
  display.print(sourceName[source]);
  display.setTextSize(3);
  display.setCursor(0, 14);
  display.print(reading);
  display.setTextSize(1);
  display.print(sourceUnit[source]);
  display.setCursor(0, 42);
  display.print("Needle at ");
  display.print((int)currentAngle);
  display.print((char)247);
  // Draw the same gauge on screen
  int cx = 100;
  int cy = 52;
  display.drawLine(cx - 24, cy, cx + 24, cy, SSD1306_WHITE);
  float radians = (180 - currentAngle) * PI / 180.0;
  display.drawLine(cx, cy,
  cx + cos(radians) * 22,
```

```
  cy - sin(radians) * 22, SSD1306_WHITE);
  for (int mark = 0; mark <= 4; mark++)
  {
    float a = (180 - (minAngle + (maxAngle - minAngle) * mark / 4.0)) * PI / 180.0;
    display.drawPixel(cx + cos(a) * 25, cy - sin(a) * 25, SSD1306_WHITE);
  }
  display.setCursor(0, 54);
  display.print("D3 changes source");
  display.display();
  delay(30);
}
```

What you should see: Turn the potentiometer and the servo arm sweeps across the printed scale. The OLED shows the value and the exact needle angle. Press the first button to switch the meter to the light sensor instead. The needle glides into position rather than jumping.

Servo Controlled Pointer Display

Goal: Use the servo to point at one of several labelled categories rather than a continuous scale.

A continuous meter can rest anywhere; a category pointer must land exactly on one of five positions. The angles are stored in an array, so choosing a category is choosing an index, categorical physical indexing. Because the pointer should not hover between two labels while a reading sits on a boundary, the sketch also requires the new category to hold for a moment before the servo moves, which stops a twitching needle.

Online page with troubleshooting: geekobot.in/projects/titan-kit-102-servo-controlled-pointer

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Categorical physical indexing
- Fixed positions instead of a continuous sweep
- Settle time to stop boundary flapping
- Manual override alongside automatic control

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Servo	Servo signal	**GPIO26**	
Servo	Servo red / brown	**5V / GND**	
DHT11	DHT11 DATA	**GPIO14**	
DHT11	DHT11 VCC / GND	**3V3 / GND**	
Button	Step button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO33**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <DHT.h>
#define DHTPIN 14
#define DHTTYPE DHT11
#define CATEGORIES 5
Servo pointer;
DHT dht(DHTPIN, DHTTYPE);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int stepButton = 12;
int buzzer = 33;
// One fixed angle per category
int categoryAngle[CATEGORIES] = {20, 55, 90, 125, 160};
const char* categoryName[CATEGORIES] =
{
  "COLD", "COOL", "IDEAL", "WARM", "HOT"
};
float upperLimit[CATEGORIES] = {18, 22, 27, 31, 99};
int category = 2;
int candidate = 2;
unsigned long candidateSince = 0;
unsigned long settleTime = 1500;
bool manualMode = false;
float temp = 0;
unsigned long lastRead = 0;
int categoryFor(float t)
{
  for (int i = 0; i < CATEGORIES; i++)
  {
    if (t < upperLimit[i]) return i;
  }
  return CATEGORIES - 1;
}
void moveTo(int index)
{
  pointer.write(categoryAngle[index]);
  tone(buzzer, 1200, 80);
  Serial.print("Pointing at ");
  Serial.print(categoryName[index]);
  Serial.print(" (");
  Serial.print(categoryAngle[index]);
  Serial.println(" degrees)");
}
void setup()
{
  pinMode(stepButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  pointer.attach(26);
  pointer.write(categoryAngle[category]);
  Serial.begin(9600);
  dht.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  if (digitalRead(stepButton) == LOW)
  {
    manualMode = true;
    category = (category + 1) % CATEGORIES;
    candidate = category;
    moveTo(category);
    delay(350);
  }
  if (millis() - lastRead > 2000)
  {
    lastRead = millis();
    float t = dht.readTemperature();
    if (!isnan(t)) temp = t;
    if (!manualMode)
    {
```

```
      int wanted = categoryFor(temp);
      if (wanted != candidate)
      {
        candidate = wanted;
        candidateSince = millis();
      }
      else if (candidate != category && millis() - candidateSince > settleTime)
      {
        category = candidate;
        moveTo(category);
      }
    }
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("POINTER DISPLAY");
  display.setCursor(84, 0);
  display.print(manualMode ? "MANUAL" : "AUTO");
  display.setTextSize(2);
  display.setCursor(0, 14);
  display.print(categoryName[category]);
  display.setTextSize(1);
  display.setCursor(0, 34);
  display.print("Temp ");
  display.print(temp, 1);
  display.print("C arm ");
  display.print(categoryAngle[category]);
  display.print((char)247);
  // Show all five positions with the active one filled
  for (int i = 0; i < CATEGORIES; i++)
  {
    int x = i * 25 + 3;
    display.drawRect(x, 44, 21, 12, SSD1306_WHITE);
    if (i == category)
    {
      display.fillRect(x + 2, 46, 17, 8, SSD1306_WHITE);
    }
  }
  display.setCursor(0, 58);
  display.print("D3 steps manually");
  display.display();
  delay(60);
}
```

What you should see: The arm points at IDEAL at room temperature. Warm the sensor by hand and after a moment the arm swings to WARM. Five boxes on screen show which position the arm is in. Press the first button to step the pointer manually through every category.

Stepper Motor Position Memory System

Goal: Count steps to keep track of absolute position, save named positions to EEPROM, and return to them exactly.

A stepper has no position sensor, so the sketch must count. Every step command adds or subtracts one from a running total, and that total is the position, which is why the motor must never be turned by hand while powered off, and why the count is meaningless until it is homed. The 28BYJ-48 takes 2048 steps per revolution through its gearbox, so each step is about 0.176 degrees. Saving that count to EEPROM lets the position survive a power cut.

Online page with troubleshooting: geekobot.in/projects/titan-kit-103-stepper-motor-position-memory

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x 28BYJ-48 Stepper Motor & ULN2003 Driver Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Step counting for absolute position
- State saving to EEPROM with put() and get()
- Long press and short press from one button
- Why an open-loop motor must never be moved by hand

Wiring

Part	Pin	ESP32	Note
Stepper driver	IN1	**GPIO33**	
Stepper driver	IN2	**GPIO32**	
Stepper driver	IN3	**GPIO5**	
Stepper driver	IN4	**GPIO23**	
Stepper driver	+ (5V)	**L298N +5V**	The motor's 5V comes from the L298N regulator, not the ESP32
Stepper driver	− (GND)	**L298N GND and ESP32 GND**	The common ground is required
Stepper driver	Motor socket	**28BYJ-48 white connector**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Jog anticlockwise	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

Button	Jog clockwise	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Save / recall	**GPIO27**	The other leg goes to GND (the sketch uses the internal pull-up)
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	The 5V jumper stays on
L298N	GND	**Battery (−)**	

> The stepper's 5V comes from the L298N's on-board regulator, fed by the 2 x 18650 pack. The ESP32 only sends the four step signals.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Stepper.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <EEPROM.h>
#define STEPS_PER_REV 2048
#define SLOTS 3
#define EEPROM_BASE 200
#define MAGIC 77
// Note the pin order for a 28BYJ-48 on a ULN2003 board
Stepper motor(STEPS_PER_REV, 33, 5, 32, 23);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int leftButton = 12;
int rightButton = 14;
int memoryButton = 27;
long position = 0;
long saved[SLOTS] = {0, 0, 0};
int slot = 0;
void saveToEEPROM()
{
  EEPROM.update(EEPROM_BASE, MAGIC);
  for (int i = 0; i < SLOTS; i++)
  {
    // A long needs four bytes
    EEPROM.put(EEPROM_BASE + 1 + i * 4, saved[i]);
  }
  Serial.println("Positions written to EEPROM");
}
void loadFromEEPROM()
{
  if (EEPROM.read(EEPROM_BASE) != MAGIC)
  {
    return;
  }
  for (int i = 0; i < SLOTS; i++)
  {
    EEPROM.get(EEPROM_BASE + 1 + i * 4, saved[i]);
  }
  Serial.println("Positions loaded from EEPROM");
}
void moveTo(long target)
{
  long difference = target - position;
  Serial.print("Moving ");
  Serial.print(difference);
  Serial.println(" steps");
  motor.step(difference);
  position = target;
}
float degrees_()
{
  return position * 360.0 / STEPS_PER_REV;
}
void setup()
{
  pinMode(leftButton, INPUT_PULLUP);
  pinMode(rightButton, INPUT_PULLUP);
  pinMode(memoryButton, INPUT_PULLUP);
  motor.setSpeed(12); // RPM
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  loadFromEEPROM();
}
void loop()
{
  if (digitalRead(leftButton) == LOW)
  {
    motor.step(-16);
    position = position - 16;
  }
  if (digitalRead(rightButton) == LOW)
  {
```

```
    motor.step(16);
    position = position + 16;
  }
  if (digitalRead(memoryButton) == LOW)
  {
    unsigned long pressStart = millis();
    while (digitalRead(memoryButton) == LOW)
    {
      delay(10);
    }
    if (millis() - pressStart > 800)
    {
      // Long press saves the current position
      saved[slot] = position;
      saveToEEPROM();
    }
    else
    {
      // Short press recalls, then advances the slot
      moveTo(saved[slot]);
      slot = (slot + 1) % SLOTS;
    }
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("POSITION MEMORY");
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print(degrees_(), 1);
  display.setTextSize(1);
  display.print((char)247);
  display.setCursor(78, 18);
  display.print(position);
  display.print(" st");
  display.drawLine(0, 32, 127, 32, SSD1306_WHITE);
  for (int i = 0; i < SLOTS; i++)
  {
    display.setCursor(0, 36 + i * 9);
    display.print(i == slot ? ">" : " ");
    display.print(" Slot ");
    display.print(i + 1);
    display.print(": ");
    display.print(saved[i] * 360.0 / STEPS_PER_REV, 1);
    display.print((char)247);
  }
  display.display();
  delay(20);
}
```

What you should see: Hold the first button or the second button and the motor jogs, with the angle counting up or down. Hold the third button for a second to save the current position into the highlighted slot. Tap the third button to drive straight back to the saved position. Power off and on: the saved positions are still there.

Stepper Motor Angle Preset Selector

Goal: Jump straight to preset angles and compare that with relative moves, so the difference between absolute and relative positioning is obvious.

Absolute positioning means go to 180 degrees, wherever you are now; relative means turn 180 degrees from here. Press Go four times on a 90 degree preset and the two modes end up somewhere completely different, absolute stays put after the first move, relative keeps going round. Seeing that side by side is the point of the experiment, because the same distinction governs CNC machines, plotters and robot arms.

Online page with troubleshooting: geekobot.in/projects/titan-kit-104-stepper-motor-angle-preset-selector

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Tactile Push Button Switch
- 1 x 0.96" OLED Display 128x64
- 1 x 28BYJ-48 Stepper Motor & ULN2003 Driver Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Absolute versus relative positioning
- Choosing the shorter route to a target angle
- Wrapping an angle into a 0–360 range
- Converting between degrees and motor steps

Wiring

Part	Pin	ESP32	Note
Stepper driver	IN1	**GPIO33**	
Stepper driver	IN2	**GPIO32**	
Stepper driver	IN3	**GPIO5**	
Stepper driver	IN4	**GPIO23**	
Stepper driver	+ (5V)	**L298N +5V**	The motor's 5V comes from the L298N regulator, not the ESP32
Stepper driver	− (GND)	**L298N GND and ESP32 GND**	The common ground is required
Stepper driver	Motor socket	**28BYJ-48 white connector**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
Button	Select preset	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)

Button	Go	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Absolute / relative	**GPIO27**	The other leg goes to GND (the sketch uses the internal pull-up)
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	The 5V jumper stays on
L298N	GND	**Battery (−)**	

> The stepper's 5V comes from the L298N's on-board regulator, fed by the 2 x 18650 pack. The ESP32 only sends the four step signals.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Stepper.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define STEPS_PER_REV 2048
#define PRESETS 5
Stepper motor(STEPS_PER_REV, 33, 5, 32, 23);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int selectButton = 12;
int goButton = 14;
int modeButton = 27;
int presetAngle[PRESETS] = {0, 45, 90, 180, 270};
int preset = 0;
long position = 0; // In steps
bool absoluteMode = true;
int moves = 0;
long stepsFor(int degrees)
{
  return (long)degrees * STEPS_PER_REV / 360;
}
float currentDegrees()
{
  float d = position * 360.0 / STEPS_PER_REV;
  while (d >= 360) d = d - 360;
  while (d < 0) d = d + 360;
  return d;
}
void go()
{
  long target;
  long movement;
  if (absoluteMode)
  {
    // Nearest equivalent of the target angle
    long fullTurns = position / STEPS_PER_REV;
    target = fullTurns * STEPS_PER_REV + stepsFor(presetAngle[preset]);
    movement = target - position;
    // Take the shorter way round
    if (movement > STEPS_PER_REV / 2) movement = movement - STEPS_PER_REV;
    if (movement < -STEPS_PER_REV / 2) movement = movement + STEPS_PER_REV;
  }
  else
  {
    movement = stepsFor(presetAngle[preset]);
  }
  Serial.print(absoluteMode ? "Absolute" : "Relative");
  Serial.print(" move of ");
  Serial.print(movement);
  Serial.println(" steps");
  motor.step(movement);
  position = position + movement;
  moves = moves + 1;
}
void setup()
{
  pinMode(selectButton, INPUT_PULLUP);
  pinMode(goButton, INPUT_PULLUP);
  pinMode(modeButton, INPUT_PULLUP);
  motor.setSpeed(12);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
}
void loop()
{
  if (digitalRead(selectButton) == LOW)
  {
    preset = (preset + 1) % PRESETS;
    delay(300);
  }
  if (digitalRead(modeButton) == LOW)
  {
    absoluteMode = !absoluteMode;
```

```
    Serial.println(absoluteMode ? "ABSOLUTE mode" : "RELATIVE mode");
    delay(300);
  }
  if (digitalRead(goButton) == LOW)
  {
    go();
    delay(300);
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("PRESET SELECTOR");
  display.setCursor(74, 0);
  display.print(absoluteMode ? "ABSOLUTE" : "RELATIVE");
  display.setTextSize(2);
  display.setCursor(0, 12);
  display.print(currentDegrees(), 0);
  display.setTextSize(1);
  display.print((char)247);
  display.print(" now");
  display.setCursor(0, 32);
  display.print("Preset: ");
  display.print(presetAngle[preset]);
  display.print((char)247);
  display.print(absoluteMode ? " (go to)" : " (turn by)");
  // Preset row
  for (int i = 0; i < PRESETS; i++)
  {
    int x = i * 25 + 2;
    display.drawRect(x, 42, 22, 11, SSD1306_WHITE);
    if (i == preset)
    {
      display.fillRect(x, 42, 22, 11, SSD1306_WHITE);
      display.setTextColor(SSD1306_BLACK);
    }
    display.setCursor(x + 3, 44);
    display.print(presetAngle[i]);
    display.setTextColor(SSD1306_WHITE);
  }
  display.setCursor(0, 56);
  display.print("Moves ");
  display.print(moves);
  display.print(" D4 = go");
  display.display();
  delay(20);
}
```

What you should see: Select the 90° preset in absolute mode and press Go: the shaft moves to 90°. Press Go again and nothing happens, it is already there. Switch to relative mode and press Go repeatedly: it advances 90° each time. The current angle wraps neatly from 359° back to 0°.

Automated Conveyor Simulation

Goal: Run the stepper as a factory conveyor: continuous motion, indexed batches, jams, and a live count of items processed.

Real conveyors run in two ways. Continuous mode never stops, which suits bulk material; indexed mode advances a fixed distance, pauses for work to happen, and advances again, which suits assembly. Both are built here from the same primitive, a small batch of steps issued each pass through loop() rather than one long blocking move, so the sensor and buttons stay live. That is the same non-blocking discipline used throughout this manual, applied to a motor.

Online page with troubleshooting: geekobot.in/projects/titan-kit-105-automated-conveyor-simulation

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x 0.96" OLED Display 128x64
- 1 x Generic Infrared Obstacle Avoidance Sensor Module (Active Low) – LM393 Comparator, 2-30cm Range
- 1 x 28BYJ-48 Stepper Motor & ULN2003 Driver Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Continuous motion loops
- Batching steps to keep the loop responsive
- Indexed advance-and-pause cycles
- Detecting a fault from the absence of expected events
- Combine levels, give the RFID access log a Bluetooth download, or let the gesture menu drive the stepper conveyor.
- Add a DS3231 real-time clock so every log entry carries a real date and time instead of seconds since power-up.
- Replace the simulated compass in Experiment 7 with a real magnetometer on the same I2C bus.
- Move the RFID and configuration data into EEPROM so cards can be enrolled without re-uploading the sketch.
- Try a proper FFT library on a faster board and compare it with the zero-crossing estimate from Experiment 25.
- Close the loop, add an end-stop switch to the conveyor so the stepper can home itself instead of trusting its count.
- Design your own project from scratch: write down the inputs, the outputs, and the decision in between, then wire and code it. Conclusion If you have worked through these experiments, you have travelled a very long way. You began by making one LED blink on a 3.3V dual-core microcontroller and ended by driving a conveyor line that counts its own output and notices when it jams. Along the way you met every core idea in physical computing, digital and analog signals, PWM, bitmapping, thresholds and state machines, and then the ideas that separate a working prototype from a demonstration: calibration, filtering, hysteresis, non-volatile storage, and protocols designed for a machine at the other end rather than a human. More importantly, you have seen the same techniques reappear in unrelated places. The rolling average that smoothed a temperature graph is the one that stopped a noise alarm firing at a door slam. The two-threshold hysteresis that

steadied a hover switch is the one that stopped a gesture menu flickering. The parallel arrays behind a library catalogue are the ones behind an attendance register. Recognising a familiar technique wearing unfamiliar clothes is most of what engineering experience actually is.

Wiring

Part	Pin	ESP32	Note
Stepper driver	IN1	**GPIO33**	
Stepper driver	IN2	**GPIO32**	
Stepper driver	IN3	**GPIO5**	
Stepper driver	IN4	**GPIO23**	
Stepper driver	+ (5V)	**L298N +5V**	The motor's 5V comes from the L298N regulator, not the ESP32
Stepper driver	− (GND)	**L298N GND and ESP32 GND**	The common ground is required
Stepper driver	Motor socket	**28BYJ-48 white connector**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	
IR sensor	IR sensor OUT	**GPIO13**	
IR sensor	IR sensor VCC / GND	**3V3 / GND**	
Button	Start / stop button	**GPIO12**	The other leg goes to GND (the sketch uses the internal pull-up)
Button	Mode button	**GPIO14**	The other leg goes to GND (the sketch uses the internal pull-up)
Passive buzzer	Buzzer (+)	**GPIO27**	
Passive buzzer	Buzzer (−)	**GND**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	The 5V jumper stays on
L298N	GND	**Battery (−)**	

> The stepper's 5V comes from the L298N's on-board regulator, fed by the 2 x 18650 pack. The ESP32 only sends the four step signals.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Stepper.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define STEPS_PER_REV 2048
Stepper belt(STEPS_PER_REV, 33, 5, 32, 23);
Adafruit_SSD1306 display(128, 64, &Wire, -1);
int sensorPin = 13;
int runButton = 12;
int modeButton = 14;
int buzzer = 27;
bool running = false;
bool indexed = false; // false = continuous, true = indexed
int batchSteps = 32; // Steps issued per pass - keeps loop responsive
long indexLength = 512; // Steps between pauses in indexed mode
long stepsThisIndex = 0;
unsigned long pauseUntil = 0;
unsigned long pauseLength = 1500;
long totalSteps = 0;
int itemsCounted = 0;
int speedRPM = 10;
bool lastSensor = HIGH;
unsigned long lastItem = 0;
unsigned long lastMovement = 0;
bool jammed = false;
void setup()
{
  pinMode(sensorPin, INPUT);
  pinMode(runButton, INPUT_PULLUP);
  pinMode(modeButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
  belt.setSpeed(speedRPM);
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  lastMovement = millis();
}
void loop()
{
  if (digitalRead(runButton) == LOW)
  {
    running = !running;
    jammed = false;
    lastMovement = millis();
    tone(buzzer, running ? 1400 : 700, 150);
    Serial.println(running ? "Conveyor started" : "Conveyor stopped");
    delay(300);
  }
  if (digitalRead(modeButton) == LOW)
  {
    indexed = !indexed;
    stepsThisIndex = 0;
    pauseUntil = 0;
    Serial.println(indexed ? "Indexed mode" : "Continuous mode");
    delay(300);
  }
  // Count items passing the sensor
  bool sensor = digitalRead(sensorPin);
  if (sensor == LOW && lastSensor == HIGH && millis() - lastItem > 250)
  {
    itemsCounted = itemsCounted + 1;
    lastItem = millis();
    tone(buzzer, 1800, 50);
    Serial.print("Item ");
    Serial.println(itemsCounted);
  }
  lastSensor = sensor;
  // Move the belt in small batches so nothing blocks
  if (running && !jammed && millis() > pauseUntil)
  {
    belt.step(batchSteps);
    totalSteps = totalSteps + batchSteps;
    lastMovement = millis();
```

```
    if (indexed)
    {
      stepsThisIndex = stepsThisIndex + batchSteps;
      if (stepsThisIndex >= indexLength)
      {
        stepsThisIndex = 0;
        pauseUntil = millis() + pauseLength;
        tone(buzzer, 1000, 100);
      }
    }
  }
  // A jam is the belt running with nothing reaching the sensor
  if (running && !indexed && millis() - lastItem > 20000 && itemsCounted > 0)
  {
    if (!jammed)
    {
      jammed = true;
      tone(buzzer, 400, 800);
      Serial.println("JAM - no items detected");
    }
  }
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("CONVEYOR LINE");
  display.setCursor(74, 0);
  display.print(indexed ? "INDEXED" : "CONTINUOUS");
  display.setTextSize(2);
  display.setCursor(0, 12);
  if (jammed) display.print("JAMMED");
  else if (!running) display.print("STOPPED");
  else if (millis() < pauseUntil) display.print("PAUSED");
  else display.print("RUNNING");
  display.setTextSize(1);
  display.setCursor(0, 32);
  display.print("Items: ");
  display.print(itemsCounted);
  display.setCursor(64, 32);
  display.print("Turns: ");
  display.print(totalSteps / STEPS_PER_REV);
  // Belt animation
  display.drawRect(0, 42, 128, 12, SSD1306_WHITE);
  if (running && millis() > pauseUntil && !jammed)
  {
    int offset = (totalSteps / 8) % 16;
    for (int x = offset; x < 126; x = x + 16)
    {
      display.fillRect(x + 1, 44, 6, 8, SSD1306_WHITE);
    }
  }
  display.setCursor(0, 56);
  if (indexed && millis() < pauseUntil)
  {
    display.print("Work pause ");
    display.print((pauseUntil - millis()) / 100 / 10.0, 1);
    display.print("s");
  }
  else
  {
    display.print("D3 run D4 mode");
  }
  display.display();
}
```

What you should see: Press the first button and the belt turns continuously with a moving pattern on screen. Break the IR beam and the item counter increases with a click. Press the second button for indexed mode: the belt advances, pauses 1.5 seconds, then advances again. Leave it running with no items for 20 seconds and it reports JAMMED.

Wi-Fi Light Switch

Goal: Switch three LEDs on and off from a phone browser, with the ESP32 running its own Wi-Fi network and web page.

The ESP32 starts a Wi-Fi access point, so the phone joins the board directly and no router is needed. A small web server answers the phone: the home page lists the three lights, and tapping one opens /toggle?led=1, which flips that LED and sends the phone back to the updated page.

Online page with troubleshooting: geekobot.in/projects/titan-kit-106-wifi-light-switch

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 3 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Starting a Wi-Fi access point with WiFi.softAP()
- Serving a web page with the WebServer library
- Reading a value from the address with server.arg()
- Building HTML in a String

Wiring

Part	Pin	ESP32	Note
LED	Red LED anode	**GPIO27**	Through a 220 ohm resistor
LED	Yellow LED anode	**GPIO26**	Through a 220 ohm resistor
LED	Green LED anode	**GPIO25**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> The page has no login beyond the Wi-Fi password: change the password in the sketch before you use it in a shared space.

Code

```
#include <WiFi.h>
#include <WebServer.h>

const char* ssid = "Titan-Lights";      // the network the ESP32 creates
const char* password = "titan1234";     // at least 8 characters
int leds[3] = {27, 26, 25};
bool state[3] = {false, false, false};
const char* names[3] = {"Red", "Yellow", "Green"};
WebServer server(80);

String page()
{
  String h = "<!DOCTYPE html><html><head><meta name='viewport' content='width=device-width, initial-
scale=1'>";
  h += "<title>Titan Lights</title><style>body{font-family:sans-serif;text-align:center}";
  h += "a{display:block;margin:12px auto;padding:18px;width:70%;border-radius:12px;color:#fff;text-
decoration:none;font-size:22px}</style>";
  h += "</head><body><h2>Titan Kit Lights</h2>";
  for (int i = 0; i < 3; i++)
  {
    h += "<a style='background:" + String(state[i] ? "#2e7d32" : "#757575") + "' href='/toggle?led=" +
String(i) + "'>";
    h += String(names[i]) + (state[i] ? " is ON" : " is OFF") + "</a>";
  }
  h += "</body></html>";
  return h;
}

void handleRoot()
{
  server.send(200, "text/html", page());
}

void handleToggle()
{
  int i = server.arg("led").toInt();
  if (i >= 0 && i < 3)
  {
    state[i] = !state[i];
    digitalWrite(leds[i], state[i] ? HIGH : LOW);
    Serial.printf("%s LED %s\n", names[i], state[i] ? "ON" : "OFF");
  }
  server.sendHeader("Location", "/");
  server.send(303);
}

void setup()
{
  Serial.begin(9600);
  for (int i = 0; i < 3; i++)
  {
    pinMode(leds[i], OUTPUT);
  }
  WiFi.softAP(ssid, password);
  Serial.print("Join the Wi-Fi network ");
  Serial.print(ssid);
  Serial.print(" and open http://");
  Serial.println(WiFi.softAPIP());
  server.on("/", handleRoot);
  server.on("/toggle", handleToggle);
  server.begin();
}

void loop()
{
  server.handleClient();
}
```

What you should see: The Serial Monitor prints the network name and the address 192.168.4.1. Join the Titan-Lights Wi-Fi on a phone (password titan1234) and open that address. Each tap on a button switches that LED and the button turns green or grey.

Wi-Fi Weather Station

Goal: Read temperature, humidity and air pressure and show them on a web page that any phone or laptop on your Wi-Fi can open.

The DHT11 measures temperature and humidity and the BMP280 measures air pressure. The ESP32 joins your home or school Wi-Fi and runs a web server: the home page shows the latest readings and refreshes itself every five seconds, and /data returns the same numbers as JSON for a spreadsheet or app.

Online page with troubleshooting: geekobot.in/projects/titan-kit-107-wifi-weather-station

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x GY-BMP280-3.3 Precision Altimeter Atmospheric Pressure Sensor Module
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Joining a Wi-Fi network with WiFi.begin()
- Reading two sensors and keeping the last good value
- Serving HTML and JSON from the same sketch
- Converting pascals to hectopascals

Wiring

Part	Pin	ESP32	Note
DHT11	VCC	**3V3**	
DHT11	GND	**GND**	
DHT11	DATA	**GPIO4**	
BMP280	VCC	**3V3**	
BMP280	GND	**GND**	
BMP280	SDA	**GPIO21**	
BMP280	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include <Adafruit_BMP280.h>
#include <DHT.h>

const char* ssid = "YOUR_WIFI_NAME";        // your 2.4 GHz Wi-Fi
const char* password = "YOUR_WIFI_PASSWORD";
DHT dht(4, DHT11);
Adafruit_BMP280 bmp;
WebServer server(80);
float temp = 0, hum = 0, pres = 0;
unsigned long lastRead = 0;

void readSensors()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  if (!isnan(t)) temp = t;   // keep the last good reading if one read fails
  if (!isnan(h)) hum = h;
  pres = bmp.readPressure() / 100.0; // pascals to hPa
}

void handleRoot()
{
  String h = "<!DOCTYPE html><html><head><meta name='viewport' content='width=device-width, initial-
scale=1'>";
  h += "<meta http-equiv='refresh' content='5'><title>Titan Weather</title>";
  h += "<style>body{font-family:sans-serif;text-align:center}div{font-size:28px;margin:14px}</style>
</head><body>";
  h += "<h2>Titan Weather Station</h2>";
  h += "<div>Temperature: " + String(temp, 1) + " &deg;C</div>";
  h += "<div>Humidity: " + String(hum, 0) + " %</div>";
  h += "<div>Pressure: " + String(pres, 1) + " hPa</div>";
  h += "<p>Refreshes every 5 seconds</p></body></html>";
  server.send(200, "text/html", h);
}

void handleData()
{
  String j = "{\"temperature\":" + String(temp, 1) + ",\"humidity\":" + String(hum, 0) + ",\"pressure\":"
+ String(pres, 1) + "}";
  server.send(200, "application/json", j);
}

void setup()
{
  Serial.begin(9600);
  dht.begin();
  if (!bmp.begin(0x76))
  {
    Serial.println("BMP280 not found: check SDA and SCL, or try address 0x77");
  }
  WiFi.begin(ssid, password);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Open http://");
  Serial.println(WiFi.localIP());
  server.on("/", handleRoot);
  server.on("/data", handleData);
  server.begin();
  readSensors();
}

void loop()
{
  server.handleClient();
  if (millis() - lastRead > 2000) // the DHT11 needs 2 seconds between reads
  {
```

```
    lastRead = millis();
    readSensors();
  }
}
```

What you should see: The Serial Monitor prints the board's address on your network, for example http://192.168.1.42. Open it on a phone on the same Wi-Fi: temperature, humidity and pressure appear. The numbers update by themselves every five seconds.

Wi-Fi Colour Lamp

Goal: Pick any colour on a phone and the WS2812 LED ring lights up in it, with a brightness slider.

Each LED on the WS2812 ring has its own tiny controller, so one data wire sets all eight colours. The ESP32 runs its own Wi-Fi network and a page with the phone's colour picker. The picker sends a colour such as #ff8800; the sketch splits it into red, green and blue numbers and sends them to the ring.

Online page with troubleshooting: geekobot.in/projects/titan-kit-108-wifi-colour-lamp

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x WS2812 RGB LED Ring
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Driving addressable LEDs with the Adafruit NeoPixel library
- Turning a hex colour into red, green and blue values
- Using HTML form inputs (colour and range)
- Capping brightness to stay within the USB power budget

Wiring

Part	Pin	ESP32	Note
LED ring	5V / VCC	**5V**	The ring's LEDs need 5V; brightness is capped in the sketch
LED ring	GND	**GND**	
LED ring	DIN (data in)	**GPIO18**	Use the DIN pad, not DOUT

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> The page has no login beyond the Wi-Fi password: change the password in the sketch before you use it in a shared space.

Code

```
#include <WiFi.h>
#include <WebServer.h>
#include <Adafruit_NeoPixel.h>

#define RING_PIN 18
#define LEDS 8
Adafruit_NeoPixel ring(LEDS, RING_PIN, NEO_GRB + NEO_KHZ800);
WebServer server(80);
const char* ssid = "Titan-Ring";
const char* password = "titan1234";
int r = 0, g = 80, b = 255;
int bright = 60; // 5-150: keeps the ring's current low enough for USB power

void showColour()
{
  ring.setBrightness(bright);
  for (int i = 0; i < LEDS; i++)
  {
    ring.setPixelColor(i, ring.Color(r, g, b));
  }
  ring.show();
}

void handleRoot()
{
  char hex[8];
  snprintf(hex, sizeof(hex), "#%02x%02x%02x", r, g, b);
  String h = "<!DOCTYPE html><html><head><meta name='viewport' content='width=device-width, initial-
scale=1'><title>Titan Ring</title>";
  h += "<style>body{font-family:sans-serif;text-align:center}input{width:80%;height:60px;margin:10px}
</style></head><body>";
  h += "<h2>LED Ring Colour</h2><form action='/set'>";
  h += "<input type='color' name='c' value='" + String(hex) + "'><br>Brightness<br>";
  h += "<input type='range' name='b' min='5' max='150' value='" + String(bright) + "'><br>";
  h += "<input type='submit' value='Light it'></form></body></html>";
  server.send(200, "text/html", h);
}

void handleSet()
{
  String c = server.arg("c"); // for example #ff8800
  if (c.length() == 7)
  {
    long v = strtol(c.substring(1).c_str(), NULL, 16);
    r = (v >> 16) & 0xFF;
    g = (v >> 8) & 0xFF;
    b = v & 0xFF;
  }
  if (server.hasArg("b"))
  {
    bright = constrain(server.arg("b").toInt(), 5, 150);
  }
  showColour();
  Serial.printf("Colour %s, brightness %d\n", c.c_str(), bright);
  server.sendHeader("Location", "/");
  server.send(303);
}

void setup()
{
  Serial.begin(9600);
  ring.begin();
  showColour();
  WiFi.softAP(ssid, password);
  Serial.print("Join ");
  Serial.print(ssid);
  Serial.print(" and open http://");
  Serial.println(WiFi.softAPIP());
  server.on("/", handleRoot);
  server.on("/set", handleSet);
  server.begin();
}

void loop()
```

```
{
  server.handleClient();
}
```

What you should see: The ring lights blue at start-up. Join the Titan-Ring Wi-Fi (password titan1234) and open 192.168.4.1. Choose a colour, press Light it and the whole ring changes; the slider dims or brightens it.

Internet Clock on OLED

Goal: Build a clock that sets itself: the ESP32 asks an internet time server for the exact time and shows it on the OLED.

After joining your Wi-Fi, configTime() asks a time server (NTP) for the time and keeps the ESP32's own clock in step. The offset of 19800 seconds turns world time (UTC) into Indian time (UTC + 5:30). getLocalTime() then fills a date and time structure, and strftime() turns it into text for the screen.

Online page with troubleshooting: geekobot.in/projects/titan-kit-109-internet-clock-oled

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- How NTP sets a clock over the internet
- Time zones as an offset in seconds
- Formatting time and date with strftime()
- Big and small text on an OLED

Wiring

Part	Pin	ESP32	Note
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <WiFi.h>
#include <time.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(128, 64, &Wire, -1);
const char* ssid = "YOUR_WIFI_NAME";
const char* password = "YOUR_WIFI_PASSWORD";
const long utcOffset = 19800; // India: UTC + 5 h 30 min = 19800 seconds
const char* days[7] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
int lastMinute = -1;

void setup()
{
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println("Connecting to Wi-Fi");
  display.display();
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  configTime(utcOffset, 0, "pool.ntp.org", "time.google.com");
  Serial.println(" connected, asking the time server");
}

void loop()
{
  struct tm now;
  if (!getLocalTime(&now))
  {
    Serial.println("Waiting for the time server");
    delay(1000);
    return;
  }
  char hhmm[6], secs[3], date[16];
  strftime(hhmm, sizeof(hhmm), "%H:%M", &now);
  strftime(secs, sizeof(secs), "%S", &now);
  strftime(date, sizeof(date), "%d %b %Y", &now);
  display.clearDisplay();
  display.setTextSize(3);
  display.setCursor(4, 6);
  display.print(hhmm);
  display.setTextSize(1);
  display.setCursor(104, 20);
  display.print(secs);
  display.drawRect(4, 34, 120, 5, SSD1306_WHITE);
  display.fillRect(4, 34, now.tm_sec * 2, 5, SSD1306_WHITE); // fills across each minute
  display.setCursor(4, 44);
  display.print(date);
  display.setCursor(4, 55);
  display.print(days[now.tm_wday]);
  display.display();
  if (now.tm_min != lastMinute)
  {
    lastMinute = now.tm_min;
    Serial.printf("%s %s\n", date, hhmm);
  }
  delay(200);
}
```

What you should see: The OLED shows Connecting to Wi-Fi, then the time in large digits. Seconds tick in the corner and a bar fills across each minute. The date and the day of the week show underneath.

IR Remote Light Ring

Goal: Use the kit's TV-style remote to change the LED ring's colour, brightness and effects.

The remote flashes an invisible infrared code for each key. The IR receiver turns the flashes back into pulses, and the IRremote library decodes them into a key number. The sketch looks that number up: keys 1 to 6 pick a colour, up and down change brightness, 0 starts a spinning rainbow and OK switches the ring off.

Online page with troubleshooting: geekobot.in/projects/titan-kit-110-ir-remote-light-ring

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x IR Receiver Module (VS1838B / TSOP)
- 1 x WS2812 RGB LED Ring
- 1 x ESP32 38-Pin Expansion Board

What you learn

- How an infrared remote sends codes
- Decoding keys with the IRremote library
- Choosing an action with switch and case
- Hue-based rainbow colours

Wiring

Part	Pin	ESP32	Note
IR receiver	VCC	**3V3**	
IR receiver	GND	**GND**	
IR receiver	OUT / S	**GPIO35**	
LED ring	5V / VCC	**5V**	The ring's LEDs need 5V; brightness is capped in the sketch
LED ring	GND	**GND**	
LED ring	DIN (data in)	**GPIO18**	Use the DIN pad, not DOUT

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <IRremote.hpp>
#include <Adafruit_NeoPixel.h>

#define IR_PIN 35
#define RING_PIN 18
#define LEDS 8
Adafruit_NeoPixel ring(LEDS, RING_PIN, NEO_GRB + NEO_KHZ800);
// key codes of the kit's 17-key remote (NEC): if yours differ, read them in the Serial Monitor
const uint8_t K1 = 0x45, K2 = 0x46, K3 = 0x47, K4 = 0x44, K5 = 0x40, K6 = 0x43, K0 = 0x19;
const uint8_t KUP = 0x18, KDOWN = 0x52, KOK = 0x1C;
uint32_t colour = 0;
int bright = 50;
bool rainbow = false;
uint16_t hue = 0;

void fillRing(uint32_t c)
{
  for (int i = 0; i < LEDS; i++)
  {
    ring.setPixelColor(i, c);
  }
  ring.setBrightness(bright);
  ring.show();
}

void setup()
{
  Serial.begin(9600);
  IrReceiver.begin(IR_PIN);
  ring.begin();
  colour = ring.Color(0, 0, 255);
  fillRing(colour);
  Serial.println("Point the remote at the receiver and press a key");
}

void loop()
{
  if (IrReceiver.decode())
  {
    uint8_t key = IrReceiver.decodedIRData.command;
    bool repeat = IrReceiver.decodedIRData.flags & IRDATA_FLAGS_IS_REPEAT;
    IrReceiver.resume();
    if (!repeat)
    {
      Serial.print("Key code 0x");
      Serial.println(key, HEX);
    }
    switch (key)
    {
      case K1: colour = ring.Color(255, 0, 0); rainbow = false; break;
      case K2: colour = ring.Color(255, 90, 0); rainbow = false; break;
      case K3: colour = ring.Color(255, 200, 0); rainbow = false; break;
      case K4: colour = ring.Color(0, 255, 0); rainbow = false; break;
      case K5: colour = ring.Color(0, 0, 255); rainbow = false; break;
      case K6: colour = ring.Color(160, 0, 255); rainbow = false; break;
      case K0: rainbow = true; break;
      case KUP: bright = min(bright + 10, 150); break;
      case KDOWN: bright = max(bright - 10, 5); break;
      case KOK: colour = 0; rainbow = false; break;
      default: break;
    }
    if (!rainbow)
    {
      fillRing(colour);
    }
  }
  if (rainbow)
  {
    for (int i = 0; i < LEDS; i++)
    {
      ring.setPixelColor(i, ring.gamma32(ring.ColorHSV(hue + i * 65536L / LEDS)));
    }
    ring.setBrightness(bright);
    ring.show();
```

```
    hue += 256;
    delay(20);
  }
}
```

What you should see: Each key press prints its code in the Serial Monitor. Keys 1 to 6 colour the ring red, orange, yellow, green, blue and purple. Up and down change brightness, 0 spins a rainbow, OK switches off.

Wi-Fi Plant Monitor

Goal: Check how thirsty a plant is from your phone: the soil moisture shows on the OLED and on a web page.

The soil sensor gives a lower or higher voltage as the soil gets wetter or drier. The ESP32 reads it on GPIO34, an ADC1 pin that still works while Wi-Fi is on, turns it into a percentage with your two calibration numbers, and shows it on the OLED and on a page served from its own Wi-Fi network.

Online page with troubleshooting: geekobot.in/projects/titan-kit-111-wifi-plant-monitor

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Capacitive Soil Moisture Sensor – 3.3V/5V Analog Output, PH2.54-3P Interface
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- Calibrating an analog sensor with two known points
- Why ADC1 pins are used when Wi-Fi is on
- Showing the same data on a screen and a web page
- Status thresholds (dry, OK, wet)

Wiring

Part	Pin	ESP32	Note
Soil sensor	VCC	**3V3**	
Soil sensor	GND	**GND**	
Soil sensor	AO	**GPIO34**	
OLED	VCC	**3V3**	
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Push only the sensor's prongs into the soil and keep water away from the ESP32 and the OLED.

> The page has no login beyond the Wi-Fi password: change the password in the sketch before you use it in a shared space.

Code

```
#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(128, 64, &Wire, -1);
WebServer server(80);
const char* ssid = "Titan-Plant";
const char* password = "titan1234";
int soilPin = 34;      // ADC1: still readable while Wi-Fi is on
int dryValue = 3000;   // reading in dry air: note yours from the Serial Monitor
int wetValue = 1300;   // reading in a glass of water
int percent = 0;

String status()
{
  if (percent < 30) return "Dry - water me";
  if (percent > 80) return "Wet";
  return "OK";
}

void handleRoot()
{
  String h = "<!DOCTYPE html><html><head><meta name='viewport' content='width=device-width, initial-
scale=1'>";
  h += "<meta http-equiv='refresh' content='5'><title>Titan Plant</title>";
  h += "<style>body{font-family:sans-serif;text-align:center}b{font-size:64px}</style></head><body><h2>My
Plant</h2>";
  h += "<b>" + String(percent) + "%</b><p>" + status() + "</p></body></html>";
  server.send(200, "text/html", h);
}

void setup()
{
  Serial.begin(9600);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  WiFi.softAP(ssid, password);
  server.on("/", handleRoot);
  server.begin();
  Serial.print("Open http://");
  Serial.println(WiFi.softAPIP());
}

void loop()
{
  server.handleClient();
  static unsigned long last = 0;
  if (millis() - last < 1000)
  {
    return;
  }
  last = millis();
  int raw = analogRead(soilPin); // 0-4095
  percent = constrain(map(raw, dryValue, wetValue, 0, 100), 0, 100);
  Serial.printf("Raw %d, moisture %d%%\n", raw, percent);
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("Soil moisture");
  display.setTextSize(3);
  display.setCursor(0, 16);
  display.print(percent);
  display.print("%");
  display.setTextSize(1);
  display.setCursor(0, 46);
  display.print(status());
  display.setCursor(0, 56);
  display.print(WiFi.softAPIP().toString());
  display.display();
}
```

What you should see: The OLED shows the moisture percentage and Dry, OK or Wet. Join Titan-Plant (password titan1234) and open 192.168.4.1 to see the same reading. Water the pot and the number rises within a few seconds.

Barometer and Altitude Meter

Goal: Measure air pressure and height above sea level with the BMP280, and watch the pressure trend that hints at coming weather.

Air pressure falls by about 12 hPa for every 100 m you climb, so the BMP280's pressure reading also gives an altitude. readAltitude() works it out from the standard sea-level pressure of 1013.25 hPa. The sketch keeps the reading from ten minutes ago, and a falling pressure is shown as a down arrow: often a sign of rain.

Online page with troubleshooting: geekobot.in/projects/titan-kit-112-barometer-altitude-meter

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x GY-BMP280-3.3 Precision Altimeter Atmospheric Pressure Sensor Module
- 1 x 0.96" OLED Display 128x64
- 1 x ESP32 38-Pin Expansion Board

What you learn

- How pressure changes with height
- Reading a BMP280 over I2C
- Comparing a reading with an older one to find a trend
- Drawing arrows with triangles on the OLED

Wiring

Part	Pin	ESP32	Note
BMP280	VCC	**3V3**	
BMP280	GND	**GND**	
BMP280	SDA	**GPIO21**	
BMP280	SCL	**GPIO22**	
OLED	VCC	**3V3**	The OLED shares the same four I2C wires
OLED	GND	**GND**	
OLED	SDA	**GPIO21**	
OLED	SCL	**GPIO22**	

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Code

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(128, 64, &Wire, -1);
Adafruit_BMP280 bmp;
float seaLevel = 1013.25;   // hPa: set today's sea-level pressure for a more exact altitude
float oldPressure = 0;
unsigned long lastTrend = 0;
int trend = 0;              // -1 falling, 0 steady, 1 rising

void setup()
{
  Serial.begin(9600);
  if (!bmp.begin(0x76))
  {
    Serial.println("BMP280 not found: check the wires, or try 0x77");
    while (true) { }
  }
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { while (true) { } }
  display.setTextColor(SSD1306_WHITE);
  oldPressure = bmp.readPressure() / 100.0;
  lastTrend = millis();
}

void loop()
{
  float hpa = bmp.readPressure() / 100.0;
  float metres = bmp.readAltitude(seaLevel);
  float temp = bmp.readTemperature();
  if (millis() - lastTrend > 600000UL) // every ten minutes
  {
    float change = hpa - oldPressure;
    trend = change < -0.5 ? -1 : (change > 0.5 ? 1 : 0);
    oldPressure = hpa;
    lastTrend = millis();
  }
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0, 0);
  display.print(hpa, 1);
  display.setTextSize(1);
  display.print(" hPa");
  display.setCursor(0, 26);
  display.print("Altitude: ");
  display.print(metres, 0);
  display.print(" m");
  display.setCursor(0, 40);
  display.print("Temp: ");
  display.print(temp, 1);
  display.print(" C");
  display.setCursor(0, 54);
  if (trend < 0)
  {
    display.print("Falling: rain likely");
    display.fillTriangle(116, 40, 126, 40, 121, 50, SSD1306_WHITE);
  }
  else if (trend > 0)
  {
    display.print("Rising: fair weather");
    display.fillTriangle(116, 50, 126, 50, 121, 40, SSD1306_WHITE);
  }
  else
  {
    display.print("Steady");
  }
  display.display();
  Serial.printf("%.1f hPa, %.0f m, %.1f C\n", hpa, metres, temp);
  delay(1000);
}
```

What you should see: The OLED shows pressure in hPa, altitude in metres and temperature. Carry it up two floors of stairs and the altitude rises by about 6 m. After ten minutes an arrow shows whether the pressure is rising, steady or falling.

Robot Car First Drive

Goal: Build the Titan robot car and teach it its first moves: forward, back, turns and a square.

The L298N has two H-bridges, one per wheel. Two direction pins per motor choose forward or back, and a PWM signal on ENA or ENB sets the speed from 0 to 255. drive(left, right) hides all of that: positive numbers drive a wheel forward, negative ones backwards, and opposite signs spin the car on the spot.

Online page with troubleshooting: geekobot.in/projects/titan-kit-113-robot-car-first-drive

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- How an H-bridge reverses a motor
- Setting motor speed with PWM
- Writing one drive() function for all moves
- Using the ESP32's BOOT button as a start button

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
// Robot car first drive: press BOOT on the ESP32 to start the routine
// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

int startButton = 0; // the BOOT button (GPIO0), LOW when pressed
int speed = 180;

void move(const char* name, int left, int right, int ms)
{
  Serial.println(name);
  drive(left, right);
  delay(ms);
  drive(0, 0);
  delay(400);
}

void setup()
{
  Serial.begin(9600);
  motorsBegin();
  pinMode(startButton, INPUT_PULLUP);
  Serial.println("Press BOOT to start");
}

void loop()
{
  if (digitalRead(startButton) == HIGH)
  {
    return;
  }
  delay(1000); // time to take your hand away
  move("Forward", speed, speed, 1000);
  move("Back", -speed, -speed, 1000);
  move("Spin left", -speed, speed, 500);
  move("Spin right", speed, -speed, 500);
  for (int side = 0; side < 4; side++)
  {
    move("Square side", speed, speed, 800);
    move("Quarter turn", speed, -speed, 380); // tune this time until the turn is 90 degrees
  }
  Serial.println("Done: press BOOT to run again");
}
```

What you should see: Nothing moves until you press the BOOT button on the ESP32. The car drives forward, reverses, turns left and right, then drives a square. The Serial Monitor names each move as it happens.

Wi-Fi Robot Car

Goal: Drive the robot car from a phone browser: hold an arrow to move, let go to stop, with a speed slider.

The car runs its own Wi-Fi network and a page with five big buttons. While a button is held, the page sends its command every 200 ms; the car stops by itself if no command arrives for 600 ms. That way the car can never run away if the phone drops the connection or the finger slips off.

Online page with troubleshooting: geekobot.in/projects/titan-kit-114-wifi-robot-car

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Driving motors from a web page
- Sending repeated commands with JavaScript fetch()
- A watchdog timer that stops the car safely
- Mixing speed and direction

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

> The page has no login beyond the Wi-Fi password: change the password in the sketch before you use it in a shared space.

Code

```
#include <WiFi.h>
#include <WebServer.h>

// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

WebServer server(80);
const char* ssid = "Titan-Car";
const char* password = "titan1234";
int speed = 180;
unsigned long lastCommand = 0;

const char PAGE[] = R"HTML(<!DOCTYPE html><html><head><meta name='viewport' content='width=device-width,
initial-scale=1, user-scalable=no'>
<title>Titan Car</title><style>body{font-family:sans-serif;text-align:center;user-select:none}
button{width:28vw;height:28vw;margin:1vw;font-size:9vw;border-
radius:16px;border:0;background:#1b286e;color:#fff;touch-action:none}
input{width:80%}</style></head><body><h2>Titan Car</h2>
<div><button data-d='f'>&#9650;</button></div>
<div><button data-d='l'>&#9664;</button><button data-d='s'>&#9632;</button><button data-d='r'>&#9654;
</button></div>
<div><button data-d='b'>&#9660;</button></div>
<p>Speed <input type='range' id='v' min='100' max='255' value='180'></p>
<script>
let timer = null;
function send(d) { fetch('/go?d=' + d + '&v=' + document.getElementById('v').value); }
document.querySelectorAll('button').forEach(b => {
  b.onpointerdown = () => { send(b.dataset.d); clearInterval(timer); timer = setInterval(() =>
send(b.dataset.d), 200); };
  b.onpointerup = b.onpointerleave = () => { clearInterval(timer); send('s'); };
});
</script></body></html>)HTML";

void handleGo()
{
  String d = server.arg("d");
  speed = constrain(server.arg("v").toInt(), 100, 255);
  lastCommand = millis();
  if (d == "f") drive(speed, speed);
  else if (d == "b") drive(-speed, -speed);
  else if (d == "l") drive(-speed * 2 / 3, speed * 2 / 3);
  else if (d == "r") drive(speed * 2 / 3, -speed * 2 / 3);
  else drive(0, 0);
  server.send(200, "text/plain", "ok");
}

void setup()
{
  Serial.begin(9600);
  motorsBegin();
  WiFi.softAP(ssid, password);
  server.on("/", []() { server.send(200, "text/html", PAGE); });
  server.on("/go", handleGo);
```

```
  server.begin();
  Serial.print("Join Titan-Car and open http://");
  Serial.println(WiFi.softAPIP());
}

void loop()
{
  server.handleClient();
  if (millis() - lastCommand > 600) // no command for 0.6 s: stop
  {
    drive(0, 0);
  }
}
```

What you should see: Join Titan-Car (password titan1234) and open 192.168.4.1. Hold the up arrow and the car drives forward; let go and it stops. The slider sets the speed, and the car stops by itself if the phone loses the connection.

Bluetooth Robot Car

Goal: Drive the robot car from a Bluetooth car-controller app on an Android phone, with diagonal moves and ten speeds.

Most Bluetooth RC car apps send one letter per button: F, B, L and R for the four directions, G, I, H and J for the diagonals, S for stop and 0 to 9 for speed. The HC-05 passes those letters to the ESP32's second serial port, and a switch statement turns each letter into a drive() call. The car stops if the app goes quiet for a second.

Online page with troubleshooting: geekobot.in/projects/titan-kit-115-bluetooth-robot-car

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Reading a Bluetooth serial link on Serial2
- A one-letter command protocol
- Curved turns by giving the wheels different speeds
- Stopping safely when the signal is lost

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**GPIO16 (ESP32 RX2)**	
HC-05	RXD	**GPIO17 (ESP32 TX2)**	The ESP32 sends 3.3V, which the HC-05's RXD accepts directly

> The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
// Commands from Bluetooth RC car apps: F B L R, G I H J (diagonals), S stop, 0-9 and q speed
// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

int speed = 180;
unsigned long lastCommand = 0;

void setup()
{
  Serial.begin(9600);
  Serial2.begin(9600, SERIAL_8N1, 16, 17); // HC-05 TXD -> GPIO16, RXD -> GPIO17
  motorsBegin();
  Serial.println("Pair with HC-05 and connect from the app");
}

void loop()
{
  if (Serial2.available())
  {
    char c = Serial2.read();
    lastCommand = millis();
    Serial.println(c);
    int slow = speed / 3;
    switch (c)
    {
      case 'F': drive(speed, speed); break;
      case 'B': drive(-speed, -speed); break;
      case 'L': drive(-speed, speed); break;
      case 'R': drive(speed, -speed); break;
      case 'G': drive(slow, speed); break;   // forward left
      case 'I': drive(speed, slow); break;   // forward right
      case 'H': drive(-slow, -speed); break; // back left
      case 'J': drive(-speed, -slow); break; // back right
      case 'S': drive(0, 0); break;
      case 'q': speed = 255; break;
      default:
        if (c >= '0' && c <= '9')
        {
          speed = map(c - '0', 0, 9, 100, 240);
        }
        break;
    }
  }
  if (millis() - lastCommand > 1000) // the app went quiet: stop
  {
    drive(0, 0);
  }
}
```

What you should see: Pair the phone with HC-05 (PIN 1234) and connect from a Bluetooth RC car app. Arrow buttons drive the car, the diagonals curve it and the slider changes the speed. Every command shows in the Serial Monitor.

Obstacle Avoiding Robot Car

Goal: Let the car explore on its own: it drives until the ultrasonic sensor sees a wall, then backs off, turns and carries on.

The HC-SR04 at the front measures the distance ten times a second. While the way is clear the car drives forward, slowing down as it gets closer. Under 20 cm it stops, reverses briefly and turns, choosing left or right in turn so it does not get stuck in a corner, then looks again.

Online page with troubleshooting: geekobot.in/projects/titan-kit-116-obstacle-avoiding-robot-car

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Solderless Breadboard
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Measuring distance with sound echoes
- Making decisions from a sensor reading
- Slowing down in proportion to distance
- Alternating turns to escape corners

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	

HC-SR04	TRIG	**GPIO4**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal

> The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
// Obstacle avoiding car: press BOOT to start, BOOT again to stop
// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

int trigPin = 4, echoPin = 5;

float distanceCm()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  unsigned long t = pulseIn(echoPin, HIGH, 25000); // give up after 25 ms (about 4 m)
  if (t == 0) return 400;                          // nothing in range
  return t * 0.0343 / 2;                           // speed of sound, there and back
}

int startButton = 0;
bool running = false;
bool turnLeft = true;

void setup()
{
  Serial.begin(9600);
  motorsBegin();
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(startButton, INPUT_PULLUP);
  Serial.println("Press BOOT to start");
}

void loop()
{
  if (digitalRead(startButton) == LOW)
  {
    running = !running;
    drive(0, 0);
    Serial.println(running ? "Running" : "Stopped");
    delay(800);
  }
  if (!running)
  {
    return;
  }
  float d = distanceCm();
  Serial.print("Distance: ");
  Serial.println(d);
  if (d > 20)
  {
    int s = d > 60 ? 200 : map((int)d, 20, 60, 130, 200); // slow down near walls
    drive(s, s);
  }
```

```
  else
  {
    Serial.println(turnLeft ? "Wall: back off and turn left" : "Wall: back off and turn right");
    drive(0, 0);
    delay(200);
    drive(-160, -160);
    delay(400);
    if (turnLeft) drive(-170, 170);
    else drive(170, -170);
    delay(450);
    turnLeft = !turnLeft;
    drive(0, 0);
    delay(150);
  }
  delay(60);
}
```

What you should see: Press BOOT and the car drives forward. It slows as it nears a wall, stops at about 20 cm, backs off and turns. The Serial Monitor prints the distance and each decision.

IR Remote Robot Car

Goal: Drive the robot car with the kit's infrared remote: hold an arrow to move, number keys set the speed.

While a remote key is held, it keeps sending a short repeat code about nine times a second. The sketch drives for as long as codes keep arriving and stops 250 ms after the last one, so letting go of the key stops the car. The number keys 1 to 9 set the speed and OK stops at once.

Online page with troubleshooting: geekobot.in/projects/titan-kit-117-ir-remote-robot-car

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x IR Receiver Module (VS1838B / TSOP)
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Using IR repeat codes for hold-to-drive
- A timeout that turns no signal into stop
- Mapping keys to actions
- Reading key codes from the Serial Monitor

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	
IR receiver	VCC	**3V3**	
IR receiver	GND	**GND**	
IR receiver	OUT / S	**GPIO35**	Mount it facing backwards or up, so the remote reaches it while the car drives away

> The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <IRremote.hpp>

// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

#define IR_PIN 35
// key codes of the kit's 17-key remote (NEC): if yours differ, read them in the Serial Monitor
const uint8_t KUP = 0x18, KDOWN = 0x52, KLEFT = 0x08, KRIGHT = 0x5A, KOK = 0x1C;
const uint8_t DIGITS[9] = {0x45, 0x46, 0x47, 0x44, 0x40, 0x43, 0x07, 0x15, 0x09}; // keys 1-9
int speed = 180;
unsigned long lastSignal = 0;

void setup()
{
  Serial.begin(9600);
  motorsBegin();
  IrReceiver.begin(IR_PIN);
  Serial.println("Hold an arrow key on the remote to drive");
}

void loop()
{
  if (IrReceiver.decode())
  {
    uint8_t key = IrReceiver.decodedIRData.command;
    IrReceiver.resume();
    lastSignal = millis();
    Serial.print("Key 0x");
    Serial.println(key, HEX);
    if (key == KUP) drive(speed, speed);
    else if (key == KDOWN) drive(-speed, -speed);
    else if (key == KLEFT) drive(-speed * 2 / 3, speed * 2 / 3);
    else if (key == KRIGHT) drive(speed * 2 / 3, -speed * 2 / 3);
    else if (key == KOK) drive(0, 0);
    for (int i = 0; i < 9; i++)
    {
      if (key == DIGITS[i])
      {
        speed = map(i, 0, 8, 110, 255);
        Serial.print("Speed ");
        Serial.println(speed);
      }
    }
  }
  if (millis() - lastSignal > 250) // key released: stop
  {
    drive(0, 0);
  }
}
```

What you should see: Hold the up arrow and the car drives; release and it stops. Left and right spin the car, down reverses. Keys 1 to 9 change the speed and the Serial Monitor prints every code.

Follow Me Robot Car

Goal: Make the car follow your hand like a pet: it keeps a set distance, moving forward when you step away and backing off when you come close.

This is a simple control loop. The car wants to stay 20 cm from your hand. The error is the measured distance minus 20: a big positive error means drive forward fast, a small one slowly, and a negative error means reverse. Inside a 5 cm dead band it stops, so it does not twitch back and forth.

Online page with troubleshooting: geekobot.in/projects/titan-kit-118-follow-me-robot-car

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Solderless Breadboard
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Proportional control: speed from the size of the error
- Why a dead band stops jitter
- Ignoring readings that are out of range
- Smoothing a sensor with an average

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	

HC-SR04	TRIG	**GPIO4**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal

> The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
// Follow-me car: keeps 20 cm from your hand. BOOT starts and stops it.
// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

int trigPin = 4, echoPin = 5;

float distanceCm()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  unsigned long t = pulseIn(echoPin, HIGH, 25000); // give up after 25 ms (about 4 m)
  if (t == 0) return 400;                          // nothing in range
  return t * 0.0343 / 2;                           // speed of sound, there and back
}

int startButton = 0;
bool running = false;
float target = 20;  // cm to keep from the hand
float band = 5;     // no movement within 5 cm of the target
float gain = 8;     // speed added per cm of error

void setup()
{
  Serial.begin(9600);
  motorsBegin();
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(startButton, INPUT_PULLUP);
  Serial.println("Press BOOT to start");
}

void loop()
{
  if (digitalRead(startButton) == LOW)
  {
    running = !running;
    drive(0, 0);
    delay(800);
  }
  if (!running)
  {
    return;
  }
  float d = (distanceCm() + distanceCm() + distanceCm()) / 3; // average of three
  float error = d - target;
  int s = 0;
  if (d > 60)
  {
    s = 0; // nothing to follow
  }
```

```
  else if (abs(error) > band)
  {
    s = constrain((int)(error * gain), -200, 200);
    if (s > 0 && s < 110) s = 110;     // motors need a minimum to move
    if (s < 0 && s > -110) s = -110;
  }
  drive(s, s);
  Serial.printf("Distance %.0f cm, speed %d\n", d, s);
  delay(50);
}
```

What you should see: Press BOOT, then hold your hand in front of the sensor. Move your hand away slowly and the car follows; move it closer and the car backs off. With nothing in front within 60 cm it waits.

Gyro Straight Line Robot Car

Goal: Use the MPU6050 gyroscope so the car drives dead straight and turns exactly 90 degrees, then drives a perfect square.

Two motors are never exactly the same, so a car left to itself drifts. The gyroscope measures how fast the car rotates; adding that up over time gives the heading. While driving, the sketch speeds up one wheel and slows the other in proportion to the heading error. For a turn it spins until the heading has changed by 90 degrees.

Online page with troubleshooting: geekobot.in/projects/titan-kit-119-gyro-straight-line-robot-car

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x GY-521 MPU6050 6-Axis Gyro and Accelerometer Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Integrating a gyroscope to get a heading
- Removing gyro drift with a start-up calibration
- Correcting a course with proportional control
- Turning by angle instead of by time

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	
MPU6050	VCC	**3V3**	
MPU6050	GND	**GND**	
MPU6050	SDA	**GPIO21**	
MPU6050	SCL	**GPIO22**	Mount the board flat on the chassis, chip side up

> The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

> The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

> Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

Adafruit_MPU6050 mpu;
int startButton = 0;
float heading = 0;   // degrees, positive = turned left
float gyroBias = 0;  // the gyro's reading when still
unsigned long lastMicros = 0;
int base = 170;      // cruising speed
float kp = 6;        // correction per degree of error

void updateHeading()
{
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  unsigned long now = micros();
  float dt = (now - lastMicros) / 1000000.0;
  lastMicros = now;
  heading += (g.gyro.z - gyroBias) * dt * 57.2958; // rad/s to degrees
}

void straight(float target, unsigned long ms)
{
  unsigned long start = millis();
  while (millis() - start < ms)
  {
    updateHeading();
    int fix = (int)((heading - target) * kp); // turned left: speed up the left wheel
    drive(base + fix, base - fix);
    delay(5);
  }
  drive(0, 0);
  Serial.printf("Side done, heading %.1f\n", heading);
}

void turnTo(float target)
{
  while (heading < target - 2)
  {
    updateHeading();
    drive(-150, 150); // spin left
    delay(5);
  }
  drive(0, 0);
  delay(300);
}

void setup()
{
```

```
  Serial.begin(9600);
  motorsBegin();
  pinMode(startButton, INPUT_PULLUP);
  if (!mpu.begin())
  {
    Serial.println("MPU6050 not found");
    while (true) { }
  }
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  Serial.println("Calibrating: keep the car still");
  float sum = 0;
  for (int i = 0; i < 400; i++)
  {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    sum += g.gyro.z;
    delay(5);
  }
  gyroBias = sum / 400;
  lastMicros = micros();
  Serial.println("Ready: press BOOT");
}

void loop()
{
  updateHeading();
  if (digitalRead(startButton) == HIGH)
  {
    delay(5);
    return;
  }
  delay(1000);
  heading = 0;
  lastMicros = micros();
  for (int side = 0; side < 4; side++)
  {
    straight(side * 90, 1500);
    turnTo((side + 1) * 90);
  }
  Serial.println("Square done");
}
```

What you should see: Keep the car still for two seconds after power-on while it calibrates. Press BOOT and it drives a square with sharp 90-degree corners. The Serial Monitor prints the heading during each side.

Auto Braking Smart Wi-Fi Robot Car

Goal: The Titan Kit finale: a phone-driven car with a live distance read-out, automatic braking before walls and LED-ring headlights and brake lights.

This project joins the Wi-Fi car, the ultrasonic sensor and the LED ring. The page sends drive commands as in project 114 and asks the car for its status five times a second. Before every forward move the car checks the distance: under 20 cm it refuses to go forward, turns the ring red and beeps, but still lets you reverse or turn away.

Online page with troubleshooting: geekobot.in/projects/titan-kit-120-smart-wifi-robot-car-auto-brake

Parts

- 1 x ESP32-WROOM-32 38-Pin Development Board
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Solderless Breadboard
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 1k and 2k resistors
- 1 x WS2812 RGB LED Ring
- 1 x PCB Mounted Active Buzzer Module
- 1 x ESP32 38-Pin Expansion Board
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Joining several projects into one system
- A status endpoint that a page polls with fetch()
- Safety rules that override the driver
- Using an LED ring as lights

Wiring

Part	Pin	ESP32	Note
L298N	IN1	**GPIO27**	
L298N	IN2	**GPIO26**	
L298N	IN3	**GPIO25**	
L298N	IN4	**GPIO33**	
L298N	ENA	**GPIO14**	Pull the ENA jumper cap off first, so the ESP32 sets the speed
L298N	ENB	**GPIO32**	Pull the ENB jumper cap off first
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and ESP32 GND**	The common ground is required
L298N	+5V	**ESP32 5V (VIN) pin**	Powers the ESP32 from the pack when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	
HC-SR04	VCC	**5V**	

HC-SR04	GND	**GND**	
HC-SR04	TRIG	**GPIO4**	
HC-SR04	ECHO	**GPIO5**	Through a divider: 1k from ECHO to GPIO5 and 2k from GPIO5 to GND, because ECHO is a 5V signal
LED ring	5V / VCC	**5V**	The ring's LEDs need 5V; brightness is capped in the sketch
LED ring	GND	**GND**	
LED ring	DIN (data in)	**GPIO18**	Use the DIN pad, not DOUT
Buzzer	Buzzer (+)	**GPIO23**	
Buzzer	Buzzer (−)	**GND**	

The motors run from the 2 x 18650 pack (7.4V) through the L298N, and the L298N's 5V output powers the ESP32 when the USB cable is out. Switch the pack off whenever the USB cable is plugged in for an upload.

The ESP32 is a 3.3V board: never connect a 5V signal straight to a GPIO pin.

Lift the wheels off the table for the first test, and switch the battery holder off before you change any wiring.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

The page has no login beyond the Wi-Fi password: change the password in the sketch before you use it in a shared space.

Code

```
#include <WiFi.h>
#include <WebServer.h>
#include <Adafruit_NeoPixel.h>

// L298N: IN1-IN4 set each wheel's direction, ENA/ENB its speed (0-255)
int in1 = 27, in2 = 26, in3 = 25, in4 = 33;
int enA = 14, enB = 32;

void drive(int left, int right) // -255 (full reverse) to 255 (full forward)
{
  digitalWrite(in1, left > 0 ? HIGH : LOW);
  digitalWrite(in2, left < 0 ? HIGH : LOW);
  digitalWrite(in3, right > 0 ? HIGH : LOW);
  digitalWrite(in4, right < 0 ? HIGH : LOW);
  analogWrite(enA, constrain(abs(left), 0, 255));
  analogWrite(enB, constrain(abs(right), 0, 255));
}

void motorsBegin()
{
  int pins[6] = {in1, in2, in3, in4, enA, enB};
  for (int i = 0; i < 6; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  drive(0, 0);
}

int trigPin = 4, echoPin = 5;

float distanceCm()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  unsigned long t = pulseIn(echoPin, HIGH, 25000); // give up after 25 ms (about 4 m)
  if (t == 0) return 400;                          // nothing in range
  return t * 0.0343 / 2;                           // speed of sound, there and back
}

#define RING_PIN 18
Adafruit_NeoPixel ring(8, RING_PIN, NEO_GRB + NEO_KHZ800);
WebServer server(80);
const char* ssid = "Titan-Car";
const char* password = "titan1234";
int buzzer = 23;
int speed = 180;
float distance = 400;
bool braked = false;
char current = 's';
unsigned long lastCommand = 0, lastPing = 0;

const char PAGE[] = R"HTML(<!DOCTYPE html><html><head><meta name='viewport' content='width=device-width,
initial-scale=1, user-scalable=no'>
<title>Titan Car</title><style>body{font-family:sans-serif;text-align:center;user-select:none}
button{width:28vw;height:28vw;margin:1vw;font-size:9vw;border-
radius:16px;border:0;background:#1b286e;color:#fff;touch-action:none}
#d{font-size:28px}</style></head><body><h2>Titan Smart Car</h2>
<div><button data-d='f'>&#9650;</button></div>
<div><button data-d='l'>&#9664;</button><button data-d='s'>&#9632;</button><button data-d='r'>&#9654;
</button></div>
<div><button data-d='b'>&#9660;</button></div><p id='d'>-</p>
<script>
let timer = null;
function send(d) { fetch('/go?d=' + d); }
document.querySelectorAll('button').forEach(b => {
  b.onpointerdown = () => { send(b.dataset.d); clearInterval(timer); timer = setInterval(() =>
send(b.dataset.d), 200); };
  b.onpointerup = b.onpointerleave = () => { clearInterval(timer); send('s'); };
});
setInterval(() => fetch('/status').then(r => r.text()).then(t => document.getElementById('d').textContent
= t), 200);
```

```
</script></body></html>)HTML";

void lights(uint8_t r, uint8_t g, uint8_t b)
{
  for (int i = 0; i < 8; i++)
  {
    ring.setPixelColor(i, ring.Color(r, g, b));
  }
  ring.show();
}

void apply()
{
  braked = (current == 'f' && distance < 20);
  if (braked)
  {
    drive(0, 0);
    lights(255, 0, 0);
    digitalWrite(buzzer, HIGH);
    return;
  }
  digitalWrite(buzzer, LOW);
  if (current == 'f') drive(speed, speed);
  else if (current == 'b') drive(-speed, -speed);
  else if (current == 'l') drive(-speed * 2 / 3, speed * 2 / 3);
  else if (current == 'r') drive(speed * 2 / 3, -speed * 2 / 3);
  else drive(0, 0);
  if (current == 'b') lights(255, 0, 0);       // reversing: red
  else if (current == 's') lights(40, 40, 40); // parked: dim
  else lights(255, 255, 255);                  // driving: headlights
}

void handleGo()
{
  current = server.arg("d").charAt(0);
  lastCommand = millis();
  apply();
  server.send(200, "text/plain", "ok");
}

void handleStatus()
{
  String s = String(distance, 0) + " cm" + (braked ? " - AUTO BRAKE" : "");
  server.send(200, "text/plain", s);
}

void setup()
{
  Serial.begin(9600);
  motorsBegin();
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzer, OUTPUT);
  ring.begin();
  ring.setBrightness(80);
  lights(40, 40, 40);
  WiFi.softAP(ssid, password);
  server.on("/", []() { server.send(200, "text/html", PAGE); });
  server.on("/go", handleGo);
  server.on("/status", handleStatus);
  server.begin();
  Serial.print("Join Titan-Car and open http://");
  Serial.println(WiFi.softAPIP());
}

void loop()
{
  server.handleClient();
  if (millis() - lastPing > 60)
  {
    lastPing = millis();
    distance = distanceCm();
    apply(); // brakes even if no new command arrives
  }
  if (current != 's' && millis() - lastCommand > 600) // phone went quiet: stop
```

```
  {
    current = 's';
    apply();
  }
}
```

What you should see: Join Titan-Car (password titan1234) and open 192.168.4.1: the distance to the nearest wall shows under the buttons. Driving forward towards a wall, the car brakes at 20 cm by itself, the ring turns red and the buzzer beeps. Reverse or turn away and the ring goes back to white headlights.

WHAT NEXT

Keep building

Every project in this book follows one pattern: read an input, make a decision, drive an output. The Wi-Fi pages, the RFID systems and the robot car are the same idea with more parts. Now remix them:

- Send the weather station readings to the robot car's web page.
- Add the RFID reader to the Wi-Fi car so only your card unlocks it.
- Log the BMP280 and DHT11 readings every ten minutes and draw a graph on the OLED.
- Use the MPU6050 on your wrist as a tilt remote for the car over Bluetooth.

More than 1,900 free project guides with parts lists, wiring and code: **geekobot.in/projects**

Geekobot
Imagine. Build. Innovate.

Build. Learn. Innovate.