

# RaspTankPro — Student Programming Guide

---

This guide explains how to write programs for the robot using the **sandbox environment**. You will use Python and a simple set of commands — no knowledge of hardware or electronics is required.

---

## Table of Contents

1. [Hardware Overview](#)
  2. [Getting Started](#)
  3. [Movement](#)
  4. [Sensors](#)
  5. [Servos and the Arm](#)
  6. [Visual Odometry \(Position Tracking\)](#)
  7. [Display and LEDs](#)
  8. [Example Programs](#)
  9. [Running the Hardware Tests](#)
  10. [Troubleshooting](#)
- 

## 1. Hardware Overview

Component	Description
<b>Drive motors</b>	Two motors (left and right) control movement and turning
<b>Servos</b>	Five servos: arm rotation (left-right), arm (up-down), hand (up-down), gripper (open-close), camera tilt (up-down)
<b>Ultrasonic sensor</b>	Measures distance to the nearest obstacle (in metres)
<b>MPU6050</b>	Combined gyroscope and accelerometer (angular velocity + acceleration)
<b>Camera</b>	Raspberry Pi camera, used for the video stream and visual odometry
<b>LED strip</b>	16 RGB NeoPixel LEDs
<b>OLED display</b>	Small SSD1306 screen with 6 lines of text

---

## 2. Getting Started

Powering on the robot

1. The robot turns on automatically when connected to a power source.
2. Connect a keyboard, mouse, and monitor if you need to write or edit programs.
3. **Thonny IDE** opens automatically with `sandbox.py` loaded — this is your programming environment.

Writing and running your program

1. Define any helper functions you need in the **functions area** above `run()` — there is an example function there to show the format.
2. Write your main program inside the `run()` function.
3. Press **F5** (or click the Run button in Thonny) to execute your program.
  - The web control server is stopped automatically when you press Run.
  - The robot's hardware is now exclusively available to your program.
4. Press the **Stop button** (or **F2**) at any time to stop the robot safely.

## Switching between Web Server and Sandbox mode

The file `~/startup.sh` (`/home/pi/startup.sh`) controls what the robot runs on boot. Open it in a text editor (e.g. `nano ~/startup.sh`) and swap the comments:

**Web Server mode** (default — control the robot from a browser):

```
# Comment this line when working in sandbox:  
sudo python3 /home/pi/adeept_rasptankpro/server/webServer.py  
  
# Uncomment this line when working in sandbox:  
#sudo python3 /home/pi/adeept_rasptankpro/server/sandbox.py
```

**Sandbox mode** (robot runs your `run()` function automatically on boot):

```
# Comment this line when working in sandbox:  
#sudo python3 /home/pi/adeept_rasptankpro/server/webServer.py  
  
# Uncomment this line when working in sandbox:  
sudo python3 /home/pi/adeept_rasptankpro/server/sandbox.py
```

Save the file and reboot. The robot will start in the selected mode.

## Emergency stop

If the robot runs away or behaves unexpectedly and you cannot reach Thonny:

- Double-click the **EMERGENCY STOP** icon on the desktop.
- This immediately kills your running program and stops the motors.

## 3. Movement

All movement functions accept a `speed` (0–100) and an optional `duration` in seconds. If `duration` is omitted, the robot keeps moving until you call `robot.stop()`.

```
robot.forward(speed=50, duration=2.0) # Move forward for 2 seconds at half speed  
robot.backward(speed=50, duration=1.0) # Move backward for 1 second  
robot.turn_left(speed=50, duration=0.5) # Pivot left (right wheel drives, left
```

```
stops)
robot.turn_right(speed=50, duration=0.5) # Pivot right (left wheel drives, right
stops)
robot.spin_left(speed=40, duration=1.0) # Spin in place counter-clockwise
robot.spin_right(speed=40, duration=1.0) # Spin in place clockwise
robot.stop() # Stop immediately
robot.wait(1.5) # Block for 1.5 s – motors keep running if
already on
```

## Function reference

### `robot.forward(speed=50, duration=None)`

Move forward.

- `speed` — motor power, 0–100 (default 50)
- `duration` — seconds to move; if `None`, moves until `stop()` is called

### `robot.backward(speed=50, duration=None)`

Move backward. Same parameters as `forward`.

### `robot.turn_left(speed=50, duration=None)`

Pivot left. The right wheel drives forward; the left wheel stops.

### `robot.turn_right(speed=50, duration=None)`

Pivot right. The left wheel drives forward; the right wheel stops.

### `robot.spin_left(speed=50, duration=None)`

Spin counter-clockwise in place. Both wheels move at the same speed in opposite directions.

### `robot.spin_right(speed=50, duration=None)`

Spin clockwise in place.

### `robot.stop()`

Stop all motors immediately.

### `robot.wait(seconds)`

Block execution for the given number of seconds — the Python program does nothing during this time. **Any motors already running will continue running.** Use `robot.stop()` before or after `wait()` if you want the robot to stay still.

---

## 4. Sensors

## Distance sensor

```
distance = robot.get_distance()  
print(f"Obstacle is {distance:.2f} metres away")
```

### robot.get\_distance() → float

Returns the distance (in **metres**) to the nearest obstacle directly in front of the sensor. Values above ~2 m may be inaccurate. Returns 0 if no echo is received.

---

## Gyroscope

```
gyro = robot.get_gyro()  
print(f"Rotation - x: {gyro['x']:.2f} y: {gyro['y']:.2f} z: {gyro['z']:.2f}  
°/s")
```

### robot.get\_gyro() → dict

Returns angular velocity in **degrees per second** as a dictionary with keys '**x**', '**y**', '**z**'.

Axis	Meaning
x	Rolling (tilting side to side)
y	Pitching (tilting forward/backward)
z	Yawing (rotating left/right)

Returns `{'x': 0, 'y': 0, 'z': 0}` if the sensor is not connected.

---

## Accelerometer

```
accel = robot.get_accel()  
print(f"Acceleration - x: {accel['x']:.2f} y: {accel['y']:.2f} z:  
{accel['z']:.2f} g")
```

### robot.get\_accel() → dict

Returns linear acceleration in **g** ( $1\text{ g} \approx 9.81\text{ m/s}^2$ ) as a dictionary with keys '**x**', '**y**', '**z**'. When the robot is flat and still,  $z \approx 1.0$  (gravity) and  $x \approx y \approx 0$ .

Returns `{'x': 0, 'y': 0, 'z': 1}` if the sensor is not connected.

---

## 5. Servos and the Arm

All servo functions take a **position** from **-1.0** (one extreme) to **+1.0** (the other extreme), with **0.0** being the centre.

```
robot.set_arm_rotation(-0.5) # Rotate arm halfway to the left
robot.set_arm(0.8)           # Raise arm 80% of the way up
robot.set_hand(0.5)          # Raise hand halfway up
robot.set_gripper(-1.0)       # Close gripper fully
robot.set_gripper(1.0)        # Open gripper fully
robot.set_camera_tilt(1.0)    # Tilt camera fully up
robot.reset_servos()         # Return all servos to centre
```

### Function reference

#### **robot.set\_arm\_rotation(position)**

Rotate the arm left or right (turret rotation).

- **-1.0** = full left, **0.0** = centre, **+1.0** = full right

#### **robot.set\_arm(position)**

Move the arm joint up or down.

- **-1.0** = fully down, **0.0** = centre, **+1.0** = fully up

#### **robot.set\_hand(position)**

Move the hand (wrist/forearm) up or down.

- **-1.0** = fully down, **0.0** = centre, **+1.0** = fully up

#### **robot.set\_gripper(position)**

Open or close the gripper.

- **-1.0** = fully closed, **0.0** = centre, **+1.0** = fully open

#### **robot.set\_camera\_tilt(position)**

Tilt the camera up or down.

- **-1.0** = full down, **0.0** = centre, **+1.0** = full up

#### **robot.reset\_servos()**

Move all servos back to their centre position (position **0.0**).

## 6. Visual Odometry (Position Tracking)

Visual odometry estimates the robot's position by analysing movement between consecutive camera frames. It uses **FAST feature detection** and **Lucas-Kanade optical flow** to track how the scene shifts from frame to frame, then calculates the camera's 3D translation.

### Important limitations

**Monocular scale ambiguity:** A single camera cannot determine real-world distances without an external reference. The x, y, z values returned by `get_position()` are in **relative units**, not metres. The scale depends on `absolute_scale` (default 1.0). If you need real-world distances, you need to calibrate this value against a known distance.

**Drift:** Visual odometry accumulates small errors over time. Long paths will show drift from the true position.

**Camera conflict:** The web control server also uses the camera. `start_odometry()` requires the server to be stopped first — this happens automatically when you press F5 in Thonny.

### Usage

```
# Start tracking position
robot.start_odometry()

# ... move the robot ...
robot.forward(speed=40, duration=3.0)

# Read current position (x, y, z in relative units)
x, y, z = robot.get_position()
print(f"Position: x={x:.2f} y={y:.2f} z={z:.2f}")

# Reset origin to current position
robot.reset_position()

# Stop tracking when done
robot.stop_odometry()
```

### Function reference

`robot.start_odometry(focal_length=537.0, pp=(320.0, 240.0), scale=1.0, show_debug=False)`

Start position tracking in the background. All parameters have defaults — pass only the ones you need. For example, `robot.start_odometry(scale=4.7)` is valid.

- `focal_length` — camera focal length in pixels (default 537.0 for the Pi camera at 640×480)
- `pp` — optical centre (cx, cy) in pixels (default (320.0, 240.0))
- `scale` — scale factor applied to each translation step (default 1.0)
- `show_debug` — if `True`, opens two windows while odometry runs: a live camera feed with tracked feature points and a 2-D trajectory map. Useful for verifying the camera works and the estimated path

makes sense. Default `False`.

Raises `RuntimeError` if the camera cannot be opened.

`robot.get_position() → (x, y, z)`

Return the latest position estimate as a tuple of three floats. The origin is the robot's position when `start_odometry()` was called (or `reset_position()`). Raises `RuntimeError` if `start_odometry()` has not been called.

`robot.reset_position()`

Reset the current position to `(0, 0, 0)`. Raises `RuntimeError` if `start_odometry()` has not been called.

`robot.stop_odometry()`

Stop position tracking and release the camera.

---

## 7. Display and LEDs

### LED strip

The robot has 16 RGB LEDs. Set them all to any colour with three 0–255 values (red, green, blue).

```
robot.set_led_color(255, 0, 0)      # Red
robot.set_led_color(0, 255, 0)      # Green
robot.set_led_color(0, 0, 255)      # Blue
robot.set_led_color(255, 165, 0)    # Orange
robot.led_off()                   # Turn off
```

`robot.set_led_color(r, g, b)`

Set all LEDs to an RGB colour.

- `r, g, b` — red, green, blue values, each 0–255

`robot.led_off()`

Turn all LEDs off (equivalent to `set_led_color(0, 0, 0)`).

---

### OLED display

The robot has a small SSD1306 OLED screen with 6 lines of text (line 1 at the top, line 6 at the bottom).

```
robot.show_display(1, 'Hello!')
robot.show_display(2, f'Dist: {distance:.2f} m')
```

```
robot.show_display(3, f'Speed: {speed}')
robot.clear_display()           # Clear all lines
```

### **robot.show\_display(line, text)**

Write text to a display line.

- **line** — 1 (top) to 6 (bottom)
- **text** — any value; numbers are converted to strings automatically

### **robot.clear\_display()**

Clear all six lines of the display.

---

## 8. Example Programs

Example 1 — Move forward and stop before an obstacle

```
def run():
    SAFE_DISTANCE = 0.30  # metres

    robot.forward(speed=40)    # Start moving (no duration – keep going)

    while True:
        distance = robot.get_distance()
        print(f"Distance: {distance:.2f} m")

        if distance < SAFE_DISTANCE:
            robot.stop()
            print("Obstacle detected! Stopped.")
            break

    robot.wait(0.05)  # Small delay between readings
```

---

Example 2 — Arm rotation sweep with distance readings

```
def run():
    print("Scanning...")
    positions = [-1.0, -0.5, 0.0, 0.5, 1.0]

    for pos in positions:
        robot.set_arm_rotation(pos)
        robot.wait(0.5)  # Wait for servo to settle
        distance = robot.get_distance()
        angle_label = f"{int(pos * 90)}:{+d}°"
        print(f" {angle_label}: {distance:.2f} m")
```

```
robot.reset_servos()
print("Scan complete.")
```

---

### Example 3 — Log sensor data to a file

```
def run():
    import csv
    import time

    LOG_FILE = '/home/pi/sensor_log.csv'
    DURATION = 10.0 # seconds

    print(f"Logging sensors for {DURATION} s → {LOG_FILE}")

    with open(LOG_FILE, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(['time_s', 'distance_m',
                        'gyro_x', 'gyro_y', 'gyro_z',
                        'accel_x', 'accel_y', 'accel_z'])

        start = time.time()
        while time.time() - start < DURATION:
            t = time.time() - start
            dist = robot.get_distance()
            gyro = robot.get_gyro()
            accel = robot.get_accel()

            writer.writerow([
                round(t, 3), dist,
                gyro['x'], gyro['y'], gyro['z'],
                accel['x'], accel['y'], accel['z'],
            ])
            print(f"t={t:.1f}s dist={dist:.2f}m "
                  f"gyro_z={gyro['z']:.2f}")
            time.sleep(0.1)

    print("Done. Open the CSV file in a spreadsheet to analyse the data.")
```

---

## 9. Running the Hardware Tests

To verify that all hardware is working correctly:

```
# In sandbox.py, replace the contents of run() with:
from robot_test import run_all_tests
```

```
def run():
    run_all_tests(robot)
```

The test script will check each component in sequence and print [ PASS ] or [ FAIL ] for each one. The full test takes about 30 seconds and physically moves the robot, so make sure there is space around it.

---

## 10. Troubleshooting

### "Cannot open camera" when calling `start_odometry()`

The camera is in use by another process (usually the web server). Make sure you ran the program through Thonny (F5), which stops the server automatically before starting your code.

### I2C / sensor errors at startup

The MPU6050 gyro/accelerometer communicates over I2C. If it fails to initialise, `get_gyro()` and `get_accel()` return zero values instead of crashing. Check that the sensor is connected and that I2C is enabled (`sudo raspi-config` → Interface Options → I2C).

### Servos do not move

The servo controller uses I2C (same bus as the gyro). Check the wiring and that I2C is enabled. If only some servos fail, there may be a PWM channel wiring issue.

### Robot does not stop when F2 is pressed

Thonny sends SIGTERM to the running script. `sandbox.py` catches this signal and calls `robot.cleanup()`, which stops the motors. If the robot still moves, use the **EMERGENCY STOP** desktop shortcut.

### GPIO warnings at startup

Raspberry Pi GPIO may print `RuntimeWarning: This channel is already in use`. This is harmless — it means the GPIO was not released cleanly by a previous run. The robot will still work correctly.

### Motors spin but robot does not move straight

The two motors may have slightly different efficiencies. Use a higher `speed` value or adjust the `radius` parameter via `move.move()` directly for fine-tuned control (advanced use).