LE/ESSE 2220 Algorithmic and Computational Methods

Lab 4: Joystick Space Navigation

(Fall 2025-2026)

Z. ARJMANDI (ZARARJ@YORKU.CA)

Review

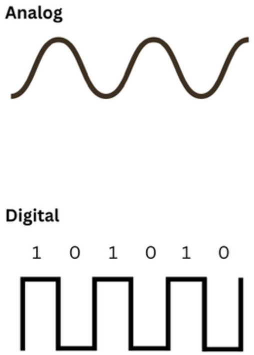
Raspberry Pi GPIO – Inputs & Outputs

Inputs (read, sense)

- **Digital Input (0 / 1, LOW / HIGH)**
  - Button, switch, motion sensor
  - Data type: **Boolean (True/False)**
- **Analog Input** (needs external ADC)
  - Temperature sensor, potentiometer
  - Data type: **Integer/Float** (e.g., 0–1023)

Outputs (write, interact)

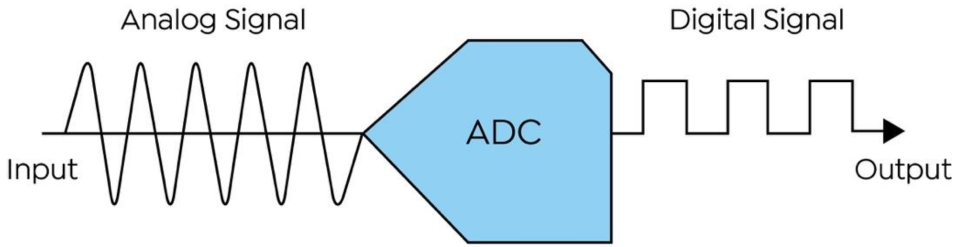
- **Digital Output (0 / 1, OFF / ON)**
  - LED, buzzer, relay
  - Data type: **Boolean (True/False)**
- **PWM Output (Pulse Width Modulation)**
  - LED dimming, motor speed control, servo angle, **Buzzer tone generation**
  - Data type: **Duty Cycle % (0–100)**

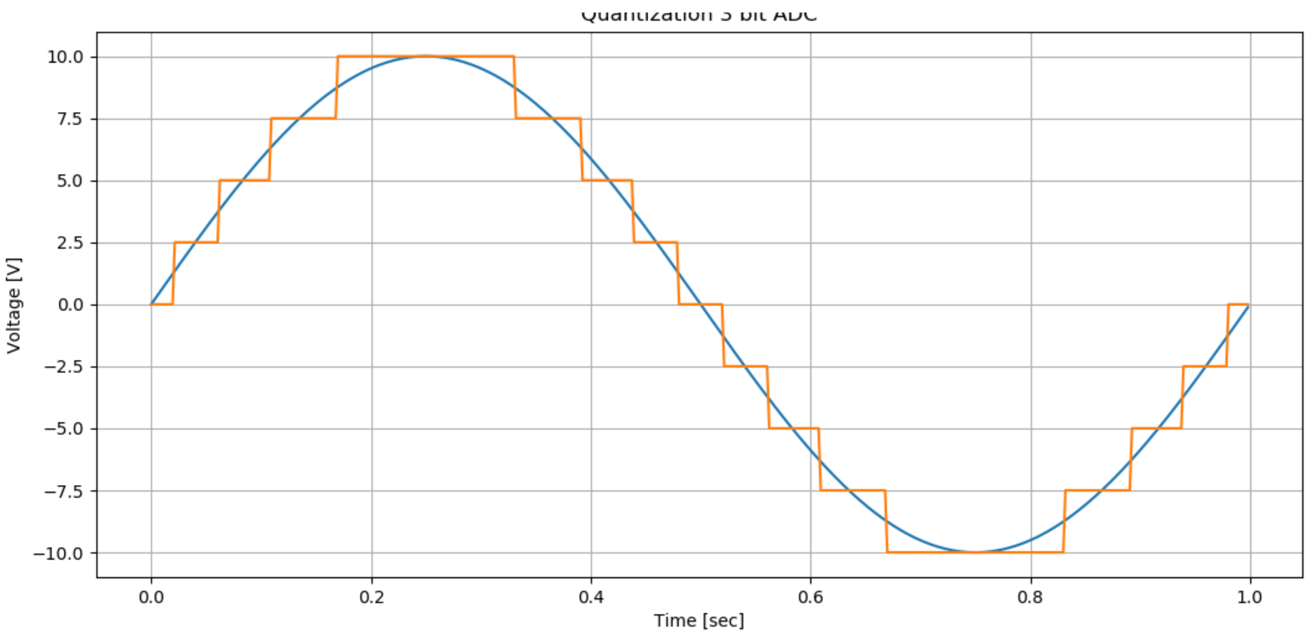


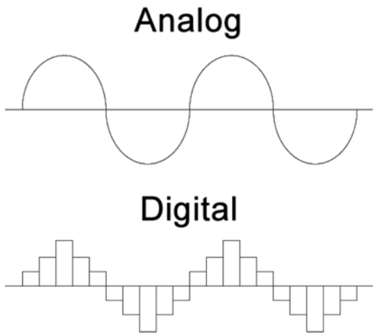
Inputs: ADC (Analog-to-Digital Converter)

- **Conversion:**
  - ADC maps input voltage into a digital number
  - Example (10-bit, 3.3 V):
    - 0 V → 0
    - 1.65 V → ~512
    - 3.3 V → 1023

(2^10=1024)



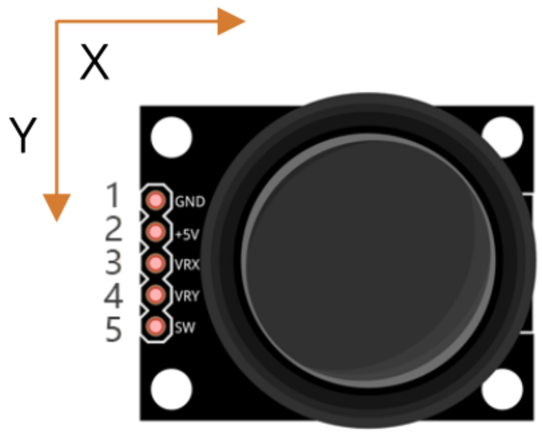


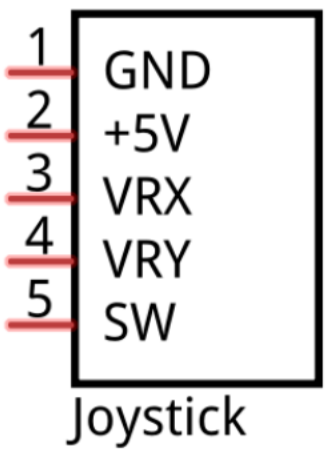


Lab 4

Joystick

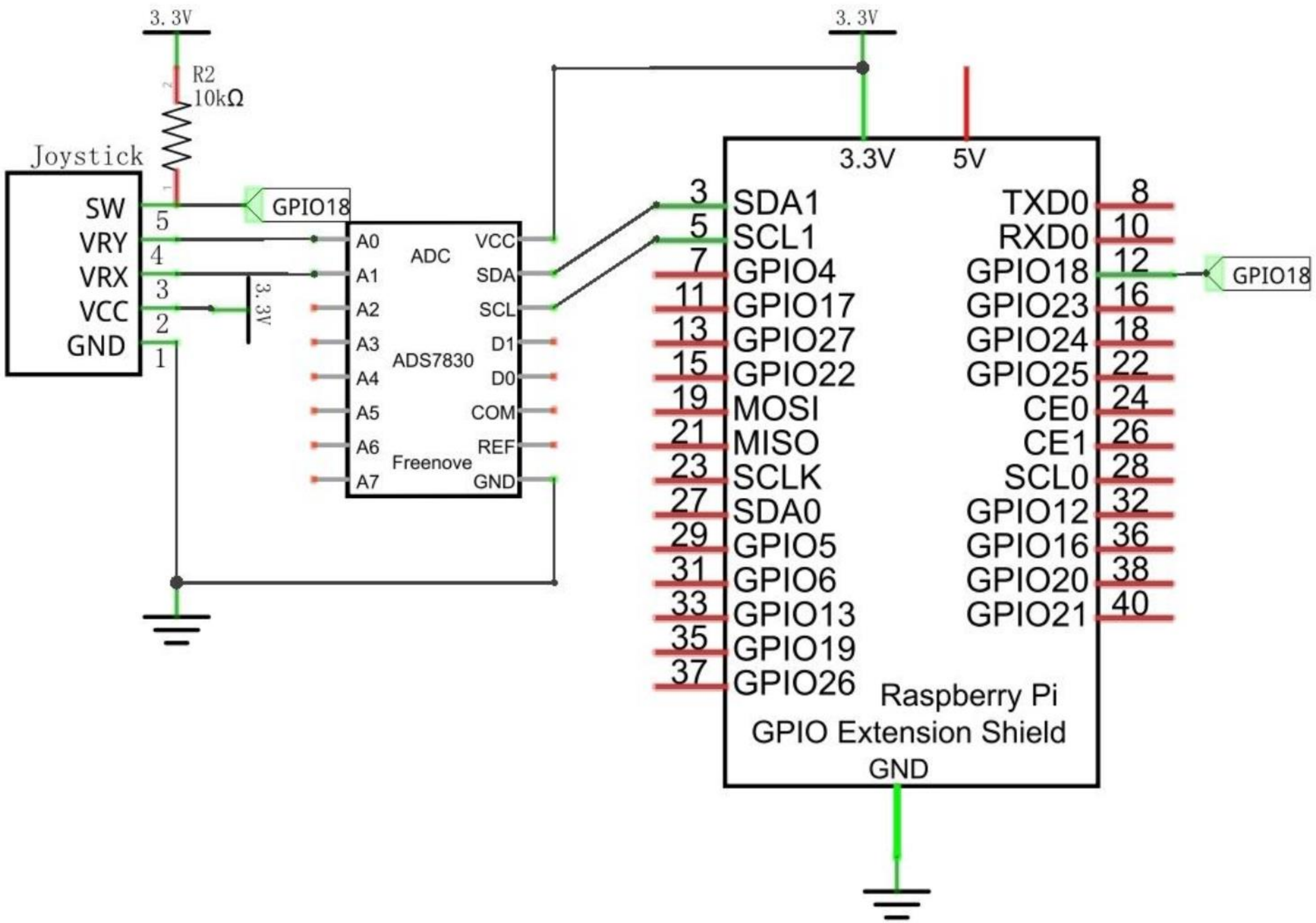
Lab 4: Joystick Space Navigation





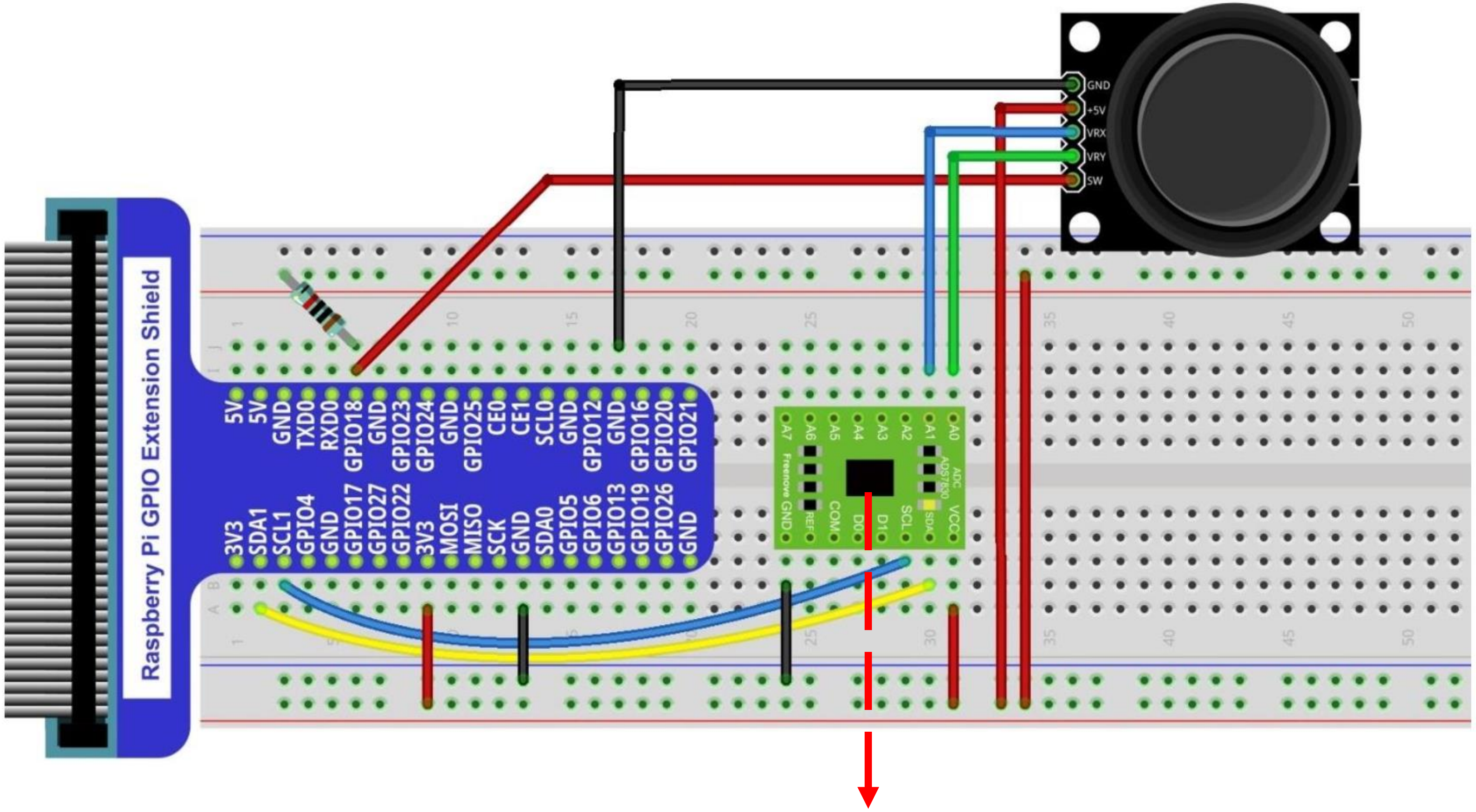
Lab 4: Joystick Space Navigation

- For more details on this circuit, please refer to Tutorial Chapter 12 (available on E-Class).



Lab 4: Joystick Space Navigation

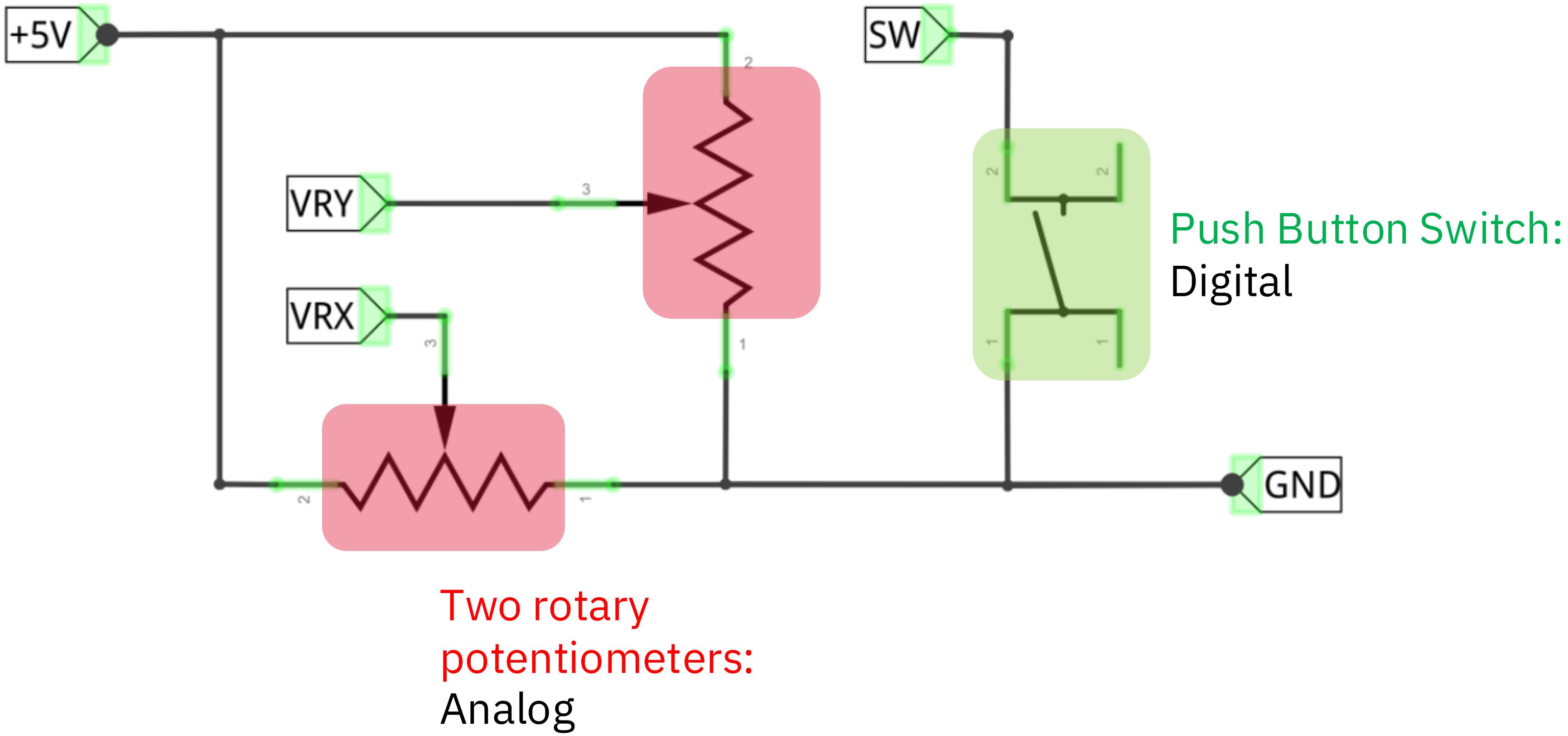
- Build the Circuit



Double-check for the
correct orientation.

Lab 4: Joystick Space Navigation

- For more details on this circuit, please refer to Tutorial Chapter 12 (available on E-Class).



Lab 4: Joystick Space Navigation

- **Task:**
  - Build the circuit
  - Double-check wiring before running code.
- **Run the starter joystick code**
  - Make sure you can read X, Y, Z values.
- **Follow the steps**
  - Add each feature step by step until the ship can reach the target!
  - Print Reached the target and exit the loop and finish program
- **Tune the parameters**

Inputs & Constants

- **Given:**
  - **Joysteak reading from ADC:** Val_x, Val_y (0–255),
  - **User Inputs:** Starting point: (Pos_x, Pos_y)
  - **User inputs:** Target location: $(target_x\ , target_y)$
- **Constants that needs to be tuned:**
  - CENTER = 128,
  - SCALE = 0.05, **(Tune it)**
  - TOL = 3.0. **(Tune it)**
- **Bonus:**
  - DEADZONE = 10, **(Tune it)**
  - Factor = 1.5 **(Tune it)**

Steps

Step 1 — Center the joystick readings

$$dx = Val_x - 128$$

$$dy = Val_y - 128$$

Explanation

- The **ADS7830** is an **8-bit ADC**. $(2^8 = 256)$
- **Joystick → ADC → re-center (code)**
- $\begin{cases} 0\text{ V} & \rightarrow & 0 & \rightarrow & -127 & \text{(Left)} \\ 2.5\text{ V} & \rightarrow & 127 & \rightarrow & 0 & \text{(Center)} \\ 5\text{ V} & \rightarrow & 255 & \rightarrow & +127 & \text{(Right)} \end{cases}$
- This makes the math **symmetric around zero**, so left/right and up/down are treated equally.

Steps

Step 2 — Normalize to a unit range

$$nx = \frac{dx}{127} \times \ Scale$$

$$ny = \frac{dy}{127} \times \ Scale$$

Explanation

- Converts joystick to **−1 … +1** scale.
- Makes code hardware-agnostic (same math if ADC changes).
- SCALE is your **speed knob** (how far you move each loop).

Steps

- **Step 3 — Simple step-based:**

$$Pos_x = Pos_x + nx$$

$$Pos_y = Pos_y + ny$$

Explanation

- Each loop, the new position is just the old position plus the joystick offset.
- nx is the joystick's raw displacement from the center.
  - Negative → move left
  - Positive → move right

Steps

Step 4 — Check arrival to the planet (no sqrt)

$$d^2 = (target_x - Pos_x)^2 + (target_y - Pos_y)^2$$
$$d^2 \leq TOL^2$$

Explanation

- TOL: tolerance is how close you must be to count as "reached."

Steps (Bonus)

- **(Optional) Press Z button to add extra thrust.**

If button pressed:

$$nx = 1.5 \ * nx$$

$$ny = 1.5 \ * ny$$

Explanation

- Easy extension for engagement: a **thrust boost** when Z is pressed.
- Keep factor small (e.g., 1.2–2.0) for control.

Steps (Bonus)

- **(Optional but recommended) Apply a dead zone**

$$dx = \begin{cases} 0 & if\ |dx| < Deadzone \\ dx & Othersie \end{cases}$$

$$dy = \begin{cases} 0 & if\ |dy| < Deadzone \\ dy & Othersie \end{cases}$$

Explanation

- Real joysticks “wiggle” near center.
- Deadzone prevents **unintended drift** when sticks are released.

Part 4 Report Format (short, personal, verifiable)

Setup
- Attach a clear photo of your circuit (showing the joystick connected to ADC and Raspberry Pi).
- Copy and paste your final code (main program).
- Include meaningful comments in your code explaining each part.

Demo
- Ask your TA or instructor to check your program running on the hardware.

Observations
- Describe how joystick tilt (X/Y) changed your ship's movement.
- How did adjusting SCALE or TOL affect control and arrival?
- Did the ship drift when you released the joystick? Why?

Analysis
- Explain in your own words:
- What is the **starting point** and how is it set?
- What is the **target location** and how do you check arrival?
- Why do we use TOL (tolerance) instead of requiring the exact coordinates?
- What would happen if you didn't use scaling and just added the raw joystick values to the position?

Bonus (Optional)
- Add a **deadzone** to remove drift when the joystick is released.
- Use the joystick **button (Z)** as a "boost" or "reset target."
- Experiment with different tolerance values or speeds. Describe how it changes the landing challenge.

Rubric

- **Circuit Setup** – 2 pts
  - safe connections.
- **Code and Demo** – 4 pts
  - Code runs correctly with meaningful comments – 1 pt
  - Show spaceship moving with joystick and reaching target planet (live demo to TA/instructor) – 3 pts
- **Observations** – 2 pts
- **Analysis** – 2 pts
- **Bonus (optional, up to +2 pts)**
  - Add **deadzone** to fix drift – +1 pt
  - Use joystick button (Z) as "boost" or "reset target" – +1 pt