Robotics Architectures and Modeling Assignment

Instructions

- Read all questions (Q1–Q4) carefully before attempting.
- Provide answers in structured format with diagrams where required.
- Maintain clarity and brevity (avoid unnecessary length).
- Submit written content as a single compiled PDF. [Report.pdf]
- Include everything in a single zip file. [RollNo_Name_Assignment1.zip]

Q1 — Architecture Trade-off Analysis (SPA vs Reactive vs Hybrid) [15 Marks]

Scenario Framing (choose ONE)

- Hospital delivery robot (meds from pharmacy → ward)
- Domestic assist robot (fetching objects in a cluttered apartment)
- Office courier robot (mail delivery across floors with elevators & moving people)

Tasks

1. **Short scenario framing ($\frac{1}{4}$–$\frac{1}{2}$ page)** [2 Marks]
   - Define the mission, constraints (map, humans, dynamics, safety).
   - State what counts as "success."
2. **Architecture sketches (1 page total)** [5 Marks]
   - SPA: draw a simple activity or sequence diagram.
   - Reactive: draw a state machine sketch.
   - Hybrid: draw a layered/combined diagram showing mode switching. (Include 1–2 sentences of explanation per architecture.)
3. **Evaluation matrix ($\frac{1}{2}$ page)** [4 Marks]
   - Use 4–5 criteria only (e.g., robustness, optimality, response time, safety, ease of ROS integration).
   - Give scores (1–5) and 1–2 sentence justifications.
4. **Decision & rationale ($\frac{1}{2}$ page)** [4 Marks]
   - State which architecture fits your chosen scenario best.
   - Mention one strength and one limitation you accept.

Q2 — Drawing a Picture with a Turtlebot [30 Marks]

Problem Definition

Design and implement a program that will cause a TurtleBot in the ROS `turtlesim` simulator to draw a picture of the front elevation of a house, comprising:

- Roof
- Front wall
- One window
- One door

The house should have the dimensions shown in Figure 1a. The window and door should be placed exactly as shown. All dimensions are specified in the units of the ROS `turtlesim` simulator. The bottom-left corner of the house should be positioned at the coordinates specified in an input file.

The picture must be drawn by controlling the turtle's movement (no teleportation). However, teleportation may be used to (re-)position the robot prior to drawing each element of the house.

Input File

The program should read an input file `assignment1Input.txt`.

This file contains a sequence of pairs of coordinates (x, y), one pair per line, specifying the position of the bottom-left corner of the house.

The `turtlesim` environment should be reinitialized after drawing each picture, erasing the current drawing, but only after the user is prompted to continue to the next picture (so they can inspect the output before proceeding).

Input Format

- A sequence of lines, each comprising a pair of coordinates specifying the position at which the bottom-left-hand corner of the house should be drawn.

Output

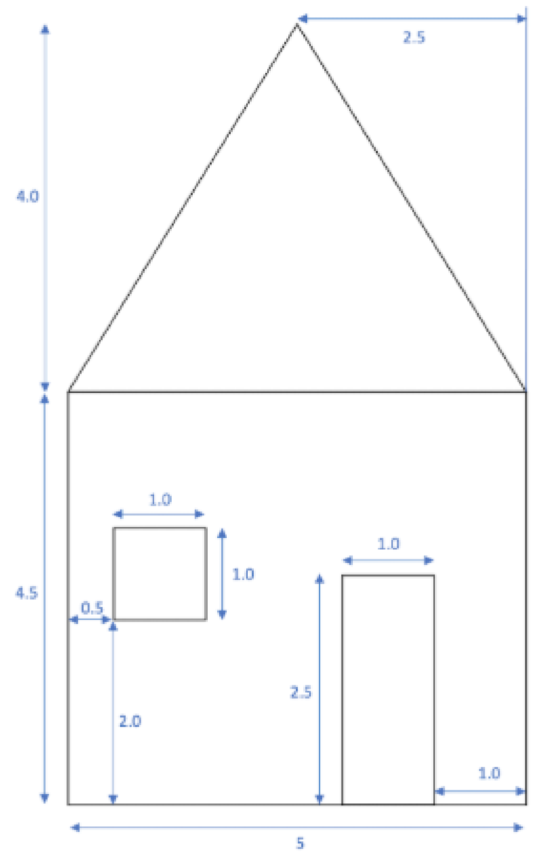
The TurtleBot should draw a sequence of pictures of a house at the locations specified in the input file.

Sample Input

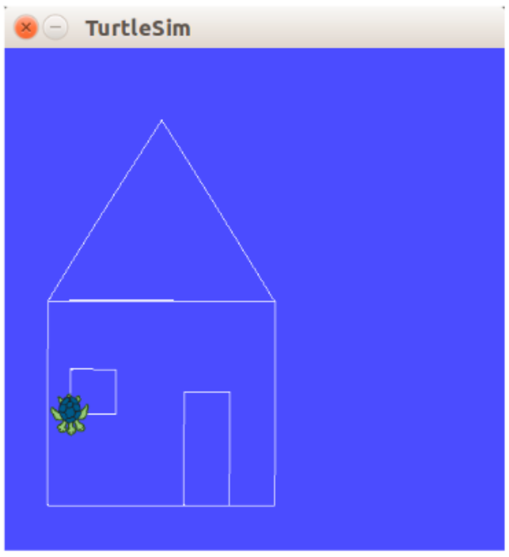
```
1.0 1.0
3.0 2.0
```

Sample Output

A picture of the house drawn by a TurtleBot at location (1.0, 1.0). See Figure 1b.



(a) Dimensions of the house.



(b) Example of house drawn by TurtleBot at location (1.0, 1.0).

Figure 1: House specification (left) and TurtleBot drawing (right).

Q3 — Coding Assignment: Navigation Architectures (SPA vs Reactive vs Hybrid) [30 Marks] (2 world x 3 architectures)

Goal

Compare three classical robot control architectures by implementing them for TurtleBot navigation in Gazebo worlds:

- **SPA (Sense–Plan–Act):**
  - Compute a global plan to the goal once at the start.
  - Execute without replanning, even if blocked.
- **Reactive:**
  - Use sensor data only (LiDAR).
  - Implement a local obstacle avoidance strategy (e.g., follow-the-gap, Braitenberg).
  - No global map or preplanned path.
- **Hybrid:**
  - Start with a global plan.
  - If progress stalls (e.g., robot stuck for >5s), switch into reactive mode.
  - After clearing the obstacle, replan and resume.

Worlds

Out of the possible worlds, we can have these two predefined ones:

- `turtlebot3_stage_1.world` as your **Corridor World (W1)**.
- `turtlebot3_house.world` as your **Apartment World (W2)**.

Both are sized for TurtleBot3 Burger (0.7m doors, 11×11 world).

Tasks

1. Implement three separate nodes inside a package `ri_nav_compare/`:
   - `spa_nav.py`, `reactive_nav.py`, `hybrid_nav.py`
2. Configure launch files to run each architecture in each world with a fixed start + goal pose.
3. Conduct 6 runs total:
   - 2 worlds × 3 architectures.
4. Collect metrics for each run:
   - **Success (0/1):** robot within 0.3 m of goal.
   - **Time (sec):** from start to goal or timeout (max 180s).
   - **Near-collisions (#):** number of frames where `min(/scan)` $<$ 0.22m.
   - **Path length (m):** estimated from odometry.

Deliverables

Submit a zip archive with the following structure:

```
ri_nav_compare_<firstname>.zip
  |-- src/      spa_nav.py, reactive_nav.py, hybrid_nav.py
  |-- launch/   spa.launch.py, reactive.launch.py, hybrid.launch.py
  |-- worlds/   corridor.world, apartment.world
  |-- results.csv
  \-- REPORT.md
```

Results

`results.csv` should contain rows with:

```
world, approach, trial, success, time_sec, path_length_m, near_collisions
```

`REPORT.md` ($\leq$ 1 page) should include:

- One comparison table (averaged metrics per architecture × world).
- 3–4 bullet points describing trade-offs (robustness vs efficiency).
- A paragraph: "Which architecture is best suited for W1 vs W2, and why?"

Hints / Constraints

- Nav2 may be reused for SPA & Hybrid planning.
- For Reactive: implement a basic gap-following controller.
- Hybrid = SPA + a timeout monitor: if no progress in 5s → cancel goal → run Reactive for 5s → replan.
- Keep it simple: evaluation is based on comparison & reflection, not perfect performance.

Q4 — Coding/Modeling: UML for the Robot Waiter [25 Marks]

Scenario

You are designing a Robot Waiter that serves water in a café. Your task is to model this system using UML diagrams to capture different perspectives of its operation.

Tasks

1. Use Case Diagram

- Identify the actors (e.g., customer, robot waiter, barista, environment).
- Show the main tasks/interactions (e.g., take order, fetch water, deliver to table, avoid obstacles).

2. Activity Diagram

- Model the workflow of the serving process.
- Include decision points (e.g., obstacle detected → avoid vs. ask for help).
- Show concurrency where relevant (e.g., sensing and navigation running in parallel).

3. State Machine Diagram

- Capture the robot waiter's operational modes (e.g., Idle → Take Order → Navigate → Deliver → Return).
- Include transitions triggered by events (e.g., "order received", "obstacle detected", "battery low").

Extension Task

Go beyond standard UML:

- Propose one novel UML extension (e.g., a stereotype, tag, or constraint) that captures a cognitive robotics aspect such as:
  - Attention focus
  - Uncertainty
  - Learning/adaptation
- Clearly justify why this extension is useful (what it adds beyond standard UML).
- Show a sketched example of how it would be applied in one of your diagrams.

Submission

- Three UML diagrams (Use Case, Activity, State Machine) in standard notation.
- One additional diagram (or annotated version) demonstrating your proposed extension.
- A short written justification (5–8 sentences) explaining the extension's purpose and benefit.
