

Project 2: Robot Wall Following by Reinforcement Learning

Assigned: February 27, 2024

Deliverable 1 (Problem Formulation) due: March 06, 23:59:59

Deliverable 2 (Reinforcement Learning) due: March 15, 23:59:59

Deliverable 3 (Project Report) due: March 27, 23:59:59

Note: Project 2 has a four-day late submission period for each deliverable.
Late submissions will lose 25% points per day of that specific deliverable.

In the project, you will teach an autonomous wheeled robot to follow walls and avoid running into obstacles by designing a representation of the environment from sensor observations and implementing a reinforcement learning algorithm. You will use the Gazebo simulation in ROS to simulate an omnidirectional robot named Triton, and use an environment map that is provided to you for training and testing the robot. You will use a 2D LiDAR on the robot to perform sensing, and the robot is controlled using steering and velocity commands. Students are required to program this project using Python in ROS Noetic running on Ubuntu 20.04 LTS (i.e., the same development environment used in Project 1). In addition, students must write a report following the format of standard IEEE robotics conferences using \LaTeX .

Please **START EARLY!**

1 The Wall Following Problem

Wall following is a common strategy used for navigating an unfamiliar environment. This capability allows an autonomous robot to navigate through open areas until it encounters a wall, and then to navigate along the wall with a fixed distance. The successful wall follower should traverse all circumferences of its environment at least one time without getting either too close or too far from the walls, or running into obstacles. Several methods were used to address this problem, such as control-rule based methods and genetic programming. In this project, students are required to solve the problem by defining a representation of the environment and implementing a reinforcement learning method.

In this project, we use the following informal definitions: following a wall means that at time t , the moving robot is at a desired distance d_w from the wall, and that this distance is maintained at time $t + 1$. During wall following, the robot must be moving. A robot that maintains a distance d_w from the wall at time t and $t + 1$, but is not moving, does not fit the definition of following a wall. For this project, students will define the value of d_w (e.g., 0.75m). For the purposes of this project, we will consider a robot that covers longer distances while following a wall to be better than a robot that only follows a wall very slowly (Note that we didn't just say we prefer a robot that covers longer distances; The robot should only be rewarded if it covers longer distances while also following the wall). Note also that the robot can follow a wall from either side (right or left), except that once it is following a wall on one side (say, the right), it should continue to follow the wall on that side until the wall is lost. If you prefer, you may restrict the definition to be only right-side wall following or only left-side wall following (meaning that the robot only follows walls on one side, and never on another), if this makes your robot learn faster or better. All these issues may affect your reward function.

2 Gazebo Simulation of Triton in ROS Noetic

This project will use a simulated Triton robot as illustrated in Fig. 1. The Gazebo simulator of Triton can be set up by following instructions in the GitLab repository:

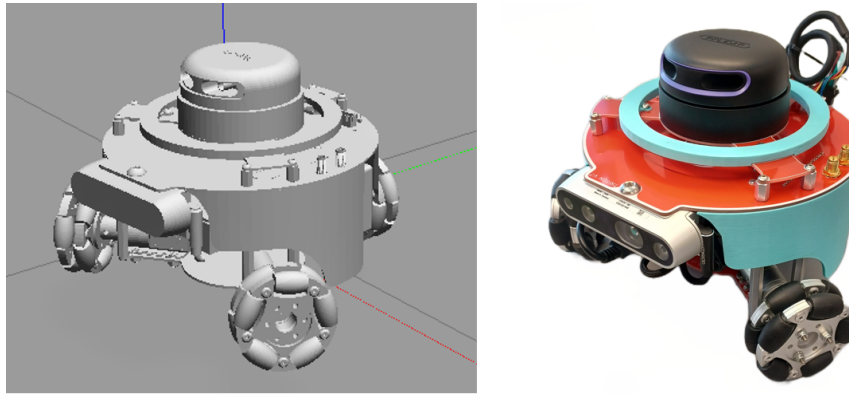


Figure 1: The Triton robot and its simulation in Gazebo.

https://gitlab.com/HCRLab/stingray-robotics/stingray_sim.

The Gazebo simulation allows for visualizing the Triton robot and the world, controlling the robot via ROS topics and services, and visualizing robot sensing and navigation path in RViz. Students need to follow the instructions in the repo or watch the instruction video, and become familiar with basic functions before implementing your package.

The default simulation environment can be launched via:

```
$ wall_following.launch
```

which is available in the directory: *stingray_sim/launch/*.

After launching, you will see a simulation environment that is similar to Fig 2. To experiment with other maps, you need to place a new world file under the *worlds/* directory in the *stingray_sim* package and call `roslaunch` with a *world_file* argument.

The robot movement is controlled using the ROS topic `/cmd_vel`. This topic accepts a message of type `geometry_msgs/Twist`, which is defined as:

https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html

The LiDAR readings of the robot is obtained by listening on the topic of `/scan`, which uses the message of type `sensor_msgs/LaserScan`. The structure of this message can be found at:

https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html

Communication services between ROS and Gazebo are necessary to let the robot repeatedly explore the environment. You can find all available services while the simulation environment is launched by typing the command `rosservice list` in a new terminal. You are likely to use the following services in this project:

- `/gazebo/reset_simulation`
- `/gazebo/get_model_state`
- `/gazebo/set_model_state`
- `/gazebo/unpause_physics`
- `/gazebo/pause_physics`

Explanation of each service can be found in the following link. Examples of calling services in Python are presented in ROS Tutorial 1.1.15.

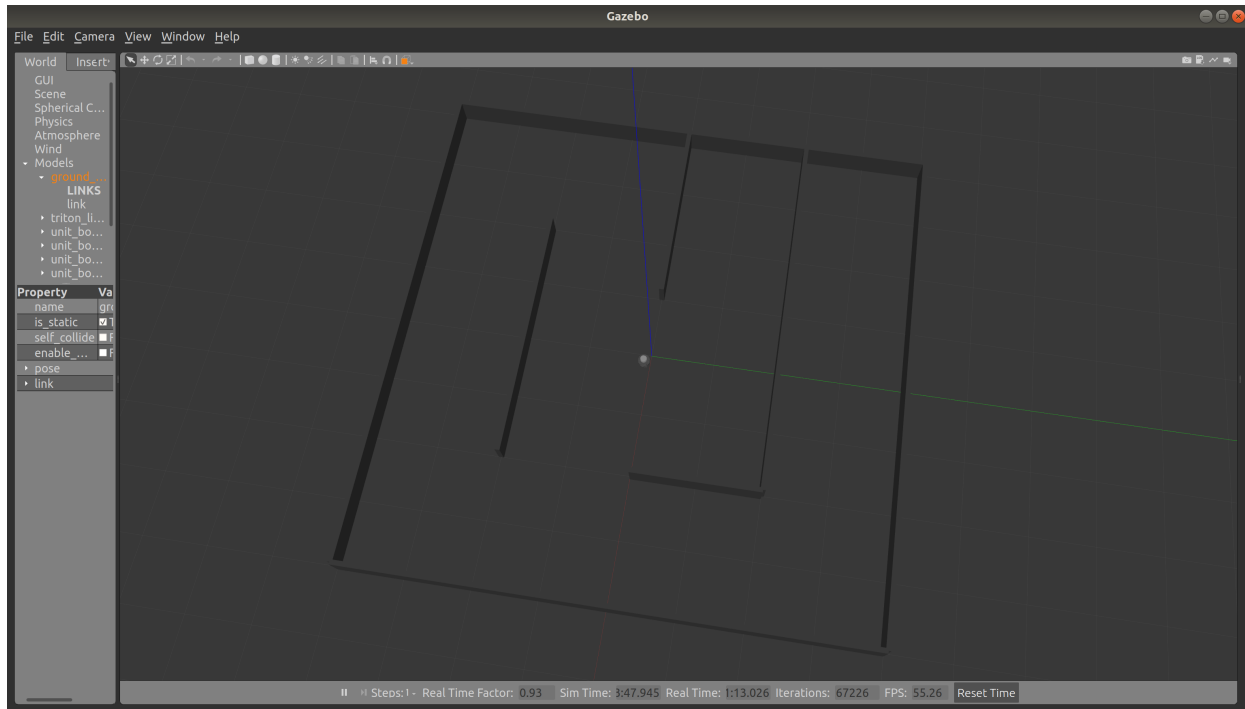


Figure 2: Gazebo simulation of the Triton robot and the experiment environment for wall following.

https://classic.gazebosim.org/tutorials?tut=ros_comm.

3 Deliverable 1: Problem Formulation and Representation

As part of the project, students are responsible for formulating the wall following problem. Specifically in this part of the project, students will need to determine how to represent the input state (based on the input LiDAR data) and the motor actions, and the best level of discretization of these values. Please note that you are NOT allowed to use the grid-world representation of states. States should be a function of sensor values (and other measurements, as appropriate). As always, there is not just one single way to define states and actions. Explore various design possibilities and find the one that you believe works best, or survey the literature on this topic.

Based upon the discretized states and actions, students are required to implement a Q-table and manually set the values of the Q-table to determine the policy. The manually selected Q-values will be applied for two purposes:

- Allowing the robot to follow a straight wall using these manually defined Q-table as the policy.
- Using these manually defined Q-values to encode expert knowledge to facilitate Q-learning in Deliverable 2.

In Deliverable 1, students are required to demonstrate that the manually defined Q-values as the policy enable the robot to follow a straight wall. Accordingly, the ROS package you submit in Deliverable 1 should allow a user (e.g., the grader, TA or instructor) to test the policy manually defined in the Q-table for the Triton robot to follow a straight wall in Gazebo. This means that (1) your submitted ROS package in Deliverable 1 should include your manually defined Q-table, and (2) a launch file designed to start the whole simulation to show the capability of robot following a straight wall in Gazebo.

What to Submit for Deliverable 1:

Students must submit a single tarball, named *P2D1_firstname_lastname.tar* (or *.tar.gz*), to the portal named P2-D1 in Canvas. The tarball must contain the following **two items**:

- Your **ROS package** (that must include all necessary package files, including Python scripts, launch files, *package.xml*, *CMakeLists.txt*, world files, etc.) to demonstrate the robot capability of following a straight wall. The launch file must automatically start your demonstration in Gazebo (as illustrated in Fig. 2). The *package.xml* file must provide sufficient information of your submitted package, and clearly describe how to use the launch file to run the demonstration.
- A short **demo video** showing that the robot successfully follows a straight wall (no need to implement or show the capability of turning around wall corners; following a *straight* wall is sufficient for this deliverable). You can either submit a video or provide a link to the video (e.g., on YouTube). If you are submitting a video, make sure the video size is less than 20M. You can use *ffmpeg* to speed up the video in Ubuntu. If you choose to provide a link, you are responsible to ensure that the video link allows public access.

4 Deliverable 2: Reinforcement Learning

In this deliverable, students will implement reinforcement learning algorithms. Before starting to code your algorithms, you need to read the relevant lecture slides or the reinforcement learning textbook to understand them. As it may take a while for the reinforcement learning algorithms to converge, please start early!

We discussed two classic reinforcement learning algorithms, including Q-learning (Fig. 3) and SARSA (Fig. 4). This Deliverable 2 mainly focuses on expanding the ROS package you already developed in Deliverable 1 and implementing the reinforcement learning algorithms to update the Q-values through learning (instead of using manually defined values). In particular, your reinforcement learning algorithms should satisfy the following requirements:

- Temporal Difference (TD) with a pre-defined learning rate (i.e., *StepSize*) must be implemented to update the Q-values in an incremental and iterative fashion.
- ϵ -greedy policy must be implemented with a pre-defined ϵ value to balance exploration and exploitation.

These requirements are already included in the provided Q-learning and SARSA algorithms. You may use your manually defined Q-values in Deliverable 1 as the initialization, or set all Q-values to 0 in order to let the algorithms to learn from scratch without prior knowledge.

As part of the deliverable, you are responsible for fine-tuning how you will represent the input state and the motor actions, and the best level of discretization of these values. In addition, you will need to decide the reward function you'll use for learning; presumably the robot will have a positive reward for following a wall and a negative reward for running into a wall. The robot may also have a component of the reward function that rewards moving longer distances along a wall, rather than have the robot creep along the wall. You may also define your reward function any way you like, including the use of scaled or graduated rewards as a robot moves toward or away from a wall, if you find such rewards helpful. Moreover, you can decide yourself where the starting position of the robot will be for each learning episode. You'll need to determine how many episodes are needed for your program to learn.

Q-learning (off-policy TD control)

```
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
```

Figure 3: The Q-learning algorithm.

Sarsa (on-policy TD control)

```
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A'$ 
  until  $S$  is terminal
```

Figure 4: The SARSA algorithm.

What to Submit for Deliverable 2

Students are required to implement and submit both Q-Learning (Fig. 3) and SARSA (Fig. 4), and compare them. For Deliverable 2, you must submit a single tarball, named *P2D2_firstname_lastname.tar* (or .tar.gz) to the Canvas portal named P2-D2. The tarball file must contain the following **three items**:

- Your **ROS package** (including Python scripts, launch files, package.xml, CMakeLists.txt, world files, etc.) that implements both Q-learning and SARSA. Your ROS package should include TWO options for each algorithm – one option is the reinforcement learning algorithm in learning mode, and the second option is to test the best learned policy for an arbitrary robot starting position. This means that your package submission should include the best policy you found during your own training, so that you can use those during testing mode.

Your package should allow the user to specify:

1. which algorithm to use (e.g., Q-learning or SARSA), and
2. whether the code should be run in the training mode or the testing mode.

It is up to you to design the interface to allow the user to specify which mode the code should be in, e.g., by implementing multiple launch files or by implementing one launch file with arguments. The package.xml file must provide sufficient information of your package, and clearly describe how to use the launch file(s) to run the two algorithms in your package in either training or testing mode.

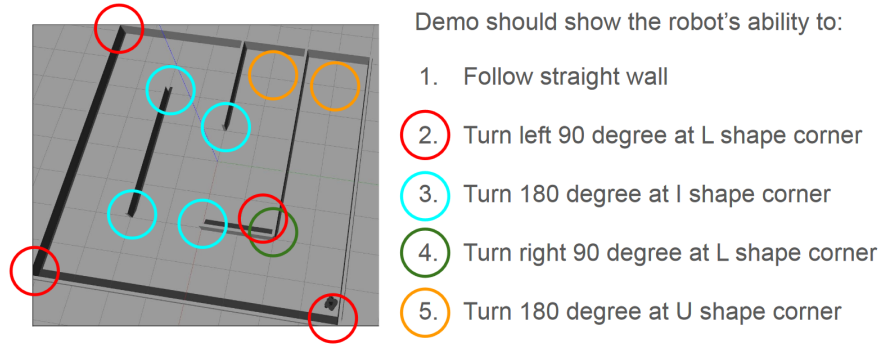


Figure 5: The five scenarios in which the robot should be able to follow the wall using the learned policy.

- A **demo video** showing that your policy learned using your Q-learning algorithm enables the robot to successfully follow the wall in ALL FIVE scenarios defined in Fig. 5. You can either submit a video or provide a link to the video. If you submit a video, make sure the video size is less than 20M. You can use *ffmpeg* to speed up the video in Ubuntu. If you choose to provide a link, you are responsible to ensure that the video link allows public access. Please clearly label the algorithm that is demonstrated in the submitted video.
- A **demo video** showing that your SARSA policy learned using your algorithm enables the robot to successfully follow the wall in ALL FIVE scenarios defined in Fig. 5. You can either submit a video or provide a link to the video. If you submit a video, make sure the video size is less than 20M. If you choose to provide a link, you are responsible to ensure that the video link allows public access. Please clearly label the algorithm that is demonstrated in the submitted video.

In addition, during your implementation and testing, you are suggested to collect the following information to compare both algorithms (that will need to be analyzed and compared in your report in Deliverable 3):

- Rate of convergence of learning (e.g., in terms of the number of episodes).
- Effect of different reward assignments on the quality and speed of learning.
- Change of the accumulated reward as the number of episodes increases.

In addition, you may analyze the effect of the layout of the learning environment (if you want to try different environments), or the effect of different state and motor output representations (e.g., using more or less resolution) on the quality and speed of learning. You do not need to submit the analysis results in Deliverable 2, but the information will be used in Deliverable 3 when you write your project report.

5 Deliverable 3: Project Report

As the last deliverable of the project, you must prepare a single 4-6 page project report/paper (in pdf format, using \LaTeX and the 2-column IEEE style for robotics conferences, which is similar to this project write-up). You can use the \LaTeX template *ieeeconf.zip* at: <https://ras.papercept.net/conferences/support/tex.php>.

Design: Your report should discuss the details of both Q-learning and SARSA algorithms that you designed. For each learning algorithm, you need to discuss:

- How the sensing field of view is divided?
- How state, action, and reward are defined?
- How your Q-table is defined and initialized?
- What are the values of model parameters, including ϵ , α , and γ used in your Q-learning implementation?

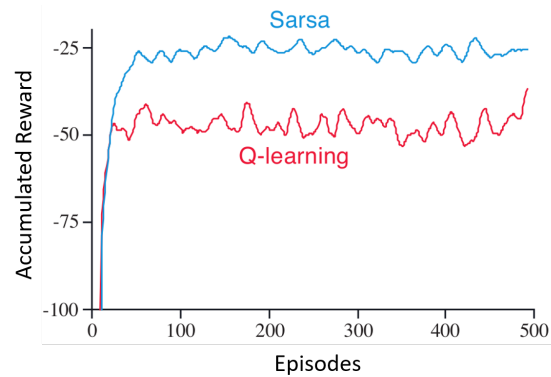


Figure 6: An example (and just an example) of quantitative results, using accumulated reward as the performance measure. Your results may be noisier and/or take a larger number of episodes for the learning methods to converge.

Experiment: Your report needs to include the following experimental results:

- Your report should include experimental results to make it clear that the robot has in fact learned. Results need to be quantitative, but not qualitative. This means that you have some concrete measure (e.g. accumulated reward, as shown in the example in Fig. 6) by which to evaluate your results. You may depict the performance of Q-learning and SARSA based upon your measure in the same figure to compare both methods.
- Your report should include qualitative results (e.g., snapshots from your demo video) to demonstrate that your learning methods enable robots to address the five situations of wall following described in Fig. 5.
- Your report should compare Q-learning and SARSA. For example, how many episodes were used in training by Q-learning and SARSA to obtain a good solution? How much time did the training take for each of the learning methods?

Organization: Your paper should includes the following:

- An abstract containing 200 to 300 words to summarize your findings.
- A brief introduction describing the robot wall following task, and describing your RL solutions at the high-level.
- An approach section that describe your implementations of Q-learning and SARSA. You may include answers to the **Design** questions in this section.
- An explanation of the experimental results. Include all experimental results required by the **Experiment** section. Use figures, graphs, and tables where appropriate to present quantitative and qualitative results.
- A conclusion section to discuss the significance of the results, any insightful observations, and future work that you believe would improve the learning.

The reader of your paper must be able to understand what you have done and what your learning methods do without looking at the code itself. The paper should have the “look and feel” of a technical conference paper, with logical flow, good grammar, sound arguments, and illustrative figures.

What to Submit:

The project paper you turn in must be in the pdf format in a single file generated using L^AT_EX. Your report must address all requirements described in this subsection. Name your paper *P2D3_firstname_lastname.pdf* and submit it to the Canvas portal named P2-D3.

6 Grading

Your grade will be based upon the quality of your package implementations, the demo demonstrations, and the documentation of your findings in the report. Project 2 will be graded as follows (for a total of 10 points):

- Deliverable 1: 2.5/10 – You should have a working implementation of the ROS package, meaning that your code is implemented as a ROS package, your code runs without crashing in ROS and Gazebo, and performs robot wall following using manually defined Q-values. You must also submit a video demo to demonstrate that the robot is able to follow a straight wall.
- Deliverable 2: 5/10 – You should have a working implementation of the required reinforcement learning algorithms, meaning that your code of the required algorithms is implemented as a ROS package, runs without crashing in ROS and Gazebo, learns Q-values in the learning mode, and performs robot wall following in ALL FIVE scenarios using your learned Q-values in the testing mode. You should also create and submit a video demo for each of the required algorithms to demonstrate that the robot is able to follow a wall in all five scenarios defined in Fig. 5 using the policy learned by the respective algorithm.
- Project report/paper: 2.5/10 – All required contents must be included in the report. The report must be prepared in \LaTeX using the IEEE robotics conference styling and submitted in the pdf format. Figures and graphs should be clear and readable, with axes labeled and captions that describe what each figure and graph illustrates.