CS 489/689 Autonomous Racing: Lab 3 Follow the Gap

Section 1001, Fall 2023

1 Lab Description

In this lab, groups will learn to use (1) reactive methods to follow the gap.

2 Learning Outcomes

Outcome 1 Create a reactive gap following algorithm

3 Rubric

Simulator: Complete One Lap of `levine_blocked`	10
Simulator: Complete One Lap of `levine_obs`	10
On Vehicle: Complete One Lap of TBE-B Second Floor	50
Find-Max Gap Implementation	5
Find Best Point Implementation	5
Project Archive	20
Total	100

4 Submission Instructions

A completed project is first demonstrated to the teaching assistant in both the simulator and running on the vehicle.

Project demonstrations will be held during the TA office hours on September 23rd. They include implementations demonstrated in simulator and on vehicle. In the simulator, the vehicle will be tested in two maps provided in the `gap_follow` node: `levine_blocked` and `levine_obs`. The vehicle will first be set along the track on the *right side* of the `levine_blocked` map with the vehicle facing North. Once the vehicle has successfully completed a lap in `levine_blocked`, the map will be swapped with `levine_obs` and demonstrated. See Figure 1 below for an idea of the expected start pose. The `gap_follow` node, once started, should display a start message to terminal while the vehicle drives a complete lap around both map hallways.

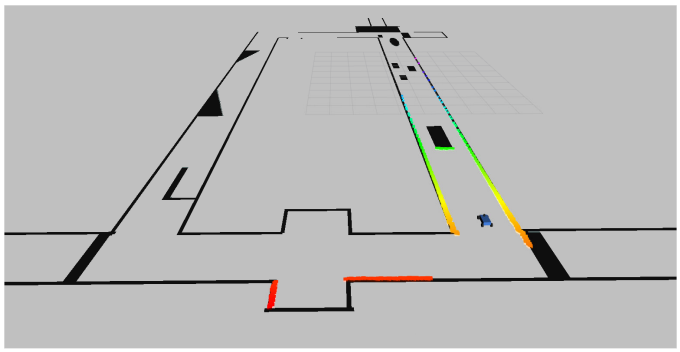

Figure 1: Starting Pose of the Vehicle for Simulation Enviornment

Be sure to implement the necessary changes to `reactive_node.py` to make it integrate with the vehicle before submission, these are listed in Section 8.0.2. It is expected for students to have tested their code on vehicle, before submission, to ensure it runs properly.

Submissions of the `gap_follow` package to canvas will be accepted after a successful demonstration. Within this package, should exist 2 python scripts. `reactive_node.py` and `reactive_node_vehicle.py`, for the simulator for the vehicle respectively.

5 Lab Description

In this lab, students will implement a reactive follow the gap algorithm. This will allow the vehicle to react to static and dynamic obstacles on the track. Students are encouraged to try and implement different reactive methods or a combination of several, but the python script lays out the foundation for a generic implementation. The accompanying slide deck, Day-03-Part-01-Reactive-Methods-Follow-the-Gap.pptx, and script outline provide hints for a more competitive implementation.

5.1 Follow the Gap

Naively, a gap following implementation seeks to find the largest gap between obstacles and send the vehicle towards the farthest point away from the vehicle that lies within that gap. This, however, does not account for safety. If the vehicle's geometry is not considered collisions are possible.

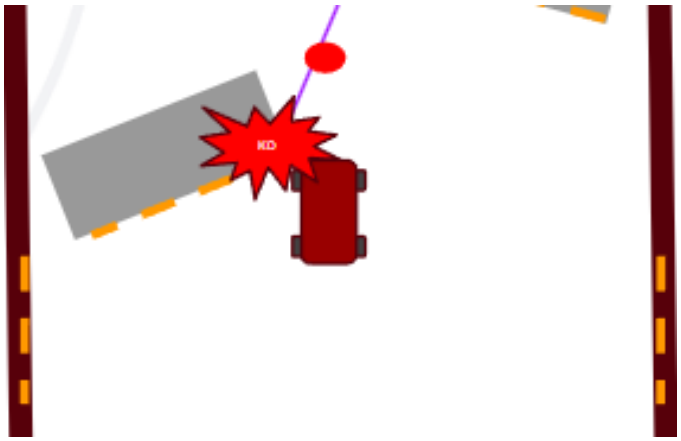

Figure 2: Vehicle collision due to following the farthest point (red circle) laying too close to an obstacle

To account for the vehicle's dimensions, it will be projected onto the obstacle at the closest point to the vehicle. The radius of the vehicle will need to be taken and saved, and a circle projected as in Figure 3 below. All points within this "safety bubble" will be considered as the obstacle, and set to 0. All other non-zero points are considered our "gaps".

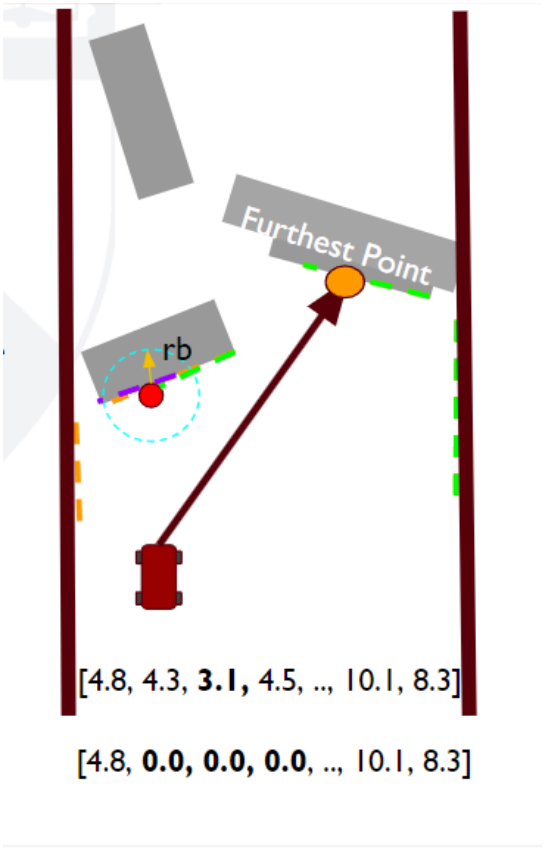

Figure 3: Projection of the vehicle radius onto the closest point

The `steering_angle` will be set towards the farthest point.

6 Gap Follow

This subsection is based upon the F1TENTH Lab on Follow the Gap: https://github.com/f1tenth/f1tenth_labs_openrepo/tree/main/f1tenth_lab4. Begin by cloning the F1TENTH Open Repository outside of the F1TENTH Gym container directory created earlier.

```
gordon@f1sim:~$ cd ws/gym-one/
gordon@f1sim:~/ws/gym-one$ cp -r ~/git/f1tenth_labs_openrepo/f1tenth_lab4/gap_follow .
gordon@f1sim:~/ws/gym-one$ ls
f1tenth_gym_ros  safety_node  wall_follow  gap_follow
```

Stop the docker composition and edit the `docker-compose.yaml` file of the composition, include a volume for the newly copied `gap_follow` directory.

```
gordon@f1sim:~/ws/gym-one/f1tenth_gym_ros$ emacs -nw docker-compose.yml
>>> SNIP <<<
  volumes:
    - .:/sim_ws/src/f1tenth_gym_ros
    # Add the next entry
    - /home/gordon/ws/gym-one/gap_follow/:/sim_ws/src/gap_follow
>>> SNIP <<<
```

Restart the docker and re-enter the simulator container before building the `gap_follow` package.

6.0.1 Subscribers and Publishers

Note the following topic names for your publishers and subscribers:

`LaserScan: /scan`
`AckermannDriveStamped: /drive` (`drive.steering_angle` and `drive.speed`)

6.0.2 Reactive Node Template

```python
import rclpy
from rclpy.node import Node

import numpy as np
from sensor_msgs.msg import LaserScan
from ackermann_msgs.msg import AckermannDriveStamped, AckermannDrive

class ReactiveFollowGap(Node):
    """
    Implement Wall Following on the car
    This is just a template, you are free to implement your own node!
    """
    def __init__(self):
        super().__init__('reactive_node')
        # Topics & Subs, Pubs
        lidarscan_topic = '/scan'
        drive_topic = '/drive'

        # TODO: Subscribe to LIDAR
        # TODO: Publish to drive

    def preprocess_lidar(self, ranges):
        """ Preprocess the LiDAR scan array. Expert implementation includes:
            1.Setting each value to the mean over some window
            2.Rejecting high values (eg. > 3m)
        """
        proc_ranges = ranges
        return proc_ranges

    def find_max_gap(self, free_space_ranges):
        """ Return the start index & end index of the max gap in free_space_ranges
        """
        return None

    def find_best_point(self, start_i, end_i, ranges):
        """Start_i & end_i are start and end indicies of max-gap range, respectively
        Return index of best point in ranges
            Naive: Choose the furthest point within ranges and go there
        """
        return None

    def lidar_callback(self, data):
        """ Process each LiDAR scan as per the Follow Gap algorithm & publish an AckermannDriveStamped Message
        """
        ranges = data.ranges
        proc_ranges = self.preprocess_lidar(ranges)

        # TODO:
        #Find closest point to LiDAR

        #Eliminate all points inside 'bubble' (set them to zero)

        #Find max length gap

        #Find the best point in the gap

        #Publish Drive message


def main(args=None):
    rclpy.init(args=args)
    print("WallFollow Initialized")
    reactive_node = ReactiveFollowGap()
    rclpy.spin(reactive_node)

```

```
    reactive_node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

scripts/reactive_node.py

7 Changing the Simulator Map

Located within the downloaded `gap_follow` node are two new maps, `levine_blocked` and `levine_obs`. These maps, shown below, will be used to test the `gap_follow` node.

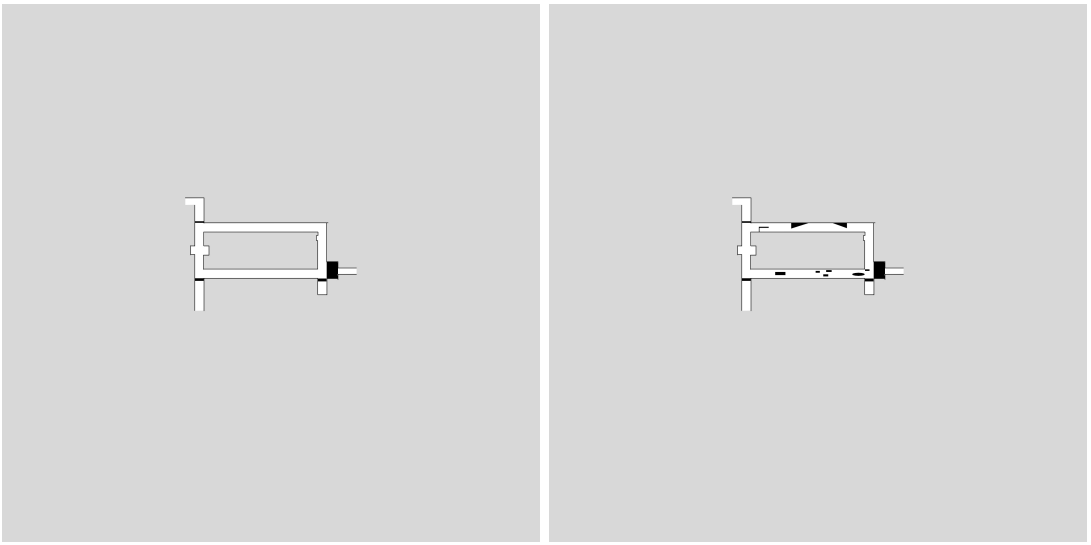

Figure 4: New Simulator Maps- `levine_blocked`(*left*) and `levine_obs`(*right*)

To change to one of these two maps, locate the desired `.png` and `.yaml` map files and add them to the `f1tenth_gym_ros/maps` directory. Once copied over, the directory should be populated as below.

```
gordon@f1sim:~/ws/gym-one/f1tenth_gym_ros/maps$ ls
levine_blocked.png   levine_obs.png   levine.png   Spielberg_map.png
levine_blocked.yaml  levine_obs.yaml  levine.yaml  Spielberg_map.yaml
```

Next, change the map path in `f1tenth_gym_ros/config/sim.yaml` to the desired map.

```
gordon@f1sim:~/ws/gym-one/f1tenth_gym_ros/config$ emacs -nw sim.yaml
>>> SNIP <<<
# map parameters
map_path: '/sim_ws/src/f1tenth_gym_ros/maps/levine_obs'
map_img_ext: '.png'
>>> SNIP <<<
```

After any changes to the `sim.yaml` file, the docker will need to be restarted, re-entered and rebuilt.

8 Demonstrations

Project demonstrations will be held during the TA office hours on September 23rd. Each group will have 45 minutes to work with the vehicle and must have their simulator ready and running for presentation before their allotted time. Please make sure to have a working implementation before arriving. If extra time is needed at the end of a group's designated window, groups will round-robin through remaining time.

8.0.1 On Simulator Demonstration

A satisfactory demonstration includes the following:

- While driving autonomously, the vehicle completes one lap of `levine_blocked` without colliding into a wall or obstacle
- While driving autonomously, the vehicle completes one lap of `levine_obs` without colliding into a wall or obstacle

In presentation of the simulator, start your `gap_follow` node in an adjacent terminal to the simulator preview. Have the node output a message to terminal indicating that the node has successfully been started. *Note: Continual terminal output delays node processing and may impact the correct operation of the vehicle. Provide terminal output for the successful launching of the node and no more.*

8.0.2 On Vehicle Demonstration

The presentation on vehicle will be held on a track set up in the classroom. The vehicle will be expected to complete the following tasks:

- While driving autonomously, the vehicle completes one lap of the second floor of TBE-B without colliding into a wall or obstacle.

When deploying to the vehicle, account for differences in the ROS API, sensor location, and physical parameters of the vehicle. The suggested radius of the vehicle frame is 6 inches.

Lab 3 Resources

F1TENTH Lab 4: Follow the Gap The F1TENTH group's description of Lab 4: Follow the Gap. Instruction URL: https://github.com/f1tenth/f1tenth_labs_openrepo/tree/main/f1tenth_lab4

`LaserScan` Message Quick link to the ROS `LaserScan` topic reference data. Instruction URL: http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html

`AckermannDriveStamped` Message Quick link to the ROS AckermannDriveStamped topic reference data. Instruction URL: http://docs.ros.org/en/melodic/api/ackermann_msgs/html/msg/AckermannDrive.html