Disparity Extender

Welcome to the second half of the workshop. Now that the boring part is out of the way, the real fun can begin. In this section you will be taking everything you learned over the past while and implement your own disparity extender algorithm! You will then race your algorithms against each other in a virtual environment and the winner will get to do something special at the end. But enough preamble, let’s get started!

# Prerequisites

There are several perquisites you need to have installed on your laptop before you can get started

* Python 3.8 or higher
* A code editor (Visual Studio Code is recommended)
* Clone the workshop repository
* The gym module (installation instructions below)
* NumPy and Matplotlib (optional but may help)

Installation instructions are mostly the same between systems, however if there are any differences, we will point it out to you. Feel free to ask us as well if you have any troubles with the installations

## Cloning the workshop repository

Cloning the workshop repository can be done in a number of ways. For those of you who have git installed on their laptop create a folder on your desktop, open cmd/terminal and run:

*git clone https://github.com/RacingAI/Reactive-Methods-Workshop.git*

This will download the repository to the folder.

For those who do not have or are unsure of using git, the repository can also be downloaded from the GitHub website. Navigate to this link:

[*https://github.com/RacingAI/Reactive-Methods-Workshop.git*](https://github.com/RacingAI/Reactive-Methods-Workshop.git)

and click on the green button that says “Code” and click “Download Zip”. Extract the zip to a folder on your laptop, name this folder *“Reactive Methods Workshop”*.

## Installing Gym

Open cmd/terminal and navigate to folder where you cloned the repo using:

*cd /PATH/TO/* Reactive Methods Workshop

Then run the command:

*pip install --user -e gym/*

This will install the gym module and all its required dependencies.

## Numpy and Matplotlib (OPTIONAL)

Everything in this workshop can be done with base python, however some may find some of the array functions in numpy useful. Matplotlib is a useful graphing module, it may help you visualize what your algorithm is doing. These can be installed by running the command:

*pip install numpy matplotlib*

## Testing the Environment

Now that you have all the prerequisites set up, run the testing code to make sure that everything is set up right. This will also give you an opportunity to see how the simulator works and how you can add in your own driver class.

To run the simulator, navigate to:

*/Reactive Methods Workshop/src/*

Then run the file labeled *“simulator.py”.* You can do this by pressing the “run” button in your code editor or by opening a cmd/terminal window in that directory and run the command:

*python3 simulator.py*

If everything is set up correctly you will see two windows pop up, as shown below. The first window with the purple rectangle (driver) is the simulation environment. The second window with the red rectangle is a visualizer so you can see in real time the LiDAR scans from the car

A screenshot of a computer

Description automatically generated with medium confidence

You will see the purple car move as the simulator steps forward in time.

# Understanding the Simulator

There are a few things you need to know about the simulator to be able to effectively participate in this workshop. There are a set of imports at the top of the simulator.py file, these are needed for the simulator to run. DO NOT CHANGE OR REMOVE THESE.

## Importing the Driver

You will notice on line 19, there is a line which says:

*from drivers.starting\_point import SimpleDriver*

Let’s deconstruct this line. The word in green is the folder location of the file that you want to import. The word in red is the name of the file to import and finally; the word in orange is the class in the file that you want to import. So, the line above in “English” means “Import the SimpleDriver class from the file called ‘starting\_point’ which is in the ‘drivers’ folder”.

When testing your driver, you need to have a similar line to the one above. Place your file in the ‘drivers’ folder located at”:

*Reactive Methods Workshop/src/drivers*

Then in the *simulator.py* file add a line like the one on line 19 to import your driver.

## Changing Racetracks

The mark of any good algorithm is its ability to adapt to different situations. That’s why we have provided you with a number of different racetracks to choose from. You can change the track you’re using by modifying line 25. Your choices include:

* TRACK\_1
* TRACK\_2
* TRACK\_3
* OBSTACLES
* TRACK\_CUSTOM

# Coding the Algorithm

The now that all the setup is done, you can start coding your own disparity extender! The rest of this document will just be a refresher of whatever was mentioned in the PowerPoint. You can refer to here if you ever get stuck. However, don’t just copy the steps below. At each step, you should consider “Why am I doing this step?” or “What happens if I change this or maybe don’t include that”. Approach this with a healthy level of inquisitiveness and if you can make any improvements to the algorithm, then go ahead! We want to see what sort of innovations you can come up with!

*[The challenges are optional]*

## Step 1: Limiting the FOV

Limiting our field of view is important for two reasons. Firstly, we can get rid of information we don’t care about. For example, we don’t care about the LiDAR scans behind us since our main goal is to go forward. We also don’t care about the LiDAR scans at our 9 and 3 o’clock positions and a little bit around them. The reason being, if there was a disparity in that region, we wouldn’t be able to turn quickly enough to face that disparity. In reality, every autonomous car as a non-zero turning circle that we have to be aware of. Secondly, limiting our FOV improves processing speed. If we don’t have as many LiDAR scans to consider, our algorithm becomes faster. While this is not so important in our virtual simulator, in real races shaving a second or two off your algorithm could be the difference between winning and losing.

*Hint 1: This is done in the preprocess\_lidar function*

*Hint 2: Maybe using python array slicing might be useful here*

## Step 2: Finding Disparities by Thresholding Differences

The next step in the algorithm is to find all the disparities by thresholding the differences. Remember the differences being negative doesn’t help us. So how would we ensure that the differences are always positive?

*Hint 1: For this you need to implement the get\_differences and get\_disparities*

*Hint 2: Maybe the abs function might be useful*

*Challenge: Can you find the index of the disparities using numpy and one line of code?*

## Step 3: “Extending” the Disparities

But remember we can’t just pick any disparity we find. As shown in the PowerPoint we still run the risk of running one of the walls. So, we need a way of finding “safe” disparities that we can choose. We can do this by taking a disparity and looking at its immediate neighbors. There will be one “closer” and one “farther” scan. Starting at the closer scan we calculate how many lidar scans we need to overwrite so that it covers half the width of the car from the distance you’re at. Once you figure out this angle you can find out how many lidar scans that equates to. Then overwrite those many scans with the “closer” distance. You can overwrite “further” distances with “closer” ones. But never “closer” ones with “father” ones.

*Hint 1: You need to implement get\_num\_points\_to\_cover, cover\_points and use both of those in extend\_disparities to finish this step*

## Step 4: Steering Angle and Speed

Now that the possible list of disparities is narrowed significantly, we can now choose the disparity we want to follow. We can choose the longest disparity and based on its index in the original scans array, we an also figure out the steering angle. The speed function is where you can really let your creativity show. You can just have the function return a constant value or you can create a function that outputs the speed based on the length you’re travelling ect. How you implement the speed function is entirely up to you

*Hint 1: You need to implement the get\_steering\_angle and the get\_speed function*

## Step 5: Run the Algorithm

Now that you have coded all the helper functions, you need to implement the “run\_algorithm” function. Here you place the helper functions you coded before and return two numbers, the speed and then the steering angle. It would also be wise here to figure out how many radians there are per LiDAR point as this number is heavily used throughout your code.

*Hint 1: You need the functions preprocess\_lidar, get\_differences, get\_disparities, extend.disparities, get\_steering\_angle and get\_speed*