Dawson Horvath
Ben Huckell

ENPH 353 Final Report - Team 8

Competition Overview:

Each team will control one robot attempting to navigate a gazebo simulation world. Robots will be judged on their ability to properly maneuver the course, as well as successfully read the locations and license plate numbers of the cars it encounters.

Requirements:

- Build a ros gazebo controller framework used to structure the publishing of commands and subscription to messages
- Process images from robot front camera to determine the useful information from the background noise
- Build a OCR neural network capable of detecting characters from images of license plates
- Return position and plate number of parked cars
- Avoid all pedestrians and moving objects, while staying on the proper roadway

High Level Design:

Design Decisions:

- Callback structure will include multiple asynchronous callbacks, all responsible for independent functionalities of the robot
- OpenCV image processing to extract key features from camera image feed. Use distinct features to control robot operation, including navigation through the use of a PID algorithm, pedestrian and truck detection, and license plate imaging
- Use of a Convolutional Neural Network (CNN) to detect and read characters of the gathered parked car images, including both parking locations, as well as license plate numbers

Data Processing Architecture:
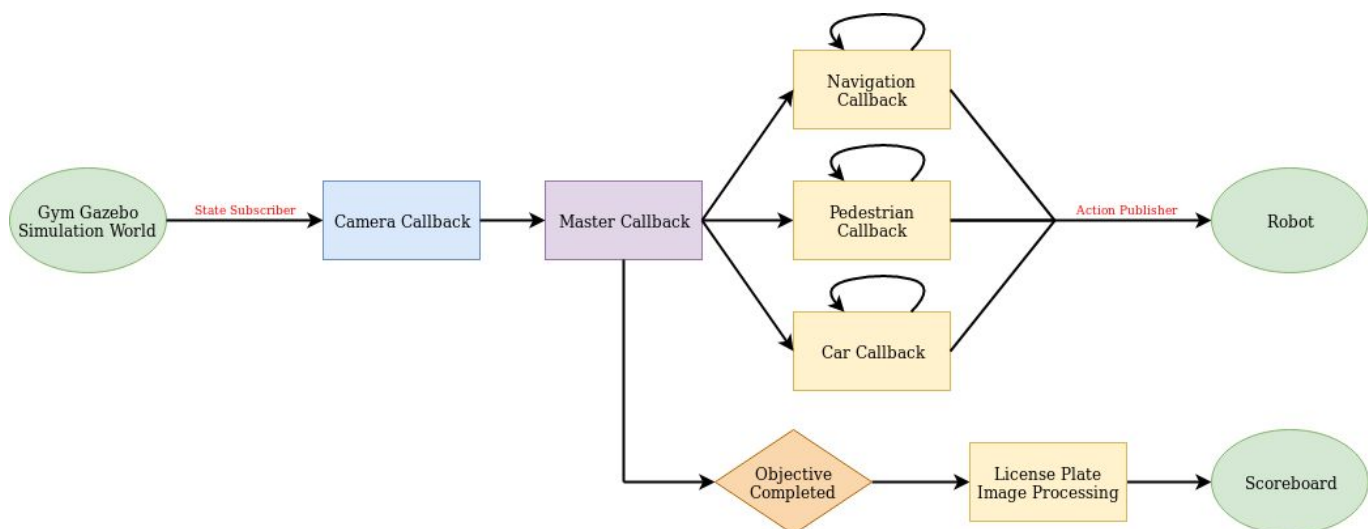
The above diagram is a simple flow chart of our data processing architecture. It is meant to highlight the pipeline that exits between the input images received from our camera and the output commands we send to our robot. The

camera callback subscribes to the rospy raw image stream to gather its input. The master callback handles the processing of this image to identify and separate key features that would be useful for our asynchronous (denoted in yellow) callbacks. It also handles the logical flow of our program, and the distribution of the processed images to their appropriate callbacks. The three main asynchronous callbacks - Navigation, Pedestrian, Car, all perform further image processing before sending output commands to the robot. We designed this system with function and callback modularity in mind, and kept separate aspects of the output decision making process independent from one another.

This led to an important element of our design approach - the implementation of asynchronous callbacks. This allowed our three asynchronous callbacks to run with cycle times independent of the other two callbacks. Through testing, we found that our navigation callback was highly affected by the cycle time of our main loop. By separating our functions into their constituent parts, we saw much lower cycle times and much smoother PID control.

When the robot finishes its complete path while navigating the course, the "Objective Completed" condition becomes true within our code. The master callback then directs the program flow to process the gathered licence plate images from the car callback, run them through our neural network, and output the location and number of the license plates it has seen.

Lower Level Design:
Camera Callback
Extremely simple callback that subscribes to the rospy camera image node, and closes the communication bridge between the gym-gazebo world and the image input to our system. Images are not processed in this callback, but rather are just passed along through to the master callback. Establishes the gateway between the simulation and our code.

Master Callback
The master callback function controlled the main logic of our program loop. It was responsible for beginning all of our asynchronous callbacks, and keeping track of our state variables. To control the functionality of our system, and to ensure that we kept track of our "progress" through the map, we needed a way to indicate where we were on the map.

Rather than creating a sophisticated state machine, we decided to monitor our position through the use of state flag variables. For example, our navigation algorithm includes a state flag variable that indicates whether our robot is on the inside or outside loop. Understanding and tracking this is crucial to the successful navigation of the course. There are different actions that the robot must take in order to navigate the various obstacles. In order to properly execute these unique actions, our state flag variables provided us with all of the information we needed to locate ourselves on the map, and identify what actions are required at that specific location.

Within our master callback, we performed various, general, image processing algorithms to extract key features that would be of use to our asynchronous callbacks. These are listed below.
- Roadside white lines/Roadway paths
- Blue cars
- Roaming truck
- Any red within the image

By extracting some of these key features now, we are able to dedicate the three main asynchronous callbacks specifically to feature detection. This increases the cycle time of the callbacks, and keeps our code very modular. This way, if any of the filtered images are shared among the callbacks, we only need to filter them once, rather than many times.

Dawson Horvath
Ben Huckell

Navigation Callback

Navigation was done with a hysteresis controller because the action space was discrete in decision. The hysteresis controller was built with the ability to follow each edge of the road for the purpose of allowing access to all parts of the map. We used this methodology along with a PID control algorithm to accurately maneuver the course.

This ability to follow individual sides of the road was done by splicing the input image up into a left and right half that ignored the other line. Once half the image was ignored we then took slices of the images to build subsections which we could find the location of the line in. This way the controller could see further ahead and predict turns more accurately.
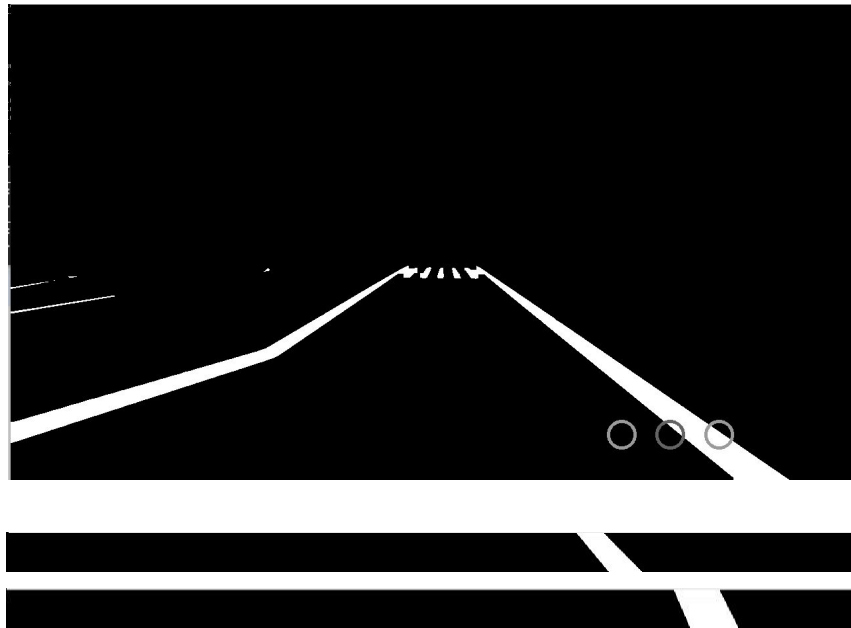
Figure 1: This figure shows the original filtered image, as well as two slices that were taken out of the image and then averaged in order to find the location of the line. This helped predict future changes in the direction of the road. The darker circle in the middle is the average and the two lighter circles are the location of the line in the slices.

The hysteresis controller works in a way that has an ideal set point but has a buffer over the set point in each direction where an amount of error is acceptable. This makes the controller less sensitive and smoother for control purposes. The controller, while over the hysteresis section of the setpoint, will always be turning towards the line it is set to follow even if it doesn't currently see that line in view. Below is an image showing this.

Dawson Horvath
Ben Huckell

Figure 2: *Showing the robot turning toward the line that it is supposed to be following despite being out of frame.*

Pedestrian Callback

While the robot is navigating the course, the pedestrian callback is being repeatedly called. The condition for the pedestrian sequence to begin is for a red line to be seen at a certain location in the image frame. The figure below shows the approximate condition needed to trigger the pedestrian stop sequence.
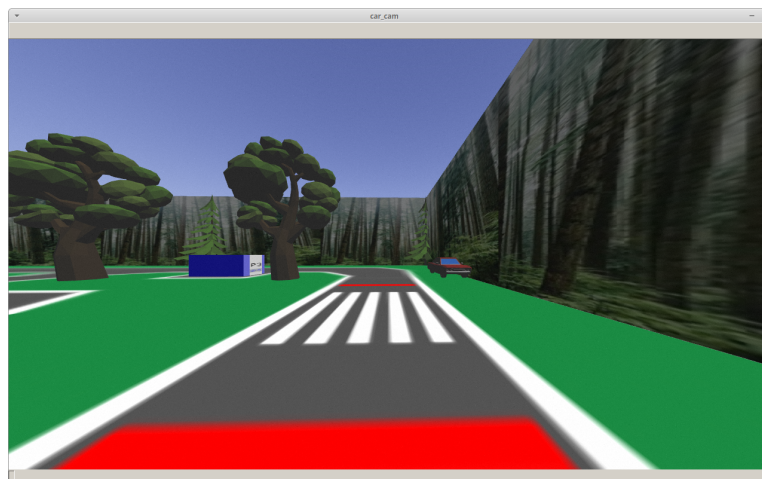
Figure 3: Pedestrian stop condition from the red line before the crosswalk

After we have successfully stopped, we must ensure that the pedestrian is not within the road lines. If we proceed and this condition is not satisfied, we run the risk of crashing into the pedestrian, thus docking points from our score total.

The first approach we considered was to extract the pedestrian from the image by noticing the colors present on the pedestrians clothing. This would have been an acceptable approach, however this did not prove very robust since changing the clothing color of the pedestrian would cause our algorithm to fail.

Our actual approach relied on monitoring for moving objects within the frame through the use of background subtraction. By continuously subtracting the current frame from the past frame, we would be able to see which

pixels were changing. We filtered the image to show a purely black frame, with any moving objects depicted in white. Following this, we eroded and dilated the image to form a "blob" of white on the screen rather than multiple white spots, each of which is independently finding pixel changes. This allowed us to find the contours of the image using OpenCV's "find contours" function, and draw bounding boxes around the pedestrian.

To determine when it was safe for our robot to proceed through the crosswalk, we color coded the pedestrian bounding box at different times in its path. When our image processing saw any motion within a certain area of the frame, a red bounding box would be drawn. This is shown below.
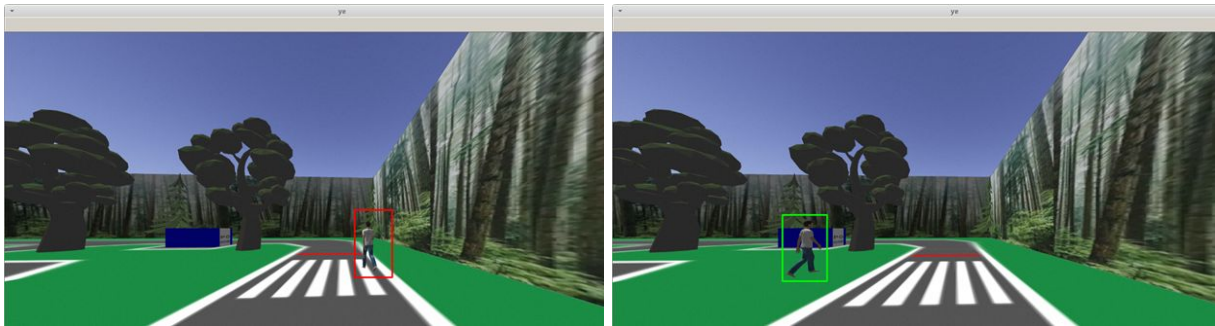
Figure 4: Left image shows pedestrian moving along the crosswalk, Robot will not proceed in this state(red bounding box). Right image shows the pedestrian condition where it is safe for robot to pass crosswalk. Pedestrian has completed a trip across the road, and has stopped moving for at least 3 frames.

Our "proceed forward" condition relied on a very specific sequence of events to happen. First, the algorithm must see at least 15 frames of red. This threshold is in place to protect against random box jitters when the pedestrian is stopped on either side of the road. This ensures that the robot sees the pedestrian pass in front of itself at least once before continuing. All possible starting positions of the pedestrian are covered with this algorithm, and we were able to achieve almost a 100% success rates at pedestrian crossings.

When the pedestrian reaches the end of its cross, the background subtraction yields simply a black screen (since nothing is moving). If the robot has seen 15 frames of motion, and then sees no motion for at least 3 frames, a green bounding box is drawn around the pedestrian, and the robot proceeds to safely cross the intersection. This bounding box is depicted in the above figure as well.

Car Callback

The Car Callback had two major functionalities - roaming truck/parked car detection, and image capturing. Both functionalities were crucial to our performance in the competition.

Roaming Truck Detection

Our ability to detect and stop for the roaming truck in the middle loop was required to safely enter the inner ring and capture images of the last two license plates. This algorithm was quite similar to our pedestrian detection algorithm with some small differences.

In the case of the pedestrian, we looked for a red line at a specific pixel location on our screen. When entering the inner loop, it was much more difficult to find a robust method to determine where to stop and begin checking whether the truck is in our way. We eventually noticed that when our robot is approximately halfway between the outer and inner loops (while entering), the image feed would have a few frames where the road took up an entire

Dawson Horvath

Ben Huckell

horizontal portion of the screen. Therefore, to identify where to stop and begin looking for the truck, we simply checked for the point where our "roadway paths" filtered image from our main callback, took up an entire row of our image pixel matrix. The following diagram shows this.
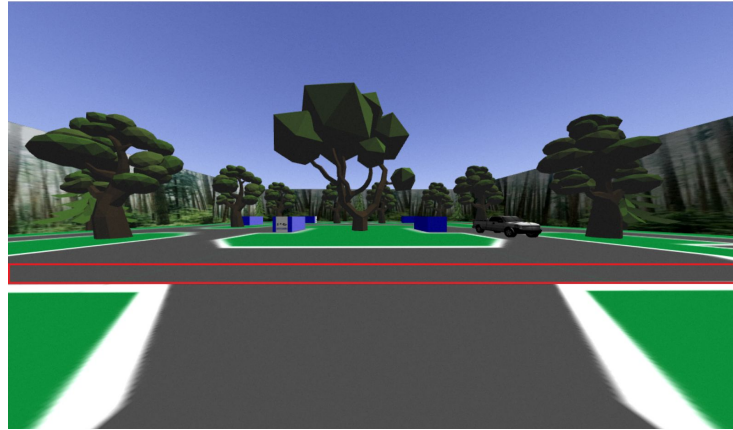
Figure 5: Stop condition for entering the inner loop. When the image has an entire row filled with roadway, our robot will stop, and begin looking for the roaming truck to identify whether it is safe to proceed.

To determine whether the moving truck was in our way, we applied the same principles of background subtraction to isolate the moving object from the rest of our image. Through some tuning, we found a minimum bounding box size that represented how close the truck was. For example, if the truck was on the far side of the inner loop, the bounding box drawn would be quite small and below the threshold size - our robot would be allowed to proceed. If no bounding box was drawn, either the truck was far enough away or it was not in our field of view. In either case, our robot was free to proceed. In addition, since we were always entering the inner loop with the truck travelling the same direction, we constricted the portion of the image that we look for moving objects to the right half of the screen. This served two purposes. First, it neglected any movement of the truck if it had already passed the entrance T-intersection. Second, it allowed the robot to begin line following again once the truck passes our right-half field of view (in the event that it *waits* for it to pass). In this case, the truck was travelling along a collision path, our robot successfully stopped, and then proceeded once the truck was not seen in the right half portion of the screen.
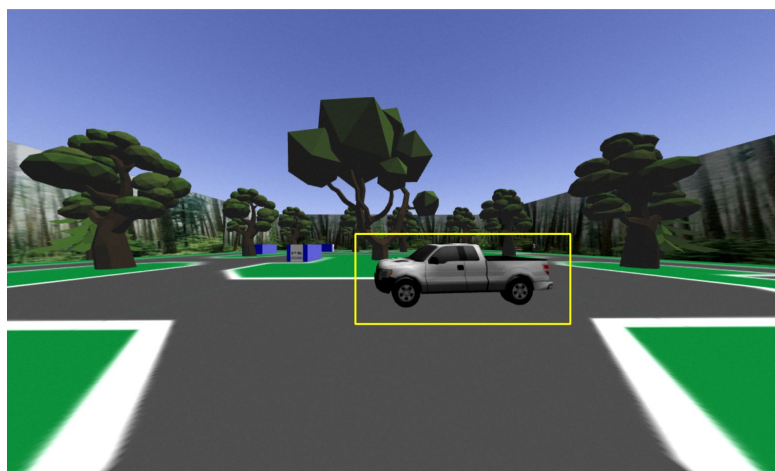
Figure 6: Bounding box drawn around roaming truck. Robot will not proceed while the car is moving in its field of view and is above a certain threshold size. Once the truck moves into the left side of the image, the robot will continue on its original path.

Dawson Horvath

Ben Huckell

Parked Car Detection and Image Capturing

The competition strategy we decided on relied on our robot navigating the course, taking pictures of the license plates of parked cars, and then finally running our captured images through a neural network to extract parking location and license plate numbers.

Throughout our competition path, our car callback used a filtered image processed in the master callback to identify any cars within the active frame. To identify cars we simply pull out a specific range of hsv values from the image corresponding to the blue color of the parked cars. We are then able to draw a bounding box around these cars by finding the contours of the image. In the case of car detection, not only are the bounding boxes included for visual effects, but the calculated position of this box also determines the image that is captured. We use the bounding box to determine how to crop our images such that only the car is in the capture. These images are processed, however this crop makes the job of extracting meaningful information in the license plate image processing callback much simpler.

License Plate Image Processing

After our robot has navigated the course and ended its run, the images that were taken by the car callback are sent through an image processing algorithm. Some examples of the images gathered by the car callback are shown below. It is important to note that while these images may be slightly blurry, our algorithm saves all of the pictures that a parked car is present in. This means approximately 10-15 images are captured per car. This increases the overall chances that at least one image is of good quality, and we can properly extract the information required.
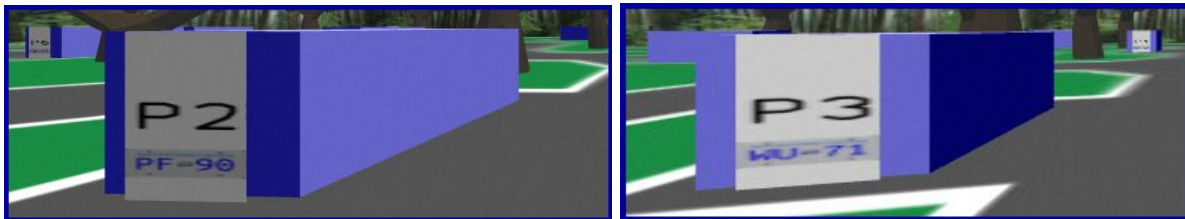
Figure 7: *Captured and cropped images from car callback*

Figure 8: *Perspective transformed images showing the extracted license plates and parking spot number from the above saved images*

The image processing had multiple layers. The main layer involved using hsv range values to pull out specific regions of interest of the images. This was mostly done within the master callback, but some small modifications

were done within the image processing callback. The following layer was used for detecting specific components from the subparts of the already filtered images - specifically the white box on which all of the required information was present. This specific region often appeared at a slight angle (given the location of the car relative to the license plate), so we also applied a perspective transform to get a clearer view of the number and letters. The next layer processed these images using an ROI detection algorithm involving the detection of contours relating to each character. Finally, we individually processed those characters with the trained convolutional neural network to find what they were.

Neural Network Training

The training set used was only a single image of each character we needed to find. These characters were preprocessed in a random image data generator. Changes included blurring, downsampling to reduce range of resolution, shifting, scaling, and changing contrasts and saturations. This made the training set very similar to the validation images which were collected from the real world (very low resolution with varying contrasts and brightness with slightly varying hues). This type of augmentation allowed us to build a random batch to train on very easily making our training set very effective for its purposes without much effort involved in the collection and classification of images.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_____
activation_1 (Activation)    (None, 62, 62, 32)        0
_____
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_____
conv2d_2 (Conv2D)            (None, 29, 29, 32)        9248
_____
activation_2 (Activation)    (None, 29, 29, 32)        0
_____
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_____
conv2d_3 (Conv2D)            (None, 12, 12, 64)        18496
_____
activation_3 (Activation)    (None, 12, 12, 64)        0
_____
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_____
flatten_1 (Flatten)          (None, 2304)              0
_____
dense_1 (Dense)              (None, 128)               295040
_____
activation_4 (Activation)    (None, 128)               0
_____
dropout_1 (Dropout)          (None, 128)               0
_____
dense_2 (Dense)              (None, 36)                4644
_____
activation_5 (Activation)    (None, 36)                0
=================================================================
Total params: 328,324
Trainable params: 328,324
Non-trainable params: 0
_____
```

Figure 9: *Summary of our character classifier, We used an adam loss optimizer with a softmax activation trying to optimize the accuracy metric.*

After we trained our CNN to detect numbers and letters seen on the license plates we looked into a validation set to quantify the accuracy of the network. This was done by running our robot through the course multiple times, and capturing images of cars with the exact same algorithm we will be running in our competition. By labelling a set of these images, running them through the network, and observing the results, we could see how well our CNN was working. After multiple iterations of this process we were able to achieve a license plate accuracy of 98%.

Dawson Horvath
Ben Huckell

<u>Extra Information</u>
Difficulties Faced
- Initially, most issues faced involved understanding the file structure and framework of the ros gazebo packages while trying to set up a reinforcement learning environment. Once this was understood then it wasn't a problem despite having a steep learning curve
- Tuning parameters required lots of iteration. We were able to successfully eliminate all time-based routines from our control system, but we had many parameters, such as HSV values for object detection, that required lots of tuning.

Reinforcement Learning
We gave a valiant effort at implementing a reinforcement learning approach as our controller, the methods attempted were the following; DQN, DDQN, and a genetic algorithm approach with cross genetic/random mutation.

General gym structure

Gym is a well known framework for reinforcement learning in simulations. Gym uses a framework which represents an environment, the environment is given by a set of functions that control the simulation world whatever type of simulation you are doing whether it be ATARI or gazebo. This is done by having functions that allow resetting of the world, moving an actor another step in the world, rendering the simulation, and closing the world. These functions also return information on the actors current state (whether it is doing the right thing), whether the actor needs to be reset, and its observation space (what it sees).

Defined environment variables

Observation states (OBS): In our case, it was either a 84x84 image stack with 4 consecutive frames or just the image unstacked. Frame stacking is done to give more information about velocity and acceleration of the actor.

Reward function: Involved promoting behaviour towards the middle of the lane determined from an added overhead camera on the robot. This added camera feed looked for a line added to the world png and read its orientation in comparison to the bow of the robot to determine how far from the ideal orientation the robot was. This then was tabulated in a way to give an absolute value score biased towards being aligned and centered with the road

Exploration decay: After playing with a few exploration decay functions varying from linear to exponential with a decay of 0.998, also tried an exponential decay that spikes back up in value after a certain period.

Attempted models

DQN: The attempted deep Q learning Network used the turtlebot DQN structure and reward function with hopes that the existing framework would give a working product. We had to abandon this approach as the deadline prevented pursuing this approach much further.

Genetic Algorithm: Uses an approach very similar to real evolution with crossing over alleles between the most successful models, then mutating new models from the child of the best models. This repeats until a desired result is achieved. We attempted to try this method in the gym gazebo framework as well, but it lead to some very interesting results such as the robot doing circles in a region of the road it could go straight, and only turning every few actions - leading to a triangular shaped path. This made this method seems much less robust than we would like and the controllability/predictability seemed a bit worse than we would like so we never explored this method much further.

Dawson Horvath
Ben Huckell

DDQN: This was the most pursued method of RL for this project. DDQN is a double deep Q learning Network. It relies on having two seperate Q state estimators giving unbiased Q-value estimates of the actions selected by the network. This is done by having a target Q network and the main network where the main network chooses the actions and the other evaluates its choice. A lot of time was spent trying to figure out how to optimize this network, and whether the network we built, based off the ATARI DDQN, was working properly. Initially the results were terribly random and did not converge in any way. This can be seen below.
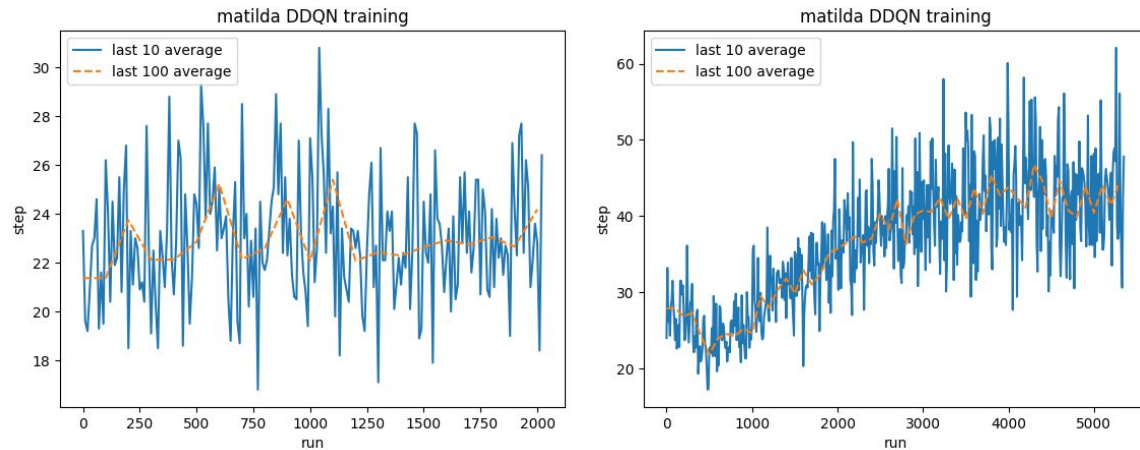
Figure 10: *Original random training despite large number of runs and slightly improved but still poor results from the improved model, still needs a lot of work to be effective in any way at the task.*

Now there are a few obvious issues with this training, the epsilon decay is way to slow being linear for the application we are using it for. There was also a major issue with the reward function not being consistent with the orientation of the robot in all positions on the map. Now after playing with the model a bit we were able to improve the results to the ones also shown above.

These results were still terrible but at least the network showed improvement despite the fact it started converging before being able to do the required task (This was approximately 6 hours of training at an RTF of 7). This is where the lack of knowledge in RL came into play after this point even with trying a few different ideas and approaches we never got results to be much better than what is shown above, we switched to trying a DQN network shortly after this and never went much deeper into the model than this since we were near the deadline.

Changes we would have made
- We wish we could have spent more time learning how reinforcement learning works and how to optimize it. This would have saved a considerable amount of time tuning our algorithm for various corner cases. If we had more time, or had dedicated more time to reinforcement learning, we are confident we could have gotten it working for the competition
- While our state flag variables did provide us with a fair amount of flexibility in our ability to handle different situations in different ways, the structure became slightly disorganized by the end of development. Many state flag variables were being used, and their meanings became somewhat blurred. In the future, we would like to develop an actual state machine to control our robot. This would allow our code to be more organized, and much easier to understand, read, and reproduce.