Autonomous System Post Disaster Route Obstruction Assessment

Steven Brewer
Robotic Engineering
Worcester Polytechnic Institute
Worcester, MA, United States
scbrewer@wpi.edu

Trevor Sherrard
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, United States
twsherrard@wpi.edu

Nithin Kumar
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, United States
nkumar2@wpi.edu

Javier Sanguinetti
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, United States
jasanguinetti@wpi.edu

Brian Wells
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, United States
bkwells@wpi.edu

***Abstract*—This work outlines a preliminary design of an autonomous system that is to be deployed after a hurricane impacts a region. The system assesses blocked roads and updates a command center to assist first responders in route planning and cleanup efforts. The system used UAV drones (agents) that learned on camera, depth, and IMU data to explore roads through curriculum based reinforcement learning (RL). A control subsystem was developed that took planned paths as inputs from the RL agent and controlled the UAV pose which utilized a unique control scheme to improve smooth tracking. Road segmentation, and obstruction detection was implemented in simulation to assist the agent in path planning. Once an obstruction was detected the GPS coordinates and image of the obstruction in question were relayed to a command center, where path planning for first responders could be performed. Multiple point-to-point planners for fast and "sufficiently optimal" path generation were examined. Ultimately, Bidirectional RRT with smoothing was implemented, which yielded positive results in multiple maps generated from real cities. Using curriculum reinforcement learning, it was found that the agent was able to learn progressively more complex behaviors, culminating with the learning of basic altitude keeping, collision avoidance, and road intersection navigation.**

***Index Terms*—Reinforcement Learning, Motion Planning**

I. Introduction

Hurricanes effect the lives on people living in the United States of America, in particular people living in the south and east coast. A hurricane has sustained winds of 74 mph (64 knots) to greater than 111 mph (96 knots) [9]. On average a hurricane causes $20.5 billion in damage to homes, buildings, and roads [9]. The average hurricane produces 6-12 inches of rainfall resulting in severe flooding [9]. Road blockages slow emergency workers ability to navigate to victims of these storms. A system of Unmanned Aerial Vehicles (UAV) deployed after the storm can assess road blockages and route plan for first responders. This autonomous system would reduce response times which in turn saves lives.

A quadcopter is a type of unmanned helicopter with four rotors in an 'X' arrangement in which half the rotors spin clockwise while the other half spin counterclockwise. Their light weight, compact design, and maneuverability make them ideal for unmanned areal vehicles (UAV) in both commercial and military settings.

Utilizing multiple UAVs in a system, makes the network fault tolerant and increase the area that can be covered. Traditionally multi-robot systems were solved with hand-built algorithms. Reinforcement Learning (RL) and multi-robot systems merge into an interesting machine learning problem which can solve trajectory planning of the multi-robot system. Due to time and hardware constraints the drone training was limited to a single UAV.

The action space is comprised of the drone/UAV moving 5 meters up, down, left, right, forward, back or rotating $90°$. The observation space consists of drone camera (front, downward, rear) images, Inertial Measurement Unit (IMU) data, and distance sensor data. Camera data was converted to segmentation masks which highlighted the roads. The drone's decisions were passed to a control subsystem which took the planned path as inputs and controlled the UAVs pose.

In order to have a more comprehensive understanding of the operating environment, the ability to generate segmentation maps is paramount. Image segmentation is the process in which individual pixels are labeled within an image as belonging to a given class. This is in contrast to object recognition, which attempts to find objects within an image, and produce a bounding box describing the location and size of the object. Generated segmentation maps are particularly helpful when an agent is attempting to isolate a non-bounded area of a given image. That is, when trying to detect the ground versus the sky in an image acquired from a UAV, it is desirable to have a mask representation of these areas, rather than a bounding box. Robust image segmentation techniques typically employ a deep learning architecture to generate segmentation masks. These models typically convolve the image into dense feature spaces. These feature spaces are then passed through a decoder, which typically up-convolve them into usable segmentation class masks. In the case of this work, the resulting image segmentations are used to inform the Deep Q-Learning network of the current state of the UAV's operating environment.

The quadcopter is an under actuated system. The thrust from the four propellers and motors are the only control inputs for the vehicle, which has 6 degrees of freedom. The system is also inherently unstable. This leads to the requirement to have a closed loop control system in order to stabilize the flight of the quadcopter. One of the objectives of the controller subsystem was to attain closed loop control of the orientation of quadcopter in both hover and constant velocity modes. An improved dynamic model was utilized for the development of the plant model which incorporated rotor drag force into the calculation of the pitch, roll and yaw rates. [15] The second objective was to achieve improved trajectory tracking in order to transition between waypoints commanded by the higher level subsystems. A modified controller was developed which implemented yaw and pitch mixing to implement coordinated turns through the transitions of consecutive paths. The trajectory tracking utilized Dubin's curves to enable constant velocity through the turns. This pared nicely with the inner control loop's constant velocity control tuning. [14]

Reinforcement learning involves an agent that must learn about a dynamic environment through trial and error and choose the best outcome to maximize reward. In the standard reinforcement learning model the agent has the choice in action based on the state received from the environment. The goal of the agent is to determine the optimal behavior. The agent is trained to complete a specific task with the goal of generalizing to similar tasks.

The tasks are modeled as Markov decision processes (MDP). An MDP is memoryless, meaning given the current state the future is independent of the past states [6]. The environment and agent states S consist of $s_1, s_2, s_3, \ldots s_t$ where $s \in S$. The set of all possible actions the agent can make A consists of $a_1, a_2, a_3, \ldots a_t$ where $a \in A$. r is the immediate reward after transition from state s to s' where $s' = s_{t+1}$ after taking action a. Reinforcement learning is stochastic in nature and therefore the probability of transitioning from s to s' given action a, $P(s'|s, a)$, is calculated. The goal is to learn a policy π that maximizes the cumulative reward. This is done by selecting the action a with the greatest probability of transitioning the agent to the goal state.

For this work the AirSim simulator was used in conjunction with the Unreal Engine 4 [10]. The simulator was selected based on Collins et al's review of robotic simulators [11]. AirSim's goal is to reduce the gap between simulation and implementation for autonomous systems [8]. The agent was trained on the Neighborhood environment. An example of the simulator can be seen in Figure 1. The simulator provides many sensor options to interface with. Each drone utilized the front, bottom and rear cameras, IMU data and distance sensors.

In this study, we consider path planning for first responders as the final step in the pipeline. Once the swarm of drones have identified road blockages, these sites will be prioritized in terms of distance to the command center. Then point-to-point planners will be generated to guide the first responders (on the road) from the command center to each blockage. For training and benchmarking purposes, we will use binary obstacle maps of real cities. We will seek to implement some popular path planning algorithms including A* for optimal path generation and variations of rapidly-exploring random trees (RRT) to determine the best fit for this application.

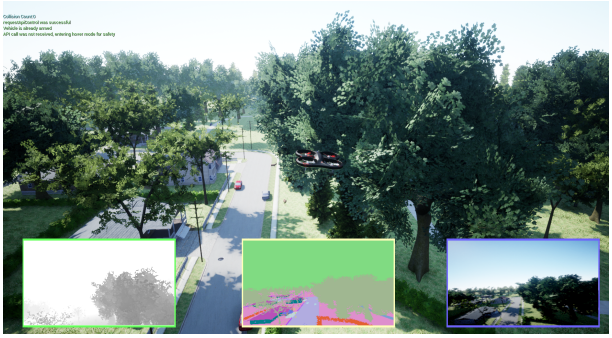

Fig. 1. Neighborhood Environment Example where the three sub-images are the Drone's front camera data. The left is depth data, middle is image segmentation, and the right is an RGB image. All data was generated in real time

II. Literature Review

A. Controls

There are many open source controllers available that implement feedback control of a quadcopter. The approach taken for this project was to use a first principles approach to the dynamics and controls in order to achieve a deeper understanding of the underlying aerodynamics and control of a quadcopter. This required searching for a reference that was tailored for someone with robotics and feedback controls training but not necessarily with an aerospace background. An excellent resource was found that provided that bridge from controls to aerospace knowledge. [14] The techniques outlined were specifically for a fixed wing aircraft but provided sufficient insight and methodology that made it possible to adapt the information for the derivation of the quadcopter plant model. An additional improvement was added which incorporated propeller drag force in the calculations for the pitch, roll and yaw rates. [15] This improved the realism of the model and provided an opportunity for support of accelerometer modelling and Kalman filter development and tuning. The accelerometer support is left for future development.

B. Reinforcement Learning

Reinforcement Learning has been of great interest in the robotics community and the machine learning world recently. Mnih et al [4] proved that neural networks were capable of human level control of atari games using a Deep Q-Network (DQN). The DQN was a pivotal moment because it opened the door to using neural networks in reinforcement learning to accomplish tasks previously thought non-commutable. Following the DQN, van Hasselt [3] showed improved performance and learning stability with a Double Deep Q-Network (DDQN). DDQN uses two networks, an evaluation network and a target network, to compute the temporal difference error used in the learning step in reinforcement learning. DDQN also introduced the idea of using batch learning which allows for the algorithm to become temporally independent and learn in a pseudo off-line fashion. Both DQN and DDQN

are suited to environments where discrete actions can be taken, Lillicrap et al [2] addressed this by extending the Deterministic Policy Gradient (DPG) of Silver et al. [7] with Deep Deterministic Policy Gradient, thus allowing for an Actor-Critic reinforcement learning regime. DDPG suffers from stability issues in the weight biases within the network and thus Fujimoto et al. [1] introduced the Twin Delayed Deep Deterministic Policy Gradient (TD3). TD3 introduced the idea of having two critics and taking the minimum value between the two when completing a learning step. This forces the network to undervalue state-action pairs which does not get backpropagated, whereas, DDPG was over-valuing the state-action pairs leading to tremendous compounded error.

C. Path Planning

There has been extensive work done in the literature in the field of path planning. In particular, the approach for searching for the best-path first is credited to [16]. The A* algorithm is complete and optimal but one practical drawback of this search is the computation time since all nodes have to be stored to memory. This is certainly a consideration for our application, since we would like to generate these point-to-point planners without too much delay.

RRTs are a relatively newer class of algorithms that are able to efficiently search non-convex, high-dimensional spaces [17]. Though we are only searching in 2D space for our problem, we sought to investigate the benefit of this search over A*. There have been several extensions of RRT search, including bidirectional RRT and an optimal variant, RRT* [18]. We will explore the former in this study.

III. Problem Description

A. Problem Statement

Design an autonomous system that will be deployed after a hurricane that will assess blocked roads and update a command center to assist first responders in route planning and cleanup efforts.

B. Requirements

The system requirements are as follows:

- System shall be designed for post hurricane in clear weather and operate during daylight.
- Wind is limited to a max speed of 10 kph.
- Systems shall operate in urban and suburban areas.
- UAV are to be used for road obstruction classification and localization.
- All predictions, control, and path planning shall be performed locally on the UAV.
- System shall identify road obstructions.
- Systems shall deliver GPS coordinates and images of obstruction to first responders.
- Communication between autonomous system and command center shall operate without the use of cellphone towers.
- System shall perform path planning route for first responders.
- Communication between UAVs is limited to 100 meters.

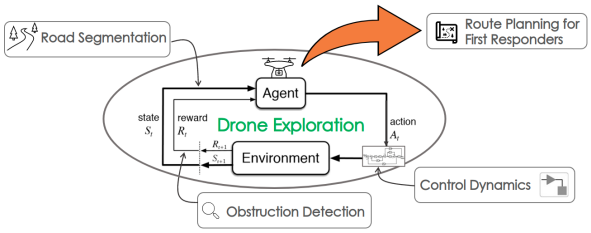

Fig. 2. Subsystem flow diagram

- Communication between UAV and command center is limited to 5 km.
- UAV shall operate continuously for 15 minutes.
- UAV shall fly no higher than 120 meters above the ground.
- UAV shall be capable of avoiding no fly zones.
- UAV speed is limited to less than 30 kph.
- UAV computational timeout is reach after 3 minutes at which point the UAV returns to the command center.

C. Goals

The goal of this project to develop an autonomous system for route obstruction detection and first responder route planning to a preliminary design level. This report outlines the proof of concept that could be further developed. The goal is to outline the system processes required to accomplish the problem statement. Systems were designed to drive down risk of implementation. Some systems that were lower risk such as object detection and image segmentation are to implemented during a detailed design. For now utilizing the simulators built in function help supply the drone with the necessary information.

IV. Methodology

The UAV/drone/agent needed to learn to explore the environment following roads in search of road obstructions which it would report back to a command center for first responder route planning. To accomplish these tasks the work flow was broken down as follows: The drone was at a high level controlled by the reinforcement learning agent. The agent took an action which was passed to the basic control system. The control system took in the took in the agent's desired location and controlled the drone's position and orientation to achieve the agent's required position. The environment provided the agent the new state and the reward for the actions took. The obstruction detection subsystem provided part of the reward function to the agent. The state data required road segmentation prior to being given to the agent. The agent then relayed the obstruction detection to the command center if one was detected. The obstructions fed the route planner information used in selecting the best route. The subsystem flow diagram is illustrated in Figure 2.

A. Controls

The plant/physical model used was originally developed for RBE-502 Robot Control. The modelled quadcopter had a mass of 1kg which is in the range a DJI Mavic 3. This was the right size to support the sensor suite

Fig. 3. Example Simulation Render. Shows an example of quadcopter dynamics simulation using the Unreal Engine renderer.

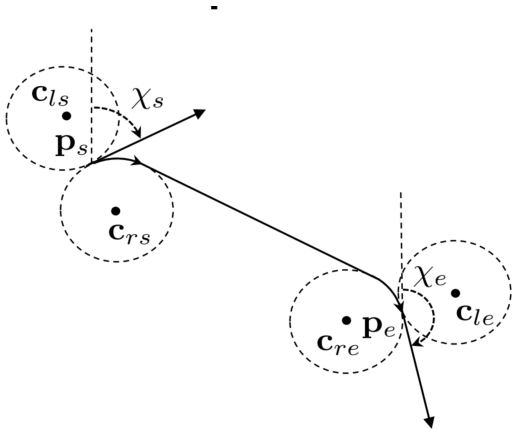

Fig. 4. Example of a Dubin's Curve. Shows an example of Dubin's curve for generating a constant velocity path.

required while still small enough to support the swarm behavior envisioned for the project. The original control was tuned to hover adequately but not track trajectories well. The position controller simply mapped movement in x to changes in pitch and movement in y to changes in roll. The rendering was done using Matlab plotting to display a primitive quadcopter while flying basic circles and figure eights. While this was sufficient for exercising the developed plant model it was not sufficient for the purposes of this project. The first step was to adapt the existing trajectory input, controller and plant model to use the UAV toolbox in Matlab. The only piece of the UAV toolbox that was utilized was the rendering component which supported using Unreal Engine (UE) to create a scenario and render the quadcopter and scenery. This provided a much higher fidelity view of the flight path and performance of the quadcopter. An example a simulation using this rendering engine is shown in Figure 3.

The control was next updated to yaw the quadcopter in the direction of flight. In this way the camera on the quadcopter was always pointing in the direction of motion which is exactly the behavior needed for this application. The transform developed is shown below in the equations.

$$pitch = x * sin(\pi/2 - \psi) + y * cos(\pi/2 - \psi) \quad (1)$$

$$roll = x * cos(\pi/2 - \psi) + y * sin(\pi/2 - \psi) \quad (2)$$

These equations were used to preprocess the commanded x and y position and yaw the quadcopter in the direction of motion and transform the change in x and y into a combination of pitch and roll. Once the control was updated to point the camera in the direction of travel an additional updated was needed. Previously, the control followed waypoints by accelerating to a target velocity and then as it approached the waypoint it would decelerate. Once stopped at the waypoint it would yaw in the direction of the next way point and accelerate towards the new direction. While this piece-wise approach worked it was quite time consuming to accelerate then decelerate and yaw for each waypoint. The solution to the problem was to update the preprocessing of the waypoints to generate Dubin's curves to transition from one direction/orientation to the next. An example of a Dubin's curve is shown in Figure 4. [14] This meant that now the controller would continue to fly at a constant velocity from waypoint to

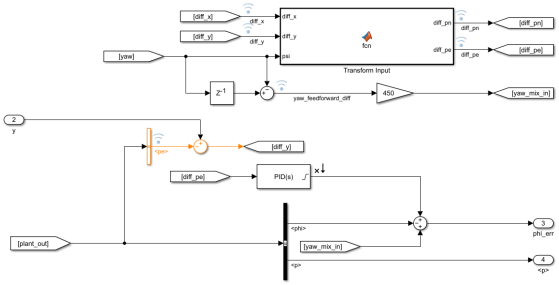

Fig. 5. Implementation of yaw/roll mixing. Shows implementation of feedforward term needed to support coordinated turns.

waypoint and generate a curve to smoothly transition from one direction to the next. One additional update to the control was needed to complete the trajectory tracking improvement. In order for the quadcopter to successfully follow the commanded curve through a turn, an additional feedforward term was added to the controller. This entailed mixing roll with changes in yaw. In this way, a coordinated turn was achieved. This caused the quadcopter to bank into the turns which improved tracking performance. The mixing feedforward implementation is shown in Figure 5.

B. Obstruction Detection

As previously mentioned, detecting obstacles within the environment is critical to inform the first responder path planners of the hazards in the operating environment. In the scope of this work, obstacles are target objects that are detected within image feeds taken directly from onboard cameras on the UAV. This detection capability was implemented with both a deep learning solution and a solution making use of provided AirSim functionality.

For the deep learning based solution, an open source implementation of yolov4 model was investigated for use as the detection engine. It was quickly determined that using a deep learning approach for this proof of concept provided little value over provided functionality within AirSim. The primary driver behind this decision was that the resource cost to re-train the model on images taken within the simulation environment and validate it would

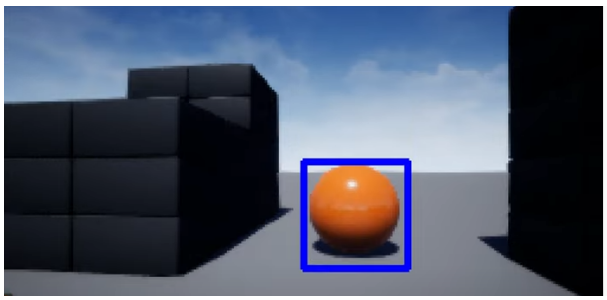

Fig. 6. AirSim Based Object Detection

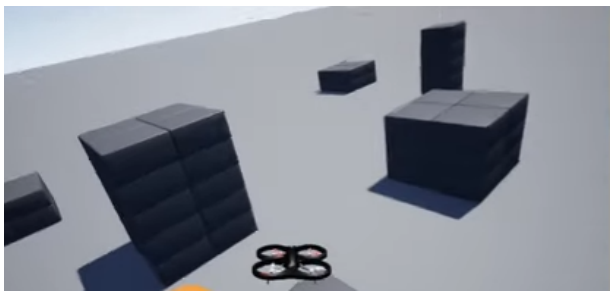

Fig. 7. Exocentric Drone View Of Scene

Fig. 8. Segmentation Mask Generated From Egocentric View Of Scene

be too high given the limited scope and time allotted for this work.

The AirSim simulation environment allows client software to quickly extract the bounding boxes of object meshes within a given AirSim environment. This is done by providing AirSim with the mesh ID of targets of interest. AirSim maintains an understanding of the state of the UAV within the environment, and once a target objects comes into view of a given camera within a given distance, these bounding boxes are returned to the aforementioned client software. A set of ROS nodes were developed that wrapped this object detection functionality for use within the developed system. Figure 6 in this section depicts an example detection of an orange ball object within an AirSim test environment.

C. Road Segmentation

As mentioned in previous sections, the utilization of image segmentation in the development of the aforementioned road following behaviour is required. The results of this segmentation operation are used as inputs to the Deep Q-Learning training process. As in the case of obstruction detection, this segmentation capability was implemented with both a deep learning solution and a solution making use of provided AirSim functionality.

For the deep learning solution, an open source implementation of a multi U-Net model was investigated for use. This model was pretrained on images taken from aerial perspectives from UAVs. As mentioned in previous sections, the U-Net model operational principle relies on the idea of encoder-decoder. The encoder repeatedly convolves input images into dense feature spaces. The decoder then up-convolves these feature spaces into usable class masks. When depicted graphically, the model takes on a ”U” shape, hence the name. The output of the model is an NxMxD tensor, where NxM is the input image size and D is equal to the number of one-hot encoded class channels. In one-hot encoding, each class is given a separate image channel, where each pixel value is set to the confidence that the corresponding pixel at location (N, M) belongs to the class in question. The index of the class channel with the maximum confidence value can then be assigned to each pixel. This results in a new NxM image where each pixel in a value between zero and D, and corresponds directly to the inferred class label per pixel. It was found through inspection that the preliminary performance of the pretrained U-Net model would not suffice for the Deep Q-Learning training process. As such, the provided AirSim functionality was utilized in the final implementation.

As in the case with object detection, AirSim provides built in functionality to generate segmentation maps from images taken within the simulation environment. This functionality was wrapped in a ROS node for use in the final implementation of the overall system. Figures 7 and 8 in this section depict an exocentric view from the UAV in the simulation environment, and a segmentation map generated from it’s egocentric view of the same scene, respectively. Note the the segmentation map shown in figure 8 is generated looking for meshes matching the stacked cubes seen in figure 7.

D. Reinforcement Learning

1) Agent to Environment Interaction: The UAV was trained in the neighborhood environment in AirSim. Once the environment was launched the drone was connected where it could receive state data and interact. The segmentation ID’s were set such that road, houses, yards, and trees were masked to unique numbers. At the start of an episode:

- The weather was set to a sunny clear day.
- The UAV moved to the current home position.
- The GPS data history was set to zero.
- The first image of the state was generated.
- Clock started

The agent with the initialized state data made its first action. The drone moved based on the action. The environment was stepped, meaning that it returned to the agent the reward and the new image state data as well as the determination if the episode was complete. The information was all saved to the agents memory for sampling and learning. The process of action, state data, reward, and learning was repeated until the episode was

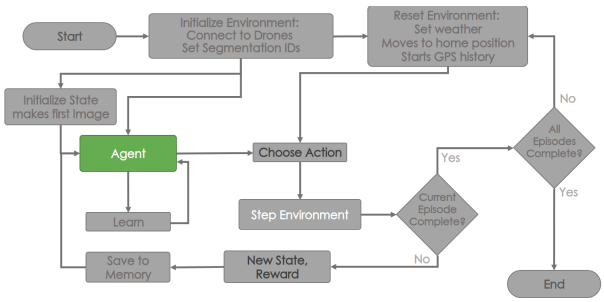

Fig. 9. Agent Flow Chart. Shows the process of the agent interacting with the environment, this includes setting up the environment to match system requirements.

Fig. 10. Aerial view of the suburban neighborhood environment.

completed. If the episode was done the episode was reset meaning the bullet-ed list above was completed. When the last episode was completed the simulation ended. The agent's learning was then assessed on a test location and the results were reported. The flowchart of the process can be seen in Figure 9.

The aerial view of neighborhood environment the drone interacted with can be seen in Figure 10. Position 0,0 was at the center of the middle intersecting road. The home positions the drone moved were at 'T' intersections. The drone was moved 20 meters from the 'T' and a no fly zone was placed behind the agent. This positioning encouraged the drone the learn to make the decision to turn quickly. Allowing the agent to go straight for too long in the learning process enforced the action making it harder to learn when to turn later down the road.

2) Agent Design: There are several control systems that could be implemented for the task. The first being hand tuned features. Hand tailored control is good for repeatable environments or repeatable tasks. They have historically been used to allow robots to complete tasks however they do not generalize well to complex environments. The system is overly specialized and if the environment changed slightly, the system fails.

Q-Learning adds the ability for the system to learn about the environment. The state action pair was stored in a Q-table that was used to look up the probability of reward. Q-Learning generalizes better than hand tailored control and might work, however, the complexity of the state-action space was such that no known computer could handle a policy table for all combinations.

Deep Q-Learning (DQN) trains a neural network to learn and approximate the policy function. Because the policy is approximate, the model generalizes to different obstacles. However, DQN suffers form a problem with over-fitting. This leads to unstable learning.

To overcome the issues with DQN Hasselt et. al [3] introduced Double Deep Q-Learning (DDQN). DDQN used two neural networks, one that is adjusted every learning step and one that was adjusted at a much slower frequency, for instance, every 500 learning step. The second network, called the target network, was copied from the current learning network, called the evaluation network, and used in the learning algorithm to provide a stable target policy approximation. This secondary network disassociates the learning steps from time and provides a stable target for the network to use when calculating the value of state-action pairs. DDQN was selected as the high level control system for the task of road following and obstruction detection.

A separate, but equally important mechanism to decouple learning from time is the replay buffer. Using the replay buffer allows the storage $[State, Action, Reward, New\ State, Termination]$ sequences and then randomly sample from these sequences so the network will learn from time invariant samples. For the DDQN the replay buffer / memory size was 2,500 $[State, Action, Reward, New\ State, Termination]$ sequences.

Because the agent was receiving state data in the form of images, convolutions neural networks were required. The requirement for using on board resources for computations led to the selection of the MobileNetV3 architecture. It is fast, and has significantly less parameters than a comparable residual neural network design while achieving similar results on the ImageNet dataset. Pretrained weights were used on the convolutional layers. Because the state data had five channels and the typical RGB image only has three, the first two channel weights were duplicated for channel four, and five. This technique was shown to preform better than random normal weight initialization [13]. A mean and standard deviation of 0.5 were used to normalize the state data consistent with how MobileNetV3 handles images.

The agent's decisions was broken into seven discrete actions. The agent could move five meters in the $x, y, z, -x, -y, -z$ or rotate $90°$. The agent would select one of these actions at random, gradually overtime the randomness of selection was reduced to a minimum of 10% meaning 90% of the time the agent was exploiting the maximum reward while 10% of the time it was still exploring. During testing of the agent the probability of random action was set to zero.

3) Agent State Data: The drone's state was a five channel image. The first channel provided the drone with distance sensor data to aid in the decision making process.

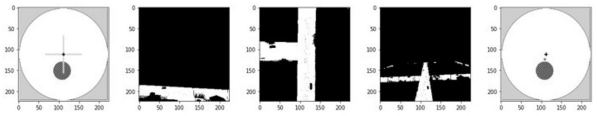

Fig. 11. Agent state with binary road segmentation.

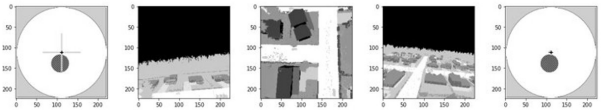

Fig. 12. Agent state full segmentation class ID's shown

The 100 meter radius around the drone was shown in white with hashed circles representing the no fly zones. At the center of the image was a plus sign shape where each of the four line represented the distance to an object within 40 meters. if an object was shown with 40 meters one of the four lines would shorten. If the object was within 7 meters an 'x' would be displayed. The color of the distance sensor lines was related to the drone's altitude.

The second, third, and four channel of the image were segmentation maps for the front, bottom, and rear camera's respectively. Initially in training the images were binary and returned white for roads and black for all other class ID's as seen in Figure 11. After 800 episodes of training the state images transitioned to including other class ID's such as the sky, trees, houses, grass, etc. The full segmentation state image can been seen in Figure 12.

The fifth and last channel in the state image was the location map. At the center was the drone's current position. The home location is shown as a small circle, other drones were shown as a star. The explored locations of the current or other drone show up in a grey-scale line where the color is mapped to the reward at those locations. Any no fly zones were displayed as a hashed circle. The environment was build such that several no fly zones could be added to any given map. the map data shows a 100 meter radius from the drone. An illustration of the map with a legend can seen in Figure 13.

4) Agent Reward: The drone/agent reward structure was made from several different functions. The first reward function was for drone altitude (z-height) as seen in Figure 14. The reward exponentially penalizes the drone for flying too low or flying towards the limit of 120 meters. Electrical controls were also put in place to prevent the drone from being able to fly above the 120 meter limit. 30 meters was selected as the optimal flight height because it was sightly taller than the average telephone pole height which would have the drone avoid being tangle in telephone wires but also close enough to the road to properly observe road

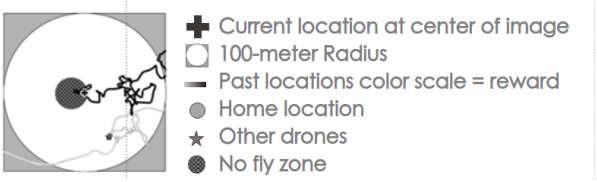

Fig. 13. Location Map details

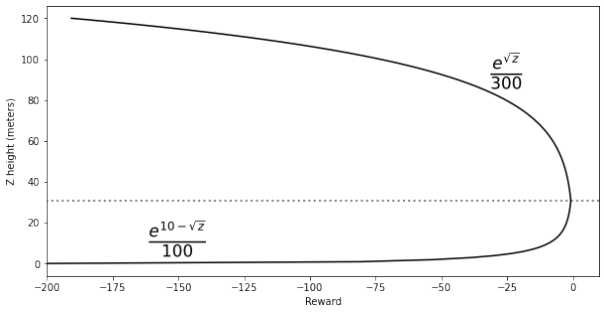

Fig. 14. Height reward

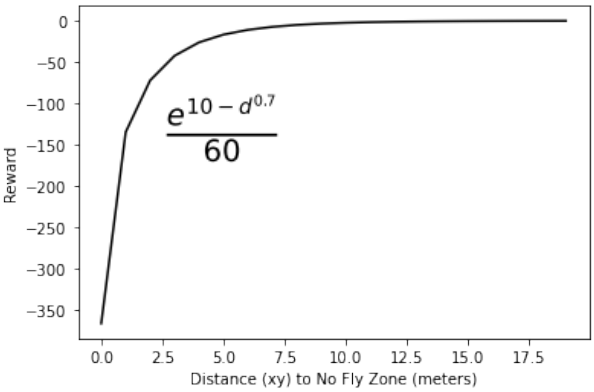

Fig. 15. Penalty for being close to a no fly zone. -4,000 was the penalty for entering the no fly zone

obstructions. Although a road obstruction classifier was not implemented at this stage of the project the planned reward for finding one was set to +100 points.

To motivate the agent to explore a penalty was incurred for being within 5 meters of any previous x, y position it or any other drone it is in contact with has visited. This penalty was additive, -5 points for each repeating location. The penalty kicked in after the first 4 actions. Given that actions were 5 meters location changes and the roads in the simulation were approximately 20 meters wide, this encouraged the agent to travel along the road. The backtracking penalty was capped at -500 points.

In an emergency certain areas may be currently being worked on my first responders or airfields could be bringing the region effected by the hurricane supplies. The drone should avoid these areas and as such a penalty was incurred for approaching the area as seen in Figure 15. If the agent entered the no fly zone if would receive -4,000 points. The agent was not trained with other drones in the system at this phase of the project. However the reward for drones maintaining 100 meters of contact was implemented in the code and the penalty can be seen in Figure 16. The drone was slightly penalized for being too close to other drones as the positions would be similar to previous positions and would result in duplicate obstruction classification. The agent was exponentially penalized as it got beyond 50 meters from another drone with large penalties kicking in at 80 meters and beyond.

The road reward was a 100x100 pixel box at the center of the downward facing camera on the drone. For initial training the reward was the percent of pixels classified

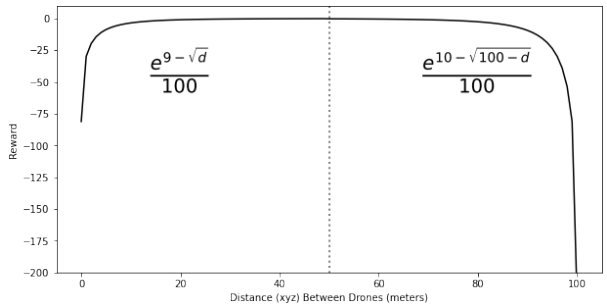

Fig. 16. Reward for the distance between drones

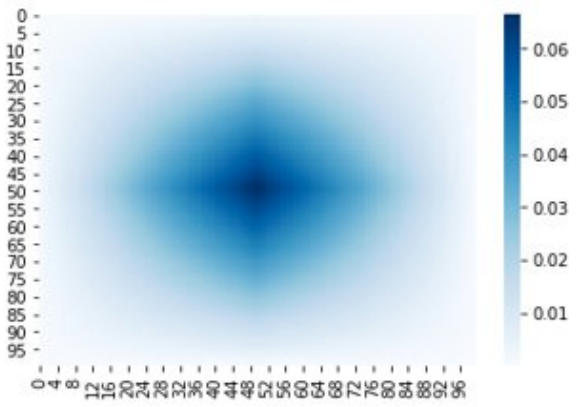

Fig. 17. Reward for the drone being over the road. 100x100 pixel box in the center of the drone's bottom camera

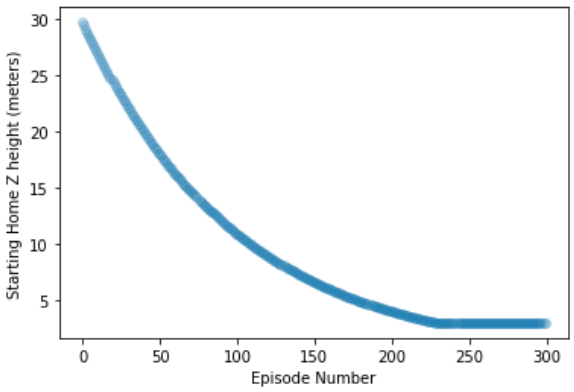

Fig. 18. Starting home height of the drone during the 10 second curriculum

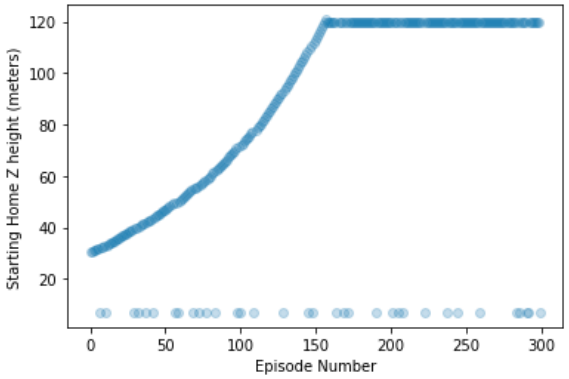

Fig. 19. Starting home height of the drone during the 15 second curriculum

as road such that 61% equated to +61 points. After the policy was learned the road reward increased such that the center pixels were weighted more than ones farther from the center as seen in Figure 17. The values seen in the figure were multiplied by +100.

5) Episode Completion: The episode lengths was dependant on where in the learning curriculum agent was. Episode time ranged from 10 to 60 seconds. A timer stated at the beginning of each episode, the start time was compared to the current time during as the environment stepped through the state reward tuples. The episode ended once the time ran out.

The episode ended in the event of a collision. The AirSim environment had an API for collision detection which was called at each time step. An imminent collision also ended the episode. The distance sensors in the front,right, left, rear, and below the drone were used to monitor imminent collisions. If an object was within 1 meter or less of the drone the episode ended.

Run away penalties were monitored. If an individual time step penalty was greater than -10,000 the episode was terminated. If the cumulative reward for the episode was greater than -100,000 then the episode ceased.

6) Agent Curriculum: Learning the optimal reward from the environment proves challenging due to the large number of interactions required. Data sampling in a sequence for the purpose of learning a problem that may otherwise be too difficult to learn from scratch is referred to as curriculum learning [12]. Curriculum learning has been shown to be effective and was deployed for this problem. The agent was tested throughout each phase of the curriculum to confirm the desired learning was taking place.

To start the episode length was set to 10 seconds and only the height reward seen in Figure 14 was used. The camera data produced only road segmentation as seen in Figure 11. The agent was trained for 250 episodes. The episode length was then increased to 15 seconds and trained for an additional 250 episodes for a total of 500 episodes. The starting height was at the goal height initially and gradually lower for the 10 second episodes at rate seen in Figure 18. For the 15 second episodes the height starting position gradually rose away from 30 meters but with a 10% probability dropped to 7 meters as seen in Figure 19

During the first 500 episodes the agent explored randomly starting at 100% of the and decremented by 0.0001 each action until the agent was taking 10% random actions.

After the agent learned the optimal height, the episode length was increased to 20 seconds with the full reward structure. The image segmentation again only produced roads as seen in Figure 11. The road reward was the percent of pixels that referred to the road in the 100x100 pixel box in the center of the bottom camera. The reward was such that if 61% of the pixels were road then the reward was 61. The drone agent was trained for 100 episodes. This processes was repeated with an increased episode length of 35 seconds at 100 episodes and then 40 seconds at 100 episodes for a total of 800 episodes of

training.

The agent was again trained for 40 second episodes with the full image segmentation as seen in Figure 12 and a road reward structure as seen in Figure 17. Training was done for 300 episodes. This process was then repeated for 60 second episode lengths for 300 episodes. In total the agent learned for 1400 episodes for a total time of 12 hours and 42 minutes.

As the agent moved farther from the home position it continued to take random actions with a probability of 10%. Once the episode length was changed a new the agent would enter new areas and do little exploring, to combat this the epsilon greedy policy was reset for the time in the new area. For example if the agent went from 40 to 60 second episodes, the agent would follow the optimal policy for 90% of the actions for the first 40 seconds. For the remaining episode time it would act randomly starting at 100% of the time and decrement the randomness by 0.001 each action until it again reached 10% exploration. The end episode exploration probability was tracked between episodes such that the following episode it would resume exploration with the probability it left off at.

E. Path Planning

The goal of the path planning subsystem is two-fold. First, given a list of GPS coordinates of target road blockages (obtained from drone exploration), prioritize the order to visit each goal state from the command center such that the total distance is minimized. Second, generate point-to-point planners between consecutive goal states. Ideally, these point-to-point planners would be optimal to minimize the total distance covered by first responders, but another consideration is computation time.

In order to test our implementation of various search algorithms, we needed a set of sample maps that corresponded to real-world roads/paths. We converted city maps to a binary map from Google Maps using the Static Map API. Goal prioritization was accomplished as follows:

1) Add the command center to the queue. Starting at the command center, find the shortest straight-line distance to each road blockage and select the one with the shortest distance. Add this to the queue.
2) Find the shortest straight-line distance to each road blockage from the second entry in the queue. Add the blockage with the shortest distance to the queue.
3) Keep repeating this until all blockages are accounted for.

We initially decided to implement A* search algorithm due to its completeness and optimality. The pseudocode is given in Fig. 20.

The heuristic used for sorting and to determine the next node to explore is the sum of the distance from the start node to the current node and the Euclidean distance from the current node to the goal node. Due to the nature of the map, which often contains straight line paths, the new nodes to explore were set to 1 m and 5 m in all 8 directions around the current node (total of 16 new nodes considered each time a node was explored). This provided a good balance between turning corners and traversing straight roads. In practice however, we noticed significant variation

```
FORWARD_SEARCH
1   Q.Insert(x_I) and mark x_I as visited
2   while Q not empty do
3       x ← Q.GetFirst()
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' not visited
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

Fig. 20. Pseudocode of basic forward search algorithm. The modification for A* search lies in the GetFirst() command, which specifies a best first sort.

```
RDT_BALANCED_BIDIRECTIONAL(q_I, q_G)
1   T_a.init(q_I); T_b.init(q_G);
2   for i = 1 to K do
3       q_n ← NEAREST(S_a, α(i));
4       q_s ← STOPPING-CONFIGURATION(q_n, α(i));
5       if q_s ≠ q_n then
6           T_a.add_vertex(q_s);
7           T_a.add_edge(q_n, q_s);
8           q'_n ← NEAREST(S_b, q_s);
9           q'_s ← STOPPING-CONFIGURATION(q'_n, q_s);
10          if q'_s ≠ q'_n then
11              T_b.add_vertex(q'_s);
12              T_b.add_edge(q'_n, q'_s);
13          if q'_s = q_s then return SOLUTION;
14      if |T_b| > |T_a| then SWAP(T_a, T_b);
15  return FAILURE
```

Fig. 21. Pseudocode of a bidirectional rapidly-exploring random tree (RRT).

in computation time from our implementation of A*. Our hypothesis for this is that some optimal paths involve taking non-direct paths to the goal (including some side-streets) that are not explored until our sorting heuristic sufficiently explores the wrong path. Furthermore, the 1 m and 5 m set distances did not always work well in all maps and had difficulty in rounding sharp corners.

Due to the graph-like structure of the map, we decided to consider implementing a rapidly-exploring random tree (RRT). The optimal modification to this search algorithm (RRT*) was considered but we wanted to see if we could get sufficiently optimal paths without the need for a complete planner. We found RRT worked relatively well and decided to see if the search performance could be improved by implementing bidirectional RRT. The idea here is to create two trees (list of nodes and vertices), one from the start node and another from the goal node. The pseudocode for the bidirectional RRT search is given in Fig. 21.

In practice, we found this worked well in terms of computation time (around 7x less time on average compared to A*). However, there was still an issue with the optimality of the path. The main reason for this was due to the random directions the tree was exploring, resulting in occasionally having nodes in side-streets that were parents of nodes that were on the generated path. In order to smooth this, once the path was generated, we modified

it as follows:

1) Check if there is a valid straight-line, collision-free path from the first to the last node (node 1 to node N).
2) If so, return the path consisting of the first and last node only (path: [1,N]). If not, check if there is a valid straight-line, collision-free path from the first to the second to last node (from 1 to N-1).
3) Keep decrementing this final node searched by 1, until a valid path is found, say at the kth iteration. Once found store the path from node 1 to N-k.
4) Once found, search if there is a straight-line, collision-free path from the new start node to goal nodes (node N-k to node N). Repeat this until the last node N is reached and connect all the generated paths.

V. Results and Discussion

A. Controls

Several iterations of the controller were developed. The first iteration concentrated on improving the tracking performance in separate degrees of freedom. First the hover height was turned. Next pitch and roll were re-tuned then finally the yaw tuning was updated. The performance of the initial controller is demonstrated in the first sequence flown in the video in the appendix. The second iteration improved the control with enabling the camera to track the direction of flight. This was demonstrated in the second flight shown in the video. Each of these took the path and treated it in a piece wise fashion therefore the paths traversed by these controls were not applicable to be compared with the final two iterations. The third iteration of the controller included using a Dubin's curve to transition from one path to the next. The results of the tracking using this control are shown in Figure 22. A capture of the controller in mid-turn is shown in 24. This control performed well as far as maintaining straight line flight and proper orientation even through fast stops. It however struggled with tracking through turns. The final version of the controller incorporated yaw and roll mixing in order to produce a coordinated turn. The plot of the turn using this controller is shown in Figure 23. A capture of the controller in mid-turn approximately in the same position as the previous capture is shown in Figure 25. The addition of the coordinated improved the tracking through the turn without impacting the performance of the controller in other areas.

B. Reinforcement Learning

Once the drone completed the 10 second training curriculum where only the height reward was turned on, the model was then loaded into a new location and tested. The test simulation initially ran for 30 seconds with the drone starting on the ground. The average reward of ten tests was plotted over time in Figures 26 below.

In Figure 26 the drone started low and learned to gain altitude, however because the episode length was short the drone over shot the desired position. Note that in the simulation recorded height values are negative.

The frequency of actions taken over the ten 30 second tests were plotted in Figure 27. The drone learned to climb

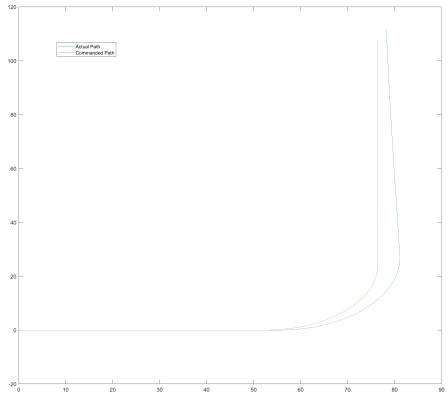

Fig. 22. Path using Dubin's curve without yaw / roll mixing.

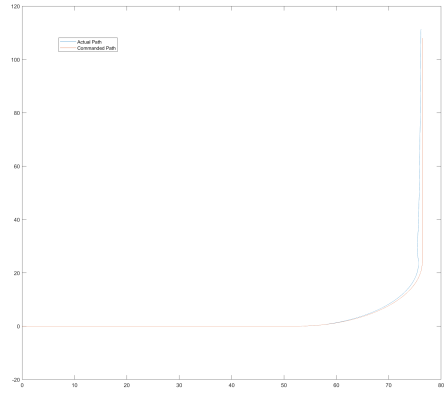

Fig. 23. Path using Dubin's curve with yaw / roll mixing for a coordinated turn.

Fig. 24. Turn using Dubin's curve without yaw / roll mixing.

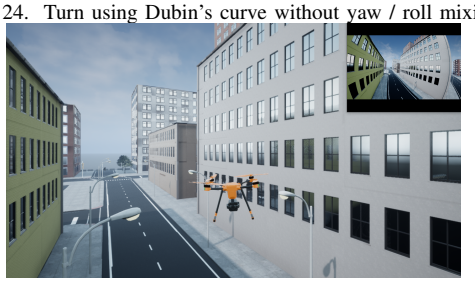

Fig. 25. Turn using Dubin's curve with yaw / roll mixing for a coordinated turn.

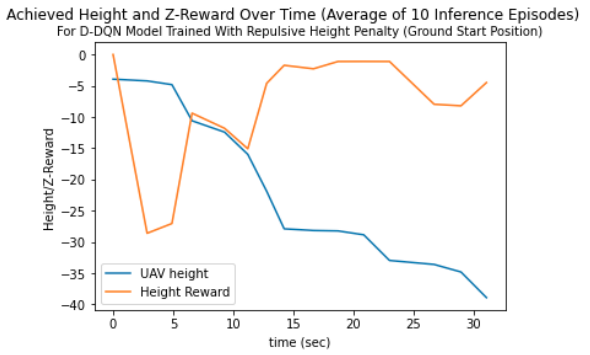

Fig. 26. Z height and reward of drone when trained for the 10 second curriculum. All non height reward was removed and the test simulation was run for 30 seconds from the ground. Note the Z height in the simulation is negative

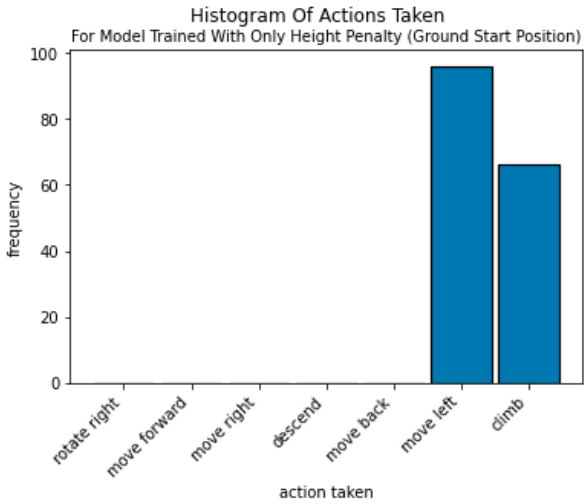

Fig. 27. Histogram of action take for the 10 second curriculum where the drone started from the ground

and had a preference for going left. A no fly zone in the simulation was added but because there was no reward or penalty associated with entering the no fly zone the agent did not avoid it. The average X, Y position the agent took was plotted over the environment road and no fly zones which can be seen in Figure 28.

Similar to the 10 second curriculum the process was repeated for the 15 second curriculum. This time in stead

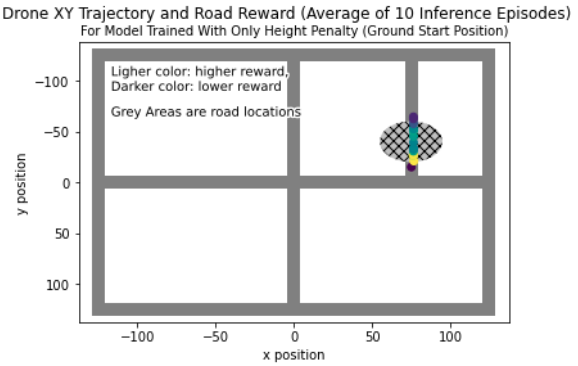

Fig. 28. Average X,Y position of drone for the 15 second curriculum when starting on the ground

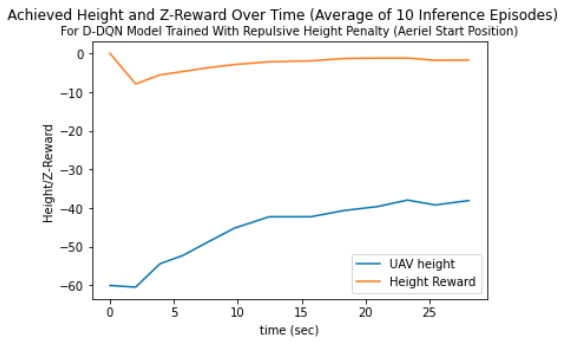

Fig. 29. Z height and reward of drone when trained for the 15 second curriculum. All non height reward was removed and the test simulation was run for 30 seconds from an aerial height of 60 meters. Note the Z height in the simulation is negative

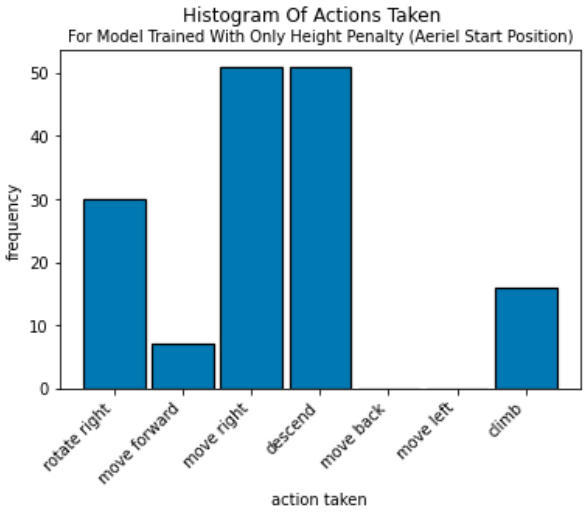

Fig. 30. Histogram of action take for the 15 second curriculum where the drone started from an aerial height of 60 meters

of starting form the ground the UAV started at 60 meters in the air. The average reward of ten tests was plotted over time in Figures 29.

In Figure 29 the drone started too high and lowered itself to the optimal height of 30 meters. Again note that in the simulation recorded height values are negative. The actions taken by the drone were then plotting in a histogram seen in Figure 30.

With the added 5 seconds of training the drone learned the optimal height and had time to randomly explore. The agent primarily descended but along had time to move forward, right and rotate. The average X, Y position the agent took was plotted over the environment road and no fly zones which can be seen in Figure 31. It can be seen from the figure the drone avoided the no fly zone by random chance.

All rewards and penalties were turned on and the agent was trained for 20 seconds. The road reward was the percent of pixels that referred to the road in the 100x100 pixel box in the center of the bottom camera. After training was complete the agent was tested in a new location ten times. Each test lasted for 90 seconds so the agent had 70 seconds of new and never explored before experience. The X, Y potion of the drone was plotted in Figure 32.

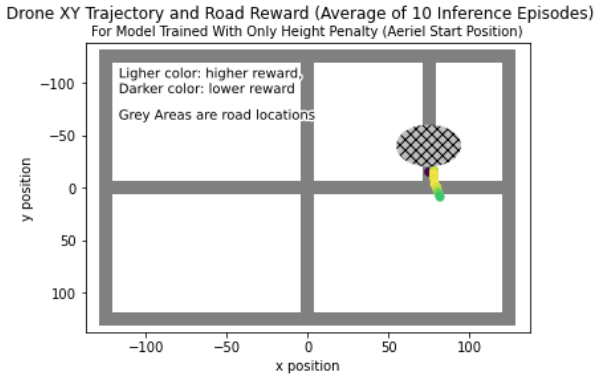

Fig. 31. Average X,Y position of drone for the 15 second curriculum when starting from an aerial height of 60 meters

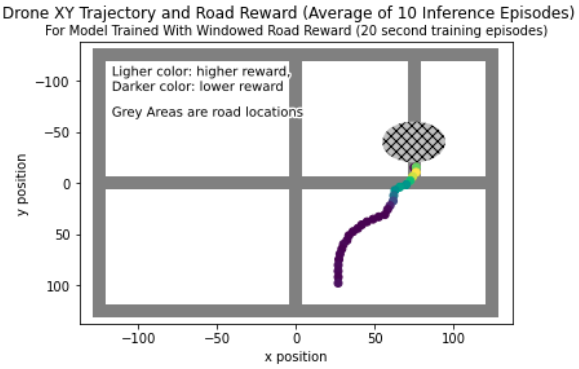

Fig. 32. Drone average X, Y position when trained for 20 seconds with all rewards and penalties turned on. The road reward was set to percent of pixels associated with the road

The agent did a good job of initially following the road but after the 20 second mark begins getting off course. The agent can be seen moving back and to the right of the map in Figure 33.

The agent used the 20 second curriculum model as a starting point and continued to train for an additional 100 episodes with each episode having a length of 35 seconds. The road reward again was the percent of pixels

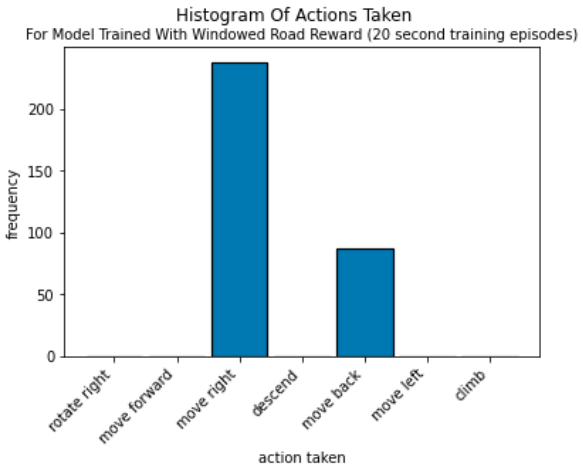

Fig. 33. Histogram of action take for the 20 second curriculum where all the rewards are turned on and the road reward was set to percent of pixels associated with the road

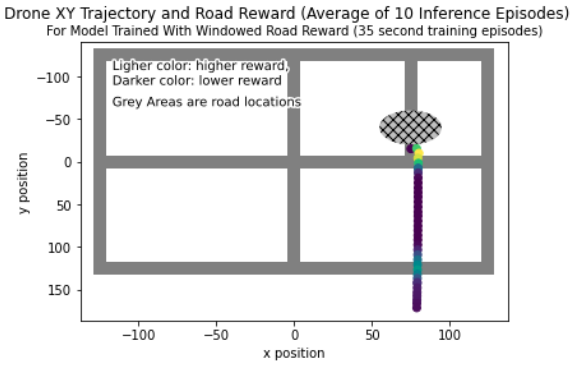

Fig. 34. Drone average X, Y position when trained for 35 seconds with all rewards and penalties turned on. The road reward was set to percent of pixels associated with the road

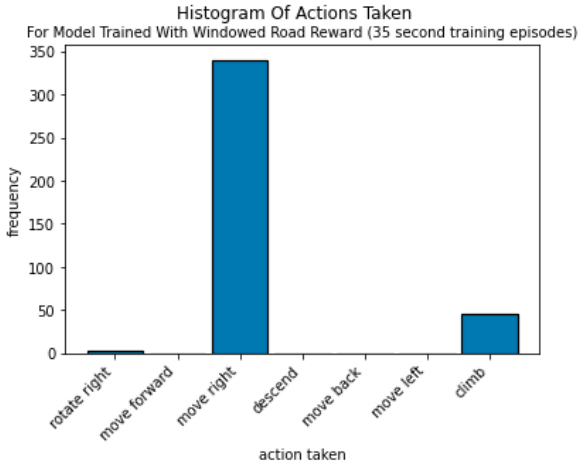

Fig. 35. Histogram of action take for the 35 second curriculum where all the rewards are turned on and the road reward was set to percent of pixels associated with the road

that referred to the road in the 100x100 pixel box in the center of the bottom camera. After training was complete the agent was tested in a new location ten times. Each test lasted for 90 seconds so the agent now had 55 seconds of exploration beyond its training episodes. The X, Y potion of the drone was plotted in Figure 34.

The agent essentially learned to move only right as seen in Figure 35. This is was the inspiration for resetting the epsilon greedy exploration for the delta time between the starting model and the current episode length.

The agent used the 35 second curriculum model as a starting point and continued to train for an additional 100 episodes with each episode having a length of 40 seconds. The epsilon greed exploration was reset from 20 to 40 seconds. The road reward remained the percent of pixels that referred to the road in the 100x100 pixel box in the center of the bottom camera. The same test setup as previous was performed. The X, Y potion of the drone was plotted in Figure 36.

With the epsilon greedy reset the agent learned to move right and forward as seen in Figure 37. The agent had 50 seconds of exploration beyond the trained episode length and struggled to make the turn when the road made a 'T'.

The agent used the 40 second curriculum model as a starting point and continued to train for an additional 100

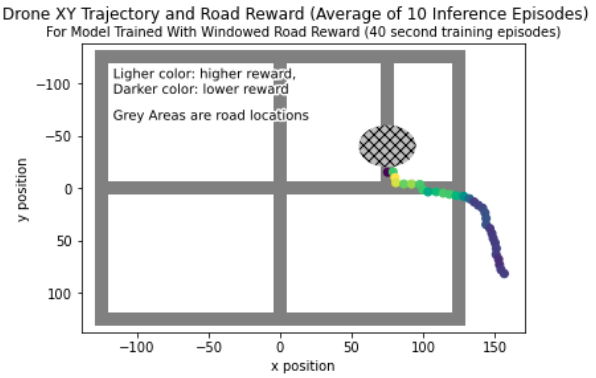

Fig. 36. Drone average X, Y position when trained for 40 seconds with all rewards and penalties turned on. The road reward was set to percent of pixels associated with the road

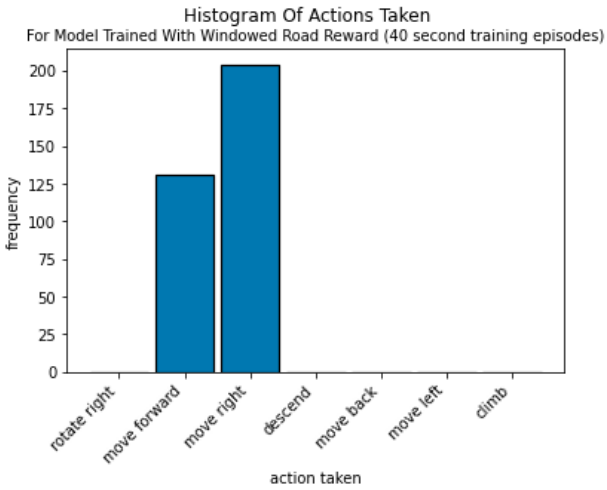

Fig. 37. Histogram of action take for the 40 second curriculum where all the rewards are turned on and the road reward was set to percent of pixels associated with the road

episodes with each episode having a length of 60 seconds. The epsilon greed exploration was reset from 40 to 60 seconds.The agent over-fit and continued to move forward when the road made a 'T' as seen in Figure 38.

For the agent to continue to learn, two parameters were changed. First the full image segmentation was

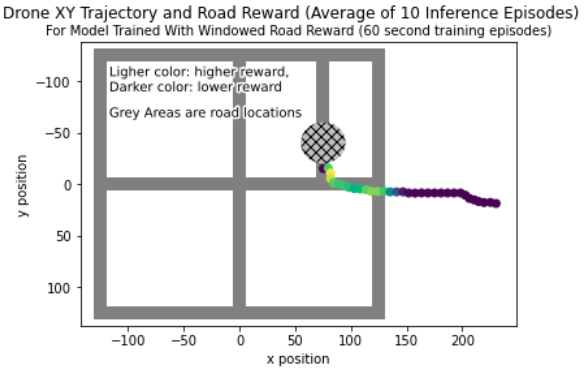

Fig. 38. Drone average X, Y position when trained for 40 seconds with all rewards and penalties turned on. The road reward was set to percent of pixels associated with the road

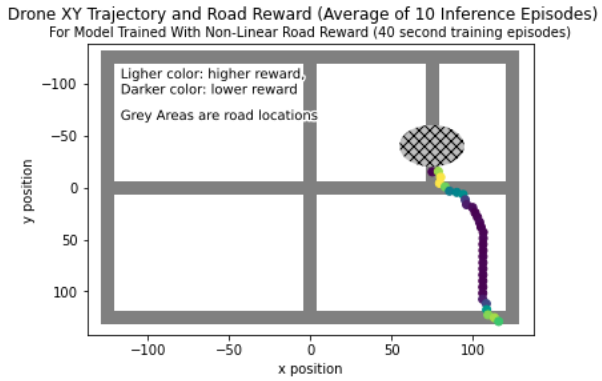

Fig. 39. Drone average X, Y position when trained for 40 seconds with all rewards and penalties turned on. The road reward was set to nonlinear reward seen in Figure 17 multiplied by 100 and summed

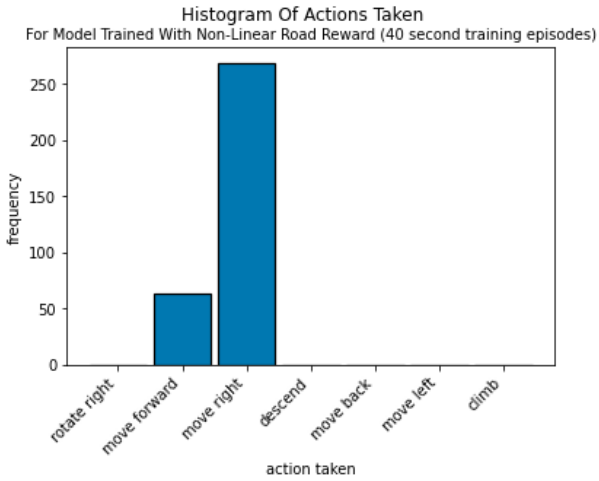

Fig. 40. Histogram of action take for the 40 second curriculum where all the rewards are turned on and the road reward was set to the nonlinear reward seen in Figure 17 multiplied by 100 and summed

implemented to include the sky, tree, houses and lawns etc. Secondly the road reward was changed from the percent of pixels in the 100x100 center window of the of the bottom camera to the non-linear road reward that weighted pixels closer to the image center with higher values than the ones farther away as seen in Figure 17. The values in Figure 17 were multiplied by 100 and summed to calculate the reward.

The agent used the 40 second curriculum model as a starting point and train for 300 episodes with each episode having a length of 40 seconds. Because the difference in state input data the agent was retrained to take random actions starting at 75% of the time and decrementing by 0.0005 until a value of 10% was reached. After training was complete the agent was tested in a new location ten times. Each test lasted for 90 seconds so the agent again had 50 seconds of exploration beyond its training episodes. The X, Y potion of the drone was plotted in Figure 39.

The agent learned to move forward and to the right as seen in Figure 40. The UAV had high variation in it's path but on average did not make it to the 'T' in the road.

The agent used the 40 second nonlinear road reward curriculum model as a starting point and continued to train for an additional 300 episodes with each episode having

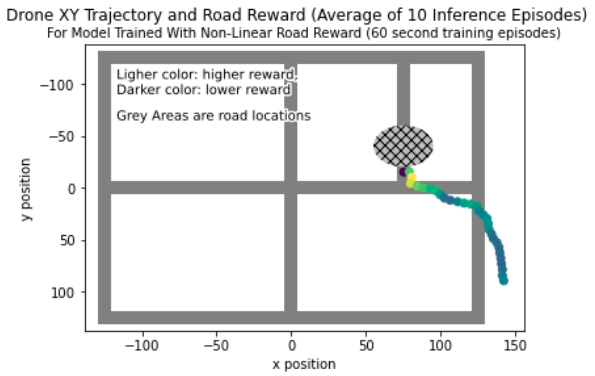

Fig. 41. Drone average X, Y position when trained for 60 seconds with all rewards and penalties turned on. The road reward was set to nonlinear reward seen in Figure 17 multiplied by 100 and summed

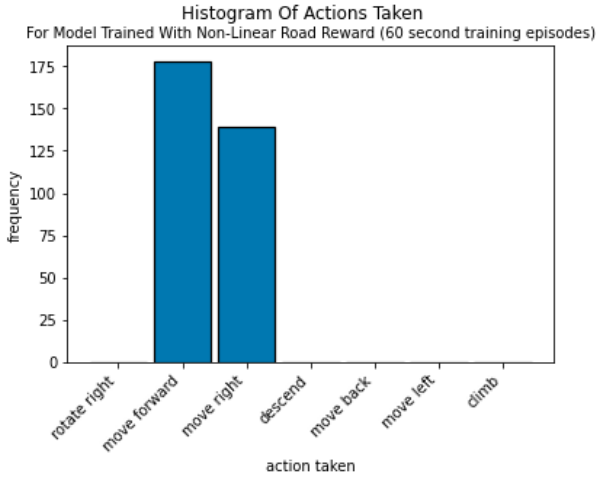

Fig. 42. Histogram of action take for the 40 second curriculum where all the rewards are turned on and the road reward was set to the nonlinear reward seen in Figure 17 multiplied by 100 and summed

a length of 60 seconds. The epsilon greed exploration was reset from 40 to 60 seconds of each episode. The road reward remained the nonlinear reward described in the 40 second non linear road reward curriculum. Again after the training was complete the agent was tested in a new location ten times. Each test lasted for 90 seconds so the agent now had 30 seconds of exploration beyond its training episode length. The X, Y potion of the drone was plotted in Figure 41.

The agent learned to move forward and to the right more evenly as seen in Figure 42 but was shown to be able to navigate the the turn in the road.

This method of incremental training improvement has shown promise as the direction to continue to explore going forward. The next steps would be to have the drone operate in new location and different environments. The AirSim package offers an urban environment that works exclusively on windows that could be further explored. Implementation details would need to be worked through because the code was written on Linux and it cannot be guaranteed the operating systems are fully compatible. Additional training data would provide a rich and more robust model that is ultimately better at following the road, avoiding obstacles, and detecting road obstructions.

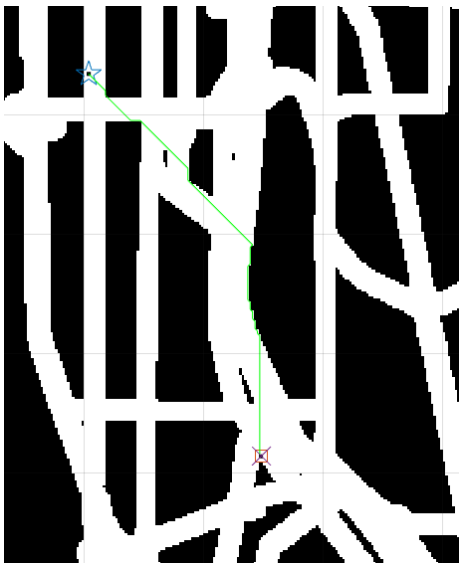

Fig. 43. Example of A* planning search in a binary map of Seattle. Here we initialize the start (red square) and goal (blue star) randomly. Optimal path (based on the allowable actions) is shown in green.

Multiple drones interacting with the environment and training on one computer would require hardware upgrades to further training. The goal before adding in multiple drones was to have a single UAV able to operate correctly. The method detailed here has shown success and the next step would be to add in the reward function seen in Figure 16, update GPS history data with shared drone information, and share current drone positions when within range. The code for this is written but without more powerful hardware it is currently untested.

C. Path Planning

Here we will share some results from both the A* and bidirectional RRT search algorithms we implemented. Fig. 43 demonstrate an instance of using A* for a single goal. Note that this path is optimal under the allowable actions (1 m and 5 m in 8 directions around each node). Decreasing these values helped generate shorter distance paths, at the expense of computation time. Fig. 44 demonstrates the smoothed paths generated after bidirectional RRT on 3 random goals. Each path is colored differently (red, green, blue) for clarity.

VI. Conclusion

Reinforcement learning was used to train a UAV to follow a road and search for road obstructions. An advanced reward structure was developed that was learned via curriculum learning. A five channel image that leveraged camera data that was segmented to show roads, distance sensors and mapping of previously explored areas. From the state data it was shown the drone learned the optimal height and to follow and explore roads. Multiple drones trained in a more diverse environments are the next step in the design of the system. More powerful hardware is required to implement the network while simultaneously running the simulation. This design showed promise as the proof of concept and the methodology should be continued in later design phases.

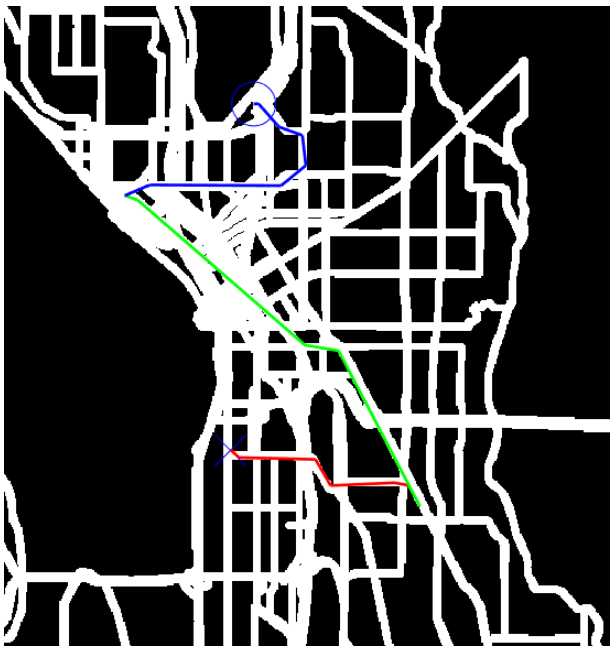

Fig. 44. Example of bidirectional RRT planning search in a binary map of Seattle. Here we initialize the start (blue cross) and 3 goals randomly. Goal prioritization determines the order and 3 point-to-point paths are generated shown in red, green, and blue.

For Path planning, we implemented two search algorithms to generate point-to-point paths on a binary map of real-life cities. The performance of A* and bidirectional RRT were evaluated and we found that searching in the grid-like environment benefited the use of bidirectional RRT. Furthermore, we implemented smoothing to generate shorter length paths resulting in a computationally inexpensive search algorithm for a dense grid-like environment.

Sampling methods such as Probabilistic Roadmaps (PRM) could work well for applications where multiple paths are queried for the same static environment and could be explored in future work.

REFERENCES

[1] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," arXiv:1802.09477 [cs, stat], Oct. 2018, Accessed: Dec. 05, 2020. [Online]. Available: http://arxiv.org/abs/1802.09477.

[2] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," arXiv:1509.02971 [cs, stat], Jul. 2019, Accessed: Jul. 06, 2020. [Online]. Available: http://arxiv.org/abs/1509.02971.

[3] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," p. 7.

[4] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning," IEEE Signal Process. Mag., vol. 34, no. 6, pp. 26–38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.

[6] M. Wiering and M. van Otterlo, Eds., Reinforcement Learning, vol. 12. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[7] Silver, David, et al. "Deterministic Policy Gradient Algorithms." International Conference on Machine Learning, PMLR, 2014, pp. 387-95. proceedings.mlr.press,http://proceedings.mlr.press/v32/silver14.html.

[8] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," May 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[9] "Hurricane costs," NOAA Office for Coastal Management. [Online]. Available: https://coast.noaa.gov/states/fast-facts/hurricane-costs.html: [Accessed: 16-Apr-2022].

[10] Epic Games, (2019), Unreal Engine, Retrieved from https://www.unrealengine.com

[11] J. Collins, S. Chand, A. Vanderkop and D. Howard, "A Review of Physics Simulators for Robotic Applications," in IEEE Access, vol. 9, pp. 51416-51431, 2021, doi: 10.1109/ACCESS.2021.3068769.

[12] Narvekar, Sanmit and Peng, Bei and Leonetti, Matteo and Sinapov, Jivko and Taylor, Matthew E. and Stone, Peter, "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey" 2020 Retrieved from https://arxiv.org/abs/2003.04960

[13] Shah, Rutav and Kumar, Vikash "RRL: Resnet as representation for Reinforcement Learning" 2021 Retrieved from https://arxiv.org/abs/2107.03380

[14] R.Beard, and T. McLain, "Small Unmanned Aircraft: Theory and Practice", Princeton University Press, 2012

[15] R. Leishman, J. Macdonald, R. Beard, and T. McLain," Quadrotors and accelerometers: State estimation with an improved dynamic model," IEEE Control Systems, vol. 34, no. 1, pp. 28-41, 2014

[16] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

[17] LaValle, Steven M.. "Rapidly-exploring random trees : a new tool for path planning." The annual research report (1998)

[18] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research. 2011;30(7):846-894. doi:10.1177/0278364911406761

VII. APPENDIX

A. Reinforcement Learning

All code for the reinforcement learning agent can be found: https://github.com/Post-Obstruction-Assessment-Capstone/Reinforcement-Learning

The drone success road follow behavior video can be found: https://www.youtube.com/watch?v=b2sTqRlmgoA

B. Controls

All code for the dynamics and controls can be found: https://github.com/bkwells87/DroneSimulation

Video demonstrating the progression of the feedback controller and dynamics simulation can be found: https://youtu.be/YQ-M$_{PyCv}$0E

C. Obstruction Detection

All code generated for obstruction detection can be found: https://github.com/Post-Obstruction-Assessment-Capstone/Route-Obstruction-Detection

A video demonstrating the detection of an obstacle within a test AirSim environment can be found: https://youtu.be/GJ$_AXwldamY$

D. Image Segmentation

All code generated for image segmentation can be found: https://github.com/Post-Obstruction-Assessment-Capstone/Drone-Road-Segmentation

A video demonstrating the segmentation of an object class within a test AirSim environment can be found: https://youtu.be/r18uvHY2Fos

E. Path Planning

Matlab code for the bidirectional RRT path planning and binary obstacle maps for cities can be found here: https://github.com/Post-Obstruction-Assessment-Capstone/Path-Planning