Deep Reinforcement Learning for Aerial Obstacle Avoidance using Monocular RGB Images

Submitted To

Dr. Nuria González-Prelcic
Naeem Akl

Prepared By

Armand Behroozi
Ali de Jong
Sean Kirmani
Yuriy Minin
Josh Minor
Taylor Zhao

EE464 Senior Design Project
Electrical and Computer Engineering Department
University of Texas at Austin

Spring 2018

CONTENTS

CONTENTS (Continued)

TABLES

FIGURES

EXECUTIVE SUMMARY

This paper explores the design of a system for autonomous aerial robot navigation using reinforcement learning (RL) using only monocular RGB images. We do path planning via a learned implicit geometric model of the world instead of explicit 3D scene reconstruction. Our solution pairs a neural net trained for obstacle avoidance using RL with a proportional integral derivative (PID) controller that helps the drone reach a goal position.

Our development happened exclusively in simulation which allowed us to collect many samples without worry of damaging the environment or the drone. The process began with training the drone to navigate randomly generated mazes and after initial success we chose to move to a more realistic environment. We moved to a model of our team members' apartment in which most of our training and testing occurred. Despite working exclusively in simulation, we always kept transfer to a real drone in mind, adding domain randomization to combat learned simulation bias.

The model is trained on image data and its output is combined with the output of the PID controller to chose the next action. The PID controller tries to turn the drone and fly directly toward the goal. The model, however, attempts to avoid obstacles and is often at odds with the PID controller. We try to balance the two to deviate from the direct path when necessary, but still reach the goal efficiently.

Evaluation of the model was promising, showing that our solution is on par with other work in the industry. We evaluated the rate of successfully reaching the goal, qualitative observation of learned behaviour, and unit tests of particularly difficult navigation tasks. Our solution is in no way ready to be commercialized, but does show promising results as a research endeavor.

We were able to achieve our goals within the given time and budget constraints. Our focus was to create a low cost solution by only using a single RGB camera for input data, this allowed us to use only about 16% of the allotted budget for purchasing the drone. Throughout the development process we evaluated the scope and goals of our project, and ultimately were able to create a complete system for training and testing within the course of an academic year.

Autonomous UAVs pose many safety and ethical concerns which we considered in the design choices we made. Our main consideration was drone collisions with a static indoor environment. Hopefully our solution would be safe and comfortable for users and observers if actually commercialized.

Improvements to this project could be made by implementing a universal value function approximator (UVFA), hindsight experience replay (HER), and autoencoding depth. UFVAs add parameterizable goals to our policy inputs. The idea behind HER is to replay
each episode with a different goal than the one the agent was trying to achieve. Lastly, training with ground truth depth data in simulation would allow the model to encode that into its edge weights, but could still make predictions from only image input.

1.0 INTRODUCTION

Our project was to explore the design of a system for autonomous drone navigation using only input data from a single camera. The purpose of this document is to summarize the results of our research and present it in a concise but detailed manner. This project was put forth by Dr. Nuria Gonzalez who was seeking to develop systems for autonomous drone navigation. We were given the opportunity to explore the area in the direction that we saw fit for the purposes of a research project. In particular we chose to explore the area of developing end to end models using state of the art deep reinforcement learning strategies. Our system operates as a black box, going directly from images to controls, an AI for obstacle avoidance.

We will begin by explaining the design problem as well as the requirements for creating such a system for autonomous navigation. Included is a detailed list of the project's goals and motivations as well as a brief summary of prior art on the subject of autonomous indoor navigation tasks. Next will follow a summary of our design using deep reinforcement learning as well the iterative process we went through in developing our model. There can found a detailed description of the neural network architecture as well as training and simulation system design. Our design went through several iterations as we continued to evolve the architecture and inputs/outputs of our model, the largest change to which was adding a PID controller to work in tandem with the output of our network.

We then describe the process of evaluating and testing system performance as we continued our iterative process. In the end, we find that we achieve success rate near state of the art systems developed by other groups. Finally, we enumerate several time concerns as well as safety and ethical considerations before concluding with a series of technical suggestions on future development of our existing system. We believe that local navigation using monocular images is possible using existing methods.

2.0 DESIGN PROBLEM

This section will summarize our design space as well as our functional and operational goals and requirements. The functional requirements of our system focus on developing viable point to point navigation achieving comparable results to other research in the field while having the underlying system being our novel end to end model solution instead of more established methods. We have set various other requirements for our project that involve creating subsystems that help in the development of such a system. These include the simulation subsystem as well as the testing framework and domain randomization module of our project.

2.1 Prior Work in Aerial Obstacle Avoidance

Most navigation algorithms can be divided into two major approaches: data-driven methods and mesh-driven geometric methods [1]. The classical robotics approach most commonly used are geometric approaches. Geometric methods create a 3D model of the environment and make decisions assuming the model is noiseless and that its information is completely accurate. This also depends on traditional expensive and heavy additional depth sensors such as LIDAR or depth cameras. Data-driven algorithms have become more popular more recently and are being explored in academia as machine learning trends are at a high. The benefit of the data-driven methods is that they assume a noisy, uncertain environment, which helps them generalize to new situations. The trade off is that machine learning methods are harder to implement.

Looking at geometric solutions our group doubts the robustness of such models due to the assumptions they make about the operating environment which are vastly different from the real world [1,2]. In general, we believe that the data driven approaches will be more robust to unique environments due to their ability to operate with noisy data and in uncertain scenarios. With the open ended format of our research project we decided that working on a data driven approach would be a more challenging and interesting approach to tackle, but would also help explore and further research into aerial obstacle avoidance for robotics in general. This was a major motivator in deciding our eventual approach.

2.2 Design Objectives

Our functional requirements consist of three distinct modules and layers of implementation. We have the navigation system which will depend on a simulator submodule, the test bench framework for evaluating performance as we iterate over models, and finally we have the domain randomization module which works closely with the simulator submodule. Together these three components work together to create our iterative pipeline as we make changes in the model's parameters in order to explore what works best for our use case.

Our primary system design goal is successful point to point navigation in a simulation environment. Since we will be developing a data driven approach using deep reinforcement learning, we will need an efficient environment for data collection. Collecting 100,000 plus samples for every model that we would like to train in the real world would be impractical and dangerous while the drone has not learned a decent obstacle avoidance policy. A practical simulation framework needed to be made in order to make training efficient. Our neural model samples point of view images from the drone in simulation as the input to its model network architecture. This report will go into greater detail into the inner working of that system in Section 3.

After we trained the model, we needed a way to evaluate it. A suite of unit tests seemed like a good option for black box testing. Our unit tests give the drone a series of start and end goals that require it to navigate non-trivial obstacles. We wanted a simple system that evaluates performance and allows us to compare the model over time and compare different models against each other. The suite needed to be easily extensible, making it easy to add new test cases.

Despite working only in simulation, we wanted to keep real world transfer in mind by implementing domain randomization. This module allows the model to train with a diverse set of texture and lighting configurations. The goal of domain randomization is not to create a realistic environment, but rather to give the drone lots of different variations. While this increases the problem's complexity in training, it would reduce the expected performance drop when transfering to a real drone flying in a real environment.

Table 1. Specifications and relative metrics.

Metric	Description	Minimum Requirements	Stretch Goals
Success Rate	What percentage of time does the drone reach its goal.	50% in simulation.	50% on a live drone.
Tolerance to new environments	Measure the average difference in performance when changing the training environment.	The average success rate should be comparable with newly introduced textures and lighting.	The average success rate should be comparable with newly introduced geometry.
Check for non-trivial learned patterns	Ensure that the drone has learned constructive behavior.	Check that the drone has not learned to stop midair or fly in small circles.	Compare performance to other solutions in research or the marketplace.

There are three primary metrics that are used to measure success in the performance of our system enumerated in Table 1. Success rate is the overall metric for how often the drone reaches its destination. It is the primary overall measurement for the success of our system. Tolerance to new environments is another important factor for evaluating the eventual ability of our drone to transfer to real life. Even though we were not ready to transfer to a real environment due to limitations in time, it remains an important problem specification. Finally, it is important to include a sanity check for non-trivial behavior such as flying in circles. Behavior such as this was found to be very common in early iterations of our design. Table 1 enumerates our minimum requirements and our eventual hopes for the success of our system.

3.0 DESIGN SOLUTION

This section will go into a technical deep dive of our autonomous navigation solution. Our final solution configuration and interaction chart is shown here in Figure 1. To summarize, the simulation environment, built in Gazebo interacts with the neural network and PID controller, which then work in tandem to plan the drone's trajectory through the environment. During training, we backpropagate changes in the network architecture dependent on a chosen reward function and our reinforcement learning algorithm. When we are done training a model, we can feed it into our unit testing framework in order to compare the performance of model changes to previous iterations.

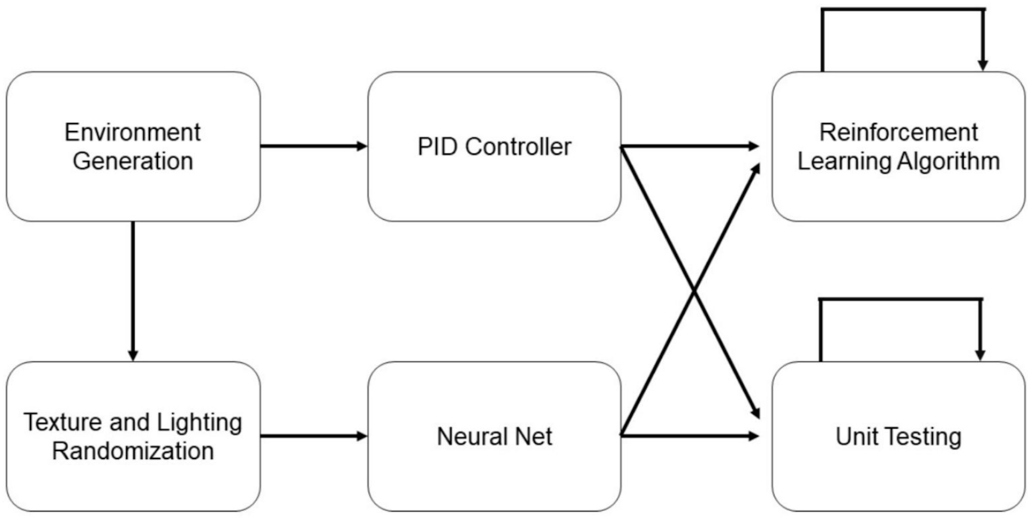

Environment Generation
PID Controller
Reinforcement Learning Algorithm
Texture and Lighting Randomization
Neural Net
Unit Testing

Figure 1. System architecture.

3.1 Simulation Environment

We leverage a powerful existing framework for robotics called Robot Operating System (ROS) which integrates closely with Gazebo, a physics and graphics simulator built specifically for robotics applications. Since ROS is an open-source modular environment, we leverage the state estimation package for localization and drivers from the Technical University in Munich's robot vision group for the Parrot AR Drone. Within Gazebo, we have created a scale-accurate model of two of our members' apartment for the drone to train in. In addition, textures and lightning are randomized for each point to point flight through the environment to help the model be more robust to new environments and aid in transfer to the real world.

3.1.1 *Apartment Model*

We have created a to-scale representation of an apartment as an environment for the drone to train in. An overhead image of the simulated area is shown in Figure 2. This was done in part to eventually transfer to real-world once the policy reaches a satisfactory level of performance. The apartment contains 4 primary areas for the drone to spawn in, the dining room, kitchen, living room, and the laundry room.

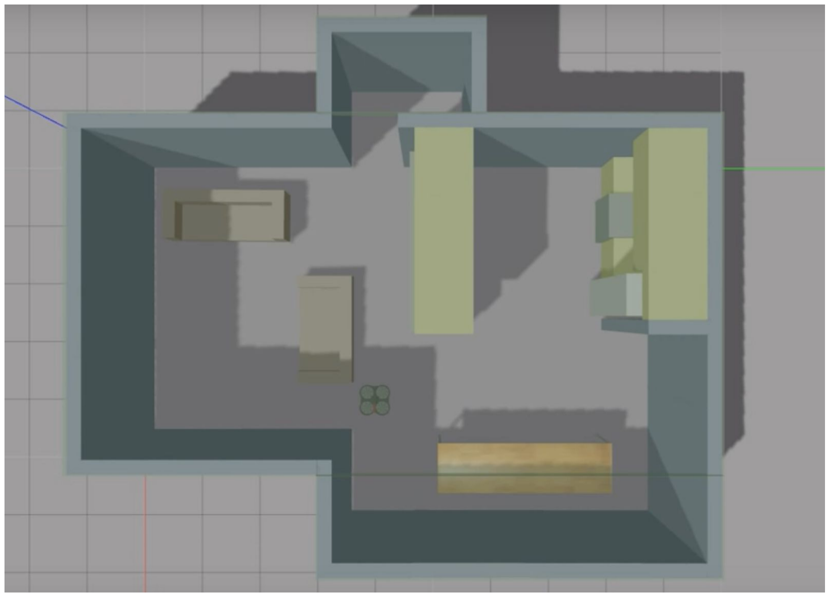


Figure 2. Simulated apartment environment.

3.1.2 *Texture and Lighting Randomization*

Running alongside the training system we have a python script which randomizes the lighting and texture of all surfaces in the model world. The simulation randomization script is called to randomize the scene lighting and object materials. Our simulation has one directional light source, and in order to randomize lighting, we change its roll, pitch, and yaw each to a value between 0 and 30 degrees. This points the light in different directions, which causes various lighting conditions in the apartment. Then, the script randomly chooses from a library of textures that we have generated and makes objects appear as if they are made from different materials. For transfer to real world scenarios it is important to have a diverse set of training data, not necessarily photo-realistic images. [3,4,5]

3.2 Reinforcement Learning

In reinforcement learning, we have some state space S and a action space A. If at time t we are in state $s_t \in S$ and take action $a_t \in A$, we transition to a new state $s_{t+1} = f(s_t, a_t)$ according to a dynamics model $f : S \times A \rightarrow S$. The goal is to maximize rewards summed over the visited state: $\sum_{t=1}^{T-1} r(s_t, a_t, s_{t+1})$. Model-based RL algorithms assume you are given (or learn) the dynamics model f. Given this dynamic model, there are a variety of model-based algorithms. We consider a method that performs the following optimization to choose a sequence of actions and states to maximize rewards:

$$max_{\ a_{1:T-1} s_{1:T}} \sum_{t=1}^{T-1} r(s_t, a_t, s_{t+1})\ s.t. f(s_t, a_t) = s_{t+1}$$

We learn our transition dynamics $f_\theta(s_{t,} a_t)$ parameterized by a vector θ over some horizon H. At time t in state s_t, the next action a_t is selected by solving the finite-horizon control problem:

$$argmax_{A_t^H} E[\sum_{h=0}^{H-1} \gamma^h R(\widehat{s}_{t+h}, a_{t+h})] \; s.t. \; \widehat{s}_{t'+1} \sim f_\theta(\widehat{s}_{t'}, a_{t'}), \; \widehat{s}_t = s_t$$

Since planning for large H is expensive and often undesirable due to model inaccuracies, planning is typically done in a model predictive control (MPC) fashion in which the optimization is solved at each time step, the first action from the optimized action sequence is executed, and the process is repeated. Standard model-based algorithms then alternate between gathering samples using MPC and storing transitions (s_t, a_t, s_{t+1}) into a dataset D, and updating the transition model parameter vector θ to maximize the likelihood of the transitions stored in D.

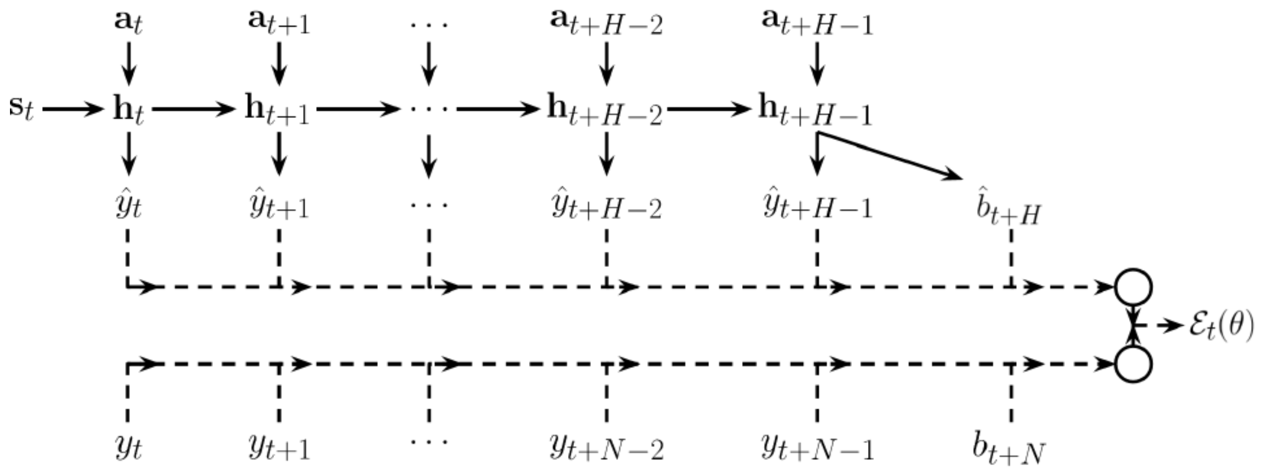


Figure 3. A generalized computation graph for model-free, model-based, and hybrid reinforcement learning algorithms.

Our navigation computation graph is defined as follows. The model first outputs a planned trajectory $a_{t:t+H-1}$, and then we output the probability we will collide at each future state $y_{t:t+H-1}$ as well as the probability our best case scenario action will collide in a b_{t+H}. We want to choose actions that maximize utility in the future b_{t+H}. Fundamentally, this model is used to predict the probability that a given sequence of actions will collide in the future.

In addition, defining our time horizon determines how far into the future we attempt to plan our trajectories without collision. In the purely model-free version, we would set H=1, or one time unit into the future. However, this is often impractical as we care if our actions will cause us to collide in the future beyond just a single-step before collision. The model-free approach can be thought of as the greedy optimization problem. As $H \rightarrow \infty$, our method is completely model-based, however as mentioned above, planning for a large H is expensive and our model inaccuracies accumulate.

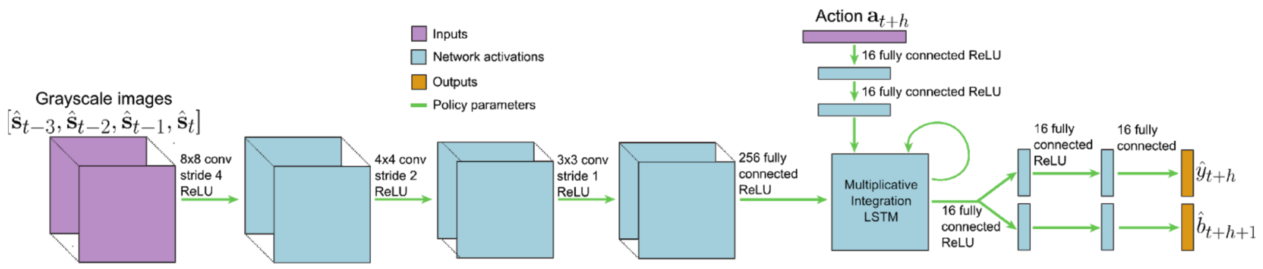


Figure 4. Recurrent neural network computation graph for robot navigation policies.

The network takes as input the past four grayscale images, which are processed by convolutional layers to become the initial hidden state for the recurrent unit. The recurrent unit is a multiplicative integration LSTM. From h = 0 to H-1, the recurrent unit takes as input the processed action and the previous hidden state, and produces the next hidden state and the RNN output vector. The RNN output vector is passed through final layers to produce the model outputs $\widehat{y}_{t+h}$ and $\widehat{b}_{t+h+1}$. See Figure 3 for a diagram representation of the network architecture.

Given the outputs our our navigation computation graph, we now need to define how to task of collision-free robot navigation is encoded into the policy evaluation function J. If the model outputs are values, then the policy evaluation function is simply the value $J(s_t, A_t^H) = \sum_{h=0}^{H-1} \gamma^h \widehat{y}_{t+h} + \gamma^H \widehat{b}_{t+H}$. If the model output quantities are collision probabilities, then the policy evaluation function needs to somehow encourage the robot to move through the environment. We assume that the robot will be travelling at some smooth speed, and therefore the policy evaluation function needs to evaluate which actions are least likely to result in collisions $J(s_t, A_t^H) = \sum_{h=0}^{H-1} -\widehat{y}_{t+h} - \widehat{b}_{t+H}$.

In addition to the model horizon, we must decide the label horizon N. The label horizon N can either be set to the model horizon H, or to some value $N > H$. Although setting the horizon N to be larger than the model horizon H can increase learning speed, as N-step Q-learning often demonstrates, the learning algorithm then becomes on-policy. This is an undesirable property for robot navigation because we would like our policy to be able to trained with any kind of data, including data gathered by old policies or by exploration policies. We therefore set the label horizon N to be the same as the model horizon H.

Finally, to train our model using dataset D, we need to define the loss function between the model outputs and the labels. Using samples $(s_t^H, A_t^H, y_t^H) \in D$ from the dataset, if the model outputs and labels are values, the loss function is the standard Bellman error $\varepsilon_t(\theta) = \| \sum_{n=0}^{N-1} \gamma^n y_{t+n} + y^N b_{t+h} - J(s_{t,} A_t^H) \|_2^2$, in which $b_{t+H} = max_{A^H} J(s_{t+H,} A^{H)}$. If the model outputs are collision probabilities, the loss function is the cross entropy loss

$\varepsilon_t(\theta) =- [\sum_{h=0}^{H-1} y_{t+h} log(\widehat{y}_{t+h}) + (1 - y_{t+h}) log(1 - \widehat{y}_{t+h}) + b_{t+H} log(\widehat{b}_{t+H}) + (1 - b_{t+H}) log(1 - \widehat{b}_{t+H})]$,

in which $b_{t+H} = min_A \frac{1}{H} \sum_{h=0}^{H} \widehat{y}_{t+H+h}$ represents the lowest average probability of collision the robot can achieve at time $t + H$. We note that these probabilities can also be learned using a mean squared error loss, but in practice using the cross entropy loss seems to be more stable.

We opted to use an actor-critic method for our approach. When using policy gradients, or actor-only methods, we are optimizing directly on the action we are trying to execute. When using value functions, or critic-only methods, we are optimizing our actions with respect to the value of our visited states. Actor-critic methods combine both these ideas by learning both a policy function and a value function. At each training iteration, we update the value of our states based on our sampled data, then update our actor to determine the best action to take given our new estimate of the value of our states. The intuition behind the method is that we as people pick an action to execute, but also have an internal critic that models the value of our own actions. As we gain more experience, our model of the world improves and we update our actions appropriately to operate on our underlying model of the world. In practice, these methods tend to reduce variance in our action space. There are many popular variants of actor critic methods such as Deep Deterministic Policy Gradients (DDPG) [6], Advantage Actor-Critic (A3C) [7], Hierarchical Actor Critic (HAC) [8], Actor-critic with Experience Replay (ACER) [9], and Soft-Actor Critic (SAC) [10] to name a few. Our method most closely resembles A3C.

In our implementation, our robot state space $S \in \Re^{9216}$ is the scaled the intensity map of our previous four input frames down to 64×36, and our robot action space $A \in \Re^{25}$ represented a 5×5 quantization of the domain from $[-1, 1]$ to represent a deviation in the gaz and turning speed. The actual action we execute however is a combination of both our PID output to go in a straight line from our start location to the goal location, and the intuition is that our network will learn to output some deviation from this line that goes around obstacles. The robot observes the current image and selects an action every $dt = 0.25$ seconds, and our model is optimal over a horizon $H = 16$, corresponding to 4 seconds. We opted to model outputs as collision probabilities, thus our labels are binary classes of whether we did or did not collide at a given timestep t.

3.3 PID Controller

This model is to be used in conjunction with a higher level navigation planner, a Proportional Integral Derivative (PID) controller. Because we are using Gazebo to simulate our environment, we are implicitly given a coordinate system that the drone can use to localize itself. While it is a difficult task to localize the drone in the same manner in reality, since all of our work is done in simulation, we decided to leave the task of real world localization to future researchers. Working on a coordinate system, we are able to define an error in distance from our current state and goal state. Our PID solution uses two separate control loops. One minimizes the error in height, while the other corrects for the drones direction and simultaneously moves it toward the goal. The implementation details can be found in the PID control implementation section below.

4.0 DESIGN IMPLEMENTATION

This section discusses the tradeoffs and implementation decisions that arose when creating and simulating the autonomous navigation system. First, we motivate the use of simulation, choice of simulation environment, and use of domain randomization. Then, the training of the neural network and its integration with the proportional integral differential (PID) controllers is examined.

4.1 Use of Only Simulated Data in Training

When first approaching the problem of training the drone, the team had wanted to use real images from the drone's camera to train its neural network. However, reinforcement learning requires the drone to crash many times as part of its training, potentially damaging the drone and its environment. Furthermore, the network would require hundreds of thousands of samples to train on, which are time-consuming and difficult to gather in real life. This makes using purely simulated images an attractive solution for gathering training data. One drawback to this approach, however, is that a model can overfit to small differences between a simulated environment and the real world, so additional work must be done to help the model perform in real environments. The next two sections discuss how we chose our simulation environment to speed up training and facilitate transfer of our model to the real world.

4.2 Simulation Environment

After proving that the drone could avoid simple simulated obstacles such as pillars in an otherwise empty world, the team wanted to create a more complex environment for the drone to train in. It was important that the environment be similar to the the one the real drone would fly in, so it could learn a strong prior of its surroundings from training. Initially, the team envisioned flying the drone in the hallways outside of Dr. Gonzalez’s office, so we created hallway worlds and a procedurally generated a random maze of hallways as shown in Figure 5.

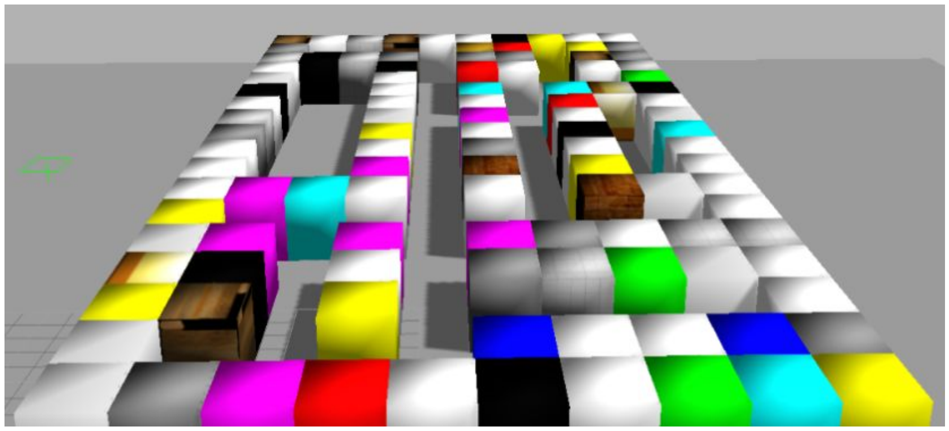


Figure 5. Randomly generated box maze.

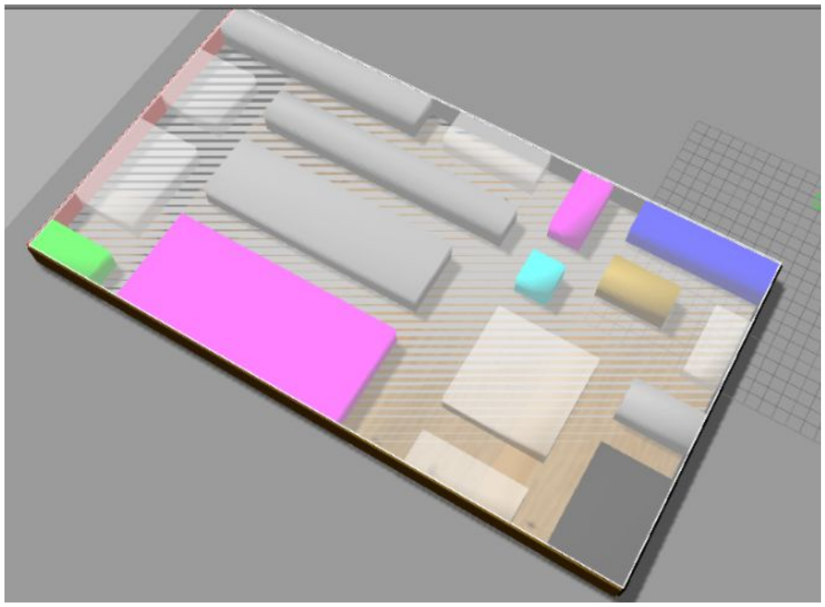


Figure 6. Randomly generated maze with merged boxes.

The maze, however, took approximately 20 seconds to render, which was much longer than a training episode. This led us to seek optimizations and merge boxes to reduce the number of rendered objects as shown in Figure 6.

With merging, the environment rendered four times faster; however, a five second rendering time still more than doubled training time. We could have continued to make improvements on the idea (e.g. generate a new maze only every 100 episodes), but we learned that we would not be allowed to fly the drone in the EERC, which led us to model an environment we could eventually fly the drone in. The team chose to recreate the apartment of two of our team members who owned a drone, so they could train a model and immediately test it if needed. We measured the dimensions of the apartment and modeled the floor plan and large furniture as shown in Figure 7. The team purposely left out smaller objects, such as pots and pans in the kitchen, so the drone could not simply memorize its training environment and would be forced to avoid obstacles whose positions it could not learn.

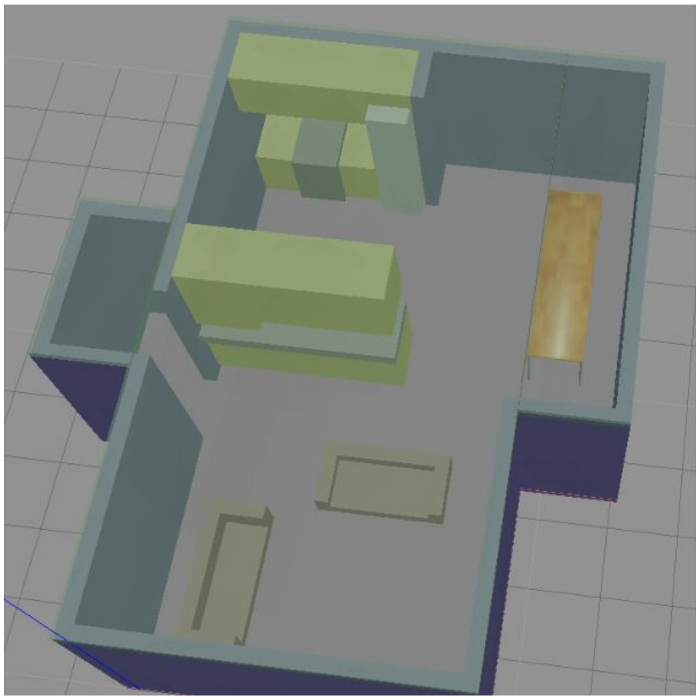


Figure 7. Apartment simulation environment.

4.3 Texture and Lighting Randomization

Even though the drone's training and real-world flight environment were similar, the team was still concerned about the performance of the learned policy outside of simulation. Prior work had demonstrated that even small differences between a simulated and real-world environment could lead to significant performance drops [3, 4]. This led us to implement a technique called domain randomization, where aspects of an environment are randomized, with the idea that enough diversity will cause the model to view the real world as just another random environment [3, 4, 5]. The team chose to only randomize scene lighting and object materials because those had shown promising results when explored in research [3, 4, 5].

Implementing domain randomization was challenging because, initially, we could only change the material of an object by deleting it and spawning a new one. This is undesirable because, as mentioned previously, more spawned objects translates to higher training time. Eventually, we were able to avoid constantly removing and spawning the same objects by passing messages between our robot software framework, Robot Operating System (ROS), and the simulator, Gazebo. Later, we were even able to avoid explicit message passing and created a plugin to change object materials that the simulator automatically checked whenever updating the world.

Although the team was able to increase the speed at which domain randomization was performed, applying domain randomization increases the complexity of the learning problem, which causes training to take five times longer to converge. This constraint coupled with the team's lack of computing resources made it difficult to properly train the reinforcement learning model. We tried to circumvent this challenge by only partially training the model when making a modification and observing how quickly the success rate increased. If given more time, the team would tackle this problem using a curriculum learning approach where the neural network is first trained on the non-randomized environment to initialize its weights and is later fine-tuned with randomization.

4.4 Training the Neural Network for Local Planning

During the preliminary stages of implementation, we envisioned a drone navigation system with two planners: a global planner to get it from its start position to destination and a local planner to avoid obstacles. The team decided to only focus on the local planner, since our physical drone did not have GPS or a method of determining its position in an environment. Due to this paradigm, the local planner was trained by spawning the drone randomly in the apartment environment and rewarding it for maximizing time without collision. The team quickly discovered that the drone just learned to spin in a circle to avoid colliding with obstacles because it had no incentive to explore the rest of the apartment. To avoid this behavior, the team decided to implement a global planner in simulation where ground truth positions can easily be extracted. At this point, the goal of the project shifted from having the working model on a physical drone to having a functional system in simulation that could be transferred to a real drone.

The global planner was implemented as two PID controllers and would cause the drone to face and move towards its goal while the local planner would output an offset from this straight line to avoid collisions along the way. The addition of a goal during training forced the drone to explore more regions of the apartment, and consequently encounter more obstacles, during each episode and made it impossible for the drone to only fly in a circle.

Training the local planner while the global planner is active, however, poses some challenges. During training, the local planner might output an offset to avoid an obstacle while the global planner fights this output and attempts to continue moving in a straight line. Despite the local planner outputting the correct action, if a high enough weight is placed on the global planner's output, the drone may still collide. Then, when the model attempts to learn from this example, it will see that its output led to a collision, so next time, it will choose a different, less optimal action. The team attempted to alleviate this problem by tuning the PID controllers' parameters to make them more under-damped to reduce the priority of reaching the goal. Another solution would be to conduct a parameter sweep of the relative weights of the global and local planners on the drone's movement to find the optimal priorities for avoiding collisions and reaching the goal. We did not attempt this, however, because we lacked the sufficient computing resources.

4.5 PID Controller Implementation

After realizing that the drone was flying in circles and was not exploring its environment during each episode, the team implemented a global planner using PID control, a commonly used control method. Since the drone has only one front facing camera and we are working in 3D space, we decided to split this complex problem into two separate pieces, utilizing two logically separate PID controllers. The first controller minimizes the difference between the target and current heights. The second turns the drone to face its goal, and then proceeds to minimize the $\widehat{x}$ and $\widehat{y}$ distance to that goal. The second controller is more complex than the first because it has to correct for an angle, theta, which is the angle between where the drone is facing and where it needs to go, as well as the $\widehat{x}$ and $\widehat{y}$ distance.

To solve this problem, our solution uses non-holonomic controls, which means that the drone must be partly facing its goal in order to begin moving in either the $\widehat{x}$ or $\widehat{y}$ directions. The forward velocity of the drone is determined by scaling a constant value by the cosine of theta mentioned earlier. Thus, as theta approaches 0 and the drone turns to face its goal, its velocity increases. Conversely, as theta approaches 90 degrees and the drone is facing further away from its goal, its velocity decreases. After tuning the PID controllers in our implementation, the drone correctly faces and moves towards a target, allowing us to train with start and goal positions.

The the governing formula for PID control is $u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$.

Using this model we structured our code to replicate its functionality. In this formula the next effort u at time t, is determined by the sum of three separate functions, each with an associated constant denoted with a K. The first component is the proportional response, which is simply the constant K_p times the current error at time t. The next is the integral component, which looks at the summation of all the previous error up to time t, and again multiplied by a separate constant K_i. Finally there is a derivative component, which looks at the current slope of the error at time t, multiplied by a final constant K_d. The combination of these three components yields a stable state solution for both components of our control system when properly tuned. The act of tuning refers to setting the constants such that the error goes to zero as quickly as possible. We hand tuned our controllers this semester, which was difficult but ended up yielding positive results.

5.0 TEST AND EVALUATION

Since our design is essentially one large module and there are many paths that qualify as a success, we have developed three different methods to assess the model's effectiveness. The first is to evaluate the success rate gathered during the training process to evaluate how the model is progressing as it trains and whether or not it is making the right advancements and progressions over time. The second method is to observe the simulation and watch for non-trivial, learned obstacle avoidance, for example, going over the counter or around a corner. The final testing method discusses a suite of basic unit tests developed to score the model quantitatively. Evaluating the accumulated reward allows us to look at internals or our design and observation and unit tests allow us to black box test the system without knowing the internals.

5.1 Success Rate Evaluation

Once we added a PID controller the reward function ceased to be an accurate measure of the overall performance since the drone was expected to fly from point A to point B which could vary in length. We opted for a more direct qualitative measure with the success rate of the drone in reaching its objective. These metrics provide a quantitative measure of the performance of our model, and can be monitored over the course of training time.

We did see a meaningful increase in success rate during training. At the beginning of training the model we saw about a 33% success rate without any learned edge weights in our neural network.

As the model trained, however, we saw the success rate peak at around 52% and settle to 48% after 7 days of training. This indicates that our model did in fact learn meaningful behaviour and improved its performance by training on paths through the apartment with randomized textures and lightings. See Figure 8 for the graph of success rate over time. The darkened orange line is a smoothing of the actual data that allows us to better see the trend in the rates.

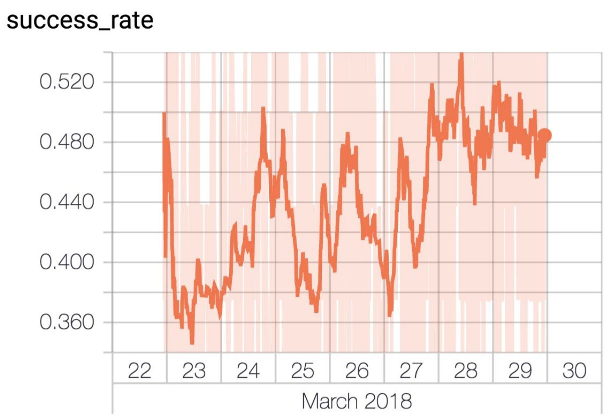


Figure 8. Success rate.

While a 50% success rate is not nearly ready for a commercializable product, it is on par with other cutting edge research. A similar research project using a long-short term memory (LSTM) as part of their network saw a 63% success rate using only RGB images [11]. Domain randomization also affects our success rate results. While it helps with transfer to the real world, it makes our results in simulation look lower since it increases the problem's complexity. If we were to overfit to the simulated apartment model we might see higher success rates but it would introduce a simulation bias that would result in a dramatic performance drop when transfered to the real world. Domain randomization makes it more difficult to learn, but ultimately makes the model more robust to handle variations in the real environment. Ultimately, a 50% success rate is an encouraging result from this initial training and is comparable to other work in the field.

5.2 Qualitative Observation of Performance

Since our model is being trained in simulation, we can visually observe its flight patterans and environment. This provides sanity checks for trivial learned patterns, and confirmation that the environmental randomization is functioning correctly. It is also encouraging to see the drone flying in meaningful patterns and avoiding obstacles successfully during the training period. We also used visual inspection to evaluate the proper functionality of the domain randomization.

Based off of our visual inspection of the generated environments,we confirmed that modifications of simulated objects was occuring correctly. In terms of obstacle avoidance, we saw some meaningful behaviours, but we still have a bit of tweaking to do. See Appendix A for an image of the drone flying between the kitchen counter and cabinets successfully. Seeing behaviours like this is encouraging but is still inconsistent enough to warrant further development.

5.3 Unit Testing

Since there are various ways to define a navigation task success and there are no standard benchmarks for systematic testing, unit tests are an attractive way to evaluate the system. By testing the drone with different navigation tasks such as “fly between two couches”, we can compare the performance of different models or the same model over time and observe how changing algorithms or model hyperparameters affects collision avoidance in various scenarios. Additionally, since reinforcement learning collects data from trial and error and the drone starting position and goal are randomized during training, two training instances with the same parameters can yield dramatically different models as slight variances compound over days of training [12]. This makes it even more imperative to have a systematic method for testing.

The unit tests are defined by a start and end position and are chosen such that the drone must avoid an obstacle in its path to reach the goal. This provides “black box” testing to evaluate system performance without assessing individual modules such as the reward or PID controller parameters. Once the overall system is tested, we can note how results differ between different models, learning algorithms, and parameters, using these differences to optimize the navigation system.We chose a set of tests that force the drone to avoid some sort of difficult obstacle. These include flying around a corner, through tight spaces, around table legs and more. A map of some difficult example start and goal positions can be found in Appendix B. Our model does have difficulty with these unit tests which led us to analyze what that means about our approach.

The results of our unit tests may seem disappointing but actually provide some insights about our design. Firstly, the success of the free space test verifies that the PID controller is functioning properly and that controls are correctly transmitted to the drone. Secondly, the concept of a start and end goal was a recent design change, so the model was previously optimizing for flight time without collision. Currently, instead of attempting to maximize flight time, the model outputs an offset from a straight line in order to avoid obstacles on the way to the goal. This means that optimizations in architecture and parameterization for the new problem definition could potentially lead to higher success rates. Thirdly, random paths through the apartment (determined by random start and end positions) are more likely to be in free space than to be obstructed by obstacles, which means that the drone acquires less experience flying in tight quarters. However, these are exactly the conditions we test it in. In order to improve model performance, the drone needs to be exposed to more scenarios during training where it must avoid obstacles to reach its goal. Lastly, although the drone fails many of the unit tests, these tests may not be representative of the difficulty of tasks that would normally be required of an autonomous drone. Instead of flying from a living room to a small laundry room that allows the drone only .25m of free space on either side, it might be more reasonable to expect it to navigate wide hallways near the ceiling, so it has little chance of colliding into humans.

6.0 TIME AND COST CONSIDERATIONS

Taken as a whole, we were able to complete our project within the allotted time and budget. Our project was largely ahead of schedule to begin with, as there was quite a bit of prototyping done before the final semester of the project had begun. Much of the development environment was configured prior to the start of implementation, such that we were all able to begin working from a fresh environment. This streamlined much of our development for this semester, however we did run into some bottlenecks. Namely, with updating our operating systems, tuning our PID controller, and introducing texture randomization. With respect to the cost of our project, from our design choices prior to this semester, the changes we made took our cost down from well over $1000, down to just $160 which aligned well with our goal of creating a low cost solution.

Although we had a well constructed development environment setup for this semester, we had to update our operating systems running in virtual machines to account for a new stable release of Gazebo which we wished to use. This turned out to be much more daunting of a task than previously expected, as many group members were unable to successfully run Ubuntu 16.04 in a virtual machine. After a week of deliberation, we decided to run this new OS on a partitioned hard drive, completely resolving any issues we were having attempting to run on a virtual machine.

Once we had a clean slate to start on with our new operating systems, the next bottleneck which slowed the progress of our project was implementing our PID controller, a much more difficult task than was previously thought. While it was relatively trivial to implement the equation aforementioned in the PID controller implementation section, tuning this controller proved to be a difficult task. We found that changing any one of the constants even by a slight amount would cause the motion of the drone to become non-smoothed, and it frequently would overshoot it’s goal. One of the reasons for this is that we were sharing constants for both pieces of the controller. We found that separating the constants out for both controllers increased the precision we had in tuning the control system. After spending a day tuning this new controller we were able to reach a steady state solution and the rest of the development could continue. All in all this puts us only a couple of days behind schedule.

Aside from the PID controller, much of our focus this semester was devoted to help generalize the model so it could transfer to the real world without overfitting to a certain room and textures. Domain randomization was difficult to get working because the documentation for Gazebo was hard to decipher and we had to read through all of their text files and code to find what we needed to change. They already had a library of colors and textures but it was very limited in quantity and we wanted hundreds of different looks (from brick to tree leaves). Lighting was easier to implement, so we were able to train our model over this semester with the small library of textures Gazebo provided, but were only able to get domain randomization finished in early April. This did set us back quite a bit, because training on a diverse set of textures and colors is reported to take 15 times as long to train a model before it reaches a steady state, if it does at all [3,4,5]. Given that this began working so late, it took so long to train models that we could only evaluate results after weeks of training. The only way we could speed up this process was to train multiple models on separate machines simultaneously, in the hopes that just one would converge and produce non-trivial learned behavior.

While our design costs were speculated to be well over $1000 at the end of last semester, by altering our design at the beginning of this semester we drastically reduced the overall cost of our project. Our only cost for the semester was the purchase of a real drone to fly. The Parrot AR Quadcopter 2.0 Edition only cost $159.99. Given that we had a budget of $1000 dollars for this project, our team's design choices to reduce the project expenditures is commendable. We considered buying a processor or GPU and attaching it to the drone, but since we only worked in simulation, this became irrelevant. In the future, purchasing a GPU to place on the drone would allow us to speed up the rate of decision making for the drone if it was flying in real life, however even with the purchase of a board like the Nvidia Jetson, our project would have still come in under budget.

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN

Autonomous drone navigation presents a wide set of possible safety and ethical considerations. Within the scope of our project, our team deliberated how to maximize the safety for our team members and those around us, as well as develop a solution that would minimize the ability for our design solution to be used for any ill intentioned actions. We outlined three points of consideration in this area; drone collisions with objects or people, airspace cluttering in the presence of many autonomous drones, and security and privacy.

Given the fact that propellers can easily slice or break objects they come into contact with, avoiding collisions was of paramount concern. In order to mitigate the possibility of drone collisions damaging people or property, we performed the entirety of our project in simulation. While we were never comfortable with transferring our model to the drone itself, we trained using texture and lighting randomization in order to minimize simulation bias with the intention of reducing the likelihood of collision induced with the uncertain conditions the drone would face in the real world.

With the increase of drone development, people fear the pervasiveness of drones in the airspace. In a study conducted in order to provide insight into how society perceives the the development and scope of civil drones, it was found that, “the most widely adopted vision of civil drone development was one of proliferation to the point where they are a near constant presence” [13]. Our team understood that the development of autonomous drones would lead to a greater number airborne drones, as a user could launch multiple drones without needing their active control. While our technology could enable more drone usage, if the uses are limited to commercial, military, and public service use, we believe there wouldn’t be extensive crowding.

Another key concern with the development of autonomous drones is security and privacy. If ill-intentioned individuals are able to remotely override drones, especially those militarily equipped, the results could be catastrophic. While data security is out of the scope of our design project, it is an issue that production grade autonomous drones will have to solve moving forward. Privacy and security of individuals in the use case of surveillance is another aspect of our project which we would consider in extensions of this project. Individuals may not want to appear in the video stream of a drone. While this may be the case for some, in the aforementioned study, it was found that, "while some felt uncomfortable with [surveillance] as an intrusion of privacy, these feelings were often balanced against the increased security offered, for example through police access to CCTV footage." Many today have come to accept the increased probing of institutions into their lives [14]

8.0 RECOMMENDATIONS

Our largest limitations exists in our inability to generalize well to many goals and the curse of sample-inefficiency. Our main recommendation for handling generalization is to remove our PID controller, and do goal-centric reinforcement learning by adding our position and goal as inputs to our policy and value networks.

One idea is to use Universal Value Function Approximators (UVFA) is an extension to DQN where there is more than one goal we may try to achieve [14]. Let Q be the space of possible goals. Every goal $g \in G$ corresponds to some reward function $r_g : S \times A \rightarrow R$. Every episode starts with sampling a state-goal pair from some distribution $p(s_0,\ g)$. The goal stays fixed for the whole episode. At every timestep the agent gets as input not only the current state, but also the current goal $\pi : S \times G \rightarrow A$ and gets the reward $r_t = r_g(s_t, a_t)$. The Q-function now depends not only on a state-action par but also on a goal $Q^{\pi}(s_t, a_t, g) = E[R_t|s_t, a_t, g]$. Training an approximation to the Q-function using direct bootstrapping using the Bellman equation and a greedy policy derived from it can generalize to previously unseen state-action pairs.

One ability humans have, unlike the current generation of model-free RL algorithms, is to learn almost as much from achieving an undesired outcome as from the desired one. Imagine that you are learning how to play hockey and are trying to shoot a puck into a net. You hit the puck, but it misses the net on the right side. The conclusion drawn by a standard RL algorithm in such a situation would be that the performed sequence of actions does not lead to a successful shot, and little (if anything) would be learned. It is however possible to draw another conclusion, namely that this sequence of actions would be successful if the net had been placed further to the right.

This intuition is formalized by the technique called Hindsight Experience Replay (HER) which allows the algorithm to perform exactly this kind of reasoning [15]. It is applicable whenever there are multiple goals which can be achieved, e.g. achieving each state of the system may be treated as a separate goal.

HER may be seen as a form of implicit curriculum as the goals used for replay naturally shift from ones which are simple to achieve even by a random agent to more difficult ones. However, in contrast to explicit curriculum, HER does not require having any control over the distribution of initial environment states. Not only does HER learn with extremely sparse rewards, it also performs better with sparse rewards than shaped ones.

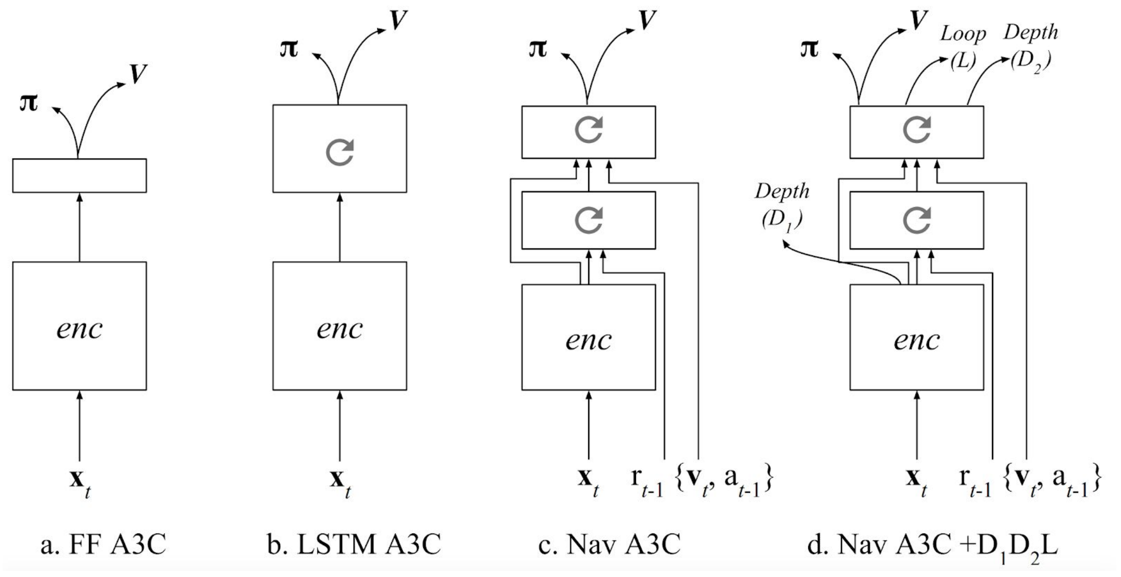


Figure 9. Different architectures for learning navigation policies in complex environments.

The final suggestion we have is to add an auxiliary loss function into our neural network to help provide some additional signal. Specifically for navigation tasks, we can draw insights from NavA3C which added a depth prediction loss into their networks in order to implicitly autoencode depth information into the policy [16]. They find that for navigation tasks, it often converges faster and better because we start to implicitly reason about which parts of the image have free-space. Another impressive feature of this idea is that when we train in simulation, we can get ground truth depth information from our z-buffer at training time, but at test time when we put our policy on the drone we are able to leverage a policy trained with depth data using only an RGB image. Figure 9b illustrates how our current system is implemented, and Figure 9c and Figure 9d illustrate how our model might change when we autoencode depth into our convolutional layers.

9.0 CONCLUSION

At the start of senior design, we were presented with a open ended question on how to achieve autonomous navigation. Our goal was to create an end to end system for local obstacle avoidance. In many ways, we succeeded in meeting specifications for point to point navigation using a novel tandem system of a PID controller and neural model. We were also able to create a simulation environment with domain randomization for other autonomous drone research groups to use. We plan on open sourcing our code so other robotics researchers can apply randomization to their own environments to train their models in simulation.

We were able to meet all of our goals set from the beginning of the semester (fly a drone autonomously in simulation from a point A to a point B). However, we weren't able to fly the drone in real-life, we were hindered by permits and limitations in model accuracy. However, we failed to meet many of our stretch goals. The most prominent of which was that we hoped to fly the drone in real life with our obstacle avoidance model onboard. Unfortunately we were not confident in our models performance for real world transfer and did not even begin with the process of transferring the model to the actual drone. This is definitely something that can be accomplished if we had more time on the project.

We are really proud of what we have developed over the course of these two semesters. Our design uses state of the art methods of developing control systems that we believe have great inherent advantages to methods in classical robotics. Unfortunately, there is still much research to be done in the field of reinforcement learning and its applications in robotics before its performance matches that of classical geometric methods.

REFERENCES

[1] J. Kober, J. Bagnell and J. Peters, "Reinforcement learning in robotics: A survey", 2013.

[2] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, S. Levine, “Uncertainty-Aware Reinforcement Learning for Collision Avoidance”, 2017.

[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World,” ArXiv170306907 Cs, Mar. 2017.

[4] F. Sadeghi and S. Levine, “CAD2RL: Real Single-Image Flight without a Single Real Image,” ArXiv161104201 Cs, Nov. 2016.

[5] J. Tobin, W. Zaremba, and P. Abbeel, “Domain Randomization and Generative Models for Robotic Grasping,” ArXiv171006425 Cs, Oct. 2017.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” ArXiv:1509.02971 Cs, Sep. 2015.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning.” ArXiv:1602.01783 Cs, Feb. 2016

[8] A. Levy, R. Platt, K. Saenko, “Hierarchical Actor-Critic,” ArXiv:1712.00948 Cs, Dec. 2017.

[9] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, N. de Freitas, “Sample Efficient Actor-Critic with Experience Replay,” ArXiv:1611.01224 Cs, Nov. 2016

[10] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” ArXiv:1801.01290 Cs, Jan. 2018

[11] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, “Cognitive Mapping and Planning for Visual Navigation,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[12] Lanctot, M. and Zambaldi, V. (2017). A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning.

[13] P. Boucher, "'You Wouldn't have Your Granny Using Them': Drawing Boundaries Between Acceptable and Unacceptable Applications of Civil Drones," Science and Engineering Ethics, vol. 22, no. 5, pp. 1391–1418, Oct. 2016.

[14] T. Schaul, D. Horgan, K. Gregor, D. Silver, "Universal Value Function Approximators," 2015 Journal of Machine Learning Research (JMLR), 2015.

[15] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, "Hindsight Experience Replay," ArXiv:1707.01495 Cs, Jul. 2017.

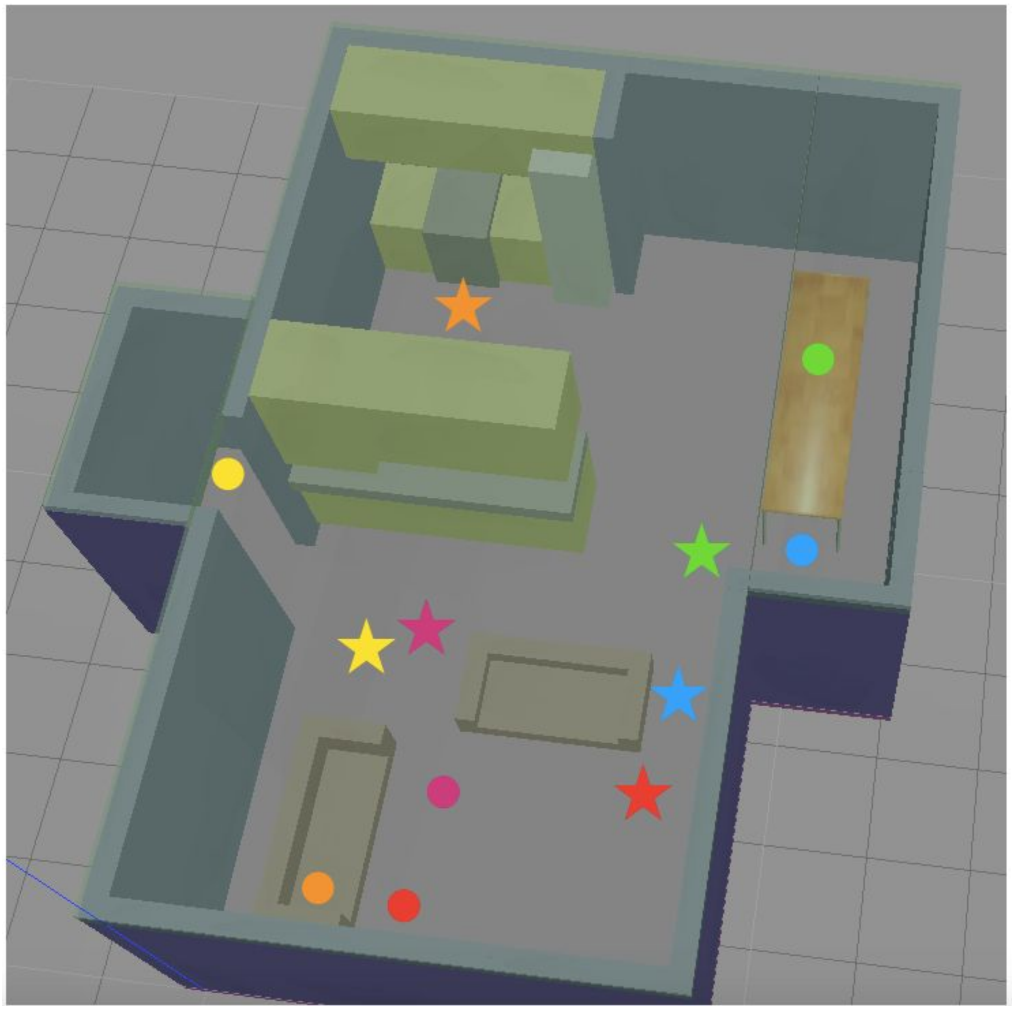
[16] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, R. Hadsell, "Learning to Navigate in Complex Environments," ArXiv:1611.03673 Cs, Nov. 2016.

APPENDIX A: NON-TRIVIAL LEARNED NAVIGATION

APPENDIX A: NON-TRIVIAL LEARNED NAVIGATION

APPENDIX B –MAP OF UNIT TESTING STARTS AND GOALS

APPENDIX B: MAP OF UNIT TESTING STARTS AND GOALS



Circle is the starting position, star is the goal position.

Color Key:

Across Living Room	🟥
Through Kitchen	🟧
Exit Laundry Room	🟨
Under Table	🟩
Around Corner	🟦
Between Couches	🟪