Nicolas Lopez Bravo

Independent Study for UCF - CRCV

UAV Path Planning Final Report

Introduction

UAV's are a becoming more useful and common in military as well as the daily life applications. In the USA, the FAA gives measures to inform hobbyists to follow certain guidelines to promote the safety usage of UAV's. Previous research as in "Autonomous navigation for low-altitude UAVs in urban areas" by Thomas Castelli et al. described a form of navigation in which it would be safer to fly UAV where they defined different safe zones on a map. The goal of that work was creating the shortest path that would go over safe zones most of the time and go over the danger zones with minimal time. In their work, they used A* algorithm [put a citation here] to compute that safest and shortest distance where they used different weights for the different zones on a map.

This independent study aimed to extend that the work of Castelli et al. to study the performance of deep reinforcement learning algorithms on performing the same tasks for UAV path planning. Similar to the approach followed in [1], in this work, we also assume a UAV with video camera flying over a given geographical region, for which both the geodetically accurate reference image of the area and the GIS data of buildings and roads in the area are available. The dataset used in safe UAV flight includes geo-locations from WPAFB, PVLabs, downtown Orlando, FL and UCF. The image used for this test is of downtown Orlando as shown in Fig 1.

Related Work

Q learning is an approach that utilizes Markov Decision Processes; the goal of Q learning [A] is to learn a policy which tells an agent which action to take under which circumstances as the agent gets experience of the environment. The work in [6] shows that Q learning converges to an optimal policy over experience. In Q learning there is a weight to rewards where rewards closer have greater value to the agent. Deep Reinforcement Learning is the application of Q learning with a neural network as the value approximation function. Deep Q Learning has then evolved to Dueling DQN [3] and Double DQN.[4] In [2] AI Safety Gridworlds by DeepMind they present a suite of reinforcement learning environments that have different factors such as safe exploration, reward gaming, self-modification. They explore the classic grid world problem by adding a twist to it.

Our Approach

In our approach, we will be tackling the path planning problem similar to the classic grid world example of AI, in which an agent must walk through cells (or in our case pixels) to achieve a goal. We take our input data as a Google API generated image as shown in Fig 1. We then scan the image and create a 2D array of rewards based on the color of the image as inked in by the Google API. The image has been stitched together from several other images to achieve a high resolution, for our purposes we have resized the image to a resolution of 300 x 300 pixels for simplicity of training.

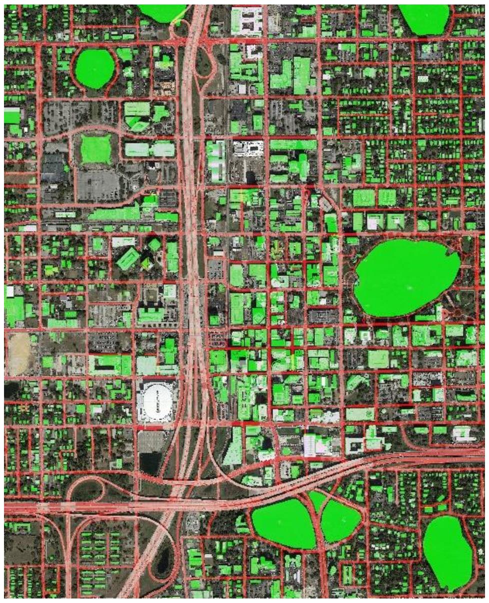


Figure 1

The architecture of this Deep Reinforcement Learning agent is that of Q learning and using a deep network to approximate the Q values. We use experience replay as a form of decoupling sequential information from path planning and being able to determine q values of future locations without bias. The input, or observation, of the agent is defined as a $\triangle$ distance from the target normalized with the pixel length of the image. We then train the network from the experienced actions and determine the Q value using the bellman equation as described in [5].

Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

[5]

Definitions

D = Replay Memory

N = Memory Capacity

A = action

R = reward

Transition = [state, reward, action, next state]

Equation 3 = Bellman Equation

Method

Fitting this algorithm to our case, we utilize the array of rewards as the environment, for which the Neural Network will determine Q values to store in replay memory. The Q values will be updated during training, we choose Q-values using an epsilon greedy policy where epsilon is 0.9. The list of available actions in our case 8 directional control.

Actions = [UP, DOWN, LEFT, RIGHT, UP-LEFT, UP-RIGHT, DOWN-LEFT, DOWN-RIGHT]

Our Pixel Ratio for these movements have been determined as 1 pixel per movement as the original pixel ratio from [1] is **0.229 m/pixel,** given the Velocity **V** = 5m/s, the minimum pixel movement is that of **1 Pixel.** When resized to 300 x 300 the pixel ratio decreases largely, however to ensure movement we bound the lower limit to 1 pixel.

Our means to achieve our goal involves creating a carefully designed Reward function that allows the agent for error but at the same time incentivizes convergence. These values have been determined empirically and have delivered the best results.

We have determined the rewards for each location as follows:

Road Reward:	- 1
Building Reward:	1
Neutral Reward:	0

Reward = array[x,y] + (D/(H*U))

H = the height of the image

U = Resolution

D = Negative Euclidean distance from the goal

If D < Straight Line distance then D = e^{d}

Experiments

In our experimental results we see the evolution of the agent throughout different reward functions. In these following images the A* algorithm's trajectory is presented in yellow, while the DQN is presented in purple.

Initially we explored a plethora of different reward values for buildings in rewards including the original 100, 20, 5 from [1], but as we will see below the current function provided the best results. We also have included results for starting/ending locations with great distances.

Initial tests with no training

In these following experimental results, we run the agent without any training to see the pure randomness that the agent follows.

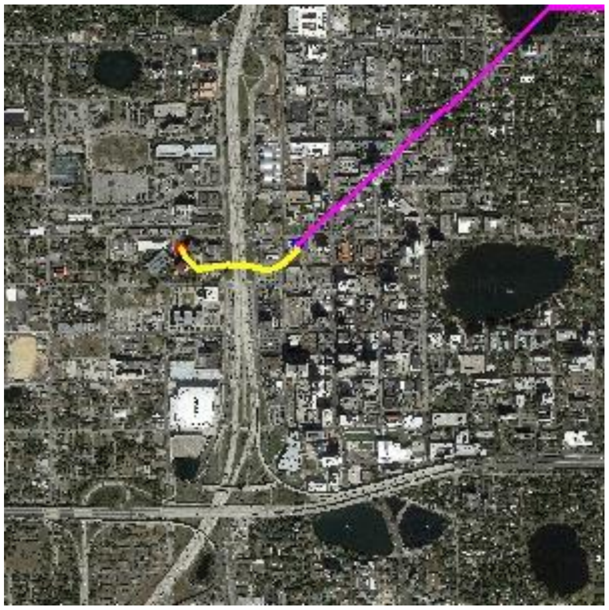


Figure 2

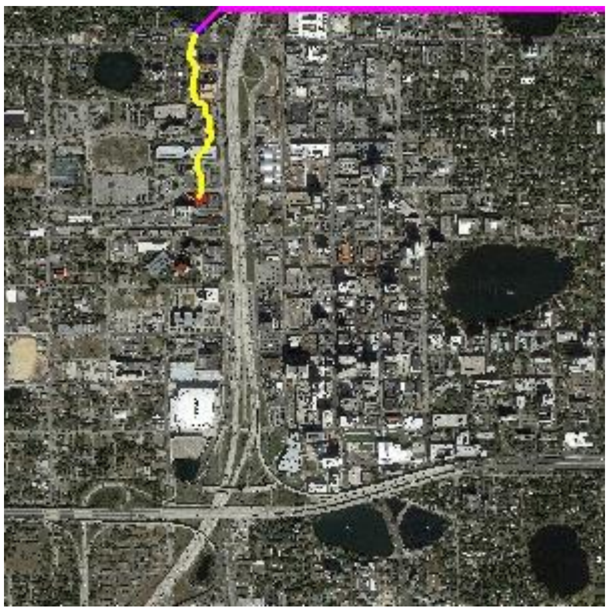


Figure 3

Final tests with present reward function

In these final images, we can see a clear overlap of the agent with the previous algorithm, these two cases by metrics the agent performed better in the static case than the previous algorithm.

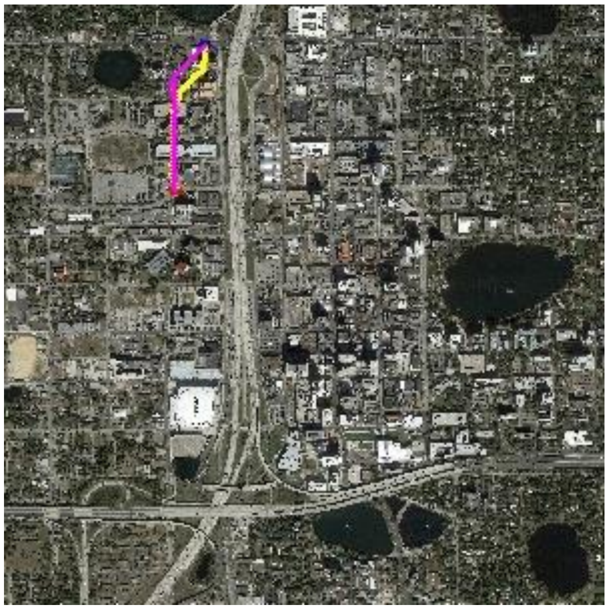


Figure 4

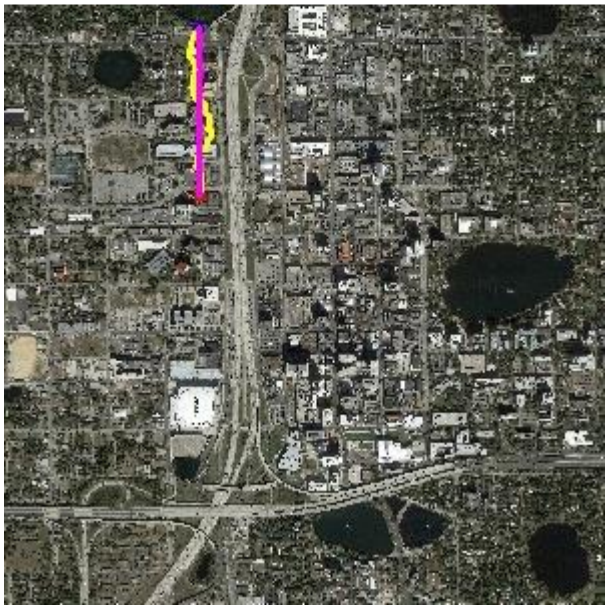


Figure 5

Tests for Long Distances

In these tests, we have colored the map as shown above with the respective reward values given, any areas colored green are considered buildings and have a positive reward of 1, while roads are color coded red and have a negative reward of -1. Transparent areas have no color

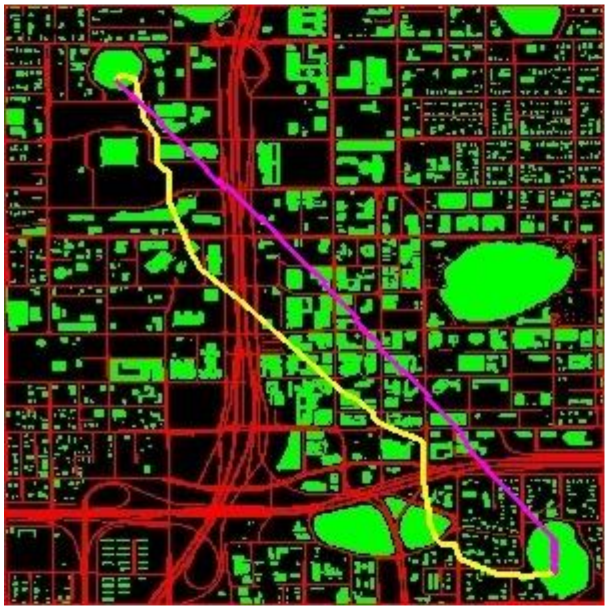


Figure 6

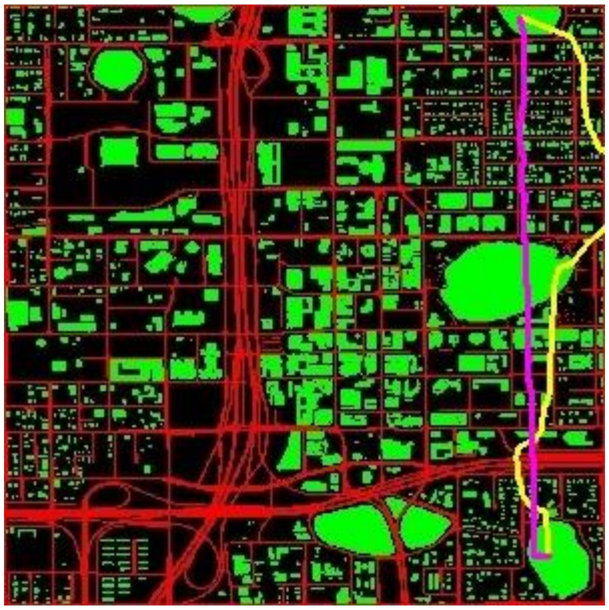
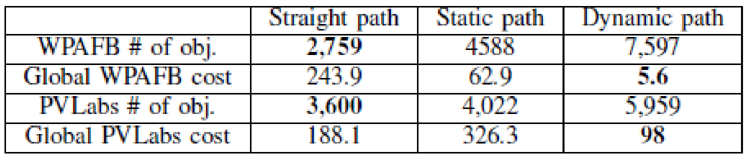


Figure 7

Results

In the Safe UAV [1] paper by Castelli et al. Results are given by testing with real footage of cars moving through an image for which the results are presented below. The purpose of our experiment is to try to replicate/improve results. To do this we've compared the previous algorithm against our new one.

TABLE I
SAFETY ESTIMATION RESULTS FOR WPAFB AND PVLABS



	Straight path	Static path	Dynamic path
WPAFB # of obj.	**2,759**	4588	7,597
Global WPAFB cost	243.9	62.9	**5.6**
PVLabs # of obj.	**3,600**	4,022	5,959
Global PVLabs cost	188.1	326.3	**98**

In our cumulative results below, we see that the DQN agent overall spends more time on safer trajectories than the previous algorithm, avoiding roads more so and being able to navigate without sacrificing too much of distance as we see in our experimental results as well. In our tests we did not test with real footage, but with a plethora of starting and ending locations for our agent to explore and learn the map.

Static Path	**Buildings**	**Neutral**	**Roads**	**Success Rate**
A*	0.5	.998	0.667	0.722333333
DQN	0.83	0.83	0.83	0.83
Straight Line	.998	0.667	0.83	0.832333333

Conclusion

We conclude that DQN offers an alternative solution to the path planning problem that allows for further exploration of traversing a map in a safe way. The advantages to this method is its simple approach allows for easy testing and no need for an internet connection for path planning, just the required input. Not to mention the ever-growing potential of AI and the branch of Deep Reinforcement Learning can improve performance as end-to-end solutions come up. In this independent study we have successfully replicated static path planning for UAVs. Given that dynamically UAVs must always avoid in coming cars in their FOV, traditional dodging path planning can be placed and then the static path re-embarked.

References

[1] Autonomous navigation for low-altitude UAVs in urban areas by Castelli et al. https://arxiv.org/abs/1602.08141

[2] AI Safety Gridworlds by DeepMind https://arxiv.org/pdf/1711.09883.pdf

[3] Dueling Network Architectures for Deep Reinforcement Learning

By Google, https://arxiv.org/pdf/1511.06581.pdf

[4] Reinforcement Learning with Tensorflow by Morvan Zhou,
https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow

[5] Playing Atari with Deep Reinforcement Learning by Mnih et al. https://arxiv.org/abs/1312.5602

[6] Q learning a simple proof by Melo,
http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf

[A] Reference Table

Agent = An AI program with a purpose.

Q-table = A lookup table with values that determine quality of position-action

Reward = Score of an agent due to its action and result

Actions = The set of available moves an agent can take