UAV Obstacle Avoidance Using Q-Learning

Katherine Glasheen, Marc Gonzalez, Shayon Gupta, Travis Hainsworth, & Ramya Kanlapuli Rajasekaran

Motivation

- Unmanned aircraft vehicles (UAVs) offer a low-cost, efficient platform for advancing surveillance, delivery, and science capabilities
- To exploit such capabilities, UAVs need to be capable of autonomous decision-making to keep them safe while navigating through national airspace
- These decision policies must avoid static and dynamic obstacles, such as severe weather, wildlife, buildings, and other aircraft, while progressing on their mission path.

Australian post delivers package with UAV https://www.itnews.com.au/news/australia-post-trials-parcel-delivery-by-drone-418239

Israel encounters problems with UAVs in national airspace http://sicnoticias.sapo.pt/mundo/2017-11-23-Israel-enfrenta-problema-de-infiltracoes-de-drones-nas-cadeias

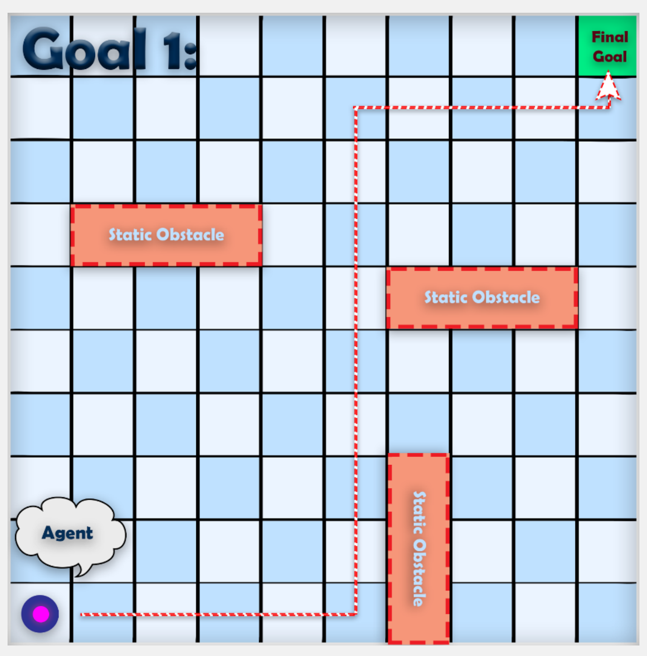
Overview

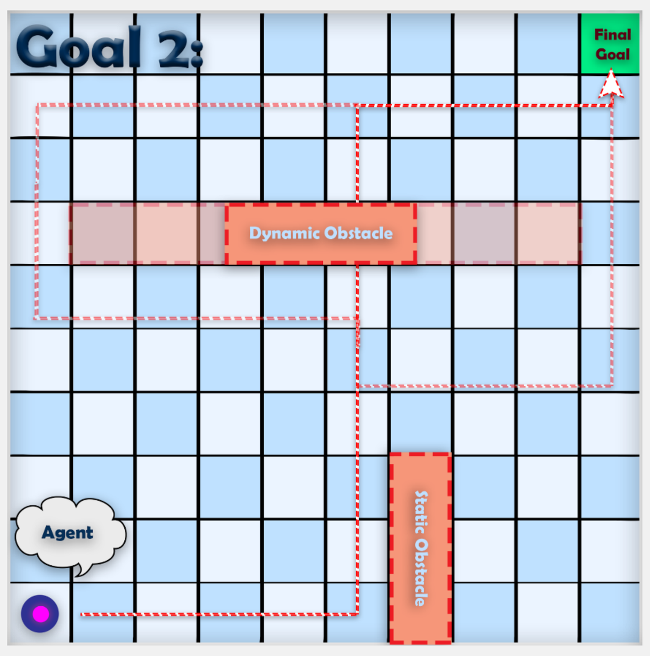
Q-learning algorithms prove effective for autonomous path-planning applications.

In Q-learning techniques, utilization functions generate values for actions taken over the course of the navigating agent's path, awarding positive scores to decisions avoiding danger and negative scores for collisions.

Researchers demonstrate the success of Q-learning for automated video-game strategies and autonomous vehicle navigation.

This research focuses on applying two Q-learning models to decide movements at discrete time intervals in order to avoid obstacles and complete mission paths



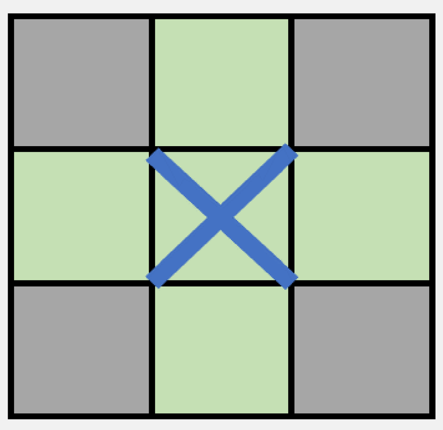


Problem Approach

1. Agent enters an unknown discretized grid world.
2. Agent observes the space within its action horizon (up, down, left, right) and assigns state-action values based on the observations

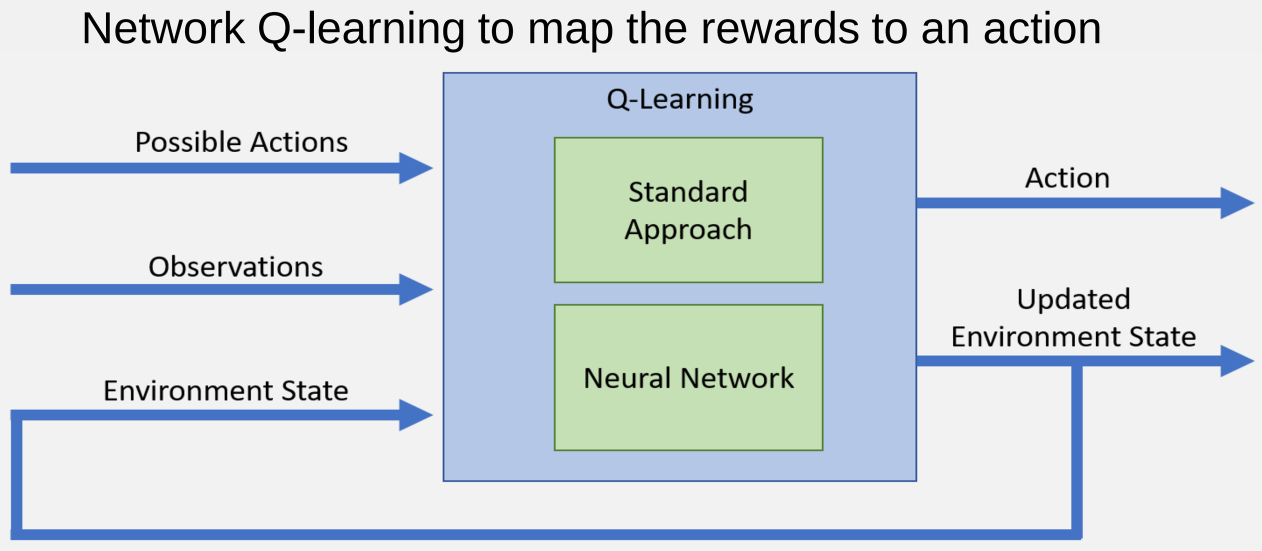
Grid Cell	Reward Value
Goal	100
Adjacent to goal	50
Obstacle/wall	-100
Adjacent to obstacle	-50
unobserved	0

Designers choice of rewards assigned to entities of the grid world



Agent observation horizon

3. Agent employs either Standard Q-learning or Neural Network Q-learning to map the rewards to an action



4. Vehicle dynamics are employed to move the agent given the commanded action

5. Steps 2-5 repeats until agent reaches the goal

Standard Q-Learning Using Lookup Table

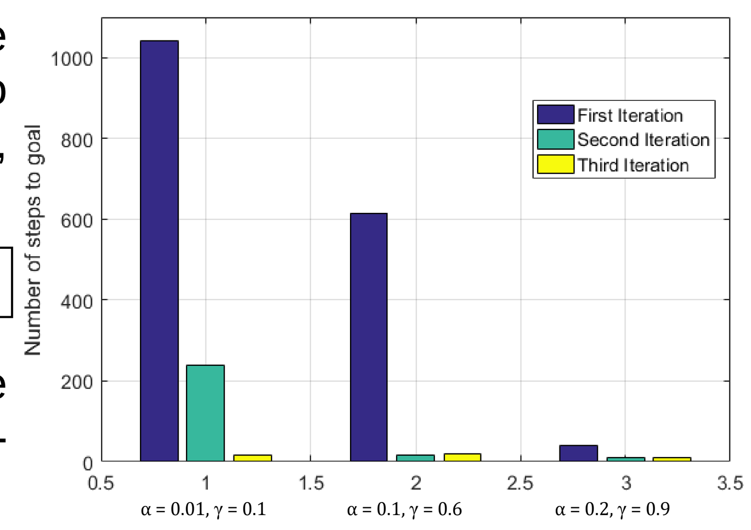
The agent represents knowledge of the environment in a Q-matrix. It updates the entries of the matrix, which map to environment states and possible actions, based on observations and prior knowledge:

$$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha(R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a))$$

At each grid space, the agent will choose the state-action pair that yield's the highest Q-value:

$$a_t = \max_a Q(s_t, a)) \quad a\epsilon\{up, down, left, right\}$$

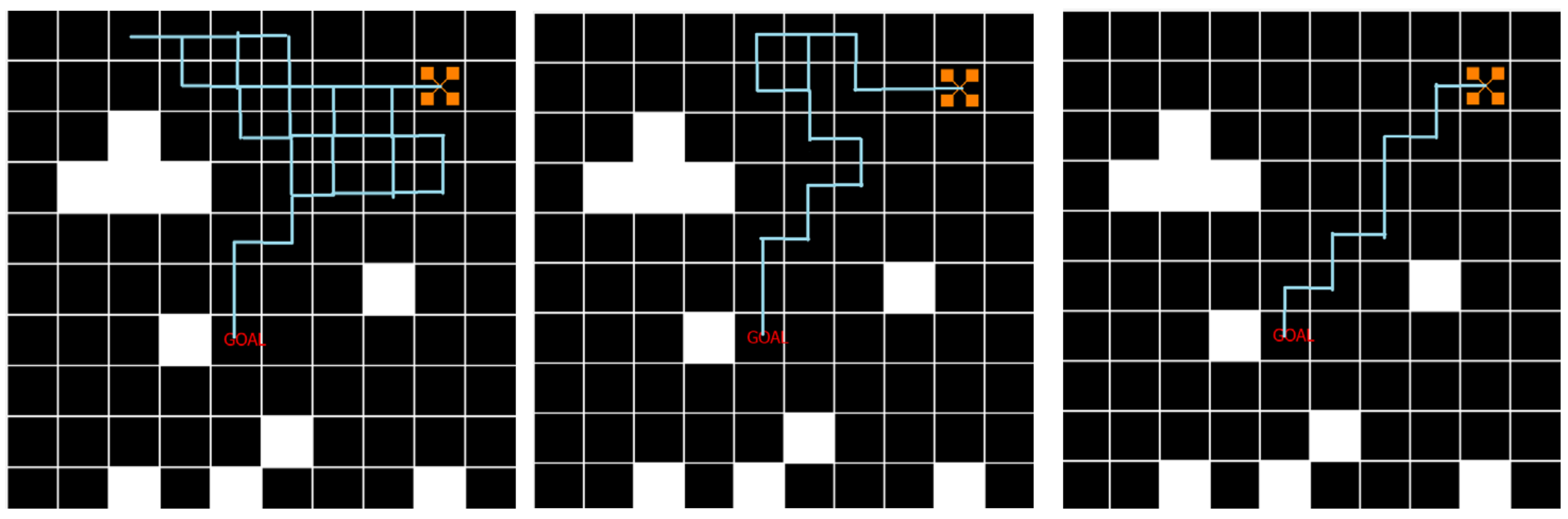
Parameter Evaluation



A lower learning rate, α, and discount factor, γ allows the agent to explore the environment, while a higher α and γ cause comparatively quick convergence with little exploration

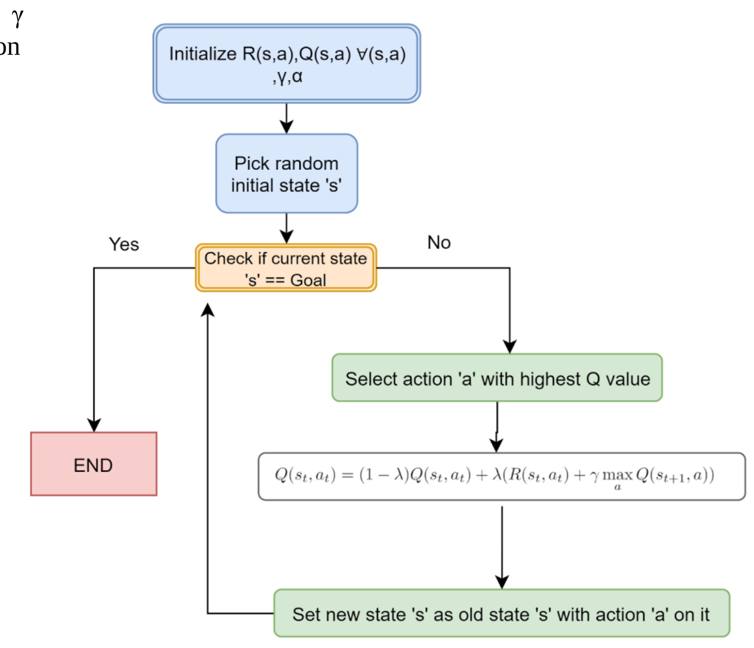
Summary

- Converges quickly on optimal paths
- Objective is to converge on optimal *policies* (as opposed to paths)
- To learn policies, the Q-matrix will be trained through random initializations of agent position

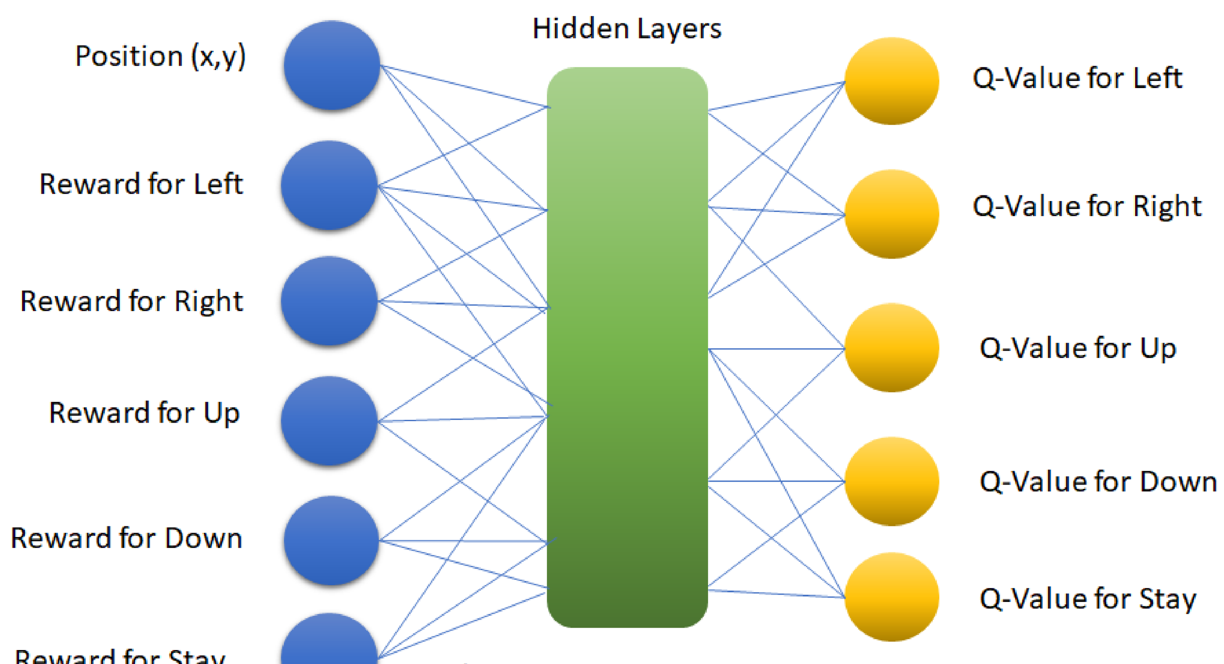


Iteration 1 *Iteration 2* *Iteration 3*

In an 10x10 grid world, the agent converges to the optimal path in less than 5 training games



Q-Learning Using Neural Network



* Rewards are anticipated based on observations of the next state

Preliminary Results

Training: Over 50 hours of training data generation, over 60 hours of model training

Model: Currently designed to end the game as quickly as possible, instead of being designed to reach the goal

Training

Utilized Imitation Learning by training from the actions of a Random Agent's successful game runs

Next Steps

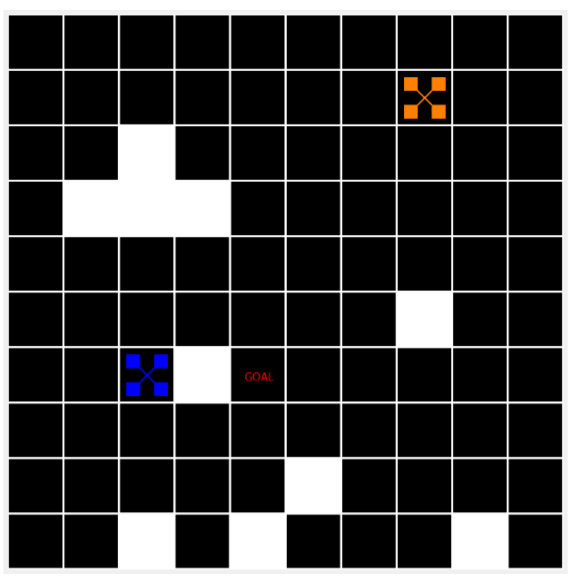
- Decrease the size of the hidden layers to improve computation time and accuracy
- Generate more training data with an improved method
- Utilize GPUs for training to decrease computation time

Evaluation Metrics

$$Success\ Rate = \frac{\#\ of\ goal\ reaches}{\#\ of\ games\ played}$$

$$Optimality\ of\ Success = \frac{\sum_{i=1}^{\#\ Games} \frac{Optimal\ Path\ Length}{Actual\ Path\ Length}}{\#\ of\ games\ played}$$

Dual Agent System

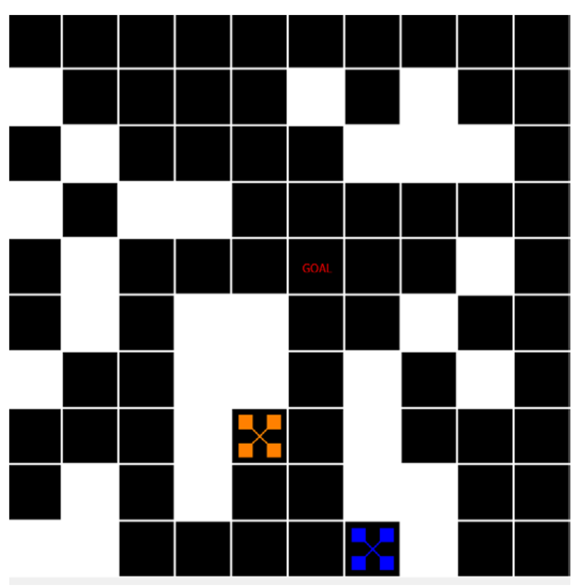


Two agents can exist in the same grid world environment. Both agents can employ a random action model, standard learning model, or neural network learning model

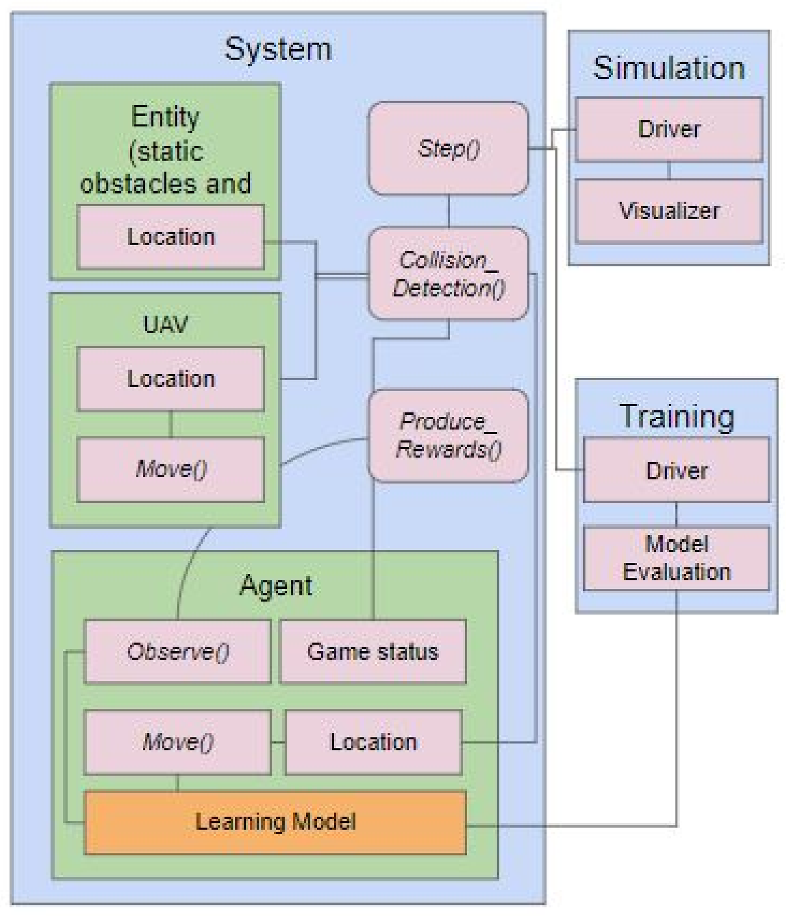
Progress Update

Both agents have been implemented using the standard Q-learning method.

The goal of Agent 1 is fixed, and the goal of Agent 2 is Agent 1.



Simulation Design



The system was designed in modules to allow for expansion and modification.

Parameters include environment size, number of obstacles, agent initial location, goal location

Vehicle Dynamics

action:	Up	Down	Left	Right
outcome:	Up - 75% Down - 5% Left - 10% Right - 10%	Up - 5% Down - 75% Left - 10% Right - 10%	Up - 10% Down- 10% Left - 75% Right - 5%	Up - 10% Down- 10% Left - 5% Right - 75%

Acceptance-rejection sampling is used to model vehicle dynamics. Generate a random number between 0 and 1. If it is less than the threshold value (table), take action. If not, stay.

Conclusions

Most Exciting Aspects

- Evaluating performance
- Using visualizer as a tool
- Successfully implementing reinforcement learning techniques
- Building infrastructure that we can continue to apply to future endeavors

Challenges

- Building initial infrastructure
- Generating training data and developing models – did not have the infrastructure to do this quickly
- Containing the agent within the grid world

Project Evaluation

- Our initial infrastructure focused on modularity, which had a huge impact on project expandability.
- Both methods show promise. With more time we could keep expanding and improving