PAWS Navigation: Reinforcement Learning for Path Planning in Unknown Environment

Samantha Johnson

University of Maryland, College Park

ENPM690 – Robot Learning

# Abstract

words

# Introduction

PAWS (Psychological Assistance and Wellness System) is in development to be an in-home mental health robotic pet companion. This companion is geared toward helping individuals with moderate to severe mental health conditions who would benefit from constant supervision and stimulation from an external source. The target consumer is a young to middle aged adult, who may live alone or are in a situation where there is limited supervision, and who also suffers from a metal health disorder such as depression, anxiety, bipolar disorder, or PTSD. The goal of this product is to provide companionship, motivation, emotional stabilization, activity notifications, and emergency resources to aid in the recovery process for an individual suffering from a mental illness. The PAWS Bot hopes to speed up the recovery time and decrease the relapse rate for these individuals, as well as reduce the high number of suicides that occur amongst this group.

PAWS is a wheeled mobile pet-like robot that is capable of interacting with its human companion through various sensors. One of the main development points for the PAWS Bot is enabling its ability to safely navigate in the human’s home. The robot needs the capability to respond to a human’s distress signal and traverse to their location quickly to provide assistance. This navigation component is a complex problem to solve because the manufactured robot will have no information about the home in which it will exist. This means the robot needs to be trained ahead of time to be able to consistently react in an unknown environment. To address the complexity of needing to operate in an unknown environment, it was determined that a machine learning technique would be utilized to train the robot’s navigation component.

The machine learning method used for the assistance and navigation component of PAWS will be reinforcement learning. This will allow the robot to acquire feedback from its environment and human in order to make decisions on where it should go. The algorithm investigated for usage in the navigation component was Double-DQN, which combines the techniques of Q-learning with Neural Nets. The robot and simulation was created using Coppelia Sim (formerly VREM) and programmed using Python, through access with the simulation’s Remote API. It will be expected that in the real world the human the robot is assisting will be using a wearable technology to be able to transfer data that the robot will use for localization. The robot will also be equipped with several other sensors to facilitate its navigation and assistance abilities.

# Background and Related Work

Robot path planning is a highly researched field which has resulted in many path planning algorithms that enable a robot to autonomously navigate to a location. Some well known algorithms like Dijkstra and A\* utilize a discrete grid of the environment and mark the vertices in this map with a cost based on distance or a heuristic function. Both of these algorithms are complete, in that they are able to find the shortest path between the start and goal, which the robot can then travel. In contrast there are sampling based algorithms such as RRT (Rapidly-exploring Random Trees) which creates an expanding tree from the robot’s start position, until it reaches a goal. This algorithm does not necessarily find the optimal path, but will find a path to goal if one does exist. (<http://correll.cs.colorado.edu/?p=965> Path Planning page 75+)

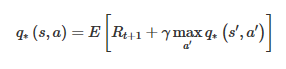
The previously discussed path planning algorithms all have a major element in common; they require a known map of the environment. There are methods that can allow a robot to create a map of its environment for use in path planning, such as utilizing SLAM (Simultaneous Localization and Mapping), but for the PAWS robot it is desired that it can consistently act in an unknown environment without a map. There has been research done into methods that allow a robot to traverse an unknown environment but this field is a complicated and still evolving one. It has previously been proposed to use some application of Evolutionary and Hybrid algorithms to solve this path planning problem, such as Artificial Intelligence, Genetic Algorithms, and Fuzzy Logic, but there is minimal research in the application of Reinforcement Learning to solve this problem (mobile robots motion path planning in unknown environments.pdf).

Some research in path planning utilizing reinforcement learning has been done with the application of a DDQN (Double Deep Q-Network) as the planning algorithm. Originally proposed by DeepMind In the papers examined, the algorithm used for the DDQN was the same, but the main things that differed were their environments and reward functions. In one paper, “Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning”, a DDQN is applied to a mobile robot in an attempt to plan in an unknown, dynamic environment. The work is able to show that…. IDK FIX IT LATER

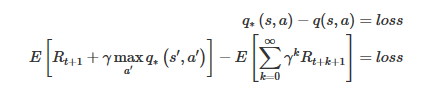
# Approach

The objective for the PAWS Bot at the end of training is to be able to navigate in an unknown environment and reach a changing goal in a timely manner. Based on the promising results of research utilizing reinforcement learning with a DDQN for path planning in unknown environments, it seemed beneficial to utilize this algorithm for the development of the PAWS navigation system. Along with this algorithm, the PAWS bot was created in simulation so that the training could be conducted.

## Double Deep Q-Network

The algorithm for a DDQN combines the search for the optimal policy, that the Q-learning provides, with the generalization ability and the application to large state spaces that neural networks provide. Together this algorithm allows for Q-learning to be expanded from the tabular method, to something more complex. The fundamental aspect of DDQN is the Bellman Equation. This equation explains the optimal Q-function, which defines the optimal return for a relationship between states and actions.

Bellman Equation: <https://deeplizard.com/learn/video/rP4oEpQbDm4>

The neural network in this algorithm functions to approximate this optimal Q function by minimizing the loss between the values output from the network, and the target values.

Network Loss: <https://deeplizard.com/learn/video/0bt0SjbS3xc>

Calculating the target values for the loss function can be tricky because the target function is what is being predicted. Using the network being trained to also calculate the target values causes the neural net to essentially eat its own tail during training. To resolve this, it is common to introduce a target network which is used solely to calculate the target values of the Bellman Equation. This target network is then updated on some interval to match the weights of the trained network. The introduction of this target network is what makes this algorithm a *Double* DQN (<https://deeplizard.com/learn/video/xVkPh9E9GfE> ).

The DDQN also utilizes the Epsilon Greedy Strategy which is what enables the implementation of exploration vs. exploitation to the algorithm. This strategy pushes the agent to explore the environment early on in the training phase, to potentially find a more rewarding action than the one that is being chosen by the DDQN. As the training phase goes on, the epsilon value decays, making it more likely to exploit the information it has learned through training ( <https://deeplizard.com/learn/video/mo96Nqlo1L8> ).

The last major component of the DDQN algorithm is called Experience Replay. This technique is accomplished by storing the state, action pairs, as well as some additional information from the environment, into Replay Memory, which is then accessed in random batches to train the network. This method is used to reduce the correlation caused by training the network on consecutive samples (<https://deeplizard.com/learn/video/Bcuj2fTH4_4> ).

All of the above components get applied to create the DDQN algorithm which was utilized for the PAWS navigation system. This algorithm is generalized below.

1. Initialize the simulation
2. Copy the policy network to the target network
3. For each episode:
   1. Initialize the environment
   2. For each time in max time:
      1. Select an action based on e-greedy strategy
         1. Random or from neural net
      2. Execute action in simulation environment
      3. Return reward obtained and next state
      4. Store last experience in replay memory
      5. Perform neural net training
         1. Select batch of memories to replay through policy network
         2. Utilize the target net to obtain values for the next state
         3. Calculate the loss between the output Q values and target Q values
         4. Gradient descent to minimize loss
      6. Decay epsilon
      7. After a set time update the target net weights to match the policy net weights

<https://deeplizard.com/learn/video/xVkPh9E9GfE>

## The Environment

The environment for the PAWS Bot consists of free space, obstacles, and a goal. To simulate a changing environment the obstacles and initial robot location were kept static, but the goal location changed around the map. This allowed for a new look at the environment every episode iteration. The environment is confined to a set space and closed off by walls.

The reward structure for the environment was complicated to devise. It was determined that the robot should learn to avoid obstacles, traverse to a goal, and reach the goal in a timely manner. So it seemed a simple 1 or 0 reward function would not be sufficient to encompass the scope of the problem. To accomplish the determined goals a function was created that incorporates aspects of distance and time to calculate the reward as shown below.

To summarize, if the robot reaches the goal state, it receives the goal reward, which is a large positive value to entice completion of the task. If the robot encounters an obstacle it receives the obstacle reward, which is a large negative value to encourage not crashing into obstacles. For all other spaces the reward is a composite of , which creates a positive value for moving towards the objective and a negative value if moving away. , which creates a higher value for being in spaces nearer to the objective. And , which is a negative value that increases as the time limit reaches expiration. The α are weights which sum to one and allow for tuning of the reward function.

## The Agent State

The robot agent needs the capability to observe its local area for obstacles and needs knowledge of the goal it is trying to reach. Due to these needs a state was created that combines 4 Boolean proximity sensor inputs that sense at north, south, east, and west based on the robots point of view, the normalized vector pointing between the human location and robot’s location, and the normalized time since the episode began. The state can be represented as the below, where the Z coordinate of the vector is excluded due to the problem being planar in motion.

# Implementation

# Results

# Analysis

# Conclusions

# Future Work

# Bibliography

Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning (PDF)