Artificial Intelligence Project – Implementation Report

Motion Planning for Robot

Submitted by :

Manan Jethwani (2019282)

Srishti Singh (2019346)

Introduction

Artificial Intelligence is a branch of Computer Science that seeks to understand the essence of Intelligence and produce a new kind of Intelligent Machine that can respond in a similar way to human intelligence, with research in the field including Robotics, Language Recognition, Image Recognition, Natural Language Processing and Expert Systems.

Motion Planning, as a branch of robotics and computer science, was first studied in mid 1960's primarily from a geometrical point of view and for robots with limited abilities. The basic and pure formulation of the problem only focuses on the geometric features of the motion and ignores many issues such as the robot's kinematic and dynamic constraints, uncertainty or incompleteness in the geometric data, control strategies for executing the motion, etc. A Path is generally regarded as a sequence of positions (or more accurately, configurations) which can guide a robot from its starting position towards its goal position. As such, a path has no temporal characteristic. The term Path Planning usually refers to finding a free path, that is, a path which has no intersection with any obstacle in the workspace. Alternatively, a Trajectory is an expression of a path as a function of time; i.e., it enforces the robot to be located on a certain point on a path at a certain time. The term Trajectory Planning, therefore, implies planning the motions of a robot on a path with regard to the element of time. Motion planning, on the other hand, is a term which in its general sense refers to path planning or trajectory planning, and in its specific sense is more akin to trajectory planning.

Collision detection is the most important factor of path planning. A planner should guarantee a collision-free movement of the robot; otherwise, the system will fail to function properly due to a crash of the hardware. This must be done automatically by the planner or by a human programmer. Local collision avoidance is important when the robot is moving in an unknown or poorly-known environment, and requires a feedback control to avoid paths that contain collisions. Since the paths generated by a planner should be navigated by robots and every robot has its own kinematic and dynamic limitations, the planner must devise a path that is collision-free and efficient regarding to some performance criteria such as velocity of links or joints, actuator forces, energy consumption, proximity to obstacles, and path traver.

The problem is challenging from a machine learning perspective due to the maintenance of a balance between the obstacle avoidance behaviour and goal-seeking behaviour. Emphasis on obstacle avoidance can lead the robot reasonably far away from the goal, making the overall task difficult, while an emphasis on goal seeking canrisk the robot being too close to the obstacles. Often the robot needs to select a proper homotopic class of getting to the goal and avoid being trapped amidst obstacles.

The objective is to achieve a specified task like avoiding obstacles, taking the shortest the more feasible approach to hardware after comparing different approaches to it.

Motivation

Most of the research done in this field is to move the bot in a physical environment such that it moves aimlessly avoiding all static as well as dynamic obstacles. We took inputs from the research already done and thought of optimizing it in the sense that the bot can reach the predefined destination efficiently.

We took motivation from the work done in this field and thought that how might can we enhance the motion of robot such that its path travelling can become more precise and accurate.

This project is significant and unique as it adds a specific goal or the destination such that the bot moves in th environment and stops after reaching the goal or its destination avoiding static as well as dynamic obstacles it encounters in its path.

Literature Review

Motion planning plays an important role in the field of robotics. This section discusses the advancements made in the research and development of various algorithms in motion planning in the past five years. Most of the recent motion planning algorithms are based on random sampling algorithms. More effective algorithms such as optimization-based, Probabilistic Movement Primitives (ProMPs)-based and physics-based methods are feasible research directions to explore to improve their effectiveness.

Many efforts have been conducted in robotic research for solving the fundamental problem of motion planning which consists of generating a collision-free path between start and goal position for a robot in a static and completely known environment, where there could be obstacles.

Motion planning in dynamic environments was originally addressed by adding the time dimension to the robot's configuration space ,assuming bounded velocity and known trajectories of the obstacles solved the planar problem for a polygonal robot among many moving polygonal obstacles by searching a visibility graph in the configuration space.

Machine Learning techniques have been widely used for the navigation of mobile robots moving towards a region of interest while avoiding obstacles. Surprisingly, there is a sparingly little literature on the use of machine learning techniques for navigating the robot towards a precisely defined goal configuration amidst static and dynamic obstacles. The need to have the robot reach a precise configuration is needed for applications like placing the robot for charging, robots carrying mobile manipulation, etc. Use of deep learning is must in order to achieve such goals.

KeyWords - motion planning, machine learning, neural networks, socially aware robot navigation, supervised learning, reinforcement learning

Setup

To recreate the collision scenario and use reinforcement learning a basic environment was needed to test and improve upon the neural network model, for this purpose we used pygame library and created following scenario which consisted of 4 things -

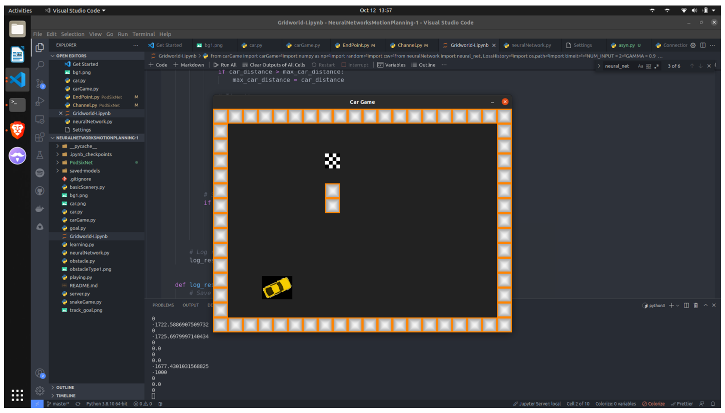
1) Car - The car is basically moved using either of 2 methods
   a) makeMove(self, action) - action can have one of 4 values namely (0,1,2,3) which makes the player car make one of the 4 actions that are i) rotate left by 5 degree j) move forward by 20 units k) rotate right by 5 degree l) move backward by 5 units
   b) update(self, actions) - update uses makeMove underneath to perform an action, but it also returns the current state of the game including distance from various objects and goals as well as it returns the reward for the previous action depending upon the state.

2) Obstacle - During the initialization phase the obstacle is randomly placed on the board and is a crucial part for building our model (the reward for colliding with an obstacle is -500).

3) Goal - The final position to reach is the goal and it has a reward of +5000.

4) borders - similar to obstacles the collision cost is -1000 and the borders are used to create a confined space.

The overall game looks as given below in visualisation mode -

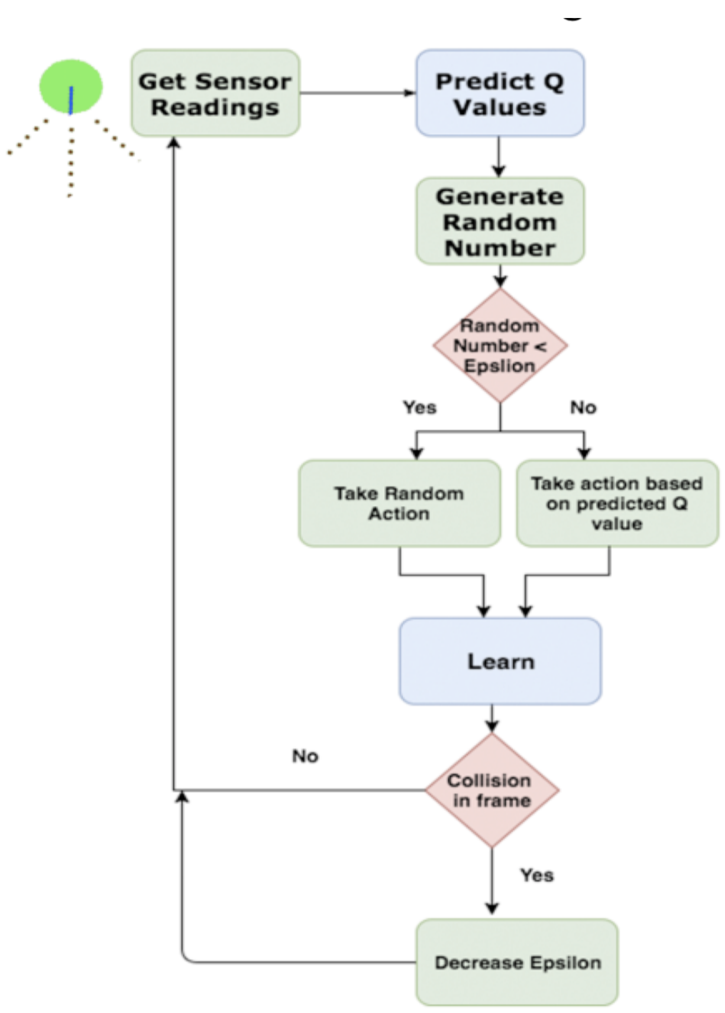

Gridworld-I.ipynb - NeuralNetworksMotionPlanning-1 - Visual Studio Code
Car Game
Run All
Clear Outputs of All Cells
Variables
Outline
PROBLEMS
OUTPUT
python3
Jupyter Server: local
Cell 2 of 10

```
from carGame import carGame⏎import numpy as np⏎import random⏎import csv⏎from neuralNetwork import neural_net, LossHistory⏎import os.path⏎import timeit⏎⏎NUM_INPUT = 2⏎GAMMA = 0.9 ...
```

```
            if car_distance > max_car_distance:
                max_car_distance = car_distance
```

```
0
-1722.5886907509732
0
-1725.6979997140434
0
0.0
0
0.0
-1677.4301031568825
-1000
0
0.0
0
```

Approach



The basic approach is to use the Q-learning approach which comes under reinforcement learning.

We used a neural network with 2 layers consisting of 2 input nodes and 4 output nodes, output nodes have 4 values for the 4 actions as mentioned earlier. Whereas 2 input nodes denote the current state of the car and it's distance from goal and obstacle.
Neural network used here is a sequential neural network model which is best suited when network layers do not need to interact with each other and there is only one tensor output. As for our project we used various densities of our 2 neural network layers and checked for the best suited model. Following are the list of parameters used for various models -

1) Nn_params (neural network densities) = [1000, 1000] , [512, 512] , [256,256]

2) Batch Size - [40, 100, 400]

   - The batch size here denotes the amount of actions to be considered as a single batch from the generated memory.

3) Buffer - [10000, 50000]

   - The Buffer size denotes the amount of replay frames to be stored in memory.

Procedure

Procedure can be summarized as follows -

1. For a set of params generate a neural network model using `neural_net()` method.
2. Before starting the training of model we go into observation period
3. The observation period involves letting the car take random actions and reach a random set, this is necessary for the model to be trained to take up any possible positions and generate best results and the history of states and actions taken is stored to be used ahead in training as well.
4. After the observation period comes to an end we move to random sampling and generate samples of specified batch sizes and generate mini batches out of these samples (X_train and Y_train).
5. After the observation period we use an epsilon which help us decide either to take random actions or use prediction of our model, this epsilon is decreased under 2 circumstances
   a. Either a collision occurs which means we need to depend more over predictions rather than random actions
   b. Or in case the prediction model is maturing and we need to perform more predicted actions.
6. In this period we train our model and perform actions based on previous state using predicted Q value and use it to generate new state and keep on improving using reinforcement learning.
7. We keep on generating rewards for each prediction which is used in training the model, the pseudo code for the same is provided below.

Pseudocode

```
1.  x,y = getSpriteCoodinates()
2.  Theta = getSpriteAngle()
3.  readings = getSensorReadings(x, y, Theta)
4.  if collision_with_obstacle:
5.          return -500
6.  else if collision_with_goal:
7.          game.pause()
8.          return 5000
9.  else:
10.         goalID, angleD = getGoalDistance, getAngleDiffWithGoal()
11.         return -5 + (readings/10) + 0.5*goalID - angleD
```

Conclusion

Using The reinforcement learning method, we can conclude that various neural network node density can affect the decision making capabilities of the model and uptil now as we keep decreasing the number of nodes we see a gradual increase in the performance of the model, Our next task for the upgradation would be to run various models under various densities to compare and find best suitable model.

References

[1] Abhinav Khare, Ritika Motwani, S. Akash, Jayesh Patil Rahul Kal, Learning the Goal Seeking Behaviour for Mobile Robots, 2018 3rd Asaia-Pacific Conference on Intelligent Robot Systems (ACIRS).

[2] Cheng Yan 2020, Research on Path Planning of Robot Based on Artificial Intelligence Algorithm, Journal of Physics: Conference Series.

[3] Bhushan Mahajan - Department of Computer Science and Engineering - G.H.Raisoni College of Engineering, Nagpur, Punam Marbate - Department of Computer Science and Engineering - G.H.Raisoni College of Engineering, Nagpur, IJCSN International Journal of Computer Science and Network.