Bryant Pong

CSCI-4480

Undergraduate

11/25/14

**Robotics 1 – Final Project Writeup**

**Approach – Path Planning Algorithm**

MATLAB unfortunately is not a suitable programming language for generic programming. After doing some research, I learned that since MATLAB is written in Java, Java code can be natively run in MATLAB. Therefore, the Path Planning Algorithm was written in Java.

In Computer Science, graph algorithms are some of the more used algorithms. Path planning becomes a much easier task if the current workspace can be mapped to an m x n grid. One of the most famous shortest-path algorithms is Dijkstra’s Algorithm, which is a greedy algorithm that tries to find the shortest path from two nodes in a graph that does not result in a cycle. However, a shortcoming of Dijkstra’s Algorithm is that obstacles are not taken into account. For obstacle avoidance, the A\* algorithm suits the task. A\* is a modified form of Dijkstra’s Algorithm that uses a heuristic to help avoid obstacles. For our project, we used the Manhattan Distance as our heuristic; the Manhattan Distance is the sum of the X and Y distances from the current location to the target location.

Pseudo-code for A\* is shown below:

**Listing 1:** Pseudo-code for the A\* Algorithm

**Explored**: List of nodes/spaces that have been explored

**InitialState**: Initial State of the Robot. Contains initial coordinate of the robot and an empty list of the previous steps to this state.

**PQ:** Priority Queue Data Structure holding the states to explore for the robot. Sorts by Manhattan Distance of each state.

PQ.add(InitialState)

While PQ is not empty:

NextState: The state from the top of PQ to explore

Explored.add(NextState)

If NextState == EndState

Return NextState.stepsToState

Endif

NextStatesToExplore: List of all possible states that can be reached from NextState

For I = 0 to length(NextStatesToExplore):

If(NextStatesToExplore[i] not in Explored):

NextStatesToExplore[i].stepsToState.append(NextState)

PQ.add(NextStatesToExplore[i])

Endif

Endfor

If PQ is empty:

Return NoSolution

EndIf

Endwhile

**Derivation – A\* Algorithm**

Our A\* Implementation is written in Java in the file “AStar.java”. To run native Java code in MATLAB, we first needed to install the Java Development Kit (JDK) corresponding to MATLAB 2012, the version of MATLAB used on the lab computers. The JDK version required is 1.6.0\_17.

MATLAB provides bare-bones functions to support native Java code. After compiling the AStar.java file with the JDK tools a class file is produced (AStar.class). In MATLAB, Java classes can then be used by using the “javaaddpath” command.

Unfortunately, our A\* implementation uses Java Vector data structures. MATLAB only provides a function (javaArray()) that creates Java Array data structures. We wrote up a Java class (ArrayToVector.java) to take in a Java Array object and return a Vector copy of the Array.

The A\* Algorithm is used in our project in the following way:

1) From Sean’s Image Processing code, construct a m x n Java Character Array denoting obstacles with an “X”, free spaces as “0”, the starting location as “S”, and the ending location as “E”.

2) Construct an ArrayToVector object to convert the m x n Array into an m x n Vector.

3) Run the A\* Algorithm on the Vector representation of the map. The return value is either a Linked List of (X, Y) coordinates for a path from the starting to ending point, or NULL for no solution.

**Conclusion**

Path planning for all robots, both mobile and stationary, continues to be an important but difficult challenge. When the element of computer vision is added, the complexity