e-Yantra Robotics Competition - 2018

NS Task 1 Report 1508

Author Name(s)	**Gudapati Nitish, Deepak S V, Jinesh R, Himadri Poddar**
Team ID	**1508**
Date	**28/11/2018**

Q1. Describe the path planning algorithm you have chosen.

Dijkstra's algorithm. Dijkstra's algorithm is an algorithm for finding the shortest path between nodes in a graph. The algorithm creates a tree of shortest paths from the starting node, the source to all other nodes in the graph/adjacency matrix. The edge between every node has weights. The weight of a node is the sum of weights of edges it has traversed from the start node. The algorithm searches for route to destination, connecting nodes, giving first preference to the nodes having least weights. When searching in this manner we can be sure that once the path search reaches the destination node, the route it followed will be the best route, since it connected nodes of least weightages. Dijkstra`s algorithm is more preferred in places where nodes are quite scattered, with edges having distinct weightages.

Q2. Describe the algorithm's specific implementation i.e. how have you implemented it in your task?

These are the sequence of steps employed to implement the algorithm in the task:

1. An adjacency matrix (int maze[24][24][2]) containing the relative positions of all 24 nodes given in Task 1.2 and the weightage of edges (line connecting nodes) was made. The weightage value given to the edges were proportional to the length of edges.
2. A function (void generate_path (char start_node, char end_node)) was made, which gets the start & end nodes as parameters and stores the connecting node numbers of the shortest path in a globally accessible path array (char path[24]).
3. The function makes use of the global maze array and some more local arrays known as priority_queue (int priority_queue[24][3]) , node_pile (int node_pile[24][3]) and priority_node (int priority_node[3]), for processing.

4. The algorithm starts by storing the details of start node in priority node. The details of individual nodes stored in the least significant indexes of priority_queue, node_pile and priority_node are as follows:

    0 – Node number

    1 – Node number of previous node of its branch

    2 – Weight of node (sum of weights of edges of its branch)

5. Then the nodes branching from priority_node will be stored in priority_queue, excluding the nodes which are already present in node_pile. In case a branch node (created from different path) is already present in the priority node, the "weight" value of both nodes will be compared and the node with lesser weight value alone will be stored in the priority queue.
6. Now the priority_queue will be sorted on the basis of weights of the nodes present, with the lowest weighing node getting the least index value.
7. Now the priority_node will be stored in node_pile and the priority_node is replaced with the 1st node (index 0) present in priority_queue.
8. Now the priority_queue is moved up by 1 step, deleting the previous value present in the index 0 of priority queue.
9. Step 5,6 and 7 are repeated till the priority_node becomes the end_node.
10. Now path matrix is generated by backtracking from the end_node stored in priority node, via the latest node stored in node_pile till the start_node present in node_pile. Backtracking is done by connecting the previous_node of each node till start_node is reached and simultaneously storing the connecting node numbers in the path array.
11. This generated path array is the output generated by the function and will be used by the program to make the bot traverse via all the connecting nodes specified in the path matrix till the end_node is reached.

YouTube Link: https://youtu.be/e7YdW_GPqsQ