TASK 3 : LABYRINTH MAZESOLVER

Debjoy Saha

***Abstract*— Navigation through a 3D maze using computer vision library opencv and python library pyautogui**

This is a project that has applications in dynamic path planning in 3D environment, given indicators of correct path as camera input. It is a small step towards creating an autonomous bot that can maneuver maze like environment and navigate to the destination.

I. INTRODUCTION

The task was to navigate through a maze, run with python 2.7 and tkinter dependancy (https://github.com/parakh08/Maze.git),solved using OpenCV in python3, numpy and pyautogui. In brief, approach followed was determining position of pixel having maximum green intensity in image at a certain point moving towards it. More or less the same approach was followed throughout the gameplay, except for times faced by an obstacle where an alternate obstacle clearance algorithm was used. Alternate methods of determining intermediate points in the image was thought of to reduce number of collisions, but was abandoned due to complexity.

Run by command : python3 task3.py python2 maze.py

II. PROBLEM STATEMENT

The Problem was navigating to the destination using W(forward), A(left), S(right), D(backward), ¡, ¿ (left/right translation) commands. The game interface is available in directory.

The colour of the walls turn more green as you move towards the finish(white). Using this information, the agent must determine its direction of motion and move using above controls.

III. RELATED WORK

None known.

IV. INITIAL ATTEMPTS

Initial attempts involved a path based approach, obtaining the path using image segmentation and deciding upon the overall direction of motion on the intensity of green pixel value. It would have allowed for thorougher search for all possible turns and ways. However, in favour of a simpler and faster solution, wall based approach was preferred.

V. FINAL APPROACH

The final proposed solution involved finding the greenest wall among all the walls. This involved first cropping the screenshot to reduce it only to game window. Then findwall() function was used to segment the wall from sky and path. Findgreenest() function was used to further segment the image to keep only the greenest wall i.e the wall towards whih we must move.

Then using contour function, the centre of that wall was determined. In case of multiple such walls, only one was retained. D and A keys were used to move such that that pixel remain approximately at the centre. Then the bot moves forward.

On facing an obstacle, the agent translates left or right randomly until it obtains clearance from walls (clearobstacle()), and once again the above algorithm is applied. On facing an obstacle head-on, it does a larger random turn, so as to expose other walls and move accordingly.

VI. RESULTS AND OBSERVATION

The proposed solution is a very primitive approach to gameplay and has bugs in obstacle algorithms. The game is able to manuever to the destination in easier maps i.e not involving dead ends, with few manual inputs only. However tougher maps can not be manuevered with present algorithm.

VII. FUTURE WORK

Necessary changes should be made to allow agent to manuever across dead ends, and make the gameplay fully autonomous. Also, pixel colour intensity based algorithm can be combined with a path based algorithm to produce a smoother path with less number of collisions (Determine direction of movement required and plan a path to that point using path based information). Bugs in obstacle clearance algorithms need to be fixed.

CONCLUSION

The discussed approach was very primitive and leaves ground for improvement.

REFERENCES