Treasure Hunt

CSE 360 Final Project Report

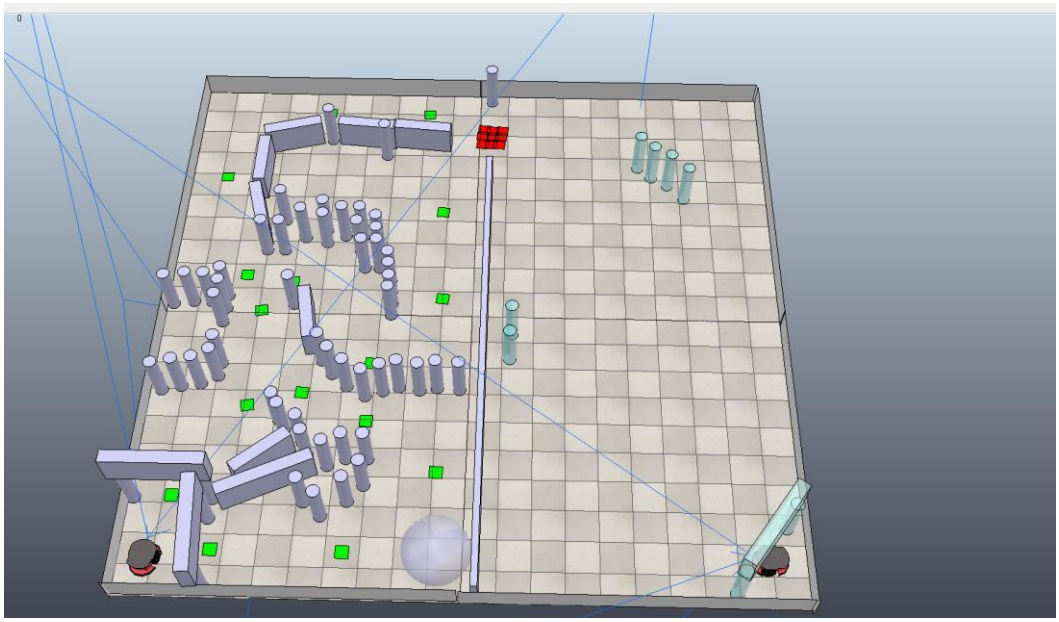
Jitong Ding

Motivation

Nowadays, there is a popular game called “Escape room” which askes players to find the clues that help them get out the rooms. This game is not only popular in the video game but also many mongers bring this game to the reality. Due to the popularity of this game for human, I am wondering if the robotics could play this kind of game which is not as difficult as what the human play. Thus, I am going to employ this kind of similar game for robotics, but it will not be as hard as the real one and I will change the game to be the “Treasure Hunt” which is similar to “Escape room”. Let us see what I can achieve.

Description

Based on my idea, there are two different environments, one a very complicated robot environment with many hints, which are green planes, and another an empty environment with several obstacles and no hints. The Treasure is at the top of the middle line of the whole environment, which are nine red planes.



As the plot shown above, on the left side, there are two different starting directions and two different routines to the destination so there will be 4 ways to achieve goal. On the right side, there is no green plane and routine so the robot can be placed at any place. There are two “Pioneer_p3dx” robots in the environment, which both have a version sensor and many ultrasonic sensors. The robot on the left will follow the hints to find the treasure but the another need to find the treasure by itself without any hint. In the end, the project will find which robot will win.

Method to Control two Robots

Declare two Client ID to control the two robots. One is **simRemoteApi.start(19999)** and another one is **simRemoteApi.start(19998).** Using two different scripts to control the robots is an easy way to manipulate and make any addition change to the specific robot.

Map and Locate

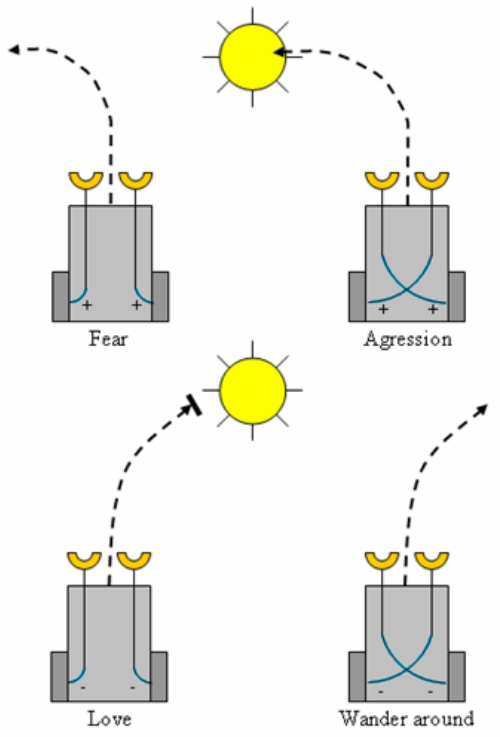
Map: Read the image from the camera and save into the list but the data from camera is inversed. In order to keep the accuracy, the image data keeps as original without flipping. Because the camera needs to detect the red planes and green planes, they are two functions to extract the red and green from the image and store the pixels where are red or green in the lists with “cv2.inRange()” function from OpenCV package. There is one more pre-process for images need to be done before using, the images pixel needs to be compressed into a list of size 16 because the original image list is 256 which is too large to calculate with braitenberg numbers.

Locate: Every robot has eight ultrasonic sensors. Read the distance information between the robots and the obstacles and other objects. Processing the sensor data, if the robot is surrounding by obstacles, transforming original distance data into scale from 0 to 1 will be easy for the later calculation. Moreover, if there is no obstacle nearby the distance will be assigned to be 0.

Method to Avoid the Collision with Obstacles

In the class, professor has already a varied way to avoid the collision like mapping and locating the whole environment before moving for every time, which will take a lot time, so I used “Braitenberg vehicle algorithm” which Phd Xu mentioned in his presentation.

Braitenberg vehicle algorithm: There are four basic model in this algorithm, “Fear”, “Aggression”, “Love”, and “Wander around” model.



For avoiding collision, the “Fear” model should be employed. As the picture shown, when the robot car detects there is an obstacle on the right side of robot, the right sensor will get the distance information and then control the right wheel to accelerate the velocity based on the Braitenberg numbers. The same procedure for the left wheel. As a result, both wheels accelerate the velocity and different direction with the obstacle so it will get away from the obstacle.

```
[0.25, 0.03, -0.22, -0.45, -0.64, -0.88, -1.6, -1.5]
[-1.2, -1.0, -0.8, -0.6, -0.4, -0.2, 0.0, 0.2]
```

First list controls the left wheel and second controls the right one. Because the distance sensor list’s size is 8, there are 8 numbers in every list. As the lists shown, some number is positive but some is negative, because each number deals with different sensor input which have various distance to the obstacles. The left wheel begins as positive because it is farthest from the obstacle and also its original moving direction is opposite to the obstacle. After the first number, the numbers will become smaller and smaller to be negative. The same concept for the second list. Moreover, the two lists should be symmetry but there are too many obstacles in the game environment, so I make the lists to be suitable for my project. The list will be updated to be a general one.

Method to Design Hints

The green planes in the environment are the “Hints” designed for robot. When camera detects the green planes, it will inform the robot to approach the green plane and then pass it. After then, the robot knows it needs to move forward to see next hint or the destination. For approaching green planes, the Braitenberg vehicle algorithm is also used here but this time it employs the “Aggression” model: when the left sensor detects the objects is on the left, it will control the right wheel to approach the target and it is same for right sensor with left wheel.

```
[0.8, 0.75, 0.7, 0.65, 0.6, 0.55, 0.5, 0.45, 0.4, 0.35, 0.3, 0.25, 0.2, 0.15, 0.1, 0.05]
[0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8]
```

The first list is the braitenberg number for left but it will be assigned to the right wheel and the second presenting the right sensor is assigned to left wheel. Because the first number in the first list is dealing with the farthest sensor on the robot, it will be the biggest and the number after will be smaller and smaller. It is same explanation for second list.

Method to Arrive Destination and Stop

The “Treasure” is designed as nine red planes together so when the robot sees the red planes, it will move to the red planes. The method of approaching the destination which is red planes is also “Aggression” model with same braitenberg number as green plane.

For stopping the robot at the destination when it win the match, the robot will check its relative position with the red plane position so if the relative position for both X and Y axis are smaller than 0.09, the robot will stop.

Conclusion

Based on ten times running the matches with different starting points for both robots, the right-side robot wins three times of the games and the left side robot wins seven times of the matches. As a result, if there is no hint for robot, the robot needs to spend more time to find the way to the destination.

Future Plan

1. Improve the Braitenberg vehicle algorithm to be more precise.
2. Add moving robots as obstacles in the environment
3. Add more robots as players
4. Make the robots could pick a random direction when they are in the crossing points.
5. Add different robots to start at the same place to make a match

Reference

Github link: https://github.com/djt1998/CSE360/tree/master/Final_Project

Final_Project_1.py for “Pioneer_p3dx” Final_Project_2.py for “Pioneer_p3dx#0”

Video link: https://drive.google.com/drive/folders/1TZr0wiwYn47pd-2q34QTA-sug0acsK-k?usp=sharing

Braitenberg vehicle: https://en.wikipedia.org/wiki/Braitenberg_vehicle