Mobile Robots - Navigation with Camera

Matias Cinera
Computer Science Department, *University of South Florida*
Tampa, Florida, 33612, USA
cinera@usf.edu

I. Introduction

This document is the report of my 4th lab from my Spring 2023 class CDA-6626 Autonomous Robots, as a graduate student. The individuals who are overviewing this project are our professor Dr. Alfredo Weitzenfeld (Professor in the department of Computer Science) and Mr. Chance Hamilton (Teaching Assistant & Graduate Student)

II. Motion to Goal

Motion to goal describes the process of finding a path for a robot from its initial position to a desired goal. For my implementation, instead of precomputing a path I just rotated the robot it was perpendicular to the goal and used a PID to stop 5in from the goal. To achieve this, I used the camera (provided by the TA, implemented with the camera module in Webots) and used its ***getRecognitionObjects(), getPositionOnImage(), & getPosition().***

Figure 1: Camera color object recognition

getRecognitionObjects() will return 1 if the object is on the field-of-view of the camera and 0 if it's not. ***getPositionOnImage()*** will return the horizontal position of the object (only if it's present the in the camera's FOV), relative to the camera resolution (80x80). ***getRecognitionObjects()*** returns the distance from the camera to the object in meters. The algorithm consists of rotating the robot until the goal in the FOV, then using the distance to the goal as an input for the PID.

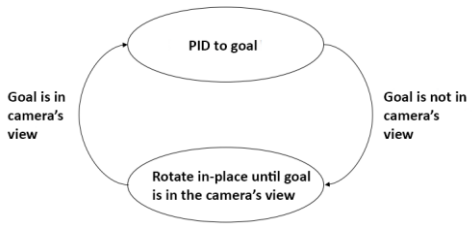

Figure 2: Motion to Goal state diagram

III. Bug Zero

Bug-zero can theoretically be implemented by combining a Wall-Following and Motion-to-Goal algorithm. Which is true, by simply following the logic of the state diagram in **Figure 3.**

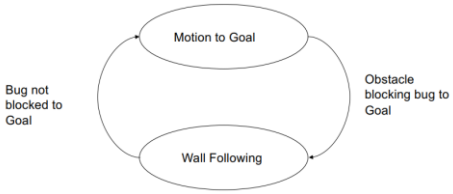

Figure 3: Bug-Zero state diagram

A. Bug Zero - Algorithm

However, there are a couple edge cases that need to be considered when implementing this algorithm.

- While MG, the goal is suddenly not in camera FOV
- The goal is not in camera view at the start of the algorithm.

Both can be solved the same way. Rotate in place until the goal is the camera's view. If after rotating 360° the goal was not found, go forward until the robot is close to a wall, then call the WallFollow algorithm.

```
WF = False
MG = False
if goal is in view:
    MG = True
while robot timestep:
    if WF:
        wallFollow()
        if position of goal is on the center of the camera:
            MG, WF = True, False
    else if MG:
        front_dist = front distance sensor
        goal_dist = distance to goal
        if robot is 5inches from goal:
            exit program
        else if front_dist if too close to a wall & front_dist < goal_dist:
            rotate in-place 90* to the left
            WF, MG = True, False
        else if goal is in view:
            motionToGoal()
        else:
            WF, MG = False, False
    else:
        rotate in place until goal is found
        if goal was found:
            MG, WF = True, False
        else: (goal was not found)
           go forward until the robot is close to a wall
            WF, MG = True, False
```

Figure 4: Bug-Zero Pseudo-code of my implementation

IV. Conclusions

The only problem when implementing this project was finding the edge cases of Bug-Zero. Assuming the simulation could be stopped at any instance, and the robot could be repositioned, the algorithm should adapt and still be able to find the goal. Cases such as moving the robot while Wall Following or Motion to the goal are handled by my algorithm.

References

[1] Webots Reference Manual. Webots. (n.d.). Retrieved January 23, 2023, from https://www.cyberbotics.com/doc/guide/imu-sensors?version=develop