The Iby and Aladar Fleischman Faculty of Engineering
Tel Aviv University

ZumoPi Obstacle Avoidance

Project Number: 21-1-1-2322
Names: Ido Arad , Meitar Amir
Advisor: Arkadi Rafalovich

Introduction

The project revolves around a robot based on a Raspberry Pi board. Using a single fish-eye camera attached to the front of the robot, our goal was to implement an obstacle avoidance algorithm that will supervise the movement of the robot and intervene when necessary. The obstacles to avoid are plastic balls of different known colors in a controlled environment.

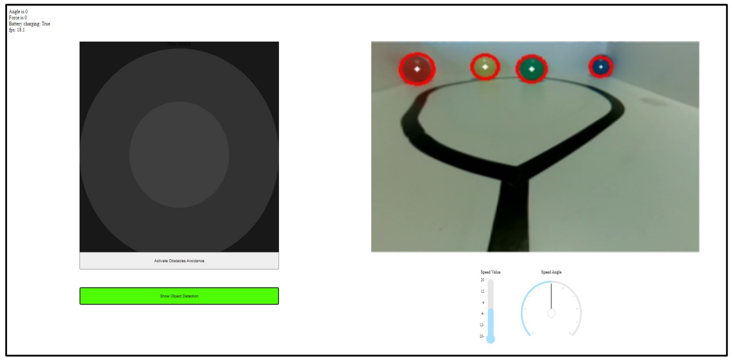
Motivation

The robot is part of the ZumoPi learning environment developed by our advisor. Its goal is to serve future students in labs and research. Our job was to continue developing the robot while providing feedback to our advisor in further improving the environment.

Implementation

In order to implement obstacle avoidance, we wrote an algorithm in Python using the OpenCV libraries according to the following steps:

1. Finding the camera parameters ,calibrating the fish-eye camera and detecting the obstacle.
2. Estimating the distance of the obstacle from the robot in accordance with the geometric structure of our system.
3. Calculating forbidden boundaries - we used the geometric features of our system in order to examine whether a collision will occur if a specific angle is chosen by the user.


Angle is 0
Force is 0
Battery charging: True
fps: 18.1

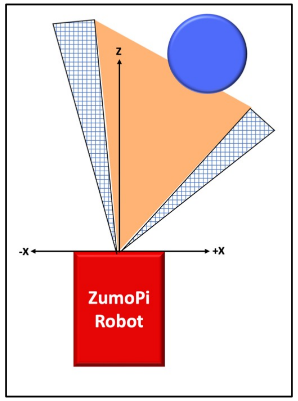
A weight function in our design will have factor in defining the error range of the boundaries and the angle of the controller's intervenience. We wanted the controller to have the most effect when the robot is closer to the obstacle, so we chose a function $w(z)$ that satisfies the following:

- $w(z \to 0) = 1$
- $w(z \to \infty) = 0$

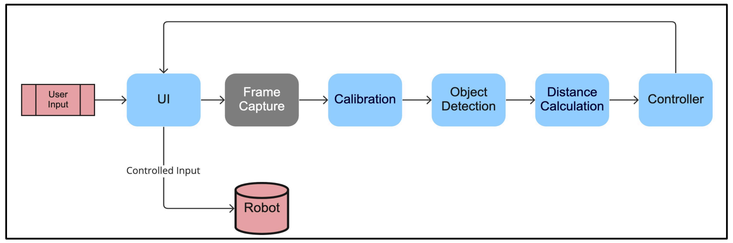
The function we chose:

$$w(z) = \frac{1}{1+\alpha^{z-\beta}}$$

4. Adjusting the robot's movement (if needed) according to its current state. If the user's commands fit in the forbidden boundaries, then intervenience is necessary, and the controller will adjust the movement angle according to the distance from the obstacle.

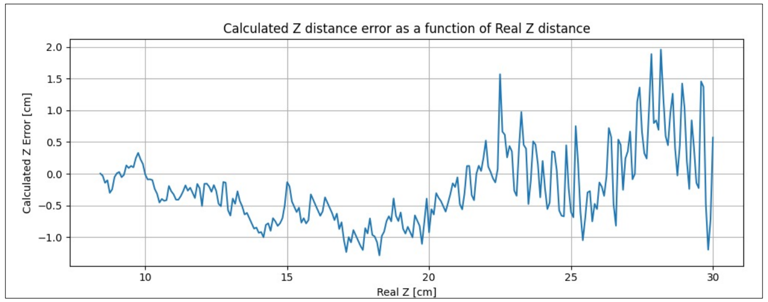


To conclude our implementation, the following block diagram represents the process that happens repeatedly for each frame:



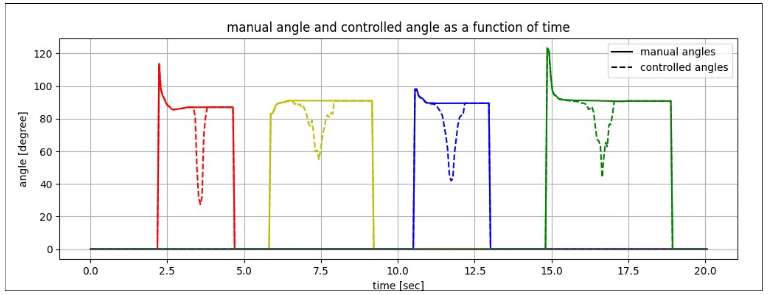
Results

As expected, our algorithm was able to adjust the user's commands in order to avoid obstacles.
The following graph describes the error of the distance estimation in the vertical axis with respect to the real distance of the obstacle from the robot:



It's noticeable that the calculated distance stays within a $\pm 2[cm]$ range from the real distance which is satisfactory to perform obstacle avoidance.

The following graph describes the manual input and the controller input as a function of time for every possible obstacle color:



The obstacles were placed such that the controller intervenes and changes the angle of the robot's direction when getting close to an obstacle.

Conclusions

Our algorithm works properly on most scenarios and allows obstacle avoidance.
When working on the project we noted a significant limitation regarding the current system. Obstacle avoidance will occur only if the obstacle is in the camera's frame. This means that there are cases where the robot might hit an obstacle that wasn't in sight.