Object oriented programming and software engineering

Waiter-bot

Mikołaj Suchoń 405821
Michał Jaros 407223

Attached to this report is the flowchart of all functions explaining which function depends on which function, library or data file.

Also attached is the whole code required to present a proof of concept on any pc with python3 and required libraries. Only thing required to change in code for operation on any PC is to change the path of the camera in RobotJourney. The code has been written with clarity and modularity in mind. The naming of functions, variables and code comments hopefully describe the functioning of the code unambiguously.

1. Principle of operation

The goal of the project is to build a system for controlling the food delivery robot operating in an indoor restaurant. The user chooses from GUI the starting point and the endpoint of the robot path. The robot navigates thanks to the lines on the floor of the restaurant and QR codes on the intersection of each line. Robot is equipped with a camera, and thanks to image processing functions described below and OpenCV library it is able to detect the line as well as to recognize the qr code and obtain necessary information from it. The software uses a predetermined map of the restaurant.

2. SimpleGuiMod:

It is least complicated mode out of these three. It opens the window in which we can insert the starting point and end point. GUIi has been created thanks to PySimpleGUI library.

3. HowToGetThere:

In this module we have the function which takes as arguments previously returned by the SimpleGuiMod. Then it opens the file with a map of all paths. Map is the list of all paths possible to take by the robot. One path is the list of intersections which is a tuple which has two values: intersection ID and angle of turn that needs to be performed on that intersection.

Based on the parameters from the user it chooses which path to use. As proof of concept we chose to have one base and 3 tables and one intersection. Robot can travel from the base to each table and from each table to the base. For now it cannot travel between the tables but it is only a matter of expanding the map and HowToGetThere library of possible routes. Logic will stay the same.

4. RobotJourney:
This is the most complicated part. Up to this point the purpose of the modules was just getting input data that will be used by the RobotJourney mode. Thanks to that mode our robot can move along the chosen path and interact with the environment around it.
Here we have the RobotJourney function which takes as argument path chosen previously.

At the beginning we define code frequency which tells us how often our code refreshes. Then we define the time after which the robot will initiate the findLine function if the line has been lost. It is one second in this case. Thanks to that our robot won't stop moving if it loses a path for less than a second which naturally happens due to glitches. Then we have camera settings. In the next part, code unpacked the path object into two lists. One for the qr code IDs (intersections and path ends) and one for the angles of turn. The actual process of controlling the robot starts in a while loop. Now we start from getting the image from the camera. After that we gather information needed for steering the robot. Cx and Cy are the coordinates of the middle of the object (line detected). In order to get the object we first convert the image from the camera to grayscale and then we use a series of thresholding and blurring operations to extract the line. Cx is the parameter that is important for steering the robot. For controlling two wheeled robots following the line only x-axes is relevant.

Additionally we use the CheckForCollison function. It uses data gathered from the ultrasonic sensor. If the external object is too close to the sensor, the function returns true. If collision equals true the robot stops for half a second and we overwrite coordinates Cx and Cy as 0. In that case the robot thinks that he did not detect the line. It causes the robot to stop for 15 seconds and then it starts looking for the line again. The same part of code is used if the robot actually loses the line.
What if our robot found the line and there was no collision. We set the speed of the robot. Then based on the cx parameter and speed parameter whileOnline function steers the robot according to the following rule: „If the centre of the object is to the right, turn right, if the centre of the object is to the left, turn left, if the centre of the object is in the middle continue the journey”. This is the line following part.

Then we have part responsible for navigating by the qr codes. At first we are looking for the qr code using the function QR_Read. This function allows us to find the orientation of the QR code in space (we draw 3 orientational points of the qr code and calculate the angle) and to read the information included in this qr code. If the ID from encountered code matches the ID of expected IDs, this ID is saved in the list of previously encountered intersections. It is important because it eliminates the problem with reading the same ID twice due to buffering errors. When the robot sees the proper ID for the first time it saves it in a list, thanks to that if it sees the same ID two Times in the row it ignores the second ID and does not stop in the same place doing the same instruction all over again. If the code ID matches expected ID and it is not the same code as previously and the robot saw that the amount of intersections it has encountered is the number of intersections it was supposed to encounter it means that it is at the end of the path and needs to stop.