Obstacle/Crash Avoidance Feature

DCRC Project

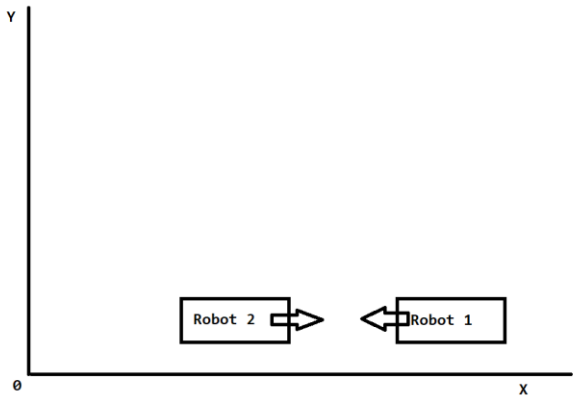
Safety Course Level 3

Background and Current Situation are explained in previous document!

Execution:

The obstacle avoidance feature was created in order to recognize if there is going to be a crash between two robots and prevent it. Because this goal is too broad, I have set the scope together with our project mentor Pablo as follows:

Two robots are facing each other and going in each other's direction along the X axis.



Colab localization system

The goal of the test case is to detect an obstacle via the lidar and compare the lidar data to the colab list of AGVs, which contains the distance to each AGV and the angle to each of them. So each robot has that list and knows the distance to all the other and the angle to all the other. By comparing that to the lidar we know if what is in front of us is actually a robot and which exact robot it is. When the robots "see" that there is another robot in front of them they stop and compare their tasks' priorities. The robot with the higher priority waits and the one with the lower gets a new goal to move along the Y axis so it can get out of the way. After the robot with the lower priority task gets out of the way the other robot continues on its path and the robot with the lower priority is given back its original goal and it continues on its way. In this way we prevent a crash and have a system for managing robot coordination for that particular test case.

The way the feature works is that the robots are subscribed to their Lidar and get the data of their surroundings. Because I defined the test case as the two robots facing each other, in the Lidar callback method I check if the distance to the nearest object at the 0^{th} angle is less than 5 centimeters, meaning that we check for an obstacle in front of the robot that is nearer than half a meter. Once we find such an obstacle, the robot has to define if the obstacle in front of it is another robot and that is done by getting the list of AGVs from the colab localization software via TCP and filtering those, which are in radius of 0.5 meters. After that the robot checks if the robots that are in this radius are within a 5 degrees range of the front of the robot to filter the list further. If there is a robot in that satisfies those two conditions then the conclusion is that the obstacle in front is a robot. In order to prevent a crash the robots uses the emergency stop to pause its task and stop in place. After that the robot gets the priority of the task of the robot in front of it and its own task priority, which is done with the help of topics. The information about the priorities comes from the blackboard and if the robot has a greater priority than the one in front of it, it waits for a while. If the robot has a lower priority than the one in front of it then it is given a new temporary goal to move along the Y axis in order to make space for the other robot to pass. After it has reached the temporary goal then the robot is given again its original goal and it continues on its path and in that way both robots have avoided a crash. The sequence diagram and screenshots of the code can be seen below:

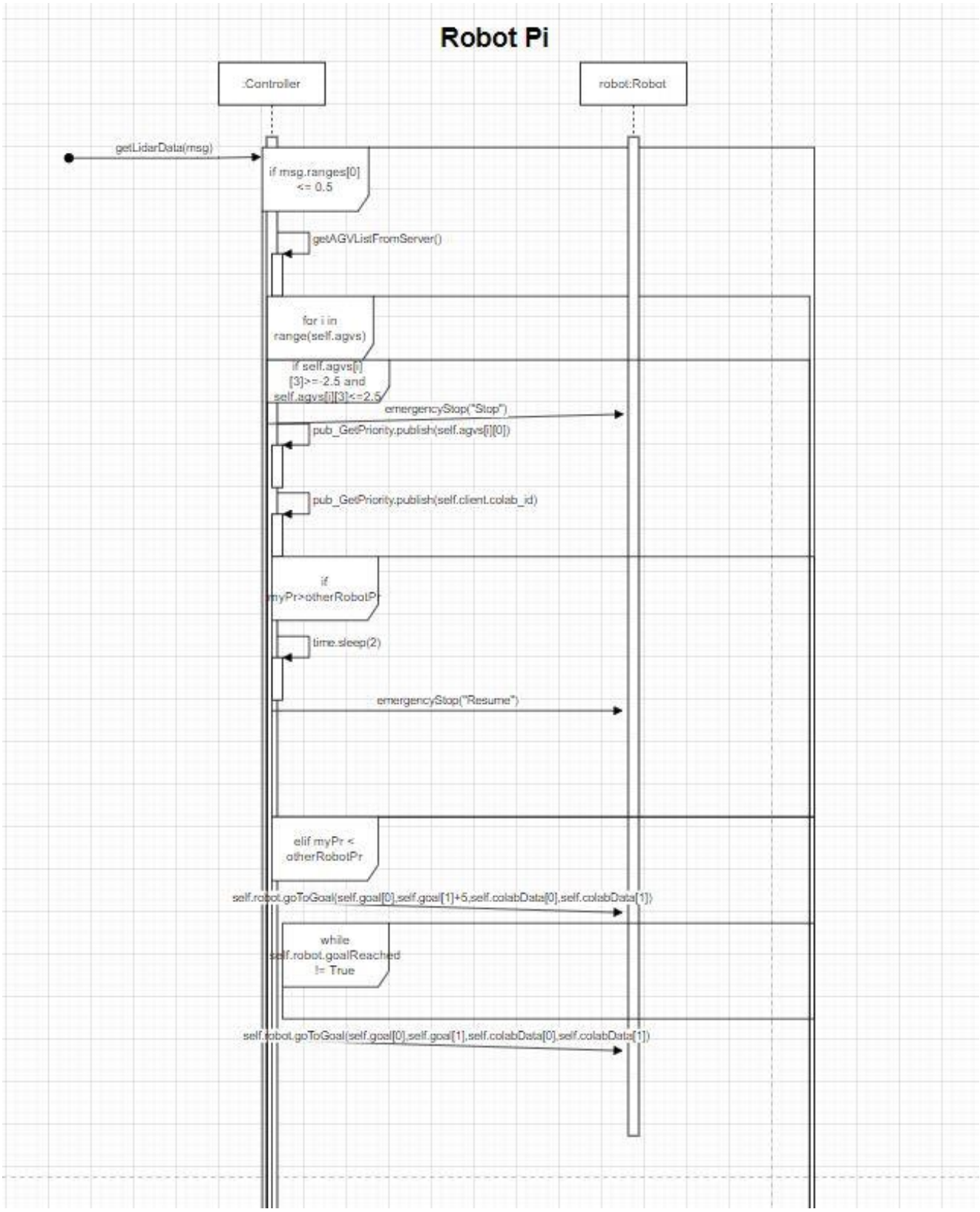
```
def getAGVListFromServer(self):
    #get robots in radius from colab via tcp
    self.agvs = self.client.getAgvs()
    #filter robots in 50cm range
    for i in range(len(self.agvs)):
        if self.agvs[i][1] >= 0.5:
            self.agvs.pop(i)

def getLidarData(self,data):
    print(len(data.ranges))
    #filter AGVs in 50cm range from lidar data
    if data.ranges[0]<=0.5:
        #get agvs in radius around this robot from colab
        self.getAGVListFromServer()
        #self.agvs contains robots in 50cm range according to colab data
        for i in range(len(self.agvs)):
            #filter AGVs from colab in 5 degrees range
            #if there is a robot that fulfulls the check then that is what the lidar detected
            if self.agvs[i][3]>=-2.5 and self.agvs[i][3]<=2.5:
                #stop current task to avoid crash
                self.robot.EmergencyStop("Stop")
                #get priority of tasks of both robots and compare them
                self.pub_GetPriority.publish(self.agvs[i][0])
                otherRobotPr = self.priority

                self.pub_GetPriority.publish(self.client.colab_id)
                myPr = self.priority

                if myPr > otherRobotPr:
                    #if this robot's priority is greater then we wait for the other to get out of the way
                    time.sleep(2)
                    self.robot.EmergencyStop("Resume")
                elif myPr < otherRobotPr:
                    #if this robot's priority is not greater then we give it an intermediate goal
                    #in order to get it out of the way, the goal x stays the same, the goal y is increased
                    #so the robot moves out of the way
                    self.robot.goToGoal(self.goal[0],self.goal[1]+5,self.colabData[0],self.colabData[1])
                    #we wait until the goal is achieved
                    while self.robot.goalReached != True:
                        print("getting out of the way")
                    #after it has gotten out of the way we give it its initial goal
                    self.robot.goToGoal(self.goal[0],self.goal[1],self.colabData[0],self.colabData[1])

def CheckGoal(self,msg):
        if msg.data[0] == self.robot.robot_id:
            self.colabData = self.client.getPreprocessedData()
            self.goal = [msg.data[1],msg.data[2]]
            self.robot.goToGoal(self.goal,self.colabData[0],self.colabData[1])
```


Robot Pi
:Controller
robot:Robot
getLidarData(msg)
if msg.ranges[0] <= 0.5
getAGVListFromServer()
for i in range(self.agvs)
if self.agvs[i][3]>=-2.5 and self.agvs[i][3]<=2.5
emergencyStop("Stop")
pub_GetPriority.publish(self.agvs[i][0])
pub_GetPriority.publish(self.client.colab_id)
if myPr>otherRobotPr
time.sleep(2)
emergencyStop("Resume")
elif myPr < otherRobotPr
self.robot.goToGoal(self.goal[0],self.goal[1]+5,self.colabData[0],self.colabData[1])
while self.robot.goalReached != True
self.robot.goToGoal(self.goal[0],self.goal[1],self.colabData[0],self.colabData[1])

Result:

As I explained in the previous document, the current situation is such that the robot navigation to goal is not working on Ubuntu with ros and only works on raspbian. Furthermore, the ros bridge and ssh are not working so it is not possible to assign a task to robot. This feature too cannot be tested as in order to test even a basic version of it I will need ros2 for the lidar topic and robot navigation. This is impossible because the navigation doesn't work on ros2 and I can't get the lidar data without ros2 so it is a deadlock. What I decided to do in order to at least show some basic prototype is to simulate it in gazebo. I successfully got the lidar data but I couldn't publish messages on /cmd_vel to the turtlebot3 for some reason and therefore I couldn't make it move aside to make space for the other robot. I am confident that with a bit of debugging, the feature can be easily implemented once the issues with the GPIO library and ros are solved.

The full code for the feature can be found on: https://github.com/fontysrobotics/DCRC / The feature is only on the RobotPi branch / The TCP server and client I created can also be found in the GIT repository