Robohockey documentation

Team 2
Reza Aghideh, Jaakob Lidauer, Bálint Máté

March 27, 2019

Contents

1 First milestone

1.1 The problem

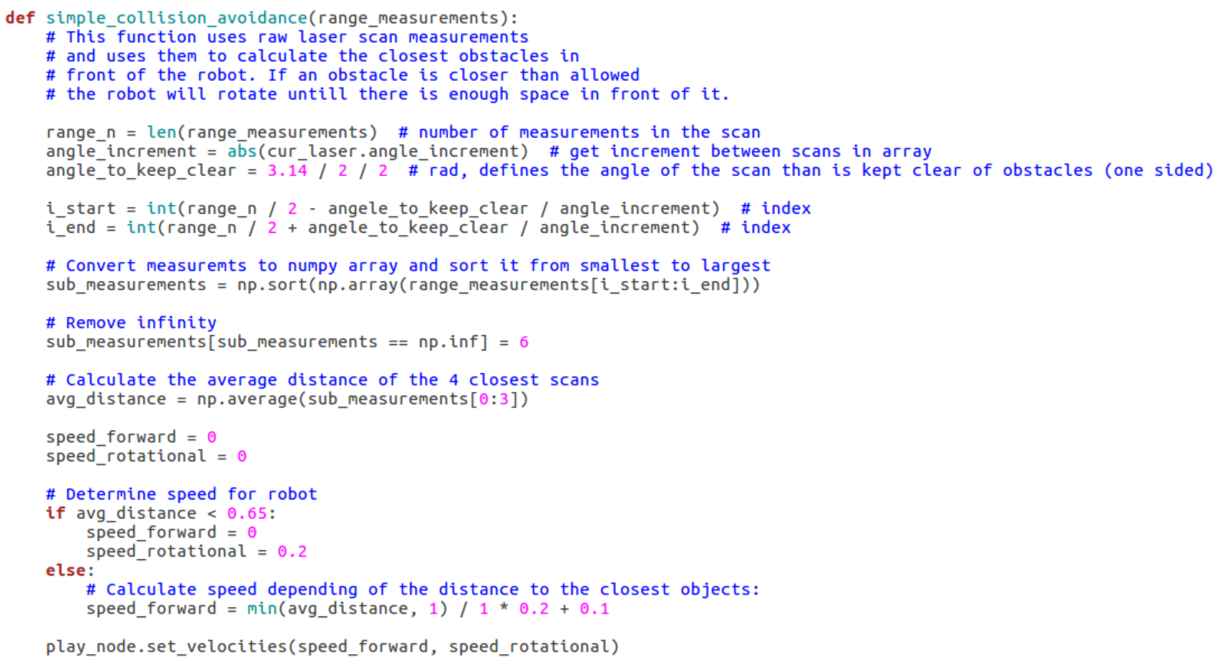
The goal of milestone 1 was to program the robot to perform very basic collision avoidance. The robot should drive randomly around and avoid obstacles. We solved this problem by using the data provided by the lidar sensor.

1.2 Our approach

Our solution to this problem was to program an algorithm that moves the robot forward if no obstacles are closer than allowed in front of the robot. If an obstacle gets too close the robot will rotate roughly around its center axis until there are no obstacles closer than allowed.

This algorithm is very simple and the robot can get in some cases stuck, especially when there are low obstacles (wires, chargers) that the lidar is not able to detect. However, in most cases the collision avoidance works surprisingly well. This is why choose the approach, it performs good compared to its simplicity. During the testing of the robot we encountered small issues with the robot node on the robot, it did not always getting a connection to the Rosaria module, but after a few resets it started working.

1.3 The code



```python
def simple_collision_avoidance(range_measurements):
    # This function uses raw laser scan measurements
    # and uses them to calculate the closest obstacles in
    # front of the robot. If an obstacle is closer than allowed
    # the robot will rotate untill there is enough space in front of it.

    range_n = len(range_measurements)  # number of measurements in the scan
    angle_increment = abs(cur_laser.angle_increment)  # get increment between scans in array
    angle_to_keep_clear = 3.14 / 2 / 2  # rad, defines the angle of the scan than is kept clear of obstacles (one sided)

    i_start = int(range_n / 2 - angele_to_keep_clear / angle_increment)  # index
    i_end = int(range_n / 2 + angele_to_keep_clear / angle_increment)  # index

    # Convert measuremts to numpy array and sort it from smallest to largest
    sub_measurements = np.sort(np.array(range_measurements[i_start:i_end]))

    # Remove infinity
    sub_measurements[sub_measurements == np.inf] = 6

    # Calculate the average distance of the 4 closest scans
    avg_distance = np.average(sub_measurements[0:3])

    speed_forward = 0
    speed_rotational = 0

    # Determine speed for robot
    if avg_distance < 0.65:
        speed_forward = 0
        speed_rotational = 0.2
    else:
        # Calculate speed depending of the distance to the closest objects:
        speed_forward = min(avg_distance, 1) / 1 * 0.2 + 0.1

    play_node.set_velocities(speed_forward, speed_rotational)
```

Figure 1: Collision avoidance

The code used for implementing the simple control algorithm is shown in Figure 1. As seen in the code the, first the range measurements are cropped to a desired sector in front of the robot, in this case 90 degrees. After that the measurements are sorted and an average of 4 closest measurements is calculated, this is done to get a more reliable understanding of the closest objects. Finally, based on the *avg_ distance* the speed of the robot is determined.

1.4 Work division

The code was mainly written by Jaakob, because of ROS setup problems with others. However, the code was tested and optimized with the whole team.