Report 1: Algorithmic obstacle avoidance

Introduction

The problem requires the development of an appropriate algorithm which enables an e-puck robot to navigate from a starting location A to a target location B while avoiding any obstacles on its way. The task is executed in a specific environment which is illustrated in Figure 1.

The suggested approach consists of solving two sub-problems:

1. Navigating from start location A to target location B.
    - Navigation and orientation
    - Monitoring and updating current location
2. Avoiding obstacles.

Resolving these sub-problems provides a holistic solution since the initial problem can be interpreted as an iterative process of orienting and driving in direction of the target and avoiding obstacles.

Methods

In this section we will discuss the solutions to the sub-problems in detail. As already mentioned bringing them together will provide the complete solution of the task.

Orienting and navigating to target. Provided that we have a start location and a target location defined as (x,y) with an additional start angle $\alpha \in [0; 2\pi]$, we can calculate the distance between the two locations on straight line. Additionally, we must consider the initial orientation of the robot since we need to have the robot facing the target.

By using the Pythagorean theorem we can obtain a measure of the distance on straight line between two given locations $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$:

$$d = \sqrt{(x_2 - x_1)^2 + (y2 - y1)^2} \tag{1}$$

In order to obtain the angle between the robot and the target we can utilize the inverse trigonometric functions. Inverse trigonometric functions invoke an angle by given angle's trigonometric ratios (or sides of triangle). The *atan2* function matches our purpose the best since it invokes a four-quadrant angle in the interval $[\pi; -\pi]$ instead of the standard *arctan* interval of $[-\pi/2; \pi/2]$. We can obtain the allocentric angle to the target θ by calculating $atan2(y_2 - y_1; x_2 - x_1)$:

Finally, in order to get the measure of the angle of rotation needed to face the target we need to calculate $\theta - \alpha$ where α is the current orientation of the robot and θ is the angle to the target. Figure 2 illustrates the applied Pythagorean theorem and the direction of the target between two points in the environment.

Monitoring and updating current location. Every time the robot encounters an obstacle, it needs to change it's moving direction in order to avoid the object. As a result, the robot drives off the straight path leading to the target. At this point we need to recalculate the angle to the target and the distance to it from the current position of the robot and reorient the agent towards the target. Therefore, we need to keep track of the current position and orientation of the robot at any point of time.

We can accomplish this by integrating sensor information over time. The wheel sensors return the traversed distance by each wheel in pulses.

Based on the pulse information from the wheels we can distinguish three different types of movement - in place rotation, movement on straight line and arc movement. By obtaining the pulses traversed by the wheels for a time interval we can determine the type of movement executed in this period. Note that we need to convert the pulses measure into millimeters before engaging into location and angle updates.

Obtaining left and right wheel pulses which sum up to 0 means that the robot was executing a in place rotation in this interval of time. In this case we do not need to update our location since the rotation does not require any movement on the x or y axes. However, we should update the current angle. After converting the pulse measure to millimeters we can trivially find the change in the angle $\Delta\phi$ for the current time interval Δt using the following formula:

$$\Delta\phi = \frac{d}{r_{robot}} \tag{2}$$

While rotating in place we consider the robot itself being a circle and thus the traversed distance will correspond to the length of an arc and $\Delta\phi$ will be the angle corresponding to this arc. d is the traversed distances by the wheels in millimeters and r_{robot} is the distance between the wheels divided by 2 or the radius of the circle.

In order to update the current angle of the robot we need to add the angular displacement to the last known angular orientation.

Movement on straight line is characterized by having equal distances traversed by both wheels, in other words receiving equal pulse rates for the left and the right wheel for a time interval Δt. In this case we do not need to update the current angle but we need to update the position of the robot. Using the following formulas gives us the displacement of the robot:

$$\Delta x = d.cos(\phi); \Delta y = d.sin(\phi) \tag{3}$$

Figure 3 shows how these dependencies are actually obtained - they are based on the trigonometric dependencies in a 90deg triangle. d is the traversed distance by the wheels in millimeters.

Executing an arc movement generally means that both wheels have traversed different distances. This case is con-

sidered to be the most complicated one as it requires updating both current location and angle. In order to obtain current displacement we need to calculate the egocentric displacement first. Once having the egocentric displacement we can calculate the displacement in the global coordinate system.

$$\Delta\phi = \frac{d_2 - d_1}{d_{wheels}} \tag{4}$$

$$x_{ego} = r.sin(\Delta\phi); y_{ego} = r.(1 - cos(\Delta\phi)) \tag{5}$$

$$r = \frac{\frac{(d_1+d_2)}{2}.d_{wheels}}{d_2 - d_1} \tag{6}$$

$$x_{allo} = x_{ego}.cos(\phi) - y_{ego}.sin(\phi) \tag{7}$$

$$y_{allo} = x_{ego}.sin(\phi) + y_{ego}.cos(\phi) \tag{8}$$

Where, d_1 and d_2 are the traversed distances by the right and the left wheel and d_{wheels} is the distance between the wheels of the robot.

In order to find the current location in the global coordinate system we need to add the current displacement to the last known position - the origin of the egocentric system (valid for both straight and arc movement). As previously mentioned we also need to update the current angle by adding the angular displacement to the last-known angular orientation:

$$location = [x_{last} + x_{allo}; y_{last} + y_{allo}] \tag{9}$$

$$angle = \phi + \Delta\phi \tag{10}$$

Sensors and avoiding obstacles. Estimating the proximity to objects is done by the 8 infrared sensors mounted on the robot. An additional calibration step is needed so that the acuity of distance estimation to objects in the specified environment is maximized. The calibration is an important step since it allows us to adjust the sensor outputs according to our environment in the sense of material and distances. Furthermore, the original IR sensor readings are described by an exponential function which causes problems in mapping values and thus is not useful for reading out corresponding values. We can solve this limitation by approximating the original function to a linear one. This is done by a logarithmic regression which yields distance in cm:

$$d(v) = -a.log_e(v) - b \tag{11}$$

v is the normalized value from the sensor, a and b are chosen according to the measured sensor characteristic.

Figure 4 and Figure 5 show the obtained raw values from a single sensor after running calibration sequence respectively with an wooden obstacle and a mirror. As you can see the proximity values vary with the change of material - in this case the standard deviation seems to be larger for wooden objects Because of such variations, calibrating the sensors according to the environment is an important step.

After the calibration we can use the readings from the sensors,normalize them in the interval [0,1], turn them into cm and obtain the proximity to an object in direction of a certain sensor. The normalization step basically maps the smallest value from the original value interval to 0 and the biggest one to 1. This way every value can be expressed as a number in the interval [0,1]. Figure 6 shows the processed proximity values of all the sensors after calibration, normalization and conversion. The shown values suggest that there is an obstacle in about 4 cm ahead of the robot. The current solution uses only the frontal sensors(num.1 and num.8) for estimating distances to objects ahead. If an object is in less than 4cm ahead, the robot stops, performs a 60deg turn in spot and proceeds with a straight movement before reorienting to the target.

Results

The discussed algorithm uses only forward movement on straight line and rotation on spot. Therefore the resulting behavior of the robot are an iterative sequence of repetitive movements which is executed until the target is reached. The sequence can be sum up up as follows:

1. In place rotation. Orienting towards target.
2. Movement on straight line towards target until an obstacle is met or target is reached.
3. If obstacle is met - in place rotation, forward movement until obstacle is passed.

Discussion

Wheel sensors inaccuracies. From time to time the robot will display sort of inaccuracies in the sensor data from the wheels. For most of the time this is not going to impact the overall behavior of the robot. However, in some cases it may be beneficial to set up little margins when checking the type of movement so such inaccuracies are avoided as much as possible. For example we can add a margin of 2 pulses in the straight movement check, this way readings in which the pulses of the wheels have up to 2 pulses difference will be also considered as a straight movement.

Turning direction when avoiding an obstacle. Using the IR sensor output on both sides of the robot, we can estimate on which side the obstacle is closer to the robot. This way we can turn in the direction in which the obstacle is further away and thus have a smoother navigation and a straightforward path to the target location.

Figures

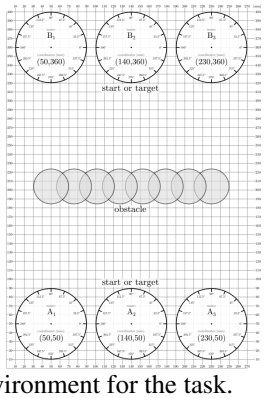

Figure 1. The environment for the task.

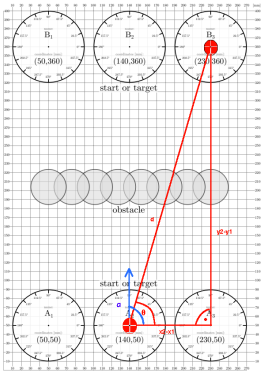

Figure 2. Graphically applies the Pythagorean theorem and illustrates the discussed angles.

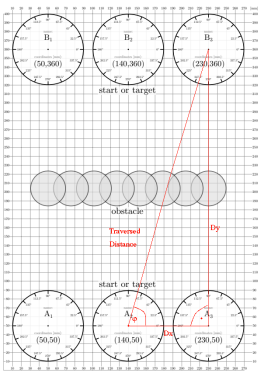

Figure 3. Graphically shows how we come up with the position displacement equations in the case of a straight movement. Start location A2 target location B3.

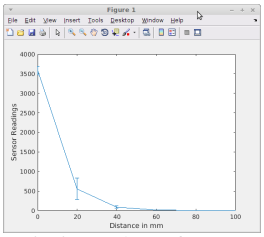

Figure 4. Raw proximity values from a single sensor after the calibration sequence with an wooden obstacle

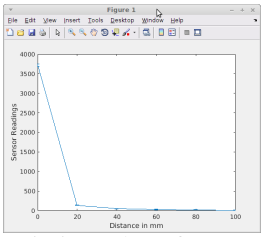

Figure 5. Raw proximity values from a single sensor after the calibration sequence with a mirror.

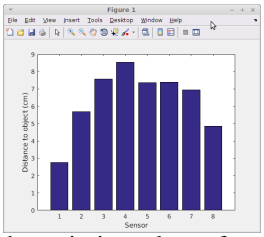

Figure 6. Obtained proximity values after normalization and conversion. Showing an obstacle in about 4cm ahead of the robot.