2020

Robotics – CE215

ASSIGNMENT 2

SHINSKIY, MAXIM 1804336

Table of Contents

Introduction

The goal of the assignment is to learn the use of classes of leJOS NXT Brick to write a software code for the robot. The robot will travel from one corner of an arena to another and avoid obstacles on its way. Other than using the motors, the robot will also make use of multiple sensors such as, ultrasonic and colour sensors, to detect objects on the route and find the waypoints on the path. In the meantime, the car will produce the output as a trajectory displayed on its LCD screen and write it to the .csv file to analyse the path completed afterwards.

Robot Behaviour

The code starts with initializing all required variables and constructors of the classes used for travelling and navigating such as, `DifferentialPilot pilot`, `OdometryPoseProvider opp`, `Navigator nav` and `ColorSensor colorSensor`, `UltrasonicSensor sonicSensor` for attaching and initializing the sensors.

The main part of the code that is responsible for the logic of the car, consists of few while loops. The biggest one checks whether the route has been finished, next while loop allows to check for several conditions and execute other functions at the time when the car is moving.

Travelling

To make the car move from one side of an arena to another I used `Navigator` class that also required the `DifferentialPilot` and `OdometryPoseProvider` class to travel. For the pilot class I recalibrated the values, due to loss of the robot in the period between the assignments. The robot starts off with heading 90 degrees, for that I set the new pose in odometry class, `opp.setPose(new Pose(0, 0, 90));`.

```
//Pilot
DifferentialPilot pilot = new DifferentialPilot(3.37, 17.25, LEFT, RIGHT);
OdometryPoseProvider opp = new OdometryPoseProvider(pilot);
pilot.addMoveListener(opp);

//Navigation
Navigator nav = new Navigator(pilot);
nav.setPoseProvider(opp);
```

By creating waypoints objects I could set a path for robot to follow. Nevertheless I didn't add them to the path and then asked the car to follow, otherwise instead I asked the car to follow one after the other, that way I could explore the region at the waypoints (at 150, 150 and 0,0) to find the red square. To switch in between the checkpoints, I have if statements. Initially the car will go to the set waypoint, which is set by `currPoint = points[0];` and `nav.goTo(currPoint);`. After the conditions in the while loop are met the `currPoint` variable changes to the other point, then the car stops to exit the while loop that is controlled by `nav.isMoving()` and then it follows the next point.

In the while loop that has `nav.isMoving()` I use the sensors to detect the colour tile on the floor and an object in front.

To find the obstacle it uses the ultrasonic sensor. In a while loop it has an if condition to check the distance from the sensor to any object ahead. If the distance is less than 15 centimetres the robot stops, turns 90 degrees to the right and goes in an arc around the item. It is given that it's 10 centimetres in diameter and 15 centimetres tall, therefore as arguments for arc function in the code I use 20 as radius and 180 for angle. After the manoeuvre is complete the car follows the current waypoint.

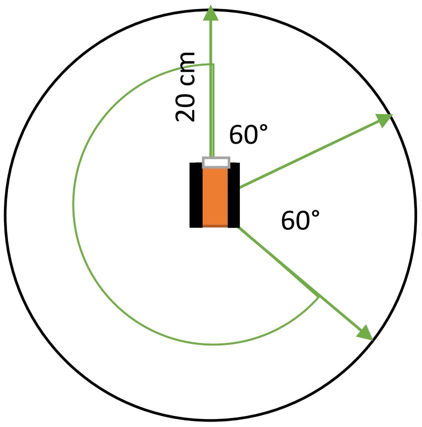
```
if (sonicSensor.getDistance() < 15) {
    nav.stop();
    pilot.rotate(-90);
    pilot.arc(20, 180);
    nav.goTo(currPoint);
}
```

In order to switch to the next waypoint robot first needs to find the red spot on the floor. Colour sensor is responsible for that job. This sensor class has a method `getColorID()` that returns integer value as an id of a colour, from the table in leJOS documentation, I found that red colour has id of 0. Therefore, if when travelling is done, the robot lands on the red spot, it will return 0 and conditions of an if statement are going to be met. Inside a statement the point is switched to next one and `nav.stop();` to stop the robot, so it leaves the while loop, so it could go to the next point. To terminate the movement the brick checks what point had been reached, if it's the final one, then the boolean variable is changed to exit the while loop.

```
//If the sensor detects red colour at the waypoint
if (colorSensor.getColorID() == 0 && nav.pathCompleted()) {
    if(currPoint == points[0]) currPoint = points[1];
    else finish = true;
    nav.stop();// move to next waypoint
```

Exploring

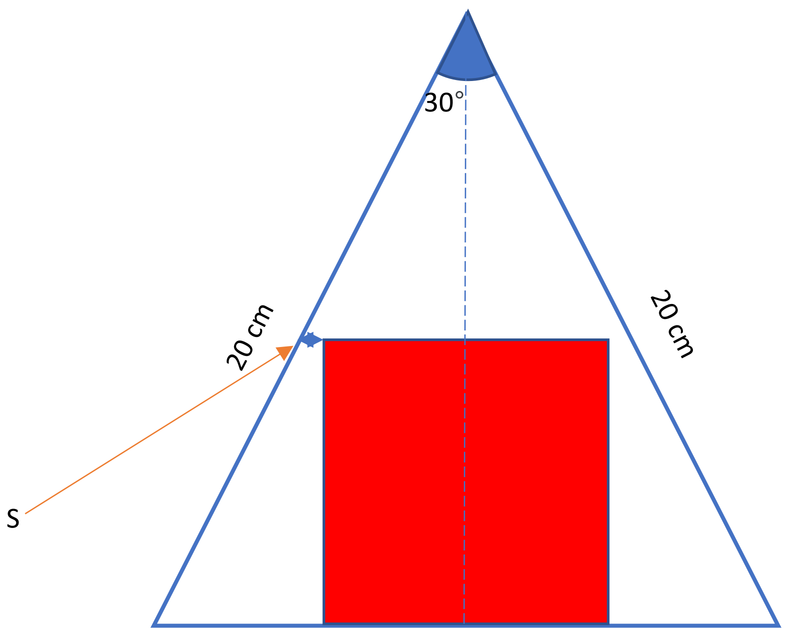
The movement of the car is not accurate enough and it is hard to make it stop at exact position of the red marker. In this case, the exploring algorithm is used. To find the spot robot will move forward and backward all around itself to scan the area in a radius of 20 centimetres around it. I chose this value after having few testing rounds, at which I observed that car doesn't go too far off the target.



Few instance variables need to be initialized first, `int s = 20` constant for magnitude of distance, `int a = 60` constant for magnitude of angle of rotation, `boolean found = false` for checking the condition for while loop, `int count = 0` to keep track of direction of movement. After, in a while loop, it is intended that car will go forwards then swap the sign to negative to go backwards, then turn by 30 degrees and do movement repeatedly until the red square found. The colour sensor checks for the data in a while to supply the program constantly with information.

```
//Beginning of exploring algorithm
boolean found = false;
int s = 20;   //distance, constant magnitude
int a = 60;     //angle of rotation, constant
int count = 0;  //constant, to keep track of number of cycles complete
while (!found) {
    //if the number of COUNT is even then increase the angle
    if (count % 2 == 0 && count != 0) pilot.rotate(a);
    pilot.travel(s);
    s *= -1; //reverse the direction of travel
    while (pilot.isMoving()) {
        if (colorSensor.getColorID() == 0) {
            found = true;
            if(currPoint == points[0]) currPoint = points[1];
            else finish = true;
            nav.stop();
        }
    }
    count++;
}
```

The constants for magnitude of distance and angle of rotation were chosen to increase the pace of searching with sufficient area coverage. Assuming that the robot is at most 20 centimetres away from the target and that the red spot is a 10 by 10 square, I chose the 30 degrees angle to be most suitable. With greater angle the square could be possibly missed, with smaller it would take more time than needed. To prove that let's take a triangle with the two equal sides of 20 centimetres which are our paths of searching for the red colour. If the square is still inside the area, the worst-case scenario would be if it touches the lower side of a triangle and equally apart from each side.



We can now find the distance that the square is going to be away from the track in the worst case, call it S. The line going straight down is going to be equal to 17.3 cm from $20 * \cos(30) = 10\sqrt{3} = 17.32$ then subtract 10 from that and we get 7.32. Last thing left is to find how far is the middle line form the side and subtract the half of the length of the side of square. So $7.32 * \tan(30) = 4.23$. That means that square is a little bigger than the triangle, so it won't be missed.

Data Recording

Another requirement of the assignment was to record all the movement of the car to the project it on the LCD screen and to write the coordinates to the .csv file to inspect afterwards.

As in previous project, I used `FileOutputStream fileStream` and `DataOutputStream dataStream` to get the data to file. First, I create a file to which the data is going to be written. Then initialize the output stream, all in one try-catch block.

```
File file = new File("CE215ass2data.csv");
FileOutputStream fileStream = null;
DataOutputStream dataStream = null;

//Creation and setup of file and streams
try {
    file.createNewFile();
    fileStream = new FileOutputStream(file);
    dataStream = new DataOutputStream(fileStream);
} catch (IOException e) {
    e.printStackTrace();
}
```

Data obtained by odometry class with `opp.getPose().getX()` and `opp.getPose().getY()` is then written as a string into a file. The file will appear to have to columns with coordinates for x, y of robot.

To draw the path of robot on the robot's screen I took the same code from before which fitted into the screen. The code is inside try-catch block as well.

```
try {
    LCD.setPixel(1 + (int) opp.getPose().getX() / 3, 1 + (int) opp.getPose().getY() / 
3, 1);
    dataStream.writeChars(opp.getPose().getX() + ", ");
    dataStream.writeChars(opp.getPose().getY() + "\n ");
    Thread.sleep(50);
} catch (IOException | NullPointerException | InterruptedException e) {
    e.printStackTrace();
}
```

Coordination of Behaviours

Navigator class combines two other class to get information from servomotors and information about dimensions of the car. The odometry class tracks the encoder counts to in motors to keep track of relative location of robot, differential pilot class brings the dimensions to calculate the angles and distance to create an accurate trajectory. In navigator they are used to improve the accuracy and create an easier way to manipulate robot by adding other functions like waypoint and building a route. These features allow us to create a product that can complete a simple navigation task, going from point A to point B. Nevertheless, the accuracy of it is still not perfect. For this instance, another behaviour is brought up. By exploring the space around it, the robot will find the goal position with high possibility. In this assignment the task was made more difficult by putting an obstacle somewhere in the arena. To avoid this obstacle vehicle is equipped with ultrasonic sensor that uses sounds pulses to detect the distance to object in front of it. The code combines the use of all these features of the robot to complete the task. For further analysis and demonstrative purposes, the trajectory is printed onto the screen and recorded in the file. The data obtained can be examined to increase the accuracy and precision of the truck.

Conclusion

Unfortunately, due to unpredicted circumstances and labs closure I was unable to finish the code and all the testing for the assignment, as well as I am unable to provide any evidence other than code to support my words. The tests have been done on some parts of the code, more particularly on all other than exploring behaviour part that has been written without any testing nor debugging. Some other minor parts were also re-written to fit the concept of exploring. Without that, many trials were done, and results showed that car could go from origin the other side and then back with obstacle avoidance without any problem but wasn't stopping exactly at the red spot so that wasn't checked. Part with recording the data was complete but not fully tested, particularly, writing into the file. It was taken from previous assignment, so I assume that it is working.

After completing the assignment, I got skills in programming a differential drive robot and putting theoretical knowledge from lectures into practice.

Appendix

```java
import lejos.nxt.*;
import lejos.robotics.localization.OdometryPoseProvider;
import lejos.robotics.navigation.*;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class ass2 {
    public static void main(String[] args) {
        //Motors
        final NXTRegulatedMotor LEFT = Motor.C;
        final NXTRegulatedMotor RIGHT = Motor.A;

        boolean finish = false;

        //Define starting & goal points
        Waypoint[] points = new Waypoint[]{new Waypoint(0, 0), new Waypoint(150,
150)};
        Waypoint currPoint;

        //Attach sensors to the ports
        ColorSensor colorSensor = new ColorSensor(SensorPort.S4);
        UltrasonicSensor sonicSensor = new UltrasonicSensor(SensorPort.S1);

        //File & Streams
        File file = new File("CE215ass2data.csv");
        FileOutputStream fileStream = null;
        DataOutputStream dataStream = null;

        //Creation and setup of file and streams
        try {
            file.createNewFile();
            fileStream = new FileOutputStream(file);
            dataStream = new DataOutputStream(fileStream);
        } catch (IOException e) {
            e.printStackTrace();
        }

        //Pilot
        DifferentialPilot pilot = new DifferentialPilot(3.37, 17.25, LEFT, RIGHT);
        OdometryPoseProvider opp = new OdometryPoseProvider(pilot);
        pilot.addMoveListener(opp);

        //Navigation
        Navigator nav = new Navigator(pilot);
        nav.setPoseProvider(opp);
        nav.singleStep(true);

        opp.setPose(new Pose(0, 0, 90));
        currPoint = points[0];

        while (!finish) {
            nav.goTo(currPoint);
            while (nav.isMoving()) {
                if (sonicSensor.getDistance() < 15) {
                    nav.stop();
                    pilot.rotate(-90);
                    pilot.arc(20, 180);
                    nav.goTo(currPoint);
                }
                //If the sensor detects red colour at the waypoint
                if (colorSensor.getColorID() == 0 && nav.pathCompleted()) {
                    if(currPoint == points[0]) currPoint = points[1];
                    else finish = true;
                    nav.stop();// move to next waypoint
                    //If the waypoint has been reached but not at the red spot
                } else if (nav.pathCompleted() && colorSensor.getColorID() != 0) {
                    //Beginning of exploring algorithm
                    boolean found = false;
                    int s = 20;   //distance, constant magnitude
                    int a = 60;     //angle of rotation, constant
                    int count = 0;  //constant, to keep track of number of cycles
complete
                    while (!found) {
                        //if the number of COUNT is even then increase the angle
                        if (count % 2 == 0 && count != 0) pilot.rotate(a);
                        pilot.travel(s);
                        s *= -1; //reverse the direction of travel
                        while (pilot.isMoving()) {
                            if (colorSensor.getColorID() == 0) {
                                found = true;
                                if(currPoint == points[0]) currPoint = points[1];
                                else finish = true;
                                nav.stop();
                            }
                        }
                        count++;
                    }
                }
                try {
                    LCD.setPixel(1 + (int) opp.getPose().getX() / 3, 1 + (int)
opp.getPose().getY() / 3, 1);
                    dataStream.writeChars(opp.getPose().getX() + ", ");
                    dataStream.writeChars(opp.getPose().getY() + "\n ");
                    Thread.sleep(50);
                } catch (IOException | NullPointerException | InterruptedException e)
{
                    e.printStackTrace();
                }
            }
        }
        try {
            fileStream.close();
            dataStream.close();
        } catch (IOException | NullPointerException e) {
            e.printStackTrace();
        }
        Button.waitForAnyPress();
    }
}
```