```java
/**
 * Created with IntelliJ IDEA.
 * User: Emmanuel Amodu
 * Date: 02/03/23
 * Time: 12:57
 * To change this template use File | Settings | File Templates.
 */
import robot.Kinematics;
import robot.Control;
import robot.Robot;
import utils.Delay;
import utils.ScatterPlotter;
import org.jfree.ui.RefineryUtilities;
import java.awt.*;
import java.nio.file.NotDirectoryException;
public class Test {
    private  static Robot robot;
    private ScatterPlotter scatterPlotter;
//    private Walk
//    private  static Control control;
    /**
     * Title : Lab 1
     * Description : Implement the robot control methods below, that will be your tools for the conduction of your assignment.
     **/
    /**
     * Method : odometryModel()
     * Purpose : Tangent-node localisation method.
     **/
    private static double  gamaTh = 2.9;
    private static double GAMMA_A = 200.0;
    private static double gamaDis = 200;
    private static final int TURN = 0;
    private static final int MOVE = 1;
```

```
    private int step = MOVE;
    private int i = 0;
    private double turnvel;
    private double intial_angle=0;
    private static double NODE[][] =
        {
            {2400, 0},
          {2400, -2800},
            {4000, -2800},
             {4000, -3900} ,
           { -1950, -3900},
           { -1950, -1100},
             {-4000, -1100}
        };
    public Test(Robot robot)
    {
        this.robot = robot;
//      wallFolow =_new
//      this.vel = vel;
        scatterPlotter = new ScatterPlotter("Laser Scanner", "Scatter Plot", "X", "Y", "Data", Color.BLUE, false);
        RefineryUtilities.centerFrameOnScreen(scatterPlotter);
        scatterPlotter.setVisible(true);
    }
    public boolean odometryModel(double vel)
    {
        double x = NODE[i][0];
        double y = NODE[i][1];
       double th = getAngle(x, y);
        double next_th=   get360(robot.kinematics.getTh());
//      turnvel = vel;
        switch(step)
        {
            case TURN:
            {
```

```
//            System.out.println("\nMy angle " + th);
            robot.control.turnSpot(turnvel/2);
//
            if(isAngularDestination(th))
            {
//                control.stop();
              robot.control.stop();
              step = MOVE;
            }
          }break;
          case MOVE:
          {
            robot.control.move(vel);
            if(isDist(x, y))
            {
//                control.stop();
              robot.control.stop();
//                robot.control.move(vel);
              step = TURN;
//                i++;
//                robot.control.turnSpot(vel/2);
//                robot.control.turnSpot(i/2);
//                System.out.println(i);

              if(++i== NODE.length){
                return (true);
              }
              double new_x = NODE[i][0];
              double new_y = NODE[i][1];
              double next_t = newgetAngle(new_x, new_y, x,y);
              System.out.println("\nOld angle " + th);
              System.out.println("\nNext angle " + next_t);
              //get next angle
              if((next_t>=180 || next_t==0) && (th <90 || th >=320)) {
```

```
                    turnvel = -vel;
                }  else {
                    turnvel = vel;
                }
                //get the velocity
//                  Double vel = vel;
            }
        }break;
    }
    return(false);
  }
  /**
   Method LabExercises::track()
   *Purpose : To perform target tracking using 3 discrete zones.
   *Parameters: vel: The robot velocity.
   *ReturnsTrue if detecting a target, false otherwise.
   Notes Make use of the camera sensors and blob detector.
   **/
  public boolean track(double vel)
  {
    if((robot.sensor.getBlobX() > 0) && (robot.sensor.getBlobX() < (robot.sensor.getImageWidth()/
        3))){
      robot.control.turnSpot(+vel);
      System.out.println("Now robot moves Left <");
      return(true);
    }
    // Validate right image zone and turn right:
    else {
      if((robot.sensor.getBlobX()> ((2 * robot.sensor.getImageWidth())/3)) && (robot.sensor.getBlobX() <
robot.sensor.getImageWidth())){
        robot.control.turnSpot(-vel);
        System.out.println("Now robot moves Right ");
        return(true);
      } else {
```

```
            double f_vec[] = {
                    robot.sensor.getSonarRange(3),
                    robot.sensor.getSonarRange(4)};
            double min = Math.min(f_vec[0], f_vec[1]);
            if(min <= GAMMA_A) {
                robot.control.stop();
             System.out.println("Now robot is stopping ==>");
                Delay.ms(100);
            }
            else robot.control.move(vel);
            System.out.println(" Robot is Moving  ");
            return (false);
        }
    }
  }
  /**
   * Method : avoid()
   * Purpose : Avoid, Decollide, Untrap.
   **/
  public boolean avoid(double vel)
  {
    // [1]Collect left/right sensor ranges:
    double l_vec[] = {
            robot.sensor.getSonarRange(1),
            robot.sensor.getSonarRange(2),
            robot.sensor.getSonarRange(3)
    };
    double r_vec[] = {
            robot.sensor.getSonarRange(4),
            robot.sensor.getSonarRange(5),
            robot.sensor.getSonarRange(6)
    };
// 12]Calculate the left/right min sensor vector:
    double l_min = Math.min(Math.min(l_vec[0], l_vec[1]), l_vec[2]);
```

```
        double r_min = Math.min(Math.min(r_vec[0], r_vec[1]), r_vec[2]);
        System.out.println("Left min  " + l_vec);
        System.out.println("irght min " + r_min);
        // [3]Validate if an obstacle is detected left and turn right:
        if( l_min < GAMMA_A) {
           System.out.println("Left is lower");
           robot.control.turnSpot(vel);
           return (true);
        }
//       {robot.control.turnSpot(this.vel);return (true);}
        else
//       ?/ 14]Validate if an obstacle is detected right and turn left:
           if(r_min< GAMMA_A){
              System.out.println("Right is lower");
              robot.control.turnSpot(vel);
              return(true);
// [S]Otherwise, invoke decollider:
           } else {
              boolean initPose = (robot.kinematics.getX() == 0) && (robot.kinematics.getY() == 0);
              boolean zeroVel = (robot.kinematics.getLeftVel() == 0) && (robot.kinematics.getRightVel()== 0);
              System.out.println("Initial Vel " + initPose + " Get Robot X " + robot.kinematics.getX() + " Get Robot Y " +
robot.kinematics.getY());
              System.out.println("Zero Vel " + initPose + " Get Robot Vel Lef " + robot.kinematics.getLeftVel() + " Robot
Vel right" + robot.kinematics.getRightVel());
              if(zeroVel && !initPose)   {
                 robot.control.move(-vel); // Move backward.
                 Delay.ms(2000);
                 double p = Math.random();
                 if(p<0.5) robot.control.turnSpot(vel);
                 else robot.control.turnSpot(-vel);
                 Delay.ms(2000);
                 System.out.println("Should Turn");
              } else {
                 robot.control.move(vel); // Move forward.
                 System.out.println("Should Move forward");
```

```
            }
        }
    return(false);
}
/**
 * Method : main()
 * Purpose : To run the robot.
 **/
public void main(String args[])
{
    boolean omFlag = false;
    new Run(args);
    while(true)
    {
        if(!omFlag)
        {
            omFlag = odometryModel(100);
        }
        else
        {
            if(!avoid(100))
                track(100);
        }
        Delay.ms(100);
    }
}
/**
 * Method : getAngle()
 * Purpose : To get the tangent angle that points to a pair of coordinates.
 * Parameters : - x : The x coordinate.
 * - y : The y coordinate.
 * Returns : The angle to turn.
 * Notes : None.
 **/
```

```
    public double initial_angle(double x, double y)  {
        return  get360(Math.toDegrees(Math.atan2(x, y)));
    }
    public double newgetAngle(double nx, double ny, double x, double y)  {
        return  get360(Math.toDegrees(Math.atan2(y-ny, x-nx)));
    }
    public double getAngle(double x, double y)  {
//        return Math.toDegrees(Math.atan2(x, y));
//        return getTheta();
        double get360 = get360(Math.toDegrees(Math.atan2(y-robot.kinematics.getY(), x-robot.kinematics.getX())));
//        System.out.println(get360);
        return get360;
    }
    public double get360(double th)  {
//        return Math.toDegrees(Math.atan2(x, y));
//        return getTheta();
        return (th - (360* Math.floor(th / 360.0)));
    }
    /**
     * Method : getRadToDeg()
     * Purpose : To transform radians to degrees.
     * Parameters : - th : The theta angle.
     * Returns : The degrees.
     * Notes : None.
     **/
//    public double getRadToDeg(double th)
    /**
     * Method : get360()
     * Purpose : To normalise the robot's th into [0, 360].
     * Parameters : None.
     * Returns : The normalized angle.
     * Notes : None.
     **/
    public double getTheta()   {
```

```
        return 1;
    }
    /**
     * Method : isAngularDestination()
     * Purpose : To check if the robot has reached its angular destination.
     * Parameters : - th : The theta angle.
     * Returns : true when the angle (th) is reached, false otherwise.
     * Notes : A threshold ensures that the robot will not miss the angle owing to the drift error.
     **/
    public boolean isAngularDestination(double th) {
//        double th_new = 0;
        double newth =   get360(robot.kinematics.getTh());
        if( (newth >= (th - gamaTh)) && (newth <= (th + gamaTh)))
            return true;
//        System.out.println(robot.kinematics.getTh());
        return false;
    }
    public double th_new(double t_new)  {
//        return Math.toDegrees(Math.atan2(x, y));
//        return getTheta();
        return (t_new - (360* Math.floor(t_new / 360))); //where % is the modulous sign
    }
    /**
     * Method :LabExercises::mapBuilder() * Purpose: To implement a mapping algorithm using the robot's front sonar
* Parameters: None. ring
     *Returns :Nothing.
     * Notes
     : The algorithm builds a 2D map and displays it.
     **/
    public void mapBuilder() {
        for(int i=0; i<8; i++) {
            double sonarR = robot.sensor.getSonarRange(i);
            double sonarX = robot.sensor.getSonarX(i);
            double sonarY = robot.sensor.getSonarY(i);
```

```
        double sonarTh = Math.toRadians(robot.sensor.getSonarTh(i));
      double robotTh = Math.toRadians(robot.kinematics.getTh());
        if((sonarR > 100.0) && (sonarR < 5000.0) ) {
          //calculate sonar detected local instance
          double  Xp = sonarX + Math.cos(sonarTh) * sonarR;
          double  Yp = sonarY + Math.sin(sonarTh) * sonarR;
//            calculate the trigonometric components for the robot's left/right side sonars'
          double Xg = Xp * Math.cos(robotTh) - Yp * Math.sin(robotTh);
          double Yg = Xp * Math.sin(robotTh) + Yp * Math.cos(robotTh);
          //calculate sonar detected global instance
          Xg = Xg + robot.kinematics.getX();
          Yg = Yg + robot.kinematics.getY();
          //plot point map instances
          scatterPlotter.series.add(Xg,Yg);
//            System.out.printf("")
        }
      }
    }
    /**
     * Method : isLinearDestination()
     * Purpose : To check if the robot has reached its linear destination using Euclidean distance.
     * Parameters : - x : The x destination coordinate.
     * - y : The y destination coordinate.
     * Returns : true when the destination (x, y) is reached, false otherwise.
     * Notes : The linear threshold is added to the x, y so that to reach a node more precisely.
     **/
    public boolean isDist(double x, double y) {
      if(Math.sqrt(Math.pow(x-robot.kinematics.getX(),2) + Math.pow(y-robot.kinematics.getY(),2)) <= gamaDis)
        return true;
      return false;
    }
}
```