

## **Practical 1**

**Aim:** Write a program to create a robot i) With Gears ii) Without Gears and move it forward, left and right.

### **Description:**

1. Gear()  
Creates a gear instance with right motor plugged into port A, left motor plugged into port B.
2. backward(int duration)  
Starts the backward movement for the given duration (in ms) and stops.
3. forward(int duration)  
Starts the forward movement for the given duration (in ms) and stops.
4. left(int duration)  
Starts to rotate left (center of rotation at middle of the wheel axes) for the given duration (in ms) and stops.
5. right(int duration)  
Starts to rotate right (center of rotation at middle of the wheel axes) for the given duration (in ms) and stops.
6. setSpeed(int speed)  
Sets the speed to the given value (arbitrary units).
7. stop()  
Stops the movement.
8. NxtRobot()  
Creates a turtle robot instance.

## Source Code:

i) With Gear

```
import ch.aplu.robotsim.*;

public class Practical2 {

    public Practical2() {

        NxtRobot nxtRobot = new NxtRobot();

        Gear gearBox = new Gear();

        nxtRobot.addPart(gearBox);

        gearBox.setSpeed(100);

        while(true){

            gearBox.forward(200);

            gearBox.left(280);

            gearBox.right(300);

        }

    }

    public static void main(String[] args)

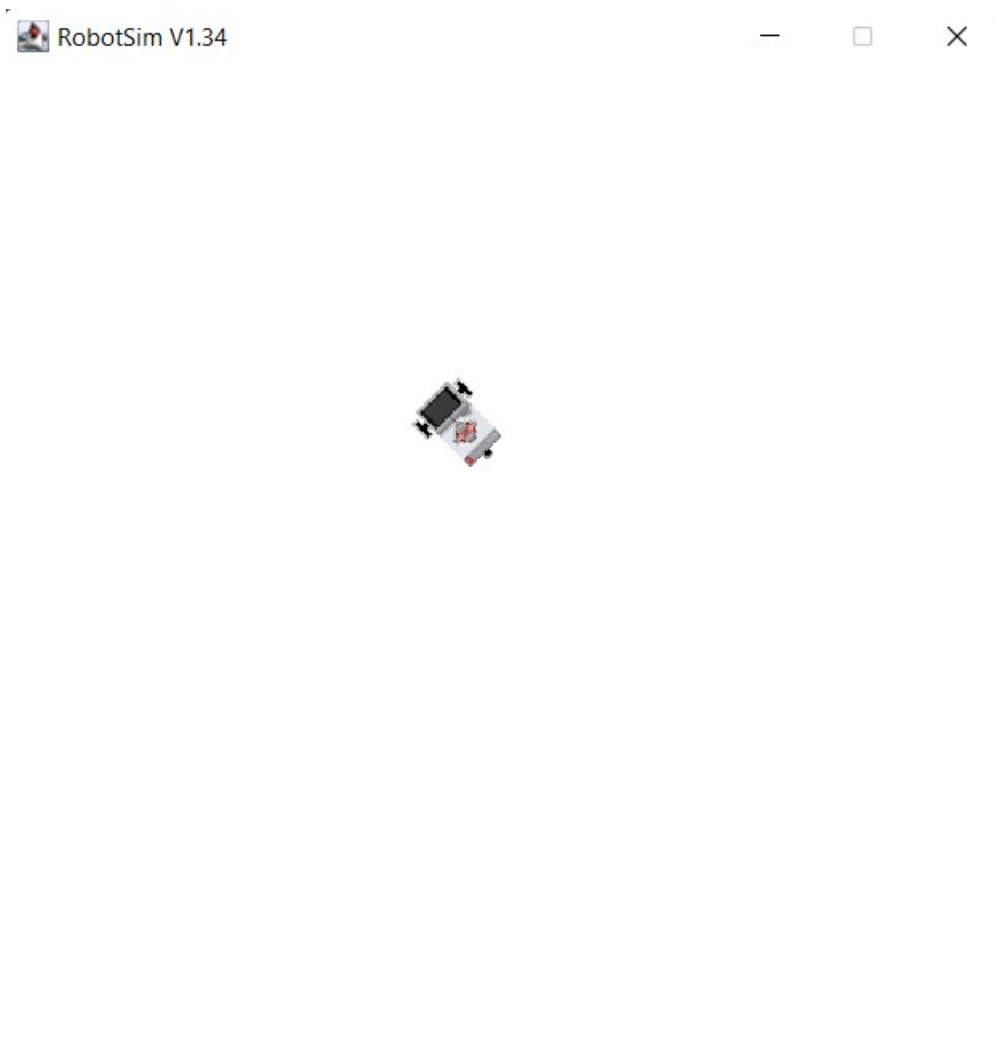
    {

        new Practical2();

    }

}
```

## Output:



## ii) Without Gears

```
package Robotics;

import ch.aplu.robotsim.TurtleRobot;

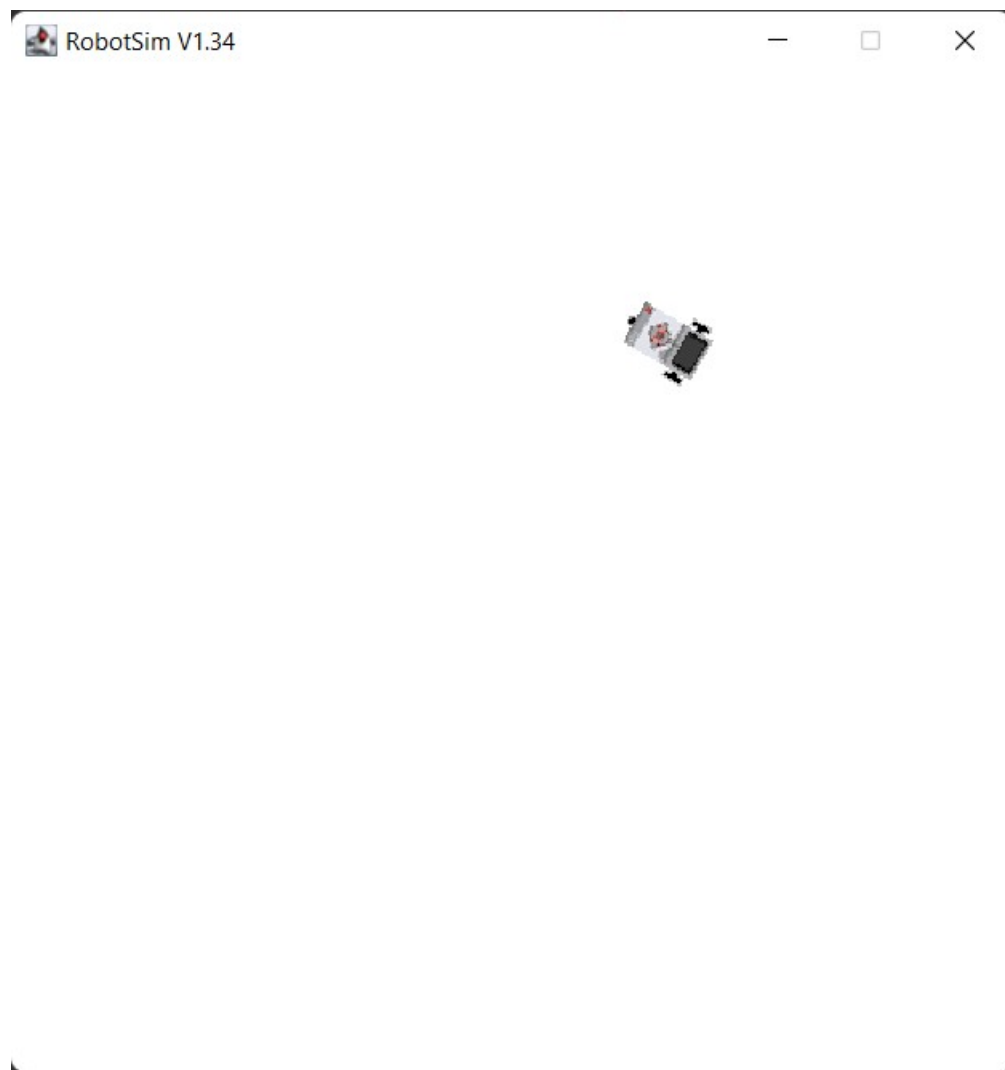
public class RobotMove {

    RobotMove()

    {
        TurtleRobot t = new TurtleRobot();
        t.forward(100);
        t.right(30);
        t.forward(50);
        t.left(90);
        t.forward(70);
    }

    public static void main(String[] args){
        new RobotMove();
    }
}
```

## Output:



## **Practical 2**

**Aim:** Write a program to create a robot with two motors and move it forward, left and right.

### **Description:**

1. Motor(MotorPort port)  
Creates a motor instance that is plugged into given port.
2. MotorPort A  
A motor port for a motor connected to port A.
3. static MotorPort B  
A motor port for a motor connected to port B.
4. static MotorPort C  
A motor port for a motor connected to port C.
5. delay(int duration)  
Suspends execution of the current thread for the given amount of time (unless the game grid window is disposed).

### **Source Code:**

```
import ch.aplu.robotsim.*;

public class RobotMotor {

    public RobotMotor() {

        NxtRobot nxtRobot = new NxtRobot();

        Motor motorLeft = new Motor(MotorPort.A);
        Motor motorRight = new Motor(MotorPort.B);

        nxtRobot.addPart(motorLeft);
        nxtRobot.addPart(motorRight);

        motorLeft.forward();
```

```
motorRight.forward();  
//Right  
Tools.delay(1000);  
motorRight.stop();  
  
Tools.delay(1110);  
motorRight.forward();  
  
//Left  
Tools.delay(1000);  
motorLeft.stop();  
  
Tools.delay(1000);  
motorLeft.forward();  
  
Tools.delay(1000);  
  
Tools.delay(1000);  
motorLeft.stop();  
  
Tools.delay(1000);  
motorLeft.forward();  
}  
  
public static void main(String[] args){  
    new RobotMotor();  
}  
}
```

**Output:**



### **Practical 3**



**Aim:** Write a program to create a robot to perform square using a while loop, doing steps with a for loop.

**Description:**

1. TurtleRobot()  
Creates a turtle robot instance.

**Source Code:**

```
import ch.aplu.robotsim.*;

public class Square {

    public Square() {

        TurtleRobot turtleRobot = new TurtleRobot();

        while(true)

        {

            turtleRobot.forward(100);

            turtleRobot.right(90);

            //turtleRobot.exit();

        }

    }


    public static void main(String[] args){

        new Square();

    }

}
```

**Output:**

 RobotSim V1.34



## **Practical 4**

**Aim:** Write a program to create a Robot using a light sensor to follow a line.

**Description:**

1. RobotContext()  
Creates a RobotContext instance.
2. LegoRobot()  
Creates a robot with its playground using defaults from RobotContext.
3. LightSensor(SensorPort port)  
Creates a sensor instance pointing downwards connected to the given port.
4. LightSensor(SensorPort port, boolean upwards)  
Creates a sensor instance connected to the given port.
5. addPart(Part part)  
Assembles the given part into the robot.

**Source Code:**

```
import ch.aplu.robotsim.*;

public class Practical4 {
    static {
        RobotContext.setStartPosition(50, 470);
        RobotContext.useBackground("sprites/road.gif");
    }
    public Practical4() {
        LegoRobot legoRobot = new LegoRobot();
        Gear gearBox = new Gear();
        LightSensor lightSensor = new LightSensor(SensorPort.S3);

        legoRobot.addPart(gearBox);
```

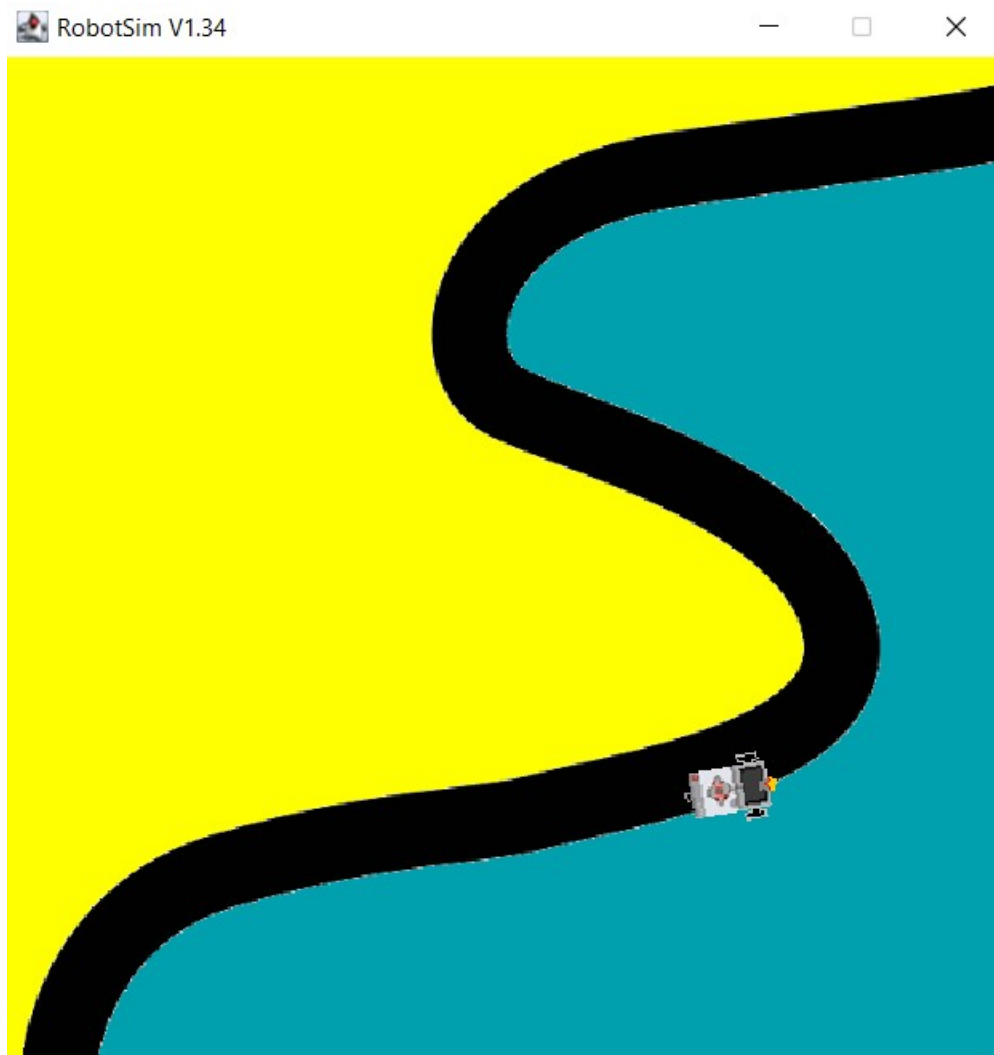
```
legoRobot.addPart(lightSensor);

gearBox.forward();
gearBox.setSpeed(100);

while (true) {
    int lightSensorValue = lightSensor.getValue();
    if(lightSensorValue < 100)
        gearBox.forward();
    else if(lightSensorValue > 350 && lightSensorValue < 750)
        gearBox.leftArc(0.05);
    else if(lightSensorValue > 800)
        gearBox.rightArc(0.05);
}
}

public static void main(String[] args) {
    new Practical4();
}
}
```

**Output:**



## **Practical 5**

**Aim:** Write a program to create a Robot that makes a circle using two motors.

## **Description:**

1.

## **Source Code:**

```
package Robotics;

import ch.aplu.robotsim.*;

public class RobotCircleMotor {

    public RobotCircleMotor() {
        NxtRobot nxtRobot = new NxtRobot();
        Motor motorLeft = new Motor(MotorPort.A);
        Motor motorRight = new Motor(MotorPort.B);

        nxtRobot.addPart(motorLeft);
        nxtRobot.addPart(motorRight);

        while(true) {


            motorLeft.forward();
            motorRight.forward();
            Tools.delay(300);

            motorLeft.stop();
            Tools.delay(300);

        }
    }
}
```

```
}  
  
public static void main(String[] args){  
    new RobotCircleMotor();  
}  
}
```

### **Output:**

 RobotSim V1.34

— □ ×



### **Practical 6**

**Aim:** Write a program to create a path following robot.

## **Description:**

1. leftArc(double radius)  
Starts to move to the left on an arc with given radius.
2. leftArc(double radius, int duration)  
Starts to move left on an arc with given radius for the given duration (in ms) and stops.
3. right(int duration)  
Starts to rotate right (center of rotation at middle of the wheel axes) for the given duration (in ms) and stops.
4. rightArc(double radius)  
Starts to move to the right on an arc with given radius.
5. void rightArc(double radius, int duration)  
Starts to move right on an arc with given radius for the given duration (in ms) and stops.

## **Source Code:**

```
package Robotics;

import ch.aplu.robotsim.*;

public class PathFollower
{
    public PathFollower()
    {
        NxtRobot robot=new NxtRobot();
        Gear gear=new Gear();
        LightSensor ls1=new LightSensor(SensorPort.S1);
        LightSensor ls2=new LightSensor(SensorPort.S2);
        robot.addPart(gear);
        robot.addPart(ls1);
        robot.addPart(ls2);
    }
}
```



```
gear.forward();
```

```
while(true)
```

```
{
```

```
    int rightValue=ls1.getValue();
```

```
    int leftValue=ls2.getValue();
```

```
    int d=rightValue - leftValue;
```

```
    if(d>100)
```

```
        gear.rightArc(0.1);
```

```
    if(d < -100)
```

```
        gear.leftArc(0.1);
```

```
    if(d > -100 && d < 100 && rightValue > 500)
```

```
        gear.forward();
```

```
}
```

```
}
```

```
public static void main(String args[])
```

```
{
```

```
    new PathFollower();
```

```
}
```

```
static
```

```
{
```

```
    NXTContext.setStartPosition(250,490);
```

```
    NXTContext.setStartDirection(-90);
```

```
    NXTContext.useBackground("sprites/path.gif");
```

```
}
```

```
}
```

## Output:



## Practical 7

Aim: Write a program to create a Robot to resist obstacle.

## **Description:**

1. `setStartDirection(double direction)`  
Sets the Nxt starting direction (zero to EAST).
2. `static void setStartPosition(int x, int y)`  
Sets the Nxt starting position (x-y-coordinates 0..500, origin at upper left).
3. `useBackground(java.lang.String filename)`  
Use the given image as background (playground size 501 x 501).
4. `TouchSensor(SensorPort port)`  
Creates a sensor instance connected to the given port.

## **Source Code:**

```
import ch.aplu.robotsim.Gear;
import ch.aplu.robotsim.LegoRobot;
import ch.aplu.robotsim.RobotContext;
import ch.aplu.robotsim.SensorPort;
import ch.aplu.robotsim.TouchSensor;

public class Practical6 {

    static {
        RobotContext.setStartPosition(15, 240);
        RobotContext.useObstacle(RobotContext.channel);
        RobotContext.setStartDirection(30);
    }

    public Practical6() {
```

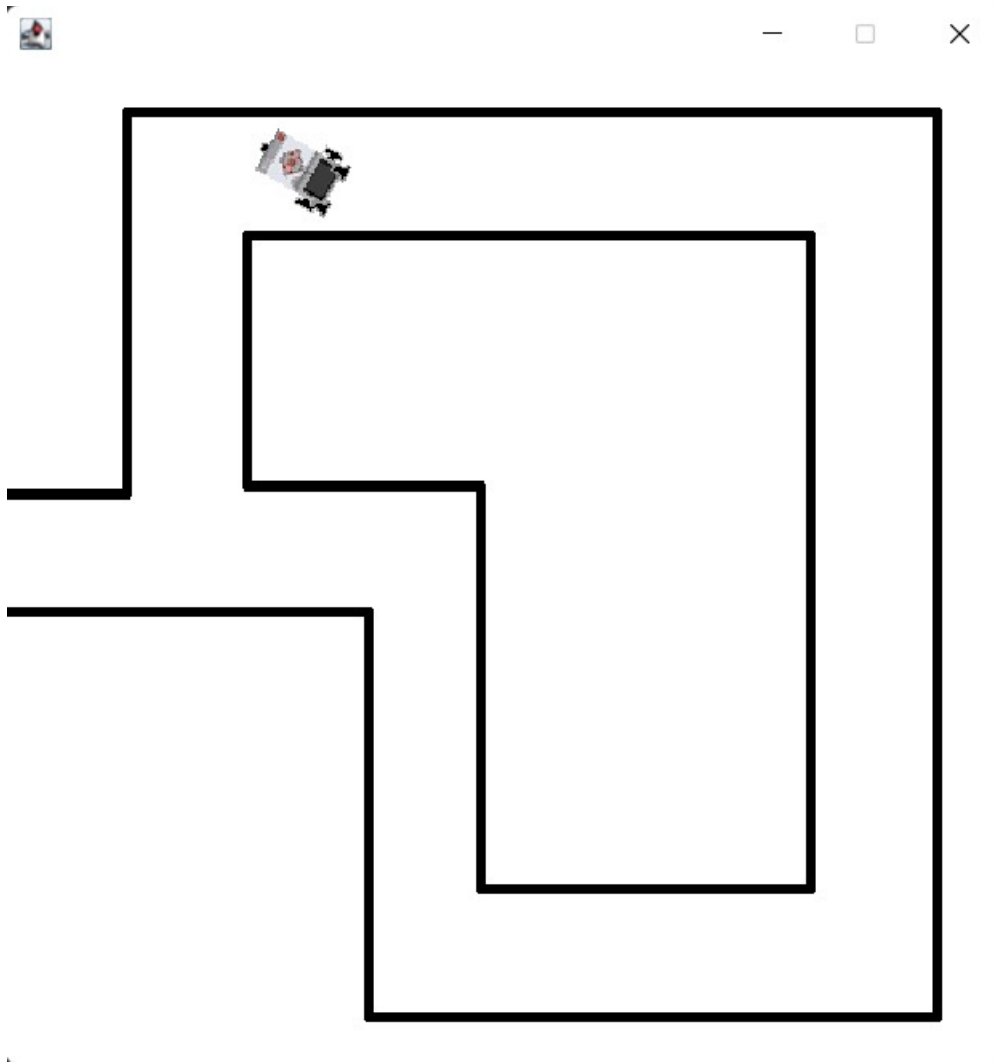
```
LegoRobot legoRobot = new LegoRobot();  
Gear gear = new Gear();  
TouchSensor touchSensorL = new TouchSensor(SensorPort.S2);  
TouchSensor touchSensorR = new TouchSensor(SensorPort.S1);  
  
legoRobot.addPart(gear);  
legoRobot.addPart(touchSensorL);  
legoRobot.addPart(touchSensorR);  
  
gear.forward();  
while(true) {  
    if(touchSensorL.isPressed() && touchSensorR.isPressed()) {  
        gear.backward(40);  
        gear.right(400);  
        gear.forward();  
    }  
    else {  
        if (touchSensorL.isPressed()) {  
            gear.backward(40);  
            gear.right(400);  
            gear.forward();  
        }  
        else if(touchSensorR.isPressed()) {  
            gear.backward(40);  
            gear.left(400);  
            gear.forward();  
        }  
        else {
```

```
        gear.forward();
    }
}
}
}

public static void main(String[] args) {
    new Practical6();
}

}
```

**Output:**



### **Practical 8**

**Aim:** Write a program to create a Robot with Ultrasonic Sensor.

## **Description:**

1. UltrasonicSensor(SensorPort port)  
The port selection determines the position of the sensor and the direction of the beam axis.
2. setProximityCircleColor(java.awt.Color color)  
Sets the color of the circle with center at sensor location and radius equals to the current distance value.
3. useTarget(GGBitmap bm, java.awt.Point[] mesh, int x, int y)  
Creates a target for the ultrasonic sensor using the given GGBitmap.
4. setStartPosition(int x, int y)  
Sets the Nxt starting position (x-y-coordinates 0..500, origin at upper left).

## **Source Code:**

```
package Robotics;

import ch.aplu.robotsim.*;
import java.awt.Color;
import java.awt.Point;
import java.awt.color.*;

public class Ultra_Ex {
    public Ultra_Ex()
    {
        LegoRobot robot= new LegoRobot();
        Gear gear = new Gear();
        robot.addPart(gear);
        UltrasonicSensor us= new UltrasonicSensor(SensorPort.S1);
```

```
robot.addPart(us);  
us.setProximityCircleColor(Color.lightGray);
```

```
double arc=0.5;  
gear.setSpeed(50);  
gear.rightArc(arc);  
boolean isRightArc=true;
```

```
int oldDistance=0;
```

```
while(true)  
{  
    Tools.delay(100);  
    int distance = us.getDistance();  
    if (distance ==-1)  
    {  
        continue;  
    }  
    if (distance<oldDistance)  
    {  
        if(isRightArc)  
        {  
            gear.leftArc(arc);  
            isRightArc =false;  
        }  
        else  
        {  
            gear.rightArc(arc);  
            isRightArc=true;
```



```

        }
    }
    oldDistance=distance;

}

}

static
{
    Point[] mesh_bar=
    {
        new Point(10,200), new Point(-10,200),
        new Point(-10,-200), new Point(10,-200)
    };
    RobotContext.useTarget("sprites/bar1.gif",mesh_bar, 200,250);
    RobotContext.useTarget("sprites/bar1.gif",mesh_bar, 300,250);
    RobotContext.setStartPosition( 250,460);

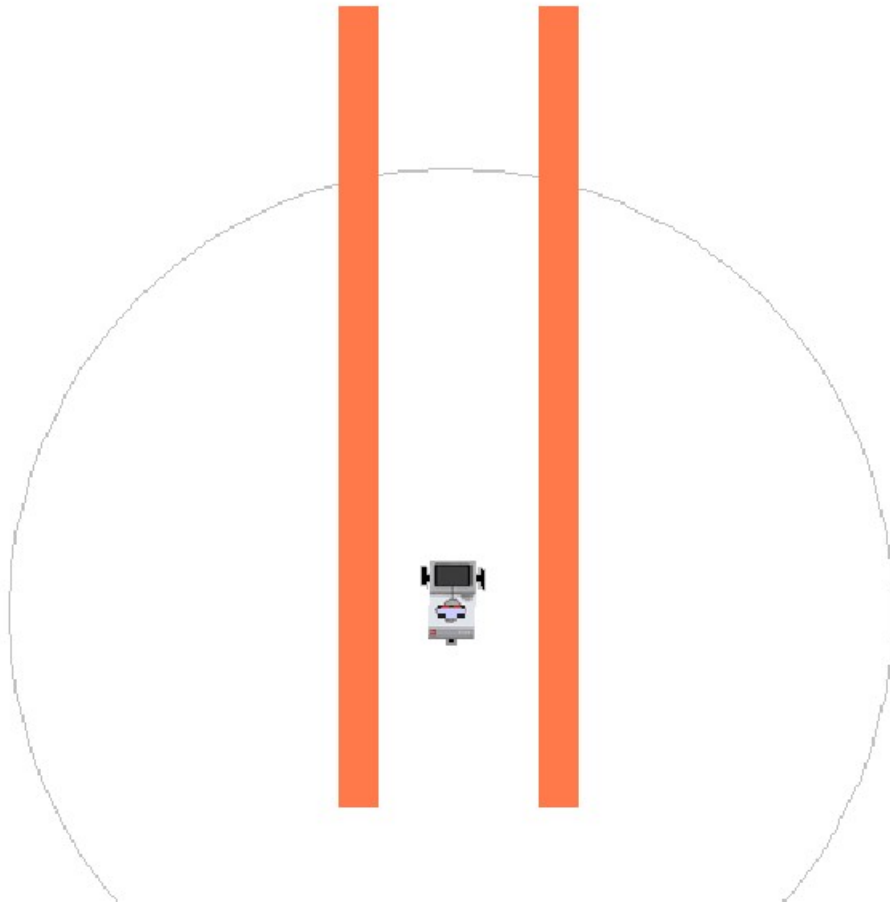
}

public static void main(String args[])
{
    new Ultra_Ex();
}

}

```

### **Output:**



## Practical 9

**Aim:** Drag and Bot Simulator Demo.

**Description:**

Drag & Bot as a software platform for development of ROS-based industrial applications

Drag & Bot is the software platform for simple, graphical setup and programming of robot systems.

Drag & Bot enables manufacturing companies to flexibly and economically automate small lot sizes.

Drag & Bot pursues to be for ROS what Windows was for MS-DOS.

Drag & Bot STUDIO: develop

- Website running on browser
- Intuitive
- Graphical

Drag & Bot RUNTIME: produce

- IPC-based execution environment
- ROS is running inside

Drag & Bot STUDIO: develop

- Website running on browser
- Intuitive
- Graphical

Drag & Bot RUNTIME: produce

- ROS running inside on the cloud
- Includes a robot simulator with different URDF models and inverse kinematics
- Includes a gripper and a machine tending interactive environment

Drag & Bot allows to create ROS-based industrial applications with an easy interface for the user.

Users are able to use, configure and reprogram industrial cells.

With Drag & Bot you can fast prototype new applications.

New hardware or software modules can be integrated in a couple of days or even hours.

Drag & Bot transforms industrial robots into flexible production tools.

Our software is the easiest way for flexible programming of robot systems.

It enables your workers to use robots and other machines independently for automation.

Drag & Bot is an app for controlling industrial robots.

The software is for use by production and industrial companies, enabling system programming with little effort.

A special feature of the app is that it lets staff program tasks independently, which allows them to adapt robots to change production tasks and benefit from automation advantages.

Staff should ideally be able to use available robots like they would use tools for performing specific tasks.

## **Practical 10**

**Aim:** Pickup object using Drag & Bot Simulator

**Description:**

## **Why drag&bot for flexible robots in pick & place applications?**

As different as manufacturing companies and their production processes are, the handling of parts of different materials, shapes and properties is present almost everywhere. In almost every operation, parts must be removed from bins, boxes, containers, workpiece carriers or grids and placed back in a different orientation or exact positioning.

When done manually by a worker, this can quickly become very monotonous and tedious. That is why more and more smaller companies are using robots for these tasks. Pick&Place industrial robots do not get tired and make fewer mistakes. They are easy to acquire, quick to install. They often have a high cycle time, increasing throughput in production.

The processes are often simple, a standard gripper on the arm is often sufficient – special sensors and intermediate stations are not necessary. This saves investment costs and leads to a fast implementation of the projects. Robots in the pick&place area have one of the best return-on-investment (ROI) calculations compared to other robot applications. The use of robots often pays for itself in less than a year.

With drag&bot, pick&place applications are easy to operate by workers, flexible to use and maximally cost-efficient. More than every second robot sold worldwide is now used in the pick&place sector – and the trend is rising!

## **Loading/unloading machines**

Machine loading with robots is one of the most common pick-and-place tasks. We work with many machine builders who want to offer a flexible automation solution to their machines. Typical extensions of these robot systems are cleaning, quality checks and/or marking of the workpieces. The video shows an automatic bending press machine from Placke, equipped with a Fanuc LR Mate 200iD and a vacuum gripper.

## **Assignment 1**

**Aim:** Write a program to create a robot to follow a track using light sensor.

**Source Code:**

```
import ch.aplu.robotsim.*;

public class Assignment6 {
    static {
        RobotContext.setStartPosition(80, 438);
        RobotContext.useBackground("sprites/track.png");
    }

    public Assignment6() {
        LegoRobot legoRobot = new LegoRobot();
        Gear gearBox = new Gear();
        LightSensor lightSensor = new LightSensor(SensorPort.S3);

        legoRobot.addPart(gearBox);
        legoRobot.addPart(lightSensor);

        gearBox.forward();
        gearBox.setSpeed(100);

        while (true) {
            if(lightSensor.getValue() > 10){
                gearBox.forward();
            }
            else{
                gearBox.rightArc(0.03);
            }
        }
    }

    public static void main(String[] args) {
```

```
new Assignment6();  
}  
}
```

**Output:**



**Assignment 2**

**Aim:** Write a program to create a robot to make a circular motion using while loop.

**Source Code:**

```
import ch.aplu.robotsim.TurtleRobot;

public class RobotCircle {
    RobotCircle()
    {
        TurtleRobot t = new TurtleRobot();
        while(true){
            t.forward(3);
            t.right(3);
        }

    }

    public static void main(String []args){
        new RobotCircle();
    }
}
```

**Output:**





## Assignment 3

**Aim:** Write a program to create a robot using gears and perform a circular motion.

**Source Code:**

```
import ch.aplu.robotsim.Gear;
import ch.aplu.robotsim.NxtRobot;

public class RobotGearCircle {
    public RobotGearCircle() {
        NxtRobot nxtRobot = new NxtRobot();
        Gear gearBox = new Gear();
        nxtRobot.addPart(gearBox);

        gearBox.setSpeed(500);
        while(true){
            gearBox.rightArc(0.5);
        }
    }

    public static void main(String[] args){
        new RobotGearCircle();
    }
}
```

**Output:**

