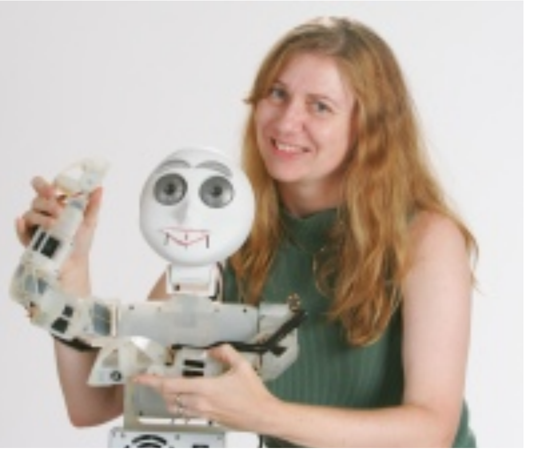
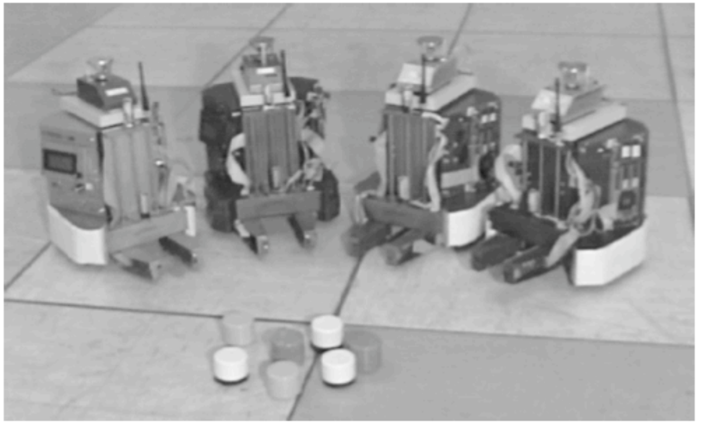
Fred Martin

- Sequential strategies
- Reactive strategies

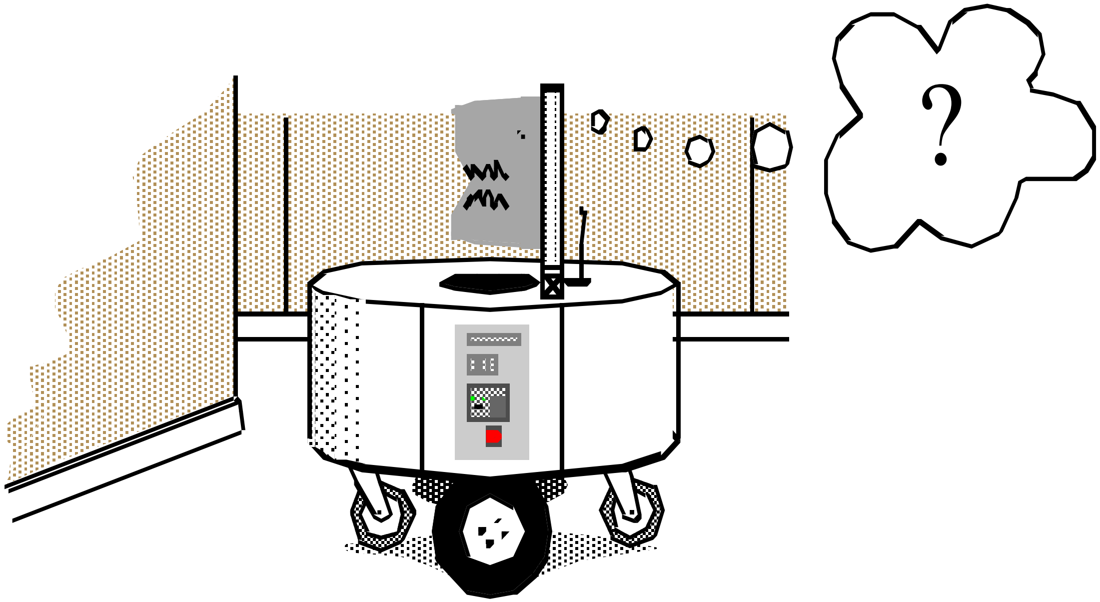




Maja Mataric

- Deliberative approach
- Reactive approach
- Behavior-Based approach

Figure 6: The family of four IS Robotics mobile robots used in the group behavior experiments. The robots are equipped with IRs, contact sensors, grippers, position sensors, and radio communication.



Where am I? *Localization*

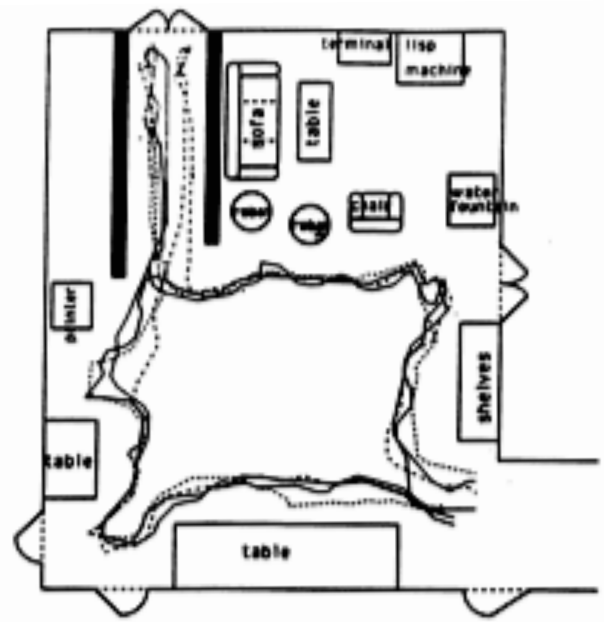
Where have I been? *Map making*

Where am I going? *Mission planning*

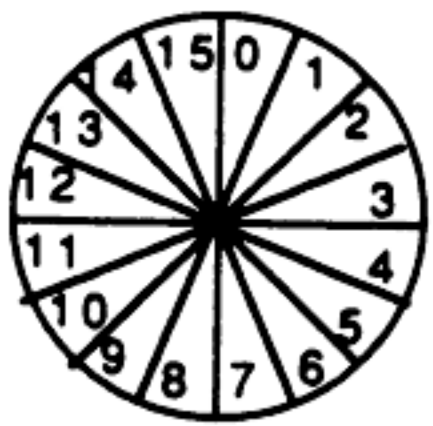
What's the best way there? *Path planning*

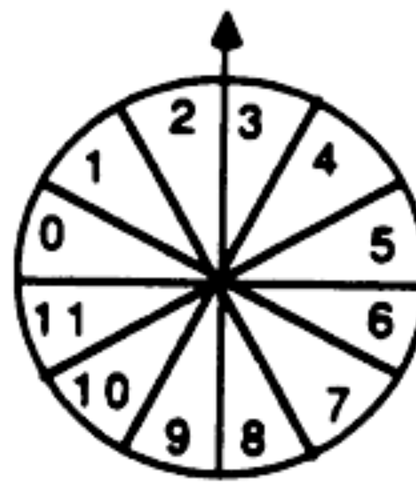
Integration of Representation Into Goal-Driven Behavior-Based Robots

Maja J. Mataric

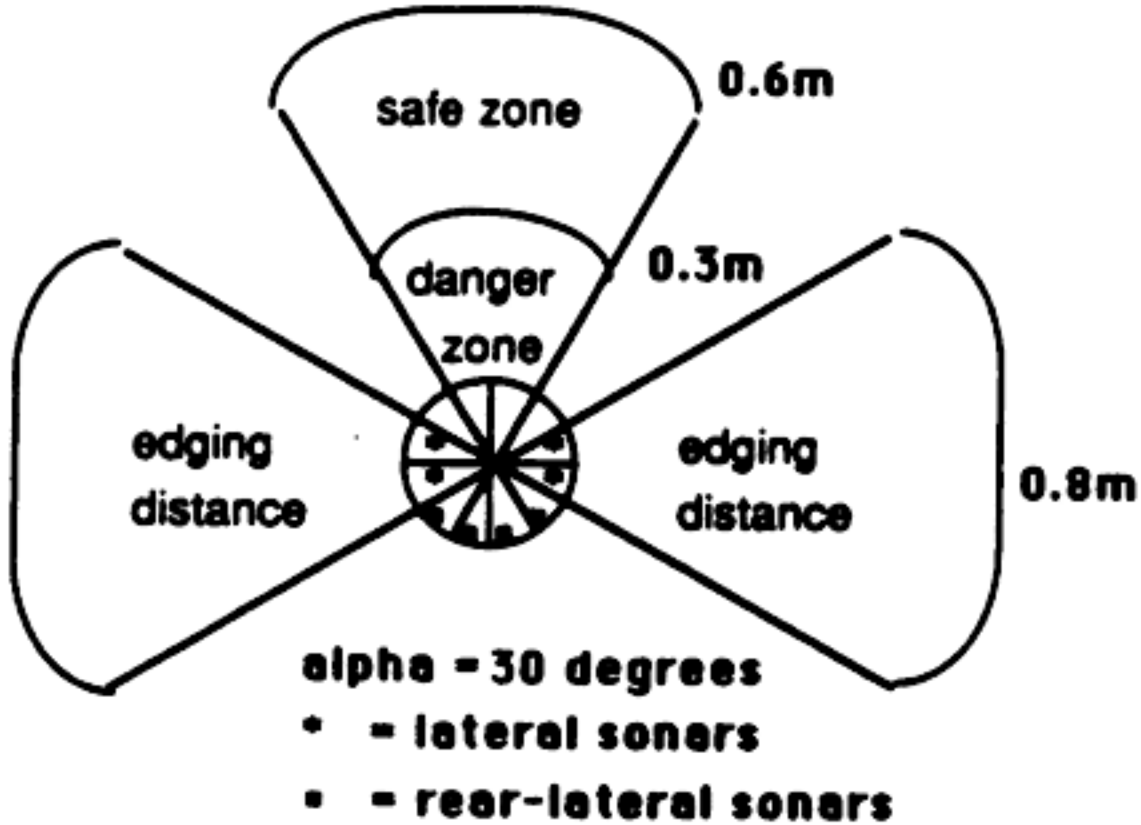


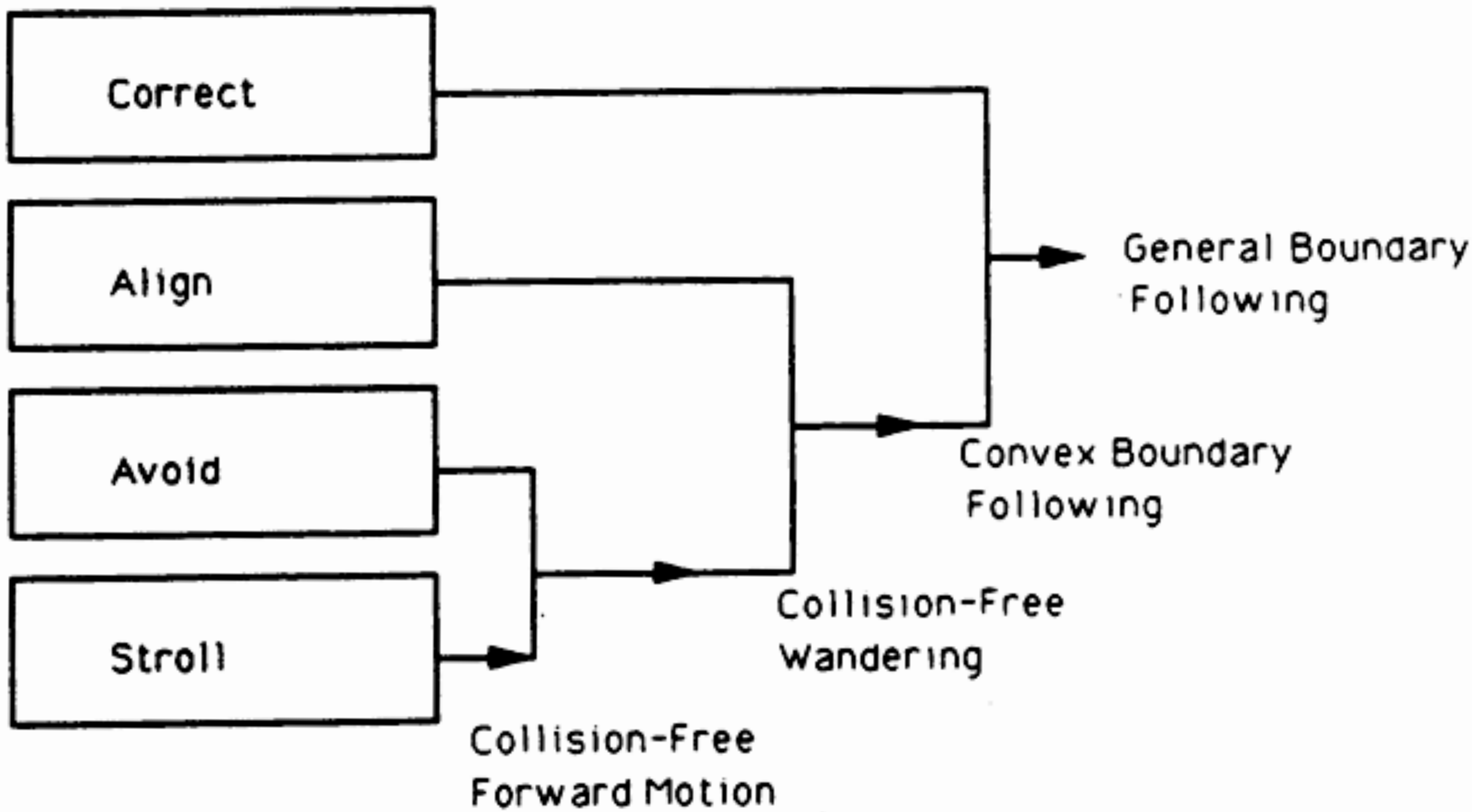
Shaft encoders
Sonic range finders
Compass

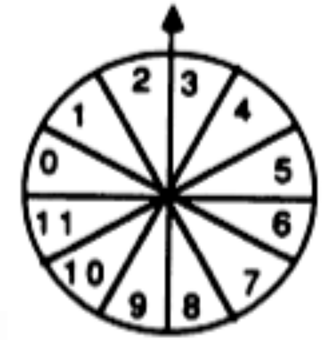




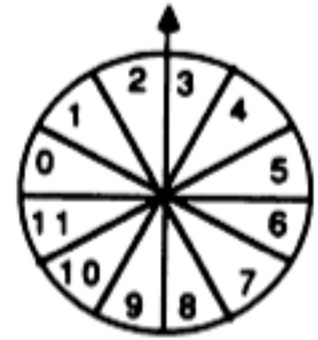
Perceptual zones for sonic range finders




Correct
Align
Avoid
Stroll
General Boundary Following
Convex Boundary Following
Collision-Free Wandering
Collision-Free Forward Motion




```
(defbehavior stroll
   (cond
        ((and (< = (min (sonars 1 2 3 4)
         danger-zone))
              (not stopped))
           (stop))
        ((and (< = (min (sonars 1 2 3 4)
         danger-zone))
              (stopped))
           (move backward))
        (t
           (move forward))))
```




```
(defbehavior avoid
   (cond
      ((and (< = (sonar 1 or 2) safe-
        zone)
            (< = (sonar 3 or 4) safe-
             zone))
          (turn left))
      ((< = (sonar 3 or 4) safe-zone)
          (turn right))))
```

1 or 2 ?

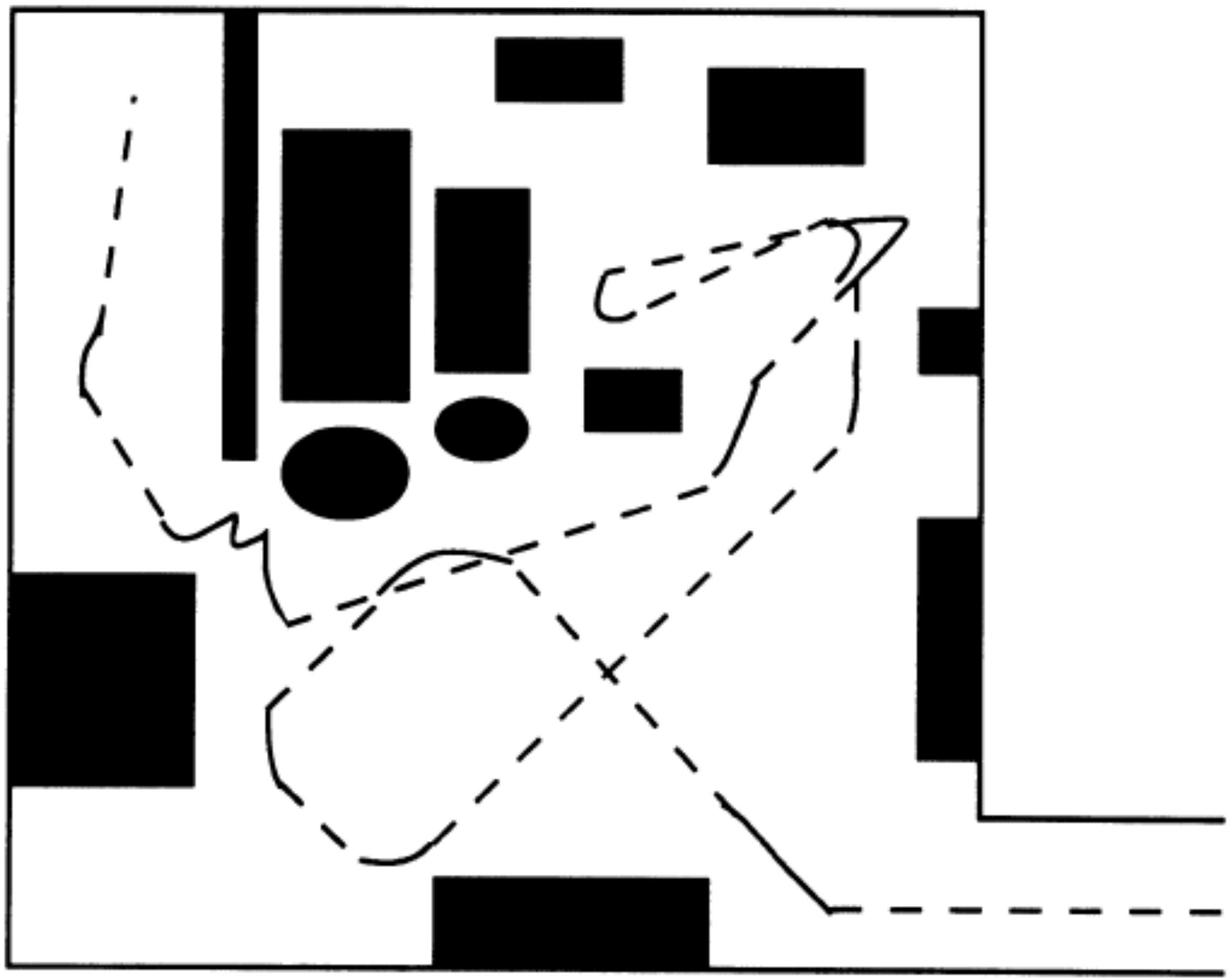
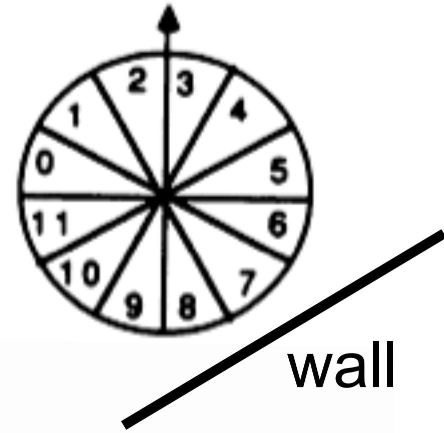
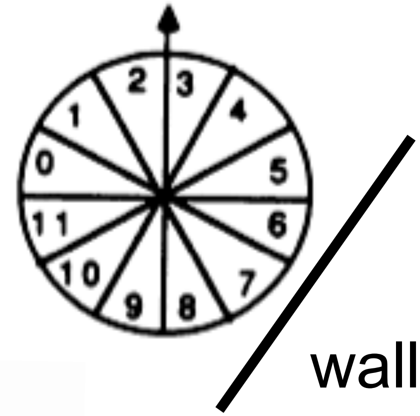


Figure 5.4: The performance of the combined stroll and avoid behaviors. Stroll produces straight-line path segments shown with dashed lines. Path segments generated by avoid are shown with continuous lines.

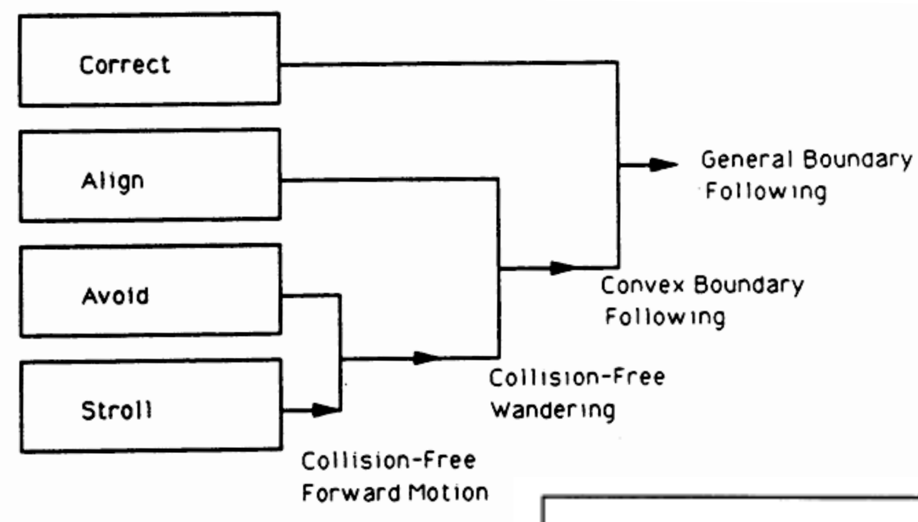



```
(defbehavior align
   (cond
       ((and (< (sonar 7 or 8) edging-
        distance)
             (> (sonar 5 or 6) edging-
              distance))
          (turn right))
       ((and (< (sonar 9 or 10) edging-
        distance)
             (> (sonar 11 or 0) edging-
              distance))
          (turn left))))
```




```
(defbehavior correct

(cond
      ((and (< (sonar 11) edging-
       distance)
              (> (sonar 0) edging-
               distance))
         (turn left))
      ((and (< (sonar 6) edging-distance)
              (> (sonar 5) edging-
               distance))
```

turn right



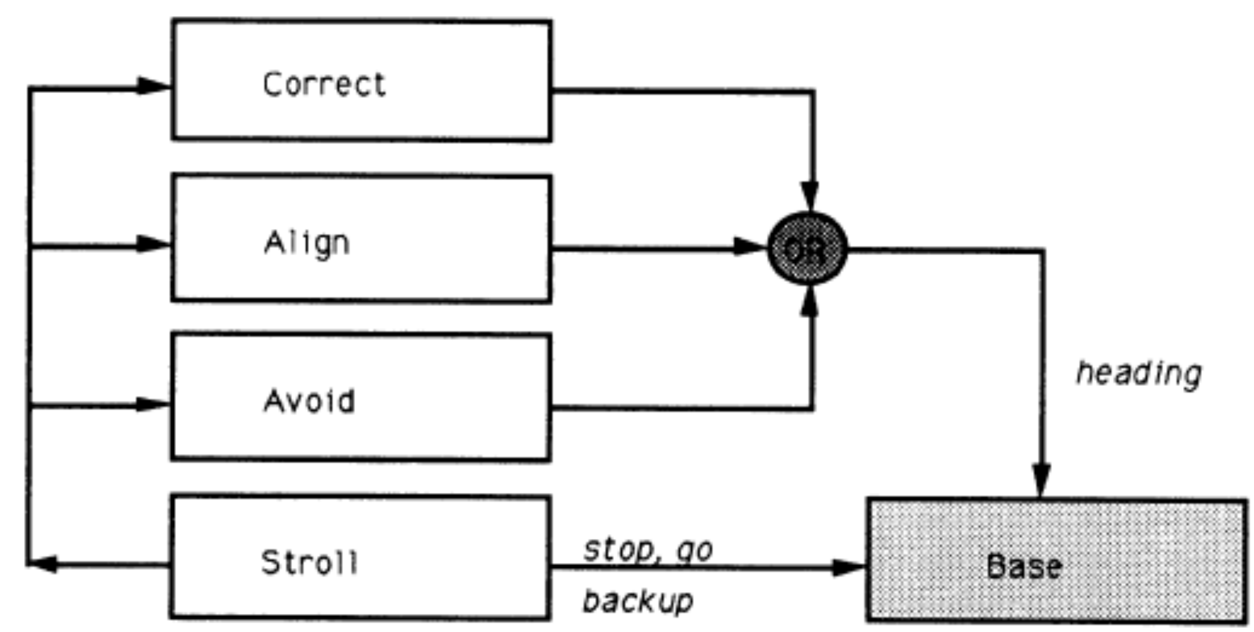
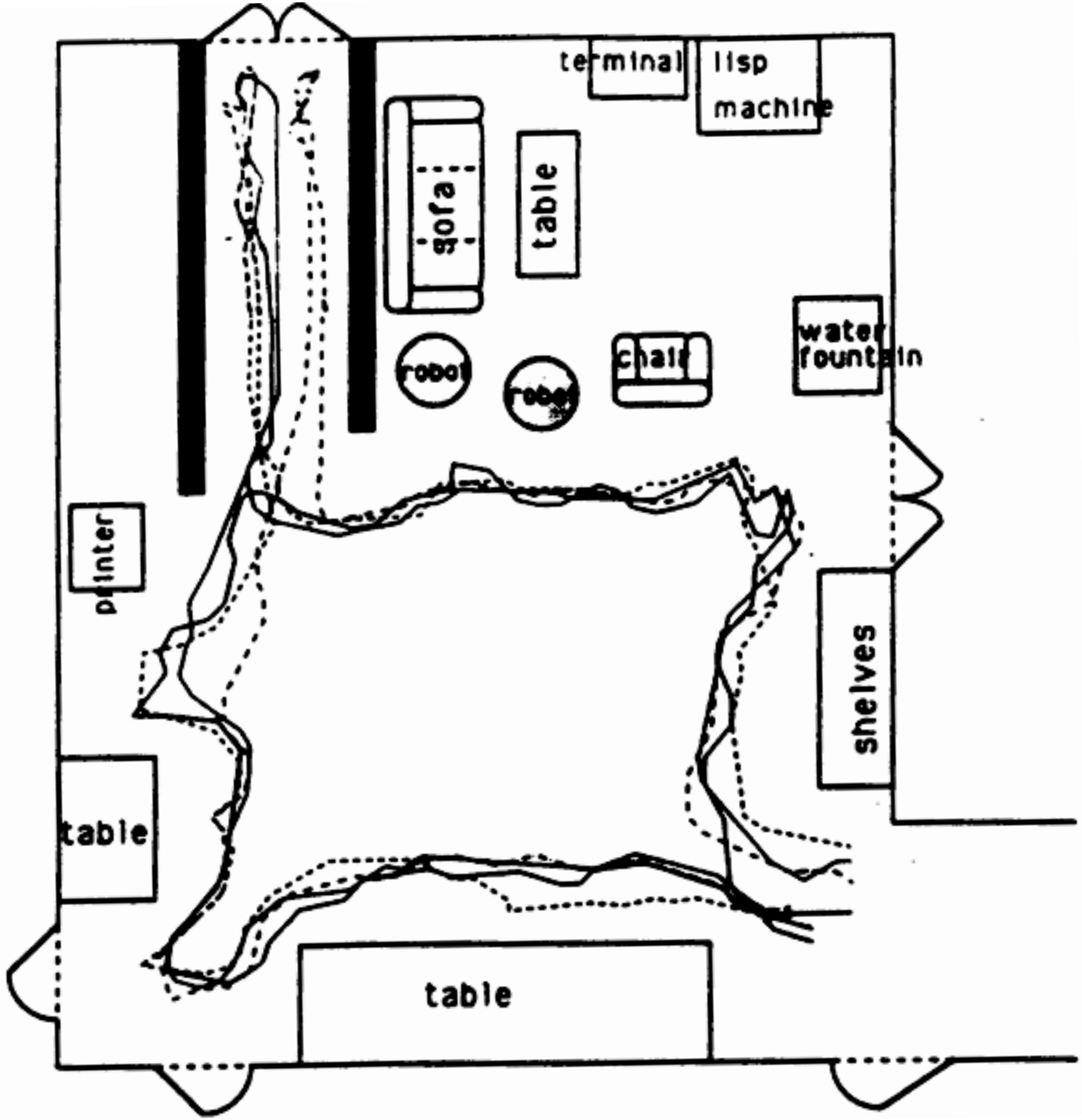
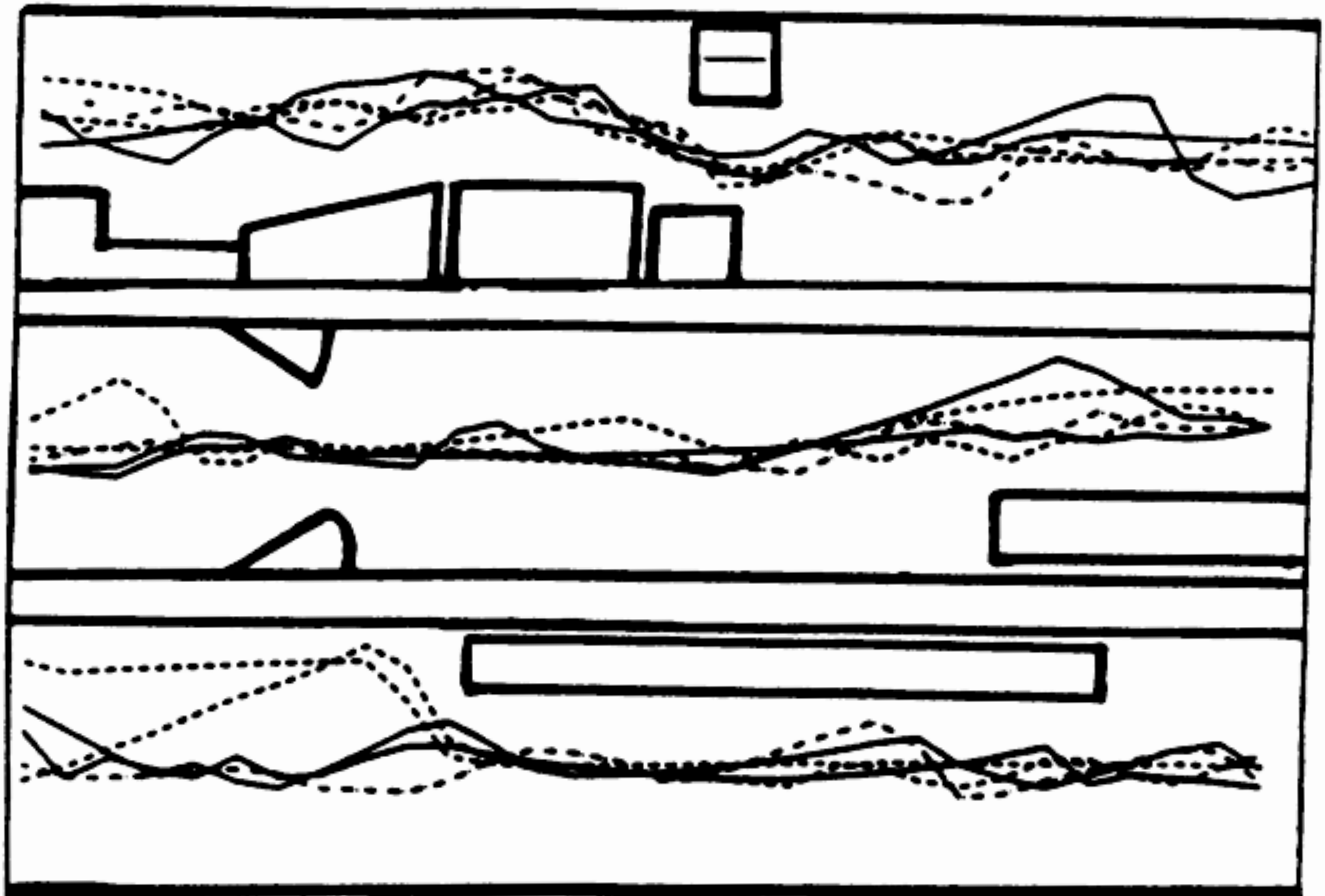
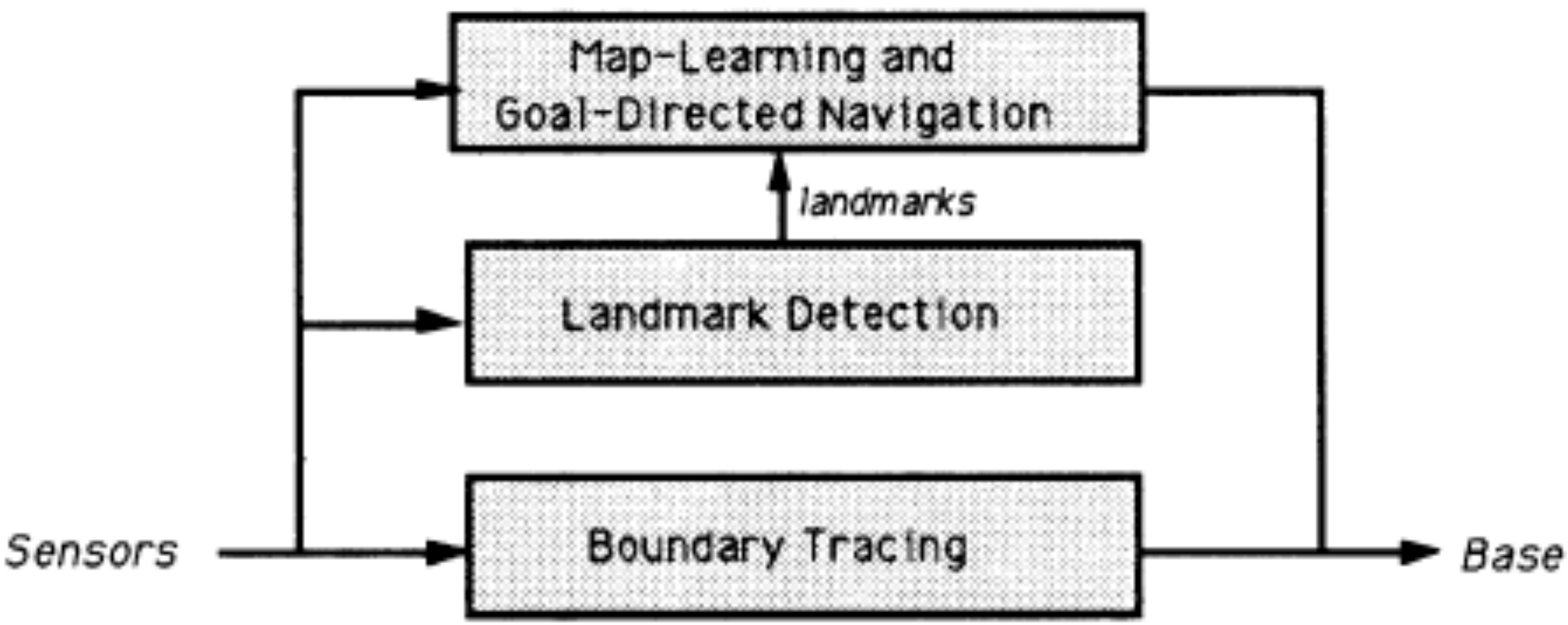


Figure 5.8: A schematic illustrating the implicit arbitration among the low-level navigation behaviors. Since the conditions triggering each of the four low-level behaviors are mutually exclusive, no explicit arbitration is needed.


terminal
lisp machine
sofa
table
robot
robot
chair
water fountain
printer
shelves
table
table



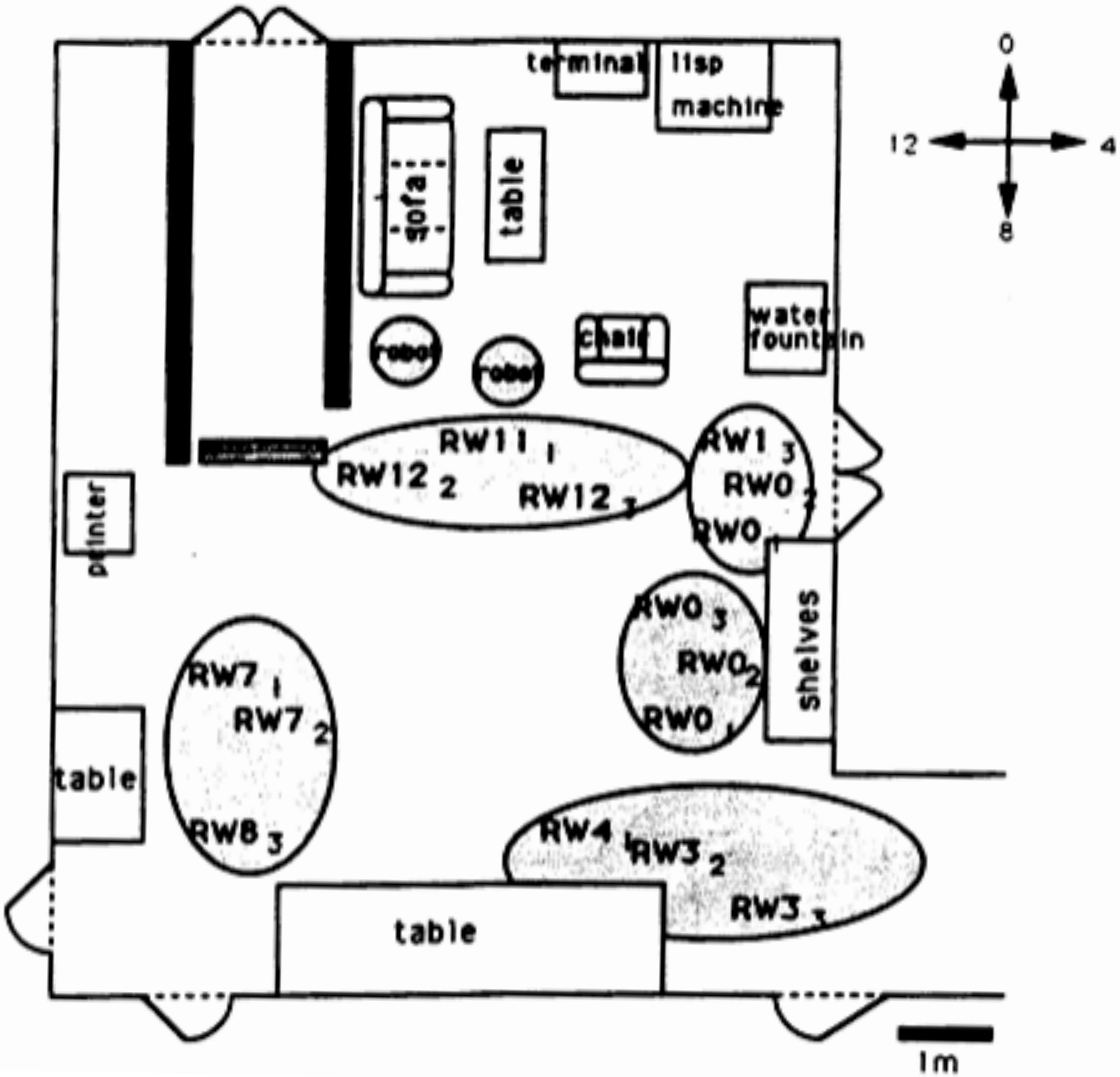

Map-Learning and
Goal-Directed Navigation
landmarks
Landmark Detection
Sensors
Boundary Tracing
Base

Landmark detection

Landmarks are walls either on one side or on both sides: LW, RW, C.

Wall detection: average compass bearing stable and repeatedly short readings on its side then confidence counter incremented

confidence counter > threshold

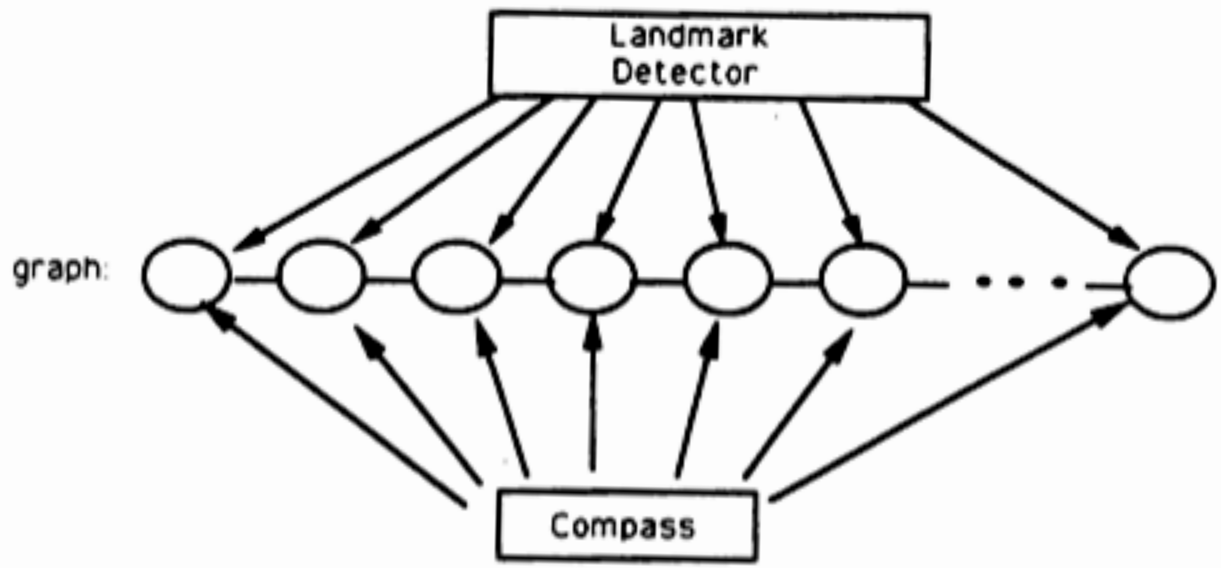


terminal
lisp machine
sofa
table
robot
robot
chair
water fountain
0
12
4
8
RW11 1
RW12 2
RW12 3
RW1 3
RW0 2
RW0 1
printer
RW0 3
RW0 2
RW0 1
shelves
RW7 1
RW7 2
table
RW8 3
RW4 1
RW3 2
RW3 3
table
1m

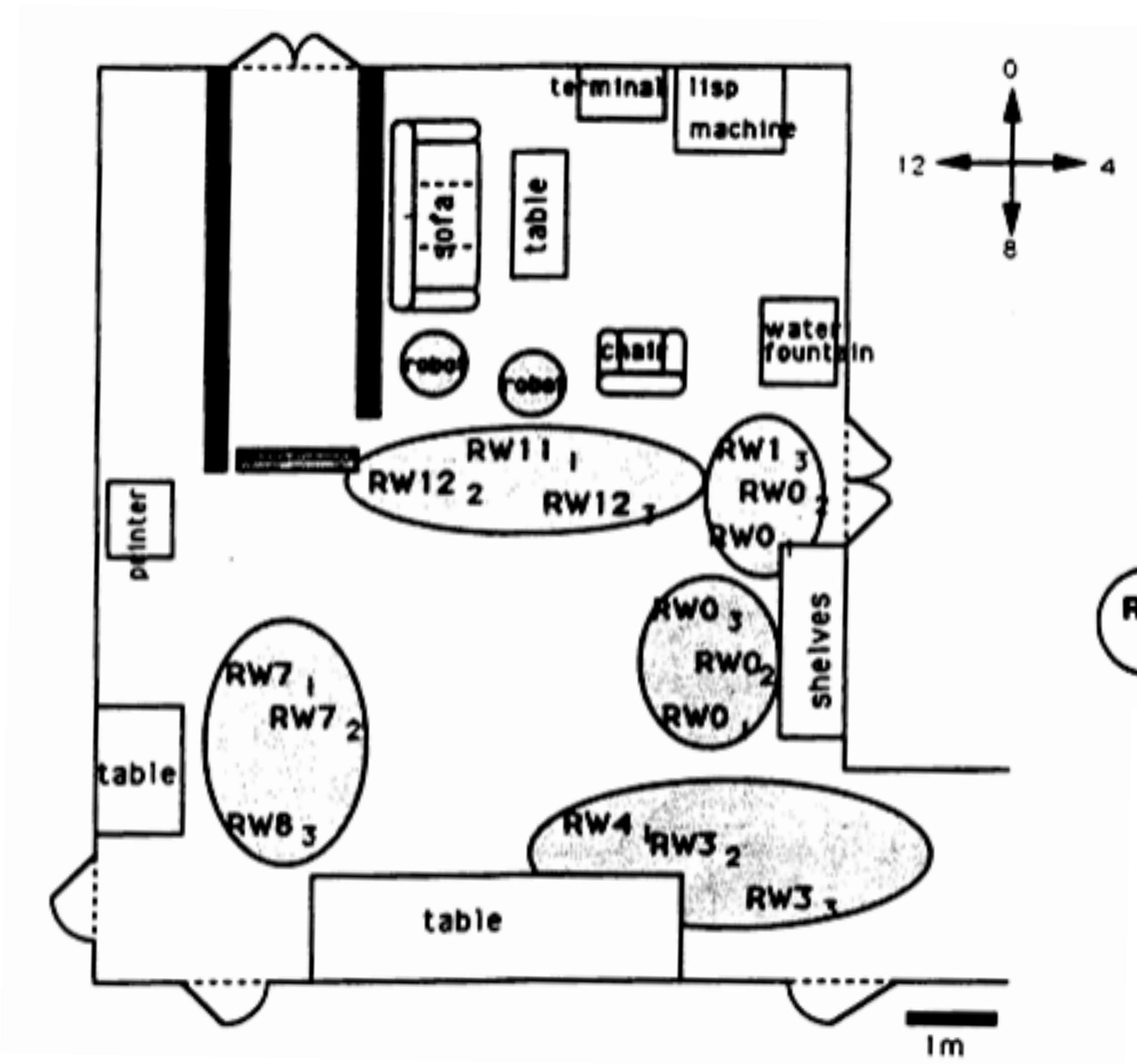
Landmark descriptor

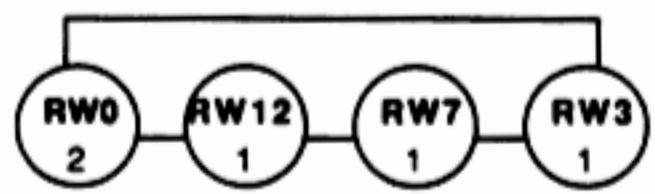
<T,C,L,P>

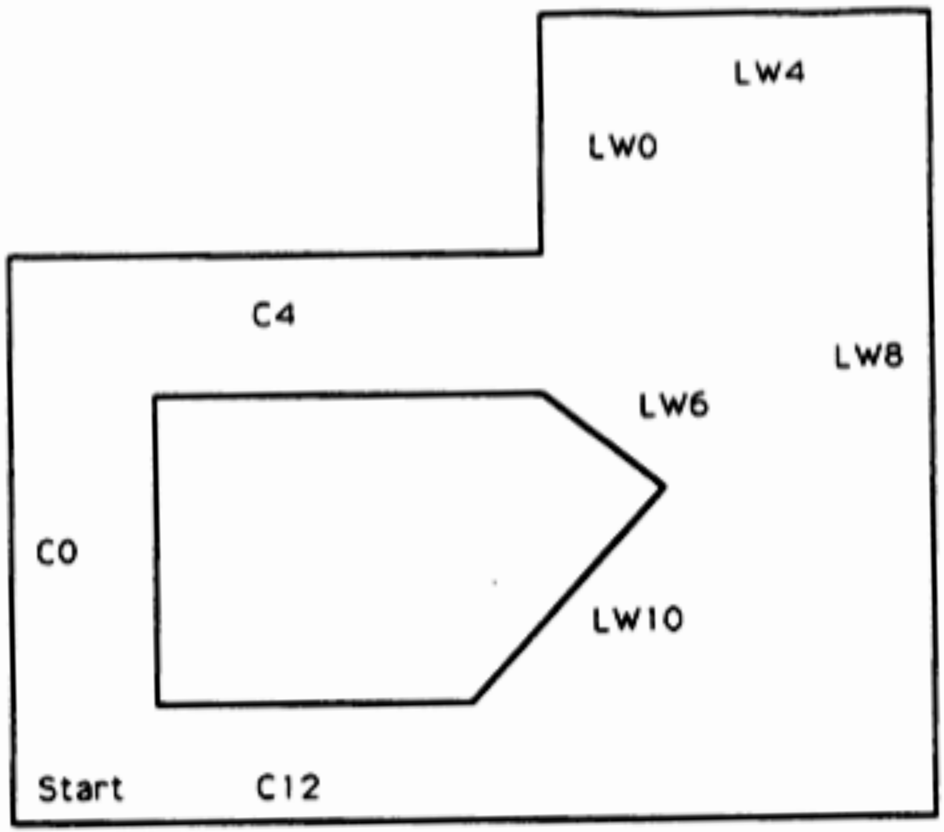
T: LW, RW, C, I
C: compass bearing
L: length
P: position (x,y)


Landmark Detector
graph:
Compass

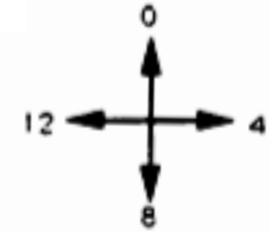
Map building



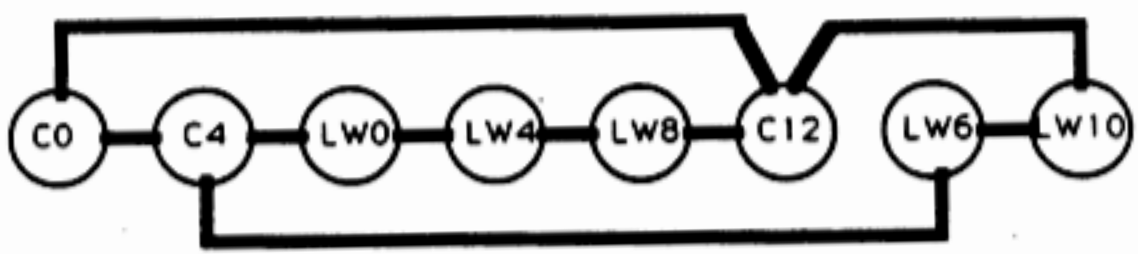




LW4
LW0
C4
LW8
LW6
C0
LW10
Start
C12

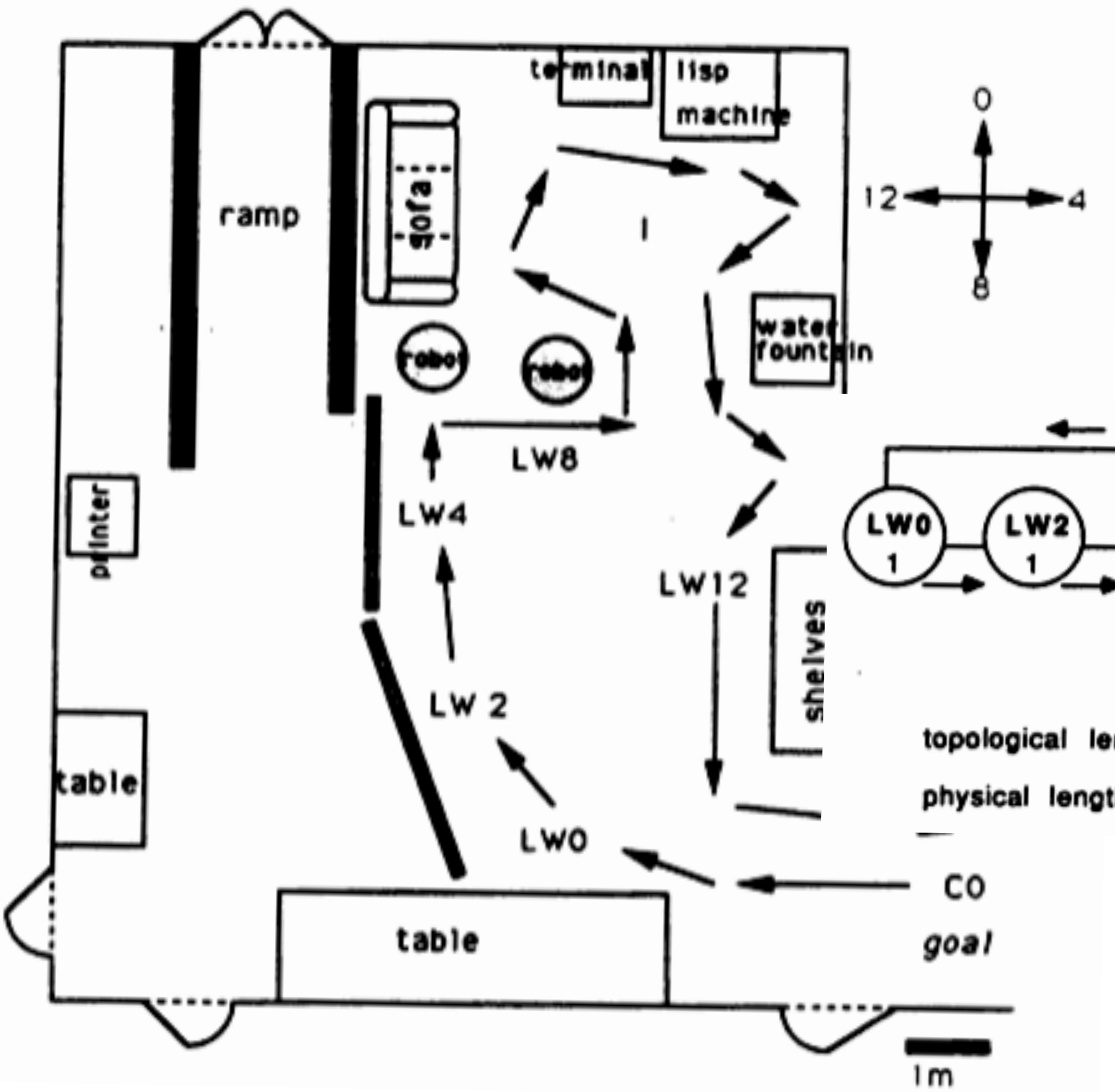


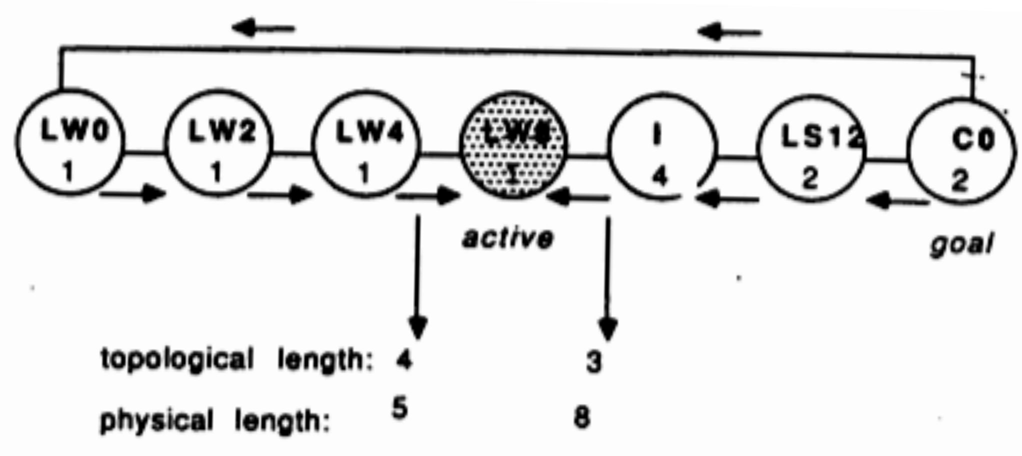
0
12
4
8



C0
C4
LW0
LW4
LW8
C12
LW6
LW10

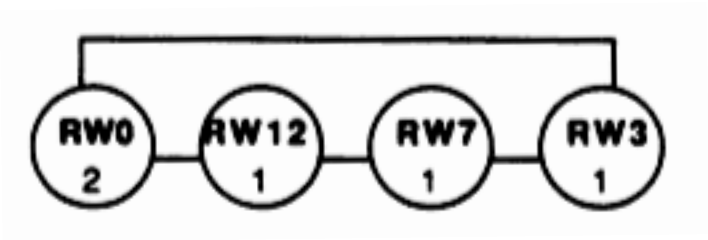
Path planning



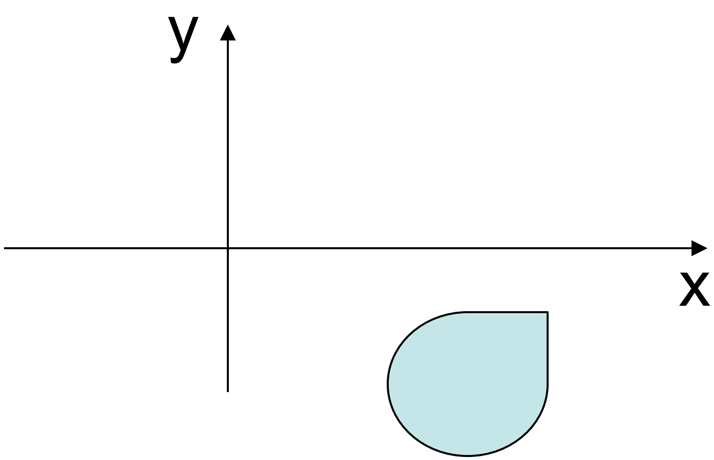


Representation of space

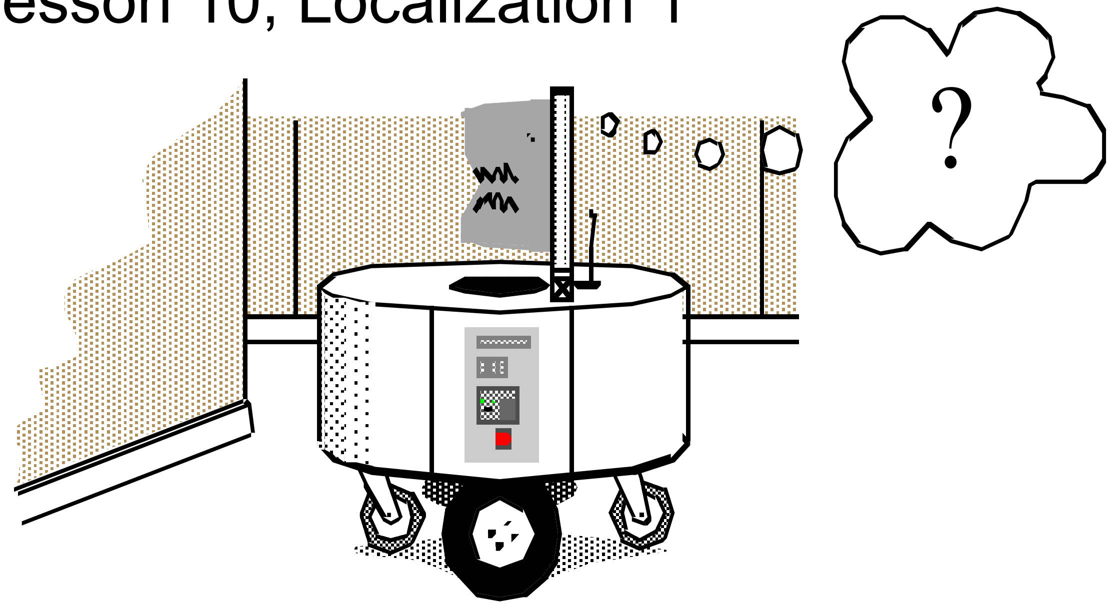
Topological map



Cartesian map



Lesson 10, Localization 1



Where am I? *Localization*

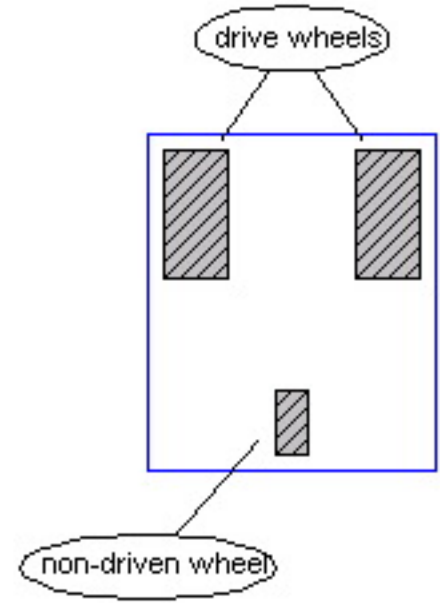
Where have I been? *Map making*

Where am I going? *Mission planning*

What's the best way there? *Path planning* lejos.robotics.navigation

Class DifferentialPilot

java.lang.Object
  └ **lejos.robotics.navigation.DifferentialPilot**



Example of use of come common methods:

```
DifferentialPilot pilot = new DifferentialPilot(2.1f, 4.4f, Motor.A, Motor.C, true);  // parameters in inches
pilot.setRobotSpeed(10);  // inches per second
pilot.travel(12);         // inches
pilot.rotate(-90);        // degree clockwise
pilot.travel(-12,true);
while(pilot.isMoving())Thread.yield();
pilot.rotate(-90);
pilot.rotateTo(270);
pilot.steer(-50,180,true); // turn 180 degrees to the right
while(pilot.isMoving())Thread.yield();
pilot.steer(100);          // turns with left wheel stationary
Delay.msDelay(1000;
pilot.stop();
```

lejos.robotics.localization

Class OdometryPoseProvider

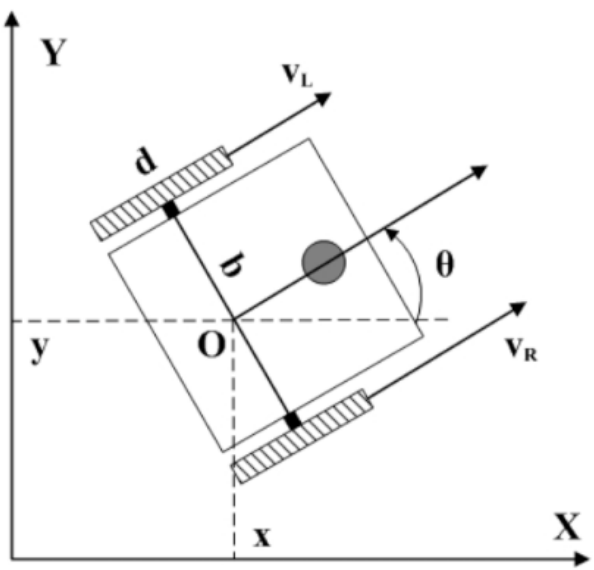
java.lang.Object
  └ **lejos.robotics.localization.OdometryPoseProvider**

All Implemented Interfaces:
PoseProvider, MoveListener

```
public class OdometryPoseProvider
extends Object
implements PoseProvider, MoveListener
```

A PoseProvider keeps track of the robot `Pose`. It does this using odometry (dead reckoning) data contained in a `Move`, which is supplied by a `MoveProvider`. When the PoseProivder is constructed, it registers as listener with its MoveProvider,

```
public class OdometryPoseProvider implements PoseProvider, MoveListener
{

  private float x = 0, y = 0, heading = 0;
```



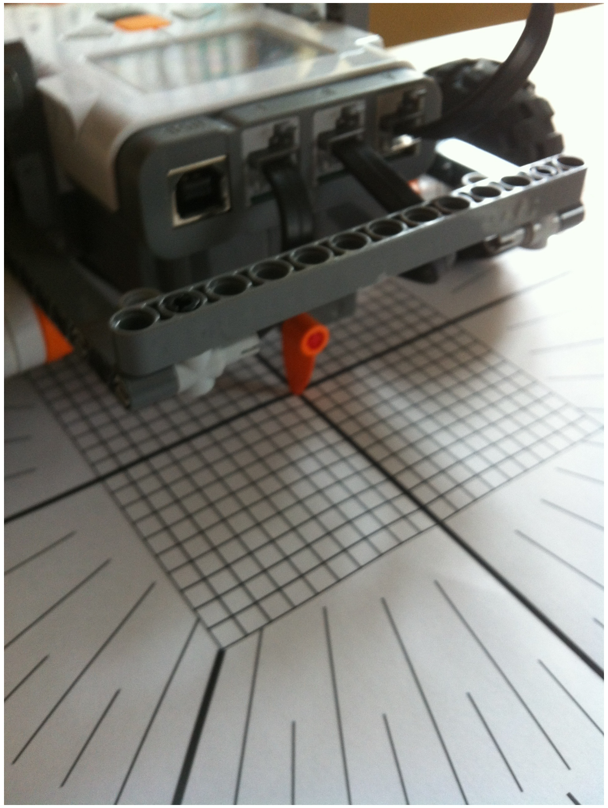
```java
public class OdometryPoseProvider implements PoseProvider, MoveListener
{

  private float x = 0, y = 0, heading = 0;



 /**
  * called by a MoveProvider when movement starts
  * @param move - the event that just started
  * @param mp the MoveProvider that called this method
  */
 public void moveStarted(Move move, MoveProvider mp)
 {
   angle0 = 0;
   distance0 = 0;
   current = false;
   this.mp = mp;
 }

/**
 * called by a MoveProvider when movement ends
 * @param move - the event that just started
 * @param mp
 */
public void moveStopped(Move move, MoveProvider mp)
{
  updatePose(move);
}
```

PilotSquare

```
double wheelDiameter = 5.5, trackWidth = 16.0;
double travelSpeed = 5, rotateSpeed = 45;
NXTRegulatedMotor left = Motor.B;
NXTRegulatedMotor right = Motor.C;

DifferentialPilot pilot = new DifferentialPilot(wheelDiameter, trackWidth, left, right);
OdometryPoseProvider poseProvider = new OdometryPoseProvider(pilot);
Pose initialPose = new Pose(0,0,0);
RConsole.open();
pilot.setTravelSpeed(travelSpeed);
pilot.setRotateSpeed(rotateSpeed);
poseProvider.setPose(initialPose);

LCD.clear();
LCD.drawString("Pilot square", 0, 0);
Button.waitForAnyPress();

for(int i = 0; i < 4; i++)
{
    pilot.travel(20);
    show(poseProvider.getPose());
    Delay.msDelay(1000);

    pilot.rotate(90);
    show(poseProvider.getPose());
    Delay.msDelay(1000);
}
```

PilotSquare

```
double wheelDiameter = 5.5, trackWidth = 16.0;
double travelSpeed = 5, rotateSpeed = 45;
NXTRegulatedMotor left = Motor.B;
NXTRegulatedMotor right = Motor.C;

DifferentialPilot pilot = new DifferentialPilot(wheelDiameter, trackWidth, left, right);
OdometryPoseProvider poseProvider = new OdometryPoseProvider(pilot);
Pose initialPose = new Pose(0,0,0);
RConsole.open();
pilot.setTravelSpeed(travelSpeed);
pilot.setRotateSpeed(rotateSpeed);
poseProvider.setPose(initialPose);

LCD.clear();
LCD.drawString("Pilot square", 0, 0);
Button.waitForAnyPress();

for(int i = 0; i < 4; i++)
{
    pilot.travel(20);
    show(poseProvider.getPose());
    Delay.msDelay(1000);

    pilot.rotate(90);
    show(poseProvider.getPose());
    Delay.msDelay(1000);
}
```

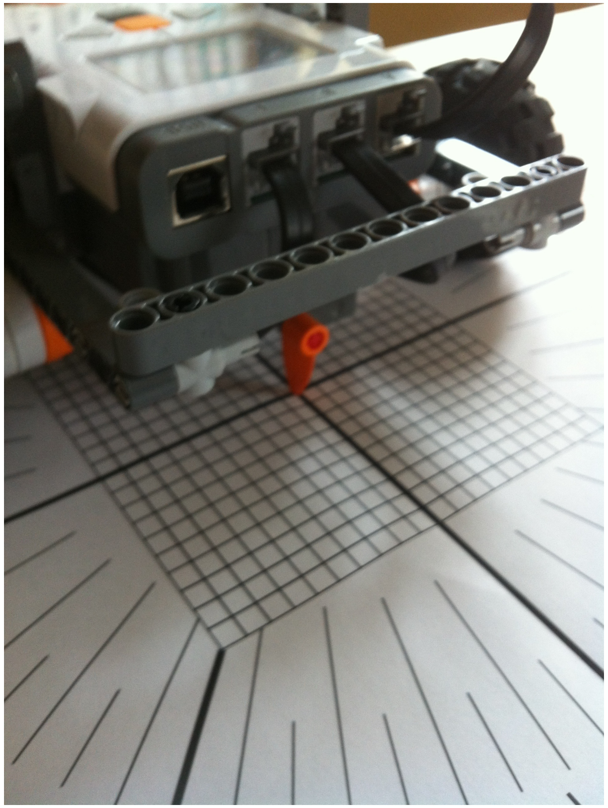


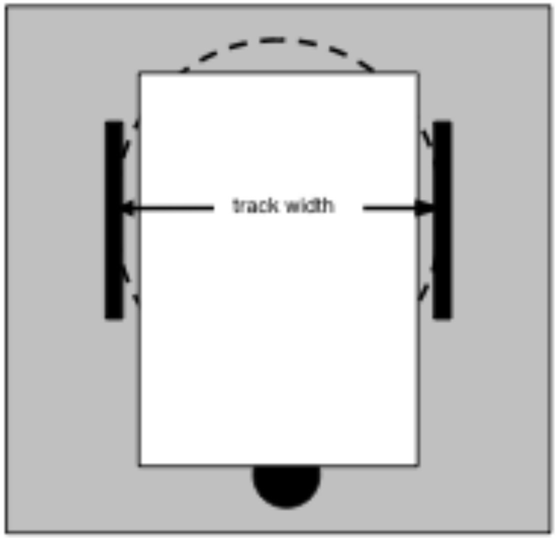
PilotSquare

Systematic odometry errors

Non-systematic errors

```
double wheelDiameter = 5.5, trackWidth = 16.0;
double travelSpeed = 5, rotateSpeed = 45;
NXTRegulatedMotor left = Motor.B;
NXTRegulatedMotor right = Motor.C;

DifferentialPilot pilot = new DifferentialPilot(wheelDiameter, trackWidth, left, right);
OdometryPoseProvider poseProvider = new OdometryPoseProvider(pilot);
Pose initialPose = new Pose(0,0,0);
RConsole.open();
pilot.setTravelSpeed(travelSpeed);
pilot.setRotateSpeed(rotateSpeed);
poseProvider.setPose(initialPose);

LCD.clear();
LCD.drawString("Pilot square", 0, 0);
Button.waitForAnyPress();

for(int i = 0; i < 4; i++)
{
    pilot.travel(20);
    show(poseProvider.getPose());
    Delay.msDelay(1000);

    pilot.rotate(90);
    show(poseProvider.getPose());
    Delay.msDelay(1000);
}
```





Move in a straight line

travel(20)

Rotate on-the-spot

rotate(90)

$$c_m = \pi D_n / n C_e \tag{1.2}$$

where

c_m = conversion factor that translates encoder pulses into linear wheel displacement
D_n = nominal wheel diameter (in mm)
C_e = encoder resolution (in pulses per revolution)
n = gear ratio of the reduction gear between the motor (where the encoder is attached) and the drive wheel.

We can compute the incremental travel distance for the left and right wheel, $\Delta U_{L,i}$ and $\Delta U_{R,i}$, according to

$$\Delta U_{L/R,\, i} = c_m N_{L/R,\, i} \tag{1.3}$$

For completeness, we rewrite the well-known equations for odometry below (also, see [Klarer, 1988; Crowley and Reignier, 1992]). Suppose that at sampling interval I the left and right wheel encoders show a pulse increment of N_L and N_R, respectively. Suppose further that

$$c_m = \pi D_n / nC_e \tag{1.2}$$

where

c_m = conversion factor that translates encoder pulses into linear wheel displacement
D_n = nominal wheel diameter (in mm)
C_e = encoder resolution (in pulses per revolution)
n = gear ratio of the reduction gear between the motor (where the encoder is attached) and the drive wheel.

We can compute the incremental travel distance for the left and right wheel, $\Delta U_{L,i}$ and $\Delta U_{R,i}$, according to

$$\Delta U_{L/R,\,i} = c_m N_{L/R,\,i} \tag{1.3}$$

and the incremental linear displacement of the robot's centerpoint C, denoted ΔU_i, according to

$$\Delta U_i = (\Delta U_R + \Delta U_L)/2. \tag{1.4}$$

Next, we compute the robot's incremental change of orientation

$$\Delta\theta_i = (\Delta U_R - \Delta U_L)/b \tag{1.5}$$

where b is the wheelbase of the vehicle, ideally measured as the distance between the two contact points between the wheels and the floor.

For completeness, we rewrite the well-known equations for odometry below (also, see [Klarer, 1988; Crowley and Reignier, 1992]). Suppose that at sampling interval I the left and right wheel encoders show a pulse increment of N_L and N_R, respectively. Suppose further that

$$c_m = \pi D_n / n C_e \tag{1.2}$$

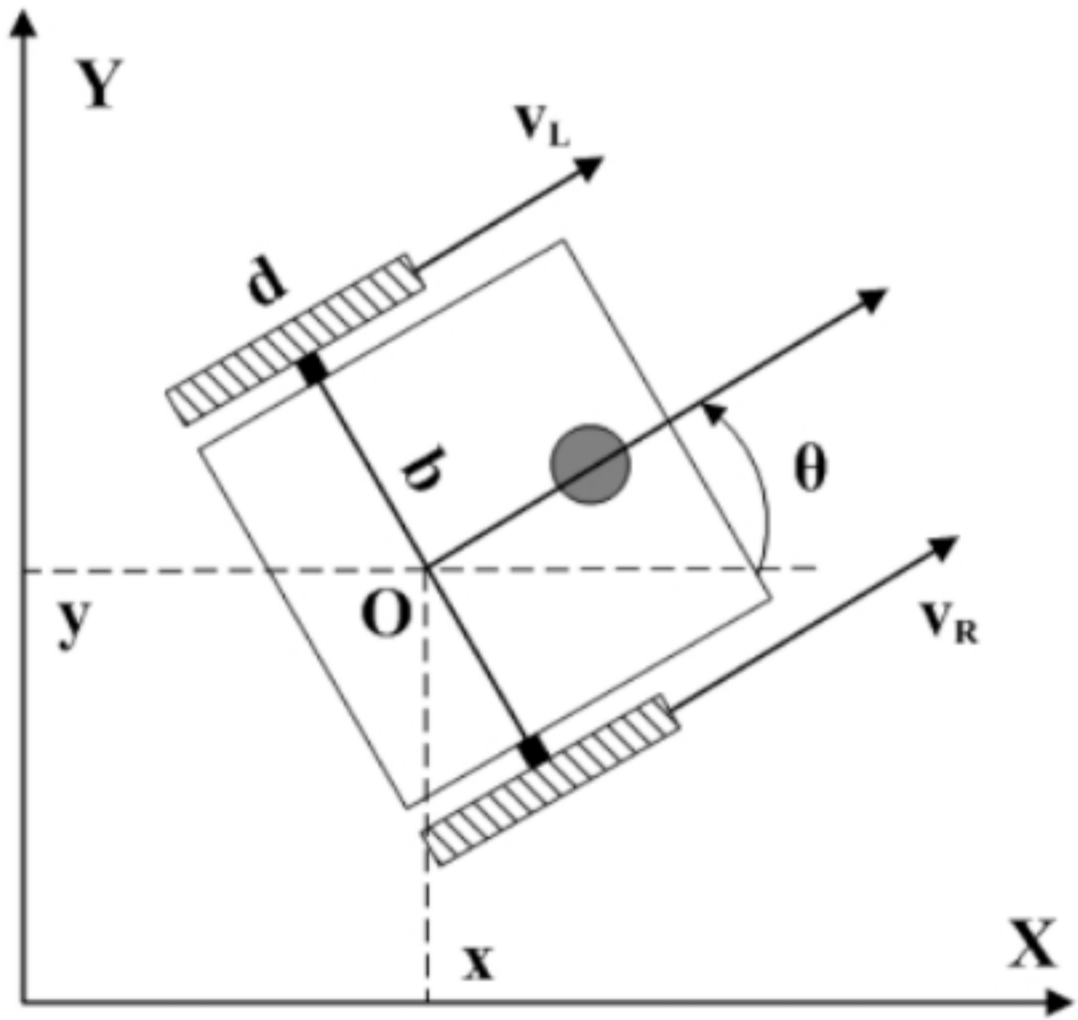
where

c_m = conversion factor that translate

D_n = nominal wheel diameter (in mm

C_e = encoder resolution (in pulses pe

n = gear ratio of the reduction gear be drive wheel.



We can compute the incremental tra according to

$$\Delta U_{L/R,\, i} = c_m N_{L/R,\, i}$$

and the incremental linear displacement

$$\Delta U_i = (\Delta U_R + \Delta U_L)/2.$$

Next, we compute the robot's increment

$$\Delta\theta_i = (\Delta U_R - \Delta U_L)/b$$

where b is the wheelbase of the vehicle, points between the wheels and the floor.

Calibrate the wheel diameter and the track width

- Start with `wheelDiameter`. Make the vehicle travel e.g. 50 cm and adjust the wheel diameter until the vehicle travels as close to 50 cm as possible. If the vehicle will not run in a straight line on a smooth surface, there may be a small difference in the diameters of the two wheels. Then the constructor with different diameters for the left and right wheels should be used.
- After having adjusted the wheel diameters then make the vehicle rotate a given angle e.g. 180 degrees and adjust the `trackWidth` until the vehicle rotates as close to the given angle as possible.

$$\Delta U_{L/R,\,i} = c_m N_{L/R,\,i} \tag{1.3}$$

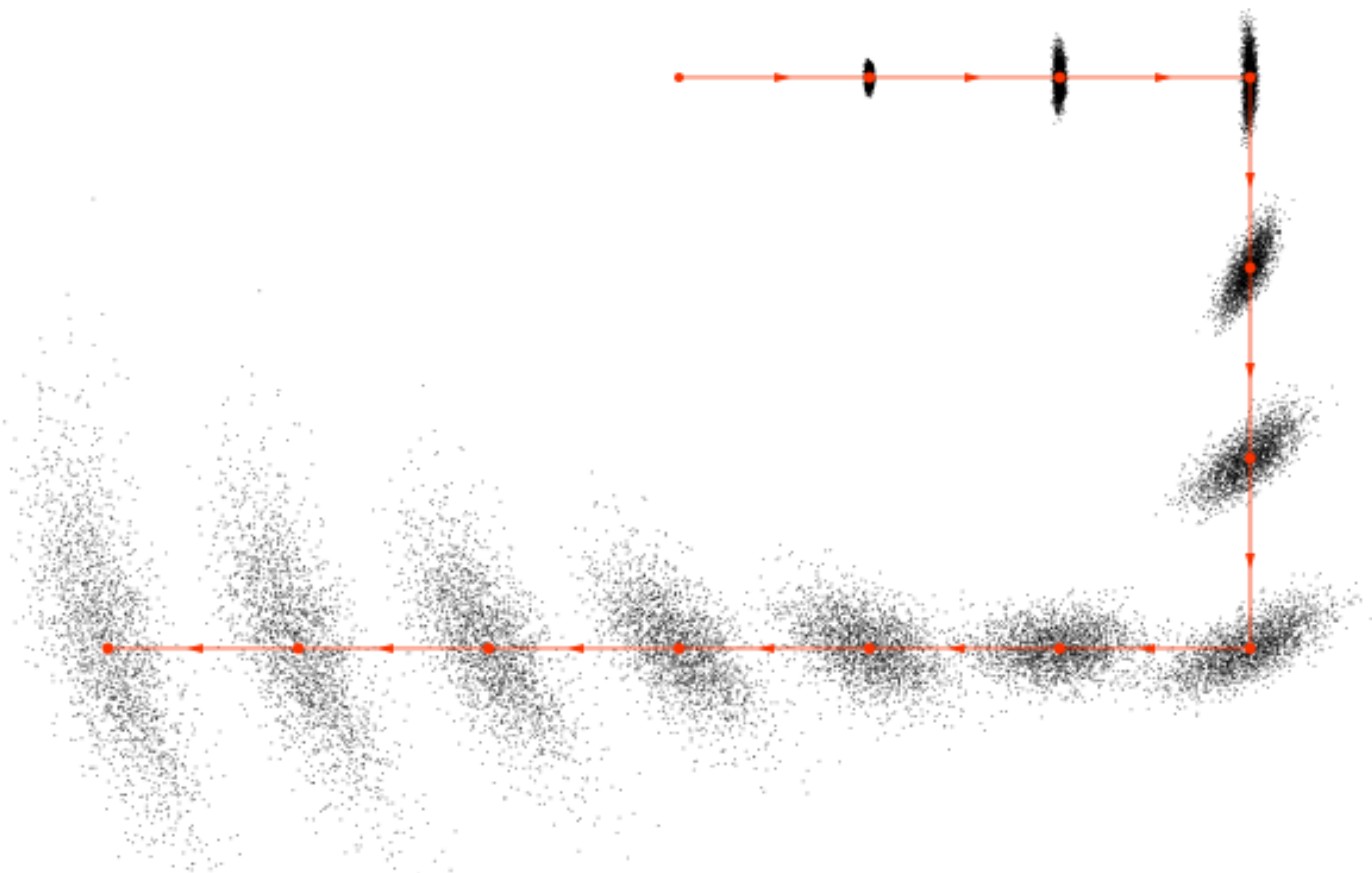
and the incremental linear displacement of the robot's centerpoint C, denoted ΔU_i , according to

$$\Delta U_i = (\Delta U_R + \Delta U_L)/2. \tag{1.4}$$

Next, we compute the robot's incremental change of orientation

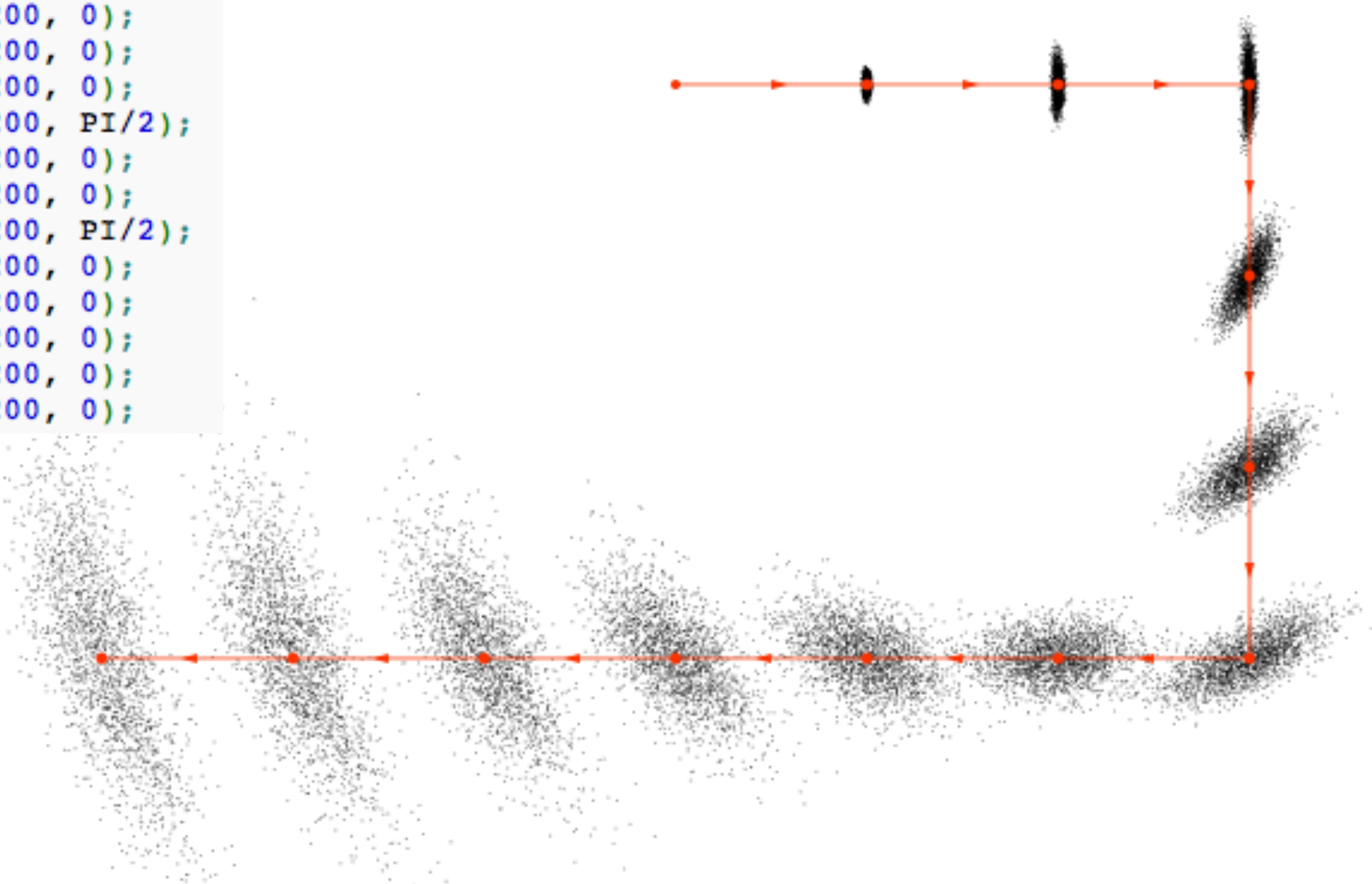
$$\Delta\theta_i = (\Delta U_R - \Delta U_L)/b \tag{1.5}$$

Position tracking by means of particle filters



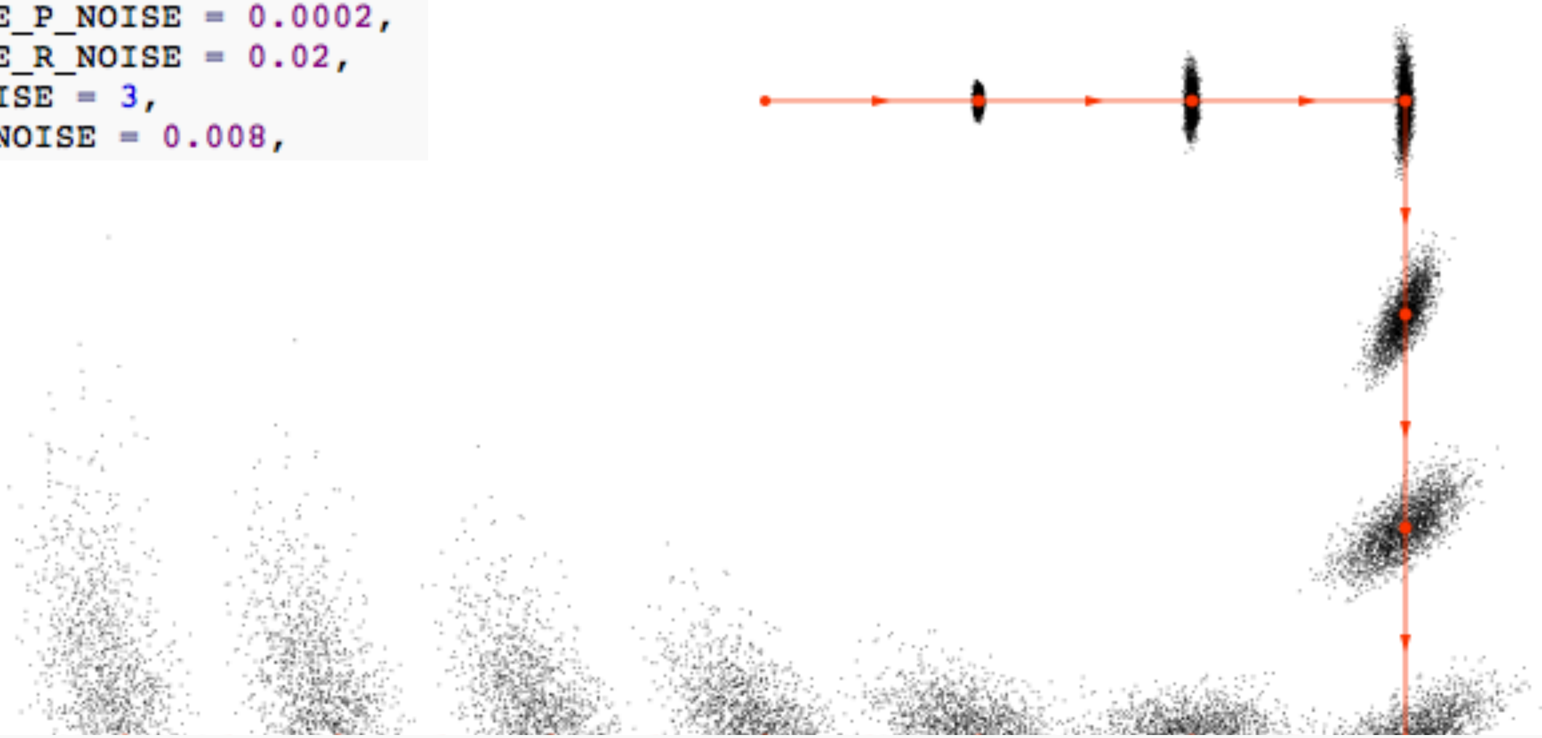
Position tracking by means of particle filters

```
move(200, 0);
move(200, 0);
move(200, 0);
move(200, PI/2);
move(200, 0);
move(200, 0);
move(200, PI/2);
move(200, 0);
move(200, 0);
move(200, 0);
move(200, 0);
move(200, 0);
```



Position tracking by means of particle filters

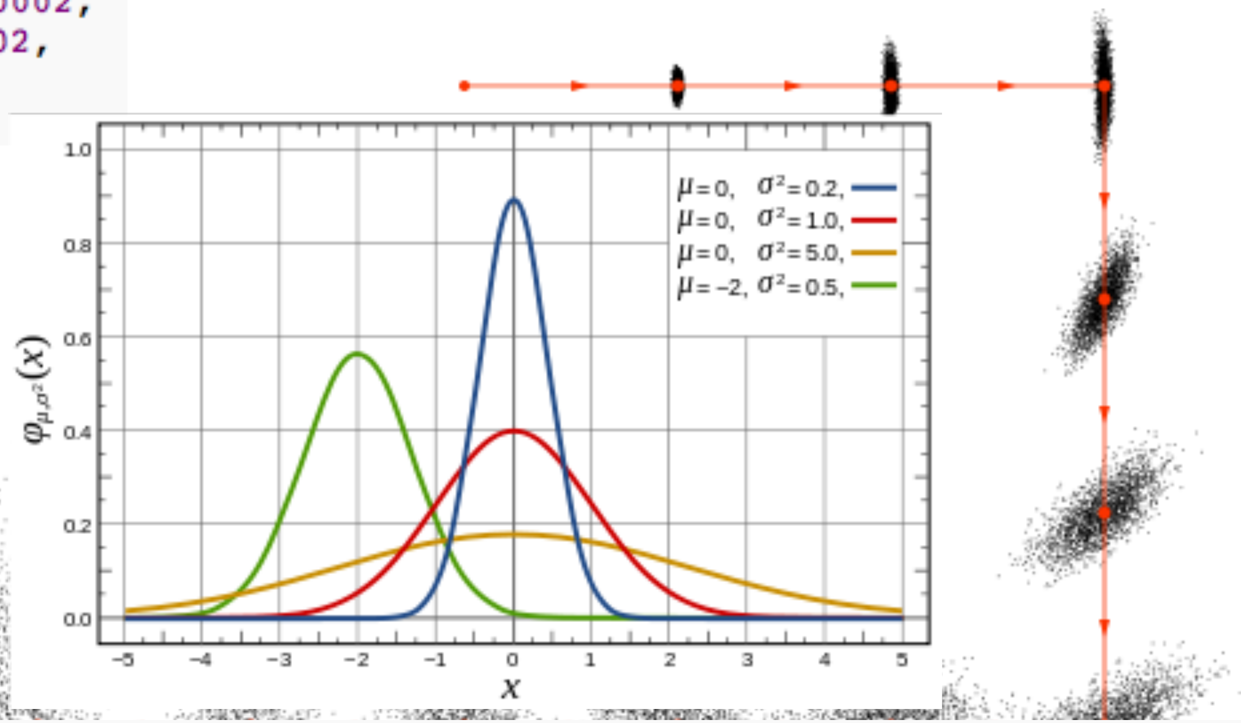
```
ANGLE_NOISE = 0.04,
ANGLE_P_NOISE = 0.0002,
ANGLE_R_NOISE = 0.02,
R_NOISE = 3,
R_P_NOISE = 0.008,
```



```
void move(long double r, long double t) {
    qwer.theta += t;
    qwer.x += r*cos(qwer.theta);
    qwer.y += r*sin(qwer.theta);
    for(int i=0; i<M; i++) {
        z[i].theta += t + r*ANGLE_P_NOISE*noise() + ANGLE_NOISE*noise() + t*ANGLE_R_NOISE*noise();
        z[i].x += r*cos(z[i].theta) + R_NOISE*noise() + r*R_P_NOISE*noise();
        z[i].y += r*sin(z[i].theta) + R_NOISE*noise() + r*R_P_NOISE*noise();
```

Position tracking by means of particle filters

```
ANGLE_NOISE = 0.04,
ANGLE_P_NOISE = 0.0002,
ANGLE_R_NOISE = 0.02,
R_NOISE = 3,
R_P_NOISE = 0.008,
```




```
void move(long double r, long double t) {
    qwer.theta += t;
    qwer.x += r*cos(qwer.theta);
    qwer.y += r*sin(qwer.theta);
    for(int i=0; i<M; i++) {
        z[i].theta += t + r*ANGLE_P_NOISE*noise() + ANGLE_NOISE*noise() + t*ANGLE_R_NOISE*noise();
        z[i].x += r*cos(z[i].theta) + R_NOISE*noise() + r*R_P_NOISE*noise();
        z[i].y += r*sin(z[i].theta) + R_NOISE*noise() + r*R_P_NOISE*noise();
```

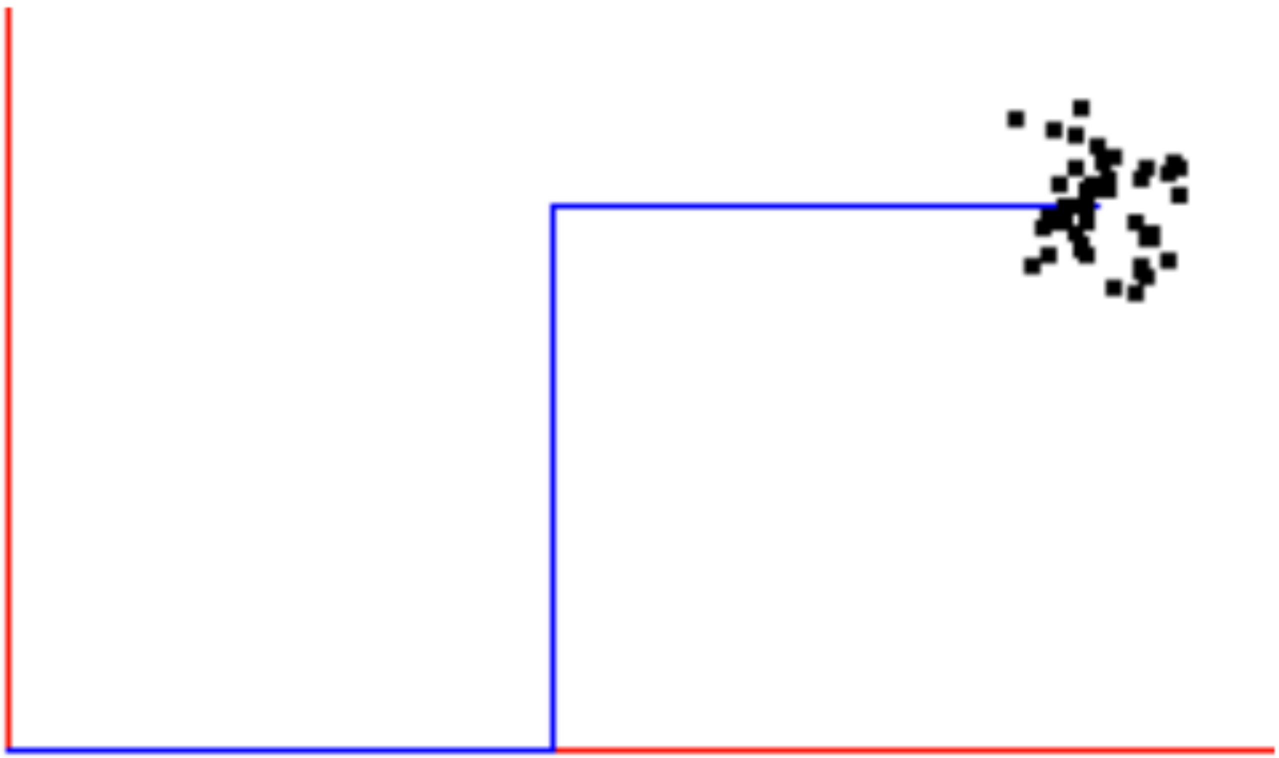
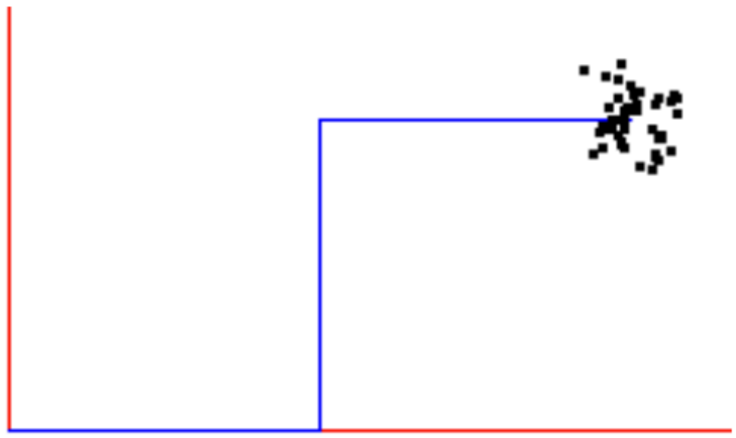
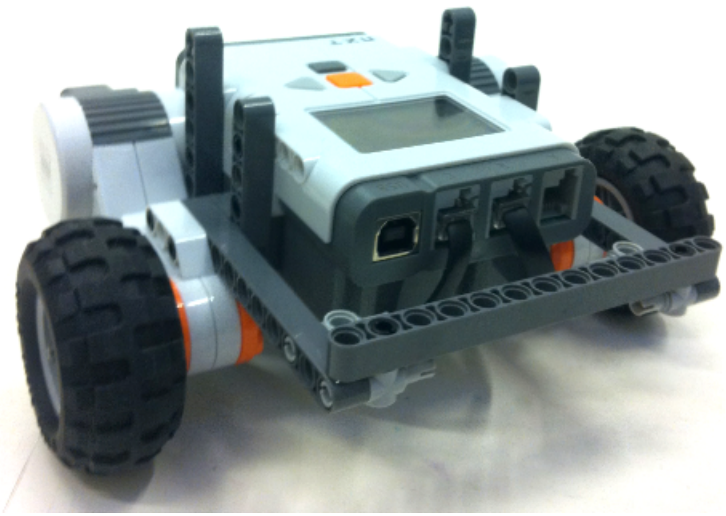


Figure 5 The distribution of likely positions for a non-sensing vehicle after the route travel(100), rotate(90), travel(100), rotate(-90) and travel(100).

Position tracking by means of particle filters

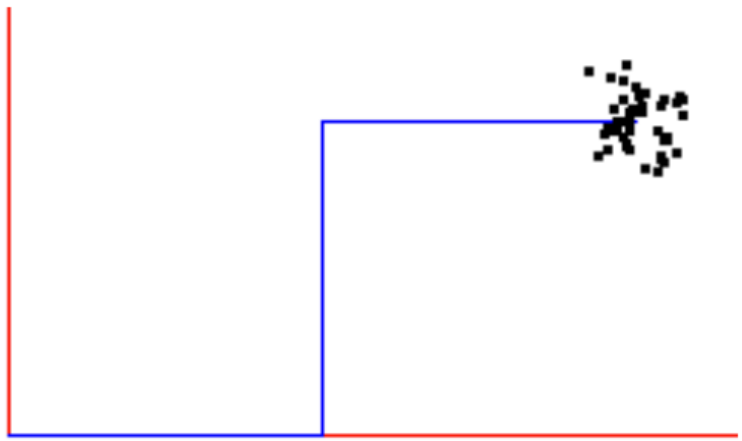


PilotMonitor



PilotRoute

Position tracking by means of particle filters



```
/**
  * Apply the robot's move to the particle with a bit of random noise.
  * Only works for rotate or travel movements.
  *
  * @param move the robot's move
  */
 public void applyMove(Move move, float distanceNoiseFactor, float angleNoiseFactor)
 {

   float ym = (move.getDistanceTraveled() * ((float) Math.sin(Math.toRadians(pose.getHeading()))));
   float xm = (move.getDistanceTraveled() * ((float) Math.cos(Math.toRadians(pose.getHeading()))));


   pose.setLocation(new Point(
                  (float) (pose.getX() + xm + (distanceNoiseFactor * xm * rand.nextGaussian())),
                  (float) (pose.getY() + ym + (distanceNoiseFactor * ym * rand.nextGaussian()))));
   pose.setHeading(
      (float) (pose.getHeading() + move.getAngleTurned() + (angleNoiseFactor  * rand.nextGaussian())));
   pose.setHeading((float) ((int) (pose.getHeading() + 0.5f) % 360));

 }
```

Monte Carlo localization