University of Edinburgh - School of Informatics

IAR - Task 1 Report

Jakob Calero - s0948339
Samuel Neugber - s0821562

October 4, 2012

1 Abstract

In a relatively static environment it may be possible to find all states a robot can be in and reduce them to conditions in the control code. This approach worked fine for our simple robot to do its simple task; it followed walls and avoided obstacles using a subsumption architecture based on the conditions we identified. A different approach we attempted to use was to directly inhibit the motor speeds using the information from the infra-red sensors. This resulted in smoother, but less predictable movement. Since the robot was more difficult to control using this method, we concentrated mainly on the conditional control to achieve the task.

2 Introduction

We approached the problem of navigating a small robot in a known environment using two different methods: Discrete, functional behaviour and more continuous, differential control, as described in the first few chapters of Vehicles[1, p.6-9]. The Khepera II robots are circular, two-wheeled robots, which come equipped with 8 infra-red sensors arranged as seen in *figure 2.1*.

The goal was to move the robot around in a static and enclosed testing arena (*figure 2.2*) while following walls and avoiding obstacles in unknown positions. The functional behaviour followed a simple subsumption control architecture, directly prioritising actions depending on conditional cases. The differential control on the other hand set the wheel speeds as a function of the values returned by the IR sensors. Both approaches were worked on simultaneusly and reviewed over time until one approach was chosen due to performing the task best.

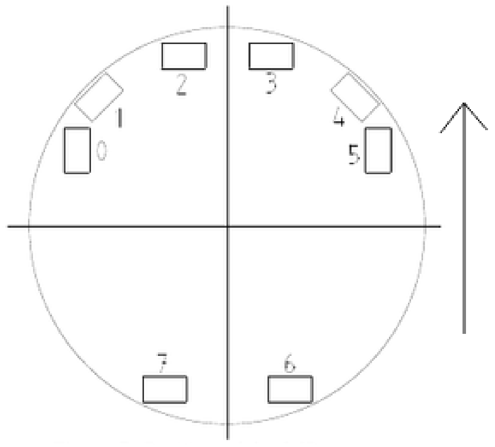

Figure 2.1: Diagram of IR Sensors on Khepera robot

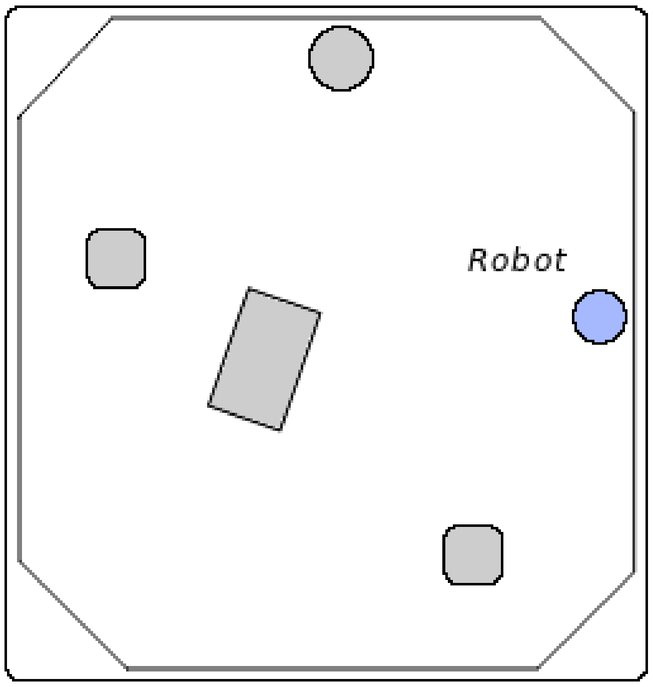

Figure 2.2: Map of driving area with example obstacle placements

3 Methods

3.1 Gathering Data

The IR sensors on the Khepera have a range of values between 0 and 1020. Experimentation showed that when nothing is close to the robot the sensor values drift at around 50. Sensor values of 200+ describe a sensible value representing the robot being near an object (within 2cm) while values of 500+ mean that we are too close to an object. A range between 180 and 250 where the values we chose as a good distance when trying to follow a wall without getting too close or too far away from it. From these values we then extrapolated if necessary.

3.2 Functional Control

As seen in *figure 3.1* our control code for this methodology uses just four if-statements.

```
if FrontSensor is close to object OR RightSensors are close to object then
    Command: Rotate left
end if
if FrontSensor is close to object OR LeftSensors are close to object then
    Command: Rotate right
end if
if LeftSensor is closer than 250 but further away than 180 from object then
    Command: Rotate left
end if
if RightSensor is closer than 250 but further away than 180 from object then
    Command: Rotate right
end if
Command: Move forward
```

Figure 3.1: Functional Control Algorithm

Given the order each condition is checked the robot will always turn whenever there is something directly in front of it, as illustrated by the first two conditions above. In the same two conditions the right and left sensors are also checked but only for a very close distance, triggering only when collision is imminent and not while travelling along a wall. These two simple cases result in the behaviour of avoiding object collision and the tendency of following straight objects (walls) given that as soon as the front sensor is not detecting anything, forward movement is resumed.

The final two conditions attempt to capture the fact that the robot may turn away from a wall too much when avoiding collision or drift slightly.

The robot should rotate back towards the wall if it has moved too far away. A cap to that distance is used as we do not want the robot to start rotating when it's *very* far away from a wall as that might indicate it's not following a wall to begin with.

3.3 Differential Control

The second approach we took to solve the task ahead was one described in the first few chapters of Vehicles[1, p.6-9]. Our goal was that, ideally, the robot should always be moving and do so as smoothly as possible to maximise efficiency and avoid "stuttering" movement. The control code in *Appendix 6.2* shows that in order to do so, each iteration of the control loop resets the wheel speed to a maximum value and then reduces the speed of the wheel on the opposite side of the corresponding IR sensor if certain thresholds are met. The higher the value of the sensor, the more the speed of the corresponding wheel is reduced.

4 Results

4.1 Functional Control

Using functional control, the behaviour of the robot was close to being deterministic, reliably following walls with 1-2cm distance. It managed to avoid obstacles to avoid collision and return to the wall searching pattern immediately after *(figure 4.1)*.

A few problems still remain however. One case was that it got stuck in narrow passages, such that the first two rules in the control loop were repeatedly activated in succession. An additional case where the robot would not explore the entire map if objects were placed in a square-like pattern, resulting in the robot moving in a loop between the objects without being able to get out *(figure 4.2)*.

4.2 Differential Control

Differential control did not work as intended in most cases, so we did not pursue it for long. The robot did turn smoother than it had using functional control, but it was also harder to control. The robot usually turned away too much when avoiding collision, which meant that it rarely followed a wall. This method did however have the benefit that the robot did not get stuck in between even the closest gaps, but nicely navigated through.

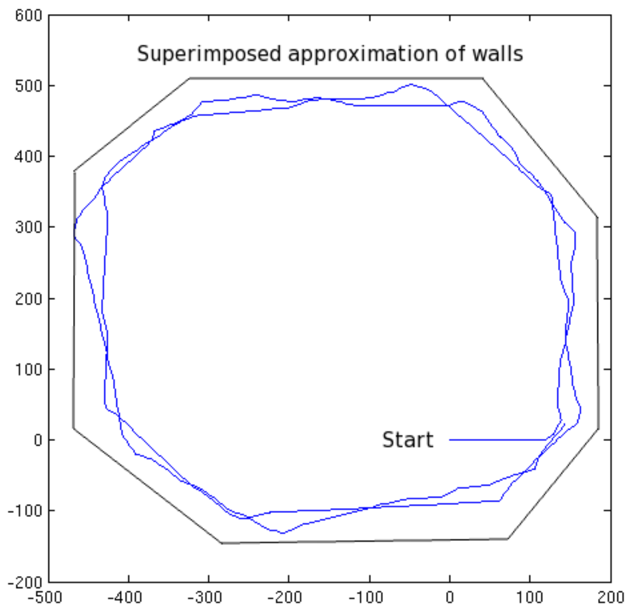

Figure 4.1: Tracking of robot movement based on odometry data during functional control. No obstacles along the walls.

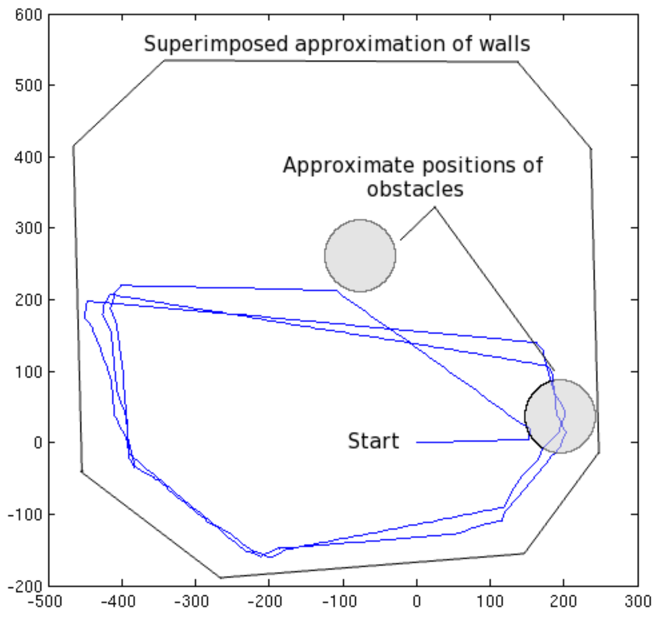

Figure 4.2: Illustrating the "*loop*" problem mentioned above.

5 Discussion

In an environment as small and predictable as the one given in this task, good rule based behaviour is possible since most conditions the robot will be in can be extracted and mapped accordingly and good results follow therein, as shown above. However, the more complex the task and environment get, the harder it will be to achieve such a design. In these cases, differential control, (or even better, full proportional-integral-derivative control) and more dynamic basic behaviour will be useful. Conditional if-cases will run into problems like the *indecisiveness* of our above problem when navigating narrow corridors and are generally very messy to maintain.

References

[1] Valentino Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, A Bradford Book, 1986.

6 Appendix

6.1 Functional Controller - Source

```
function positions = controlMain(s, speed)
  turnspeed = round(speed/3);

  setCounts(s,0,0);
  oldWheelCountsL = 0;
  oldWheelCountsR = 0;
  angle = 0;
  xPos = 0;
  yPos = 0;

  positions = [xPos,yPos];

  while (true)
    pause(0.05);

    % read sensor values
    ir = readIR(s)
    % rename individual sensors
    leftIR1 = ir(1);
    leftIR2 = ir(2);
    midLeftIR = ir(3);
    midRightIR = ir(4);
    rightIR2 = ir(5);
    rightIR1 = ir(6);
    % update Odometry
    wheelCounts = readCounts(s);
    wheelDeltaL = wheelCounts(1) - oldWheelCountsL;
    wheelDeltaR = wheelCounts(2) - oldWheelCountsR;
    oldWheelCountsL = wheelCounts(1);
    oldWheelCountsR = wheelCounts(2);

    angle = angle - 0.5*(wheelDeltaL - wheelDeltaR)/330;

    if(angle > 2*pi) angle = angle - 2*pi; end
    if(angle < 0) angle = angle + 2*pi; end
    xPos = xPos + 0.5*(wheelDeltaL + wheelDeltaR)
     * 0.08 * cos(angle);
    yPos = yPos + 0.5*(wheelDeltaL + wheelDeltaR)
     * 0.08 * sin(angle);

    positions = cat(1,positions,[xPos,yPos]);

    % Adjust Left based on centre sensor
    if ((midRightIR > 150 || rightIR1 > 450 || rightIR2 > 600))
      stop(s)
      turn(s,-turnspeed,turnspeed)
      continue
    end

    % Adjust Left based on centre sensor
    if ((midLeftIR > 150 || leftIR1 > 450 || leftIR2 > 600))
      stop(s)
      turn(s,turnspeed,-turnspeed)
      continue
    end

    % Wall re allignment
    if (leftIR1 < 250 && leftIR1 > 180)
      stop(s)
      turn(s,-turnspeed,turnspeed)
      continue
    end

    if (rightIR1 < 250 && rightIR1 > 180)
      stop(s)
      turn(s,turnspeed,-turnspeed)
      continue
    end

    go(s,speed);
  end
end
```

6.2 Differential Controller - Source

```
while (true)
```

```
    irSens = readIR(s);
    irR1 = irSens(6);
    irR2 = irSens(5);
    irL1 = irSens(1);
    irL2 = irSens(2);
    irC1 = irSens(3);
    irC2 = irSens(4);

    lSpeed = 6;
    rSpeed = 6;

    if(irR1 > 200)
      lSpeed = lSpeed - round(irR1/100);
    end

    if(irL1 > 200)
      rSpeed = rSpeed - round(irL1/100);
    end

    if(irC1 > 200 || irC2 > 200)
      lSpeed = lSpeed - ((irC1 - irC2)/(abs(irC1 - irC2)))
        * round(irC1/100)
      rSpeed = rSpeed - ((irC1 - irC2)/(abs(irC1 - irC2)))
        * round(irC2/100)
    end

    if(lSpeed > 6) lSpeed = 6; end
    if(rSpeed > 6) rSpeed = 6; end

    setSpeeds(s,lSpeed,rSpeed);
    pause(0.05);
end
stop(s);
```