University of Edinburgh - School of Informatics

IAR - Task 2 Report

Jakob Calero - s0948339
Samuel Neugber - s0821562
October 15, 2012

1 Abstract

A dynamically changing task usually requires a dynamic solution. In this task we are asked to follow a dynamically moving light source while avoiding obstacles in an arena. We have therefore revisited the approach of setting motor speeds as a direct function of the sensor values and assuming obstacle detection from the same conditional approach we used in our previous task. The robot will move towards a sufficiently close lightsource in a smooth trajectory while at the same time avoiding obstacles.

2 Introduction

In the last task we explored two different control methods for our robot: one which acted on certain threshold-conditions of the infra-red (IR) sensors, and one which directly mapped the values from the sensors to motor speeds. Our findings were that the first approach (conditional control) works well enough in relatively static environments and is a little easier to reason about than the other approach.

This task, on the other hand, required our robot to exhibit more dynamic behaviour in order to follow a light which could frequently change its position. We therefore went back and changed the control method to a more direct function between sensor values and motor control as in our second method in the last task. Using the photo sensors on the robot we calculate a relative angle it should be facing and that angle is then taken as input to the functions which detemine the speed of the individual wheels. The obstacle detection is handled by a conditional approach based on the IR sensors and is taken as higher priority than approaching the light source.

3 Methods

3.1 Light Direction Detection

Our robot has 8 sensors detecting levels of light at a value range of 0 to 500, which are arranged as seen in (*figure 3.1*). From this information we ultimately want to estimate a direction the light could be originating from by combining the values and orientations of the individual sensors.

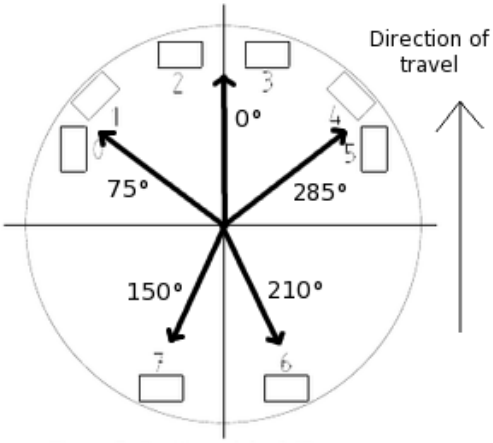

Figure 3.1: Sensors and their virtual angles used to estimate angle of the light source.

As we initially can estimate the angles each sensor is pointing at we can use the sensor values as weights for each of the vectors between the robot and the sensor direction. Summing the vectors would give us a general direction that the strongest light would be coming from (*figure 3.2*). However, given that there are disproportionately many sensors detecting light from the front than from the back this information would be skewed. To offset this we look at the front sensors in pairs, taking the highest value between the two and the angle between them.

Now, this would give us a general direction, but considering we're dealing with visible light we're likely to get a lot of interference from other light sources. We therefore also normalise the weights to get the relative size of the final directional vector. A short directional vector would indicate a more even spread of readings between sensors and thus a higher likelyhood that it's simply ambient light it's reading and not the light source it's supposed to follow.

$$light\_direction = \sum_{i=1}^{\#ofsensors} \mathbf{sensor\_directions_i} * \mathbf{weights_i}$$

Figure 3.2: General formulae for calculating the final direction of the light source.

3.2 Differential Control

The next step our control program performs is to convert the vector we have calculated on the basis of the light sensors into an angle ranging between -180 and 180 degrees. Given the placement of the sensors, a negative angle will indicate a lightsource to the right and vice versa. We then directly inhibit the speed of the wheel on the side we want to turn to, to the point of getting opposing wheel speeds when the absolute angle is greater than 90 degrees.

We additionally use the scale of the vector as an indicator to only move towards a lightsource we are relatively certain about. The robot will only move if it computes a certainty of over 0.05, where certainty is within a range of 0 to 1 and simply taken as the length of the final direction vector as computed in (*figure 3.2*).

Another special case we take into account is when there is a lightsource directly behind the robot. This case is a little problematic since the robot has a blind spot between the side-most front sensors and the back sensors which will cause the light angle to be skewed when in that spot and cause the robot to occasionally loose its trail. To avoid this we add a special case when the light is detected as coming from behind (> 130 or < -130) which turns the robot by a certain amount ($\sim 180°$) skipping the blind spot completely.

3.3 Obstacle Avoidance

In the previous two sections we have established the means to follow a lightsource. In order to avoid obstacles we had originally planned to copy the mechanism described in 3.1 to extract a vector from the IR sensors which points in the opposite direction of the obstacle. Ideally, this vector would have been combined with the vector from our lightsource detection, completely overriding the influence of the light when an the robot comes too close to an obstacle.

In the end however we decided to save some time by simply adding our obstacle avoidance code from the last task, which overrides our differential control in case certain thresholds of the IR sensor values are met.

4 Results

From our experiments we have found that our robot will detect a light source shining at it from roughly 70 cm away, due to the fact that we take into account the scale of the vector we calculate. Additionally, we found that the average error in the angle we measure versus the actual angle of the lightsource towards the robot is Y degrees.

Regardless/because (¡- we don't know yet) of the reliability we achieve from the method we use to find the lightsource, our robot moves towards the lightsource quickly, smoothely and without oscillations in direction of movement. In the presence of obstacles the movement may not be as smooth due to the discrete conditional cases, but it prevents the robot from colliding with an object in almost all cases.

There are just two cases which we have encountered where our control method is not perfect yet. The first case has the robot turning towards the lightsource, only to end up with the light shining on the robot's blind spot. The robot may therefore not detect sufficient light, fall back into the standart behaviour and drive out of the beam's reach. The second case is a problem we have inherited from reusing our obstacle avoidence code. In the case where there are obstacles to the left and to the right of the robot, but not in front, it can loop between the two cases of turning away from the left obstacle, and turning away from the right obstacle.

5 Discussion

6 Appendix

6.1 Direction of Light Detection - Source

```
function [angle,certainty] = getLightAngle(s)
%angles = [170, 135, 95, 85, 45, 10, 300, 240];
angles = [75, 0, 285, 210, 150];

vector1 = [cosd(angles(1)), sind(angles(1))];
vector2 = [cosd(angles(2)), sind(angles(2))];
vector3 = [cosd(angles(3)), sind(angles(3))];
vector4 = [cosd(angles(4)), sind(angles(4))];
vector5 = [cosd(angles(5)), sind(angles(5))];
%vector6 = [cosd(angles(6)), sind(angles(6))];
%vector7 = [cosd(angles(7)), sind(angles(7))];
%vector8 = [cosd(angles(8)), sind(angles(8))];

%vectors = [vector1; vector2; vector3; vector4; vector5; vector6; vector7; vector8
vectors = [vector1; vector2; vector3; vector4; vector5];
```

```
sensorVals = readAmbient(s);

weights = 1 ./ sensorVals;

weights = weights .* 1/sum(weights);

%for i=1:size(vectors)
%    vectors(i,:) = vectors(i,:) .* weights(i);
%end
vectors(1,:) = vectors(1,:) .* max(weights(1),weights(2));
vectors(2,:) = vectors(2,:) .* max(weights(3),weights(4));
vectors(3,:) = vectors(3,:) .* max(weights(5),weights(6));
vectors(4,:) = vectors(4,:) .* weights(7);
vectors(5,:) = vectors(5,:) .* weights(8);

direction = sum(vectors,1);
certainty = norm(direction);

costheta = dot(direction, [1,0])/(norm(direction)*norm([1,0]));
angle = rad2deg(acos(costheta));

if (direction(2) < 0)
    angle = -angle;
end
```

6.2 Controller - Source

```
function positions = control2Alternate(s,speed)
turnspeed = round(speed/2);

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

positions = [xPos,yPos];

iteration = 0;
pauseTime = 0.05;

oldLSpeed = speed;
oldRSpeed = speed;

while (iteration < 1000)
    pause(pauseTime);
    pauseTime = 0.05;

    iteration = iteration +1;

    % 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
% odo 1: Get wheel measurements
wheelCounts = readCounts(s);
wheelDeltaL = wheelCounts(1) - oldWheelCountsL;
wheelDeltaR = wheelCounts(2) - oldWheelCountsR;
oldWheelCountsL = wheelCounts(1);
oldWheelCountsR = wheelCounts(2);

% odo 2: calculate angle
%           +      =    - 0.5*(vleft - vright)/(widthfactor)
% 2 x 40mm radius * 0.08mm per wheelcount != 'width factor' of 330, which I ju
angle = angle - 0.5*(wheelDeltaL - wheelDeltaR)/330;

if(angle > 2*pi) angle = angle - 2*pi; end
if(angle < 0) angle = angle + 2*pi; end


% odo 3: calculate position
% x      x +   x =x + 0.5*(vleft + vright) cos(  )
% y      y +   y =y + 0.5*(vleft + vright) sin(  )
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]);

if((rightIR1 > 250 || rightIR2 > 400) && (leftIR1 > 250 || leftIR2 > 400) && (
    setSpeeds(s,speed,speed);
    continue;
end

% Adjust Left based on centre sensor
if (midRightIR > 150 || rightIR1 > 250 || rightIR2 > 400)
    stop(s)
    turn(s,-turnspeed,turnspeed)
    continue
end

% Adjust Left based on centre sensor
if ((midLeftIR > 150 || leftIR1 > 250 || leftIR2 > 400))
    stop(s)
    turn(s,turnspeed,-turnspeed)
    continue
end
```

```
    [a,c] = getLightAngle(s);

    if(c < 0.05)
        setSpeeds(s,speed,speed);
        continue;
    end

    if(a > 0)
        rSpeed = speed;
        lSpeed = max(round(speed - (12 * abs(a) / 180) -1),-speed);
    else
        lSpeed = speed;
        rSpeed = max(round(speed - (12 * abs(a) / 180) -1),-speed);
    end

    if(oldLSpeed ~= lSpeed || oldRSpeed ~= rSpeed)
        setSpeeds(s,lSpeed,rSpeed);
        oldLSpeed = lSpeed;
        oldRSpeed = rSpeed;
    end
    if((a > 130 || a < -130) && c > 0.1)
        pauseTime = 1.5;
    end
end
stop(s);

%%%%%%%%%%%%%%%

%% CLEANUP %%%
%closeConnection(s);

%%%%%%%%%%%%%%%

end
```

References

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