IAR Reort
Task 1

Babak Esmaeili, David Fullerton

October 7, 2015

Abstract

We have written a program for a Khepera robot that allows the robot to following along a wall while avoiding colliding with obstacles. When the robot is first released into the arena it will drive until it detects a surface, if the obstacle is detected with the front two IR sensors the robot will turn counter-clockwise on the spot until its right side if facing the wall. However if the robot first encounters an object on its left side, it will curve away to the right. Once the robot detects an object using sensor 6, the sensor on its right side, the robot follows the surface of the object attempting to keep within a set distance away from it, we use PID controls for this. If the robot is placed in an empty arena this will result in the robot driving until it reaches the wall and then following the wall in a counter-clockwise direction. Some limitations with our design are that the robot only uses one sensor to measure the distance from the wall when using PID control also if the robot meets a small object before a wall it will follow the edge of that object.

1 Introduction

The task was to design a controller for a Khepera robot that would navigate an arena while avoiding collisions with obstacles. If possible the robot would tend to follow long walls and avoid following the edge of smaller obstacles. We approached this by deciding the robot would favour keeping an object a set distance away from its right hand side. As for how the robot would decide if what it sensed was a wall or small object we were unsure how to approach this. One idea was that when a sensor received a reading above a threshold we would also check the adjacent sensors as to what their values were. If they were also above a threshold then it would suggest a wall, if not then a small object. We didnt use this as it would only work for very small objects and for the front middle four sensors. We were late to start this project and so referred to a previous course, IVR, and course work we had done in the past to help our approach to this task.

2 Methods

First, we had decide the value for the distance threshold, which would determine robots closeness to the wall. The most important IR sensor in our case is number 6, which is the sensor that is used to measure the closeness to the wall. The robots distance threshold is set the threshold to 150. If it detects a wall (or an obstacle) close to it, it will go to PID control in order to maintain the distance threshold to the wall.

Main Controller Algorithm
If haven't detected the wall => Keep going Straight
If reached a corner => Turn left on the spot
If detecting an obstacle on the left side => Curve to the right
Else go to PID control and maintain the distance to the wall

Algorithm 1 PID Controller

```
while N ≠ 0 do
  if Obstacle on left detected then
    X ⇐ X × X
    N ⇐ N/2
  else {N is odd}
    y ⇐ y × X
    N ⇐ N − 1
  end if
end while
```

One challenge we faced during the experiments was the buffering problem. Due to some power problems and possibly sending too many commands, our robot crashes. The reason for that is sending unwanted command to the robot but without printing it or in other words keeping the buffer clean. First we made sure that whenever we are about to set the speed wheels same as the previous loop, we avoid sending that command again and simply go to the next loop. We are also pausing the program for 0.1 s at the end of the loop to make sure there is enough time spaces between commands.

3 Results

Our robot is able to follow the wall properly and avoid any obstacles both on its left side and right side. In trials, each trial including on full round with some obstacle in the way, it got stuck n times, next to a grey object. We observed that it detects the darker objects, much later than the lighter objects. In other words, it must be very close to the dark object in order to detect it, which is the moment that there is sudden huge change in the sensor values. The two main problems it faced during problems was the occasional crashing, and wiggling too much around grey or dark objects due to the delay in detecting them.

We managed to reduce the crashing by the method described in the previous section. The reason for the struggle in following dark walls was the PID control. In the first approach only PD control was used, as we observed the robot doing a reasonable job. In order to reduce the wiggling with the grey objects, we increased the constant multipliers for D and I parameter.
We have experimented obstacle avoidance and wall following using both Turning on the spot method and Curved path method. We observed that the using curved path performs quite better than turning right at the spot. Using curved turning, our robot only got syck n times, while using turning on the spot method, it got stuck m times. In general, curved paths are better than turning on the spot, as a fluid motion is more desirable and turning on the spot slows down the movement.

4 Discussion

One improvement that can be done is giving the robot a side to follow. At the moment, it always follows the wall clockwise, even if the initial situation is one where following anti-clockwise is easier and more sensible. We can turn the other way around by using IR sensor 1 instead of IR sensor 6. We can make side an argument, or even better. we can compare the sensor values from both sides and check which side the wall is and then follow the way which is more sensible.

If the robot is positioned in the center of the arena (with some obstacles in the middle), Ideally, if it detects an obstacle, it should go around it and then go straight until it reaches the wall and then following the wall. However, what occurs at the moments is the robot circling that obstacle and treating it as the wall. In other words, the robot cannot differentiate between an obstacle and the wall. One possible solution to overcome this problem is odometry. By tracking its position, if the robot is reaching the same position in a short amount of time, it can conclude that the object next to it is an obstacle and not the wall. Thus, it can keep going straight after one turn until it finds the wall.

5 Appendix

```
% Code for webots that instructs the bot to move forward to a surface
% then follow the edge of that surface avoid objects

TIME_STEP = 64;
direction = 'None';          % Direction for the current frame
speed = 4;                   % General speed of the robot.
default_dist = 120;          % Closer to wall <=> Bigger number
Kp = 9/default_dist;         % Parameter for P control
Ki = 2/default_dist;         % Parameter for PI control
Kd = -4/default_dist;        % Parameter for PD control
prev_errors = zeros(1,100); % Used for PI control
steps = 1:length(prev_errors);
found = 0;                   %If the robat has reached a surface or not
n = 0;                       %Used to count instructions
current_motion = [0,0];      %Current motion of bot


delete(instrfindall)
s = openConnection           %Open connection to khepra bot

% Calling MATLAB desktop versionwb_differential_wheels_set_speed(1, -1);
desktop;

% Main loop:
% Perform simulation steps of TIME_STEP milliseconds
while 1
  sensor_values = readIR(s)

  % Get distance to wall on the right
  dist = sensor_values(6);
```

```
%Before an object is found, move forwards
if ~found
  direction = 'Straight';
end

% The next conditions handle these situations:
% To avoid head-on collisions
if (max(sensor_values(3),sensor_values(4))>default_dist ...
|| sensor_values(5) > 150)
  direction = 'Left Turn';
  found = 1;
% To avoid obstacles on the left
elseif (max(sensor_values(1),sensor_values(2))>150)
  direction = 'Right Curve';

  found = 1;
% If there are no obstacles - go to PID control
elseif found
  direction = 'PID Control';
end

if strcmp(direction, 'Straight')
  if ~(current_motion(1) == speed && current_motion(2) == speed)
    go(s, speed);
    disp('Straight!')
    current_motion = [speed, speed];
  end
elseif strcmp(direction, 'Left Turn')
  if ~(current_motion(1) == -speed && current_motion(2) == speed)
    setSpeeds(s, -speed, speed)
    disp('Left Turn!')
    current_motion = [-speed, speed];
  end
elseif strcmp(direction, 'Right Turn')
  if ~(current_motion(1) == speed && current_motion(2) == -speed)
    setSpeeds(s, speed, -speed)
    disp('Right Turn!')
    current_motion = [speed, -speed];
  end
elseif strcmp(direction, 'Left Curve')
  if ~(current_motion(1) == speed/2 && current_motion(2) == speed)
    setSpeeds(s, speed/2, speed)
    disp('Left Curve!')
    current_motion = [speed/2, speed];
  end
elseif strcmp(direction, 'Right Curve')
  if ~(current_motion(1) == speed && current_motion(2) == speed/2)
    setSpeeds(s, speed, speed/2)
    disp('Right Curve!')
    current_motion = [speed, speed/2];
  end
elseif strcmp(direction, 'Stop')
```

```
    if ~(current_motion(1) == 0 && current_motion(2) == 0)
      go(s,0)
      disp('Stop!')
      current_motion = [0,0];
    end
  elseif strcmp(direction, 'PID Control')
      % For P Control
      err = default_dist - dist;
      % For PI Control
      prev_errors = [err prev_errors(1:(length(prev_errors)-1))];
      int = trapz(steps,prev_errors)/length(steps);
      % For PD Control
      dev = diff(prev_errors);
      dev = mean(dev);
      % Do final calculations
      v = floor(Kp*err + Ki*0*int + Kd*dev);

      if v > 7
        v = 7;
      end
      if v < -7
        v = -7
      end

      if ~(current_motion(1) == speed+v && current_motion(2) == speed+v)
        setSpeeds(s, speed+v, speed-v);
        current_motion = [speed+v,speed-v];
      end
      disp('PID Control!')
  else
      disp(['Something wrong! Recieved command: ' direction])
  end

  n = n + 1;
  pause(0.1);
end
go(s, 0);
disp('Stop!')
```