ESTIMATION OF TIME TO COLLISION

DECEMBER 20, 2021

Submitted to: Prof. Dr. Kemalettin Erbatur

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Autonomous Mobile Robotics (ME 525) Lab

TABLE OF CONTENTS

INTRODUCTION

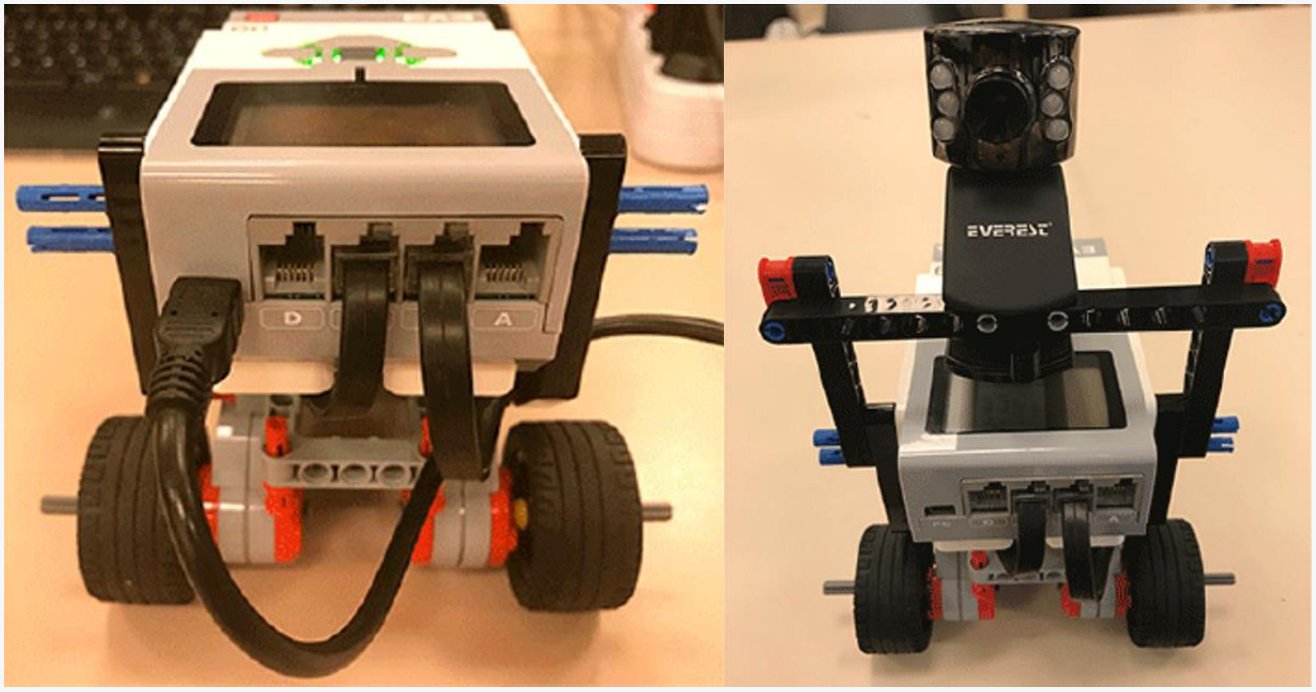
This lab involves the construction of a two-wheeled differential drive robot equipped with a camera followed by using MATLAB to estimate the time to collision with an obstacle. The obstacle in this case is a simple rectangular black pattern on a paperboard.

The major aim of this lab is to take sequential images of the obstacle during relative motion between the robot and the obstacle, calculate the height of the obstacle (after extracting its corners), and use this data alongside other defined parameters to calculate how long it will take for collision to occur.

MATLAB's support package for cameras is used to take the images, while Harris corner detection is applied using the related function in the Computer Vision toolbox.

PROCEDURE

Firstly, the differential drive robot is assembled using the necessary parts as shown in the figure below and the camera is mounted:



The major components used are:

- Two large motors - Two castor wheels - Two actuated wheels
- The EV3 Brick - A camera

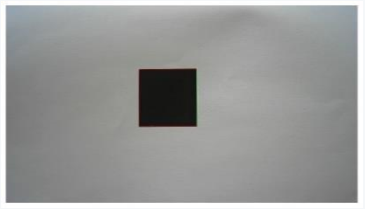
MATLAB is the underpinning software for this application.

To perform the necessary tasks, a program is written in MATLAB. Prior to this, a USB connection is established between the robot and the PC on which the MATLAB code is written.

MATLAB Code

After creating a connection to the Ev3 and camera, we perform the following:

- Three complementary MATLAB scripts are written – main.m, getImgFindcorners.m, and calculateTTCandMove.m



getImgFindcorners.m

- With this script, an image of the obstacle is taken and converted to grayscale.
- Using the inbuilt Harris features function in the Computer Vision Toolbox, the strongest corners within the image are extracted.
- The coordinate of each corner is obtained using the max and min functions. This is used to plot the rectangle, after which the length of the right edge is calculated (using the norm function).

calculateTTCandMove.m

- The time to collision is calculated here using the equation below:

$$TTC = \frac{L_1}{\frac{L_1 - L_2}{\Delta t}} = \frac{L_1 \Delta t}{L_1 - L_2}$$

Where L_1 and L_2 are the heights of the right edge in sequential images and Δt is the time between each acquired image.

- A key piece of detail is to set TTC to zero whenever the robot is stationary. This could simply be implemented by checking the condition $L_1 = L_2$.
- The test cases are now checked using a switch-case implementation.
- This is followed by a calculation of the distance travelled by the robot using the following.
- After every motion run within the while loop, some computations are performed using the encoder speeds for the left and right motors($\triangle E_r$ and $\triangle E_l$):
  - i. The distance travelled by each wheel is calculated by:

$$D_r = 2\pi R \frac{\triangle E_r}{360}, \qquad D_l = 2\pi R \frac{\triangle E_l}{360}$$

  - ii. With the above results, the distance travelled by the center of the wheels and the change in angle are calculated:

$$D_c = \frac{D_r + D_l}{2}, \qquad \Delta\theta = \frac{D_r - D_l}{L}$$

- The distance is now stored the counter is incremented for the next run.

$$x(k+1) = x(k) + D_c.cos\left(\theta(k) + \frac{\Delta\theta}{2}\right)$$

calculateTTCandMove.m

- The pseudo code for this script is as follows:

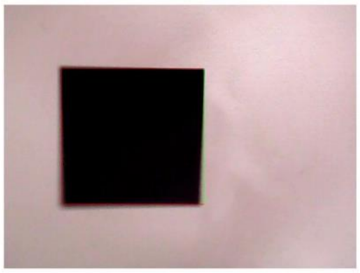
```
main.m
    clear all; close all; clc;
    Define and verify Lego connection;
    Define motor connections;
    Reset encoders;
    cam = webcam(1);
    cam.Resolution = '320x240';
    while 'down' is NOT pressed do
        getImgFindCorners;
        pause(1/25);
    end
    Adjust motor speed and start;
```

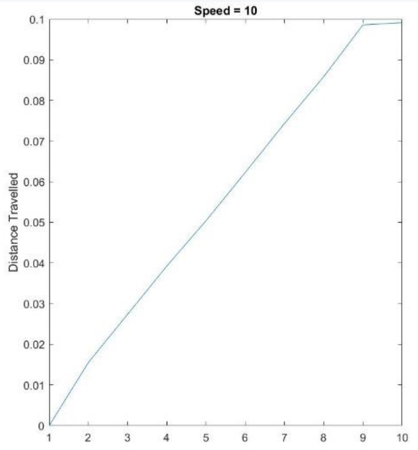
```
while 'up' is NOT pressed do
    start=tic;
    subplot(1,2,1);
    getImgFindCorners;
    Assign the calculated length to $L_1$ ;
    pause(deltaT);
    getImgFindCorners;
    Assign the calculated length to $L_2$ ;
    calculateTTCandMove;
    subplot(1,2,2);
    Plot the travelled distance;
    elapsed=toc(start);
    pause((1/25)-elapsed);
end
Stop motors;
```

RESULTS AND COMMENTS

The following results were obtained from the implementation of the MATLAB code on the differential drive robot equipped with a camera:

Test Case 1:

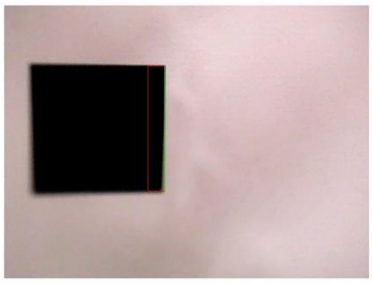


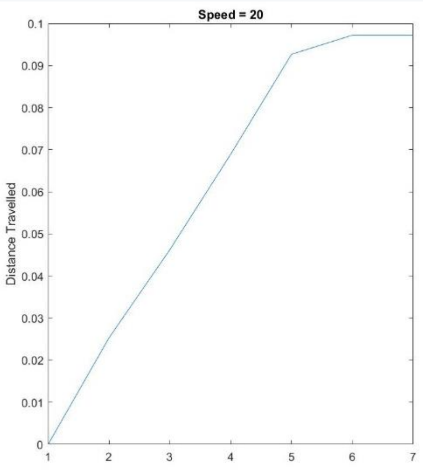


The image taken by the camera, although not crisp, was good enough for the Harris Corner detection algorithm. As expected, the distance plot is almost a straight. I noticed that adjust the castor wheels before motion really helps avoid uneven trajectories. The end of the curve (where the deviation occurs) signifies when the robot stopped at a safe distance away from the obstacle.

Test Case 1 - For a higher speed (20):

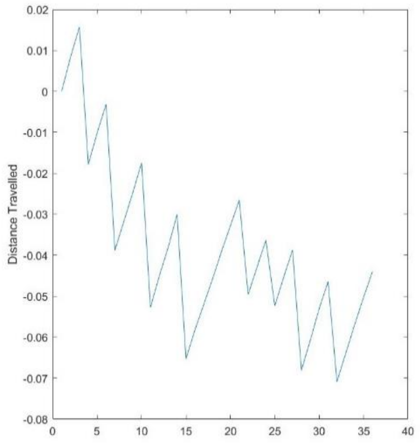
As we increase the speed, the obstacle detection algorithm struggles with perform as seen in the picture on the right:



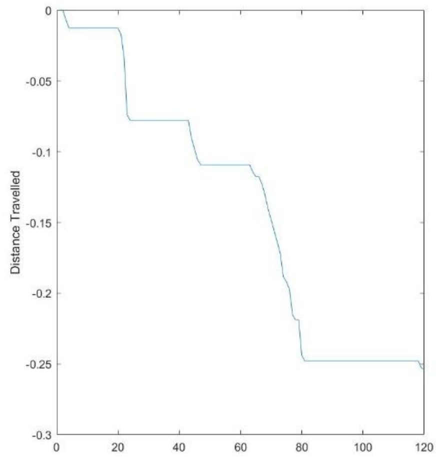


The image taken by the camera becomes blurring (*see the left edge above*) due to the vibration caused by increase in speed. This affects the Harris corner detection since the input image is now quite noisy. We can observe this in the drawing of the rectangle around the black pattern.

Test Case 2:

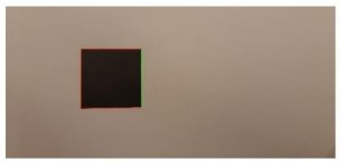
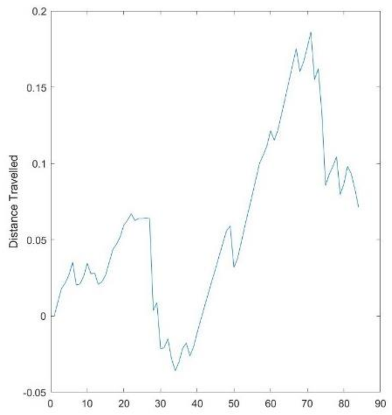


This case proved quite challenging because of the change in the orientation of the castor wheels when robot stops and moves forward/backward. The fluctuations in the distance plot show the alternating forward and reverse motions.



Test Case 3:

In this case, the flat regions show when the robot was stationary; while downward slopes show when the robot moved backward once the obstacle was within collision distance. As seen from the above, this particular case was repeated four times.

Test Case 4:

This is another challenging case because both the obstacle and the robot are moving. Non-uniform readings here arise as a result of the changing orientation of the castor wheels and the non-steady hands which affects the obstacle detection.

DISCUSSION

Q1: This is included in the previous section.

Q2: Initially, the time to stop the robot was determined intuitively after observing outputs for the calculation of TTC. It was noticed that placing a condition of $0 < TTC < 1$ was sufficient to stop the robot at a safe distance from the obstacle.

Q3: This is also included in the previous section.

Q4: At the beginning of the experiment, a random Δt would be chosen to see the performance and values of the lengths obtained. After this, the farthest distance of the robot from the obstacle on the experiment area is measure and the speed set in the code would be taken into cognizance to determine an appropriate Δt. In this case 1 second was good enough for Δt.

Q5: When the obstacle disappears from the view of the camera, the robot keeps moving while the algorithm tries to detect other corners in the scene and use the measurements from these for TTC.

CONCLUSION

This lab includes a condensed study on the estimation of the time to collision of a robot (equipped with a camera) with an obstacle. The Harris detection algorithm from the Computer Vision Toolbox in MATLAB was used to detect the corners of the obstacle, before its height was calculated. Four cases were tested to determine the flexibility of the time to collision code – i) obstacle stationary and robot moving, ii) case (i) done repeatedly by reversing the robot's motion just before collision, iii) Robot kept stationary while the obstacle moves, and iv) Both obstacle and robot moving. It was determined that better results are obtained when the robot moves at low speeds due to relative stability of the robot and the attached camera. Major error-inducing factors encountered were castor wheel orientation changes, vibrations at higher speeds, non-steady hands when the obstacle is moved, etc. In conclusion, we note that the camera is an **exteroceptive and passive sensor**.

APPENDIX

getImgFindcorners.m

```
img = snapshot(cam);
I = rgb2gray(img);
corners = detectHarrisFeatures(I);
corners = corners.selectStrongest(20);
corners = corners.Location;
X = corners(:,1); Y = corners(:,2);
imshow(img); hold on;

% Left Line
xL = [min(X) min(X)];
yL = [min(Y) max(Y)];
plot(xL, yL, 'Color', 'r'), hold on;
% Bottom Line
xd = [min(X) max(X)];
yd = [max(Y) max(Y)];
plot(xd, yd, 'Color', 'r'), hold on;
% Top Line
xu = [min(X) max(X)];
yu = [min(Y) min(Y)];
plot(xu, yu, 'Color', 'r')
% Right Line
xR = [max(X) max(X)];
yR = [min(Y) max(Y)];
plot(xR, yR, 'Color', 'g')

length = abs(max(Y) - min(Y));
```

main.m

```
clear all; close all; clc;

mylego = legoev3('usb');
```

```
% Create connections for the left (B) and right (C) motors.
Rightmotor = motor(mylego, 'C');
Leftmotor = motor(mylego, 'B');
% Reset the encoders
resetRotation(Rightmotor);
resetRotation(Leftmotor);

cam = webcam(1);
cam.Resolution = '640x480';

while ~readButton(mylego, 'down')
    getImgFindCorners;
    pause(1/25);
end

% Set the speed for the motors and start
speed = 10;
Rightmotor.Speed = speed;
Leftmotor.Speed = speed;
start(Rightmotor);
start(Leftmotor);

% The Position and Orientation Arrays.
x_array = [];
theta_array = [];

x_array(1) = 0;
theta_array(1) = 0;

% Robot Parameters
R = 0.02;
L = 0.1;
% Counter
i = 1;
deltaT = 0.1;
while ~readButton(mylego, 'up')
    resetRotation(Rightmotor);
    resetRotation(Leftmotor);
    % To change the time period
    pause(0.1)
    start = tic;
    % Plot the rectangle and calculate the length(s)
    subplot(1, 2, 1);
    getImgFindCorners;
    L1 = length;
    pause(deltaT);
    getImgFindCorners;
    L2 = length;

    calculateTTCandMove;

    % Travelled Distance
    % Read the Encoder values
    El = double(readRotation(Leftmotor));
    Er = double(readRotation(Rightmotor));

    % Calculate the required parameters
    Dl = 2*pi*R*(El/360);
    Dr = 2*pi*R*(Er/360);

    Dc = (Dr+Dl)/2;

    delta_theta=(Dr-Dl)/L;

    theta_array(i+1) = theta_array(i) + delta_theta;
    x_array(i+1) = x_array(i) + Dc*cos(theta_array(i) +
(delta_theta/2));

    % Plot the distance travelled
    subplot(1, 2, 2);
    plot(x_array)
    ylabel('Distance Travelled')
    title(['Speed = ', num2str(speed)])

    elapsed = toc(start);
    pause((1/25) - elapsed);
    % Increment the counter for data storage
    i = i+1;
end
% Stop the motors
stop(Rightmotor);
stop(Leftmotor);
```

calculateTTCandMove.m

```
% Check if the robot is stationary and calculate the time to
collision
if L1 == L2
    TTC = 0;
else
    % Calculate the time to collision using the formula
    TTC = (L1*deltaT)/abs(L1 - L2);
end
```

```
test_case = 1;
% Move the robot according to the cases
switch test_case
    case 1 % Test Case 1
        if TTC > 0 && TTC < 1
            stop(Rightmotor);
            stop(Leftmotor);
        end
    case 2 % Test Case 2
        tic
        while ~readButton(mylego, 'right')
            if TTC > 0 && TTC < 1
                speed = -10;

                Rightmotor.Speed = speed;
                Leftmotor.Speed = speed;
            else
                speed = 10;

                Rightmotor.Speed = speed;
                Leftmotor.Speed = speed;
            end
        end
    case 3 % Test Case 3
        if TTC > 0 && TTC < 1
            speed = -10;

            Rightmotor.Speed = speed;
            Leftmotor.Speed = speed;
        end
    case 4 % Test Case 4
        if TTC > 0 && TTC < 1
            speed = -10;

            Rightmotor.Speed = speed;
            Leftmotor.Speed = speed;
        else
            speed = 10;

            Rightmotor.Speed = speed;
            Leftmotor.Speed = speed;
    otherwise
        warning('Unknown Case!')
end
```