ME 425 LAB 4 Report
Computer Vision On Turtlebots

Arda Gencer
34124
Instructor: Mustafa Ünel

Fall 2025-2026
Due Date: 10/12/2025

1 Introduction

,

In the Lab # 4 of ME 425, an introduction to vision and how image processing can be used in robot motion control was made. To do this, a paper with a black square was used in a way that the robot would detect how close it is to collide with the paper (Time To Collision) and adjust its speed accordingly.

2 Procedure

First of all, the connection to the Turtlebot was made just like the other labs. After this, 3 different MATLAB scripts were implemented. One script would read the image and detect the square, one would calculate the TTC and adjust the speed accordingly, and one would run the main algorithm of the program and manage the subscriber/ publishers and also record TTC data.

1. **getImgFindCorners.m:** This script reads the image from the subscriber and converts it to black and white format. Then, it crops % 30 of the image from all sides since only focusing on the center of the image is enough for this application. Then, it uses the "detectHarrisFeatures" function of the ROS Computer Vision Toolbox to detect the corners in the region of interest. After this, if the square is seen (i.e. 4 dominant corners are detected in the roi, it will set a variable true to indicate that the robot can see the square. It will also plot the image and the square on a figure.

2. **calculateTTCandMove.m:** This script calculates the TTC by using the l_1 and l_2 information that is received from the main script. If l_1 is approximately equal to l_2, it sets TTC as 100 since this means that the robot is almost stationary. Else, it calculates TTC according to the following formula:

$$TTC = \frac{l_1 \Delta t}{l_1 - l_2} \text{ where } \Delta t \text{ is the sampling time.}$$

   Then, if TTC is smaller than 10, the robot decides it is too close to the target and stops- decreases its velocity depending on the task. If it is greater than 10, the robot moves forward or stops(for task 2).

3. **main.m:** In this script; the connection, node creation, and subscriber- publisher declarations were made. Then, the sampling time and exeTime are defined. After that, a loop is ran until the end of the execution time. Inside this loop, the first script is ran repeatedly and the robot checks if the square is detected or not. If the square isn't detected, the robot waits for 0.04 seconds and runs the first script again. If the robot can detect the square, l1 and l2 are assigned and second script is ran to determine the TTC and set the velocities accordingly. The TTC is also then saved to a vector for plotting. After the execution is done, the TTC values for whole execution is plotted.

3 Results

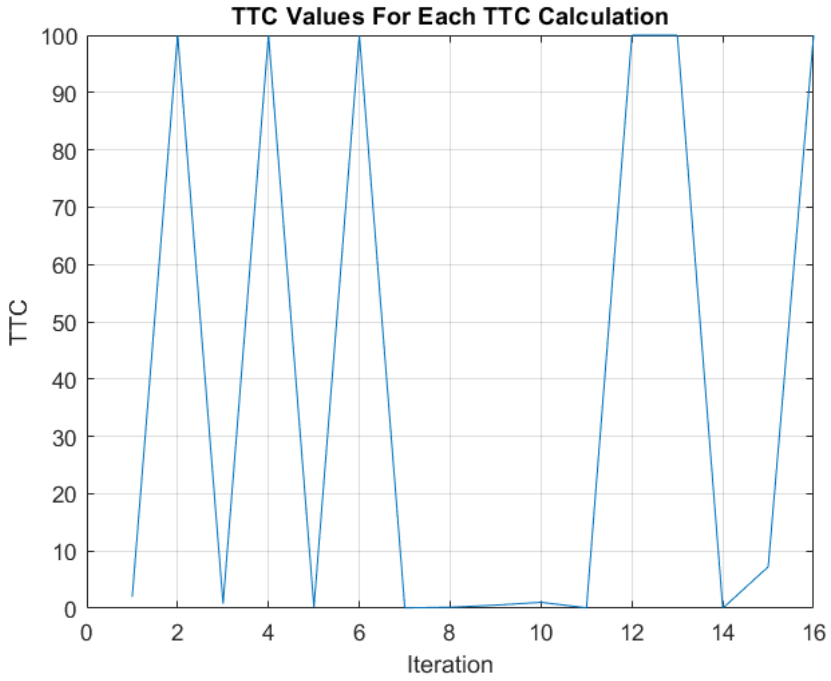

Figure 1: The TTC Values Recorded For Task 1

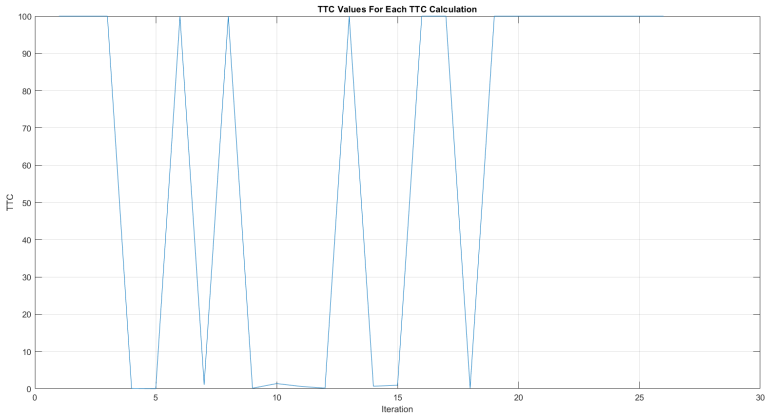

Figure 2: The TTC Values Recorded For Task 2

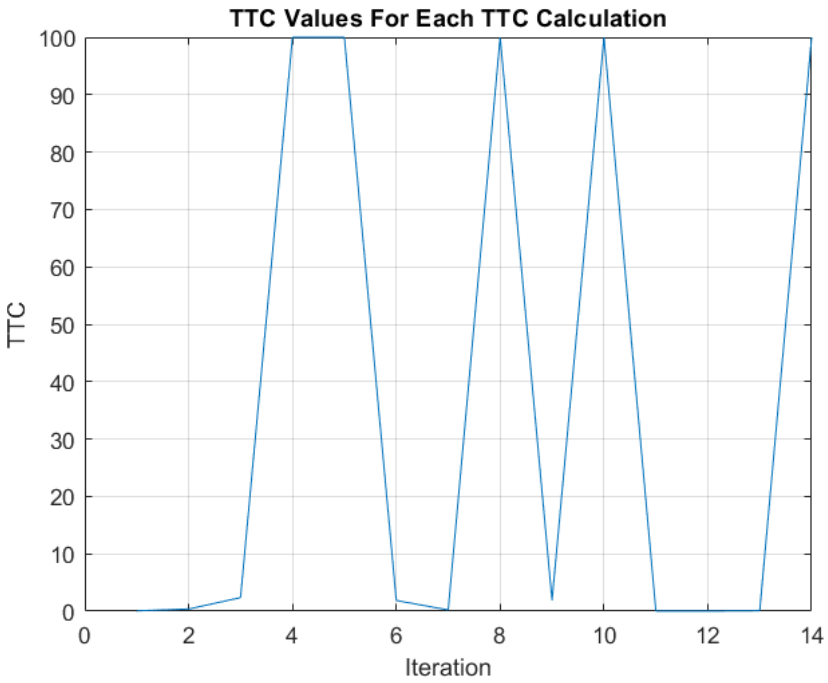

Figure 3: The TTC Values Recorded For Task 3

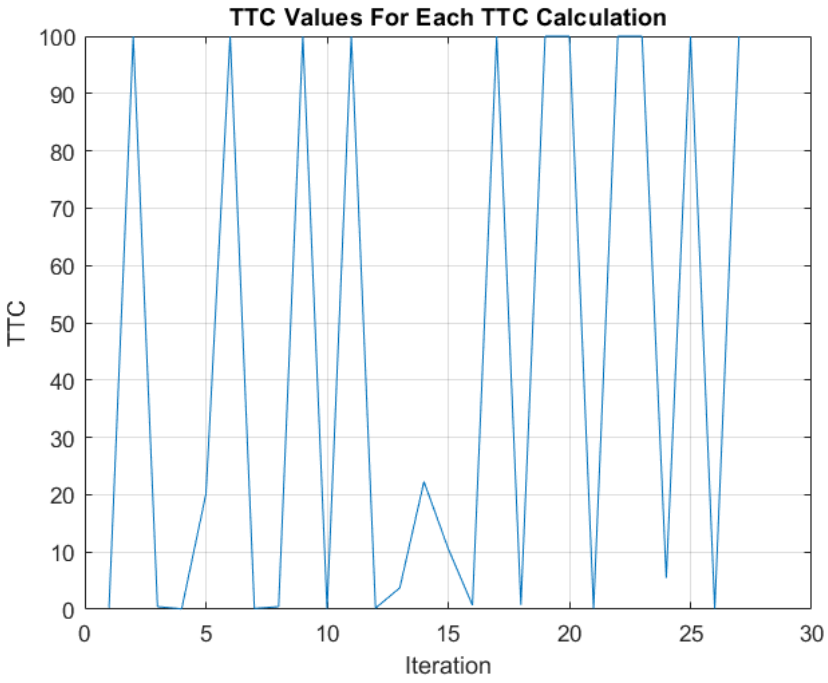

Figure 4: The TTC Values Recorded For Task 4

Link To The Application Of Task 1: https://drive.google.com/file/d/1xDHMuLngZN_JjB8SZL7H4ji8eAsRQjWg/view?usp=sharing

Link To The Application Of Task 2: https://drive.google.com/file/d/1q9Yc76k-MbHYKm0fZN
view?usp=sharing
Link To The Application Of Task 3: https://drive.google.com/file/d/1rdfOhBuEACSHtzIfbY
eA2uIkwN/view?usp=sharing
Link To The Application Of Task 4: https://drive.google.com/file/d/1ZEaFqzTWYGCIYnGg0z
view?usp=sharing

4 Conclusion

In this lab, a solid introduction to computer vision algorithms was made and how image data can be used for motion control was demonstrated by using TTC and controlling the robot's motion according to this data that was received.

5 Discussion

1. **Complete all the Test cases and comment on your results:** Overall, the robot could stop when it got too close to the black square. However, due to connection issues between MATLAB and the Turtlebot, the program couldn't receive the camera data in time and this resulted in weird calculations of TTC and the robot ended up stopping abruptly or colliding with the paper.

2. **How did you determine the TTC threshold for stopping the robot?:** The TTC threshold was determined experimentally by testing the robot and looking at whether the robot would collide with the paper(in which case threshold was increased) or if the robot stopped too early(in which case threshold was decreased). In the end, the threshold was set to 10 seconds.

3. **In Test 1, how did changing the TurtleBot3's forward linear velocity affect the stopping distance and TTC curve?:** Increasing the linear velocity usually causes the robot to move closer to the obstacle/ colliding with it. Decreasing the speed allows the robot to produce more accurate results that is closer to TTC. This could be because decreasing velocity allows more time for the information to by sent to the MATLAB and for velocity commands to reach back to the Turtlebot.

4. **How did you choose the sampling interval Δt?:** I used the default value 0.01 that was given in the code. This value proved sufficient enough results so I didn't change the value. However, if the program ran too slow or too fast, the sampling time could be changed accordingly.

5. **What happens to your TTC calculation when the obstacle leaves the camera field of view (e.g., too close, too far, or moved sideways)?:** I wrote my code in such a way that if the camera stops detecting the square, TTC will not change. In this case, the robot will continue moving as if no obstacle is seen.

6 Appendix (All MATLAB Scripts That Were Used):

Listing 1: getImgFindCorners.m. Gets image data, crops it and extracts the corners from it.

```
imgMsg = receive(imgSub, 10);
img = rosReadImage(imgMsg);
I = rgb2gray(img);
[rows, cols] = size(I);

%Crop to Center (Region of Interest)
roiPercentage = 0.4;
r_min = floor(rows*(1-roiPercentage)/2); r_max = floor(rows*(1+
    roiPercentage)/2);
c_min = floor(cols*(1-roiPercentage)/2); c_max = floor(cols*(1+
    roiPercentage)/2);
I_roi = I(r_min:r_max, c_min:c_max);
corners = detectHarrisFeatures(I_roi);
corners = corners.selectStrongest(20);
if corners.Count == 4
    objectDetected = true;
    %Offset corners back to original image coordinates,
    X = corners.Location(:,1) + c_min; Y = corners.Location(:,2) + r_min;

    %Now you have x and y coordinates of the detected corners (as vectors)
    imshow(img); hold on;
    %Plot the rectangle in red
    rectangle('Position', [min(X), min(Y), max(X)-min(X), max(Y)-min(Y)],
        ...
         'EdgeColor', 'r', 'LineWidth', 2);
    %Plot the right edge in green
    line([max(X), max(X)], [min(Y), max(Y)], 'Color', 'g', 'LineWidth', 2)
        ;
    %Calculate the length of the right edge
    rightEdgeLength = r_max - r_min;

    hold off;
else
    objectDetected = false;
    imshow(img);
end
```

Listing 2: calculateTTCandMove.m. Calculates TTC with the l_1 and l_2 values and adjusts the robot's speed accordingly.

```
% Calculate TTC
TTC = min(100,l1 * deltaT / abs(l1 - l2));
%Save TTC to the array
TTCArray = [TTCArray, TTC];
% Move the robot according to Test cases
if TTC < 10
    goBackDuration = 4;
    timer = tic;
    while toc(timer) < goBackDuration
        velMsg.linear.x = -0.05;
    end
else
    velMsg.linear.x = 0.05;  % Normal speed
end
send(velPub, velMsg);
```

Listing 3: main.m. Runs the main loop off the program and determines whether to execute second script or not depending on if the robot can see the black square or not.

```
%% Environment Setup
clear; close all; clc;

clear node; % clear previous node handle if it exists

setenv('ROS_DOMAIN_ID','39'); % Set to your robots ROS_DOMAIN_ID (check the robots ID card)
setenv('ROS_LOCALHOST_ONLY', '0'); % 0 implies multi-host communication
setenv('RMW_IMPLEMENTATION','rmw_fastrtps_cpp'); % Middleware

% Create a unique node name for this MATLAB session
node = ros2node('TTC');

%% Subscribers And Publishers
%Setup /cmd vel publisher
velPub = ros2publisher(node,"/cmd_vel","geometry_msgs/Twist", ...
"Reliability","reliable","Durability","volatile","Depth",10);

velMsg = ros2message(velPub);

%Setup /image raw/compressed subscriber
imgSub = ros2subscriber(node,"/image_raw/compressed","sensor_msgs/CompressedImage", ...
"Reliability","besteffort","Durability","volatile","Depth",10);

%% The Program
deltaT = 0.01;
exeTime = 100;
t0 = tic;
TTCArray = [];

getImgFindCorners;
```

```
while toc(t0) < exeTime
    while objectDetected == false  %If no object is detected
    getImgFindCorners;
    send(velPub, velMsg);
    pause(1/25);
    end

    while objectDetected == true
    getImgFindCorners;
    %Assign the calculated length to L1
    l1 = max(Y) - min(Y);
    pause(deltaT);
    getImgFindCorners;
    %Assign the calculated length to L2
    l2 = max(Y) - min(Y);
    calculateTTCandMove;
    end
end

%Stop motors;
velMsg.linear.x = 0;
velMsg.angular.z = 0;
send(velPub, velMsg);

%% Plotting TTC
figure;
plot(TTCArray);
title("TTC Values For Each TTC Calculation");
xlabel("Iteration");
ylabel("TTC");
grid on;
```