Rochester Institute of Technology

Principle of Robotics Project – Fall 2016

# Dynamic Path Following Robot

## Using vision based controls

## Mission Statement

This project explores making a wireless automatic guided robot which does not require any kind of intrusive modifications to be made to the environment, apart from installing an overhead camera. Generally environments that make use of automatic guided vehicles (AGVs) have to plan the path(s) where the robots should go before installing the tracks, like magnetic strips or metal tracks; this is an investment even before using the robots. If any change to the paths is required to be made, then more cost is incurred. In this paper a four wheeled differential drive robot has been controlled wirelessly to follow paths drawn on a graphical user interface within a workspace of 1.8m by 1.4m. The goal of this project is to control the robot by correcting its orientation through camera feedback. Error analysis to investigate how well the robot follows the path drawn have been taken.. The estimated error of the robot is within a few centimeters of the path and can be reduced by modifying various thresholds.

## People

| STUDENTS | TEACHING ASSISTANTS | PROFESSOR |
|---|---|---|
| *Amar Bhatt, Rasika Kangutkar* | *Alex Synesael , Mazin Ali* | *Dr. Ferat Sahin* |

## System Overview

This system consisted of four main sub-components; Robot, OpenCV, MATLAB, and the Environment workspace itself. Fig.1 shows the entire system architecture and work flow using these four main components.

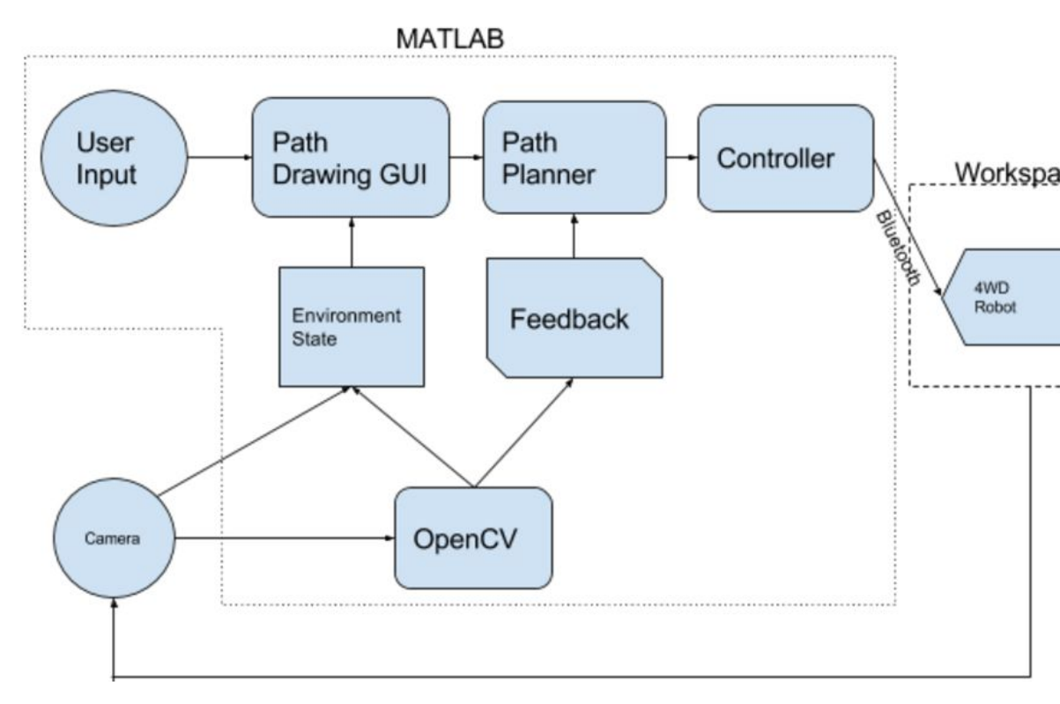


Fig. 1 System Architectire

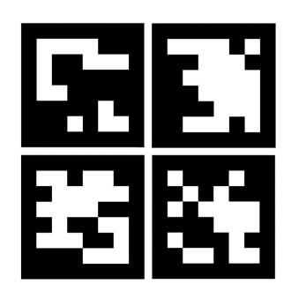


Fig. 2 Four ArUco Markers

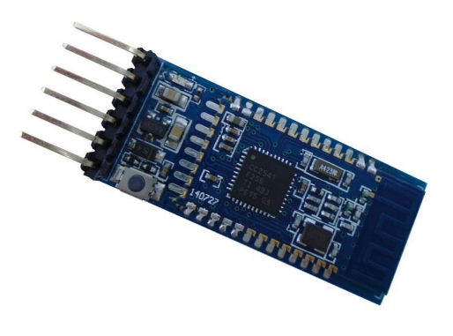


Fig. 3 HC- 06

Fig. 4 Logitech Webcam

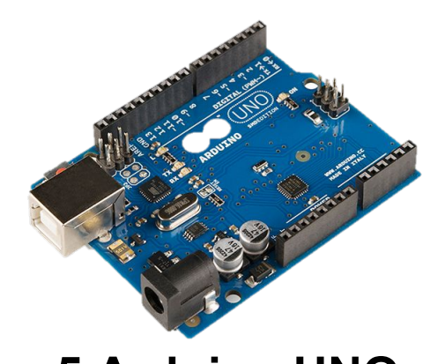


Fig. 5 Arduino UNO board

### Differential Wheel Drive Robot

Four-wheeled differential drive robotics platforms are commonly used in navigation applications. The system used in this setup was the 4WD Rover I from Lynxmotion, as shown in Fig. 5.

### MATLAB

Here, MATLAB is used for the GUI, interface with OpenCV and the camera, for path creation sending serial commands to the robot via Bluetooth.

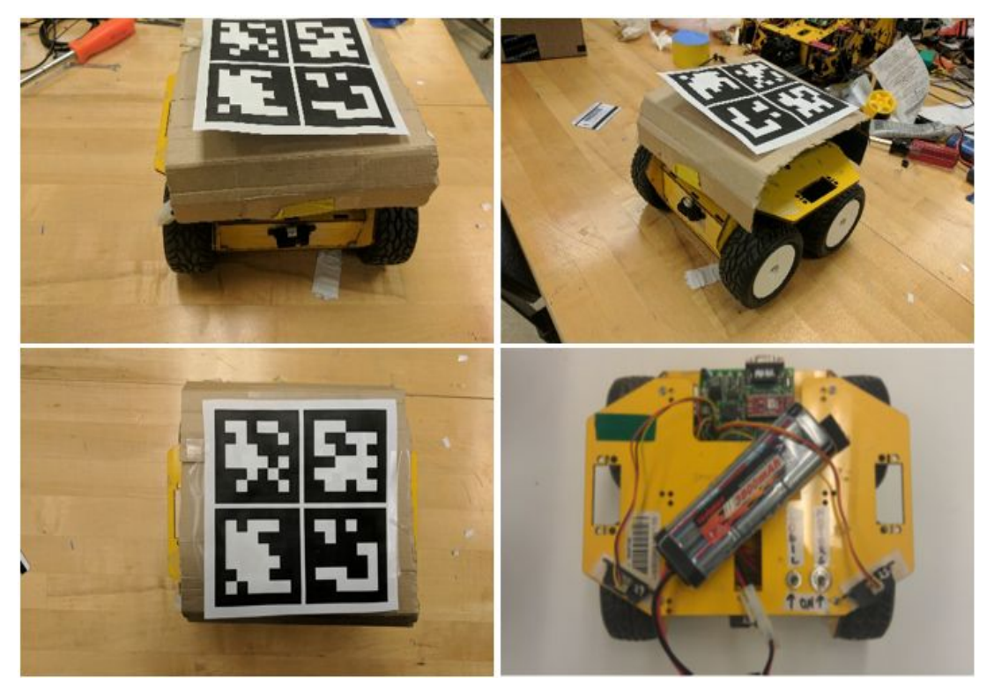


Fig. 6 4WD Rover from Linxmotion

### OpenCV

OpenCV is a comprehensive computer vision toolbox used for a variety of real-time applications such as tracking, facial recognition, segmentation, and classification. ArUco markers are pattern-based black and white squares that can be easily identified through OpenCV. For this application four ArUco markers were used as shown in Fig. 2. This marker array was printed and mounted flat on top of the robot so that the center of the array was at the center of the robot. The top two markers were facing in the direction of the front of the robot and were used to determine the orientation of the robot.

### Environment

A webcam was placed on the ceiling of the room which was about 8 feet from the ground and faced perpendicular to the floor. The webcam used is shown in Fig. 4. The webcam was used with a resolution of 640 px by 480 px. This created view-able workspace of about 1.8m (6 feet) by 1.4m (4.5 feet).

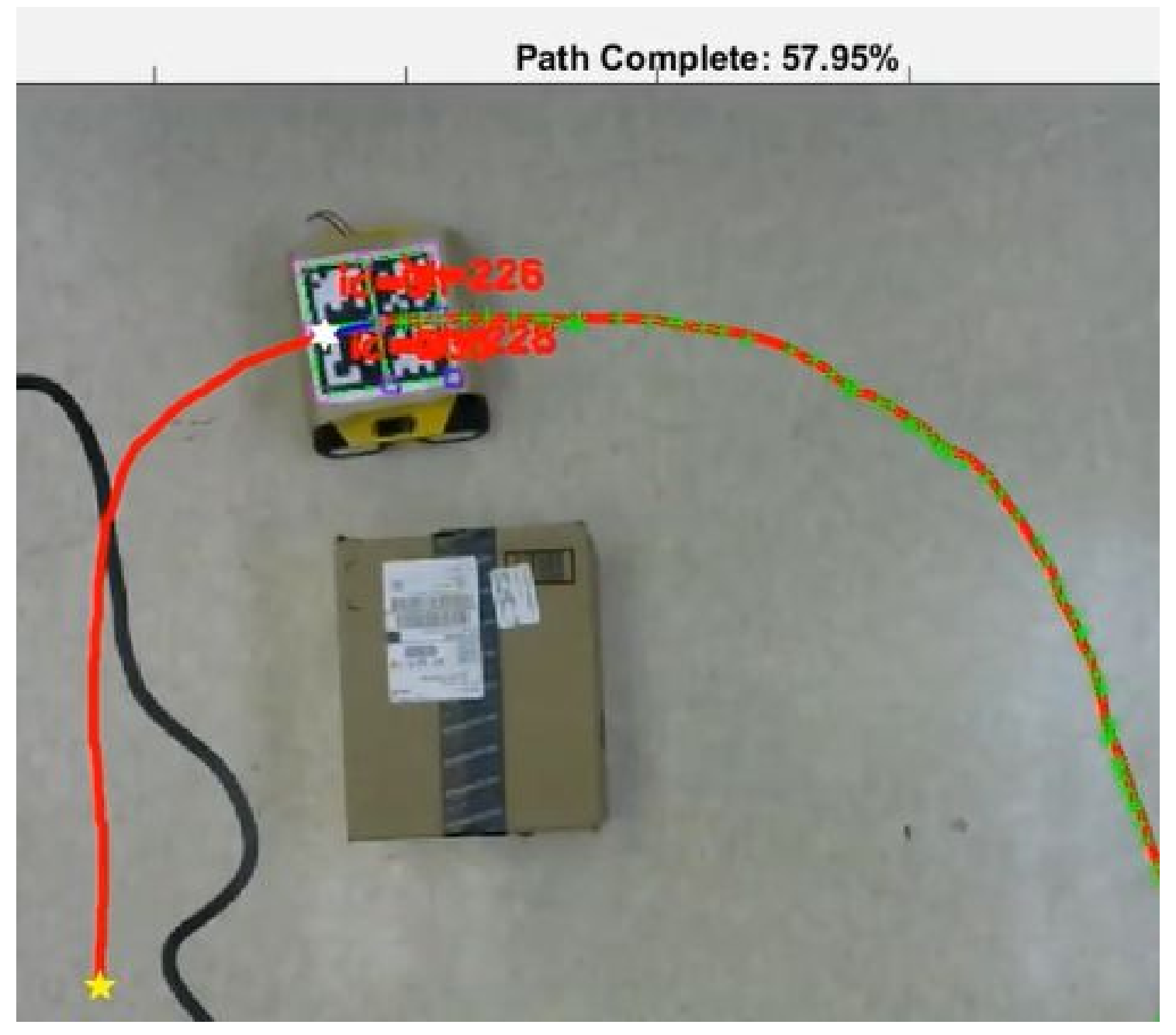


Fig. 7 Robot following the red path drawn in the GUI; yellow star indicates goal, white star indicates subgoal, green indicates points covered by robot.

## Methodology

As an overview, the procedure of running this system takes into account all four main components. First, the camera is set-up to view a bird's-eye view of the the workspace. The robot is then placed within that workspace with its ArUco markers Fig. 2. The application is then started using MATLAB. OpenCV functions are used to detect the ArUco markers and determine the millimeter to pixel ratio since the ArUco markers are a known size. Once the robot is detected the user is prompted to draw a path over the current image. This path is then fed to the path planner code where the robot iteratively follows the following steps:

1. The robot faces desired point
2. The robot goes towards point
3. Feedback from OpenCV to determine whether or not robot meets success thresholds
4. The robot's movement is adjusted to meet threshold if needed
5. Once the point is reached, the desired point moves to the next point

Once the robot completes the desired path the errors from the robot movement versus the ground truth path are found. The user is then prompted to draw another path. This basic procedure is shown in the flowchart in Fig. 8.

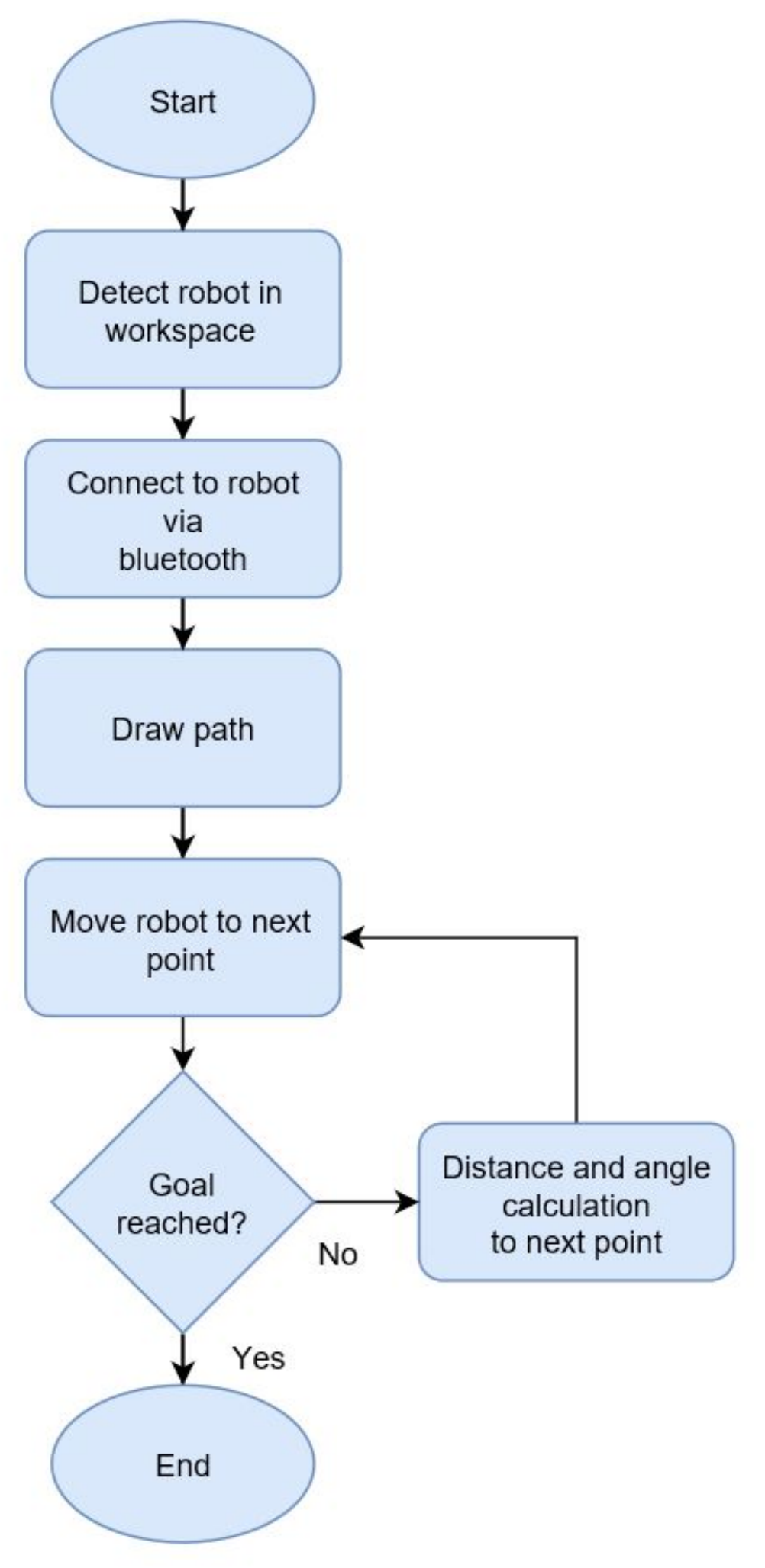

Start
Detect robot in workspace
Connect to robot via bluetooth
Draw path
Move robot to next point
Goal reached?
No
Distance and angle calculation to next point
Yes
End

Fig. 8 Flowchart of Procedure

### Thresholds

To ensure the robot follows the path accurately, a threshold was created for the allowable distance it was from the current point. Also, an orientation threshold was defined such that Vo direction of the robot and Vg desired direction of the robot were close to one. If Vo•Vg = -1 then the robot is facing in the opposite direction of the point and needs to turn 180 degrees.

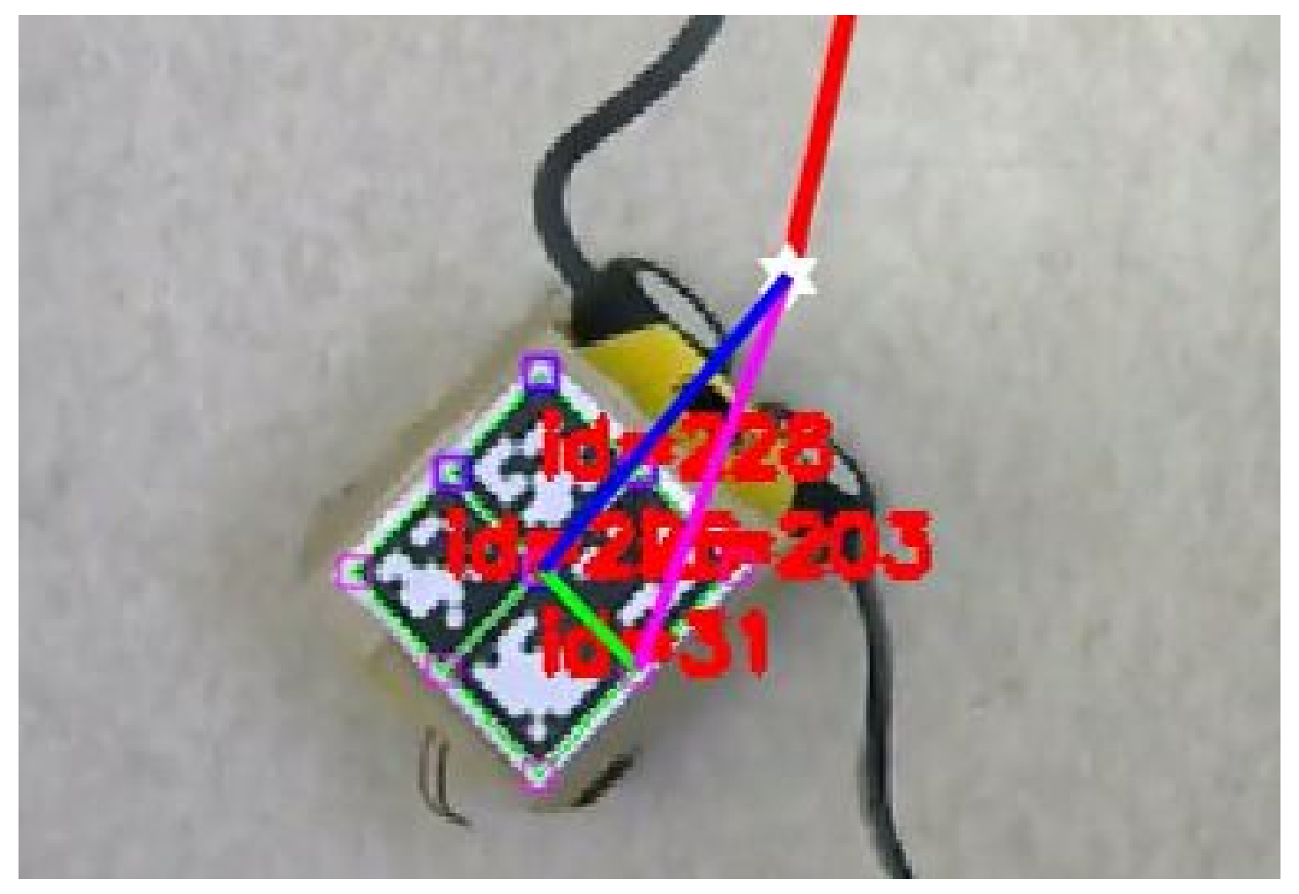
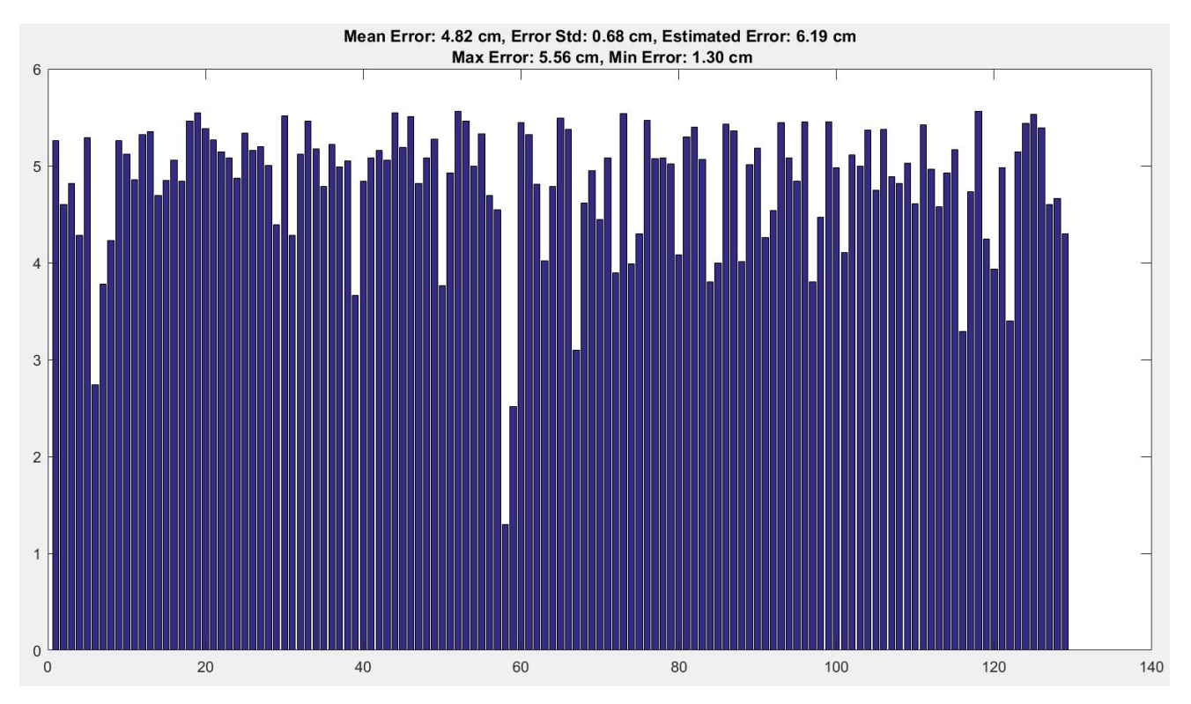


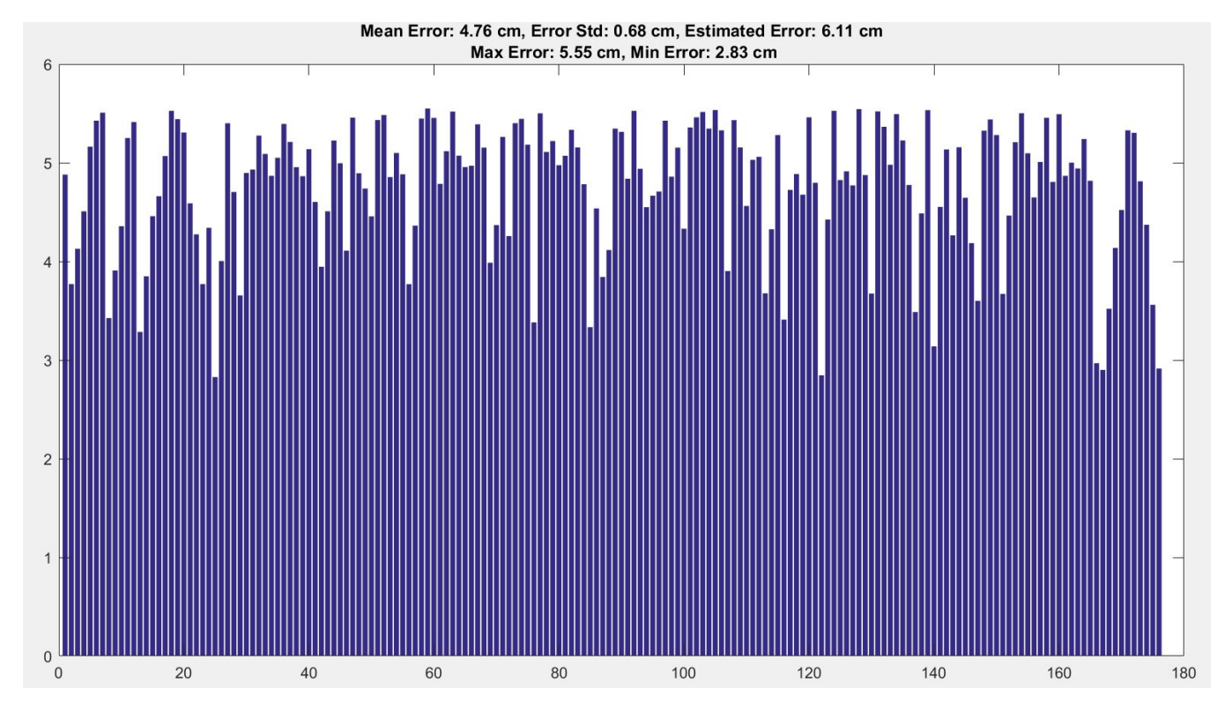
Fig. 9 Setting Robot Heading Direction:green indicates current heading direction of robot Vo; blue indicates desired heading direction point Vg; magenta indicates difference between the directions which is the vector from the front of the robot to the desired point Vn.

## Tests

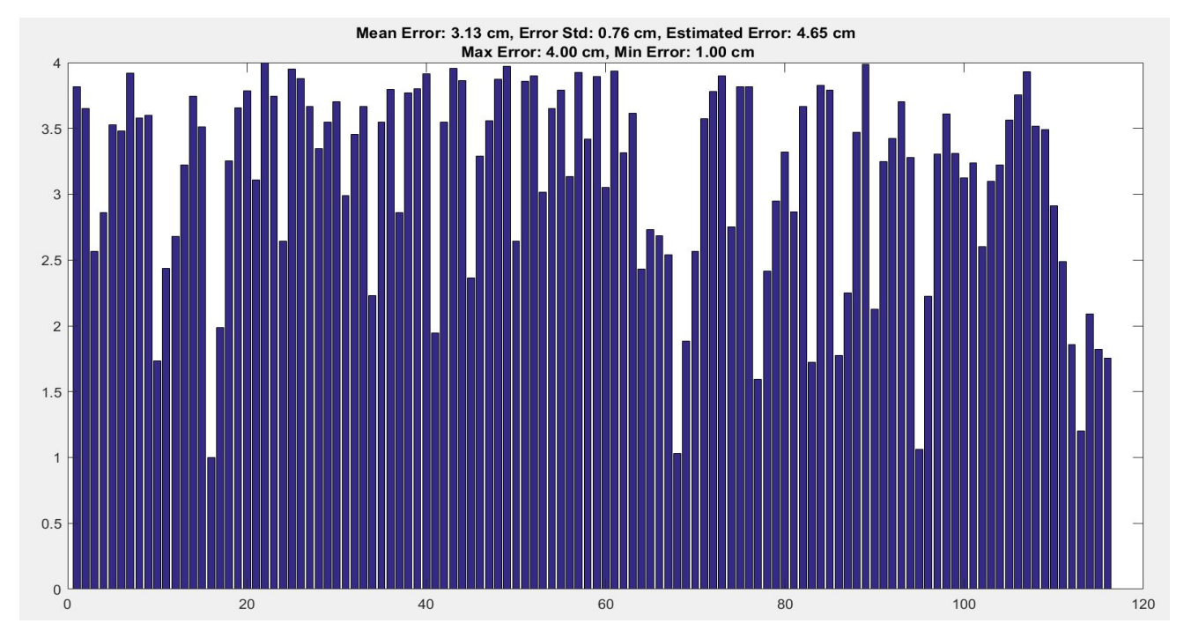
By varying the position and orientation thresholds multiple tests were carried out to see how well the robot follows the path drawn. The following figures display the mean, maximum, minimum, standard deviation and estimated error of the system with various thresholds.



(a)



(b)



(c)

ERROR ANALYSIS

| $a^*$ | $d^*$ | STD | Mean Error | Max Error | Min Error | Estimated Error |
|---|---|---|---|---|---|---|
| 0.95 | 20 | 0.68 | 4.82 | 5.56 | 1.30 | 6.19 |
| 0.99 | 20 | 0.68 | 4.76 | 5.55 | 2.83 | 6.11 |
| 0.99 | 15 | 0.76 | 3.13 | 4.01 | 1.00 | 4.65 |
| Error in cm, $d^*$ in px<br>$a^*$= orientation threshold, $d^*$ = distance threshold | | | | | | |

(d)

Fig. 10. (a) Position Threshold: 20px Orientation Threshold: .95, (b) Position Threshold: 20px Orientation Threshold: .99, (c) Position Threshold: 15px Orientation Threshold: .99, (d) Summary of Error Analysis

## Results

As the threshold for orientation increased the estimated error decreased. However, the largest contributor to the decrease in estimated error was the decrease in the distance threshold. Upon performing further tests, it was found that if the distance threshold is decreased then the orientation threshold must be increased, otherwise the robot would reach each point on the path but it will be unable to accurately produce the path due to the over-correction it will need to do. The orientation threshold contributes to lowering the over-correction needed. It was also found that lowering the distance threshold too much would result in the robot not being able to reach a point. This is because while the ArUco marker detection is robust, it is not very stable and does shift by a few pixels causing errors in the system when the distance threshold is too small. Another threshold needing to be modified was the time threshold. This threshold was used to determine how often movement commands would be sent to the robot. If it was too high the robot would overshoot points, if it was too low then the robot would get overloaded with commands. Therefore, a threshold of 50 ms was used after testing various values. The table in Fig. 10 (d) summarizes the error analysis of the tests.

## Conclusion

The dynamic path drawing system created worked very well. The robot showed that it could follow a path with very little error. By changing the distance and orientation thresholds the accuracy of the robot can be further improved. To solve the issue of a non-constant center point, the system can take an over time average of the center point to keep it steady. Another method would be to use an ArUco marker by itself for the center of the robot. Some future developments to this system could be obstacle avoidance. This would require identifying obstacles in the environment, avoiding them, and finding the path again. This can be done through obstacle detection using OpenCV and the camera, or on the robot level using ultrasonic sensors. Another possible advancement would be to control more than one robot in the workspace.