Dynamic Path Following Robot using Vision Based Control

Amar Bhatt and Rasika Kangutkar

Abstract— **This paper 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 path(s) 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 robot is controlled by correcting its orientation through visual feedback from a camera. Error analysis was performed to investigate how well the robot followed the path drawn. The estimated error of the robot is within a few centimeters of the path and can be reduced by modifying various thresholds.**

Keywords: ArUco, automatic guided vehicle, Bluetooth, camera, fiducial marker, differential drive, OpenCV, vision control

I. INTRODUCTION

Robots whose path is controlled by the user are termed as automatic guided vehicles (AGVs). AGVs follow paths using lasers, cameras, or are physically attached to these paths. These mobile robots follow predefined routes and so, are heavily used in industries for materials handling. Apart from transporting loads, these kinds of robots are used in risky environments and explorations, like space, mines and underwater. For guiding some AGVs, lines, magnetic tracts, or wires are installed on warehouse floors. Hence, making use of these robots requires an initial investment. Vision-based AGVs follow a physical line or markers in a room: landmark based navigation. Combining these ideas a robotic system was created for dynamic path following using vision based control.

This paper explores making a wireless automatic guided robot that does not require any kind of intrusive modifications to be made to the environment apart from installing an overhead camera. The camera above the workspace tracks the robot continuously and is used to help correct it to follow a predefined path. The user is allowed to define the path the robot should follow in a GUI created in MATLAB.

Using a live image of the environment from a bird's-eye-view, the robot is tracked in real-time using ArUco fiducial markers. The live images are fed to a software GUI where the user can freehand draw any path they desire directly on top of the image. The system then sends commands to the robot to move it along this path by using the camera and ArUco markers as feedback control. Doing this, the user can draw complicated paths around a room and around obstacles for the robot to navigate.

The following sections will cover related works in the field, an overview of the system, the procedure implemented, results achieved, and concluding thoughts with future work.

II. RELATED WORKS

[1] discusses the importance of AGV over manually operated forklifts and describes their importance in materials handling. The authors go over how implementation of track based AGVs require modification to the factory environment that will be used. AGVs relying on a laser guidance system are extremely expensive and rely on the installation of laser reflectors. To overcome the hurdles these automatic guided solutions pose, the authors develop a vision based system by fusing a low cost camera, inertial sensors, encoders, and a gyroscope for realizing automatic driving.

Though vision systems decrease the amount of modifications that need to be made to the environment when tracks are used, these systems require localization. With track-based methods the robot is attached physically to it, but with vision systems the position of the robot in the environment must be known at all times so that it can be given paths to move on.

There are many methods for vision based localization. In [2] the authors discuss the challenges of natural landmark localization in a warehouse environment since such a setting offers few unique points of interest. Making use of artificial landmarks and placing them strategically is discussed in this paper. [10] discusses how making use of stereo vision and artificial landmarks they were able to localize the robot.

Apart from relying on landmarks, a vision system can rely on painted black lines to follow the right path. In [3] the authors discuss the robots reaction when a line is discontinuous. Using image processing and edge detection techniques the problem is solved. Another localization method is discussed in [4]. Making use of global localization with RFID technology and then Kalmann filters over particle filters with a vision system provides a reinforced way to localize a robot.

Despite making use of artificial landmarks for localization, situations where no landmarks is visible or even multiple landmarks may occur. Now multiple comparisons need to be carried out to figure out where the robot is exactly. [5] makes use of Bluetooth low energy beacons to position itself. Depending on the strength of signal received the robot can localize itself thereby being able to navigate on the predetermined path in the environment.

Whether the AGV system is set up to use RFIDs, vision systems or cameras, the actual behavior of the robot needs to be known i.e. how well is it following the given path. Having a navigation performance evaluation of AGVs is thus

needed for comparing systems and defining the efficiency of the system. [6] discusses the evaluation of AGV systems. One way it achieves this is by comparing the ground truth path to the physical path that was traveled. It also defines a procedure and test strategy to normalize results across the AVG research spectrum.

Apart from having robots help in transporting materials in warehouses, user controlled robots may be needed during disasters, mines, underwater explorations, space expeditions, inspections, working in nuclear areas, etc. [7] shows how robots are controlled with feedback from a fixed video camera. [7] is able to evaluate whether the robot is following the defined trajectory and the amount of error.

[8] discusses remote Bluetooth/WiFi controlled line following robot that is able to avoid obstacles using infrared sensors. The user is able to control the robots motion over a GUI.

In [9] the authors discuss how it is important for multiple robots to interact with each other without any collisions and have developed an autonomous tracking algorithm. In this a single robot is made to be the head of the pack and the rest are follower robots, decisions of the follower robots are made by training a neural network. It inputs image features and distance from the head robot to move left, right, stop, forward, backward.

Considering the range of problems in [1], [2], [3], [5] this paper tackles the challenges of an automatic guided robot by using ArUco markers to continuously correct the position and orientation of the robot with live streaming video data. Since there is an overhead camera the task of localizing the robot with landmarks is removed. The importance of evaluation of this system is realized from [4] and thus, tests have been carried out to show how reliable it is.

III. SYSTEM OVERVIEW

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

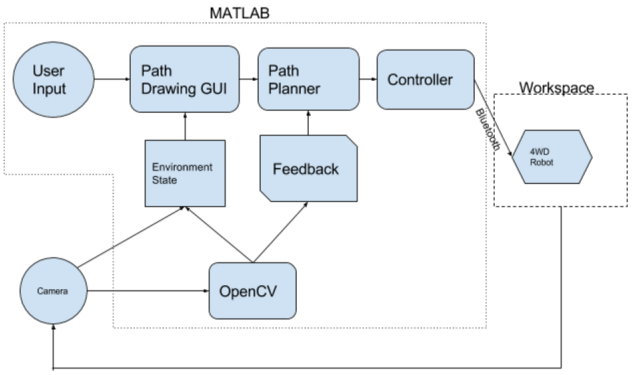

Fig. 1. System Work Flow

A. 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. 2.

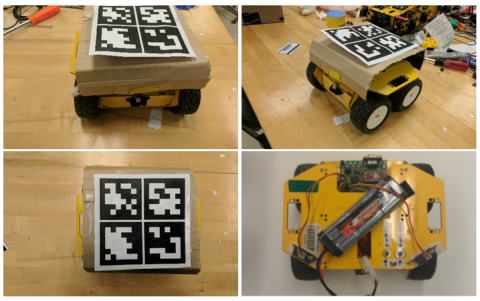

Fig. 2. 4WD Rover I from Lynxmotion

In this type of system each pair of wheels can be controlled independently from another pair. Therefore, this allows the robot to rotate about its center, which gives it greater control and speed on sharp angled turns. To control these motors, pulse width modulation (PWM) signals are given to each motor. This type of signal is usually at a fixed frequency and is a digital square wave ranging from 0V to V_{max}. V_{max} for the specific application performed was 7.2V. The signal determines on and off time for the motor based on its duty cycle. A duty cycle of 50% will result in the motors being powered at half their capacity. A duty cycle of 10% will result in the motors being powered at a tenth of their capacity. A duty cycle of 100% will result in the motors being powered at full capacity which looks like a DC signal at 7.2V. These duty cycles are shown in Fig 3. To control the duty cycle, an Arduino Uno was used which is the main controller used for the four-wheeled robot. A value of 0 to 255 was used to set the duty cycle between 0 and 100%. In this case 0 would represent 0% or full stop, 128 would represent 50% or half speed, and 255 would represent 100% or full speed.

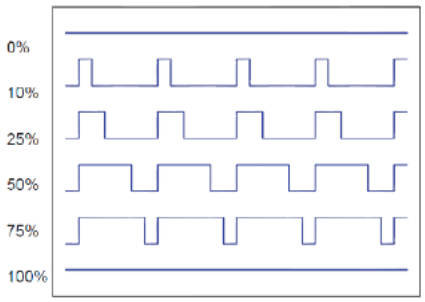

Fig. 3. Various duty cycles on PWM signals

This robot was programmed through the Arduino to take in serial commands to determine motion. Single character commands were used as shown in Table I. To take in these serial commands, a Bluetooth module was connected to the Arduino's TX and RX lines. The module used was the HC-06, which is a low-power slave Bluetooth module. Once

the Arduino receives a movement command it sends the appropriate PWM signals to the Pololu Dual MC3926 motor controller board which was wired directly to the motors, battery, and Arduino Uno.

TABLE I
MOVEMENT COMMANDS

Command	Function	Wheel Movement
F	Move Forward	All wheels turn in same direction, forward
L	Turn Left	Left side wheels reverse, right side wheels forward
R	Turn Right	Right side wheels forward, left side wheels reverse
S	Stop	Duty cycle of all wheels set to 0%, no movement

B. OpenCV

OpenCV is a comprehensive computer vision toolbox used for a variety of real-time applications such as tracking, facial recognition, segmentation, and classification. It is originally written in C++ but wrappers have been made to allow it to be used in Python and MATLAB. For this system, the only component of OpenCV being used is video stream capture and the ArUco Marker detection. ArUco markers are pattern-based black and white squares that can be easily identified through OpenCV. Since OpenCV knows what patterns to look for, it can identify ArUco markers in an environment and determine the corners of them as well as the orientation. For this application four ArUco markers were used as shown in Fig. 4. This marker array was printed to the size 14.5 cm by 14.5 cm and was 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 as a reference to determine the orientation of the robot.

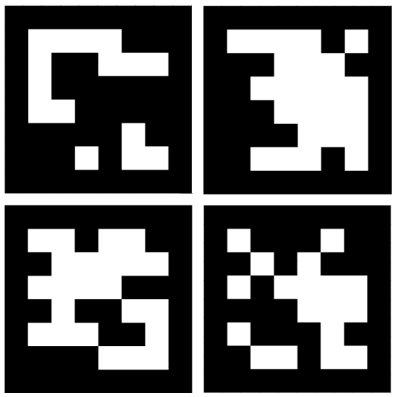

Fig. 4. ArUco Array of four markers

C. MATLAB

MATLAB is a powerful computing platform. It uses a scripting based language which is ideal for fast prototyping. It has also been built to be computationally efficient especially in terms of matrix math and neural networks. In this system, MATLAB is used to create the main GUI, interface with OpenCV, interface with the camera, path creation and planning, and sending serial commands to the robot via Bluetooth.

D. Environment

The environment that the robot resides in can be variable. However, the surface on which the robot roams needs to be navigable by the robot chassis. Also, the environment needs to be well lit so that the markers can be detected. 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. 5. The webcam itself 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). The robot was placed within this static workspace. The webcam was connected to the computer running the MATLAB application through USB.

Fig. 5. Webcam used for tracking

IV. PROCEDURE

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 workspace. The robot is then placed within that workspace with its ArUco markers. 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:

- The robot faces desired point
- The robot goes towards point
- Feedback from OpenCV to determine whether or not robot meets success thresholds
- The robot's movement is adjusted to meet threshold if needed
- 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 6.

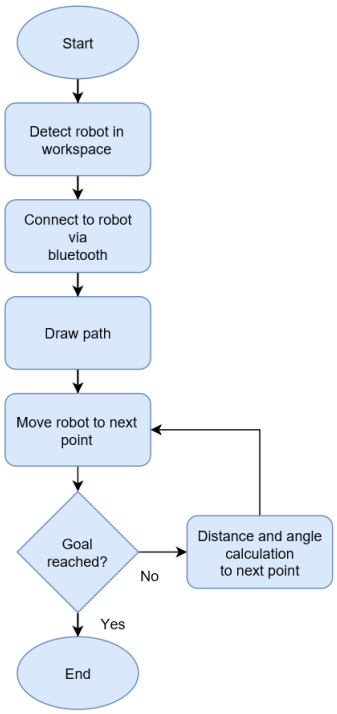

Fig. 6. Basic Flowchart of Procedure

A. Detecting ArUco Markers

To detect the 4 ArUco markers on the robot, the ArUco marker library within OpenCV was used. This library contains a function *detectMarkers* which takes in an image (that contains the markers), a dictionary of marker definitions, and some optional parameters. It then returns the corner locations and ID numbers for all markers detected. Once the corners and IDs are found, as shown in Fig. 7, the center point for the array is also found. This is done by using a centroid

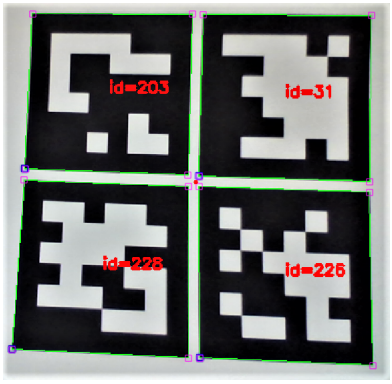

Fig. 7. Annotated ArUco Array of four markers

function that uses the points of the 4 corners closest to the center of the image. This is shown in Equation 1 and 2 where n is equal to 4 to represent the four points used. This center point is used to determine the center of the robot, which is also its turning axis.

$$center_x = \frac{\sum_{k=0}^{n} x_k}{n} \tag{1}$$

$$center_y = \frac{\sum_{k=0}^{n} y_k}{n} \tag{2}$$

B. Drawing a Path

Once the robot and the markers are detected in the frame of the workspace the user is prompted to draw a path. This is done by using the built in *imfreehand* function on the current image of the workspace. This allows the user to freely drawing any path within the workspace as shown in Fig. 8. Once the user draws the path the path is represented by a

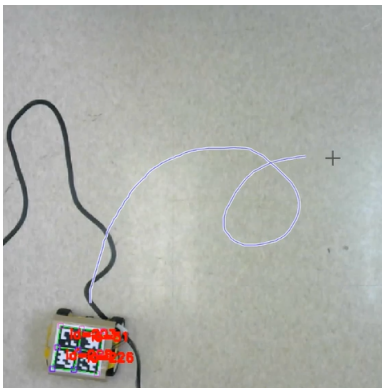

Fig. 8. Path Drawn by user in GUI

solid red line with a white star at its first point and a yellow star at its end point.

C. Accuracy Thresholds

To ensure the robot follows the path accurately, thresholds were created for the allowable distance it was from the desired point and its orientation with respect to the next point. To identify the distance in pixels between the desired point and the robot, denoted as d, the euclidean distance between the desired point and the center point of the ArUco marker array was found. This equation is shown in 3, where x_c and y_c are the pixel coordinates for the center of the robot, and x_g and y_g are the pixel coordinates for the desired point (current point on path trying to be reached). If the value of d was less than the threshold (by default was 20 pixels) then the robot met the distance criteria.

$$d = \sqrt{(x_c - x_g)^2 + (y_c - y_g)^2} \tag{3}$$

The second criteria was the orientation of the robot to the next point. This was important to meet so that the robot was always following the trajectory of the path. To do this 3 vectors were found and plotted. The first vector was from the center of the robot to the desired point (V_g). The second vector was from the center of the robot to the front of the robot between ArUco markers 31 and 203 (V_o). The third vector was from the front of the robot to the desired point (V_n). These vectors are shown in Fig. 9.

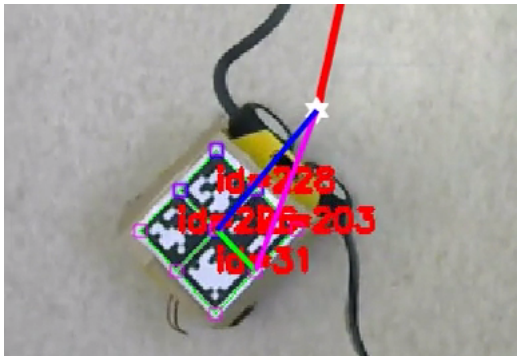

Fig. 9. Setting Robot Heading Direction: white star indicates desired point; green indicates current heading direction between center and front of robot (V_o); blue indicates desired heading direction between center and desired point (V_g); magenta indicates difference between the directions which is the vector from the front of the robot to the desired point (V_n)

To use these vectors to modify the robot's current heading, they were first normalized by dividing the vector elements (x and y) by the magnitude of the vector. The dot product was then taken between V_o and V_g as shown in Equation 4. Where a is a value between 1 and -1. The desired orientation value of the robot is 1 which means that the robot's heading is facing towards (parallel) the desired point. At -1, the robot is facing the opposite way of the desired point. An orientation threshold was defined to be close to 1 to make the robot more accurate in path following.

$$a = V_o \cdot V_g \tag{4}$$

The a value gives us information on heading but does not tell the optimal way for the robot to turn. The system could have been setup to turn left every time the heading was less than the threshold until the orientation threshold was met, but that would not be optimal. To determine the direction the robot should turn to make the most optimal decision to meet the orientation threshold Equation 5 is used. Where the two parameters of the *atan2* function are the *sin(θ)* and *cos(θ)*. This function returns the four quadrant inverse tangent that will give a θ value between 180° and -180°. This gives us the exact angle between the V_g and V_n vectors to determine which direction to turn. Since this angle is in reference to the current heading of the robot, a positive θ value will mean that the robot needs to turn left, and a negative θ value will mean that the robot needs to turn right.

$$\theta = atan2(V_{g_x} * V_{n_y} - V_{g_y} * V_{n_x}, V_{g_x} * V_{n_x} + V_{g_y} * V_{n_y}) \tag{5}$$

Therefore, using the values d, a, and θ the robot's movement is altered using Algorithm 1. To ensure that the robot has enough time to act before another command is sent to it the movement commands were only sent once a certain time threshold was met. Therefore, some movement commands were skipped to allow the robot's movement to be smooth and to ensure that the serial buffer on the robot would not be overflown.

Algorithm 1 Robot Next Move

```
if d < distanceThreshold ∧ a > orientationThreshold then
    desiredPoint ← nextPoint
else if a < orientationThreshold then
    if θ > 0 then
        TurnLeft
    else
        TurnRight
    end if
else if d > distanceThreshold then
    MoveForward
end if
```

D. Following the Path

Using Algorithm 1, the robot went to each point on the path drawn by the user. As it did this, the center point of the robot was plotted every time a point was reached. Using this data the, the overall error of the physical robot path was determined. Fig. 10 shows a snapshot of a robot following a path.

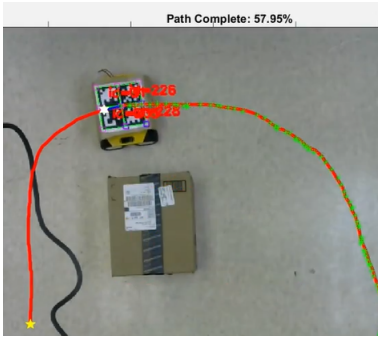

Fig. 10. Robot following the Path: white star indicates next point to reach; green markings show robot's past positions, yellow star indicates goal

V. RESULTS

To determine the overall accuracy of the robot, error between the physical path the robot took and the actual path was measured. This accuracy was improved by changing the distance and orientation thresholds for the system. At the end of each path taken, the error magnitude for each point is displayed as well as the standard deviation of error, mean error, maximum error, minimum error, and estimated error. The estimated error is defined as the error that will be found 95% of the time or two standard deviations (σ) away from the mean (μ), as shown in Equation 6.

$$Est.Error = \mu + 2 * \sigma \tag{6}$$

Three trials were tested with varying threshold values. Fig. 11, 12, and 13 show several paths with their corresponding error charts.

As shown in these figures, the robot's path seems to follow the path drawn by the user. Table II shows these results in

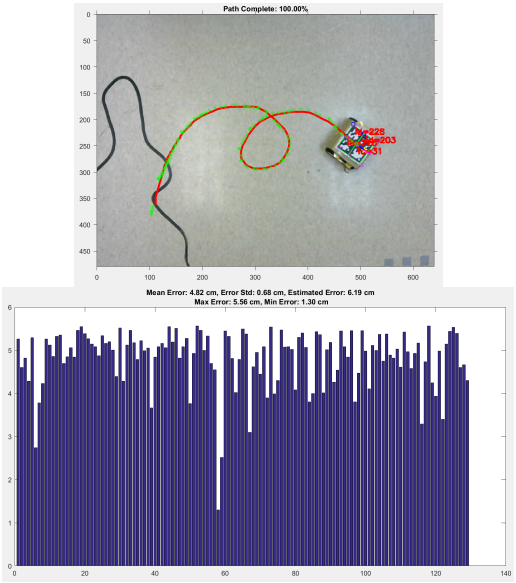

Fig. 11. Results for robot motion with distanceThreshold = 20px and orientationThreshold = 0.95

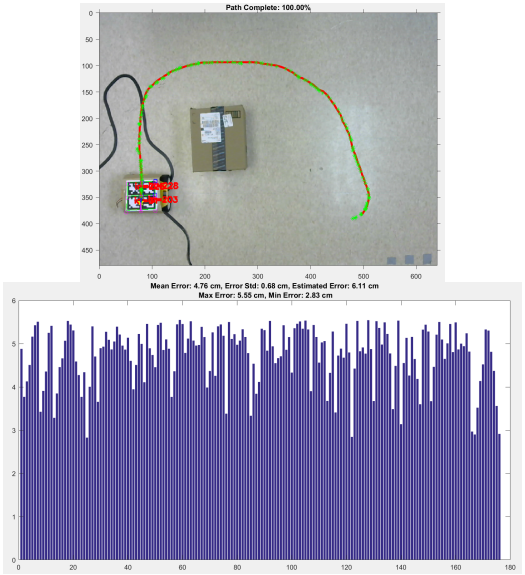

Fig. 12. Results for robot motion with distanceThreshold = 20px and orientationThreshold = 0.99

terms of standard deviation of error, mean error, maximum error, minimum error, and estimated error. Each of these values is converted to centimeters from pixels by using the known size of the workspace and the ArUco marker array.

TABLE II
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
a^*= orientation threshold, d^* = distance threshold

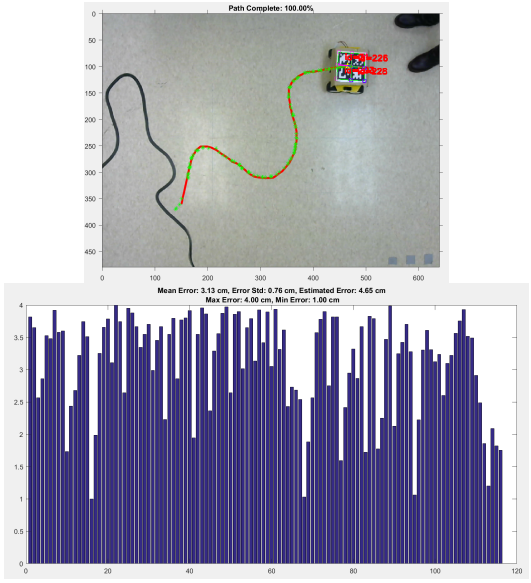

Fig. 13. Results for robot motion with distanceThreshold = 15px and orientationThreshold = 0.99

As shown, 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.

VI. CONCLUSION AND FUTURE WORK

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 improvement to the system can be to smooth out the user's path drawing so that the robot can take a smooth path without the having to adjust to the user's hand drawn line that will not necessarily be smooth. Currently the workspace of this system is limited to an area 1.8 m by 1.4 m. To increase the area covered by the system, multiple cameras can be installed. The ability to control a robot over a larger area would help using such a setup in the industry. Using this system continuous assessment of the system would also be possible. Another possible advancement would be to control more than one robot in the workspace.

REFERENCES

[1] L. Li, Y. H. Liu, M. Fang, Z. Zheng and H. Tang, "Vision-based intelligent forklift Automatic Guided Vehicle (AGV)," 2015 IEEE International Conference on Automation Science and Engineering (CASE), Gothenburg, 2015.

[2] X. Gao, J. Wang and W. Chen, "Land-mark placement for reliable localization of automatic guided vehicle in warehouse environment," 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), Zhuhai, 2015.

[3] S. Butdee and A. Suebsomran, "Automatic guided vehicle control by vision system," 2009 IEEE International Conference on Industrial Engineering and Engineering Management, Hong Kong, 2009.

[4] C. Roehrig, A. Heller, D. Hess and F. Kuenemund, "Global Localization and Position Tracking of Automatic Guided Vehicles using passive RFID Technology," ISR/Robotik 2014; 41st International Symposium on Robotics, Munich, Germany, 2014.

[5] Jae-Ho Lee, J. Uk-Jin and Youn-Sik Hong, "Indoor navigation for an automatic guided vehicle with beacon based relative distance estimation," 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), Vienna, 2016.

[6] R. Bostelman, T. Hong and G. Cheok, "Navigation performance evaluation for automatic guided vehicles," 2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA), Woburn, MA, 2015.

[7] R. Carelli, J. Santos-Victor, F. Roberti, S. Tosetti, Direct visual tracking control of remote cellular robots, Robotics and Autonomous Systems, Elsevier, Lisboa, Portugal, 2006.

[8] S. Mandal, S. K. Saw, S. Maji, V. Das, S. K. Ramakuri and S. Kumar, "Low cost arduino wifi bluetooth integrated path following robotic vehicle with wireless GUI remote control," 2016 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, 2016.

[9] H. Pangborn, S. Brennan and K. Reichard, "Development and applications of a robot tracking system for NIST test methods," 2013 IEEE Systems and Information Engineering Design Symposium, Charlottesville, VA, 2013.

[10] H. Zhang, L. Zhang and J. Dai, "Landmark-Based Localization for Indoor Mobile Robots with Stereo Vision," 2012 Second International Conference on Intelligent System Design and Engineering Application, Sanya, Hainan, 2012.