Developing Vision-enabled Autonomous Driving Capabilities For Competitive Robotics

Natnael Kelkay
Montgomery Blair High School

Gordon Franken
Intelligent Automation Inc

Justin Hudis
University of Maryland

Abstract

Our project focused on the capabilities of Limelight; an easy-to-use smart camera for the FIRST Robotics Competition (FRC). We developed code to generate a path for a robot given a target. We started by tuning parameters for computer vision-based target detection and characterizing the performance of the Limelight's target detection. This procedure involved receiving feedback from the Limelight by locating the vision target's position. We did this by using a mini-robot connected to the Limelight, and by collecting live feedback through a dashboard called Shuffleboard. With Shuffleboard, we were able to record and export data from the Limelight as a Comma-separated values (CSV) file. We calculated the mean and the standard deviation of each dimension measured to determine noise and bias. We then used linear regression to examine the effects of the target's distance from the Limelight in each dimension on the resultant error. The next step for this project is implementing the vision pipeline to go from target detection to path planning/following. We have written code for the path generation that is currently under testing and development.

1 Introduction

In FIRST Robotics Competition (FRC), an international high school robotics competition, vision is a big part of the game. Each year there has been consistent use of vision targets for the robots to use for the game challenges. This research project presents a way to improve teams' capabilities in the areas of computer vision and autonomy. As FIRST started moving away from the pure autonomous period, they switched towards vision targets. Recently, there is a relative benefit of vision-based autonomy versus remote operation via camera, which highlights the need to improve our vision and autonomy capabilities. The object of this research is to develop a program that will allow a robot to generate a path on its own, given a target. There are many methods to calculate a robots forward and inverse kinematics. The literature on Introduction to Autonomous Mobile Robot covers different topics, but the topics relevant to our research are Mobile Robot Kinematics and Planning and Navigation. The paper provides a method for calculating the forward kinematics of a robot and suggests how to apply that to a control software for an instance of mobile robot hardware. Our work also emphasizes the importance of understanding the mechanical aspect of the robots behavior before writing any code. It helped define the various methods we could use during the start of this project. The article on Reactive Path Deformation for

Nonholonomic Mobile Robots covers methods on how to calculate paths for robots that depend on the path taken in order to achieve their targets. The core idea of their approach is to perturb the input functions of the system along the current path in order to modify this path (Lamiraux, pg 1). The article helped bring forth the idea of generating a basic low resolution path to the target, and having that path split up into different sections. Once the robot gets to the first section it will create a more detailed path and use it. When it gets to the second section, it will do the same until it reaches its target.

The vision targets we tested with are made of retro-reflective tape. For this research, we used two vision targets only a few inches apart in the space of a triangle. The camera we used for this research was a Limelight. It was created specifically for FRC robots which makes it easier for this research. The research required access to a robot and laboratory to conduct the tests. The experiment consisted of a visual target being placed at different positions to test the capabilities of the camera. We were able to conclude from the data how the magnitude of x, y, and theta affect the error of the Limelight's target detection and to what extent.

2 Methodology

2.1 Tuning Parameters for Computer Vision-based Target Detection

The Limelight vision pipeline is comprised of five tuning tabs: input, thresholding, contour filtering, output, and 3D. Thresholding was the main focus for conducting the experiments—it included variables such as hue, saturation, and value that allows us to throw away any pixels that were not in a specific color range. The result of thresholding is generally a one-dimensional image in which each pixel is either "on" or "off." After tuning the hue, saturation, and value, a clear image of the vision target is visible for conducting experiments.

2.2 Performance Characterization of Target Detection

2.2.1 Initial configurations and finding the Limelight's vision range

We first collected data from the Limelight to identify the best position for accurate vision detection. In order to get readings from the Limelight, we had to connect it to a working robot. The robot we used is called Robot-in-a-box (RIAB). It was developed by the Montgomery Blair High School robotics team. Once the Limelight was connected to the robot, we opened a software called Driver Station. Driver Station station is a software that must be used to drive FRC robots. After the limelight was running we used our own vision target to see how far and to what extent it could detect the target. We tested every position possible around the limelight and the range it could pick up the vision target was in a cone shape.

2.2.2 Experimental setup

The vision targets are two reflective strips taped on to a wooden platform as shown in figure 1. One pair of the vision target is 5 in. (14 cm) long by 2 in. (5 cm) wide strips of 3M 8830 Scotchlite Reflective Material. Strips are angled toward each other at 14.5 degrees with a tolerance of approximately 1 degree in respect to the part to which it's adhered (FRC 2019 Game Manual).

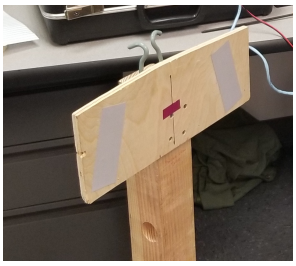


Figure 1: Vision target with retro reflective tapes

The setup for this experiment is as seen in figure 2. We attached the robot-in-a-box to the Limelight and computer and laid it out on a table. The Limelight was attached to the edge of the table and shined toward the vision target. On the floor we added marks each equally spaced out 12 inches from each other. As shown in the image below, there were about 29 possible testing locations.

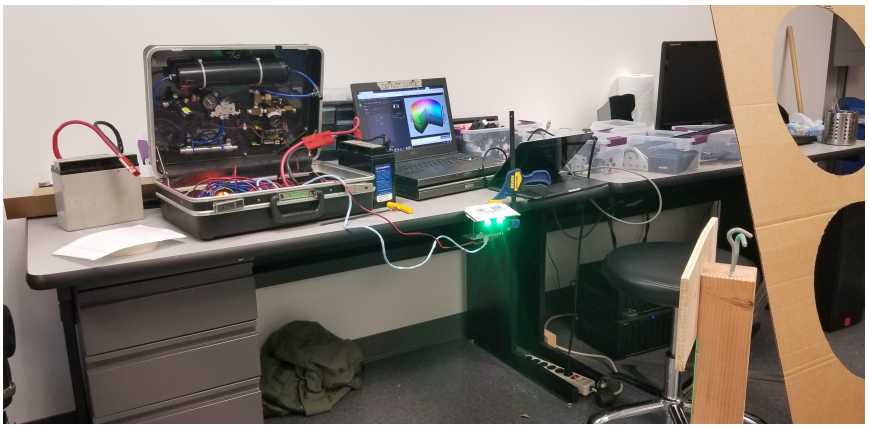


Figure 2: Experiment setup with the Limelight and the vision target

We also had degrees measured on a paper to align the vision target when testing. We Included 0º, 10º, 15º, 20º, 30º, 40º, and 60º in the testing. We ran into problems with some software issues that did not allow detection when the Limelight was turned to the right, so all of the data was collected by turning the Limelight to the left.

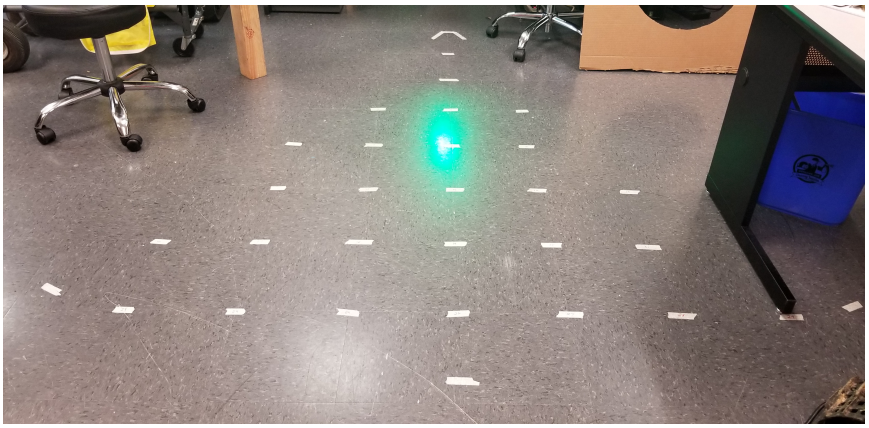


Figure 3: Limelight's cone/range of detection. The marks on the floor indicate the possible vision target detection positions for the limelight.

2.2.3 Experimental variables and ranges they were sampled

Through the Shuffleboard, we are able to receive live feed back from the network tables of the Limelight. One of the variables we recorded was the camtran—It stores the results of a 3D position solution, 6 numbers: Translation (x,y,y) and Rotation (pitch,yaw,roll). The only data points that were relevant to this characterization were the x, y and yaw. The x, y, and yaw give us delta x, delta y, and delta theta relative to the Limelight and the vision targets. Other critical variables were the time stamps. The time stamps were laps on a timer the indicated the data we want to examine.

2.2.4 Raw data was collected

The PnP point-based pose estimation given by the Limelight's pipeline, made it easy to obtain the x, y, and theta values. The Shuffleboard had a recording feature, so once the Limelight was connected and running, the shuffleboard was able to record the values we wanted—specifically the camtran. During the experiment, we placed the vision targets on the 29 testing locations one by one for about 2 seconds each. To account for the timings for each location, we used a timer. Whenever we had a good detection signal from the limelight, we would lap the timer to indicate that the 2 seconds after was the data we wanted to use for analysis. For each location, we recorded 7 times for each degree, so that was a total of 14 seconds of data for each location.

2.2.5 Raw data was post-processing

We exported the recording as a comma-separated value (CSV) file and uploaded it to an Excel file. Since the raw data given had many data points, we had to open it as an Excel file and filter out the camtran values. Once that was done, we used Python to do the data calculations. The first task before running the code was to convert the timestamps from seconds to milliseconds (since the shuffleboard stores time in milliseconds). Then we

stored the x and y values of each position/location in order (in feet X is lateral distance, Y is distance in front). The tested angles of the target were also stored in variables.

The database was split up into two sections, one with outliers and the other without. We did this to compare and contrast the difference between the margin of errors from the data with and without the outliers. The outliers that were omitted for the no outlier data were generally the ones from distances far from the vision target—it was either inconsistent or undetectable.

2.3 Code Testing setup

The vision code we used to generate a path was written on a computer called Nvidia Jetson model TK1. The code was written in C. We connected the Jetson (figure 4) to the robot so the both platforms can communicate.

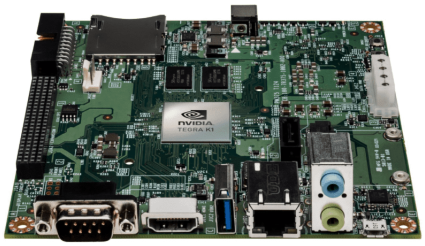


Figure 4: Jetson TK1—used to store and run path generation code written in C

The robot code runs the execution code while the Jetson runs the code for the path generation. To access the code in the Jeston we had to use a monitor (figure 5). When the code was ready there was no need to use the monitor because all the code can be accessed through the Jetson.

The basic set-up is shown in figure 5. To connect the robot to the Jetson we used an ethernet cable, and to connect the robot to the computer we used wireless connection through a radio, but wired connection was open as a possibility.

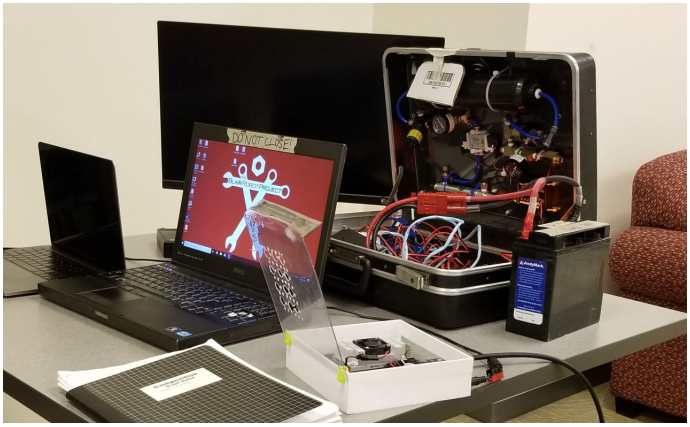


Figure 5: Code communication setup. The Monitor is connected to the Jetson, the Jetson is connected to the robot, and the robot is connected to the laptop.

3 Results

3.1 Analyzing mean and std deviation to determine noise and bias

The mean and standard division was calculated by time intervals we set for each test. Going back to figure 3 (Limilght's cone) we can see the 29 spots the decision target was placed to test the limelight. For each point, determined the angel the vision target was at, the delta x from its poison to the vision target, and the delta y from its position. For every position test, we only considered the first two seconds after it was placed on the position. Because of the abundance of data, we took the averages and standard deviation for those time intervals. The standard deviation told us how far away the actual value was from the mean. Having that figure allowed us to create another dataset without the outliers. We took out all the standard deviations that were too high and placed the filtered data into a new sheet.

3.2 Using linear regression to examine the effects of the measurement variables on resultant error

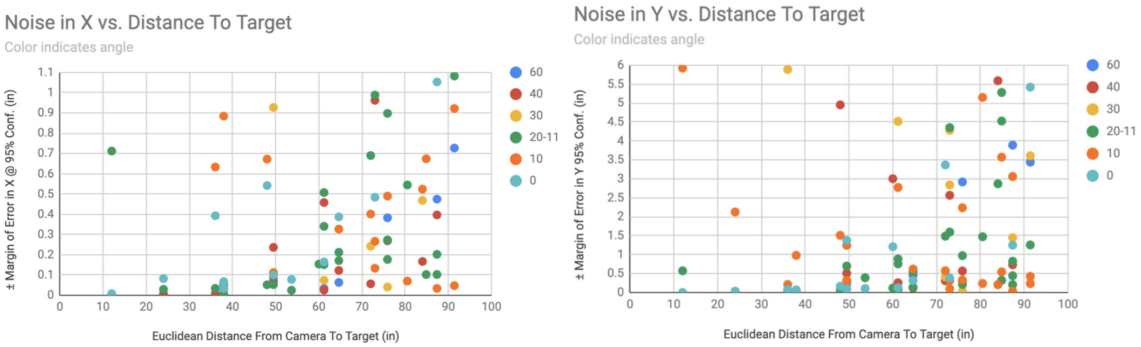
Linear Reg Coefficients							
With Noisy Outliers				Without Noisy Outliers			
	Abs. X Error	Abs. Y Error	Abs. Theta Error		Abs. X Error	Abs. Y Error	Abs. Theta Error
Abs. X	-0.08	0.06	-0.27	Abs. X	-0.41	0.01	-0.41
Abs. Y	0.31	0.08	0.20	Abs. Y	0.46	0.14	0.24
Abs. Theta	0.68	0.15	0.11	Abs. Theta	0.84	0.18	0.07

Figure 6: Table of the linear regression coefficients for the errors in the x, y and theta axis

Using linear regression, we set out to determine how efficient the limelight was. The linear regression coefficients with the noisy outliers data show that most of the values are greater than 0. This means that as the vision target goes further away in the x and y plane, the detection gets less accurate. For the absolute x value, the values for Abs. X Error and Abs. Theta Error was negative in both cases (with and without outliers), which indicated that the detection got better when the vision target was further from its original positions.

3.3 Data Analysis

In figures 7a and 7b, you can see the graph of the margin of error in x and y versus the Euclidean distance from the camera to target both figures in inches. The blue dot represents the data points collected for 60^o to the left. The red dots represent the data points collected for 40^o to the left and so on. With these graphs, we are able to confirm the conclusion that the Limelights detection works best when the vision targets are closer to it. The graph has an increasing curve trend. As the distance the vision target was from the limelight



(a) Noise in X vs. Distance to target

(b) Noise in Y vs. Distance to target

Figure 7: Graphs of the errors in the x and y axis of the vision target to the Limlight

increased, the margin of error in both x and y also increased. There is a cluster at the bottom of both graphs which is a good thing because that means the margin of error is not too high for most data points. There are also several of outliers, some even off the chart.

The graph for the vision target's angle from the limelight's detection (figure 8) also shows a similar trend. There is the same cluster at the bottom of the graph and the few outliers above the cluster. The x, y, and theta error graphs being similar was a good indication that there was only one variable or factor that we needed to worry about. From the analysis of the linear regression's coefficients we concluded that as the vision target gets farther away from the limelight, the margin of error in x, y, and theta increase.

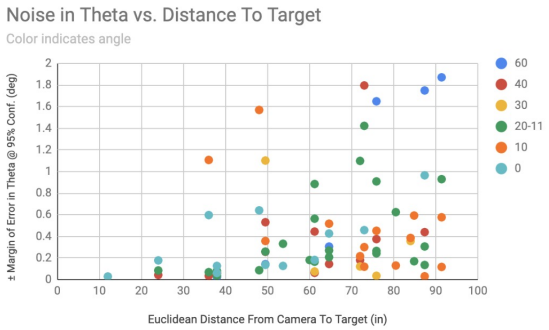


Figure 8: Noise in theta vs. Distance to targe

4 Discussion

Our research project will help improve FRC robotics vision capabilities. It will allow the robot to get a better reading from the limelight and have it better characterized. We are still testing code and will have a product that will allow the robot to generate a path on its own and travel to the target position on its own without having to rely on the driver. We also plan to expand on this by applying advanced computer science topics like machine learning to it. With machine learning, the robot will be able to not only generate a path and travel to it, but if it encounters an obstacle, it will generate a path to reach the goal. In addition to the path generator. We plan to create a way for the robot to detect other robots.

Detecting other robots will allow the robot to avoid them and easily pass through them. One aspect of an FRC game is defense. If the other team has a strong defense, having this detection capability will allow a robot to maneuver through them.

References

[1] Siegwart, R., Nourbakhsh, I. R. (2004) *Introduction to Autonomous Mobile Robots*. United Kingdom: Bradford Book.

[2] J. Borenstein and Liqiang Feng *IMeasurement and Correction of Systematic Odometry Errors in Mobile Robots*. in IEEE Transactions on Robotics and Automation, vol. 12, no. 6, pp. 869-880, Dec. 1996.

[3] F. Lamiraux, D. Bonnafous and O. Lefebvre *Reactive Path Deformation for Nonholonomic Mobile Robots*. in IEEE Transactions on Robotics, vol. 20, no. 6, pp. 967-977, Dec. 2004.

[4] Limelight *Limelight Documentation (2017, October 21)*. Retrieved from http://docs.limelightvision.io/en/latest/.

[5] FIRST *FRC 2019 Game Manual. (2019). FIRST FRC*. Retrieved from https://firstfrc.blob.core.windows.net/frc2019/Manual/2019FRCGameSeasonManual.pdf