High performance Vision Tracking

(An alternative to Limelight)

By Bill Kendall
Reviewed and augmented by Team 4499, The Highlanders

Preface

The very first thing the reader should understand is, the purpose of this paper is not trying to denigrate Limelight in any way whatsoever. At the time of this writing, Limelight sets the standard that all vision tracking systems are measured against.

The purpose of this paper is to simply provide an alternative to using Limelight that meets the following goals: 1) The same 90 FPS image acquisition as Limelight, 2) The same, or better, NetworkTables update rate as Limelight, 3) Similar image acquisition to NT update latency, 4) Fully user built/written pipelines, 5) < $200 price tag.

All of these goals are achievable! That said, at the time of this writing, the Global Chip Shortage has made acquisition of the components used a bit difficult, but not impossible as can be seen below.

The base hardware for this system is comprised of:

1)Raspberry Pi 4 with at least 2GB of ram. (Many alternative configurations will work.) *
2)Raspberry Pi Camera v. 2.1 (Other CSI options can work)
3)8+GB Micro SD card
4)2 or more 3W LEDs (Green is preferred but not mandatory) Other options may work as well.
5)DC to DC Constant Current LED driver.

** **Please read the supplemental information in the rear of this document. After review of this document by a couple EE's on Team 4499, The Highlanders, some of their suggestions will be added there.**

Table of Content

Supplement based on feedback from 4499, The Highlanders

Introduction

As mentioned already, the goal of this paper is to provide a vision tracking alternative to Limelight that has similar performance. Limelight is being referenced as the standard because, at this time, Limelight is the highest performing option currently available. Much of the inspiration for this project comes from Limelight.

There are two differences in the approach that will be covered in this paper to the traditional USB Web Camera based tracking system. These two differences allow the performance improvements needed to match the Limelight performance levels. The first is the use of a “Raspberry Pi Camera”, or one of the many CSI camera alternatives available. The second is using multi-threading to allow the camera to run at full speed without blocking. Blocking is what happens when one software process has to wait until another process completes. For example, the image processing needs to wait until the image acquisition process completes.

When it comes to coding, Python, OpenCV and PyNetworkTables will be covered. This combination has been chosen simply because that is what the author is familiar with. Fortunately, Python bindings for OpenCV perform well enough that they can achieve the stated performance goals. PyNetworkTables is the mechanism used to transfer tracking data to the RoboRio.

Potentially, you may be able to achieve better performance by coding in C++, but performance gains achieved by C++ may also be negated by limitations in the sync rate of either NetworkTables or PyNetworkTables. By *default*, NetworkTables and PyNetworkTables sync to the server, the RoboRio, at 100ms and 50ms intervals respectively. PyNetworksTables can have this interval reduced, but this is not recommended. The tradeoff is at the risk of network bandwidth saturation. However, whether you use NetworkTables or PyNetworkTables, the fastest possible sync rate is achieved by simply sending all cached data once per processing loop. Both NT and PNT are rate limited to 10ms intervals. Thus, both can actually send as fast, if not faster, than the maximum frame rate attainable from the camera.

One last consideration, there is potentially a risk of data corruption, leading to loss of functionality, when the power to the RPi is suddenly removed as happens when the main breaker on the robot is turned off. This issue needs to be considered and addressed if possible. Options for addressing this will be presented in Ch. **5**.

Prerequisites

The first thing you will need is a Raspberry Pi 4. This can come in a few different configurations. Due to the current Global Chip Shortage, acquiring one can be a bit tricky. The recommended minimum configuration is a **Raspberry Pi 4B with 2GB of ram**. **
The platform used for validation of performance for this paper was a Raspberry Pi Compute Module 4 with 2GB ram, Wifi, and 32GB eMMC. Additionally, a CM4 IO board to mount the Compute Module to. These were selected because that was all that was available at the time this project was undertaken.

The next item you will need is a Raspberry Pi Camera. The v2.1 version is the recommended camera to use. It is capable of acquiring images at 640 X 480 @ 90FPS. There are several alternatives to this specific module. Some come with a larger lens, some not. Just make sure that whichever module CSI camera you choose, it is able to support at least 90FPS @ 640 X 480. (For example, a module with a Sony IMX219 sensor)

Once you have the RPi 4 and the camera, you will need to set up the RPi OS. At the time of writing, the latest RPi OS image, 64bit Bullseye, the default Camera support, libcamera, does not have Python bindings yet. It is anticipated these bindings will be available by the end of summer. With this in mind, there are at least two options. 1) Enable “Legacy Camera” support, then install Picamera. 2) Use an earlier stable Pi OS like Buster. This is the approach selected for this project. If Python bindings become available under Bullseye by the time this is published, an effort will be made to include the results of testing with those (“Picamera2”).

The next step is to install OpenCV. You can find several guides to follow with just a simple Google search. This guide by QEngineering was used. Please follow the path that covers your OS. https://qengineering.eu/install-opencv-4.5-on-raspberry-pi-4.html
Just be prepared, this process can take quite a while to perform.

** *After validating that the performance levels that are pointed out above are attainable with the configuration suggested, the same process was tested on a RPi 4 with **1GB** Ram. As it turns out, it too was able to meet the stated performance levels! That said, it does take quite a bit longer to set up than a SBC with at least 2GB of Ram.*

Consideration as to how you want to operate the Raspberry Pi on the robot network should be taken. ******Please see Supplement bullet #2***

Coding

Aside from the hardware, software is where the real magic takes place. It is your code that determines how fast you can acquire images. It is your code that identifies and tracks targets. It is your code that provides the targeting information to the Robot Control System so that actions can take place. All of these processes are fully in your control!

That said, there are other considerations you need to keep in mind when you develop your code. This paper will endeavor to address as many of these considerations as is reasonable. Code will be provided to help you get started. The hope is, **YOU** will take these as starting points and continue to develop above and beyond what this paper covers. ******Please see Supplement bullet #1***

An ideal exposure is one where the camera sees only the target you want to track. The closer you can get to achieving an ideal exposure, the better. Much can be done before any exposure is taken to help the camera achieve the ideal exposure. Currently FIRST is using retroreflective tape as Vision Targets. To take advantage of this, we need to use illumination to see these targets. The brighter, the better. For example, Limelight uses 6 green LEDs as a light source. You can do the same with variations you deem optimal. Different color, different LED count, different LED style are all options. Just keep in mind the game rules, and the fact that the light output needs to be consistent. No flickering or fading. Many of these options will be covered a bit more in Ch. **5**.

The first consideration is how to set the exposure of the camera. In addition to the ideal exposure only showing the desired target, the ideal exposure for target tracking will be as short as possible. This reduces or eliminates motion blur. There are a few different settings you can use to achieve the shortest exposure possible. Shutter speed and Camera ISO are the primary two that impact exposure time the most.

The next consideration is being able to separate the target information from the rest of the image content. The technique for target identification that will be covered identifies the target based on color, intensity, and shape. You will want to make certain the color stays consistent. Therefore, you will want to lock down the White Balance to prevent that process from changing the color values. So, manually setting color gains, specifically Blue and Red, will need to be performed. Additionally, automatic exposure settings can easily change the image generated in fractions of a second. This is good for photography and video, but not for target tracking in a FIRST Robotics competition. Thus, you need to be in control of these variables as well.

With these things in mind, to achieve the fastest performance, the recommendation is to run the image acquisition from the camera in a separate thread. This allows the camera to run at full speed without interfering with the image processing and tracking data upload processes. The Pi Camera is capable of acquiring 90 FPS at 640 x 480. USB cameras can not keep up with that level of performance. When you look at the Camera Server script, you can see how this “multi-thread” process is done. The image processing and data upload thread simply grabs the latest image from the acquisition thread and processes it. In reality, the image processing actually runs faster than the acquisition thread. Therefore, the same image is occasionally processed multiple times. This is far better than missing frames! This also means that a lot more targeting data can be extracted from the images and still not reduce the overall performance of the system. Take a look at all the additional information that is available from the Limelight camera. BE INSPIRED, and see if you can figure out how get all the information you need.

https://docs.limelightvision.io/en/latest/networktables_api.html

There are three Python scripts included with this project. The first one allows you to set the camera exposure, as mentioned above, and saves the values to a file. The second reads those values and uses them to set the camera exposure so that you can then calibrate the tracking values to the specific target(s). The targeting calibration values are also saved to a file. The third is the actual Camera Server script that will run during matches and sends the targeting data to the RoboRio. The Server script uses both the exposure calibration values and the target calibration values from the saved files. This allows tracking to start immediately when the robot is powered on.

The expectation is that these scripts will only be the basis on which you begin. The hope is that they will inspire you to go much further. That said, they will work as a minimum starting point for competition. ***_**Please see Supplement bullet #1**_**

Testing and Calibration

This section will be broken down into 3 main sections. Each section will follow one of the three included scripts. They will be covered in the order that they should be performed when calibrating the camera system for use in competition. It is HIGHLY RECOMMENDED that you get very familiar with each script, how it operates and what its purpose is. Doing so will help ensure you have a successful vision tracking system. ******Please see Supplement bullet #3 and #5***

1)"PiCamera Exposure Cal.py"

This is a non threaded, blocking script. FPS is really not important in this process. The purpose of running this script is to tune the Pi Camera's settings to get as close to the "Ideal Exposure" as possible.

Notice in lines 13-15 that the camera is preset to not use Automatic White Balance, preset for an ISO of 800, and finally, manual exposure. This allows the user to set an exposure that is as short as possible (ISO 800) and lock the exposure time, and set the color gains to optimal levels that will not change.

This script also includes a mouse click feature that will print out the GREEN value of the pixel that is selected by L-Clicking on the image. RED or BLUE values may be displayed if the user chooses to edit the code for that purpose. The purpose of this feature is to make sure the selected color is not at or above saturation. (255). The suggested value should be around 200-225. This leaves some overhead for variations. Calibrating for this value is done by adjusting the trackbar sliders. After each adjustment, verify the value of the selected pixel.

2)"PiCameraThreadCalTargets.py"

This script runs multithreaded. In this thread, the current FPS is printed out to help the user determine if the image processing code, plus displaying of the image, causes the system to run slower than the goal of 90FPS. This script currently does not include PiNetworkTables. It would be a simple process to add this feature. Just see the next script, PiCameraThreadingServer.py, to see how it is done.

In the "PiVideoStream" class, you will see that both AWB and Exposure modes are turned off. You will also see that the ISO is set. Make sure these modes and values are set the same as in the PiCamera Exposure Cal.py script. Additionally, note that the "framerate" is NOT defined. This allows the camera to free run at its maximum framerate.

This script also uses the mouse to help aid with the calibration. Just click the target in the image. This will set the Hue, Saturation, and Value sliders. You may want to modify the “CalWidth” value a bit, but it should be close enough to get you started. Just fine tune the Upper and Lower Hue, Saturation, and Value sliders to get the cleanest visualization of the target in the “Binary” image. These sliders are located on the “HSV” window.

Once you have a clean binary target, adjust the remaining sliders on the “HSV” window until the target is identified by a box around it and the crosshairs on it in the “Camera” window.

Depending on the target shape, size, and number of sides, you may find all the sliders will need to be adjusted. By examining the script’s code, you should be able to determine exactly what each slider’s purpose is. You also may want to modify these functions in a way that optimizes the code for your specific needs.

3)”PiCameraThreadingServer.py”

This final script is the one you will want to run during competition. There really is nothing that is needed to be done with this script in the way of calibration. Now, if you have modified either of the first two scripts, you will need to modify this one to match. For example, the IP address of the NT/PNT server will need to be set. The typical format for the IP address should be “10.TE.AM.2”.

This script will launch and then “wait” until it is able to reach the NT/PNT server. Once connection to the NT/PNT server is made, the script will continue. It will begin sending tracking data almost immediately. If no target is visible, the “targeting” variable will be sent with a value of “0” and “x” and “y” will remain at the previous value. Once a target is recognized, “targeting” will be set to “1” and the “x” and “y” will take on their new values. All three values will be sent every processing cycle, limited to every 10ms by NT/PNT.

To get this script to launch on boot up, a typical approach is to add a line to “rc.local”. The line would look something like:

“***python /home/pi/vision/PicameraThreadingServer.pi &***”

The “&” is important to include. It allows the rc.local file to continue without waiting for the python script to complete.

Options

Two optional items that were mentioned previously were:

1) UPS to prevent data corruption during power down.
2) LED to illuminate the target.

There is a possibility that the memory card, or eMMC memory, depending on which Raspberry Pi version you use, can be corrupted when power is removed without properly shutting down the Raspberry Pi. Therefore, it is highly recommended that precautions are taken to prevent this from happening. Using a UPS is likely the simplest option. The one that was chosen for testing purposes was the x728 v2.1 by Geekworm. Depending on which batteries are placed in it, it can easily keep the RPi running for more than 30 hours once power is removed. Once the batteries are drained, it will automatically shutdown the RPi even if you forget to do it. It does have a button onboard that can power off the RPi as well. Thus you have plenty of options available to you. ******Please see Supplement bullet #4***

The second consideration you will need to look into is; How do you plan to illuminate the target? With FIRST's current process, Retroreflective tape is used to define the Vision Target. Various shapes and sizes have been used in the past. Until FIRST chooses to do otherwise, such as using ArUco markers, a subject for another paper maybe, there currently is no reason to assume this pattern will not continue. Thus, you need to find a way to illuminate the target with consistently and sufficiently bright light. As mentioned before, the brighter, the better. There are MANY options for this. Color does not need to be green, but there are good reasons to choose it. For one, cameras are designed to respond to color similarly to the human eye. Therefore, they are usually most sensitive to green light. So, using green means you can attain shorter exposures due to this increased sensitivity.

To make sure the light is consistent in brightness, a Constant Current Drives should be used. If at all possible, the driver should also include a Buck/Boost circuit to provide stable voltage to the LED(s) even when the battery on the robot sags due to high loading.

Several options have historically been available from SparkFun in the past, and may be in the future. Due to the chip shortage, the supplies for both LEDs and Constant Current LED drivers from them has dried up. The same LEDs are still available from Amazon as are multiple options for driver/regulators. Links to a couple options will be made in the next section. Keep in mind, the LEDs linked below can get quite hot!

Parts list and links

How to install OpenCV https://qengineering.eu/install-opencv-4.5-on-raspberry-pi-4.html\

Limelight NetworkTables API
https://docs.limelightvision.io/en/latest/networktables_api.html

SparkFun
https://www.sparkfun.com/

25 Green 3W LEDs (non affiliate link)
https://smile.amazon.com/gp/product/B07PGRRD5N/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1

Automatic Boost Buck Converter Module (non affiliate link)

https://smile.amazon.com/gp/product/B00JKG57T4/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1

Raspberry Pi UPS module (non affiliate link)
https://smile.amazon.com/gp/product/B087FXLZZH/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&th=1

Supplement

This supplement is the result of feedback received from a couple of professional EE's on Team 4499 after they reviewed the original document.

1) The use of Python is sub-optimal. It is a convenient and easy language to use, but it does have its share of shortcomings. In reality, it does not actually perform Multi-Threading. That said, it can perform well enough to achieve the stated goals of this project. The quote from Team 4499: "*The python code seems a little naive to me, async functions would probably be a better solution than threading, they are still only using one OS thread due to the python GIL. This approach does manage to hide I/O delays probably as effectively but they could avoid the confusion over double processing frames if they rethought the threading model."* It is suggested that this code is just a starting point. Please consider optimizing the code and use another language, such as C++ to achieve optimal performance.
2) From Team 4999: "*Generally, we like to put devices such as a pi on a static IP so that we can easily access them via ssh (eg. 10.44.99.11)*...*When setting up the network remember that the FMS has ports 5800 - 5810 open for team use. These ports are super helpful for setting up debugging tools, http servers, etc. ...I would also consider changing the default SSH port on the PI to 5800. The FMS blocks port 22, so if you need to manually fix something before a match it is nice to have.*"
   While this is not a MUST DO, it is HIGHLY RECOMMENDED! Configure the Raspberry Pi with a static IP address! Typically 10.TE.AM.11 is used. While it is not in the scope of this document to give details how to do this, a quick Google search will return several guides such as this one:
   https://howchoo.com/pi/configure-static-ip-address-raspberry-pi
   As far as the FMS ports go, yes using these ports provide easy remote access. That said, practically every connection made to the Pi is performed on the robot. Thus, changing the ports is nice, but may not be necessary. YMMV.

3) Suggestion from Team 4499: "*I would highly recommend developing a network-based calibration tool that doesn't require the students to connect a monitor to the Pi. It makes things way easier at competition.*" This is a very valid point! Dragging a monitor to the field is a really difficult option.
   During development of this project, VNC was used extensively to remotely access the Raspberry PI. It not only allows remote access, it also allows file

transfers when needed. It is a simple process to add support for VNC..
https://www.realvnc.com/en/blog/how-to-setup-vnc-connect-raspberry-pi/

4) Depending on your reading of R602 (2022 Game Manual), adding batteries to a UPS may not be allowed. Please keep this in mind when choosing your option to protect the Raspberry Pi from possible corruption.

5) Last quote from Team 4499: "*It would be useful to layout a simple logging framework in the sample scripts. This will help to encourage students to use logging, which is super useful when the code inevitably crashes or fails for some weird reason.*" This is ABSOLUTELY a great recommendation! In fact, print statements were used extensively in the creation of these scripts, but removed once the code was working. Please feel free to add any type of logging or debugging processes you feel are helpful. It is just good practice!