Team 1100 Vision Guide

# Introduction

Welcome to the endless world of fun and joy that is FRC Vision! Each year, the game changes, which necessitates a change in the vision system of the robot. This is a comprehensive guide that should teach you the ins and outs of Team 1100 FRC Vision setup. With the introduction of Apriltags, the computer vision aspect of the competition varies less from game to game. However, a sound understanding of computer vision can open a variety of possibilities for autonomous routines.

# Team 1100 Vision Capability

As of 2023, the team 1100 vision program uses a Raspberry Pi to process both Apriltags and images based on color. The former can be used to locate the robot on the field, while the latter can be used to track game or field elements (a very limited form of object detection). When multiple field elements are detected, the closest element is chosen as the desired target. The program can handle up to two cameras but can only process feed from one camera at a time. This enables the user to switch between two different cameras facing different directions. The program also cannot process both Apriltags and game elements at the same time (yet).

# Vision Hardware Setup

Shown below is the vision setup for 2023 Infinite Recharge, which can be used as a reference for how to set up the hardware. The vision processing takes place on a Raspberry Pi. Instructions for configuring a new RPi can be found [here](https://github.com/Team1100/FRCRasPiVisionApp). Bothfigured, the RPi needs to be placed on the robot in a safe location. The both the roboRio and the RPi need to be connected to the radio with separate ethernet cables. The RPi can be powered via a usb connection from the roboRio. If a newer RPi is used (which may require more amperage than the roboRio can provide), power can be acquired from the \_\_\_\_ . The cameras can simply be plugged into the RPi’s usb ports.

* **Important note:** The cameras need to be separate models. If identical models are used, the vision program will be unable to differentiate the two.

# Connecting to the Raspberry Pi

In-depth details on how to connect to the Raspberry Pi can be found on [Team 1100’s Vision Repository](https://github.com/Team1100/FRCRasPiVisionApp). However, there are two main ways to control and edit the Raspberry Pi. The first is via a web interface, which provides a high-level control over cameras, vision feed, and a program log. The second is via an ssh connection which provides direct access to the RPi’s file system. Once the hardware has been set up, the robot can be turned on.

* **Important note:** Many resources and information can be found on the wpilib [documentation](https://docs.wpilib.org/en/stable/docs/software/vision-processing/index.html). I would highly recommend reading through many of those pages, as they provide a broad perspective on how the system goes together. This is vital for understanding the system at a more technical level, which will hopefully be provided here.

## Web interface

After connecting to the robot’s Wi-Fi, the web interface can be accessed by going to either <http://frcvision.local/> or <http://wpilibpi.local/>. Frcvision is the link used by older generations of the raspberry pi, while wpilibpi is used by newer editions. An overview of the web interface can be found [here](https://docs.wpilib.org/en/stable/docs/software/vision-processing/wpilibpi/the-raspberry-pi-frc-console.html).

## SSH Connection

To edit the python application contents, a command line interface is needed. In GitBash (or other similar linux terminals). If you know how to navigate a raspberry pi, it can be connected to by running this ssh command: ssh [pi@frcvision.local](mailto:pi@frcvision.local) (or wpilibpi.local). However, doing so requires a knowledge of navigating a Linux environment. Instead, a more user-friendly way to connect is through WinSCP (an app that can be installed through windows), or FileZilla for macOS, both of which provide graphical interfaces. Instructions for how to use these apps can be found [here](#_Using_WinSCP_&).

* Username: pi
* Password: 1100pies
* **Important note:** In order to edit via a ssh connection, the Web Interface needs to be open and the RPi needs to be set to Writable.

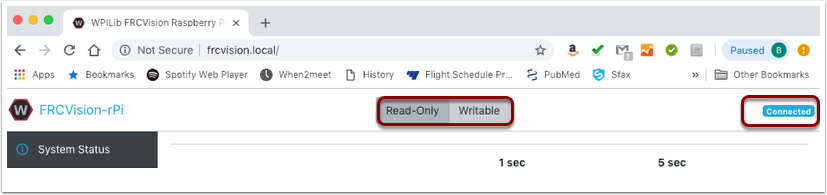
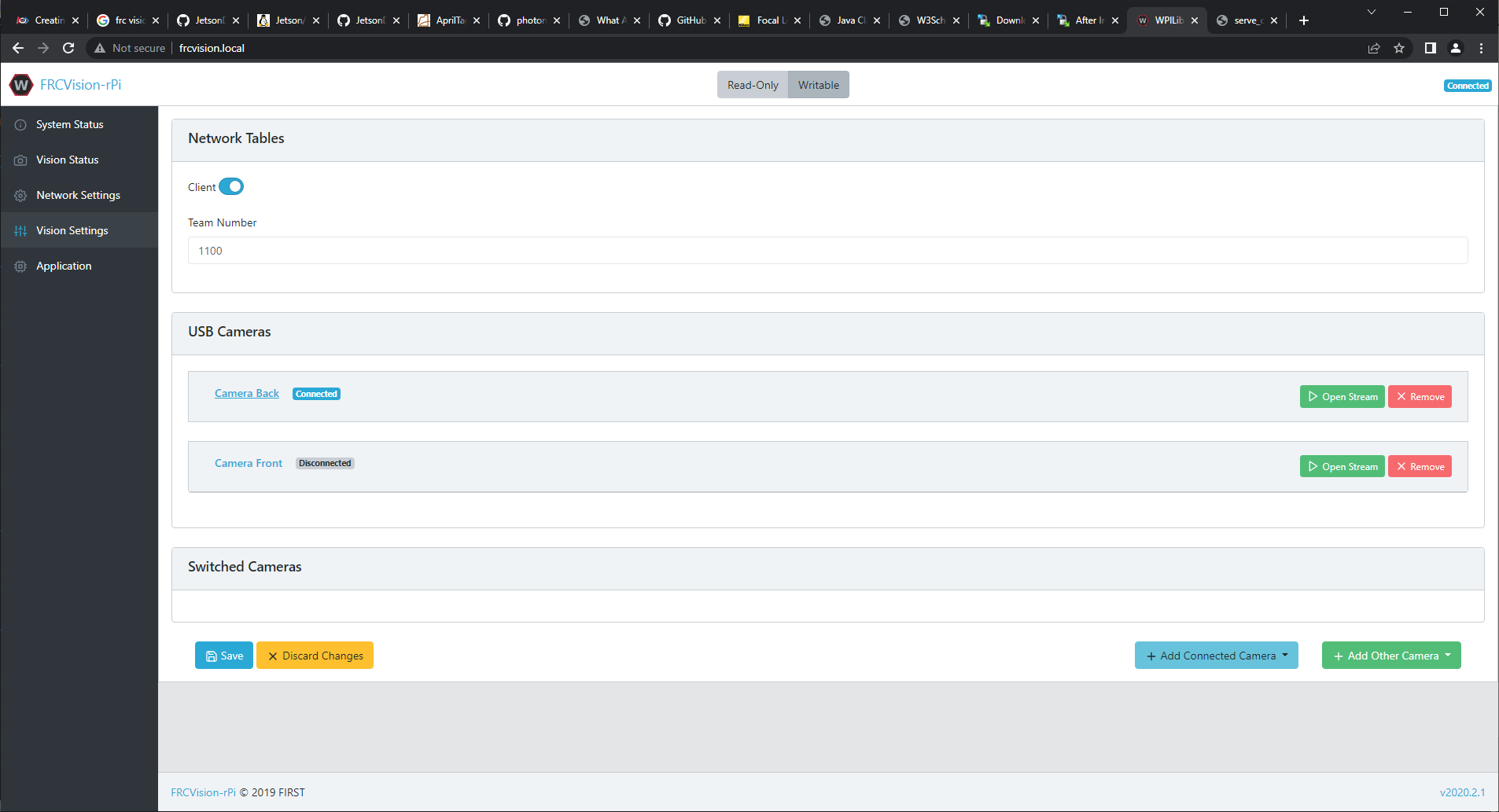


Figure : <https://docs.wpilib.org/en/stable/docs/software/vision-processing/wpilibpi/the-raspberry-pi-frc-console.html>

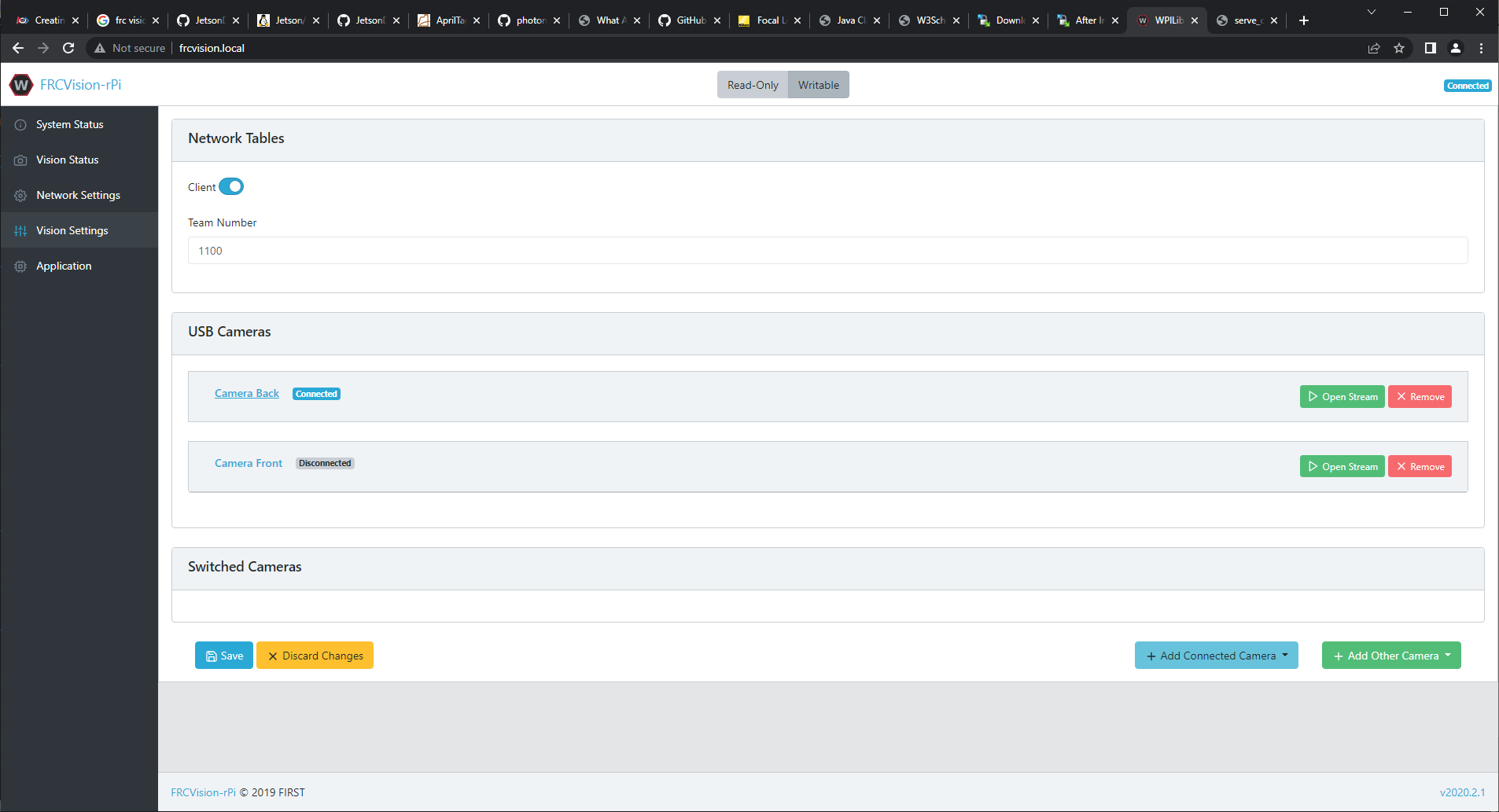
# Setting up Cameras

Make sure a camera is plugged into the RPi. If you are adding multiple cameras, it would be easiest to do so with only one camera connected to the RPi at a time. Open the Web interface and move to the **Vision Settings** page.



Figure

Make sure the team number is correct. If it needs to be changed, set the RPi to writable, change the number, and click save at the bottom.



Figure

Next (making sure only one camera is plugged in), click A screenshot of a computer

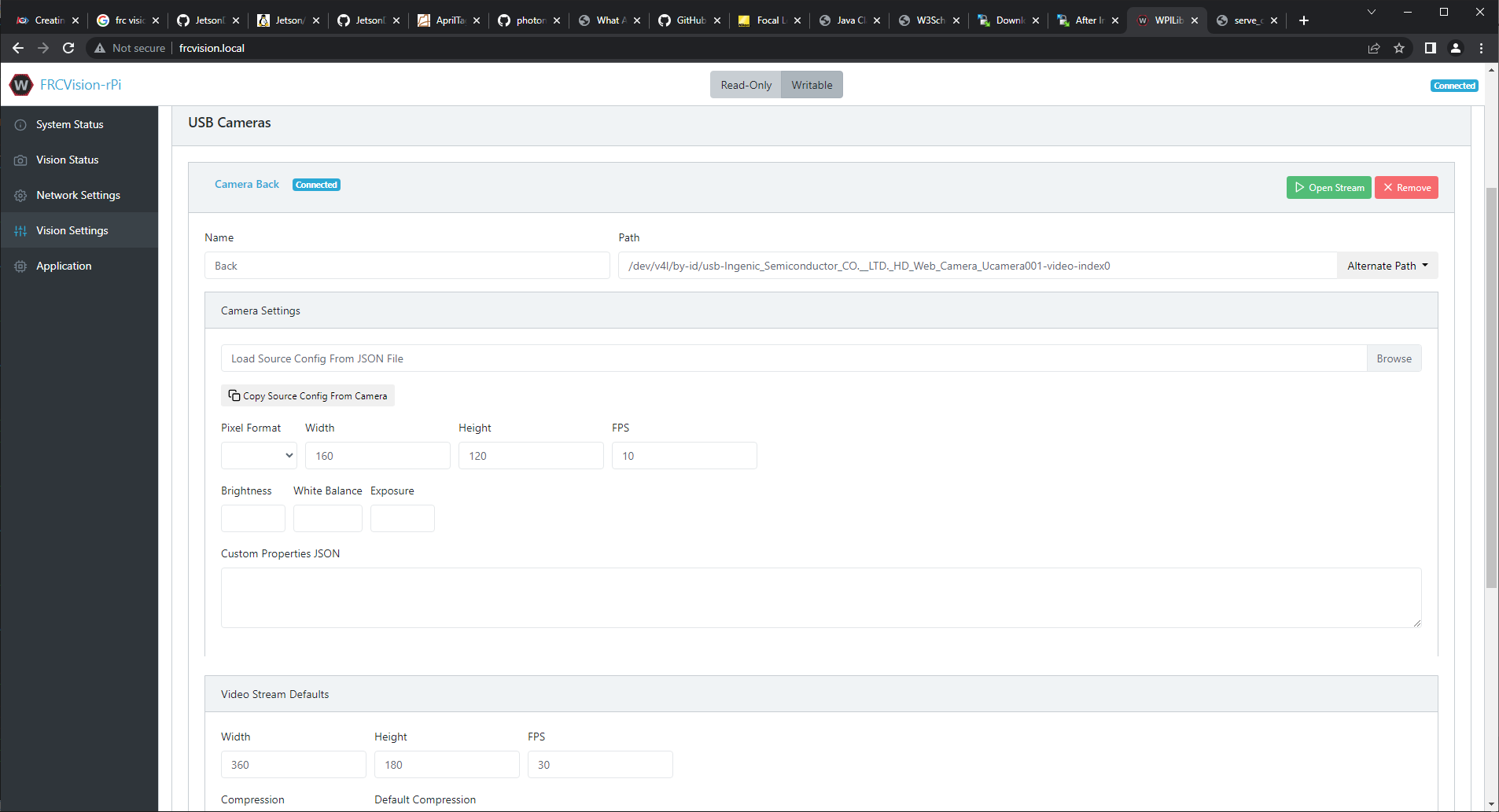
Description automatically generated at the bottom of the page. This will open a selection pane similar to the one seen in the image below. Selecting the *by-id* option (underlined in green) will associate the camera with a specific ID.

A screenshot of a computer

Description automatically generated

Figure

After a path has been selected, a new camera should appear, labeled “Camera New”. The example images were taken from the 2023 system, which already has two cameras set up. “Camera Front” and “Camera Back”. Clicking on the blue camera name will open a settings box. Shown below are the settings for “Camera Back”, from the 2023 system. Once all the settings have been updated, make sure the RPi is set to writable, and click save at the bottom.



5

4

1

3

2

Figure

1. The name of the camera. This can be changed to help the user identify which camera is being edited.
2. The ID of the camera. This will be used later in the vision python program to tell it where to pull its camera feed.
3. The desired image settings for the camera. Images will be processed at the size set here. Technically, any size can be set. However, each camera has a preferred ratio of width to height. Clicking on the green A screenshot of a computer

   Description automatically generated button opens a video feed of the camera, along with a table of supported video modes. If possible, choose one of Video Modes to be the width, height, and FPS. If a smaller resolution is needed, try to keep the ratio between the width and the height the same. For example, the 2023 system was run on a RPi 3. This struggled to process images at a high resolution, so the width and height were set to 160 and 120, respectively. Multiply each by 4 and the resolution would be 640 by 480. Thus, the same ratio was maintained, but at a lower resolution. A low resolution presents a tradeoff. The images can be processed faster and take less bandwidth to send over the network. However, that sacrifices image quality, which can make it more difficult for the program to detect objects, especially Apriltags.
4. The video stream settings. These are the settings for the video stream that is sent over the network. Ideally, this would be less than the actual image resolution, since sending large images over the network takes a significant amount of bandwidth. Doing so can create trouble during competition.
5. The last setting is the amount of compression applied to the Video Stream. Increasing the compression will lower the amount of bandwidth used. However, it will also take more processing power and slow down the vision program. If a newer RPi is used, processing power is in abundance. However, if an older pi is being used, you need to avoid slowing the processing down significantly. If the program only processes at 1 fps, the desired data will be outputted too slowly to be useful.

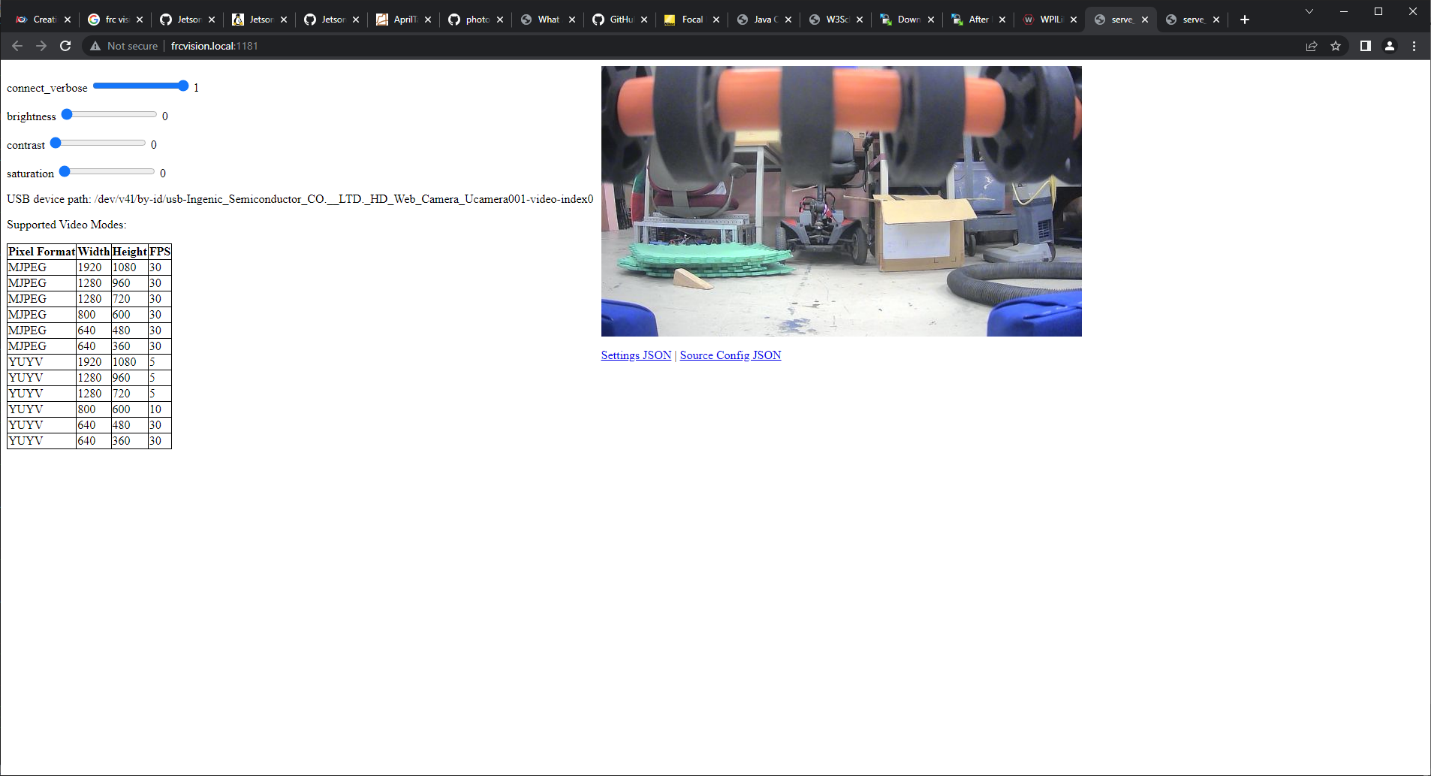


Figure 6: Sample view from the 2023 system of a camera stream. The supported video modes can be seen in the table on the bottom left.

## Connecting Multiple Cameras

If two cameras are needed, the process above needs to be repeated for the second camera. Make sure to unplug the first camera before doing so. This will avoid any confusion about which connected camera the system is setting up.

## Updating the Python Program

Now that the cameras have been set up, the python application needs to be updated with the new camera IDs, which allows it to identify them. Follow [these](#_Using_WinSCP_&) instructions in the quick reference guide to open the python application. Search “Starting Automatic Capture”. The code following that line creates a new camera. Copy the camera path from the web application (number two in *Figure 5*). Paste it between the two single quotes after “path=” (replacing the path that is already there). If two cameras are needed, copy the path of the second camera and insert it on the line where “camera2” is created. Then, search “Set Number of Cameras” to set the variable “numberOfCameras” to 2.

# Vision Subsystem Overview

This section provides a broad overview of the Vision subsystem in the robot code (Using code from the [2023 FRC Charged Up Repository](https://github.com/Team1100/FRCChargedUp/blob/main/src/main/java/frc/robot/subsystems/Vision.java)).

## The Constructor

Starting with the constructor, two arrays are initialized with a variety of important constants that are used to track color targets. A detailed description of each of the constants can be found in the next section.

## *Vision getInstance()*

This method sets up sliders on ShuffleBoard to adjust Vision Processing constants. These are used during calibration. It also posts an integer, “Target Apriltag ID”, to SmartDashboard. This tells the vision program which AprilTag to target.

## *double getTargetOffset()*

This method returns the number of pixels between the center of the target and the center of the camera. It can be used to turn towards the target.

## *boolean isAprilTagTargetFound()*

This returns a true or false value based on whether or not an AprilTag target has been detected by the camera. This can be used to check if an AprilTag is detected by the camera. If it returns false, either the AprilTag is too far away, it is out of view of the camera, or the camera is looking for a different AprilTag ID.

## *void setTargetAprilTag(int apriltagID)*

This method can be used to tell the vision program what AprilTag ID to look for. If used, it first needs to be commented out of the *periodic()* method. Currently, the Vision program constantly pulls the desired ID from SmartDashboard. Thus, if the user wants the robot to set the ID, the periodic method will simply reset the target ID to whatever is on SmartDashboard. Commenting this line stops the constant updates. Note that this will also completely remove the ability to set the ID from SmartDashboard.

## *void setDetectionMode(int detectMode)*

This method tells the program which type of target it should be looking for. Constants at the top of the Vision subsystem can be used as parameters.

* DETECTING\_NOTHING
  + If used, vision program detects nothing
* DETECTING\_COLOR
  + If used, vision program detects objects based specific color and size restraints
* DETECTING\_APRILTAG
  + If used, vision program detects AprilTags based on the variable *aprilTagTarget*, which is set either on SmartDashboard or in the code via the *setTargetAprilTag()* method.

## *setCamera(int camInUse)*

Indicates to the vision program which camera to used (if multiple have been set up). Constants at the top of the Vision subsystem can be used as parameters. **Note:** The constants’ names can be adjusted to better represent the specific cameras in used (I.E. CAMERA\_BACK, CAMERA\_FRONT)

* CAMERA\_1
  + If used, vision program processes the video stream from the first video camera.
* CAMERA\_2
  + If used, vision program processes the video stream from the second video camera.

## int getTargetAprilTag()

Returns the specific ID that the vision program is looking for.

## void periodic()

Updates debugging read-outs for the *offset* variable on SmartDashboard. It also pulls the user inputted “Target AprilTag ID” from SmartDashboard and updates the vision program.

# Calibration and Configuring the Vision Processing Constants

## HSV Tolerances

### Overview

There are two parts to calibrating the color detection constant. The first involves a series of minimum and maximum [HSV values](https://www.lifewire.com/what-is-hsv-in-design-1078068) (from 0 to 255 for Hue, Saturation, and Value) that indicate which color to accept. For example, setting *hueMin* to 30 and *hueMax* to 255 accepts all potential targets with a hue greater than 30 and less than 255.

### Setup

To do the initial calibration (in the robot room), turn on the robot, connect to the wifi, make sure the RPi is fully set up, and open both Driver Station and Shuffleboard. Make sure the robot is connected and operating, and then open the web interface for the RPi. Restart the vision program by switching to the Vision Status page and pressing the red X Kill button. If you want to view the console output, toggle the “Enable” button at the top right of the console output box. If everything restarts correctly, it should wait for five seconds, connect to the cameras, and then connect to NetworkTables. The console will print “Connected!” to indicate a successful connection.

* If it simply prints “Waiting” and nothing else, then it is trying to connect, but failing. The most common causes are either issues with the firewall blocking the connection, or an incorrect team number somewhere on the robot or RPi.

Once connected, navigate to Shuffleboard. Switch to the “Vision” tab. In the top left corner, click on the two greater-than signs to expand the drawer. Click on the CameraServer menu to expand it. Several options should be present. The two important video streams are “visionCam” and “maskCam”. Right clicking each will open a selection pane that allows you to select Show as: Camera Stream. This will open a widget on Shuffleboard that streams the video.

* Note: If the Vision tab doesn’t show up, make sure an instance of the Vision subsystem is created in the “Robot.java” file. Use [2023 ChargedUp](https://github.com/Team1100/FRCChargedUp/blob/main/src/main/java/frc/robot/Robot.java) for reference.

##### Vision Cam

* visionCam is a camera stream of fully processed video. When processing for color, it will draw a bounding box around the target. When processing AprilTags, it will outline the AprilTag in red and print the ID in the center.

##### Mask Cam

* maskCam only streams when processing for color. A mask is a black and white view of the video feed: white indicating which colors are included in the processing and black indicating which colors are excluded from the image. This inclusion and exclusion are determined by the HSV constraints explained above. For instance, the 2023 ChargedUp robot was calibrated to detect purple cubes. The values listed below were configured to isolate all colors except purple. Thus, the purple cube would show on the maskCam as a white square, and everything else would be black. (Unless there happened to be another purple object in view, in which case it would also show as white)
  + Hue Minimum: 109.92
  + Hue Maximum: 153
  + Saturation Minimum: 103
  + Saturation Maximum: 255
  + Value Minimum: 103
  + Value Maximum: 255

### Calibrating the HSV Constants

Both camera streams can be expanded as needed. Also present in the Vision tab should be six slider widgets. Each is associated with one of the HSV constraints. If the default values from a previous year are kept, the sliders should reflect those values. To start from scratch, reset each slider to their minimum/maximum values. All the min values should be at 0 and all the max values should be at 255. Once everything has been reset, the maskCam should be completely white. Pick one of the min tolerance and slowly drag the slider to the right. At the same time, watch the maskCam. Drag the slider until some of the black starts eating into the object in question. Then, slowly back the slider up until most of the object is white. Pick a max tolerance (ideally the same HSV value as the min tolerance) and slowly drag to the left. As with the min tolerance, drag the slider until the object begins to turn black, and then back it up until the object is mostly white. Repeat this process with the last four values.

* Note: As the light changes in the room or as the robot changes its position relative to the object, the color can shift quite dramatically. If the program is calibrated too finely, it will lose track of the object as these color shifts take place. Thus, caution on the side of accepting more color than you reject. This will allow for a greater range of colors and will account for these color shifts. Even if it does detect objects other than the desired game piece, the next step of calibration will work to isolate the target even further.
* Click [here](0001-1685.mp4) to watch a video recording of the calibration process.

Record all of the calibration values (taking a picture of the screen works perfectly fine). To plug the values into the program, go to the Vision subsystem, search “HSV Constants”, and update the values in the “defaultHSV” array.

## Target Filtering Constants

At this point, the vision program has been able to isolate objects of a specific color. These show up as the white blobs on the mask (also called contours). However, to accommodate for changing light conditions, the HSV tolerances can’t be too specific. An unfortunate consequence is that the program fails to isolate objects that are a similar color. For example, if the game piece is blue, the mask will also pick up the blue bumpers of the opposing alliance. As each frame is processed, the vision program creates a list of the contours (or potential targets) seen in the mask. Thus, more constraints are necessary to pick which contours in the list are the right object. The vision program has a variety of ways to decide which contour on the mask is the true target. An overview of several of these techniques can be found here.

### Overview

#### Minimum Area

Before any further filtering happens, any contour that is too small is rejected. This filters out the smallest anomalies. Be careful about setting this too high, or it will reject targets that are far away (and thus appear small to the camera).

* **Note:** This value also changes based on the camera resolution set in the RPi web interface. The higher the resolution, the larger the minimum area can be.

#### Area Ratio

As the program loops through the list of contours, it calculates two different areas for each. The first is the actual area of the contour. The second is the area of the bounding box created around the contour. The area ratio is simply the contourArea divided by the boundingArea. Let’s look at an example. Imagine two objects are detected.

##### Perfect circle:

* Radius: 10 pixels

20 px

##### Rectangle:

10 px

* Width: 40 pixels

40 px

* Height: 20 pixels

|  |  |
| --- | --- |
| Circle  Contour Area  Bounding Area  **Area Ratio** = | Rectangle  Contour Area  Bounding Area  **Area Ratio** = |

In simplified terms, the area ratio of an object describes how much of the bounding box is filled by the contour. To indicate to the program what area ratio to be looking for, the user sets up an ideal Area Ratio and an Area Ratio Tolerance. The ideal Area Ratio is the expected area ratio for the object. In the example above, the ideal Area Ratio for the circle is 0.785. However, the program will never detect a perfect circle. Depending on how finely tuned the HSV tolerances are, the contour area will be slightly greater or less than it should be. It also changes based on the cameras distance to the object. This is where the Area Ratio tolerances come in. Any object that is less than the ideal Area Ratio + tolerance and greater than the ideal Area Ratio - tolerance is allowed to pass on to the next level of filtering.

* Ex: Circle
  + Ideal Area Ratio: 0.785
  + Area Ratio Tolerance: 0.2
  + Range: (0.585, 0.985)
  + Contour Area Ratio of 1.0: Not the target
  + Contour Area Ratio of 0.63: Potentially the target

#### Aspect Ratio

After passing the Area Ratio test, the next filter is Aspect Ratio. Aspect Ratio is the ratio between the width and height of the bounding box around the contour (in this case ). Just as with the Area Ratio, it has an ideal Aspect Ratio and an Aspect Ratio tolerance.

|  |  |
| --- | --- |
| Circle  **Aspect Ratio** = | Rectangle  **Aspect Ratio** = |

Aspect Ratio describes how oblong an object is. Tall objects have small (decimal) aspect ratios. Square objects have aspect ratios close to 1.0. Wide objects have large aspect ratios.

* Ex: Rectangle
  + Ideal Aspect Ratio: 2
  + Aspect Ratio Tolerance: 0.2
  + Range: (1.8, 2.2)
  + Contour Aspect Ratio of 1.3: Not the target
  + Contour Aspect Ratio of 1.9: Potentially the target

#### Coordinate Position

Coordinate Position filtering eliminates objects that are detected in a position on the screen that is very unlikely. For instance, if the camera is pointed horizontally and the target is high up on a wall, any contour that is detected at the bottom of the video stream is unlikely to be the target. By this point, aspect and area filtering should have done the bulk of the work. Thus, using this method of filtering is not necessary. In fact, it needs to be used with caution. If the tolerances are too small, the actual target can easily be filtered out accidentally. The field presents so much variability that the target is often not where it is expected to be. For instance, during 2023 ChargedUp, the target was accidentally being filtered out because the camera had been shifted a few degrees, thus putting the target out of its expected range. Also, an objects Y coordinate can change dramatically depending on how far the robot is to the target. So, even if you are confident that the target will stay within a certain range on the screen, choose larger tolerances than necessary. There are four values associated with Coordinate filtering.

* Ideal Y Coordinate
  + Describes where the object should be on the y-axis.
  + **Note:** this value needs to align with the pixel coordinate system (zero being at the top of the screen)
* Y Coordinate tolerance
  + Describes how much variability is allowed on the y-axis.
* Ideal X Coordinate
  + Describes where the object should be on the x-axis.
  + **Note:** A value of -1 will tell the program to choose the center of the video stream as the ideal coordinate.
* X Coordinate tolerance
  + Describes how much variability is allowed on the x-axis.

**Note:** To remove coordinate filtering, set the tolerances to wider than the video stream dimensions. This will accept contours anywhere on the screen.

### Setting the Target Filtering Constants

All the constants are stored in the Vision subsystem in an array called “colorDetectConstants” (search “Color Detection Constants” to find it). Start by calculating each ideal constant. Some targets are easy. For instance, the cube from 2023 ChargedUp should have an Area Ratio of 1, an Aspect Ratio of 1, and so on. However, if the program were used to detect the yellow cones instead, it becomes more difficult.

1. Print out an image of the target/object that is like what the camera would view.
2. Approximate the dimensions of the bounding box with a ruler.
3. To find the area of the object itself, use basic geometric shapes. For instance, a triangle could be used to approximate the side view of a cone.
4. Use this area and the bounding box dimension to calculate the ideal Area Ratio and Aspect Ratio.
5. Set both the ideal y-coor and ideal x-coor to -1

After the ideal constants have been calculated, choose very wide tolerances for all the filters. Finally, choose a very low value for the minimum area. Push the code to the robot. Once that has finished, [restart the vision program](#_Setup). Point the camera a single target, ideally with a background that isn’t cluttered with other objects. With wide tolerances and few other distractions, the program should easily detect the target (indicated by a bounding box being drawn around the target on visionCam). When a target is detected, the program publishes data on that object to NetworkTables (which can be viewed on Shuffleboard. Sometimes the data can be hidden behind other widgets, so try moving widgets around if they aren’t showing up). These include the actual Aspect Ratio, Area Ratio, and coordinates. Use these values to update the ideal filtering constants.

# Operation during Competitions

## Setup for Computer Connection

Given that an ethernet connection during competitions is required to interface with the robot, connecting to the RPi involves a different hardware setup. Unfortunately, the radio doesn’t have enough ports to connect the laptop, the robot, and the RPi together. So, an external ethernet switch is required. The team should have a cheap switch that is powered off a USB connection to the RIO. Any other network switch requires the user to cut its power wire in half and plug the bare wares into the VRM. Once a switch has been acquired and powered, unplug the RPi and the RIO ethernet cables from the radio and plug them into the switch. Complete the connection by plugging a computer into the switch via its own ethernet cable. This setup places the computer, the RPi, and the RIO on the same network. The user should be able to open Shuffleboard, Driver Station, and the RPi web interface as usual.

## Calibration

Calibration is required at the beginning of each competition to adjust for the field lighting conditions. For instance, the lights at home may be a warm white. However, the lights on competition fields are often cool white. This changes the perceived color of the game objects and so new HSV constants need to be found. To begin, use the hardware setup described above. Then, place the robot on the field with the desired object in view of the camera. Finally, complete the [calibration process](#_Calibrating_the_HSV) as normal.

## Setup for a Match

The network switch is only necessary when a computer needs a connection with the RPi and the robot. Thus, the regular hardware setup can be used (as described in the [Vision Hardware Setup](#_Vision_Hardware_Setup) section).

* **Note:** When multiple potential targets are in view, the vision program picks the target that shows as largest on the screen. This should pick the target that is closest to the robot. This sorting is performed near line 433 (search “key=lambda”) of the python program. The program has a list of targets, which it sorts from smallest to largest using the “boundingArea” variable. Note that the target ID (the target being chosen) is simply the last target in the list. So, after having been sorted from smallest to largest, the largest target is chosen. If some other method of sorting is needed (i.e. choosing the right most target, lowest target, etc), switch the “target.boundingArea” out with a different variable from the TapeTarget class. Below are a few options.
  + left to right: target.x
  + top to bottom: target.y
* **Note #2:** If a different side of the target list is needed (i.e. choosing the smallest target instead of the largest target, or leftmost target instead of the rightmost, etc) change “len(self.tapeTargetList)-1” to “0”. This will select the first value of the target list instead of the last.

TODO:

Update line numbers in the Raspberry Pi Repository

Mention restarting shuffleboard and repushing code to the robot if widgets don’t show up

Can manually change the networktable values on vision tab (have to change apriltag ID on smartdashboard)

# Quick Reference Guide

## Using WinSCP & FileZilla

### WinSCP

Click here and download the most recent version of the software. Run the setup application and choose all the default settings. Before opening the application, make sure the RPi is fully set up and the computer is connected to the robot Wi-Fi. When opened, WinSCP should automatically open a pop-up like Figure 8. If not, then click “New Tab” in the top left corner of the screen. Fill in the information required. The host name would be either “frcvision.local” or “wpilibpi.local”, depending on how recent the pi was aquired. The username is “pi” (unless someone changed it from the default). The password is “1100pies” (unless somone changed it, or a newer pi is being used. The password on brand new RPi’s is “raspberry”). Clicking the arrow next to Login provides the option to save this information. Once everything has been filled out, go ahead and click Login. It may require the password again, and then you should be through.

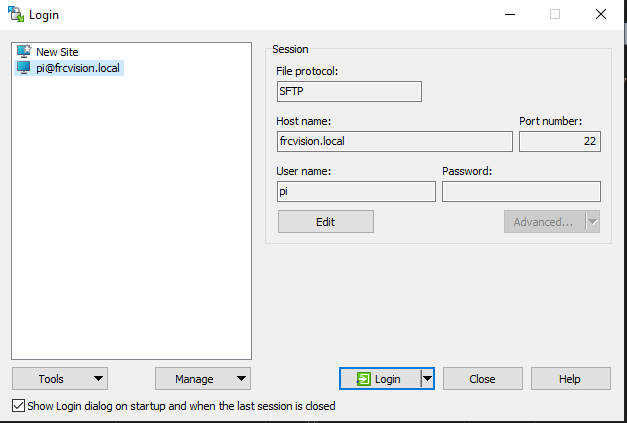


Figure7

The left pane represents the file system on your device. Navigate to the team repository for the year and open the “vision” folder. The path should be similar to the path in the blue bar on the top left. The right pane represents the file system on the RPi. Double click on the “python-app” folder to open it.

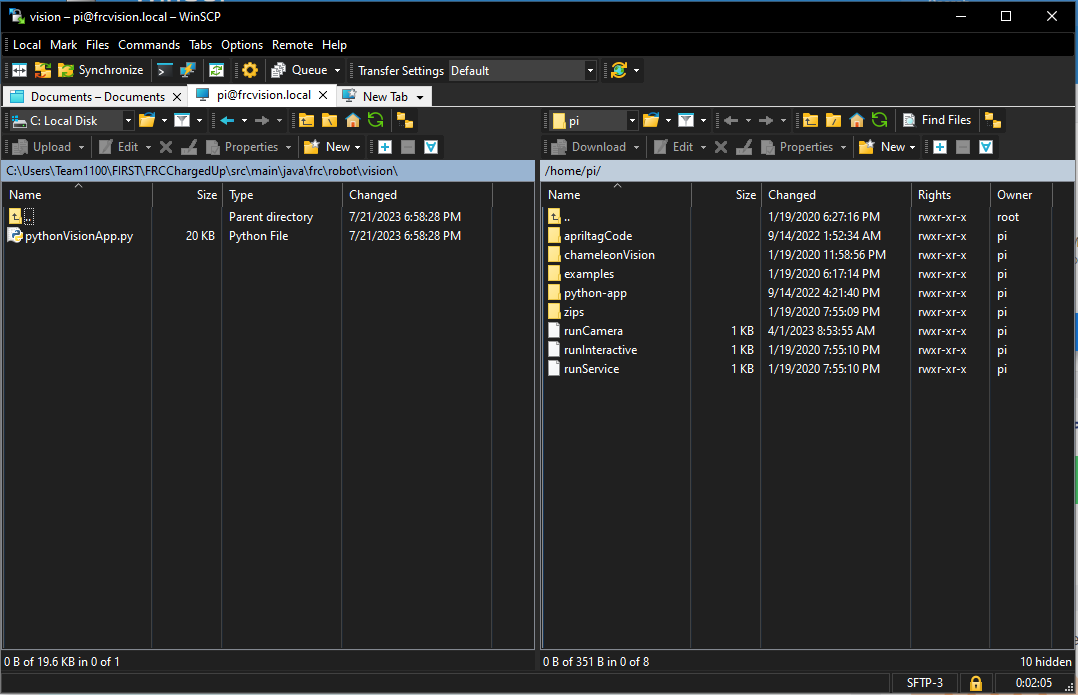


Figure 8

Before making any edits, make sure the RPi is set to writable. Double clicking on “pythonVisionApp.py” will open a text editor that allows you to edit the file. Alternatively, the python file on the computer can be edited. Then, clicking and dragging it to the opposite side will update the python file on the RPi.

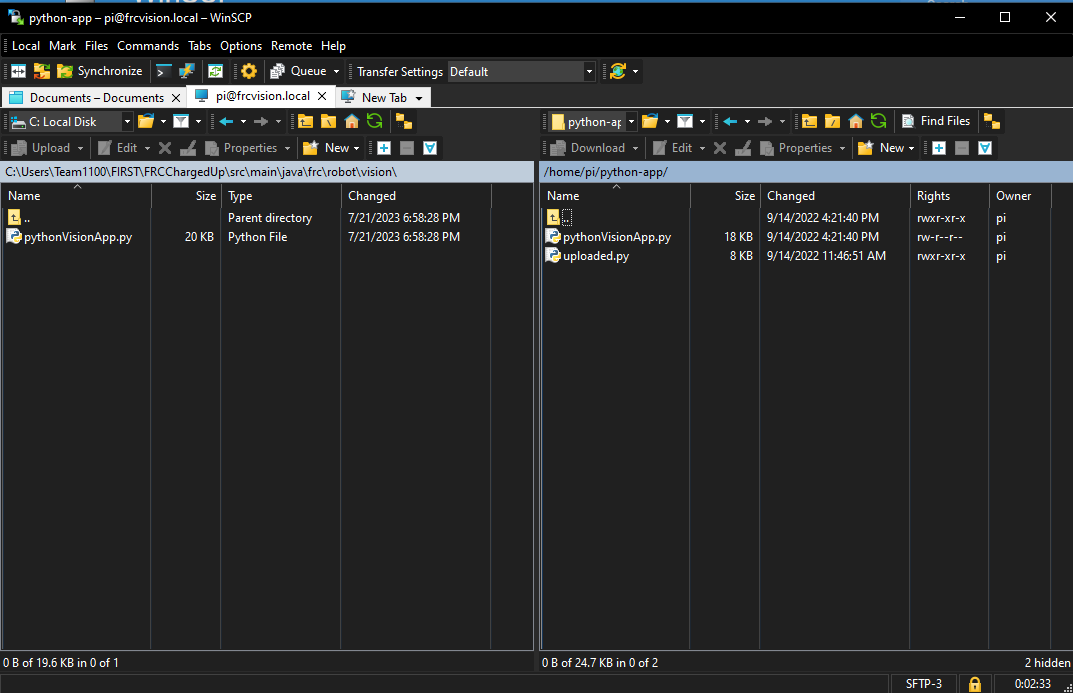


Figure 9