

The Design Simple Guide to JeVois for FRC - Part 1

By Anand Rajamani

Table of Contents

[Introduction](#)

[Goals](#)

[Before we begin](#)

[Step 1: Setting up the JeVois](#)

[Step 2: Downloading supporting software](#)

[AMCap](#)

[Arduino](#)

[GRIP](#)

[Project Files](#)

[Step 3: Setting up the JeVois file system](#)

[Accessing the JeVois file system](#)

[File System Details](#)

[Step 4: Using GRIP and generating the code](#)

[Starting GRIP](#)

[How to use GRIP](#)

[The “Blur” block](#)

[The “CV extractChannel” block](#)

[The “CV Threshold” block](#)

[The “Mask” block](#)

[The “Normalize” block](#)

[The “HSV Threshold” block](#)

[The “Erode” block](#)

[The “Dilate” block](#)

[The “Find Contours” block](#)

[The “Filter Contours” block](#)

[Generating Code](#)

[Putting code on to the JeVois](#)

[Step 5: Running vision code](#)

[Step 6: Adjusting camera settings \(exposure, brightness, etc.\)](#)

[Step 7: Tweaking the JeVois code](#)

Introduction

Vision in FRC has traditionally been a massive challenge in FRC. Integrating camera, control system, and vision processing is a massive challenge to undertake, one that many teams struggle and fail with each year. Paying hundreds of dollars for coprocessors and setting up specialty cameras, only to find that there's not enough time to get it to play nice with the robot code in time for competition, is a cycle many teams know all too well.

Recently, however, small integrated camera modules have started to become popular. The Pixy (CMUcam5) and openMV cameras were pioneers in this field, but more recently a game-changing camera appeared: the JeVois. The JeVois Smart Machine Camera is a small CPU/GPU/Camera combo that features a highly optimized Linux-based OS that can run OpenCV code in Python or C++ very efficiently. The JeVois is not a direct replacement for a powerful coprocessor like the Nvidia Jetson series, but the JeVois can be put to work within minutes of pulling it out of the box. It is also (at time of writing) available with same-day shipping with Amazon Prime if ordered in the morning, or next-day shipping with Prime if ordered in the evening for only \$50.

The purpose of this guide is to learn how to use the JeVois in a manner that suits FRC as fast and as painlessly as possible, especially for people who do not consider themselves experienced programmers.

This paper integrates with the GitHub project found here:

<https://github.com/asid61/JeVois-For-FRC>

Goals

1. Have the JeVois stream video to your computer
2. Communicate with the JeVois via the Arduino Serial Monitor
3. Set up GRIP to work with the JeVois
4. Reorder the JeVois file system to make programming as easy as possible
5. Run GRIP-generated Python code on the JeVois to recognize retroreflective targets

All of this can be set up in well under a day. Part 2 of this guide details on moving data to the RoboRIO once the vision pipeline is working.

Before we begin

What do you need to get started? Not too much:



- A Windows computer with an SD card slot
 - If you don't have an SD card slot, [USB-SD](#) adapters are readily available
- [8GB+ micro-SD card](#)
 - Make sure you have a micro-SD to regular SD adapter if you need one

- [A JeVois](#) - also available from [Amazon](#) for the same price
- Mini-USB B cable, [NOT a Micro-USB](#)
- [A green led ring](#) - also from [Andymark](#)
 - Optional, but it makes recognizing retroreflective targets FAR easier. Always have this when using on a robot.
 - A 12v power supply will probably be necessary for this
- [A mount](#)
 - Also optional, but you'll need one sooner or later.
- A vision target

Step 1: Setting up the JeVois

First, we're going to follow a process similar to the [JeVois Quickstart Guide](#). Download the JeVois SD Card image from <http://jevois.org/start/start.html>. Also download Etcher from here: <https://etcher.io/>.

The "SD Card Image" is basically the Operating System for the JeVois. The JeVois does not ship with an OS on it, so you'll need to add one. That's where Etcher comes in. Etcher lets you put the OS onto the JeVois in a way that Windows cannot on its own.

 Etcher-Setup-1.2.1-x64.exe	12/9/2017 8:00 PM	Application	51,298 KB	
 jevois-image-latest-8G.zip	12/9/2017 7:59 PM	Compressed (zipp...	603,268 KB	

The Etcher installer and the JeVois Image

Once you've downloaded the image, you'll need to extract it from the zip file. This usually takes a few minutes, so now would be a good time to also run the Etcher installer.

Once the files are extracted, you may see more than one image if you've downloaded one before. Use the latest image, in this case 1.6.0.

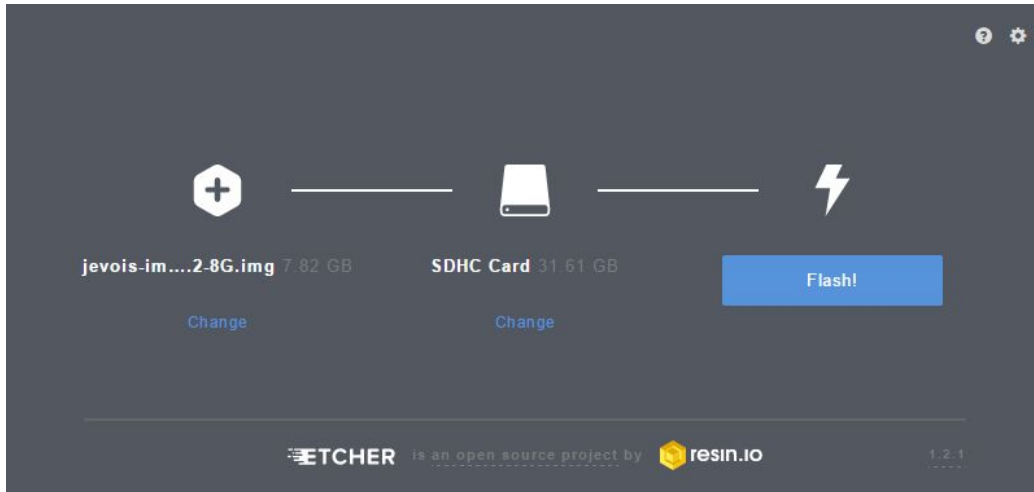
 jevois-image-1.5.2-8G.img	10/24/2017 10:08 ...	Disc Image File	7,639,040 KB
 jevois-image-1.5.2-8G.md5	10/24/2017 10:08 ...	MD5 File	1 KB
 jevois-image-1.6.0-8G.img	12/9/2017 8:08 PM	Disc Image File	7,639,040 KB
 jevois-image-1.6.0-8G.md5	12/9/2017 8:08 PM	MD5 File	1 KB

The extracted images

Insert your SD card into your computer or SD card reader. Now, we are ready to flash the image onto the SD card.

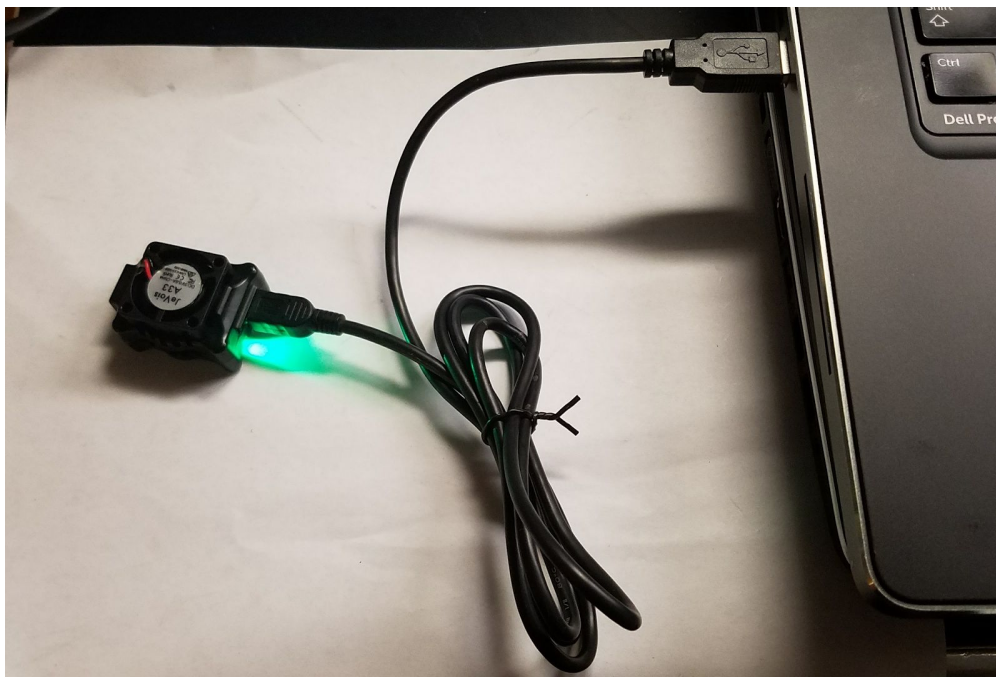
Open Etcher and select the .img file as your image, and select your SD card to be the drive. Here, I'm using a 32GB SD card.

Click "Flash!" to flash the image onto the SD card. This process usually takes a few minutes.



The correct Etcher setup

Congratulations! The JeVois is now ready for use in any application. Put the microSD into the JeVois SD card slot and use the mini-USB cable to connect it to your computer. Make sure to push the microSD card into the JeVois until you hear it click.



The JeVois plugged in

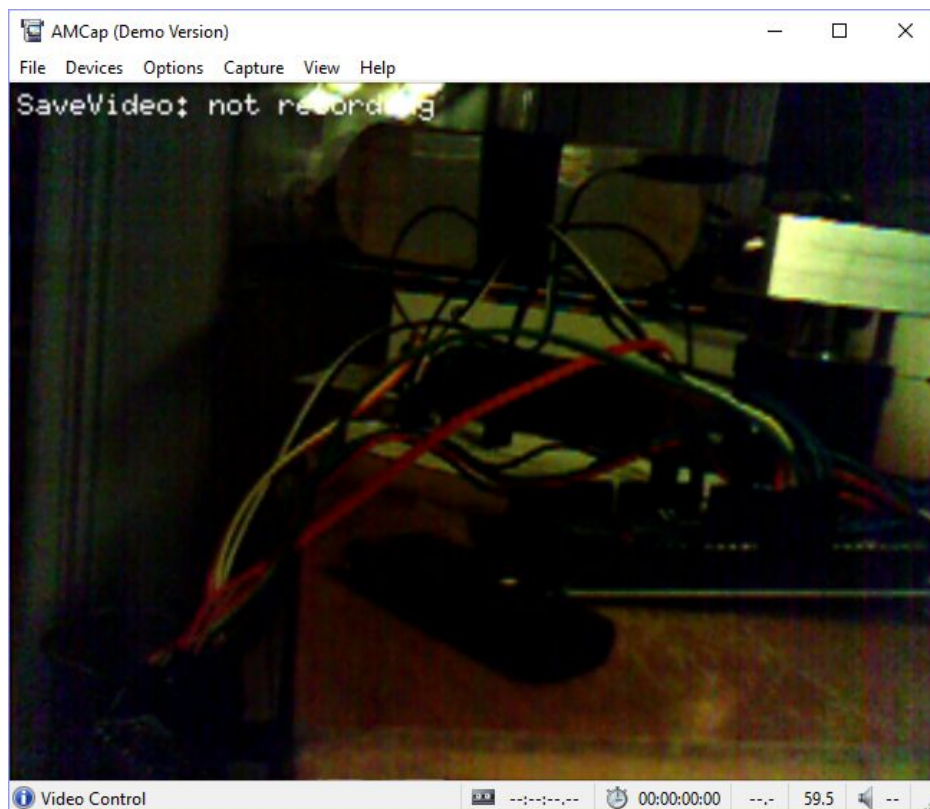
Step 2: Downloading supporting software

Now that the JeVois is ready for use, you'll need some software to make vision tuning as easy as possible. We're going to use three programs to do this: AMCap, Arduino, and GRIP. Make sure your JeVois is plugged in for these steps!

AMCap

We're going to start out by downloading AMCap. AMCap is a webcam capture program with the ability to select the resolution and framerate of the camera. We will need this to select the operating mode of the JeVois.

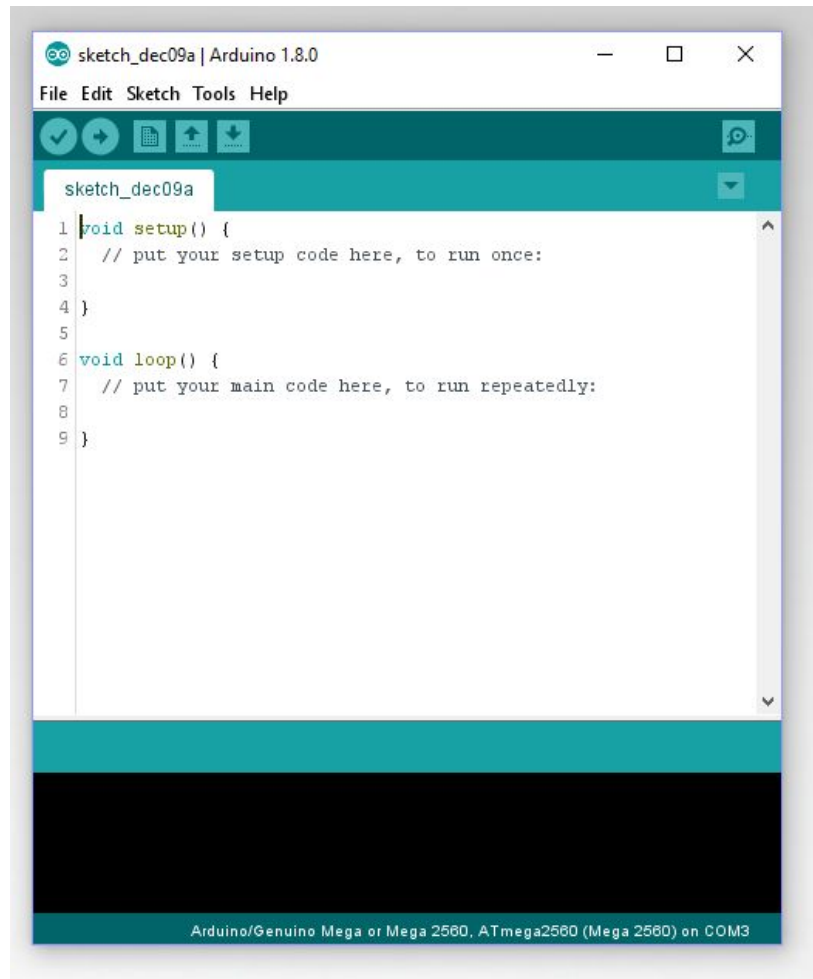
First, download AMCap here: <https://amcap.en.softonic.com/download>. Extract the downloaded folder and run "setup.exe" (the only file in the folder). Run AMCap, and you'll see a video feed pop up. Make sure you are only running one instance of AMCap at once; trying to run two or more will make the second one you open show an error.



The default AMCap JeVois video feed - success!

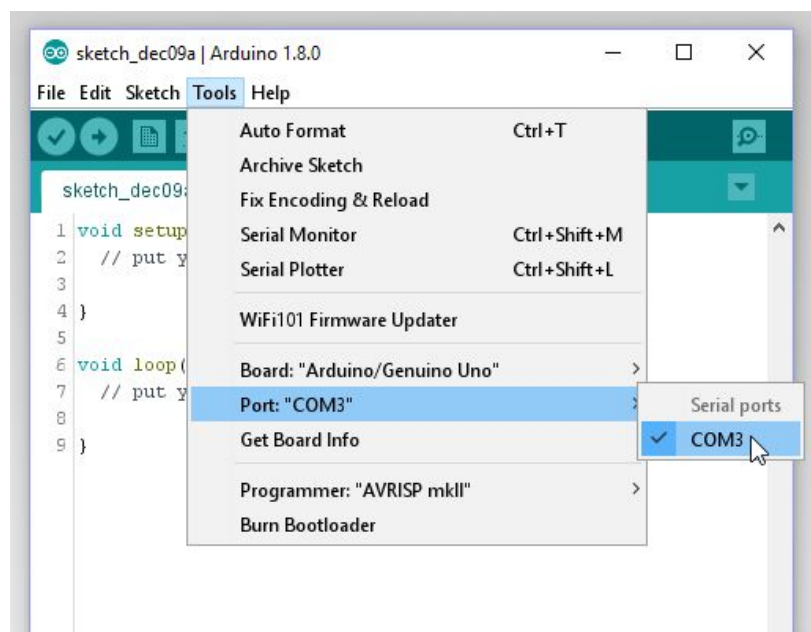
Arduino

Next, download Arduino 1.8 from here: <https://www.arduino.cc/en/Main/Software>. The “Windows Installer” will let you run an .exe and install Arduino. We won’t be using Arduino to actually program anything, but we will be using it for its Serial monitor. Once you’ve installed Arduino, open it.



The Arduino window

Next, go to Tools -> Serial Port -> COMx. The number of the COM port (“x”) doesn’t really matter; just pick the highest one on the list. Having only a single COM port listed is ok. This step basically just tells Arduino which Serial device it’s talking to, so if you have multiple JeVois or Arduino boards plugged in you can select which one to send to.



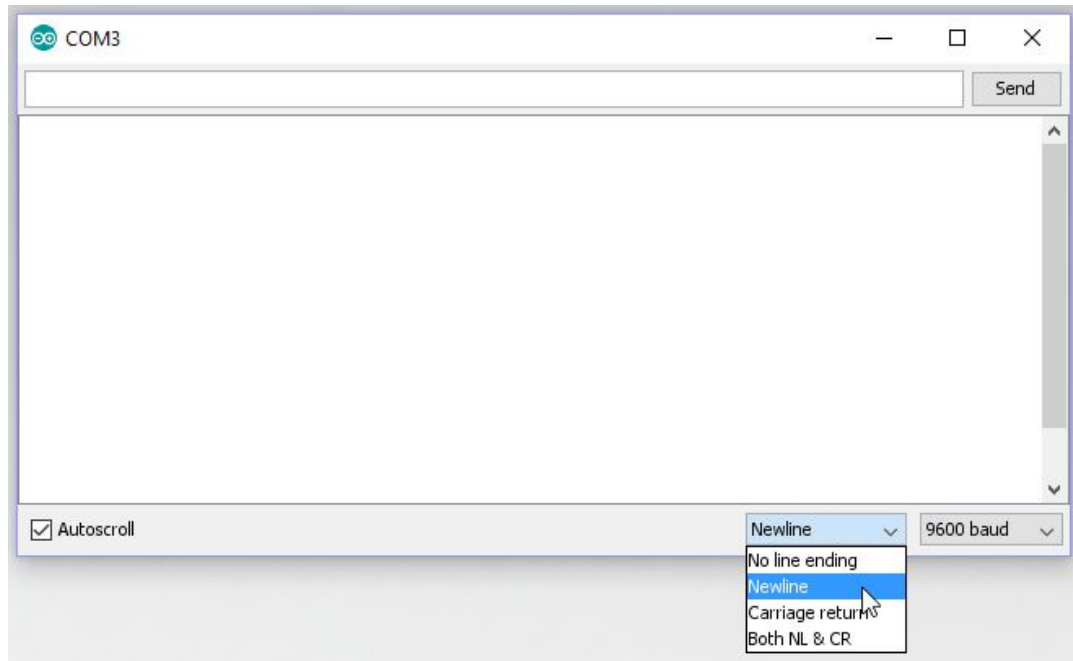
Picking the COM port

Next, click on the Serial Monitor button in the top right, or go to Tools -> Serial Monitor.



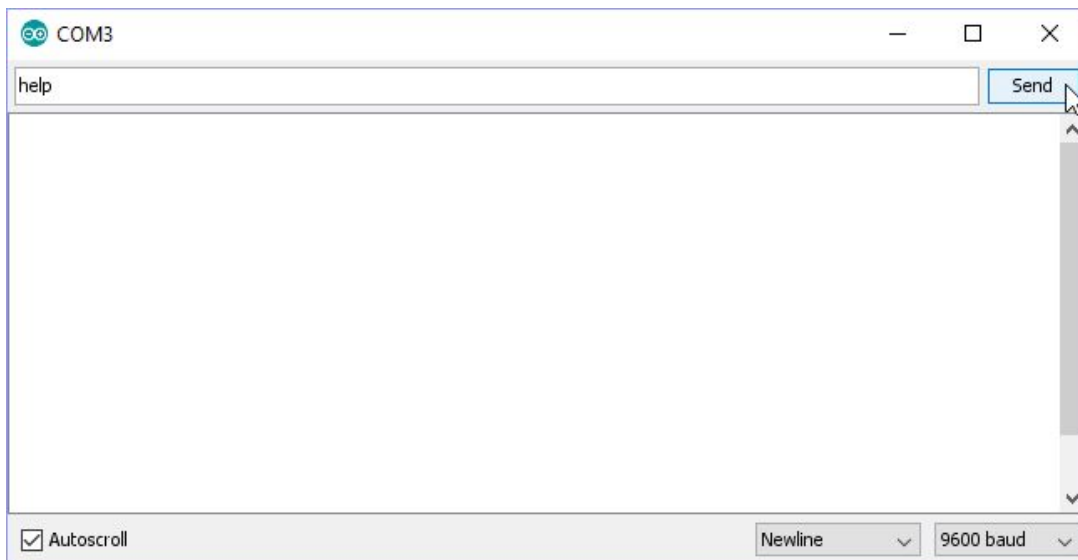
The Serial Monitor button

Make sure that “Newline” is selected and that the baud rate is set to 115200 baud. This is very important. The JeVois expects Newline mode for line endings and it will not work without it selected! 115200 baud is the data rate, which in this case means that it can transmit data at 115kb/s. This is not the same as the video stream data rate! 9600 baud will work as well, but 115200 is much faster.



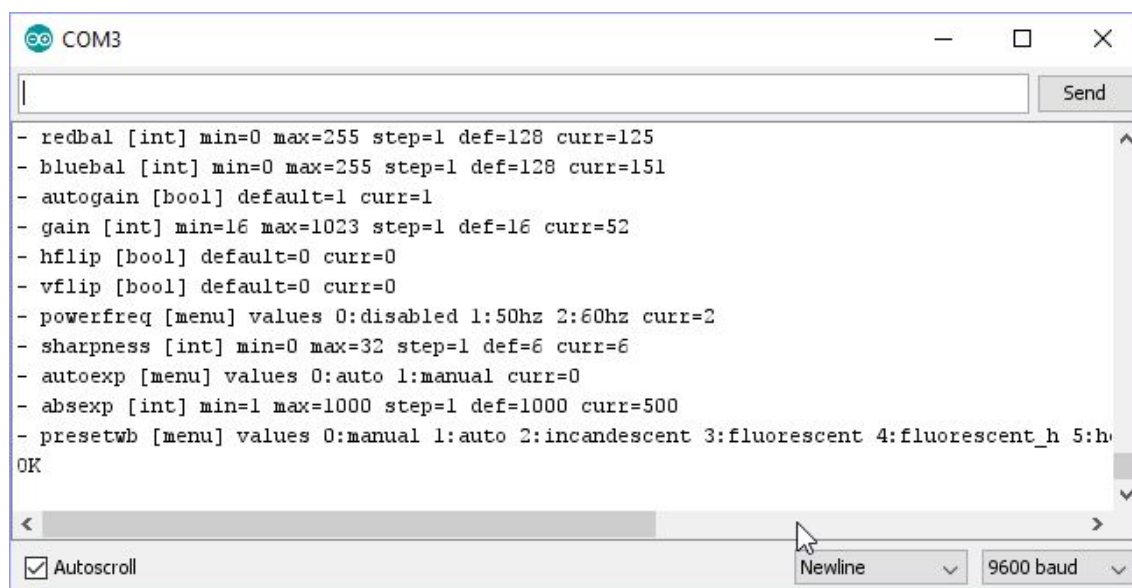
Selecting the line ending and baud rate

Next, type in “help” (without quotation marks) in the top and hit “Send”.



Typing commands into the Serial Monitor

You should see a long list of text come up. Don't worry about what it says for now; we'll come back to the Serial monitor later when we program the JeVois. For now, it's enough that the JeVois is sending something back.



Success! The JeVois is responding to Serial commands.

What we've just done is we've established a communication between computer and JeVois. You can use the Serial Monitor to do various things with the JeVois, including view files or set the camera output (among many other things).

GRIP

Finally, we're going to install GRIP. GRIP stands for Graphically Represented Image Processing engine, and acts as a Graphical User Interface (GUI) for our vision code. Developed by WPI, it is extremely easy to use, with a click-and-drag interface that lets you select filters and operations to perform on a webcam feed for image processing. Install GRIP from here: <https://github.com/WPIRoboticsProjects/GRIP/releases>. Select the x64.exe if your computer is 64-bit and x86_32.exe if your computer is 32-bit.

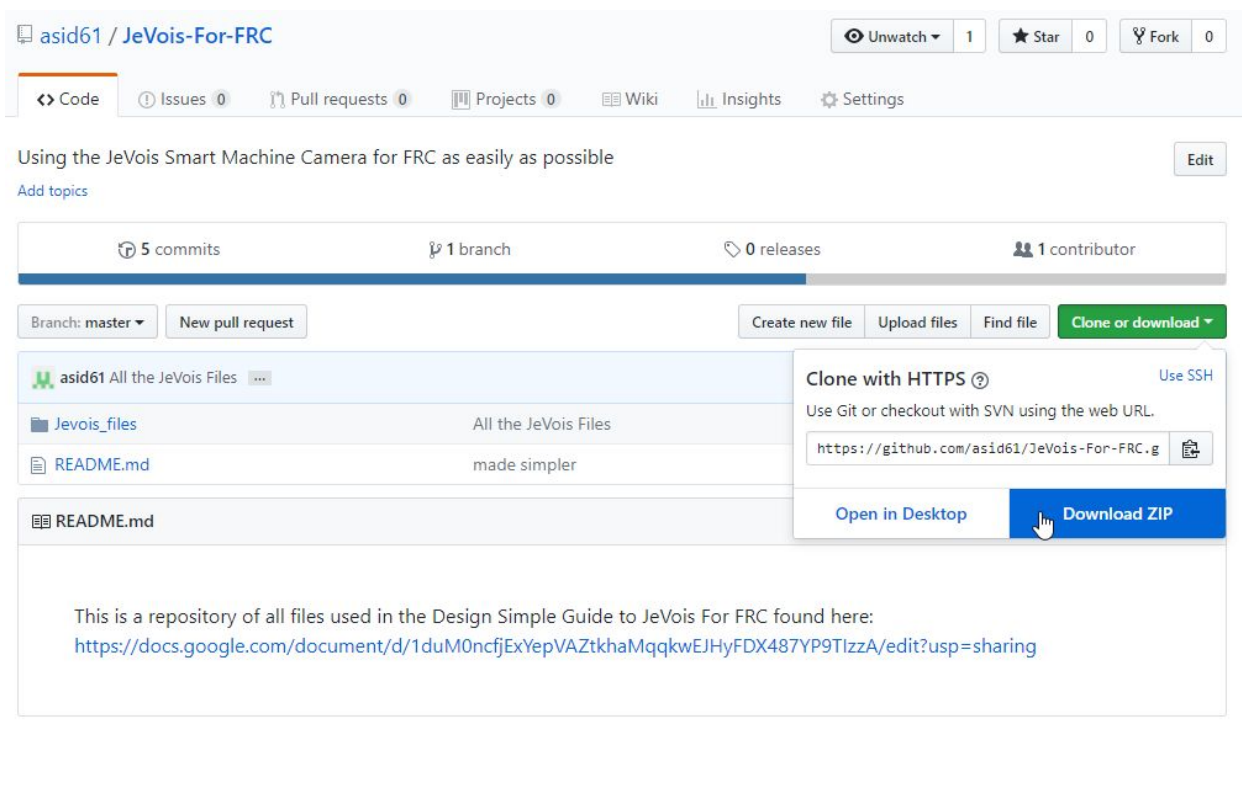
grip-1.5.2-x64.deb	167 MB
GRIP-v1.5.2-x64.dmg	115 MB
GRIP-v1.5.2-x64.exe 64-bit	87.4 MB
GRIP-v1.5.2-x64.pkg	107 MB
GRIP-v1.5.2-x86_32.exe 32-bit	122 MB
Source code (zip)	
Source code (tar.gz)	

Selecting the correct installer.

Project Files

Finally, download the project folder from GitHub. This is where all of the files used for the JeVois are stored, including our premade GRIP pipeline.

Download the project folder from here: <https://github.com/asid61/JeVois-For-FRC>. Click on the green “Clone or download” button and click on “Download ZIP”.



asid61 / JeVois-For-FRC

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Using the JeVois Smart Machine Camera for FRC as easily as possible Edit

Add topics

5 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

asid61 All the JeVois Files ...

Jevois_files	All the JeVois Files
README.md	made simpler

README.md

Clone with HTTPS ⓘ Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/asid61/JeVois-For-FRC.g>

Open in Desktop Download ZIP

This is a repository of all files used in the Design Simple Guide to JeVois For FRC found here:
<https://docs.google.com/document/d/1duM0ncfjExYepVAZtkhaMqqkwEJHyFDX487YP9Tizza/edit?usp=sharing>

Downloading the project files

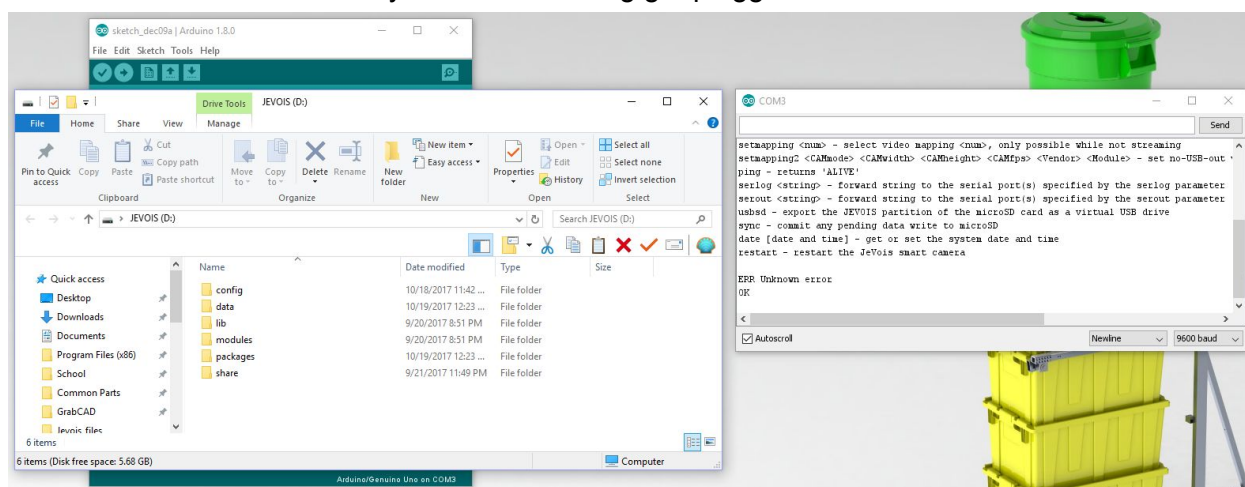
Extract the files from the zipped folder once you’ve downloaded it. Place the folder somewhere easy to find, such as the Desktop.

Step 3: Setting up the JeVois file system

Here, we're going to do a couple of quick mods to the default JeVois files that came with the SD image we flashed way back in Step 1. JeVois ships with a few dozen vision processing examples to examine. But because we're only going to be running one or two vision programs on this, we don't need all of the extra examples provided with the image, and in fact they can cause problems down the line as well.

Accessing the JeVois file system

First, make sure that all webcam programs including GRIP and AMCap are closed, as they will interfere with opening the JeVois file system. Start Arduino and open up the Serial Monitor. Type "help" and press Send to make the sure JeVois is working. Then type in "usbsd" (no quotation marks) and hit "Send". If done correctly, you should see an "ok" pop up in the Serial monitor before Windows tells you that something got plugged in.



The JeVois file system is now open after sending the "usbsd" command.

"usbsd" simply tells the JeVois that it needs to act like a USB stick and show its files. If you go to the project folder we downloaded in Step 2, you'll find a folder labeled "Jevois Filesystem". Open it.

Name	Date modified	Type	Size
Jevois Filesystem	12/9/2017 11:08 PM	File folder	
Vision Images	12/9/2017 11:08 PM	File folder	
grip_constants.py	12/9/2017 11:08 PM	Python File	13 KB
JeVois startup settings.txt	12/9/2017 11:08 PM	Text Document	1 KB
JeVois_CAD.STEP	12/9/2017 11:08 PM	STEP File	1,340 KB
JeVois_GRIP.grip	12/9/2017 11:08 PM	GRIP File	7 KB

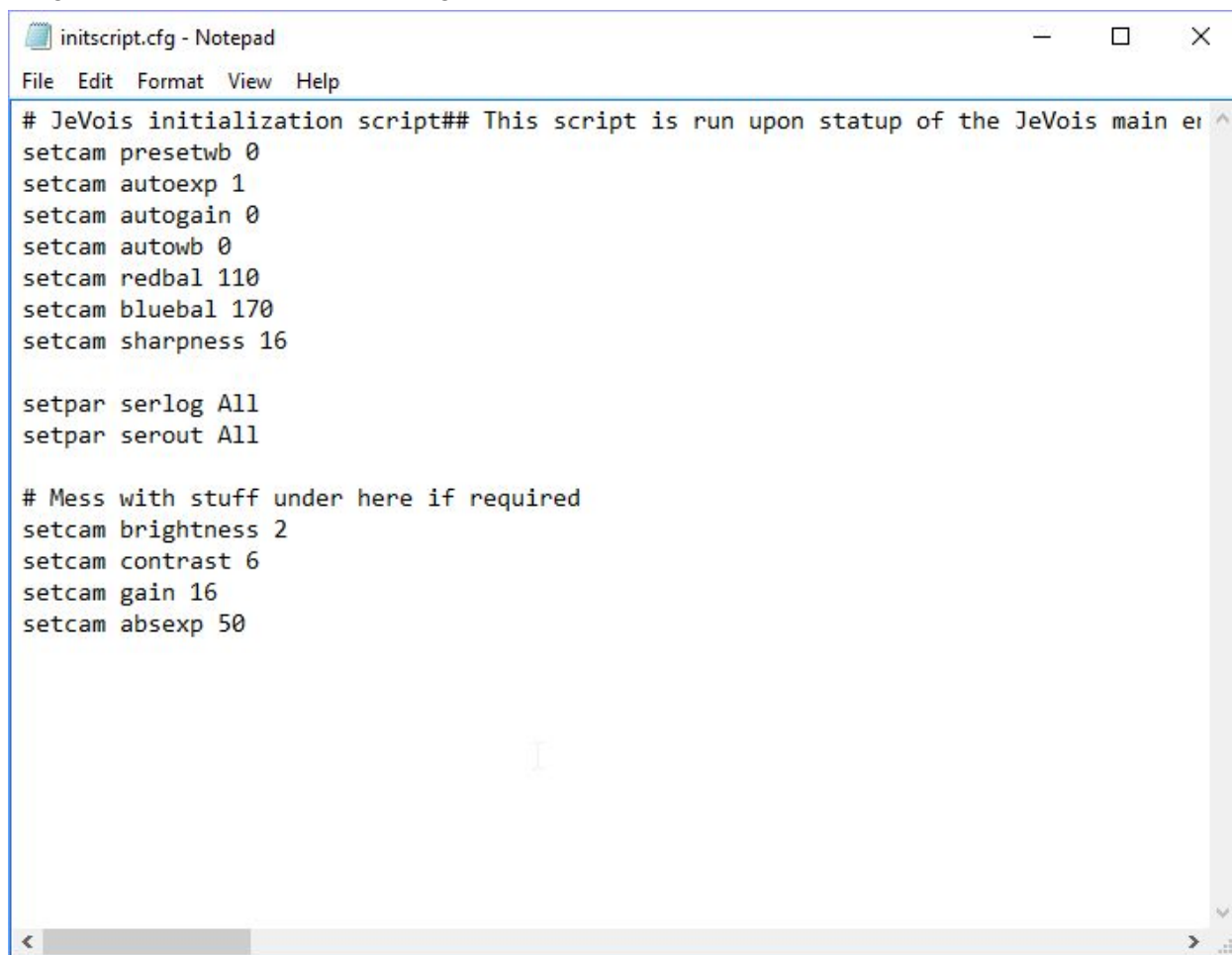
The "Jevois Filesystem" folder

Inside, there are two folders: “config” and “modules”.

Navigate to the JeVois drive (not the project folder) and delete the “config” and “modules” folders. Replace these with the ones from the project folder. This will delete all of the extra examples and replace them with the working vision programs.

File System Details

The “modules” folder can remain untouched for now, but the “config” folder has some interesting things in it. One is the “initscript.cfg” file.



```
# JeVois initialization script## This script is run upon statup of the JeVois main er
setcam presetwb 0
setcam autoexp 1
setcam autogain 0
setcam autowb 0
setcam redbal 110
setcam bluebal 170
setcam sharpness 16

setpar serlog All
setpar serout All

# Mess with stuff under here if required
setcam brightness 2
setcam contrast 6
setcam gain 16
setcam absexp 50
```

“Initscript.cfg”

The initscript file contains the commands that the JeVois will execute upon startup. A full list of commands can be found here under “AVAILABLE CAMERA CONTROLS”:

<http://jevois.org/doc/UserCli.html>

For right now, just worry about the line at the end that says “setcam absexp 50”. This sets the exposure of the camera, or in other words, how much light it takes in per frame. The range is from 1-1000, so the default setting of 50 is quite low. If you are not currently using a bright green

led ring, you may want to change this value to 300 or 400. After startup, these values can be changed via the Serial interface.

Now close the JeVois file system. Type “restart” into the Arduino Serial Monitor to trigger a restart on the JeVois. You will need to close the Serial Monitor and wait a few seconds before being able to reopen it, as the JeVois will not begin Serial communications again until the Monitor has been restarted.

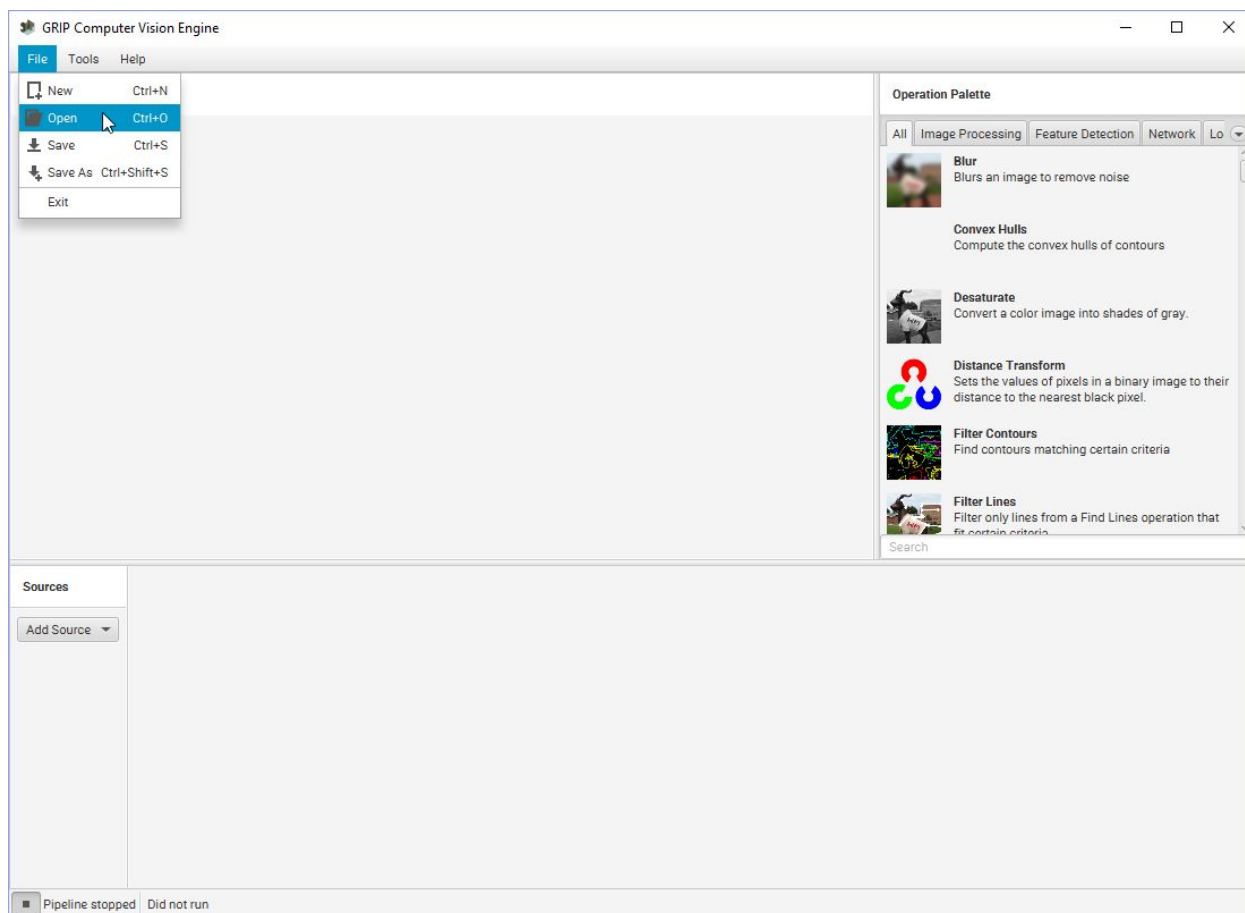
Step 4: Using GRIP and generating the code

It's time to get started with vision!

Starting GRIP

At this point, make sure AMCap is closed until you are instructed to open it, as having it open may cause errors with streaming in GRIP. Do not change the GRIP pipeline by adding or removing blocks, as the final JeVois code will need to be tweaked if edits are made in GRIP. Message Anand Rajamani directly if you have further questions.

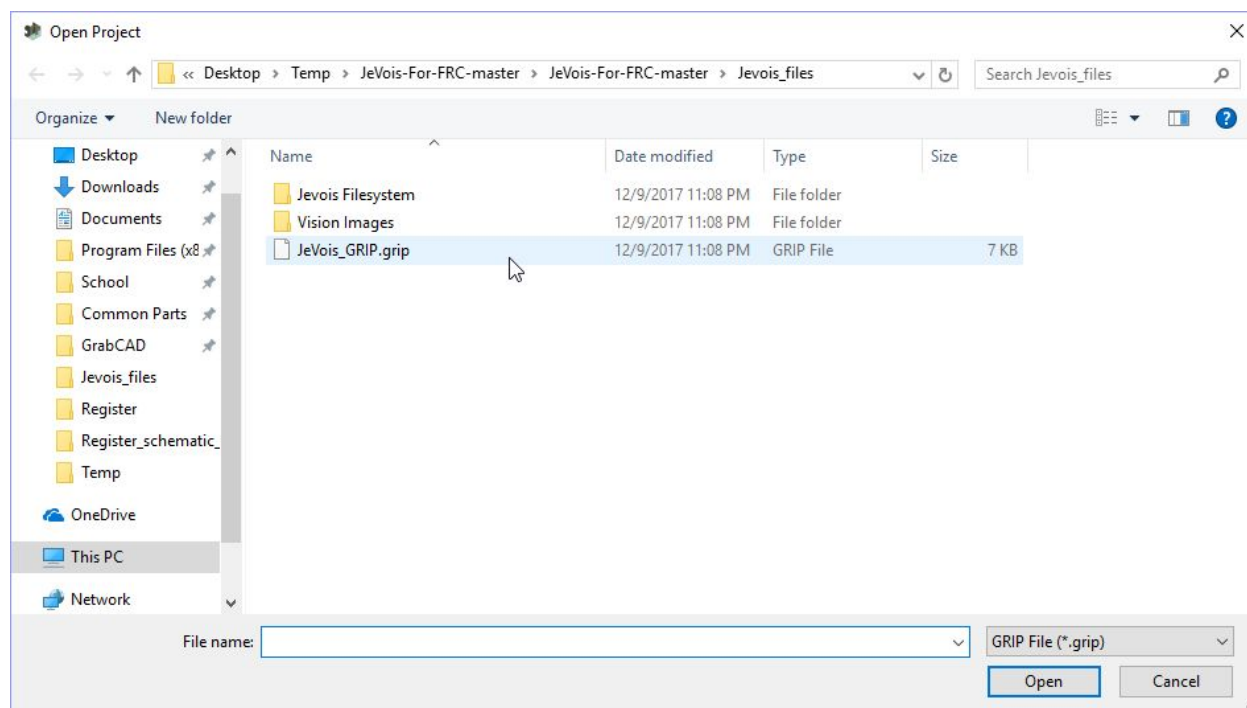
Start up GRIP. Make sure Arduino is also started, with the Serial Monitor open. Once GRIP loads, go to File -> Open.



Opening the GRIP file.

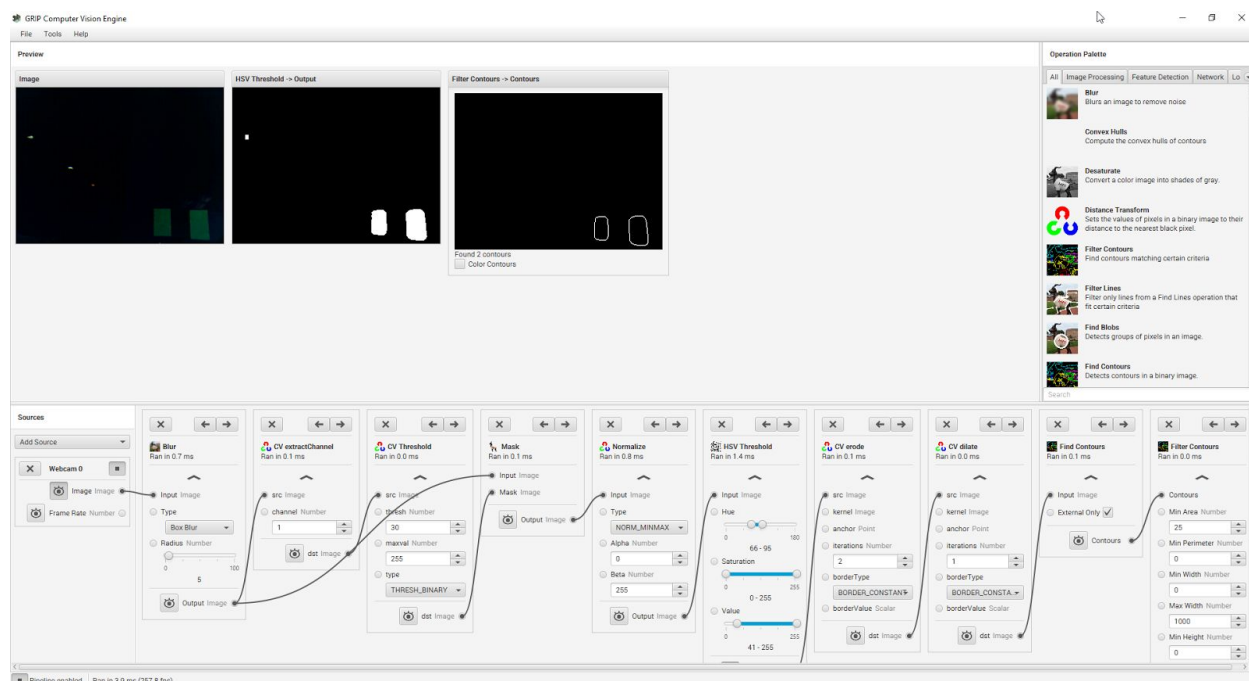
Locate the “JeVois_GRIP.grip” file in
JeVois-For-FRC-master\JeVois-For-FRC-master\Jevois_files. Your file path may be

slightly different, so if you can't find it use the search bar in the top right of the Explorer to find it.



The JeVois_GRIP.grip file

Once it's open, if your JeVois is connected to your computer via USB (which it should be), you'll see a series of video feeds and a whole bunch of stuff at the bottom.

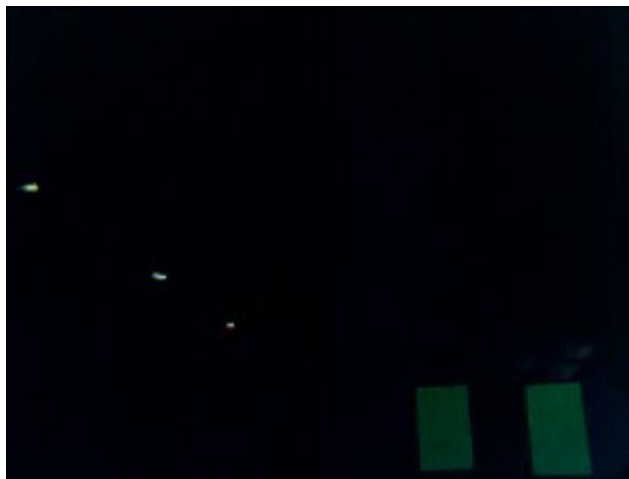


View of GRIP after opening the .grip file

You may see only black or see very little. This is due to the exposure setting in the “initscript.cfg” files from Step 3- a setting of 50 is quite low. For reference, here is a comparison between what the JeVois sees and what a human sees:



A picture of the setup



The way the JeVois sees the same room.

Even the bright light fixtures are just tiny twinkles! Using one or multiple bright green LED rings is critical to getting good performance out of your vision system. A good LED system can block out 99% of all noise, leaving only the retroreflective target within view. Even the one shown above would be classified as perhaps average, but it is suitable for the purposes of this tutorial.

How to use GRIP

There are many tutorials on how to use GRIP on the GRIP homepage like [this one](#), so we won't go into too much detail here on how to use GRIP specifically. We recommend looking at the GRIP tutorials to increase your understanding, however, we will explain what each block at the bottom does. Note that the input image may have changed slightly between the camera screenshots shown here due to accidental bumps.

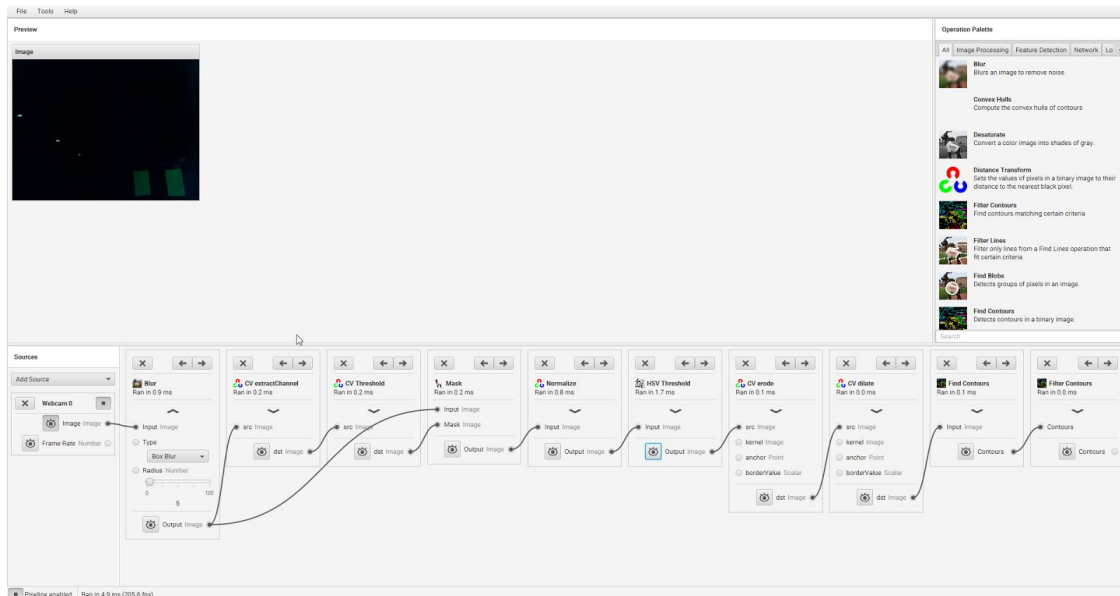
First of all, you can hide or show the output of each block by clicking on the eye icon.



The eye icon. Depressed means that the feed is being shown above the pipeline

We're going to start by minimizing all the filters at the bottom and un-depressing the eyes to hide all the feeds except for the Webcam. Webcam 0 is the JeVois in my case. You may need to click on "Add source" and try Webcam 1 if your computer has a built-in

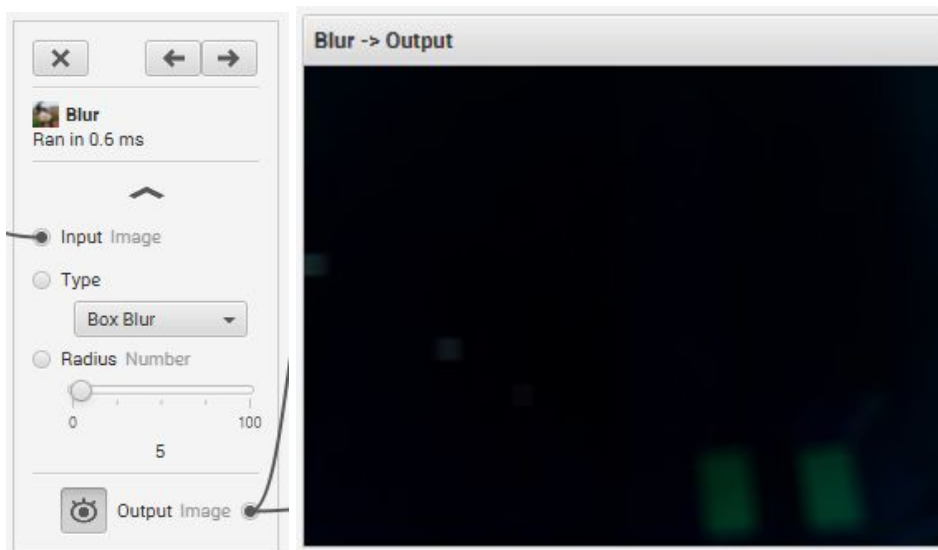
webcam; see the tutorial for GRIP linked above if you have trouble with selecting a webcam.



Using full screen view makes it easier to see the entire pipeline.

The collection of blocks at the bottom is known as a “pipeline”. Each block represents a different operation that is being performed on a frame of video, and feeds into the next block. The curvy lines represent connections between blocks. Let’s take a look at each block in order.

The “Blur” block

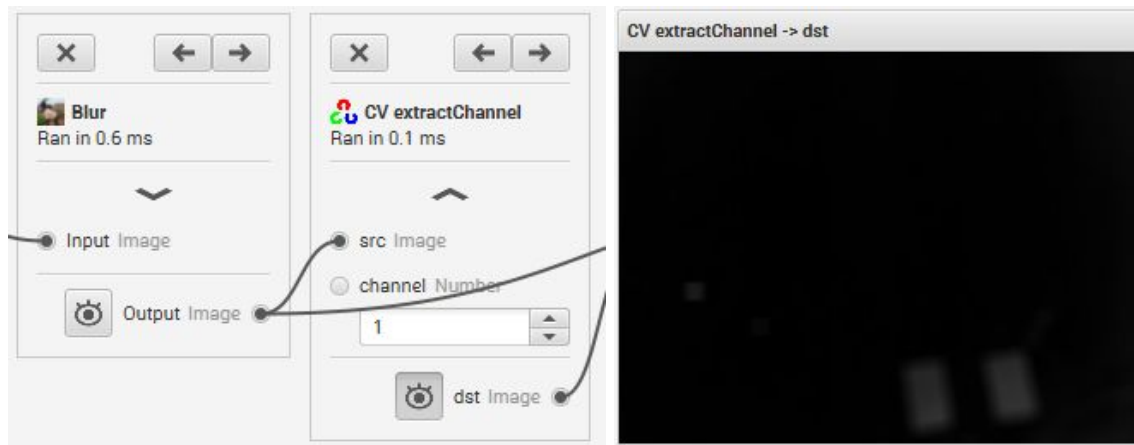


The first block in the pipeline apart from the webcam input is the Blur. Blurring an image helps reduce problems by taking out a lot of white noise or small gaps that might be present in an

image. A radius of 5 works fine for this example, but if the resolution is change you may need to change the radius to reflect that. Play around with the radius to see what it does to the image. Note how we use a “Box Blur” for the type. **Do not change this**, as changing the blur type can screw up the way the JeVois interprets the code much later on. If you want to change the blur type you’ll need to edit code manually later to suit the type.

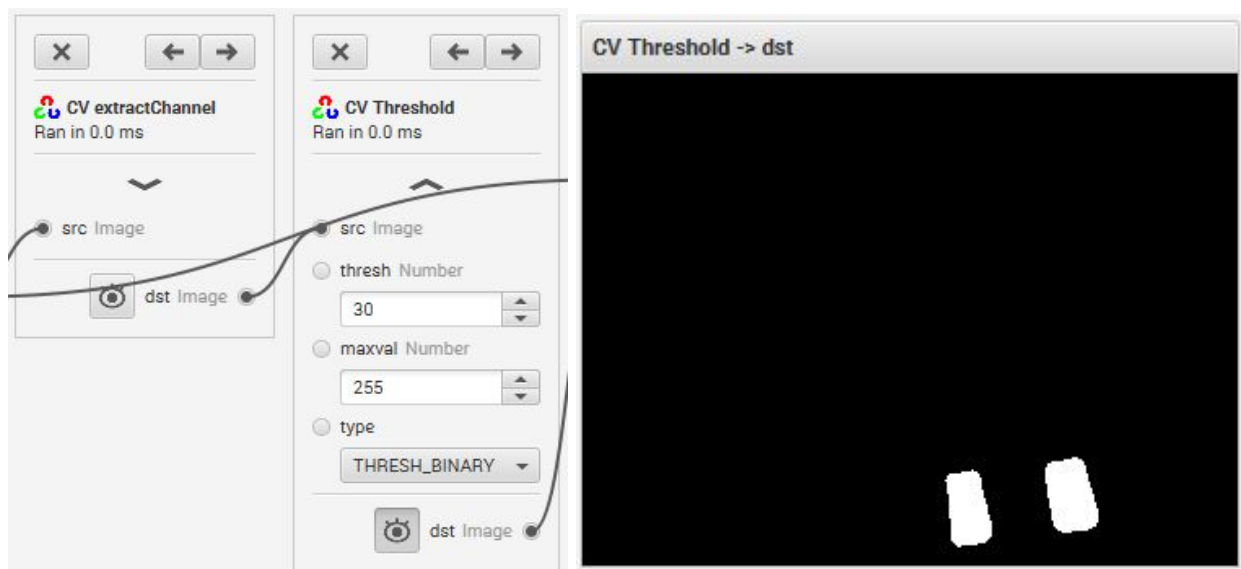
Box Blur is the fastest type of blur and does well at removing noise. More advanced blurs like the Median Filter can be extremely effective at filtering out noise, but take up much more processing, often enough to slow the frame rate down by 10-20 FPS on the JeVois side. If you are ok with a hit like that, use anything up to a Median Filter to see how it affects your results. The Bilateral Filter runs too slowly to justify usage with the JeVois.

The “CV extractChannel” block



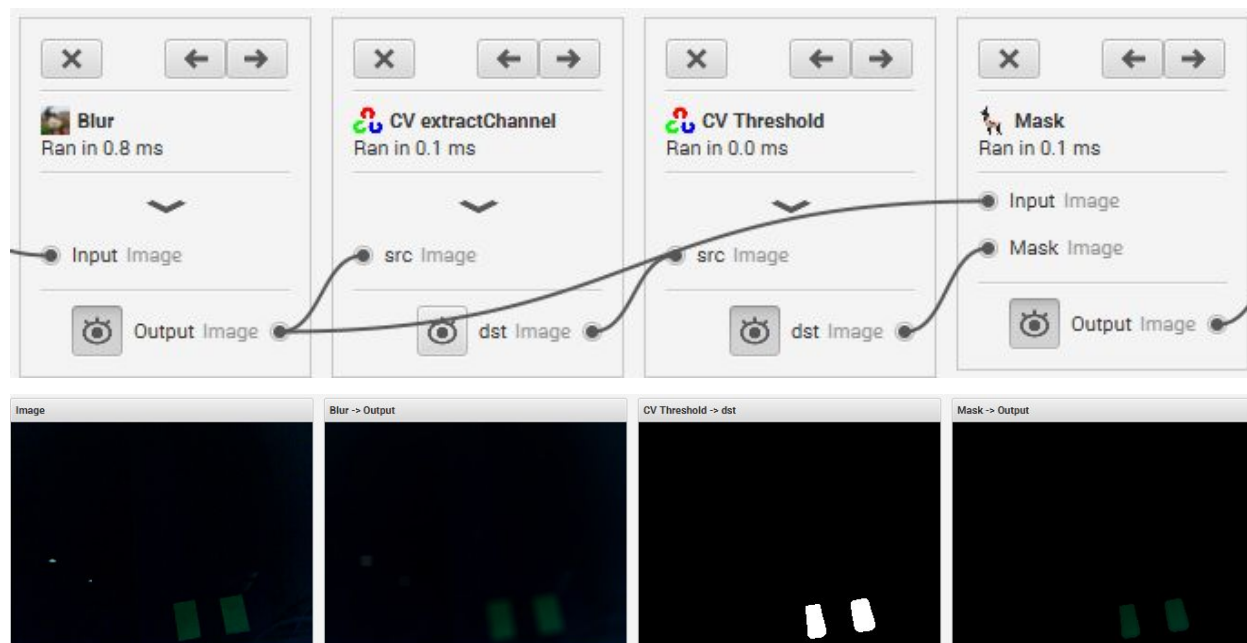
The CV extractChannel block takes out a particular “channel” from the image. One color is one channel, so red, green, and blue all have their own channels. In this case, we are using bright green LEDs, and channel 1 happens to correspond to that. You can change the channels settings and see the effects. Notice how the image is blurred, as the blurred image is what is being fed into the extractChannel block.

The “CV Threshold” block



The CV Threshold removes anything from the image under a certain value. Lighter pixels are higher values than darker ones. In this case, the minimum is set at 30 and the maximum is set at 255 (which is fully white), with a binary threshold being applied. This makes everything from 30-255 white and anything 0-29 black. You’ll need to mess with the threshold until the part of the image you like is all there. Remember, some extra noise isn’t all bad.

The “Mask” block



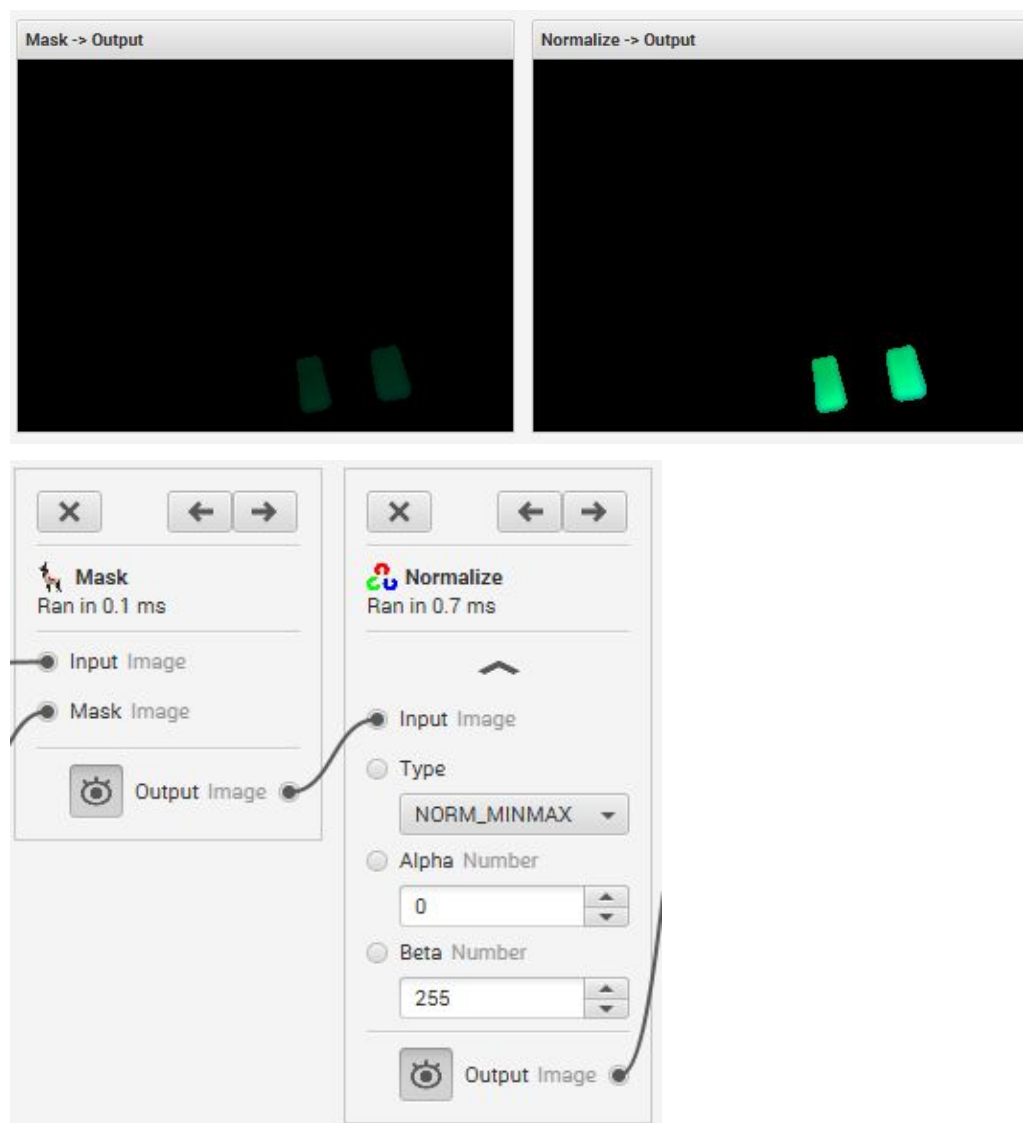
The first stage of the pipeline

The Mask block takes in a binary image (a white/black image, no greys) and uses it to mask off sections of another image. The CV Threshold is the mask, and the original blurred image is the image to be masked. Notice how the small light specks from the light fixtures disappear between the original image and the output of the Mask- this is because the CV Threshold masks off anything not inside that white area. This can be extremely useful when there's a lot of noise stemming from light fixtures and other small light sources.

This seems like a lot of steps to remove a small amount of noise, but it actually barely consumes any processing power. On the computer all of these blocks together ran in about 1ms, or 1,000 FPS - insignificant compared to the JeVois' default 60fps frame rate. The pipeline will run slower on the JeVois than on the computer, but the processing time required is still quite small, on the order of 2ms.

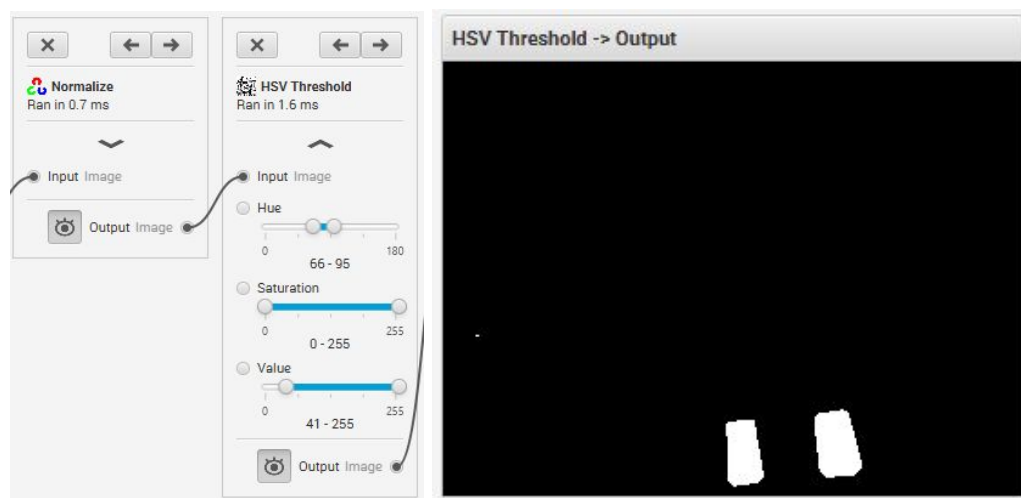
Next, let's look at the target recognition sequence blocks.

The “Normalize” block



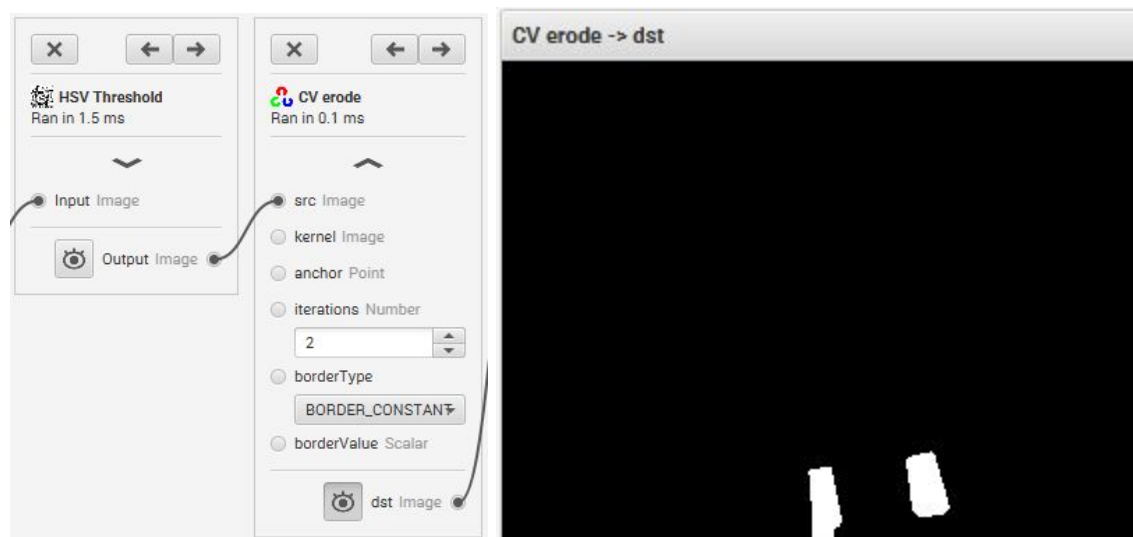
The Normalize block takes the brightest pixel from the input and scales it to a different brightness, then scales all other pixels the same amount. Alpha is what the brightness of the darkest pixel should be, and Beta is the brightness of the lightest pixel. This makes a very crisp image for the next step and stops the distance to the target or ambient light level from being a huge factor.

The “HSV Threshold” block



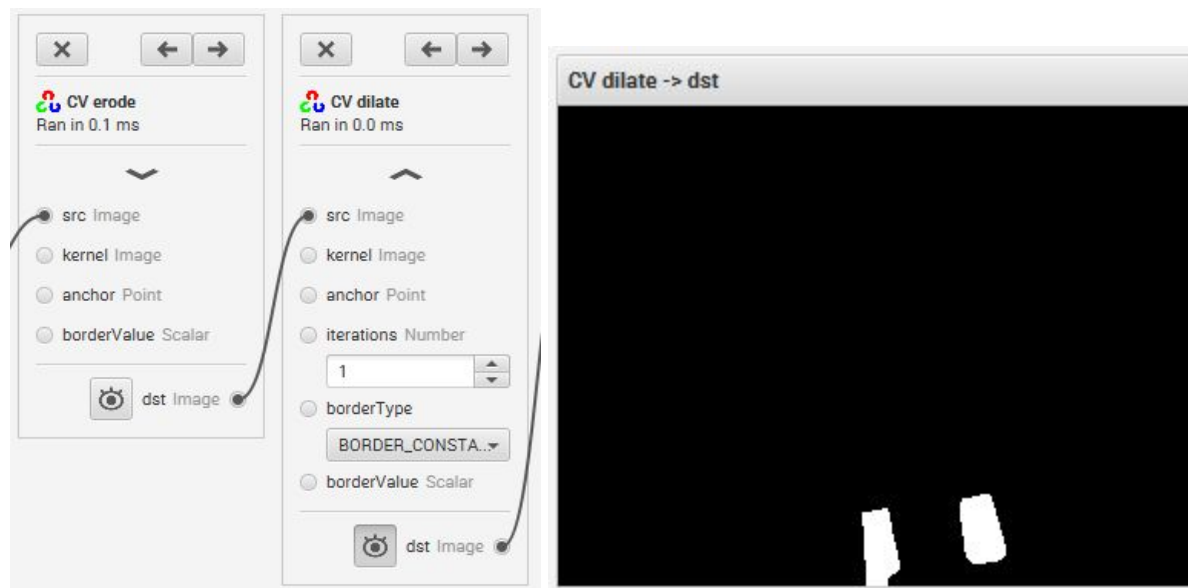
This is where the bulk of the tuning process is. This HSV Threshold filters out anything not within a specified range of Hue, Saturation, and Value. Hue and Saturation are typically tuned first, followed by Value. Value is the brightness level to look for in the image, and is the most important to tune correctly. Hue and Saturation remain relatively constant regardless of how far or close your target is, but Value can change fairly easily. As a result, leaving a bit of extra range on Value may be a good idea.

The “Erode” block



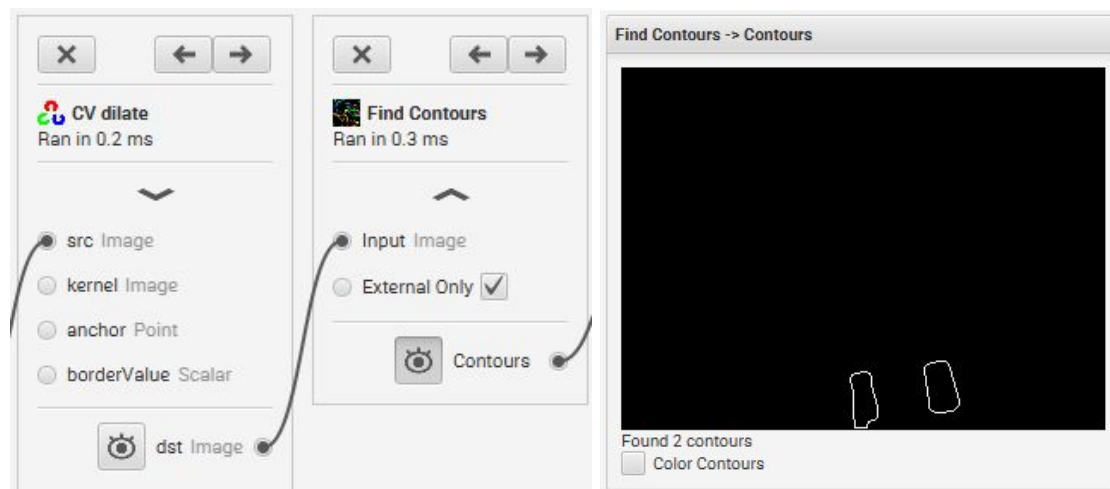
The Erode block eats away at the outside of the shapes left after the HSV threshold. It removes a lot of small white noise. The “iterations” is the number of times it runs; the more it runs, the more area is eaten away by the erode.

The “Dilate” block



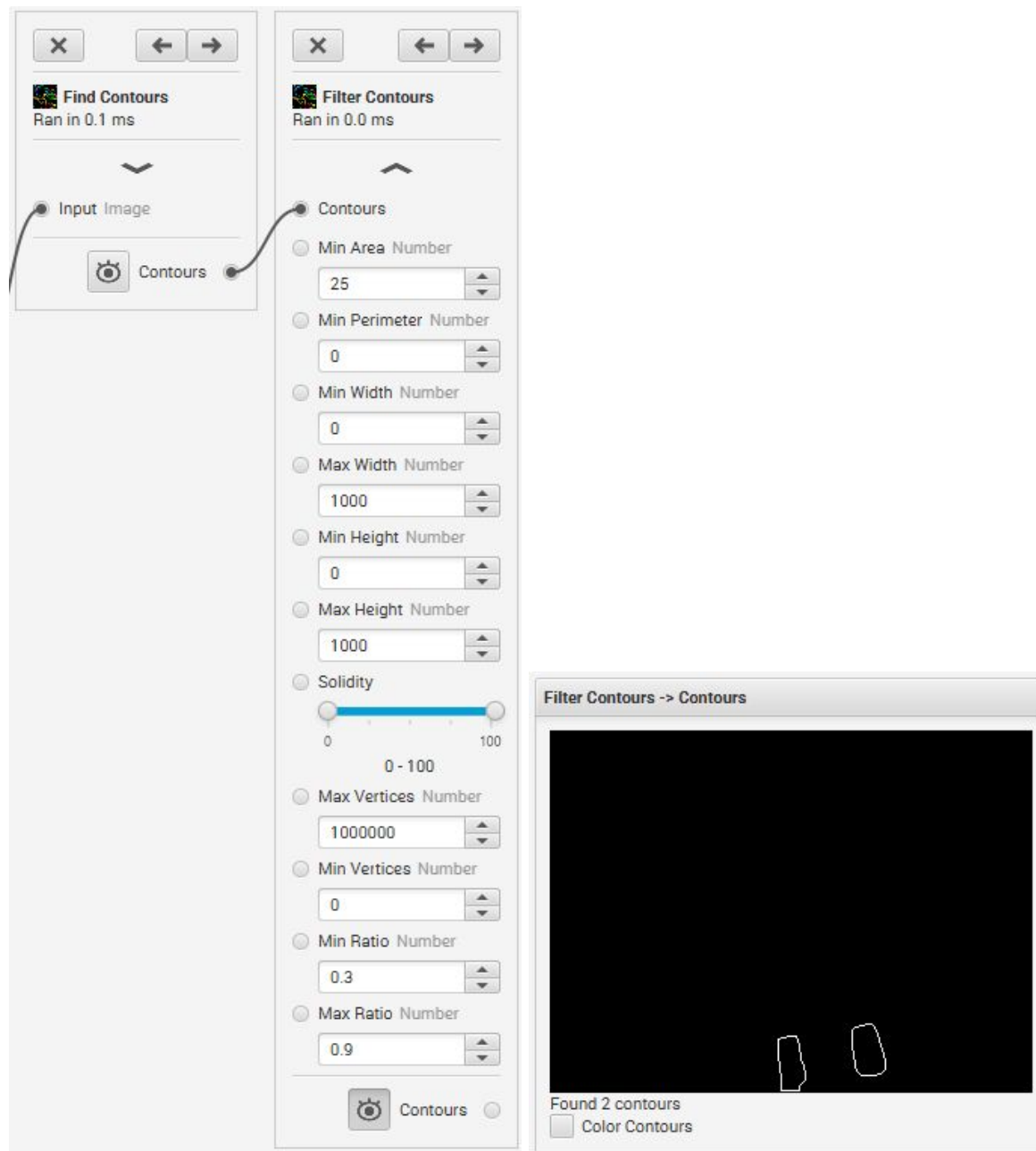
“Dilate” is the opposite of erode. It takes any shapes and enlarges them. This repairs any “damage” done by the erode in the last step. The number of iterations corresponds to the amount the blocks get larger.

The “Find Contours” block



The “Find Contours” block does just that- it finds all contours. A contour is any closed shape. Finding contours is best done after all of the filtering, so we only get the contours we care about. We can do some post-filtering on the JeVois in code, but getting the majority out of the way here is very easy to do by simply adjusting the sliders. Note how the previous blocks all output an “Image”. This outputs “Contours”.

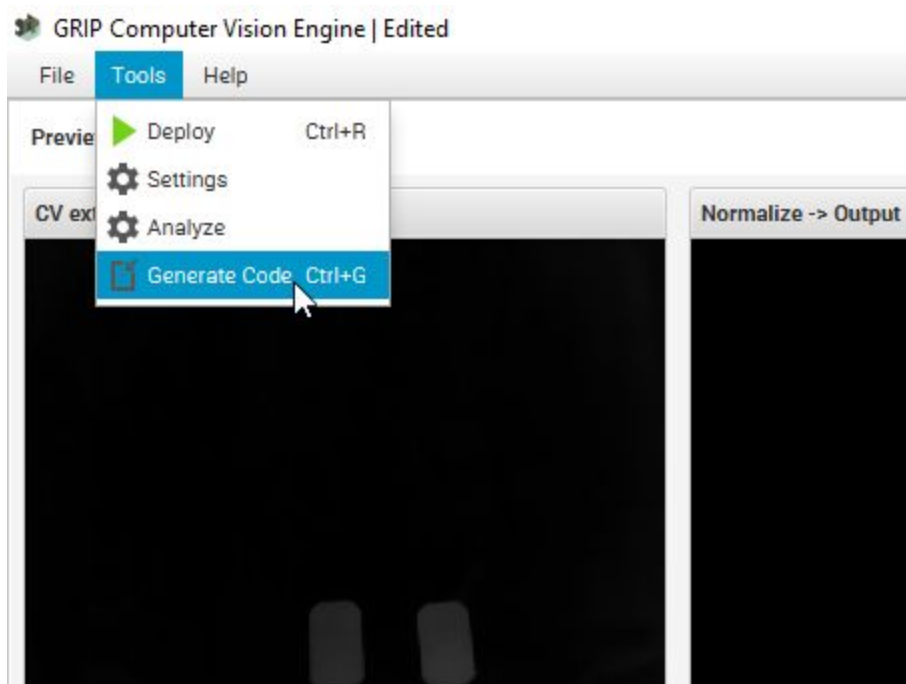
The “Filter Contours” block



The filter contours block filters out contours that we don't want. "Min Area" sets the minimum size of a contour in terms of the number of pixels it encapsulates, filtering out very small contours that made it through the filters in the previous steps. Perimeter, width, etc. all do similar things. Min and Max ratio refer to the minimum and maximum Width/Height ratio of the contours. In this case, we are looking for tall, skinny rectangles, so any W/H ratio above 0.9 is suspicious. Likewise, anything under 0.3 is suspicious purely because it is *too* tall and skinny. This can be a great tool for fitting contours to the shape they will be on the field.

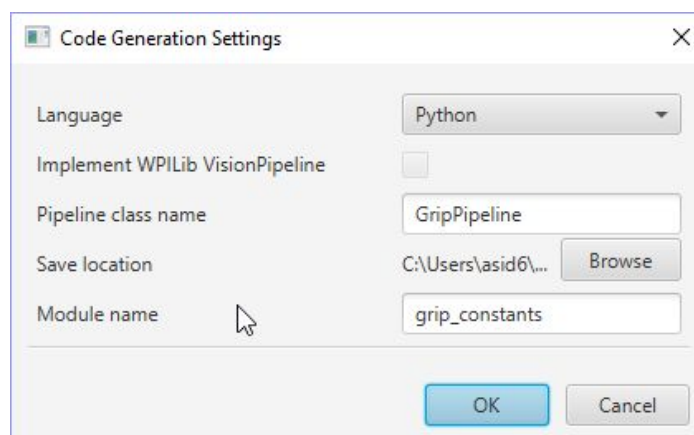
Generating Code

Once you've tuned your GRIP settings to filter out contours that represent the image you are trying to sense, it's time to put that code on the JeVois! First save (ctrl+s), then go to Tools -> Generate Code.



The Generate Code button

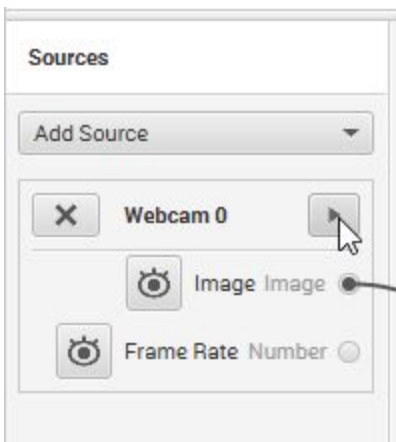
You'll see a small window pop up. Select the folder you want the output code to be in (the JeVois project folder is a good choice) using the "Save location" box and make sure "Python" is selected for the language. The class name can be whatever you want; we will not use it. "Module name" will become the name of our file; call it "grip_constants".



The Code Generation Settings

Putting code on to the JeVois

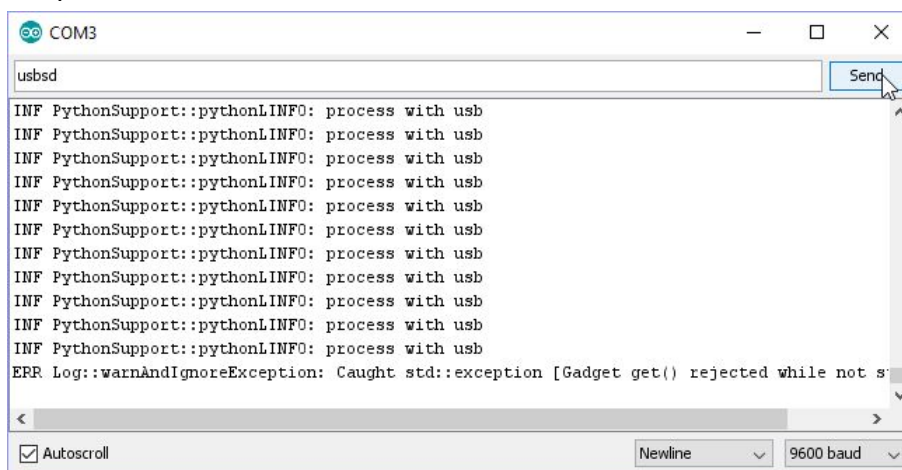
Before continuing, stop any camera feeds to GRIP by clicking on the Stop button on the left side under “Sources”. The Stop button will then turn into a slowly flashing Play button.



A stopped camera feed

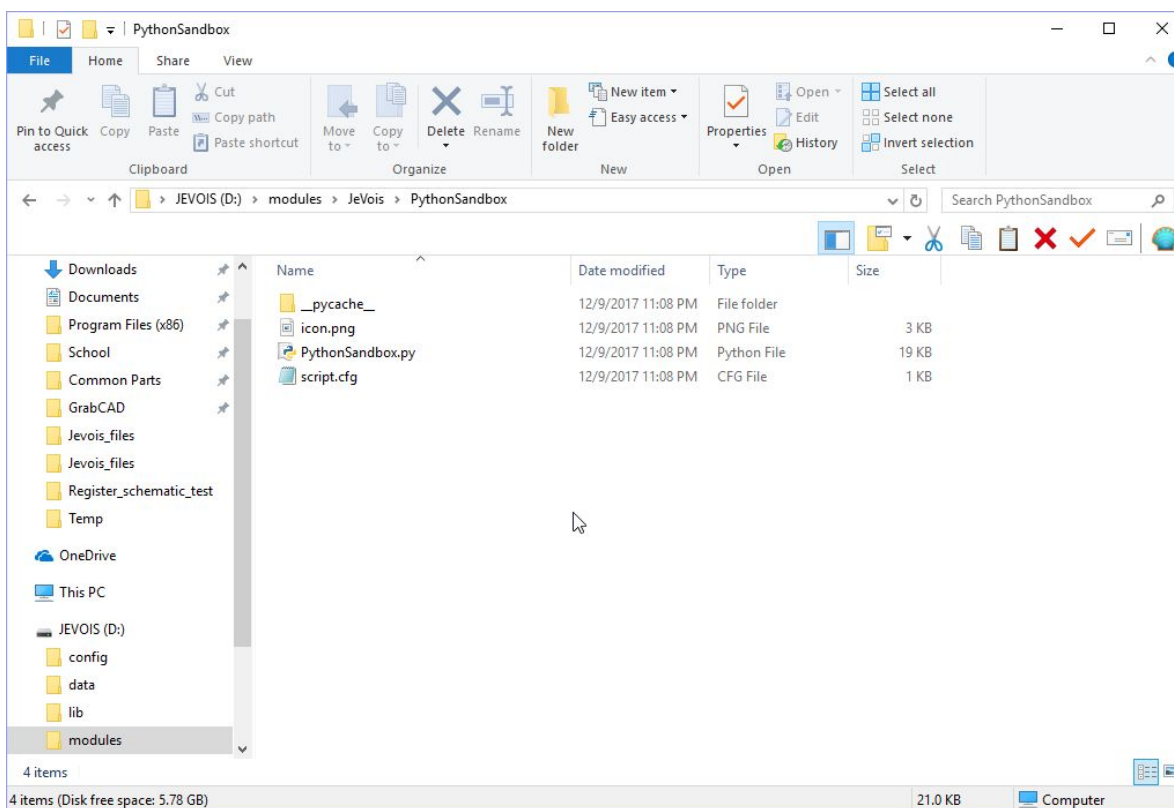
If any camera feed is active, the JeVois will not allow you to access the file system.

Next, go to the Arduino Serial Monitor (which should still be open), type in “usbsd”, and hit Send like we did in Step 2.



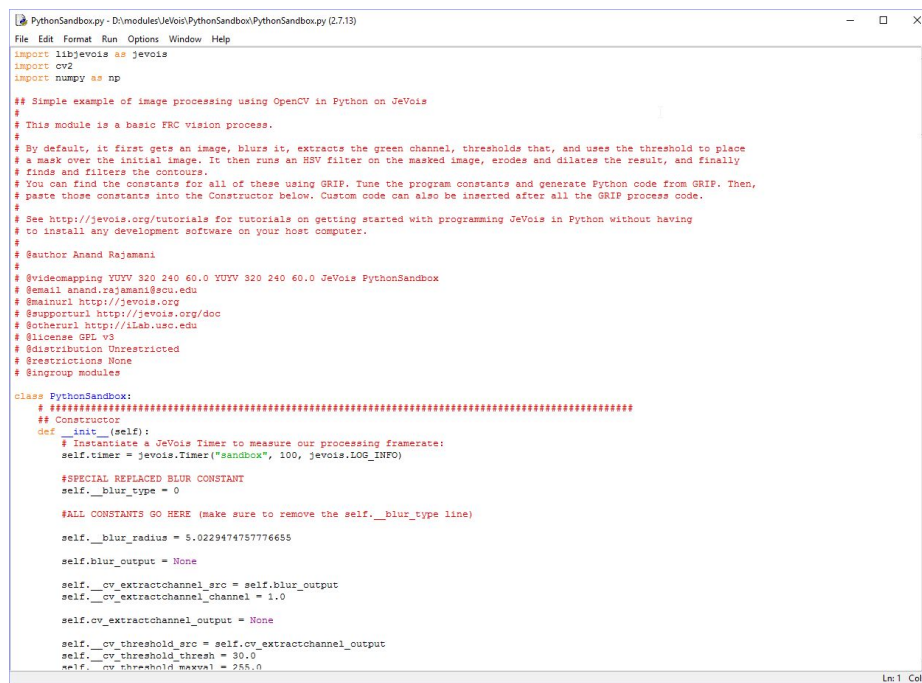
Running usbsd again

Inside the JeVois drive, navigate to the folder Modules -> JeVois -> PythonSandbox.



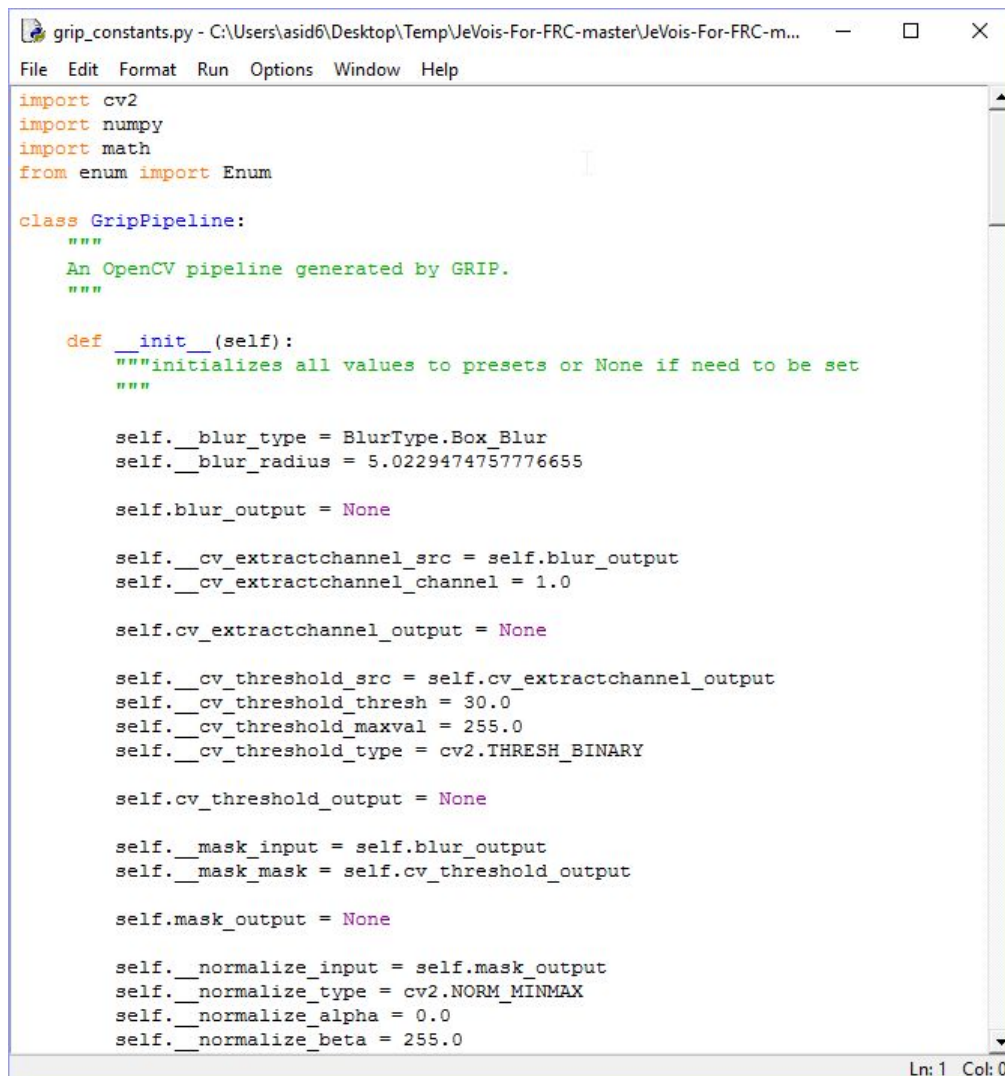
The PythonSandbox folder

Right click on "PythonSandbox.py" and edit it in any text editor. I use IDLE.



Editing "PythonSandbox.py"

Scroll down to where it has the line that says “ALL CONSTANTS GO HERE”.
Now, open up the Python file we generated from GRIP.



The screenshot shows a Python IDE window titled 'grip_constants.py - C:\Users\asid6\Desktop\Temp\JeVois-For-FRC-master\JeVois-For-FRC-m...'. The window contains the following Python code:

```

import cv2
import numpy
import math
from enum import Enum

class GripPipeline:
    """
    An OpenCV pipeline generated by GRIP.
    """

    def __init__(self):
        """initializes all values to presets or None if need to be set
        """

        self.__blur_type = BlurType.Box_Blur
        self.__blur_radius = 5.0229474757776655

        self.blur_output = None

        self.__cv_extractchannel_src = self.blur_output
        self.__cv_extractchannel_channel = 1.0

        self.cv_extractchannel_output = None

        self.__cv_threshold_src = self.cv_extractchannel_output
        self.__cv_threshold_thresh = 30.0
        self.__cv_threshold_maxval = 255.0
        self.__cv_threshold_type = cv2.THRESH_BINARY

        self.cv_threshold_output = None

        self.__mask_input = self.blur_output
        self.__mask_mask = self.cv_threshold_output

        self.mask_output = None

        self.__normalize_input = self.mask_output
        self.__normalize_type = cv2.NORM_MINMAX
        self.__normalize_alpha = 0.0
        self.__normalize_beta = 255.0

```

The status bar at the bottom right of the window indicates 'Ln: 1 Col: 0'.

The GRIP-generated Python code

We are not going to copy over all of the constants for the blocks to the JeVois. Select everything inside of the “def __init__(self)” function EXCEPT for the “self._blur_type” line. **You cannot copy over the blur_type line to the JeVois.**

```
import cv2
import numpy
import math
from enum import Enum

class GripPipeline:
    """
    An OpenCV pipeline generated by GRIP.
    """

    def __init__(self):
        """initializes all values to presets or None if need to be set"""

        self._blur_type = BlurType.Box Blur
        self._blur_radius = 5.0229474757776655

        self.blur_output = None

        self._cv_extractchannel_src = self.blur_output
        self._cv_extractchannel_channel = 1.0

        self.cv_extractchannel_output = None

        self._cv_threshold_src = self.cv_extractchannel_output
        self._cv_threshold_thresh = 30.0
        self._cv_threshold_maxval = 255.0
        self._cv_threshold_type = cv2.THRESH_BINARY

        self.cv_threshold_output = None

        self._mask_input = self.blur_output
        self._mask_mask = self.cv_threshold_output

        self.mask_output = None

        self._normalize_input = self.mask_output
        self._normalize_type = cv2.NORM_MINMAX
        self._normalize_alpha = 0.0
        self._normalize_beta = 255.0

        self._cv_dilate_src = self.cv_erode_output
        self._cv_dilate_kernel = None
        self._cv_dilate_anchor = (-1, -1)
        self._cv_dilate_iterations = 1.0
        self._cv_dilate_bordertype = cv2.BORDER_CONSTANT
        self._cv_dilate_bordervalue = (-1)

        self.cv_dilate_output = None

        self._find_contours_input = self.cv_dilate_output
        self._find_contours_external_only = True

        self.find_contours_output = None

        self._filter_contours_contours = self.find_contours_output
        self._filter_contours_min_area = 25.0
        self._filter_contours_min_perimeter = 0.0
        self._filter_contours_min_width = 0.0
        self._filter_contours_max_width = 1000.0
        self._filter_contours_min_height = 0.0
        self._filter_contours_max_height = 1000.0
        self._filter_contours_solidity = [0.0, 100]
        self._filter_contours_max_vertices = 1000000.0
        self._filter_contours_min_vertices = 0.0
        self._filter_contours_min_ratio = 0.3
        self._filter_contours_max_ratio = 0.9

        self.filter_contours_output = None

    def process(self, source0):
        """
        Runs the pipeline and sets all outputs to new values.
        """
        # Step Blur0:
        self._blur_input = source0
        (self.blur_output) = self._blur(self._blur_input, self._blur_type, se

        # Step CV_extractChannel0:
        self._cv_extractchannel_src = self.blur_output
```

The range to copy

Delete all the constants from the JeVois PythonSandbox.py file.

```
PythonSandbox.py - D:\modules\JeVois\PythonSandbox\PythonSandbox.py (2.7.13)
File Edit Format Run Options Window Help
# finds and filters the contours.
# You can find the constants for all of these using GRIP. Tune the program constants and generate Python code from GRIP. Then,
# paste those constants into the Constructor below. Custom code can also be inserted after all the GRIP process code.
# See http://jevois.org/tutorials for tutorials on getting started with programming JeVois in Python without having
# to install any development software on your host computer.
#
# @author Anand Rajamani
#
# @videomapping YUVY 320 240 60.0 YUYV 320 240 60.0 JeVois PythonSandbox
# @email anand.rajamani@acu.edu
# @mainurl http://jevois.org
# @supporturl http://jevois.org/doc
# @otherurl http://ilab.usc.edu
# @license GPL v3
# @distribution Unrestricted
# @restrictions None
# @ingroup modules

class PythonSandbox:
    """
    Constructor
    """
    def __init__(self):
        # Instantiate a JeVois Timer to measure our processing framerate:
        self.timer = jevois.Timer("sandbox", 100, jevois.LOG_INFO)

        #SPECIAL REPLACED BLUR CONSTANT
        self._blur_type = 0

        #ALL CONSTANTS GO HERE (make sure to remove the self._blur_type line)

        self._blur_radius = 5.0229474757776655

        self.blur_output = None

        self._cv_extractchannel_src = self.blur_output
        self._cv_extractchannel_channel = 1.0

        self.cv_extractchannel_output = None

        self._cv_threshold_src = self.cv_extractchannel_output
        self._cv_threshold_thresh = 30.0
        self._cv_threshold_maxval = 255.0
        self._cv_threshold_type = cv2.THRESH_BINARY

        self.cv_threshold_output = None

        self._mask_input = self.blur_output
        self._mask_mask = self.cv_threshold_output

        self.mask_output = None

        self._normalize_input = self.mask_output
```

Before deleting constants

```

*
# See http://jevois.org/tutorials for tutorials on getting started with programming JeVois in Python without having
# to install any development software on your host computer.
#
# @author Anand Rajamani
#
# @videomapping YUYV 320 240 60.0 YUYV 320 240 60.0 JeVois PythonSandbox
# @email anand.rajamani@scu.edu
# @mainurl http://jevois.org
# @supporturl http://jevois.org/doc
# @otherurl http://ilab.usc.edu
# @license GPL v3
# @distribution Unrestricted
# @restrictions None
# @ingroup modules

class PythonSandbox:
    # *****
    ## Constructor
    def __init__(self):
        # Instantiate a JeVois Timer to measure our processing framerate:
        self.timer = jevois.Timer("sandbox", 100, jevois.LOG_INFO)

        #SPECIAL REPLACED BLUR CONSTANT
        self.__blur_type = 0

        #ALL CONSTANTS GO HERE (make sure to remove the self.__blur_type line)

        #END CONSTANTS

    # *****
    ## Process function with USB output
    def process(self, inframe, outframe):
        # Get the next camera image (may block until it is captured) and here convert it to OpenCV BGR by default. If
        # you need a grayscale image instead, just use getCvGRAY() instead of getCvBGR(). Also supported are getCvRGB()
        # and getCvRGBA():
        source0 = inimg = inframe.getCvBGR()
        outimg = inimg = inframe.getCvBGR()

        # Start measuring image processing time (NOTE: does not account for input conversion time):
        self.timer.start()

        *****
        """
        Runs the pipeline and sets all outputs to new values.
        """
        # Step Blur0:
        self.__blur_input = source0
        (self.__blur_output) = self.__blur(self.__blur_input, self.__blur_type, self.__blur_radius)

        # Step CV_extractChannel0:
        self.__cv_extractchannel_src = self.__blur_output

```

After deleting constants

Note the “#END CONSTANTS” line that marks where to delete up to.

Paste the constants we copied from the GRIP-generated code into where the constants used to be in the PythonSandbox.py file.

```

class PythonSandbox:
    # *****
    ## Constructor
    def __init__(self):
        # Instantiate a JeVois Timer to measure our processing framerate:
        self.timer = jevois.Timer("sandbox", 100, jevois.LOG_INFO)

        #SPECIAL REPLACED BLUR CONSTANT
        self.__blur_type = 0

        #ALL CONSTANTS GO HERE (make sure to remove the self.__blur_type line)

        self.__blur_radius = 5.0229474757776655

        self.__blur_output = None

        self.__cv_extractchannel_src = self.__blur_output
        self.__cv_extractchannel_channel = 1.0

        self.__cv_extractchannel_output = None

        self.__cv_threshold_src = self.__cv_extractchannel_output
        self.__cv_threshold_thresh = 30.0
        self.__cv_threshold_maxval = 255.0
        self.__cv_threshold_type = cv2.THRESH_BINARY

        self.__cv_threshold_output = None

        self.__mask_input = self.__blur_output
        self.__mask_mask = self.__cv_threshold_output

```

The pasted code

Save the Sandbox.py file and close it. Type the “restart” command into the Arduino Serial Monitor and let the JeVois restart. Close the Serial Monitor and wait a few seconds before reopening it to let the JeVois start again.

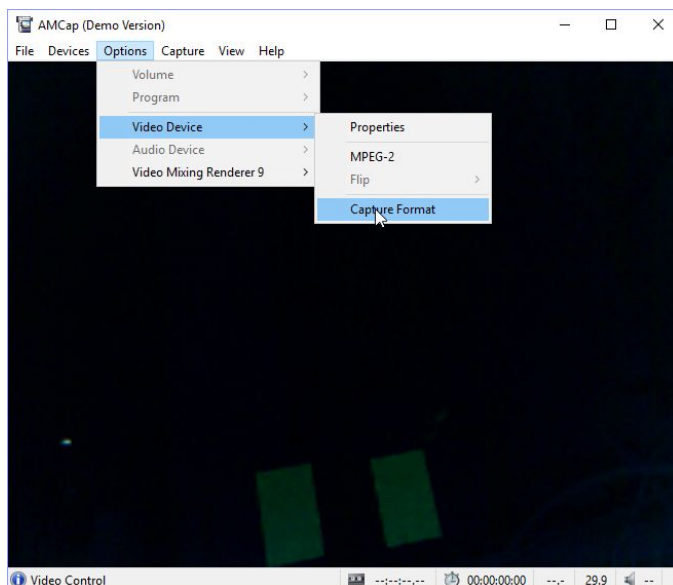
Congratulations, you now have vision code on your JeVois! If you need to re-tune your vision program, simply re-tune the GRIP code using the sliders and dropdown menus, generate code again, and copy/paste the constants!

You cannot tune on the fly when using Python this way, but if you have Linux there is JeVois documentation that details how to program the JeVois using C++ or Python.

Step 5: Running vision code

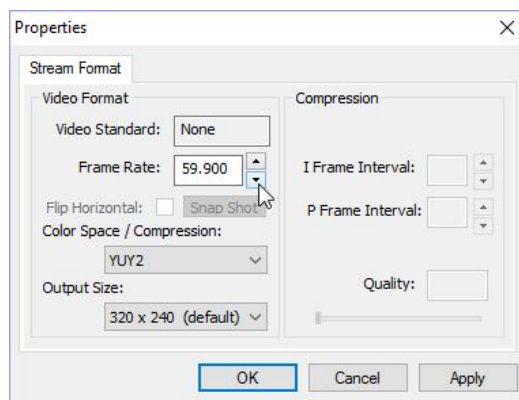
To view your JeVois vision code stream, open up AMCap. If it fails to open, make sure no other streaming programs are running. Otherwise, try restarting the JeVois in case it was not restarted properly. Make sure the correct webcam is selected in the “Devices” menu.

By default, it will show you a camera feed of what the JeVois is seeing. Click on Options -> Video Device -> Capture Format to change the output format.



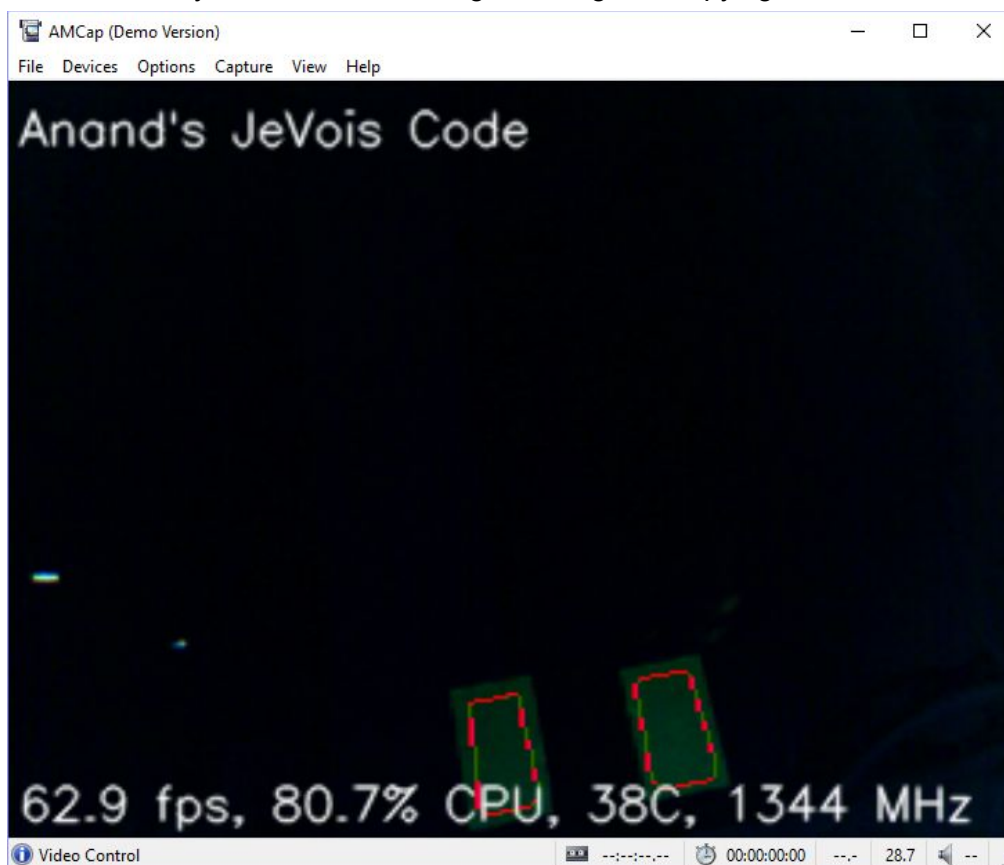
The AMCap screen Capture Format button

The frame rate on the window can be changed to 59.9 or 60.0 FPS. 60.0 FPS simply shows a camera feed, and is the default one that GRIP selects as an input for tuning. 59.9 FPS will show the output of the vision program, which highlights the two largest contours in red by default and displays the effective framerate of the vision program. Clicking the up and down arrows next to framerate will toggle between the two modes.



The Capture Format window

Hit “Ok” to watch your vision program in action! If you feel like you need to re-tune it, Just go back to GRIP and tune your sliders before regenerating and copying the code.



Vision at work recognizing retroreflective tape

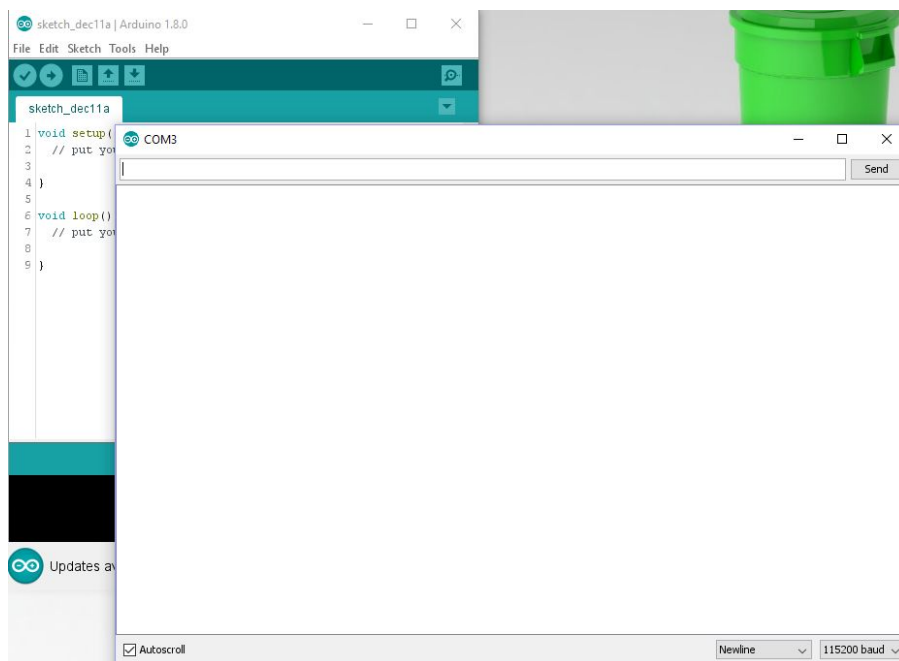
The default program recognizes contours according to the settings you set in GRIP, outlines them in red, and sends the location and area of all contours over the hardware Serial and USB from largest area to smallest area.

For most application this is enough to have working vision, but for advanced users code tweaking can be performed. See [Step 7](#) for more details.

Step 6: Adjusting camera settings (exposure, brightness, etc.)

Now that you've got your basic code working, you need to tune it! First, let's tune the exposure on our camera. Note that for this section, you **can** have AMCap or GRIP running to view the camera feed as you adjust the parameters over the Serial Monitor. GRIP is suggested so that you can also tune parameters if necessary.

Plug in the JeVois. Then, open Arduino and the Arduino Serial Monitor. Note that the JeVois takes a few seconds to show up in the list of COM Ports.



The Arduino Serial Monitor. Note that Newline and 115200 baud are shown in the bottom right.

I usually type “help” and Send that first to make sure the JeVois is connected. If it doesn't respond, closing the Serial Monitor and checking the COM port will fix the problem.

There are a number of commands you can use to tune the camera settings, but the most useful are setting exposure and setting gain. It's also possible to adjust the white balance if you check the JeVois docs here: <http://jevois.org/doc/>.

When you send the “help” command, the JeVois will send back lots of information. At the end of this is the set of camera controls.

AVAILABLE CAMERA CONTROLS:

```

- brightness [int] min=-3 max=3 step=1 def=0 curr=2
- contrast [int] min=0 max=6 step=1 def=3 curr=6
- saturation [int] min=0 max=4 step=1 def=2 curr=2
- autowb [bool] default=1 curr=0
- dwb [int] min=0 max=1 step=1 def=0 curr=0
- redbal [int] min=0 max=255 step=1 def=128 curr=110
- bluebal [int] min=0 max=255 step=1 def=128 curr=170
- autogain [bool] default=1 curr=0
- gain [int] min=16 max=1023 step=1 def=16 curr=16
- hflip [bool] default=0 curr=0
- vflip [bool] default=0 curr=0
- powerfreq [menu] values 0:disabled 1:50hz 2:60hz curr=2
- sharpness [int] min=0 max=32 step=1 def=6 curr=6
- autoexp [menu] values 0:auto 1:manual curr=1
- absexp [int] min=1 max=1000 step=1 def=1000 curr=500
- presetwb [menu] values 0>manual 1:auto 2:incandescent 3:fluorescent 4:fluorescent_h 5:horizon 6:daylight 7:flash 8:cloudy 9:shade curr=0
OK

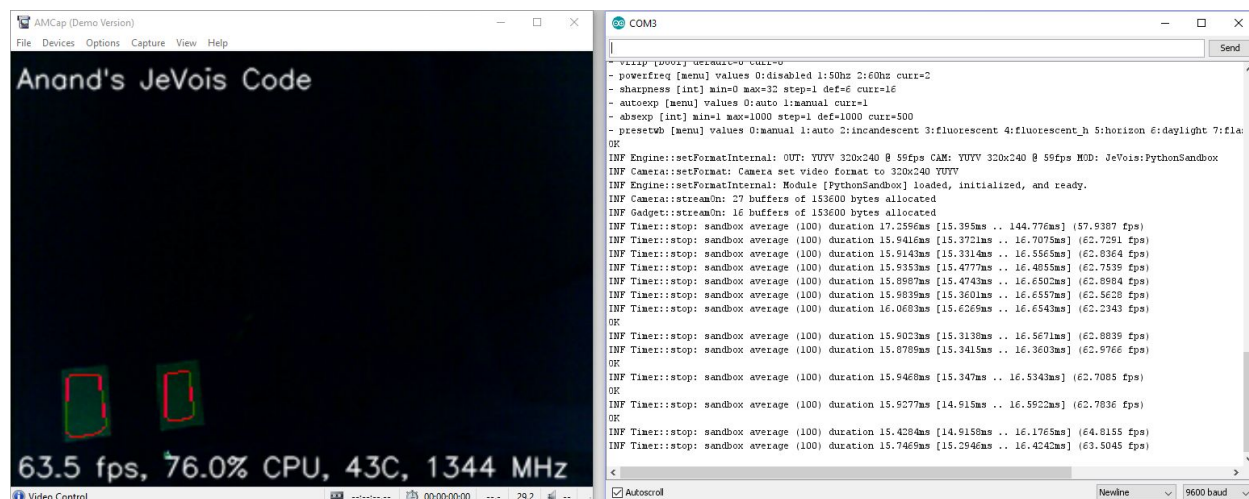
```

Camera controls

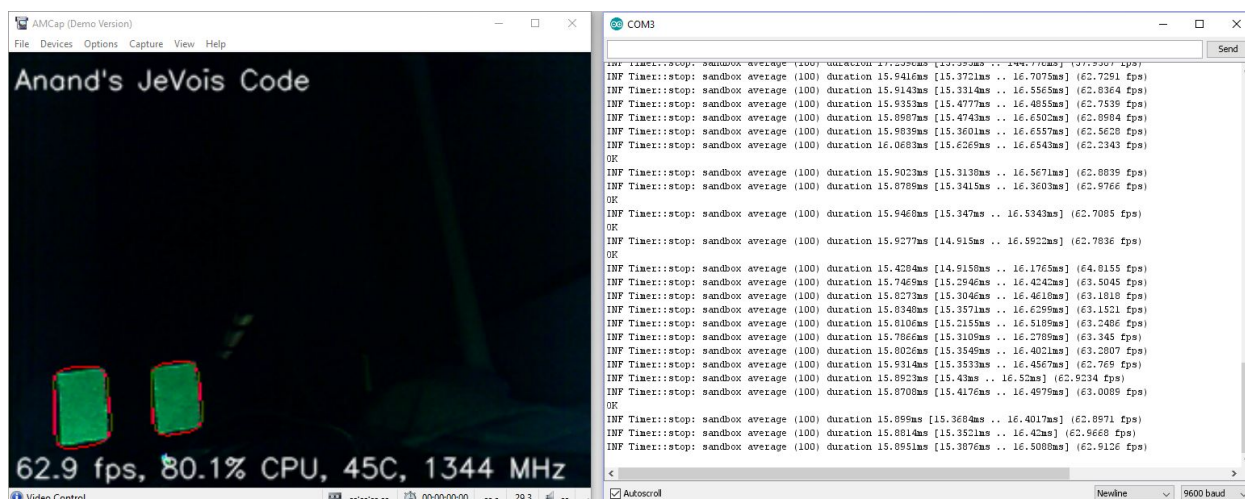
The [“initscript.cfg” file](#) contains several of these commands, to be executed upon startup. “setcam” tells the JeVois that you want to set a camera setting. So to set a camera control, type in this format:

```
setcam <control> <value>
```

For example, “setcam absexp 100” would set the exposure to 100 (on a scale of 1-1000). For most applications the only thing you’ll need to set is the exposure via the “setcam absexp <number>” command, but it’s easy to experiment with over the Serial monitor. Experiment with the exposure until you find a setting that works consistently.

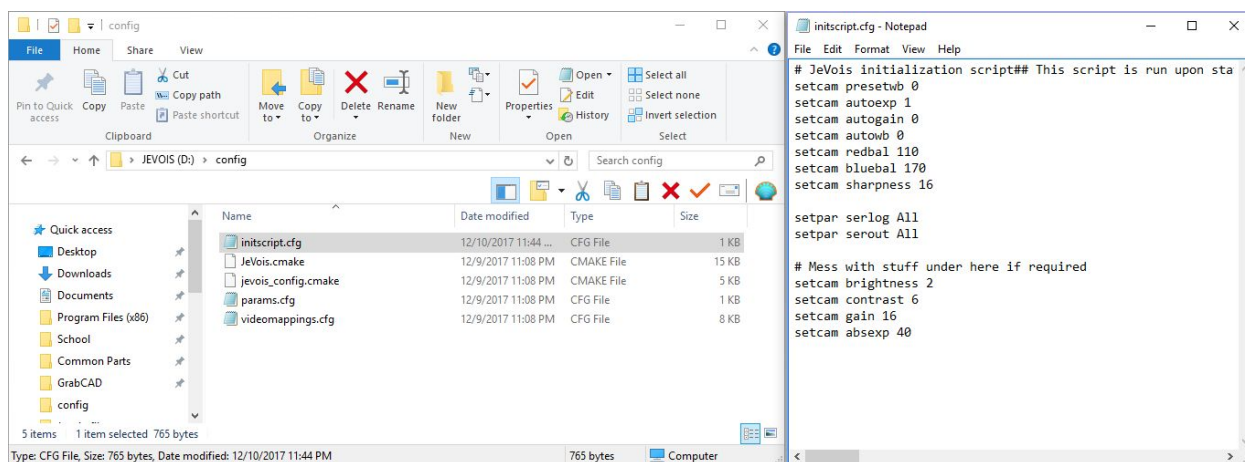


The camera view before tuning exposure



The camera view after tuning exposure

Once you've found a setting that you like, where the target is consistently found by the camera, close AMCap and any other streams like GRIP. Then send the "usbsd" command to open up the JeVois filesystem. There, under config/initscript.cfg you can edit the "setcam absexp 50" line to be whatever value of exposure you found worked the best.



Setting the final exposure in the initscript.cfg file.

In the next section, we'll discuss how to tweak the code.

Step 7: Tweaking the JeVois code

For 99% of users there should be no reason to have to tweak the code found on the JeVois. However, if you want to run custom OpenCV functions like a medianBlur or perform more post-processing on the JeVois, you'll need to edit the vision program that's running on the JeVois.

Changing the Blur style

The only part of the GRIP pipeline Steps 1-6 do not give you control over is the Blur type. This is because GRIP tries to generate enums for the blur type, which are not supported by the JeVois.