

Off Board Vision Processing in Java

This article describes how to get OpenCV, the WPILib Vision library, and NetworkTables, and how to build Java vision targeting programs that run on coprocessors that run alongside the roboRIO. This currently supports the following platforms:

- Windows
- Raspberry Pi running Raspbian
- Generic Armhf devices (such as the BeagleBone Black or the Jetson)

It has been designed to be easy to setup and use, and only needs a few minor settings to pick which system you want to be ran on. It has samples for interfacing with NetworkTables and WPILib Vision Library (CsCore) from any device, along with performing OpenCV operations.

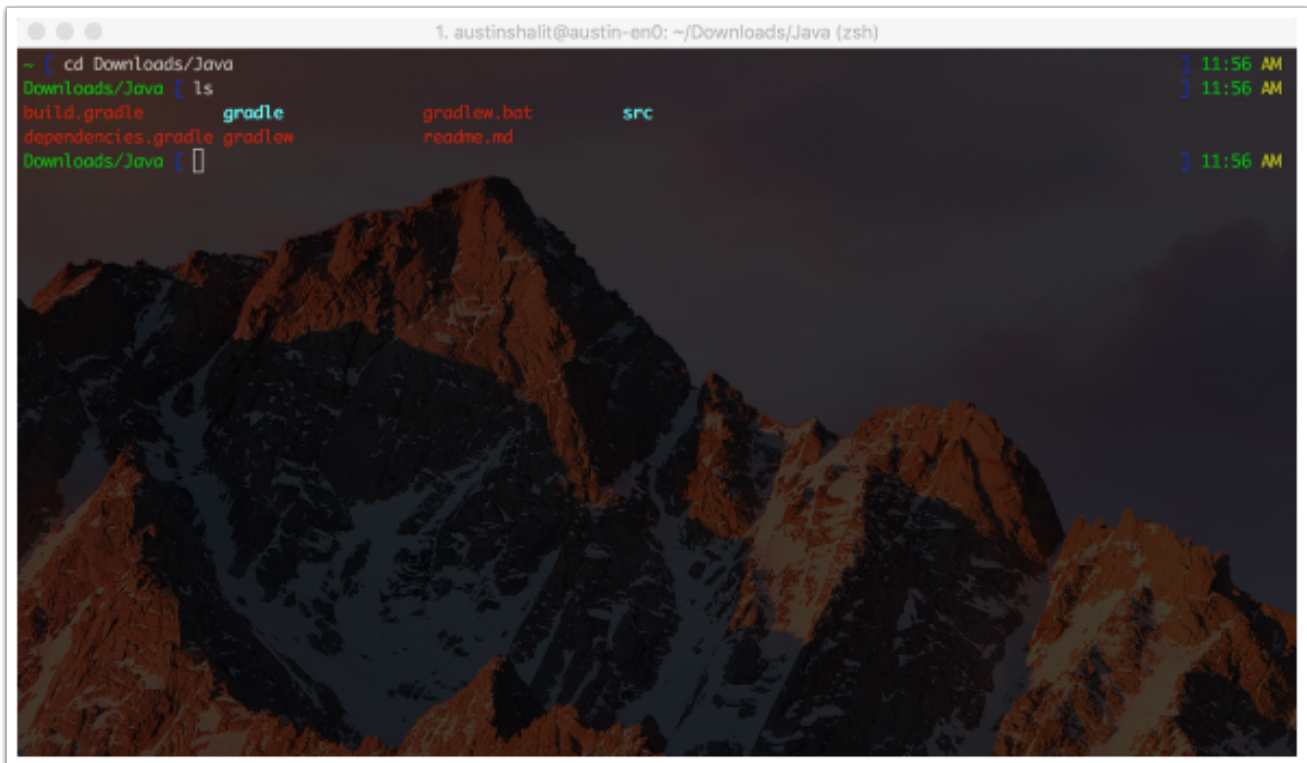
The system is setup to use Gradle to run the builds. Gradle is a cross platform build system, which only requires Java to run.

Getting Started

There are five steps to get your coprocessor configured for vision processing as shown below. This is just a template. You will need to add meaningful code to it for it to do what you want. After following the steps in this document you will have a zip file that you can copy on to your coprocessor, extract, and run.

1. You can develop your vision program on a laptop or your coprocessor. Download the zip file located at <https://github.com/wpilibsuite/VisionBuildSamples/releases> to the device where you want to do your development.
2. Extract the zip and open a command prompt to the extracted folder

Off Board Vision Processing in Java

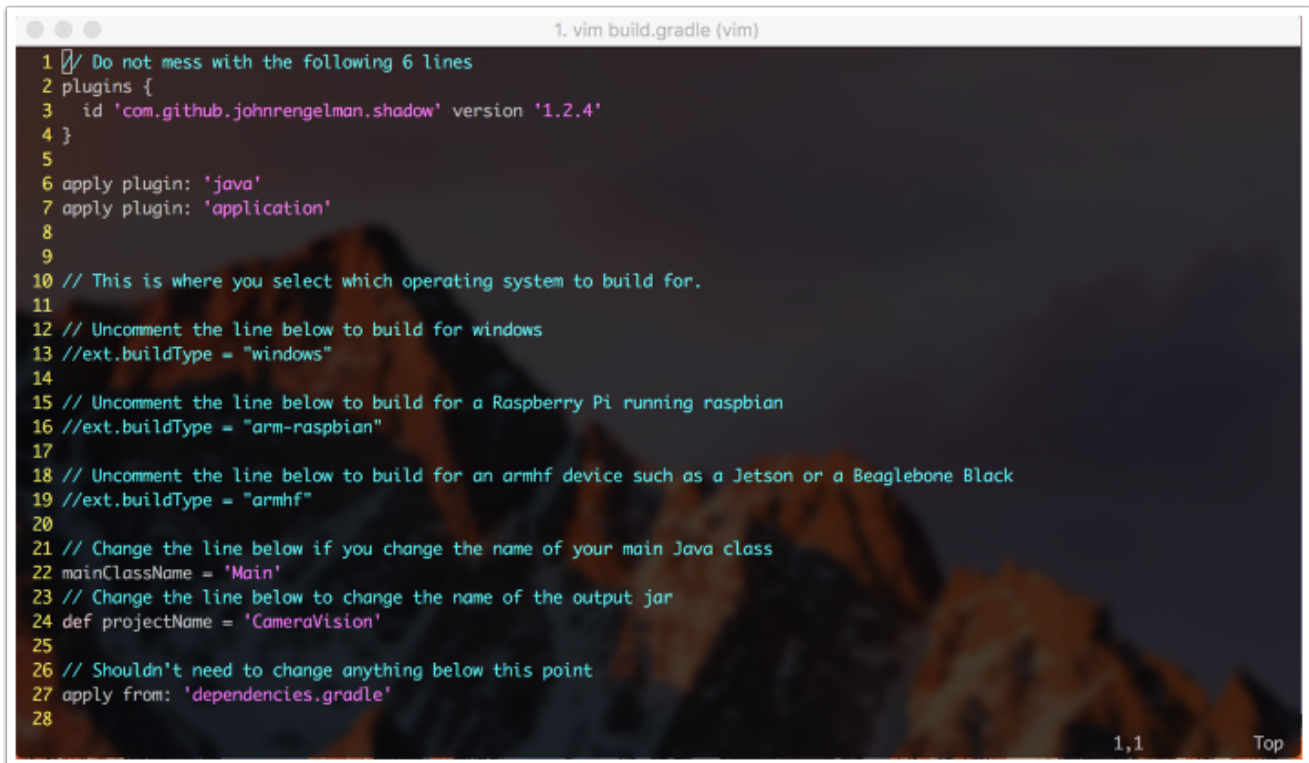


Selecting which system to build for

As there is no way to autodetect which system you want to build for, such as building for a Raspberry Pi on a windows desktop, you have to manually select which system you want to build for.

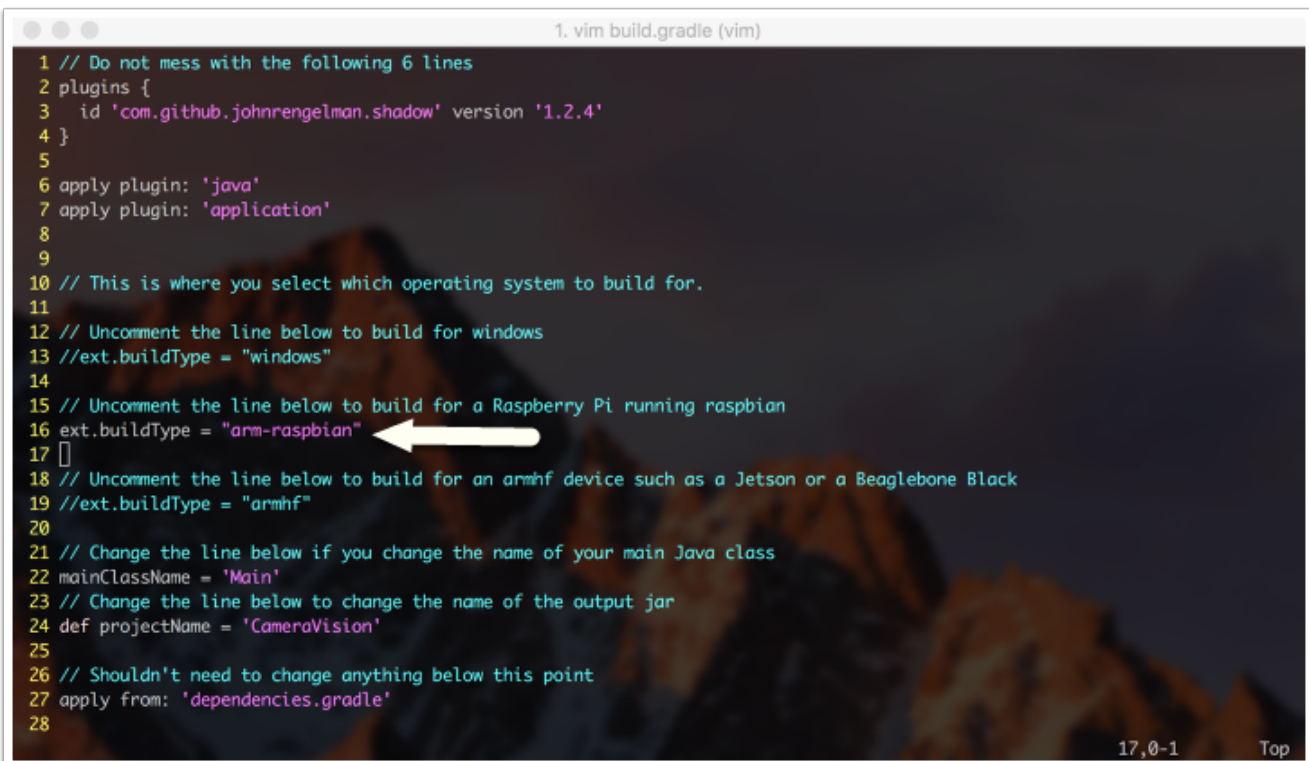
1. Open the `build.gradle` file

Off Board Vision Processing in Java



```
1 vim build.gradle (vim)
1 // Do not mess with the following 6 lines
2 plugins {
3   id 'com.github.johnrengelman.shadow' version '1.2.4'
4 }
5
6 apply plugin: 'java'
7 apply plugin: 'application'
8
9
10 // This is where you select which operating system to build for.
11
12 // Uncomment the line below to build for windows
13 //ext.buildType = "windows"
14
15 // Uncomment the line below to build for a Raspberry Pi running raspbian
16 //ext.buildType = "arm-raspbian"
17
18 // Uncomment the line below to build for an armhf device such as a Jetson or a Beaglebone Black
19 //ext.buildType = "armhf"
20
21 // Change the line below if you change the name of your main Java class
22 mainClassName = 'Main'
23 // Change the line below to change the name of the output jar
24 def projectName = 'CameraVision'
25
26 // Shouldn't need to change anything below this point
27 apply from: 'dependencies.gradle'
28
```

2. To select your device uncomment the line it is on by removing the `//`



```
1 vim build.gradle (vim)
1 // Do not mess with the following 6 lines
2 plugins {
3   id 'com.github.johnrengelman.shadow' version '1.2.4'
4 }
5
6 apply plugin: 'java'
7 apply plugin: 'application'
8
9
10 // This is where you select which operating system to build for.
11
12 // Uncomment the line below to build for windows
13 //ext.buildType = "windows"
14
15 // Uncomment the line below to build for a Raspberry Pi running raspbian
16 ext.buildType = "arm-raspbian"
17
18 // Uncomment the line below to build for an armhf device such as a Jetson or a Beaglebone Black
19 //ext.buildType = "armhf"
20
21 // Change the line below if you change the name of your main Java class
22 mainClassName = 'Main'
23 // Change the line below to change the name of the output jar
24 def projectName = 'CameraVision'
25
26 // Shouldn't need to change anything below this point
27 apply from: 'dependencies.gradle'
28
```

Off Board Vision Processing in Java

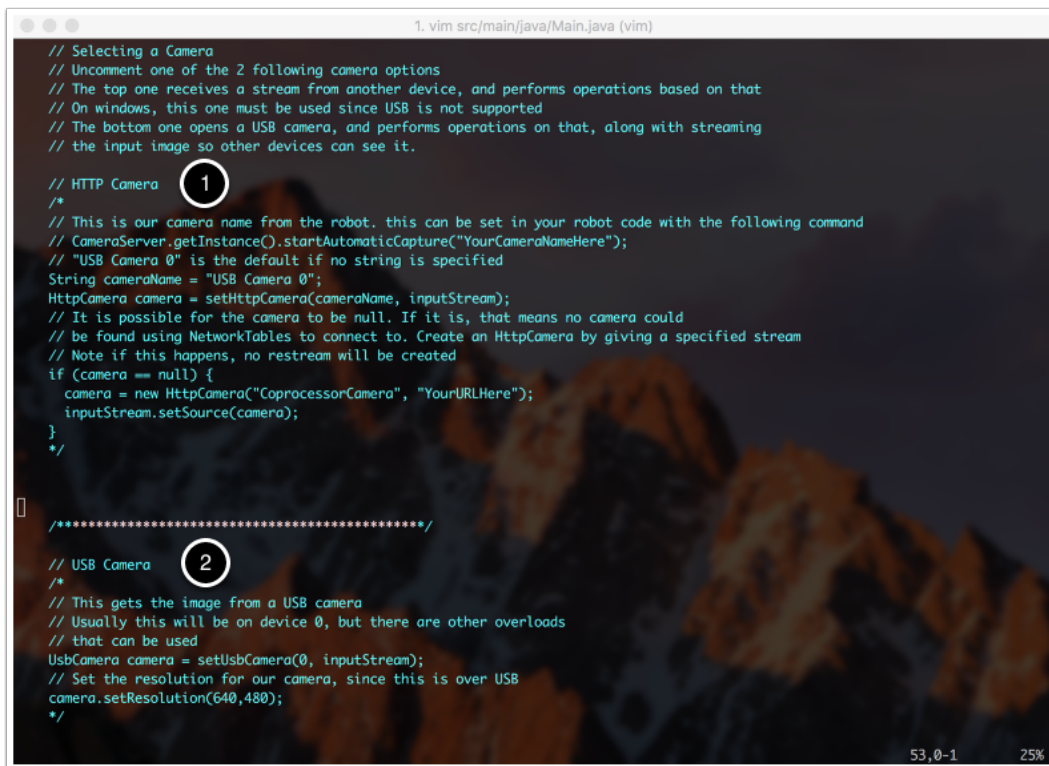
3. Save and exit

Selecting camera type

The sample includes 2 ways to get a camera image:

1. Stream from roboRIO (HTTP Camera)
2. USB Camera plugged into your coprocessor (not supported on Windows)

To select between the types, open the `Main.java` file in `src/main/java`, and scroll down to the line that says "Selecting a Camera". Remove the block comments around the type of camera you are going to use. If using the HTTP Camera, make sure to replace the "YourURLHere" with the URL to the stream of the camera. In the screenshots below, we have selected a USB Camera.



```
1 vim src/main/java/Main.java (vim)

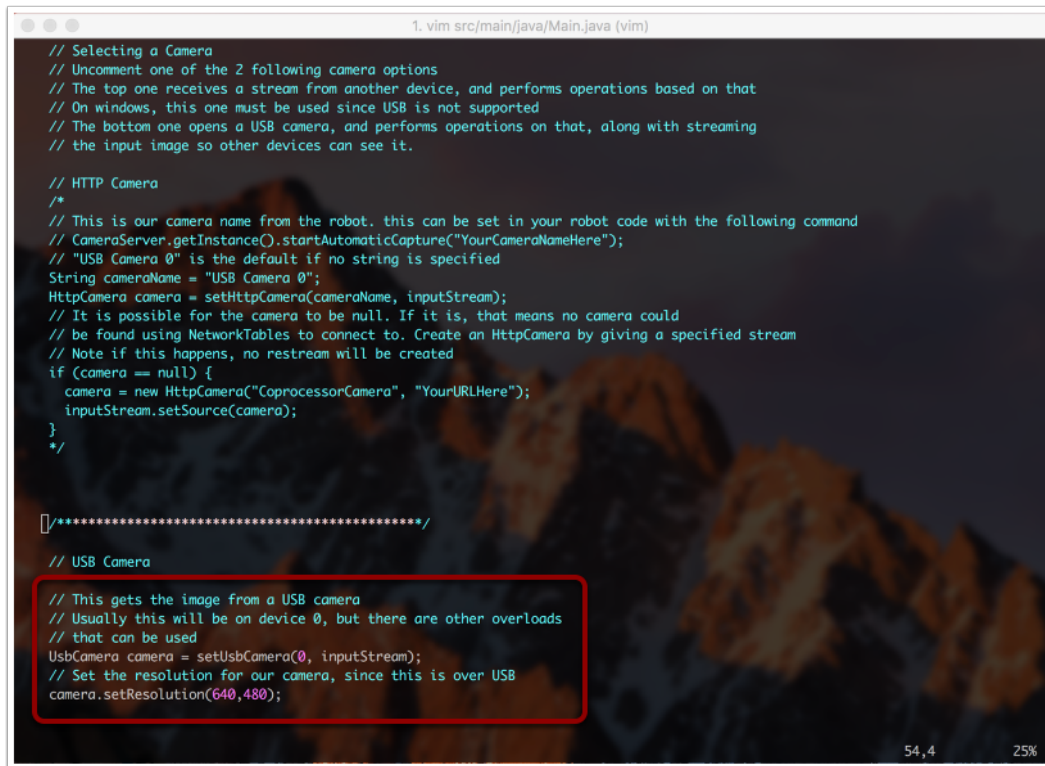
// Selecting a Camera
// Uncomment one of the 2 following camera options
// The top one receives a stream from another device, and performs operations based on that
// On windows, this one must be used since USB is not supported
// The bottom one opens a USB camera, and performs operations on that, along with streaming
// the input image so other devices can see it.

// HTTP Camera 1
/*
// This is our camera name from the robot. this can be set in your robot code with the following command
// CameraServer.getInstance().startAutomaticCapture("YourCameraNameHere");
// "USB Camera 0" is the default if no string is specified
String cameraName = "USB Camera 0";
HttpCamera camera = setHttpCamera(cameraName, inputStream);
// It is possible for the camera to be null. If it is, that means no camera could
// be found using NetworkTables to connect to. Create an HttpCamera by giving a specified stream
// Note if this happens, no restream will be created
if (camera == null) {
    camera = new HttpCamera("CoprocessorCamera", "YourURLHere");
    inputStream.setSource(camera);
}
*/

//*****/

// USB Camera 2
/*
// This gets the image from a USB camera
// Usually this will be on device 0, but there are other overloads
// that can be used
UsbCamera camera = setUsbCamera(0, inputStream);
// Set the resolution for our camera, since this is over USB
camera.setResolution(640,480);
*/
```

Off Board Vision Processing in Java



```
1. vim src/main/java/Main.java (vim)

// Selecting a Camera
// Uncomment one of the 2 following camera options
// The top one receives a stream from another device, and performs operations based on that
// On windows, this one must be used since USB is not supported
// The bottom one opens a USB camera, and performs operations on that, along with streaming
// the input image so other devices can see it.

// HTTP Camera
/*
// This is our camera name from the robot. this can be set in your robot code with the following command
// CameraServer.getInstance().startAutomaticCapture("YourCameraNameHere");
// "USB Camera 0" is the default if no string is specified
String cameraName = "USB Camera 0";
HttpCamera camera = setHttpCamera(cameraName, inputStream);
// It is possible for the camera to be null. If it is, that means no camera could
// be found using NetworkTables to connect to. Create an HttpCamera by giving a specified stream
// Note if this happens, no restream will be created
if (camera == null) {
    camera = new HttpCamera("CoprocessorCamera", "YourURLHere");
    inputStream.setSource(camera);
}
*/

//*****/

// USB Camera
// This gets the image from a USB camera
// Usually this will be on device 0, but there are other overloads
// that can be used
UsbCamera camera = setUsbCamera(0, inputStream);
// Set the resolution for our camera, since this is over USB
camera.setResolution(640,480);
```

Building and Running

This sample program only serves video as an http mjpg stream to a web browser or your SmartDashboard and serves as a template. To add your own vision code, edit the java file and add your code there.

To build the project, run the command `./gradlew build`. This will place the output files into the output directory. From there, you can run either the .bat file on windows or the shell script on unix in order to run your project on the local system. If you are programming on another computer, copy the zip file to the system you plan to run the vision code on, extract the zip file, and run the .bat or shell script to run the program.

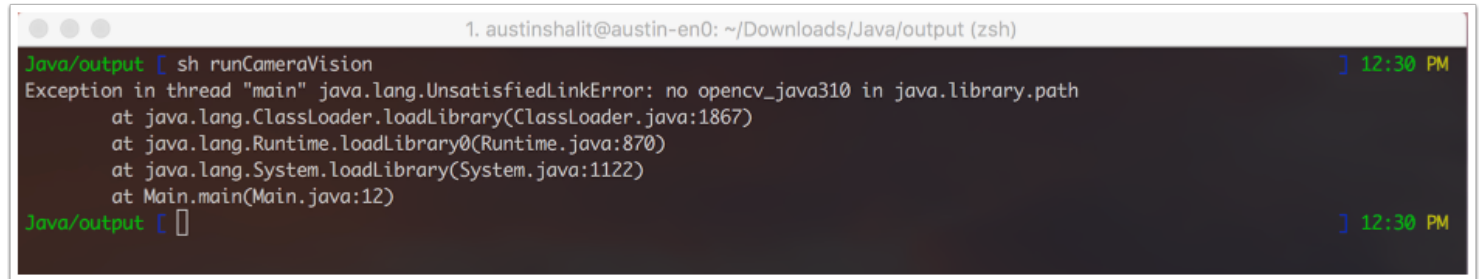
Building Offline

An initial build with an internet connection is needed to download dependencies. After the initial build is complete, builds can be run without an internet connection by adding '--offline' to the build command:

```
./gradlew build --offline
```

Off Board Vision Processing in Java

Troubleshooting



```
1. austinshalit@austin-en0: ~/Downloads/Java/output (zsh)
Java/output [ sh runCameraVision ] 12:30 PM
Exception in thread "main" java.lang.UnsatisfiedLinkError: no opencv_java310 in java.library.path
    at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1867)
    at java.lang.Runtime.loadLibrary0(Runtime.java:870)
    at java.lang.System.loadLibrary(System.java:1122)
    at Main.main(Main.java:12)
Java/output [ ] ] 12:30 PM
```

You are not on the system you specified in the `build.gradle` file. Please modify the `build.gradle` to match your system and run `./gradlew clean`.