

Vision processing on an Arm coprocessor

Coprocessors running vision has been one of the most discussed items in FRC the past few years. On the roboRIO, the new CameraServer helps makes vision and streaming easy. For 2017, beta artifacts for the 3 libraries required (ntcore, cscore and opencv) has been added for running on Arm based coprocessor boards. With these additions, it is easier than ever to get vision working on an FRC robot.

Note: it is our priority to support WPILib Suite on a roboRIO we recognize that running our vision and networking libraries on a coprocessor will be a popular and important choice for teams. To make that easier we provide documentation and prebuilt libraries that should make it easier to write your vision programs.

Picking a board and camera

There are many choices out in the marketplace for arm based coprocessors. However, for FRC you will most likely come across 2. The NVIDIA Jetson has been provided on FIRST Choice for a few years now, and is a great choice for vision processing. The other option is the highly popular Raspberry Pi. Both of these options are very viable options, and either choice is highly recommended. Note that when choosing a Pi, we heavily recommend a Pi 3. A Pi 2 works, but the additional processing power is very useful. The Pi 1 and Pi Zero are not working platforms because of their old architecture. When using a Pi, we recommend adding both a heatsink and a fan to the device, in order to keep your framerates high and your temperatures low.

For cameras, there are a lot more options. We recommend either a Microsoft or a Logitech camera. A Microsoft Lifecam has been offered on FIRST Choice in the past, and is a great option. If you are running on a Raspberry Pi, it is actually possible to get the Raspberry Pi Camera working with CameraServer as well. This has the advantages of being off the USB bus, so you don't have to worry about running out of USB bandwidth. This requires running the following command at boot to enable it.

```
sudo modprobe bcm2835-v4l2
```

You can follow the instructions at <http://www.richardmudhar.com/blog/2015/02/raspberry-pi-camera-and-motion-out-of-the-box-sparrowcam/> to see how to enable that setting on startup.

Building for your device

Building your code for the arm coprocessors is more complicated than building vision for the roboRIO. Java is a little easier, and we have an example Gradle build setup linked below that

Vision processing on an Arm coprocessor

already is setup for building with the WPILib provide artifacts. However, there is still some complication, as different Jars must be used for different devices. However, this can easily be done from a local development machine without any special compilers.

C++ adds much more complication. WPILib only provides a compiler that works on the roboRIO, and that compiler will not work for any known coprocessor board. Because of this, you will either have to find a cross compiler, or compile natively on the device.

Choosing the right package

3 different ARM artifacts are provided:

- arm is built with the FRC compiler, and targets the roboRIO, and other arm soft float targets. This will not work on any of the common coprocessor boards, and should just be used for the roboRIO.
- armhf is built with the Linaro 4.9 GCC cross compiler for gnueabihf. These artifacts target most common coprocessor boards, such as the NVidia Jetson and the BeagleBone Black. The builds will work for some Raspberry Pi builds (Ubuntu Mate has been the tested one), but WILL NOT work properly on raspbian.
- arm-raspbian is built with the raspbian 4.9.3 GCC cross compiler provided by the raspbian team. It will ONLY work on raspbian on the Pi 2 or Pi 3. It will not work on the Pi 1 or Pi Zero, nor will it work on any other armhf board.

In summary, here is a list of some tested boards, and the artifacts to chose from:

- Raspberry Pi 2 or 3 running Raspbian OS: arm-raspbian
- Raspberry Pi 2 or 3 running other OS: armhf (note some of these might require arm-raspbian, it really depends on the OS.)
- BeagleBone Black: armhf
- CHiP: armhf
- NVidia Jetson: armhf
- roboRIO: arm
- Raspberry Pi 1 or Zero: Not working

If you want to be sure which artifact your board works with, download the following zip <http://first.wpi.edu/FRC/roborio/coprocessors/ArtifactDetect.zip>. Inside are an executable for armhf (detect-hf) and an executable for arm-raspbian (detect-raspbian). Run each executable, with a USB camera connected (On Pi, the Pi Camera will work with one slight change, see Picking a board and camera above). If you get a message saying "Pure Virtual Method Called", then that means that is not the proper artifact, and try the other one. If neither one works, you will need to build all artifacts for your device in order to work properly.

Vision processing on an Arm coprocessor

Building Java

All artifacts have Java bindings built for them, and the Java JARs include the native libraries by default, and automatically extract them. To build Java, just point your build system to use the Maven artifacts, using the specific qualifier for your device. The Maven artifact locations can be found on this page. http://wpilib.screenstepslive.com/s/4485/m/wpilib_source/l/480976-maven-artifacts

In addition, a zip of a sample gradle build project can be found at the following link <https://github.com/wpilibsuite/VisionBuildSamples/releases>. Please see the readme file in that zip for more information.

Building C++ Natively

When building for C++, the native artifacts all include the header files, and shared or static libraries for you to link to. Note that if choosing the static libraries, wpiutil is a dependency for both ntcore and cscore that must be linked to as well. The artifacts can be found on the WPILib maven server, with locations on this page. http://wpilib.screenstepslive.com/s/4485/m/wpilib_source/l/480976-maven-artifacts

We do not currently have an example project for doing this, however one might be available in the future.

Finding the cross compiler toolchains for building C++

It is recommended that the output executable be built natively on the device, however sometimes you might need to cross compile. Below are where we have found cross compilers that should work for linking to the libraries. Note only the linux ones have been tested.

- armhf
 - Linux: https://releases.linaro.org/components/toolchain/binaries/4.9-2016.02/arm-linux-gnueabi/gcc-linaro-4.9-2016.02-x86_64_arm-linux-gnueabi.tar.xz
 - Windows: https://releases.linaro.org/components/toolchain/binaries/4.9-2016.02/arm-linux-gnueabi/gcc-linaro-4.9-2016.02-i686-mingw32_arm-linux-gnueabi.tar.xz
- arm-raspbian
 - Linux: <https://github.com/raspberrypi/tools/tree/master/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabi>
 - Windows: <http://sysprogs.com/files/gnutoolchains/raspberry/raspberry-gcc4.9.2-r4.exe>

Vision processing on an Arm coprocessor

Other Languages

C++ and Java are the official languages for these tools, however we know that for vision processing teams like to have options and use what they are more comfortable with. Members of the community have been working with these tools as well, and the following languages and frameworks can be use as well. Note that any team using these alternative platforms will not have official support from WPILib and FIRST, however some of the developers of these projects are WPILib developers as well, and willing to help with these platforms.

- C#/.NET: <https://robotdotnet.github.io/tutorials/coprocessors.html>
- Python: Coming Soon