

## **Project 4 – This is a living, breathing document that I am updating a lot. For now most of the additions will be in the last two pages, the first pages are pretty solid now.**

### **Robot Hardware Setup:**

1. Over all explanation video :  
<https://montana.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=760e6fa7-4192-45a5-854f-afc90159f89a>

This video shows what you should be getting, where your robots are located, where tools, old boards, batteries will go when done.

2. How to change out the computer boards (Nvidia for Raspberry Pi boards on the robot):  
<https://montana.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=1e0dfbe1-ca8f-44a7-b5f3-afc9015518f8>

### **Hardware steps:**

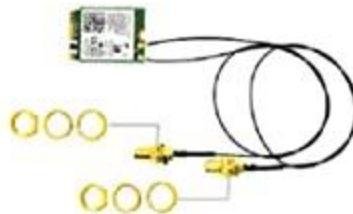
1. Make sure you have new board, new battery, new wifi card, new SD card
2. First carefully, do not break the leads, install the WiFi card in the NVIDIA board. The steps are in the image below:

# Wireless-AC8265 HOW-TO

1. Remove the two screws on Jetson Nano, detach the Jetson Nano



2. Connect the extension cable to Wireless-AC8265 on IPEX connector, tighten the nut and washer to each SMA connector



3. Remove the NIC screw onboard, insert the Wireless-AC8265 into the M.2 socket, screw it tightly



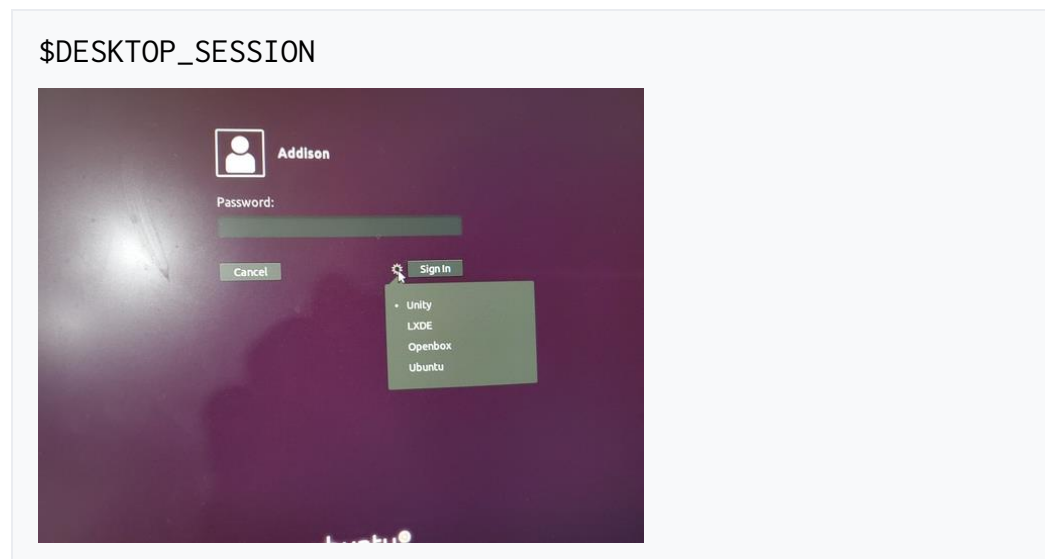
4. Insert the Jetson Nano into the socket and screw it again, connect the antennas



3. Remove the battery
4. Replace Pi board with new NVIDIA board (steps on video above).
5. Replace new battery and install the bracket over the battery.
6. Make sure no part of Nvidia board is touching the aluminum body and if clear turn on the robot with switch on back. You should see lights come on the boards, but the screen won't do much.
7. Install the software to the SD card.

## Software installation:

1. Write the image to the SD card (30 minutes) follow these steps (first four pages/steps):  
<https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit>
2. Boot the SD card.
3. Do the init process:
  - a) Computer's name should be robotXX where xx is your team number
  - b) Use tango as password
  - c) CLICK LOGIN AUTOMATICALLY unless you want to always be putting in your password.
  - d) App Partition use the default
  - e) NVPmodel use default
4. You should download this document on to the desktop of your Nvidia because you are going to be here awhile, and you will be doing a lot of cut and pasting later.
5. Go ahead and switch to do the new desktop.....
  - i. step Eight: Go to a lesser GUI so you free up some Ram



- ii. Change to LXDE desktop to save Ram

6. Open Xterm and do the update/upgrade commands

**sudo apt-get update**

**sudo apt-get upgrade**

You will get a prompt if it is ok, type Y

it will ask you if you want to keep default, just hit enter to keep default

I selected Yes on the docker I.O question, but not sure if that is correct honestly.

**sudo reboot**

7. Set the power to the lower mode:

```
sudo nvpmode1 -m 1
```

0 would be max power.

8. Setup the swap space, without this you won't get everything installed before running out of memory

```
sudo fallocate -l 4G /var/swapfile
```

```
sudo chmod 600 /var/swapfile
```

```
sudo mkswap /var/swapfile
```

```
sudo swapon /var/swapfile
```

```
sudo bash -c 'echo "/var/swapfile swap swap defaults 0 0" >> /etc/fstab'
```

```
sudo reboot
```

```
free -h    ## this will show you the swap should now be 5.9G
```

9. If you want remote desktop you need to install NoMachine on your PC and on the NVIDIA Board

[https://automaticaddison.com/how-to-set-up-the-nvidia-jetson-nano-developer-kit/#Option\\_2\\_Fastest\\_Install\\_NoMachine\\_on\\_Your\\_PC](https://automaticaddison.com/how-to-set-up-the-nvidia-jetson-nano-developer-kit/#Option_2_Fastest_Install_NoMachine_on_Your_PC)

10. Start preparing the software / drivers / dependencies for OPENCV

# Install Dependencies

Type the following command.

```
sudo sh -c "echo '/usr/local/cuda/lib64' >> /etc/ld.so.conf.d/nvidia-  
tegra.conf"
```

Create the links and caching to the shared libraries

```
sudo ldconfig
```

Install the relevant third party libraries. Each command begins with “sudo apt-get install”. I just cut and pasted each one of these sudo’s one at a time. For each one you have to give a Y to continue. I skipped this process the first time, and I got a lot of compile errors when I was doing the OpenCV and RealSense libraries.

```
sudo apt-get install build-essential cmake git unzip pkg-config  
sudo apt-get install libjpeg-dev libpng-dev libtiff-dev  
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev  
sudo apt-get install libgtk2.0-dev libcanberra-gtk*  
sudo apt-get install python3-dev python3-numpy python3-pip  
sudo apt-get install libxvidcore-dev libx264-dev libgtk-3-dev  
sudo apt-get install libtbb2 libtbb-dev libdc1394-22-dev  
sudo apt-get install libv4l-dev v4l-utils  
sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev  
sudo apt-get install libavresample-dev libvorbis-dev libxine2-dev  
sudo apt-get install libfaac-dev libmp3lame-dev libtheora-dev  
sudo apt-get install libopencore-amrnb-dev libopencore-amrwb-dev  
sudo apt-get install libopenblas-dev libatlas-base-dev libblas-dev  
sudo apt-get install liblapack-dev libeigen3-dev gfortran  
sudo apt-get install libhdf5-dev protobuf-compiler  
sudo apt-get install libprotobuf-dev libgoogle-glog-dev libgflags-dev
```

```
sudo sh -c "echo '/usr/local/cuda/lib64' >> /etc/ld.so.conf.d/nvidia-  
tegra.conf"
```

```
sudo ldconfig
```

11. Download and unzip Opencv:

```
cd ~
```

*The next two lines is one command*

```
wget -O opencv.zip  
https://github.com/opencv/opencv/archive/4.5.1.zip
```

*The next two lines is one command*

```
wget -O opencv_contrib.zip  
https://github.com/opencv/opencv\_contrib/archive/4.5.1.zip  
unzip opencv.zip  
unzip opencv_contrib.zip
```

Now do some housekeeping to help make things easier and free up more memory

```
mv opencv-4.5.1 opencv  
mv opencv_contrib-4.5.1 opencv_contrib  
rm opencv.zip  
rm opencv_contrib.zip
```

12. This one takes a while, compile Opencv  
Create a directory.

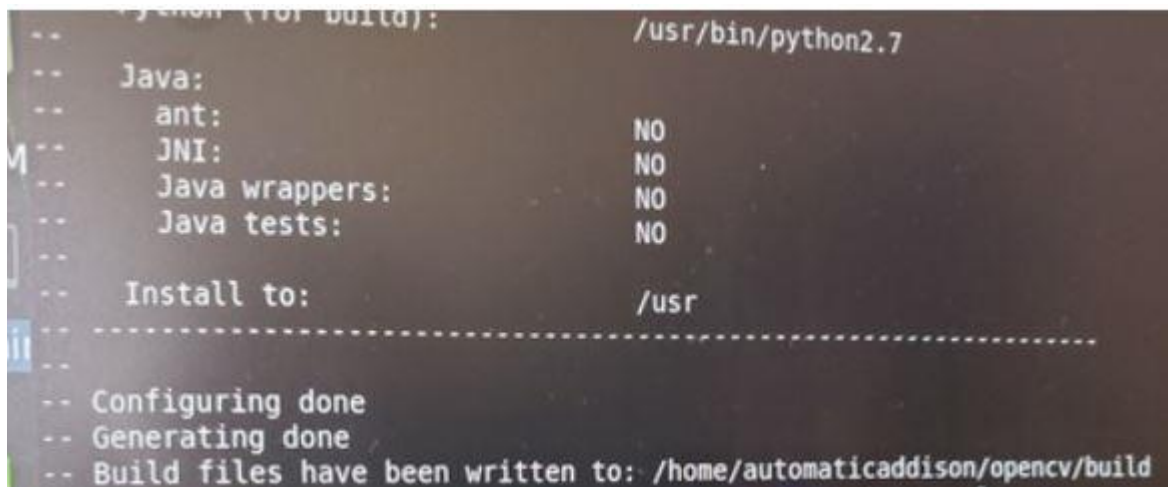
```
cd ~/opencv  
mkdir build  
cd build
```

Set the compilation directives. You can copy and paste this entire block of commands below into your terminal, it is actually only one command.

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/usr \  
-D OPENCV_EXTRA_MODULES_PATH=~/.opencv_contrib/modules \  
-D EIGEN_INCLUDE_PATH=/usr/include/eigen3 \  
-D WITH_OPENCCL=OFF \  
-D WITH_CUDA=ON \  
-D CUDA_ARCH_BIN=5.3 \  
-D CUDA_ARCH_PTX="" \  
-D WITH_CUDNN=ON \  
-D WITH_CUBLAS=ON \  
-D ENABLE_FAST_MATH=ON \  
-D CUDA_FAST_MATH=ON \  
-D OPENCV_DNN_CUDA=ON \  
-D ENABLE_NEON=ON \  
-D WITH_QT=OFF \  
-D WITH_OPENMP=ON \  
-D WITH_OPENGL=ON \  
-D BUILD_TIFF=ON \  
-D WITH_FFMPEG=ON \  
-D WITH_GSTREAMER=ON \  
-D WITH_TBB=ON \  
-D BUILD_TBB=ON \  
-D BUILD_TESTS=OFF \  
-D WITH_EIGEN=ON \  
-D WITH_V4L=ON \  
-D WITH_LIBV4L=ON
```

```
-D OPENCV_ENABLE_NONFREE=ON \  
-D INSTALL_C_EXAMPLES=OFF \  
-D INSTALL_PYTHON_EXAMPLES=OFF \  
-D BUILD_NEW_PYTHON_SUPPORT=ON \  
-D BUILD_opencv_python3=TRUE \  
-D OPENCV_GENERATE_PKGCONFIG=ON \  
-D BUILD_EXAMPLES=OFF ..
```

I got this message when everything was done building.



A terminal window showing the output of the OpenCV configuration process. The text is as follows:

```
-- Python (for build): /usr/bin/python2.7  
--  
-- Java:  
--   ant: NO  
--   JNI: NO  
--   Java wrappers: NO  
--   Java tests: NO  
--  
-- Install to: /usr  
-----  
--  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/automaticaddison/opencv/build
```

Build OpenCV. This command below will take a long time (1-2 hours), so you can go do something else and come back later. Before you start it you should turn off the screensaver so it doesn't pop on and stop the process.

```
make -j4
```

If the building process stops before it reaches 100%, repeat the cmake command I showed earlier, and run the 'make -j4' command again

13. This process took an 1 hour and 15 minutes, installing the RealSense libraries

```
# Installs librealsense and pyrealsense2 on the Jetson NX  
# running Ubuntu 18.04  
# and using Python 3  
# Tested on a Jetson NX running Ubuntu 18.04 and Python 3.6.9
```



```
sudo apt-get update && sudo apt-get -y upgrade
sudo apt-get install -y --no-install-recommends \
    python3 \
    python3-setuptools \
    python3-pip \
    python3-dev

# Install the core packages required to build librealsense libs
sudo apt-get install -y git libssl-dev libusb-1.0-0-dev pkg-
config libgtk-3-dev
# Install Distribution-specific packages for Ubuntu 18
sudo apt-get install -y libglfw3-dev libgl1-mesa-dev libglu1-
mesa-dev

# Install LibRealSense from source
# We need to build from source because
# the PyPi pip packages are not compatible with Arm processors.
# See link
[here](https://github.com/IntelRealSense/librealsense/issues/69
64).

# First clone the repository
git clone https://github.com/IntelRealSense/librealsense.git
cd ./librealsense

# Make sure that your RealSense cameras are disconnected at
this point
# Run the Intel Realsense permissions script
./scripts/setup_udev_rules.sh

# Now the build
mkdir build && cd build
## Install CMake with Python bindings (that's what the -DBUILD
flag is for)
## see link:
https://github.com/IntelRealSense/librealsense/tree/master/wrap
pers/python#building-from-source
cmake ../ -DBUILD_PYTHON_BINDINGS:bool=true
## Recompile and install librealsense binaries
```

```
## This is gonna take a while! The -j4 flag means to use 4
cores in parallel
## but you can remove it and simply run `sudo make` instead,
which will take longer
sudo make uninstall && sudo make clean && sudo make -j4 && sudo
make install

## Export pyrealsense2 to your PYTHONPATH so `import
pyrealsense2` works
export
PYTHONPATH=$PYTHONPATH:/usr/local/lib/python3.6/pyrealsense2
```

Now you should have the cameras working on your Nvidia board. To test this, download an example Python file (e.g. [opencv\\_viewer\\_example.py](#)) and run it with `python3 opencv_viewer_example.py`. You should expect to see a window with both RGBA and depth output.

Now you want to make your export permanent. Move to your home directory and edit `.bashrc`

```
cd ~
```

```
nano .bashrc
```

Paste that line at the bottom of your `.bashrc` file. Save and exit.

```
source .bashrc
```

In order to talk to the servo controller you will need to permanently change the permissions to your TTYAMO (serial port) I think you will also paste the following command into the `.bashrc` (untested)

```
chmod 777 /dev/ttyAM0
```

To turn in on April 3<sup>rd</sup>, this turn in will be a demo of you having everything working. You will demonstrate by having your robot lock on to an object being held by one of your teammates. The object being held will be what your robot is going to track. As the teammate walks toward the robot the robot will back up keeping the same distance between the robot and the teammate. If the teammate walks backwards the robot will move forward again keeping the same distance between the robot and the teammate. All demos will be on Monday, April 3<sup>rd</sup>

from 11am to 2pm. After 2pm you will be able to demo Tuesday, but it will be considered one day late, -10%.