## Table of Contents

[Table of Contents 1](#_Toc27099403)

[Abstract 3](#_Toc27099404)

[Initial Considerations 3](#_Toc27099405)

[Nvidia Jetson 4](#_Toc27099406)

[Nvidia Jetson Comparison Table 5](#_Toc27099407)

[Initial Setup of the TX2 7](#_Toc27099408)

[Jetson Inference 8](#_Toc27099409)

[Jetson Inference Installation Summary 8](#_Toc27099410)

[Jetson Inference Installation Details 8](#_Toc27099411)

[Flip the Camera Image on TX2: 9](#_Toc27099412)

[Install VS Code – OSS version for ARM processors 10](#_Toc27099413)

[Set up LIDAR 11](#_Toc27099414)

[rplidar.RPLidarException 11](#_Toc27099415)

[Install and Test rplidar SDK 12](#_Toc27099416)

[Install and Test ROS (Melodic) and ROS rplidar 13](#_Toc27099417)

[RPLIDAR Won’t stop – STILL NOT WORKING PROPERLY 15](#_Toc27099418)

[RPLIDAR Demo using ROS Viz 15](#_Toc27099419)

[Using ROS 15](#_Toc27099420)

[ROS Notes 15](#_Toc27099421)

[Connecting the Jetson to the Motors, Servos, and Sensors. 17](#_Toc27099422)

[PCA9685 Adafruit PCM 17](#_Toc27099423)

[Our Navigation Solution – In Development 18](#_Toc27099424)

[Miscellaneous Errors and Solutions 20](#_Toc27099425)

[System-wide environment variables 20](#_Toc27099426)

[/etc/environment 20](#_Toc27099427)

[/etc/profile.d/\*.sh 20](#_Toc27099428)

[Setting USB Environment Variables 21](#_Toc27099429)

[One Optimum Way to Install / Uninstall Packages 22](#_Toc27099430)

[Use CheckInstall with auto-apt 22](#_Toc27099431)

[Uninstalling 22](#_Toc27099432)

[Install MatPlotLib and SciKit – INCOMPLETE – NOT WORKING w/Python 3.6.8 23](#_Toc27099433)

[ROS Deep Learning – Dusty nv – Nvidia 23](#_Toc27099434)

[Testing 24](#_Toc27099435)

[imageNet Node 24](#_Toc27099436)

[detectNet Node 24](#_Toc27099437)

[Resizing the OpenGL Window 25](#_Toc27099438)

[Next Steps 26](#_Toc27099439)

[Complete the POC 26](#_Toc27099440)

[Building a Team to Compete 26](#_Toc27099441)

[Iterate 27](#_Toc27099442)

[Proposed Preliminary Budget 27](#_Toc27099443)

[Competition Specific Teams 28](#_Toc27099444)

## Abstract

The following describes the technical considerations and methodologies employed while trying to assemble a small autonomous vehicle capable of Simultaneous Localization and Mapping (SLAM) from LIDAR, while simultaneously, but independently employing a Single Shot Detector Neural Network receiving input from only a single on-board 2D RGB camera for semantic object detection and collision avoidance. Further, our vehicle is intended to take a cubemap set of six photos for import into the ESRI pipeline, process the imagery and SLAM data, then export for near real-time viewing on the Oculus Go virtual reality system.

## Initial Considerations

The use case scenario for our project is an autonomous vehicle for use in emergency situations by C&C first responders. The vehicle should be capable of autonomous navigation to and from a location, using only on-board capabilities after deployment so that it can be used indoors or outside. Upon arrival at its destination, the vehicle should take six photos in a cubemap formation for import into the ESRI pipeline with an eventual goal to export to VR viewing on the Oculus Go. Our goal was to create a proof of concept (poc) and explore the capabilities of some existing development systems or platforms.

We completed an initial two weeks of investigation on a variety of current technologies and development environments:

<https://github.com/NVIDIA-AI-IOT/jetbot>

<https://blogs.nvidia.com/blog/2019/03/26/jetbot-diy-autonomous-robot/>

<https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/>

<https://yadovr.com/>

<https://www.youtube.com/watch?v=wKMWjIKaU68>

<https://github.com/NVIDIA-AI-IOT/jetbot/wiki/Software-Setup>

<https://developer.nvidia.com/embedded/community/jetson-projects#keras_mobiledetectnet>

<https://developer.nvidia.com/embedded-computing>

<https://www.youtube.com/watch?v=ASIVJmlGNc8&feature=youtu.be>

<https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/?=jetsonDevkits>

<https://www.youtube.com/watch?v=FuRhYCFCaCc>

<https://www.youtube.com/watch?v=VhbFbxyOI1k>

<https://build5nines.com/raspberry-pi-4-vs-nvidia-jetson-nano-developer-kit/>

<https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html>

<https://stackoverflow.com/questions/7222382/get-lat-long-given-current-point-distance-and-bearing/7835325>

<https://www.earthdatascience.org/courses/earth-analytics-python/spatial-data-vector-shapefiles/intro-to-coordinate-reference-systems-python/>

<https://www.youtube.com/watch?v=xkut3yRL61U>

At this point in our research, we had not found any single solution which met our needs. No one solution was in our price range, while also allowing on-board AI and a LIDAR component. Over the course of the semester, two companies (Nvidia and Yahboom!) have announced kits that compare to some of our solutions, however these were not available only 3 months ago and neither would be a turn-key solution even today.

Nvidia offers an AI development platform using their Jetson Nano, with a parts list from a variety of manufacturers, however this solution required a 3D printer, and greater electronics involvement than we were initially considering (ie. soldering at a component level), and it did not include LIDAR.

MIT has a comprehensive kit and parts list; however, these units exceeded our budget.

Nvidia’s AI racecar is based off the MIT solution, putting it out of our reach as well.

## Nvidia Jetson

We selected the Nvidia Jetson TX2 and Nano development boards as the principal development platform because of its small form factor combined with its on-board support for multi-threaded machine learning libraries. These libraries are implemented in the form of TensorRT models accessed via an API using either C++ or Python, as well as PyTorch. Some functionality exists only in C++ libraries due to the nature of working with video images in memory. The API is installed as jetson.utils and they can be installed independently or as part of the jetson-inference package described below.

The TX2 processor is a Dual-Core NVIDIA Denver 1.5 64-Bit CPU and Quad-Core ARM® Cortex®-A57 MPCore processor which is the same family as the Raspberry Pi and the Arduino. What distinguishes the Jetson line is the inclusion of a GPU. The TX2 has a 256-core NVIDIA Pascal™ GPU capable of 1.33 TFLOPs.

#### Nvidia Jetson Comparison Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | [**Jetson Nano**](https://developer.nvidia.com/embedded/buy/jetson-nano) | [**Jetson TX2 Series**](https://developer.nvidia.com/embedded/buy/jetson-tx2) | | | [**Jetson Xavier NX**](https://developer.nvidia.com/embedded/jetson-xavier-nx) | [**Jetson AGX Xavier Series**](https://developer.nvidia.com/embedded/buy/jetson-agx-xavier) | |
|  | TX2 4GB | TX2 | TX2i |  | AGX XAVIER 8GB | AGX XAVIER |
| AI Performance | 472 GFLOPs | 1.33 TFLOPs | | 1.26 TFLOPs | 21 TOPs | 20 TOPs | 32 TOPs |
| GPU | 128-core NVIDIA Maxwell™ GPU | 256-core NVIDIA Pascal™ GPU | | | 384-core NVIDIA Volta™ GPU with 48 Tensor Cores | 384-core NVIDIA Volta™ GPU with 48 Tensor Cores | 512-core NVIDIA Volta™ GPU with 64 Tensor Cores |
| CPU | Quad-Core ARM® Cortex®-A57 MPCore | Dual-Core NVIDIA Denver 1.5 64-Bit CPU and Quad-Core ARM® Cortex®-A57 MPCore processor | | | 6-core NVIDIA Carmel ARM®v8.2 64-bit CPU  6MB L2 + 4MB L3 | 6-core NVIDIA Carmel Arm®v8.2 64-bit CPU  6MB L2 + 4MB L3 | 8-core NVIDIA Carmel Arm®v8.2 64-bit CPU  8MB L2 + 4MB L3 |
| Memory | 4 GB 64-bit LPDDR4  25.6GB/s | 4 GB 128-bit LPDDR4  51.2GB/s | 8 GB 128-bit LPDDR4  59.7GB/s | 8 GB 128-bit LPDDR4 (ECC Support)  51.2GB/s | 8 GB 128-bit LPDDR4x  51.2GB/s | 8 GB 256-bit LPDDR4x  85.3GB/s | 16 GB 256-bit LPDDR4x  136.5GB/s |
| Storage | 16 GB eMMC 5.1 \* | 16 GB eMMC 5.1 | 32 GB eMMC 5.1 | 32 GB eMMC 5.1 | 16 GB eMMC 5.1 | 32 GB eMMC 5.1 | |
| Power | 5W / 10W | 7.5W / 15W | | 10W / 20W | 10W / 15W | 10W / 20W | 10W / 15W / 30W |
| PCIE | 1 x4  (PCIe Gen2) | 1 x1 + 1 x4 OR 1 x1 + 1 x1 + 1 x2  (PCIe Gen2) | | | 1 x1 + 1 x4  (PCIe Gen3, Root Port &Endpoint) | 1 x8 + 1 x4 + 1 x2 + 2 x1  (PCIe Gen3) | 1 x8 + 1 x4 + 1 x2 + 2 x1  (PCIe Gen4, Root Port &Endpoint) |
| CSI Camera | Up to 4 cameras  12 lanes MIPI CSI-2  D-PHY 1.1 (up to 18 Gbps) | Up to 6 cameras (12 via virtual channels)  12 lanes MIPI CSI-2  D-PHY 1.2 (up to 30 Gbps)  C-PHY 1.1 (up to 41Gbps) | | | Up to 6 cameras (36 via virtual channels)  12 lanes MIPI CSI-2  D-PHY 1.2 (up to 30 Gbps) | Up to 6 cameras (36 via virtual channels)  16 lanes MIPI CSI-2 | 8 lanes SLVS-EC  D-PHY 1.2 (up to 40 Gbps)  C-PHY 1.1 (up to 59 Gbps) | |
| Video Encode | 250MP/sec  1x 4K @ 30 (HEVC)  2x 1080p @ 60 (HEVC) | 500MP/sec  1x 4K @ 60 (HEVC)  3x 4K @ 30 (HEVC)  4x 1080p @ 60 (HEVC) | | | 2x464MP/sec  2x 4K @ 30 (HEVC)  6x 1080p @ 60 (HEVC) | 2x464MP/sec  2x 4K @ 30 (HEVC)  6x 1080p @ 60 (HEVC)  14x 1080p @ 30 (HEVC) | 2x1000MP/sec  4x 4K @ 60 (HEVC)  16x 1080p @ 60 (HEVC)  32x 1080p @ 30 (HEVC) |
| Video Decode | 500 MP/sec  1x 4K @ 60 (HEVC)  4x 1080p @ 60 (HEVC) | 1000 MP/sec  2x 4K @ 60 (HEVC)  7x 1080p @ 60 (HEVC)  20x 1080p @ 30 (HEVC) | | | 2x690MP/sec  2x 4K @ 60 (HEVC)  12x 1080p @ 60 (HEVC)  32x 1080p @ 30 (HEVC) | 2x690MP/sec  2x 4K @ 60 (HEVC)  12x 1080p @ 60 (HEVC)  32x 1080p @ 30 (HEVC) | 2x1500MP/sec  2x 8K @ 30 (HEVC)  6x 14k @ 60 (HEVC)  26x 1080p @ 60 (HEVC)  72x 1080p @ 30 (HEVC) |
| Display | 2 multi-mode DP 1.2/eDP 1.4/HDMI 2.0  1 x2 DSI (1.5Gbps/lane) | 2 multi-mode DP 1.2/eDP 1.4/HDMI 2.0  2 x4 DSI (1.5Gbps/lane) | | | 2 multi-mode DP 1.4/eDP 1.4/HDMI 2.0  No DSI support | 3 multi-mode DP 1.2/eDP 1.4/HDMI 2.0  No DSI support | |
| DL Accelerator | — | — | | | 2x NVDLA Engines | 2x NVDLA Engines | |
| Vision Accelerator | — | — | | | — | 7-Way VLIW Vision Processor | |
| Networking | 10/100/1000 BASE-T Ethernet | 10/100/1000 BASE-T Ethernet | 10/100/1000 BASE-T Ethernet, WLAN | 10/100/1000 BASE-T Ethernet | 10/100/1000 BASE-T Ethernet | 10/100/1000 BASE-T Ethernet | |
| Mechanical | 69.6 mm x 45 mm  260-pin SO-DIMM connector | 87 mm x 50 mm  400-pin connector | | | 69.6 mm x 45 mm  260-pin SO-DIMM connector | 100 mm x87 mm  699-pin connector | |

\* The Jetson Nano module included as part of the Jetson Nano developer kit has a slot for using microSD card instead of eMMC as system storage device.

Please refer to NVIDIA software documentation online [here](https://docs.nvidia.com/jetson/index.html) for information about currently supported features.

As illustrated in the diagram below, the TX2 serves as the central platform upon which all else is installed or connected. Integrating the disparate components required a detailed understanding of both the hardware and software components of establishing communication to and from the TX2 to all the other components of the bot via multiple busses and protocols.

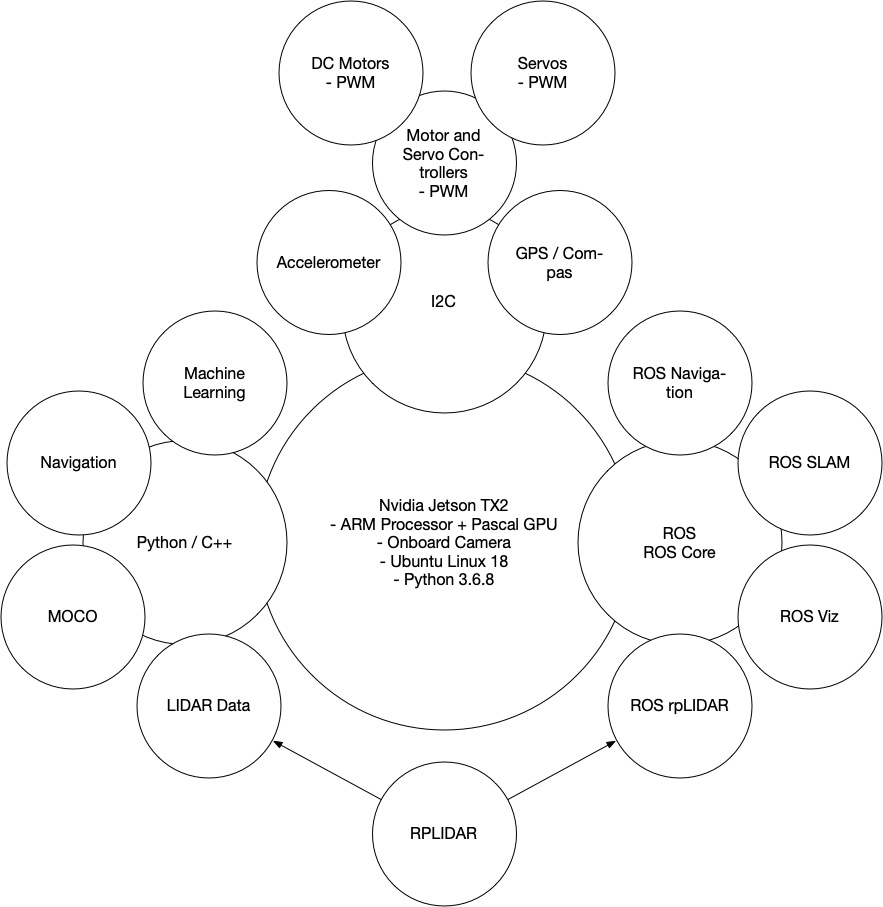


Figure 1 - GIS Bot System Overview

## Initial Setup of the TX2

1. Start Up the TX2 in Forced Reset Mode
2. Plug in to an Ubuntu Linux 16 or 18 Host Computer
3. Remember the host will probably not be ARM proc, but the TX2 is ARM aarch64 for purposes of building software packages
4. Installed Nvidia JetPack SDK on the host computer – small Lenovo w/Ubuntu 18
5. Needs a network connection
6. Check for all updates on the Host, then check again
7. Connect TX2 to host via supplied USB cable
8. Run the JetPack Installer on the Host to install on the TX2
9. Will take significant time to install JetPack on Host, download the Software and Flash the TX2 for the first time.
10. JetPack caches the downloads locally on the host for next time
11. The JetPack installation process will pause part way through
12. Requires the TX2 to complete its Initial Setup on the TX2
13. Complete setup on the TX2 only through the initial setup steps (including a static IP if you have one – but static is not required at this point)
14. Return to the Host Computer and complete the installation of JetPack software.
15. You can use the default IP address to complete the setup, but you will want to make sure you note the Username and Password of the TX2
16. Complete install will require a few hours and can run unattended; however, it will require you to click Finish and Exit to complete the install. If you click skip or go back at any point it will need to repeat the previous steps (potentially hours).
17. If the TX2 does not have an internet connection you will get “Connection Failed” errors, these can be ignored for the time being.

## Jetson Inference

The Python functionality of this project is implemented through Python extension modules that provide bindings to the native C++ code using the Python C API. While configuring the project, the repo searches for versions of Python that have development packages installed on the system and will then build the bindings for each version of Python that's present (e.g. Python 2.7, 3.6, and 3.7). It will also build NumPy bindings for versions of NumPy that are installed.

- jetson-inference documentation

### Jetson Inference Installation Summary

Download, Build and Install the jetson-inference project from Git

<https://github.com/dusty-nv/jetson-inference.git>

Follow the instructions in the Hello World tutorial to setup and test AI – cmake will be required many times – make sure these steps complete without errors

$ sudo apt-get update

$ sudo apt-get install git cmake libpython3-dev python3-numpy

$ git clone --recursive https://github.com/dusty-nv/jetson-inference

$ cd jetson-inference

$ mkdir build

$ cd build

$ cmake ../

$ make

$ sudo make install

$ sudo ldconfig

### Jetson Inference Installation Details

By default, Ubuntu comes with the libpython-dev and python-numpy packages pre-installed (which are for Python 2.7). Although the Python 3.6 interpreter is pre-installed by Ubuntu, the Python 3.6 development packages (libpython3-dev) and python3-numpy are not. These development packages are required for the bindings to build using the Python C API.

So, if you want the project to create bindings for Python 3.6, install these packages before proceeding:

$ sudo apt-get install libpython3-dev python3-numpy

Installing these additional packages will enable the repo to build the extension bindings for Python 3.6, in addition to Python 2.7 (which is already pre-installed). Then after the build process, the [jetson.inference](https://rawgit.com/dusty-nv/jetson-inference/python/docs/html/python/jetson.inference.html) and [jetson.utils](https://rawgit.com/dusty-nv/jetson-inference/python/docs/html/python/jetson.utils.html) packages will be available to use within your Python environments.

Install the selected Models

Note the amount of space required – TX2 has 8GB available – installing all the models will take a little over 2GB

Compile the Models

$ cd jetson-inference/build # omit if working directory is already build/ from above

$ make

$ sudo make install

$ sudo ldconfig

In the build tree, you can find the binaries residing in build/aarch64/bin/, headers in build/aarch64/include/, and libraries in build/aarch64/lib/. These also get installed under /usr/local/ during the sudo make install step.

The Python bindings for the [jetson.inference](https://rawgit.com/dusty-nv/jetson-inference/python/docs/html/python/jetson.inference.html) and [jetson.utils](https://rawgit.com/dusty-nv/jetson-inference/python/docs/html/python/jetson.utils.html) modules also get installed during the sudo make install step under /usr/lib/python\*/dist-packages/. If you update the code, remember to run it again.

You will need to be in the correct directory to run the examples

$ cd jetson-inference/build/aarch64/bin

1. Run the Imagenet Object ID test

$ ./imagenet-console.py --network=googlenet images/orange\_0.jpg output\_0.jpg # --network flag is optional

### Flip the Camera Image on TX2:

One of the first challenges we faced was an obvious one. The image was upside down! It required some basic research in the Nvidia developer’s forums to find that there is an error in the base code for the jetson.utils when working with the on-board camera for the TX2. It flips the image upside down on the TX2. Not only did this make the image unpleasant for viewing, but we also found that it confused the AI. For example, it would identify a person walking with their arms outstretched, as a bird. The fix for this had to be implemented in the C++ code and the package recompiled.

Set flipMethod to 0 in the file utils/camera/gstCamera.cpp: line 416

#if NV\_TENSORRT\_MAJOR > 1 && NV\_TENSORRT\_MAJOR < 5 // if JetPack 3.1-3.3 (different flip-method)

const int flipMethod = 0; // Xavier (w/TRT5) camera is mounted inverted

#else

const int flipMethod = 0;

#endif

Rebuild the package:

$ cd jetson-inference/build

$ make clean

$ cmake ../

$ make

$ sudo make install

## Install VS Code – OSS version for ARM processors

None of the previous steps includes a full featured IDE so we found the following solution to install Microsoft’s lightweight Code IDE. Code was chosen because it includes built-in support for virtual environments and Git, as well as full-featured linting and code highlighting for all the programming languages in our project. It also comes bundled with the Anaconda distribution of Python.

Install CURL

$ sudo apt-get install curl

$ curl -s https://packagecloud.io/install/repositories/swift-arm/vscode/script.deb.sh | sudo bash

$ sudo apt-get install code-oss

Code OSS will be installed in /usr/share/applications/. Once installed you can pin it to the dock.

Install the Python Extension in VS Code

Install pylint

$ sudo pip3 install pylint

Or

$ pip3 install pylint –-user

Or if using venv:

$ source venv/bin/activate

$ pip3 install pylint

## Set up LIDAR

Plug the RPLIDAR into the TX2 USB via the supplied UART to micro USB converter.

Install the rplidar libraries

sudo pip3 install rplidar

We used the following Python test code from RPLIDAR.

from rplidar import RPLidar

lidar = RPLidar('/dev/ttyUSB0')

info = lidar.get\_info()

print(info)

health = lidar.get\_health()

print(health)

for i, scan in enumerate(lidar.iter\_scans()):

print('%d: Got %d measurments' % (i, len(scan)))

if i > 10:

break

lidar.stop()

lidar.stop\_motor()

lidar.disconnect()

### 

### rplidar.RPLidarException

rplidar.RPLidarException: Failed to connect to the sensor due to: [Errno 13] could not open port /dev/ttyUSB0: [Errno 13] Permission denied: '/dev/ttyUSB0'

The solution was to add our autoboot user to the tty and dialout Group then restart

$ sudo gpasswd -a autobot tty

$ sudo usermod -a -G dialout autobot

Restart the bot

### 

### Install and Test rplidar SDK

$ git clone <https://github.com/Slamtec/rplidar_sdk.git>

$ cd rplidar\_sdk/sdk

$ make

$ cd output/Linux/Release

$ ./simple\_grabber /dev/ttyUSB0

## Install and Test ROS (Melodic) and ROS rplidar

cd to Home directory

$ cd ~

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can [follow the Ubuntu guide](https://help.ubuntu.com/community/Repositories/Ubuntu) for instructions on doing this.

<https://help.ubuntu.com/community/Repositories/Ubuntu>

Setup your computer to accept software from packages.ros.org.

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

Set up your keys

$ curl -sSL 'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0xC1CF6E31E6BADE8868B172B4F42ED6FBAB17C654' | sudo apt-key add -

Update package list

$ sudo apt update

Install ROS Desktop (Not “Desktop-Full”)

$ sudo apt install ros-melodic-desktop

Initialize ROS Dep

$ sudo rosdep init

$ rosdep update

Environment Setup - It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc

$ source ~/.bashrc

Dependencies for building packages

$ sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential

Check your installation

$ printenv | grep ROS

Source the Environment

$ source /opt/ros/melodic/setup.bash

Install catkin\_pkg

$ sudo pip3 install catkin\_pkg

Create a catkin workspace

$ mkdir -p ~/bot\_catkin\_ws/src

$ cd bot\_catkin\_ws

You may need to install cmake

download the tar ball

extract

$ ./bootstrap && make && sudo make install

The first time in any new directory you must use the following, after that you can use just cmake

$ catkin\_make -DPYTHON\_EXECUTABLE=/usr/bin/python3 -DPYTHON\_INCLUDE\_DIR=/usr/include/python3.6m -DPYTHON\_LIBRARY=/usr/lib/libpython3.6m.so

Source the new catkin overlay

$ source devel/setup.bash

To make sure your workspace is properly overlayed by the setup script, make sure ROS\_PACKAGE\_PATH environment variable includes the directory you're in.

$ echo $ROS\_PACKAGE\_PATH

$ cd /src

clone the rplidar\_ros

$ git clone <https://github.com/Slamtec/rplidar_ros.git>

$ cd ..

Confirm the overlay

$ source devel/setup.bash

Run catkin\_make

$ catkin\_make

$ roslaunch rplidar\_ros view\_rplidar.launch

Might need to change the USB port in the launch file

**ProTip** for figuring out which devices are what on USB:

Store a list of devices before plugging in Device

$ ls /dev/ > dev\_list\_1.txt

Then run this after you plug it in

$ ls /dev/ | diff --suppress-common-lines -y – dev\_list\_1.txt

### RPLIDAR Won’t stop – STILL NOT WORKING PROPERLY

Use a powered USB hub

Add 2 lines of command

$ source /home/{user}/catkin\_ws/devel/setup.bash

$ source /opt/ros/melodic/setup.bash

### RPLIDAR Demo using ROS Viz

$ cd ~/rplidar

$ source devel/setup.bash

$ roslaunch rplidar\_ros view\_rplidar.launch

## Using ROS

ROS is the Robot Operating System. It is a development framework for robotic development. ROS works off of a central core known as ROS Core. After starting the Core process, a variety of other nodes can be connected to the Core. This means that any given node can use any other node’s output for input as long as it meets the Core’s requirements for communicating.

We were able to access the point data from the rpLIDAR both via Python and via a ROS node and ROS Viz. The data from the LIDAR arrives in the form of an angle (theta) and a distance (range).

It was our intent to use ROS to provide a solution for navigation and SLAM. Unfortunately, while implementing the solution, it was discovered that the ROS Navigation packages require odometry data and/or an Intel or AMD 64-bit processor we did not have. Our current solution was stuck in a nether world of software / hardware incompatibility.

### ROS Notes

If you are ever having problems finding or using your ROS packages, make sure that you have your environment properly setup. A good way to check is to ensure that [environment variables](http://wiki.ros.org/ROS/EnvironmentVariables) like [ROS\_ROOT](http://wiki.ros.org/ROS/EnvironmentVariables#ROS_ROOT) and [ROS\_PACKAGE\_PATH](http://wiki.ros.org/ROS/EnvironmentVariables#ROS_PACKAGE_PATH) are set:

printenv | grep ROS

Need to run this in every new shell unless added to .bashrc

source /opt/ros/melodic/setup.bash

To make sure your workspace is properly overlayed by the setup script, make sure ROS\_PACKAGE\_PATH environment variable includes the directory you're in.

echo $ROS\_PACKAGE\_PATH

Make sure to allow access to the USB

sudo chmod 666 /dev/ttyUSB0

WARNING: The script jupyter-console is installed in '/home/autobot/.local/bin' which is not on PATH.

Consider adding this directory to PATH or, if you prefer to suppress this warning, use –no-warn-script-location.

## Connecting the Jetson to the Motors, Servos, and Sensors.

Both the Nano and the TX2 list a 40-pin Raspberry Pi style connector on the development board. As we would discover, there are actually variations between the Pi version and the Nvidia versions, and even some differences in pins between the Nvidia models. This meant significant customization was required to use the hardware diagrams and software programs written by any other projects. It also meant that the motor control unit that was intended to control the bot was not usable in its current configuration via a simple 40 pin connector cable. It would require a re-mapping of pins to make this combination work with our TX2 or Nano developer boards.

We were able to make the bot move easily during week 3 when we set up the bot with a Raspberry Pi controller instead of the Nvidia boards. However, due to the difference in pin configurations, simply swapping out the Raspberry Pi board for the Nano or TX2 would not work. And the Pi does not have the processing power to handle the on-board AI. So, we decided we would use the Nvidia Jetson Racecar solution for motor and servo control.

The TX2 and NANO do not have the same support for on-board Pulse Wave Modulation (PWM) via the GPIO pins on the 40pin connector that the Pi has, and they have no software PWM via GPIO whereas the Pi has PWM control for every pin. There is a library available to mimic the software solution, and we would use that in conjunction with a separate PWM control module via I2C. The Nvidia development boards use the I2C standard for communication. This allows for more powerful types of serial communication than PWM, however it makes it a little trickier to use the smaller “hobby” servos and motors that our project was employing at this stage.

This communication between the logic on the processors and controlling the vehicle proved to be a significant pain point during this phase of the project. The lack of any definitive source for an appropriate component list combined with a steep learning curve with regards to the communications protocols supported by the individual components and boards proved to require significantly more learning and testing than originally anticipated. After successfully moving the bot in week three, the difficulties faced later in the process were all that more trying.

### PCA9685 Adafruit PCM

<https://github.com/jetsonhacks/JHPWMDriver>

<https://www.jetsonhacks.com/2016/01/11/imu-and-pwm-driver-over-i2c-for-nvidia-jetson-tx1/>

<https://github.com/adafruit/Adafruit_CircuitPython_ServoKit>

sudo pip3 install adafruit-circuitpython-servokit

## Our Navigation Solution – In Development

We are currently pursuing a custom solution, utilizing Python as the core programming language.

We have written custom Python code to convert between a variety of coordinate systems to enable the bot to be geolocated when possible, but capable of operation indoors or without any GPS signal when necessary.

The primary location data is LAT/LON to provide for global functionality. We wrote a python module to convert from LAT/LON to a cartesian space of X/Y/Z coordinates. This model uses the radius of the Earth and a Spheroid shape, rather than a more accurate Ellipsoid model, because we were implementing it for poc purposes only, and we wrote the code. However, the transforms doing the trigonometry can and should be easily swapped out for a more optimized library before moving to the next phase of competition or development.

R = 6371007.1810 # Radius of Earth in Meters per IUGG Spheroid - not WGS84 Ellipsoid

Further, our code is also capable of converting the radial and distance measurements from the LIDAR to and from the other two coordinate systems. In summary, we have a working package capable of handling all the required conversion to interchange any of three coordinate systems: LAT/LON , Cartesian X/Y/Z , Radial Theta/Distance.

The basic starting point of the navigation algorithm is to convert any starting GPS coordinates into X/Y/Z space then minimize the difference in the current location with respect to the destination, until the bot reaches its goal.

The navigation code seeks to emphasize the data coming from the Jetson TX2’s onboard 2D RGB camera at 720p resolution. That information is fed into a Single Shot Detector neural network, based off the jetson.inference API. We have installed multiple TensorRT models on the Jetson capable of segmentation, detection, and classification. The models can be further tuned or substituted with custom models.

Figure 2 shows the flow from the onboard camera through the VRAM where it is accessed directly via CUDA as well as a NumPy array for color space transformations. This is the recommended methodology by Nvidia at this time for working with images in Array space. For us it also provides a simple solution for ingesting images intended for use in a machine learning environment.

From VRAM, the image can also be viewed directly using OpenGL. This allows for remote human operation of the bot using a standard joystick for control, as well as assisting in understanding what the camera is showing the AI.

A close up of a map

Description automatically generated

Figure - Image Processing to Navigation Solutions

## Miscellaneous Errors and Solutions

Throughout the project we have been challenged by a range of errors. These errors can often be deceptive, and often required significant research to solve. In fact, there were multiple occasions where the errors could only be remedied by changing source code or required a developer to point us in the right direction. We have listed some of the more universal errors, and some techniques we used to resolve them. Some of these may be well known to those more familiar with the Linux operating system, but they are included here for completeness.

### System-wide environment variables

One challenge was the use of USB. USB is a hot swappable standard, and assigns IDs based on the order that a given device is plugged in or recognized at startup. This is very friendly from a user standpoint, allowing one to plug and unplug any number of devices on the same port. However, when using the USB IDs dynamically in our programs, we found it necessary to assign variables that could be used system-wide so that it did not matter if a given component was unplugged or when it was detected in the startup order.

A suitable file for environment variable settings that affect the system as a whole (rather than just a particular user) is /etc/environment. An alternative is to create a file for the purpose in the /etc/profile.d directory.

#### /etc/environment

This file is specifically meant for system-wide environment variable settings. It is not a script file, but rather consists of assignment expressions, one per line.

FOO=bar

Note: Variable expansion does not work in /etc/environment.

#### /etc/profile.d/\*.sh

Files with the .sh extension in the /etc/profile.d directory get executed whenever a bash login shell is entered (e.g. when logging in from the console or over ssh), as well as by the DisplayManager when the desktop session loads.

You can for instance create the file /etc/profile.d/myenvvars.sh and set variables like this:

export JAVA\_HOME=/usr/lib/jvm/jdk1.7.0

export PATH=$PATH:$JAVA\_HOME/bin

## 

### Setting USB Environment Variables

1. Find out what's on ttyUSB:

$ dmesg | grep ttyUSB

2. List all attributes of the device. Use your device ID instead of x.

$ udevadm info --name=/dev/ttyUSBx –attribute-walk

Pick out a unique identifier set, eg idVendor + idProduct. You may also need SerialNumber if you have more than one device with the same idVendor and idProduct. SerialNumbers ought to be unique for each device.

Info for our UART Adapter

ID\_VENDOR=Silicon\_Labs

ID\_VENDOR\_ID=10c4

ID\_MODEL\_ID=ea60

symlink for our bot : uart\_bridge

3. Create a file /etc/udev/rules.d/99-your\_file.rules with something like this line in it:

SUBSYSTEM=="tty", ATTRS{idVendor}=="1234", ATTRS{idProduct}=="5678", SYMLINK+="your\_device\_name"

4. Load the new rule:

$ sudo udevadm trigger

5. Verify what happened:

$ ls -l /dev/your\_device\_name

will show what ttyUSB number the symlink went to. If it's /dev/ttyUSB1, then verify who owns that and to which group it belongs:

$ ls -l /dev/ttyUSB1

Then a final double check:

$ udevadm test -a -p $(udevadm info -q path -n /dev/your\_device\_name)

### One Optimum Way to Install / Uninstall Packages

One significant challenge throughout the course of the project was the need to custom compile most of the software packages, and or code snippets. We were constantly presented with multiple authoritative solutions for a given installation. There are many “bad habits” that can easily become the norm when attempting to make all the pieces work together.

Given the unstable nature of a development environment, it is often necessary to uninstall after an incomplete installation. The variety of dependencies, and the need to compile much of the software specifically to this machine means that there will often be dependencies that need to be installed before a given package can be compiled. If a package fails to compile, it can leave files scattered around the system, making removal of a partial installation very difficult. The following method attempts to mitigate this scattering of files by tracking the installation, making the reversal of the process much easier.

We use checkinstall instead of just install to create the list of files, and we use run ./configure instead of just ./configure to get a dialog box about any dependencies that are available for install.

### Use CheckInstall with auto-apt

You can use auto-apt when you want to build a simple package from source with checkinstall. You need to have [auto-apt](https://help.ubuntu.com/community/AutoApt) installed!

Instead of

$ ./configure

you use:

$ auto-apt run ./configure

If the dependencies are available, a dialog box opens and asks you to install them.

The rest remains the same

$ make

$ sudo checkinstall

### Uninstalling

The installed package can then also easily be removed via [Synaptic](https://help.ubuntu.com/community/SynapticHowto#head-9a2bcc5a697205e980d6b8b3cac02f799e1bd5f0) or via the terminal:

$ sudo dpkg -r packagename

Example:

$ sudo dpkg -r pidgin

<https://help.ubuntu.com/community/CheckInstall>

## Install MatPlotLib and SciKit – INCOMPLETE – NOT WORKING w/Python 3.6.8

sudo apt-get update

pip3 install --upgrade setuptools  
sudo pip3 install -U setuptools  
sudo apt-get install libpcap-dev libpq-dev  
sudo pip3 install cython

sudo apt-get update

sudo apt-get install -y build-essential libatlas-base-dev

sudo apt-get install gfortran

sudo pip3 install -U scikit-learn

sudo pip3 install git+<https://github.com/scikit-learn/scikit-learn.git>

python -m pip install --user numpy scipy matplotlib

## ROS Deep Learning – Dusty nv – Nvidia

<https://github.com/dusty-nv/ros_deep_learning>

$ sudo apt-get install ros-melodic-image-transport

$ sudo apt-get install ros-melodic-image-publisher

$ sudo apt-get install ros-melodic-vision-msgs

Then, create a Catkin workspace (~/catkin\_ws) using these steps:  
<http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment#Create_a_ROS_Workspace>

$ cd ~/bot\_catkin\_ws/src

$ git clone https://github.com/dusty-nv/ros\_deep\_learning

$ cd ../

$ catkin\_make

**Note**: if you do this after the rplidar installation, it will see both cmake files and do them both.

### Testing

Before proceeding, make sure that roscore is running first:

$ roscore

### imageNet Node

First, to stream some image data for the inferencing node to process, open another terminal and start an [image\_publisher](http://wiki.ros.org/image_publisher), which loads a specified image from disk. We tell it to load one of the test images that come with jetson-inference, but you can substitute your own images here as well:

$ rosrun image\_publisher image\_publisher \_\_name:=image\_publisher ~/jetson-inference/data/images/orange\_0.jpg

Next, open a new terminal, overlay your Catkin workspace, and start the [imagenet](https://github.com/dusty-nv/ros_deep_learning/blob/master/src/node_imagenet.cpp) node:

$ source ~/catkin\_ws/devel/setup.bash

$ rosrun ros\_deep\_learning imagenet /imagenet/image\_in:=/image\_publisher/image\_raw \_model\_name:=googlenet

Here, we remap imagenet's image\_in input topic to the output of the image\_publisher, and tell it to load the GoogleNet model using the node's model\_name parameter. See [this table](https://github.com/dusty-nv/jetson-inference/blob/master/docs/imagenet-console-2.md#downloading-other-classification-models) for other classification models that you can download and substitute for model\_name.

In another terminal, you should be able to verify the [vision\_msgs/Classification2D](http://docs.ros.org/melodic/api/vision_msgs/html/msg/Classification2D.html) message output of the node, which is published to the imagenet/classification topic:

$ rostopic echo /imagenet/classification

### detectNet Node

Kill the other nodes you launched above, and start publishing a new image with people in it for the [detectnet](https://github.com/dusty-nv/ros_deep_learning/blob/master/src/node_detectnet.cpp) node to process:

$ rosrun image\_publisher image\_publisher \_\_name:=image\_publisher ~/jetson-inference/data/images/peds-004.jpg

$ rosrun ros\_deep\_learning detectnet /detectnet/image\_in:=/image\_publisher/image\_raw \_model\_name:=pednet

See [this table](https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2.md#pre-trained-detection-models-available) for the built-in detection models available. Here's an example of launching with the model that detects dogs:

$ rosrun image\_publisher image\_publisher \_\_name:=image\_publisher ~/jetson-inference/data/images/dog\_0.jpg

$ rosrun ros\_deep\_learning detectnet /detectnet/image\_in:=/image\_publisher/image\_raw \_model\_name:=coco-dog

To inspect the [vision\_msgs/Detection2DArray](http://docs.ros.org/melodic/api/vision_msgs/html/msg/Detection2DArray.html) message output of the node, subscribe to the detectnet/detections topic:

$ rostopic echo /detectnet/detections

## Resizing the OpenGL Window

**This functionality is not yet implemented in the jetson.utils.**

When using the OpenGL window via jetson.utils, the window will display the image in the top left of the screen and fill the rest of the display with black. After significant research on how to resize the window using OpenGL, we learned that that functionality has actually not been implemented yet via the jetson.utils API. In order to change the window size, we edited the C++ code of the API and recompiled. The following changes still only hardcode the size of the window, rather than truly providing a general use fix for the API.

/home/autobot/gis\_bot/jetson-inference/utils/display/glDisplay.cpp

starting at line 155:

// Uncomment to use the Display size instead of Custom (ie. Camera) Size

// const int screenWidth = DisplayWidth(mDisplayX, screenIdx);

// const int screenHeight = DisplayHeight(mDisplayX, screenIdx);

// Comment out the below two lines if using the above code

const int screenWidth = 1300;

const int screenHeight = 750;

After editing the code, we re-compiled and re-installed jetson-inference

$ cd jetson-inference/build

$ make clean

$ make

$ sudo make install

## Next Steps

### Complete the POC

We are at the final stages of a viable proof of concept for our use case. While not capable of fully autonomous operation, we will seek to complete a moving poc over the coming weeks. This bot should be capable of navigating a hallway at CGU, or the outside path north of the Academic Computing building at CGU, taking the required cube map photos, and returning to base. It will perform SLAM using LIDAR and rudimentary navigation based off the AI. Finally, the images should be imported to ESRI tools and viewed on the Oculus Go.

Future improvements to this core set of functionalities will lie in optimizing the route-finding algorithms, further tuning of the AI models, development of ROS modules, and improvements to the motion control algorithms. However, due to the non-generalizable aspects of this type of development, these steps should happen with the specific hardware intended for use moving forward. This will make for both a more robust solution, as well as one that is far more generalizable for real-world use by first responders, C&C, emergency response, hostage rescue, and warfighter scenarios.

### Building a Team to Compete

We have completed a valuable research and understanding phase of development. We have completed initial programs and algorithms to address multiple navigational challenges. We have identified compatible hardware and software and have created custom code to fill many of the gaps in functionality. And we have done so for well under $1000 and in less than a semester, much of which can be carried up to the next level of development.

The next phase of development and research would be an intermediate step towards a goal of competing in multiple national and international challenges. Two challenges that would require some funding support in order to allow teams from the 7 Colleges to compete would be the DARPA Subterranean Challenge and the Lockheed Martin Drone Racing Challenge. Both of the competitions accept teams from individuals or institutions, and include teams from entities like JPL, Carnegie Mellon, MIT, CalTech, and many others, as well as private enterprises setup solely for purposes of the competitions. 1st place prizes are over $1M for each competition.

Building a team to compete at that level will likely be a six-month process, and can be approached incrementally. The MIT racecar platform has become the de facto development standard for racecar AI. While not robust enough for any off-road challenges, it is capable of supporting most types of indoor challenge, including speed tests, and it has sufficient size to carry multiple batteries, sensors, and processors, making it an ideal platform for development and research. The approximate cost of assembling this type of vehicle is around $2000. An ideal scenario would be to have multiple small teams, each building one car apiece.

To compete nationally, the team(s) will require ready access to a 3D printer from the outset, something equivalent to, or better than the Dremel DigiLab 3D45 (approx. $1500 on Amazon) with a 6.7” x 10” x 6” printable volume and remote viewing and printing for shared use in a lab. The lab need only have enough space for a shared workbench with standard tools and amenities – preferably two stations on one bench, sharing some less-frequently used but still necessary tools and equipment like various shapes of screwdriver head, or an oscilloscope. Although used infrequently, when they are needed, they are the only tool that can do the job ($3000 total). Each station should have one small Linux box with a monitor, keyboard, and mouse; and a second monitor, keyboard and mouse for use on the bot. (approx. $1250/ea or $2500 for both). Test environments can be found outside this lab.

### Iterate

Moving forward from this point we should be able to iterate on our solutions and deliver substantial gains with regard to the navigational coding and algorithms, as well as creating optimized datasets for training and tuning of the machine learning models. Further, the machine learning models need not be trained locally, allowing us to take advantage of cloud solutions (Google, AWS, Azure) on an as-needed or when-available basis.

Using the racecar sized vehicle as a baseline, we can introduce variations for locomotion like flying drones, tracked vehicles, and multi-pedal solutions. This should make competition at the national and international levels possible from a technical standpoint, however the cost of custom machined and one-off industrial caliber vehicles will be substantial. By employing a two-tiered approach, we can continue to improve software and algorithmic solutions while minimizing any on-going expenses. We can also compete in the DARPA Virtual Challenge for no significant additional charge.

#### Proposed Preliminary Budget

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Item** | **Description** | **Unit Cost** |
| 1 | Racecar | Chassis, Wheels, Parts, Electronics, Sensors, Motors, and Servos to build the hardware aspects of the vehicle | $ 2,000 |
| 2 | 3D Printer | Dremel Digilab 3D45 Award Winning 3D Printer, Idea Builder with Heated Build Plate to Print Nylon, ECO ABS, PETG, PLA at 50 Micron Resolution | $ 1,529 |
| 3 | Work Bench | 10'+ workbench with appropriate AC power, usb power, lighting, two stools, drawers, static mats, misc | $ 2,500 |
| 4 | Bench Tools | Driver Set and Bits, pliers x 3, wire cutters x 2, wire strippers x 2, hex set, digital soldering iron, hand held multi-meter, workbench dc power supply, misc. | $ 500 |
| 5 | Ocilloscope | PicoScope 4444 Ocilloscope | $ 1,350 |
| 6 | MultiMeter | Fluke 8808A 120V 5.5-Digit Digital Bench Multimeter | $ 800 |
| 7 | Workstation | 2 x Monitors, 2 x Keyboard, 2 x Mouse, wall mounts | $ 1,250 |
| 8 | Field Kit | Driver set and bits, pliers x 3, wire cutters x 2, wire strippers x 2, hex set, digital soldering iron, multi-meter, cart, 2 monitors, monitor stands, led light, tool boxes, ups, misc. | $ 5,000 |
| 9 | Generator | Portable AC/DC Generator | $ 2,000 |
|  |  | **Totals:** | **$ 16,929** |

### Competition Specific Teams

While most of the development can be used universally, the two competitions mentioned above both have very different operating demands and optimizations. One is a small flying drone capable of very high-speed precision movement for a very short period of time (<30s). The other is any viable solution for very large (multiple kilometers of distance) and varied (underground and urban) SLAM.

The Lockheed Martin competition will likely require time on the cloud as its major expense, as well as replacement drones - unfortunately, we can count on crashing a few drones when expressly trying to go as fast as we can through small spaces. However, most of the other expenses can be considered sweat equity. We could expect a year’s budget to be a minimum of $7500 dollars to replace up to 5 drones, and some time (TBD) on a cloud platform.

The DARPA competition will require a rugged vehicle(s). While there are few specific design requirements, the demands of the terrain define some parameters. We do not have a specific design for the vehicle at this point, but it is likely that it will involve some kind of a terrestrial bot carrying significant battery power, imaging, sensing, and AI capability, that can also act as a mobile base station for a small flying drone capable of reaching elevated obstacles like up/down a ladder. The combination must be capable of extended movement (many kilometers) through a variety of wet and dry terrain. This will necessitate some significant hardware expense to compete at this level. Minimum costs for purposes of budgeting for this competition would include significant metal/carbon-fiber fabrication, an on-board computer, and other industrial-grade sensors, motors, and servos likely costing $50,000-$150,000+. This would be in addition to any time on the cloud.