Autonomous Navigation Powered by Jetson TX2 and Robot Operating System

Matthew Haddad, Shem Cheng, Linda Thao, Justin Santos
Graduate Advisor: Aaron Schlichting
Faculty Advisor: Hao Jiang, Ph.D.
San Francisco State University, San Francisco, CA

Abstract

This project was a 10-week research internship conducted at San Francisco State University (SFSU) over the Summer of 2019. The goal was to learn and conduct research about various aspects of applied robotics development in Linux through the use of the Robot Operating System (ROS) software and the embedded computing board, Jetson TX2. By combining the Jetson TX2 with an Arduino and other hardware components, we were able to create a framework in ROS for autonomous navigation aided by computer vision, which was implemented with TensorRT and OpenCV.

1. Introduction

The origin of what would become autonomous driving can be traced to the 1920s when demonstrations of driverless "phantom autos" drew spectators from cities across America.[8] These cars were considered driverless by the fact there were no humans in the car, but were remotely controlled by a human instead of a computer. One promise of autonomous driving that attracted people was safety. As Fabian Kröger noted in his report, *Automated Driving in Its Social, Historical and Cultural Contexts*, mass motorization of America began in the 1920s which led to an insurgence of automobile related accidents. As driver error was blamed as the primary cause of these accidents, the possibility of eliminating the role of a human driver appealed to many.[9] In recent years, advances in convolutional neural networks (CNNs) and powerful embedded computing boards have made computer vision for autonomous navigation feasible with real time image processing.[24]

The goal of our group was to use the Jetson TX2 through ROS to create an autonomous robot capable of obstacle avoidance through sonar and computer vision. The remainder of this paper is organized as follows: Section 2 is a literature review of two previous experiments, one that designs a deep learning network for computer vision and another that utilizes computer vision for a mobile robot. Section 3 introduces our embedded computing board, NVIDIA's Jetson TX2 and our reasoning for using it in our research. Section 4 describes the other hardware components used for our robot. Sections 5 and 6 introduce ROS and the specific packages used for our project. Section 7 outlines the different approaches to computer vision and machine learning that were attempted. Section 8 details the procedures undertaken in our project. Finally, Sections 10 and 11 outline our results and conclusion of our project.

2. Literature Review

2.1 Using Convolutional Neural Networks to Enhance Image Recognition with Deep Learning

Convolutional Neural Networks By using image depth information with deep learning architecture design image recognition algorithms, it is possible to train a program to identify objects on the road regardless of variations of weather, lights, and shadows. Most deep learning methods can be used to enhance image recognition accuracy at the price of speed. With the use of efficient convolutional neural networks, however, deep learning can be used to improve image recognition speed and accuracy.[18]

2.2 DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car

DeepPicar is a small autonomous car that mimics NVIDIAs full size car DAVE-2. This vehicle makes use of a Convolutional Neural Network (CNN) for navigation. By taking data from its camera and processing it through the neural network, it could learn in real time from the surrounding environment while detecting and navigating around objects. The final architecture contained, "9 layers, 27 million connections, and 250K parameters."[15] Using partitioning to protect CNN workload resulted as ineffective because of the limited processing power of the Raspberry Pi 3 (RPi3) used for processing in their project. The RPi3 was capable of running the CNN, but would experience up to 11.6X slowdown because of shared resource contention. [15] The Jetson TX2 has larger cache memory and random access memory (RAM), which makes it better suited to our real-time application of autonomous navigation. [19]

3. Jetson TX2

3.1 Why Jetson

Because autonomous driving requires large amounts of parallel processing power, a large number of graphics processing units (GPU) cores can be utilized for more efficient neural networks and image processing. [12]

GPUs are known for their efficient capabilities to process large sets of data, specifically images, rapidly. This is a useful and necessary tool when developing autonomous cars since the computing system needs to filter thousands of images of the car's environment quickly to have the vehicle react accordingly to the situation. For our project, we are utilizing and reviewing NVIDIA's Jetson TX2. This embedded computing board was designed and marketed specifically for "high performance AI at the edge," housing both the graphics processing unit (GPU) and central processing unit (CPU) on the same chip.[22] By transferring data quickly through the

chip's system fabric, parallel processing using both the GPU and CPU can be achieved in real time. This collaboration allows for faster image processing which will then aid with developing the deep neural network for the autonomous car.

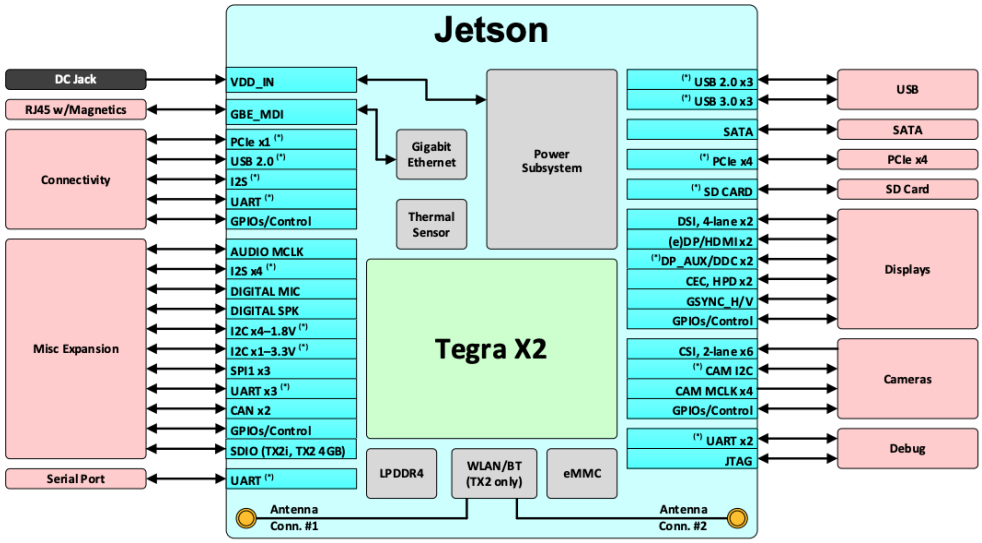

Figure 1: Jetson TX2 Series Module Block Diagram [23]

The Jetson TX2 is the second generation embedded computing board developed by Nvidia as part of their Nvidia Jetson series. Using the Tegra X2, the Jetson TX2 boasts better power efficiency while having higher processing power than its predecessor. This makes it ideal for applications which demand high powered computations in environments where power is limited. Examples of such applications include autonomous vehicles, drones, and virtual reality.

3.2 Jetson TX2 GPU

NVIDIA's Jetson TX2 includes the Tegra X2 system-on-chip (SoC) design processor, where all components are contained on a single chip. The Tegra X2 incorporates a quad-core 2.0-GHz 64-bit ARMv8 A57 processor, a dual-core 2.0-GHz superscalar ARMv8 Denver processor, and an integrated Pascal GPU. The integrated GPU lets the GPU share dynamic random access memory (DRAM) with the CPU. This allows both the GPU and CPU to run more efficiently on low power, between 5 watts at max efficiency and 15 watts at max performance. This efficiency allows for minimal cooling and provides additional space. This low-power system was designed for accelerating machine learning making it a reliable choice for mobile robotics applications. [24]

3.3 Capabilities

We were given the option of working with either a Jetson TX1 or TX2, we decided that a TX2 would be a better option for our project. It is designed to run on Ubuntu 16.04 or 18.04. As the successor to the Jetson TX1, the Jetson TX2's main advantage is its high performance throughput and power efficiency. [20]

The TX2 uses Deep Learning to become autonomous through training and inference. During the training phase, the network receives large sets of data so it can start to recognize patterns based on the examples given. Inference uses all of these tools to make predictions using the datasets from training for implementing image recognition, object detection, and segmentation.[25] Image recognition groups photos into object types so it can identify the object accurately. Object detection is a more focused level of recognition by using a camera to identify an item in a live surrounding space. With segmentation, the camera can use edge detection to map its surrounding space. It is possible because the Jetson TX2 has an encoder and decoder for 4K pixels at 60 frame rates per second.

4 Arduino & Hardware

4.1 Microcontroller & Shields

In order to not draw processing power away from the Jetson's Machine Learning algorithm, we used an external microcontroller (the Arduino Mega) in order to control all of our system's hardware.[26] In addition, the General Purpose Input Output (GPIO) pins on the Jetson are unable to output pulse width modulated (PWM) signals to motors. To control the motors, we use the Arduino Motor Shield Rev3, a driver module that attaches to the Arduino Mega and is created specifically to allow efficient control of the motors through the Arduino IDE.[6] On top of the motor shield was the Arduino Sensor Shield V4.0.[27] This shield was used primarily to expand the number of connections available for providing 5V of power and a connection to ground. Through an FCTP chip, the Arduino can translate its serial data into a usb connection that interfaced with the Jetson.[26]

4.2 Sensors & Actuators

The motors we used to drive our robot were both the Pololu 50:1 Metal Gearmotor 37Dx70L mm with 64 CPR Encoder. [1] These brushed DC motors have integrated quadrature encoders that trigger 64 voltage spikes over each full rotation of the motor shaft. This translates to 3200 counts per revolution after accounting for the 50:1 gear ratio of the motors. These voltages can be converted through a C script to determine the degrees of rotation of each motor, making sure each wheel is performing as expected. [34]

For sensing its environment, the robot used two kinds of sensors. First is the HC-SR04 sonar sensor.[28] This sensor has two terminals, one which emits an ultrasonic wave and another that detects the wave and calculates the amount of time that has passed. By altering the basic kinematics equation speed = distance / time and using the speed of sound at which the wave travels, the distance of the nearest object can be calculated. The final formula is distance = (speed of sound) * time / 2. The formula is divided by two as the total distance includes the wave propagating forward and returning back to the sensor.

The final type of sensor was a Logitech 720p camera, which plugs directly into the Jetson TX2 via usb.[29] This camera is what enabled the machine learning to take place through image processing. The raw camera images could be taken in and processed through ROS and the Caffe or TensorRT deep learning framework to produce a final output. For our project the camera was trained to recognize humans situated at various distances, and react accordingly.

5. ROS - Robot Operating System Overview

The main implementation of all software for the robot control was done through ROS, the Robot Operating System. ROS is an integrated package manager for Linux designed for efficient cross-platform communication and open-source robotics development. ROS programming involves creating and connecting nodes with specific functions through topics to allow the nodes to exchange information.

5.1 Why ROS

ROS is a robust software framework that contains many features that assist in creating robots. Their unique communication system for message passing, robot geometry and description libraries for defining dimensions, and robust tools for pose estimation, localization, and navigation, make it a powerful ecosystem for precisely controlled robotics projects. Its language independence, agnostic libraries, and flexible compatibility allows researchers and developers to share and use nodes and packages regardless of their software frameworks or the language of their program. To date there are over 3000 packages publicly available for download which provide frameworks for various kinds of sensing and movement[30]

5.2 Catkin

ROS has a very specific structure that is designed around its custom build program, called catkin. Catkin is built on top of the CMake compiler with additional Python macros for added efficiency. [2] All files for a project are stored inside a catkin workspace, which allows an entire project to be compiled at once. The src folder contains the top-level CMakeList.txt file, which can have specified parameters to override other CMake instructions. In our case, the top-level CMake file had to be edited to set the default compiler as C++. For the compiler to know

what files to build and in what languages, all executable files are created inside packages. Each package must contain two files, CMakeList.txt, and package.xml. These files specify the necessary compiler, the location of the build target, as well as all dependencies and their locations.

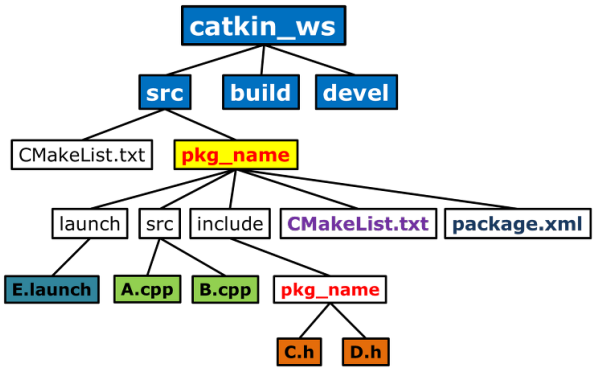

Figure 2: Organization of catkin workspace [14]

5.3 Nodes

Inside a catkin package, the src folder contains the package's "nodes", the main robot control implementation files. These files are generally written in C++ or Python but catkin supports many languages such as Lisp, Javascript, and Ruby. If a node has dependencies on specific libraries, their header files must be included in the include directory, as well as having their location specified in the CMakeList.txt file. The standard dependencies for most nodes are roscpp and rospy for compiling C++ and Python, as well as the std_msgs package for general communication. Finally, the launch folder contains launch files that can use the ROS tool roslaunch to launch multiple nodes at once. [13]

5.4 Rostopics, Publishers, Subscribers

Once multiple nodes are launched, they need to be able to communicate with each other. ROS uses a proprietary transport layer called TCPROS that uses standard TCP/IP sockets for sending messages. This communication takes place over topics, which are named channels for exchanging data of a specified data type. On any topic, nodes can be specified as either a publisher or a subscriber. Publishers push new messages to a topic, while subscribers can take in and process data on a topic. [21]

5.5 ROS Master Node

Nodes do not automatically communicate with other nodes. This is why a ROS Master Node is required. A Master Node can be started by calling the roscore command, or alternatively can be automatically launched through launch scripts. Once a master node is active, new nodes

first register themselves with the master and establish the topics to which they will publish and subscribe. From there, publisher nodes form direct connections with their subscribers. There are more advanced communication techniques called services and actions which involve two-way communication between two nodes, which we did not utilize in our project.[13]

5.6 Gazebo

Gazebo is a robotic simulator for designing, training, and testing algorithms for artificial intelligence. With the model editor we can create and implement our autonomous robot into open space to allow further testing. By creating a URDF file with specified parameters for each link and node in the model, "All simulated objects have mass, velocity, friction, and numerous other attributes that allow them to behave realistically. " [7] Users can also design their own environment with different obstacles and pathways to simulate a robot's response to its environment.

6 ROS Packages

6.1 Rosserial

Rosserial was the most important package that we used for our project. Rosserial is a is a protocol for wrapping ROS messages to be sent over a serial port. This was used to send and receive data from the serial port that was connected to the Arduino Mega. The package is capable of managing 25 publishers and 25 subscribers over the serial port, which are all combined into a single packet for sending information.[40] The packet has header and tail portions that allows messages from multiple topics to be sent in the same packet of serial information. In order to use rosserial, a ROS master node must first be running. Then, the ROS-side server is established using the rosserial_python library that is included in the package. From there, nodes can be written to take data from sensors or control actuators via the serial port.

6.2 NVIDIA Jetson Developer Toolkit

The NVIDIA Jetson Toolkit is a metapackage that was developed as a joint project between NVIDIA and Cal Poly SLO.[31] It acts as a robotics framework for the Jetson TX1 embedded board and the ROS Jade Linux tools. It includes example scripts for low level robot control, as well as drive inference to interface with the machine learning done through Caffe. While the framework provided useful reference materials, much of it did not directly compile into our project when we started. A lot of troubleshooting for our project involved editing files created by this metapackage. We were using a Jetson TX2 with ROS Kinetic, which caused package mismatches and dependency errors because of the different hardware and software. The package deals with implementing low level control of the hardware through arduino, then sending messages over rostopics to establish a differential drive. Following in Figure 3 is an

excerpt of the code to control the hardware. First the motor driver is initialized, which in our case was the Arduino Motor Shield R3. This was different than the motor shield used in the base project, so the libraries had to be updated and the CMakelists.txt changed to reflect new dependencies. Next, the baud rate is set for the hardware. The baud rate determines the frequency of serial communication and must correspond exactly to the rate at which messages are sent and received in ROS. Finally, the node is initialized and all of the component hardware is set up as either a publisher or a subscriber to their corresponding topics. The advertise function is used to create a publisher in a node, and subscribe will initialize a subscriber.

```
void setup() {
  md.init();
  nh.getHardware()->setBaud(115200);

  nh.initNode();

  nh.subscribe(motor_right_speed_sub);
  nh.subscribe(motor_left_speed_sub);

  nh.advertise(motor_right_current_pub);
  nh.advertise(motor_left_current_pub);

  nh.advertise(encoder_left_pub);
  nh.advertise(encoder_right_pub);

  for(uint8_t i = 0; i < SONAR_NUM; i++) {
    nh.advertise(sonar_pub[i]);
  }
```

Figure 3: Rosjet.ino setup excerpt [31]

Once this sketch has been uploaded through the Arduino IDE, rosserial will be able to publish and subscribe to the topics listed above, controlling the robot.

6.3 Jet_Control & Diff_Drive_Controller

Jet_Control is another metapackage for ROS that contains a number of tools for implementing control systems in robotics. Serial communication then allows a differential drive system to be implemented.[39] Differential drive is an important concept in many ROS applications as it allows a robot to take advantage of a specific data type, called geometry_msgs/Twist. This message is formed from two vectors, linear and angular, which each contain three entries for x, y, and z. These entries describe commands for how far the robot should move in a certain direction, and how much they should rotate about each axis. For our project we only used the linear.x and the angular.z portion, which describe the velocity at which the robot should move forward and angular velocity about its center axis.

7 Computer Vision

7.1 Caffe & DIGITS

On this project we worked in parallel with a computer engineering team that was focused on creating a custom neural network for autonomous navigation. The algorithm used was Caffe, a deep learning framework developed at UC Berkeley. The framework was created by Yangqing Jia for his PhD, where it is described as, "A BSD-licensed C++ library with Python and MATLAB bindings for training and deploying general-purpose convolutional neural networks and other deep models efficiently on commodity architectures." [3] In order to implement this framework, NVIDIA DIGITS was utilized to build and visualize the DNN (deep neural network), and collect and manage training data. Once the computer engineering team created their custom network, we analyzed and edited their C++ code to enable a stable drive inference based on the incoming camera data for use on their TX1 system.

7.2 TensorRT & OpenCV

Because the computer engineering team referenced in section 7.1 utilized different software versions in creating their neural network, we had to implement our own Deep Neural Network (DNN) to process image data. To do this, we took advantage of the TensorRT programmable inference accelerator, released by NVIDIA. This program is built on CUDA, a GPU computation language that works directly with the NVIDIA embedded system architecture to effectively utilize the Jetson TX2's GPU-based computing power. Instead of collecting our own training data, we used the inference model GoogLeNet for image recognition. This then had to be integrated through OpenCV and a ros_deep_learning node to interact with our robot.

OpenCV (Open Source Computer Vision) is a library of functions for C++ made for accelerating machine learning and artificial intelligence applications. According to Ivan Culjac in IEEE, OpenCV "is extensively used around the world, having >2.5M downloads and >40K people in the user group." [4] Using this library, a Python file was created inside the catkin workspace that could convert between image types. Finally, the raw camera data could be fed through the GoogLeNet inference model and used to determine the motion of the robot.

8 Procedure

8.1 Hardware Assembly & Jetpack Installation

When we began this project, no one on our team had knowledge of software development on Linux, or working with the Robot Operating System. Because of this, the first week of our program involved learning the layout of Ubuntu Linux, ROS, and catkin workspaces, as well as working through the ROS tutorials provided on the ROS wiki. [13] After some initial research[16], we decided that it would be most advantageous to use ROS Kinetic, supported on Ubuntu 16.04,

instead of using the newest version ROS Melodic, available on Ubuntu 18.04. The reasoning for this was that the Kinetic release has been around for longer while still being one of the more recent releases. As an open source platform, there are more packages and more community support available through forums. [16]

Next we flashed the Jetson TX2 with Jetpack 3.3 and installed ROS Kinetic and its required tools. The Jetson was flashed from a host computer with the OS being transferred over usb and the other components and libraries being sent over ethernet. Finally we could connect the TX2 to a monitor to view the file system and verify a correct installation.

8.2 Setup of Arduino & Catkin Workspace

Once our Operating System was online and ROS had been installed, we began development on our robot. First we attached the Arduino Motor Shield and connected the wiring for our two DC motors. Next, we wrote a file in C to control the motion of the motors. After that was successful we connected the HC-SR04 ultrasonic sensor to the Arduino and made use of the library NewPing to read distance measurements taken from the sensor. [36]

The last hardware system that had to be configured were the encoders on the motors. Though we could not initially interpret the encoder data, the voltages they transmitted could be converted to degrees of rotation by implementing a C script involving attaching interrupts to each of the encoder pins and converting the number and order of the incoming voltages to rotation. After researching encoder implementations and libraries [33], we wrote our own interrupt functions that could track the number of full rotations of each wheel accurately to each full rotation. [32] However, they were not accurate to smaller degrees of rotation. We then implemented the Encoder Library written by Paul Stoffregen for quadrature encoders, which performed very well for our application. [34]

Once the hardware components were functioning properly through the Arduino IDE, we moved on to integrating the hardware with our ROS environment. There are several packages available for interfacing between Arduino and ROS, and the package we decided to use first was ros_arduino_bridge, a metapackage for transmitting data over the serial port.

We created nodes for all of the hardware and ran them through ROS to make sure they could perform as intended. Then, we wrote a launch script to launch the individual hardware nodes together. After this produced compile errors, we realized that when running ROS projects that involve Arduino or other microcontrollers, only a single sketch can be uploaded to the board at one time. This is because when a rosserial node is running, it needs complete control of the Arduino's serial bus. [40] We then consolidated all of our hardware control into a single sketch and created a new ROS node to interface with the hardware. We ran into problems uploading this

code as the CMake compiler could not find or recognize the libraries we were using with Arduino. When we specified the full file path to the header files, those files were found, but then their dependencies could not be found. We repeated this process several times until realizing that there must be a more efficient way to integrate Arduino libraries with ROS.

We then decided to start using the rosserial package to attempt a new form of communication from the Jetson to the Arduino. We first tested a node set up to interface with a sonar sensor and publish its data on a rostopic. We were able to do this successfully using the NewPing library by including the library header in the ROS package's include folder and specifying the dependency in the CMakeLists.txt. By the end of the third week, all our hardware code was consolidated into a single sketch and was set up to run all required components through ROS. However, we were getting compilation errors during the catkin_make related to package dependencies that were vague and hard to parse.

We decided to revisit a folder that had been shared with us that involved a joint project created by Cal Poly SLO and NVIDIA that had presentations and files related to building computer vision projects through ROS. While we had gone over the presentations and documents included in the different modules, we previously had not been able to locate the source code for the project. There we found a rosjet folder that contained source code for the 6 nodes required to drive the TX1 based robot. Once we located these files, we imported them onto the Jetson to adapt them for our application on a TX2.

8.3 Development of Software Nodes & Troubleshooting

Using the NVIDIA Jetson Resources was both good and bad for our project. As there was a software mismatch between our ROS and what was used in the project, we had to change the dependent packages to newer versions and edit some code that used outdated syntax. After installing the required packages for ROS Kinetic, we were soon able to run a simulation of the robot in the Gazebo simulator.

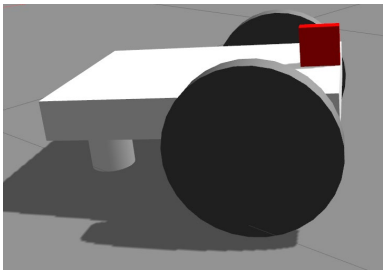

Figure 4: jet_gazebo robot simulation [31]

While the simulation of the robot could launch successfully, running the script jet_real.launch that was meant to control the physical hardware nodes proved to be more difficult. At this point, we ran into an additional problem with the Jetson TX2. The board had

been powered off, and when restarted it launched in a low graphics mode with limited functionality. Eventually after some troubleshooting we decided it would be best to save our files onto a usb drive a re-flash the operating system onto the board using Jetpack.[35] We later discovered that this problem had been caused by the Jetson being shut off without all running processes having ended, which somehow altered the initial boot sequence when it was turned back on. Despite attempts to avoid this issue in the future, the Jetson had to have its operating system reinstalled over the course of the project another 7-8 times due to various other issues such as corrupted files or incomplete installations. [37]

When we were not dealing with hardware problems, we had to manage the hardware setup in the catkin_ws. The NVIDIA package contained a file rosjet.ino that controlled the hardware functions through Arduino. As detailed in the overview the of Jetson Developer Toolkit, the main hardware update required was updating the motor driver library and related files. However, the package seemed to rely on a compilation environment called PlatformIO that was configured within the workspace.[38] We spent some time trying to work with the PlatformIO debugger and compiler system to communicate the messages published from the script onto Arduino and into ROS. The scripts would compile through the build environment, but would not be uploaded onto the board. Eventually we decided to scrap the file system for PlatformIO implementation and upload the sketch directly through the Arduino IDE.

Once the other CMake compile errors were resolved, we could run our launch file to run the nodes that implemented our project.

9.4 Implementation of Computer Vision

Once the jet_real.launch file could successfully launch and control hardware, we attempted to connect it with a computer vision framework. Though we had been working alongside the computer engineering group referenced in section 7.1 that was training a custom neural network for robot navigation, their build environment was too different from ours so their code could not be ported easily so we decided to use the deep learning framework included in Jetpack, TensorRT. We could then take advantage of pretrained image recognition inference models, such as GoogLeNet, which allowed us to recognize many different kinds of objects.

Once we installed and tested our inference model with some sample images, it had to be connected to ROS through a ROS node. There is a ROS package called ros_deep_learning that was developed for integrating TensorRT with a usb camera through ROS.[11] This should have allowed the model to analyze raw data from the usb camera, but unfortunately there was a data type mismatch. The camera was outputting images in the rgb8 format, but the inference node required images to be in the bgr8 format for TensorRT. To work around this, an additional node had to be created by utilizing OpenCV. The OpenCV library contains a built in function to

convert between image types, so a new package was created in ROS to implement this additional layer to process the camera data. Though we could use a prebuilt python script, the conversion node had to be compiled through CMake, which produced an error linking the target libraries.

10 Results

The physical result of our project is the assembled robot pictured below. It includes all hardware components, the most important of which are the Jetson TX2 and Arduino Mega. It also has a motor shield and sensor shield connected to an ultrasonic sensor and two DC motors. Everything was mounted on the aluminum chassis using either double sided tape or nuts and bolts.

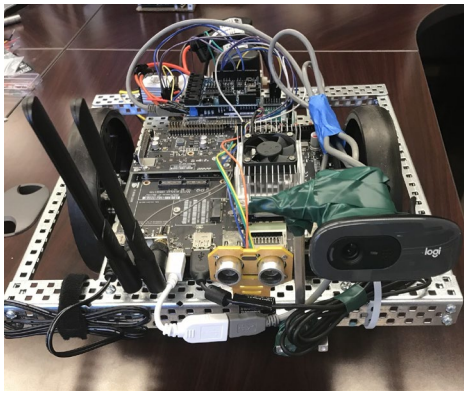

Figure 5: Completed robot assembly

In terms of software implemented, several features were successful. The motors, encoders, and sonar sensor were all connected onto Arduino shields which then sent and received data over the serial port through ROS. That data from the sensors could be read through ROS, and commands can be sent to the motors on either the /arduino/motor_left_speed and /arduino/motor_right_speed topics or the /cmd_vel topic which takes advantage of the robot's differential drive algorithm.

A Logitech 720p usb camera was also attached directly to the jetson, where its live data could be viewed over the /usb_cam/image_raw topic once the node was launched. Two packages were successfully installed for computer vision, jetson-inference and ros_deep_learning. Jetson-inference allowed us to utilize vision primitives such as imageNet, detectNet, and segNet for image recognition, localization, and segmentation. Ros_deep_learning allows these inference algorithms to be connected to a ROS node, where the usb camera can be fed through the computer vision node to produce an output.

Unfortunately, we were unable to connect the camera stream to the motor controls by the end of the ten week program. However, all components were successfully implemented

individually and we feel that with a week or two more we would have been able to run a homogenous navigation system.

11 Conclusion

In conclusion, this project provided an effective framework for research and applied development in Linux. Our team learned valuable skills related to engineering research, as well as producing tangible results from our project. Working with the Jetson TX2 proved challenging but rewarding when the computer vision could be applied in real time. For peripheral hardware, we were able to convert low level voltage data from the sonar sensor into usable distances, as well as feed the raw image data from the camera through an inference node to perform image recognition. The cmd_vel topic could be utilized to convert linear and angular velocity commands into pulse width modulated signals to be sent to each motor, allowing precise control of robot navigation. Overall, we were able to successfully apply the concepts learned in our research and development process to write and adapt code for mobile robotics on the Jetson TX2 through use of the Robot Operating System. We learned valuable software engineering skills related to troubleshooting different kinds of errors. The implementation of this project changed significantly over the program's duration, which led to us gaining a more in depth understanding of ROS and some of the low level implementation for the hardware components. In our final version we used libraries for the sonar[36], motor control[1], encoder readings[33], and differential drive algorithm[39], whereas during development we wrote scripts in C, C++, or Python to execute these functions manually. We also gained research insight into connecting convolutional neural networks for computer vision[18] to a ROS controlled robot through a ros_deep_learning node. [11]

12 Citations

[1] "Pololu - 50:1 Metal Gearmotor 37Dx70L Mm with 64 CPR Encoder." *Pololu Robotics & Electronics*, www.pololu.com/product/2824.

[2] "Catkin." *Catkin - Catkin 0.7.18 Documentation*, docs.ros.org/kinetic/api/catkin/html/index.html.

[3] Jia, Yangqing, et al. "Convolutional Architecture for Fast Feature Embedding." *ACM Digital Library*, ACM, 3 Nov. 2014, dl.acm.org/citation.cfm?id=2654889.

[4] Culjak, Ivan, et al. "A Brief Introduction to OpenCV." *IEEE Conference Publication*, IEEE, 2012, ieeexplore.ieee.org/abstract/document/6240859.

[5] Antsaklis, P.J., et al. An Introduction to Autonomous Control Systems - *IEEE Journals*; Magazine, ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=88585.

[6] Arduino Motor Shield Rev3, store.arduino.cc/usa/arduino-motor-shield-rev3.

[7] Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator - *IEEE Conference Publication*, ieeexplore.ieee.org/abstract/document/1389727.

[8] "'Phantom Auto' will tour city". The Milwaukee Sentinel. Google News Archive. 8 December 1926. Retrieved 8 August 2019

[9] "Autonomous Driving - Technical, Legal and Social Aspects: Markus Maurer." Springer, Springer-Verlag Berlin Heidelberg, www.springer.com/de/book/9783662488454.

[10] Cashmore, Michael, et al. "ROSPlan: Planning in the Robot Operating System." Association for the Advancement of Artificial Intelligence, www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/viewPaper/10619.

[11] Dusty-Nv. "Dusty-Nv/Jetson-Inference." GitHub, github.com/dusty-nv/jetson-inference/blob/master/docs/deep-learning.md.

[12] "NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge." NVIDIA Developer Blog, 5 Sept. 2018, devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/.

[13] "ROS Wiki." Ros.org, wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

[14] "ROS Wiki." Ros.org, wiki.ros.org/ROS/Introduction.

[15] Bechtel, Michael G., Elise McEllhiney, Minje Kim, and Heechul Yun. "DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car." ArXiv.org. 07 Feb. 2018. 08 Aug. 2019 <https://arxiv.org/abs/1712.08644v2>.

[16] Quigley, Morgan, Brian Gerkey, and William D. Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.

[17] Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System." In *ICRA workshop on open source software*, vol. 3, no. 3.2, p. 5. 2009.

[18] Krizhevsky, Alex, et al. "ImageNet Classification with Deep Convolutional Neural Networks." *NIPS Proceedings*, Neural Information Processing Systems (NIPS), 2012, papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[19] Knauerhase, Rob (2008). "Using OS Observations to Improve Performance in Multicore Systems". *IEEE Micro*. doi:10.1109/mm.2008.48.

[20] Kanokwan Rungsuptaweekoon, et al.*Evaluating the Power Efficiency of Deep Learning Inference on Embedded GPU Systems - IEEE Conference Publication*, ieeexplore.ieee.org/abstract/document/8257866.

[21] "ROS Wiki." Ros.org, wiki.ros.org/ROS/Topics.

[22] "NVIDIA Jetson TX2: High Performance AI at the Edge." *NVIDIA*, www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.

[23] "DATA SHEET NVIDIA Jetson TX2 Series System-on-Module." NVIDIA, https://developer.download.nvidia.com/assets/embedded/downloads/secure/tx2/Jetson%20TX2%20Module%20Data%20Sheet/Jetson_TX2_Series_Modules_DataSheet_v1.2.pdf

[24] Amert, Tanya, et al. “GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed.” *2017 IEEE Real-Time Systems Symposium (RTSS)*, 2017, doi:10.1109/rtss.2017.00017.

[25] Gatys, Leon A., et al. “Image Style Transfer Using Convolutional Neural Networks.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, https://ieeexplore.ieee.org/document/7780634.

[26] “Arduino Mega 2560 Datasheet”, Arduino. https://www.robotshop.com/media/files/pdf/arduinomega2560datasheet.pdf

[27] “Shields.” *Arduino*, https://www.arduino.cc/en/Main/ArduinoShields

[28] “Ultrasonic Ranging Module HC - SR04.” *Sparkfun,* https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[29] “Logitech C270 HD Webcam, 720p Video with Built-in Mic & Lighting Correction.” *Logitech C270 HD Webcam, 720p Video with Built-in Mic & Lighting Correction*, www.logitech.com/en-us/product/hd-webcam-c270.

[30] “ROS Wiki.” Ros.org, wiki.ros.org/ROS/APIs.

[31] “NVIDIA / Cal Poly Robotics Teaching Kit with Jet”, *Cal Poly San Luis Obispo, NVIDIA,* 2016

[32] “How to Read Data from a Rotary Encoder with ATmega328.”*Arduino Stack Exchange*, 2015, arduino.stackexchange.com/questions/11962/how-to-read-data-from-a-rotary-encoder-with-atmega328.

[33] “Reading Rotary Encoders.” *Arduino Playground - RotaryEncoders*, playground.arduino.cc/Main/RotaryEncoders/.

[34] “Encoder Library.” *PJRC*, www.pjrc.com/teensy/td_libs_Encoder.html#optimize.

[35] Alvarado, Luis. “How to Fix ‘The System Is Running in Low-Graphics Mode’ Error?” *Ask Ubuntu*, 2013. askubuntu.com/questions/141606/how-to-fix-the-system-is-running-in-low-graphics-mode-error.

[36] Eckels, Tim. “NewPing Library.” *Arduino Playground - NewPing Library*, dev.playground.arduino.cc/Code/NewPing

[37] *“*JetPack 3.3 failed to install CUDA on TX2*” Devtalk.nvidia.com*, devtalk.nvidia.com/default/topic/1044733/jetson-tx2/jetpack-3-3-failed-to-install-cuda-on-tx2/.

[38] Platformio. “Platformio/Platformio-Core.” *GitHub*, 24 July 2019, github.com/platformio/platformio-core.

[39] “ROS Wiki.” Ros.org, wiki.ros.org/ROS/diff_drive_controller.

[40] “ROS Wiki.” Ros.org, wiki.ros.org/ROS/Rosserial.