Contents

List of Figures

List of Tables

Preface

Chapter 1: Introduction to DATA61, CSIRO

This chapter describes the history and main business activities of this company.

Chapter 2: Training Experience

Chapter 3: Conclusion

Acknowledgment

Abarajithan Gnaneswaran,

Undergraduate,

Department of Electronics and Telecommunication Engineering,

University of Moratuwa.

1 Introduction to Training Establishment

1.1 Company Overview

Commonwealth Scientific and Industrial Research Organization (CSIRO) is the Australian federal government agency for scientific research and development. CSIRO has its headquarters in Canberra, Australia and several branches across the world, with over 5500 employees. CSIRO is known for the development of Wi-Fi, Atomic absorption spectrography and the polymer banknote which have changed the lives of millions of people around the world.

CSIRO consists of many parts: Agriculture and Food, Data61, Energy, Land and Water, Mineral Resources...etc with research centers in several cities of Australia. DATA61 is a part of CSIRO that aims on developing a data driven future for Australia. DATA61 consists of multiple groups: robotics and automation group (RAG), data privacy group, mobile security group, distributed sensor networks...etc.

I worked in the Pullenvale (Brisbane) branch of CSIRO. It is called the 'Robotics hub of Australia' due to the large number of robotics projects, facilities and researchers present in the Pullenvale branch. The robotics and automation group of CSIRO is known worldwide for their state-of-the-art SLAM (Simultaneous Locomotion and Mapping) algorithms.

Fig. 1.1. DATA61 logo

1.2 Company History

It was formed in 2015 by merging NICTA (National Information and Communications Technology Australia Ltd) with CSIRO's data science section.

1.3 Organization Structure and Hierarchy

1.4 Areas of Interest

1.5 Current Situation

The RAG of CSIRO was recently selected as one of the six teams worldwide for the DARPA Subterranean Challenge by United States Department of Defense. Therefore the next four years of research in Robotics in CSIRO will be more focused on developing robots that can simultaneously map and navigate underground tunnels, caves and mines without GPS or reliable communication with humans. For this task, incorporating machine learning into the workflow of algorithm development and testing is of paramount importance for all researches in RAG. I addressed this problem by developing an efficient end-to-end pipeline for this and demonstrating it through two projects.

1.6 Impacts on Sri Lankan Industry

1.7 SWOT Analysis

1.7.1 Strengths

1.7.2 Weaknesses

1.7.3 Opportunities

1.7.4 Threats

1.8 DARPA Subterranean Challenge

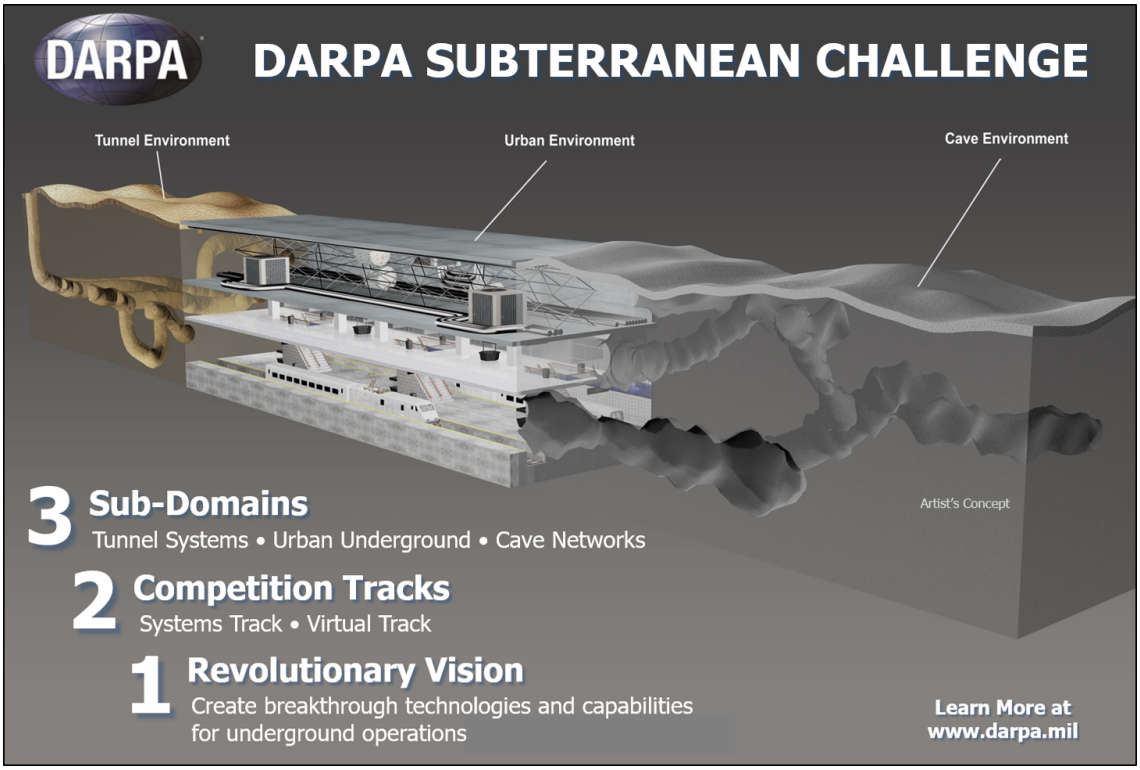

Fig. 1.2. CSIRO focuses on DARPA challenge

1.9 Usefulness to the Country

1.10 Suggestions to Improve the Company

2 Training Experience

2.1 How I got the Opportunity

After graduation, I wanted to continue doing higher studies and become an academic, rather than settling for a job at a company. Therefore, for my internship, I applied for research opportunities in universities and institutes around the world. I got positive response from two or three institutes, one of them being CSIRO. I sent my CV to Dr. Navinda Kottege from RAG, DATA61, CSIRO in February 2018, requesting a research internship opportunity. He asked me to complete a set of 3 timed tasks online to assess my skills in programming and algorithms. He then interviewed and offered me the position as research intern student in CSIRO for 6 months.

Initially I was informed that I am being assigned to the project titled "Computer vision based off-board autonomous UAV Navigation" under the supervision of Mr. Frederick Pauling, a highly capable and friendly senior engineer in DATA61. I was informed that knowledge in ROS (Robot Operating System) and Tensorflow would be necessary, so I spent few weeks learning the basics before the internship.

However, when I arrived at CSIRO, Mr. Frederick Pauling had been promoted into the Group Leader of RAG (Robotics and Autonomous systems Group), to lead the cutting edge robotics research in Australia. Therefore, I could not be assigned into the said project under his supervision. As a result, Uvindu and I was assigned under the supervision of Mr. Nicolas Hudson.

Mr. Nicolas Hudson arrived CSIRO only few weeks before us, after working as a senior roboticist in NASA's Jet Propulsion Laboratory, Boston Dynamics and Google's Machine Learning Division. In CSIRO he wanted to streamline the workflow of the RAG group and incorporate Machine Learning tools into their workflow seamlessly. He asked us to work with him in one of his experimental projects: "Learning Transfer Across RGB, Thermal and IR Modalities in CNNs". After about a week, Uvindu requested for a project that is more focused on hardware. Hence, he asked us to modify Trailnet for autonomous indoor navigation.

2.2 Trailnet: A Classification Network for Autonomous Trail Navigation

2.2.1 Trailnet: An Introduction

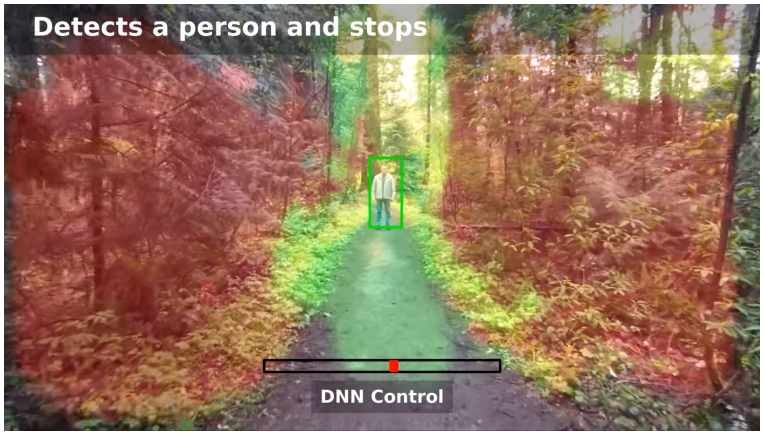

Fig. 2.1. NVIDIA's Trailnet Navigating a Drone

In 2017, four researchers published a paper titled "Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness" [3] (Trailnet paper in short) with the funding of NVIDIA. The paper describes the following:

- Merits of approaching autonomous navigation as a classification problem
- Architecture of their Trailnet CNN, a modified version of Resnet-18. [2]
- Data collection techniques for the Trailnet CNN
- Training Trailnet with a relatively small dataset
- Hardware hierarchy and the command flow between cameras, NVIDIA Jetson TX2 [1] running ROS and the flight control board.
- Usage of YOLO for obstacle detection and algorithm for obstacle avoidance.
- Results and observations after flying the quadcopter autonomously in the forest trail for several kilometers.

2.2.2 Navigation as Classification

Their model was based on a concept discussed in a 2016 paper : approaching autonomous navigation as a classification problem. That is, given an RGB image the CNN would output three probabilities: that of the camera facing left, right or center with respect to the trail. The key advantages in this approach are:

- The effect of noise introduced by human errors during data collection on training are minimized due to discretization.
- Data collection and labeling is straightforward
- Performance can be fine tuned, by adjusting K1 and K2 accordingly. See Figure: 2.2
- The depth of the required neural network is less, compared to the regression network that provides similar accurate performance.
- Can train the relatively shallow network with a relatively small dataset and shorter training time.
- Can be implemented on low powered devices.

In the Trailnet paper They choose Resnet 18 as the basis for their architecture since it is small enough to be run on real time in a power-limited device like Jetson TX2. Resnets (Residual Networks) are special kinds of deep neural network that uses "short circuits" between layer outputs to prevent the problem of vanishing gradients, as a network gets too deep. By employing this technique, researchers have been able to create networks that are thousands of layers deep and still outperform shallower networks. Resnet-18, Resnet-50...etc are popular variants of applying this technique on deep convolutional neural networks.

Trailnet is not an RCNN. That is it does not remember past inputs nor correlate current inputs with past and future values for prediction. It is a simple CNN that gives a twist command based on the current image. Input to trailnet is a 320 x 180 x 3 RGB image and the outputs are six softmax nodes connected to the output of a slightly modified resnet-18. The six output layers signify the probability of the given image facing left, center and right and the robot (or UAV)

being aligned left, center and right on the path. Weighted (by adjustable constants k1, k2) sum of these probabilities provide the angular twist command, which is used to steer the robot. This additional consideration of alignment, prevents the UAV slowly drifting off the center of the trail and crashing with tree branches near the trail edges. Together, the facing and align probabilities correct the course of the UAV to stay in the center of the path. YOLO and SLAM are used for obstacle avoidance.

Data collection was done using a camera rig with 3 cameras facing at three angles (left, center and right). The rig was carried by a person along a forest trail. The video feed from each camera had been then labeled accordingly. Similarly align to left, center and right data has been collected. A pretrained network (previously trained on IDSIA dataset of 40,000 images) had been fined tuned with this collected data. After training, Trailnet was run with ROS (Robot Operating System) and Caffe on Jetson TX2 on board the UAV. Jetson TX2 receives the video frame from camera, processes it and sends it sends command to the flight controller.

2.2.3 Building Trailnet from scratch

One objective of their project was to showcase the capability of NVIDIA's Jetson TX2 high level controller board. Therefore, they had used Caffe framework to build the network and DIGITS framework to train it. However, our key objective in working with this was to build a unified pipeline which all scientists in DATA61 can use. Since most of them were familiar with TensorFlow and since it is the state of the art framework today, I rebuilt the 20-layer network in TensorFlow-Keras and trained it from scratch in CSIRO's supercomputer as I built the pipeline.

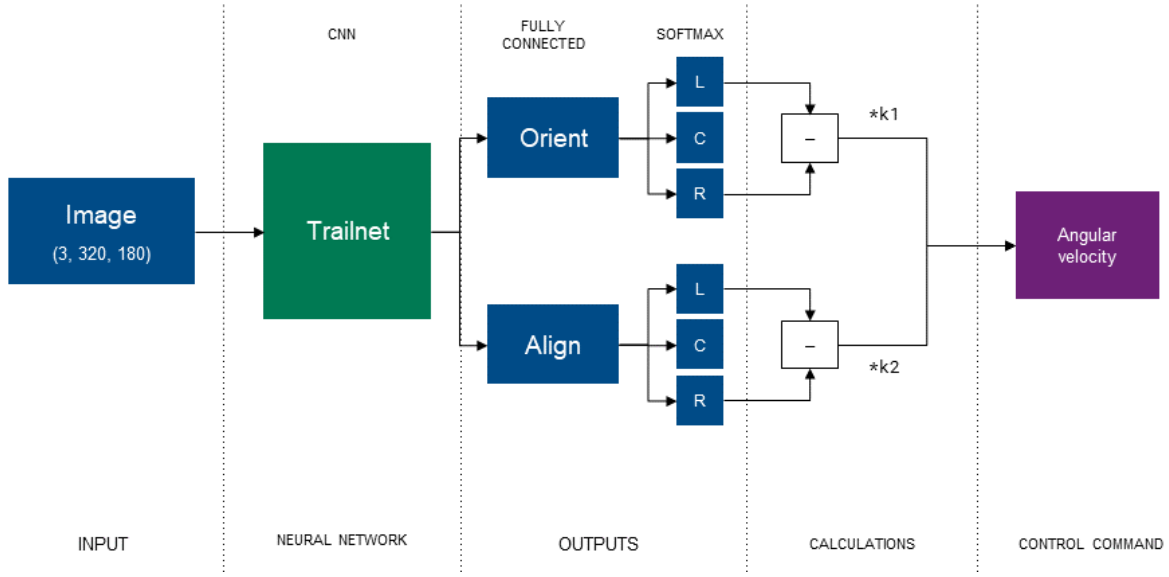

Fig. 2.2. Simplified Trailnet Architecture and Post Processing

2.2.4 Data collection and training Trailnet from scratch

Our goal with this project was to train a robot navigate indoor hallways as a demonstration of our end-to-end pipeline. Hence we mounted three cameras on the robot, facing center, left and right by 30 degrees. We took the robot along the hallways of CSIRO using a remote control and recorded the image stream data as ROS bags. In each hallway, we took the robot on three paths: center aligned, left aligned and right aligned. We then extracted the image stream into an image dataset by taking one image every second (1 fps) from the image stream. The resulting hallway dataset consisted of 120,000 images.

Fig. 2.3. IDSIA Dataset of 40,000 images

Fig. 2.4. Hallway Dataset of 120,000 images

The 120,000 images were stored in the supercomputer and used to train the network. First, align output nodes of Trailnet were frozen and the network (with facing output nodes only) was trained on the IDSIA dataset of 40,000 images. Then, the same configuration was trained on the hallway facing dataset. Finally, the facing-output nodes were frozen and the rest (align-output nodes) were trained on the hallway-align dataset.

Training the network was a laborious process prone to errors. The supercomputer sessions automatically terminate every few hours, requiring me to stay by the CSIRO provided desktop throughout the process. I stayed overnight for five days alone in the office to train this and the other networks.

2.2.5 Deployment on a Robot and Testing

The trained model was optimized into a tf-trt graph (See section: 2.4.6) and executed inside a python based ROS node. The latency was 20 ms, which was enough to process an input image stream at 30 fps during inference. My ROS node also performs necessary calculations and publishes a velocity message (type: geometry_msgs/Twist) to a topic that is subscribed by the motor controller and the predictions (type: Float32MultiArray) for debugging. I also designed it in a way that the constants K1, K2 can be tuned by publishing the constants to a topic that is subscribed by the Trailnet ROS node.

After setting up this way, the robot was tested for its ability to navigate the hallways autonomously. It's response to external disturbances was checked by kicking the robot in either direction, moving it off the center of the track. K1, K2 constants were tuned to provide the shortest response time while maintaining a steady speed when undisturbed. We also created a visualization technique, where the predictions and commands of the robot can be visualized with the image stream. Following images show the testing process and the corresponding visualization.

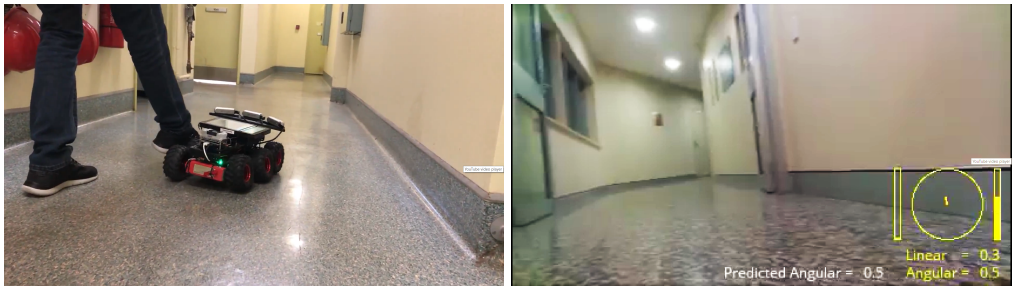

Fig. 2.5. Our indoor Trailnet CNN reacting to external disturbances

THe robot was tested in different hallways, including ones from where we did not collect training data. The robot performed remarkably well in cluttered hallways also, showing its robustness . In addition to that, the response to 90 degree corners was also remarkable, where the robot smoothly turned to follow the hallway.

Fig. 2.6. Our CNN reacting to corners

The results were presented to the fellow scientists in a Robotics Reading Group meeting as a demonstration of the end to end pipeline I designed.

2.3 Building Wallie: A Hardware Platform for Data Collection and Deployment

The Robotics and Autonomous Systems Group of DATA61 is about field robotics. Therefore, it is not sufficient to demonstrate a concept theoretically or just by simulations. Instead, the members are expected to demonstrate our systems and results in complex real world environments.

Therefore, we needed a robot platform on which we can collect data and perform our tests. However, the available platforms of CSIRO were all in use for bigger projects. Hence we were requested to build a robot platform based on Pololu Wild Thumper chassis. This platform was previously assembled by our seniors: Isuru and Tharindu. However, as we describe below, we had to change almost every component of that to create a new robot to suit our specific needs. Our friend from Chile named this robot Wallie, in reference to the Wall-E movie and the fact our robot was initially built to follow walled hallways.

Fig. 2.7. Wallie: The Robot

Firstly, we need a way to fix cameras in the robot for data collection and testing. I sketched a small camera rig to hold three Intel Realsense cameras, each facing: center, left and right at 30 degree angles from center. Samith Ashan, my coworker helped designed it in solidworks. That, a platform for the Jetson TX2 board and a holder for the Li-ion battery were then 3D printed.

2.3.1 Designing the Power Distribution System

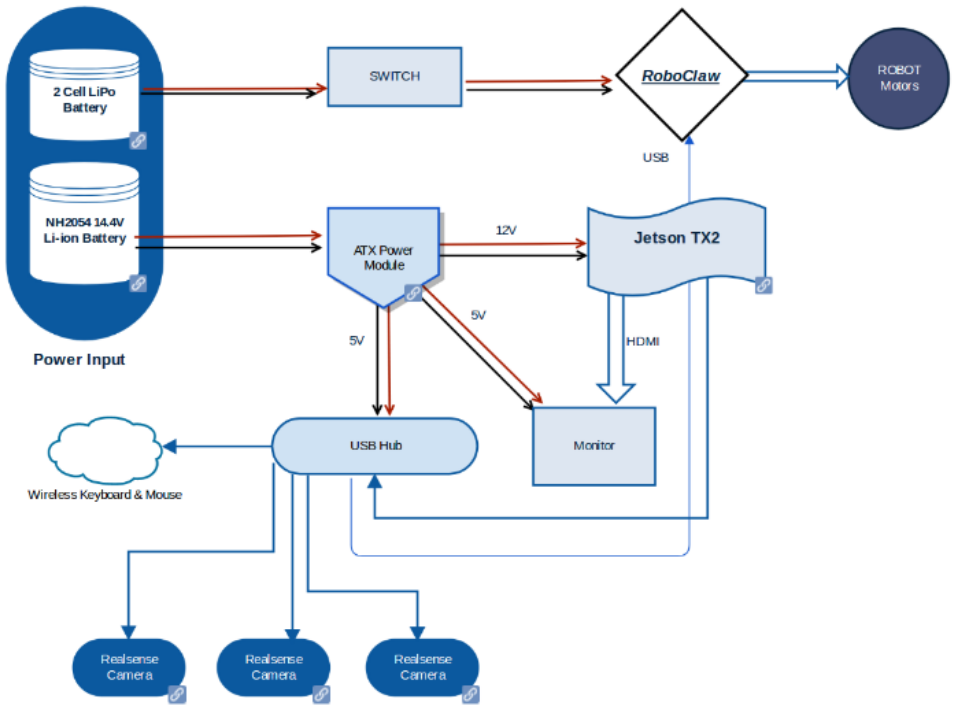

Fig. 2.8. Wallie: Hardware Hierarchy

We then designed a power distribution system for the robot, to deliver power from two batteries to all the sensors and actuators. We consulted Dr. Navinda and Mr. James Brett and came of with the following system. A 3 cell Li-Po delivers power to the Roboclaw Motor Controller, to which six 12V, 2.5D high torque motors are connected through a switch and a fuse (to prevent damage to the motor controller in case if the motors start stalling, as it happened once). The cameras and IMU are powered through an active USB hub. The USB hub LCD display and NVIDIA Jetson TX2 are powered through a robust power supply (12V, 5V) which is connected through a switch to the 7.4V Li-ion battery. Having two separate power sources for high-level and low-level systems helped through decoupling by reducing the chances of power failures from one network damaging the component in the other.

We documented the design of the robot in CSIRO's confluence wiki pages, so that fellow researchers can use it in the future for their own activities. Most of our time in CSIRO (about 75%) was spent in building and debugging the robot platform.

2.3.2 NVIDIA Jetson TX2

Jetson TX2 is device released by NVIDIA as a solution to run computation-intensive programs on board of a low power consuming device. This credit-card sized device contains CPU that can run AMD64 Linux, a powerful GPU that can run multiple neural networks at once using just 7.5W of power. DATA61 RAG is starting to unify and standardize their sensor and actuator APIs. Hence, they were considering NVIDIA's Jetson TX2 and Xavier as potential candidates for the high level controller on their multiple types of wheeled, legged and aerial robots.

Table. 2.1. NVIDIA Jetson TX2 Specifications

GPU	256 CUDA cores of Pascal Architecture
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2
Memory	8 GB 128 bit LPDDR4, 59.7 GB/s
Data storage	32 GB
USB	USB 3.0
Connectivity	Gigabit Ethernet, 802.11ac WLAN, Bluetooth

Hence we began using Jetson TX2 as the high level controller with Orbitty carrier board. We flashed it with NVIDIA's Linux for Tegra kernel and set it up with Jetpack 3.2.1 (Updated to Jetpack 3.3 later).

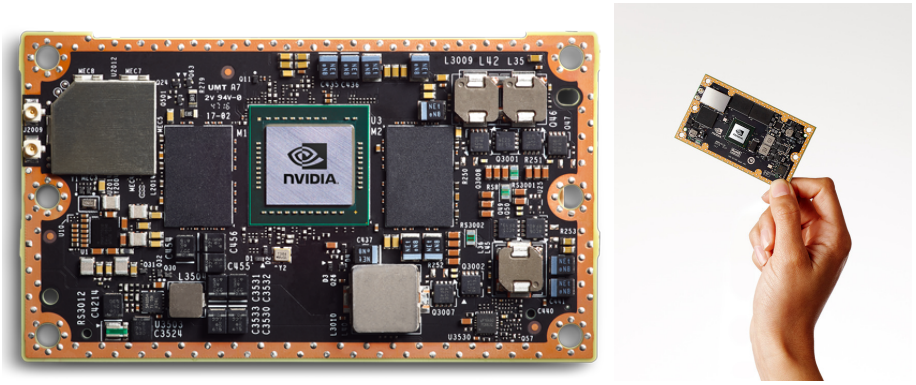

Fig. 2.9. NVIDIA Jetson TX2

2.3.3 Intel Realsense Depth Camera D435

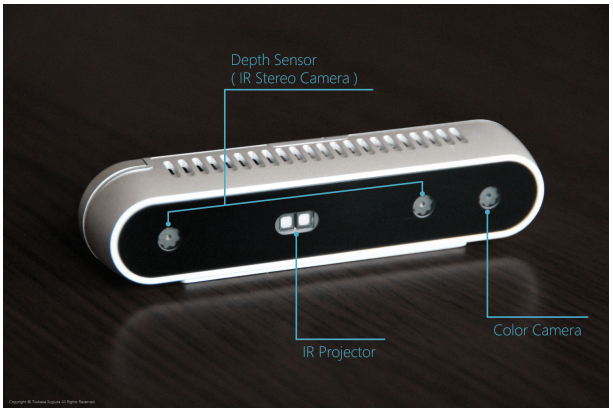

Fig. 2.10. Intel Realsense D435

Intel Realsense depth camera is able to output image streams of RGB, IR and Depth images as per following specifications. Due to the small size and versatility, RAG members considered using this as he de-facto vision sensor in low power robots. It can be connected to Jetson TX2 via a USB-C cable and there is an official ROS package from Intel with which it can communicate via serial and publish the image streams and diagnostic information.

Table. 2.2. Intel Realsense D435 Specifications

Depth FOV (Horz, Vert, Diag)	85.2°x 58°x 94°(+/- 3°)
Depth Stream Output Resolution	Up to 1280 x 720
Depth Stream Output Frame Rate	Up to 90 fps
Maximum Range	Approx.10 meters
RGB FOV (Horz, Vert, Diag)	69.4°x 42.5°x 77°(+/- 3°)
RGB FOV (Horz, Vert, Diag)	1920 x 1080 at 30 fps
Connectors	USB 3.0 Type - C

We used three of these cameras, each facing 30°left, right and center to collect image streams in ROSbags. We used RGB image stream to train our network, since most other robots do not have depth cameras as vision sensors. The depth data we collected was used by our coworker along with our IMU data for SLAM purposes.

2.3.4 Roboclaw Motor Controller

The wild thumper chassis we obtained had the arduino-based TREX motor controller on it. I spent few days trying to come up with an algorithm that can map the remote-control pulses to the PWM commands for smooth maneuvering. I succeeded partially. However, due to the problems described in the next section, we switched to Roboclaw controller, which mixes RC pulses into PWM using their proprietary algorithm.

Table. 2.3. Pololu Roboclaw Motor Controller Specifications

Motor channels	2
Control interface	USB; TTL serial (2-way), RC pulses; PWM
Minimum operating voltage	6 V
Maximum operating voltage	34 V
Continuous output current per channel	30 A
Peak output current per channel	60 A

The main advantage for choosing roboclaw was the availability of a compatible ROS package. However, we then found that the package was not being maintained for the past few years. I spent multiple weeks debugging the package and rewriting some of the driver code.

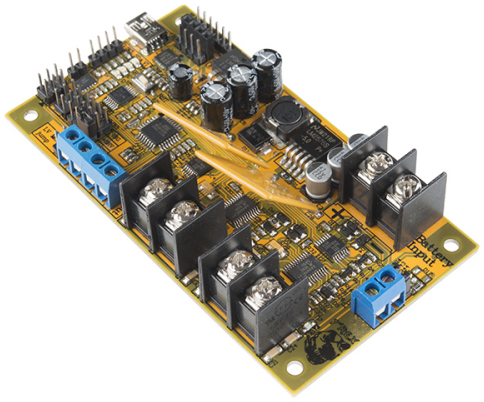

Fig. 2.11. TREX Motor Controller

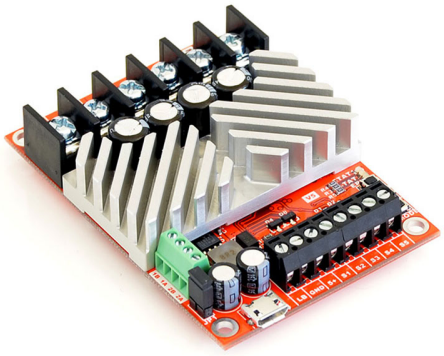

Fig. 2.12. Roboclaw Motor Controller

2.3.5 Lord Microstrain IMU

2.3.6 Problems Faced and Solutions

Firstly, when we assembled wallie with TREX as the low level controller and a custom algorithm to mix RC signals into PWM values for motor control, we found that it was quite difficult to maneuver the robot. Using remote control, we could not make it take tight turns at corners and the robot did not turn smoothly in carpeted floors due to friction. We found three problems that resulted in this:

- Center of mass of the assembled robot was not exactly at the geometric center.
- RC mixing algorithm is not robust enough
- The MOSFET switch used between the battery and TREX limits the current, reducing the torque of motors.

To solve these issues, I improved the mixing algorithm, Uvindu reassembled the robot and we removed the switch. The performance improved, but after few hours of test run, the TREX controller burnt off. Dr. Navinda instructed us to troubleshoot it and submit a report. After intensive testing and discussion with James Brett, Pubudu Aravinda and Samith Ashan, we identified the key issue as the in-built fuses of the old TREX board that failed to function as a large current was drawn when motors stalled on the rough carpet floor. As a result, the hall sensor on the controller got burnt, short circuiting and damaging two MOSFETs on the same side of the H-bridge. Dr. Navinda accepted the report and ordered the Roboclaw controller as replacement (since TREX boards are outdated and unavailable in the market).

The prime advantage of the new Roboclaw was the associated ROS package. However, we soon found that the public package was severely outdated and not maintained. I contacted the manufacturers of Roboclaw and got a copy of the ROS package they use internally. That package also had several issues which we struggled with, throughout the project. After a week of debugging, I realized a main issue was that their driver was not thread safe. That is, when the ROS node receives a message, it runs the specified callback function, which is initiated on a new thread. If multiple such function calls are made in quick succession, it results in multiple threads that try to access the same resources (serial port). This results in unexpected frequent crashes. I changed

the driver package to be thread safe and fixed some issues with incompatibility between velocity commands and odometry readings.

The D435 Realsense cameras are also not error free. We noticed that their output serial stream hangs unexpectedly and the only known solution is to unplug and plug them back. After some reading, I found that this is a hardware bug addressed by Intel as unfixable on this device. Therefore, I recommended Nick to not rely on these devices for critical tasks, such as the DARPA subterranean challenge.

When collecting data for hill climbing, I noticed a large variance (noise) in the IMU readings. Since our neural networks consider only the current inputs to provide instantaneous output velocity, noise in such an input would hinder the training process as the optimizer struggles to find correlation. I noticed that this high variance was due to the small slope of the hills and the increase in the percentage error due to that. To avoid this, I suggested fixing the IMU at an elevated angle to the horizontal on the robot.

We also faced issues with Jetson TX2, since the orbitty development board which we had to use (due to its small form factor) had its own modified Linux kernel. Many drivers were not directly supported for this and we had to modify the kernel to install certain drivers.

2.4 Designing and Implementing an Efficient End-toEnd Pipeline for Machine Learning in Robotics

2.4.1 Need for a standard ML pipeline in DATA61

Robotics and Autonomous Systems Group of CSIRO's DATA61 consists of world class engineers who have specialized in field robotics. Over their multiple years of experience in research and engineering, each of them have formulated an efficient workflow for themselves. Workflow is the general procedure where they collect data, process them in specialized software or write programs for custom processing, build software that can run real time on an existing robot to process the environment and make decisions, debugging their products or code and so on.

Today, with the rise of machine learning in several fields, it is a valuable tool for a field roboticist since it is fundamentally suitable to address the common problem in field robotics: processing complicated real world through noisy sensors and making high level decisions. Many of these leading roboticists have realized that and have already started using machine learning in their projects. However, in DATA61, so far, there is no unified framework that acts as a common ground to the different researchers working in different projects. Also, many of them currently use their local computers (powered by GPUs) for machine learning tasks out of habit and convenience, rather than utilizing the supercomputing resources available at CSIRO. And some of them who have figured out the procedure of using a supercomputer follow certain practices which are fine in local computers, but inefficent when done on supercomputers.

When Nicolas Hudson (our supervisor) took position in DATA61, he set out with the task of streamlining the machine learning (and other) procedures in DATA61 into unified frameworks, such that different researchers can work together, share their knowledge on a common ground and follow practices which are efficient and that maximize the utilization of the resources at disposal. One of the main motives behind such a task of streamlining is the DARPA Subterranean Competition (see: section 1.8) on which different groups of DATA61 will be working together for the next few years.

2.4.2 Pipeline: An Overview

Therefore, we were requested to create an end to end pipeline for machine learning. This pipeline should be compliant with the current general workflow of most roboticists in RAG and should be efficient and scalable. We were asked to use the latest versions of software and the most general hardware controllers so that our pipeline can be effectively generalized. It is end-to-end in the sense that includes best practices in collecting and storing data, building models and training them with large volumes of data and finally deploying the trained models on power-constrained high level controllers which are to be mounted on robots.

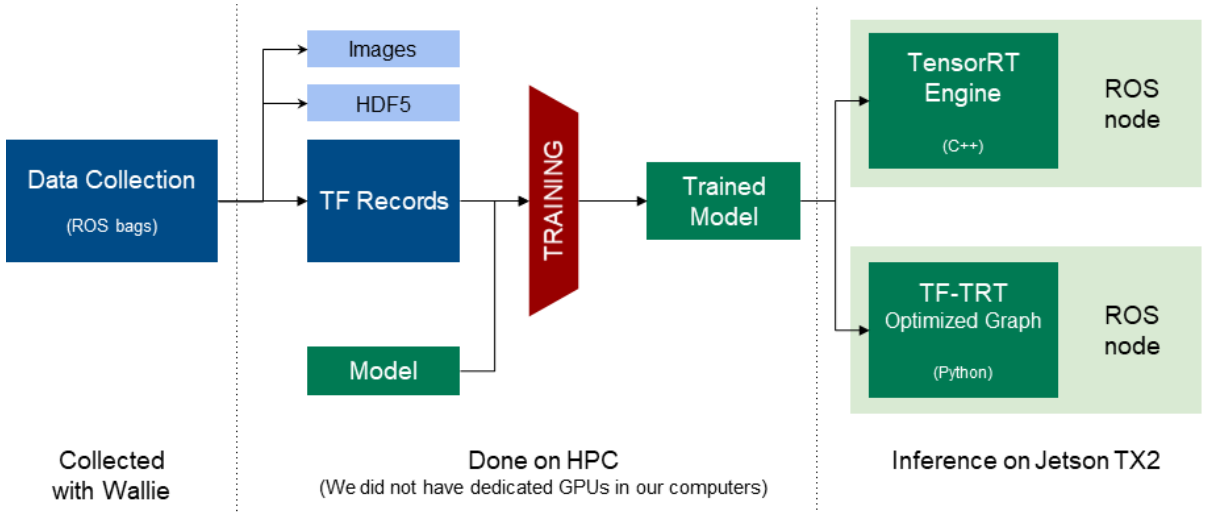

Fig. 2.13. End-to-End Pipeline we built

We were asked to do the two projects: Trailnet and Hillnet using this pipeline while perfecting the pipeline itself as a proof of concept. We were also asked to document our processes extensively in the internal wiki pages of CSIRO along with reusable code. In addition we were requested to create simple code snippets that demonstrate the key points of each part of the pipeline and finally explain and demonstrate our pipeline to the fellow scientists at a Robotics Reading Group meeting.

2.4.3 Collecting and Storing Data

In field robotics research, ROS (Robot Operating System) is used as the common platform or middleware to connect sensors and actuators of a robot. ROS packages are available for most of the professional high-end sensors and data from such sensors can be recorded in 'ROSbags'. ROSbag (.bag) is a file where all the activities of a ROS session was recorded in time-ordered format. They can be generated by a built in ros package. The user is able to record any session into a rosbag and play them later. ROS and ROSbags are used as the de-facto standard by the roboticists at CSIRO to collect data for analysis.

These ROSbags are unnecessarily huge in size and time-ordered. This poses difficulty in directly using them as a data storage format for machine learning. Due to their size, it is not possible to fit them into memory and due to their structure, it is not possible to perform multiple parallel reads from the same file in disk to accelerate the training process. Being time-ordered is an advantage for training RCNNs, but for training simple CNNs, we need the data to be completely shuffled for faster convergence when training.

Storing as Images

Therefore, the first solution is to generate a dataset of thousands of individual images (JPG, PNG) sorted into folders representing various classes. Such a dataset of images can be thoroughly shuffled easily, by simply shuffling a string-list of their filenames. Loading all the images into the memory at once is impossible even for 40,000 images. Instead, one can write scripts to read and queue images from disk, but it is cumbersome and error-prone.

In addition to that, storing thousands of images in the HPC (High Performance Computing or Supercomputer) is highly discouraged. When it is done, the performance gains of processing the image faster is offset by the overhead delay of reading and processing the image headers of each JPG or PNG image and results in a slowdown. Also, storing a large 'number of files' in network drives puts a strain on the existing file management systems. I had my access revoked multiple times for accidentally storing a large number of files in my personal folders of the supercomputer.

Storing as HDF files

Due to these issues, I spent a few days experimenting with serialized file formats such as HDF5. They are ideal in the way that they are read from disk directly (not from memory) and queuing operations can be performed with little difficulty to boost the reading speed. However, a major problem with HDF files is the inability to render themselves to a shuffle operation. Since the data is serialized, to shuffle an existing HDF5 dataset, the entire dataset has to be read into the memory, which is impossible. Then the only solution is to shuffle before writing a dataset. But given that ROSbags can be read only in a time-ordered manner, this is also not possible.

2.4.4 Storing as TF Records

Then I turned to TF records, a dataset format recommended by the TensorFlow framework, which is the standard framework for ML in CSIRO RAG. I found TF records to be ideal for our purpose due to several reasons. Firstly, TFRecord is a serialized file format, that supports reading from disk. Also, the data stored in TFRecords can be bundled, like bundling attributes into an object in Object Oriented Programming. That is, the input image and the corresponding label, IMU data, velocity can all be bundled into a single TFexample, stored and retrieved for training. In addition to that, tensorflow offers multiple operations that can be performed on tfrecords to shuffle them, queue them and read them effectively without reading the entire file. In a nutshell, tfrecords were designed to be used in supercomputers and to solve the problems with other methods.

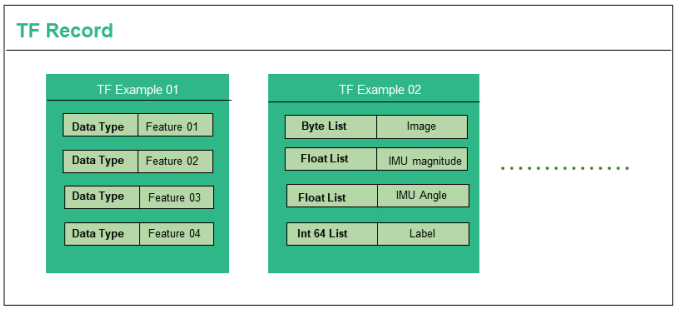

Fig. 2.14. Structure of a TF Record

A TFRecord dataset can be composed of thousands of tfexamples (tfexample is conceptually similar to an object in OOP, in reality is a protobuf defined in tensorflow). Each TFexample is a bundle of multiple features (a feature is conceptually similar to an attribute in OOP) that describe the TFexample. A feature can be of only one of three datatypes as allowed by tensorflow: ByteList, FloatList and Int64List. For efficient shuffling, one needs to split the dataset into multiple tfrecords. That is, as the raw file (such as ROSbag) is traversed in time-order, first 500 tfexamples should be written into one record in time-ordered manner, second 500 into next and so on. This is also because, having millions of examples inside one tfrecord slows down the read operation and the recommended optimum size of the tfrecord is around 100 MB.

2.4.5 Training on Supercomputers

The set of tfrecords written that way can be read in the following manner. First, the list of tfrecord files in a dataset is shuffled. Then we start reading from all the tfrecords in parallel. Here, we can split the dataset into training and validation randomly. This read operation can be queued easily (prefetching). This pipeline can then be effectively shuffled (by shuffling files inside a prefetch buffer from each file and then interleaving them across the read heads from multiple files), repeated and decoded using a function we specify (to convert the tf datatypes to the datatypes we use, perform operations such as BGR to RGB conversion in images, changing dimensionality...etc), then batched and released as the input for training a model.

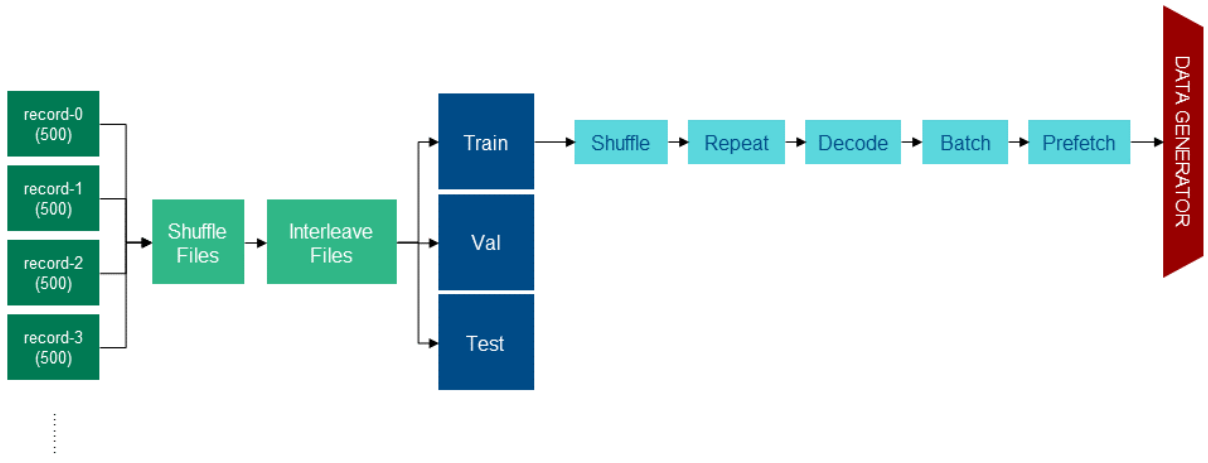

Fig. 2.15. Data Input Pipeline with TFRecords for training

A key insight here is that each of the above mentioned operations: shuffle, interleave, decode, batch, prefetch, train-val split are all implemented as tensorflow ops (operations). That is, tensorflow follows a dataflow programming paradigm (like verilog HDL). We specify the operations that should be performed on data and the connections between those operations (a tensorflow graph) through our python code, but these operations are not performed as control flows through those respective lines in the code. Instead an abstract graph is formed as control flows through those statement. Finally, when we start a tensorflow session and run it (sess.run()), the graph gets executed and data starts flowing through it in parallel (much like verilog HDL). The above mentioned operations are also performed this way. We specify the transformations that should be done on the dataset in an abstract manner. Tensorflow forms a graph and performs those transformations in an efficient parallalized way that maximizes the use of resources at disposal.

TFRecords also yield well into the training of models in tensorflow and tensorflow-keras. Both APIs support generators, a type of python functions that mimic an iterator (like a list) to provide their input, output data pair for training. Such a generator can be created to read a set of tfrecords using the above ops and yield one pair at a time as output. This blends seamlessly into the training procedure.

Unfortunately tfrecords are not very intuitive to use. Not many roboticists are not familiar with the unique datatypes and the dataflow programming paradigm of tensorflow. Advocating the use of tfrecords as the common dataset storage format and showing others how they can be included in their pre-established custom workflows was a key part in the development of the pipeline. I helped many of my coworkers by creating tools to visualize, write and read tfrecords. I also documented these procedures extensively for the future reference in CSIRO's wiki pages.

2.4.6 TensorRT: Deployment on a low power device

Machine learning frameworks such as TensorFlow and Caffe are built to be optimized for training a neural network model. They use high precision datatypes and typically utilize a large amount of memory, GPU and processing resources even to perform inference using a trained model. In addition, neural networks are typically built and trained by data scientists without a software engineering background who prefer high level languages like python and generally are not focused on writing fast, efficient code to be run on resource-limited devices. This poses a key problem in machine learning: these models are generally not portable to devices that have limited memory, processing power and use little power.

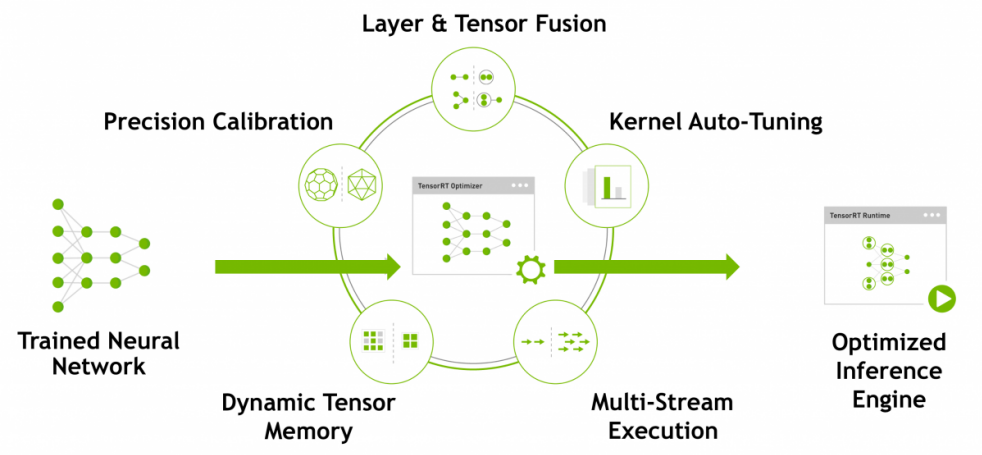

Fig. 2.16. TensorRT in a nutshell

To address this problem, NVIDIA released TensorRT as their own library. TensorRT is NVIDIA's programmable inference accelerator, that can optimize any supported model for inference on devices with limited resources. When a model is optimized, first the full-precision float32 datatype is converted to half-precision float16. Then, tensorRT traverses the tensorflow graph and finds subgraphs that can be optimized. That is, a convolution operation followed by activation and max-pooling operations is combined into a single tensorRT operation that can be executed in very few clock cycles. These tensorRT operations are then assigned to various parts of the target device: the CPU, GPU and special DLA (Deep Learning Accelerator) chips and queued for parallel operation.

Therefore, by converting a tensorflow graph (optimized for training) into tensorRT graph (optimized for inference), one can obtain a model that can have very low latency during inference, used very little memory without a significant loss of accuracy. This model can then be deployed for inference on NVIDIA's embedded hardware such as Jetson TX2.

TensorRT is released both as a C++ API and a Python API. However, Jetson TX2 only supports the C++ API of TensorRT. The following figure shows the procedure I followed to convert a tensorflow graph into a tensorRT graph for inference. After conversion, I created C++ ROS nodes for Trailnet and Hillnet to accept messages from sensors, process them with the model and output velocity commands for the motor controller. The C++ API is extremely fast: it could process a 320 x 180 x 3 image through a resnet with 20 convolution layers within 5 ms, allowing a processing rate upto 200 frames per second.

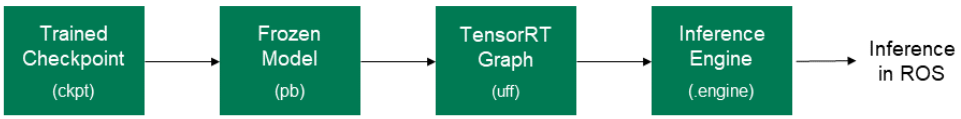

Fig. 2.17. Deployment Pipeline: C++

However, not all models can be converted into tensorRT graphs for inference. That is, only a small subset of tensorflow's operations are supported by tensorRT for inference. For example, batch normalization, a common operation used in many networks, was not supported in tensorRT 3.0. Therefore, in order for our pipeline to be generalized, I also looked into a way to convert optimize any tensorflow graph for inference. Google and NVIDIA joined hands to solve this issue and have provided tensorflow-tensorRT (TF-TRT) as a part of tensorflow framework since 2017. TF-TRT can analyse any tensorflow graph (any neural network) and choose the subgraphs (operations) that can be accelerated by tensorRT. The unsupported operations are then perform through tensorflow in an unoptimized manner. Result is a model that is slower than pure tensorRT and faster than pure tensorflow during inference. Following is the workflow for this.

Fig. 2.18. Deployment Pipeline: Python

I created a python ROS node also to accept messages from sensors, process them with the model and output velocity commands for the motor controller. The same network had a latency of 20 ms in the python node. That is about 50 frames per second. The detailed procedure and my scripts used for these conversions were documented thoroughly in CSIRO's wiki pages.

2.4.7 Problems Faced and Solutions

When I began building the pipeline, the major problem I faced was that none of my coworkers were familiar with tfrecords. I had questions about the best practices in shuffling and the order of operations, which adequate answers were not provided in the documentation. Therefore, I started to experiment with a simple dataset of integers from 1 to 50,000. I brainstormed my ideas and tried different configurations of writing, reading and shuffling and figured out the best way of doing it efficiently.

Then, there was a problem when we found ROS was not installed on the supercomputer as a package and we are not allowed to install new packages there. This meant that we cannot play the ROSbags to extract them in the supercomputer. We tried to extract the ROSbags into tfrecords and then pushing the tfrecords into the supercomputer, but it was a cumbersome task and the storage on our local machines were not sufficient for that. Our coworker Micheal gave me a docker container image he has built for a similar task. Docker container is a virtual environment which one could build with necessary packages and run on a computer as an abstract layer. I spent several days building and rebuilding a docker based on that, to include rosbag, opencv and cvbridge python packages and used it to convert ROSbags to tfrecords in the supercomputer, without running ROS.

When working with the C++ interface of tensorRT, I worked overnight at office to build the first draft of the ROS node: the pipeline to process the image and perform inference and faced with a TensorRT error: "dimensions don't match for the inputs of add layer". I spent a week debugging this issue: raised an issue in NVIDIA's developer forum, rechecking dimensions of my network, building a residual block with pure tensorflow ops and the problem persisted. I decided it was a bug. After a week they confirmed it indeed was a bug and asked me to update our software to the latest version released few weeks prior to that. We did, and hence solved that issue.

2.5 Hillnet: An Experimental Attempt at Utilizing ML for Hill Climbing

After successfully demonstrating indoor-Trailnet built from scratch, trained and deployed with my pipeline, our supervisor Nick described about his idea of experimenting with a algorithm to climb hills while avoiding obstacles using computer vision. We were asked to come up with a system where two different types of inputs are merged: scaler inputs representing the direction of the slope and the RGB image input from a camera to output a velocity command to control the motor controller. Trailnet, which itself was based on resnet-18 was chosen to be modified to build such a neural network.

2.5.1 Preprocessing IMU and Velocity Data

The LORD Microstrain IMU sends its data through serial to its ROS package, which publishes the IMU readings in two types: as a quarternion and set of cartesian x,y,z vector components of perceived acceleration. I decided to use the vector components to calculate the magnitude and direction of the gravity vector projected on the horizontal plane of the robot. Direction $\theta \in [-\pi, \pi]$ was measured with respect to the heading direction and then converted to $[0, 1]$ range using the sigmoid function to match the range of the other normalized inputs. I chose to output the angular velocity as a float $\in [0, 1]$ using a sigmoid output node (in classification approach) and scale it to the necessary angular Velocity.

2.5.2 Data Collection

As per the instructions of our supervisor, Uvindu collected data in the slopes around CSIRO campus during the last few weeks of the internship. He placed the robot on the bottom of the hill and drove the robot following the steepest ascent. When he faced an obstacle, he went around the obstacle and continued to follow the steepest ascent. Image streams from the camera, IMU data and the odometry reading from the motor encoders were recorded in ROSbags.

Fig. 2.19. Data Collection to Train Hillnet

2.5.3 Merging Scaler and Image Inputs

For this task, it was necessary to decide how the two types of inputs: scaler and image are combined in the neural network. Our supervisor proposed a method, where the IMU values are concatenated to the flattened output of the convolution layers, followed by few fully connected layers. The inspiration for this idea comes from the insight that deeper layers of a deep neural network identify higher level features and therefore the flattened average pooling output of the CNN should be containing information about by how much should the robot turn to avoid the obstacle. Therefore, concatenating the IMU inputs, which also tell by how much the robot should turn to follow the hill and following it with few fully connected layers might result in the network learning an OR operation between two inputs.

However, a research paper on merging these kind of inputs proposed an alternative method, where the scaler inputs are broadcasted into a matrix with the right size, processed by few fully connected layers and added elementwise to the output of an intermediate layer in the CNN. The insight behind this is the fact that the output of the fully connected layers might act like a mask

on the image, effectively clouding and directing the decision process of the CNN. After some consideration and experimentation, we decided to follow our supervisor's method since it was more suitable for our task.

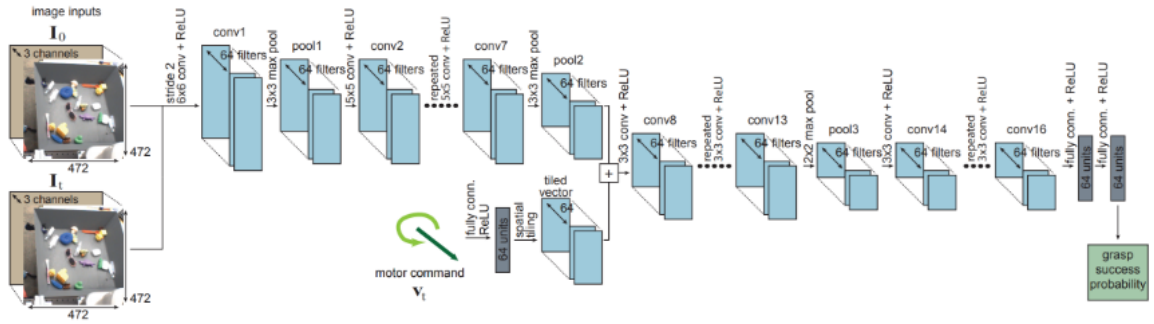

Fig. 2.20. Merging by Broadcast and Add Elementwise as a Mask

2.5.4 Regression Approach

Next we brainstormed on how to post-process the output of the network to steer the robot. During the discussion, our supervisor first suggested to use the regression approach. That is, to build a network with a single output node that has no activation function, so the output value can be directly used as the angular velocity command to steer the robot. This is quite straightforward to build, train and test. The neural network can be trained with IMU and Image data and output as the angular velocity from odometry reading when driven by remote control during data collection.

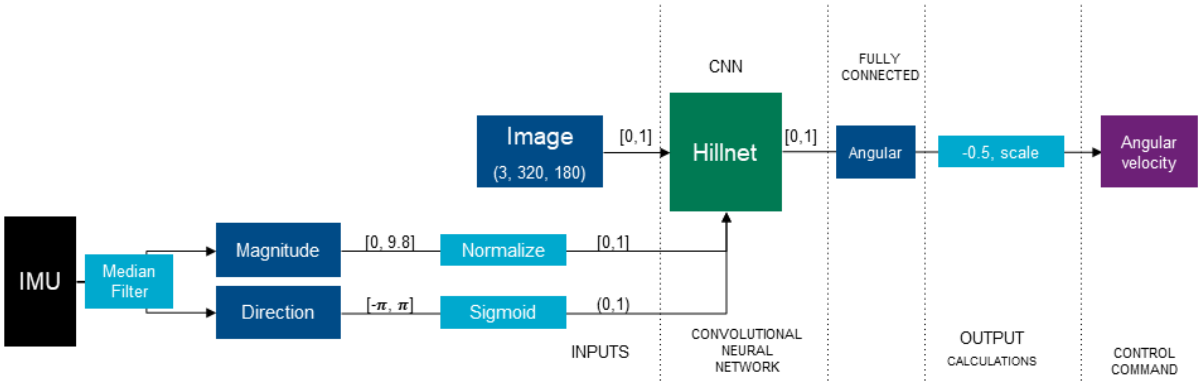

Fig. 2.21. Hillnet Regression Architecture

2.5.5 Classification Approach

During that discussion, I suggested trying a classification approach similar to that of trailnet. First advantage was the ability to fine tune the output by adjusting the constants. With regression, either it works or not. But with classification, we could fine tune it to work as we wish. Then, the effect of human error introduced in collecting data can be made insignificant by quantizing the angular velocity into classes: left, center and right.

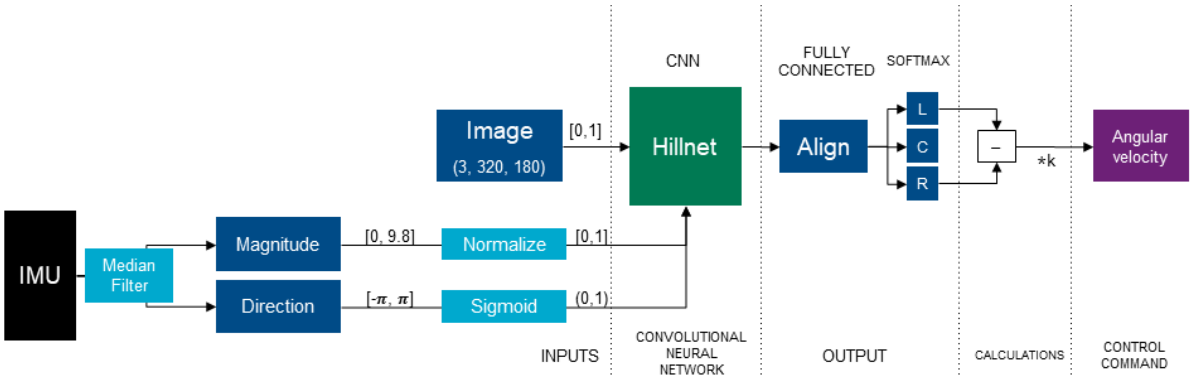

Fig. 2.22. Hillnet Classification Architecture

Next, I proposed a scheme of augmenting data to effectively triple the amount of collected training data. That is, first we record image streams from all three cameras. Then, in the input pipeline, I made changes to add +30°and -30°to the IMU angle and use it as datapoints along with the image streams from left and right cameras respectively. This would train the network to NOT turn away from the steepest slope, unless an obstacle is encountered, I argued. Our supervisor accepted the idea and asked me to work on both classification and regression in parallel to compare their merits.

2.5.6 Problems Faced and Solutions

First problem we faced was the human errors in data collection. Since the hills had a small slope, Uvindu had difficulty in visually identifying the highest-slope direction and steering the robot towards it. When we analysed the collected data using the visualization techniques, we found there was a steady state error by up to 5°-10°. We tried collecting more data with a couple of days remaining to end the internship.

Another anomaly I observed from the visualization was the fact the IMU input had a high variance (noise). This was due to facts the slope was gentle (high percentage error) and the spring loaded suspension system of Wallie was causing it to wobble, introducing low frequency vibrations. This was critical, since our neural network does not remember nor correlate with the past inputs, but considers only the current inputs to give instantaneous outputs. Such a noisy input will prevent the training process from converging to a global maxima. Hence I suggested mounting the IMU at an angle to reduce the percentage error. After experimenting with mean and median filters of various lengths, I implemented a median filter of length 50 to smoothen the noise.

On the last few days of the internship, while debugging the network, I accidentally noticed that our collected dataset had an unhealthy disparity. That is, only 0.15% data accounted for avoiding obstacles, while the rest 99.85% accounted for climbing following the steepest hill. This is a classic problem in data science where the neural network simply learns to suggest "go forward" and be right 99.85% of the time! I had discussed a similar potential problem with the supervisor at the beginning of the project, where I raised concerns that "we are not showing the robot which input-output combinations are wrong. we are showing only what is right". Our supervisor assured that "the robot will learn what is wrong, when you turn the robot to face the hill after avoiding an obstacle". However, the percentage of that kind of data was dwarfed by the "straight climbing" data in the dataset. I discussed this with Micheal and our supervisor and started implementing my idea of artificially boosting the frequency of "avoiding obstacle" data in the input pipeline.

However, we were running out of time by the end of the internship to try all possible ideas and experiment with all the possibilities. I stayed for multiple days overnight at office to try and finish as much as possible, but we couldn't try everything within that short time. Spending most of our time on building and debugging the robot platform could be one of the reasons for our time being limited for exploring new ideas with Hillnet. However, we realized that this is quite common in experimental field robotics, where scientists get to spend most of the time struggling with the hardware issues.

2.6 Life at CSIRO

CSIRO, being a world class research institute, thrives to create a stress-free work environment that encourages people to socialize and to boost creativity. There is a workplace culture in DATA61 to bring cakes (or any equivalent sweets) for all the coworkers if one gets married, arrive at CSIRO, leave CSIRO, has a birthday and so on. I shared Sri Lankan sweets (sent by my mother in mail) for my birthday and cakes with everyone for arriving at and leaving CSIRO.

2.6.1 Reading Groups and DATA61 Meetings

Every friday, a small meeting called "Robotics Reading Group" is hosted, where one scientist explains his current project to everyone who attends the meeting. This way, we get to know the latest technology that is being developed in different parts of DATA61 and new projects that are being started. This meeting also allows the scientist to be questioned, so that he can derive insights from the audience and refine his procedures in the future. Also, once in a fortnight our supervisor Nick holds a "ML Robotics" meeting for all engineers working on machine learning. There we discuss our current ideas and issues to help each other. In addition to these, there are monthly meetings with the entire CSIRO (branches from all over Australia join via video conferencing), where new developments are discussed. One such meeting had the lead scientist from NASA's Insight Mars Lander mission as the chief guest explaining the challenges faced in their mission and taking our questions on the matter.

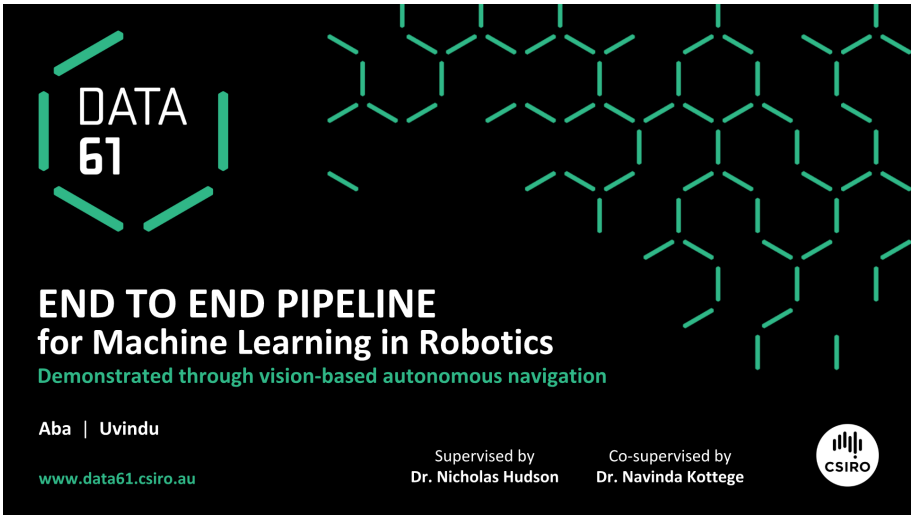

Fig. 2.23. Presenting the Pipeline in Robotics Reading Group

2.6.2 Presenting the Pipeline at Reading Group to the Scientists

Uvindu and I presented the pipeline to other scientists during one of the last Reading Group meetings of the year. It was well received, thanks to the support of our supervisor who encouraged others to use our pipeline in their workflow. Many asked questions and were convinced of the merits of such a unified framework. I had a few scientists reaching out to me on the following days asking me to develop visualization tools to complement the pipeline.

2.6.3 DATA61 Live Event

DATA61 Live is an event held annually to showcase the science and technology innovations of DATA61 from all over Australia. In 2018, it was held in Brisbane, in our city. The theme was: 'Adapting to Disruption'. We signed up as volunteers and apart from volunteering, we had a chance to attend many lectures, talks and forums. It was a great experience.

Fig. 2.24. DATA61 LIVE Event

3 Conclusion

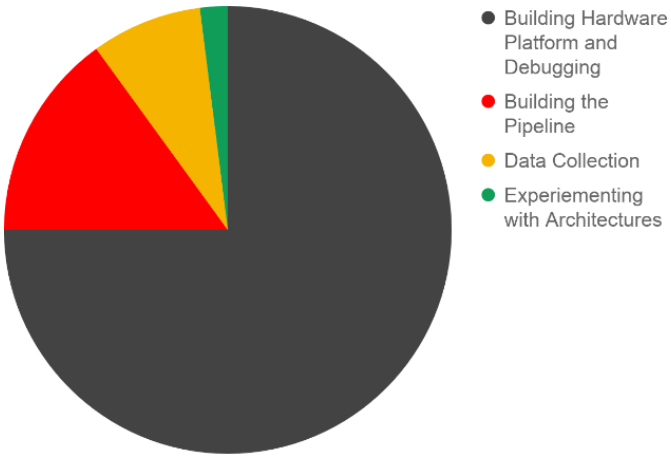

Fig. 3.1. Overview of our time spent

References

[1] Nvidia jetson tx2 module. https://developer.nvidia.com/embedded/buy/jetson-tx2. Accessed: 2018-06-28.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. 2017.