Semantic Segmentation for Terrain Roughness Estimation Using Data Autolabeled with a Custom Roughness Metric

Shastri Ram

CMU-RI-TR-18-34

July 14, 2018

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
George Kantor
Kris Kitani
Sankalp Arora

Submitted in partial fulfillment of the requirements for the degree of Masters of Science in Robotics.

Copyright © 2018 Shastri Ram

Abstract

Traditional methods for off-road terrain estimation use two approaches. The first approach estimates terrain as it is traversed, while the second approach estimates terrain before it is traversed. The first approach accurately measures the terrain because it directly captures the effects of the terrain on the vehicle. However this approach cannot predict the terrain ahead of the vehicle. On the other hand, the second approach is able to predict the terrain ahead of the vehicle, but since it relies on indirect terrain features, this approach cannot accurately measure the effects of the terrain on the vehicle. This thesis proposes to combine the best of both approaches by developing two metrics for classifying roughness, an inertial measurement unit based roughness metric and a point to plane distance roughness metric. Images autolabeled with these roughness metrics are then trained with deep learning network designed for semantic segmentation. The result is a system which was trained without the subjectivity of a human, and with the ability to predict the vehicle's response to the future terrain at a higher degree of accuracy than traditional systems.

Acknowledgments

Firstly I would like to thank God for blessing me with this prestigious opportunity to pursue a Masters of Robotics at Carnegie Mellon University, and for giving me the strength, energy and fortitude to complete my courses, my research and this thesis. I am greatly thankful for my core support system, my mom, Yasmin Ram, dad, Daniel Ram, brother, Vinai Ram and my girlfriend, Thalia Singh. My parents and my brother were always there to offer kind words of encouragement and keep me focused on my goal. Though my girlfriend is in a different country, she kept me company every day and night ensuring that I never felt alone and always felt her love. I am indebted to my advisor Professor George Kantor whose wisdom and guidance kept me on track and helped me overcome the obstacles I faced. I am also thankful to my co-advisor Professor Kris Kitani who also helped steer me in the right direction throughout my thesis and research. I am thankful for the Yamaha Motor Corporation for allowing me to use their offroad robotic platform developed at the Field Robotics Center. I also received counsel from other members of the Robotics Institute community, but in particular I would like to thank Srinivasan Vijayarangan, Dong-Ki Kim, Bill Drozd, Sankalp Arora and Daniel Maturana for the great assistance they offered.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The world is experiencing a robotics and autonomy revolution. These changes are occurring both on the hardware and software fields. Improvements in manufacturing processes, innovation in design, miniaturization of components, and use of new materials are driving down the cost of components such as vision sensors, motors, actuators and processors. Every year either new algorithms are being developed or being improved upon, that challenge and even surpass the previous year's state of the art result.

Of the multitude of robotics research and development presently occurring, autonomous driving stands out from the rest. There could be many reasons for this. Firstly, autonomous vehicles receive the most attention. The reason for this is due to the physical and visual nature of autonomous vehicles. While there are a multitude of devices such as smart phones, home assistants and even refrigerators running complex machine learning algorithms, they do not receive as much attention because these algorithms are running in the background, away from the view of the public eye.

On the other hand the public gets to see and even use various aspects of vehicular autonomy. Many car manufacturers have some level of autonomy as standard on their vehicles. Examples of this range from self parking features, lane keeping abilities and of course autonomous driving under certain conditions. Furthermore companies such as Uber and Waymo have autonomous vehicles on the road which the public can use.

Another reason why autonomous vehicles are being developed at such a rate could be due to the increased benefits that vehicular autonomy provides. Human beings can only process a certain limited amount of information at a time. However a computer can process much more information and at a faster rate. A vehicle outfitted with many sensors and a powerful computer can monitor its environment better than a human. Google's and Tesla's self driving cars have logged millions of miles with only a few incidents of crashes or fatalities. The equivalent number for human drivers is astoundingly high. As such autonomous vehicles can improve the safety of everyday driving. Autonomous vehicles can improve the efficiency of the transportation system. For example one can envision a city in the not too distant future, where all the vehicles are autonomous. Such a city would have no traffic, less accidents and so productivity and efficiency

can increase.

From a researcher's point of view, vehicular autonomy is a very interesting problem. It is the perfect marriage of all different fields of robotics such as computer vision, path planning, state estimation, deep learning and controls that need to interact and work harmoniously with each other. What's even more appealing is that fact that these fields are being applied to solve a real world problem. This again alludes to the previous points before.

While most of the autonomous driving hype has been centered around urban driving, there is a fast emerging sector of autonomy which is offroad autonomous driving. Mining and construction companies such as Caterpillar have autonomous trucks for haulage and mining applications as seen in [1] and [2]. These autonomous equipment improve productivity and safety in the mines and construction sites. On the more recreational side of things, Yamaha has been working on an autonomous offroad vehicle in collaboration with the Field Robotics Center [3].

In off-road autonomous navigation, the ability to understand the terrain is of utmost importance to enable a robot to successfully navigate. As opposed to urban navigation, during off-road navigation the terrain itself is a hazard. An incorrectly planned path can get the vehicle stuck in an area of soft ground, or a high speed over rough terrain can cause the vehicle to loose control and crash or overturn. As such, various methods over the years have been developed to estimate the terrain to enable better planning and adapt the controls of an autonomous vehicle in an outdoor off-road setting.

Traditional methods of off-road terrain estimation use some type of learning network to predict discrete classes of terrain such as short grass, tall grass, dirt and trees. Other methods of learning which can give more detailed, but still discrete classes, use on board sensors to measure the terrain roughness, and then predicts the terrain type. There also exist non-learning based approaches which use geometry and physics based equations along with inputs from sensors which give a measure of the terrain roughness or traversability.

Within the learning realm, there have been varying approaches to classifying terrain. These approaches can be split into two categories, vision based and proprioceptive sensor based. Vision based systems use input from some vision sensor such as a camera or LIDAR and either use algorithms or learning methods to predict the terrain. Proprioceptive sensor methods use a sensor that is capable of measuring the effects of the terrain on the vehicle as it traverses the terrain. Such sensors include inertial measurement units, suspension stroke sensors or microphones. Again with this data, either some algorithm or learning method is applied to predict the roughness of the terrain.

Vision sensors have the benefit that they can predict the terrain before the robot drives on the actual terrain. However, they have the disadvantage that they cannot predict the underlying terrain profile. Proprioceptive sensors such as inertial measurement units (IMUs) or suspension stroke sensors have the benefit that they can predict the underlying terrain, but unfortunately they are only able to do so when the robot is already actually driving on the terrain. What if we can combine the benefits of these two types of sensors? This is the first question that this thesis aims to explore.

We approach this problem by developing roughness metrics derived from the robot's sensors, in particular the vision sensor and the inertial measurement unit. However instead of the usual labels such a grass, trees, rough trail or smooth trail, we seek to develop a roughness metric that describes the nature of the terrain itself to a finer degree than the broad type of labels previously

mentioned. By having labels which characterize the roughness of the underlying terrain, we can project these labels onto the image and so fuse the information from both sensors.

As with any learning system, in order to train the system, there must be a labeled dataset. The bane of any researcher in deep learning is the task of hand labeling images. Usually a researcher will sit for hours on end labeling images, or spend money hiring services such as Amazon Mechanical Turk [4] or people such as undergrads to hand label the data. Apart from the time and/or money spent hand labeling datasets, there is also the issue of human subjectivity. For example when considering outdoor terrain, what one person may perceive as smooth, another person may think of as rough.

A possible solution to overcome this problem is autolabeling. An autolabeling system takes the raw data as input and automatically labels the data according to some metric. The benefits of this include saving resources such as time and money, as well as removing the human subjectivity from the equation. This is the other question that this thesis seeks to delve into. We develop a system that autolabels images with the roughness metrics previously described, thus removing human subjectivity from the system.

In order to do this, the path the robot drove would be calculated through dead reckoning. Then for each image, the path traveled from that point of time the image was taken, would be projected onto the image. The calculated roughness metric would then be superimposed onto the path that the robot drove, forming the ground truth data. These autolabeled ground truths and images would be used to train a deep learning network that would be able to predict the roughness of off-road terrain given an input image.

A network which is capable of predicting the roughness of the underlying terrain to a higher level of detail than traditional systems can be beneficial. For example a path planning system can improve its paths by minimizing the jerk to the robot and its passengers. It could also enable a controls system to adjust its controls so that the vehicle drives more slowly over the rougher areas reducing the possibility of an accident.

The rest of this thesis illustrates the methodology, algorithm, data, experiments and results of this entire process in fine detail.

1.2 The Robotic Platform

The robotic platform used for data collection and research for this thesis was the Yamaha Autonomous System Off Road Vehichle, shown in Figure 1.1, officially known as the Yamaha AS-X1, [3]. It was developed as a joint collaboration between Yamaha Motors and the Robotics Institute Field Robotics Center from May 2015 to August 2017.

While the robot has quite a bit of sensors, the two that play the most integral role in this thesis are the Multisense S21 and the Xsens MTi-20, which are depicted in Figure 1.2. The Multisense S21 is a stereo camera designed by Carnegie Robotics, [5]. The Multisense was configured to run at 10Hz and outputs a 1024x544 pixel RGB image and the corresponding depth map, calculated by the on-board processor, as ROS topics. The depth image is the same size as the raw image, but each (x,y) location of the depth image holds the calculated distance the corresponding pixel in the raw image is from the camera. The Xense MTi-G-700, [6], is a inertial measurement unit which measures important data such as magnetic field, angular velocity, linear velocity, linear

Figure 1.1: Yamaha autonomous systems off-road vehicle, Yamaha AS-X1.

acceleration, roll, pitch and yaw. This data is also output as ROS topics at a rate of 200Hz. Of particular interest for this thesis are the angular velocity, linear velocity and angular acceleration.

The Yamaha AS-X1 was the primary data collection device and data was collected during August 2017. The robot had the capability to be commanded to drive at a specified speed, while being manually steered by a safety driver. The control system of the robot would maintain the specified speed until the command is canceled. In order to collect data, the robot was commanded to drive at speeds of 10, 20, 30 and 40 kilometers per hour (kmph) on different parts of the Gascola test site. All of the data from sensors were logged as ROS bag files.

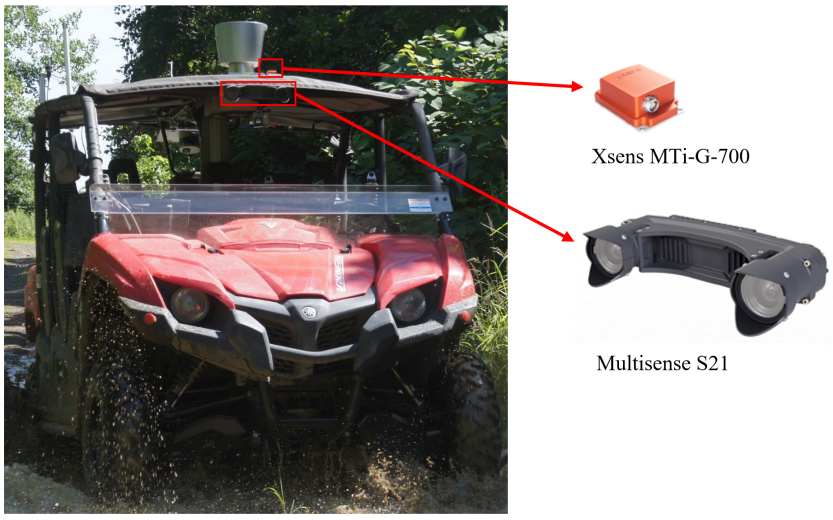


Figure 1.2: Location of the Multisense S21 and Xsens MTi-G-700.

Chapter 2

Related Work

While there exist many examples of work which estimate terrain, they can be divided into two main categories. The first category, methods that estimate terrain as the vehicle drives over it, accurately measures the terrain because these approaches directly capture the effects of the terrain on the vehicle. However, since the vehicle must be traversing the terrain and cannot see the future terrain, these methods are unable to predict the terrain ahead of the vehicle. Methods such as these use sensors like inertial measurement units (IMU), suspension stroke sensors and acoustic based sensors. The second category of work, methods that estimate terrain before the vehicles drive over it, are predictive because they look at the area ahead of the vehicle and make an estimation of the terrain. However, they are unable to accurately measure the effects of the terrain on the vehicle because they rely on indirect terrain features rather than the direct response of the vehicle. Methods such as these use a vision sensor such as a camera or lidar.

This chapter describes the work done in these two domains. Finally it gives a summary of our contribution, which proposes to combine the two approaches by using learning to train a predictive network labeled with direct accurate training data.

2.1 Methods that Estimate Terrain as it is Traversed

One of the most notable areas where an IMU was used to determine characteristics of terrain is in [7]. Here, Staven et al use the z-axis acceleration data from an IMU as a measure of shock to the vehicle. However instead of using the raw data, g (9.81 ms^{-2}) was subtracted and then the data was filtered. The filtered data was used as a measure of shock applied to the vehicle due to the terrain. They found that there was a near linear correlation between the filtered data and the velocity of the vehicle. As such, a velocity plan with learnable parameters, was developed that took into account the current speed of the robot and shock applied to the vehicle. The parameters of this were derived by training with a supervised learning system. This method proved to work quite well and the researchers credit their success in the DARPA Grand Challenge to the algorithm. We use this method of filtering the IMU data to develop an IMU based roughness metric.

Data from an IMU was once again used in [8], but instead of estimating roughness, it was used to classify six different types of terrain. An experimental cart equipped with the IMU was

driven on different types of terrain. From one second segments of the raw z-axis acceleration data, feature representations were extracted, using three methods, log power spectral density, 128 point Fast Fourier Transform and a custom designed feature representation. Each of these types of feature representations were used to train a support vector machine (SVM) to predict the terrain type. It was found that their custom designed feature greatly outperformed the other types of feature representations considered. They note that if this method were to be applied to a vehicle with a suspension system, they recommend to mount the IMU on the axis of the wheel or some other part of the vehicle not affected by the suspension.

Weiss et al continue their work in [9]. Here an IMU is mounted on an outdoor robot , which is driven on different terrain at three different speeds. Features were extracted from the acceleration data and are used to train different learning methods such as a probabilistic neural network (PNN), Brook's Method, SVM, k-nearest neighbors classifier, decision trees and a naive Bayes classifier. Each learning method was trained with each set of data collected at different speeds, and then with all the data. They found that the SVM produced the best results. However they found that training separate learning methods for data at each different speed collected resulted in better performance than training using all the data from all the speeds. It is important to note that the maximum speed used was 0.6 ms^{-1} which is almost 20 times slower than the speeds driven when data was collected for this research.

A speed independent vibration algorithm was used in [10] to classify terrain with data from a suspension control arm mounted IMU. The vertical acceleration from the IMU is passed through an inverse tire and suspension model to estimate a profile of the terrain elevation profile as a function of time. Vehicle displacement is estimated by integrating the speed, which is then combined with terrain profile in time, to derive the terrain elevation profile as a function of displacement. Principal component analysis was used to extract features from the terrain profile which is then used to train a SVM. This algorithm involved the use of extensive knowledge of signals and frequency systems analysis to generate a terrain elevation profile. While this method seems complicated, we extrapolate the idea of modeling terrain profile as a measure of roughness.

More recently, some interesting work was done in [11] where acceleration data from all 3 axes of an IMU was used to classify 14 different classes of data. Besides normalization, no other processing was done to the raw data. The robot was driven over different terrain at a fixed speed and logged the IMU data. Each sample for each terrain corresponded to 1 second of recording time. Since the data was a temporal sequence, recurrent neural networks (RNN) and long short term memories (LSTM) were used to train the data. An improvement in classification accuracy from the previous state of the art was achieved.

One of the most interesting methods for classifying terrain was used in [12]. Here they attach a microphone near the wheel of a robot and record the sound made by the wheel when driving on different terrain. Spectrograms from the audo signals, augmented with white Gaussian noise and varying signal to noise ratios were used to train novel deep convolutional neural network architecture to learn features from the spectrograms, in order to classify nine different types of terrain.

2.2 Methods that Estimate Terrain Before it is Traversed

The work presented so far, with the exception of [7], has mainly focused on extracting features from an IMU to use for training a terrain classifier. However, research into the use of different type of sensors for estimating terrain has also been conducted. In [13] three novel force prediction models were developed which used the height of a point from a 3D point cloud to predict the reactive force exerted by terrain on the vehicle. This force was used to calculate the maximum allowable traversal velocity of the robot. The aim was to develop a less aggressive velocity profile to prevent equipment damage due to the transmitted force. It must be noted that this work was all done in simulation using simulated artificial data and never implemented on a real robot or tested with real world data.

An uneveness point descriptor, extracted from a 3D point cloud produced by a stereo camera, was developed in [14]. It which took into account both the inclination and roughness of the local surface. The descriptor was obtained by first finding the normals of all points in the point cloud by fitting a plane to the neighborhood of points around a query point. The sum of normals for each point within the neighborhood of the query point was found. The descriptor then became the average of the normals, along with each component of the summed normal. The descriptor was successful in illustrating the uneveness of terrain for both indoor and outdoor settings. While we are unable to use a full descriptor as a label for ground truth in deep learning, the method of fitting a plane to points from a point cloud, and finding the normal can be helpful in developing a profile of the terrain akin to [10].

While all of the work presented so far used one senor, that is, an IMU, stereo camera or microphone, work has been done in fusing the data from these sensors to predict terrain. In [15] Weiss et al fuse the results from a SVM trained with texture based features extracted from images, and another SVM trained with features extracted from IMU data. First the system predicts the terrain ahead of the robot with the vision classifier, then when the robot traverses the terrain, the output of the vibration classifier is fused with the vision classifier to get a better prediction. The fused results give a 10% increase in accuracy over the separate methods. While the success of this method gives credibility to the fusion of vision and IMU data, it is worthy to point out that Weiss et al are classifying different types of terrain, not specifically the roughness of the terrain.

In [16], a fusion of data from a camera and lidar were used to segment an image into four different classes of terrain: rough, smooth, vegetation and no information. The vegetation and no information classes were hand labeled but the rough and smooth labels were obtained by calculating the mean point to plane distance, as a measure for roughness, for points within a voxel. Their novel network contained a 2D and 3D stream which are fused by projecting the 3D features to the image space. Roughness and porosity features calculated from the 3D point cloud were used in the 3D stream while the RGB image was used in the 2D stream. The aim of the Kim et al was to ensure accurate terrain classification through different seasons of year.

To this end, two data sets, one during summer and the other during winter were collected. SegNet, a pixelwise semantic segmentation network was used as a benchmark to compare their results. While their results were a little better than SegNet for the summer dataset, they greatly outperformed SegNet in the winter dataset. However it is interesting to note that only the summer dataset was used for training. It could be possible that if SegNet were also trained with the winter dataset, it would achieve better results. However their result shows good generalization over

different seasons for a network trained from data of only one season.

This paper highlights some interesting points. Again we see the use of the plane fitting used to describe roughness. However, in contrast to other methods of terrain recognition which use classification for a patch of terrain, this method uses full image segmentation to classify all the terrain in front of a robot. Image segmentation first came to fame in [17], where Long et al show that a fully convolutional network trained end to end, pixels to pixels achieve high performance on segmentation tasks. Their approach involved convolutionalizing fully connected layers in a network and introducing skip connections to form high resolution feature maps.

SegNet, [18], is another semantic segmentation network. It uses an encoder-decoder structure with a novel upsampling layer which uses indicies from the max pooling layers to upsample the feature maps. This network outperformed other segmentation methods at the time the paper was written. These networks have been adopted to solving other segmentation tasks and have been shown to achieve great success as in [16]. By using pretrained weights, learned on one dataset, the network can achieve good success on a new completely dataset by fine tuning the weights through training with the new dataset. This transfer learning is described in [19], where they use activation features extracted from a deep convolutional neural network to generalize to novel tasks.

The work done in [20] and [21] are the closest work we have seen to the method we propose. In [20], Mou et al use a SVM terrain classifier trained on vibration data to label image patches with one of five different terrain classes. A Gaussian mixture model (GMM) approach was used to create a vision classifier, which was then trained on the labeled images. To give the system the ability to look ahead, features were extracted from the height of points obtained from a laser scanner, hand labeled, and then trained on a SVM to also classify terrain. They used a naive Bayes classifier to fuse the predictions of the vision and laser classifier. Similarly in [21], Brooks et al, use the prediction from a vibration trained SVM classifier to label terrain patch images with 3 classes of terrain. From the terrain patch images, features were extracted and then trained with another SVM classifier. It is important to note that in both these cases, the robot moved at a slow constant velocity.

2.3 Summary of Our Contribution

Thus far, most methods have extracted features from IMU data to predict classes of terrain such as grass, asphalt, dirt, gravel etc. Since we seek to label an image with a roughness metric derived from IMU data we are unable to use feature based representations. Furthermore, our aim is to gain a deeper understanding of the terrain, by accurately capturing the effect of the terrain on the vehicle, rather than just broad classes which are unable to provide such accuracy. Staven et al [7] have shown the potential of the IMU to estimate roughness through simply filtering the IMU data. We aim to follow a similar method used in [20] to label images with roughness classes created from the IMU data, but trained with a more recent method such as semantic segmentation.

The work presented here which use IMU data to predict terrain have all dealt with robots moving at low velocities. However our data consists of the robot driving at much higher and varying velocities. It is possible that a deep convolutional neural network may be able to learn weights to segment the image independent of velocity. If not we may have to look to terrain

profiling methods as in [10] or methods of incorporating velocity into training as in [9] or [16].

While [10] used the IMU data to estimate the terrain profile, they also extracted features to be trained to classify terrain. Furthermore in this work a suspension mounted IMU was used so that a direct measure of the terrain could be obtained. While we are unable to directly do this, we can use the concept of estimating the terrain profile to create a model of the terrain using a point cloud from a stereo camera similar to [16] and [14]. In [9], Weiss et al trained different SVMs for predicting the terrain at different velocities. This is one possible method to handle the varying velocities. Another method would be to include velocity as an input into the network similar to [16] which used multiple inputs, such as roughness and porosity, into the network to achieve good classification performance.

Chapter 3

The Roughness Metric

The roughness metric gives a measure of the traversability of the terrain. It must be such that rough areas have a higher value than the smoother areas. For this thesis, two roughness metrics were considered. The first one, the energy of the acceleration along the z-axis of the robot, was developed with inspiration from [7]. The second metric, point to plane distance, is a method of obtaining the terrain profile. It is a method used in previous work such as [14] and [16].

3.1 Roughness Metric Based on the Linear Acceleration along the Vertical Axis of the Robot

When driving a vehicle we often describe a road or terrain as rough when we experience a lot of vertical motion as the terrain or road is being traversed. When the wheels hit a bump, rocks or some other sort of rough terrain, the impact is transferred from the wheel to the suspension system. While the suspension system does dampen some of this shock and vibration, it is still transferred from the suspension system to the body of the vehicle and the vehicles moves in an up and down motion. This effect becomes more pronounced for off-road vehicles which have stiffer suspensions to deal with rough terrain. In other words, while the suspension system of an off-road vehicle is much more robust that a typical passenger vehicle, it gives a much rougher ride.

The Yamaha AS-X1 shown in 1.1 was originally designed as an off-road utility vehicle, and as such was made with a tough off-road suspension system. Therefore by quantifying the vertical motion of the vehicle as it traverses the terrain, a measure of roughness could be obtained. As shown in Figure 1.2 the Xsens inertial measurement unit is approximately located at the center of the robot. Among other things, it measures the linear acceleration along the z-axis of the robot, which is a perfect measure of the vertical motion of the robot as it traverses the terrain.

3.1.1 Derivation of the IMU Based Roughness Metric

The inertial measurement unit measures the accelerations at a rate of 200 Hz. In addition to measuring the accelerations felt by the vehicle along the z-axis, it measures g (9.81ms^{-2}), along that axis. Since we wanted to isolate the roughness felt along the axis, g was subtracted from the raw

readings. The initial idea was that the roughness metric could simply be the acceleration readings themselves. Figure 3.1 shows the raw IMU reading, and then the reading with g subtracted from it.

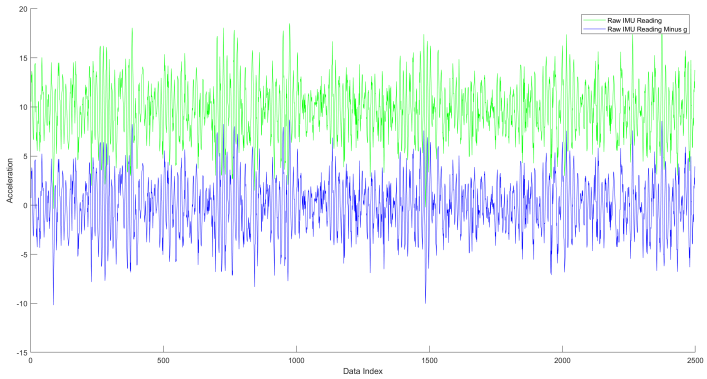


Figure 3.1: Raw IMU reading and raw IMU reading with g subtracted.

As can be seen in Figure 3.1 it is clear that the g subtracted IMU reading is too noisy, and some type of filtering must be done. As such the second idea was to filter the data by applying a simple moving average to the last 100 data points (0.5 seconds), after g was subtracted. This can be seen in Figure 3.2 as the blue line.

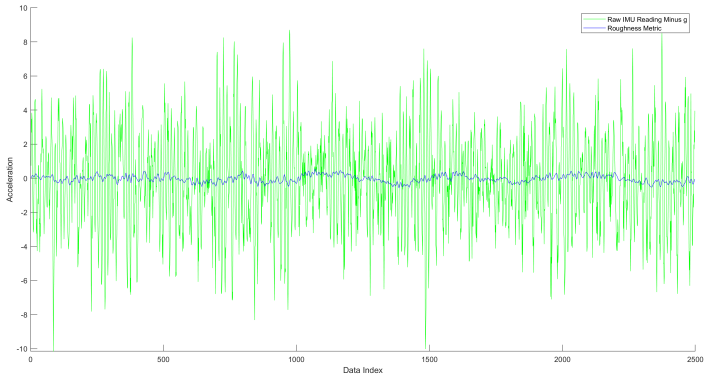


Figure 3.2: Roughness metric calculated from a simple moving average of the square of the last 0.5 seconds of data.

When the robot encounters rough terrain, it oscillates up and down due to the suspension system, and consequently there are both positive and negative accelerations. Since the simple moving average uses the raw reading with g subtracted, the negative values cancel out the positive ones and the resulting roughness metric does not differentiate the rough and smooth surfaces well. The simple moving average of the square of the signal would take into account both positive and negative accelerations, and solve this issue. Again, the simple moving average was taken over the last 100 data points (0.5 seconds) of the data. Figure 3.3 shows this result.

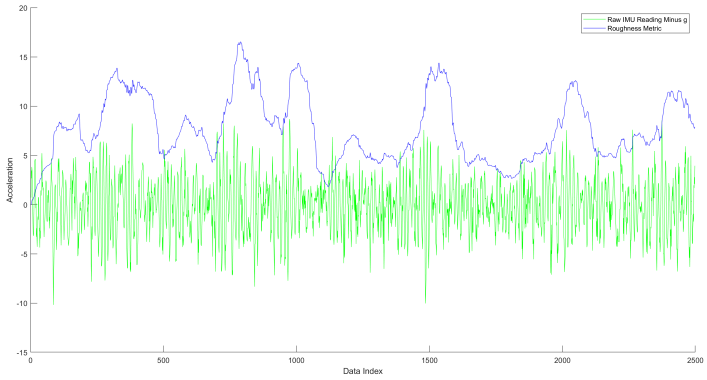


Figure 3.3: Roughness metric calculated from a simple moving average of the square of the last 0.5 seconds of data.

However, it can be seen that the roughness metric seems out of sync with the raw data. This was solved by using a simple moving average of the last 50 data points, or last 0.25 seconds of data instead as shown below. This method clearly differentiates between the rough and smooth terrain and seems more in sync with the raw data. Figure 3.4 shows this result.

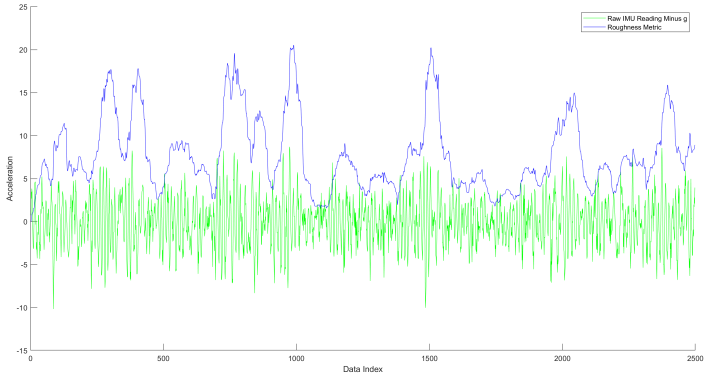


Figure 3.4: Roughness metric calculated from a simple moving average of the square of the last 0.25 seconds of data.

After labeling the terrain using this roughness metric, as described in chapter 4, the next step would be to train a convolutional neural network to learn how to associate the terrain with the roughness metric. Since this roughness metric is represented by continuous values, it would be necessary to train the convolutional neural network for regression. However, as described in 5.3.1, this did not produce the desired results, so the network was switched to be trained for classification, as discussed in 5.3.2. However in order to do so, the roughness metric would have to split into categories.

3.1.2 Categorizing the IMU Based Roughness Metric

It was first determined to categorize the continuous roughness metric by simply rounding the roughness metric to the nearest integer. When this was done, we found that there were values ranging from 0 to 160. It is not sensible to have 161 classes of roughness. This would make it very difficult for the network to learn to differentiate these values. We found that 0-35 gave a fine enough granularity between the roughness readings. A roughness of 35 represented very vigorous vertical motion and subsequently very rough terrain. Any roughness above 35 would be even worse. Therefore, it was not necessary to try to characterize roughness beyond 35 since it already represented very rough terrain. It would be more necessary to be able to classify smoother and less rough terrain. As such values which were 35 or greater were simply set to 35. Additionally, 36 classes were more feasible to be learned by a network. Figure 3.5 shows the how this roughness metric compares with the raw IMU reading, with g subtracted, and the continuous derived roughness metric. As it can be seen, it follows the trend of the roughness quite well.

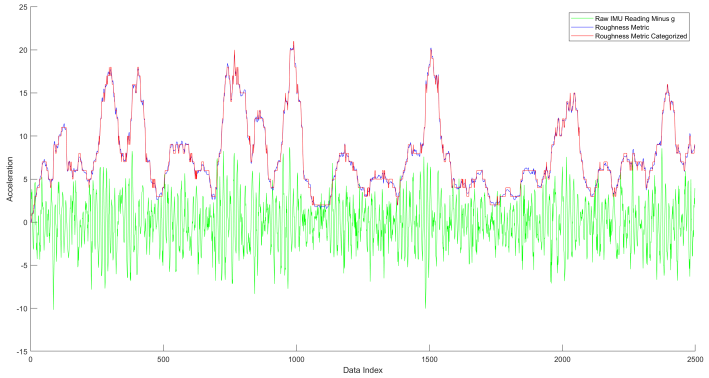


Figure 3.5: Roughness metric categorized from 0-35.

After training the network with image patches with these categories, the results were not as expected. As discussed in Section 5.3.2 the accuracy of the results improved if some misclassification were allowed. As such we determined that having such fine categories for roughness were not necessary. Especially for a vehicle like the Yamaha AS-X1, we really just need to determine the terrain which rough, smooth and partially rough. As such it was determined to re-categorize the roughness metric once again. The roughness metric was re-categorized as shown in Table 3.1. Figure 3.6 shows how the re-categorized roughness metric compares against the raw IMU reading with g subtracted and the continues derived roughness metric. Again, it can be seen here that the re-categorized roughness metric follows the roughness well.

Previous Roughness Metric Categories	New Categories
0-8	0
9-17	1
18-26	2
27-35	3

Table 3.1: Table showing previous roughness metric categories and the re-categorized roughness metric categories.

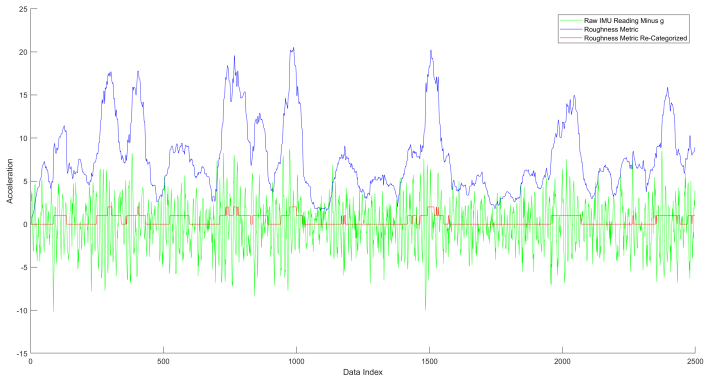


Figure 3.6: Previous roughness metric re-categorized

3.1.3 IMU Based Roughness Metric Refinement

The categorized roughness metric previously discussed was used to label the images of the terrain. These labels along with the raw RGB images were used to train the convolutional neural networks as described in chapter 5. However we found that this was not producing the desired results. One issue we realized was that even though we used a small window size for the simple moving average, there was still a small delay between a high raw IMU reading and the corresponding roughness metric. This small delay could possibly be contributing to a decrease in performance of the networks. As such, we decided to use a smoothing filter, and then apply the roughness categorization as discussed in section 3.1.2.

Two types of filters were considered, a Gaussian smoothing filter and a mean smoothing filter. Both these filters operated over a window size of 50 data points, that is, 25 previous data points, and 25 data points ahead. This contrasted with the simple moving average which was applied to the previous 50 data points. The benefit of this method is that there was not any delay between the raw IMU reading and the corresponding roughness metric. The mean smoothing filter shown in figure 3.7, smoothens the raw IMU reading averaging the readings over the window. The Gaussian smoothening filter, show in figure 3.8 applies a Gaussian weighted mean to the points over the window.

Looking at these plots, it can be seen that the Gaussian filter smoothens the data much more than the mean smoothing filter, and this is reflected in the final roughness metric. For instance between data points 500 to 1000, where the mean smoothing filter resulted in some spikes in the

Figure 3.7: Mean smoothing filter.

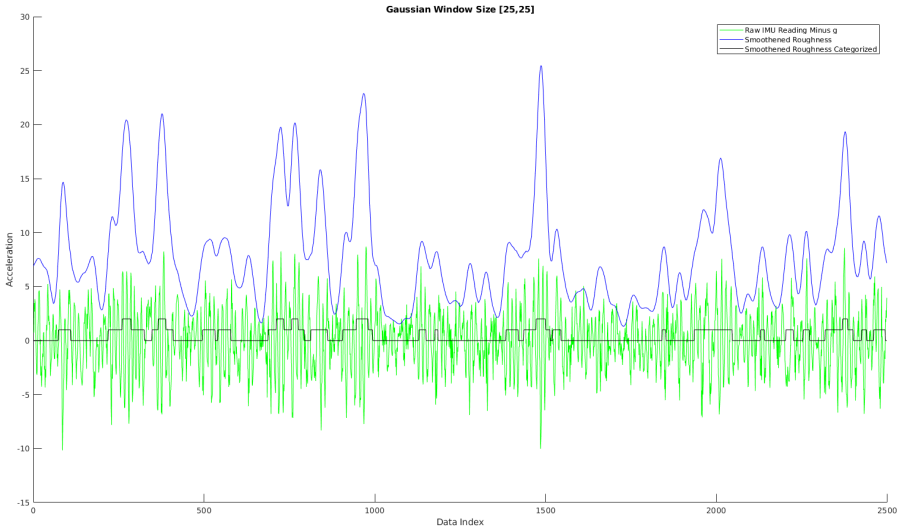


Figure 3.8: Gaussian smoothing filter.

roughness metric, the Gaussian smoothing filter resulted in a more steady roughness metric. It was unclear as to whether this could be a benefit or a disadvantage to training a deep learning network. On one hand it could be possible that Gaussian smoothing filter is not capturing the true roughness, and the mean smoothing filter is capturing the true roughness better. On the other hand, having such fluctuation in the roughness, could be confusing to a learning network. As such we decided to use both to train the semantic segmentation networks, and analyzed the results.

3.1.4 Discussion of the IMU Based Roughness Metric

It is important to note that this roughness metric is dependent of the velocity of the robot. The faster the robot moves over a particular piece of terrain, the rougher it will feel, and this will be reflected in the roughness metric. For example if the robot is driving slowly (10 kmph) over an area with small pebbles, it would feel smooth and there would not be much vertical movement of the robot. As such, there vertical acceleration measured by the IMU would be low. Consequently, this terrain would get a label of 0. However if the robot drove over the same area much faster, for example 40kmph, there would be more vertical accelerations measured in the IMU and hence the labels would be higher, possibly 1 or 2.

As such, this introduces some inconsistency in the labeling, and this can affect the learning process. A convolutional neural network takes the image as input and tries to find the association between the pixels and the labels. From a pure visual standpoint, it can be easily seen how the network would be confused based on the example above. In the example, the network would see one image with terrain labeled 0, and then another image with the exact same terrain, but a different label. It is clear that this inconsistency can confuse the network. As such, the velocity of the robot must be included as an input into the network to help with this inconsistency.

Another important point to note with this roughness metric is a major assumption that was introduced due to the use of the IMU. The IMU is located at the center of the robot. As previously discussed the roughness felt is transferred from the wheels to the suspension system to the body of the robot which the IMU is attached to. Consequently the roughness measured by the IMU is a combination of the roughness experienced by all four wheels. However, it is possible that the wheels may not all be on the same terrain. For example it is likely that the left side of the robot is on a rough patch of terrain while the right side is on smoother terrain. Figure 3.9 shows an example of this.

Since one of the aims of this thesis is to have the images be autolabeled, and the robot has no way of measuring the individual roughness experienced by each wheel or each side of the robot, we must make the assumption that the roughness of the terrain being measured is constant. This means that pose of the robot is labeled with the roughness metric. In an image, the pose is represented by a line or row of pixels, and therefore all pixels in the row must receive the same roughness metric label. This assumption is called the terrain consistency assumption.

Lastly another consideration for this roughness metric, is the sensors involved. All that is needed is an IMU. Given that these devices are relatively cheap, it means that this roughness metric can be derived "cheaply" for any robotic application since expensive devices such as LIDAR or stereo cameras are not needed.

Figure 3.9: Example of the different types of terrain traversed by the robot at the same time.

3.2 Roughness Metric Based on Point to Plane Distance

The initial idea was to solely used the IMU based roughness metric, but after training networks using this metric, some of the its shortcomings were realized. As such we thought it prudent to develop another roughness metric that could possibly overcome the weakness of the IMU based roughness metric, and to serve as a comparison against ours. Point to plane distance was the other metric considered for roughness. The idea of this metric is to measure the profile of the terrain. Quite simply, this metric takes a point from the point cloud, finds a neighborhood of points within a specified radius of the selected point, and fits a plane to the neighborhood of points. The point to plane distance is then the distance of the selected point from the fitted plane. The idea of fitting a plane to a neighborhood of points was used in [14]. However instead of using the point to plane distance, Bellone et al used the normals of each point within the neighborhood to derive an uneveness terrain descriptor. More recently in [16] Kim et al used the mean point to plane distance as an input into the multimodal network. Having seen the success of this method, we determined it would also be a useful metric to describe roughness.

3.2.1 Generating the 3D Point Cloud From the Selected Area of the 2D Image

In order to find the point to plane distance, a point cloud is needed, which is a three dimensional array of the points in space. From section 4.3 we have the area of the image, as shown in figure 3.10b, in front of the robot for which we want to find the point to plane distance of each point represented by a pixel. In order to do this, the pixel in the image frame must be transformed to a 3D point cloud in the multisense frame.

Given a pixel $\mathbf{x_p} = (x_p, y_p)$, in the image frame, we seek the projection of that pixel, $\mathbf{x_c} = (x_c, y_c, z_c)$ in the camera (Multisense) frame. We already have z_c from the depth map produced by the Multisense, the focal lengths, (f_x, f_y), of the Multisense, as well as the principal point (p_x, p_y). Given all this information, x_c and y_c can be calculated as given by equations 3.1 and 3.2 respectively.

$$x_c = \frac{x_p - p_x}{f_x} z_c \tag{3.1}$$

$$y_c = \frac{y_p - p_y}{f_y} z_c \tag{3.2}$$

By apply this equation to every pixel in the selected area, a point cloud can be generated corresponding to the selected area. From the point cloud, the point to plane distance for each point in the point cloud can be calculated.

3.2.2 Derivation of Point to Plane Distance

A stated by [22], plane is defined by:

$$\mathbf{n}.(\mathbf{x} - \mathbf{x_r}) = 0 \tag{3.3}$$

Where the normal vector $\mathbf{n} = (a, b, c)$, passes through the reference point $\mathbf{x_r} = (x_r, y_r, z_r)$ and $\mathbf{x} = (x, y, z)$. Expanding equation 3.3, we get:

$$ax + by + cz + d = 0 \tag{3.4}$$

$$d = -ax_r - by_r - cz_r \tag{3.5}$$

d is the dot product of the reference point and the normal, and is given by equation 3.5. Therefore to precisely define a plane, we need a normal $\mathbf{n}$ and a reference point $\mathbf{x_r}$. To do this, the method of first order 3D plane fitting is applied using Principal Component Analysis as defined in [23], which solves for the best fitting plane given a set of points. In particular to solve for the normal, we follow the method outlined in [14]. To solve for the plane, at least three points are needed, and therefore must define a neighborhood of points as:

$$\mathbb{X}^n = \left\{\mathbf{x_i} \in \mathbb{R}^3 : \|\mathbf{x_i} - \mathbf{x_q}\| \leq r_k\right\} \tag{3.6}$$

Where r_k is the kernel radius, which is the maximum distance of a neighor point from the query point $\mathbf{x_q}$. With the neighborhood of n points, we define the reference point $\mathbf{x_r}$ as the centroid, $\bar{\mathbf{x}}$ of the neighborhood of points.

$$\mathbf{x_r} = \bar{\mathbf{x}} = \frac{1}{n}\sum_{1}^{n} \mathbf{x_i} \tag{3.7}$$

With the reference point derived from equation 3.7, the normal $\mathbf{n}$ is obtained by analyzing the Eigenvectors of the covariance C of the neighborhood of points $\mathbb{X}$, where C is given by:

$$C = \frac{1}{n}\sum_{1}^{n} (\mathbf{x_i} - \mathbf{x_r})(\mathbf{x_i} - \mathbf{x_r})^T \tag{3.8}$$

The Eigenvectors are the principal components of $\mathbb{X}$, and therefore form the orthogonal subspace of the neighborhood of points, [14]. The Eigenvector corresponding to the smallest Eigenvalue is the normal $\mathbf{n}$, of the best fit plane through the points in $\mathbb{X}$. With $\mathbf{n}$, we have solved for (a, b, c), and with $\mathbf{x_r}$, we can solve for d as shown in equation 3.5.

Given the equation of the plane, the distance of the query point $\mathbf{x_q} = (x_q, y_q, z_q)$ from the plane is given by equation 3.9:

$$\texttt{dist} = \frac{|ax_q + by_q + cz_q + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{3.9}$$

CloudCompare, [24], is a standard by which all point cloud operations are compared against. CloudCompare has a built in roughness metric calculation tool which calculates the roughness as the distance of a point from the best fit plane obtained from the neighboring points within a specified kernel radius from the query point. Though they do not specify which plane fitting method is used on the program, we thought it prudent to perform a comparison between the output of our point to plane roughness calculation and the output from CloudCompare.

In order to do this, area of terrain requiring labels was selected using the method in Section 4.3. Figure 3.10a shows the image of terrain, and figure 3.10b shows the region which was converted to a point cloud.

(a) (b)

Figure 3.10: (a):RGB image (b):Trapezoidal (non-red) area selected to find point to plane roughness.

This point cloud was then run through our roughness calculation code, and also through CloudCompare's roughness calculation tool, both with a kernel radius of 0.2 meters. The output roughness from our code was plotted against the output of the CloudCompare, and the equation best fit line through the points was found. This is shown in figure 3.11, with the equation of the best fit line is approximately $y = x$. As can be seen, there are some outlier points, however, most of the points do agree with each other quite closely. This means that our roughness matched extremely well with CloudCompare's roughness since the best fit line was a line with gradient 1 through the origin. This proves that our code works as expected.

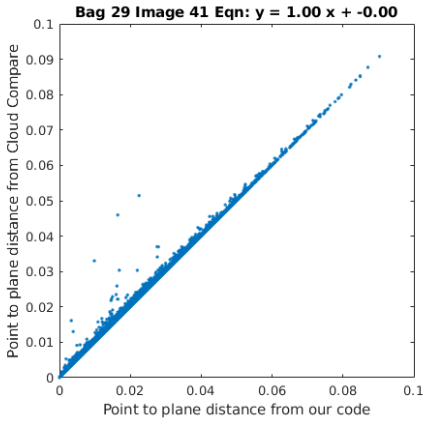


Figure 3.11: Our point to plane roughness distance output vs CloudCompare roughness output.

Additionally, a visual comparison of the results were done. To do this, the roughness calculated from both our code and CloudCompare were re-projected back onto the image frame. The images were visually inspected for consistency.

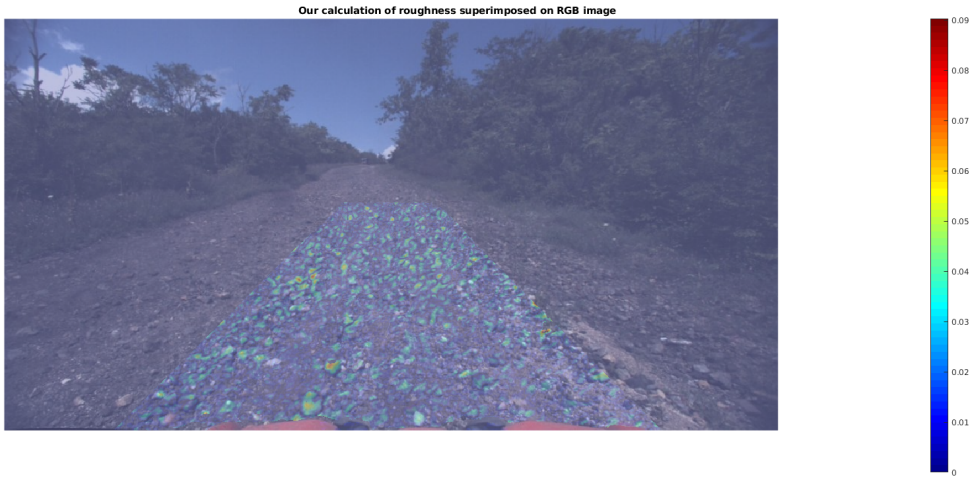


Figure 3.12: Our point to plane roughness projected onto the terrain image.

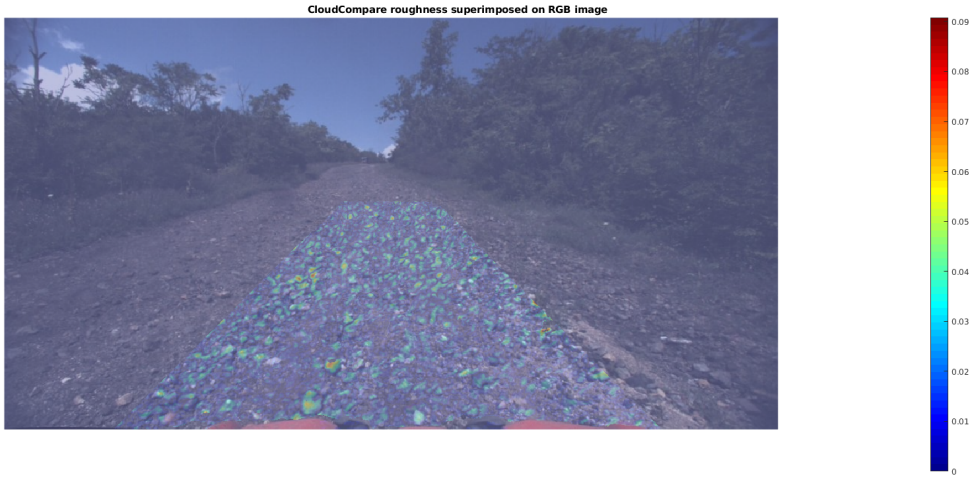


Figure 3.13: CloudCompare point to plane roughness projected onto the terrain image.

Figure 3.12 shows the our calculated point to plane roughness projected onto the terrain image while figure 3.13 shows the result of CloudCompare roughness projected onto the terrain image. As can be seen, both image look the same which corroborates the plot from 3.11 proving

that our code is calculating the point to plane roughness correctly. Both of these comparisons were done with different images, and all follow the same pattern of success.

We have chosen to calculate this ourselves because CloudCompare is a graphical user interface, and consequently has no facility to enable scripting, or interfacing with outside programs written in other languages such as Matlab or Python. One of the aims of this thesis was to create an autolabeling system for the data. As such it was impossible to use CloudCompare for this purpose, so our version of the roughness calculation needed to be written.

3.2.3 Selection of Kernel Size

With the confidence that our method runs correctly, we needed to select the optimum kernel radius. Referring to equation 3.6, the kernel radius affects how many points are included in the neighborhood of the query point. More neighborhood points mean longer runtime per query point. The kernel radius also affects the fit of the plane. Too small of a kernel radius, means a low number points, which could have an averaging effect on the terrain, in effect smoothing out the rough areas. On the other hand, too large of a kernel radius, can have the opposite effect, over-exaggerating the roughness of the terrain. In order to select the best kernel radius, different kernel radii were used to calculate the point to plane roughness, and then the roughness was projected back onto the terrain image. The roughness was colored with a colormap spectrum such that 0cm was blue and greater than 10cm was red. This roughness coloring was compared to what our expected roughness of the terrain should be. Figures 3.14 to 3.19 show the effect of different kernel radii for one example image, corresponding to the terrain in figure 3.10a.

Figure 3.14: Kernel radius= 0.05m

Figure 3.15: Kernel radius= 0.1m

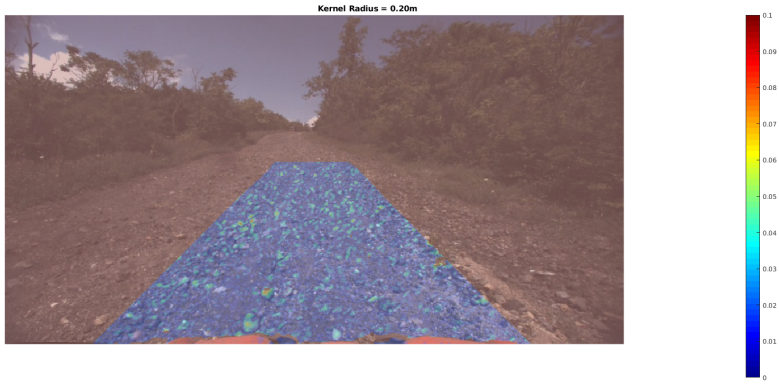


Figure 3.16: Kernel radius= 0.2m

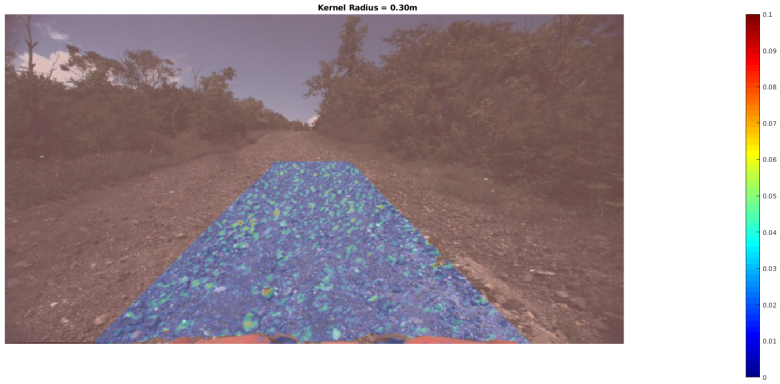


Figure 3.17: Kernel radius= 0.3m

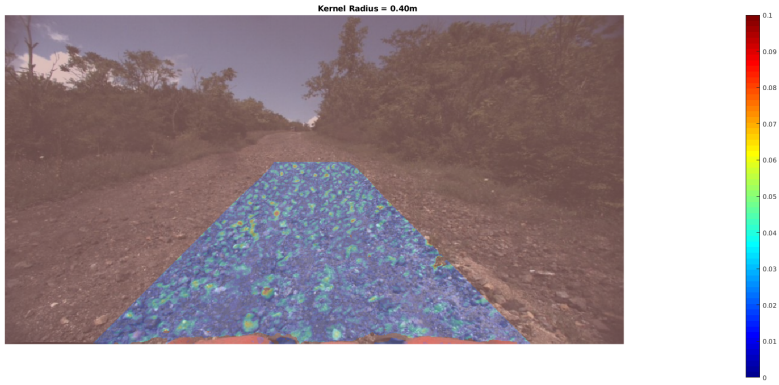


Figure 3.18: Kernel radius= 0.4m

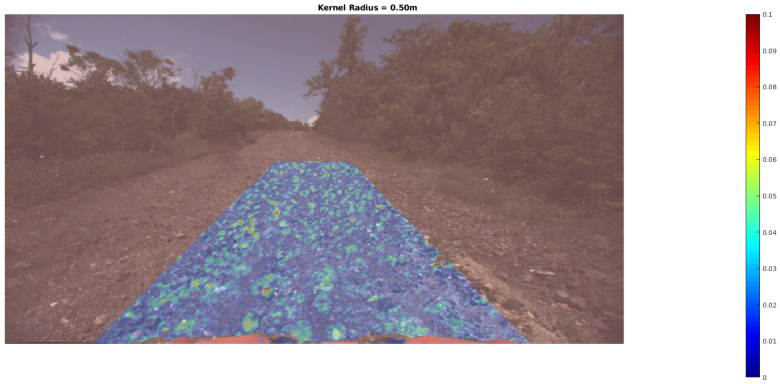


Figure 3.19: Kernel radius= 0.5m

Kernels from 0.05m to 0.5m were considered. Small kernel radii like 0.05m and 0.1m made the terrain appear smoother that it should. Larger radii showed the roughness of the terrain more clearly. Kernel radii greater than 0.5m produced an output that looked no different from a kernel radius of 0.5m. After comparing similar images for other terrain, we decided that a kernel radius of 0.2m was optimum.

3.2.4 Describing Roughness Using Point to Plane Distance

Now that we know how to calculate the point to plane distance and have selected the appropriate kernel radius, there are different ways of using point to plane distance to describe roughness. For this purpose we considered three methods of using the point to plane distance to describe roughness: absolute distance, max distance, and mean distance as used in [16]. Absolute distance is the distance of the query point from the plane that was fitted to the neighborhood of points around the query point and is given by equation 3.9.

Max distance finds the distance of each neighborhood point from the fitted plane, and then labels the distance of the query point as the maximum of those distances. It is given by equation 3.10, where $\mathbf{x} = (x, y, z)$, and $\mathbb{X}^n$ is the neighborhood of points as defined in equation 3.6.

$$\text{Max Distance} = \underset{\mathbf{x}\in\mathbb{X}^n}{\operatorname{argmax}} \left(\frac{|ax + by + cz + d|}{\sqrt{a^2 + b^2 + c^2}} \right) \tag{3.10}$$

Mean distance find the distance of all points in the neighborhood of the query point, and then labels the distance of the query point as the average of those distances. It is given by equation 3.11 where $\mathbf{x_i} = (x_i, y_i, z_i)$ and $\mathbb{X}^n$ is the neighborhood of points as defined in equation 3.6.

$$\text{Mean Distance} = \frac{1}{n} \underset{\mathbf{x_i}\in\mathbb{X}^n}{\Sigma} \left(\frac{|ax_i + by_i + cz_i + d|}{\sqrt{a^2 + b^2 + c^2}} \right) \tag{3.11}$$

Figures 3.20 to 3.22 show the selected area from figure 3.10a labeled with the the different point to plane roughnesses just described. The colorbar to the right color the labels according to the distance, where 0m is colored dark blue and 0.2m is colored dark red. the same color scale was used for all point to plane roughness to allow for better comparison.

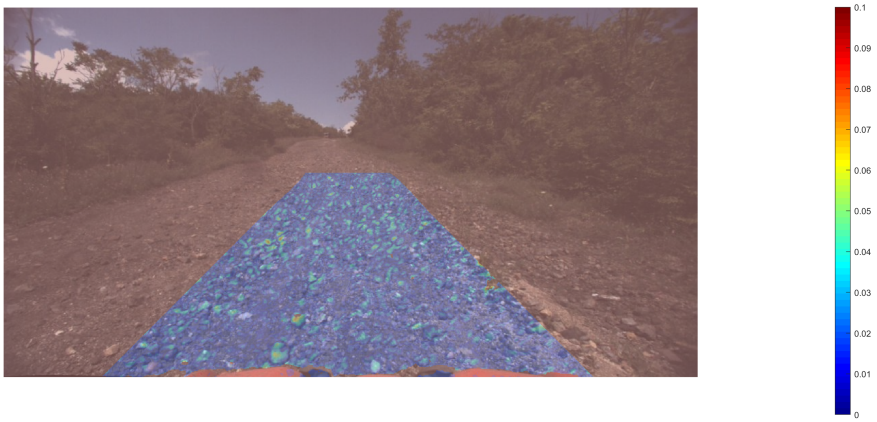


Figure 3.20: Terrain labelled with roughness as absolute point to plane distance

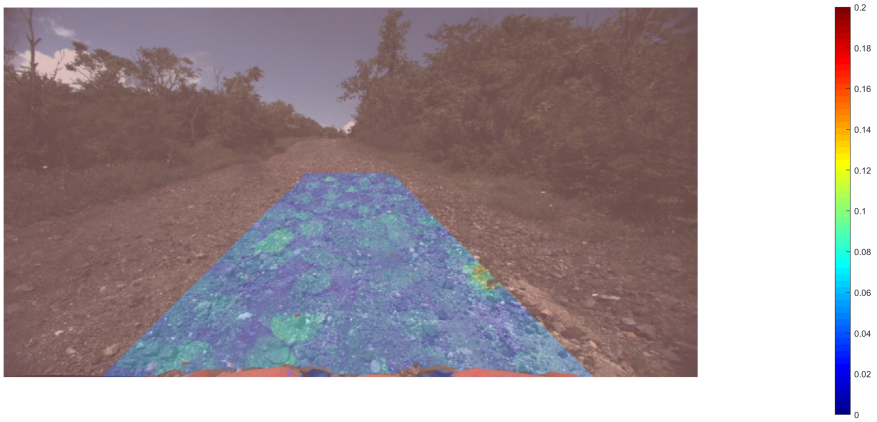


Figure 3.21: Terrain labeled with roughness as max point to plane distance

Figure 3.22: Terrain labeled with roughness as mean point to plane distance

From these images, the mean roughness produced the least informative labels. Since the mean of the points within the neighborhood was taken, this had the effect of smoothing out actual roughness, and thus gave low point to plane distances for the labels. On the other hand, the max and absolute distance roughness were more representative of the actual roughness. Max distance roughness had the effect of grouping the roughness into patches as can be seen from 3.21. In this sense, it give an upper bound or worst case scenario label for roughness. Absolute distance roughness had the opposite effect, that is, it was more focused in its labeling. It is difficult to determine whether max or absolute distance roughness would be better for learning, so therefore both were considered and used for training. Mean distance roughness was not used.

3.2.5 Categorizing Point to Plane Distance Roughness

Now that the point to plane distance for each pixel in the selected area of the image was derived, a discrete label was necessary to label the pixel. Discrete labels are needed for the ground truth for training the learning networks. These labels categorize ranges of point to plane distances, and also must be representative of the roughness of the terrain. To select these ranges, we used our own knowledge of driving on the terrain as well as comparison with the IMU labeled roughness. Since we wanted to compare the IMU roughness metric with the point to plane roughness metric, we wanted the labels between them to have some similarity. To do this, we visually compared images labeled with the IMU roughness metrics next to images labeled with the point to plane distance metric. We looked at areas of terrain which would give certain IMU roughness metric labels, and their corresponding point to plane metric labels. We chose the following categories.

Point to plane distance	Category
0m - 0.04m	0
0.04m - 0.07m	1
0.07m - 0.09	2
>0.09m	3

Table 3.2: Categories selected for point to plane roughness distances

3.2.6 Discussion of the Point to Plane Roughness Metric

As mentioned before, the idea of fitting a plane to 3D points have been applied to classifying terrain as in [14] and [16], so there is good precedence for its usage. Furthermore, because it gives a physical measure to roughness, it is one that we can easily understand, visualize and identify. For example we can easily picture a terrain labeled as category 1 which would contain small pebbles of maximum diameter 0.04m or 4cm. On the other hand, the IMU based roughness metric is a bit more abstract and more difficult to visualize or relate to. Furthermore the point to plane roughness metric is independent velocity as opposed to the IMU roughness metric.

Lastly the drawback to this metric is the hardware required. In order to get point to plane distances, we need 3D points which means we need a sensor that is able of giving us 3D points, that is, either a stereo camera or a 3D lidar. Both of these pieces of equipment can be very expensive especially when compared to the IMU roughness metric which can be computed using only a monocular camera and an IMU. One may ask why bother to use the point to plane roughness for training, when it can just be computed. The problem is computation time. Even when parallelized, point to plane distance calculations are an expensive process. Most of the computation time is spent in finding the neighbors of the query point. In a real time scenario, this can be problematic especially if the robot is driving fast and needs to determine the roughness of the terrain quickly. For this purpose a deep learning network is preferred since inference time is in the milliseconds range.

Chapter 4

Auto-labeling Images with the Roughness Metric

With the roughness metrics derived, the next task was to develop a system for autolabeling the images. This system takes in a rosbag file containing logs of the sensors necessary to derive the roughness of the terrain. In particular, these rosbags contained IMU data, odometry data and Multisense camera data. With this data, the system derives the path driven by the robot, and then projects the path onto the image. With the knowledge of the path driven by the robot, the appropriate area along the path is selected to be labeled with the roughness metric. This system is fully autonomous. The only input required from the user is to set the correct file paths, and hit run. The subsequent sections of this chapter details the working of this system.

4.1 Deriving the path driven by the robot

We make the assumption that the robot is driving on relatively flat terrain, meaning that there are no large or sudden elevation changes. As such we are only considering the robot's pose in a plane, and therefore the pose has three degrees of freedom: x, y and θ. The sensors provided the linear velocity, v, and the angular velocity, ω, of the robot. Since extreme accuracy was not necessary we decided to use the unicycle model, as described in [25], for dead reckoning to derive the path driven by the robot. The starting pose of the robot was initialized as $(x, y, \theta) = (0, 0, 0)$. Each point along this path represents the pose of the robot along the path. The update is then performed as follows:

$$\begin{aligned} \Delta t &= timeStamp(i+1) - timeStamp(i) \\ \theta &= \theta + \frac{\omega \Delta t}{2} \\ x &= x + v\cos(\theta)\Delta t \\ y &= y + v\sin(\theta)\Delta t \\ \theta &= \theta + \frac{\omega \Delta t}{2} \end{aligned} \tag{4.1}$$

Since the linear velocity and angular velocity were derived from the IMU, the coordinates calculated from equation 4.1 are in the IMU frame. This equation is applied for each bag file so that we have the entire path driven by the robot for that rosbag file. The path is stored as a 2D array in memory. In order to project the path onto the image, these coordinates need to be transformed from the IMU frame into the image frame.

4.2 Projecting the path onto the image

With the path of the robot for the entire log file derived through dead reckoning from equation 4.1, the path needs to be projected onto each image. However, firstly, the pose of the robot at the time the image was taken needed to be obtained. The index of this pose in the path array is called the $startIndex$. In order to do this, we make the assumption that the all the sensors are in sync, meaning that all sensors start logging at the start time, $t_0 = 0$ seconds, of the rosbag file. We also assume that all the sensors operate at exactly the stated configuration frequency. The Multisense S21 operates at $f_{mult} = 10$Hz, while the Xsens IMU operates at $f_{IMU} = 200$Hz.

From the rosbag file, the number of images that were taken through the duration of the log can be determined. Therefore given an image number, the time that image was taken can be calculated by applying equation 4.2.

$$t_{im} = imageNum \times \frac{1}{f_{mult}} \tag{4.2}$$

With the start time of the image, the index number of the IMU data from which we will select the path and the IMU roughness metric can now be determined. The start index is given by equation 4.3.

$$startIndex = t_{im} \times f_{IMU} \tag{4.3}$$

$startIndex$ is therefore the index of the pose in the path array at which the image was taken. All points in the path array from $startIndex$ to the end will be projected onto the image. We define the points from this start point as $path_{IMU}$. Since the IMU roughness metric is also calculated from the linear acceleration about the z-axis of the robot, it is also in sync with other data from the IMU. This means that each path point has a corresponding IMU roughness metric associated with it. This means that the $startIndex$ is also the beginning of the index of the roughness metric matrix which would used to label the image.

To transform the path points from the IMU frame to the image frame, the points first need to be transformed into the Multisense frame. With the points in the Multisense frame, the points are then projected into the image frame via the projection matrix. Figures 4.1 and 4.2 show the locations of the coordinate frames of the sensors in the robot. The robot frame is defined as the Novatel frame, which is located at the center of the robot. We were provided with the transformations of each sensor with respect to the robot frame, by the engineers who built the robot. Additionally, we were also provided with the camera projection matrix by the Multisense information rostopic.

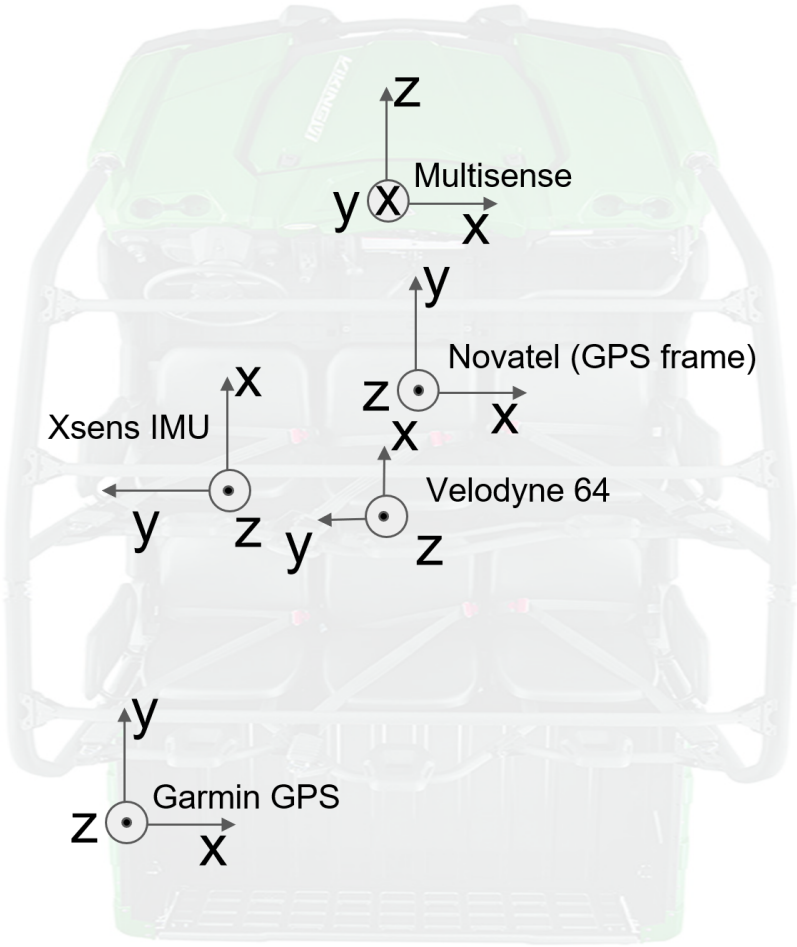


Figure 4.1: Top view of sensor frames.

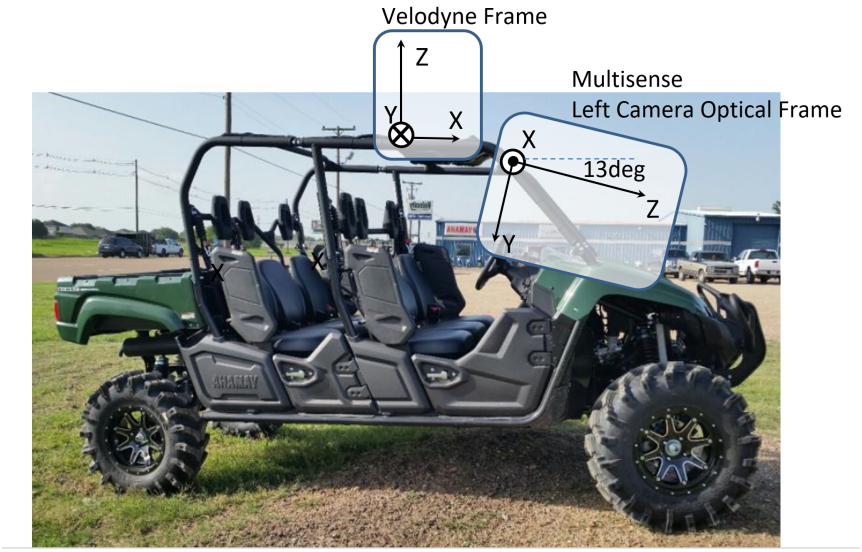


Figure 4.2: Side view of LIDAR and Multisense sensor frames.

We define the transformation of the IMU with respect to the Novatel frame as $T^{IMU}_{Novatel}$, the transformation of the Multisense with respect to the Novatel frame as $T^{Multi}_{Novatel}$ and the Multisense camera projection matrix as P. Since these are all homogeneous transformations, the path array needs to be modified to work with these transformations. Recall that the (x,y) coordinates of the path were just calculated. This is a Nx2 matrix where N is the number of path points. The path points represent the points on the terrain which the robot traversed, and since the z-axis of the IMU points upwards, the path points will have a negative value. These points were calculated from the IMU frame which is 2 meters above the ground, so therefore the z coordinate of all these points in the IMU frame is -2. A column of -2's is appended to the path points to get a Nx3 matrix. Lastly to work with the homogeneous transformations, a column of ones is appended to the path points, which would now give a Nx4 matrix, $path_{IMU}$. The points can now be transformed from the IMU frame to the Multisense frame by equation 4.4.

$$path_{Mult} = T^{Mult}_{Novatel} \times \left(T^{IMU}_{Novatel}\right)^{T} \times path_{IMU} \tag{4.4}$$

With the path points transformed into the Multisense frame, we require the points in the image frame. Therefore the path points from the Multisense frame need to be projected into the corresponding pixels in the image frame. This is simply done by multiplying the points in the Multisense frame by the Multisense camera projection matrix, P, as shown in equation 4.5.

$$path_{imageFrame} = P \times path_{Mult} \tag{4.5}$$

$path_{imageFrame}$ is a Nx3 matrix. To get the path in pixel coordinates, $path_{pixels}$, the each element in column 1 and 2 must be divided by the corresponding element in column 3. Since $path_{IMU}$ contained all points from start index to the end of the rosbag file, so too will $path_{pixels}$. This means that when the path is plotted onto the image, the path would extend beyond the boundaries of the image. To remedy this, we simply only select points which were within the boundaries of the image, that is, the x pixel coordinate must be within 1 and 1024, while the

y pixel coordinate must be within 1 and 544. The indices of these points are also noted, and the corresponding roughness metric for the image is simply the roughness metric matrix values which correspond to the indices.

Figure 4.3: Example of path projected onto the image.

Figure 4.4: Example of path projected onto the image.

Figures 4.3 and 4.4 show two examples of the path projected onto the image. In figure 4.3, the path extended beyond the boundaries of the image, and hence it was truncated. In figure 4.4 the path ended at a point within the image, so the path did not extend beyond the boundaries of the image and hence no truncation was necessary.

4.3 Selecting the area in front the robot to label

With the path projected onto the image, we now move on to the process of labeling the path with the IMU roughness metric.The label matrix is first initialized to a matrix of size 1024 x 544,

with each matrix entry being the no information class of 4 when creating labels for classification, or NaN if the labels were being created for regression. The no information class is the label given to pixels for which there is no information about roughness. Our aim was to have the labels centered along the path the robot drove in the image, and also be roughly consistent with the perspective of the image. This meant that if the robot is imagined to be driving through the image, its footprint would decrease in size as the robot moves further away from the camera. Therefore the labels must reflect this as well.

As mentioned in sections 3.1.4, the IMU based roughness metric is measuring the roughness experienced by all four wheels of the robot. While it is possible that the left and right side of the robot could experience different types of terrain, we have no way of measuring the individual roughness experience by each wheel or each side of the robot, and as such we have to make the assumption that at a particular instant in time, the terrain being measured in consistent.

Each path point represents the pose of the robot along the path, and since each pose is labeled with the roughness metric, we decided that for each path point in the image, would be represented by the row of pixels at that path point, or the y-coordinate of that path point. Taking the previous assumption into consideration, this means all pixels of that row would receive the same IMU roughness metric label. To determine how much of the row to be labeled, labeling first begins at the path point closest to the bottom of the image. The first row of labels would be centered about this point, with a width of 725 pixels, that is 362 pixels to the left and 362 pixels to the right of the path point. In all the images, the tip of the front of the robot could be seen, and from this we can get a average of the width of the robot in pixels. After comparing several images, 725 pixels was determined as the appropriate starting width of the robot.

So the first row of length 725 pixels, centered about the first path point, is labeled with the IMU roughness metric for that path point. To deal with the perspective issue, for the next path point, the length of the labeled row is decreased by one pixel on each side of the path point, that is, the next row would be 723 pixels in length and so on for the other subsequent path points. We found that this method closely followed the perspective of the image. This process is performed for all the points in the image. Figures 4.5 and 4.6 show the labels generated from this method superimposed onto the images form figures 4.3 and 4.4. The color bar to the right shows the labels, where blue represents the roughness class of 0, and red represents the no information class 4.

While this method of labeling came close to our expectations, we found that there were instances of rows labeled with the no information class in between rows labeled with the roughness classes. This can be seen in both figures 4.5 and 4.6. This mainly occurred for the rows closer to the robot. It is easy to understand why this occurs. Each path point is separated by 0.005 seconds, and if the robot is moving at constant speed, then the distance between each path point would be the same. However, when the path points are projected onto the image, the path points closer to the camera would appear further apart than the points further from the camera, due to perspective. Therefore, when labeling, the labels closer to the camera would have some unlabeled rows due to the fact that the path points were further apart.

Due to this issue, the labels also did not properly follow the perspective of the image. Our method only decreased the length of the labeled row for each subsequent path point. By having rows of the image with no path point, the length of the labeled rows did not follow the perspective of the image. Furthermore, having rows of unlabeled terrain amidst rows of labeled terrain

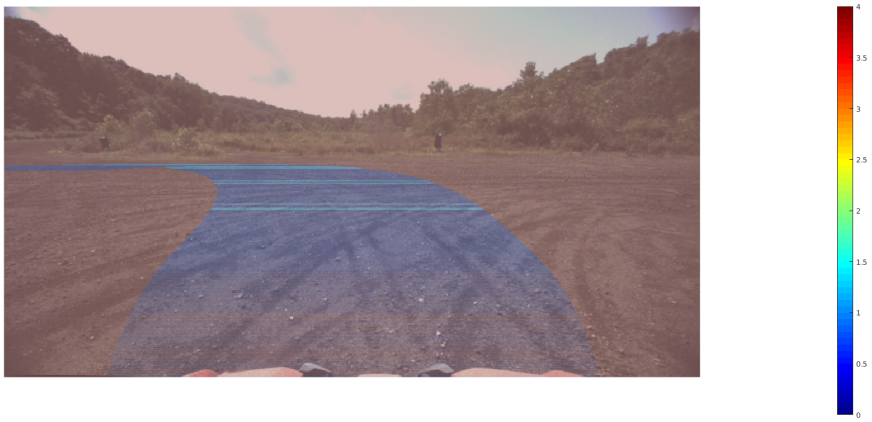


Figure 4.5: Initial IMU roughness metric labels superimposed onto the image from figure 4.3.

Figure 4.6: Initial IMU roughness metric labels superimposed onto the image from figure 4.4.

would make it difficult for learning. As such to deal with this issue, after this initial labeling, if any unlabeled rows we found between labeled ones, artificial path points were generated by interpolating between the path points that existed above and below the unlabeled rows. Similarly, the corresponding roughness metric was also generated by interpolating between the roughness of the labeled rows above and below the unlabeled ones. The result of this operation is shown in figures 4.7 and 4.8.

Lastly, we determined that the labels very far from the camera were not necessary, and con-

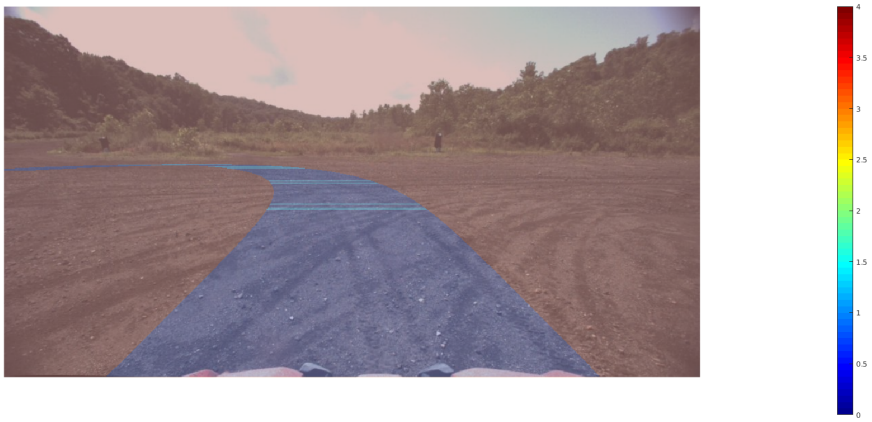


Figure 4.7: Cleaned up IMU roughness labels superimposed onto figure 4.3.

Figure 4.8: Cleaned up IMU roughness labels superimposed onto figure 4.4.

sequently labels beyond 300 pixels from the bottom of the image were discarded. It is only necessary to determine the roughness of the terrain directly in front of the robot. Therefore having labels far ahead of the robot would not be necessary. Furthermore having labels that far ahead of the robot could prove to be difficult for a convolutional neural network to learn. A large rock 50 meters ahead of the robot would only appear a few pixels in diameter in an image, while a small rock close to the camera would also have the same pixels in diameter. This type of confusion would make it difficult to learn a proper matching between terrain and roughness labels.

By comparing various images and looking at distances from the depth map, the area 300 pixels from the bottom of the image is equivalent to 15-18 meters ahead of the robot. We concluded that labeling the terrain up to that distance was good enough and thus the area 300 pixels from the bottom of the image was adequate. The final IMU roughness labels are shown in figures 4.9 and 4.10.

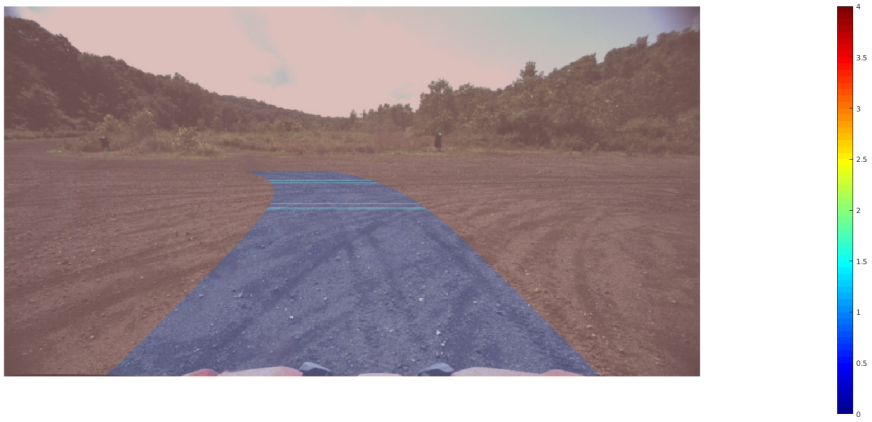


Figure 4.9: Final IMU roughness labels for figure 4.3.

Figure 4.10: Final IMU roughness labels for 4.4.

This labeling method output a label matrix of size 1024 x 544, with the area in front of the robot labeled according to the IMU based roughness metric. To get the roughness labels

according to the point to plane distance roughness metric, the same area of the image labeled with the IMU roughness is obtained by simply applying a mask to get all the pixels not labeled as the no information class. A 3D point cloud is generated from these pixels as described in 3.2.1, after which the point to plane roughness metric is obtained as detailed in 3.2. For completeness the point to plane distance roughness metric labels are for figures 4.3 and 4.4 are shown in figures

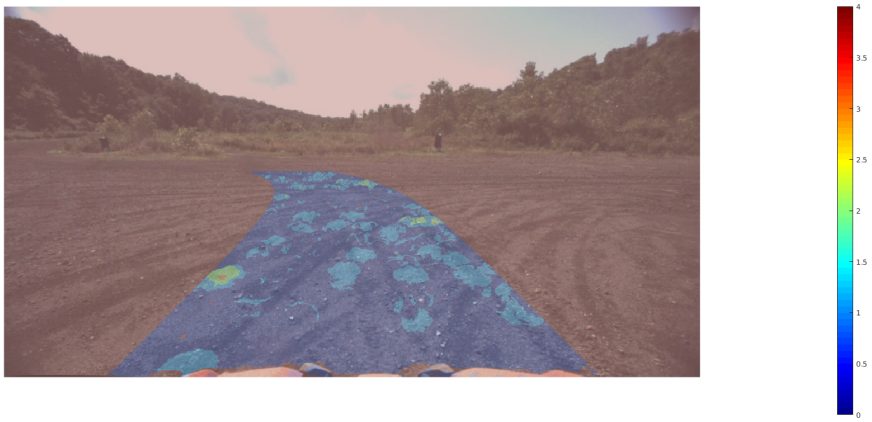


Figure 4.11: Point to plane roughness metric labels superimposed onto 4.3

Figure 4.12: Point to plane roughness metric labels superimposed onto 4.4

The entire process detailed in this chapter and the previous one was written and designed to be a fully automated system in Matlab. The system took in a bag file, extracted the depth

images, RGB images, IMU roughness metric and the path driven by the robot. For each image, the appropriate start index was found, the path projected onto the image, and the image was labeled with the IMU roughness metric. The area selected for labeling was then used to obtain the point to plane distance labels as described in section 3.2. Finally the RGB image, the IMU roughness metric labels, the point to plane distance roughness labels as well as other meta data such as velocity were saved to separate directories. The system also made it easy for a user to change the parameters such as kernel radius or type of filtering.

If the system is run to extract everything but the point to distance labels, it takes approximately 3 hours to process 135 Gigabytes of log files. If the point to plane distance labels are included it takes a couple days to run. In total we were able to get approximately 18000 images and labels which we split up 60:20:20 for training, validation and testing respectively.

Chapter 5

Training a Deep Learning Network to Learn the Roughness Metrics

With the images and labels extracted, the next step in the process was to use a learning network to predict the roughness of terrain given an image. In order to do this, various methods were considered. These methods ranged from using image patches trained on a convolutional neural network for regression to using a semantic segmentation network for full image classification. This chapter details all these methods and the results.

5.1 Evaluation Metrics

Since comparison is being done on the labels of an image, there are specific evaluation metrics which are used. We follow the evaluation metrics used in [17] and [18]. It is important to note that these metrics were only calculated on the pixels whose labels were used for training. For each image, the ground truth and predicted labels are compared pixel wise, and the number of pixels correctly classified is defined as the global accuracy. Class accuracy as defined by [17], is the number of correctly predicted pixels for each class, and class average accuracy is simply the mean of the all the class accuracies and is given by equation 5.1, where n_{cl} is the number of classes and n_{ij} is the number of pixels of class i predicted as class j.

$$\begin{aligned} t_i &= \Sigma_j n_{ij} \\ Class\ Avg\ Acc &= \frac{1}{n_{cl}} \Sigma_i \frac{n_{ii}}{t_i} \end{aligned} \tag{5.1}$$

The other, and possibly most important metric considered, is per class intersection over union, as used in the Pascal VOC 12 Challenge [26]. Intersection over union (IoU) for each class is given by equation 5.2, as defined in [17] and [26]. Mean IoU in simply the average of all the per class IoU. IoU calculates how much of the pixels predicted for a certain class were actually correctly predicted. IoU gives a better measure of performance for segmentation labels as it penalizes false positive and false negative predictions. Through the analysis of the networks trained, while the other metrics are considered, mean IoU and per class IoU are given higher consideration

than the others since they provide a better idea of the quality of the label predictions and image segmentation.

$$\begin{aligned} IoU &= \frac{true\ pos}{true\ pos\ + false\ pos\ + false\ neg} \\ &= \frac{\Sigma_i n_{ii}}{t_i + \Sigma_j n_{ji} - n_{ii}} \end{aligned} \tag{5.2}$$

5.2 Precursor Hand Labeled Patch Method

At the inception of this research, before the roughness metric was fully developed, some precursor work was done on terrain classification. This work aimed to classify an outdoor terrain as traversable, partially traversable or non traversable. In order to do this, patches of terrain were extracted and hand labeled with the aforementioned classes. Two training methods, a convolutional neural network and a support vector machine, were trained using the image patches and the labels. The convolutional neural network used the raw images for training while the support vector machine used features extracted from a visual word representation of the image patch. The purpose of this work was to set a foundation for the research and investigate the performance of using a patch based method to classify an entire image.

5.2.1 Image Preprocessing

A sample of images from different bag files were taken. The images were resized from 1024x544 to 1024x576. Each image was split up into 144, 64 x 64 subimages as shown in figures 5.1 and 5.2. Each subimage was hand labeled with 0, 1, or 2 representing traversable, partially traversable or non-traversable respectively. A traversable area was a clearly defined off-road path and its surface was usually gravel, sand, or dirt. A partially traversable area was a rough surface such as larger rocks and the edge of the road. The non-traversable areas were everywhere else including but not limited to the sky, trees, and tall grasses.

A bank of 38 filters, shown in 5.3, comprising of Gaussian, Laplacian, X-derivative, Y-derivative, and Gabor filters of varying scale and orientation were applied to the each channel of the subimages. 100 randomly selected points from each subimageâĂŹs filter responses were used to create a 114 visual word dictionary using MATLABâĂŹs K-Means algorithm. Each subimage was then re-represented as a visual word map using the words from the dictionary. Examples of this are shown in figure 5.4. It can be seen from these examples that the visual words were able to capture semantic meaning of the image scene.

Features were then extracted from each image using a 3 layer spatial pyramid histogram as in [27]. The histogram counts the number of occurrences of each visual word at each level of the pyramid in an effort to capture the spatial structure of the image. This histogram was the representation of the image, and used to train the support vector machine.

Figure 5.1: Image of terrain.

Figure 5.2: Image of terrain split into subimages.

5.2.2 Learning Methods

As mentioned previously, two learning methods were considered, a support vector machine and a convolutional neural network. The support vector machine was implemented using the libsvm library and trained with the word map histogram feature extraction. The best result used a radial basis function kernel and resulted in an accuracy of 84.62%, and confusion matrix:

$$\begin{bmatrix} 110 & 2 & 3 \\ 13 & 11 & 16 \\ 14 & 0 & 143 \end{bmatrix}$$

The second method used was a 21 layer convolutional neural network designed with inspiration from AlexNet, [28], using the raw images as input. This resulted with an accuracy of

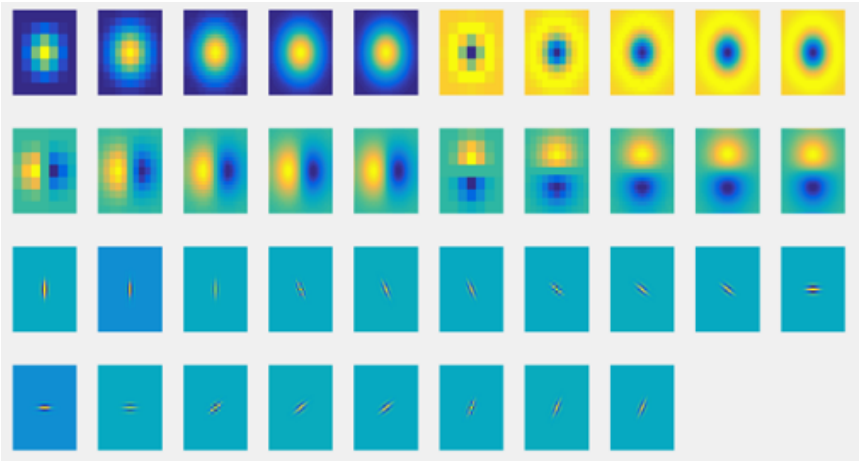


Figure 5.3: Bank of filters applied to the subimages.

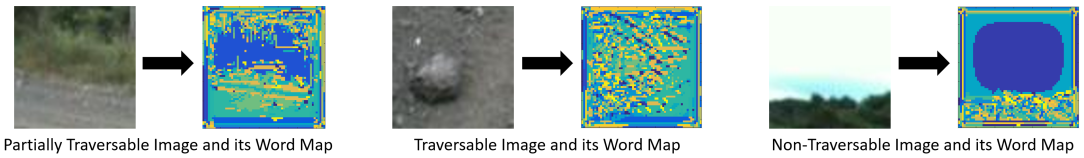


Figure 5.4: Examples of image patches and the corresponding wordmap representations.

87.18% and confusion matrix:

$$\begin{bmatrix} 109 & 5 & 1 \\ 8 & 19 & 13 \\ 6 & 7 & 144 \end{bmatrix}$$

The main improvement from the CNN was the enhanced classification of the partially traversable images. However from both confusion matrices, it is clear that the most misclassified images were the partially traversable images which contain visual elements from both types of images. This is understandable because the partially traversable areas straddle the boundary between the traversable and non traversable areas. Some of the misclassification can be attributed to the significantly fewer, partially traversable images in the training set, than traversable or non-traversable images. These issues were noted and in later iterations of the learning system, remedied by using a balanced data set and median frequency weighting.

5.3 Convolutional Neural Network using a Patch Based Method

Following on the heels of the success of the patch based method in the precursor work, our initial attempt at learning the IMU roughness metric emulated the patch based method. For each of the labeled image, samples of patches, each 64x64 pixels, were taken within the 268x724 pixels region bordered in red in 5.5. Since the patches were 64x64 we did not want to sample the area 32 pixels from the bottom on the image. If this were to be done, a technique such as padding would need to be applied, to ensure that patches sampled within that region would not

fall outside the image. We felt that this would not be a wise idea since there would introduce patches in the dataset which did not contain a true representation of the terrain, and consequently could introduce irregularities in training and hence lead to poor performance.

As it can be seen from figure 5.5, the path is not a rectangular region but rather looks like a trapezoid. If the robot turned, the path would also curve in the same direction, and the labels would have a curved trapezoidal shape. Therefore during sampling, there is the possibility of the pixels in the 64x64 patch not containing any labels, containing some labels or all containing labels. To deal with this, the number of labeled pixels in each patch was counted, and if at least 60% the patch were labeled then the patch was accepted. The label was then calculated as the average of all the labels of the labeled pixels of the patch. Since we were a calculating the label as an average, this label was assigned to the center pixel of the patch. Figure 5.5 shows the area of the image which was sampled in red, and some example patches in yellow which were accepted. The patch, along with its label, velocity, and the position of the center of the patch were saved.

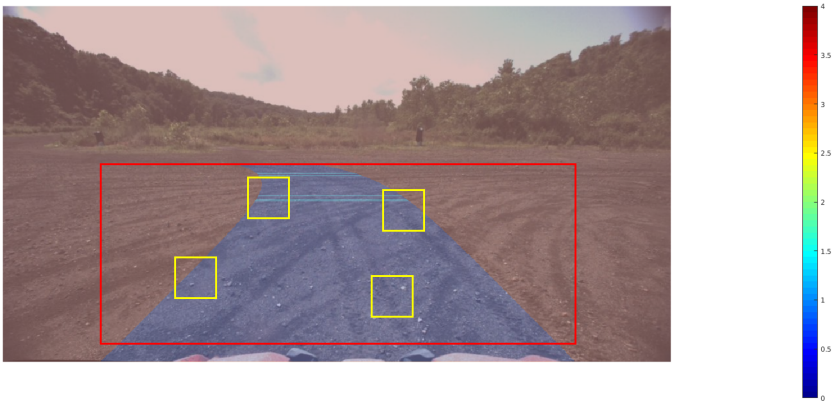


Figure 5.5: The patch sampling area shown in read, and some example patches.

In order to model the roughness of the image patches, the proposed method was to feed the image patches along with their labels through a deep convolutional neural network which would be able to learn the features of different terrain and map it the particular roughness value. Since the label was assigned to the center pixel, it was hoped the networks would learn how the area around a pixel would influence its roughness. It is important to note that roughness is dependent on the speed that the robot is moving. The faster the robot moves over a particular piece of terrain, the rougher it will feel, and this will be reflected in the roughness measure. Another important point, is that the position of the patch is important. A rough patch further in front the robot, for example with large rocks, would look similar to a patch close to the robot with small pebbles. As such the velocity, and position of the patch (x,y) would have to be included in the network.

It was theorized that by learning the roughness for image patches, the network would overcome the limits of the assumption that all the terrain under the robot was of constant roughness, and then be able to generalize to the terrain that was not traversed by the robot. First the patch based method was used along with a convolutional neural network trained for regression. After producing poor quality results, the method was switched to training the convolutional neural net-

work for classification. This section describes the methodology and results of both of these sets of experiments.

5.3.1 Regression Trained CNN

Three deep convolutional neural networks were used to model the data for regression. These models are VGG16 (with the pretrained weights), [29], Cifar10, and a custom designed network, called terrainNet as shown in figure 5.6. VGG16 and Cifar10 were both used for image classification, so the last layer of each of these networks would be modified to a fully connected layer to do regression.

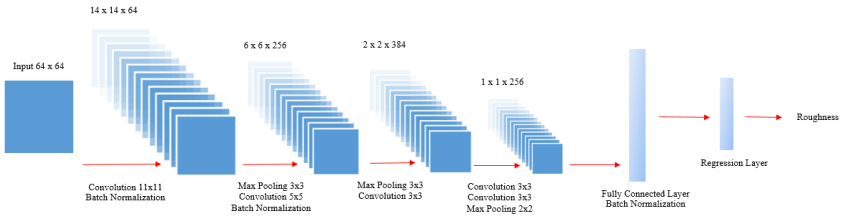


Figure 5.6: TerrainNet convolutional neural network model.

It is important to note here is that at this point in time in the research, since we were dealing with solving a regression problem, the images were labeled with the IMU roughness metric calculated from the simple moving average of the energy of the IMU signal. This version of the IMU roughness metric was described in section 3.1.1, and shown in figure 3.4. After a patch was selected, the label assigned to the patch was the average of the labels of the pixels of that patch.

For the preliminary results, it was decided to disregard the speed and position inputs, and just train the network using the images and roughness measurement. This was done to check the baseline performance on the custom convolutional neural network. The network was trained for 1000 epochs using the standard parameters of the Adam optimizer, and mean square error loss. The training and validation loss curves are shown in figures 5.7 and 5.8 respectively.

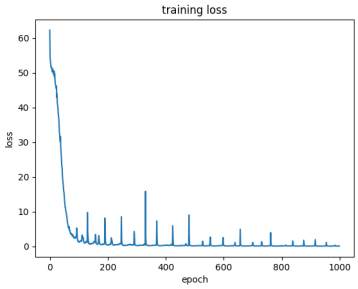


Figure 5.7: Training loss.

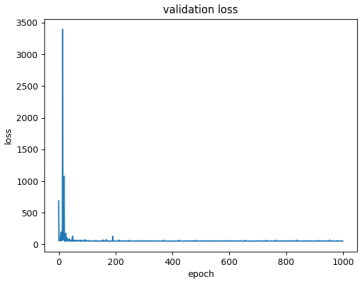


Figure 5.8: Validation Loss.

As it can be seen the training loss started off very large but eventually decreased to a value of around 0.15. The general curve looks typical, but there were strange spikes. The validation loss curve looked even more peculiar. There were a lot of large spikes, and it settled to a value

of around 53. This most likely meant that the model overfit to the training data, and quickly lost its generalization capability. Figure 5.9 shows the result of the prediction.

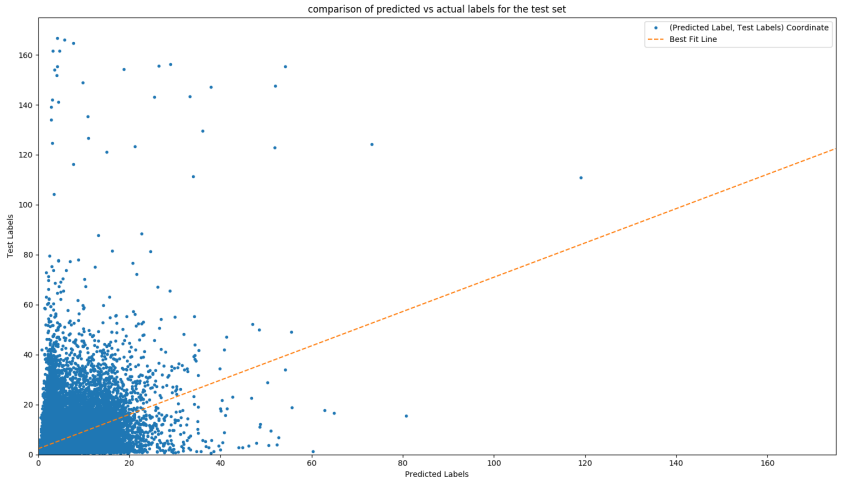


Figure 5.9: Plot of the test labels vs the predicted labels and the best fit line through the points.

Figure 5.9 shows the plot of the predicted labels against the test labels. If the predictions were close the to the test labels, the points would roughly lie along a line whose gradient is equal to 1, and have a mean square error close to 0. However as it can be seen the points were quite dispersed, and the gradient of the best fit line through them is 1.46. The mean square error of the predicted labels were 58.16. This clearly means the predictions are not good.

As previously discussed, the velocity the robot, as well as the position of the patch in the image are important pieces of information which could help improve the performance of the network. These were just concatenated onto the input of the last dense layer in the network. TerrainNet as well as VGG16 and Cifar10 were trained all with different optimizers for 20 epochs. This was done to see if there would be any sort of improvement. The results are summarized in table 5.1.

These tests revealed some important information. The inclusion of the position of the image patch, as well as the velocity helped with the stability of the network. There were no spikes in the training loss curve, as was seen previously. The validation curves for terrainNet were still noisy. Though it was not as noisy as the initial curve shown in figure 5.8, the curves did not always follow the typical validation loss curve, where the loss decreases per epoch, and there were still spikes in the curve though not as large as in figure 5.8. However, the validation curves for VGG16 and Cifar10 were much more stable and looked liked the typical validation curve that was expected. The best performing network, that is, the one with the lowest mean squared error was Cifar10 with the Adadelta optimizer. The prediction plot for this network is shown in figure 5.10.

While this plot looks better than the one in figure 5.9, the values are still clustered to the bottom left quadrant of the graph. As mentioned previously, if the predicted labels were close to the true labels, the points should lie along a line and mean square error should be close to zero. This is clearly not the case here so it was evident that the network was not learning well. The nature of solving a novel regression problem, made it difficult to determine if the a particular

Table 5.1: Summary of training the different networks for regression with different optimizers

Network	Optimizer	Gradient of best fit line	Mean squared error
terrainNet	Adam	1.088	483.63
terrainNet	RMSprop	0.969	41.24
terrainNet	Adagrad	1.010	45.98
terrainNet	Adadelta	0.909	48.43
terrainNet	Adamax	0.923	84.75
terrainNet	Nadam	0.409	84.75
vgg16	RMSprop	1.024	42.71
vgg16	Adam	1.094	38.82
cifar10	RMSProp	1.060	39.70
cifar10	**Adadelta**	1.092	**35.69**
cifar10	Adamax	1.077	37.78
cifar10	Adam	1.065	37.54
cifar10	Nadam	**1.032**	37.70
cifar10	Adagrad	0.252	160.97

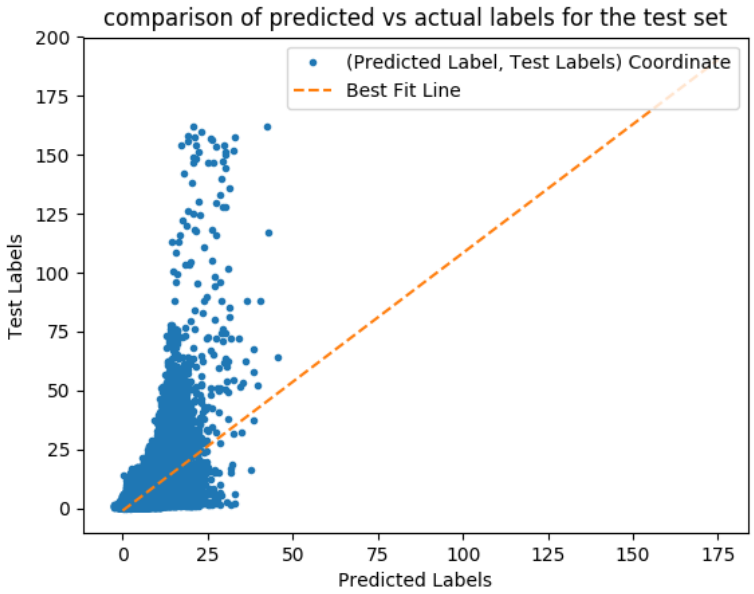


Figure 5.10: Prediction plot for the cifar10 network with adadelta optimizer

mean squared error loss was a good value or not. From the results it was clear that solving this regression problem was proving to be very difficult. The patch based method described in 5.2 worked well for classification. Furthermore, it was not necessary for the robot to know an exact value for roughness. Having different classes which represented degree of roughness within a certain range would be just as informative. As such the decision was made to switch to solving a

classification problem instead of a regression problem.

5.3.2 Classification Trained CNN

Programming-wise, the conversion to switching to solving a classification problem simply involved changing the last layer of each network to a classification layer, that is, a softmax layer the size of the number of classes. Now instead of having a label as continuous value, the labels were categorized as classes from 0-35 as described in section 3.1.2, and shown in figure 3.5. It was ensured that there were the same number of patches each class so that the dataset was balanced.

There were a couple of reasons to switch to classification. Firstly it would be more intuitive to understand the performance of the network. Secondly, it is not necessary to know the exact value of the roughness. The simple moving average IMU roughness values ranged from 0-160. A value of 70 is just as bad as a value of 50. Therefore taking all the values of roughness greater than 35 and placing them all into one class would not come at a loss to the robotic system, since it would just simply be understood that that area of terrain is very rough. The benefit gained is that the scope of learning reduces for the network.

5.3.2.1 Selection of the Best Network for Classification

To approach this in a systematic manner, the best performing network out of terrainNet, VGG16 and Cifar10 needed to be chosen. To select this, each network was trained with just the image patches and labels to get the best performance. First terrainNet was trained with the Adadelta optimizer for 20 epochs and achieved a validation accuracy of 12.2% with a validation loss of 7.30. Then terrainNet was trained with the Adam optimizer which saturated at 128 epochs with a validation loss of 10.71 and a validation accuracy of 14.73%. It was interesting to note here that with the Adadelta optimizer, the network achieved similar results to the Adam optimizer, but for less epochs.

Cifar10 and VGG16 both eventually saturated to low training and validation accuracies even with different optimizers. After 300 epochs, the Cifar10 network saturated with a training accuracy of 17% and a validation accuracy of 9%. VGG16 saturated after 3 epochs with a training accuracy of 2.77% and a validation accuracy of 2.75%. This was no better than chance. Cifar10 and VGG16 were designed for object recognition. The objects would be distinct from one another, whereas the terrain in this data set is similar with minor differences. As such these networks may not be best suited for the terrain recognition domain. As such it was determined to use terrainNet from here on since it produced better results.

5.3.2.2 Selection of Data Fusion Method

Now that the most suitable network was chosen, some investigation needed to be done into whether the concatenation of velocity and image patch position actually improved results. terrainNet network was chosen with the Adam optimizer using default values and this was trained on the balanced data for 20 epochs with batch size 512. Using the image alone, without any concatenation gave a validation loss of 3.67 and a validation accuracy of 9.31%. Using the images

and velocity concatenation gave a validation loss of 3.29 and a validation accuracy of 10.71%. With only the image and patch position concatenation, it gave a validation loss of 10.26 and a validation accuracy of 5.73%. Using both image patch position and velocity gave a validation accuracy of 5.15% and a validation loss of 10.21. Clearly using just the velocity for the concatenation gave the best results. However the model using no concatenation also gave decent results.

To compare against the performance without any concatenation, versus with just velocity concatenation effectively, k-fold cross validation would be used with k=5. terrainNet was trained using the Adadelta optimizer with a learning rate of 0.01 for 150 epochs. The results are shown in table 5.2.

Table 5.2: Summary of 5-fold validation results for using image and velocity concatenation vs no concatenation

Image and Velocity			Image Alone		
Fold	Loss	Accuracy	Fold	Loss	Accuracy
0	7.122412	0.114269	0	6.516335	0.094351
1	6.076258	0.116021	1	6.146654	0.125570
2	6.465306	0.124236	2	6.888288	0.110322
3	6.147904	0.130325	3	6.217765	0.125919
4	6.104640	0.113604	4	6.120966	0.122083
Average	6.383304	0.119691	Average	6.378002	0.115649
Std Dev	0.395057	0.006537	Std Dev	0.291476	0.012060

In summary, training the network with the velocity concatenation slightly outperforms the one without the concatenation. The losses are very similar, but the accuracy of the network with the concatenation is 0.4% better than the one without. Intuitively it does make sense that the concatenation of the velocity would improve the results of the network. The roughness is dependent on the speed at which the robot moves over the terrain. The 5-fold tests confirmed this. However, it was expected the difference in performance to be much larger. Nevertheless, velocity concatenation was chosen as the method for data fusion moving forward.

5.3.2.3 Final Results for Patch Based Method

After more than 40 tests, terrainNet was chosen as the best performing network to train the data. It was trained on 360,000 images with velocity concatenation, using the Adadelta optimizer. The first 100 epochs of the network was trained with the default learning rate of 1.0. The next 100 epochs were trained with a learning rate of 0.0001. The default learning rate was high and allowed rapid learning, however it would often cause the gradient to be too high in a particular direction. As a result the training was a bit unstable. As such, after the initial training of 100 epochs, the second training of 100 epochs used a lower learning rate to fine tune the network to settle on stable values.

This gave an accuracy of 25.7% which was the best performance so far. The confusion matrix is shown in figure 12. As it can be seen in figure 5.11, the diagonal is prominent indicating that there were many correct predictions. However, there are two points worth noting. Firstly each class has 2000 test images. Secondly the scale of the colour bar to the right is important. The brightest yellow indicates a number of 2000. So while the confusion matrix looks good, there is still quite a bit of misclassification occurring. However it is worth pointing out that a lot of the misclassification is occurring in classes nearby the true class, so there is a general clustering around the diagonal of the confusion matrix.

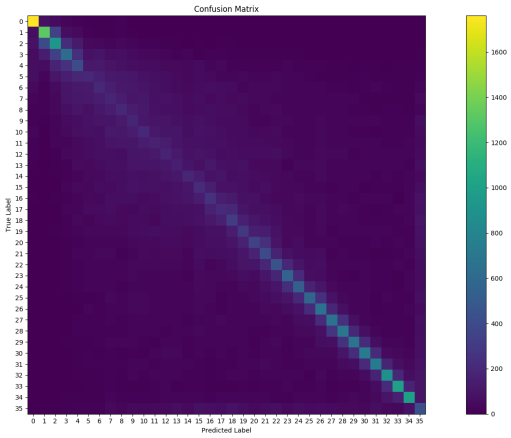


Figure 5.11: Consfusion matrix for patch based method CNN

After some introspection and discussion it was determined that 36 classes of roughness was more than enough that is needed by a robotic system to determine roughness. In other words, it is not necessary for a robotic system to know the exact value of the roughness for path planning or adaptive control. Basically the robotic system would just need to know which areas are traversable, partially traversable or non-traversable. This alludes to the precursor work done which labeled the patches with these type of labels. As long as the roughness it predicts is within the correct neighborhood, the system would perform fine. For example roughnesses of 32 and 36 are similar since on a high level, they indicate there is a very rough area, and this information is adequate for a robotic system. As such it is okay to allow some degree of inaccuracy for the roughness predictions. If we allow a misclassification of ± 1 classes, the accuracy is 42.7%, ± 2 classes, the accuracy is 51.5%, ± 3, the accuracy is 58.1% and ± 4 classes the accuracy is 63.1%.

With this in mind, the labels were recategorized as shown in 3.1. With these re-categorized labels, analysis on full image prediction was done. In order to do this, for each ground truth label, a list of the pixels which were labeled was obtained. A patch of 64x64 around each of these pixels was taken and then sent to the network for prediction. Care was taken to ensure that prediction and analysis was done only for the area for which the samples were taken, meaning that the area 32 pixels from the bottom of the image was not considered for prediction and analysis. This was done for all the training images and the results shown in table 5.3.

Table 5.3: Results for full image prediction for patch based CNN

			Per Class IoU			
Global Acc	Class Avg Acc	Mean IoU	Class 0	Class 1	Class 2	Class 3
0.52936	0.43238	0.26925	0.667	0.1659	0.0992	0.1452

5.4 Full Image Sematic Segmentation

After obtaining the results for the patch based method, we decided that a change of approach was needed. The disadvantage of using the patch based method was that spatial information was lost. Given an outdoor image, the terrain, and by extension roughness, is generally consistent within an area. It is rare that there will be a sudden change in roughness. Therefore this means, that the roughness of an image patch is influenced by the roughness and terrain of the image patches within its neighborhood. By training a convolutional neural network to learn roughness of an individual image patch, spatial correlation with nearby pixels or image patches is not being considered. Consequently potential valuable information about the terrain and labels that can be gained from these neighboring patches and pixels is being lost. This could have possibly led to the sub-par performance of the patch based method. As a result full image semantic segmentation was the next method that was considered.

Full image segmentation was implemented with good performance in [16]. More importantly a fully convolutional neural network called SegNet, [18], was used as a benchmark, and produced comparable results on the summer dataset used in [16]. As such SegNet was chosen as the network to train for full image segmentation using our dataset. SegNet's architecture, code, written in caffe-segnet, and pretrained weights were provided for public use. Additionally SegNet contained a useful feature whereby a certain class could be ignored, which from our understanding meant that that class would not be learned and subsequently it would not be able to affect the learning of the other classes. This is an important feature which we wanted to leverage since our dataset only contained labels for a specific part of the image.

Since the classes in the dataset were unbalanced, some form of class weighting needed to be applied. The code provided by the authors of SegNet allowed a user to easily set weights for class balancing without the user having to write any sort of custom loss function. All of these features made SegNet an attractive option for training our dataset. However, SegNet was trained using images of size 360x480, while the images in our dataset were 1024x544, about three times as larger. With an image size of 1024x544, it would be difficult to fit a large batch size along with the model and weights onto a GPU. As such the images and labels would either need to be resized or split up into smaller images.

5.4.1 Class Balancing

Since there was a large variance in the number pixels for each class in the data set, then class balancing needed to be performed by assigning weights to each class. These class weights are applied in the cross entropy loss function, so that the loss is weighted differently based on the

true class. For this dataset, median frequency class balancing was used as in the original SegNet paper, [18], which obtained the best results using this method. Median frequency balancing was introduced in [30] and defined according to equation 5.3, where $freq(c)$ is the number of pixels of class c divided by the total number of pixels of the images where class c occurred, which is equivalent to the number of images where c occurred multiplied by the size of the image. $median\_freq$ is the median of these frequencies [30].

$$\alpha_c = \frac{median\_freq}{freq(c)} \tag{5.3}$$

5.4.2 Training IMU Roughness Metric

5.4.2.1 Model Selection

In order to deal with the image size issue, the first idea was to split up the 1024x544 full image into 4, 264x136 images. The same would be done for the labels. SegNet would be modified to work with this size image, and then trained. We also first decided to train with just the images and labels, while ignoring the no information class, without any velocity data input, and with pretrained weights, in order to select be best setup for the model. This first model called SegNet Split was trained with stochastic gradient descent, a learning rate of 0.001 and momentum of 0.9 for 40000 iterations. The results are shown in row 1 of table 5.4. Immediately we can see a performance increase from the patch based method, shown in table 5.3. However for the class 2 and 3, which were the rougher classes, the performance was underwhelming.

The next option to deal with the image size issue was to resize the full images and labels from 1024x544 to 360x480. The process was done using the nearest neighbor method. This model, shown in row 2 of table 5.4, called SegNet Resized, was trained with the same parameters as SegNet Split. As it can be seen it achieved much better performance than SegNet Split. This could be attributed to a same reason the patch based method did not perform well. By splitting up the full image into the 4 smaller images, it has the same effect of sampling image patches, in that the spatial information was lost. Due to the fact that resizing produced better results, we continued the rest of experiments with the resized images and labels. In an effort to determine if the best optimizer was being used, the model was then trained with the Adadelta and Adam optimizer respectively using the default learning rate. The results are shown in the last two rows of table 5.4.

Table 5.4: Performance on the validation set of the different models trained.

				Per Class IoU			
Model	Global Acc	Class Avg Acc	Mean IoU	Class 0	Class 1	Class 2	Class 3
SegNet Split	0.76664	0.37302	0.2701	0.7996	0.2808	0	0
SegNet Resized	0.77484	**0.53356**	**0.38516**	**0.8214**	**0.2852**	**0.1442**	**0.2898**
SegNet Adam	**0.7924**	0.25	0.1981	0.7924	0	0	0
SegNet Adadelta	0.7807	0.34587	0.26795	0.8083	0.2411	0.0183	0.0041

From the table it is clear to see that the best performing network was SegNet Resized. For image segmentation the optimal measure of performance is intersection over union (IoU) and mean IoU. SegNet Resized had the best performance for mean IoU and for all the per class IoU's. Therefore the parameters chosen for the network were input image size of 360x480, stochastic gradient descent with a learning rate of 0.001, momentum 0.9 and 40000 iterations.

5.4.2.2 Selecting the Best Filtering Method

With the optimum model chosen, the next task was to test the network. It is important to note that at this point in time, the IMU roughness labels were generated by applying a simple moving average filter to the energy of the IMU signal. The test results for this model is shown in the first row of table 5.5. Another noteworthy points is that the no information class (class 4) was chosen to be ignored during training. As such we were curious to see what would be the effect if the no information class were included in training. Therefore another network was trained, again with the same parameters previously used, but included the no information class for training.

The results for training the network to learn all the classes is shown in row 2 of table 5.5 as Simple Moving Avg + No Info. The global accuracy and mean IoU increased from training the model without class 4. However upon further inspection, this increase was solely due to the fact that class 4 IoU was near perfect. The per class IoU for classes 0-3 were all much lower than the model trained without class 4. This high per class IoU of class 4 skewed the mean IoU to be higher than the one for the simple moving average model trained without class 4. In general, all the labels are clustered around the center of the image, so it is very likely that the network is learning this pattern, and labeling all the pixels out of this area as the no information class. We are really only concerned with the roughness classes 0-3, and it is clear that the inclusion of the no information class is reducing the ability of the network to learn the roughness classes. As such it was decided to not train the network with the no information class.

Now just considering the model trained with the roughness classes generated from the simple moving average filter (excluding the no information class), we were interested in whether the lack of performance was due to the fact that the simple moving average filter introduced a slight delay between the labels and the actual signal. Such a delay could cause the label to be applied to the wrong part of the image. This would become very apparent if there were a rough patch of terrain amidst smooth ones. Therefore two different filtering methods, Gaussian and mean smoothing, were applied as described in section 3.1.3. The model was retrained with the labels derived from these two different types of filtering and the results shown in table 5.5.

Clearly it could be seen from table 5.5 that mean smoothing produced the best results out of all the filters (not considering the model trained with the no information class) in all the metrics except for class 3 IoU, where the simple moving average filter outperformed the mean smoothing filter. The delay introduced by the moving average filter would have most likely caused some misalignment between the terrain and the labels making it difficult for the network to properly learn the labels. The Gaussian filtering would have filtered the the IMU readings too much and possibly given smooth labels to rougher terrain in some instances, but at other instances given the similar type of terrain rough labels, again confusing the network. The mean smoothing remedied both of these problems and gave better results.

To find out where the mean smoothing filter network was under-performing, it was necessary

Table 5.5: Performance on the test set, of the models trained with labels from different filters.

				Per Class IoU				
Model	Global Acc	Class Avg Acc	Mean IoU	Class 0	Class 1	Class 2	Class 3	Class 4
Simple Moving Avg	0.77124	0.54692	0.37459	0.8182	0.2964	0.0876	**0.2962**	-
Simple Moving Avg + No Info	**0.91996**	0.51571	0.40898	0.7347	0.2566	0.018	0.0695	**0.9661**
Gaussian Smoothing	0.74519	0.47418	0.33325	0.7876	0.2719	0.1378	0.1357	-
Mean Smoothing	0.81094	**0.54997**	**0.41262**	**0.8601**	**0.3459**	**0.1927**	0.2517	-

to look at the confusion matrix, shown below:

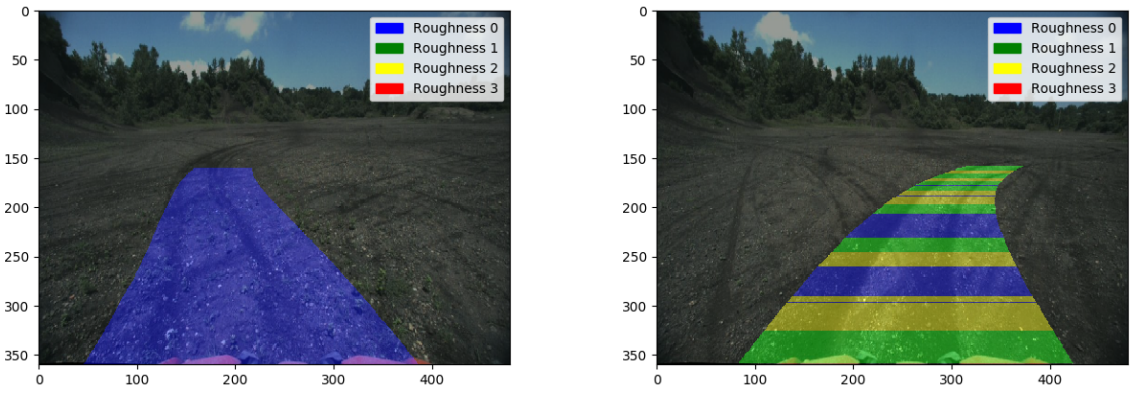
$$\begin{bmatrix} 0.8919 & 0.0994 & 0.0084 & 0.0002 \\ 0.1975 & 0.6266 & 0.1650 & 0.0109 \\ 0.0467 & 0.4846 & 0.3999 & 0.0689 \\ 0.0137 & 0.2369 & 0.4679 & 0.2815 \end{bmatrix}$$

The confusion matrix gave an idea of where the network is under-performing by showing which classes are being predicted as others. Class 0 is performing well. However class 1 is often being confused for class 0 or 2. Similarly class 2 is being mistaken for class 1. Class 3 is being confused for both class 1 and 2. Class 0 is predicted well since there are so many examples of this label. From looking at the dataset, it becomes apparent that the rougher classes (1-3) are being confused because velocity is not being taken into consideration. As mentioned in section 1.2, when the data was collected the robot was commanded to drive over the test course at speeds of 10, 20, 30 and 40 kilometers per hour. Consequently there are images in the dataset which are labeled as class 0 and class 1 at one when the robot drove slowly through the terrain, but at other times, the exact same terrain is labeled with different classes because the robot drove faster over the terrain.

Figures 5.12a and 5.12b show an example of how the IMU roughness labels of the terrain vary depending on the velocity of the robot. As such it is easy to see why such images and labels would confuse a network. Without a velocity input, the network would not be able to understand why the same image or terrain gets two different set of labels. Consequently it was necessary to find a method to incorporate velocity into the training.

5.4.2.3 Incorporating Velocity into Training

Since the robot was commanded to drive at four different velocities, 10, 20, 30, and 40 kilometers per hour it was decided to split up the dataset based on velocity. From the log files, the average velocity of the robot during the labeled path in the image could be calculated. All images and corresponding labels where the robot's average velocity was 0-10 kilometers per hour, were put into one dataset, and so on for 10-20, 20-30 and 30-40 kilometers per hour. These labels were the ones calculated from the mean smoothing filtering since that gave the best results previously. For each dataset, the class weights were recalculated and a separate network trained. These networks



(a) Robot velocity 9 km/hr.

(b) Robot velocity 19.1 km/hr.

Figure 5.12: Comparison of the IMU roughness labels based on robot velocity

all had the same parameters as before. For prediction, the appropriate model would be selected based on the average velocity of the robot for the labeled path. We called this approach SegNet Ensemble 10. Table 5.6 shows the performance of the network for this model.

Compared to the previous best performing network,SegNet with mean smoothing labels, the SegNet Ensemble 10 model outperformed it in every metric. It was clear that separating the images and labels based on the robot velocity allowed the robot to make a better differentiation between IMU roughness labels at different speeds. The confusion matrix for this network is shown below:

$$\begin{bmatrix} 0.9267 & 0.0703 & 0.0029 & 0.0001 \\ 0.1569 & 0.6996 & 0.1389 & 0.0046 \\ 0.0152 & 0.4073 & 0.5266 & 0.0508 \\ 0.0005 & 0.1069 & 0.5558 & 0.3368 \end{bmatrix}$$

It gives evidence that there is less confusion with the ensemble method especially when compared to the previous network. Class 0 performs almost perfectly. Class 1 performs a bit better but is still confused with class 0 or class 2, but to a lower amount. Similarly class 2 performs better and though it is still confused with class 1, it is confused less. Most importantly class 3 performs better, but mainly becomes confused with class 2. The confusion matrix shows an improvement in the results, and proof of less confusion occuring.

Based on this result we decided to train another ensemble set of networks called SegNet Ensemble 5. The procedure was similar as before except that the speed categories were 0-5, 5-10, 10-15, 15-20, 20-25, 25-30, 30-35 and 35-40 kilometers per hour. Its performance is shown in table 5.6. This ensemble performed similarly to SegNet 10. There was very little difference between the evaluation metrics except for the class 3 IoU, where SegNet Ensemble 10 outperformed SegNet Ensemble 5. Due to this, the mean IoU was slightly higher for SegNet Ensemble 10. The aim of ensembling was to get better predictions for the rougher classes. Therefore we concluded that SegNet Ensemble 10 was the better performing network since it

had a better mean IoU, and predicted class 3 better.

The similarity in the performance of both ensembles could be due to the fact that the labels within these finer ranges for SegNet Ensemble 5 were similar to each other. For example the labels in the dataset for 0-5 kmph and 5-10 kmph would be the similar labels. Therefore having separate networks for these datasets would be redundant and the weights learned would be close to the weights learned for one network encompassing both datasets.

Table 5.6: Performance of ensemble models on the test set.

				Per Class IoU			
Model	Global Acc	Class Avg Acc	Mean IoU	Class 0	Class 1	Class 2	Class 3
SegNet Ensemble 5	**0.86298**	0.61682	0.47933	**0.9115**	**0.4614**	**0.2726**	0.2719
SegNet Ensemble 10	0.85285	**0.62243**	**0.47949**	0.9021	0.4391	0.2606	**0.3161**
SegNet Ensemble 5 subIm	0.82923	0.56265	0.4213	0.8801	0.3835	0.2315	0.1901
SegNet Ensemble 10 subIm	0.81826	0.52445	0.39939	0.8677	0.3461	0.2024	0.1813

In an effort to elicit some better performance, we thought that due to the resizing, information was being lost, so it would be worthy to train the ensembles with all the pixels from the full image. However instead of using the entire image, the area of the image directly in front the robot was extracted. This area shown in the red rectangle in figure 5.13, was 300x726 pixels. Again two different ensemble models were trained, with the dataset prepared similarly to SegNet Ensemble 5 and SegNet Ensemble 10 respectively. The only difference was that the SegNet model itself was adjusted to accommodate the different size of input image. The results for these two ensembles SegNet Ensemble 5 subIm and SegNet Ensemble 10 subIm, are shown in table 5.6.

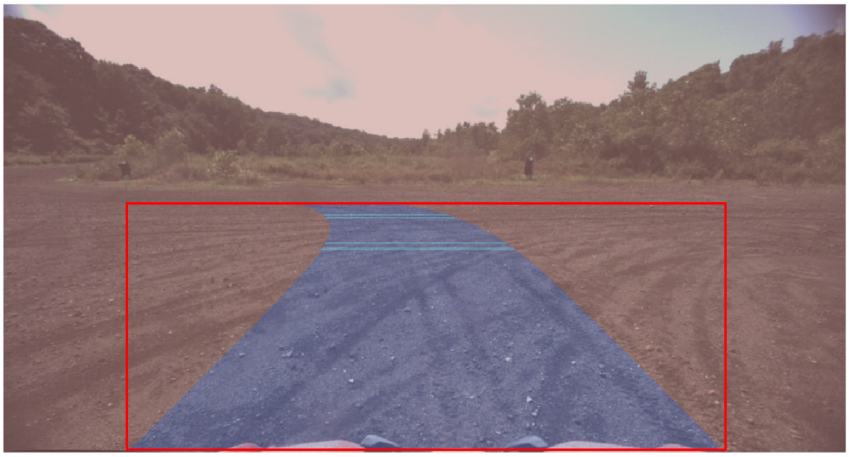


Figure 5.13: Area selected for training.

Interestingly SegNet Ensemble 5 subIm and SegNet Ensemble 10 subIm, did not perform as well as SegNet Ensemble 10. We have seen a trend that splitting up the full image by sampling patches, or extracting subimages does not seem to improve performance from simply resizing

the image. Again this could be due to the fact that by extracting subimages or image patches, valuable spatial information is lost.

Seeing the success of incorporating the velocity via ensembling, we wanted to find another method to incorporate the velocity as an input into the network itself. Unfortunately including such an input into the version of SegNet written in caffe-segnet was a difficult undertaking. However, Keras offered much more flexibility with inputs and network design. As such the decision was made to rewrite SegNet in Keras to provide the ability to modify the inputs and layers more easily.

However, this would come with some caveats. Keras did not provide a way to ignore a class as caffe-segnet did. Consequently we would have to train all the classes including the no information class. Additionally, we would have to write our own weighted cross entropy loss function, as Keras also did not provide a straightforward way of doing this. Lastly while the best attempt was made to emulate the design of SegNet in Keras, we were unable to create a similar upsampling layer to the caffe-segnet version. The upsampling layer in caffe-segnet used the max pooling indices to upsample the feature maps, [18]. To do this in Keras would involve modifying the backend code of Keras, which was beyond the scope of this research. As such we settled on using the built in upsampling layer in Keras which simply replicated the rows and columns of the feature map. Our network used the same parameters and image size during training as SegNet

Two methods of inputting velocity into the network were considered, both of which involved using a velocity mask. The velocity mask for an image had the same form as the labels. However the pixels which were labeled as class 4 (no information), were given a value of 0 in the velocity mask, and everywhere else which had a roughness label was given the value of the average velocity of the robot through the labeled path in the image. For example the velocity mask corresponding to figure 5.12b, is shown in figure 5.14. Please note that this figure is artificially colored for easy demonstration. The blue areas of this mask would be have a zero value, while the yellow areas would have a value of 5.306, which 19.1 kilometers per hour in meters per second.

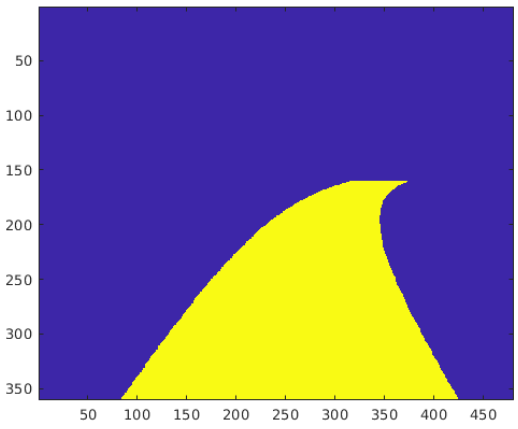


Figure 5.14: Velocity mask corresponding to figure 5.12b

The first method, called early stage fusion appended a velocity mask of size 360x480 onto

the RGB image to get an RGB-V input into the network. The second method, called late stage fusion involved adding the velocity mask to the last upsampling layer of the network. Since this network was slightly different than the original SegNet from [18], we also trained the network with just the images and labels to get a benchmark for performance. The results are for these three networks are shown in table 5.7.

Table 5.7: Performance of the SegNet-Keras models on the test set.

				Per Class IoU				
Model	Global Acc	Class Avg Acc	Mean IoU	Class 0	Class 1	Class 2	Class 3	Class 4
SegNet-Keras	0.79268	0.38364	0.22738	0.0019	0.1239	**0.0388**	0.0003	0.9719
SegNet-Keras Early Fusion	0.8	0.40756	0.24319	0.0072	0.1281	0.0352	**0.0514**	0.994
SegNet-Keras Late Fusion	**0.84157**	**0.41471**	**0.28424**	**0.2493**	**0.1666**	0.0098	0.0004	**0.9951**

Unfortunately these networks drastically underperformed all of the previous good performing networks. We theorize the lackluster performance is due to the fact that our upsample layer is not the same as the upsampling layer in the original SegNet. It must be noted that in the SegNet paper, [18], the authors also did state that upsampling via replication (the method by which Keras implements upsampling) performed quite poorly. However the main revelation from this experiment is that using late stage fusion significantly improved the result when compared to not using velocity fusion at all or early stage fusion.

5.4.2.4 Qualitative Results

Figures 5.15-5.22 show a visual comparison of the performance of the patch based results (Patch Prediction), the SegNet model trained without taking velocity into consideration (SegNet Prediction) and SegNet Ensemble 10 which accounted for velocity (SegNet Ensemble Prediction). These qualitative results support the quantitative results as SegNet Ensemble can be seen to have the best performance.

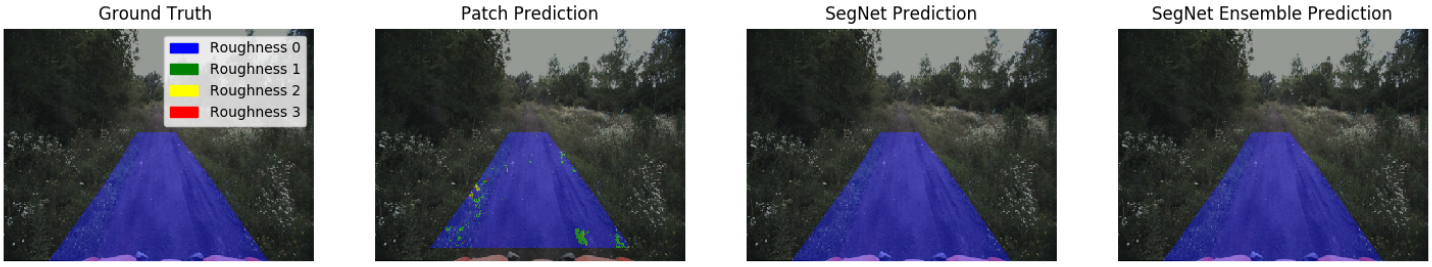


Figure 5.15: IMU roughness metric qualitative results 1.

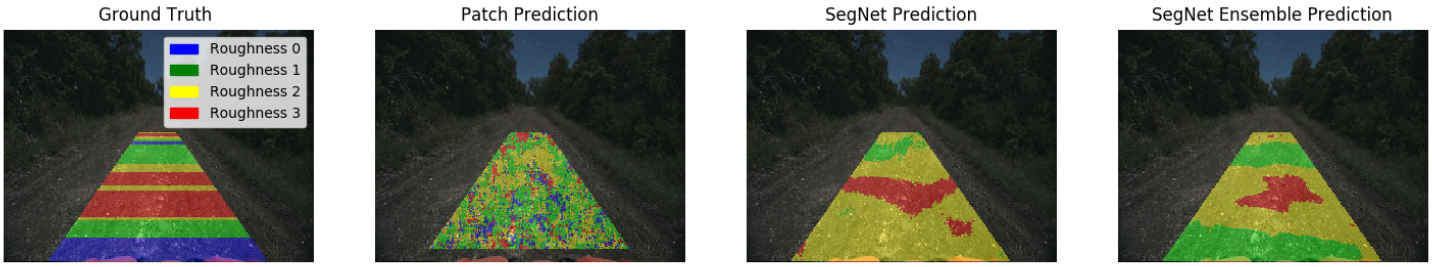


Figure 5.16: IMU roughness metric qualitative results 2.

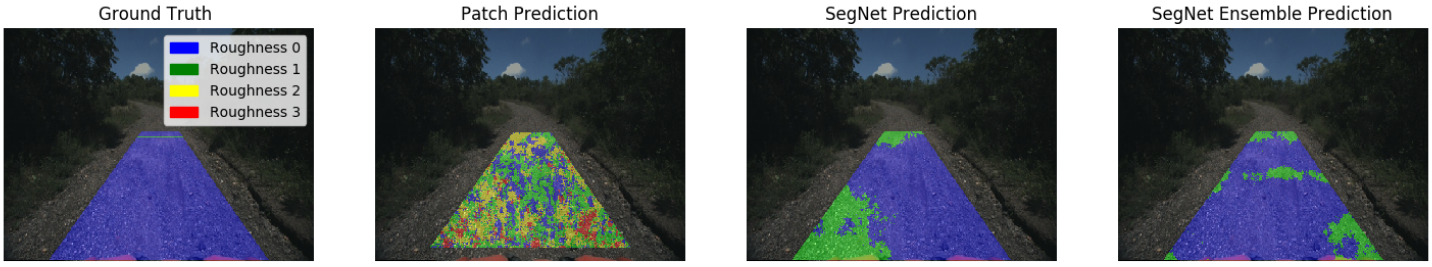


Figure 5.17: IMU roughness metric qualitative results 3.

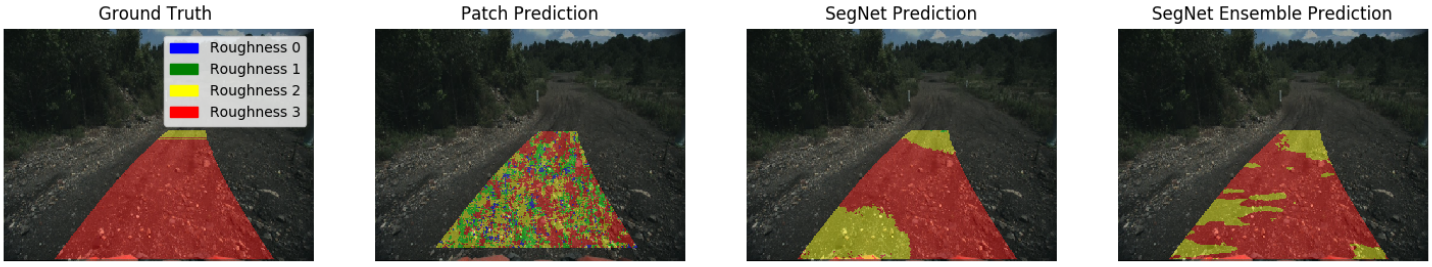


Figure 5.18: IMU roughness metric qualitative results 4.

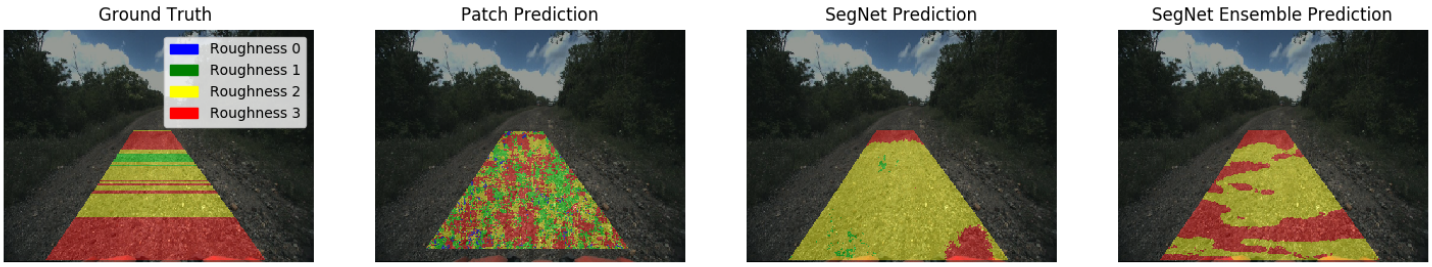


Figure 5.19: IMU roughness metric qualitative results 5.

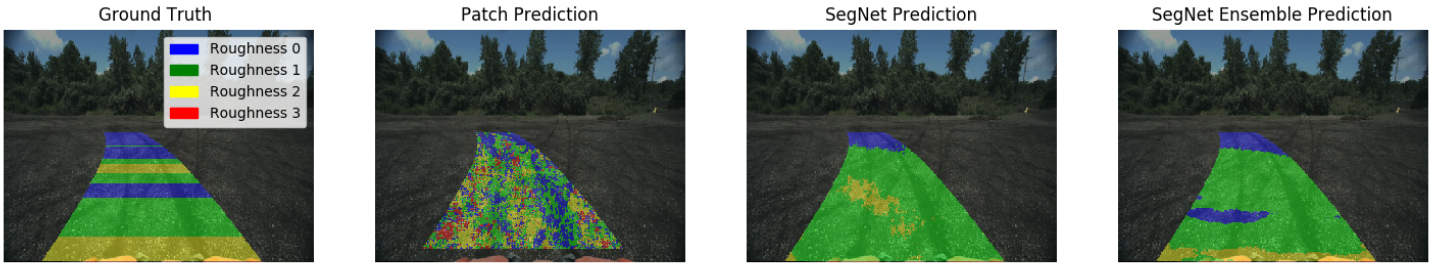


Figure 5.20: IMU roughness metric qualitative results 6.

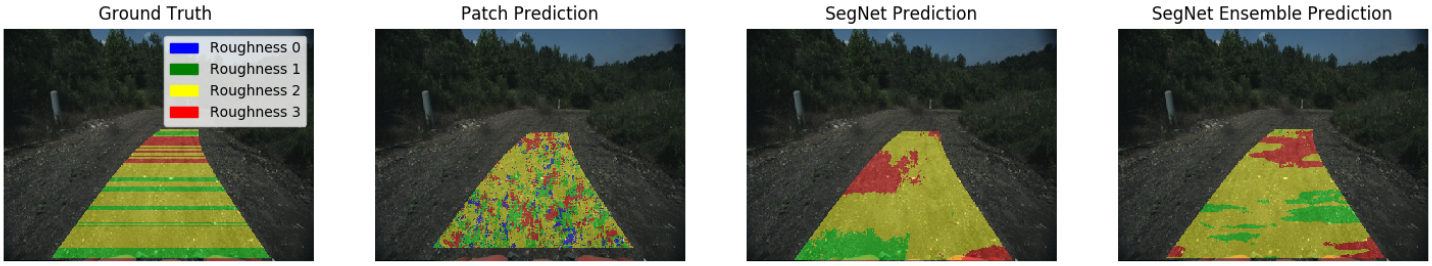


Figure 5.21: IMU roughness metric qualitative results 7.

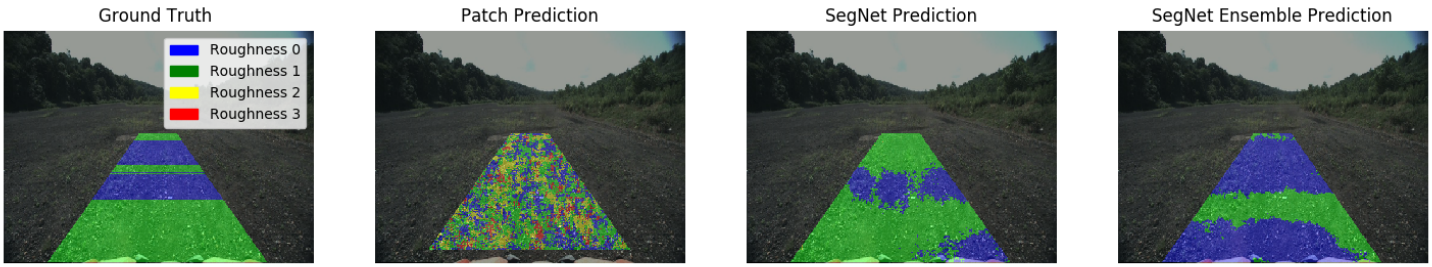


Figure 5.22: IMU roughness metric qualitative results 8.

5.4.3 Training Point to Plane Distance Roughness Metric

With the networks trained on the IMU roughness metric, the next step was to train the networks with the point to plane distance roughness metrics. From section 3.2.4, the max point to plane distance and absolute point to plane distance roughness metrics were chosen as viable candidates to describe terrain roughness. With the success of SegNet for the IMU roughness metric, SegNet was again chosen as the network to train using the point to plane dataset. Using the same model parameters, SegNet was trained using the absolute distance and max distance roughness metric labels.

As it can be seen from table 5.8, the max point to plane distance roughness metric trained network outperformed absolute point to plane roughness metric network especially in the rougher classes. These two networks were trained without the no information class (class 4). Therefore we were interested to see how the performance would change if the no information class were included in training. This model is shown in table 5.8 as Max Dist + No Info.

With the inclusion of the no information class (class 4), global accuracy, class average accuracy and mean IoU all improved from the max distance model. However it is important to note that per class IoU of class 4 for this model was almost perfect. Such a high performance offset all the other averaging metrics. With image segmentation the better measure of performance is the IoU for each class. It can be seen that the per class IoU for all the roughness classes (class 0-3) for the model trained with the no information class are all lower than the model trained without the no information class. This is similar to the performance of the IMU roughness metric when a similar experimental comparison was conducted. As we did in that case, we make the same conclusion here. We do not care about the no information class, and its inclusion in training brought no improvement to the model.

Lastly in an effort to elicit some better performance for the max distance roughness metric, we decided train the network with the 300x726 sub image of the full image, shown in figure 5.13, without the no information class. The result in shown in table 5.8 as the model named Max Dist Sub Img. Interestingly we see here that there was a slight increase in performance of this network over the network trained with the resized images. This contrasted with the IMU roughness metric which showed a decrease in performance when the sub images were used for training. In particular this network had approximately 1.1% higher mean IoU performance than the max distance model. This can be attributed to the increase in per class IoU performance for class 0 and class 3. The per class IoU performance for class 1 and class 2 were comparable for both networks.

We declare this to be the best performing network for the point to plane roughness metric. Though the Max Dist + No Info model achieved the best results for the global accuracy, class average accuracy and the mean IoU, we chose to not consider this network since the high values for the metrics were due to the large contribution of the class 4 (no information class) which skewed the results.

The point to plane distance roughness metric labels were generated from the pixels themselves. By resizing the full size image from 1024x544 to 360x480, information is lost and therefore could the reduction in performance that was observed. By taking the 300x726 sub image of the full size image, no information is lost, and the network can learn the roughness labels much better. In contrast, the labels for the IMU roughness metric were not generated from the pixels

Table 5.8: Performance of the SegNet models on the Point to Plane Roughness Metric Test Set.

				Per Class IoU				
Model	Global Acc	Class Avg Acc	Mean IoU	Class 0	Class 1	Class 2	Class 3	Class 4
Abs Dist	0.78424	0.45537	0.20451	0.7852	0.0217	0.0071	0.0041	-
Max Dist	0.59713	0.55028	0.29513	0.5764	**0.261**	**0.0871**	0.256	-
Max Dist Sub Img	0.61756	0.54464	0.30679	**0.6164**	0.245	0.0839	**0.2818**	
Max Dist + No Info	**0.87724**	**0.62713**	**0.39034**	0.5132	0.2241	0.0587	0.1956	**0.9601**

but were a function of the IMU and the robot velocity so using the sub image method would not have lead to an improvement as was seen with the point to plane distance roughness metric.

5.4.3.1 Qualitative Results

Figures 5.23- 5.30 show a comparison of the output of the best performing point to plane distance metric model, Max Dist Sub Img model, with the ground truths. From these visual results we see a pattern with the predictions. The terrain close the robot generally gets predicted as class 0, the area far from the robot gets predicted as class 2, while the area in between these distances gets predicted as class 1.

We expected to achieve better results using this metric as opposed to the IMU roughness metric since this metric hasd less variables involved and less assumptions made, however we found the opposite. While point to plane distance is a good metric for roughness, it is dependent on the accuracy, fidelity and performance of the hardware, that is, the image sensor. When compared to a high performing 3D lidar such as a Velodyne 64, the Multisense S21 has a limited range, resolution and accuracy. To compute terrain roughness via point to plane distance, we need a dense point cloud which can capture a good representation of the terrain. One possibility for the lack of performance is that point cloud generated from the depth map of the Multisense did not give the most accurate representation of the terrain, and hence so too would the roughness labels. This could account for some of the inconsistencies we noticed with the labels. Using a 3D lidar such as a Velodyne could give better labels and thus improve the results. However, the use of a 3D lidar introduces other complications computation wise such as point to pixel registration.

Another source of error could be the labeling. We used a labeling scheme similar to the IMU roughness metric to enable some comparison to be done with the point to plane distance metric. We suspect that the point to plane roughness metric could benefit from have a larger number of classes to give a better description of the terrain. Using smaller number of classes as we did could lead to inconsistent labeling, thereby confusing the network.

It may also be possible to improve the performance of the network by including the depth map as a secondary input. When generating the point clouds we noticed that the point cloud became more sparse further away from the camera. Clearly this could affect the plane fitting, and hence the point to plane distances. One solution as mentioned previously is to use a higher fidelity sensor. However it may also help to include the depth map as an input during training to

give the network a better understanding of the semantics of the terrain.

Figure 5.23: Point to plane roughness metric qualitative result 1.

Figure 5.24: Point to plane roughness metric qualitative result 2.

Figure 5.25: Point to plane roughness metric qualitative result 3.

Figure 5.26: Point to plane roughness metric qualitative result 4.

Figure 5.27: Point to plane roughness metric qualitative result 5.

Figure 5.28: Point to plane roughness metric qualitative result 6.

Figure 5.29: Point to plane roughness metric qualitative result 7.

Figure 5.30: Point to plane roughness metric qualitative result 8.

Chapter 6

Conclusion and Future Work

The aim of this thesis was to create a terrain roughness estimation system trained with images autolabeled with a roughness metric. To achieve this goal, the work was split into three tasks: develop a roughness metric, design an autolabeling system that would take images from a log file and label them with the roughness metric, and then train a deep learning network to learn the how to predict the roughness of terrain given an RGB image.

Two roughness metrics, an IMU based roughness metric, and a point to plane distance roughness metric were developed. Each of these had their benefits and drawbacks. The IMU based roughness metric could be cheaply computed, and did not require sophisticated or expensive hardware. However, due to the fact that the IMU based roughness metric calculated roughness based on the vertical motion of the robot, it was also influenced by the velocity of the robot. Furthermore since the IMU measured the motion of the whole robot, the assumption that the terrain under the robot was consistent, had to be made. This meant that for an image, all the pixels of a row had to be labeled with the roughness label. Lastly since the IMU metric was calculated from the acceleration of the robot in the vertical axis, it was difficult for a human to gain an intuitive understanding of what a particular class label meant.

The point to plane distance roughness metric worked by simply fitting planes to a 3D point cloud and finding the distance of the point from the plane that was fitted to its neighborhood. However, the process of finding the neighbors of a point within a specified radius is an expensive operation and therefore would take quite long to run on the log files. Additionally to enable calculation of the point to plane roughness metric, sensors capable of producing a point cloud were needed. Such sensors like a stereo camera or 3D lidar are quite expensive. Contrary to the IMU based roughness metric however, this roughness metric was independent of velocity and also it had an intuitive physical interpretation. This would make it easy for a human to understand what each class meant.

With the roughness metrics developed, a system to autolabel the images was designed. This system allowed a user to enter the path of a rosbag log file, set the parameters for extraction such as kernel size, filtering method, and then set to run. The system is able to save the extracted images and labels, but can also be configured to also save meta-data from the rosbag file such as robot velocity, path driven, or the depth map. This system was set to label on the area immediately in front the robot with the roughness labels, and all other areas were given a label of no information. The usefulness of this system was leveraged multiple times during the research.

For example, during the process of researching, different filtering methods for the IMU were considered, different window sizes were considered for each type of filtering, different methods point to plane distances were considered. Each of these times the right parameters were set and then the system was run without any intervention.

Next, using the the images and labels, networks were trained to learn how to predict roughness from an RGB image. Initially the idea was to learn actual value of the IMU based roughness so the networks were trained for regression. When that produced little success, the labels were categorized into classes and the networks were changed to solve a classification problem. A patch based method was implemented whereby a custom designed network called terrainNet was trained to classify 64x64 patches sampled from the full image. Velocity was included as an input into this network to cater for the dependence of the labels on the robot velocity.

With moderate success, the patch based method was switched to using a full image segmentation method using the SegNet image segmentation network. Using just the images and labels showed an improvement in performance from using the patch based method. In an effort to get better results, different filtering methods for the IMU data were explored and it was found that mean smoothing filtering produced the optimum results. Due to the dependence of the IMU roughness metric on velocity, a method was needed to somehow incorporate velocity into training. An ensemble method was used whereby the dataset was separated based on the average velocity of the robot during the path driven in the image, and a separate network trained for each of these speed separated datasets. The best performing ensemble method has 4 different networks trained for speed brackets of 0-10, 10-20, 20-30 and 30-40 kilometers per hour.

The other method considered for incorporating velocity into training, was using a velocity mask as input into the network. Due to the difficulty of doing this with the Caffe-SegNet library which SegNet was written in, it was decided to rewrite SegNet in Keras which allowed for easy modification of the network to suit our needs. Unfortunately while the best effort was made to rewrite SegNet as close as possible, the only method to replicate the upsampling layer of SegNet was to modify the backend of Keras, which was beyond the scope of this work. Nevertheless, two methods, early stage velocity fusion and late stage velocity fusion were used. With this SegNet-Keras network, though we were unable to get the surpass the performance of the ensemble method, we were able to show that late stage fusion greatly improved performance from using no fusion.

SegNet was also used to train the point to plane distance roughness metrics and it was found that using the max point to plane distance metric with a sub image from the full sized image achieved the best performance. The IMU roughness metric greatly outperformed the point to plane roughness metric. Through the course of this research we have shown an improvement in the roughness labels as well as the performance of the networks to predict the roughness labels. However as always there is room for improvement and based on the experience over the last 2 years, we have some recommendations for future work.

For the IMU based roughness metric, there is a clear dependence on the robot velocity. However, the dependence on velocity along with the previously stated assumption that the terrain under the robot is the same, resulted in inconsistent labels. This meant that there were instances where a patch of terrain would get a label of class 1 but at other times the same patch could get a label of class 3, and some examples were shown of this. With these inconsistent labels we made the dataset a difficult one, but we gladly accepted the challenge.

It is possible that by using a sensor such as suspension stroke sensors which could measure the roughness of the individual wheels, more informative labels could be created. It must be noted here that the robot, the Yamaha AS-X1 did have suspension stroke sensors which measured the movement of the suspension directly. We originally intended to use these sensors to measure roughness but it turned out that those sensors were unavailable to use by the time we were ready to collect data. With data from these sensors, it would be possible to label the path traveled by the left and right side wheel track of the robot independently, and such avoid the terrain under the robot is the same. The downside of this is that there would be less labeled area per image, and as such a larger dataset would be needed. This would be able to handle one of the reasons for the inconsistent labeling.

The other reason for the inconsistent labeling is the dependence of the IMU roughness metric on velocity. It was shown that using a velocity during late stage fusion can improve the results. Future work could focus on this by finding a way to include such a mask into the original version of SegNet written in the Caffe-SegNet library. The other option is to rewrite the original SegNet's upsampling layer in the Keras backend, to completely emulate the the original network as well as finding a way to ignore the no information class labels.

It could be possible that modifying the networks as described above may not actually work. In this case future work could be done in developing a velocity independent IMU roughness label. This means that a method would need to developed to normalize the IMU roughness metric by velocity. This may involve a deep knowledge of mechanics and signals.

The point to plane distance roughness metric holds great promise for estimating roughness since it does not have the problems of the IMU roughness metric, namely the velocity dependence and the constant terrain assumption. While we did not achieve the high performance we expected, future work could focus on using a point cloud generated from a lidar instead of one from a stereo camera. While such a method would have its own challenges such as point registration, we believe it would give a much richer point cloud which would be able to estimate the contours of the terrain more accurately. Hence the point to plane distance metric would be much more accurate and could possible get better results.

The long computation time for this roughness metric was a limiting factor. Finding the neighbor points within the search radius was the main bottleneck. It could be possible to reduce this search time by loading the point cloud using a data structure such as an Oct-Tree which can help speed up the search process. We also suspect that the point to plane distance could benefit from using a better labeling scheme with more classes. Since we wanted to follow the labeling scheme with the IMU roughness metric we only used four classes. It is possible that by using four classes we did not get a good measure of the terrain profile. Again this could have lead to inconsistent labeling. Future work could focus on using more classes to solve this problem.

Lastly since there are a lot of factors that could affect terrain roughness, it is worth exploring multimodal inputs into the network. We have solely explored using velocity as an input, but possible including a depth map, 3D registered points, vehicle acceleration in all axes or vehicle displacement in all axes could improve the network performance. We would have liked our system to be generalizable to all areas of the terrain, including those for which no data was available. However our current system only predicts well for the area directly in front the robot, that is, the trapezoidal area in front the robot which is labeled. Any terrain outside this area the network gives a random class. It is possible that by using a multimodal network, the generalization

capability of the network can be improved.

In conclusion, we have developed roughness metrics along with an autolabeling system that removed human subjectivity from the labeling process. With the autolabeled dataset, we have trained various networks to learn to predict the roughness of the terrain. While the individual components of this thesis may not have been novel, to the best of our knowledge we have developed a system that in its totality is a novel contribution to the robotics community. Although we did not achieve the high levels performance we were expecting, we followed the scientific research process in its entirety and through the course of the research continually refined the system to show an improvement in performance from its inception to the finale.

Bibliography

[1] Autonomous haulage: Making mining safer and more productive today, . URL https://www.cat.com/en_US/articles/customer-stories/mining/autonomous-haulage-making-mining-safer-and-more-productive-today.html. 1.1

[2] Improve autonomous mining safety and productivity, . URL https://www.cat.com/en_US/by-industry/mining/articles/improve-autonomous-mining-video.html. 1.1

[3] Yamaha motor research and development is developing intelligent vehicle technology, Aug 2017. URL https://www.youtube.com/watch?v=mtFchoPOrQo. 1.1, 1.2

[4] Human intelligence through an apiaccess a global, on-demand, 24x7 workforce. URL https://www.mturk.com/. 1.1

[5] Multisense s21. URL https://carnegierobotics.com/multisense-s21/. 1.2

[6] Mti-g-700 - products. URL https://www.xsens.com/products/mti-g-700/. 1.2

[7] David Stavens, Gabriel Hoffmann, and Sebastian Thrun. Online speed adaptation using supervised learning for high-speed, off-road autonomous driving. In *IJCAI*, pages 2218–2224, 2007. 2.1, 2.2, 2.3, 3

[8] Christian Weiss, Holger Frohlich, and Andreas Zell. Vibration-based terrain classification using support vector machines. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pages 4429–4434. IEEE, 2006. 2.1

[9] Christian Weiss, Nikolas Fechner, Matthias Stark, and Andreas Zell. Comparison of different approaches to vibration-based terrain classification. In *EMCR*, 2007. 2.1, 2.3

[10] Chris C Ward and Karl Iagnemma. Speed-independent vibration-based terrain classification for passenger vehicles. *Vehicle System Dynamics*, 47(9):1095–1113, 2009. 2.1, 2.2, 2.3

[11] Sebastian Otte, Christian Weiss, Tobias Scherer, and Andreas Zell. Recurrent neural networks for fast and robust vibration-based ground classification on mobile robots. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5603–5608. IEEE, 2016. 2.1

[12] Abhinav Valada, Luciano Spinello, and Wolfram Burgard. Deep feature learning for acoustics-based terrain classification. In *Robotics Research*, pages 21–37. Springer, 2018.

2.1

[13] Graeme N Wilson, Alejandro Ramirez-Serrano, Mahmoud Mustafa, and Krispin A Davies. Velocity selection for high-speed ugvs in rough unknown terrains using force prediction. In *International Conference on Intelligent Robotics and Applications*, pages 387–396. Springer, 2012. 2.2

[14] Mauro Bellone, Giulio Reina, Nicola I Giannoccaro, and Luigi Spedicato. Unevenness point descriptor for terrain analysis in mobile robot applications. *International Journal of Advanced Robotic Systems*, 10(7):284, 2013. 2.2, 2.3, 3, 3.2, 3.2.2, 3.2.2, 3.2.6

[15] Christian Weiss, Hashem Tamimi, and Andreas Zell. A combination of vision-and vibration-based terrain classification. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2204–2209. IEEE, 2008. 2.2

[16] Dong-Ki Kim, Daniel Maturana, Masashi Uenoyama, and Sebastian Scherer. Season-invariant semantic segmentation with a deep multimodal network. In *Field and Service Robotics*, pages 255–270. Springer, 2018. 2.2, 2.3, 3, 3.2, 3.2.4, 3.2.6, 5.4

[17] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 2.2, 5.1, 5.1

[18] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 2.2, 5.1, 5.4, 5.4.1, 5.4.2.3, 5.4.2.3, 5.4.2.3

[19] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014. 2.2

[20] Wei Mou and Alexander Kleiner. Online learning terrain classification for adaptive velocity control. In *2010 IEEE Safety Security and Rescue Robotics*, pages 1–7. IEEE, 2010. 2.2, 2.3

[21] C Brooks and Karl Iagnemma. Self-supervised terrain classification for planetary rovers. In *Proceedings of the NASA Science Technology Conference, Adelphi, MD, USA*, 2007. 2.2

[22] Eric W Weisstein. Plane. URL `http://mathworld.wolfram.com/Plane.html`. 3.2.2

[23] Jens Berkmann and Terry Caelli. Computation of surface geometry and segmentation using covariance techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(11):1114–1116, 1994. 3.2.2

[24] Cloudcompare. URL `http://www.cloudcompare.org/`. 3.2.2

[25] Steven Michael. LaValle. *Planning algorithms*. Cambridge University Press, 2006. 4.1

[26] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015. 5.1

[27] Svetlana Lazebnik, Cordelia Schmid, Jean Ponce, et al. Spatial pyramid matching. *Object Categorization: Computer and Human Vision Perspectives*, 3(4), 2009. 5.2.1

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 5.2.2

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5.3.1

[30] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015. 5.4.1

[31] Daniel Maturana, Po-Wei Chou, Masashi Uenoyama, and Sebastian Scherer. Real-time semantic mapping for autonomous off-road navigation. In *Field and Service Robotics*, pages 335–350. Springer, 2018.