CRANFIELD UNIVERSITY

KESTER BROATCH

LEARNING VEHICLE DYNAMICS MODELS BY SELF-SUPERVISED LEARNING

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
Autonomous Vehicle Dynamics and Control

MSc
Academic Year: 2019–2020

Supervisors: Dr Dmitry Ignatyev, Dr Argyrios Zolotas,
Jeremy Baxter (Qinetiq)
August 2020

Abstract

Mobile robots are increasingly being utilised in unstructured and off-road environments which they have no prior knowledge of. Thus, when planning trajectories or tasks it is important that the robots can make accurate and robust forward predictions about how the vehicle will interact with that terrain.

This project proposes a self-supervised learning architecture which samples and learns the vehicle and terrain relationship from experience, and can adapt as new data is collected. To create the training data the system assigns a 'traversal' score to the vehicle dynamics data which is mapped onto terrain images from a previous point in time. It is shown that this data can be used to train a neural network which can predict future vehicle behaviour, and can be easily integrated with navigation algorithms such as path planning.

In addition, a simulation environment is proposed which can be used to pre-train the network using *real* images and *simulated* vehicle behaviour in an attempt to reduce the training demand placed upon real self-supervised robots which are deployed in the field.

Acknowledgements

I'd like to take this opportunity to thank some of the kind people who helped me throughout this project.

Firstly, I'd like to thank my advisors Dr Dmitry Ignatyev and Dr Argyrios Zolotas who's generous guidance and teaching has helped me grow throughout my studies.

I'd also like to thank Jeremy Baxter from Qinetiq who suggested this interesting project, his insights and breadth knowledge on the subject proved invaluable.

Lastly, I'd like to thank my friends and family who supported my move to Cranfield. New friendships were made and old friendships strengthened.

Contents

List of Figures

List of Tables

Abbreviations

BAGDR	Berkeley Autonomous Driving Ground Robot
CNN	Convolutional Neural Network
Conv2D	2D convolution layer
DARPA	Defence Advanced Research Projects Agency
FoV	Field of View
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
LAGR	Learning Applied to Ground Robots
NASA	National Aeronautics and Space Administration
ODE	Open Dynamics Engine
ReLU	Rectified Linear Unit
RGB	Red Green Blue
ROAMS	Rover Analysis, Modelling and Simulation
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
TPU	Tensor Processing Unit

Notation

α	Rotation angle of robot about x axis
β	Rotation angle of robot about y axis
$depth$	Depth image's greyscale pixel value(between 0-255)
FoV	Camera constant describing the cameras field of view
γ	Rotation angle of robot about z axis
μ	Friction coefficient
R	Robot's rotational matrix with respect to the world frame
R_x, R_y, R_z	Robot's rotational matrices about the x,y and z world frame axes
$scale$	Camera constant which scales the depth image size to real size
$\vec{slip}$	Ratio between the robot's actual velocity and the ideal velocity
$\vec{V}_{actual}$	Robot's velocity vector
$\vec{V}_{diff}$	Difference between the robot's actual velocity and the ideal velocity
$\vec{V}_{ideal}$	Robot's ideal velocity vector (travelling over flat and level ground)
v_x, v_y, v_y	Robot's xyz velocity components in the world frame
x	x position in camera coordinate frame (left-right in image)
$Xpix$	Horrizontal pixel position on image
y	y position in camera coordinate frame (in-out in image)
z	y position in camera coordinate frame (up-down in image)
$Zpix$	Horrizontal pixel position on image

Chapter 1

Introduction

The robotics industry is in a period of flux in which their use cases are expanding beyond their origins in the structured environment of factory floors. Agriculture, health care, mining, planetary exploration and self driving cars are just a few examples. This new generation of robots are required to navigate in unstructured and dynamic environments if they are to perform effectively and safely.

In practice many modern robots perform rather poorly when a novel terrain is encountered and cannot effectively adapt their navigation plan. A well known example of this was when the NASA Mars rover 'Spirit' became trapped in a sand pit and had to end its mission (NASA, 2010) due to a poor wheel slip prediction over the martian terrain. The increased use of robots in all types of terrain (unstructured or structured) highlights the requirement for robust vehicle-terrain models which can deal with complex and dynamic scenarios.

1.1 Background and Motivation

Terrain 'traversability' assessment is the methodology which is primarily used by agents that are attempting to traverse some complex environment. This involves combining some knowledge about the terrain with some knowledge about the vehicle's own dynamics to make a prediction of the forward vehicle-terrain interaction. This prediction can then be used for things like path planning or danger avoidance.

It was found in literature that there is a large focus on the terrain models, and very little towards the vehicle models that are used in current traversability frameworks (based on a survey by Papadakis, 2013). This means that many of the systems out there work well for a confined set of vehicle-terrain interactions, but quickly break down for more novel interactions (of which there are many).

When considering the shear number of possible vehicle-terrain interactions it is not surprising then that this area of research is underrepresented using classic navigation methods, and no general solution exists. However, in recent years, machine learning methods may have provided a viable solution to tackle this issue. Machine learning offers tools that can deal with the extremely large variable sets involved in vehicle-terrain models, and has the added option of learning on the fly which allows adaption to one's environment.

1.2 Aims and Objectives

This project aims to create a framework which can sample and learn vehicle-terrain interactions from experience in a self supervised manor and use the predictions for smart path planning. This general objective can be synthesised as:

1. Collection and labelling of terrain imagery and vehicle dynamics data in a self-supervised fashion.

2. Formulation of meaningful traversal knowledge from the raw terrain and vehicle data using a terrain 'traversability' score.

3. Development of a machine learning architecture which learns the relationship between terrain imagery and traversal data.

4. Create a simulation environment which provides reasonably accurate traversal data for the purpose of network pre-training.

5. Investigation of the learning architecture's performance and shortcomings when applied to the simulation environment. Discussion of future avenues of work.

6. Integration of the traversability prediction framework with a path planning algorithm.

1.3 Contribution

In this project we take steps towards a fully self-supervised system which can learn vehicle-terrain interactions from experience.

Firstly, a 'traversability' criterion was proposed which extracted useful traversal knowledge from raw vehicle-terrain interaction data. This traversal data was mapped onto terrain images to construct a large neural network training set. Following this, a neural network was developed which learned the relationship between the traversal data and the terrain images. This network was used to predict future vehicle-terrain interactions, and was deployed in a path planning algorithm which could account for the traversability of the terrain.

Further to this, a training simulation environment was developed with the intention of pre-training models to be used later used as starting model on self-supervised robotics platforms. This simulator differed from most virtual training environments as it used *real*

terrain images along with *simulated* vehicle dynamics to train the neural network. This allows the bulk of a robot's learning to be carried out in the safety of simulation, but also reduced some of the issues with training using rendered images from simulation.

1.4 Outline of Thesis

This thesis is divided into eight chapters, organised roughly to follow the order in which the work was carried out:

Chapter 1, Introduction - States the general background of terrain traversability methods and the current challenges. Outlines the objectives of the author, the methodology adopted to achieve them, and the main contributions of the project.

Chapter 2, Literary Review - Summarises the current state of the art of robot terrain traversal methods. Promising avenues of work are analysed, and the observed strengths and limitations are presented, along with their wider dependencies on new technology.

Chapter 3, Approach - Gives an overview of the traversability prediction and data collection methods that were chosen for the project, based upon the project objectives and the literary review.

Chapter 4, Traversal Simulation - Description of the steps taken to generate the terrain model and vehicle simulations using the DeepScene (Valada et al., 2016) dataset.

Chapter 5, Traversal Data Generation - Outlines the process in which terrain traversability data was extracted from the raw simulation data and converted into training data for the neural network.

Chapter 6, Learning Traversability - Details the design and tuning of a convolutional neural network which was trained using the terrain traversal data.

Chapter 7, Results and Discussion - Details the output and performance of the traversal learning architecture.

Chapter 8, Conclusions and Future Work - Discusses the performance of the architecture with reference to the origial objectives of the project. The limitations of the architecture are acknowledged and used to outline suggestions for future work.

Chapter 2

Literary Review

This chapter gives some background information about terrain traversal methods for autonomous vehicles. In addition, the current state of the technology is presented; detailing some of the largest challenges and advancements currently taking place.

2.1 Traversability Models

The definition of terrain 'traversability' often differs between authors, but in general refers to the ability of a ground vehicle to reside over a terrain region in an acceptable state considering its current state (definition from Papadakis, 2013). In almost all cases traversability is some function of the terrain and the constraints of the vehicle.

Intelligent agents (robot or biological) make forward traversability predictions of their environment by estimating the interaction behaviour of some predicted terrain model and internalised vehicle model. The estimated traversability model can then be used for tasks such as path planning, danger avoidance or decision making.

2.1.1 Geometry Based Models

Classical methods of traversability analysis were very heavily weighted towards geometric terrain models. The main types of terrain models were elevation maps, originating from the grid based methods using sonar developed by Moravec and Elfes, 1985, which proved to be very popular for graph based path planning. These methods remain popular today with the increasing use of lidar and point cloud data (see Jaillet, Cortés, and Siméon, 2010). The issue with these models is that they only consider spatial terrain representation and typically neglect or use very simplistic vehicle representations.

2.1.2 Appearance Based Models

The recent availability of high performance image classifiers has seen an increase in appearance based traversability models. These models can be used to extract characteristics such as terrain type, roughness, slope and slip (Ayanna Howard and Seraji, 2001) from terrain images. These appearance based models can be combined with the geometric models to produce highly accurate terrain representations, for example Maturana et al., 2018. However, the same issue persists as before; no vehicle information is necessarily considered with this approach, so they are limited in accuracy.

2.1.3 Proprioceptive Based Models

The last type of traversability model is the use of 'proprioceptive' sensors to measure the vehicles internal stimulus to gain knowledge about the vehicle and terrain. For instance, Garcia Bermudez et al., 2012 used IMU vibration and Leppanen, Virekoski, and Halme, 2008 used feet strain forces for terrain classification. These methods provide useful knowledge about the vehicle model and it's constraints, but are limited in their knowledge about the terrain which is not directly below the vehicle.

2.1.4 Hybrid Models

The issue with the methods above is that they are generally limited in scope, so they can be significantly improved when they are combined into a hybrid model. Proprioceptive terrain models can be combined with exteroceptive models (external stimulus) to fuse vehicle and terrain knowledge. However, using classical navigation methods to combine these models was not practical in most cases as there is a huge number of combinations which would have to be accounted for to make the model more general. However, the recent rise in machine learning methods has provided tools which can deal with these large data sets, and has provided some very promising results in recent years.

One of the largest influences in this area was DARPA's Learning Applied to Ground Robots (LAGR) program (Jackel et al., 2006) which provided a platform which researchers could easily create hybrid traversability models from (see Shneier et al., 2008 and Andrew Howard et al., 2006). The research gained from the LAGR program was later used in the NASA Mars Rover programs to develop a system which learns the relationship between vehicle slip, vibration and terrain images (see Angelova et al., 2007 and Otsu et al., 2016).

2.2 Self Supervised Learning

The functionality of machine learned hybrid traversability models was expanded further with the idea of 'self-supervised' learning. This is where robots collect data and learn from experience without any human supervision, which allows vehicles to adapt to the terrain that they are currently traversing. The use of modern machine learning techniques and powerful on-board computers has allowed the relationship between proprioceptive vehicle models and exteroceptive terrain models to be sampled and learned in real time while the vehicle is in operation.

A number of very promising self-supervised systems have been explored in recent years. Wellhausen et al., 2019 developed a system for a legged robot which learns the relationship between foot slip and terrain images during human controlled locomotion, this knowledge is then used to plan future autonomous routes which minimise slip under-foot. Kahn, Abbeel, and Levine, 2020 take this a step further with their BAGDR system which autonomously explores and learns its environment with no human intervention required.

2.3 Learning from Simulation

One of the issues with self-supervised learning is the fact that learning must occur from experience, which can be very expensive. A vehicle learning from scratch may have to deal with crashes and tipping over before it learns enough to avoid these failures. It would therefore be desirable to pre-train a system using a simulation environment, which can then be used to initiate learning on a real vehicle (Cole et al., 2019).

Recent advancements in simulation software and gaming engines have provided a host of tools which can assist with simulation based learning techniques (Rosique et al., 2019). High fidelity physics engines, cutting edge graphics and accurate sensor models are helping to close the simulation reality gap (Jakobi, Husbands, and Harvey, 1995).

As with the Traversability Models mentioned above, NASA has been at the forefront of vehicle-terrain simulation research. Their 'ROAMS' simulator (Jain et al., 2004) accurately predicts vehicle behaviour over soft and unstructured martian terrain. This simulator was successfully used to assist in the training of a self-supervised terrain classifier (Helmick, Angelova, and Matthies, 2009) for the Mars rover projects. Another robot simulator, Gazebo (Koenig and Andrew Howard, 2004), was used by Chavez-Garcia et al., 2017 to learn the interaction of a robot with various height maps from simulations and was then successfully deployed on the real robot.

Chapter 3

Approach

This chapter gives an overview of the general approach which was settled upon given the project's objectives and literary review. The traversal prediction and data collection methods are detailed, along with their composition into a self-supervised learning architecture.

3.1 Traversability Prediction Method

The vehicle-terrain interaction model was designed to be a hybrid model that combines the input from terrain geometry (from depth images), terrain appearance (from RGB images) and vehicle dynamics data. This type of model negates the issues from unbalanced vehicle and terrain data, as discussed in the Traversability Models part of the Literary Review.

To extract a useful relationship between the terrain and vehicle data it was decided to use a convolutional neural network (CNN) to learn the relationship between the RGB and Depth images, and the vehicle data. CNNs are a very good choice for identifying patterns and features of image data, and the improved performance of on-board computers makes them well suited for real time robotics applications.

To make the system adaptable to its environment a self-supervised learning architecture was adopted. All of the data required to learn the vehicle-terrain interaction was chosen such that it could be collected and processed on-board the vehicle, without the need for human supervision.

3.2 Data Collection Method

The architecture is eventually intended for use on real vehicles, but for this project the data was collected entirely from simulations. Starting in simulation allows the model to be pre-trained before being deployed in real life, thus reducing the expensive and time consuming process of training a real self-supervised system from scratch.

The issue with the simulation based pre-training is that there can be a large reality gap (Jakobi, Husbands, and Harvey, 1995) between the pre-trained model's prediction and reality. To minimise the reality gap a *real* terrain data set (DeepScene Valada et al., 2016) was used to generate the simulation environments, which allowed the CNN to be trained with *real* images rather than rendered graphics images.

3.3 Learning Architecture

The data collection and traversability prediction methods outlined above are combined into a self-supervised learning architecture (see Figure 3.1). Firstly, in simulation or real life, the vehicle makes some pass over the terrain. The vehicle dynamics data is collected and used to calculate a traversability score for every vehicle position, which is then mapped back onto images of the traversed terrain. These images and traversability scores are used to train the CNN to predict traversability scores for new images. These predictions can be used for planning and decision making tasks further down the pipeline.

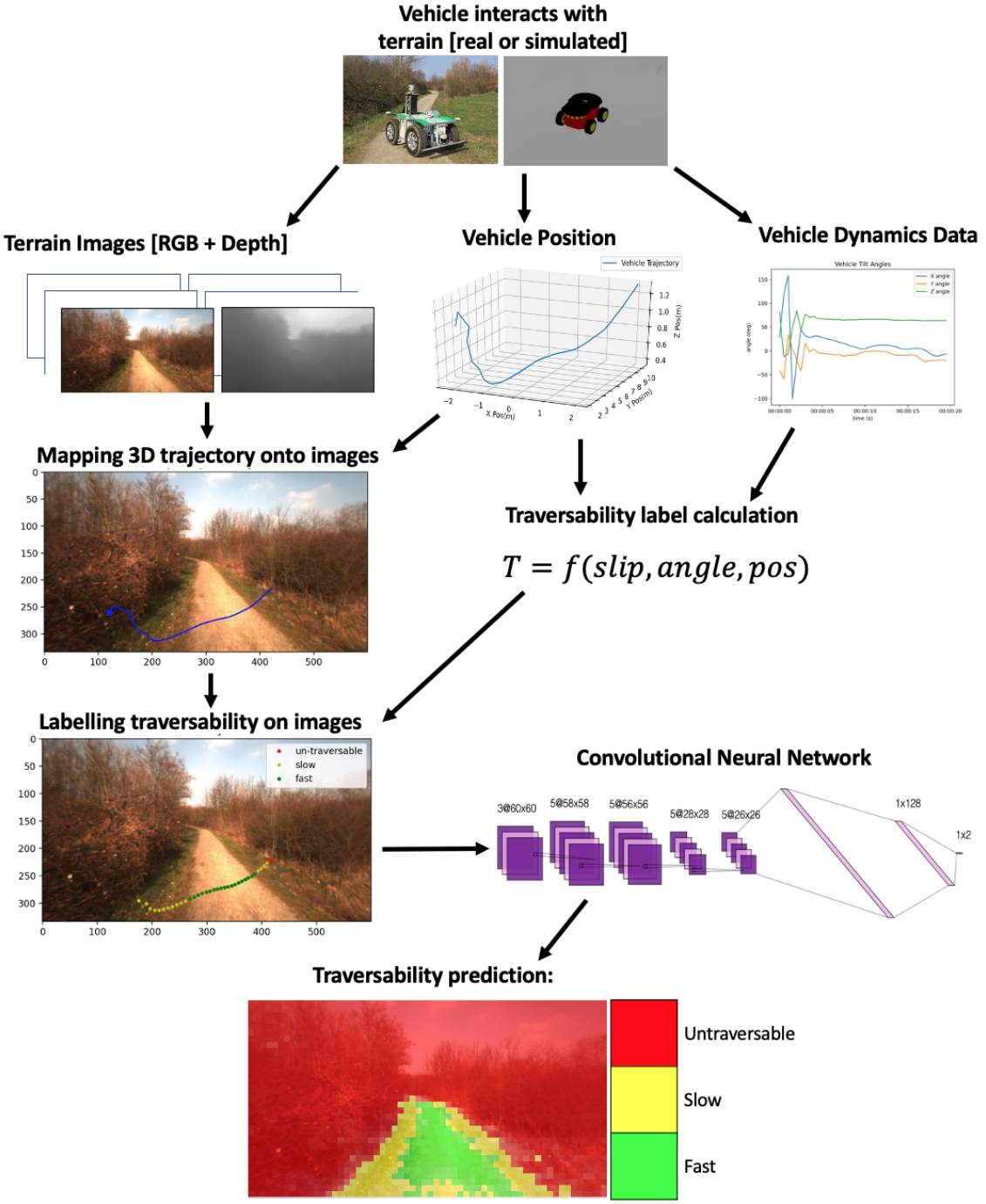


Figure 3.1: Self-supervised traversability learning architecture

Chapter 4

Traversal Simulation

This chapter outlines the steps that were taken to create the terrain and vehicle simulations in the Gazebo simulation environment (Koenig and Andrew Howard, 2004), with the intention of using the resulting model as pre-training for a real system.

4.1 Terrain Generation

The DeepScene Freiburg Forest data set (Valada et al., 2016) was used to generate terrain models for the traversal simulation. As discussed in the Approach section, the use of a real data set to generate the terrain simulations allows the CNN to be trained with real images rather than rendered graphics images, thus improving the pre-trained model's performance.

This data set consists of 350 RGB and depth images with labelled terrain classes, and a further 15,000 unlabelled raw images. A sample from this data set can be observed in Figure 4.1.

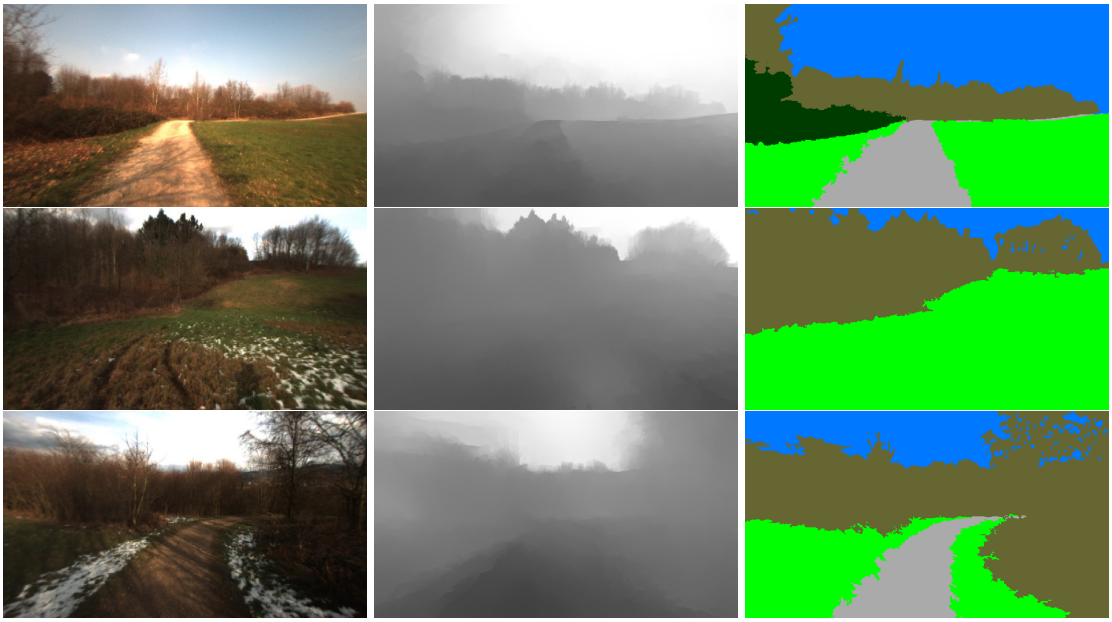


Figure 4.1: Freiburg Forest data set (Valada et al., 2016) sample - RGB, Depth and Ground Truth images

4.1.1 Terrain Point Cloud Generation

The first step in extracting the terrain geometry was to obtain the terrain's 3D point cloud. To accomplish this the 2D grayscale depth images of the terrain were transformed to 3D representations by projecting the depth information onto the camera's reference frame. To help illustrate this process Figure 4.2 shows a simplified depth image of the terrain. The pixel values and colour represents the depth of the image at that pixel position, with zero (black) representing minimum distance from the camera and one (white) representing maximum distance from the camera.

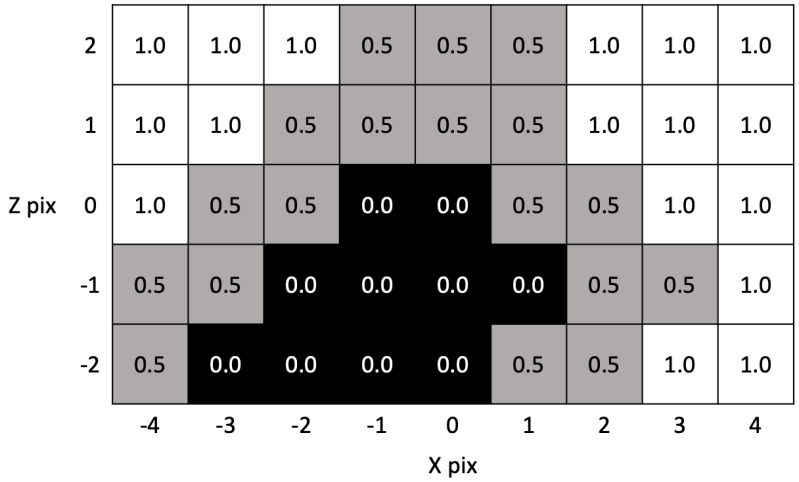


Figure 4.2: Simplified depth image showing pixel positions and example depth values at those pixels

A 3D point cloud was then extrapolated from the depth image, with the camera position at the origin of the coordinate system. Each xyz point in the point cloud was calculated for each corresponding pixel of the depth image using the the conversion equations for a kinect camera (Bo and Lai, 2014), as seen in equations 4.1 below.

$$x = \frac{Xpix \cdot depth}{FoV} \cdot scale \qquad y = depth \cdot scale \qquad z = \frac{Zpix \cdot depth}{FoV} \cdot scale \tag{4.1}$$

The field of view (FoV) and scale constant were unknown as no camera information was provided with the data set. These were set as 1500 and 1000 respectively after some experimentation produced a point cloud of reasonable dimensions. However, the value of these constants is not considered that important as the system should learn any terrain that is presented to it.

The xyz point cloud which was generated from the depth image from the first example image from figure 4.1 is shown on Figure 4.3 below.

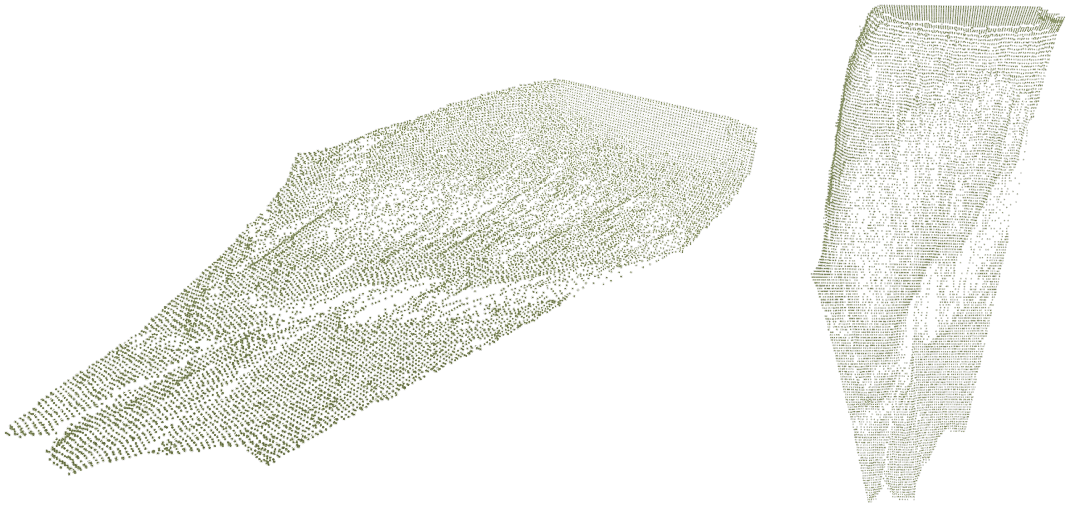


Figure 4.3: Terrain point cloud generated from a depth image

4.1.2 Terrain Mesh Generation

To make the terrain representation usable in the simulation, the point cloud had to be converted to a 3D mesh. The point cloud was first down sampled by taking the average point position in a 0.2m grid, which was followed by triangular meshing and smoothing, as seen in Figure 4.4. This process was carried out automatically using the MeshLab 3D modelling software (Cignoni et al., 2008).

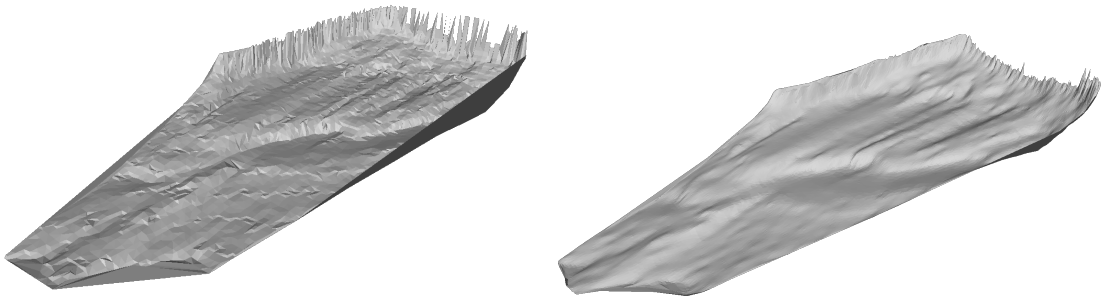


Figure 4.4: Terrain model raw mesh (left) and smoothed mesh(right)

The smoothed mesh was exported as a Collada file (.dae) as it was found to be the most stable 3D model type for use in the Gazebo simulation engine. The meshes were found to represent the terrain reasonably accurately (see Figure 4.5). These meshes were loaded

into a gazebo 'world' file where they could be loaded throughout the simulation.

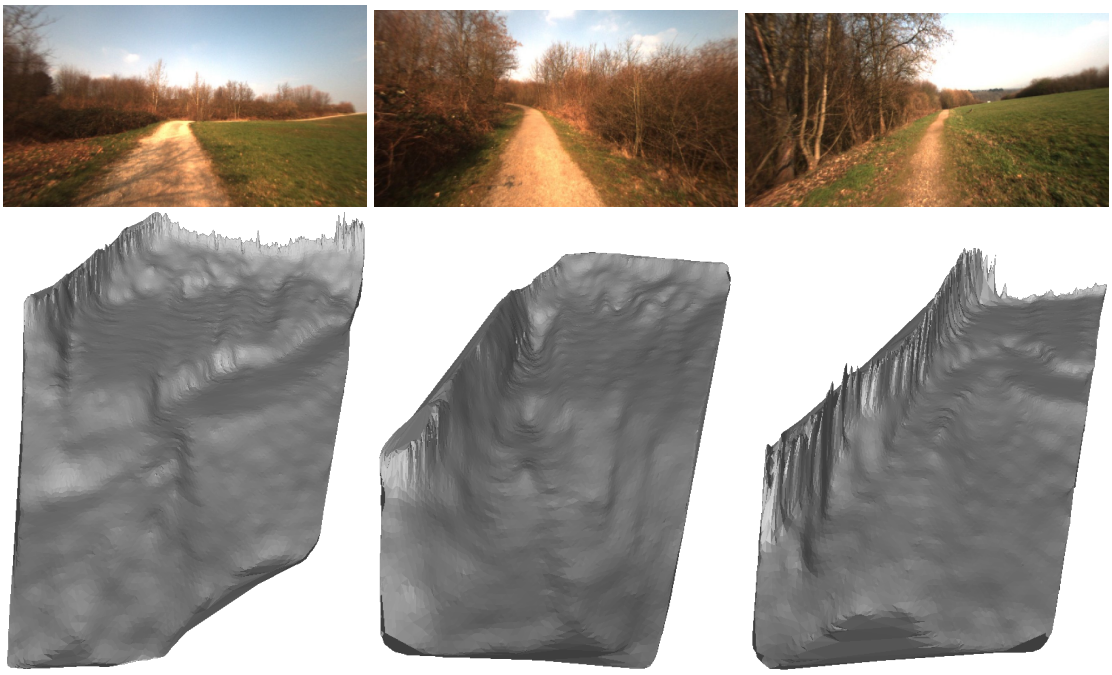


Figure 4.5: Examples of terrain meshes

4.2 Vehicle Model

The Gazebo simulator with the ODE physics engine (Koenig and Andrew Howard, 2004) was chosen to simulate the vehicle dynamics and the simulation was controlled and logged using the ROS communication architecture (Stanford Artificial Intelligence Laboratory et al., 2018). This setup was used because the Gazebo-ROS combination makes it very easy to eventually replace the simulator with a real robot. Additionally, the Gazebo-ROS ecosystem has a strong community with a lot of resources and support.

The Pioneer 3-AT robot was used for the simulations as there are a lot of pre-built gazebo and ROS models available (ROS-wiki, 2017), and could be easily expanded to the real system at a future date (see Figure 4.6). The Pioneer 3-AT is a medium sized skid-steer robot, designed specifically for outdoor exploration and research, so was ideal for this application.

Figure 4.6: Pioneer 3at robots in use (left) (ROS-wiki, 2017) and rendered in Gazebo (right)

4.3 Simulation Setup

A modified version of an open source gazebo height map simulator, designed by Chavez-Garcia et al., 2017, was used for the simulation runs. This software was designed to spawn a robot onto a gazebo elevation map and log its progress over time which provided a good starting point upon which to build the simulations.

The robot was spawned on the terrain mesh in a grid pattern moving away from the camera position, see Figure 4.7. This ensures that the model will learn the behaviour of the vehicle over the terrain which is immediately ahead of it. The vehicle uses a constant wheel surface speed of 0.5m/s and no steering input which ensures that any change in direction or speed is caused by the terrain and not by a vehicle controller.

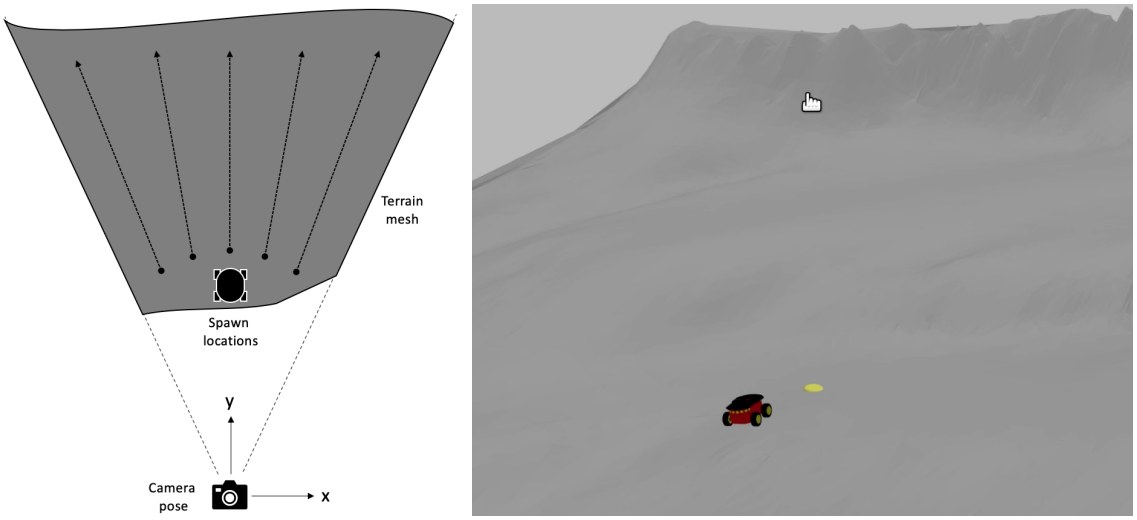


Figure 4.7: Terrain spawning in a grid pattern moving away from camera position(left), example of traversal of vehicle over terrain mesh in the gazebo simulator (right)

The various traversal paths of the vehicle over the terrain mesh are observed by projecting the 3D paths back onto the original 2D image, see Figure 4.8 below.

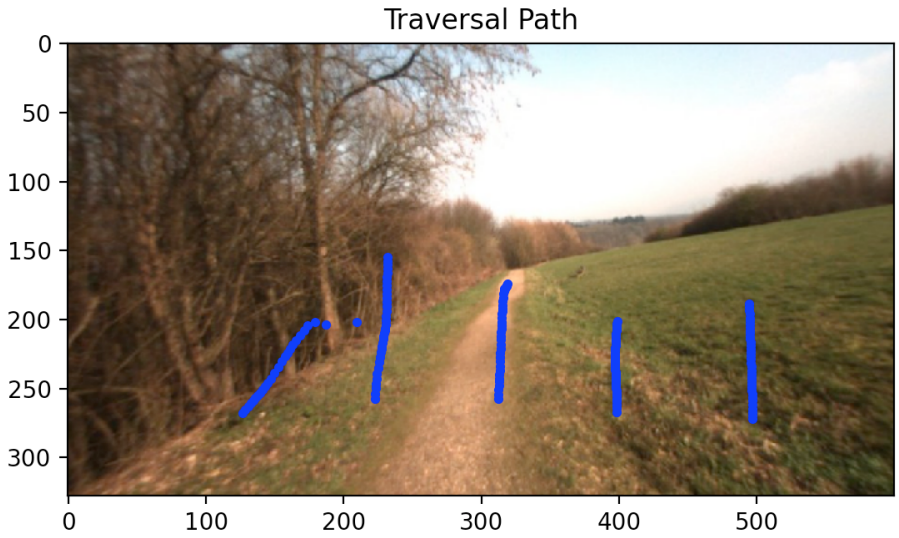


Figure 4.8: Vehicle traversal path (dark blue) projected back onto the original image

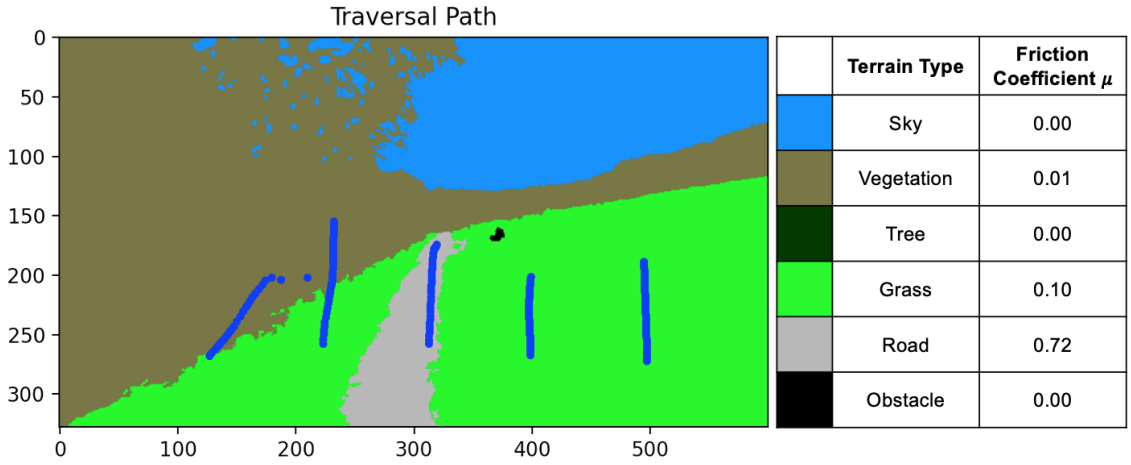
It should be noted that this method of labelling images assumes that the displacement between the camera position and the vehicle position is known perfectly, as it is in the simulator. However, it is likely that a real system will use a single robot which both records the image and traverses the terrain. A system like this would require some form

of SLAM or odometry to map the camera position to the future robot position, and would therefore incur some inaccuracies.

4.4 Friction Model

The gazebo simulator offers no easy way to vary the friction across an object's surface, so the friction of the terrain was instead simulated using a custom plugin (Gazebo-Tutorials, 2015b) which modifies the terrain friction coefficient as the simulation runs. The ODE physics engine (Gazebo-Tutorials, 2015a) was used to calculate the friction forces in the simulation. When the robot wheels and the terrain collide the smallest friction coefficient of the objects is used in the calculation.

The position of the robot on the terrain was projected onto the terrain ground truth image (as shown in Figure 4.9) for every time step. This allowed the terrain type that the vehicle was traversing to be found during the simulation, and a corresponding terrain friction to be applied. The friction plugin can be turned on and off to investigate the effect of terrain friction on the simulation. When the friction plugin is off the terrain defaults to a friction coefficient of 1.0.



	Terrain Type	Friction Coefficient μ
	Sky	0.00
	Vegetation	0.01
	Tree	0.00
	Grass	0.10
	Road	0.72
	Obstacle	0.00

Figure 4.9: Traversal paths (dark blue) projected back onto terrain ground truth image (left) and examples of terrain friction values(right)

4.5 Simulation Runs

The vehicle position, orientation, velocity and image traversal path (see Figure 4.8) were logged for every time step of the simulation using ROS and saved to csv files. In total 366 different terrain meshes were traversed, which accounts for 26,000 seconds of simulation time recorded or 13km of terrain traversed.

Chapter 5

Traversal Data Generation

This chapter outlines the process in which the terrain traversal information is extracted from the raw simulation data. The data required a fair amount of clean up and pre-processing to extract the traversal information required to generate decent training data.

5.1 Cleaning the Simulation Data

The vehicle position, orientation and velocity data was collected at 0.01s increments. It was found that when the vehicle path was projected back onto the terrain image (as seen in Figure 4.8) for every time step there were a lot of repeated pixel positions, which would cause many repeated training data points. This not only posed a training over-fitting risk further down the pipeline, but also added unnecessary computation overhead. To mitigate this issue the simulation data was down sampled to every 0.5 seconds of simulation. An example of the down sampling of the vehicle's angle about the Z axis can be observed on Figure 5.1.

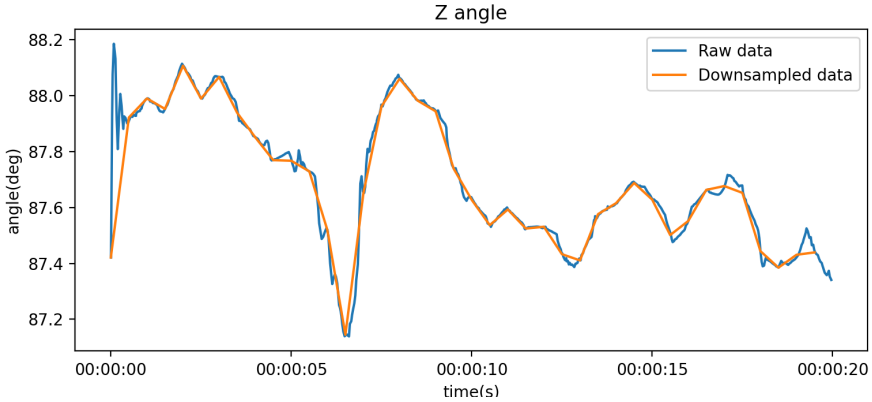


Figure 5.1: Example of raw and down-sampled simulation results, showing robot z orientation

The simulation data was also found to contain cases where the robot had fallen out of the map, or had fully flipped over and was stuck on its back. These cases were removed from the data set as they were false negatives and contained no useful information about the terrain-vehicle interaction. The first 2 seconds of simulation were also removed as the vehicle is getting up to speed.

This resulted in a total of 52,000 usable vehicle dynamics data samples which could be used to label training data.

5.2 Calculating Vehicle Slip

The vehicle slip in lateral and longitudinal directions was used as a measurement criteria for terrain traversability. In theory any terrain-vehicle interaction behaviour (or combination of behaviours) could be learned, but vehicle slip was found to be very useful for our application. The vehicle slip accounts well for the terrain slope and friction as well as the current vehicle dynamics. Initially only the slip in the longitudinal direction was considered, but it was found that this limited the scope of the terrain classification, so a combination of longitudinal and lateral slip was used.

To calculate the vehicle slip an ideal velocity vector ($\vec{V}_{ideal}$) for a vehicle travelling over flat ground with no slip was compared to the actual velocity vector ($\vec{V}_{actual}$), where,

$$\vec{V}_{ideal} = [0.5, 0, 0] \qquad \vec{V}_{actual} = [v_x, v_y, v_z] \tag{5.1}$$

The vehicle rotational matrix R was calculated from the vehicle Euler angles (α, β, γ) about the global xyz axes.

$$R = R_x R_y R_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

The ideal velocity vector was then rotated using the rotational matrix to the same direction as the vehicle velocity vector to allow the velocity difference to be calculated:

$$\vec{V}_{diff} = \vec{V}_{actual} - R \times \vec{V}_{ideal} \tag{5.3}$$

Lastly, the velocity difference vector was used to find the slip in the x,y and z directions of the vehicle as follows:

$$\vec{slip} = \frac{\vec{V}_{diff}}{|\vec{V}_{ideal}|} \tag{5.4}$$

The slip vector was then used to gain information about the vehicle-terrain interaction, as shown in Figure 5.2. A slip vector with nearly zero in all directions represents a vehicle which is travelling very close to the ideal velocity with little slip. A slip vector with non-zero x and y values represent a vehicle sliding in the longitudinal and lateral directions respectively. A slip vector with non-zero z values represent a vehicle 'sinking' into the ground, which in practice does not occur in these simulations.

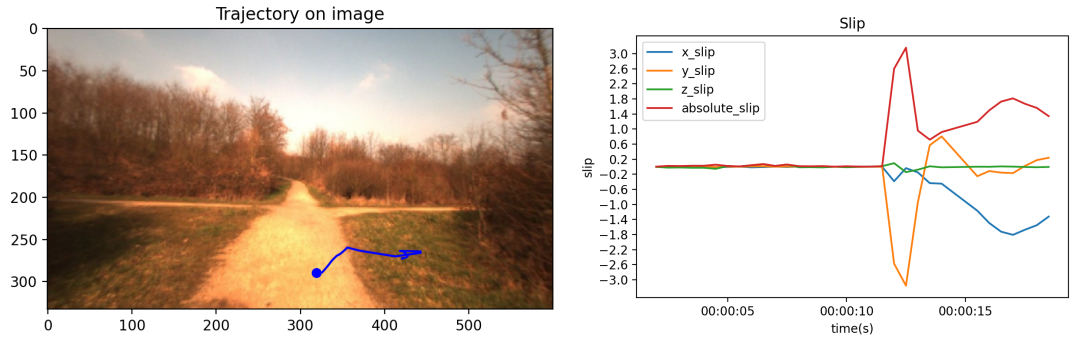


Figure 5.2: Example of vehicle slip - low slip values while travelling over road and high slip values while travelling over grass and rough terrain

5.3 Extracting Terrain Slope

The terrain slope was used alongside vehicle slip as a measurement criteria for terrain traversability. The terrain slope data provided information about regions which did not necessarily effect vehicle slip but are an important factor in considering terrain traversabilty, for instance small bumps or transition areas. The vehicle tilt also acted as a reliable threshold for identifying flips or crashes.

5.4 Terrain Patch Labelling

The terrain traversability was calculated at each data point using various thresholds for vehicle slip, terrain slope and terrain type. If the vehicle was found to be traversing the trees or sky (see Figure 4.9), or the vehicle tilt was too high then the data point was marked as 'un-traversable'. If the vehicle slip or tilt was at a moderate or low level then the data point was labelled as 'slow' or 'fast' respectively. The thresholds used in the test cases is shown in Table 5.1 below.

Table 5.1: Traversability thresholds used for labelling simulation data

Label	Vehicle Slip	Terrain Slope	Terrain type
fast	$\leq$10%	$\leq$20deg	-
slow	$>$10%	$>$20deg	-
un-traversable	-	$>$90deg	Sky or Tree

These traversability labels were then mapped from 3D simulation coordinates back onto the 2D terrain images, as seen in Figure 5.3.

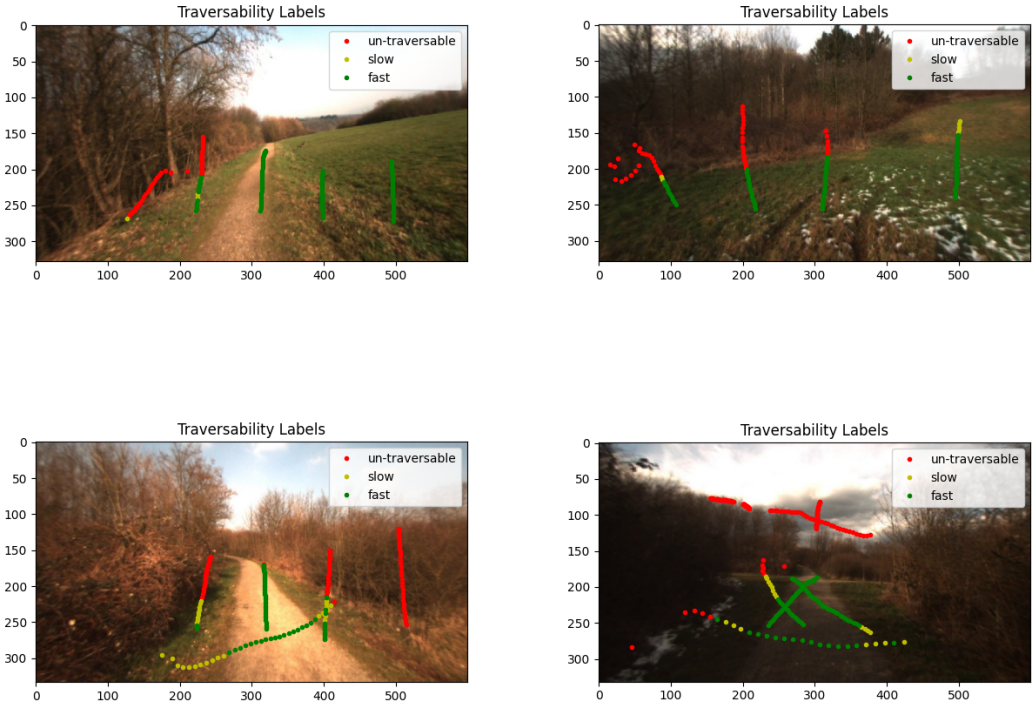


Figure 5.3: Example of the traversability labels which have been mapped back onto the terrain images (shown is labels for a uniform friction simulation)

Initial testing divided the labels into two classes, 'traversable' or 'un-traversable', but this was quickly found to have limited use-cases. A third label was added to allow more detailed terrain prediction which was required for path planning. The architecture was designed to be easily scaled to use more classes as needed.

Image patches centred around each traversability label were then cropped from the original terrain RGB and depth images, as shown in Figure 5.4 below. These cropped images were labelled for every point in the simulation data, then saved in a pandas data frame (Pandas, 2020) for later use as training data for the neural network.

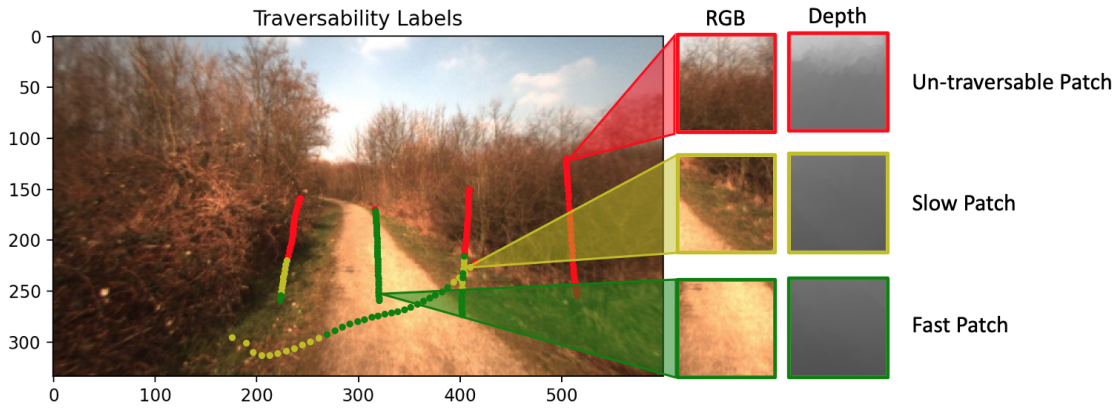


Figure 5.4: Image patches are cropped from the terrain RGB and depth images for every labelled data point

Cropped patches which overlapped the edge of the image were warped by extruding the pixel colours on the image edge (as shown in Figure 5.5). This ensured that the entire terrain image could be classified and the border was not ignored.

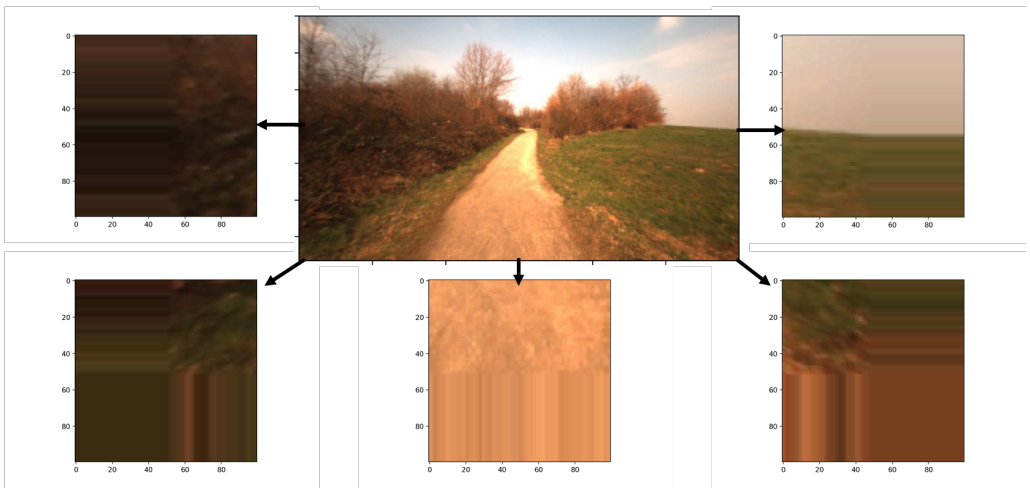


Figure 5.5: Edge patches were warped by extruding the edge pixels, this ensured that the border of the image could be classified

5.5 Artificial Labels

A number of false positive mis-classifications were observed in the first few model iterations. These mis-classifications occurred predominantly on patches where dark branches were contrasted against bright sky. To investigate this mis-classification the patch was transferred to grey scale and the intensity histogram was calculated (see Figure 5.6).

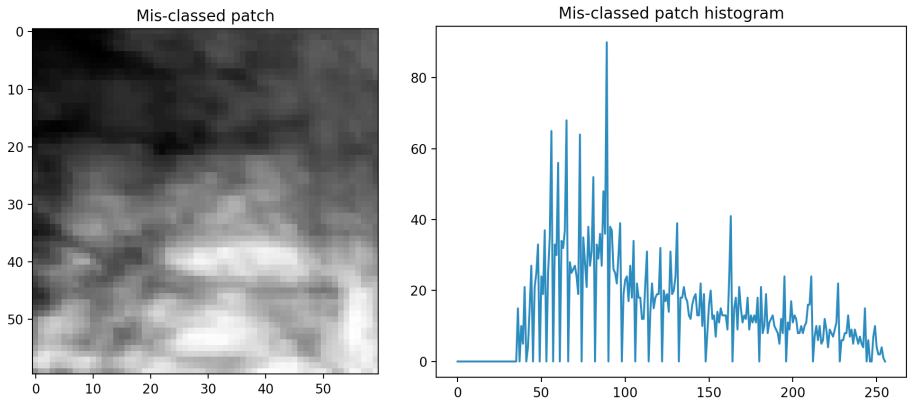


Figure 5.6: Example of a mis-classified patch and it's intensity histogram.

The histogram above shows that the image has multiple grey scale values with zero samples, which suggests that the region is highly discontinuous. Comparing this patch to similar patches (branches against a sky) which were classified correctly shows that correctly classified patch histograms are more continuous (see figure 5.7).

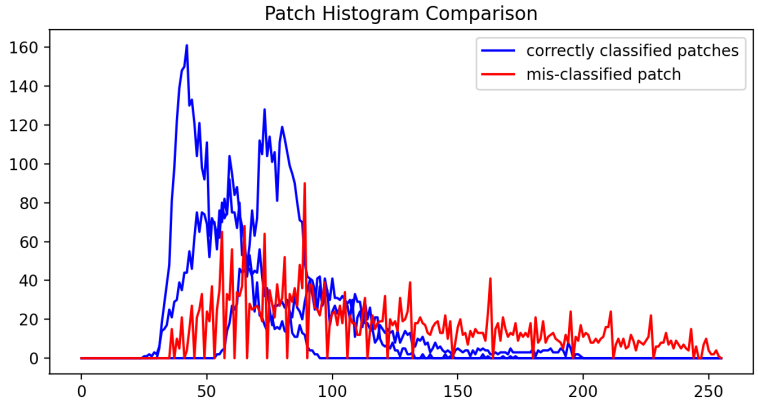


Figure 5.7: Comparison of intensity histograms of a mis-classified image patch and some semantically similar but correctly classified patches.

The comparison in Figure 5.7 above led to the hypothesis that images with a discontinuous intensity are more likely to be mis-classified. It was assumed that the reason for this is due to under-representation of these image types in the training data, which makes sense when we consider the fact that the vehicle is rarely traversing the trees during the simulations.

To solve this issue labels were added to the training data which artificially increased the representation of tree and sky images in the training data. This was achieved by using the terrain data-set's ground truth data to synthetically find and add random training labels to the trees and sky in the terrain image, as shown below.

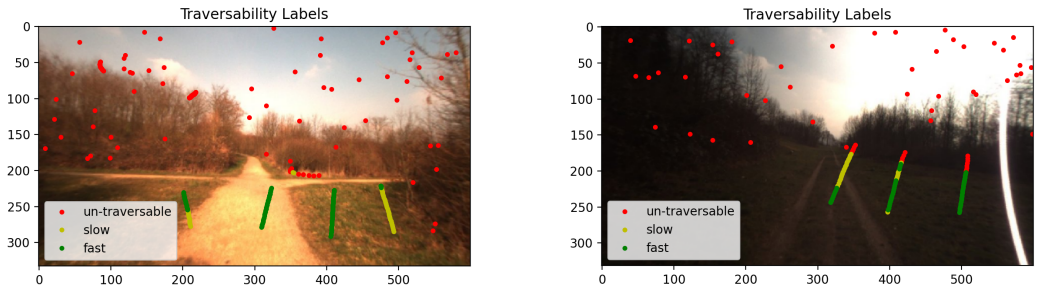


Figure 5.8: Artificial labels added to the training data to increase the representation of tree and sky image patches

Adding these artificial labels did solve the issue of the mis-classification and made the results more stable in general. This suggests that a system running on a real robot will also suffer from the same problem, which means that a mechanism to artificial add labels may be needed to run in parallel to the learning algorithm.

Chapter 6

Learning Traversability

This chapter outlines the neural network design and the process in which the terrain traversal information was used to train the network to establish a relationship between the terrain and the vehicle dynamics.

6.1 Network Design

The traversability labels which are generated from the vehicle-terrain interaction are sparse across the terrain image because the labelling only occurs along thin paths (as seen in Figure 5.3). This represents an interesting challenge as CNN's are designed for dense data across the image, and naively using sparse data does not work as the CNN is sensitive to missing data (Jaritz et al., 2018). One terrain classification system, designed by Wellhausen et al., 2019, solved this problem by applying the Mean Teacher algorithm (Tarvainen and Valpola, 2017) to their semantic segmentation CNN which accounts for sparse labels by averaging the weighting.

However, to deal with the sparse labelling issue in this project the image was cropped into discrete labelled patches (as shown in Figure 5.4 above) which could be classified,

rather than using semantic segmentation of the full image with a sparse label algorithm (as discussed above). Although labelling patches loses some of the image semantics and is less efficient, it was chosen as it is much simpler to implement than the alternative and can be applied to any size of image. The use of semantic segmentation with sparse labelling algorithms would be a useful future addition to this architecture.

The CNN architecture is based upon a height map classifier designed by Chavez-Garcia et al., 2017 which uses a Keras (Chollet et al., 2015) front end and TensorFlow backend (Martin Abadi et al., 2015). This is a standard CNN structure which is well suited for image feature recognition (the exact architecture can be seen below in Figure 6.1). The architecture uses stacked convolutional, max-pooling and fully connected layers which are all activated using ReLU functions. This arrangement allows features of the images to be extracted using kernal convolution filters at multiple levels of abstraction, which are then used as inputs to a fully connected artificial neural network.

The CNN was modified to use 100 pixel RGB and depth images input images and three traversability classes as outputs.

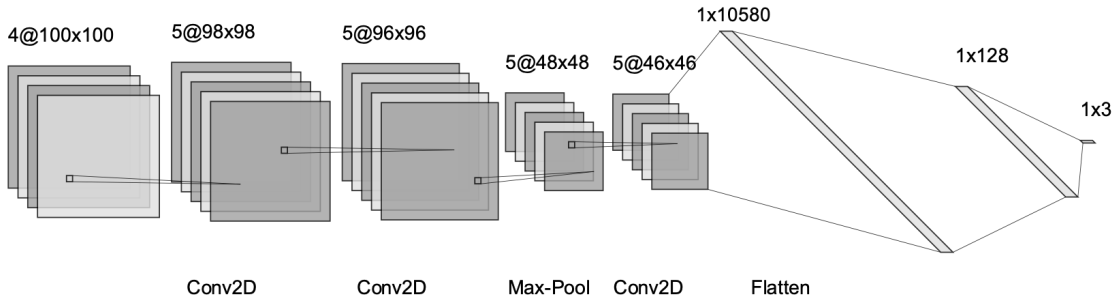


Figure 6.1: Covolutional Neural Network architecture

6.2 Training

As mentioned in the Traversal Data Generation section, there were a total of 52,000 training samples generated from the simulations. This data was first randomly mixed and then

split into training (80%) and testing (20%) sets. Class weights were then calculated for each class because the majority of the traversability images were of the non-traversable class. Initial testing without the weighted classes found that the results were very biased towards one class.

The parameter weights of the CNN were learned with back propagation by minimising the categorical cross entropy loss function with the adadelta optimiser. The adadelta optimiser uses an adaptive learning rate method which was able to deal with with the sparse classes of the traversability data set. An optimiser batch size of 64 was chosen as it was found to have the best balance of accuracy and training speed out of the conventional batch sizes of 32, 64, 128 and 256.

The model accuracy and loss metrics, of the training and testing sets, were used to monitor the CNN training progress. The accuracy gives a good indication of the difference between the predicted output and the desired output of the model, which is useful when assessing overall performance. The loss function generates a model error value which is used by the optimiser to find the global minimum, and as such, it serves as a useful metric to assess the optimisation progress and model stability.

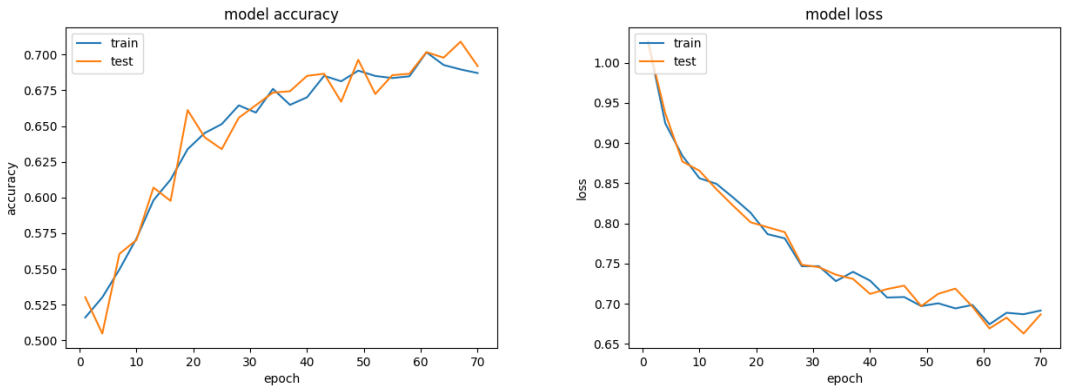


Figure 6.2: CNN training loss and accuracy

The model was trained for 70 epochs before the accuracy and loss began to plateau at an accuracy of around 70% (see Figure 6.2). The loss shows a slow decline with very

little instability as training progresses which suggests that the optimiser is successfully converging. It is observed that the testing accuracy tracks the training accuracy fairly well which may indicate that the model capacity is not high enough, and an increase in parameters (added layers) could allow a higher accuracy to be acheived in the future.

The training was carried out using the Google Colab cloud computing service (Google, 2018) which use GPUs or TPUs for artificial intelligence applications. The computing time varied depending upon the time of day and available service, but in general took around 450 seconds per epoch (690m/s per step) - so for the final training run it took around 9 hours to train from scratch.

6.3 Addressing Over Fitting

It was observed that the model was suffering from strong over-fitting for the first few model iterations which used a smaller training set (one third the size of the final data set used in Figure 6.2). This was occurring because training set was too small and the model was becoming too biased towards the training set and losing generality. The spread in accuracy and loss between the training and testing sets is a classic symptom of this, as seen in Figure 6.3 below.

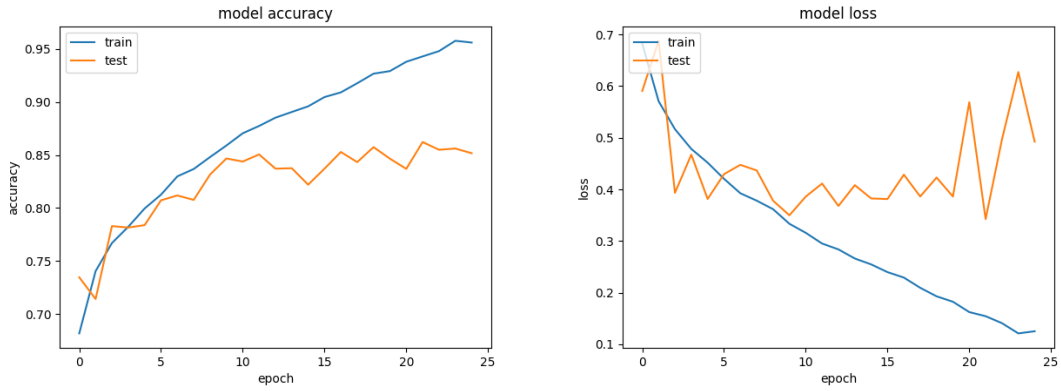


Figure 6.3: Over fitting CNN training loss and accuracy

The over fitting in later models was prevented in part by proper randomisation of the data samples and use of models that had been trained using fewer epochs (before over-fitting occurred). But the main fix for the over fitting was to increase the training set size by a factor of 3, which was used to eventually train the final model shown in Figure 6.2.

6.4 Patch size tuning

The performance of the CNN was very sensitive to the size of the training patch which was cropped from the terrain image (as discussed in the Terrain Patch Labelling section). If the patch was too small then some of the wider semantics of the terrain were lost and if it was too large then there was mis-classification at class boundaries.

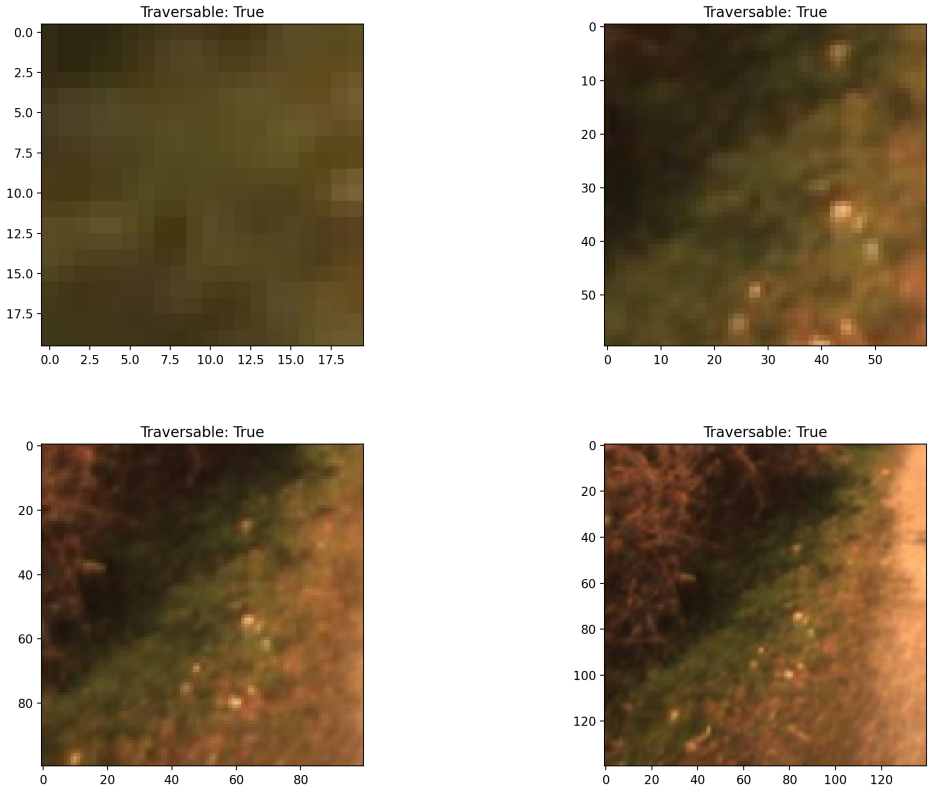


Figure 6.4: Training patches (cropped from terrain images) size comparison. Small patches lose some of the wider image semantics, and large patches can mis-represent class boundaries

It was found after some experimentation that the CNN performs best when it is trained using 100 pixel patches. This size was found to be accurate across the image as a whole and had an acceptable accuracy at the class boundaries. This compromise could be avoided in the future by adopting a semantic segmentation architecture, as discussed in the Network Design section above.

Chapter 7

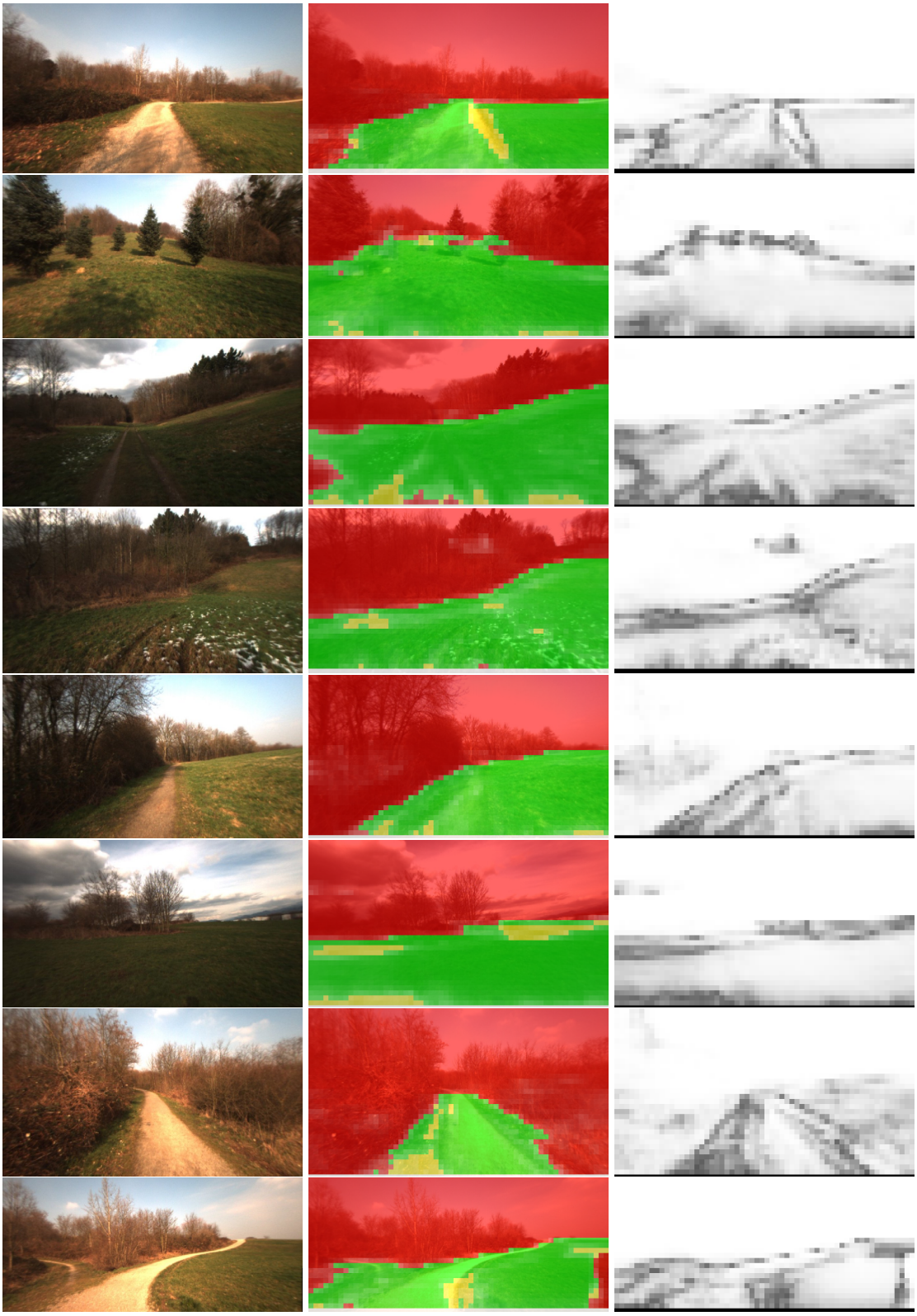
Results and Discussion

This chapter details the output and performance of the learning architecture after the CNN was trained using the vehicle-terrain interaction data set which was collected from simulations. Additionally an example of a potential path planning application of the framework is presented.

7.1 Uniform Friction Terrain

The first vehicle-terrain interaction simulation data that was used for training the CNN assumed uniform friction terrain. Doing this allowed the effects of terrain shape and terrain friction on the vehicle behaviour to be isolated and assessed individually. The friction plugin (see section 4.4) in the simulator was set so that any surface collision assumed that the friction coefficient was 1.0 between the vehicle and the terrain.

Simulation data for 366 individual terrain meshes and around 13 km of driving on uniform friction terrain was used to train the CNN. The output of the CNN when applied to test images can be observed in Figure 7.1 below.



(a) Input image (b) Traversability Class (c) Class confidence

Figure 7.1: Traversal prediction results of CNN trained with simulation data for uniform friction. Class colour: Green = fast traversal, yellow = slow traversal, red = un-traversable. Confidence colour: white = high, black = low

The system has successfully divided the terrain into traversable and un-traversable classes, and in almost all cases the trees and shrubs are classed as un-traversable and the road is classed as traversable, just as was found in the simulation data. Transition areas are also frequently predicted as slow, such as between grass and road, and between vegetation and grass. These are areas which typically have low class confidence and are moderately sloped (for example ruts on the sides of roads).

These results only indicate the effect of terrain shape on the vehicle's dynamics, so, while being somewhat useful, they are limited because they make no distinction between the effect of terrain textures, such as the surface or texture. This issue is addressed in the following section with the introduction of terrain friction.

7.2 Variable Friction Terrain

After the performance of the CNN was assessed for a uniform friction terrain the friction was varied in the simulation. This allowed the effect of terrain slip in the simulations to be isolated from the effect of terrain shape. The friction model was changed to use the following values:

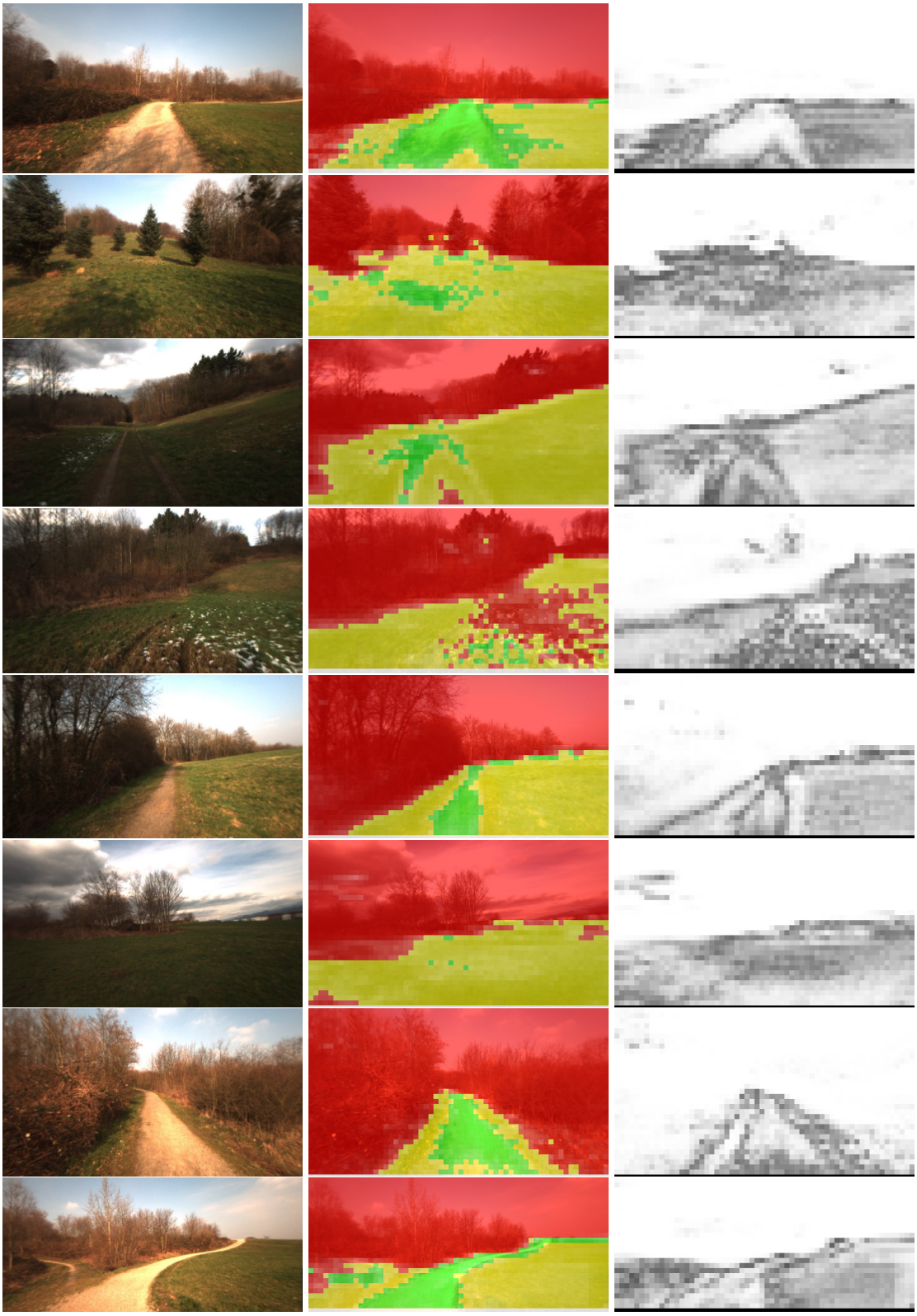
Table 7.1: Terrain friction coefficient values used in the simulation

Terrain	Sky*	Vegetation	Tree*	Grass	Road	Obstacle*
Friction μ	0.001	0.01	0.001	0.10	0.72	0.001

These friction values were loosely chosen from car tyre friction coefficient values over various terrains from the Engineering Toolbox (EngineeringToolBox, 2014), but it should be stressed that these friction values are arbitrary and intended as a proof of concept.

The results after training the CNN with variable friction data are observed in Figure 7.2.

*The sky, tree and obstacle used a very low friction rather than zero to avoid undefined simulation values



(a) Input image (b) Traversability Class (c) Class confidence

Figure 7.2: Traversal prediction results of CNN trained with simulation data with friction varied across the terrain. Class colour: Green = fast traversal, yellow = slow traversal, red = un-traversable. Confidence colour: white = high, black = low

Varying the friction across the terrain in the simulations has changed the CNN's prediction behaviour significantly from the uniform case. This suggests that the terrain simulation's friction model is successfully biasing the traversal results, and the self-supervised learning architecture is successfully learning it's environment.

The only regions predicted as fast traversal are roads and particularly flat grass patches. In most cases grass is predicted as slow traversal due to it's low coefficient of friction. The class confidence was found to be very low on the grassy regions, this is likely because the traversal data on grass is close to the class threshold between slow and fast. It is also observed on the fourth image down of Figure 7.2 that the class confidence is very low for snow on the ground, this is likely because snow is very under-represented in the data set.

7.3 CNN Performance

Since this system's intended use is for real-time robotics applications which utilise self-supervised learning, the performance of the CNN and the training properties were considered important design factors. The training and prediction would eventually have to take place on-board a robot, so computational efficiency will have reaching effects.

The CNN itself is not particularly deep nor complex, so the training time was not particularly large. The training time of a model from scratch was found to be around 9 hours using Google Colab (as mentioned above in section 6.2). It is expected that this computation time will be much lower when the model weights are pre-trained using a simulation environment, and then updated in situ as new terrain data is collected.

The CNN prediction time was recorded using a PC with an Nvidia GTX 660Ti GPU running Python 3. The system swept through the terrain image in steps sizes of 10 pixels and extracted the training patch for each step, this took around 15 seconds to collect an average 1500 patches per image. These patches were then evaluated using the CNN which

took a further 5 seconds. This performance is clearly not suitable for real time applications, and would require a large efficiency improvement such as patch prioritisation or the use of a semantic segmentation architecture.

7.4 Path Planning Example

A path planning example is presented to demonstrate how the traversability data collected from this framework could be used further down the pipeline. A simple Dijkstra algorithm is used which plans the shortest path to a goal position with traversability weightings applied between the node points.

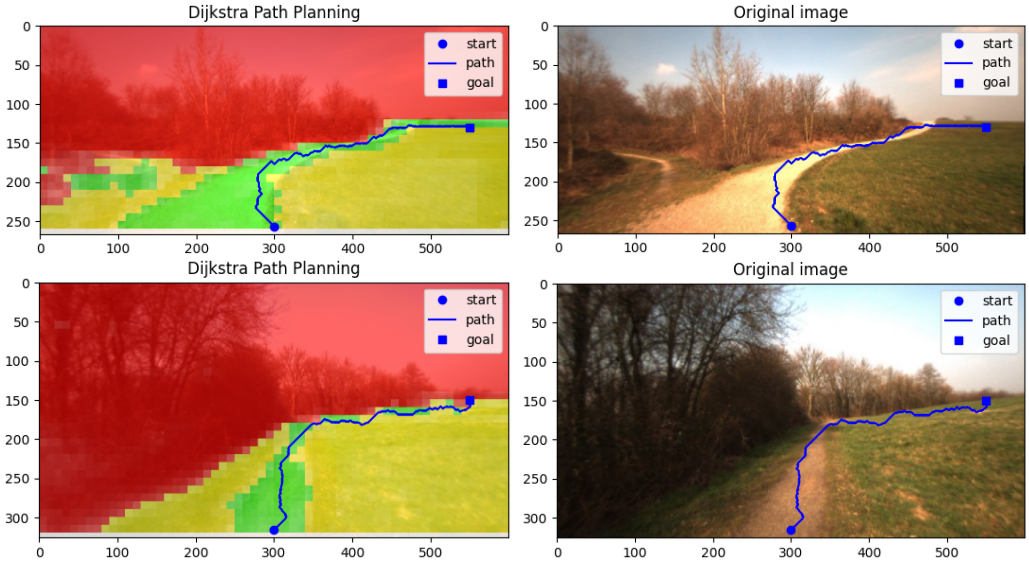


Figure 7.3: Dijkstra path planning algorithm with terrain traversability used as weightings between the graphs nodes.

The agent avoided the shortest route to the goal, which would involve crossing a slow patch of grass and road edge, in favour of staying on the road for as far as possible before crossing the grass to it's end goal. This behaviour can be easily tuned by weighting the traversability function.

Path planning is the most obvious use of this traversability knowledge, but other examples include energy conservation, danger avoidance and decision making.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis presented a self-supervised framework which learns vehicle-terrain interactions from experience. The relationship between the terrain and vehicle data, the so called 'traversability', was learned using a convolutional neural network (CNN). This learned relationship could be used to accurately predict the traversability of future terrains, which was found to be particularly useful for navigation tasks such as path planning.

A comparison was made between a framework trained using a uniform friction environment and a varied friction environment. This comparison illustrated that the classifier changes behaviour substantially as it's traversal environment changes. This characteristic is crucial for a system which is to adapt to it's terrain.

A method to generate simulated training environments from a real terrain data set was presented, which allowed the system to be pre-trained using real imagery and simulated vehicle dynamics, thus reducing the cost and safety issues of using a robot to learn from scratch.

8.2 Limitations and Future Work

There were a number of framework design issues identified which, if rectified, could expand the system's functionality significantly:

Online learning - The system currently only learns from a static training set and cannot learn dynamically as new data arrives. The addition of some mechanism to update the CNN incrementally as more data arrives would make the system more robust and capable of adapting to new environments on the fly.

Additional vehicle dynamics - The current system only learns the vehicle-terrain interaction for a vehicle at a constant speed and direction. The addition of the vehicle dynamics as an input to the CNN would allow the effect of vehicle states and configurations on the terrain traversability to be predicted.

More terrain classes - The traversal classification only used 3 discrete classes, which is not very useful for optimisation based tasks such as path planning. This classification could be more versatile if it used a higher resolution of classes or, better yet, a continuous class calculated from a regression CNN.

Improved traversability calculation - The traversability is calculated using only the vehicle's tilt and slip, which are rarely the only factors which constrain a vehicle over rough terrain. Some additional parameters could be added to its definition (suggested by Papadakis, 2013) - ground clearance, zero moment point distance, force-angle stability measure, distance stability margin or traction efficiency

Although this training framework is intended as pre-training for a real system, there are still several issues with using a simulation for the initial training:

Addition of noise - There is no noise or randomness assumed in the simulations, which can easily cause machine learning models to learn the simulation rather than

a robust and generalised model. The full extent of this effect will only be known when the model is deployed on a real system, but steps can be taken to make the model more robust, for example, artificial noise could be added to the simulations.

Safer learning maneuvers - Another issue with simulation based learning is that the system learns a lot from getting trapped or flipped in the simulation, but the same cannot be practically achieved for a real robot, especially when it is exploring a remote place such as Mars. To tackle this, the simulation could be modified to do the bulk of it's learning from safer maneuvers.

For the system to be to be successfully deployed on a real robot a number of design modifications would be required:

Real time classification - The current classifier takes around 20 seconds in total to divide an image into patches and classify each patch, which is an un-acceptable length of time for a robot operating in real time. To reduce the computation time it is expected that a CNN which uses a full image input would be required, such as semantic segmentation (as discussed in the Network Design section).

SLAM for labelling images - The simulation learning architecture knows the displacement between the previous camera position and present vehicle position at all points in time which allows the terrain images to be easily labelled, the same cannot be said for a real system. Some form of SLAM or Odometry would be required which can map the current vehicle position onto previous images of the terrain.

Unknown objects - The traversability classifier makes a prediction for all objects in the image regardless of the semantics. This works fine for terrains which are reasonably homogeneous such as forests and fields, but quickly breaks down in highly novel terrains with a lot of unusual objects such as urban environments. Some mechanism is therefore required which identifies features which the CNN is unsure about, and indicates that the agent should proceed with caution.

References

Angelova, Anelia et al. (2007). "Learning and prediction of slip from visual information". In: *Journal of Field Robotics* 24.3, pp. 205–231. DOI: 10.1002/rob.20179. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20179. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20179.

Bo, Liefeng and Kevin Lai (2014). *RGB-D (Kinect) Object Dataset*. URL: https://rgbd-dataset.cs.washington.edu/software.html.

Chavez-Garcia, R. Omar et al. (2017). "Learning Ground Traversability from Simulations". In: *CoRR* abs/1709.05368. arXiv: 1709.05368. URL: http://arxiv.org/abs/1709.05368.

Chollet, François et al. (2015). *Keras*. https://keras.io.

Cignoni, Paolo et al. (2008). "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by Vittorio Scarano, Rosario De Chiara, and Ugo Erra. The Eurographics Association. ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

Cole, Michael et al. (2019). "Are M&S Tools Ready for Assessing Off-road Mobility of Autonomous Vehicles?" In:

EngineeringToolBox (2014). *Friction and Friction Coefficients*. URL: https://www.engineeringtoolbox.com/friction-coefficients-d_778.html.

Garcia Bermudez, F. L. et al. (2012). "Performance analysis and terrain classification for a legged robot over rough terrain". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 513–519.

Gazebo-Tutorials (2015a). *Tutorial: Friction*. URL: http://gazebosim.org/tutorials?tut=friction.

— (2015b). *Tutorial: Using Gazebo plugins with ROS*. URL: http://gazebosim.org/tutorials?tut=ros_gzplugins.

Google (2018). *Google Colaboratory*. URL: https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index#scrollTo=5fCEDCU_qrC0.

Helmick, Daniel, Anelia Angelova, and Larry Matthies (2009). "Terrain Adaptive Navigation for planetary rovers". In: *Journal of Field Robotics* 26.4, pp. 391–410. DOI: 10.1002/rob.20292. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20292. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20292.

Howard, Andrew et al. (2006). "Towards learned traversability for robot navigation: From underfoot to the far field". In: *Journal of Field Robotics* 23.11-12, pp. 1005–1017. DOI: 10.1002/rob.20168. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20168. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20168.

Howard, Ayanna and Homayoun Seraji (2001). "Vision-based terrain characterization and traversability assessment". In: *Journal of Robotic Systems* 18.10, pp. 577–587. DOI: 10.1002/rob.1046. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.1046. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.1046.

Jackel, L. D. et al. (2006). "The DARPA LAGR program: Goals, challenges, methodology, and phase I results". In: *J. Field Robotics* 23, pp. 945–973.

Jaillet, L., J. Cortés, and T. Siméon (2010). "Sampling-Based Path Planning on Configuration-Space Costmaps". In: *IEEE Transactions on Robotics* 26.4, pp. 635–646.

Jain, A. et al. (2004). "Recent developments in the ROAMS planetary rover simulation environment". In: *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*. Vol. 2, 861–876 Vol.2.

Jakobi, Nick, Phil Husbands, and Inman Harvey (Jan. 1995). ""Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics,"". In: vol. 929, pp. 704–720.

Jaritz, Maximilian et al. (2018). *Sparse and Dense Data with CNNs: Depth Completion and Semantic Segmentation*. arXiv: 1808.00769 [cs.CV].

Kahn, Gregory, Pieter Abbeel, and Sergey Levine (2020). *BADGR: An Autonomous Self-Supervised Learning-Based Navigation System*. arXiv: 2002.05700 [cs.RO].

Koenig, Nathan and Andrew Howard (2004). "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, pp. 2149–2154.

Leppanen, I. M., P. J. Virekoski, and A. J. Halme (2008). "Sensing terrain parameters and the characteristics of vehicle-terrain interaction using the multimode locomotion system of a robot". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 500–505.

Martin Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: http://tensorflow.org/.

Maturana, Daniel et al. (Jan. 2018). "Real-Time Semantic Mapping for Autonomous Off-Road Navigation". In: pp. 335–350. ISBN: 978-3-319-67360-8. DOI: 10.1007/978-3-319-67361-5_22.

Moravec, H. and A. Elfes (1985). "High resolution maps from wide angle sonar". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2, pp. 116–121.

NASA (2010). *NASA to Begin Attempts to Free Sand-Trapped Mars Rover*. URL: https://www.nasa.gov/mission_pages/mer/news/mer20091112.html.

Otsu, K. et al. (2016). "Autonomous Terrain Classification With Co- and Self-Training Approach". In: *IEEE Robotics and Automation Letters* 1.2, pp. 814–819.

Pandas (2020). *Pandas.Dataframe*. URL: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html.

Papadakis, Panagiotis (2013). "Terrain traversability analysis methods for unmanned ground vehicles: A survey". In: *Engineering Applications of Artificial Intelligence* 26.4, pp. 1373–1385. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2013.01.006. URL: http://www.sciencedirect.com/science/article/pii/S095219761300016X.

ROS-wiki (2017). *Pioneer 3-at*. URL: https://robots.ros.org/pioneer-3-at/.

Rosique, Francisca et al. (Feb. 2019). "A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research". In: *Sensors* 19.3, p. 648. ISSN: 1424-8220. DOI: 10.3390/s19030648. URL: http://dx.doi.org/10.3390/s19030648.

Shneier, Michael et al. (Nov. 2008). "Learning traversability models for autonomous mobile vehicles". In: *Auton. Robots* 24, pp. 69–86. DOI: 10.1007/s10514-007-9063-6.

Stanford Artificial Intelligence Laboratory et al. (May 23, 2018). *Robotic Operating System*. Version ROS Melodic Morenia. URL: https://www.ros.org.

Tarvainen, Antti and Harri Valpola (2017). "Weight-averaged consistency targets improve semi-supervised deep learning results". In: *CoRR* abs/1703.01780. arXiv: 1703.01780. URL: http://arxiv.org/abs/1703.01780.

Valada, Abhinav et al. (2016). "Deep Multispectral Semantic Scene Understanding of Forested Environments using Multimodal Fusion". In: *International Symposium on Experimental Robotics (ISER)*.

Wellhausen, L. et al. (2019). "Where Should I Walk? Predicting Terrain Properties From Images Via Self-Supervised Learning". In: *IEEE Robotics and Automation Letters* 4.2, pp. 1509–1516.