Traversability Estimator for a Legged Robot

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Artificial Intelligence

presented by
Francesco Saverio Zuppichini

under the supervision of
Prof. Student's Alessandro Giusti
co-supervised by
Prof. Student's Omar Chavez-Garcia

June 2019

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Francesco Saverio Zuppichini
Lugano, 26 June 2019

Abstract

Effective identification of traversable terrain is essential to operate mobile robots in different environments. Historically, to estimate traversability, texture and geometric features were extracted before training a traversability estimator in a supervised way. However, with the recent deep learning breakthroughs in computer vision, terrain features can be learned directly from raw data, images or heightmaps, with higher accuracy.

We implement a full pipeline to estimate traversability that first generates a dataset entirely through simulation and then it trains a deep convolutional neural network to predict traversability with high accuracy. The method is based on the framework proposed by Chavez-Garcia et al. [5] and was originally tested on wheeled small robot. Collecting data in a simulation environment yields several advantages over the real world. The maps can be easily generated to include any features and maximize the robot exploration. Multiple simulations can be run in parallel reducing the cost for real-world hardware and increasing the dimension of the dataset. Real robots can be broken, affected by weather and failures.

The dataset was generated using thirty synthetic surfaces with different features such as slopes, holes, bumps, and walls. Then, we let the robot walks on each map on for a certain amount of time while storing its interactions. Later, we precisely crop a patch from each simulation trajectory's point that includes the whole robot's footprint and the amount of ground it should reach without obstacles in a decided time window. Then, we label each patch as traversable or not traversable based on a minimum advancement depending on the robot used. The framework is open source and the same methodology can be adopted with any type of ground robots. We test the methodology on a legged crocodile-like robot that presents challenging locomotion.

We propose and test different residual networks architecture and select the best performing one. We shows its performance with different numeric metrics on various real-world terrains. In addition, we qualitatively evaluate the network by showing the predicted traversability probability on different grounds to better visualize the acquired knowledge.

Later, we utilize model interpretability techniques to understand the network's strengths and limitations. We show its ability to properly separate samples based on their features and we discover which patches confuse the network. We visualize the ground regions that the network fails to classify and find which portion of the inputs is responsible for the wrong predictions. Finally, we test the model strength and robustness by comparing its prediction on custom patches composed by crafted features, such as a patch with a wall ahead with the ground truth obtained by running again Krock on that ground in the simulator. The results suggest that the model was able to match the ground truth in different situations.

Acknowledgements

I would like to thank my family for always supporting me. All my friends from USI and Verona. Especially Dario, one of the most positive person that I know, that helped during my Thesis always with a big smile. Thanks to Alessia and her big patience. To Nicola, Federico, Paolo and Alessandro to always make my weekends here in Italy specials. Thank you guys for making me laughing since high school.

To my advisor and co-advisor, Professor Giusti and Omar Chavez-Garcia to give me this amazing opportunity to work on a project I loved. Thank you Omar to helped me out through all these months, I could have been there without you.

Finally I would like to thank all the professors who taught me unforgettable subjects.

Contents

Chapter 1

Introduction

All living beings, including humans, need to traverse ground to eat, sleep and breed. Animals through millennia of evolution have developed different techniques to classify ground regions. Usually, local sensors are used to map new terrain to effectively navigate the environment. Humans, for instance, are equipped with a powerful traversability estimator, called vision, able to extract features from a local region, a patch, such as elevation, size and steepness and combine them to produce a local planner able to find a suboptimal path to cross.

Similarly, ground robots need to walk in order to fulfill their tasks. Taking inspiration from nature, robots must be able to correctly aggregate terrain features to predict their traversability. In this project, we consider the task to estimate the traversability of 3D terrain for a specific ground robot. Traversability can be estimated by 3D geometric, appearance features or both [17]. The first one utilizes purely geometric features to label a patch, while the second relies on categories, glass, rocks, etc, with different traversability probabilities extract mostly on camera's images. While appearance data must be collected directly from a real or simulated environment, geometric features do not directly depend on the robot's interactions with the ground. In both cases, these samples are used in supervised learning to train a traversability estimator.

In most indoor scenarios, data is collected through local sensors during a robot exploration of the environment. Thanks to their artificial design, indoor environments share similar features across the world, the flatness of the floor, stairs, and ceil. So, even if an agent is trained with samples collected in one location, due to physical limitations, the learned mapping can be still effective in different locations. Moreover, in indoor environments, traversability estimation is mostly solved with obstacle avoidance since, by design, the floor alone do not include bumps, ramps or holes. In addition, while the robot operates there may be other agents interacting with the environment, such as humans, therefore a correct function to avoid collision must properly learn to effectively move safely the robot.

On the other hand, outdoor terrains may have less artificial obstacles but they don't have a homogeneous ground making more challenging to estimate where the robot can properly travel. Also, a given patch with a specific shape may not be traversable in all direction due to the non uniform features of the surfaces. On top of that, using real-world data may be unfeasible due to the time required to move the robot on different grounds and the possibility to introduce bias if the data samples are not varied enough. Fortunately, the ground can be generated artificially and used in a simulated environment to let the robot walk in it while storing its interactions. Moreover, grounds maps can be easily obtained by using third-party services such as google maps or flying drones

equipped with mapping technologies.

We propose a full pipeline to estimate traversability tested on a legged crocodile-like robot on uneven terrain based entirely on data gathered through simulation. Data collection through simulation offers several advantages such as cost, quality and time. We generate thirty synthetic maps encoded as heightmaps with different features: bumps, walls, slopes, steps, and holes. Then, we run the robot on each one of them for a certain amount of time while storing its position and orientation and its interactions with the environment. Later, we generate a training dataset by cropping a patch for each trajectory in a way that includes the robot's footprint and the maximum possible amount of ground that can be traversed in a selected time window. We label each patch using a threshold that depends on the robot's locomotion. These patches are used to fit a deep convolutional neural network to prediction the traversability.

This thesis is organized as follow, the in chapter 2 introduces the related work. In chapter 3 gives a high-level overview of our approach. Chapter 4 carefully explains the implementation details showing the dataset generation procedure and describes different convolutional neural network architectures. Chapter 5 discuss the model selection process, showing both quantitively and qualitatively results for the best performing architecture. In chapter 6 we evaluate the model's robustness using different techniques. We prove the network's ability to learn meaningful features from the inputs, section 6.1. Then, 6.2 we visualize different inputs to understand which grounds' regions caused the wrong predictions. Lastly, in section 6.3, we test the model's robustness by creating different patches with unique characteristics. We compared the estimator's predictions to the ground truth gather the simulator. In Chapter 7 we draw the conclusion, marking some limitations of the method and describing how to tackl them.

Chapter 2

Related Work

The learning and perception of traversability is a fundamental competence for both organisms and autonomous mobile robots since most of their actions depend on their mobility [23]. Usually, visual perception is used in most animals to correctly estimate if an environment can be traversed or not. Similar, a wide array of autonomous robots adopt local sensors to mimic the visual properties of beings to extract meaningful information of the surrounding and plan a safe path through it. This chapter gives an overview of some of the existing traversability estimator approaches.

Most of the methodologies used to estimate traversability rely on supervised learning. Supervised learing is a techinique in which first the data is gathered and then a machine learning algorithm is trained to correctly predict the ground truth. These approaches can be clustered into two categories based on the inputs data: geometric and appearance based methods.

2.1 Geometric methods

Geometric methods aim to detect traversability using geometric properties of surfaces such as distances in space and shapes. These properties are usually slopes, bumps, and ramps. Since nearly the entire world has been surveyed at 1 m accuracy [22], outdoor robot navigation can benefit from the availability of overhead imagery, for this reason, recently, elevation data has been widely used to generate datasets. Chavez-Garcia et al. [5], proposed a framework to learn traversability using elevation data as input in the form of height maps. They trained a convolutional neural network to predict from a dataset, entirely generated through simulation, the traversability probability of a small ground region around the robot.

Elevation data can also be estimated by flying drones. Delmerico et al. [7] proposed a collaborative search and rescue system in which a flying robot that explores the map and creates an elevation map to guide the ground robot to the goal. They train on the fly a CNN to segment the terrain in different traversable classes. This map is later used to plan a path for the mobile robot.

2.2 Appearance methods

Appearance methods, to a greater extent related to camera images processing and cognitive analyses, have the objective of recognizing colors and patterns not related to the common appearance of terrains, such as grass, rocks or vegetation. Historically, the collected data is first preprocessed

to extract texture features that are used to fit a classic machine learning classified such us an SVM [23] or Gaussian models [22]. These techniques rely on texture descriptors, for example, Local Binary Pattern [16], to extract features from the raw data images obtained from local sensors such as cameras. With the rise of deep learning methods in computer vision, deep convolution neural network has been trained directly on the raw RGB images bypassing the need to define characteristic features.

2.3 Mixed methods

One recent example is the work of Giusti et al. [3] where a deep neural network was training on real-world hiking data collected using head-mounted cameras to teach a flying drone to follow a trail in the forest. Geometric and appearance methods can be used together, one example is the work of Delmerico et al.[6] extended the previous work [7] by proposing the first on-the-spot training method that uses a flying drone to gather the data and train an estimator in less than 60 seconds.

Also, Data can bed extracted in simulations, in which an agent interacts in an artificial environment. Usually, no-wheel legged robot able to traverse harder ground, can benefits from data gathering in simulations due to the high availability. For example, Klamt et al. [12] introduced a locomotion planner that is learned in a simulation environment.

2.4 Legged robots

On other major distinction in the field, is between different types of robots: wheeled and no-wheeled. We will focus on the later since we adopt a legged crocodile-like robot to extend the existing framework proposed by Chavez-Garcia et al. [5].

Legged robots show their full potential in rough and unstructured terrain, where they can use a superior move set compared to wheel robots. Different frameworks have been proposed to compute safe and efficient paths for legged robots. Wermelinger et al. [24] use typical map characteristics such as slopes and roughness gather using onboard sensors to train a planner. The planner uses a RRT* algorithm to compute the correct path for the robot on the fly. Moreover, the algorithm is able to first find an easy local solution and then update its path to take into account more difficult scenarios as new environment data is collected.

Due to uneven shape rough terrain, legged robots must be able to correctly sense the ground to properly find a correct path to the goal. Wagner et al. [13] developed a method to estimate the contact surface normal for each foot of a legged robot relying solely on measurements from the joint torques and from a force sensor located at the foot. This sensor at the end of a leg optically determines its deformation to compute the force applied to the sensor. They combine the sensors measurement in an Extended Kalman Filter (EKF). They showed that the resulting method is capable of accurately estimating the foot contact force only using local sensing.

While the previous methods rely on handcrafted map's features extraction methods to estimate the cost of a given patch using a specific function, new frameworks that automatize the features extraction process has been proposed recently. Lorenz et al. [15] use local sensing to train a deep convolutional neural network to predict the terrain's properties. They collect data from robot ground interaction to label each image in front of the robot in order to predict the future interactions with the terrain showing that the network is perfectly able to learn the correct features for different terrains. Furthermore, they also perform weakly supervised semantic segmentation using the same

approach to divide the input images into different ground classes, such as glass and sand, showing respectable results.

Chapter 3

Methodology

We develop a framework to learn traversability directly on ground patches for any robot using purely simulated data. The core idea, originally proposed by Chavez-Garcia et al. [5], is simple yet elegant, a robot is let travel forward with a fix controller into a simulator on different synthetic terrains. While moving it stores its interactions with the environment, namely pose and orientation. Then we crop a region of ground, a patch, around each traveled position and directly train a neural network on them to predict traversability. Figure 3.1 shows the framework's steps.

3.1 Data gathering

Collecting data through simulation has several main advantages such as variety, cost, and speed. With simulations, we can easily increase the number of data samples at any time by simply incorporating more maps or run more experiments. Contrarily, a real robot requires to first identify the suitable ground and then physically transport the machine there and finally let it walk on it. Also, real robots cannot completely explore all grounds due to the possibility to be damaged, especially in outdoors scenarios preventing the full exploration of the environment.

On the other hand, a virtual robot can be destroyed and regenerate indefinitely allowing total exploration of all not traversable samples, essential to generate a quality dataset. Of course, a realistic model controller must be implemented to simulate the real robot. Inside a simulation we are not limited by real constraint and we can design the ground to maximize the variety of robot's interactions. For instance, we can artificial design maps to include any kind of situations and challenges. Clearly, this also reduces the time required to collect the robot's interactions with the terrain. The time could be further minimized by running different simulations in parallel bypassing the demand for more physical hardware.

To create a quality dataset, we must generate a series of various surfaces and ensure their diversity. Intuitively, these maps should be small enough to allow fast exploration but big enough to include several features. Modern ground generation techniques allow to generate a rich array of terrains with different characteristics such us bumps, ramps, slopes in different levels and sizes. We adopted the same ground generation procedured proposed in the original work [5]. Figure 3.2 shows a small collection of synthetic terrains with different features we created. We represented the terrains with heightmaps, where the height component of the ground is stored as pixels in a gray image. In other worlds, these image are a top-view representation of the terrain and theirs brightness describes the height.

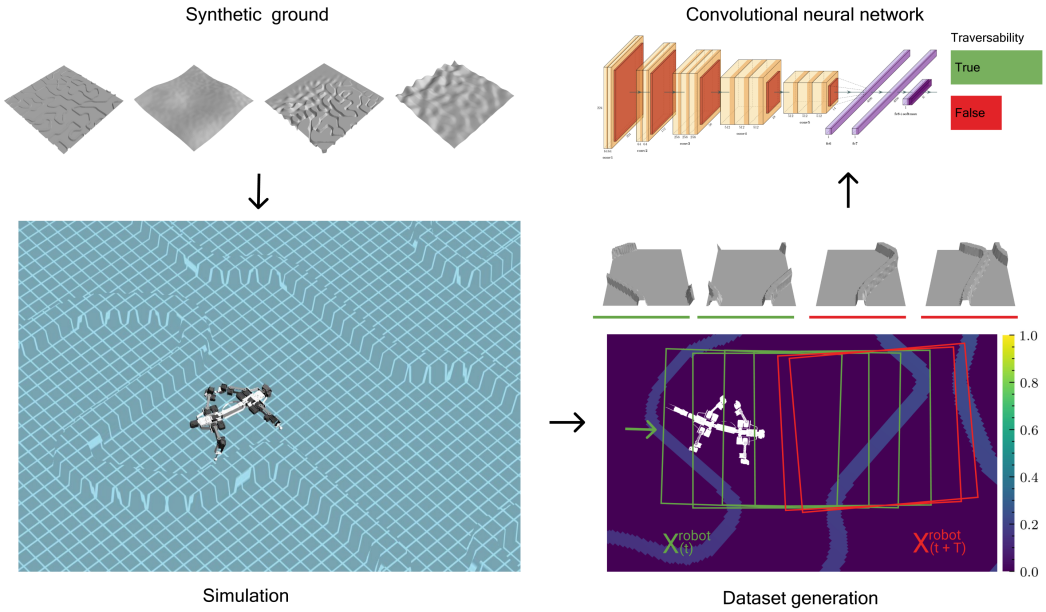
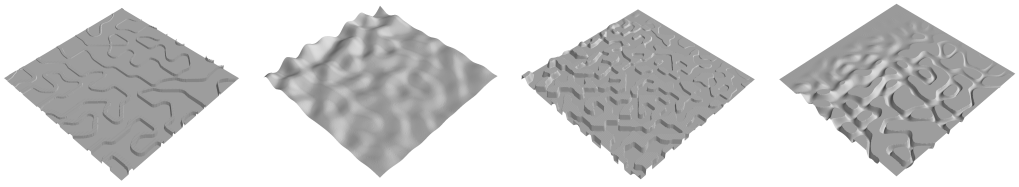


Figure 3.1. The framework's main building block in counter-clockwise order. The robot is a legged robot crocodile-like. First, we generated meaningful synthetic grounds. Then, the robot randomly spawns and walks forward with a fixed controller on the maps inside a simulated environment. While moving, it stores its interactions. Then, we crop a region of ground, a patch, for each simulation trajectory around the robot. The patch's size is calculated according to its locomotion. We labeled these images using a defined threshold. Finally, we trained a deep convolutional neural network to predict traversability.



(a) Walls. (b) Bumps. (c) Steps. (d) Rails.

Figure 3.2. Some artificially generated terrains, 10×10m, with one specific feature each.

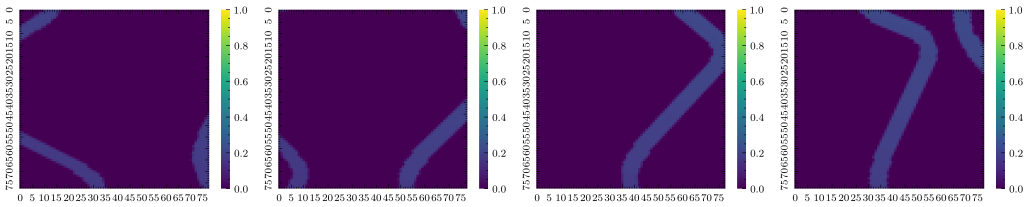
Robots interacts with the maps when it is loaded on the simulator. To ensure a correct exploration of each terrain, we randomly spawn the robot multiple times on the same map and let it walk for a fixed amount of time. Random spawn reduces the possibility to repeatedly visit the same spot and it maximizes the robot's exploration.

The robot position is tracked and stored during simulation to extract a small portion of ground, a patch, around each of its trajectory's position. The patches are cropped from the heightmap using a specific size based on the robot's footprint and its velocity. Each resulting image is labeled using a minimum advancement based on the robot's characteristic. We tested our framework on a legged crocodile-like robot called *Krock*. Due to its four legs, the robot has very unique locomotion allowing it to overcome different obstacles making estimate traversability more challenging. Figure 3.3c helps to visualize the procedure. The dataset generation process is detailed explained in section 4.2.

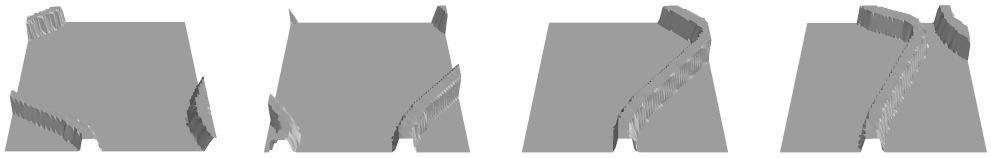
3.2 Estimate traversability

The dataset generated is used to train a deep convolutional neural network. The model estimates traversability from images representing small ground regions. We improved the original architecture by adopting a ResNet variant based on recent works [8] [9] [10]. We methodologically tested different variations of the new architecture to finally generate a model with three-time fewer parameters than the network proposed by the original paper [5] with comparable performance. Due to its smaller size, the network has a lower prediction time than the original one. Thus, it is can be easily deployed using less demanding hardware. We appraised the models' performance both quantitatively, with numerical metrics, and qualitatively, showing the traversability estimation on real-world terrains.

(a) Robot's trajectory.



(b) 2D patches topview representation.



(c) 3D patches representation.

Figure 3.3. Example of a robot's trajectory (a) extracted during training on a map with different walls. Robot's initial position is highlighted by its white silhouette. Patches borders are labeled with greed if traversable and red if not and showed as 2D (b) and 3D(c) rendered images. The robot traverses the patches from left to right.

Chapter 4

Implementation

This section talks in detail about the implementation of the framework. We will first describe the utilized robot, then discuss each step in the dataset generation. After, we describe five convolutional neural network architectures. We conclude by listing the employed software.

4.1 Robot

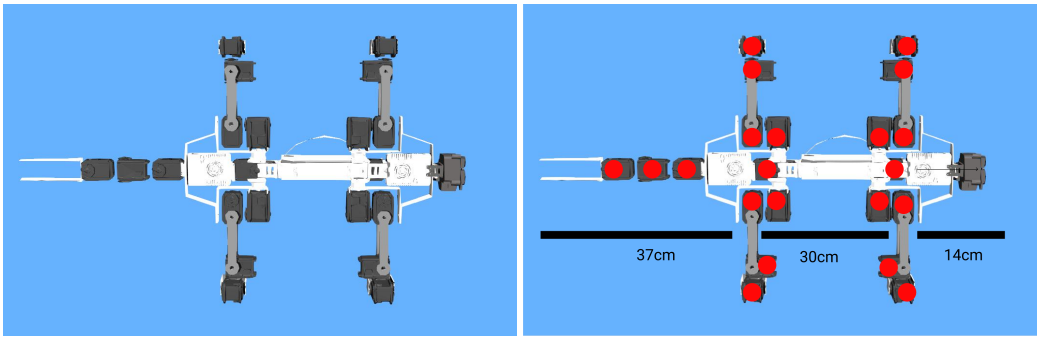
We test our framework on a legged crocodile-like robot called *Krock* developed at EPFL. Figure **??** shows the robot in real life with a cloak and in a simulated environment.

(a) Krock in real life. (b) Krock in the simulator.

Figure 4.1. *Krock*

K-rock robot was created for the purpose of monitoring real crocodiles, hence it must walk and behave realistically enough to fool the real crocodiles. Krock has four legs, each one of them is equipped with three motors in order to rotate in each axis. In addition, there is another set of two motors in the torso to increase Krock's mobility. These motors can change the robot's gait in three different configurations. The tail is composed by another set of three motors and can be used to perform a wide array of tasks. The robot is $85cm$ long, weights around $1.4kg$ and in its standard

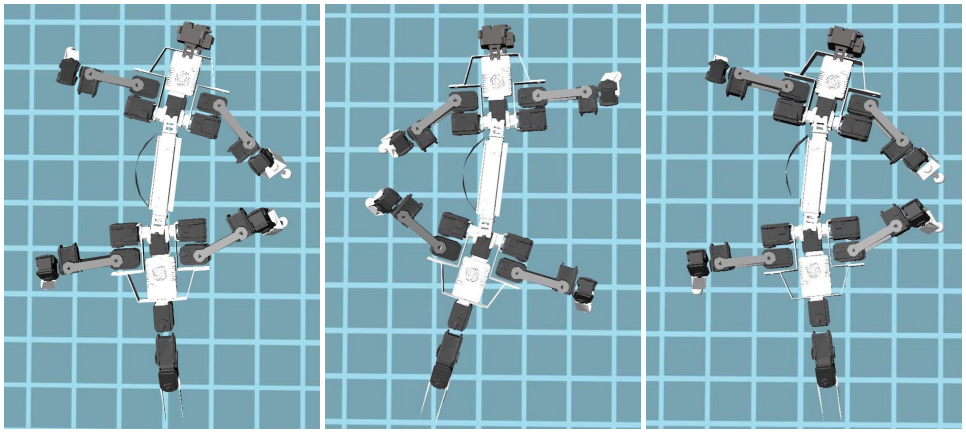
gait configuration its distance from the ground is 16cm. Figure 4.2 shows a topview of the robot. *Krock*'s moves by lifting and moving forward one leg after the other. The following figure shows



(a) Top view of *Krock* (b) Details highlighted in *Krock*

Figure 4.2. Top view of Krock to help the reader better understand its composition and the correct ratio between its parts. Each motor is highlighted with a red marker.

the robots going forward. In our framework we fix the gait configuration to normal, showed in



(a) $t = 1s$ (b) $t = 2s$ (c) $t = 3s$

Figure 4.3. *Krock* moving forward. In this gait configuration, the robot distance from the ground is 16cm. Approximately every second the robot moves one leg.

figure 4.1, where the body is at the same legs' motors height.

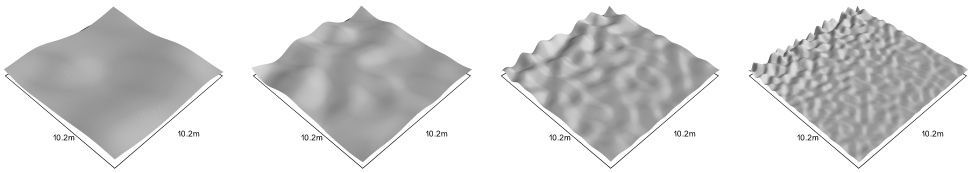
4.2 Data gathering

This section describes in detail how we create, collect and process the synthetic dataset used to train the traversability estimator with supervised learning.

4.2.1 Ground generation

We create thirty 10×10 maps with a resolution offset $0.02cm/pixel$ and a height of 1m using 2D simplex noise variant of Perlin noise [1], a widely used technique in the terrain generation litterature. We divide the maps into five main categories based on ground features: $bumps$, $rails$, $steps$, $slopes$/$ramps$ and $holes$.

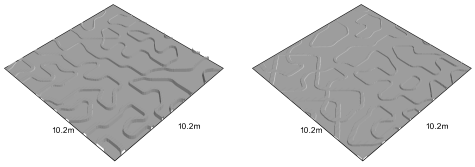
Bumps: We generete four different maps with increasing bumps' height using simplex noise.



(a) *bumps0* (b) *bumps1* (c) *bumps2* (d) *bumps3*

Figure 4.4. Bumps maps.

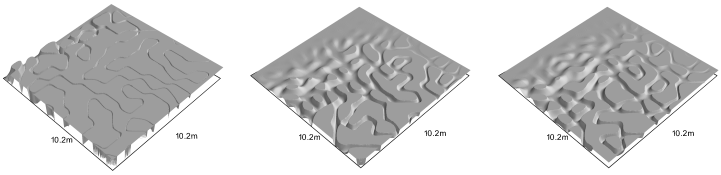
Bars: In these maps there are wall with different shapes and heights.



(a) *bars1* (b) *bars3*

Figure 4.5. Bars maps.

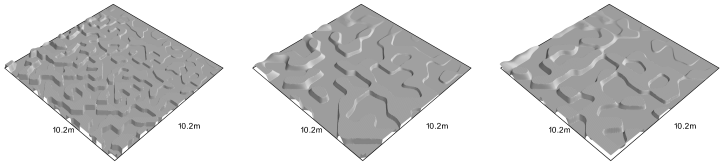
Rails: Flat grounds with slots.



(a) *rails1* (b) *rails2* (c) *rails3*

Figure 4.6. Rails maps.

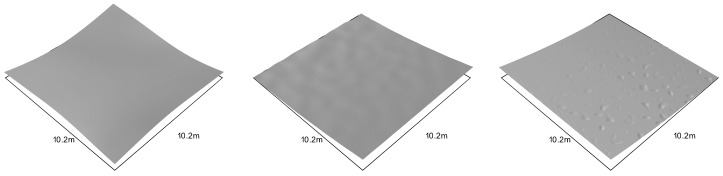
Steps: Maps with various steps at increasing distance and frequency.



(a) *steps1* (b) *steps2* (c) *steps3*

Figure 4.7. Steps maps.

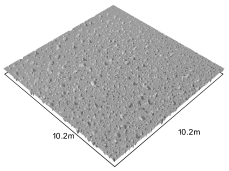
Slopes/Ramps: Maps composed by uneven terrain scaled by different height factors from 3 to 5 used to include samples where *Krock* has to climb.



(a) *ramp0* (b) *ramp1* (c) *slope_rocks1*

Figure 4.8. Slopes maps.

Holes We also included a map with holes



(a) *holes1*

Figure 4.9. Holes map.

Arc rocks We also crafted an other 10×10m map with a big rocky bump in the middle.

(a) *holes1*

Figure 4.10. Arc rocks map.

4.2.2 Real world maps

We also include real word terrains. We gather two heightmaps produced by ground mapping flying drones from sensefly's dataset, a quarry and a small village. Figure 4.11 shows the original and a 3D render of the heightmap for each terrain.

4.3 Simulation

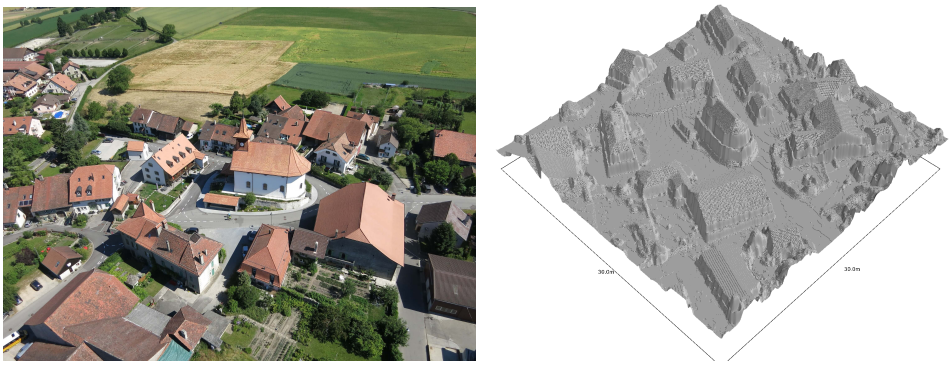
4.3.1 Setup

We use Webots as the simulation platform for the robot model and the generated terrains. The robot's controller is implemented using the Robotic Operating System, it sends data using special nodes called topics. A ROS' node embedded in the controller publishes Krock status, including its pose, at a rate of $250hz$. Due to the high value, we decide to reduce it to $50hz$ by using ROS build it `throttle` command.

4.3.2 Spawing

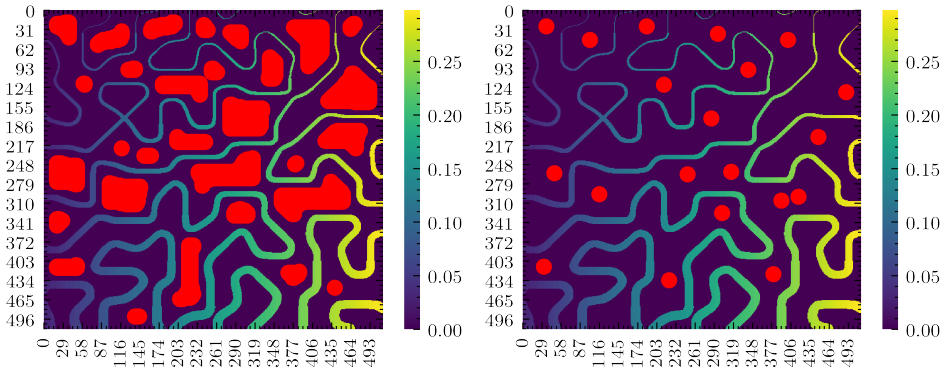
To collect Krock's Interactions with the environment, we spawn the robot on the ground and let it move forward for t seconds. We repeat this process n times per map. Unfortunately, spawning the robot is not a trivial task. For instance, in certain maps we must avoid spawning Krock on an obstacle otherwise the robot will be immediately stuck introducing noise in the dataset. To solve this problem, we define two spawn strategies, a random spawn, and a flat ground spawn strategy. The first one is employed in most of the maps without big obstacles. This strategy just spawns the robot in a random position and orientation. On the other hand, the flat ground strategy first selects suitable spawn positions by using a sliding window on the heightmap of size equal Krock's footprint and check if the mean pixel value is lower than a small threshold. If so, we store the center coordinates of the patch as a candidate spawning point. Intuitively, if a patch is flat then its mean value will be close to zero. Since there may be more flat spawning positions than simulations needed, we have to reduce the size of the candidate points. To maintain the correct distribution on the map to avoid spawning the robot always in the same cloud of points, we used K-Means with k clusters where k is equal to the number of simulations we wish to run. By clustering, we guarantee to cover all region of the map removing any bias. Picture **??** shows this strategy on *bars1*.

(a) Quarry 32×32m



(b) A small village. 20×20

Figure 4.11. Real world maps obtained from sensefly's dataset. The left images show the real world location, the right images a render in 3D of theirs heightmaps. Both images have a maximum height of 10m.



(a) Flat regions (b) K-Means with $k = 20$

Figure 4.12. An exampls of the flat spawning selection process (marked as red blobs) for the map *bars1*

4.3.3 Interactions

In each simulation, the robot is spawned using the two spawning strategies and it walks forward for a fixed amount of time. We select ten or twenty seconds depending on the map. Every five simulations we completely regenerate the robot's body to counter possible damages caused the hard traversable ground. We mov the robot only forward with a fixed controller since we cannot estimate if the robot can traverse a patch while moving sideways. Table 4.1 shows the configuration of the maps used in the simulator. For some terrain, we add three different rocky textures to create more complicated situations. For example, adding some rocks in the ramps map allowed the robot to anchor the legs to the rocks and climb them easily.

4.4 Postprocessing

To generate the dataset, we need to extract we had to extract the patches at each robot's pose of the trajectory generated at simulation time. Figure **??** shows each step in the postprocess pipeline.

4.4.1 Parse trajectories

First, we convert the robot's trajectories, stored as *.bag* files, to a more convenient data structure, pandas' dataframes. Then, we cache them into *.csv* files. This operation is done only once. After, we load the stored dataframes with the respective terrains and start the data cleaning process. We remove the first simulation's second to account for the robot spawning lag. Then, we eliminat all the entries where a part of Krock was outside the edges of the map. After the data cleaning process, we convert the robot's quaternion rotation to Euler notation using the `tf` package from ROS. Then, we extract the $\sin$ and $\cos$ from the Euler's last components and store them in one dataframe's column.

In the simulator, the position $(0, 0)$ correspond to the center of the map, while on the heightmap, like all images, $(0, 0$ is the top left corner. To later crop the the correct region from the map, we needed to convert the robot's position into the heightmap's coordinates. This part is also done once and it is cached to speed up latter runs.

Map	Height(m)	Spawn	Texture	Simulations	Max time(s)	Dataset
bumps0	2	random	-	50	10	Train
			rocks1			Train
			rocks2			Train
bumps1	1	random	-	50	10	Train
			rocks1			Train
			rocks2			Train
bumps2	1	random	-	50	10	Train
			rocks1			Train
			rocks2			Train
	2		-			Train
bumps3	1	random	-	50	10	Train
			rocks1			Train
			rocks2			Train
steps1	1	random	-	50	10	Train
steps2	1	flat	-	50	10	Train
steps3	1	random	-	50	10	Train
rails1	1	flat	-	50	20	Train
rails2	1		flat	-	10	Train
rails3	1		flat	-	10	Train
bars1	1	flat	-	50	10	Train
	2		-			Train
bars3	1	flat	-			Train
ramp0	1	random	rocks1	50	10	Train
			rocks2	50		Train
ramp1	3	random	-	50	10	Train
	4					Train
	5					Train
slope_rocks1	3	random	-	50	10	Train
	4					Train
	5					Train
holes1	1	random	-	50	10	Train
quarry	10	random	-	50	10	Test
arc rocks	1	random	-	50	10	Evaluation

Table 4.1. Maps configuration used in the simulator.

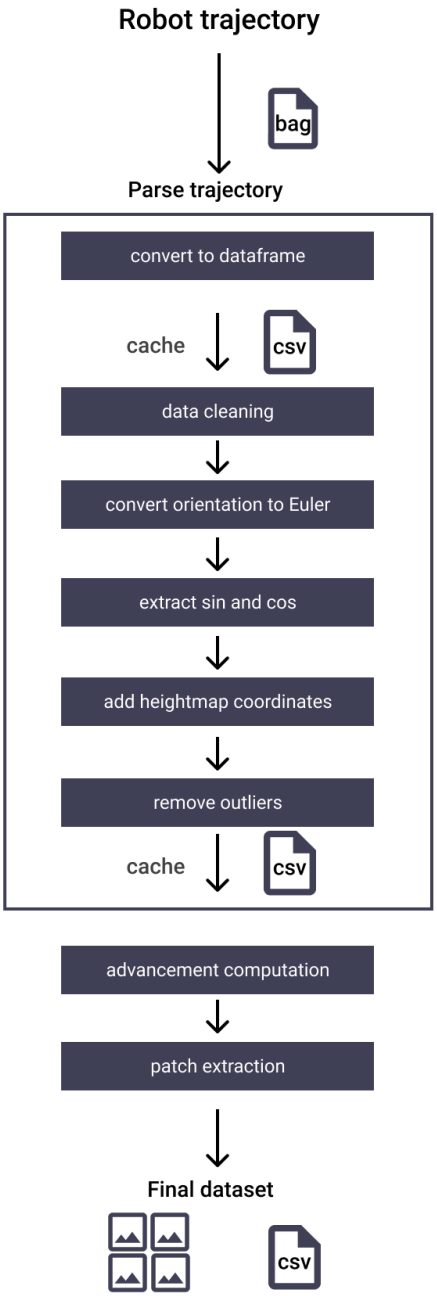


Figure 4.13. Postprocessing pipeline flow graph, starting from the top.

4.4.2 Advancement computation

To compute the robot's advancement we have to to define a time window, Δt, and consider how much the robot traveled between the currenct pose, $X(t)$ and the future one $X(t + \Delta t)$. We empirically observed in the simulator that the robot moves one leg every second. Thus, we select a time window of two seconds to both left and right legs of Krock to move once.

4.4.3 Patch extraction

Each patch must contain both Krock's footprint, to account for the ground under the robot, and a certain amount of ground region in front of it. This corresponds to the maximum possible ground Krock can traverse in a selected time window.

To discover the correct value, we must compute the maximum advancement on flat ground for the Δt and use it to calculate the final size of the patch. We compute it by running some simulations of *Krock* on flat ground and averaging the advancement getting a value of 70cm in our $\Delta t = 2$s.

Each patch must include Krock's footprint and the maximum possible distance it can travel in from the current pose $X(t)$ to the future position $X(t + \Delta t)$. Since Krock's pose was stored from the IMU located in the juncture between the head and the legs, we have to crop from behind its length, 85cm minus the offset between the IMU and the head, 14cm. Then, we have to take 70cm, the maximum advancement with a $\Delta t = 2$s plus the removed offset. The final patch size is 156×156cm that is encoded as a 78×78 px 2D gray image with a resolution of $2cm/px$. Figure **??** visualizes the patch extraction process. Lastly, we create a final dataframe containing the map coordinates, the advancement, and the patches paths per simulation and store them to disk as *.csv* files.

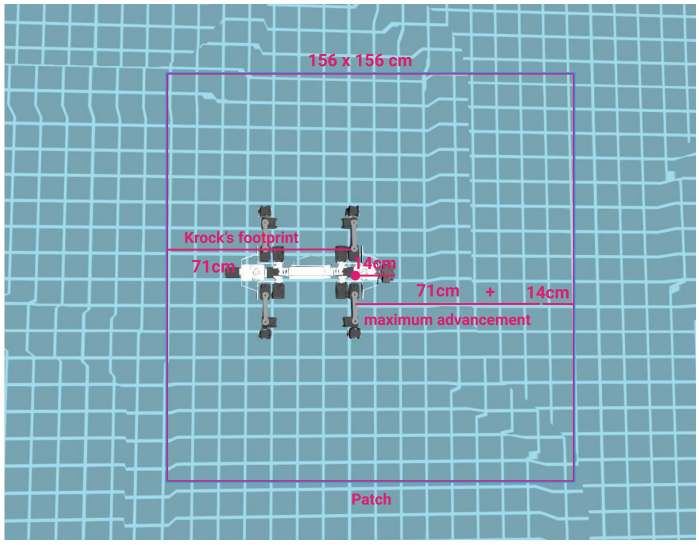
4.4.4 Final dataset

The final dataset is composed of ≈ 470K and ≈ 37K patches for the train and test set respectively. A small part of the train set, 10% is used as validation. Train and validation set's trajectories do not overlap, each set has entirely independent simulation's runs

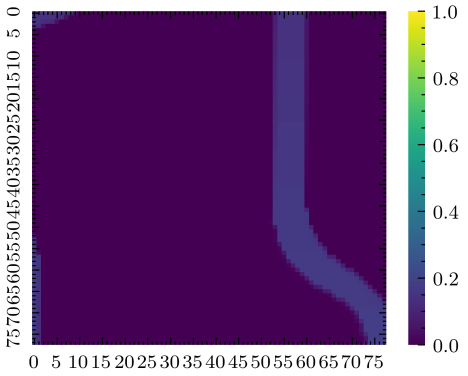
We also generate an other dataset, ≈ 38K patches, using the *arc rocks* map , to further evaluate the model. The whole pipeline took less than one hour to run the first time with 16 threads, and, once it is cached, less than fifteen minutes to extract all the patches. For completeness, we plotted in figure 4.15 the mean advancement over each map in the training set. As sanity check, we visualized some of the patches from the train set ordered by advancement, to conclude the data generation process was correct. Figure 4.17 shows a sample of fourty patches.

4.4.5 Normalization

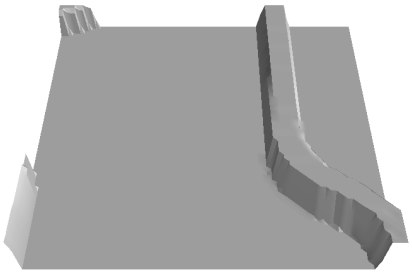
Before feeding the data to the models, at runtime, we needed to make the patches height invariant to correctly normalize different patches taken from different maps with different height scaling factor. To do so, we subtract the height of the map corresponding to Krock's position from the patch to correctly center it. Figure **??** shows the normalization process on a dummy patch.



(a) Robot in the simulator.



(b) Cropped patch in 2d.



(c) Cropped patch in 3d.

Figure 4.14. Patch extraction process for $\Delta t = 2$s. The final patch covers a region of 156×156cm and is encoded as a 78×78px 2D gray image with a resolution of $2cm/px$.

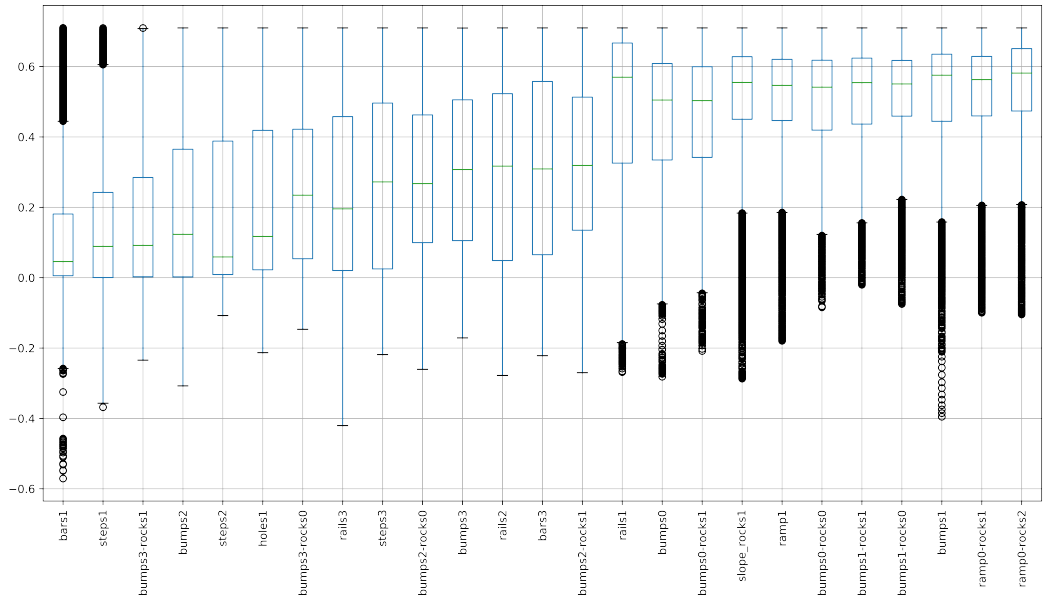
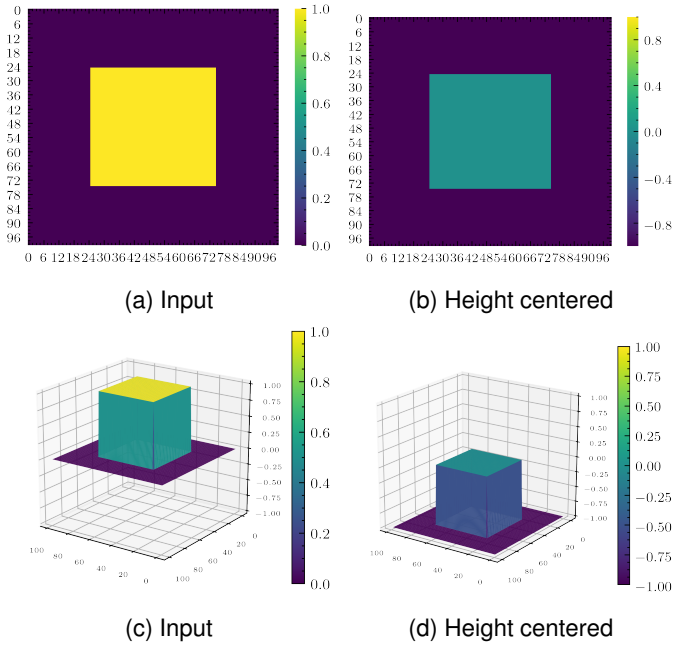


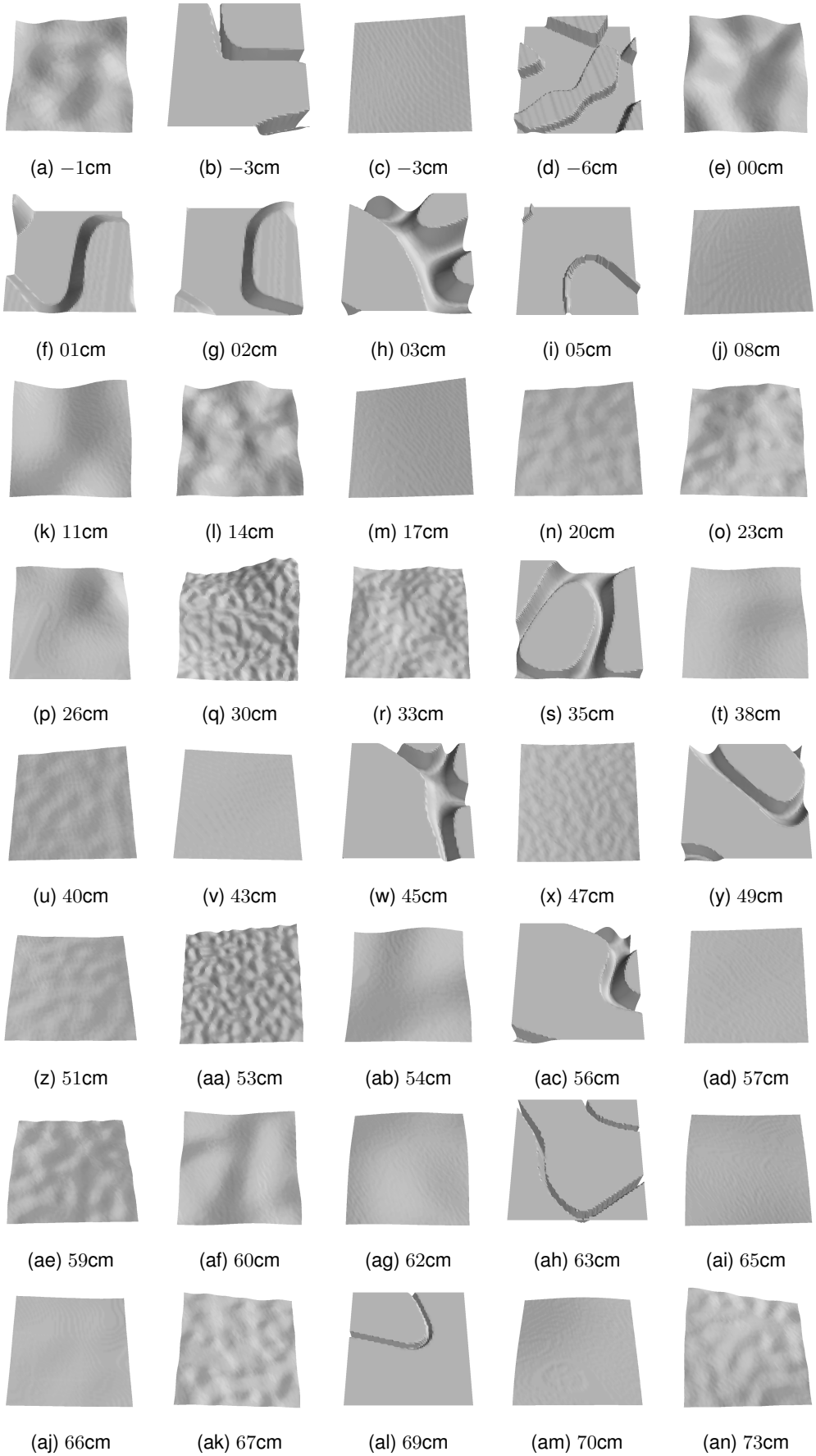
Figure 4.15. Advancement on each map with a $\Delta t = 2$s in ascendent order.



(a) Input (b) Height centered

(c) Input (d) Height centered

Figure 4.16. Normalization process. Each patch is normalized by subtracting the height value corresponding to the robot position to center its height.



(a) -1cm (b) -3cm (c) -3cm (d) -6cm (e) 00cm

(f) 01cm (g) 02cm (h) 03cm (i) 05cm (j) 08cm

(k) 11cm (l) 14cm (m) 17cm (n) 20cm (o) 23cm

(p) 26cm (q) 30cm (r) 33cm (s) 35cm (t) 38cm

(u) 40cm (v) 43cm (w) 45cm (x) 47cm (y) 49cm

(z) 51cm (aa) 53cm (ab) 54cm (ac) 56cm (ad) 57cm

(ae) 59cm (af) 60cm (ag) 62cm (ah) 63cm (ai) 65cm

(aj) 66cm (ak) 67cm (al) 69cm (am) 70cm (an) 73cm

Figure 4.17. Sample of extracted patches from the train set ordered by advancement. The first patches have mostly obstacle close to the robot's head, while the latter have smooter surfaces.

4.5 Label patches

To classify the patches we first need to label them. We need to select a threshold to consider a patch traversable if the advancement is greater or not traversable if it is lower. Ideally, the threshold should small enough to include as less as possible false positive and big enough to cover all the cases where Krock gets stuck. We empirically compute the threshold's value by spawning Krock in front of bumps and a ramp and let it walk.

Bumps We place the robot on the *bumps3* map close to the end and let it go for 20 seconds. Figure 4.18 shows Krock in the simulated environment.

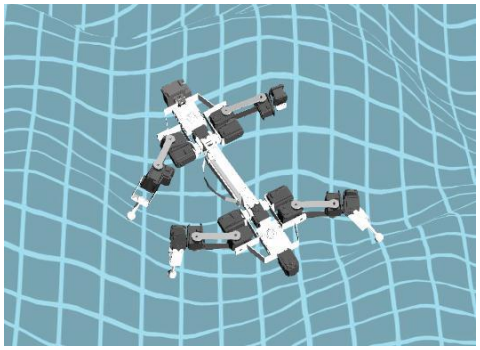


Figure 4.18. Krock tries to overcome an obstacle in the *bumps3* map.

Due to its characteristic locomotion, Krock tries to overcome the obstacle using its legs to move itself to the top but it fell backs producing a spicky advancement where first it is positive and then negative. Figure 4.19 shows the advancemnet on this map with $\Delta t = 2$, we can notice the noisy output.

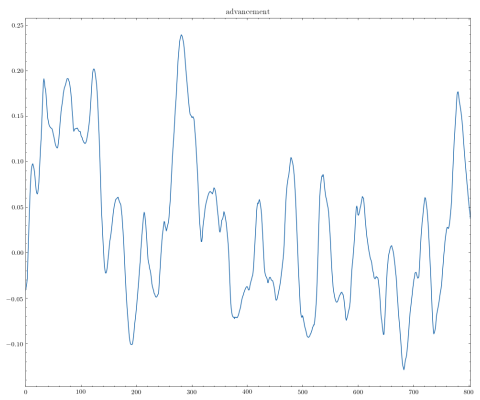


Figure 4.19. Advancement over time with $\Delta t = 2$ on *bumps3*.

A wheeled robot, on the other hand, will not produce such a graph since it cannot free itself easily from an obstacle. Image a wheeled robot moving forward, once it hits an obstacle it stops moving. So, if we plot its advancement in this situation it will look more like a step than a series of spikes.

Ramps The threshold should be also small enough to not create any false positive, patches that are traversable but were classified as not. So, we let the robot walk uphill on the $slope_rocks1$ map with a height scaling factor set to 5. We knew from empirical experiments that the robot is able to climb this steep ramp, the advancement is showed in figure 4.20. The mean ≈ 0.4cm over a

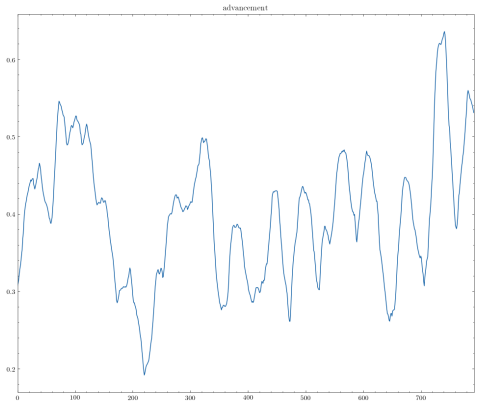


Figure 4.20. Advancement over time with $\Delta t = 2$s on *slope_rocks3*.

$\Delta t = 2$, an impressive value considering the steepest of the surface. We used this information combined with the previous experiment to choose a threshold that is beetween the upper bound of *bumps* and beetween the lower bound of *slope_rocks1*. This ensured to minimize the false positive. A good value is $tr_{\Delta t=2s} = 20cm$.

4.6 Estimator

In this section, we described the convolutional neural network utilized in the original paper. Then, we introduce residual networks and new techniques used to improve their performance. In the end, we proposed four residual network variants to be evaluated.

4.6.1 Original Model

The original model proposed by Chavez-Garcia et al. [5] is a CNN composed by a two 3×3 convolution layer with 5 filters; 2×2 Max-Pooling layer; 3×3 convolution layer with 5 filters; a fully connected layer with 128 outputs neurons and a fully connected layers with two neurons. We considered this the baseline, Figure 4.21 visualizes the architecture.

4.6.2 ResNet

We adopt a Residual Network, ResNet [8], variant. Residual networks are deep convolutional networks consisting of many stacked R̈esidual Units ¨. Intuitively, the residual unit allows the input of a layer to contribute to the next layer's input by being added to the current layer's output. Due to possible different features dimension, the input must go through and identify the map to make the addition possible. This allows a stronger gradient flows and mitigates the degradation problem. A R̈esidual Units ïs composed by a two $3x3$ *Convolution*, *Batchnorm* [11] and a *Relu* blocks. Formally

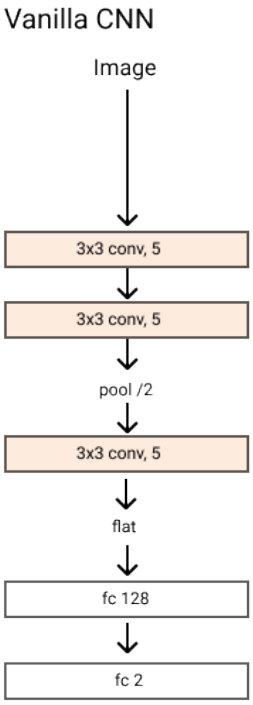


Figure 4.21. original model proposed by Chavez-Garcia et al. [5]. The rectangle represents the building block of each layer. Convolutional layers are described by the kernel size, the number of filters and the stride.

defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + h(\mathbf{x}) \tag{4.1}$$

Where, x and y are the input and output vector of the layers considered. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ is the residual mapping to be learn and h is the identity mapping. The next figure visualises the equation. When the input and output shapes mistmatch, the *identity map* is applyed to the input as

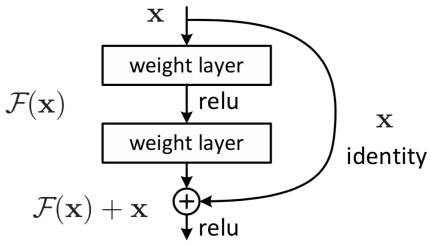


Figure 4.22. *Resnet* block [8]

a $3x3$ Convolution with a stride of 2 to mimic the polling operator. A single block is composed by a $3x3$ *Convolution*, *Batchnorm* and a *Relu* activation function.

4.6.3 Preactivation

Following the recent work of He et al. [9] we adopted *pre-activation* in each block.*Pre-activation* works by just reversing the order of the operations in a block.

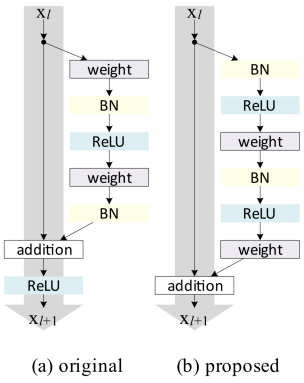


Figure 4.23. *Preactivation* [9]

4.6.4 Sqeeze and Excitation

Finally, we also used the *Squeeze and Excitation* (SE) module [10]. It is a form of attention that weights the channel of each convolutional operation by learnable scaling factors. Formally, for a given transformation, e.g. Convolution, defined as $\boldsymbol{F}_{tr} : \boldsymbol{X} \mapsto \boldsymbol{U}$, $\boldsymbol{X} \in mathbbR^{H' \times W' \times C'}$, $\boldsymbol{U} \in \mathbb{R}^{H \times W \times C}$, the SE module first squeeze the information by using average pooling, $\boldsymbol{F}_{sq}$, the it excitates them using learnable weights, $\boldsymbol{F}_{ex}$ and finally, adaptive recalibration is performed, $\boldsymbol{F}_{scale}$. Figure **??** visualizes the SE module.

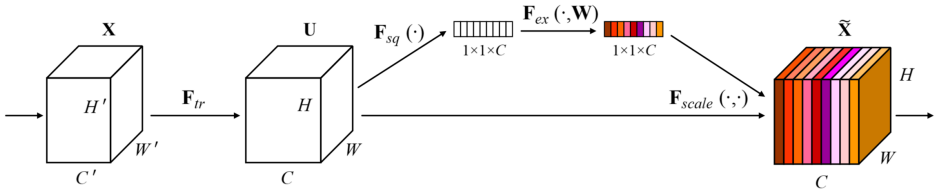


Figure 4.24. *Squeeze and Excitation*. Figure from the original paper. [10]

4.6.5 MicroResNet

The proposed network's architecture is composed by n ResNet blocks and channel incrementing factor of 2. ResNet is used to classify RGB images with 224×224 pixels so it used an aggressive convolution and max-polling operation in the first layer to reduce the input size. We decide to adopt a less aggressive convolution motivated by the smaller size of our inputs. We test two kernel sized: 7×7 and 3×3 with stride of 2 and 1 respectively. Lastly, we use LeakyReLU [4] with a negative slope of 0.1 instead of ReLU to allow a better gradient flow during backpropagation. LeakyRelu is defined in equation 4.2 .

$$LeakyRelu(x) = \begin{cases} x & \text{if } x > 0 \\ 0.1x & \text{otherwise} \end{cases} \tag{4.2}$$

We call this architecture *MicroResNet*. We propose four networks layouts: with $n = 1$, two first layers setups, 3×3 stride $= 1$ and 7×7, with and without the Squeeze and Excitation module. These architectures can be used both for regression and classification by only changing the output size of the fully connected later, 2 for classification and 1 for regression. All other layers are equals. Figure 4.25 shows the models architecture while table 4.2 adds more information. Our models have approximately 35 times less parameters than the smalles ResNet model, ResNet18 [8], that has 11M parameters. To simplicity we used the following notation to describe each architecture variant: `MicroResNet-3x3/7x7-/SE`.

4.7 Data augmentation

Data augmentation is used to change the input of a model in order to produce more training examples. Since our inputs are heightmaps we cannot utilize the classic image manipulations such as shifts, flips, and zooms. Imagine that we have a patch with a wall in front of it if we random rotate the image the wall may go in a position where the wall is not facing the robot anymore, making the image now traversable with a wrong target. We decided to apply dropout, coarse dropout, and random simplex noise in the correct degree to make them traversability invariant. To illustrate the techniques we used a dummy patch with a cube in the middle, figure 4.16.

4.7.1 Dropout

Dropout is a technique to randomly set some pixels to zero, in our case we flat some random pixel in the patch.

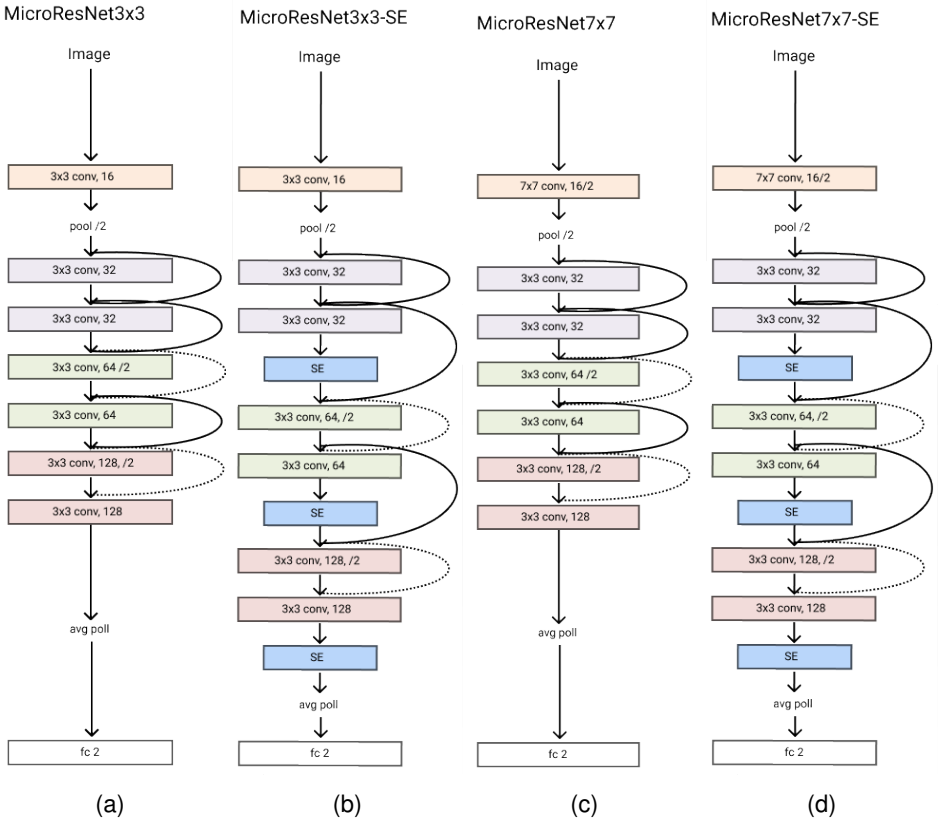


Figure 4.25. MicroResNet architectures for classification. All models have an initial features size of 16 and $n = 1$ with and without the Squeeze and Excitation module. The rectangle represent the building block of each layer. Convolutiol layers are described by the kernel size, the number of filters and the stride. Lines and dashed lines between layers represent the residual operations and the shortcut respectively.

Input	$(1, 78, 78)$			
Layers	3×3, 16 stride 1		7×7 16 stride 2	
	$2 \times$ 2 max-pool			
	$\begin{bmatrix} 3 \times 3, & 16 \\ 3 \times 3, & 32 \end{bmatrix}$ x 1			
	SE	-	SE	-
	$\begin{bmatrix} 3 \times 3, & 32 \\ 3 \times 3, & 64 \end{bmatrix}$ x 1			
	SE	-	SE	-
	$\begin{bmatrix} 3 \times 3, & 64 \\ 3 \times 3, & 128 \end{bmatrix}$ x 1			
	SE	-	SE	-
	average pool, 1-d fc, softmax			
Network Size (params)	313,642	302,610	314,28	303,250
Size (MB)	5.93	5.71	2.41	2.32

Table 4.2. MicroResNet architecture. First layer is on the top. Some architecture's blocks are equal across models, this is shows by sharing columns in the table.

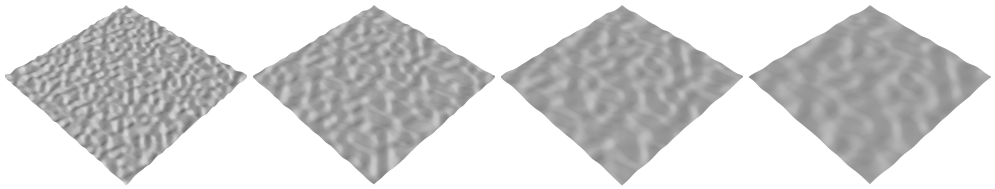
4.7.2 Coarse Dropout

Similar to dropout, it sets to zero random regions of pixels with defined boundaries. To ensure we do create untraversable patches from traversable ones we limited the coarse dropout region to only a few centimeters..

4.7.3 Simplex Noise

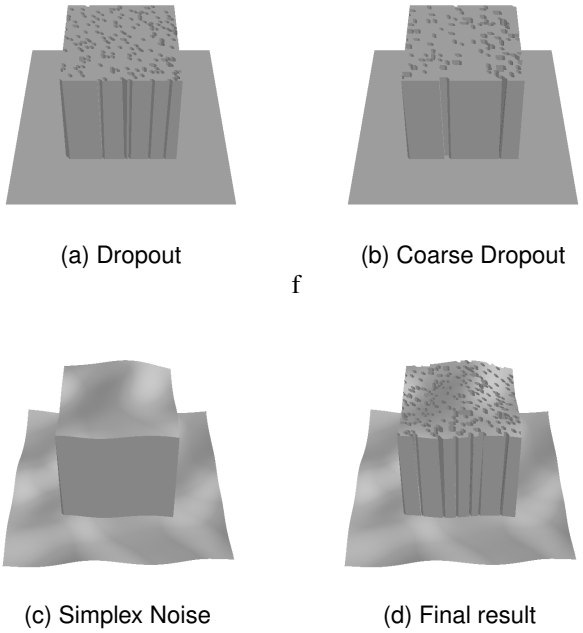
We added some noise the inputs to slightly changed the ground in order to help the network generalize better. Since it is computationally expensive, we first randomly applied the noise to five hundred images with only zeros, flat grounds. We cached them and then we randomly scaled them and added to the input image. To ensure traversability invariant, we applied simplex noise to the inputs based on their classes. For traversability grounds, we used a noise feature size $15 - 25$ while, for not traversable samples a noise size between $5 - 15$. Figure 4.26 clearly shows the grounds generated by different sizes.

Figure **??** shows the tree data augmentation techniques applied the the dummy patch. We augmented 80% of the input images with dropout and coarse dropout. Dropout has a probability between 0.05 and 0.1 and coarse dropout has a probability of 0.02 and 0.1 with a size of the lower resolution image from which to sample the dropout between 0.6 and 0.8. Simplex noise is applied to the 70% of the training data samples.



(a) Noise size = 10 (b) Noise size = 20 (c) Noise size = 30 (d) Noise size = 40

Figure 4.26. Simplex noise on flat ground. The feature size corespond to the amount of noise we introduce in the surface. A lower value produces noisy grounds, while a bigger one smoother terrains.



(a) Dropout (b) Coarse Dropout

f

(c) Simplex Noise (d) Final result

Figure 4.27. Data augmentation applied on a patch. First, we randomly set to flat some small regions of the ground. Then, we slightly mutate the surface using simplex noise.

4.8 Experiments

4.8.1 Metrics

Classification: To evaluate the model's classification performance we used two metrics: *accuracy* and *AUC-ROC Curve*. Accuracy scores the number of correct predictions made by the network while the AUC-ROC Curve represents degree or measure of separability, informally it tells how much model is capable of distinguishing between classes. For each experiment, we selected the model with the higher AUC-ROC Curve during training to be evaluated on the test set.

Regression: We used the Mean Square Error to evaluate the model's performance.

4.9 Training setup

Initially, to train the models we first use Standard Gradient Descent with momentum set to 0.95, weight decay to $1e-4$ and an initial learning rate of $1e-3$ as proposed in He et al. [8]. Then, we switch to Leslie Smith's 1 cycle policy [21] that allow us to train the network faster and with higher accuracy. The average training time is $\approx$ 10 minutes. We minimized the Binary Cross Entropy (BCE) for the classifier and the Mean Square Error (MSE) for the regressor.

4.10 Tools

We briefly list the most important tools and libraries adopted in this project. The framework was entirely developed on Ubuntu 18.10 with Python 3.6.

4.10.1 Software for simulation

ROS Melodic The Robot Operating System (ROS) [18] is a flexible framework for writing robot software. It is *de facto* the industry and research standard framework for robotics due to its simple yet effective interface that facilitates the task of creating a robust and complex robot behavior regardless of the platforms. ROS works by establishing a peer-to-peer connection where each *node* is to communicate between the others by exposing sockets endpoints, called *topics*, to stream data or send *messages*. Each *node* can subscribe to a *topic* to receive or publish new messages. In our case, *Krock* exposes different topics on which we can subscribe in order to get real-time information about the state of the robot. Unfortunately, ROS does not natively support Python3, so we had to compile it by hand. Because it was a difficult and time-consuming operation, we decided to share the ready-to-go binaries as a docker image.

4.10.2 Data processing

Numpy Numpy is one of the most popular packages for any scientific use. It allows expressing efficiently any matrix operation using its broadcasting functions. Numpy is used across the whole pipeline to manipulate matrices.

Pandas To process the data from the simulations we rely on Pandas, a Python library providing fast, flexible, and expressive data structures in a tabular form. Pandas is well suited for many different kinds of data such as handle tabular data with heterogeneously-typed columns, similar to the SQL table or Excel spreadsheet, time series and matrices. We take advantages of the relational data structure to perform custom manipulation on the rows by removing the outliers and computing the advancement.

Generally, pandas does not scale well and it is mostly used to handle small dataset while relegating b̈ig data ̈o other frameworks such as Spark or Hadoop. We used Pandas to store the results from the simulator and inside a Thread Queue to parse each *.csv* file efficiently.

OpenCV Open Source Computer Vision Library, OpenCV, is an open source computer vision library with a rich collection of highly optimized algorithms. It includes classic and state-of-the-art computer vision and machine learning methods applied in a wide array of tasks, such as object detection and face recognition. We adopt this library to handle image data, mostly to pre and post-process the heightmaps and the patches.

CNN training

Pytorch PyTorch is a Python open source deep learning framework. It allows Tensor computation (like NumPy) with strong GPU acceleration and Deep neural networks built on a tape-based auto grad system. Due to its Python-first philosophy, it is easy to use, expressive and predictable it is widely used among researches and enthusiast. Moreover, its main advantages over other mainstream frameworks such as TensorFlow [2] are a cleaner API structure, better debugging, code shareability and an enormous number of high-quality third-party packages.

FastAI FastAI is library based on PyTorch that simplifies fast and accurate neural nets training using modern best practices. It provides a high-level API to train, evaluate and test deep learning models on any type of dataset. We used it to train, test, and evaluate our models.

imgaug Image augmentation (imgaug) is a python library to perform image augmenting operations on images. It provides a variety of methodologies, such as affine transformations, perspective transformations, contrast changes, and Gaussian noise, to build sophisticated pipelines. It supports images, heatmaps, segmentation maps, masks, key points/landmarks, bounding boxes, polygons, and line strings. We used it to augment the heightmap, details are in section 4.7

4.10.3 Visualization

matplotlib Almost all the plots in this report were created by Matplotlib, is a Python 2D plotting library. It provides a similar functional interface to MATLAB and a deep ability to customize every region of the figure. All It is worth citing *seaborn* a data visualization library that we inglobate in our work-flow to create the heatmaps. It is based on Matplotlib and it provides a high-level interface.

mayavi Mayavi is a scientific data visualization and plotting in python library. We adopt it to render in 3D all the surfaces used in our framework.

Chapter 5

Results

In this section, we evaluate five convolutional neural networks: the model proposed by Chavez-Garcia et al. [5] and the four MicroResnet presented in the last chapter. We select the best performing one based on the classification score on the test set. We show the results of the same architecture trained on regression instead of classification and prove the latter yields better results. Finally, we qualitatively evaluate the best model by predicting traversability in real-world terrains.

5.1 Quantitative results

5.1.1 Model selection

We compare the four different MicroRenset architectures 4.6.5 and the Original CNN 4.6.1 propose in the original work [5] on classification. The dataset is labeled as described in 4.5. We run five experiments per architecture, and we select the best performing network using as a metric the ROC AUC. Table 5.1 shows the results for each architecture. The best performing model is MicroResNet7x7-SE with a ROC AUC of 0.896 on the test set. We suggest that the big kernel size allows the network to extrapolate more gound features in the first layer. Also, the results demonstrate the performance boots given by the Squeeze and Excitation. MicroResNet7x7-SE gains a huge boost of 0.021 compared to MicroResNet7x7.

5.2 Classification results

Table 5.2 table shows in deep the MicroResNet7x7-SE's performance on different datasets. We obstain an 95% accuracy and 0.96 AUC on the validation set and 88% accuracy and 0.89 AUC on the test set.

5.3 Regression Results

Using MicroResNet7x7-SE as regressor also yield decent results. The model scores a MSE of 0.006 on the validation and 0.020 on the test set. Table 5.3 shows in detail the score for each dataset. Figure 5.1 plots the real advancement against the regressor output for the test set. In most cases the model did not correctly predict the advancement. The regressor has several advantages

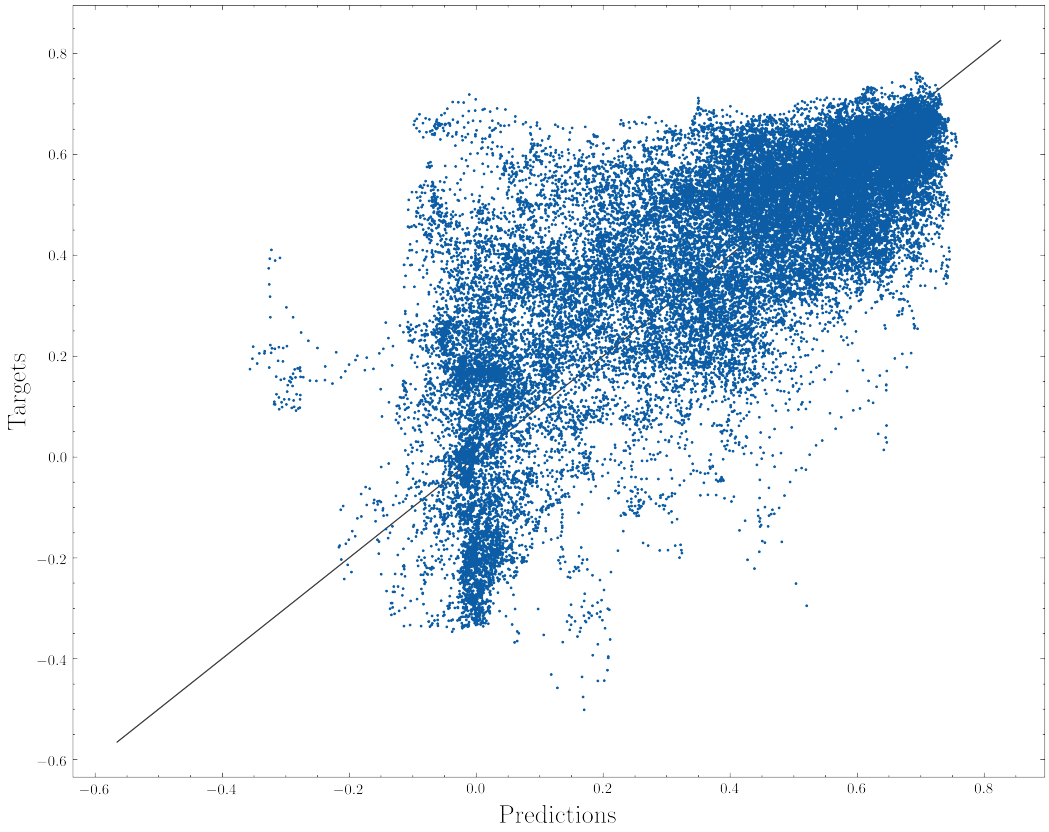


Figure 5.1. l advancement against the predicted one by the regressor for the test set. Most of the points are far away to the diagonal line showing that the regressor predicts the wrong advancement in many cases.

Model	ROC AUC		Params
	Mean	Top	
Original (baseline)	0.892	0.890	974,351
MicroResnet3x3	0.881	0.883	302,610
MicroResnet3x3-SE	0.883	0.888	313,64
MicroResnet7x7	0.867	0.875	303,250
MicroResnet7x7-SE	0.888	**0.896**	314,28

Table 5.1. Comparison of the four MicroResnet architectures 4.25 and the original convolutional neural network (Original CNN) proposed in the original work [5] on the test set. We run five experiments per architecture, we show the average and the top score for each metric. The best model is MicroResNet7x7-SE.

Dataset			MicroResNet7x7-SE		Size(m)	Resolution(cm/px)
Type	Name	Samples	ACC	AUC		
Synthetic	Training	429312	-	-	10×10	2
	Validation	44032	95.2 %	0.961	10×10	2
	Arc Rocks	37273	85.5 %	0.888	10×10	2
Real evaluation	Quarry	36224	88.2 %	0.896	32×32	2

Table 5.2. Classification results of MicroResNet7x7-SE on different datasets.

compared to the classifier, First, since we can use the same model to classify patches with a different threshold. This is done by seeing if the predicted advancement is greater or lower than a threshold. Regression is a usually feasible solution. For example, if we need to consider multiple thresholds. Classification requires the training of one classifier per threshold while regression can evaluate always the same model.

However, if we fixed a threshold, the same exactly architecture trained directly to predict traversability outperforms the regressor. Table 5.4 shows the accuracy of two MicroResNet7x7-SE on classification and regression after a threshold of twenty centimeters is selected. The classifier outperformed the regressor. For this reason, we decided to evaluate the MicroResNet7x7-SE classifier.

5.4 Qualitative results

We evaluate the classifier predictions by plotting the traversability probability on different maps in 3D. We used a sliding window to extract the patches to generate the patches. Then, we created a texture based on the traversable probability. For each map, we evaluated four headings.

Dataset			MicroResNet7x7-SE	Size	Resolution(cm/px)
Type	Name	Samples	MSE		
Synthetic	Training	-	-	10×10	2
	Validation	44032	0.006	10×10	2
	Arc Rocks	37273	0.020	10×10	2
Real evaluation	Quarry	36224	0.022	32×32	2

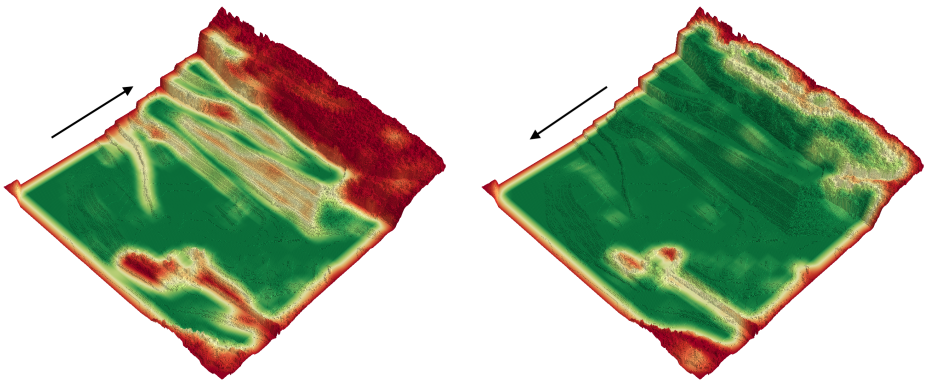
Table 5.3. Regression results of MicroResNet7x7-SE on different datasets.

	Regression	Classification
	ACC	
Top	72.8%	**88.2%**
Mean	73.6%	**87.8%**

Table 5.4. Regression and classification accuracy for the same model, MicroResNet7x7-SE, on a threshold of 20cm. The regression accuracy is computed using its output labeled with the selected threshold and the binary targets from the classification dataset. This mimic the situaion when the fixed a miniuim advancement for the robot.In this case, the classifier outperforms the regressor.

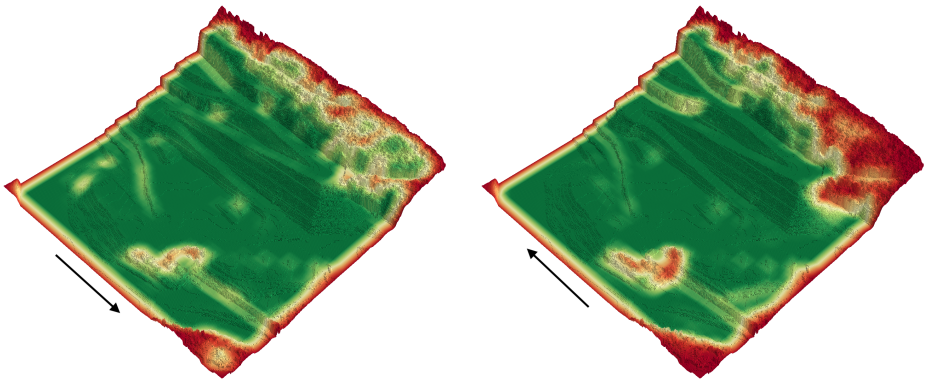
5.4.1 Quarry

We first evaluate the test set composed by the quarry map 32×32m and with a maximum height of 10m. We expect the trail on the slopes to be traversable at almost any headings, especially when Krock move from left to right and vice-versa. The top part should be hard to traverse in almost any case. Figure 5.2 shows the traversability probability directly on the map. Correctly, the lower part



(a) Robot moving from bottom to top

(b) Robot moving from top to bottom



(c) Robot moving from left to right

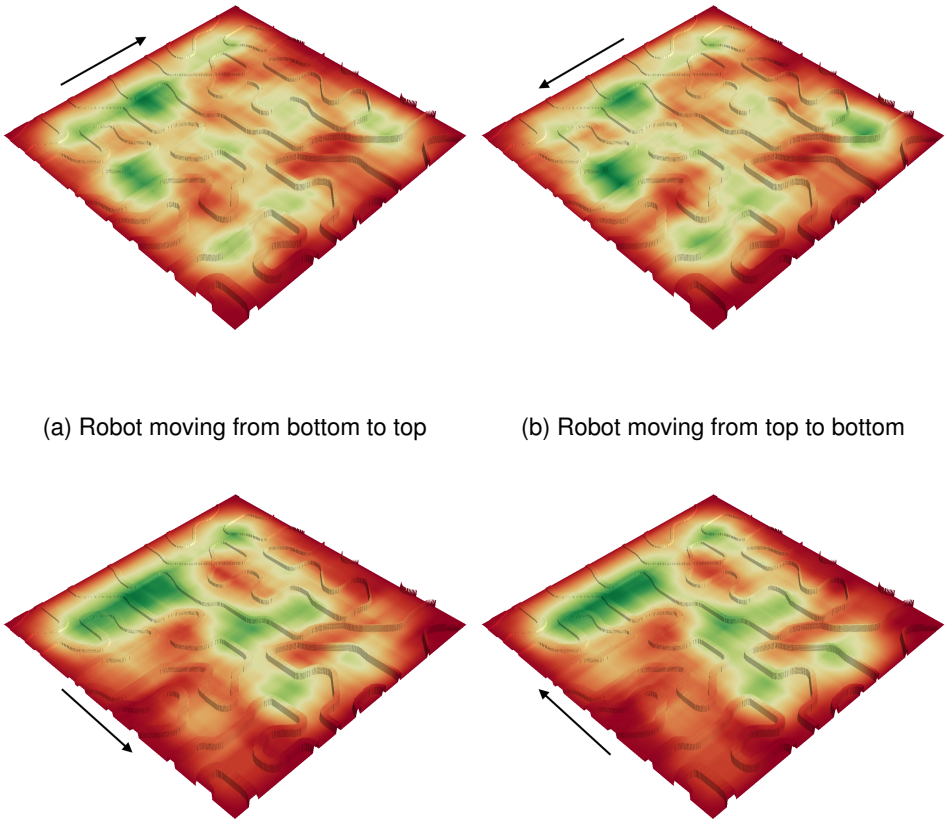
(d) Robot moving from right to left

Figure 5.2. Traversability probability on the test set, a quarry 32×32m and a maximum height of 10m, for different Krock's orientations. Red equal not traversable, green equal totally traversable.

of the map, composed by flat regions, is labeled with high confidence as traversable in all rotations. On the other hand, the traversability of the slopes and the bumps on the top region depends on the robot orientation.

5.4.2 Bars

Bars is a map composed of walls with different heights, thus we expected to be mostly not traversable at different headings. Figure 5.3 shows the traversability probabilities on the terrain.



(a) Robot moving from bottom to top

(b) Robot moving from top to bottom

(c) Robot moving from left to right

(d) Robot moving from right to left

Figure 5.3. Traversability probability on the bars map, a 10×10m , for different Krock's orientations.

Due to the high number of not traversable walls, this is a hard map to traverse the robot . Interesting, we can identify a corridor near the bottom center of the maps. This region shows how the model correctly label these patches depending on the orientation. Figure 5.4 highlights this detail.

5.4.3 Small village

We apply the same procedure to evaluate the network on a 10×10m small village map. Figure 5.5 describes its traversability for four different rotations.

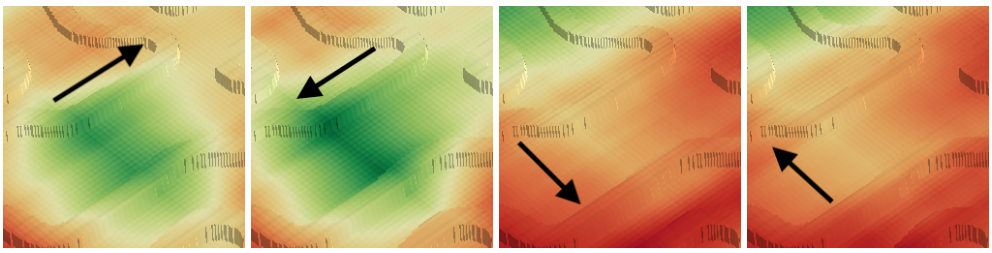
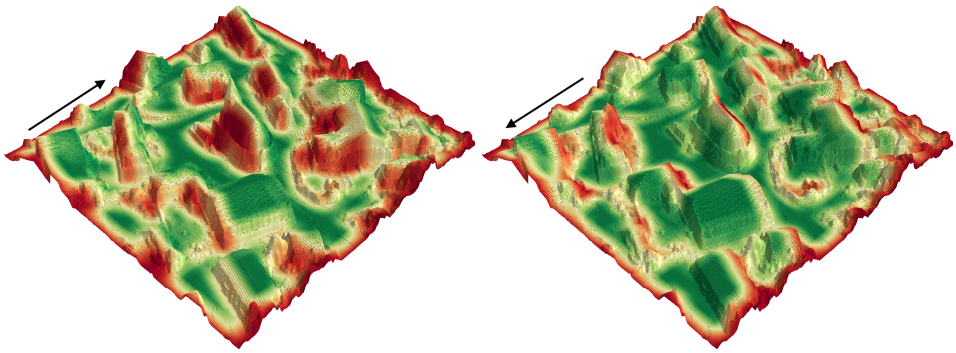
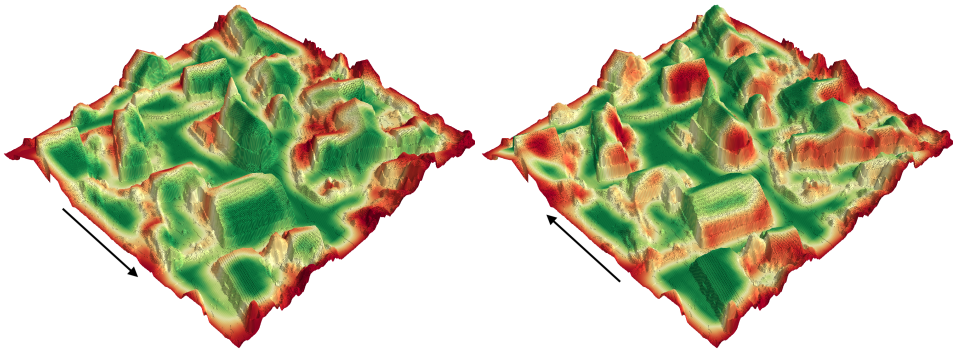


Figure 5.4. Detail of a region in the bars map where there are two walls forming a corridor. When the robot is ollowing the trail the region is label as traversable. Red equal not traversable, green equal totally traversable.



(a) Robot moving from bottom to top

(b) Robot moving from top to bottom

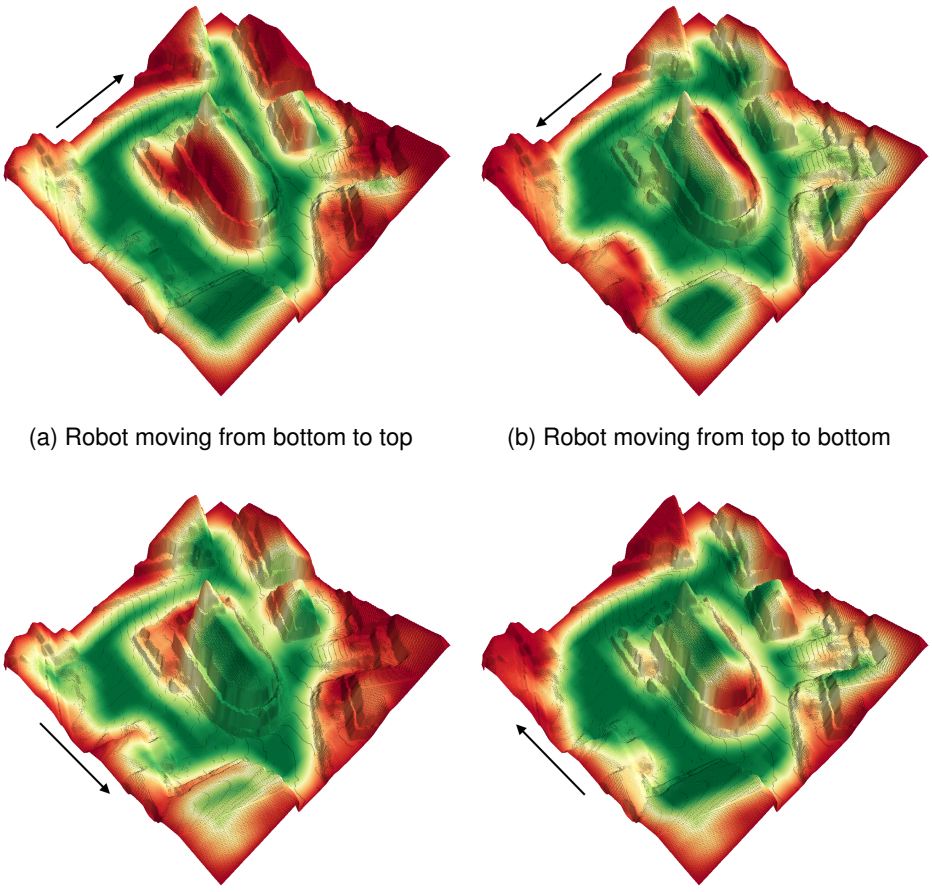


(c) Robot moving from left to right

(d) Robot moving from right to left

Figure 5.5. Traversability probability on the map of a small village for different Krock's rotation. The surface covers 30×30m and has a maximum height of 10m. Red equal not traversable, green equal totally traversable.

All the streets are labeled as traversable, while some buildings' roofs (e.g. the church's one) are traversable depending on the Krock orinetation. For example, most steep roofs are not traversable when walking uphill. On the other hand, if krock walks side by side they can be traversed. Figure 5.6 shows this behavior on the church's roof.



(a) Robot moving from bottom to top

(b) Robot moving from top to bottom

(c) Robot moving from left to right

(d) Robot moving from right to left

Figure 5.6. Detail of the church in the small village map for different Krock's rotation. All the fours images are correctly labeled accordingly to the robot orientation. In the first two images only downhill part of the roof is traversable. While in the last two the robot is able to travel till the end. Red equal not traversable, green equal totally traversable.

Chapter 6

Interpretability

In this section, we interpret MicroResNet7x7-SE's predictions using different techniquesin order to understand its ability to properly estimate traversability by classifying ground patches. We highlight its strength, robustness to understand its limitations. We evaluate the quality of our traversability estimator with different methodology. First, we showed that the model has correctly learn grounds' features and is able to separate terrains based on them. Second, we utilize the test dataset to visualize the most traversable, the least traversable and the misclassified patches. Utilizing a special method, we determine that the model always looke at the correct features in the ground even if when it fails. Finally, we craft several patches with different unique features, such as walls, bumps, etc, in order to verify the robustness of the model by comparing its predictions to the real data gathered from the simulator.

6.1 Features separability

Convolutional neural networks learn to encode images by applying filters of increasing size at each layer. The first layers learn basic features, such as edges, while the final one encodes complex shapes. The outputs in the final convolution layer are usually referred to as *features space*, consequently, a feature vector is just the output of the last layer for a given image. These last features are combined and mapped to the correct classes by one or more fully connected layers. Lee et al. [14] have visualized the features learned by the first and last layers, this is illustrated in figure 6.1. So, a

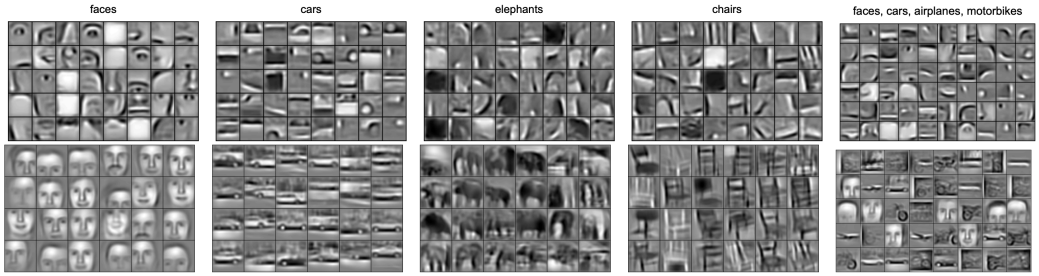


Figure 6.1. Figure from Lee et al. [14] paper where they show for fours different classes the low-level features (up) and the high-level features (down) learned by a convolution neural network.

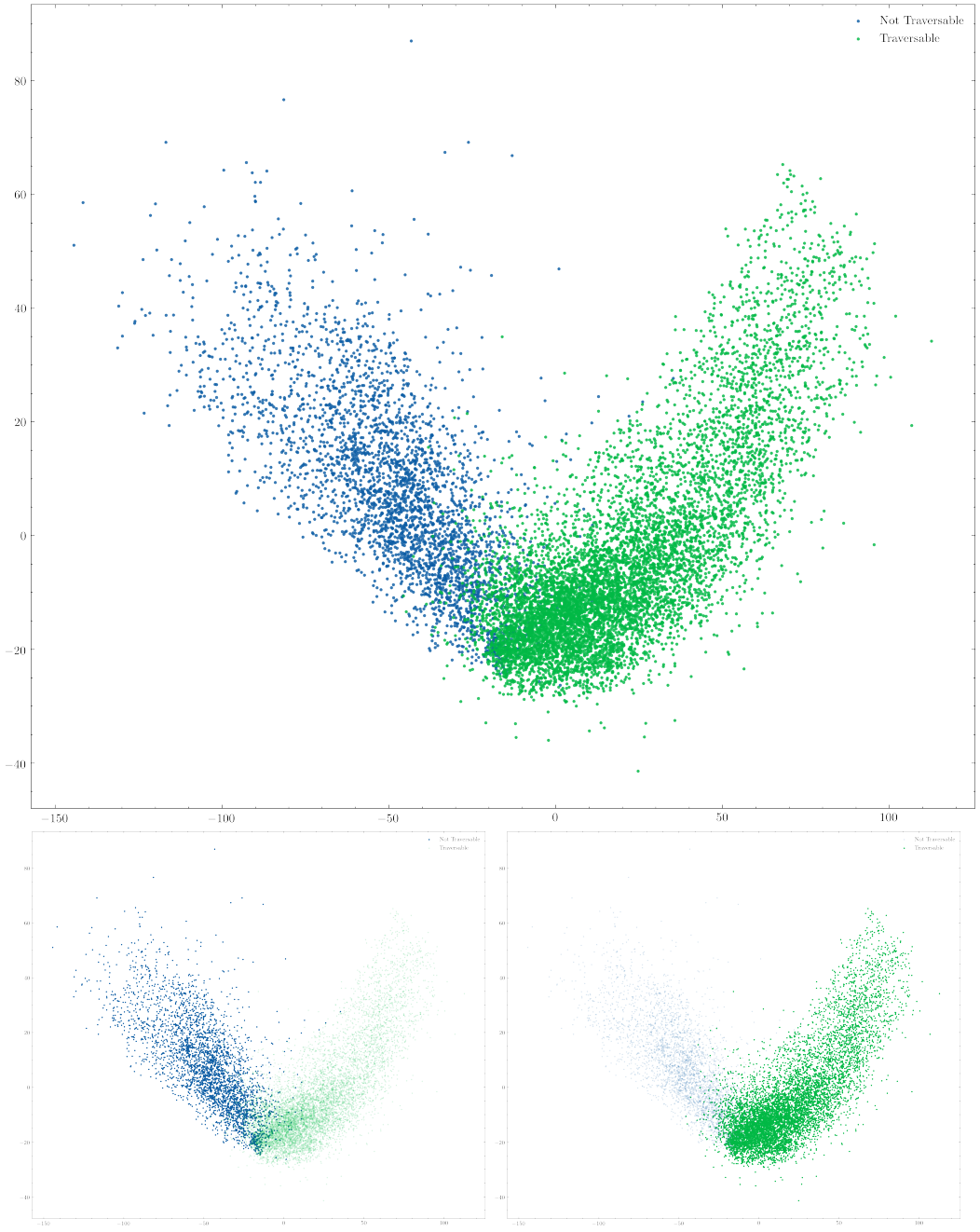
correctly trained network should be able to separate the features vectors in the features space based on their classes. Intuitively, given two classes $\mathcal{A}$ and $\mathcal{B}$, for example, *chairs* and *cars*, the high-level features for each class should not be the same. For instance, if the network believes that big wheels are features of both chairs and cars then chairs may be wrongly classified as cars. Similarly, two patches have a small and big wall in front of the robot should not be mapped in the same position in the features. Because, from a traversability point of view, have different characteristic, one has a traversable wall, the other not. However, these patches are close to each other in the features space, the model could foolishly believe they are similar and belong to the same class. One technique to discover the degree of separability is to directly visualize the features vectors for each class. In our case, MicroResNet7x7-SE, maps the inputs to a 128 dimensional feature space. Since, we cannot directly visualize such highg dimension space, we reduced the feature vectors to a two-dimension space by applying Principle Analysis Component (PCA) [20]. We investigat the features space of the model both in the train and test set.

6.1.1 Features space of the train set

Figure **??** shows the features space for 11K images sampled from the train set labeled with their classes, *traversable* and *not traversable*. We can clearly recognize two main clusters based on the labels' color, one on the left and one of the right. These points are easily separable, even by human eyes, meaning that the model was able to learn meaning features from the dataset and use to make accurate predictions. To be totally sure the center of each class' point cloud is not overlapping we plotted the density of each cluster. Clearly, there is some distance between the centers. Furthermore, we can directly plot the patch corresponding to each feature vector to identify clusters of inputs based on their similarities. Intuitively, if similar inputs are close to each other in the features space then the model also learned to effectively encode terrains features. We decid to not show all images on the same plot to avoid overcrowding the image. Instead, we cluster the points using K-Means with $k = 50$ clusters and then we took the patch that corresponded to the center point in each cluster. In this way, even by showing only a few inputs, we include all the meaningful ground types. Figures 6.4 visualizes the results. Definitely, patches with similar features are close to each other yielding a quality features encoding. On the left-top side, we can distinguish highly untraversable patches with walls/bumps in front of the robot. Going down, we encounter patches with smaller obstacles. On the plateau, there are traversable patches with small obstacles such as light bumps. Importantly, these patches are the closest ones to the not traversable ones, so they were the hardest to separate, thus, to classify. Going up on the right side, we see some grounds with small steps. Finally, on the top, we find all the downhill patches, the simplest ones to traverse.

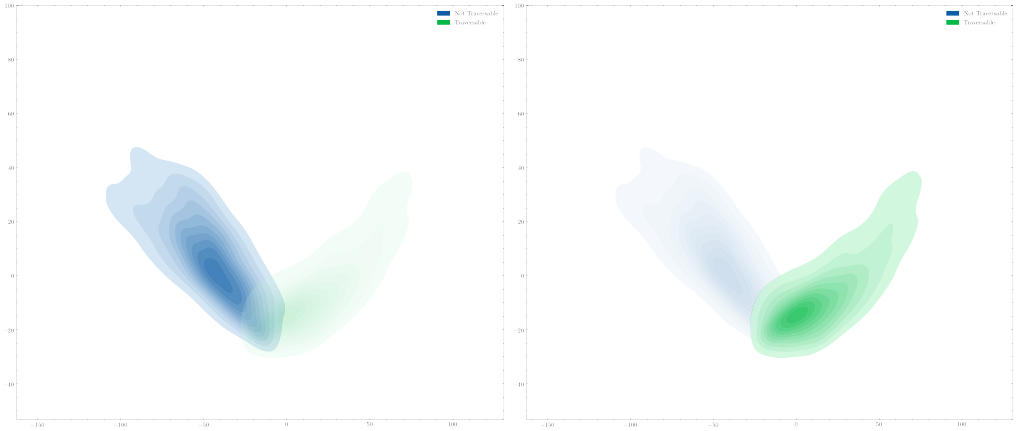
6.1.2 Features space of the test set

We can apply the same procedure on the test set. Since it is a real world quarry, this dataset is harder than the train set and present challenging situations for the robot. Figure 6.5 displays the features space after reducing its dimension to two using PCA. Interesting, the traversable patches in figure **??** are very near to each other, while the others span a very big surface. This suggests that there are many not traversable terrains with different features. The traversable points are clustered near the center, this implies that most of them share similar features. We plotted the density for each class to better understand where the most points are mapped. The two centers are really close to each other, making these samples harder to separate and some not traversable points are mixed up with the traversable ones. This explains the elevated number of false negative that lower down the AUC



(a) Not Traversable (b) Traversable

Figure 6.2. Principal Component Analysis on the features space computed using the outputs from the last convolutional layers on the train dataset. The two point clouds are perfectly separable.



(a) Not Traversable (b) Traversable

Figure 6.3. Density plot for train set features space. The more opaque the color the close to the cluster center. The centers of the cluster are not overlapping yielding a good separability and correct learning.

score on this dataset. As we did before, we can also visualize the patches by plotting them using their features coordinates. Figure **??** shows the patches directly into the features space. On the top left, from the not traversable cloud, we can see patches with a high level of bumps. Going down we find surfaces with huge walls in front of the robot while going close to the center we start to see all the traversable patches. These samples have not too steep slopes. If we move to the density center, green double shown in figure 6.6b, we encounter lots of flat patches with little obstacles. Going up on the right branch we find downhill and on the top there are falls.

To summarize, we showed how the model correctly learned to separate the inputs based on their traversability, to encode meaningful grounds' information and to map close to each other patches with similar characteristic in the features spaces. In the following section, we will take a deep look at the test set to find which patches confuse the most the model. Probably, the samples will be located between the two clusters' center where the difference between classes' features is minimum.

6.2 Test dataset exploration

After showing the model's capability of correctly separate classes' features we utilized Grad-CAM to visualize some patches inside the quarry dataset. The aim of this section is to show how the model looks at meaningful features in each input to make the prediction even when it fails. For instance, imagine we feed to the model a not traversable patch with an obstacle and the network label is as traversable. Clearly, the output is wrong but the model's error may be caused by two different situations. The first one, the model could just have ignored the obstacle and looked away, meaning that it was not even able to understand there was an obstacle in the first place. Second, the network could have correctly look at the obstacle but thought that obstacle is traversable, showing the correct ability to find and use important features in the map. In the following sections, we showed that, even when the predictions are wrong, our model always look at the most important features of each input to determine its traversability.

We divided the dataset patches into four classes based on the model's performance: worst, best,

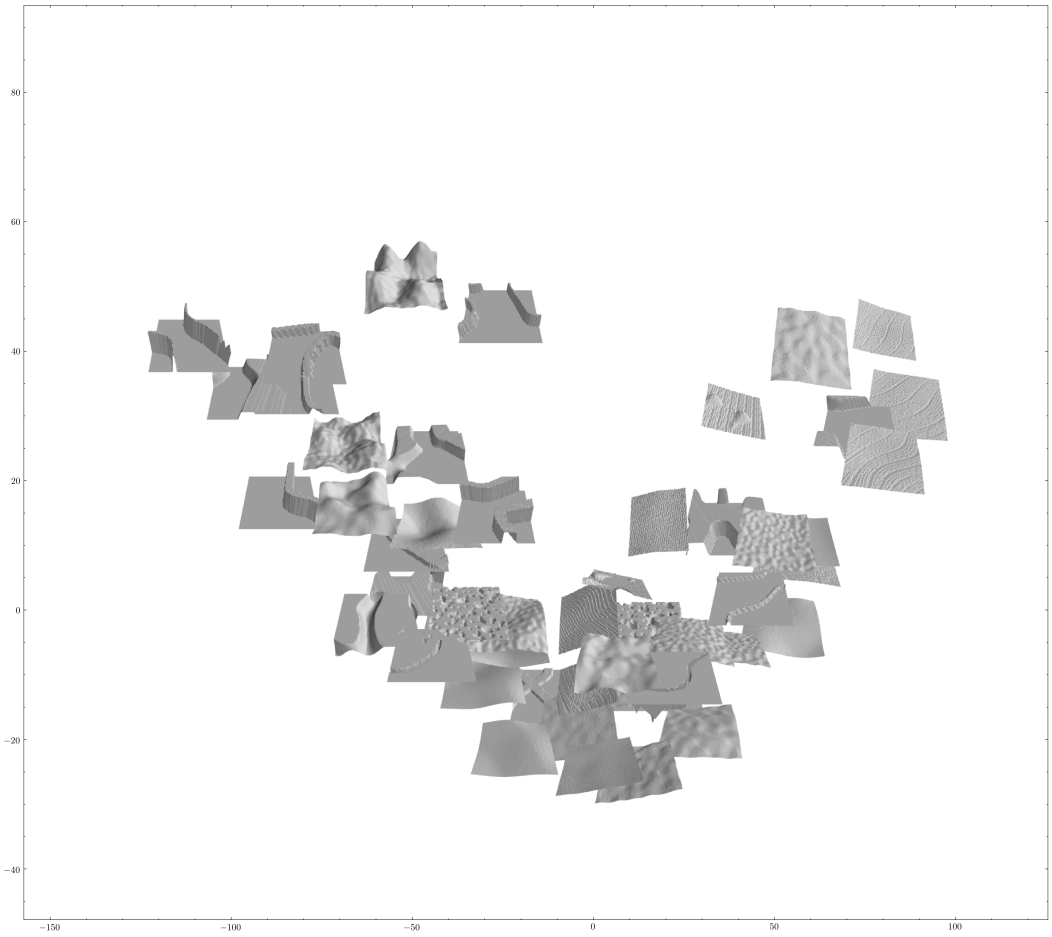


Figure 6.4. Patches plotted using the coordinate of the features vector obtained from the last convolutional layer's output and then reduced using PCA to a two-dimensional vector. Similar grounds are close to each other. From the top left in counterclockwise order, there are not traversable patches with walls, steps, and big bumps. On the plateau, we can found traversable patches with light bumps. Going up we encounter the downhills.

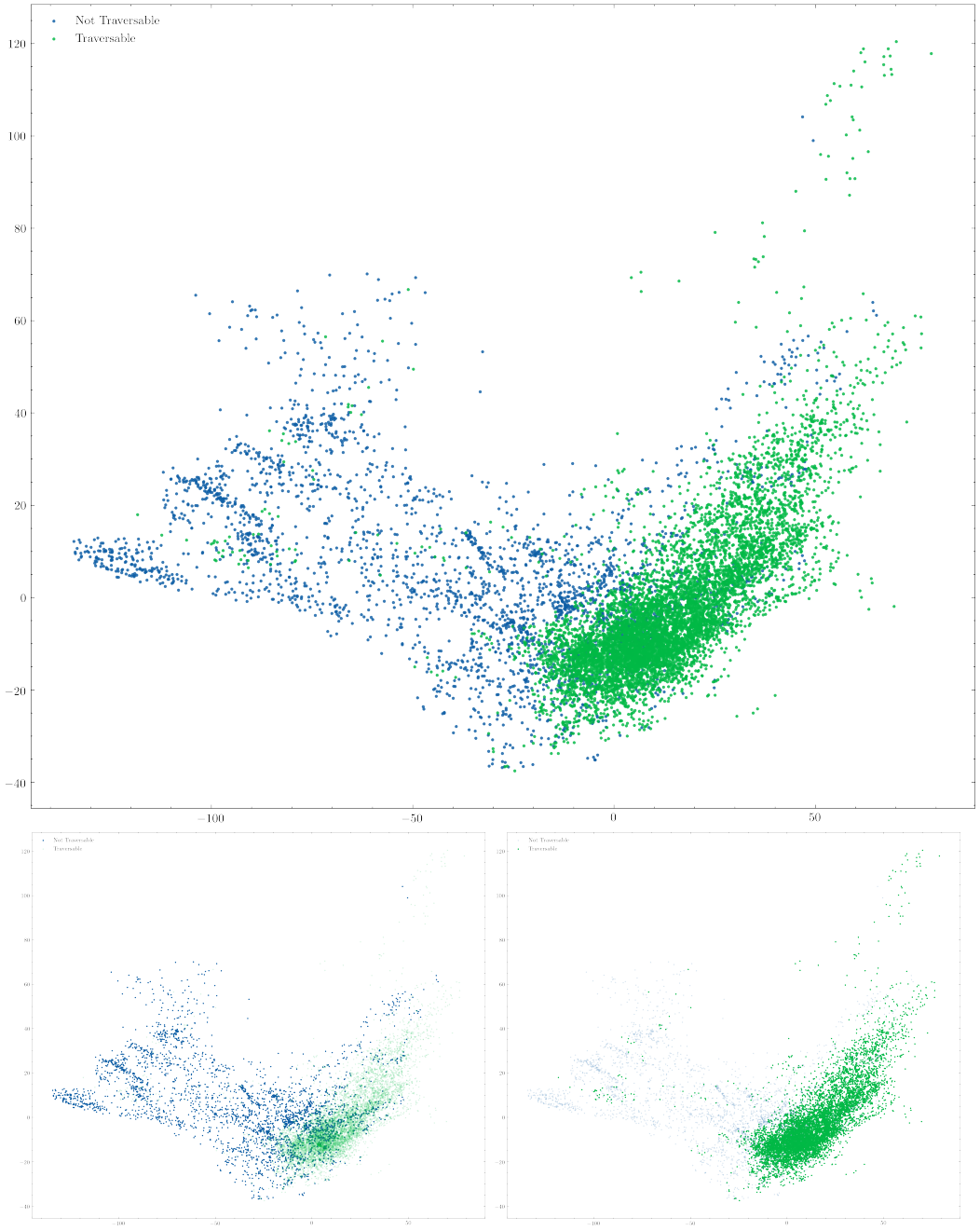
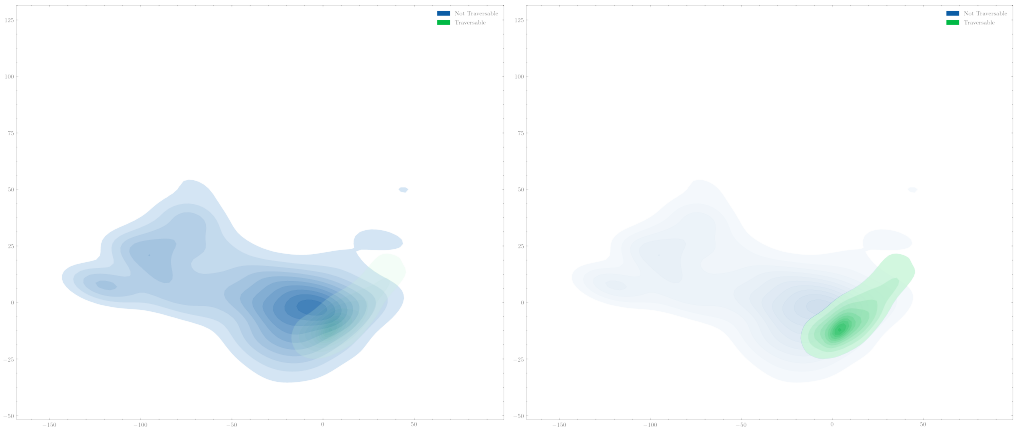


Figure 6.5. Principal Component Analysis on the features space computed using the outputs from the last convolutional layers on the test dataset. We can distinguish two main clusters. However, some points are mixed up between classes.



(a) Not Traversable (b) Traversable

Figure 6.6. ensity plot for test set features space. The more opaque the color the close to the cluster center. The centers of the clusters are close to each other yielding less separability/

false positive and false negative. Then, we took twenty inputs from these sets and applied a technique called Grad-CAM to visualize which part of the ground the model is looking. These regions are highlighted in 3D render to better visualize which spot of the inputs caused the prediction. Before starting the exploration, let us introduce the method that allowed us to identify regions of interest on the patches.

6.2.1 Grad-CAM

Before starting the exploration, let us introduce the method used to identify region of interest on the patches. Gradient-weighted Class Activation Mapping (Grad-CAM) [19] is a technique to produce v̈isual explanationsf̈or convolutional neural networks. It highlights the regions of the input image that contribute the most to the predictions. For example, given a classifier able to recognize cats and cat image, Grad-CAM will point outs the cat. In detail, the output with respect to a target class is backpropagated while storing the gradient and the last convolution layer's output. Then, a global average is applied to the saved gradient keeping the channel dimension in order to get a 1-d tensor, this will represent the importance of each channel in the target convolutional layer. After, each element of the convolutional layer outputs is multiplied with the averaged gradients to create the grad cam. This whole procedure is fast and it is architecture independent. The procedure is summarized by figure **??**

6.2.2 Traversable patches

These samples are the patches correctly predicted as traversable. We plotted twenty different inputs sampled uniformly according to the robot advancement to include as many interesting situations as possible. We recognized two main clusters of images: flat, figures 6.9a, 6.9b, 6.9f, 6.9g, 6.9h, 6.9i, 6.9j, 6.9k, 6.9o, 6.9q, 6.9s, and slopes, figures 6.9c, 6.9d, 6.9e, 6.9l, 6.9m, 6.9n, 6.9p, 6.9t. The models is mostly interested in the left part of the patches with uneven ground on the left region, 6.9a, 6.9b, 6.9c, 6.9d, 6.9e, 6.9g, 6.9j, 6.9l, 6.9m, 6.9n, 6.9o, 6.9p, 6.9r. This is due to the fact that

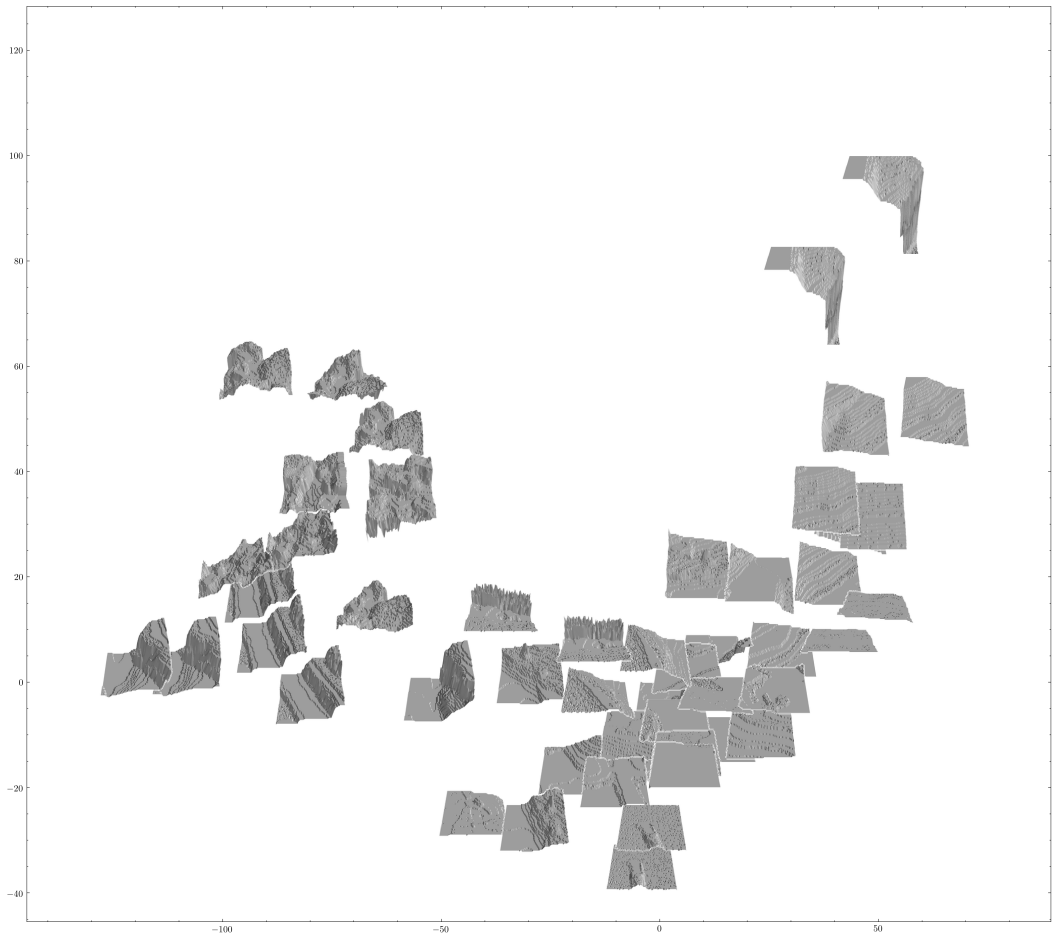


Figure 6.7. Patches that correspond to coordinates in the features space of the last convolutional layers on the test dataset. Similar grounds are close to each other. From the top left in counterclockwise order, we found not traversable patches with clearly not traversable features such us big bumps, huge obstacle towards the end. On the center, there are hardest surfaces to separate composed by slopes and small obstacles. Going up we found downhills and huge cliffs, highly traversable patches.

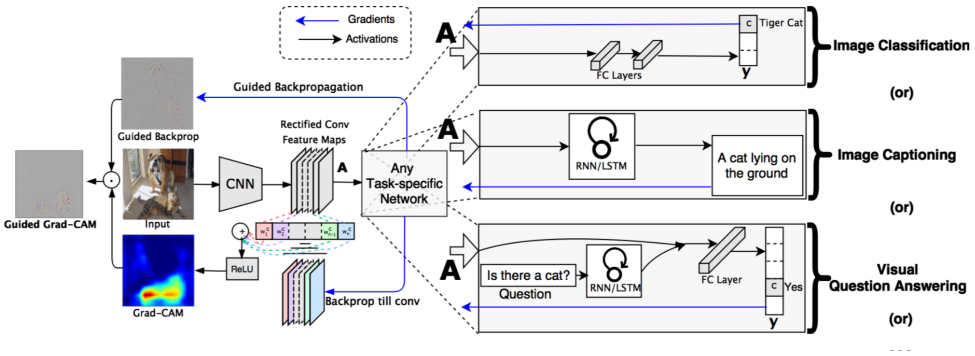
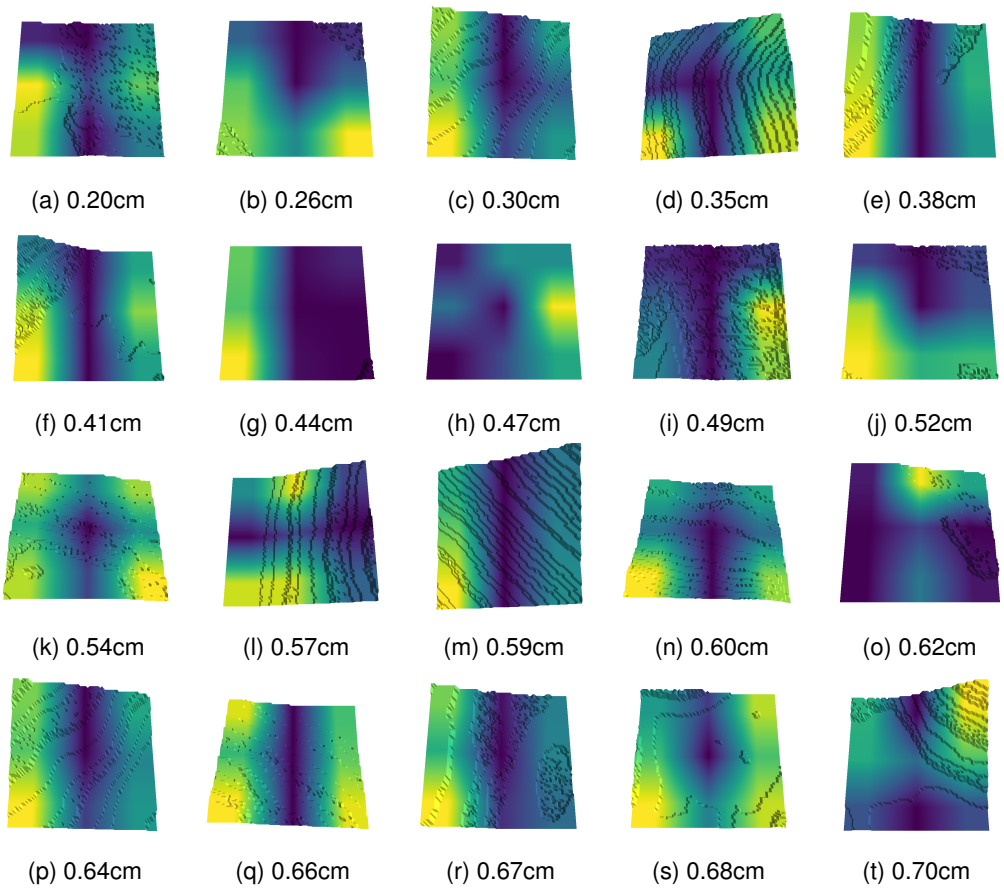


Figure 6.8. Grad-CAM procedure on an input image. Image from the original paper [19].

an obstacle in that region can block the rear legs and prevent the robot to advance. In other patches where most probably a untraversable features may also be present ahead, the model discriminates different parts of the map. For instance, figure 6.9d shows an interest region on the slope near the end and figure 6.9l on two stop of the first faily big step ahead of the robot. Similarly, also figures 6.9n, 6.9q, 6.9s.

In some situations, the model identify a possible untraversable spot only forward being sure the robot is no immeaditely stopped. There are two obvious cases, figures 6.9h , 6.9i and 6.9s. The first one is a totally flat surface, so the model looks as far as possible to check if there are obstacles. We have a similar situation in the last two patches, a surface with a some noise and a big bump respectively, where the network verifies these spots. So, rightly, the network analysis the first region of the patch that may contain an untraversable obstacle.



(a) 0.20cm (b) 0.26cm (c) 0.30cm (d) 0.35cm (e) 0.38cm

(f) 0.41cm (g) 0.44cm (h) 0.47cm (i) 0.49cm (j) 0.52cm

(k) 0.54cm (l) 0.57cm (m) 0.59cm (n) 0.60cm (o) 0.62cm

(p) 0.64cm (q) 0.66cm (r) 0.67cm (s) 0.68cm (t) 0.70cm

Figure 6.9. Traversable patches sampled from the test dataset correctly predicted by the model. We compute dthe Grad-CAM and applied as texture in the 3D render. The yellow highlights the ground region that contributed the most the model's prediction.

6.2.3 Non traversable patches

Not traversable patches correctly classified by the model. In this case, the region highlighted by the Grad-CAM more varied. Like in the previous section, in some patches, 6.10b, 6.10f, 6.10l, 6.10n, 6.10q, 6.10s, the non traversable features is directly under the robot. While, on other grounds with a atraversable left part but a big obstacle ahead, figures 6.10c, 6.10e, 6.10i, 6.10o, 6.10r, the model identified the huge obstacle as the main reason for their untraversability. There mixed cases, figures 6.10b, 6.10d, 6.10j, 6.10m. Other grounds are mostly uneven, 6.10d, 6.10k and 6.10l and the cam highlighted the biggest bumps. The last patch, 6.10h, is very interesting. There is a trail with a obstacle parallel to it. Perfectly, the network identified one part obstacle as responsabile for the prediction. Probably, these spot is the points where the robot hitted the obstacle while going forward.

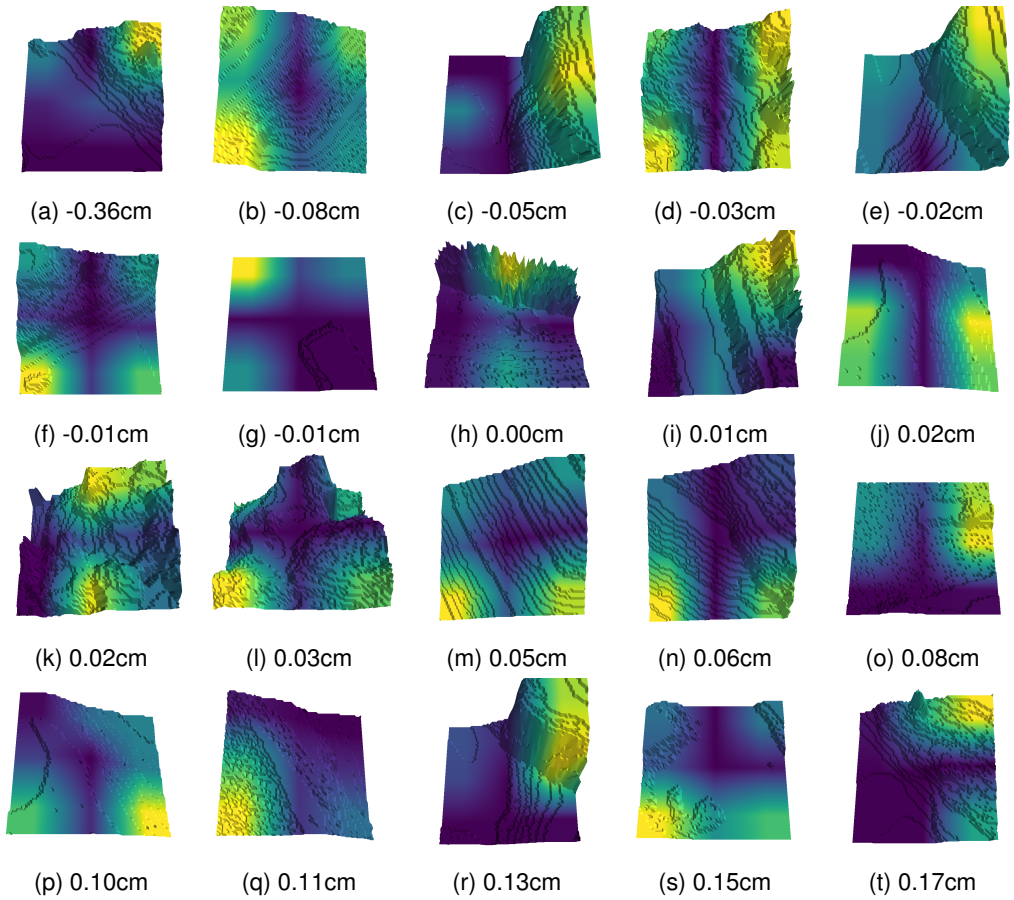
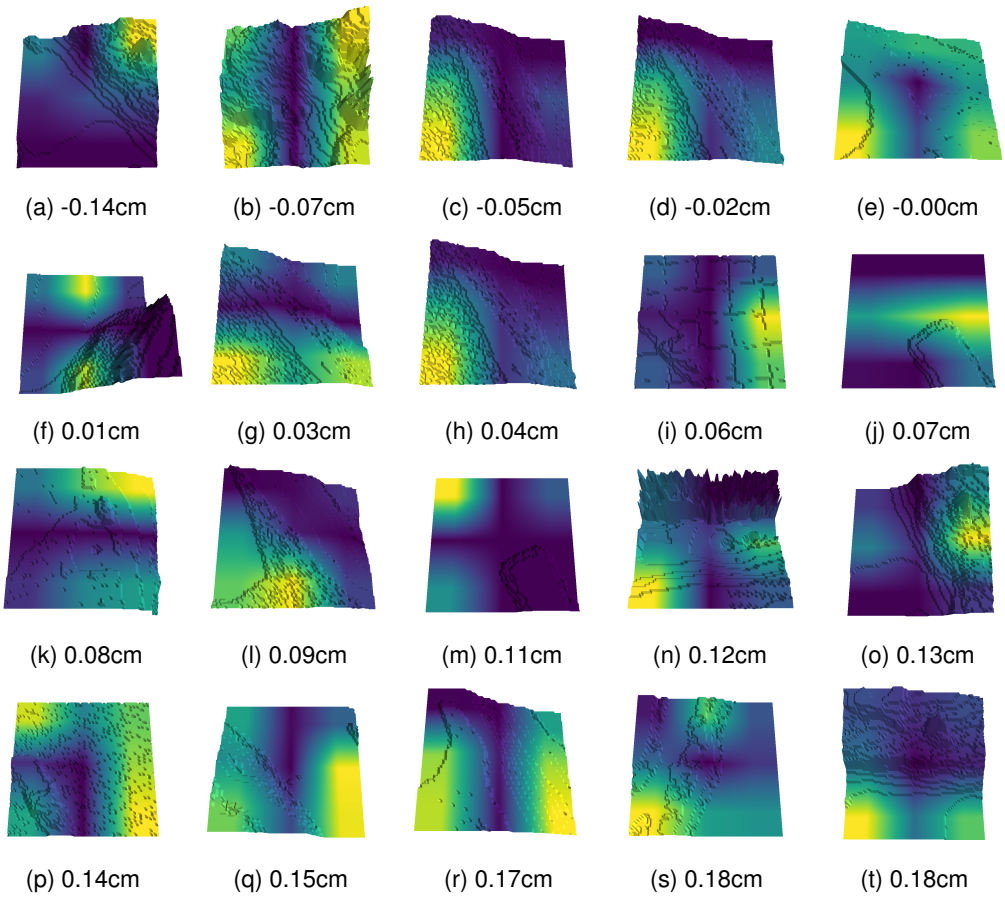


Figure 6.10. Not traversable patches sampled from the test dataset correctly predicted by the model. We compute dthe Grad-CAM and applied as texture in the 3D render. The yellow highlights the ground region that contributed the most the model's prediction.

6.2.4 False negative patches

Not traversable inputs predicted as traversable by the model. Some of the grounds are similar to ones from the last section. Figure 6.10b is almost identical to 6.11d. In this case, the model must have misclassified the first part of the map thinking it could traverse till the two bumps in the end. If so, the robot should have been able to move for more than twenty centimeters. There are two grounds with a trail, 6.11n, 6.11t. In both cases, the model looked at the initial position of the robot, left, and the final part of the patch to understand if the robot fits on the trail. Moreover, there are different slopes in which Grad-CAM highlighted the region under the robot, 6.11c, 6.11d, 6.11e, 6.11g, 6.11h, 6.11s. These regions may be hard to estimate when the sizes of the obstacles are closes to the edge cases. The model may overestimate when the size of the obstacle is just a few centimeters more than the real traversable one. Figure 6.11k has a small step on the right region, correctly highlighted by Grad-CAM. Other patches have obstacles close to the end, 6.11j, 6.11o, 6.11p, 6.11q, 6.11r. ONe patch clearly showed the model confusion, figure 6.11m. Even if the obstacle was ahead, the model discriminated and an empty spot on the top left. in general, most of

the region of interested on these patches are located on the left, close to the position of the robot's leg. Correctly, even if the prediction is wrong, the model looks at the first region of the surface, located near the legs, that can effect traversability. This shows the correct behavior even when misclassifying the inputs, meaning that the network is always looking in the correct spot.



(a) -0.14cm (b) -0.07cm (c) -0.05cm (d) -0.02cm (e) -0.00cm

(f) 0.01cm (g) 0.03cm (h) 0.04cm (i) 0.06cm (j) 0.07cm

(k) 0.08cm (l) 0.09cm (m) 0.11cm (n) 0.12cm (o) 0.13cm

(p) 0.14cm (q) 0.15cm (r) 0.17cm (s) 0.18cm (t) 0.18cm

Figure 6.11. Not traversable patches predicted as traversable sampled from the test dataset. We compute dthe Grad-CAM and applied as texture in the 3D render. The yellow highlights the ground region that contributed the most the model's prediction.

6.2.5 False positive patches

False positive patches are the most interesting inputs. They are traversable patches classified as not. By looking at the features space, **??**, we can noticed how the patches are mostly located close to the non traversable features. This explained why most of them are very close the surfaces presented before in figure 6.10. These samples included different types of terrains, some with obstacles ahead 6.12b, 6.12c, 6.12d, 6.12g, 6.12h, 6.12k, 6.12m, 6.12n, 6.12q, 6.12s, 6.12t, slopes 6.12i, 6.12j and mixed grounds, 6.12a, 6.12e, 6.12l, 6.12o, 6.12p, 6.12r. In each case, the model identify meaningful features that can effect traversability. In the slopes, 6.12i and **??**, the first part of the surface is highlighted. Similar to the traversable patches, the model tried to evaluated if the rear

legs was able to move. This is also true for 6.12f. The only patch that consufed the model is the first one, 6.12r, where it wrongly discriminated the flat region and not the obstacle. In all the others inputs, the network identifed importat region of the map that realistically could have caused the predicted not traversability.

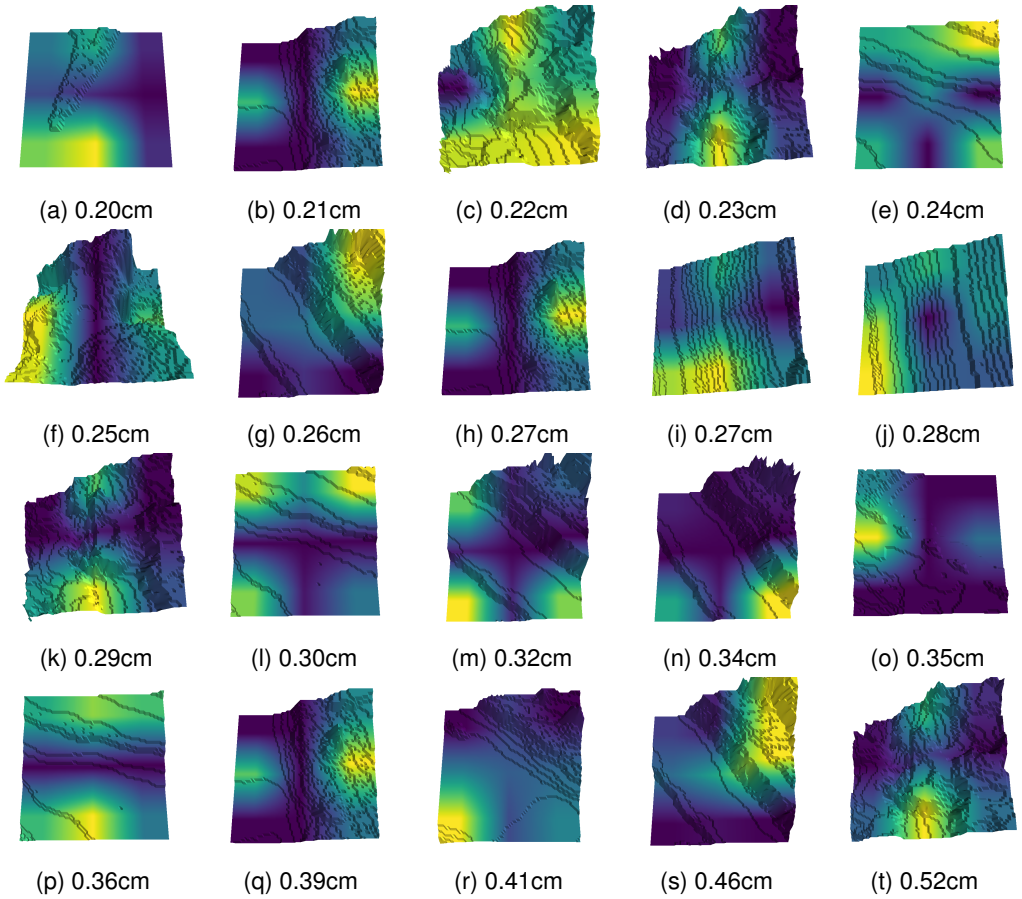


Figure 6.12. Traversable patches predicted as not traversable sampled from the test dataset. We compute dthe Grad-CAM and applied as texture in the 3D render. The yellow highlights the ground region that contributed the most the model's prediction.

6.3 Robustness

To conclude the investigation of the network's abilities, we tested the model's robustness on patches with different features, walls, bumps, ramps, and examined the model's predictions against the real robot advancement obtained from the simulator. In general, the model's outputs matched the ground truth. When the network failed, on some edge cases, it showed a certain degree of uncertainty.

6.3.1 Untraversable walls at increasing distance from the robot

We first test we performed was to place a not traversable wall in front of *Krock* at gradually moving towards the end. At a certain distance, we expected the model's prediction to be traversable even if the wall itself is too tall. Why? Because the robot will be able to travel more than the threshold before being stopped by the obstacle.

We created fifty-five different patches by first placing a wall 16cm width exactly in front of the robot and then move it by 1cm at the time towards the end. Figure 6.13 shows some of the inputs patches ordered by distance from the robot. The model's predictions are displayed in figure

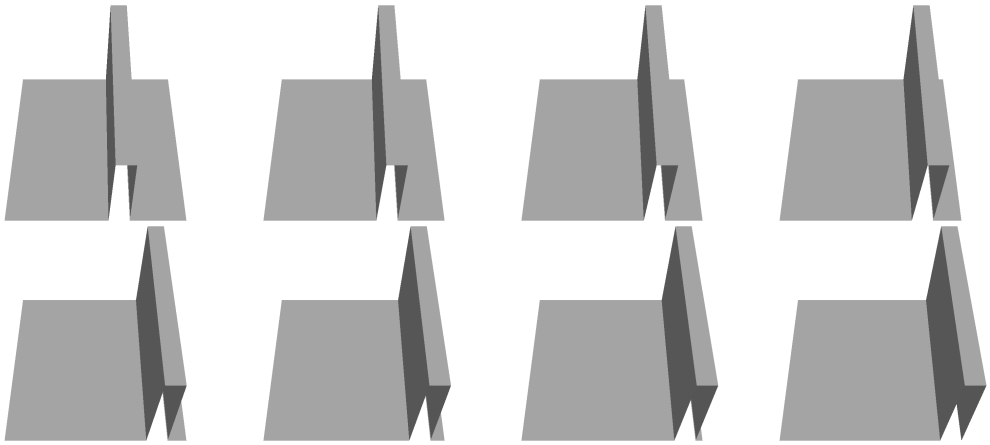


Figure 6.13. Some of the tested patches with a non traversable walls at increasing distance from the robo's headt.

6.14. We can see how the two classes invert their values around 20cm. Moreover, the predictions, are consistent and do not change multiple times. Intuitively, if a wall is traversable from a certain distance, it will still be if we place even further. To be sure the results are correct, we run the last not

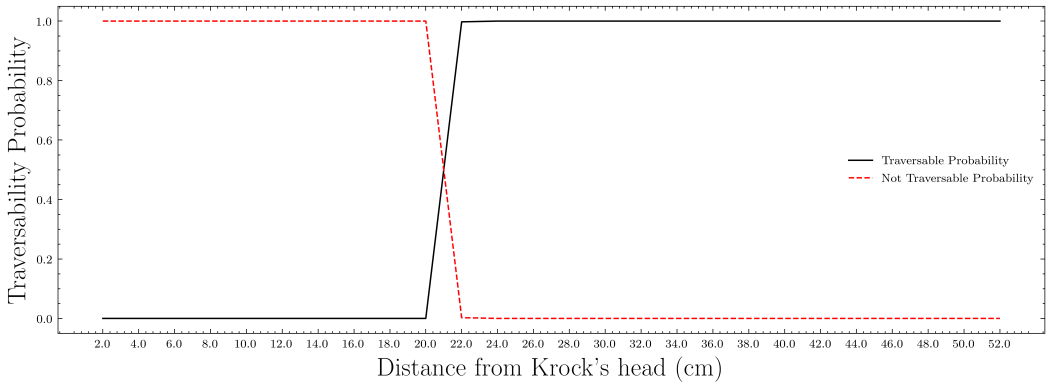
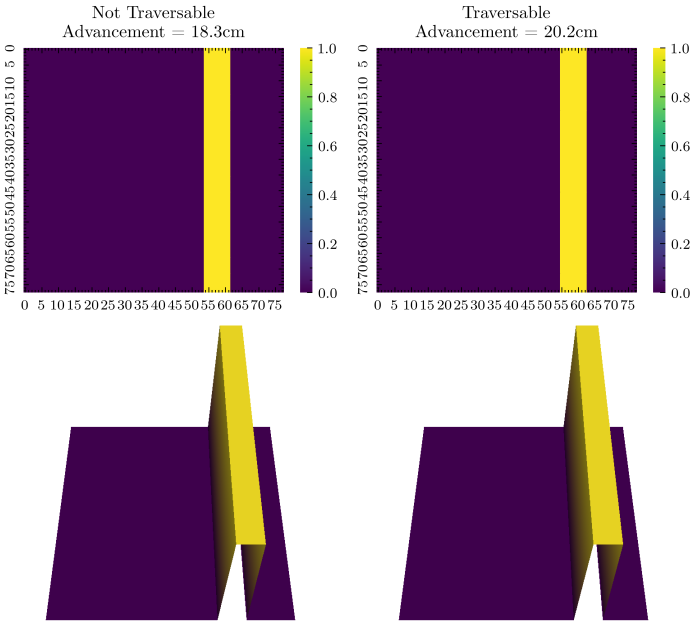


Figure 6.14. Traversability probabilities for patches with a non traversable walls at increasing distance from the robot.

traversable and the first traversable patch on the simulator to get real advancement. In the simulator, Krock advances 18.3cm on the first, not traversable patch 6.15a where the wall is at 20cm from the

robot's head. While, on the first traversable patch, figure reffig :walls-traversable-b, with a wall at 22cm, the robot was able to travel for 20.2cm. Correctly, the network's predictions are supported by the ground truth obtained from the simulation. Even more important, the model understood that the distance from the obstacle is more relevant than its height. Furthermore, we increased the wall size



(a) Distance 20cm (b) Distance 22cm

Figure 6.15. We run the last and first not traversable and traverable patch labeled by the model. Correctly, the real advancement is greater than the threshold when the distance between the robot is also greater.

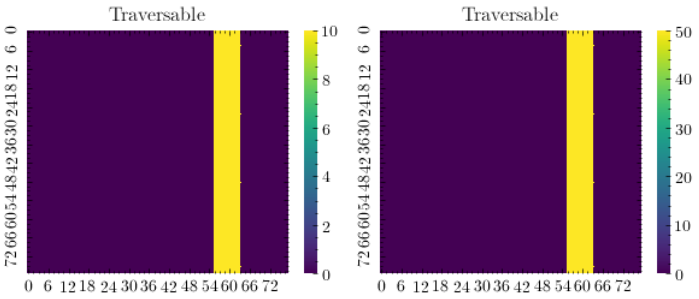
of the first traversable patch, figure 6.15b, to 10 and to 50m to see if the model will be confused. Accurately, the robot was not confused by the enourmous wall and the predictions did no change and the patches were still labeled as traversable. Figure 6.16 visualize these patches.

6.3.2 Walls at increasing height in front of the robot

After we tested the distance from Krock's and a wall, we decided to fix the obstacle position but increase its heights. We run forty patches in the simulator with a wall place exactly in front of the robot with a height from 1cm to 20cm. Figure 6.17 shows some of the inputs.

The models predicted that the walls under 10cm are traversable. We plotted the classes probabilities in figure 6.18 in which we cab observer the model's prediction change smoothly revealing a degree of uncertainty near the edge cases, $\approx$ 10cm. We compared the model's prediction with the advancement computed in the simulator using the same approach from the last section. Figure 6.19 shows the results from the last traversable patch and the first non traversable.

In the first case, the simulator outputted and advancement of 21.2cm meaning that Krock was able to overcome the obstacle, while it failed in the second case. Correctly, the predictions matched the real data.



(a) height $= 10$m (b) height $= 50$m

Figure 6.16. Two patches with a very tall wall at a distance major than the threshold. The model labels them as traversable without been confused by the huge obstacle.

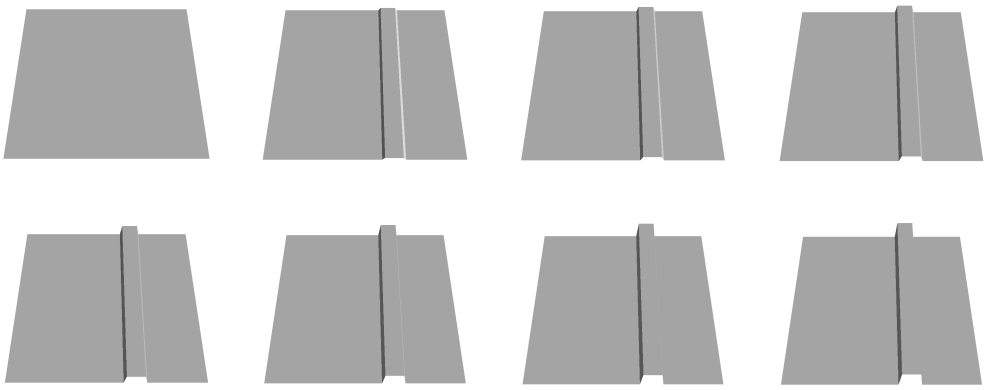


Figure 6.17. Some of the tested patches with a walls at increasing height ahead of Krock.

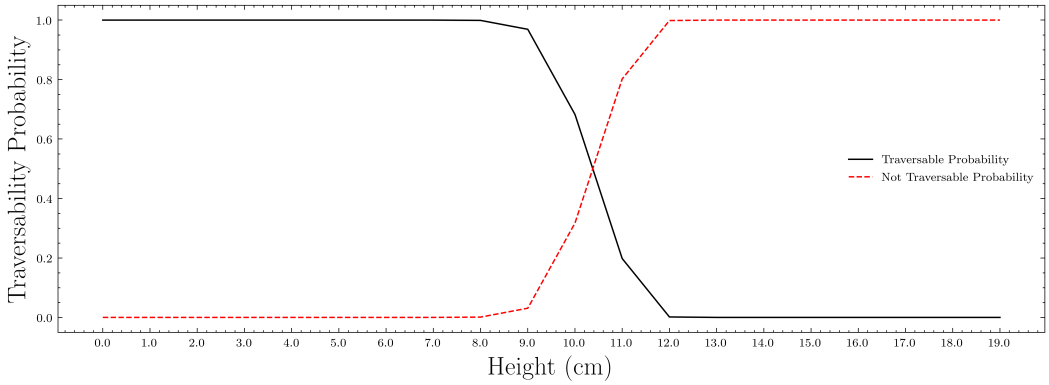
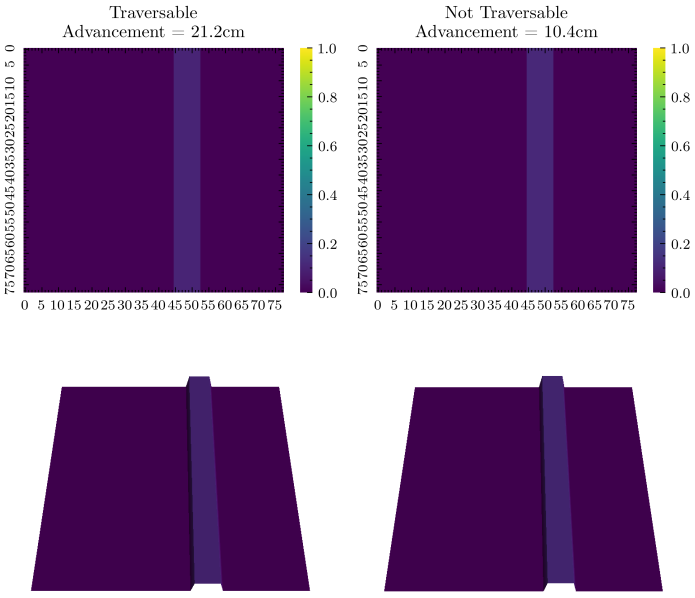


Figure 6.18. Traversability probabilities against walls height in front of Krock.



(a) height 9cm (b) Height 10cm

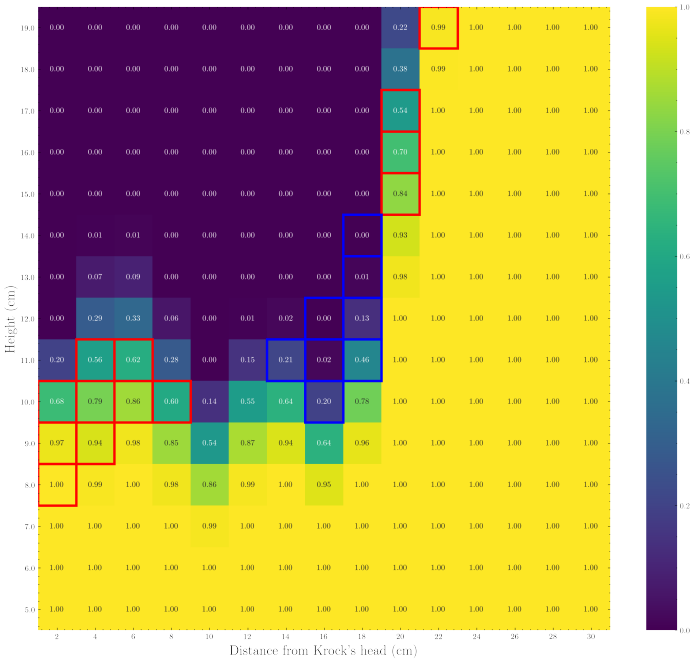
Figure 6.19. We run the last and first traversable and not traverable patch labeled by the model. Correctly the model's prediction matches the advancement from the simulator.

6.3.3 Walls with increasing height and distance from the robot

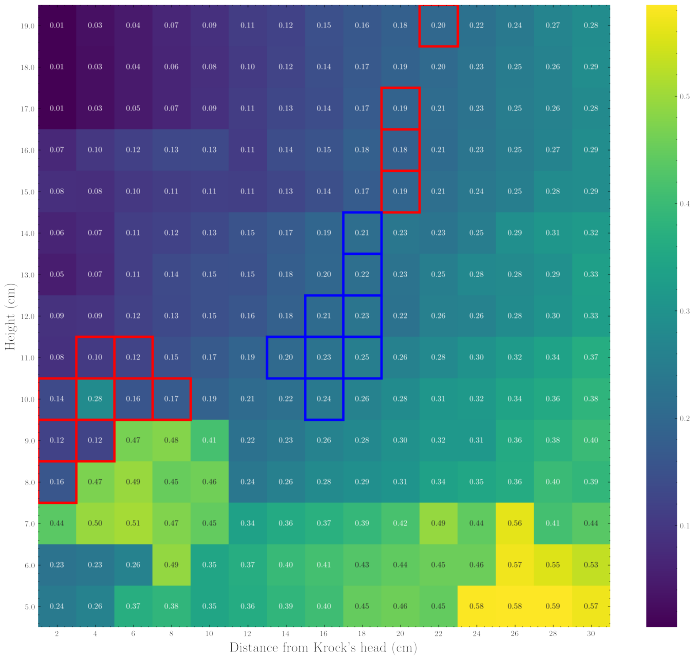
We combined the previous experiments and tested the model predictions against the ground truth for each height/distance combination. To reduce the number of samples and improve readability, we limited to consider only patches with a wall tall between 5cm and 20cm, we know from previous sections patches with a value smaller and bigger obstacle are traversable and not traversable respectively. Similar, we set the wall's distance from Krock's head between 1cm to 30cm for the same reasons. To evaluate the model's prediction, we run all the patches several times on the simulator and average the results. Figure 6.20a shows the models outputs. We highlighted the false positive and the false negative by red and blue respectively. Since we spawned the robot directly on the patch inside the simulator, the outputs may change across different runs. Sometimes, two runs on the same patch can produce slightly different advancement due to some really small changes in the initial position of the robot caused the spawning procedure or some lag. Figure 6.20b displays the real advancements average across six different runs for all the patches. Additionally, for completeness, we also display in figure 6.20c the variance between all simulation's runs to highlighted cases where the advancement changes the most across different runs:

We obtained an overall accuracy of 91%. The model failed to predict some of the edge cases. The false positives are located in two regions: on the bottom left and on the top center. The first ones are the patches with a wall just ahead Krock of heights between $\approx 8 - 11$. The second region appears when the wall is at $\approx$ 20cm, the threshold. But, even if the model failed to classify these inputs, it shows a correct degree of uncertainty. For example, most of the predictions' probability for the patches at 10cm (red cells) are less than 0.7, some of them even close to 0.5.

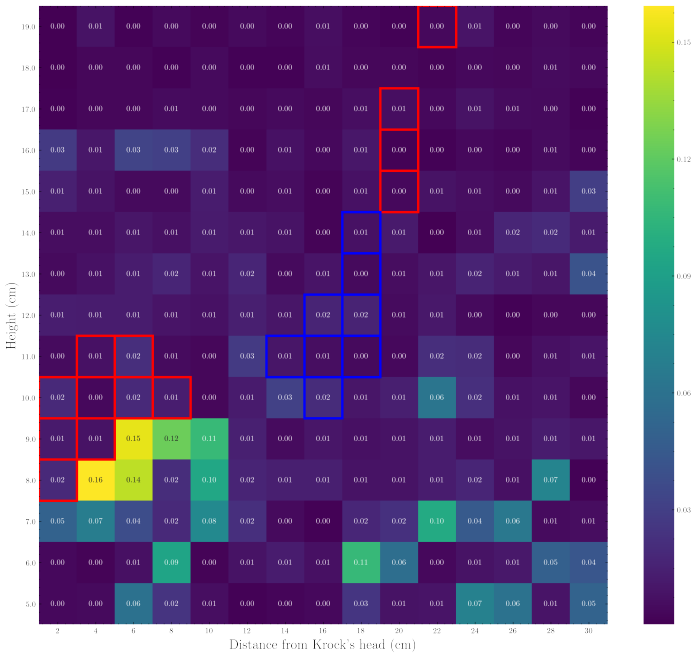
The false negative, (blue), are the inputs at distance close to the threshold and with a wall height



(a) Predictions



(b) Averaged real advancement



(c) Standard deviation of the real advancement across six different runs.

Figure 6.20. Results for all the combination of patches with a wall's distance from $2-30$cm and heights between $5-19$cm. False negatives are labeled with red while false positive with blue. The model failed to classify some edges cases when the wall is very close to Krock's head and when the wall's distance is near the threshold. The overall accuracy is 91%.

between traversable and not traversable. Even if the model wrongly classified some of the inputs, all these errors are in the edge cases where the predicted classes' probability is not maximum. Moreover, in most cases, the model's showed uncertainty, especially on the false negative. Also, the prediction changes smoothly without any spikes accordingly to the features of the terrain. This shows a correct degree of understanding of the surface inputs. For instance, If the model outputs not traversable at height of 10cm and at a distance of 16cm, then all the taller wall are correctly labeled as not traversable showing consistency and predictability.

6.3.4 Corridors

We also tested the model against corridors. We expected the model fails due to the lack of train samples similar to corridors. Only two maps in the whole dataset have featers that could generate corridors. This is confirmed by presence of only few patches with corridors in the features space, figure 6.4. We generated ten different terrains starting from two walls at 30cm from the borders. Figure 6.21 visualizes the inputs. The model scored an accuracy of just 40%. It wronly predicted

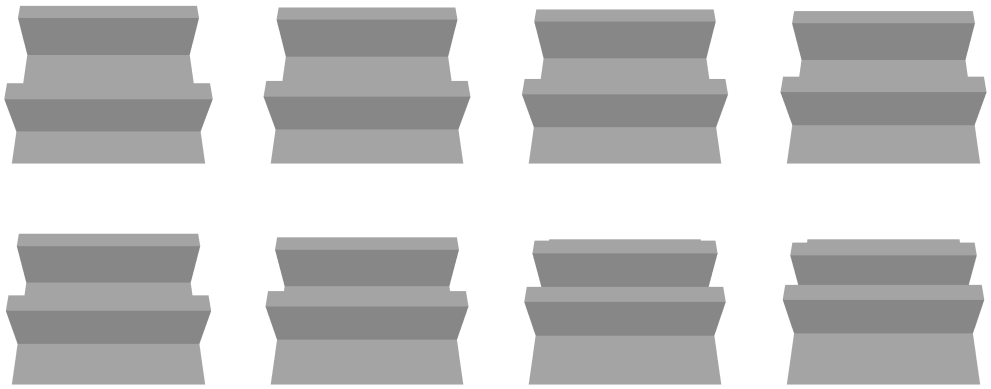


Figure 6.21. Some of the tested patches with corridors created by two walls at different distances.

that all patches expected the last two are traversable. In reality, only the first two patches are traversable, while all the others are not. When the distance between the walls reach a certain point, the robot was stuck and it was not able to move. Interesting, all the patches with corridor's width less than the robot's footprin represented an impossible situation, since the robot cannot go inside an obstacle. But, even some of these inputs were classified as traversable by the model. We computed the Grad-CAM output on last fours samples, figure 6.23 visualizes it. In the first two, wrongly predicted as traversable, the front region was highlighted meaning that the network did not consider the distance between the wallsh. in the last two patches, the network discriminated booth the under and ahead inner part of the corridor. Clearly, the model was confused. To improve the network performance on corridors we incorporate new maps in the train set.

6.3.5 Ramps

We generated twenty ramps with a maximum height from 0.25m to 4m. Figure 6.24 shows some of the inputs. The model labeled as traversable the surfaces with a maximum height less than 1m

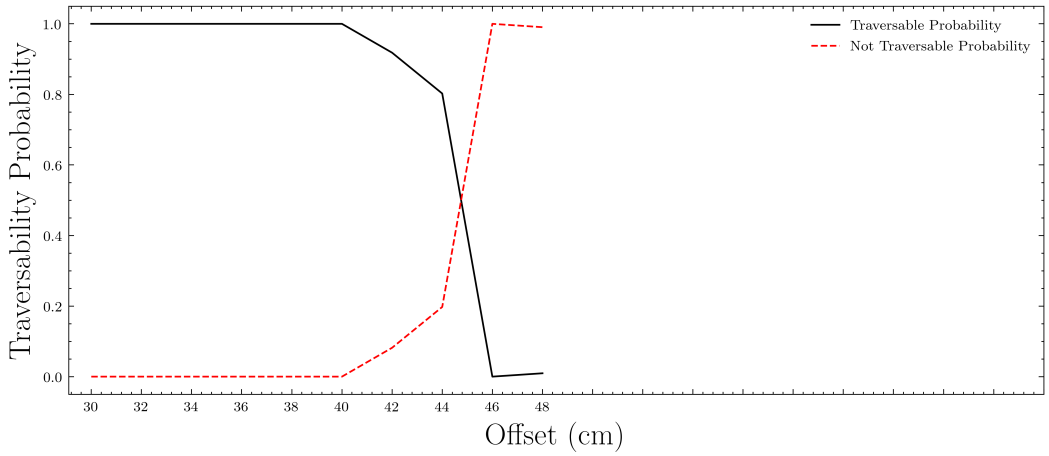


Figure 6.22. Traversability probabilities against corridor offset from the top borders.

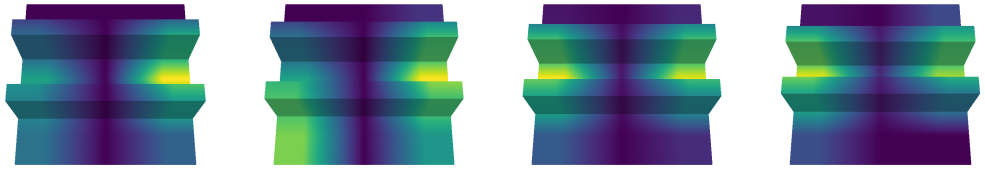


Figure 6.23. Fours patches with corridor's width smaller than Krock's footprint making impossible for the robot to traverse them. Only the last to surfaces were correct label as not traversable.

and not traversable the others, figure **??** plots the traversability probabilities against the maximum height of each ramp. Then, we tested the last traversable patch and the first not traversable with the real advancement gather from the simulator. Figure **??** shows that the model's predictions are confirmed by the ground truth values.

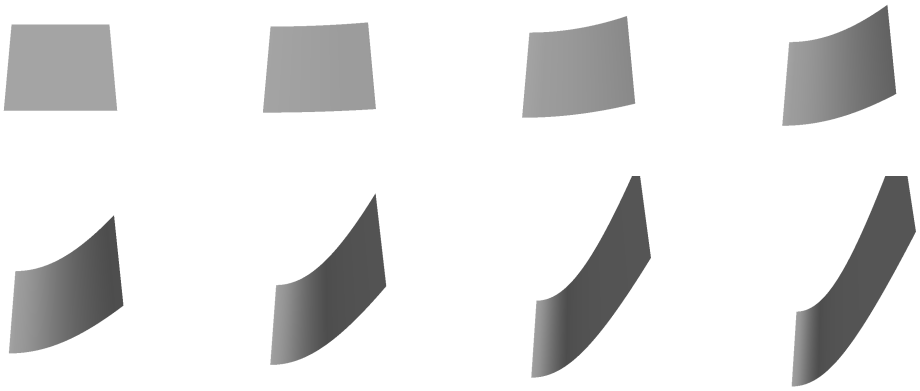


Figure 6.24. Some of the tested patches with steep ramps.

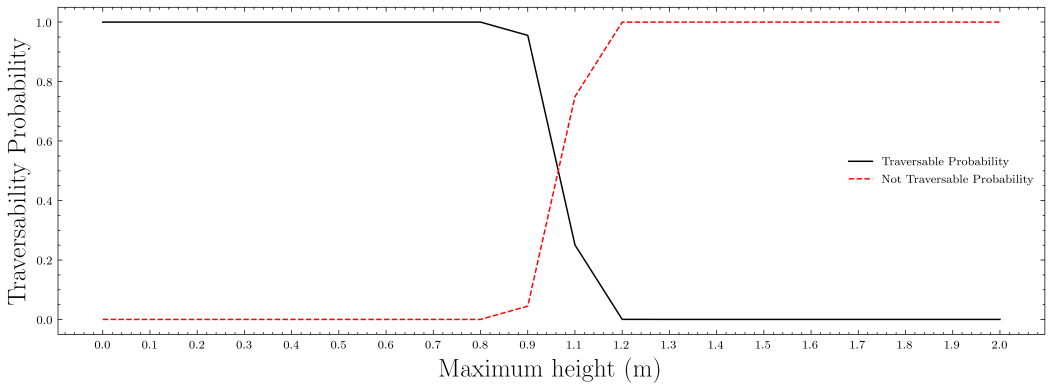
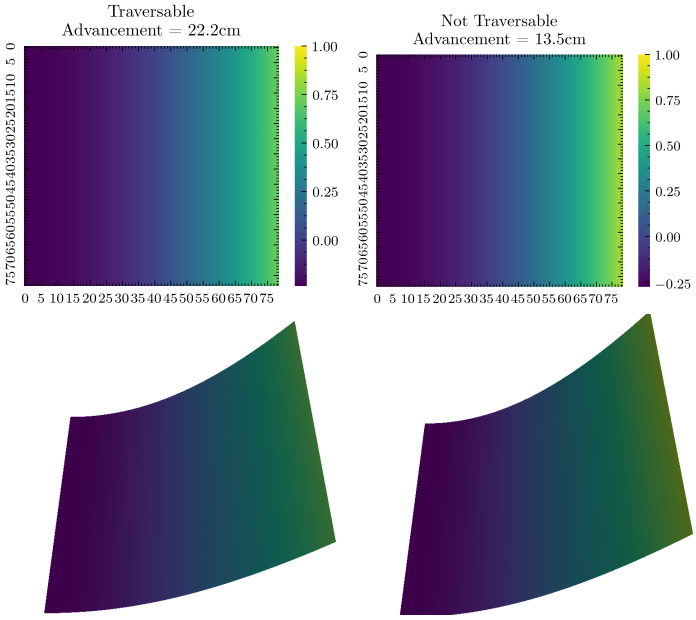


Figure 6.25. Traversability probabilities against maximum height of each ramp.



(a) Height 0.94cm (b) height 1.1m

Figure 6.26. The last traversable and the first non traversable patches with a steep ramp ahead of Krock.

Chapter 7

Conclusions and future work

7.1 Conclusions

We developed a framework for traversability estimation based entirely on simulation data and applicable to any ground robot. Simulation data allowed cheap, fast, quality and massive data gathered since 1) no real robot was used 2) simulations can be run faster than real-time and parallelize 3) maps can be engineered to maximize robots exploration and interactions with specific features 4) simulations can be run for any amount of time.

We trained a classifier to learn the robot's locomotion by supervised learning directly on ground patches. We tested the pipeline on a legged crocodile-like robot. We used a deep convolutional neural network to regress and classify the patches. We have showed both methods to be effective and we selected the latter due to its better accuracy once a threshold is fixed. We quantitatively measured the model's performance using different numeric metrics. Then, we visualized the traversability probability on real-world terrain to qualitatively evaluate the network.

Later, we interpreted the network's predictions applying several approaches. First, we proved that features were correctly aggregate and separate based on their classes. Then, we explored different samples from the test set by visualizing with part of the surfaces caused the model's prediction. We discovered that the model always analyzes the correct ground features. Finally, we analyzed the network's strength and robustness by comparing different custom patches with the real advancement computed in the simulator. The results showed that the model expressed a correct degree of uncertainty in some edge cases while been able to properly classify most samples.

7.2 Limitations

We mention three important limitations of our methodology. First, we did not take into account grounds material but only relied on the terrain's 3D as the only factor to predict traversability, with the ground activing as a rigid object. This is not true when the surface is composed of slippery or sticky elements (e.g. oil or mud) but is a realistic approximation in other scenarios composed by solid ground. The second limitation is given by the data gathering process. The performance of the model directly depends on the quality of the data gathered during simulation that may not be always adequate to replicate the real world scenarios. Lastly, not all of the robot's information is currently used to train the estimator. In the current state, we not include the legs position and we rely only on the torso's pose.

7.3 Future work

The first two problems can be tackled by additional input to the model while training (e.g. a coefficient associated with the material of the terrain). While to overcome the third, we could incorporate the pose of each leg by 1) encoding all the legs pose and including them as an input to the network 2) by generating one patch for each limb and stacking them together.

Bibliography

[1] e. a. A. Lagae. "a survey of procedural noise functions". *Comp. Graphics Forum*, 29(8):2579–2600, 2010.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning, 2016.

[3] J. G. Alessandro Giusti, D. C. Ciresan, F.-L. He, J. P. Rodríguez, F. Fontanam, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella. A machine learning approach to visual perception of forest trails. 2016.

[4] A. Y. N. Andrew L. Maas, Awni Y. Hannun. Rectifier nonlinearities improve neural network acoustic models.

[5] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti. Learning ground traversability from simulations. 2017.

[6] J. Delmerico, A. Giusti, E. Mueggler, L. M. Gambardella, and D. Scaramuzza. On-the-spot training" for terrain classification in autonomous air-ground collaborative teams. 2017.

[7] J. Delmerico, E. Mueggler, J. Nitsch, and D. Scaramuzza. Active autonomous aerial exploration for ground robot path planning. 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks, 2016.

[10] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks, 2017.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[12] T. Klamt and S. Behnke. Anytime hybrid driving-stepping locomotion planning. 2017.

[13] M. B. L. Wagner, P. Fankhauser and M. Hutter. Foot contact estimation for legged robots in rough terrain. 2016.

[14] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.

[15] W. Lorenz, D. Alexey, R. Ren, W. Krzysztof, C. L. Cesar, and H. Marco. Where should i walk? predicting terrain properties from images via self-supervised learning. 2019.

[16] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. 2002.

[17] P. Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey, 2013.

[18] M. Quigley, B. Gerkey, K. Conley, T. F. Josh Faust, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. 2009.

[19] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization, 2016.

[20] J. Shlens. A tutorial on principal component analysis, 2014.

[21] L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018.

[22] B. Sofman, E. Lin, J. A. Bagnell, N. Vandapel, and A. Stentz. Improving robot navigation through self-supervised online learning. 2006.

[23] E. Ugur and E. Sahin. Traversability: A case study for learning and perceiving affordances in robots. 2010.

[24] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krusi, R. Siegwart, and M. Hutter. Navigation planning for legged robots in challenging terrain. 2016.