Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

June 17, 2016

Using Machine Learning and Simulation to predict the traversability of a given terrain by a simple robot

Stefano Peverelli

Abstract

Autonomous vehicles in unstructured outdoor environment need to estimate the *traversability* of the terrain.
This thesis presents a new approach based on simulation and machine learning to predict whether a given 3D region is traversable by a simple ground robot.
The proposed approach has two main phases: the first consists of simulating a robot that traverses a 3D terrain in a virtual environment, determine in which areas it got stuck or capsized, and collect data from the simulations to obtain an extensive dataset of possible outcomes. The second phase instead aims to analyze the resulting dataset and use it to train a neural network model, whose purpose is, to judge the traversability of a terrain (by its 3D structure), and plan a path the robot can traverse.
By using an approach based on machine learning, there's no need to manually define heuristic rules to predict the traversability of terrain patches; moreover, the approach is applicable to any type of ground robot (wheeled/legged) as long as it can be reliably simulated.

Advisor
Prof. Luca Maria Gambardella
Assistants
PhD Alessandro Giusti, PhD Jerome Guzzi

Advisor's approval (Prof. Luca Maria Gambardella): Date:

Contents

1 Introduction

Robot navigation in outdoor environments is a very challenging task, as traversability is a really complex function of both terrain features and robot characteristics. This thesis approaches terrain scoring from a machine learning perspective: the proposed method uses a robot's experience in navigating the environment to train a classifier.
Data collection is achieved in a virtual environment, which offers a good and cheap starting point, despite of certain limitations, such as numerical approximation and the inability to resemble vegetation or sloping ground.
A main assumption of this approach is that a terrain map is available at priori (e.g. 3D radar surface reconstruction). Each simulation consists of a challenge for the robot, which, starting from an initial random point, needs to follow a predetermined path to reach a goal point. For convention we will call the square area given by the path a *patch*. The patch itself contain a lot of information about the terrain surface the robot has dealt with, steepness or presence of peaks and depressions, thus from the patch we then compute some statistics that the classifier will use during its training.
The evaluation of the quality of predictions made by the classifier, is performed by computing the area under the *receiver operating characteristic* (ROC) curve, which features both true positive and false positive rates; this evaluation phase is tested for several linear methods, and all scores are presented in the Evaluation and Results section.

1.1 State of the Art

Previous research has addressed the problem of estimating traversability for ground robots in outdoor environments, following different approaches.
Nowadays, thanks to the progress made by 3D vision sensors, a common trend is to combine acquired RGB-D images to machine learning techniques, for terrain traversability assessment.

A solution in this direction is proposed in [2]; the authors presents a new framework for online-learning achieved by the correspondence between vehicle's navigation experience and visual terrain data.
Navigation experience such successful traverses, or collisions is assessed by robot's on-board sensors like (IMU [10]), while terrain data is labeled as traversed or not by stereo imagery; automatic labeled data is then passed in input to an online classifier that trains incrementally as the robot navigates the environment.

Another example of the use of RGB-D images acquired by *time of flight cameras* is presented in [1].
The proposed method analyze and segments RGB data streams into group pixels belonging to the same physical object. Then from the depth data stream, it estimates the ground plane (by computing the *v-disparity* [8]) to obtain a first measure of traversability. From this first estimation, an arbitrary number of pixels with an high probability of belonging to the ground plane is considered, and for each of pixel, a scoring function is computed.

Another work that relies on the acquisition of RGB-D images is the one presented in [9]. Interesting is the introduction of a novel 3D descriptor, the (UPD) *uneveness point descriptor*, that observes the distribution of surface's normal vector directions using (PCA) [11] to score terrain features and detect obstacles.

The main difference between these approaches and our work is the use of simulation combined with machine learning. As virtual simulation environments are becoming more realistic and powerful, they offer a good compromise to acquire data in a cheap, but reliable way. Moreover, this approach can be tested on any type of robot (at no additional cost), as long as it can be accurately simulated.

2 Project requirements and analysis

2.1 Learning of traversability

In order to exploit machine learning techniques to create trained models that can judge the traversability of a given terrain, a dataset of terrain features has to be generated. Obtaining a dataset in the real world requires a lot of time and specific tools, instead a proper simulation environment is better suited. Among various robot simulation platforms we have chosen V-REP [3]. V-REP offers a complete simulation environment that includes physics and robot terrain interactions. In the next section the basic structure of the framework is presented.

2.2 Simulation

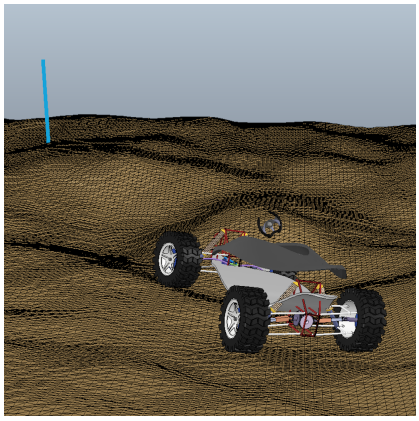

Figure 1. The robot approaching the goal point

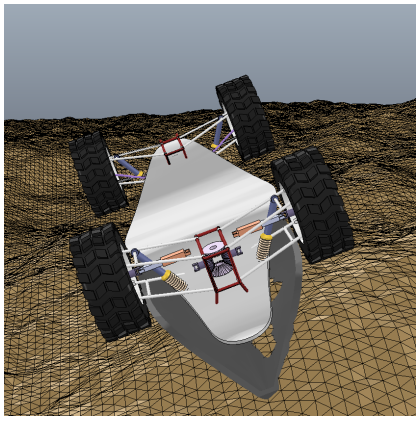

Figure 2. A possible scenario, a capsized robot

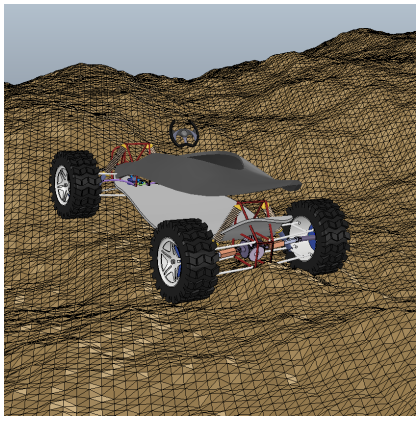

Figure 3. A possible scenario, a stuck robot

The general idea behind the simulation is to place a robot on a terrain moving towards a goal point, and to stop it when a break condition happens: either the robot has reached the goal point or it has failed to. In order to ensure this boolean condition a time limit has to be decided; thus a stuck or capsized robot will result in a failed attempt.
At each simulation time step specific data is computed, like the robot's ideal trajectory, yaw and position; the main reason behind the recording of these data is that the robot needs to be guided to the goal point, so it needs its steer to be corrected while encountering terrain irregularities. Eventually, data such as the goal point position, robot's initial and final position, elapsed time and success or failure, are saved into a CSV file. In order to simulate different scenarios, robot's velocity and distance to the goal can be passed via input to the script that handles the simulations.

2.3 Terrain Generation

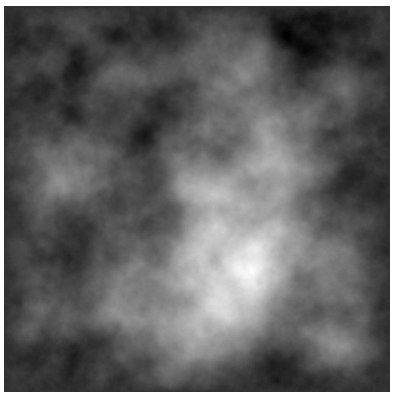

Figure 4. An example of generated heightmap

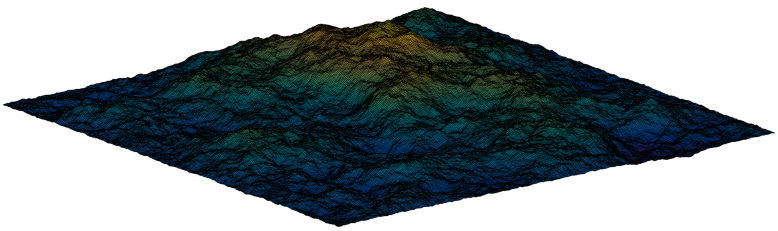

Figure 5. The corresponding 3D terrain

When simulating a virtual environment, reality must be resembled as much as possible, thus one must be as precise as possible when generating rough terrains. The task of creating random but meaningful and realistic terrains can be accomplished using just random noise and the *Gaussian blur* function, which create *heightmaps* that are then extruded to 3D terrain.

2.4 Data postprocessing

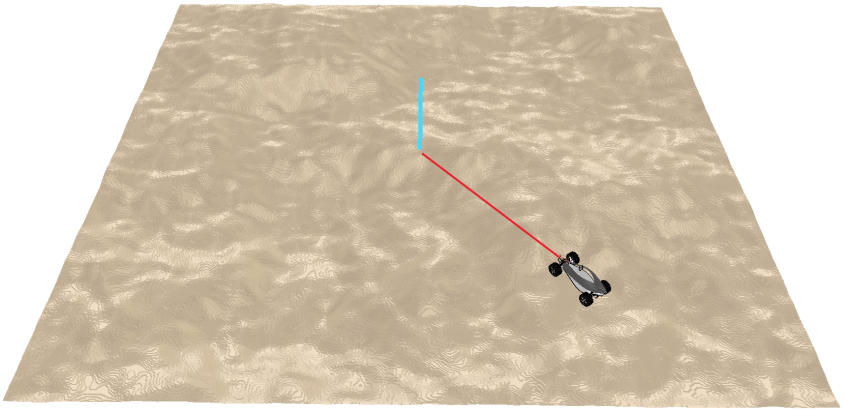

Figure 6. Simulation scenario: robot(1m x 0.5m x 0.4m) is placed in an initial random position (x,y) with a goal point at distance d (in this case d=4m), on a 10x10m heightmap. In red is highlighte the ideal path the robot will try to follow to reach the goal.

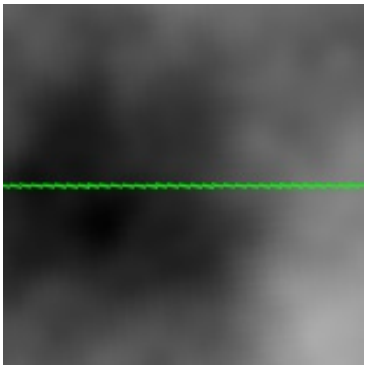

Figure 7. The cropped and rotated patch we have extracted from the robot's navigation, from which we compute the set of feature for the training dataset

In order for a classifier to actually learn to judge a given terrain, we need to extract some features from each patch that comes out from the simulations in a post-processing phase.
The concept for which a patch is traversable, strictly depends on the direction taken into account, thus the need to perform a roto-translation of each patch, so that only one direction is taken into account (from west to east). From each rotated patch then we compute the following statistics: mean, variance, mean and variance of the gradient of x, and mean and variance of the gradient of y. These are the features that compose the actual dataset.

2.5 Classification

Classification is the process that exploits what learned from labeled data and tries to predict the class of unlabeled data; labeled data is simply the training dataset obtained from the simulations and the post-processing phase, and unlabeled data are the features extracted in a post-processing phase from a testing heightmap. In order to build and test a trained model, we have chosen scikit-learn [6], a python based framework for machine learning.
Supervised learning in scikit is very straightforward; it offers a variety of linear models to work with, so there's no need to actually build a classifier from scratch.

2.5.1 Learning algorithms

For the learning process several algorithms were taken into account, from support vector machines to logistic regression, eventually focusing on this last one, which is better suited when investigating a dichotomous behavior.

2.5.2 Logistic regression

Logistic regression is a *regression model* to estimate the probability of a binary response based on one or more independent variables. The main goal of logistic regression is to find the best fitting model between the set of independent variables (features) and the binary characteristic of interest (target) using a logistic function.
The logistic function $\sigma(t)$ [Figure **??**] is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{1}$$

where t is a linear combination of multiple explanatory variables:

$$t = \beta_0 + \beta_1 x \tag{2}$$

In order to estimate the probability then, one must first estimate the coefficients. Unlike linear regression, where parameters are estimated using the method of least squares, in logistic regression they are estimated by using the *maximum likelihood estimation* method [4]. While it is possible to apply linear regression to estimate the probability, the result would be an unbounded measure, instead with logistic regression, the estimation is guaranteed to be inside the closed interval $[0, 1]$.

3 Project Design

3.1 Simulation and Data Collection

V-REP offers a variety of choices on how to interface with its core features. Figure 8 shows the main V-REP's architecture.
The are four main ways to access V-REP functionalities:

- the regular API
- the remote API
- the ROS [7] interfaces
- the auxiliary API

For this work we have chosen the regular API. The regular API is composed by several hundreds of features which can be invoked from C/C++ application or extended by custom lua functions. Since the main client application provides a simple way to handle the simulation by controlling the main script, the chosen approach was to only extend the desired functionalities by writing lua code. As mentioned in the previous section, the robot needs to be guided towards the goal point, thus the need for writing a controller.

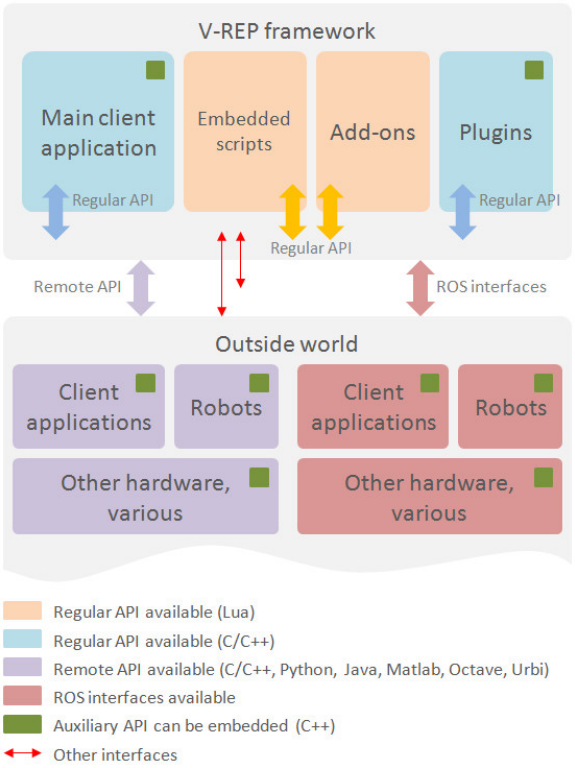

Figure 8. The V-REP API framework

3.1.1 The robot model and its controller

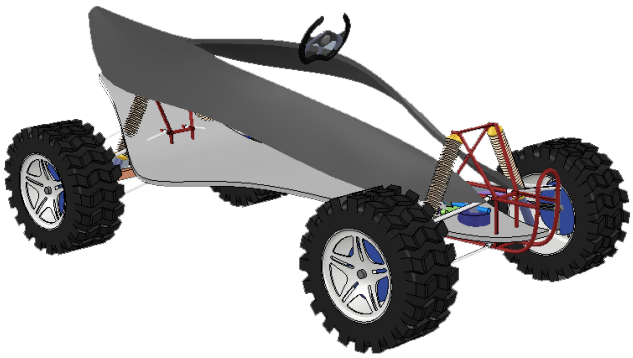

Figure 9. The "Manta with differential" model (courtesy of Qi Wang)

We used the robot model shown in Figure 9; it was chosen as a simple but also ideal representation of an off-road vehicle.
In V-REP each model has its own associated script which describes its behavior during simulation. Each script is composed by three main parts such initialization, actuation and cleanup phases.
The initialization phase is executed just once (when the simulation starts) providing a safe environment to initialize parameters like maximum time per simulation, motor velocity, maximum steer angle and more others. Table 3 lists the core parameters and corresponding values used.
The actuation part gets executed at each simulation time step ($dt = 50ms$) and is the core of each script, in which the actual behavior of the model is described.
Thus each $50ms$ robot's direction is checked, and if necessary, corrected; this process is repeated until either the

maximum time per simulation has elapsed, or the robot is in the area of interest of the goal point. At this point all necessary data is saved into a CSV file as shown below.

TRAVERSED	ELAPSED_TIME(s)	GOAL(x,y)	R_I(x,y)	R_F(x,y)
1	2.90	0.61, -0.40	0.58, -0.10	0.60, -0.35
1	3.05	0.79, -0.79,	0.52, -0.66	0.73, -0.77
0	10.00	0.38, -0.56	0.64, -0.70	0.55, -0.63
0	10.00	0.47, -0.51	0.71, -0.70	0.55, -0.55
0	10.00	0.61, -0.25	0.34, -0.13	0.33, -0.13
1	5.05	0.78, -0.15	0.62, -0.41	0.77, -0.18
0	10.00	0.61, -0.14	0.62, -0.44	0.61, -0.23
0	10.00	0.68, -0.41	0.70, -0.71	0.67, -0.49
0	10.00	0.82, -0.15	0.72, -0.43	0.77, -0.20
0	10.00	0.21, -0.53	0.48, -0.66	0.50, -0.62

Table 1. Instance of data saved from the simulations

The first column identifies whether the robot has reached the goal point or not, the second is just the elapsed time in seconds, the third column represents the screen coordinates of the goal point, and the last two columns, the robot initial and final screen coordinates.

MAX_SIM_TIME(ms)	MOTOR_VEL	MAX_STEER_ANGLE(in °)	AREA_OF_INTEREST(radius in m)	DIST_TO_GOAL(in m)
10'000	1.0	30	0.2	3.0

Table 3. Default values for robot parameters

3.1.2 The main script

In order to run a simulation, V-REP needs a scene object; this is the place where all required objects are displayed. The first task is to load a heightmap into a new scene, this can be easily done from the main menu, or via the API, although only *collada* and *wavefront* objects are currently supported.
Once the heightmap is extruded to a 3D terrain, that's when the main script gets executed. In the initialization phase the robot model is loaded an positioned in a random position onto the heightmap. Then the goal point is placed in the direction of the robot ad a distance d (defined via input). As the simulation is handled by the robot's script, the main script does not do anything in its actuation phase.

3.2 Terrain Generation

$$A = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix}$$

Figure 10. An example of a tridiagonal matrix

Terrain generation is handled by a MatLab script. The whole process is pretty straightforward, once defined the resolution (512x512) a tridiagonal matrix (Figure 10) from *Poisson's discrete equation* is generated. Then the matrix is divided by another matrix of normally distributed random numbers, and converted to a gray-scale image. At this point a *Gaussian blur* with $\sigma = 3$is applied to the resulting image. An example is shown in Figure 14.

3.3 Data postprocessing and Dataset Generation

As what we obtain from the simulations is simply raw data, it needs to pass a post-processing phase. This phase is handled by a python script which parses each line of the CSV generated from the simulations and produces a *numpy* [5] array of features and targets.
From each CSV record a patch is extracted, this patch (as Figure **??** shows) is not oriented by robot's yaw, so in order to rearrange it, the method `extract_patch` was implemented. First, a 2x3 transformation matrix is computed, then the computation is handled by an OpenCV method called `warpAffine`, which returns what is shown in Figure 7.
From the oriented patch, the previously mentioned statistics are computed, then they are stored in a *numpy* array as following:

MEAN	**VARIANCE**	**MEAN_GRAD_X**	**VAR_GRAD_X**	**MEAN_GRAD_Y**	**VAR_GRAD_Y**	**TRAVERSED**
DATA						TARGET
1.20e+02	2.70e+03	2.73e+01	8.53e+02	-1.03e+01	1.37e+03	1
1.14e+02	2.57e+03	2.83e+01	1.10e+03	1.16e+01	1.12e+03	1
7.62e+01	1.37e+03	1.40e+01	1.15e+03	2.77e+00	1.16e+03	0
9.98e+01	2.48e+03	3.01e+01	1.02e+03	-5.73e+00	1.39e+03	0
6.62e+01	9.96e+02	1.34e+01	1.37e+03	-2.04e+00	8.25e+02	0
1.38e+02	1.25e+03	1.92e+01	9.69e+02	-8.63e+00	8.67e+02	1
1.25e+02	7.60e+02	1.44e+01	7.99e+02	4.93e+00	9.31e+02	0
9.18e+01	1.45e+03	1.64e+01	1.09e+03	-1.10e+01	1.03e+03	0
1.14e+02	3.99e+02	9.33e+00	1.39e+03	-3.21e+00	6.49e+02	0
1.15e+02	4.54e+02	6.20e+00	7.77e+02	2.71e+00	8.70e+02	0

Table 4. An instance of the training dataset

3.4 Training

Training is handled by a python script which loads a dataset of 5000 samples and fits a classifier using *logistic regression*.

4 Evaluation and Results

In order to test the accuracy of the trained classifier we have computed some relevant statistics for the following linear methods:

- Nearest Neighbor (with 3 as the number of neighbors)
- Support Vector Classificiation (with 3 different penalty parameters C=0.01, C=1, and C=100)
- Linear Support Vector Classification
- Logistic Regression
- Gaussian Naive Bayes
- a dummy classifier with a "most_frequent" strategy (always predicts the most frequent label in the training set)
- a dummy classifier with a "stratified" strategy (generates predictions by respecting the training set's class distribution)

The script that handles the evaluation part, offers the possibility to apply *cross validation* which is a technique used when working only with a unique dataset for both training and testing, that prevents unpleasant mistakes such *overfitting*, by taking a random split of the dataset for testing.
Following is presented an evaluation of our classifier for the previously mentioned methods using a different testing dataset.

METHOD	AUC	ACCURACY
KNeighborsClassifier(n_neighbors=3)	0.99	0.98
SVC(C=0.01)	0.91	0.81
SVC(C=1)	0.95	0.90
SVC(C=100)	0.98	0.97
LinearSVC(C=1.0)	0.90	0.83
LogisticRegression(C=1.0)	0.96	0.93
GuassianNB()	0.88	0.83
DummyClassifier('strategy=most_frequent')	0.50	0.69
DummyClassifier('strategy=stratified')	0.49	0.57

Table 5. Evaluation of each the classifier

For each method we have computed both accuracy and *auc* reaching good scores in estimations. Below is the graphic visualization of the ROC curve for the logistic regression method:

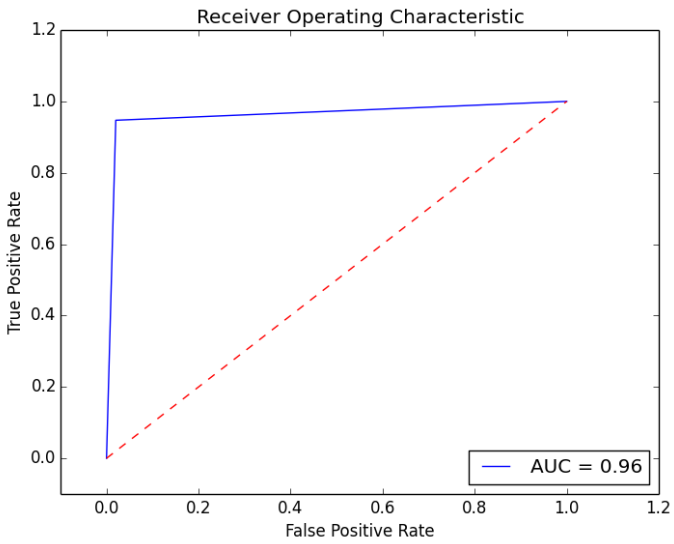

Figure 11. The ROC curve using logistic regression

As traversability depends on the direction in which a terrain is traversed, we have tried to test how our classifier behaves, analyzing 8 different directions (0 to 360 by a step of 45 degrees). Following are two experiments on two different 10 x 10m heightmaps.
Each heightmap is represented as a 10x10 grid, so each cell has dimensions 1m x 1m; the classifier estimates runs of 2m, coloring each cell with green or red depending on a positive or negative prediction.
The first heightmap contains an X-shaped tiny wall, while the second one represents a more realistic scenario.

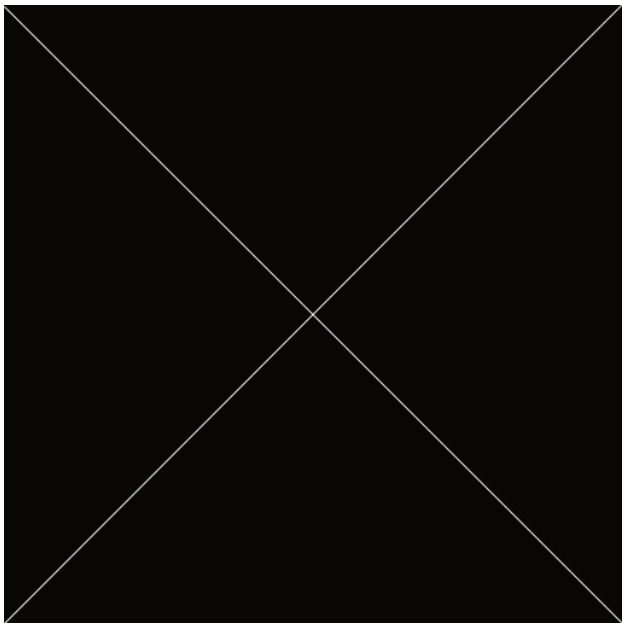

Figure 12. The first tested heightmap: there's a tiny X wall inside it

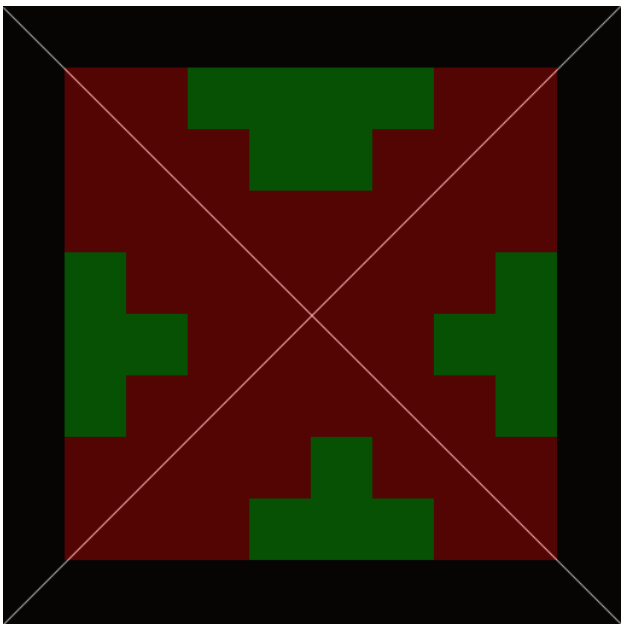

Figure 13. the heightmap traversed in ↓ direction: the classifier estimates with accuracy traversable areas.

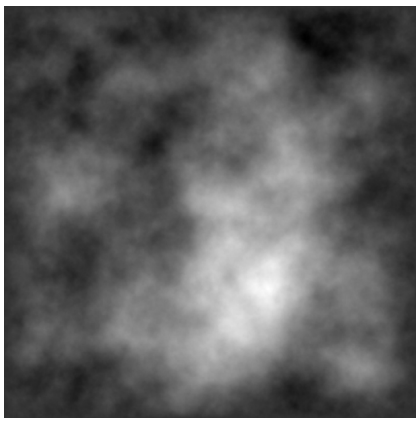

Figure 14. The second tested heightmap that represents outdoor environment

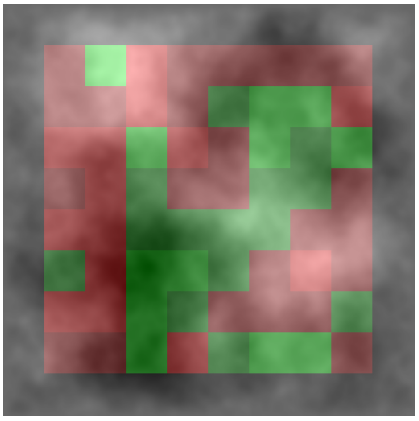

Figure 15. the heightmap traversed in → direction: here the classifier estimates correctly

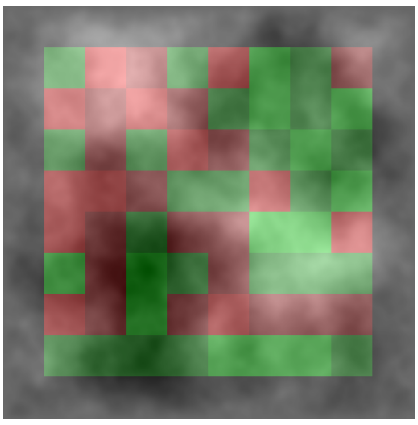

Figure 16. the heightmap traversed in ↘ direction: as in the → case, the classifier does not estimate correctly green points on the diagonal

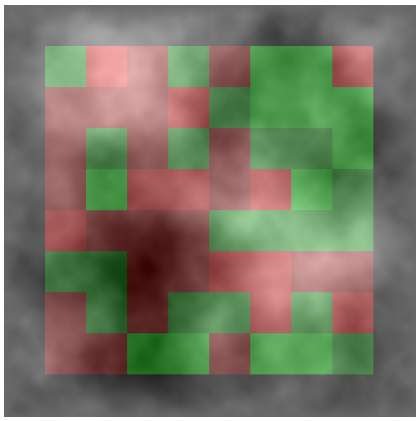

Figure 17. the heightmap traversed in ↓ direction: here the classifier estimates correctly

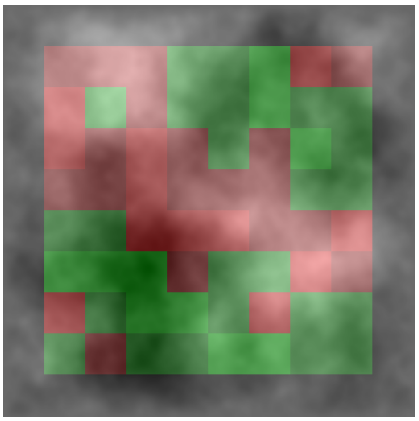

Figure 18. the heightmap traversed in ↙ direction: here the classifier estimates correctly

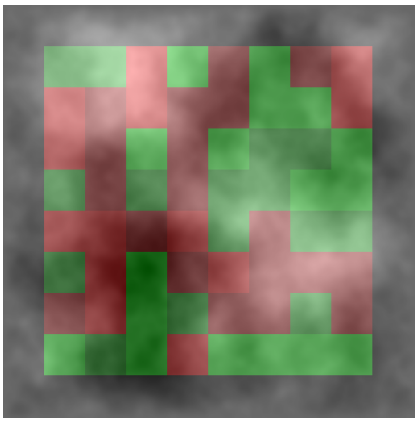

Figure 19. the heightmap traversed in ← direction: as in the → case, the classifier does not estimate correctly green points on the diagonal

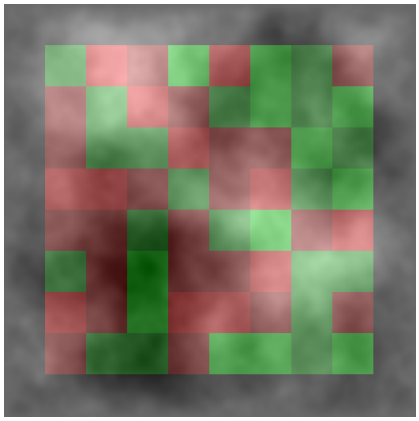

Figure 20. the heightmap traversed in ↖ direction: as in the ↘, the classifier misses 4 center patches

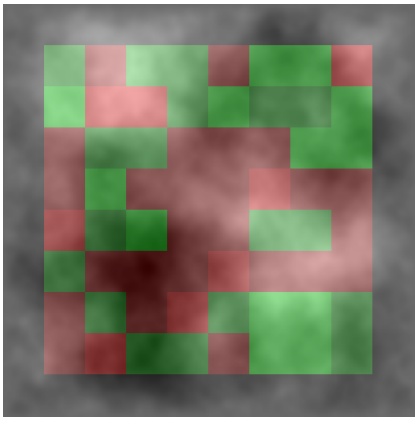

Figure 21. the heightmap traversed in ↑ direction: here the classifier estimates correctly

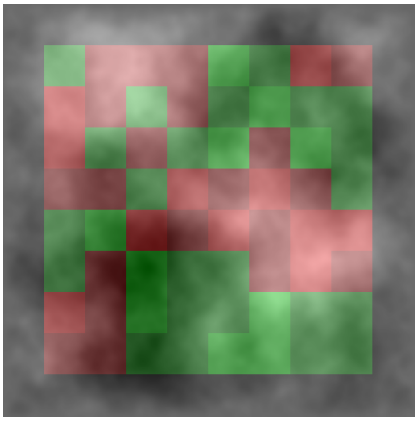

Figure 22. the heightmap traversed in ↗ direction: as in the ↙, the classifier estimates correctly

5 Conclusions and future work

This thesis presents a first approach on path planning with no human intervention. The classifier, even if composed from simple features, predicts with optimal accuracy the traversability of a given terrain, although more improvements in both data collection and learning can be done.
The main challenges in this work were the familiarization with the V-REP environment, (which like the majority of open source software, is constantly updated and does not have a vast community of users), the implementation of the controller for the robot, and understanding the principles of machine learning. Possible future research and work can be summed up in 4 different key parts:

- extend the approach to more complex robot models, (legged robots). This will require our controller to integrate control of foothold stability, in order to perform reliable simulations, and some more accurate analysis on the terrain when training the classifier.
- validate the approach for real robots, meaning doing the actual testing phase in a real outdoor environment, with a real robot, but this may require more constraints, thinking about vegetation, rocks, and other obstacles that can be encountered.
- integrate the classifier's prediction in a path planning system, a task that will also require more study, especially in the decision making phase.
- investigate the possibility to apply the system in a real world environment with the help of stereo-imagery. In other words, using the knowledge obtain by the classifier on a large variety of terrains, together with a time of flight camera for obstacle detection, to perform navigation in rough terrains.

In addition, as suggested by *Alessandro Giusti*, we are currently working in generating two datasets (both on the same terrain): one is obtained by running the robot at a normal speed, while the other one travels at 3x speed. The purpose of this experiment is to analyze how a tune in robot mobility characteristics change the traversability of the same environment.

5.1 Acknowledgments

I want to thank my advisor and professor *Luca Maria Gambardella* and my TAs *Alessandro Giusti* and *Jerome Guzzi* who have given me the opportunity to work on a really interesting and challenging project, and have helped and follow me during this whole semester. Another big thank goes to *Vasileios Triglianos* who has helped me in the correction process of this report.

References

[1] G. De Cubber D. Doroftei H. Sahli Y. Baudoin. Outdoor terrain traversability analysis for robot navigation using a time-of-flight camera.

[2] Dongshin Kim Jie Sun Sang Min Oh James M. Rehg Aaron F. Bobick. Traversability classification using unsupervised on-line visual learning for outdoor robot navigation.

[3] M. Freese E. Rohmer, S. P. N. Singh. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[4] Y X Fu and W H Li. Maximum likelihood estimation of population parameters. *Genetics*, 134(4):1261–1270, 1993.

[5] Travis E. Oliphant. Python for scientific computing, computing in science & engineering, 2007–.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[8] Didier Aubert Raphael Labayrade. In-vehicle obstacles detection and characterization by stereovision.

[9] Mauro Bellone Giulio Reina Nicola Ivan Giannoccaro Luigi Spedicato. 3d traversability awareness for rough terrain mobile robots.

[10] Wikipedia. Inertial measurement unit.

[11] Wikipedia. Principal component analysis.