1 Abstract

With this project, we estimate ground traversability for a no-wheel crocodile-like robot. We collect the robot's pose in a simulated environment using synthetic heig hmap. Then, we for each data point we crop a patch from the height map. We label them and train a deep convolutional neural network to predict their traversability. Later, we evaluate the results by visualizing different datasets and custom patches using model interoperability techniques to highlight and discover the strength and weakness of our network.

Our approach is based on an already existing methodology that we further expanded with a smaller deep convolutional neural network based on residual connection and the squeeze and excitation operator. This network is unaware of the robot's locomotion or its physical characteristics and has to learn to traverse a new environment only through data gather in the simulator.

2 Introduction

Effective identification of traversable terrain is essential to operate mobile robots in every type of environment. Today, there two main different approaches adopted in the industry to correctly plan a robot path: online and offline. The first one uses local sensors to map the surroundings while the second equip the mobile robot with an already labeled map of the terrain.

In most indoor scenarios, specific hardware such as infrared or LIDAR sensors is used to perform online mapping while the robot is exploring, this is the case of the most vacuum cleaner. In the last years, thank to the new application of deep learning in computer vision, more and more cameras have been used in robotics.

Indoor scenarios share similar features across different places shifting the problem from which ground can be traversable to which obstacle must be avoided. For example, the floor is always flat without any bumps or holes due to is artificial design. Usually, traversability must be estimated on the fly due to the high number of possible obstacles and to the layout of the objects in each room may not be persistent in time.

On the other hand, outdoor scenarios may have less artificial obstacle but a homogeneous ground making challenging to estimate where the robot can properly travel. Moreover, a given patch may not be traversable by all direction due to the not uniform features of the terrain. But, a map of the ground can be obtained easily by using third-party services such as google maps or mapped with a flying drone.

These two different scenarios have different challenges. Usually, in control environments such us indoor it is easier to move the robot on the ground but harder to perform obstacle avoidance while in outdoors scenario it is hard to first estimate where the robot is able to move. Furthermore, data gathering may not be straight forward. In indoors terrain, data is collected most of the times by driving the robot in the environment by a human or an artificial controller.

While in the outdoors scenario the data is usually gather using simulations since it is faster.

Our approach aims to estimate traversability on uneven ground, mostly outdoors scenarios. Our frameworks have two main phases, we first run different simulations by spawning the robot on synthetically created maps with different groud configurations, walls, bumps, and slopes. We record the robot position and orientation, then we crop a patch from the height map used in each run such that it includes the robot's footprint and the maximum advancement in a given time window. This value is calculated by observing the advancement of the given robot on flat ground and taking its half. So, each patch contains the current robot position on the map and the future position if the robot will move at the maximum speed. Later, to create the dataset for the classifier, we label each patch using a threshold. Finally, we train a deep convolutional neural network to fit the dataset and we evaluate it using different metrics.

To test the strength of the model we adopt model interpretability techniques such as GRAD-CAM with different inputs to better visualize the learning process.

The report is organized as follow, the next chapter introduces the related work, Chapter 2 describes our approach, Chapter 3 talks in deep about the implementation details, Chapter 4 shows the results and Chapter 5 discuss conclusion and future work.

3 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 [10]. Visual perception is known to be used in most all 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 animals to extract geometric information of the surrounding and plan a safe path through it. Different methodologies have been proposed to collect the data and then learn to correctly navigate the environment. Most of the methodologies rely on supervised learning, where first the data is gathered and then a machine learning algorithm is trained sample to correctly predict the traversability of those samples.

Among the huge numbers of methods proposed, there are two categories based on the input data: geometric and appearance based methods.

Geometric methods aim to detect traversability using geometric properties of surfaces such as distances in space and shapes. Those properties are usually slopes, bumps, and ramps. Since nearly the entire world has been surveyed at 1 m accuracy [8], outdoor robot navigation can benefit from the availability of overhead imagery. For this reason, elevation data has also been used to extract geometric information. Recently, [**1709.05368**], the work we base on, proposed a full pipeline to estimate traversability using only elevation data in the form of height maps.

Elevation data can also be estimated by flying drones. [4] 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 utilize an 'on-the-spot training' using a convolutional neural network to segment the terrain in different traversable classes.

Whereas 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. Those images can be used to directly estimate the traversability cost.

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 [10] or Gaussian models [8]. Those techniques rely on texture descriptors, for example, Local Binary Pattern [9], 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.

One recent example is [1] 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 also be used together to train a traversability classifier. [3] proposes 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.

Data can also extract in simulations, where 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, [5] proposed a locomotion planning that is learned in a simulation environment.

We can also distinguish between different types of robots: wheel and no-wheel. Since previous work [2] has been focused on wheel robot, we are now interested in no-wheel robots and in particular in a legged robot.

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. [**wermelinger2016navigation**] uses 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. [6] 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 those 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. [**wellhausen2019where**] use local sensing to train a deep convolutional neural network to predict 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

Add mirko's paper

4 Model

This section gives a high-level overview of the steps in our approach. We will first describe the data gathering, then the post-processing pipeline used to generate the dataset. After, we will introduce the model used to fit the data and then show the results with different test environments.

4.1 Robot: Krock

Our task was to apply the approach proposed in [**omar@traversability**] with a legged robot developed at EPFL named *Krock*. Figure **??** shows the robot in the simulated environment.

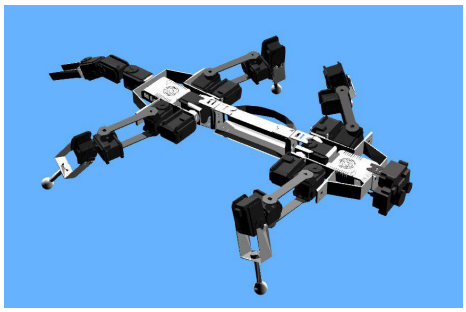


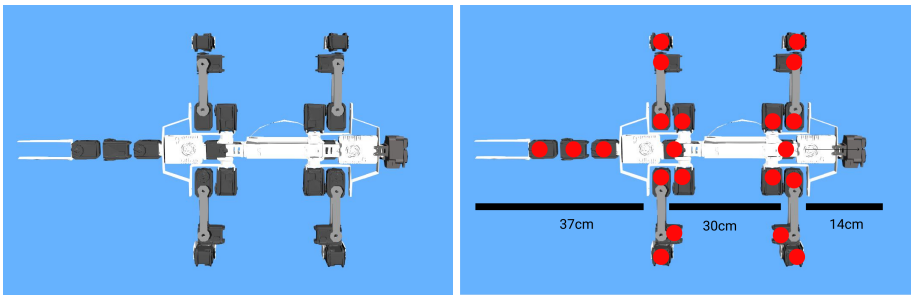
Figure 1: *Krock*

Krock has four legs, each one of them is equipped with three motors in order to rotate in each axis. The robot is also able to raise itself in three different

??

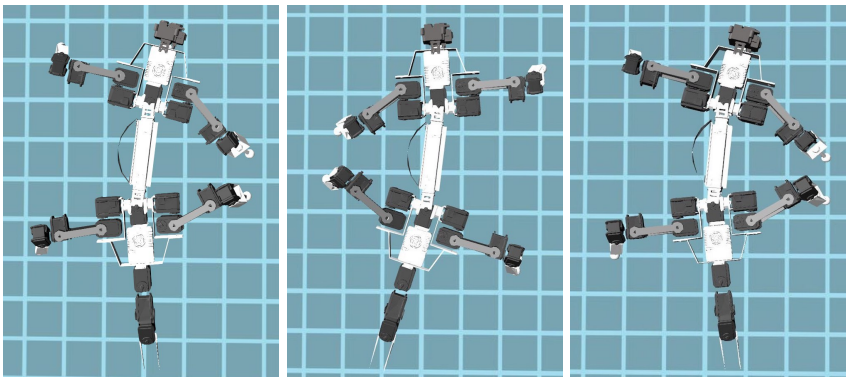
Figure 4: *Krock* differents gait configuration.

configurations, gaits, using the four motors on the body connected to the legs. In addition, there are another set of two motors in the inner body part to increase *Krock*'s move set. 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 and weights around $1.4kg$. The next figure **??** shows *Krock* from the top helping the reader understanding its composition and the correct ratio between its parts. Also, each motor is highlighted with a red marker.



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

Krock's moves by lifting and moving forward one leg after the other. The following figure shows the robots going forward.



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

Figure 3: *Krock* moving forward.

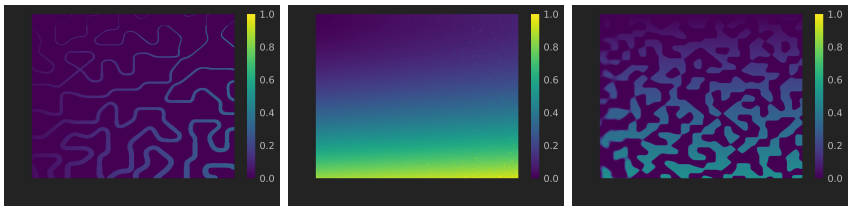
Krock can also raise its own body in three different configurations. Figure

Add Krock gait picture

4.2 Simulation

Our approach relies on simulations to collect the data needed to train the estimator. We used Webots [7], a professional mobile robot simulator, as the backbone for our framework.

We generate fifteen synthetic $10x10$ meters long heightmaps with different features, such as walls, bumps and slopes using the same techniques described in [2]. A heightmap is a gray image, formally a 2D array, where each element, pixel, represents a height value. Since each image has a range from $[0, 255]$, we also need to scale them when needed. For this reason, we associated a height scaling value for each height map that is used to increase or decrease the steepness. The next figure shows some of the maps used.



(a) *bars1* (b) *slope rocks 2* (c) *steps 1*

Figure 5: Some of the heightmaps used in the simulation.

The following figure shows the 3d rendered version of the bars heightmap.

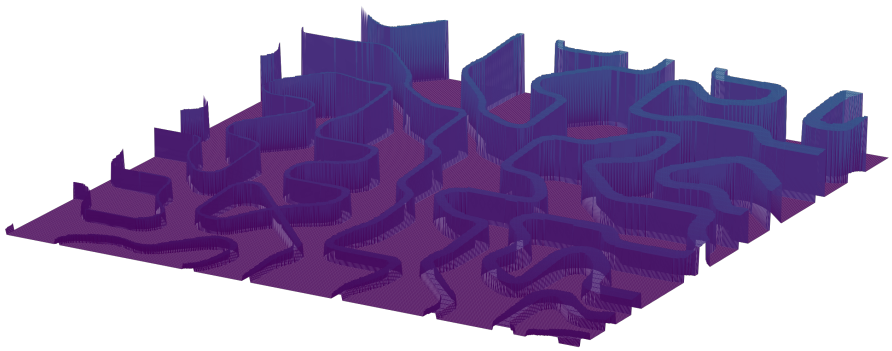


Figure 6: *Bars1* 3d map

To generate the train data, each map is loaded into the simulator, then the robot is spawned in the world and we let it walked forward for a certain amount of time or until it reaches the edge of the map. While moving, we store his pose,

that contains position and orientation, with a rate of $50hz$.

We fix the robot's gait in its standard configuration and run fifteen simulations for each map, where one simulation one robot spawns and death, with a height scale factor of one. We also decide to include steepest training samples by using one of the maps with a slope with heights factors from $[1, 7]$. The following pictures show this specific map with different scaling factors.

(a) height = 1 (b) height = 2

Figure 7: *Slope rocks 2* map with two different height scaling factor.

Krock is equipped with a controller implemented with the Robotic Operating System (ROS) software that is used as a communication channel between our framework and the simulator. Basically, ROS exposes nodes, called *topics*, in which we can listen for incoming messages. Thus, *Krock* is able to stream its pose to all the clients connected to its pose topic.

We also generate the test set, by running the robot in real-world environment, such as *Quarry* a cave map, that we later use to test the fitted model.

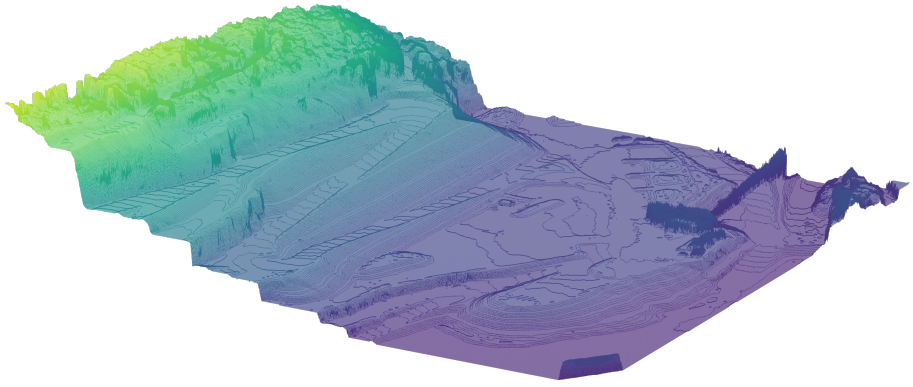


Figure 8: *Quarry* map.

By observing the simulations we notice two problems. First, *Krock*'s tail

sometime gets under the body, second, if a random spawn strategy is used, in certain maps the agent may directly fell on an obstacle.

To overcome the first problem, we decide to remove the tail after we ensure by asking the developers that this will not compromise the traversability. We immediately observe less noisy data.

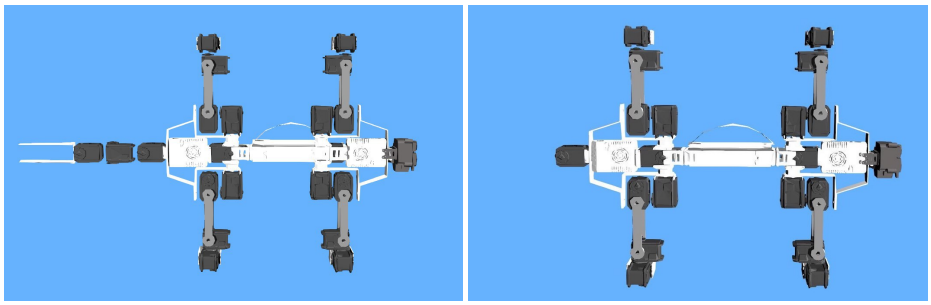


Figure 9: *Krock* with and without tail

The second issue was more challenging. If the robot spawns on an obstacle then it will be stuck for all the simulation compromising the quality of the data. This is very important for the *bars* map, where there are lots of wall of different sizes and the probability to fall on one of them is very high. So, for some maps, we guarantee that *Krock* does not directly spawn into obstacles by only spawning the robot in the flat ground first. The following figures show the result of this strategy on the *bars1* map where the robot was spawned where there is no obstacle. The following image shows different *Krock*'s spawn position in the simulator on the *bars1* map, the reader can notice that the robot is always placed where there are no obstacles under it.

Figure 10: *Krock*'s spawns on the *bars1* map.

We run a total of storing the results using ROS's *.bag* files for a total of .

add number of simulations

add size of all bags.

5 Postprocessing

In order to create the training dataset, we need to post process the data stored during the simulation. To train our classifier we need to first compute the advancement for each pose at each time for a specific time window, $pose_{it}$. Then, we can use it to crop a patch from the heightmap that was used in the simulation of the correct size. Such patch must include the whole *Krock* footprint and the maximum possible advancement in a given time window. By keeping this final goal in mind, we quickly describe each step taken to reach it. Figure 11 shows each step applied in the pipeline.

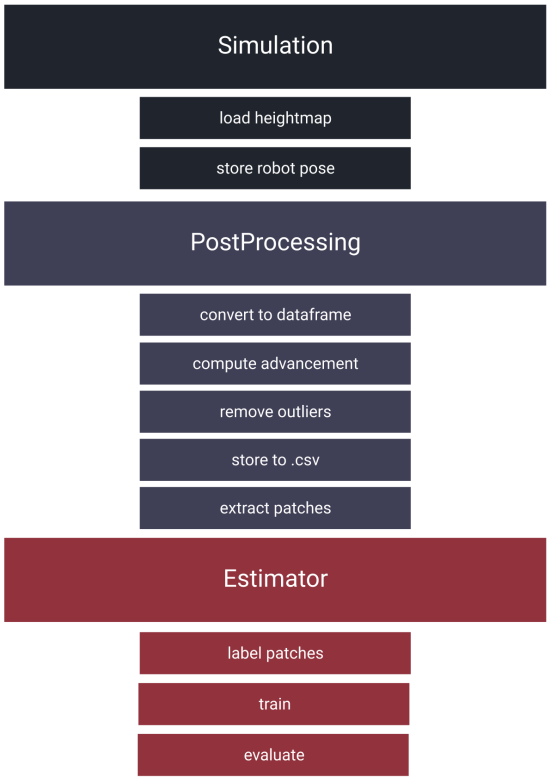


Figure 11: Pipeline

We implemented an easy to use API to define a cascade stream of function that is applied one after the other using a multi-thread queue to speed-up the process. First, we convert each *.bag* file into a *.csv* file and store it since this operation must be done only once. Then, we define a time window t and for each entry in one simulation run we compute the future advancement using booth position and angle from the pose, we also store the resulting data frame into a *.csv* file. During the process, we also clean the data by removing the samples in which the robot was upside down or out of the map.

link to the project

Finally, we crop from each heightmap the patches corresponding to each data point. To compute the patch size need to know the maximum possible advancement of *Krock* for a specific time window. In order to find it, we run *Krock* on a flat ground map and we take the mean across all of them. We observer a mean advancement of $33cm$ per second.

For a given advancement, we ensure the whole *Krock* footprint is included on the left part to take into account the situations in which an obstacle is directly under the robot. To clearly show the process, we used a simulated environment in which *krock* has a big wall in front of him and one small under his legs. The following picture shows the terrain.

add figure with the two walls

Clearly, assuming *Krock* is able to traverse 33cm of flat ground per second, in one second it won't be able to reach the obstacle, this is shown in the first picture. Increasing the time window, the taller wall appears while the small one under the robot is always correctly included.

add figure with the patch computation

6 Estimator

To classify the patches we need a class associated to each one of them, thus we must label each patch as *traversable* or *not traversable*. So, we selected a threshold, tr, and labeled each patch by $patch_{advancement} > tr$. In other words, if *Krock* was able to advance in a patch more than the threshold, then that patch is label as *traversable* and vice-versa. The pair of patches and labels represent the final dataset.

add an image with a trivial traversable and not traversable patch

We used a deep convolutional neural network to fit the training set. Convolutional Neural Network is able to map images into a features space and then to target class. Before feeding the patches into the model, we subtract in each patch the value in the middle in order to normalize them by making height invariant. Visually,

add figure with a patch not centered and centered

We evaluate different models with different hyperparameters to select the best one using the metrics score on the test set as guidelines. We then visually evaluate the network by running it on the *Quarry* map for different orientations. In order to do so, we extract patches form the ground using a sliding window and feed them to the model. Then, we create a texture where each pixel represents the traversability probability, the brighter the higher.

References

[1] Jérôme Guzzi Alessandro Giusti et al. "A Machine Learning Approach to Visual Perception of Forest Trails". In: (2016).

[2] R. Omar Chavez-Garcia et al. "Learning Ground Traversability from Simulations". In: (2017). DOI: 10.1109/LRA.2018.2801794. eprint: arXiv:1709.05368.

[3] Jeffrey Delmerico et al. "On-the-spot Training" for Terrain Classification in Autonomous Air-Ground Collaborative Teams". In: (2017).

[4] Julia Nitsch Jeffrey Delmerico Elias Mueggler and Davide Scaramuzza. "Active Autonomous Aerial Exploration for Ground Robot Path Planning". In: (2016).

[5] Tobias Klamt and Sven Behnke. “Anytime Hybrid Driving-Stepping Locomotion Planning”. In: (2017).

[6] M. Bloesch L. Wagner P. Fankhauser and M. Hutter. “Foot Contact Estimation for Legged Robots in Rough Terrain.” In: (2016).

[7] Olivier Michel. “Cyberbotics Ltd. WebotsTM: Professional Mobile Robot Simulation.” In: ().

[8] Boris Sofman et al. “Improving Robot Navigation Through Self-Supervised Online Learning”. In: (2006).

[9] Matti Pietikainen Timo Ojala and Topi Maenpaa. “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.” In: (2002).

[10] E. Ugur and E. Sahin. “Traversability: A case study for learning and perceiving affordances in robots”. In: (2010).