ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RSL Robotic Systems Lab
Prof. Dr. Marco Hutter

Semester Thesis

Deep Learning of Footholds for Rough Terrain Locomotion

Spring Term 2016

Supervised by:
Péter Fankhauser
Jemin Hwangbo
Alessandro Giusti

Author:
Wolf Vollprecht

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Deep Learning of Footholds for Rough Terrain Locomotion

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Wolf	Vollprecht

Student supervisor(s)

Péter	Fankhauser
Jemin	Hwangbo

Supervising lecturer

Marco	Hutter

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Preface

I want to express my gratitude to the ASL, the RSL and the IDSIA for the opportunity to do this tremendously interesting project together. I especially want to thank my supervisors Péter Fankauser, Jemin Hwangbo and Alessandro Giusti for their support and guidance.
My friends are a big part of my academic journey and in this sense I want to thank Isabel Schwarz, Dominic Witt, Georg Wiedebach, Maurice von Oppersdorff and Simon Scheidegger.
Without my family I would not be in the same place I am today. Thanks for always supporting me in my adventures.

Abstract

Legged robots, and specifically quadrupeds, are a highly interesting research topic since they can conquer more challenging terrain than traditional wheeled robots. One major bottleneck for climbing performance is robust foothold selection. In this semester thesis, a method to score footholds for rough terrain locomotion with a deep learning approach is reported. The required amount of data is generated by running a simulation of the quadruped walking on artificially generated terrain. We then formulate the foothold scoring as regression problem on the simulated slip distance and use the geometric heightmap information as feature. No human expert intervention is required along the learning process.

Chapter 1

Introduction

As legged robots get more advanced, the need for conclusive terrain analysis becomes more important. While intelligent control mechanisms can help the robot recover after slipping, geometric and optical inspection of the underlying terrain can prevent the slipping altogether by selecting footholds that maximize the stability during the different gait phases.
This thesis approaches terrain scoring for legged robots from a deep learning perspective. One of the challenges with deep learning is to come up with the required amount of feasible training data. This is answered by using a simulation environment that models an arbitrary three-dimensional terrain and the StarlETH robot. Within the limits of the simulation, which are governed by numerical approximations to the differential equations of the simulated physical environment, the resolution of the recording can be arbitrarily accurate and there is no need to obtain further ground truth of the robot state, which would be a major problem in real world experiments. Generating data by simulation also enables the automation of tasks like resetting the robot to gather new measurements.
The result of the simulation is a dataset which contains the heightmap around the foot and the foot position during contact with the terrain. From the foot position the target label is computed, which is chosen to be the distance from the initial contact point to the lift-off point.
The heightmap and the computed label is then used to train two regressors: A baseline support vector machine regression with a very limited set of features (consisting of the standard deviation of the heightmap in different radii around the foot and the maximum height deviation in the heightmap at the footpoint), and a convolutional neural network, a neural network architecture that learns a set of convolutional filters which are applied to extract meaningful information from the underlying (image) data to compute the regression.
Both models are then compared and the results of the regression are shown in the concluding chapter.

1.1 System Overview

StarlETH is a highly compliant quadruped robot developed at the Autonomous Systems Lab at ETH Zurich [1]. The robot has approximately the size and weight of a medium sized dog (it is about 0.6 m tall and weighs 23 kg) and is completely autonomous. The four identical legs are arranged in the X configuration and torque controlled. Each of the legs has 3 degrees of freedom (2 in the hip and 1 in the knee). Inertia of the feet is kept low by driving the joints with chain and cable pulleys and keeping the motors inside the main body. The key element of the design of the

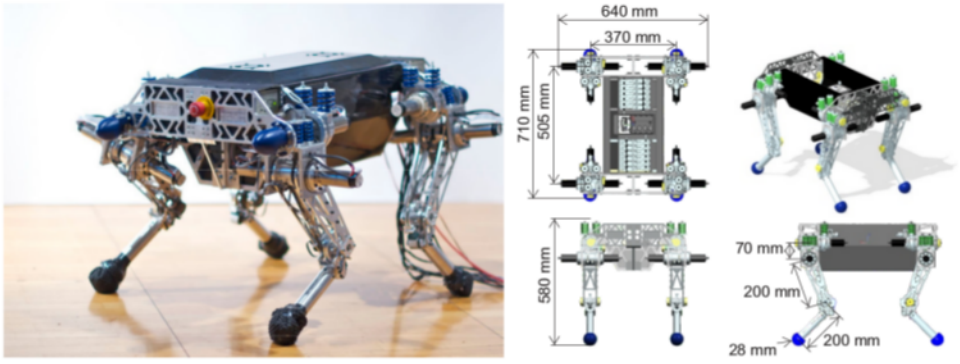


Figure 1.1: System Overview of StarlETH (image taken from [1])

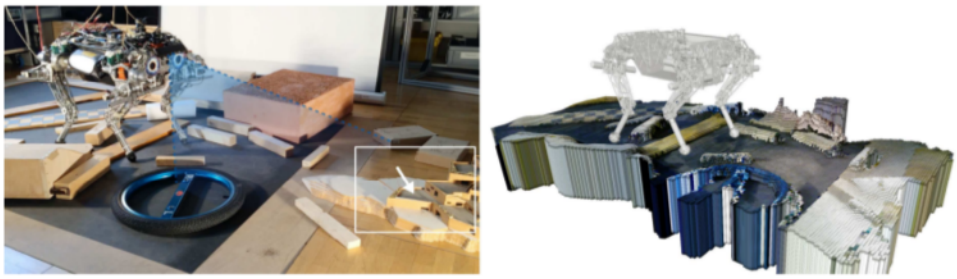


Figure 1.2: StarlETH with mounted Kinect distance sensor walks over random terrain. The terrain estimate is shown on the right

StarlETH robot are the high compliant series elastic actuators. The motors are decoupled from the joints by springs, which allows for robustness against impacts and short-term energy storage. Joint angles, motor angles, and spring deflections are continuously measured in every joint. The foot is an air filled raquet ball with an diameter of about 5 cm. Each foot also contains a force sensor that measures the contact force on the foot.
Robot centric elevation mapping has been implemented on StarlETH, which continually updates a 2.5D heightmap based on measurements from distance sensors and the motion estimate of the robot. A key feature of the elevation mapping is that it maintains an uncertainty estimation which is low in front of the robot where the measurements are being taken and relatively high in the areas previously measured. On StarlETH, the elevation mapping can be configured for different needs and with different sensors. For example, it can be used with a forward facing Microsoft Kinect for high resolution, low range elevation mapping or with a LIDAR for long range but low resolution mapping [2].

1.2 Related Work

Robot navigation planning has a long history. Many authors have been concerned with traversibility planning and local or global path finding through difficult to traverse terrain. Legged robots however propose a different challenge. While wheeled robots have several obvious limitations in regards to obstacles, legged robots can climb over obstacles within the kinematic reach of their legs. Climbing does require accurate foot positioning on the terrain and correct estimates about the foothold robustness during stance- and gait phases of the walk, since the legged robot otherwise

Figure 1.3: The SmallDog robot by Boston Dynamics (image taken from [3]).

runs into the risk of slipping and falling over.
Some of the early approaches for terrain scoring were developed for the SmallDog challenge in 2007, which was an United States DARPA project. Multiple teams competed to climb over extremely challenging terrain with a quadruped robot the size of a small dog. Most teams developed a controller consisting of a high-level planner and a low-level executing controller. All teams were given access to a high precision 3D scan of the terrain to be conquered.
The team around Kolter [4] first generate collision maps for a default stance from the 3D mesh (to prevent e.g. the knee from colliding with the terrain), and then further score the terrain by extracting and weighing features from the heightmap. The features they use are the standard deviation of height, average slope in x/y direction and maximum and minimum height relative to the center point of the patch. These 5 features are calculated for 4 patches around a center point with 5, 7, 11, and 21 pixels each. Combined with a collision indicator for each pixel, this results in a set of 22 features (they also include a constant offset to be able to shift the linear). The weights for the scorer are obtained by Hierarchical Apprenticeship Learning, where a human demonstrator scores a few key samples and the system determines corresponding weights. They report much better results using this technique than hand tuning the weight vector.
Another Little Dog competitor, the IHMC with the team around Peter Neuhaus and Jerry Pratt [3], used a combination of three different terrain scorers for their system: A scorer for the surface normal, for the local height difference and a Vee terrain scorer which scores depending on cliffs in front or behind the leg to avoid swinging into an obstacle. They mention that machine learning techniques were tried but hand-tuned solutions were deemed sufficient and engineering speed was more important towards the end of the competition.
The LittleDog team around J. Buchli and M. Kalakrishnan [5] used human expert scores and template matching to estimate foothold robustness on the 3D terrain map. The expert is shown logs of the robot running on difficult terrain and at each suboptimal foothold chooses the reachable foothold that he deemes optimal. For each score, templates on several different scales are extracted. These templates can be matched against a terrain in the evaluation to find the score of the foothold.
All three approaches have in common that human experts are needed to derive the terrain score, making it a supervised learning problem. However, in this thesis no human expert is required to identify and learn the geometric characteristics of good and bad footholds and no hand-tuning of weight vectors is used.
For the StarlETH robot, M. Hoepflinger [6] implemented an approach for unsupervised terrain scoring based on experimental results from real terrain-foot-interactions. The experimental setup consisted of a single foot applying different

amounts of force on an artificial terrain and the amount of slip was measured. Geometric features and the force distribution were then used to cluster the results by robustness, employing the k-means clustering algorithm. Since the experimental setup requires a heavy investment in terms of time required to collect the data, the final amount of experimentally collected data by M. Hoepflinger was rather small, making it unsuitable for a deep learning approach.
Belter (2009) [7] uses an approach very similar to the one in this thesis, where a multi-legged robot is simulated to learn a decision surface from experiments. In similar spirit, no human expert is needed. However, Belter et. al do not use deep learning techniques but rather try to estimate a polynomial decision surface spanned by terrain describing coefficients. The decision surface approximating function is found by a population based evolutionary algorithm.
In the field of deep learning the effectiveness of convolutional neural networks for image classification has been established and they have been proven to even outperform humans on tasks like traffic sign recognition (D. Ciresan, J. Schmidhuber 2012) [8]. An overview of recent deep learning advances is provided by J. Schmidhuber (2015) [9]. More recently, a deep learning approach with convolutional neural networks has been used to classify terrain by sound samples from a wheeled robot (A. Valada et. al, 2015) [10].

Chapter 2

Learning of Stable Footholds

In order to use machine learning techniques to create trained models that can judge the stability of footholds, a dataset has to be obtained. Since obtaining a big dataset of footholds with StarlETH in the real world requires a lot of time and preparation, a simulation environment was chosen that models the physics and the robot-terrain interactions to perform the collection of the necessary data. At the end of this chapter, the machine learning algorithms used in this semester thesis are briefly described.

2.1 Simulation

The basic idea for the data collection in simulation is to have the robot running in the simulation until some sort of break condition occurs (which could be that the robot cannot move further due to falling or because it reaches the edge of the world) and record the foot positions and the heightmap during the simulation as data. Since a bad foothold can be characterized by the amount of slippage that occurs during the stance phase of the foot, a foothold score can be calculated from our exact knowledge of the recorded foot positions (compare Figure 2.1).
Some requirements have to be made towards the simulation. Of course, it should model the real world dynamics as accurately as possible. Different terrains have to be loaded into the simulation and the terrain files should have a reasonable resolution as well as a size that allows the robot to traverse the terrain for some time. All sensors should be accurately modeled and it should be possible to read the joint positions, and especially foot positions, accurately at a high frequency.

2.1.1 Terrain Generation

For the simulation of the walking behavior on rough terrain, the specific terrain had to be generated. This could be done manually by creating rough and challenging

Figure 2.1: Sketch of the score: distance between contact and lift-off point

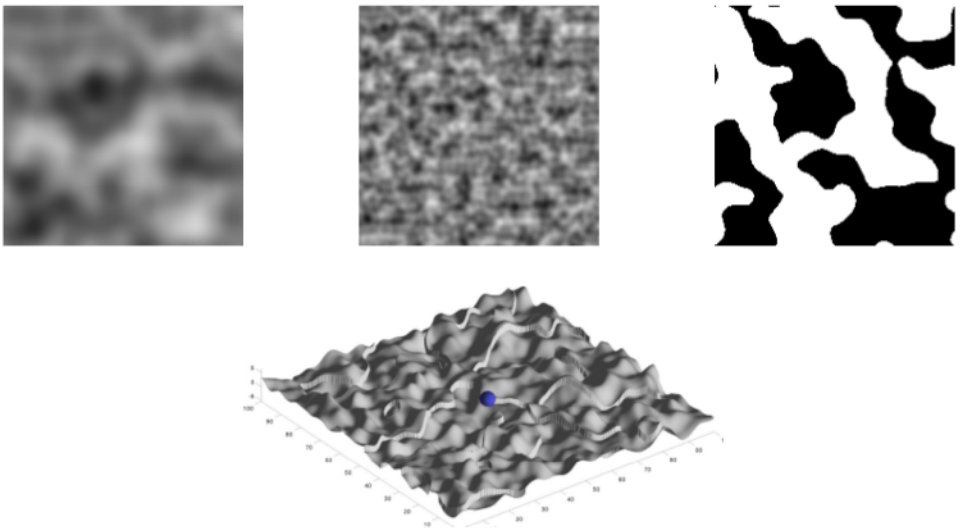


Figure 2.2: Generated heightmap with high and low frequency noise and added step

terrain with a 3D-brush. However, we chose a parameterized solution using random noise and the gaussian blur function in order to create heightmaps which are then extruded to three dimensional terrain. Since only using one noise and gaussian blur combination produces terrain structures with a very smooth gradient, several such images are combined.
The final image thus consists of an addition of two images with noise and gaussian blurs of different radii (ie. a low frequency and a high frequency source of elevation) and a third, similarly constructed but acting as a threshold (ie. adding a constant step if the gaussian blurred noise is larger than a threshold and else it stays at 0).
The maximal height deviation of the terrain was chosen to be 6 centimeters. This is a height that StarlETH can still traverse but at the same time maximizes slipping, which is desirable since bad foothold examples are needed to learn a meaningful score.
The resolution of the heightmap was chosen to be 1 pixel per centimeter, which is reasonable in terms of achievable performance with the elevation mapping and contains enough data in relation to the foot size.

2.1.2 Data Recording

During the simulation as much data as possible should be stored to be evaluated at a later stage (since the simulation is the main bottleneck in terms of consumption of time).

2.2 Data Postprocessing

2.2.1 Filtering

Since the legs of the robot are connected, but footholds are looked at as single entities, filtering the data is necessary. If one foot slips, for example, the entire system goes into an "unstable" state where the other feet also have to react to the slipped foot.
Several filtering mechanisms should therefore be implemented:

- Filtering the first and last 4 footholds. The first four footholds are discarded, because the robot is "dropped" in gazebo, and bounces back a little. This makes the beginning of a run sometimes unstable and random (during dropping the controller is not running as well, and therefore the first steps might only adjust the main frame to a horizontal position). The last 4 footholds are discarded because often the robot falls for some reason, but during "tripping" it sometimes touches down for extremely short durations or in weird positions and therefore this data is rather discarded.
- Too short runs (with fewer than 10 entries) are also filtered because it can be assumed that the robot never found into a stable walking gait.
- After slipping is detected in one foot (because the deviation of the foot position over time is bigger than a specific threshold) the following four footholds after that are filtered. It can be assumed that the robot is in a less stable position after slipping with one foot and the first slipping foot influences the behavior of the others. In this thesis the focus lies on the geometric causality and therefore these data points are discarded.
- Filtering false records and short touch-downs. Sometimes the dataset might contain footholds where the duration of the contact with the ground is extremely short. This can be either due to bugs or because the foot touches the ground only for a very short time but never really applies any force on the foot.

2.2.2 Heightmap Normalization

To effectively learn from the data, normalization should be applied. In this thesis, two normalizations are applied to the raw heightmap data.
The heightmap is rotated into the direction of the robots yaw angle. This ensures that all features of the heightmap are always perceived in the same way, in the direction of the robot, and also adds the walking direction as an implicit parameter to the dataset.
The heightmap is also normalized around the center pixel, such that the center pixel (where the foot touches the heightmap) is always at 0.5 (or a medium gray in terms of the image colors). This ensures that all deviations are projected in terms of where the foot lands at the beginning of the stance phase and not in terms of the absolute height.

2.2.3 Dataset Balancing

While it is generally advisable to have datasets as large as possible, in this case it was found out that reducing the amount of foothold samples with little slip was favorable for the results of the regression. Many footholds with little slip influence the machine learning algorithm to put more weight on footholds with little slip and judge footholds that could be good or bad so that the error will be minimized (ie. as good footholds since there are many more examples).
However, it is desirable to have a foothold classifier that reliably discards bad footholds and therefore a rebalancing step is introduced where the amount of footholds with little slip and those with heavy slip is equalized by removing samples with little slip.

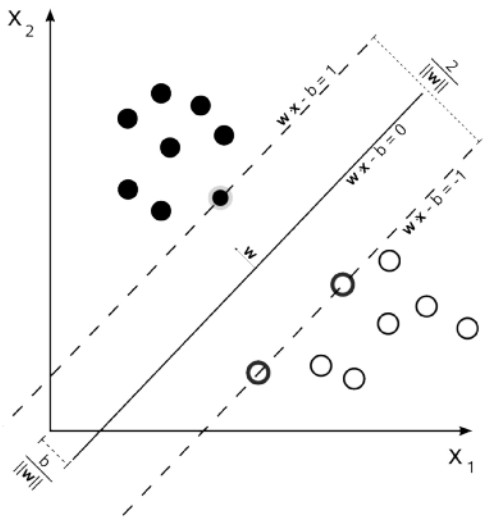


Figure 2.3: The SVM as linear classifier. Image source: Wikipedia/Cyc

2.3 Learning Algorithms

For the learning part of the semester thesis, two different learning algorithms were implemented and tested on the dataset.
A Support Vector Machine (SVM) regression with a very limited set of features was used as a baseline to compare against and derive conclusions about the dataset feasibility for learning purposes. For the deep learning part, a convolutional neural network was used.

2.4 Support Vector Machines

Support Vector Machines (SVM) in general are used as classifier with a linear decision boundary. The SVM algorithm minimizes the error of the hyperplane that seperates the classes. The hyperplane can be expressed as $w \cdot x - b = 0$, where w is the trained weight vector and x are the features of the dataset. b is a bias to shift the hyperplane to any point on the x axis.
In a two-way classification problem, all labels are $y_i \in \{-1, 1\}$. Therefore the margins of the hyperplane can be expressed as $w * x - b = 1$ and $w * x - b = -1$. It is easy to see that the margin width is $2/||w||_2$. Following from this is that minimizing the weights will maximize the margin (compare Figure 2.3).
The resulting minimization problem is:

$$\text{Minimize } \|w\|_2 \text{ subject to } y_i(w \cdot x - b) \geq 1$$

In case a linear classification boundary is not feasible the "kernel trick" can be used which allows the features to be projected according to the function specified by the chosen kernel. A common choice is the radial basis function (RBF) kernel. The resulting decision boundary is non-linear.

2.4.1 SVM Regression

ε - SVM regression (also shortened to SVR) was introduced by Vapnik in 1996 [11].
The ε - regression can be written as

$$\begin{aligned} &\text{Minimize} && \frac{1}{2}||w||_2^2 \\ &\text{Subject to} && \begin{cases} y_i - w \cdot x_i - b \leq \varepsilon \\ w \cdot x_i + b - y_i \leq \varepsilon \end{cases} \end{aligned}$$

Where y_i is a true label, and x_i is a specific vector weight. The model only cares about the data that is outside the ε margin and discards training data where the true label is close to the prediction (compare Figure 2.4). This can be interpreted as using the ε boundary around the hyperplane to separate the remaining samples into $\{-1, 1\}$ as in the 2-way classification problem and discarding those samples that cannot be clearly classified since they are close to the regression target.

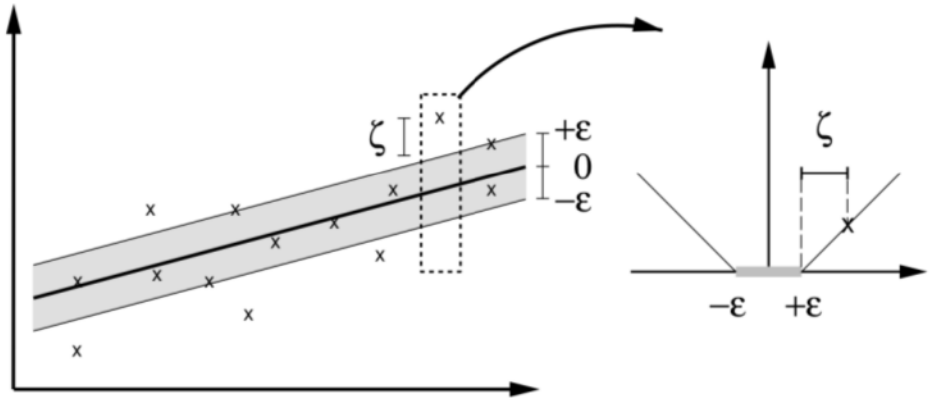


Figure 2.4: Visualizing the error for SVR and the epsilon-tube around the predicted function. Only samples outside the epsilon-tube contribute to the error. Image taken from [12].

2.5 Artificial Neural Networks

Artificial Neural Networks are a machine learning tool that is inspired by the functionality of the human brain. An input x is transformed through one or several hidden layers of the network into an output and the error is propagated back through the network to improve the network response in a supervised learning fashion.
One of the key properties of neural networks is that they can approximate any nonlinear function as response to a given input, where the possible complexity of the approximated function is dependent on the number of neurons in the network.
A very simple neural network with only one hidden layer is shown in Figure 2.5.
The neurons in the hidden and output layers are connected to neurons of the previous layer. Every connection has a specific weight, represented by the weight matrix w. The weights are "learned" in the process of training a neural network.
The equation to calculate the output of a fully connected layer in a neural network is:

$$o_a^{(i)} = \phi(\sum_k w_{ka} * o_k^{(i-1)})$$

Where w_{ka} are the neuron weights from neuron k of the previous layer to neuron a and $o_k^{(i-1)}$ is the output of neuron k in the previous layer (ie. the argument

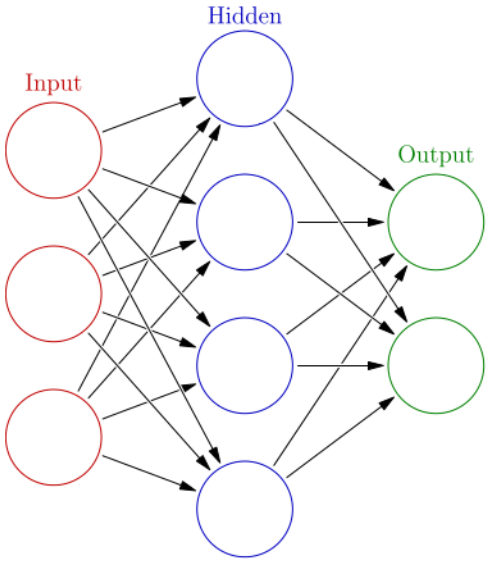


Figure 2.5: A very simple neural network with 3 input neurons, one hidden layer and 2 output neurons. Image source: Wikipedia/Glosser.ca

for the activation function is a weighted sum of all previous activations). ϕ is the activation function of the neurons. For backpropagation learning it is important that the activation is either differentiable or a subgradient exists. While the sigmoid function has been the classic activation function of the past that might correspond best to human neuron activations, the Rectifing Linear Unit (ReLU) activation function is commonly used nowadays for speed and accuracy.

$$\phi_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

$$\phi_{\text{ReLU}}(x) = \max(0, x) \tag{2.2}$$

The neural network can contain different layer architectures, such as convolutional layers (explained below).
The network weights are commonly trained by backpropagation, where the error in the output layer is propagated back to the input layer. By calculating the gradient of the error with respect to a particular weight, an equation for the necessary change to the weight can be obtained. The general equation for the particular derivative is given as:

$$\frac{\partial E}{\partial w_{ij}}$$

where E is an error function such as the squared error function for a single neuron output $E = 0.5 * (o_{output} - y)^2$.
The update rule is given as

$$\Delta w_{ij} = -\alpha * \frac{\partial E}{\partial w_{ij}}$$

What is referred to as "Deep Neural Network" is a neural network architecture that consists of many (deep) hidden layers.

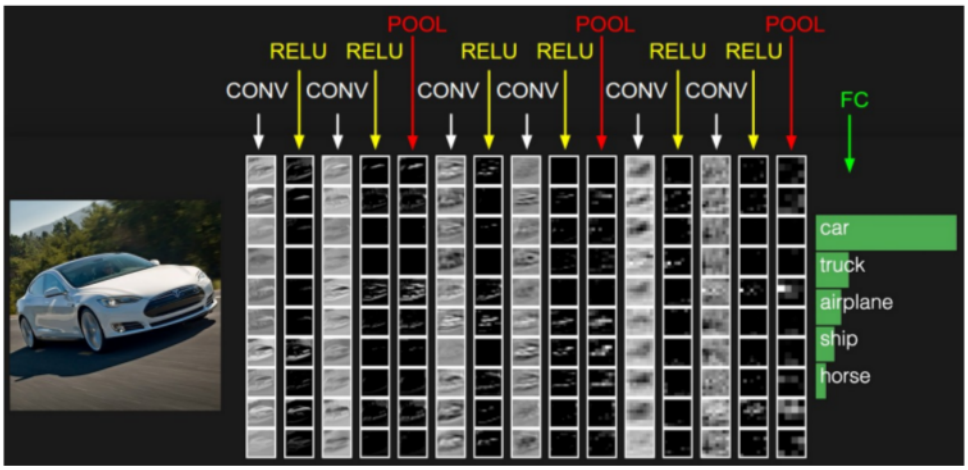


Figure 2.6: Demonstration of a convolutional network for image classification. The architecture consists of stacked convolutional layers with ReLu and Pool layers. In the convolutional layers the filters are applied. The pool layers break down the image to fewer and fewer pixels, until a final activation pattern is observed. (taken from `https://cs231n.github.io/convolutional-networks/`)

2.5.1 Convolutional Neural Networks

This explanation of convolutional layers is unclear

Not sure what "sample size" means here

Important: "reduced" w.r.t. a fully-connected layer with the same number of I/O neurons

Each convolutional layer in a neural network trains a set of filters that are repeated and overlaid on the input (in case of the first layer, the original image). The network output corresponds to the activation of the filter. The filter has a specific size (e.g. 3x3 pixels) which is the sample size and corresponds to the maximum area of the extracted feature. By training convolutional filters the amount of connections is drastically reduced, which makes it more feasible to train on large images. The output of a convolutional layer can be thought of as a three-dimensional volume, where the outputs of the different filters are arranged along the z-axis.
In a standard convolutional neural network, a so called max pooling layer is sometimes added after a convolutional layer. The max pooling layer also has a filter size and selects the outputs inside its mask with the biggest amplitude. This results in an output feature map that is smaller than the previous (e.g. a max-pool layer with filter size 2x2 and a stride of 2 effectively halves the input image by 2 in width and height).
Dropout layers are a type of layer that randomly “drop” a connection with a certain probability during training. Dropout layers have been shown to reduce the amount of overfitting that can otherwise happen in CNNs.

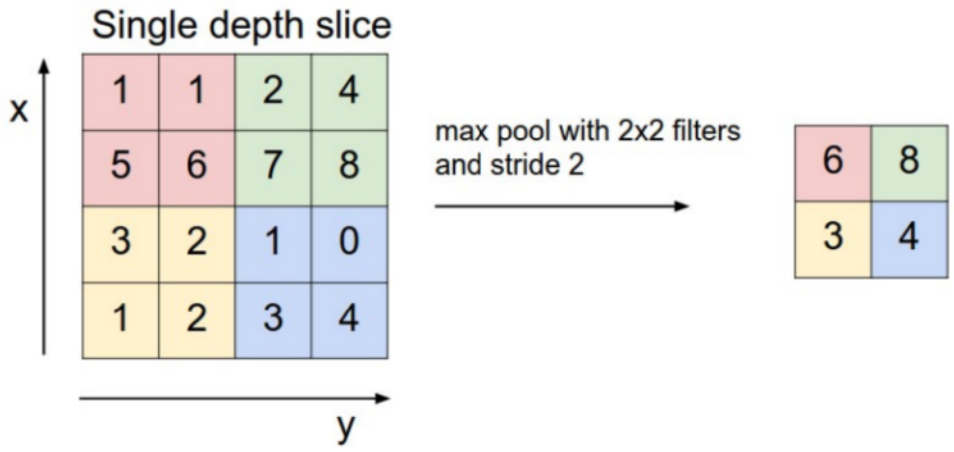


Figure 2.7: The max-pooling operation with operator size of 2 and stride of 2 (taken from `https://cs231n.github.io/convolutional-networks/`)

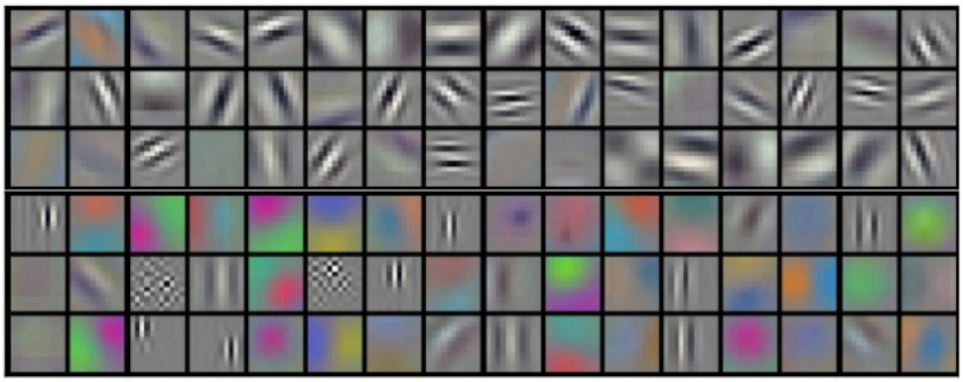


Figure 2.8: Learned convolutional filters from a paper by Krizhevsky et al. It is easy to see that some of the filters "react" to edges (taken from `https://cs231n.github.io/convolutional-networks/`)

Chapter 3

Implementation

In this chapter, the implementation of the previously mentioned components is described. The hardware-software interface of the StarlETH robot is programmed using the Robot Operating System (ROS) as middleware. All of the hardware, including the sensors, are modeled in the Gazebo simulation framework. `Rospy` offers a convenient way to implement ROS nodes in python and was used in this semester thesis.

3.1 Simulation and Data Collection

As a simulation framework for StarlETH was already implemented in the `gazebo` simulation environment and conformed to the requirements stated in the previous chapter, it was a natural choice to utilize it. The concrete implementation that binds into the simulation consists of two ROS nodes, the robot supervising node and the recorder node. A third process, the simulation supervisor which lives outside of ROS, ensures that the simulation is progressing normally and none of the subsystems has crashed or entered an unusual state.

3.1.1 Considerations for StarlETH

To evaluate the terrain in simulation, a robot specific gait needed to be selected. In this thesis the most stable, but also slowest gate was chosen, to maximize the length of each run and consequently the amount of recorded footholds. In the chosen static crawling gait, the feet are moved in the following, successive order: Front right, hind left, front left, hind right. As it is a static gait, the objective of the gait controller is to keep the center of gravity always inside the support polygon.

3.1.2 Terrain Generation

The terrain generation was implemented as a Matlab script. The output of the script are heightmaps in the PNG file format, with a size of 2049×2049 pixels. The unusual pixel size of the images stems from a gazebo requirement, as we initially experimented with loading the heightmaps directly into gazebo. That proved to be performing very badly as the loading times for heightmaps of this size and with the amount of noise was unreasonably high. Therefore, the generated heightmaps were converted using Blender (a 3D modeling software) to `*.dae` shape files, a three-dimensional description format that can be loaded as underlying terrain in gazebo. The resulting 3D model measures $20 \times 20 \times 0.06$ meters, giving the robot enough freedom to traverse the world. These dimensions make one pixel roughly equivalent

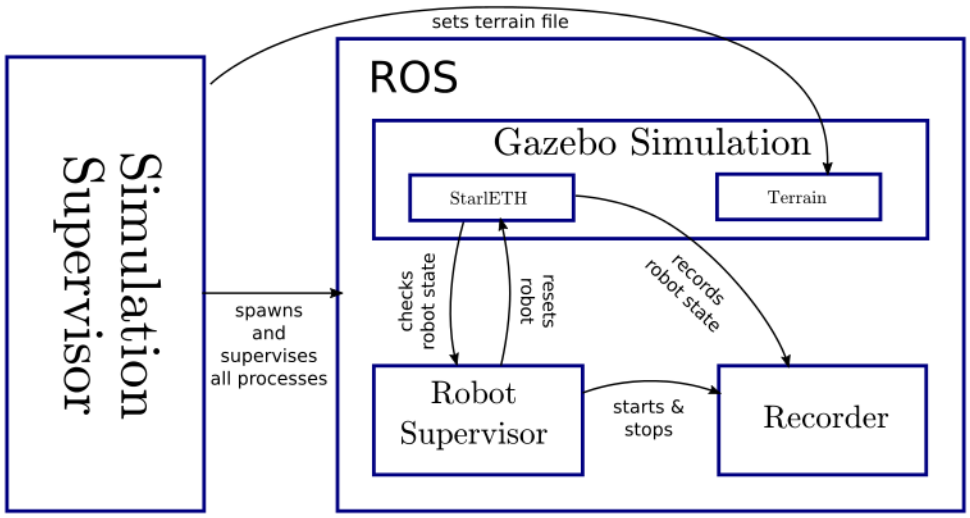


Figure 3.1: Overview of the implementation for the data collection in simulation

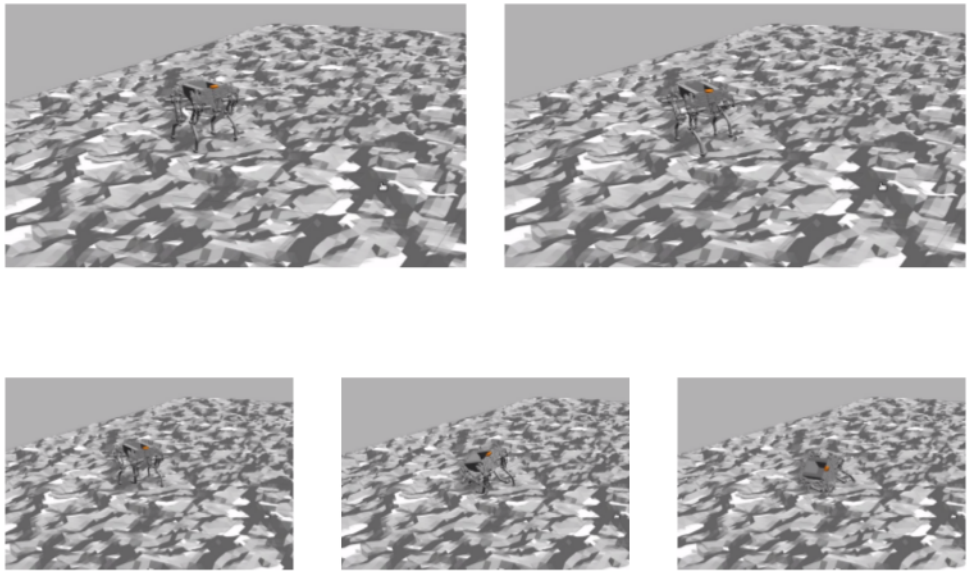


Figure 3.2: Sequence of the robot walking and slipping (with the left fore foot), and finally falling in the simulation environment

to a 1×1 centimeter square, which is also a realistic resolution for elevation mapping on the real robot.

3.1.3 Process Supervisor

One part of the system is the process supervisor. It spawns the ROS processes and continues to monitor them. If a process crashes or if it is observed that the process is ending up in a bad state, the entire simulation environment is torn down and restarted. This also happens after a certain amount of time, such that there will be no problems with memory leaks or other instabilities that manifest themselves over the long run. The simulation supervisor also selects the terrain file that will be used for the simulation. It is important to note that the process monitor watches the ROS nodes "from the outside" (ie. it's not a ROS process) and therefore it can reliably detect crashes in the ROS process, even if ROS itself is completely failing.

3.1.4 Robot Supervisor Node

The robot supervisor node is a ROS node that monitors the state of the robot and, if a major deviation happens to the robot (as defined below), restarts the recording and the robot. Additionally it is responsible for setting the gazebo simulation parameters (specifically it increases the simulation speed to three times realtime). For each run, the robot is randomly placed by the robot supervisor on the edge of the terrain (on the rim of a circle with 9.9 m diameter). The robot faces the center of the circle and is commanded to walk straight from there towards the center of the terrain. At the beginning of the run, the robot supervisor notifies the recording node to start a new recording. As soon as a major deviation of the robot is measured, the robot supervisor stops the recording and resets the robot to another randomly selected point on the circle. The major deviations are:

- Rotation about the roll or pitch axis if the angle is bigger than 40 degrees
- More than two feet in the air
- Almost on the edge of the terrain (within a 0.2 m confinement boundary)
- No movement for more than 15 seconds (this can happen when either something crashes, or the robot slips and falls, but in a way that the roll and pitch angle are not abnormally high)

3.1.5 Recorder Node

The recorder is a ROS Node that listens to the `quadruped_state` message emitted by the StarlETH state estimator during the simulation and saves the relevant information to a JSON file. Meta information, such as the current terrain that is loaded in the simulation and the current date and time are also saved.
The recorder can be activated through a service call which initializes a recording. The service call also contains the currently loaded terrain file name which is saved as metadata. During the entire recording duration, the recorder saves all timestamps and the position and rotation of the main body.
The foot position and rotation, and the force acting on the contact point is stored while the foot is in contact with the ground. Each foothold gets a unique, incrementing index and also stores the identity of the foot (ie. wether it is a left hind or right front foot). Storing an incrementing index allows the later recovery of the foothold order. When a foot is in the air, no information is stored about that particular foot.

Recorded Data

During recording the following data is stored:

- The simulated terrain name
- "Global" data consisting of:
  - Body position and rotation
  - The desired walking speed and direction
  - Current timestamp
- The footholds, each consisting of:
  - An incrementing index to be able to order the footholds
  - The foot identifier (left/right, front/hind foot)
  - The timestamps for each measurement
  - Foot contact point position and rotation over time
  - The measured force and torque at the foot contact point

Recording File Format

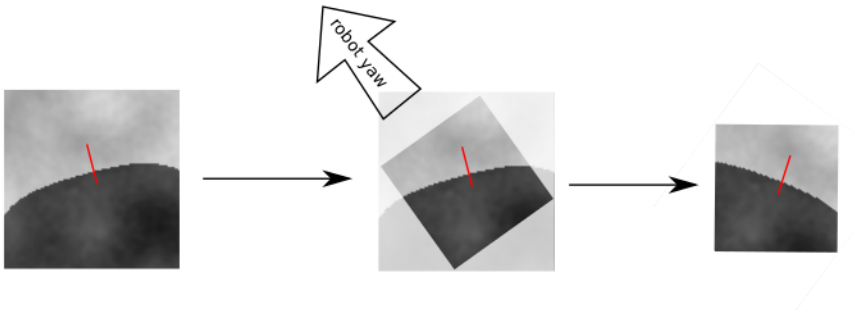
The raw data was recorded into a JSON file. JSON is a flexible, text-based key-value storage format which is convenient to use from many programming languages. Python dictionaries can natively be transformed into JSON and vice versa, making it an extremely versatile format. To save binary data in the JSON format, the format was extended to include fields for binary data which is encoded in the numpy binary format and stored as a Base64 string. For each recording session a new JSON file is created and stored. This keeps the file size of the files relatively small and prevents corruption issues that were encountered when modifying bigger files. A schema of the JSON files can be found in section 3.1.5.

3.1.6 Creating the Final Dataset

To create the final dataset, the recorded JSON files are concatenated and the entire data is saved to a single HDF5 file, which is a binary format that is very well suited to scientific data. HDF5 can be read directly in many programming languages. A main advantage over JSON is the fast reading speed and the native handling of arrays and matrices which are represented as numpy arrays in Python. While it was initially planned to record the heightmap from the elevation mapping available on the simulated robot, this did not work out due to bugs that could not be addressed in the limited time. Instead, the heightmap data was taken from the original image. Extracting and rotating the heightmap was executed during the generation of the final dataset, such that the correct heightmap data is readily available during the learning process.
The final dataset for learning is reduced to only the footholds, which consist of the heightmap, the foot position over time and the body quaternion at the time of contact. Additionally a path to the foothold source (ie. recording date and index) are stored to be able to retrieve the entire recording that belongs to the specific foothold. Data which is recorded but not used for learning (such as the foot force measurements) is ignored in the final HDF5 file.

```
{
  <recording time and date>: {
    meta: {
      terrain_id: <terrain identifier>,
    },
    {
    footholds: {
      <index>:
       {
          foot_id: one of RF|LF|RH|LH,
          foot_position: [<list of [x,y,z]>],
          timestamps: [<list of timestamps],
          force_profile: [<list of [Fx,Fy,Fz]>],
          torque_profile: [<list of [Fx, Fy, Fz]>]
       }
      }, ...
    },
    global: {
      body_position: [<list of [x,y,z]>],
      body_quaternion: [<list of [x,y,z,w]>],
      timestamps: [<list of timestamps],
      twist_linear: [<list of [x,y,z]>],
      twist_angular: [<list of [x,y,z,w]>]
    }
}
```

Figure 3.3: Schema of JSON file format



Is the cropped image size dependent on the yaw?

Figure 3.4: Rotating the heightmap about the robot yaw angle and cropping the resulting image

Heightmap Normalization

To align the heightmap with the robot, the extracted 40×40 pixel heightmap is rotated around the center with the robots body yaw and then cropped to a 20×20 pixel region to remove empty pixels from the rotation (compare Figure 3.4).
To shift the contact point to a neutral height, the center point is subtracted and the heightmap is rescaled:

OK now this is clear

Which unit is X expressed in?

$$X = 0.5 * (X - X[10, 10]) + 0.5$$

Filtering

The filtering was implemented in a straightforward fashion. However, the filtered data is contained in the final dataset, but contains an attribute consisting of a string that indicates which filters evaluated to true. This makes it easy to ignore filters or select only a few filters during the learning.

Chapter 4

Results

In this chapter the recorded dataset is discussed and the results of training the previously introduced machine learning algorithms are reported.

4.1 Dataset

To collect the dataset, an 8-core computer was running the simulation code for a total of about 3 days. The simulation speed could be increased to about 3 times of the realtime speed. This resulted in a dataset of about 1.3 gigabyte of footholds. The entire dataset contains a total of 148 274 footholds. The robot traversed 8 different heightmaps to collect the data.
The size of the dataset was mainly limited by the time the simulation could run on the computer. While a 32 core Mac Pro computer was available, it was unfortunately not possible to run the simulation in the provided VirtualBox environment due to issues with the hardware acceleration.
To check the feasibility of the dataset and the accuracy of the recorded data, several visualization tools were developed. Figure 4.2 shows a plot of several runs on a heightmap. The correlation between the edges and the amount of slip is clearly visible which shows the conclusiveness of the recorded data and the correct alignment of the data with the underlying heightmap.
In Figure 4.3 and Figure 4.1 several different foothold heightmaps are plotted, all rotated around in about the angle of the body yaw. The position of the foot during stance phase is drawn in red.
The distribution of the data is plotted in Figure 4.1. While 122 928 footholds have a deviation of 0 to 0.02 meters, only 3687 footholds slip for more than that. Using

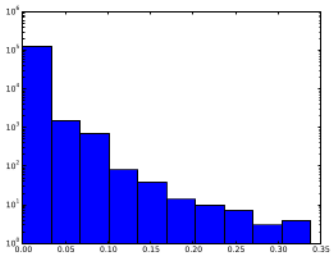


Figure 4.1: Histogram of data distribution in the raw dataset. Note the log-scale on the y axis. The x-axis is the amount of slip (training label).

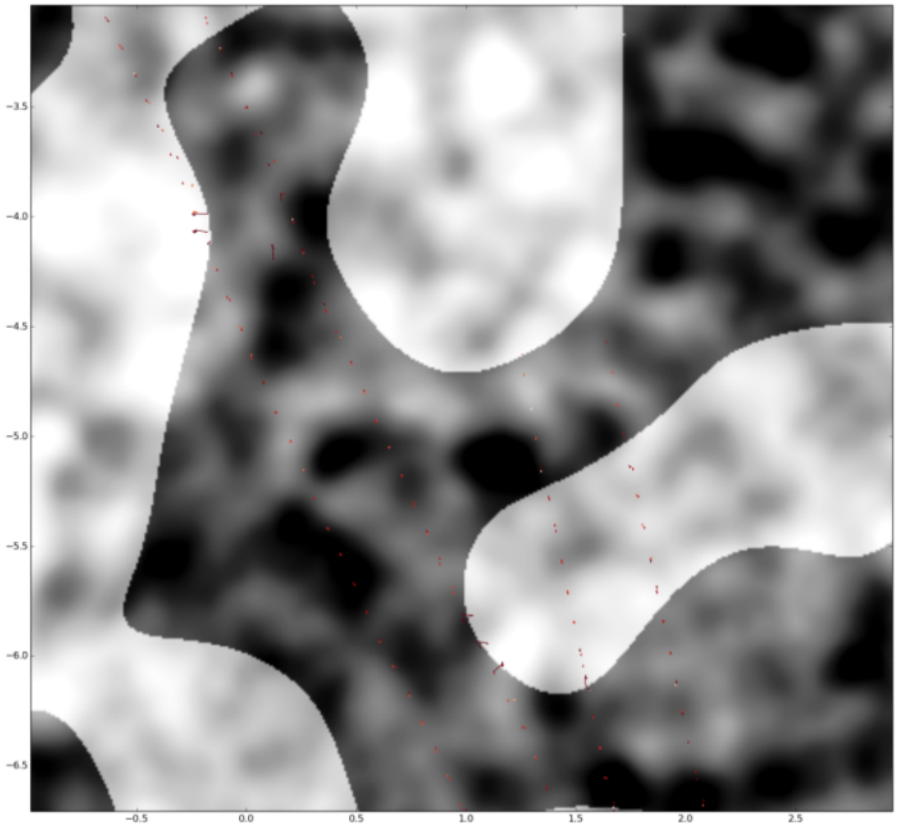
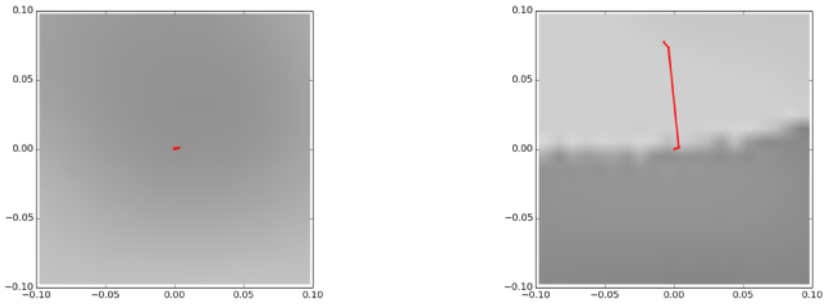


Figure 4.2: Recorded footholds plotted in red on top of the underlying terrain. Note slip occurring on the edges. The dark parts are the elevated terrain.

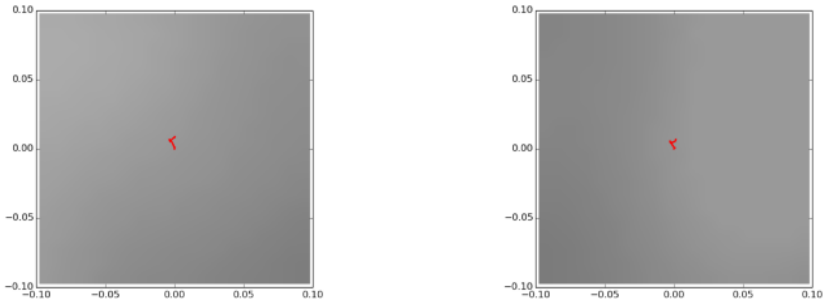
A colorbar would look good here

the described dataset resampling technique, 7374 footholds are obtained in the final dataset. For learning, the dataset is always split into a validation and a training set. The validation set consists of 20 % of the dataset, ie. containing 1475 footholds (versus 5899 footholds in the training set).

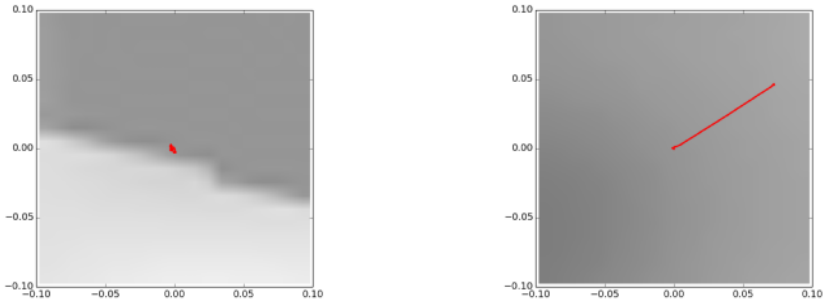
Describe how the split is defined
I'm convinced that even random sampling would not create problems here, but ML people are always very careful that the validation and training set do not overlap.
So you should argue why they don't.



Examples of very little slip and a lot of slip



Two regular footholds. The deviation stems from the motion of the contact point which moves as the ball foot "rolls" on the ground



Two "outliers": The first foothold has the step behind and therefore there is less slippage backwards possible. The second foot slips even though no height deviation is in the patch.

I understand the explanation but still it's unclear to someone new

Any idea why it slipped anyway?

Figure 4.3: Normalized footholds with the foot contact point plotted. Note that the initial contact point of the foot is always in the center of the patch.

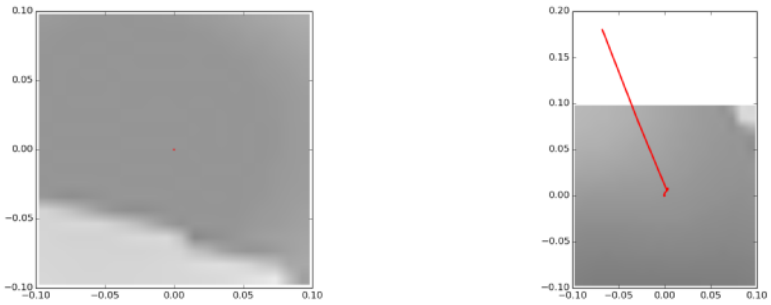


Figure 4.4: Filtered footholds

Unclear: why they were filtered?

4.2 Baseline SVM Regression

The SVM regression was implemented using the `scikit-learn` framework [13] which ships an implementation based on `libsvm` [14].

4.2.1 Features

One goal of the baseline SVM regression was to keep the amount of features deliberately small. The features with best performance and ease of implementation from Hoepflinger [6] were used.

- Standard deviation around the center of the patch with radii 2px, 4px and the entire heightmap
- The maximum height difference in the patch
- The difference between the maximum height and the height at the center

This results in a feature vector x with 5 features. Before learning the feature vector is scaled to zero-mean and unit-variance.

4.2.2 SVM Parameters

The chosen parameters were found through a grid search over a range of feasible parameter combinations. The best regression results were achieved when using

$$\begin{aligned} \varepsilon &= 0.00001 \\ C &= 1.0 \\ \text{Kernel} &= \text{"RBF"} \end{aligned}$$

where ε is the parameter for the regression tolerance boundary and C is the penalty parameter for the error term. The radial basis function kernel also showed better performance than the linear kernel.

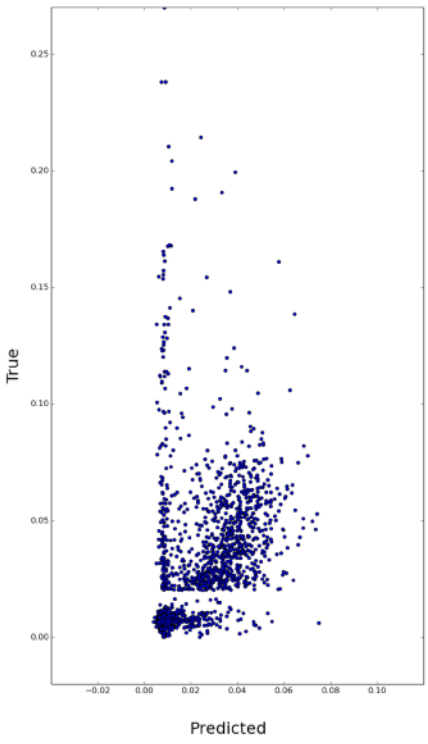


Figure 4.5: Scatter plot of SVM regression predictions of the validation set. The gap in the data stems from the rebalancing.

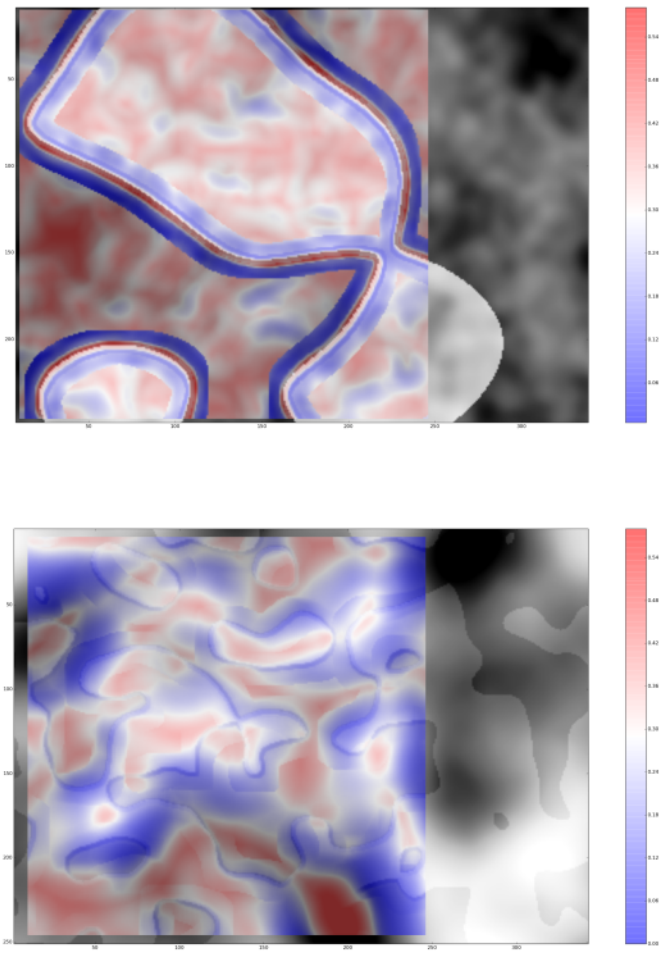


Figure 4.6: Two terrains that the SVM has never seen before. The colors correspond to the estimated slip, where red is a bad foothold (ie. a lot of slip). The terrains where scored by extracting 20x20 pixel patches and scoring them one-by-one.

Cool plots, but not easy to interpret due to two color scales mixing to each other

4.3 Training the Convolutional Neural Network

The convolutional neural network was implemented using the Brainstorm library from IDSIA [15], which makes it easy to experiment with different neural network architectures. Since it supports the CUDA acceleration framework, the neural networks can be trained very effectively on NVIDIA Graphics Processing Units (GPUs).

4.3.1 Neural Network Architecture

Some properties of the problem allow the neural network architecture to be somewhat simpler compared to architectures for generic image classification tasks like "CIFAR-100", where the best performing networks are extremely deep. For one, the rotation and scale of the input data is entirely normalized. Whereas classification of e.g. cars or planes from generic color images requires the neural network to learn features that are scale- and rotation invariant, this is not the case for the task at hand. Therefore, the neural network architecture can be simplified and contain fewer max-pooling and convolution layers while still being able to obtain meaningful results.
To avoid overfitting, use of dropout layers was made. The standard technique for image classification tasks is data augmentation, where images are randomly altered in saturation, exposure or rotation/translation to avoid overfitting and increase rotation and scale invariance. In this task this was no option since it would not make sense in the context of the normalized heightmaps. Using the dropout layers improved the regression error on the validation set significantly.
The final neural network architecture looks as follows:

- Input layer, shape 20×20
- Convolutional layer, 12 filters of size 10×10
- Max Pooling layer, kernel size 3×3, stride 2
- Dropout layer, $p_{dropout} = 0.3$
- Convolutional layer, 24 filters of size 5×5
- Max Pooling layer, kernel size 3×3, stride 2
- Dropout layer, $p_{dropout} = 0.2$
- Fully connected layer with 500 neurons
- Fully connected layer with 250 neurons
- Output, 1 neuron

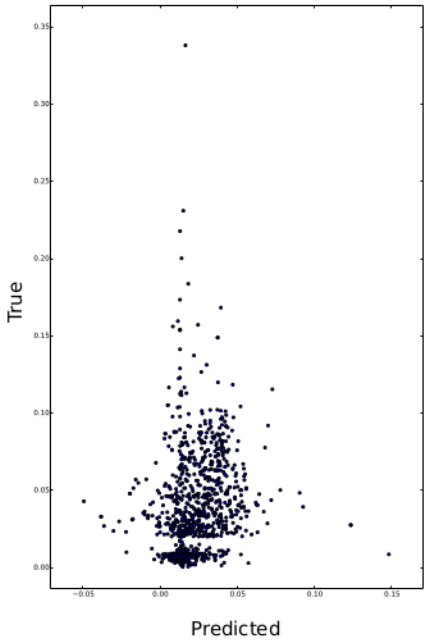


Figure 4.7: Scatter plot of CNN predictions of the validation set. Note that the regression result is negative for some examples.

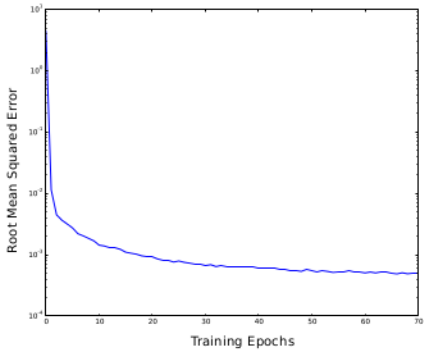


Figure 4.8: Root mean squared error over epochs of training

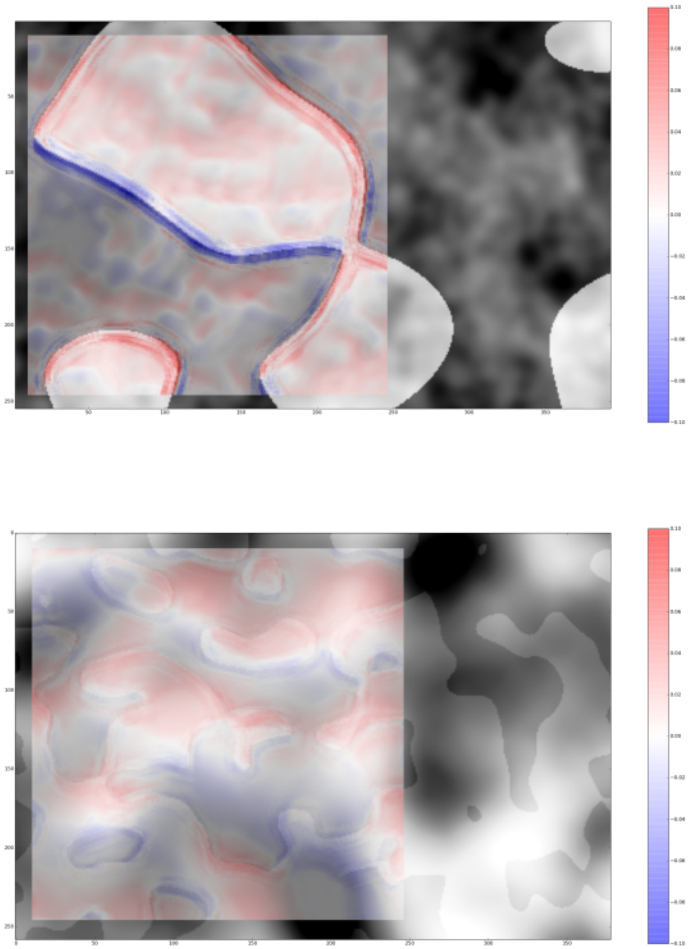


Figure 4.9: The CNN scoring two unknown terrains

As above. It would also be nice to see a comparison of the score maps from SVM and CNN (without underlying height map).

Visualization suggestion: a figure with 4 subfigures
subfigure 1: 3D representation of heightmap, with proper lighting and shading to show 3D shape
subfigure 2: 2D represenation of the same heightmap, with only isolines
subfigure 3: score map from SVM (blue-red colormap), overlayied with same isolines as abov
subfigure 4: score map from CNN (blue-red colormap), overlayied with same isolines as abov

I believe the isolines could help cross-referencing the different score maps and the heightma so we avoid the difficulty in understanding the overlayed colormaps.

Chapter 5

Conclusion

In this thesis it was shown how a system can automatically derive the geometric characterization of a good and bad foothold without human intervention. Both, the SVM machine learning algorithm and the Convolutional Neural Network successfully identify the edges in the terrain as the zones of least foothold stability, which is the intuitive correct deduction.

The biggest problem in this thesis was the collection of correct data. When further analyzing the SVM solution, most of the outliers are actually "correct" outliers in the sense that the recording is wrong, and the slip actually occurs due to some error in the simulation. In Figure 5.1 two data points that the SVM judges very wrongly are shown. In the first one, heavy slip occurs on completely flat terrain. Since the other feet seem stable, the source of this error is not clear. In the second foothold recording, a patch with a heavy edge (or, also a very high standard deviation in the center) is traversed without slipping, but the SVM assumes a high chance of slip, which is justified given the other evidence. While the simulation generally runs good, there were a few issues encountered and the robot sometimes slips and misbehaves in non-intuitive ways. While some of the bugs might be due to the Open Dynamics Engine or Gazebo, a lot of the software stack making up the StarlETH robot is academic in nature and not necessarily tested for stability in long running simulations. This leads to the conclusion that the training data contains too much noise to obtain very nice regression results.

One observation that is made in the scored terrains by the SVM is that the best foothold scores occur in a certain distance around an edge. This leads to the conclusion that on flat terrain, where no "evidence" is available as the terrain always looks the same, the slip tends towards some mean between the outliers and the true no-slip score.

This is a reasonable explanati but points to so much import ance of outliers...

The convolutional neural network solution also offers two interesting observations: First, for some heightmap patches the regression score is actually negative which is counter intuitive as negative slip does not make physical sense. And second, the negative slip always occurs after changing from high to low in the walking direction. If the negative score is interpreted as a very good foothold, then the score might make sense in so far as the feet cannot slip backwards when stepping against something behind. On the other hand, since the score is also negative "in front" of the actual edge, where the feet could slip downwards, it is questionable if this interpretation is correct or if the CNN regression is just unsuccessful at this point. It is also worth noting that this directionality is not captured with the SVM (as there are no features dependent on the direction in the feature set) and it shows the superiority of the convolutional neural network in developing its own features.

with respect to

On the second unknown terrain scored by the CNN and the SVM, both show inconclusive results. The edges in the second terrain are less pronounced as in the

Correct, but also note that we could have developed features capable of detecting directionality

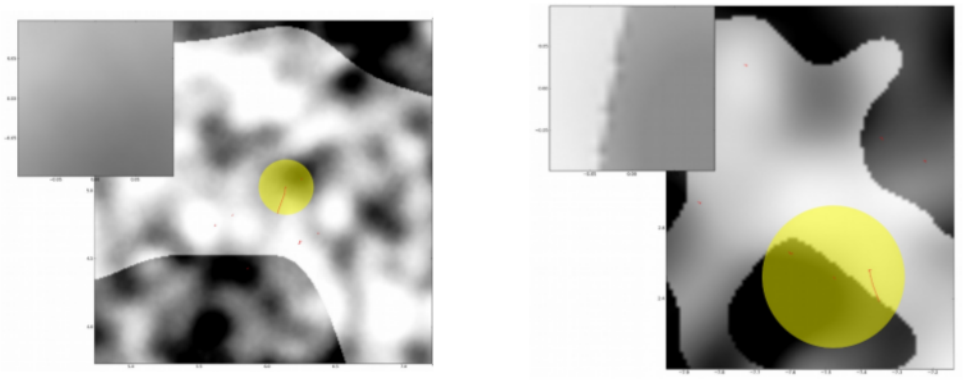


Figure 5.1: Two examples of "noise" from the validation dataset. The yellow circle highlights the foothold on the terrain, and the top left shows the heightmap as the machine learning algorithm sees it. The first example shows heavy slip even though traversing entirely flat terrain. The second example shows a very risky foothold that does not slip.

other example. The SVM seems to actually prefer the edges to the flat surroundings, highlighting further the problems with the training data. The CNN shows a foothold selection that seems to be slightly better, and the absolute scores are lower than with the stronger edges, but no reasons for the separation between good and bad footholds are obvious. Apart from the noisy training data, one possible reason for the bad performance on this particular terrain is that the selected terrains for the data collection had more pronounced edges and thus there are few to no examples of this and similar terrains in the training data (or in other words, the data set is too small).
The final root mean squared errors of the machine learning algorithms show that the CNN clearly outperforms the SVM regression, and both are better than just guessing the mean:

Method	RMSE
Mean	0.03144315885
SVM	0.03001497502
CNN	0.02209672532

This is a very reasonable conclusion.
But also note in the text that the SVM was fed with *very* simple features in this case.

5.1 Outlook

As shown above, the major problem in terms of the regression is the underlying data. While a lot of data was collected, the simulation was running somewhat instable during most of the time, with many runs with fewer than 10 footsteps (and thus, filtered). To reduce the amount of outliers in the future, it would be advisable to either stabilize the simulation software or implement a further filtering. One simple idea for filtering would be to filter all slips where almost no height deviation in the patch occurs. However, as this removes data it also introduces an interpretation at an very early stage of the learning process.
It would be interesting to use the trained scorer to traverse terrain that becomes increasingly difficult as a kind of "feedback" loop. One way to make the terrain harder is to tilt the entire terrain and make StarlETH climb uphill. This would be close to impossible without any sort of terrain scoring but might be feasible with a terrain scoring previously trained on flat land.

Interesting idea

The machine learning algorithms should also get more access to robot data, such as the body pitch and roll and the identifier of the foot to be able to include more information about the general robot state in the foothold. Also, the relation between the four feet is currently not captured in the learning process. This would be especially necessary when climbing, as the center of gravity has to stay inside the support polygon to prevent falling (which is also influenced by the foothold selection).
Training a CNN with the data from only one foot at a time and comparing the resulting regressors could lead to interesting conclusions about the influence of the foot position on the robot (ie. comparing left hind to left fore foot). Changing the regression target from the maximum distance to the angle of slip could yield another interesting regressor and the output could be fed into the distance regression as additional input.
It would be highly interesting to use the obtained scorer to select footholds in the crawling gait and then compare the resulting trajectories in simulation. Unfortunately, the time was also too short to evaluate the foothold scoring learned in simulation on the real robot hardware.

Bibliography

[1] M. Hutter, C. Gehring, M. Bloesch, M. A. Hoepflinger, C. D. Remy, and R. Siegwart, "Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *15th International Conference on Climbing and Walking Robot-CLAWAR 2012*, no. EPFL-CONF-181042, 2012.

[2] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," *Mobile Service Robotics: CLAWAR 2014*, vol. 12, p. 433, 2014.

[3] P. D. Neuhaus, J. E. Pratt, and M. J. Johnson, "Comprehensive summary of the institute for human and machine cognition's experience with littledog," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 216–235, 2011.

[4] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng, "A control architecture for quadruped locomotion over rough terrain," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on.* IEEE, 2008, pp. 811–818.

[5] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 236–258, 2011.

[6] M. A. Hoepflinger, M. Hutter, C. Gehring, M. Bloesch, and R. Siegwart, "Unsupervised identification and prediction of foothold robustness," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE, 2013, pp. 3293–3298.

[7] D. Belter, "Adaptive foothold selection for a hexapod robot walking on rough terrain."

[8] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012, pp. 3642–3649.

[9] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[10] A. Valada, L. Spinello, and W. Burgard, "Deep feature learning for acoustics-based terrain classification," 2015.

[11] A. Smola and V. Vapnik, "Support vector regression machines," *Advances in neural information processing systems*, vol. 9, pp. 155–161, 1997.

[12] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[13] 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*, vol. 12, pp. 2825–2830, 2011.

[14] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[15] K. Greff and R. Srivastava, "Brainstorm: Fast, flexible and fun neural networks," 2015, software available at https://github.com/IDSIA/brainstorm.