University of Southern Denmark

Master Thesis

Classification of terrain based on proprioception sensing for multi-legged walking robot

Author:
Bc. Martin Bulín

Supervisors:
Dr. Tomas Kulvicius
Dr. Poramate Manoonpong

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Embodied AI & Neurorobotics Lab
Faculty of Engineering

May 8, 2016

Declaration of Authorship

I, Bc. Martin Bulín, declare that this thesis titled, "Classification of terrain based on proprioception sensing for multi-legged walking robot" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"*Favorite quotation.*"

Quotation Author

UNIVERSITY OF SOUTHERN DENMARK

Abstract

Faculty of Engineering

Embodied AI & Neurorobotics Lab

Master of Science

Classification of terrain based on proprioception sensing for multi-legged walking robot

by Bc. Martin BULÍN

The abstract is a concise and accurate summary of the research described in the document. It states the problem, the methods of investigation, and the general conclusions, and should not contain tables, graphs, complex equations, or illustrations. There is a single abstract for the entire work, and it must not exceed 350 words in length.. . .

Acknowledgements

Students may include a brief statement acknowledging the contribution to their research and studies from various sources, including (but not limited to)

Their research supervisor and committee, Funding agencies, Fellow students, and Family.

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Algorithms and Code Parts

Chapter 1

Introduction

The thesis must clearly state its theme, hypotheses and/or goals (sometimes called “the research question(s)”), and provide sufficient background information to enable a non-specialist researcher to understand them. It must contain a thorough review of relevant literature, perhaps in a separate chapter.

1-2 pages intro

1.1 Problem Formulation

1 page Motivation and Research Questions

1.2 Motivation for Chosen Methods

motivation for using proprioception sensing motivation for using a neural net as a classifier

1/2 page

1.3 Hypotheses

1/2 page

1.4 Thesis Outline

1/2 page

Chapter 2

State of the Art

chapter intro

2.1 Machine Learning and Classification

Machine Learning and Classification in general, different classifiers (SVM, k-NN, RandomForest, Bayes...)

2-3 pages

2.2 Introduction to Neural Networks

neural networks from the beginning, network types, principles its usage for classification

4-5 pages

2.3 Pruning Algorithms

based on the paper Pruning Algorithms - A Survey: a summary of what has been already done, principles 1-2 pages

2.4 Terrain Classification for Legged Robots

based on the literature : a summary of what has been already done in terrain classification, summary of different methods (visual, laser, haptic, proprioception, ...)

5-8 pages

Chapter 3

Master Thesis Objectives

objectives (goals) 1/2 page

Chapter 4

Neural Network Implementation

The account of the research should be presented in a manner suitable for the field. It should be complete, systematic, and sufficiently detailed to enable a reader to understand how the data were gathered and how to apply similar methods in another study. Notation and formatting must be consistent throughout the thesis, including units of measure, abbreviations, and the numbering scheme for tables, figures, footnotes, and citations. One or more chapters may consist of material published (or submitted for publication) elsewhere. See "Including Published Material in a Thesis or Dissertation" for details.

chapter intro

overall kitt_nn framework diagram

1 page

4.1 Structural Elements

kitt_net.py, kitt_neuron.py, kitt_synapse.py

structure diagram

1-2 pages

4.2 Learning Algorithm

Backpropagation implementation in python

algorithm

1-2 pages

4.3 Graphical User Interface

GUI description and its usage

printscreen

1 page

Chapter 5

Network Pruning Algorithm

5.1 Testing Datasets

XOR Dataset

XOR dataset introduction

MNIST Dataset

MNIST dataset introduction

5.2 Minimal Structures Utilization

further MNIST analysis

figures, tables

4-5 pages

Chapter 6

Terrain Classification for AMOS II

Classification, one of the most widely used areas of machine learning, has a broad array of applications (see chapter 1). To fit a classifier to a problem, one needs to define a problem data structure. Data consists of samples and discrete targets, often called classes. The samples are sooner or later converted into so called feature vectors of a fixed length. The length of feature vectors usually determines an input of a chosen classifier and the number of classes sets an output.

The classification problem in this thesis relates to AMOS II, an open-source multi sensori-motor robotic platform (see fig. 6.2). The task is to classify various terrain types, while the only input comes from proprioceptive sensors. The overall process is based on simulation data and as chapter 4 reveals, feedforward neural networks are involved.

6.1 Overall Process Summary

The very first step is to make the AMOS II simulation run (section 6.2.2). Then a simple tripod gait controller is implemented (section 6.2.3). To generate various terrain types, the number of variable terrain quailities and their ranges are determined (section 6.3.1). Based on these qualities (parameters), a number of virtual terrains is defined (section 6.3.2) and an optimality of these parameters is briefly analysed (section 6.3.3).

Next, AMOS II (its simulation alternative) is forced to walk on every defined terrain type several times and for a sufficiently long period of time and the data from all proprioceptors are saved. This data is then verified and failing experiments are removed. The data acquisition step is parameterized by a standard deviation of an additive (Guassian) terrain noise and is run for several values.

Having the clean simulation data from all sensors, a feature vector structure is determined. Then a Gaussian signal noise is added. Finally, a dataset is created by splitting all the data into training, validation and testing sets. As it is indicated in fig. 6.1, several datesets and several classifiers are generated during the process.

An optimal neural network classifier is found. The optimal network is then pruned by the algorithm developed in chapter 5. The classification performance of developed tools is compared to *Scikitlearn-neuralnetwork* classification library [CS15].

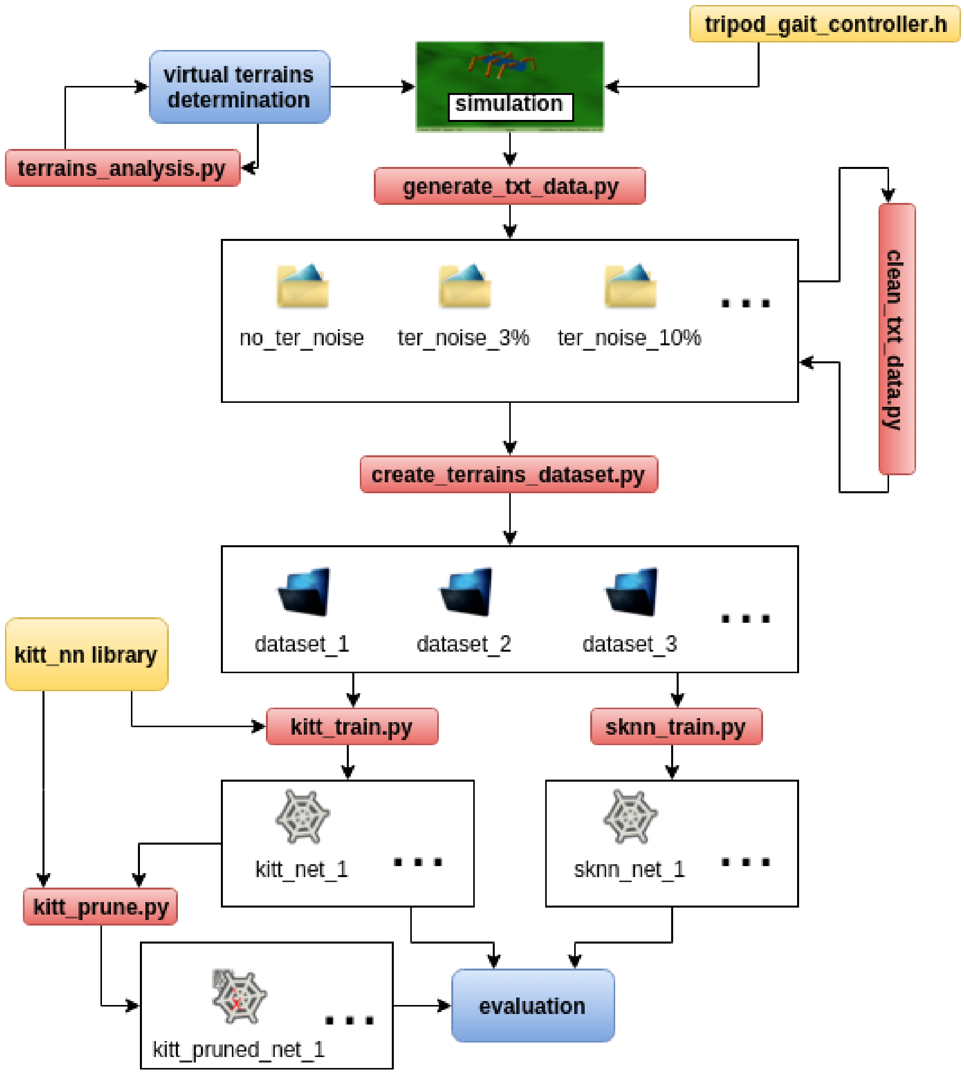

FIGURE 6.1: Terrain classification process - overall diagram.

The dataset packages may differ in these parameters:

- terrain types included (-> number of classes)
- sensors on input
- samples length (number of simulation timesteps)
- terrain noise and signal noise
- number of samples

The trained networks may differ in the following parameters:

- dataset that the network has been tested on
- neural network structure, learning rate and number of epochs

6.2 Experimental Environment Specification

Naturally, the idea of the research is to implement an online terrain classifier on the real machine. Therefore the target robot is described in the following section (6.2.1).

Nevertheless, it is usually a good idea to base the reasearch on some simulation data if a satisfactory simulator is available. In this case, *LPZ Robots* [Misc] is used (section 6.2.2).

6.2.1 Hexapod Robot AMOS II

The *AMOS II* abbreviation stands for Advanced Mobility Sensor Driven-Walking Device - version II. It is a biologically inspired hardware platform of size 30x40x20 cm and weight 5.8 Kg (see fig. 6.2). It is mainly used to perform experiments with neural control, memory and learning on a device with many degrees of freedom and to study its coordination [Misa].

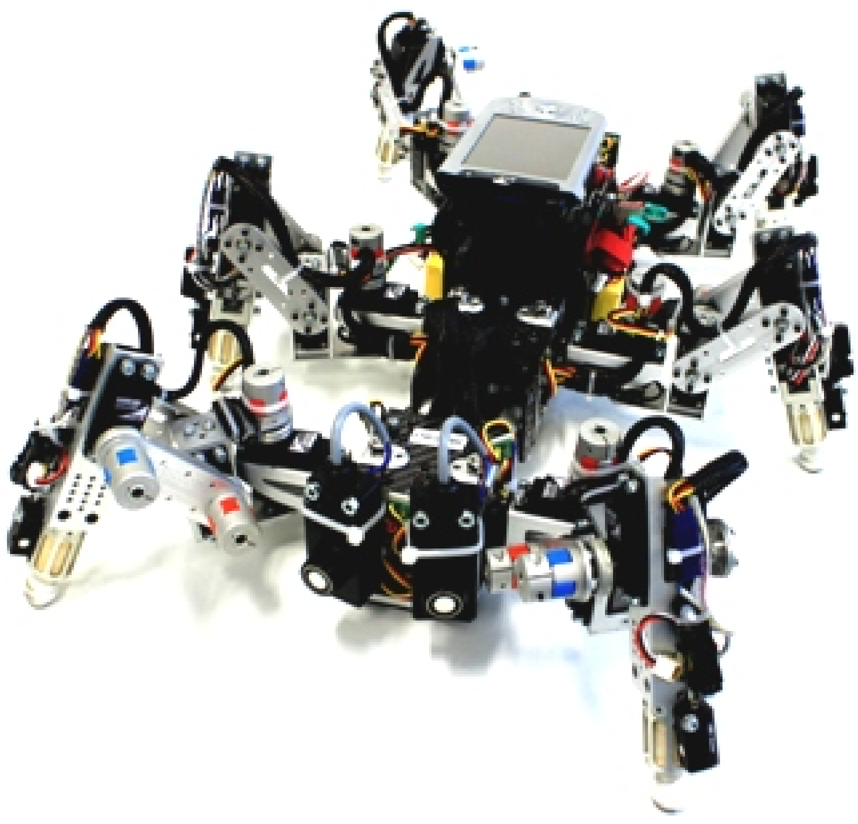

FIGURE 6.2: AMOS II. [Misa]

In general, the robot serves as a hardware platform for neural perception-action systems experiments. The body parts are modeled on the basis of robot's biological inspiration - a cockroach.

A wide range of sensors allows AMOS II to perform several kinds of autonomous behaviour. However, only the proprioceptive sensors are important for this research, therefore, we focus on angle sensors and foot contact sensors. All of them are located on robot's legs, so the leg structure is shown in fig. 6.3.

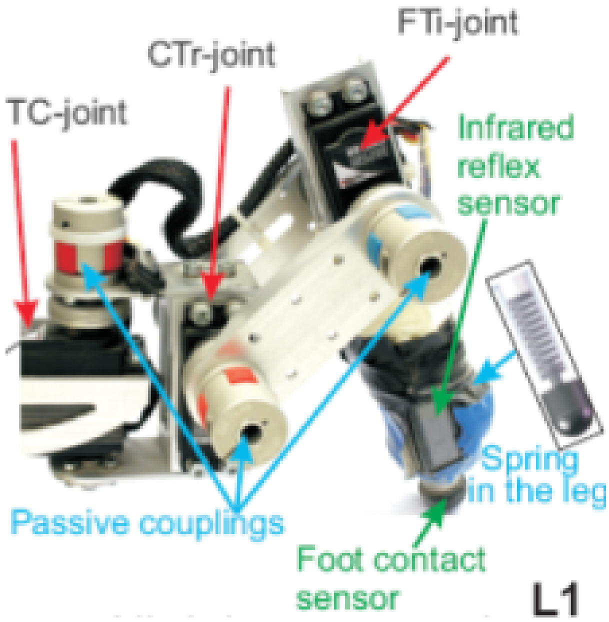

FIGURE 6.3: AMOS II. [Misa]

As figures 6.2 and 6.3 reveal, the robot has **6 foot contact sensors** in total, one on each leg. Each of them returns a value from range $[0.0, 1.0]$ depending on how strong the foot contact is - it is equal 1.0 if the robot stands on the leg with its full weight and it equals 0.0 when the leg is in the air.

There are three joints on each of the robots legs. The thoraco-coxal (TC-) joint is responsible for forward/backward movements. The coxa-trochanteral (CTr-) joint enables elevation and depression of the leg and the last one, femur-tibia (FTi-) joint is used for extension and flexion of the tibia.

These joints are physically actuated by standard servo motors. Having the servos positions, angles of the joints are known and are also considered as propriceptive sensors. As AMOS II has six legs and there are three joints on each leg, there are **18 angle sensors** in total. There is also one backbone joint angle, however, as this one is not implemented in the simulation (see section 6.2.2), it is omitted in this research.

In table 6.1 all the propriceptors, their shortcuts and original ranges are listed. The ranges are based on the individual servos locations and are explicitly set up to avoid collisions. In **??** a normalization of these ranges is discussed.

Regarding robots actuators, the servo motors can produce variably compliant motions as if each of them was driven by a pair of agonist and antagonist muscles (see [Misa] for details).

TABLE 6.1: AMOS II - Proprioceptive sensors

shortcut	*sensor description*	*original range*
ATRf	Angle sensor, Thoraco joint, Right front leg	
ATRm	Angle sensor, Thoraco joint, Right middle leg	
ATRh	Angle sensor, Thoraco joint, Right hind leg	
ATLf	Angle sensor, Thoraco joint, Left front leg	
ATLm	Angle sensor, Thoraco joint, Left middle leg	
ATLh	Angle sensor, Thoraco joint, Left hind leg	
ACRf	Angle sensor, Coxa joint, Right front leg	
ACRm	Angle sensor, Coxa joint, Right middle leg	
ACRh	Angle sensor, Coxa joint, Right hind leg	
ACLf	Angle sensor, Coxa joint, Left front leg	
ACLm	Angle sensor, Coxa joint, Left middle leg	
ACLh	Angle sensor, Coxa joint, Left hind leg	
AFRf	Angle sensor, Femur joint, Right front leg	
AFRm	Angle sensor, Femur joint, Right middle leg	
AFRh	Angle sensor, Femur joint, Right hind leg	
AFLf	Angle sensor, Femur joint, Left front leg	
AFLm	Angle sensor, Femur joint, Left middle leg	
AFLh	Angle sensor, Femur joint, Left hind leg	
FRf	Foot contact sensor, Right front leg	[0.0, 1.0]
FRm	Foot contact sensor, Right middle leg	[0.0,1.0]
FRh	Foot contact sensor, Right hind leg	[0.0, 1.0]
FLf	Foot contact sensor, Left front leg	[0.0, 1.0]
FLm	Foot contact sensor, Left middle leg	[0.0, 1.0]
FLh	Foot contact sensor, Left hind leg	[0.0, 1.0]

For purposes of this thesis, it is enough to know that it is possible to generate various gaits using the joints actuators and robot's neural locomotion control. The gait controller used for this research is described in section 6.2.3.

6.2.2 LPZ Robots Simulation

The *lpzrobots* project, developed by a research group at the University of Leipzig [Misc] under GPL license, contains many subprojects. For purposes of this thesis, the most important ones are:

selforg : homeokinetic controllers implementation framework

ode_robots : a 3D physically correct robot simulator

The project is implemented in *C++* and needs an Unix system to be run. It consists of two main GIT repositories to be forked - lpzrobots and go_robots. The overall software architecture is shown in fig. 6.4.

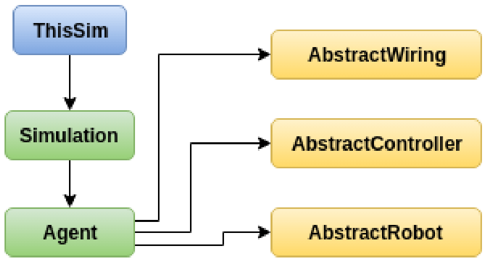

FIGURE 6.4: Software architecture for LPZRobots and GoRobots. [Misc]

To introduce the elements in fig. 6.4, *ThisSim* is an inherited class of another class called *Simulation* and is initialized everytime the simulation is launched. It integrates all elements together, controls the environment as well as the robot and sets up initial parameters.

An instance of the *Agent* class integrates all components of the agent (robot) by using the shown classes.

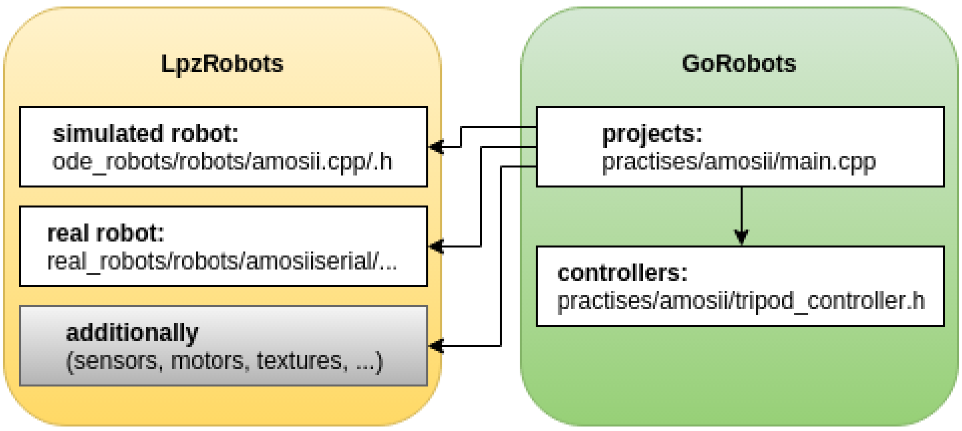

FIGURE 6.5: Structure of the two repositories (LPZRobots and GoRobots). [Misc]

In fig. 6.5 the cooperation of the two repositories is illustrated. With reference to appendix A2, one can call the *main.cpp* file from *root/simulation/mbulinai22015-gorobots_edu-fork/practices/amosii* directory as the main simulation file for purposes of the thesis. It sets up the environment with initial parameters $controlinterval = 10$ and $simstepsize = 0.01$, which means the simulation sensitivity is *10 steps* per second.

It also sets the initial camera and a robot position in the map. The robot position is chosen randomly and the reason for that is described in section 6.4. The robot fixator, which is originally implemented for AMOS II, is removed, so the robot starts walking right after the simulation is launched.

The *main.cpp* file contains all terrain types parameters introduced in section 6.3. The required terrain to be simulated is then passed to this file as an argument. Additionally, the standard deviation value of Gaussian terrain noise (details in section 6.3.4) is set as another argument. Finally, the file is ready to take one more argument, which is a simulation noise represented by a float number. In this research it is fixed to zero though and only the terrain noise combined with a signal noise is used.

The virtual vizualization of AMOS II is illustrated in fig. 6.6.

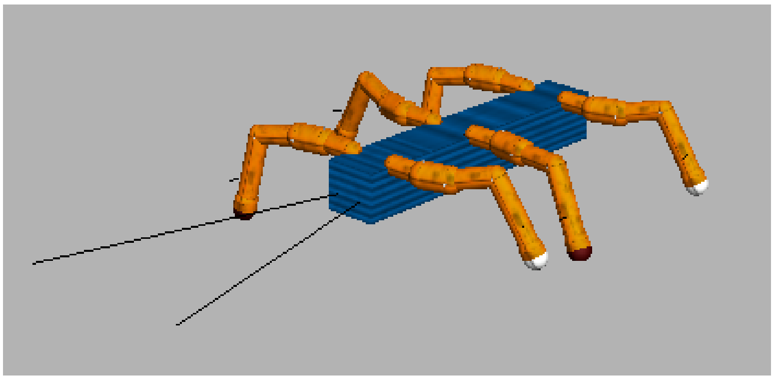

FIGURE 6.6: Simulation alternative for AMOS II.

6.2.3 Tripod Gait Controller

The main motivation for the terrain classification is to adjust the current robot's gait accordingly and save some energy thereby. It is assumed that the robot is already walking, using an implemented gait, when it tries to classify the terrain. Hence, it is needed to make the simulation agent walk as well. The starting gait is decided to be the **tripod** gait.

To generate the tripod gait, a central pattern generator (CPG) is used. [Man] It is implemented as a 2-neuron neural network right inside AMOS II (fig. 6.7).

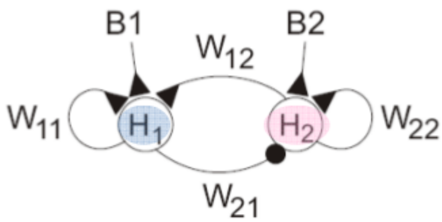

FIGURE 6.7: 2-neuron network oscillator. [Man]

To make it work in practise, *tripod_controller.h* is written. Its initial conditions and parameters are shown in part of code 6.1.

PART OF CODE 6.1: Initialization in tripod_controller.h

```
aH1 = 0; aH2 = 0;              // activities
bH1 = 0; bH2 = 0;              // biases
oH1 = 0.001; oH2 = 0.001;      // outputs
wH1H1 = 1.4; wH1H2 = 0.4;      // weights to H1
wH2H2 = 1.4; wH2H1 = -0.4;     // weights to H2
p1 = 0.35;                     // parameter for Thoraco joints
p2 = 0.3;                      // parameter for Coxa joints
```

Then, during the simulation, in *tripod_controller.h* there is a function called *step()* able to control robot's joints in every single simulation step. In this function three important actions come about.

1. **The activation function**

$$a_i(t+1) = \sum_{j=1}^{n} w_{ij} o_j(t) + b_i, i = 1, ..., n \tag{6.1}$$

   In this case, the following happens:

$$\begin{aligned} a_{H_1} &= w_{H_1,H_1} * o_{H_1} + w_{H_1,H_2} * o_{H_2} + b_{H_1} \\ a_{H_2} &= w_{H_2,H_2} * o_{H_2} + w_{H_2,H_1} * o_{H_1} + b_{H_2} \end{aligned} \tag{6.2}$$

2. **The transfer function**

$$f(a_i) = tanh(a_i) = \frac{2}{1+e^{-2a_i}} - 1 \tag{6.3}$$

$$\begin{aligned} o_{H_1} &= tanh(a_{H_1}) \\ o_{H_2} &= tanh(a_{H_2}) \end{aligned} \tag{6.4}$$

3. **Joints settings** With the reference to previous equations and variables names, the actuators are set in the sense shown in fig. 6.8. The *femur* joints (red ones) stay unchanged (set to zero). This setting generates a reliable tripod gait for AMOS II.

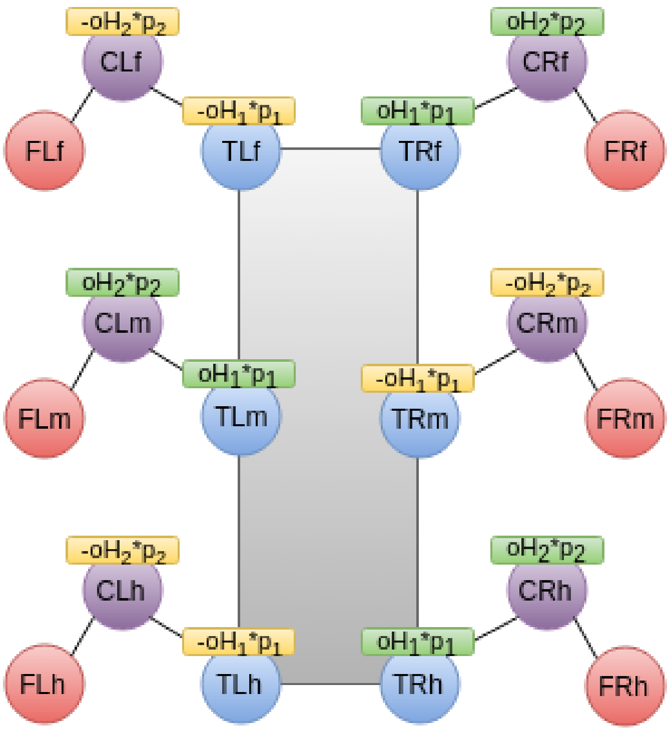

FIGURE 6.8: Tripod gait controller illustration.

6.3 Virtual Terrain Types

Since the verification is based on the simulation only, the goal is to design an authentical virtual environment. For this purpose various terrain types need to be virtually imitated.

Luckily, the **LpzRobots** AMOS II simulator supports some terrain setting. In the main simulation file (*main.cpp* - see A2), a *'rough terrain'* substance is being initialized and passed through a handle to a *TerrainGround* constructor.

Part of Code 6.2: Setting a terrain ground in main.cpp

```
Substance roughterrainSubstance(terrain_roughness, terrain_slip,
                       terrain_hardness, terrain_elasticity);
oodeHandle.substance = roughterrainSubstance;
TerrainGround* terrainground = new TerrainGround(oodeHandle,
                       osgHandle.changeColor(terrain_color),
                       "rough1.ppm", "", 20, 25, terrain_height);
```

As part of code 6.2 shows, the terrain substance is defined by four parameters: **roughness**, **slipperiness**, **hardness** and **elasticity**.

Besides the substance handle, the *TerrainGround* constructor takes six more arguments.

terrain_color : simulation ground color

"rough1.ppm" : an image in the .ppm format, a lowest common denominator color image file format [Misb], a bitmap height file

"" : texture image (not used)

20 : walking area x-size

25 : walking area y-size

terrain_height : maximum terrain height

6.3.1 Terrain Qualities

Out of the listed ground parameters, some of them are picked up and being called *terrain qualities*, as they define a specific terrain type.

It has been decided not to change the *.ppm* image for various terrains and so *rough1.ppm* is fixed. Also the walking area is set to the (big enough) final size of *20x25*. The color is variable, however, besides the simulation graphics it does not have an effect on results.

Therefore, a virtual terrain type is defined by five qualitites. Each of them is a float number from an empirically stated range [1]. (table 6.2).

[1]The upper range limits have been set up based on significant changes in the robot behaviour for various parameter values.

Table 6.2: Terrain qualities and their ranges

	min value	max value
roughness	0.0	10.0
slipperiness	0.0	100.0
hardness	0.0	100.0
elasticity	0.0	2.0
height	0.0	0.1

6.3.2 Terrains Parameters Determination

To determine a terrain type, one has to come up with the five parameters from table 6.2.

At first, the number of identifiable virtual terrain types needs to be determined. For purposes of this thesis, it has been decided to create **14 terrain types**. Their parameters (shown in table 6.3) have been set up intuitively, based on the AMOS II simulated behaviour. With respect to the qualities ranges from table 6.2, the values have been normed to (0, 1).

Table 6.3: Virtual terrain types parameters.

#	*terrain title*	*roughness*	*slipperiness*	*hardness*	*elasticity*	*height*
1	**carpet**	0.3	0.0	0.4	0.15	0.2
2	**concrete**	1.0	0.0	1.0	0.0	0.0
3	**foam**	0.5	0.0	0.0	1.0	0.7
4	**grass**	0.5	0.0	0.3	0.3	0.5
5	**gravel**	0.7	0.001	1.0	0.0	0.3
6	**ice**	0.0	1.0	1.0	0.0	0.0
7	**mud**	0.05	0.05	0.005	0.25	0.2
8	**plastic**	0.1	0.02	0.6	0.5	0.0
9	**rock**	1.0	0.0	1.0	0.0	1.0
10	**rubber**	0.8	0.0	0.8	1.0	0.0
11	**sand**	0.1	0.001	0.3	0.0	0.2
12	**snow**	0.0	0.8	0.2	0.0	0.2
13	**swamp**	0.0	0.05	0.0	0.0	1.0
14	**wood**	0.6	0.0	0.8	0.1	0.2

Colors linked to the terrains in table 6.3 are used in the simulation as well as in the figures in Results section.

6.3.3 Analysis of Chosen Parameters

In general, proper data preparation is an important part of classification tasks, hence a brief analysis is presented.

The goal is to imitate real terrains as authentically as possible and at the same time to generate such terrains, which are clearly distinguishable from each other. The more two terrains differ, the better classification results are expected.

Having five terrain qualities calls for a 5-D space, which is difficult to illustrate or even imagine. Therefore, formula 6.5 is used to compute a similarity factor of two terrain types (the five qualities are listed in table 6.2 and table 6.3).

$$SF_{t_1,t_2} = \sum_{i=1}^{5} |quality(i, t_1) - quality(i, t_2)| \tag{6.5}$$

Naturally, equation 6.5 ends up with $SF_{similar} = 0.0$ for two terrains with exactly same parameters and $SF_{different} = 5.0$ for two terrains differing most possibly.

The following figure (6.9) shows the variability (similarity factors) of generated terrains.

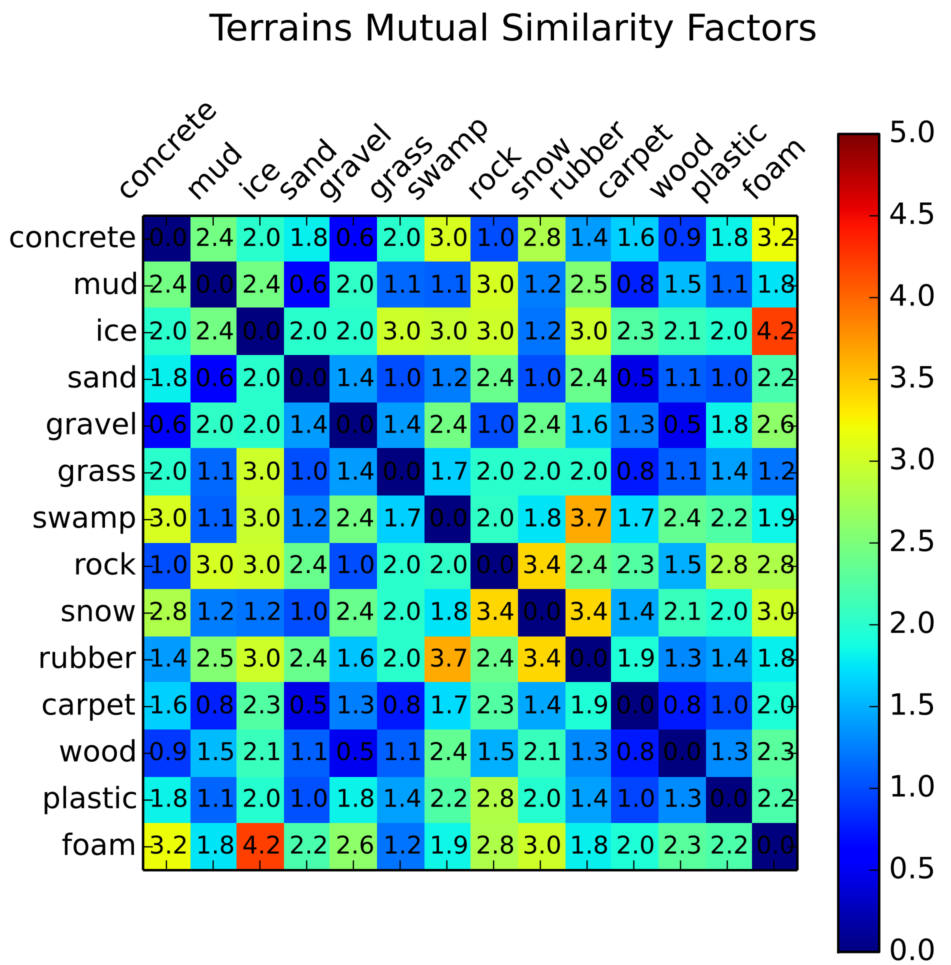

FIGURE 6.9: Variability of generated terrain types.

Based on fig. 6.9, one can say that foam is very different from ice or, for instance, sand is quite similar to mud. The surfaces are virtually generated and their authenticity has not been verified.

6.3.4 Terrain Noise

Generally, using simulation data for the very first research steps brings many benefits and it is usually the right way to start. However, the real world is always different from the simulated alternative and these disparities may influence the results significantly.

In this case, there are some virtually created terrain types based on five qualities (section 6.3.1). These parameters have been set up basically by a guess, intuitively. Therefore, one should assume that the real terrains might be distinct from the virtual ones in some ways.

Secondly, if there is a terrain defined as grass for instance, this definition cannot be general on no account. There are many types of grass and they differ from each other at least in the reffered qualities.

Consequently, there are some lines of the code added to *main.cpp* (see A2) enabling to noise the parameters shown in table 6.3. The following box (6.3) shows how it is done.

Part of Code 6.3: Adding terrain noise in main.cpp

```
terrain_roughness += fRand(-10.0*std_vol, 10.0*std_vol);
terrain_slip  += fRand(-10.0*std_vol, 10.0*std_vol);
terrain_hardness += fRand(-100.0*std_vol, 100.0*std_vol);
terrain_elasticity += fRand(-2.0*std_vol, 2.0*std_vol);
terrain_height += fRand(-0.1*std_vol, 0.1*std_vol);

// limits : params can not be negative
terrain_roughness = max(0.0, terrain_roughness);
terrain_slip = max(0.0, terrain_slip);
terrain_hardness = max(0.0, terrain_hardness);
terrain_elasticity = max(0.0, terrain_elasticity);
terrain_height = max(0.0, terrain_height);
```

The *std_vol* variable comes as an argument to *main.cpp*. It is meant to be a standard deviation of Gaussian noise in percentage. Hence, this percentage is then multiplied with the corresponding quality range and passed to the *fRand()* function in order to generate a random float number from the created range with zero mean.

The function generating the random number uses the *normal (Gaussian) distribution* with a probability density function defined as:

$$p(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{6.6}$$

In this case the mean $\mu = 0$ and the standard deviation σ is defined by a corresponding range percentage. As the values at any pair of times are identically distributed and statistically independent (and hence uncorrelated) [Wik04], a **white Gaussian noise** is being generated thereby. Additionally, there is some limits checking as the parameters cannot take negative values.

In this manner, the terrain types parameters from table 6.3 can be noised, where the magnitude of noise influence is passed as a simulation argument.

6.4 Data Acquisition

At this point the simulation is set up and ready to be launched. There are **14** virtually created terrain types (defined in section 6.3, table 6.3) and **24** robot's proprioceptive sensors (described in section 6.2.1, table 6.1) available.

Predictably, the terrain types are assumed to be classification targets (classes). Therefore, some data needs to be generated for each of these classes. This data comes from the 24 proprioceptors and one needs to find a way how to form feature vectors (classification samples) out of it (section 6.5), which is one of the most essential parts of the process.

As it is later described in more detail, several sensors values in time need to be used to catch the robot's dynamics on various terrains. Therefore, to generate a single data example, the simulation must be run for a period of time. The optimal duration is not known yet, but besides this fact, one should start thinking of generating a sufficient amount of samples for classification at this point.

The very simple way might be to let the robot walk for a long period of time and then just to cut the signals coming from sensors into many samples, based on an estimated timestep. The hitch of this approach is in initial conditions - they would become the same for every sample, which is not correct.

To keep the rightness, the simulator is launched several times in order to generate several samples for every terrain type. It has been decided to let the robot walk for **10** seconds each time. In combination with the simulation settings (see section 6.2.2), this implies **100** values for every sensor and for every simulation run - which should be more than enough.

For illustration, some data gathered from sensor *ATRf* when the robot was walking on a *concrete* surface for approximately 10 seconds is shown in fig. 6.10.

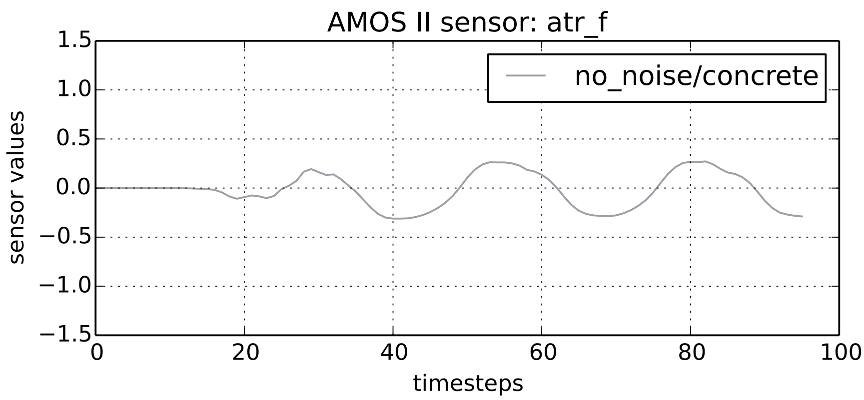

FIGURE 6.10: Data example: ATRf, concrete, 10 seconds

The *no_noise* indication in the figure legend reffers to section 6.3.4. An optimal starndard deviation value of the additive Gaussian noise is not known.

Therefore some data for several values of this parameter has been generated. The simulation has been gradually run for:

- $\sigma_p = 0.0$ (no noise)
- $\sigma_p = 0.01$ (noise 1%)
- $\sigma_p = 0.03$ (noise 3%)
- $\sigma_p = 0.05$ (noise 5%)
- $\sigma_p = 0.1$ (noise 10%)
- $\sigma_p = 0.2$ (noise 20%)

The σ_p is a standard deviation percentage, as shown in part of code 6.3, this σ_p is applied on qualities ranges and so corresponding standard deviation values σ_i are computed.

It is always recommended to store rough data before some processing, hence the simulator creates *.txt* files of structure symbolized in part of code 6.4 (with the reference to sensors shortcuts in table 6.1).

PART OF CODE 6.4: Rough sensory data files structure

```
timestep_001;ATRf;ATRm;ATRh;ATLf;...;FRh;FLf;FLm;FLh
timestep_002;ATRf;ATRm;ATRh;ATLf;...;FRh;FLf;FLm;FLh
...
timestep_100;ATRf;ATRm;ATRh;ATLf;...;FRh;FLf;FLm;FLh
```

There is a *.txt* file of this structure for every single simulation run in the *root/data/* directory (see appendix A2).

All the data files are generated by a script called *generate_txt_data.py* (A2). This script takes several arguments, like the number of jobs (simulation runs), terrain types involved or the terrain noise *std* (σ_p). Then a loop based on these parameters starts, where the simulation is launched and stopped after ten seconds each iteration. This is performed by calling a bash command (since the simulation is *.cpp* based) and then killing the called process from python. The corresponding *.txt* file is saved after each iteration by the simulation and then copied by the python script to a corresponding folder in *root/data/*.

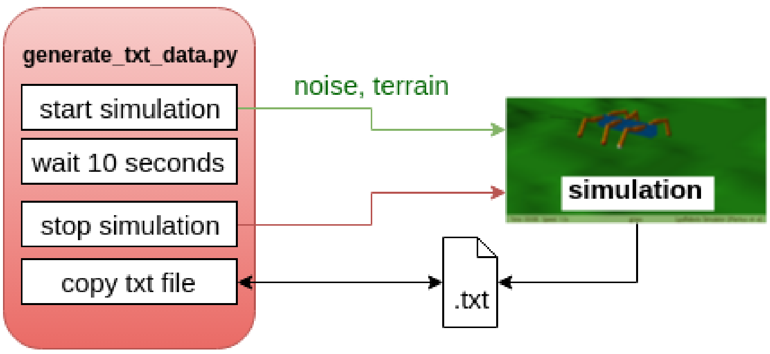

FIGURE 6.11: The process of data acquisition from the simulation.

In this manner, *.txt* files for all terrains and all mentioned σ_p are saved into a structure illustrated on fig. 6.12. Each *.txt* file contains approximately 100 lines, one for each simulation step (as shown in part of code 6.4). Every line then contains values of the 24 proprioceptive sensors.

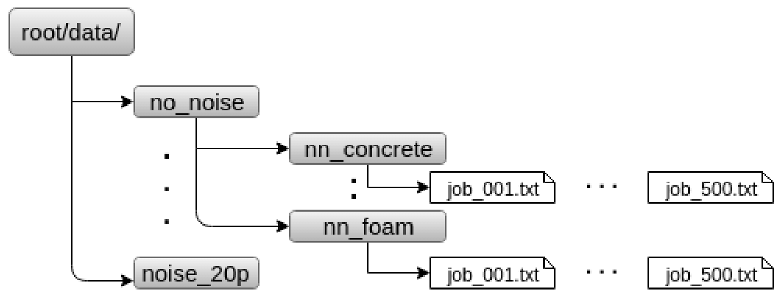

FIGURE 6.12: The structure of rough data directory.

Right after the data generation, a script called *clean_txt_data.py* (A2) is used to check the created *.txt* files. As it takes a long time to generate all the data, sometimes the simulation fails and the files are incomplete. Hence the script checks whether there are enough timesteps (at least more than 95) and also if the steps are not messed. Files that fail during the inspection are removed. As fig. 6.12 reveals, there are **500** *.txt* files for every *noise/terrain* configuration. This allows creating datasets of 500 samples per class.

6.5 Building a Feature Vector

So far, the hexapod simulation has been introduced, virtual terrain types defined and the simulator has been run on these terrains and for various values of terrain noise power. All the data has been acquired from the simulation as *.txt* files. Hence the simulation is not needed anymore and only the gathered data are used for the following processing.

Classification tasks are generally based on datasets consisting of samples and corresponding targets. The samples need to be represented in a numerical way in order to be processed by a computer and its appropriate algorithms. In machine learning, this numerical representation of an object is called *a feature vector*, an n-dimensional vector of numerical values. This section is devoted to building a feature vector out of the data gathered from proprioceptive sensors.

This part of the process is crucial as the way of feature vector compilation can influence classification results a lot. Information loss or redundant structures are quite frequent mistakes here. Therefore, as the optimal structure is not known, several possibilities are tested again and therefore some new global process parameters appear at this point (mentioned already in section 6.1).

For this particular problem, the task is to form one feature vector out of the content of one *.txt* file (got in section 6.4), as each of these files contains data for one sample (see fig. 6.13).

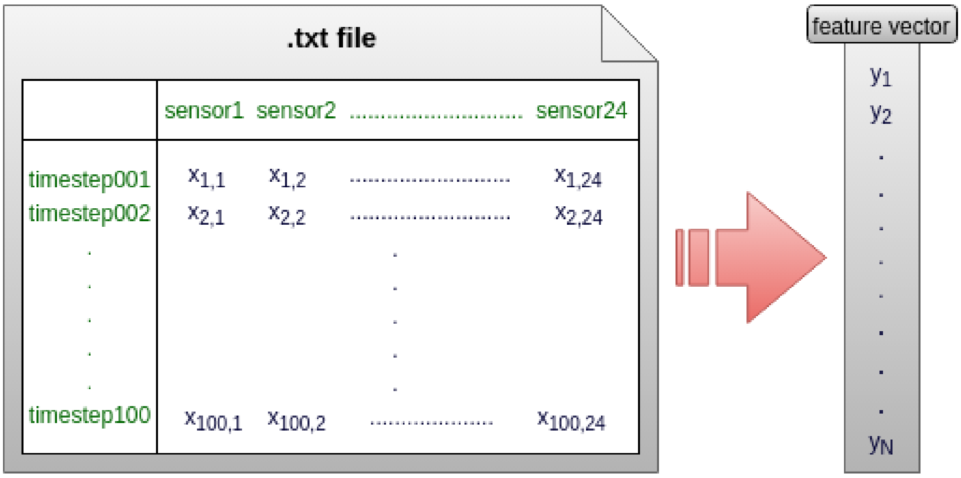

FIGURE 6.13: Forming a feature vector out of a data file.

At this point, one may ask for the reason of using several timesteps for creating one sample. It is assumed that a proper terrain classification using proprioceptors at one moment in time is at least difficult, if not impossible. Therefore the idea is to let the robot walk for a while and take down the dynamics of the sensors. Of course, the more timesteps are used for one sample, the more time the classification takes. Because of these arguments the number of timesteps is left as a global process parameter and it is a subject for later discussion.

Sensors selection defines another global parameter coming out of this section. The anticipation is that the feature vector becomes redundant using all of the 24 sensors, as many of them might behave similarly. However, for now all of them are used to show how the feature vector is built and it is also left for later discussion.

Parameters coming out of feature vector compilation and left for later discussion:

- number of timesteps used to build one feature vector (one sample)
- sensors involved in classification

Now, with reference to fig. 6.13, the question is how to transform the **2D** data from *.txt* files into **1D** vectors. The idea is to fix the *timesteps* parameter and arrange the columns of the matrix into one vector. This implies having data from all sensors one by one next to each other and forming one feature vector together.

In fig. 6.14 an example of this kind of forming is shown. For illustration there is just one terrain type (*concrete*) involved. The number of timesteps is set to 40 and all 24 sensors are used, hence a feature vector of length 960 is gained. The corresponding sensors shortcuts (see table 6.1) are added to the x-axis annotation. The 18 angle sensors are followed by the 6 foot contact sensors.

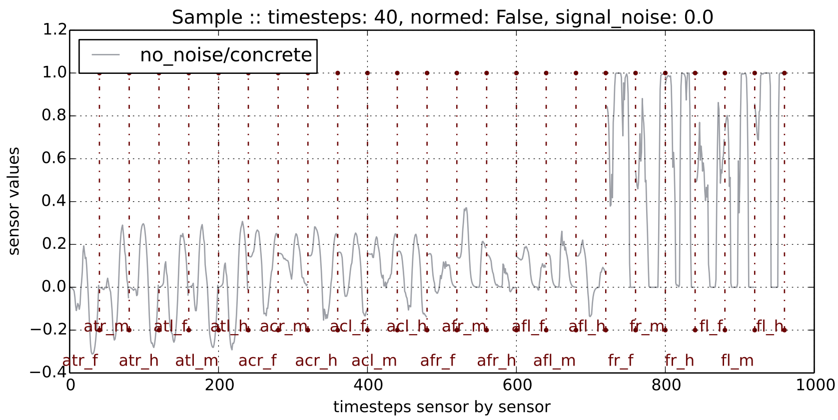

FIGURE 6.14: Example of feature vector building.

6.5.1 Normalization

It is a good manner to keep the data normed - mapped to $[0.0, 1.0]$ interval. The default range of foot contact sensors is already set to $[0.0, 1.0]$, so there is nothing to change. For the angle sensors, the following approach is used to map the data.

PART OF CODE 6.5: Data normalization

```
def norm_signal(s, a_min, a_max):
  l = [min(max(float((x-a_min))/(a_max-a_min), 0), 1) for x in s]
  return l
```

The bounds (*a_min* and *a_max* used in part of code 6.5) are defined by default sensors ranges (listed in table 6.1). Also a $[0, 1]$ interval overflow checking is added and values are adjusted if needed. This is a cover for the case ranges from table 6.1 were not accurate. The following figure (6.15) shows a normed feature vector example. The influence of normalization on classification results is another subject for the discussion.

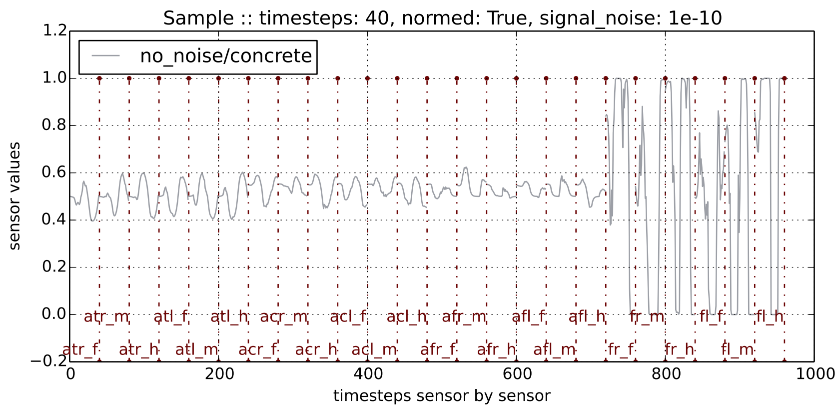

FIGURE 6.15: Example of feature vector - normed.

6.5.2 Signal Noise

In section 6.3.4 a few general reasons for noising simulation data are disscused. In that case an additive Gaussian noise is used to make the terrains definitions (from table 6.3) more complex.

For similar reasons a signal noise is added to the sensory data. In reality the mechanical sensors might shake, be influenced by environmental conditions or simply may not work as well as expected, while the data coming from the simulated sensors are always deterministic.

Right after the data normalization, a white Gaussian noise is added to the normed feature vectors. This time it is performed in Python, using the *random.normal()* function of *numpy* library to get the *normal distribution* with zero mean (part of code 6.6).

PART OF CODE 6.6: Adding signal noise in python

```
def add_signal_noise(signal):
  noise = np.random.normal(loc=0, scale=STD, size=len(signal))
  return [x+n for x, n in zip(signal, noise)]
```

Also in this case, it is difficult to estimate an optimal signal noise power (STD in part of code 6.6). Therefore it is left as another global process parameter and its influence is discussed in the results part. It is defined as a percentage of the $[0.0, 1.0]$ interval and as the signals are normed in advance, there is no need for another processing of this parameter.

6.6 Datasets Creation

Reffering to fig. 6.1, in this section the *create_terrains_dataset.py* script is introduced. At this part of the process all the data files are ready and also the form of feature vectors is determined. The task is to transform all the data into so called datasets.

There are usually three sets of data used for classification tasks - training, validation and testing data. These three sets must be disjunctive, meaning they cannot have a single element in common. All these three sets together form a dataset.

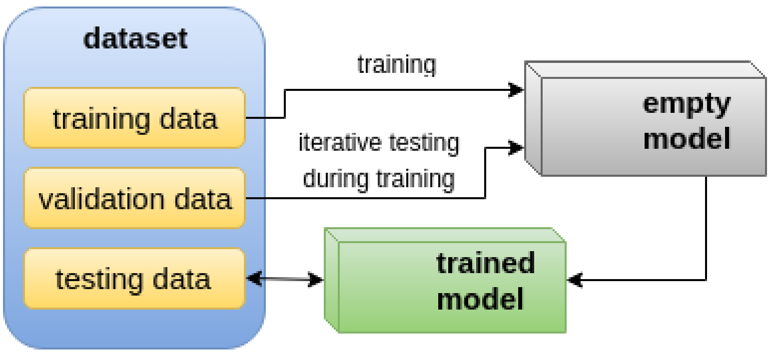

FIGURE 6.16: Three sets of data in a dataset.

Each set of data consists of samples and targets (classes). The samples are represented by normed feature vectors (section 6.5) - lists of numerical floating point values from $[0.0, 1.0]$ interval. The targets, in this case, match the names of virtually created terrain types (listed in table 6.3). Every sample must be uniformly assigned to precisely one target.

Once there are two ordered lists - a list of samples and a corresponding list of targets, these lists are split into the three sets shown in fig. 6.16. The script takes an argument called *data_split_ratio* defining the proportions among the sets sizes. Defaultly the ratio is set to generate 80% training, 10% validation and 10% of testing data.

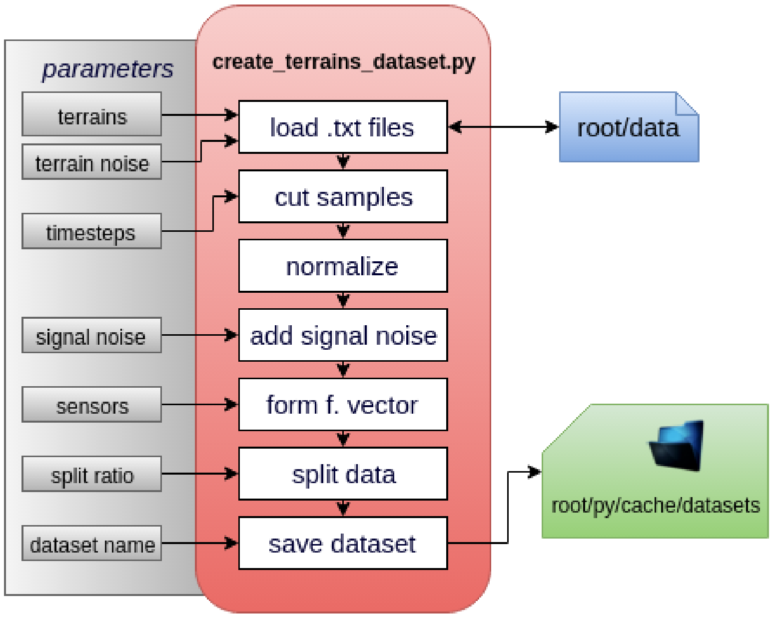

FIGURE 6.17: *create_terrains_dataset.py* : script workflow

The workflow happening inside the *create_terrains_dataset.py* script is illustrated in fig. 6.17. During the overall process description in previous sections, some global process parameters have been collected. These configurations are now passed as arguments to this script and therefore several datasets of various properties can be generated.

The datasets files are saved in directory *root/py/cache/datasets/amos_terrains_sim/* (see A2). Their structure is based on a powerful serializing and de-serializing Python algorithm implemented under a package called *pickle (cPickle)*. On the same basis a package called *shelve* is used to represent a dataset as a dictionary-link object. The files are saved with the *.ds* suffix.

The list of all generated datasets can be found in **??**. These datasets and influence of individual parameters are evaluated in the results section (**??**).

6.7 Training and Classification

Having a dataset enables to train a classifier, a machine learning tool that is able to learn some behavior on one part of some data (training and validation) and then perform similarly on another "never seen" part of the data (testing) - illustrated in fig. 6.16.

There are many classification methods differing in mathematical backgrounds and each of them has some pros and cons on various types of data. However, all of them have some general functionalities that comply with some kind of convention. For instance, there are at least two functions that every classifier should be capable of:

fit() : Fitting some data to a model. This function usually takes training samples and their targets as arguments. Additiionally, it can take some validation data and/or learning parameters. Then a model is trained.

predict() : This function is then called when a model is already trained. It takes one or more samples of testing data and returns the predicted target(s).

This convention enables testing different classification approaches on the same data in the same way. Therefore also the implemented network library *kitt_nn* (see chapter 4) provides these functions and is capable of working with datasets of the same structure as the public *.py* classifiers (discussed in sections 6.7.3 and 6.7.5).

In the overall process diagram (fig. 6.1) two scripts for training neural networks are included. The first one, *kitt_train.py*, uses the implemented network library *kitt_nn* (chapter 4) and the script called *sknn_train.py* uses a public library *Scikit-neuralnetwork*, which is described in section 6.7.3. It is important that these two scripts use the same workflow, which is illustrated in fig. 6.18.

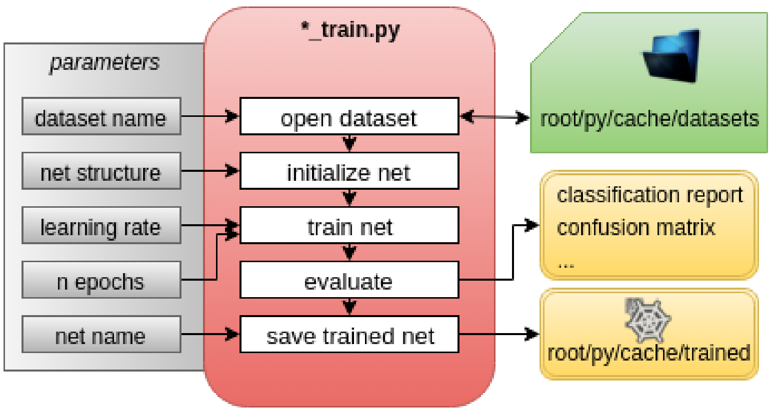

FIGURE 6.18: *kitt_train.py* and *sknn_train.py* : scripts workflow

The scripts take several arguments (firstly listed in section 6.1) that differentiate the final trained networks and their performances. The first one is the dataset that the network is trained on. This parameter brings its own configuration (see its input parameters in fig. 6.17) and so its setting parametrizes the classifier as well.

Next, one needs to define the network initial structure in sense of number of hidden layers and number of neurons in each of these layers. The input and output layers are determined by the dataset. There are many parameters to be defined for learning like *batch size*, *initial random state* etc. In this reasearch, only the learning rate and the number of epochs are used as training parameters. The learning process follows the implemented back-propagation algorithm described in section 4.2.

Finally, the trained network needs a file name, as it is saved the same way as the datasets (see section 6.6) - using the *pickle (cPickle)* package, just with the *.net* suffix.

6.7.1 Evaluation Methods

A trained network is evaluated on testing data. This evaluation provides a set of the most important classification metrics [Ped+11].

accuracy : the set of labels predicted for a sample must exactly match the corresponding set of true labels

precision : ability of the classifier not to label as positive a sample that is negative

recall : the ability of the classifier to find all the positive samples

F1 score is interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. Formula:

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{6.7}$$

confusion matrix : a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j.

6.7.2 Terrain Classification using Pruned Nets

As the overall process diagram (fig. 6.1) shows, the developed network pruning algorithm (chapter 5) is tested on the terrain datasets. The approach has been already described.

Evaluation in... **TODO:** describe the process here and reffer the results after they are gathered.

6.7.3 Scikit-learn Neural Network Library

In order to verify the functionality of implemented neural network library (chapter 4), a provided public library is used. As the official description says [CS15], this library implements multi-layer perceptrons as a wrapper for the powerful *pylearn2* library that is compatible with *scikit-learn* for a more user-friendly and Pythonic interface.

This step has been considered with the aim to test another implementation of the learning algorithm rather than to obtain better classification results. As the only learning parameters are the *net structure*, the *learning rate* and the *number of epochs*, some other default parameters of the tested network are shown in part of code 6.7.

PART OF CODE 6.7: Sknn classifier specification [CS15]

```
class sknn.mlp.Classifier(layers, warning=None, parameters=None,
random_state=None, learning_rule=u'sgd', learning_rate=0.01,
learning_momentum=0.9, normalize=None, regularize=None,
weight_decay=None, dropout_rate=None, batch_size=1, n_iter=None,
n_stable=10, f_stable=0.001,  valid_set=None, valid_size=0.0,
loss_type=None, callback=None, debug=False,  verbose=None)
```

6.7.4 Searching for Optimal Configuration (Grid Search)

TODO : describe here how the best parameters have been found using GridSearch

6.7.5 Other Classifiers

TODO : describe here how other classifiers have been tested and reffer to the results part

SVM, k-NN, RandomForest

Chapter 7

Experimental Evaluation

The account of the research should be presented in a manner suitable for the field. It should be complete, systematic, and sufficiently detailed to enable a reader to understand how the data were gathered and how to apply similar methods in another study. Notation and formatting must be consistent throughout the thesis, including units of measure, abbreviations, and the numbering scheme for tables, figures, footnotes, and citations. One or more chapters may consist of material published (or submitted for publication) elsewhere. See “Including Published Material in a Thesis or Dissertation” for details.

7.1 Network Implementation Results

7.2 Pruning Algorithm Results

7.3 Terrain Processing Results

7.3.1 Gathered Data

Data by Sensors

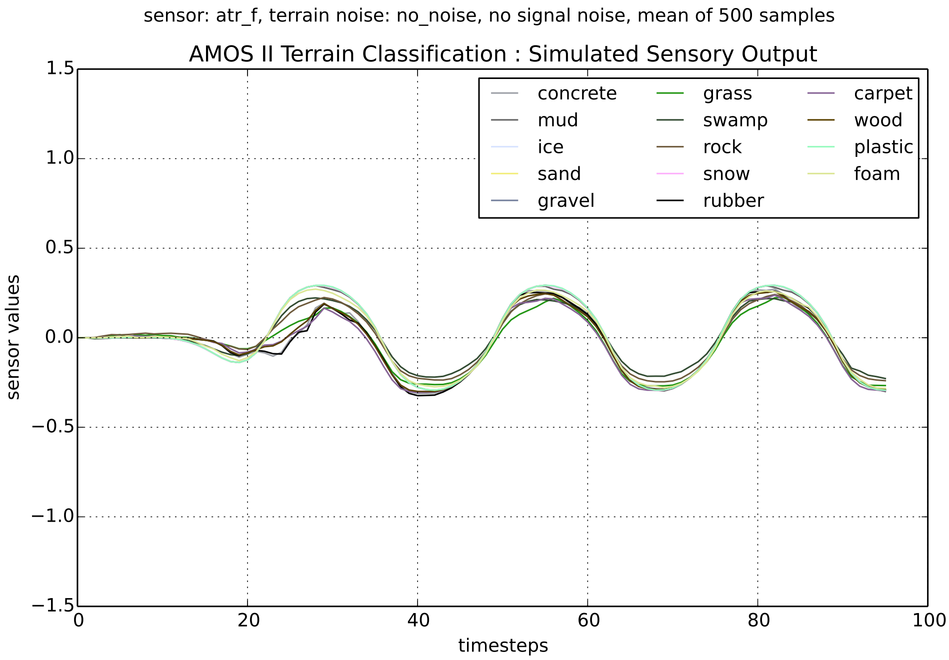

FIGURE 7.1: Sensor ATRf : mean of 500 samples, 14 terrains

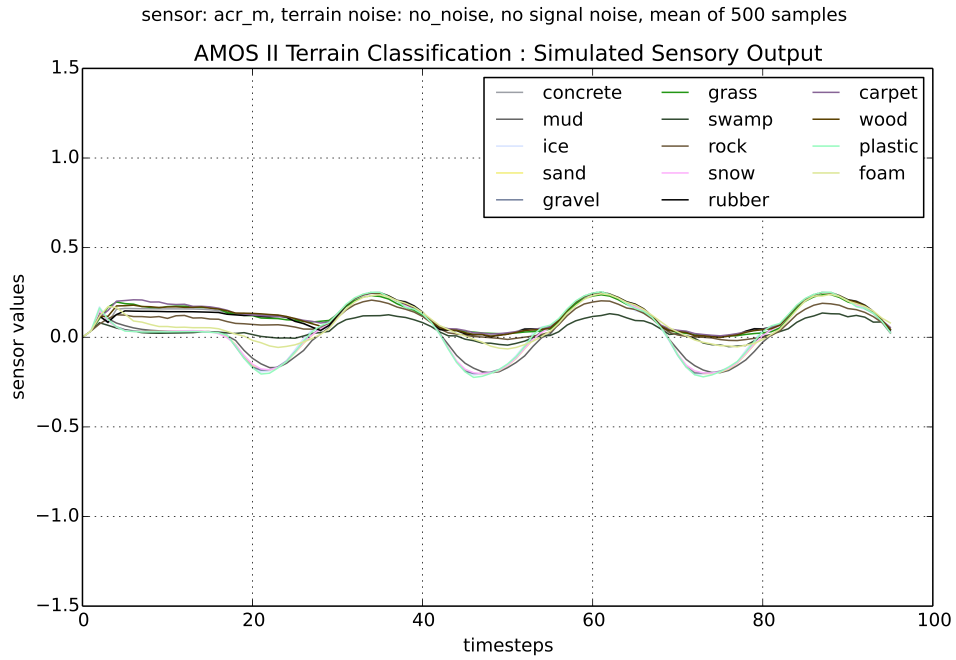

FIGURE 7.2: Sensor ACRm : mean of 500 samples, 14 terrains

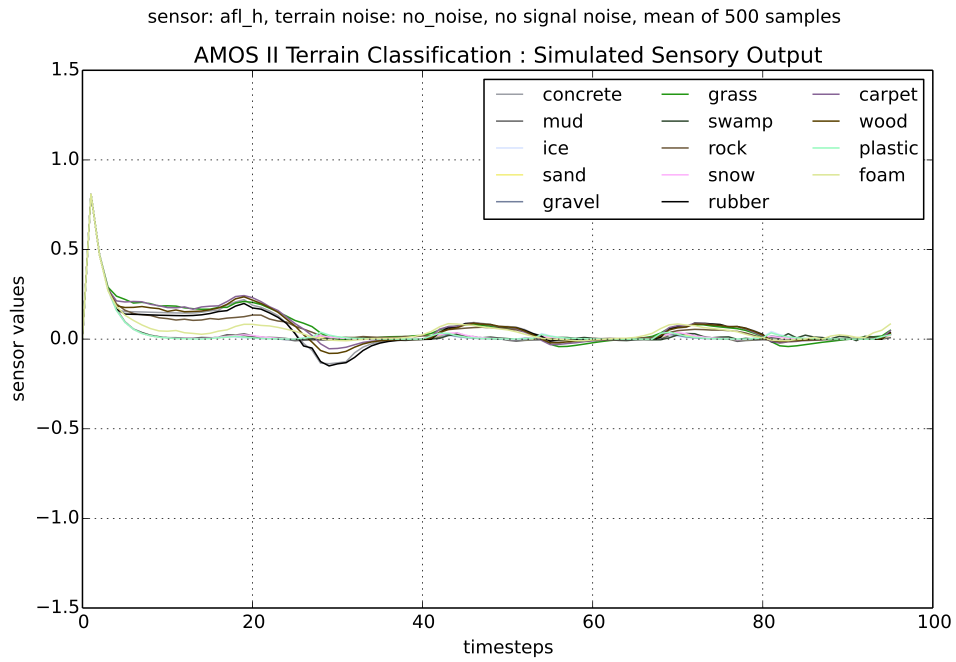

FIGURE 7.3: Sensor AFLh : mean of 500 samples, 14 terrains

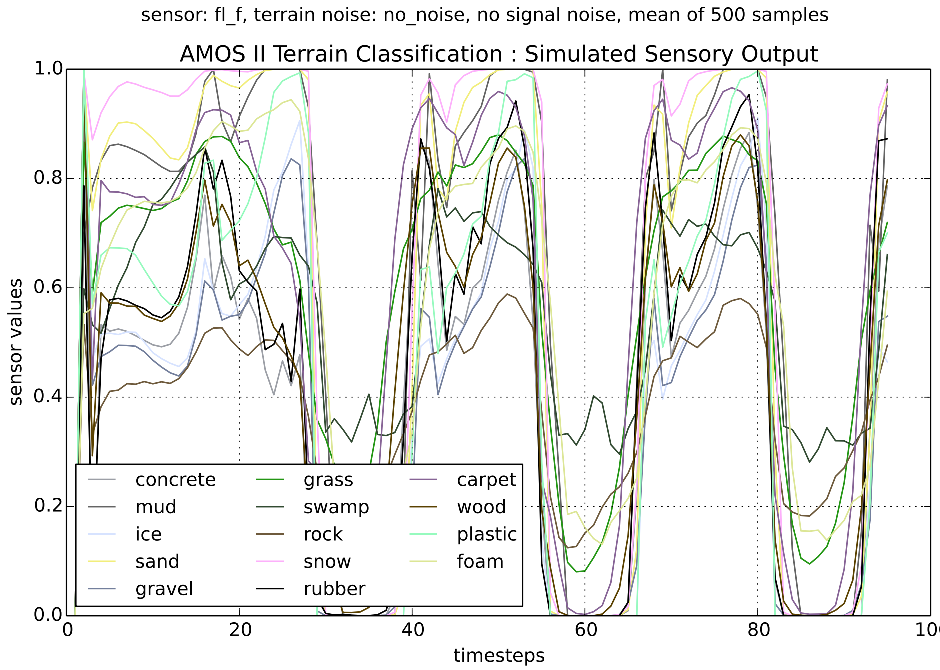

FIGURE 7.4: Sensor FLf : mean of 500 samples, 14 terrains

Built Feature Vector

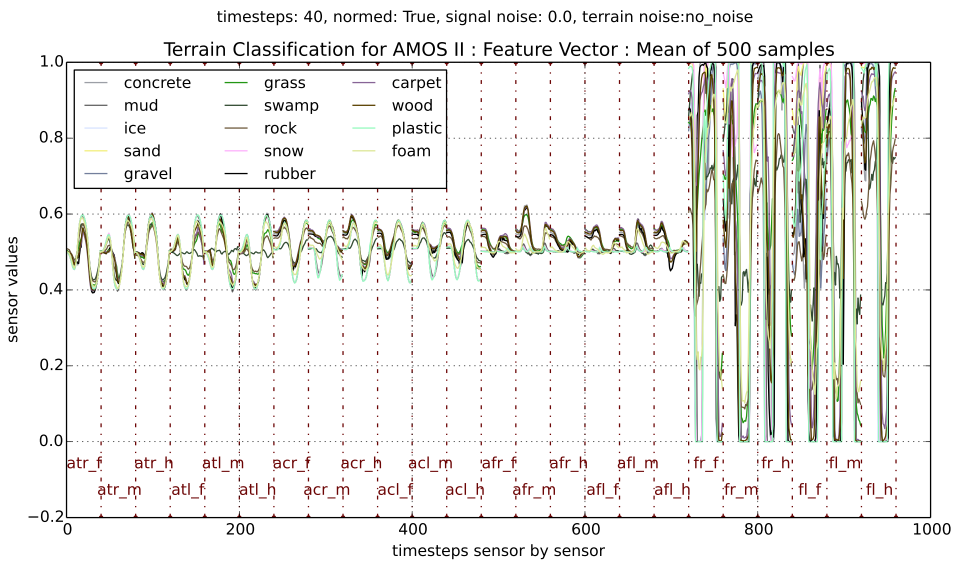

FIGURE 7.5: Feature Vector : mean of 500 samples, 14 terrains, no noise, 40 timesteps

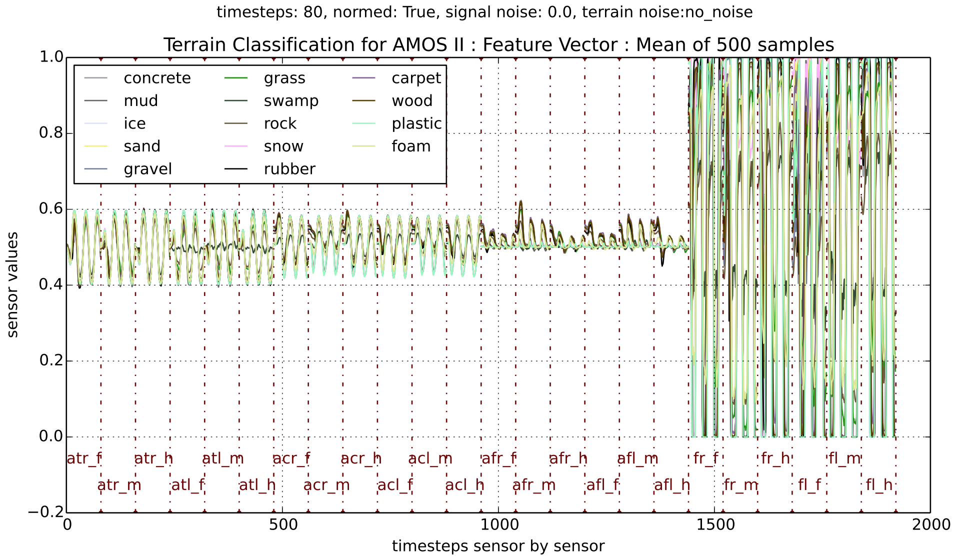

FIGURE 7.6: Feature Vector : mean of 500 samples, 14 terrains, no noise, 80 timesteps

Terrain Noise Influence

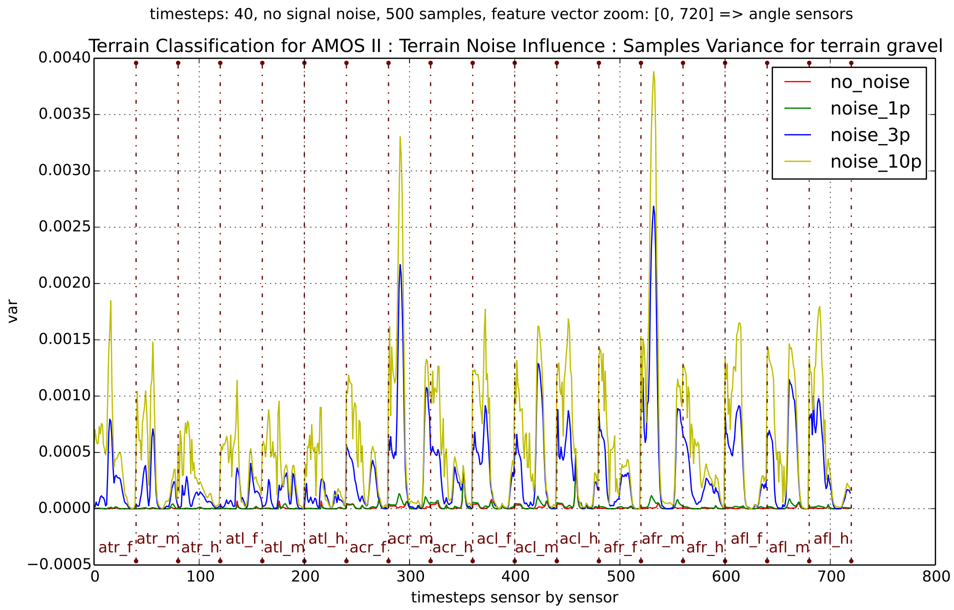

FIGURE 7.7: Terrain Noise Analysis (samples variance): 500 samples, terrain gravel, angle sensors (feature vector [0:720] for 40 timesteps)

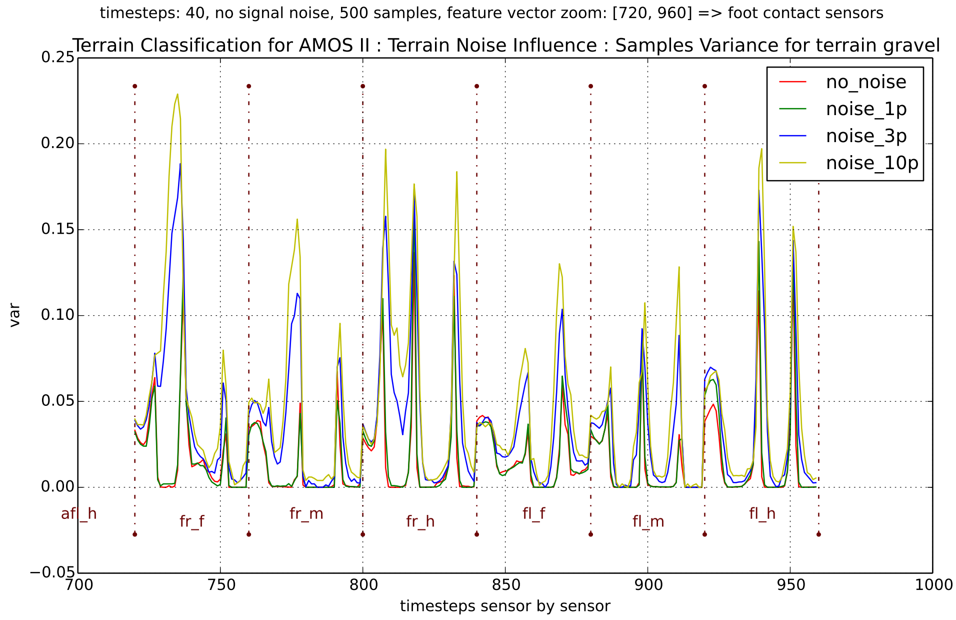

FIGURE 7.8: Terrain Noise Analysis (samples variance): 500 samples, terrain gravel, foot contact sensors (feature vector [720:960] for 40 timesteps)

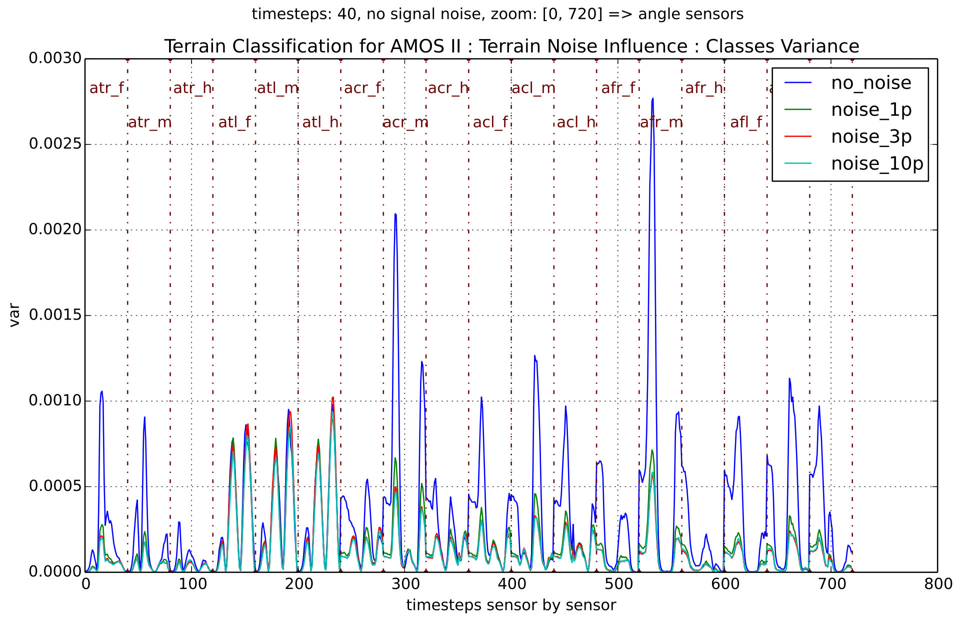

FIGURE 7.9: Terrain Noise Analysis (classes variance): means of 500 samples, 14 terrains, angle sensors

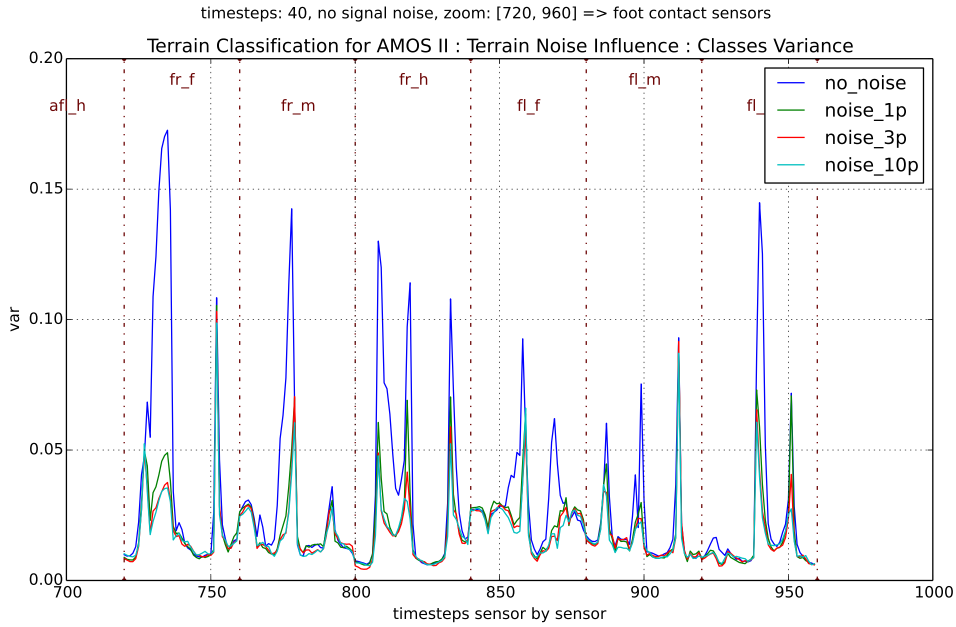

FIGURE 7.10: Terrain Noise Analysis (classes variance): means of 500 samples, 14 terrains, foot contact sensors

Signal Noise Influence

7.3.2 Generated Datasets

table

7.3.3 Classification results

7.3.4 Final Configuration

evaluation (tables and figures) of classification:

- various terrain noise standard deviation values
- various signal noise standard deviation values
- various sensors on network input (only foot, only angle...)
- various timesteps used as one sample (-> time needed for detection)
- various number of detected terrains as outputs
- various network structures
- various training parameters (epochs, learning rate, batch size...)

evaluation of neural nets as a classifier:

- comparison to other classifiers on the same data, classifiers are ready provided by sknn library

evaluation of proprioception sensing against other methods (visual, haptic, laser...):

- comparison to the results from the literature

evaluation of the pruning algorithm:

- various starting structures, ends up with the same minimal-optimal structure?
- various noise types, same minimal structure?
- speed comparisons of the fully-connected vs. pruned structure
- further analysis:
  - which sensors are redundant/crucial
  - which sensors are important for which terrain
  - comments on the minimal structure and benefits of having it

10-15 pages (many figures, tables)

Chapter 8

Discussion

Chapter 9

Conclussion and Outlook

In this section the student must demonstrate his/her mastery of the field and describe the work's overall contribution to the broader discipline in context. A strong conclusion includes the following:

Conclusions regarding the goals or hypotheses presented in the Introduction, Reflective analysis of the research and its conclusions in light of current knowledge in the field, Comments on the significance and contribution of the research reported, Comments on strengths and limitations of the research, Discussion of any potential applications of the research findings, and A description of possible future research directions, drawing on the work reported. A submission's success in addressing the expectations above is appropriately judged by an expert in the relevant discipline. Students should rely on their research supervisors and committee members for guidance. Doctoral students should also take into account the expectations articulated in the University's "Instructions for Preparing the External Examiner's Report".

2-3 pages

All references:

[Zen+13] and [Kes+12] and [XWM14] and [MK10] and [Coy10] and [Hoe+10] and [Ahm15] and [Ord+13] and [Ber+12] and [Ree93] and [SK07] and [Bel11]

Bibliography

[Ree93] R. Reed. "Pruning Algorithms - A Survey". In: *IEEE Transactions on Neural Networks (Volume:4 , Issue: 5)* (Sept. 1993), pp. 740–747. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=248452.

[Wik04] Wikipedia. *Plagiarism — Wikipedia, The Free Encyclopedia*. [Online; accessed 04-May-2016]. 2004. URL: \url{http://en.wikipedia.org/w/index.php?title=Plagiarism&oldid=5139350}.

[SK07] D. Spenneberg and F. Kirchner. *The Bio-Inspired SCORPION Robot: Design, Control and Lessons Learned, Climbing and Walking Robots: towards New Applications*. ISBN 978-3-902613-16-5, 2007.

[Coy10] E. Coyle. "Fundamentals and Methods of Terrain Classification Using Proprioceptive Sensors". PhD thesis. Florida State University Tallahassee, 2010.

[Hoe+10] M. A. Hoepflinger et al. "Haptic terrain classification for legged robots". In: *Robotics and Automation (ICRA), IEEE International Conference* 3 (May 2010), pp. 2828–2833. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5509309.

[MK10] W. Mou and A. Kleiner. "Online learning terrain classification for adaptive velocity control". In: *Safety Security and Rescue Robotics* 26 (July 2010), pp. 1–7. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5981563.

[Bel11] Dominik Belter. "Gait control of the six-legged robot on a rough terrain using computational intelligence learning and optimization methods". PhD thesis. Poznan University of Technology, Nov. 2011.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Ber+12] F. L. G. Bermudez et al. "Performance analysis and terrain classification for a legged robot over rough terrain". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* 7 (Dec. 2012). URL: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6386243.

[Kes+12] P. Kesper et al. "Obstacle-Gap Detection and Terrain Classification of Walking Robots based on a 2D Laser Range Finder". In: *Nature-inspired Mobile Robotics* (2012), pp. 419–426. URL: http://manoonpong.com/paper/2013/CLAWAR2013_Kesper.pdf.

[Ord+13] C. Ordonez et al. “Terrain identification for RHex-type robots”. In: *Unmanned Systems Technology XV* 17 (May 2013). URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1689675.

[Zen+13] S. Zenker et al. “Visual terrain classification for selecting energy efficient gaits of a hexapod robot”. In: *International Conference on Advanced Intelligent Mechatronics* 12 (July 2013), pp. 577–584. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6584154&tag=1.

[XWM14] X. Xiong, F. Worgotter, and P. Manoonpong. “Neuromechanical control for hexapedal robot walking on challenging surfaces and surface classification”. In: *Robotics and Autonomous Systems* 7 (Aug. 2014), pp. 1777–1790. URL: www.elsevier.com/locate/robot.

[Ahm15] Mohammed Nour Abdel Gwad Ahmed. “An Intelligent Architecture for Legged Robot Terrain Classification Using Proprioceptive and Exteroceptive Data”. PhD thesis. University of Bremen, June 2015.

[CS15] Alex J. Champandard and Spyridon Samothrakis. *sknn: Deep Neural Networks without the Learning Cliff.* [Online; accessed 06-May-2016; nucl.ai Conference 2015]. 2015. URL: \url{http://scikit-neuralnetwork.readthedocs.io/en/latest/}.

[Man] Poramate Manoonpong. “Adaptive Embodied Locomotion Control Systems”. Lecutre 3 - page 133 - Tripod Gait.

[Misa] *Open-source multi sensori-motor robotic platform AMOS II.* http://manoonpong.com/AMOSII.html.

[Misb] *PPM Format Specification.* http://netpbm.sourceforge.net/doc/ppm.html. Updated: 02 November 2013.

[Misc] *Research Network for Self-Organization of Robot Behavior.* http://robot.informatik.uni-leipzig.de/software/. last modified: 06. July 2015.

Appendix A1

Working Directory Structure

the mt_folder

Appendix A2

Code Documentation

Write your Appendix content here.

Appendices must be limited to supporting material genuinely subsidiary to the main argument of the work. They must only include material that is referred to in the document.

Material suitable for inclusion in appendices includes the following:

Additional details of methodology and/or data Diagrams of specialized equipment developed Copies of questionnaires or surveys used in the research Do not include copies of the Ethics Certificates in the Appendices.

Appendix A3

Detailed Results

all classification reports, confusion matrices and so on...