1 Implementation

1.1 Tools

The most important tools and libraries used in our work were:

- ROS Melodic
- Numpy
- Matplotlib
- Pandas
- OpenCV
- PyTorch
- FastAI
- imgaug
- Blender

The framework was entirely developed on Ubuntu 18.10 with Python 3.6.

1.1.1 ROS Melodic

The Robot Operating System (ROS) [**ROS**] is a flexible framework for writing robot software. It is *de facto* the industry and research standard framework for robotics due to its simple yes effective inferface that facilitates the task of creating robust and complex robot behavior regardless of the platforms. ROS work by generate a peer-to-peer connection where each *node* is to communicate between the others by exposing sockets endpoints to stream data called *topics*.

Each *node* can subscribe to received the incoming messages or publish new data on a specific *topic*. In our case, *Krock* exposes different topics, for example `/pose`, in which we can subscribe in order to get the real time informations about the state of the robot. Unfortunately, ROS does not natively support Python3, so we had to compile it by hand. Since it was not difficult and time consuming operation we decided to share the ready-to-go binaries as docker image.

where should I place the link to docker

1.1.2 Numpy

Numpy is a fundamental packages for any scientific use. Thanks to its powerful N-dimensional array object with the sophisticated broadcasting functions, it is possible to express efficiently any matrix operation. We utilised *Numpy* manipulate matrices in an expressive and efficient way.

scrape some text from the previous section

1.1.3 Matplotlib

Matplotlib is a widely used Python 2D plotting library which generates high quality figures in a variety of hardcopy formats and interactive environments across platforms. It provides a similar functional interface to MATLAB and a deep ability to customise every region of the figure. Almost every figures made in this report were produced using Matplotlib. It is worth citing *seaborn* a data visualization library that we inglobate in our work-flow. It is based on Matplotlib and it provides a high-level interface for drawing attractive and informative statistical graphics.

1.1.4 Pandas

Pandas is a Python library providing fast, flexible, and expressive data structures in a tabular form. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Today, it one of the most flexible open source data manipulation tool available. Pandas is well suited for many different kinds of data such as handle tabular data with heterogeneously-typed columns, similar to SQL table or Excel spreadsheet, time series and matrices. It provides to two primary data structures, `Series` and `DataFrame` for representing 1 dimensional and 2 dimensional data respectively.

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

1.1.5 OpenCV

Open Source Computer Vision Library, OpenCV, is an open source computer vision library with a rich collection of highly optimized algorithms. It includes classic and state-of-the-art computer vision and machine learning methods applied in a wide array of tasks, such as object detection and face recognition. With a huge community of more than fourtyseven thousand people, the library is a perfect choice to handle image data. In our framework, OpenCV is used to pre and post-process the heightmaps and the patches.

1.1.6 PyTorch

PyTorch is Python open source deep learning framework. It provides Tensor computation (like NumPy) with strong GPU acceleration and Deep neural networks built on a tape-based autograd system. Due to its *Python-first* philosophy it has a deep integration into Python allows popular libraries and packages to be used, such as *OpenCV* or *Pillow*.

Due to its simply yet expressive and beautiful object oriented API it has been adopted be a huge number of researches and enthusiastics all around the world creating a flourishing community. Its main advantage over other mainstream frameworks such as TensorFlow are a cleaner API structure, better debugging, cite TF code shareability and enormous number of high quality packages. All the neural network proposed in this project are built using Pytorch.

1.1.7 FastAI

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

1.2 imgaug

Image augmentation (imgaug) is a python library to perform image augmenting operations on images. They provide a variety of methodologies, affine transformations, perspective transformations, contrast changes and gaussian noise, to build sophisticated pipelines. They support images, heatmaps, segmentation maps, masks, keypoints/landmarks, bounding boxes, polygons and line strings.

1.2.1 Blender

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. We used Blender to render some of the 3D terrain used to evaluate the trained model.

1.3 Data Gathering

1.4 Postprocessing

1.5 Estimator

1.5.1 Vanilla Model

1.5.2 Resnet

We decide to use a Residual Network, ResNet [**he2015deep**], variant. Residual network are deep convolutional networks constisting of many stacked R̈esidual Units ̈. Intuitively, the residual unit allows the input of a layer to contribuite to the next layer's input by beeing added to the current layer's output. Due to possible different features dimension, the input must go thought and identify map to make the addition possible. This allows a stronger gradient flows and mitigates the degradation problem. A R̈esidual Units ̈is composed by a two $3x3$ *Convolution*, *BatchNorm* [**ioffe2015batch**] and a *Relu* blocks. Formally, it is defined as:

$$\mathbf{y} = \mathcal{F}\left(\mathbf{x}, \{W_i\}\right) + h(\mathbf{x}) \tag{1}$$

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

img/implementation/estimator/resnet_block.png

Figure 1: *Resnet* building block.

When the input and output shapes mistmatch, the *identity map* is applyed to the input as a $3x3$ Convolution with a stride of 2 to mimic the polling operator. A single block is composed by a $3x3$ *Convolution*, *Batchnorm* and a *Relu* activation function.

Following the recent work of He et al. [**he2015identity**] we adopt *pre-activation* in each block. Easily, *Pre-activation* reverse the order of the operation in a block.

img/implementation/estimator/preactivation.png

Figure 2: *Preactivation*

Finally, we also used the *Squeeze and Excitation* (SE) module [**hu2017squeeze**]. A form of attention that weights the channel of each convolutional operation by learnable scaling factors. The next figure visualises the SE module.

`img/implementation/estimator/se.png`

Figure 3: *Preactivation*

We select *ResNet* architecure similar to the ones originally proposed to fit the *CIFAR10* [**cifar**] dataset. We used a initial features size of 64 with three blocks of depth $n = 1$. The final model has

add learnable parameters

add model picture