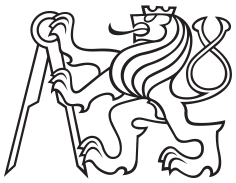
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Simulating realistic local 3D maps from geometric model

Dita Hollmannová

Supervisor: doc. Ing. Tomáš Svoboda Ph.D.
Field of study: Informatics and computer science
May 2018

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hollmannová Dita** Personal ID number: **456871**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Simulating Realistic Local 3D Maps from Geometric Models

Bachelor's thesis title in Czech:

Simulace realistických lokálních 3D map z geometrického modelu

Guidelines:

Learnable algorithms require large datasets. In outdoor robotics, these are difficult to obtain. A workaround typically includes simulations [1] coupled with a few roll-outs. Our algorithm for automatic control of robot morphology [2] uses digital elevation model (DEM) as the main input. The digital elevation model is created from depth measuring sensors like Lidar or RGB-D camera. In simulator, we can generate various terrain elements like steps, walls, staircases and use the 3D geometry for creating the DEM. The generated DEMs are used for training policies for the automatic robot control. However, data simulated from geometry only are often very different from the real ones. The main goal of the thesis is to alter the computer generated data in way that would resemble the real data. One of the possible ways may be utilzing the recent advancements in generative adversarial networks (GANs) [3] and to design an algoritm that learns a mapping model from data. The goal is not to simulate sensor per se. We want to treat the sensor and the mapping algorithm [4] as one sensor. It is, however, worth to mention that nobody applied GANs to 3D data yet.

Bibliography / sources:

[1] M. Pecka, K. Zimmermann, T. Svoboda. Fast Simulation of Vehicles with Non-deformable Tracks. In Intelligent Robots and Systems (IROS), 2017
[2] M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda. Controlling Robot Morphology from Incomplete Measurements. In IEEE Transactions on Industrial Electronics, Feb 2017, Vol 64, Issue: 2, pp. 1773-1782
[3] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, in IEEE International Conference on Computer Vision (ICCV), 2017
[4] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, Robot-Centric Elevation Mapping with Uncertainty Estimates, in International Conference on Climbing and Walking Robots (CLAWAR), 2014.

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Tomáš Svoboda, Ph.D., Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **10.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

doc. Ing. Tomáš Svoboda, Ph.D. Supervisor's signature	doc. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I would like to thank my supervisor, doc. Ing. Tomáš Svoboda Ph.D. for his guidance and advice.

I am very grateful to Mgr. Martin Pecka, who shared his tips on training the networks with me, assisted while recording the experiments, and always kindly answered all of my questions.

Also, my sincere thanks goes to doc. Ing. Karel Zimmermann, Ph.D, for inspiring new ideas that pushed this thesis forward.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within accordance with the methodical instructions for observing ethical principles in the preparation of university theses.

Prague, 25. 5 2018

Abstract

An outdoor terrain robot should be able to traverse difficult terrain autonomously. This is learned in a simulated environment to make the process faster and easier. However, the simulated sensor used for terrain mapping in this environment has always precise information, which is very different from reality where digital elevation model (DEM) is noisy and incomplete. The goal is to develop a method that would process DEM created by simulation and transform it to a more realistic DEM.

We build on top of our previous work that uses Cycle GAN networks to generate realistic DEMs from another sensor. We improve current results by adjusting parameters, preprocessing inputs, and introducing paired data.

Keywords: outdoor robotics, sensor simulation, GAN

Supervisor: doc. Ing. Tomáš Svoboda Ph.D.

Abstrakt

Terénní robot by se měl dokázat samostatně pohybovat přes obtížný terén. Učení jízdy přes překážky probíhá v simulovaném prostředí. Simulovaný senzor, který předává robotovi informace o terénu, má vždy o okolí perfektní informace, na rozdíl od reálného senzoru RealSense na robotu, ve kterém se vyskytuje šum a chybějící data. Cílem práce je vytvořit metodu, která dokáže zkreslit simulátorová data tak, aby vypadala realisticky.

Vycházíme z naší předchozí práce, která využívá Cyklické sítě GAN k vytváření realistických dat pro jiný senzor. Lepších výsledků dosahujeme úpravou parametrů, předzpracováním vstupních dat, a přidáním párovaných dat.

Klíčová slova: venkovní robotika, simulace senzoru, GAN

Překlad názvu: Simulace realistických lokálních 3D map z geometrického modelu

Contents

Chapter 1

Introduction

The outdoor robot is designed to help first responders and firefighters explore areas that can be dangerous to humans. It has several sensors, including lidar, omni-directional camera, and Intel RealSense [1], and uses continuous tracks with four individually controlled flippers, see Figure 1.2 and Figure 1.2, to ride easily across difficult terrain. A skilled operator can take advantage of the flippers and navigate the robot over various obstacles, but controlling the flippers manually is complicated and difficult to learn. The goal is that anyone can drive the robot, with flippers controlled autonomously as robot reaches an obstacle.

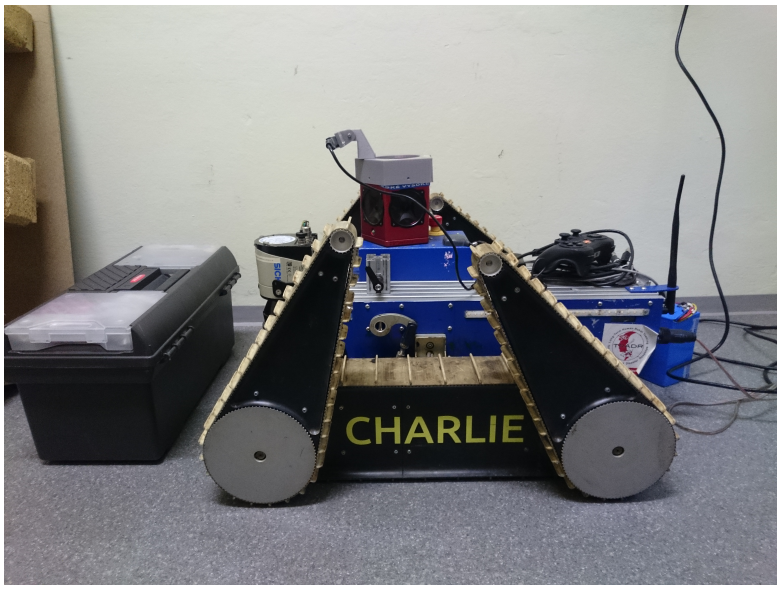


Figure 1.1: Robot with mounted RealSense sensor

To do this, the robot learns policies to control the flippers [2]. The learning takes place in a simulated environment [3], because constant human supervision is not needed there and no damage is done to the robot if it falls.

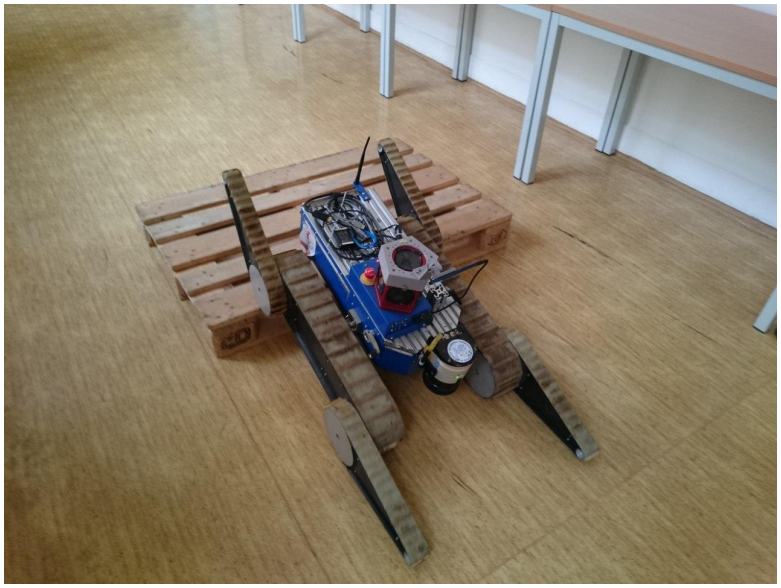


Figure 1.2: Robot using flippers to drive over a wooden pallet

Digital Elevation Model (DEM) of surrounding terrain is one of the main inputs of the adaptive traversal algorithm[2]. The DEM in simulation is created directly from the geometric representation of the terrain, therefore it is very precise. However, in reality, DEM is created by scanning terrain with RealSense and processing it with Elevation Mapping [4] ROS package [5]. These two elements are treated as one sensor in this thesis, and will be referred to as RealSense for brevity.

DEMs from RealSense are very different from simulated DEMs. They are noisy and data might be missing in some areas. Due to this, training the robot on simulated DEMs does not prepare it well for real-life obstacles.

This thesis proposes a method to process simulated DEMs, add noise and missing cells and basically transform simulator DEM enough to get a DEM that looks like it was recorded by RealSense. To do this, Cyclic Generative Adversarial Networks (Cycle GANs) are used. This is based on our recent work [6], which assumes that lidar is used as the robot's primary sensor. However, we concluded in [5] that data from RealSense have better properties for realtime flipper control in this task.

Chapter 2

Problem formulation

In this thesis, we propose a method to simulate a virtual sensor with properties similar to the RealSense sensor. It should provide realistic imperfect data, with noise and missing values. It is important that nature of this sensor is very similar to RealSense, as it will be used in simulated learning [2] to prepare the robot for real-world data. As input to our method, a Digital Elevation Model (DEM) made from a geometric terrain representation is used. The output is another DEM of the same terrain, as seen from the RealSense.

2.1 Digital Elevation Model

The DEM is a 3D terrain model on a discrete grid that maps areas closest to the robot. In [2], grid cells are 10 cm by 10 cm, and the mapped area is 2 m by 0.5 m. This yields a 20 by 5 cell grid.

For flipper control, it is crucial to have data about local terrain, and it is not that important to see further to the front. Area directly under the robot is needed for controlling rear flippers. The DEM covers thus 1.4 m (14 cells) in front of robot's center C and 0.6 m (6 cells) behind it, as is clear from Figure 2.1. Since the output of this thesis is intended to be used in [2], DEMs here have the same size and dimensions.

Elevation is measured in meters and is relative to the robot's center C.

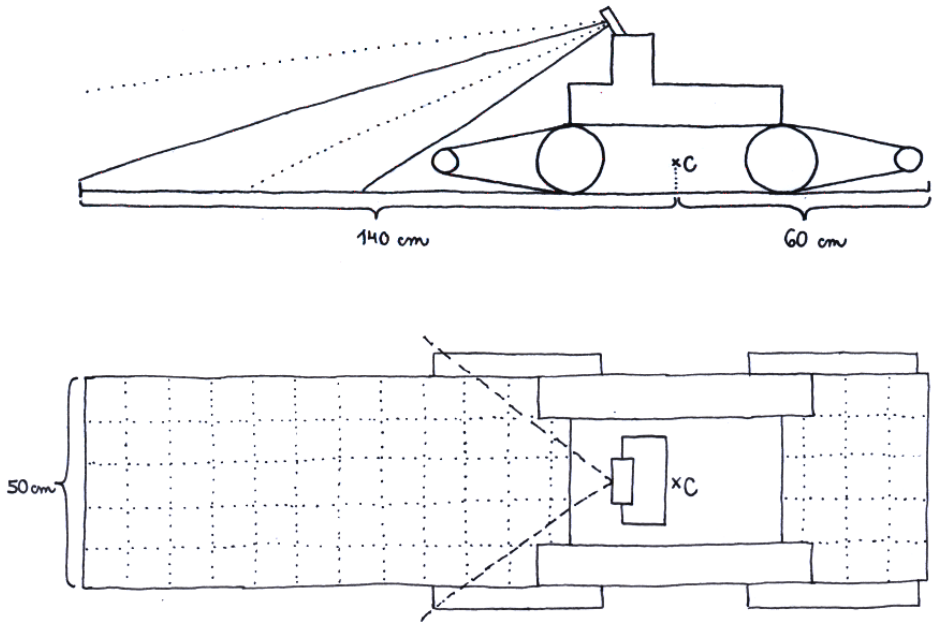


Figure 2.1: A sketch of the TRADR robot standing on DEM grid, with vague outline of RealSense's view range

Anything below the center has negative height, and anything above it is positive. Since C lies approximately 7 cm above the ground, flat terrain has constant value of -0.07 m.

DEM is saved as an array of shape $21 \times 5 \times 2$, where first 20 rows contain height information and the last row describes robot's current position. The second channel is a mask for the first channel. This is further described in chapter 5.

2.2 DEM visualization

DEM is visualized as a rectangular image, where the robot's center lies in the upper part of the image and robot is facing down. Images are visualized using `matplotlib jet` colormap. Shades of yellow, orange and red mark land above the robot, while blue and turquoise represent land below the robot's center. Robot's current height is zero - bright green. The color white is used for pixels with missing value, which are frequent in RealSense data.

In Figure 2.2, robot is standing in front of a staircase. On the left side of

the image, there is the DEM visualization of such terrain. On the right, the robot is shown on top of the DEM for perspective.

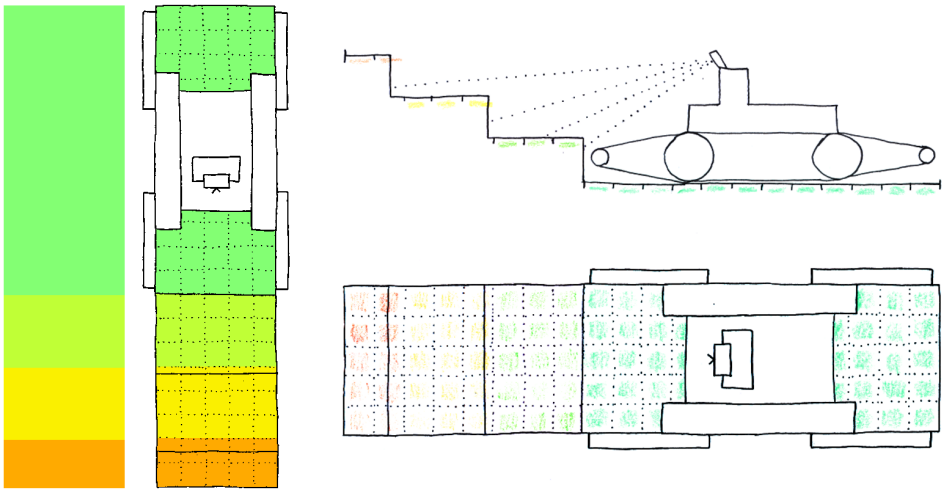


Figure 2.2: DEM for a staircase up. Left: DEM image without and with robot. Right: visualization of matching terrain, side and top view.

2.3 Characteristics of RealSense data

To transform simulated DEM to a more realistic one, we must first understand the nature of RealSense and it's errors and occlusions. One part of this is noise, which slightly changes DEM cell values, and the resulting visualization is blurry. Secondly, there are reflections and small blind spots, which are visualized as single empty cells, or rows of empty cells. The biggest issue, however, are large blind spots on the top of the stairs.

2.3.1 Missing cells and noise

The difference between RealSense DEMs and DEMs created in simulation is clear from figures 2.3 and 2.4. These show DEMs of six common terrains: flat ground, driving up the stairs, driving down the stairs, standing in front of a pallet, climbing pallet, and descending from pallet.

While simulated DEM has always clean data, RealSense is blurry. The exact number of steps can not be counted (images 2 and 3) and data for

terrain behind pallet is missing due to occlusions (images 5 and 6).

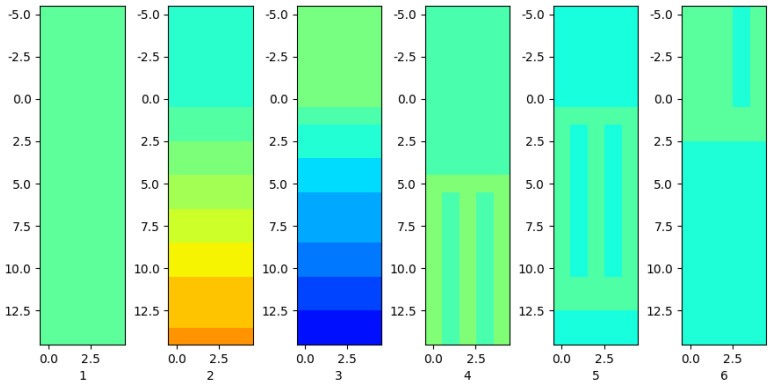


Figure 2.3: Simulated DEMs. Left to right: flat ground, climbing stairs up, climbing stairs down, standing in front of a pallet, climbing pallet, descending from pallet

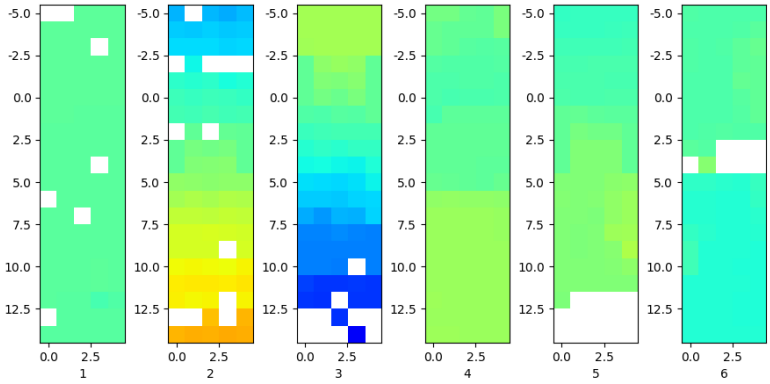


Figure 2.4: RealSense DEMs. Left to right: flat ground, climbing stairs up, climbing stairs down, standing in front of a pallet, climbing pallet, descending from pallet

2.3.2 Blind spots on a staircase

Blind spots on a staircase are caused by the way RealSense is mounted on the robot. It is important that the sensor can record area directly in front of the robot, to have precise information about closest terrain. It is also desirable to see as far as possible. To satisfy both of these requirements, the sensor is facing ground at approximately 65° [1].

[1] Angle 0° would mean the sensor is facing down, 90° would mean the sensor is looking straight forward. The sensor's exact view range is not visualized perfectly on the images, only the bottom boundary is correct

Unfortunately, the angle of stairs that are used in the dataset is very similar. Because of this, the sensor is not receiving nearly any information when robot is standing above the staircase facing down, Figure 2.5. Same thing happens when robot is driving up and reaches the upper end of the staircase. After moving forward and flipping over the edge, new data appears in DEM, but parts of the staircase remain unseen.

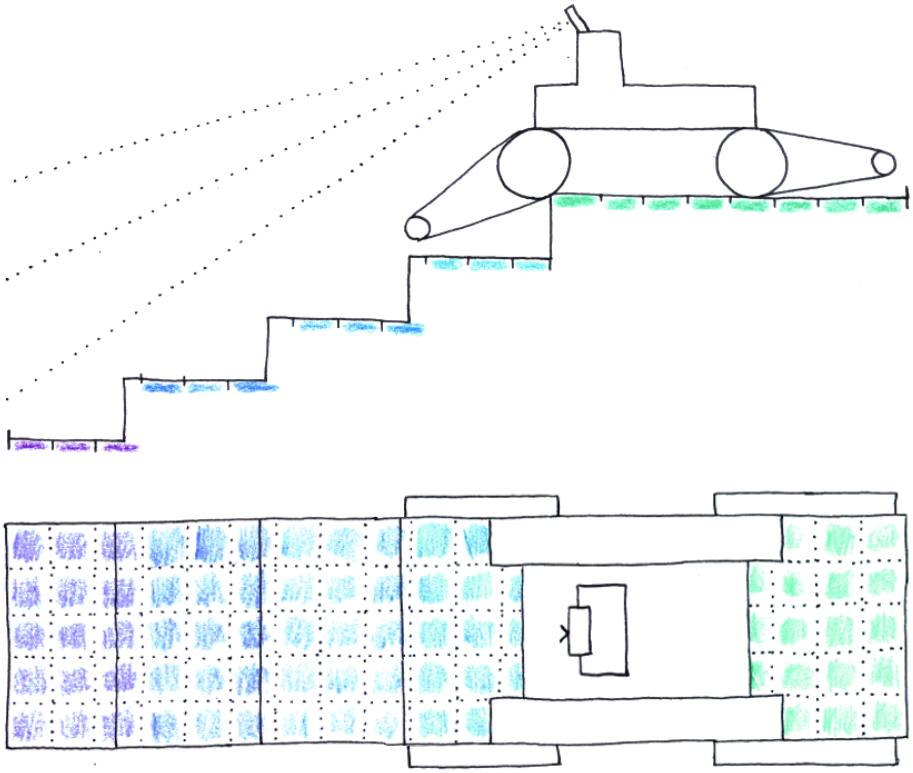


Figure 2.5: Visualization of robot standing at the top of a staircase, side and top view.

Figure 2.6 and Figure 2.7 show development of RealSense DEMs in time as the robot drives up and down over the edge of the staircase. Unlike the simulated DEMs, DEMs from RealSense have missing values, which are shown in white.

In Figure 2.6, DEMs start when robot is standing near the staircase edge, going up. On following frames, the robot comes closer to the edge, which is exactly on the line where color meets with white. On the edge, the robot tilts forward. This happens on the three frames near the end (16, 17, 18), where the edge does not move, but new color pixels appear in front of the robot. The robot is not moving forward on these frames. Afterwards, the robot moves forward and drives further away from the stairs.

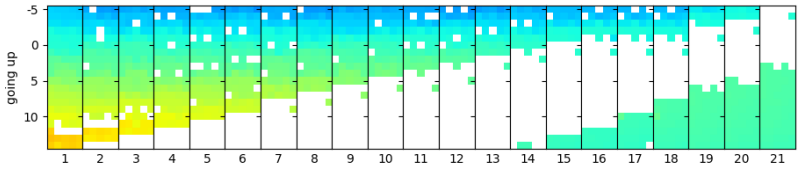


Figure 2.6: Robot does not have any DEM data when reaching top of the stairs. DEMs are ordered left to right and were captured as the robot was driving over the edge of the staircase. Labels on axis Y show distance from robot's center, labels on X show frame id in time

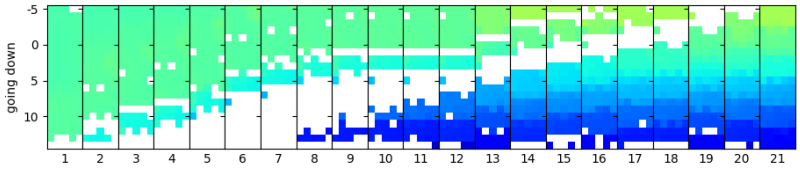


Figure 2.7: Robot does not have any DEM data when preparing to descend. DEMs are ordered left to right and were captured as the robot was driving over the edge of the staircase. Labels on axis Y show distance from robot's center, labels on X show frame id in time

Figure 2.7 shows a similar thing, but the robot starts on flat ground, approaches a staircase down, flips over the edge (frames 8 to 12) and rides a few steps down. In this case, the RealSense was able to see the first step, so the gap is smaller.

Chapter 3

Cycle GAN

This thesis proposes using Cycle Generative Adversarial Networks (Cycle GANs) to convert simulated geometric DEMs to realistic ones. Previous work [6] that uses Cycle GANs to generate realistic DEMs from a laser sensor does not perform well on RealSense data, probably due to different nature of laser's occlusions. It is used here as a starting point, and its results are compared to the new approach.

3.1 GAN

Generative Adversarial Network (GAN) is a network that can generate outputs that are similar to samples from the training dataset. Output from a well trained GAN can not be recognized from original dataset even by humans. It can be used to create new samples and expand the dataset. In [7], GAN was successfully trained on MNIST digit dataset and on CIFAR10.

GAN is created by simultaneously training two networks, Generator and Discriminator, Figure 3.1. The Generator G creates fake images based on an input noise vector, n. Fake images $\tilde{x} = G(n)$ should resemble images from original dataset. The objective of the Discriminator D is to tell $\tilde{x}$ apart from a dataset sample x.

If the network is setup correctly, the Generator will get so good at creating

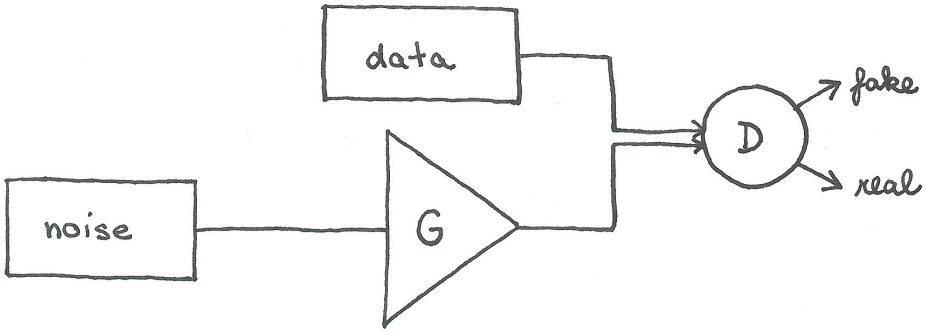


Figure 3.1: Data flow in GAN

$\tilde{x}$ that the Discriminator will no longer be able to tell x and $\tilde{x}$ apart.

3.2 Cycle GAN

Cycle GAN (Figure 3.2) goes further and uses two Generators G_X, G_Y and two Discriminators D_X, D_Y. There are also two training datasets, X and Y, each containing a different group of images. One of the datasets in [8] has photos of wild horses in X, and photos of zebras in Y.

As opposed to GANs, original images are fed to the generator instead of random noise. The Generator then transforms them to the other class, and these outputs are fed to the second generator to convert them back to the original class. This introduces two cycles into the network: the forward cycle (shown in green in Figure 3.2)$[x] \rightarrow [\tilde{y} = G_Y(x)] \rightarrow [\tilde{x} = G_X(\tilde{y})]$ and the backward cycle (shown in blue in Figure 3.2) $[y] \rightarrow [\tilde{x} = G_X(y)] \rightarrow [\tilde{y} = G_Y(\tilde{x})]$. Note that G_X is used to create fake X data from Y samples, and D_X is used to tell x apart from fake $\tilde{x}$.

The aim of Cycle GAN is to preserve "cycle consistency", which means that feeding a sample through forward (or backward) cycle will create an output that is similar to the original sample. Cycle consistency loss is defined as $L = |x - G_X(G_Y(x))|$, or the difference between the input x and generated $\tilde{x}$.

Without Cycle consistency loss, generators learn to create only one sample from target class [8], regardless of the input, which is not desirable when the goal is to transform various images from one class to various set of images in another class.

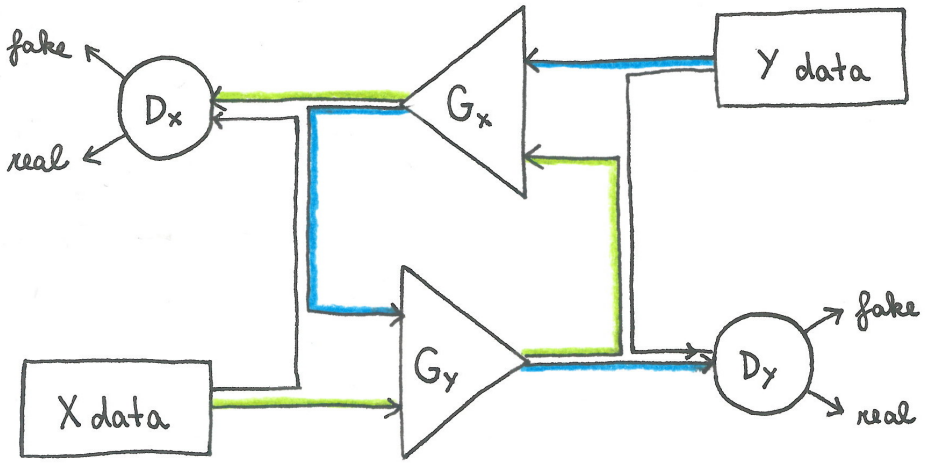


Figure 3.2: Data flow in a Cycle-GAN

Chapter 4

Network architecture

The network architecture used here is taken from the previous work [6], where Cycle GANs are used on DEMs from a laser sensor. No significant changes were done to the network.

4.1 Discriminator

Discriminator takes input of size $21 \times 5 \times 2$. It has three convolutional layers and one fully connected layer, with several instance normalizations and leaky rectified linear unit (Re-Lu) in between. The input is split to image and angular data, and these are processed separately (Figure 4.1).

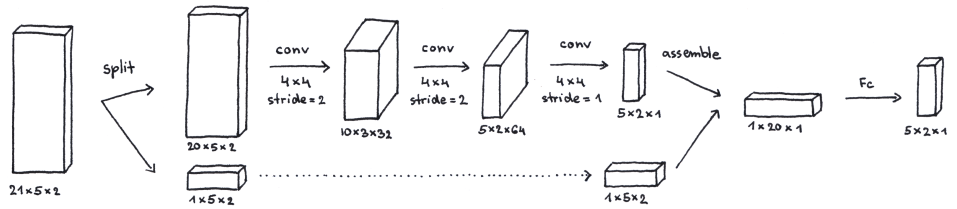


Figure 4.1: Layers of the Discriminator

The convolution layers have 4 by 4 filters and are followed by normalization and a leaky Re-LU. First two convolutions have stride 2, the last convolution has stride 1. The decision d is a matrix of size $5 \times 2 \times 1$ with labels, where

0 means `fake` and 0.9 means `real`. Error is then computed as the mean of logarithms of elements of d. Error on real data is $\epsilon = -\frac{1}{10}\sum \log(d)$, and error on fake data is $\epsilon = -\frac{1}{10}\sum \log(1-d)$.

4.2 Generator

Generator takes input of size $21 \times 5 \times 2$, which is split to DEM and angular data. DEM data is processed by 3 convolutional layers, 3 residual blocks, and three more convolutional layers. Finally, angular data is added and joined data is processed by a fully connected layer to produce the output (Figure 4.2).

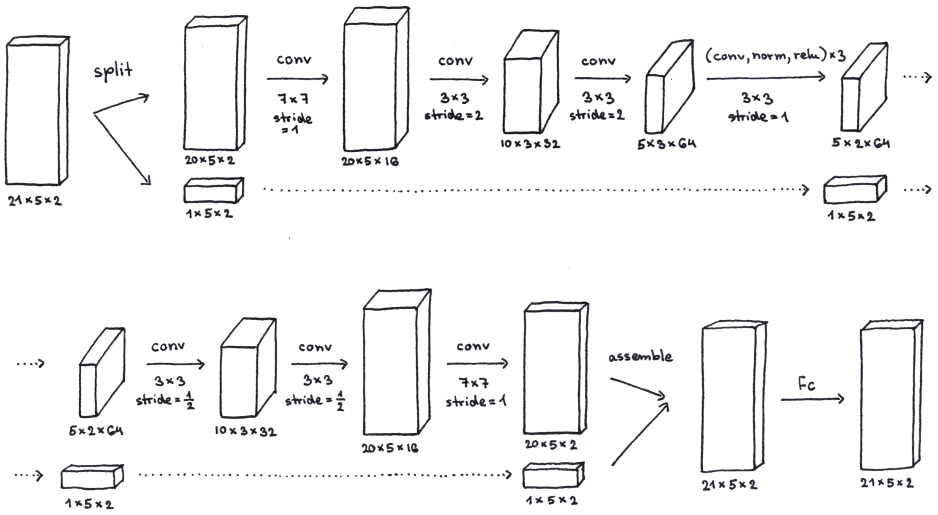


Figure 4.2: Layers of the Generator

Chapter 5

Dataset

The dataset consists of two kinds of data. Group X contains clean data from the simulated geometric environments, samples are shown on Figure 2.3 and include flat ground, going up the stairs, going down the stairs, approaching pallet, climbing on pallet, descending from pallet. Group Y has data that was recorded on the robot, using RealSense sensor by Intel and processed by Elevation Mapping [5]. Samples from similar situations are shown on Figure 2.4.

All data have the same shape $21 \times 5 \times 2$. The first channel contains the actual DEM, where height is saved as `float32` in meters, relative to robot's center position. Second channel is a mask for the first channel and has value 1 for valid pixel and -1 for missing data (not a number - `NaN`). There are no cells with missing data in X, but they are quite common in Y. The bottom (21st) row of both channels contains details about robot position: pitch and angle of all four flippers. This data is missing in Y and is replaced by a row of zeros. The data assembly is shown on Figure 5.1, where DEM is shown in green, mask is blue and angular data about robot and flippers is red.

During training, missing pixels (`NaN`s) are replaced by zeros in the first channel, and mask values range from 1 (valid cell), to -1 (`NaN`). The mask is thresholded only for visualization. It is important to remember that the error is only computed on valid cells, and cells with negative mask are considered `NaN` and their value does not matter.

Data are obtained by recording separate experiments, one in the simulator to get X and one in reality to get Y. Because the experiments are not directly

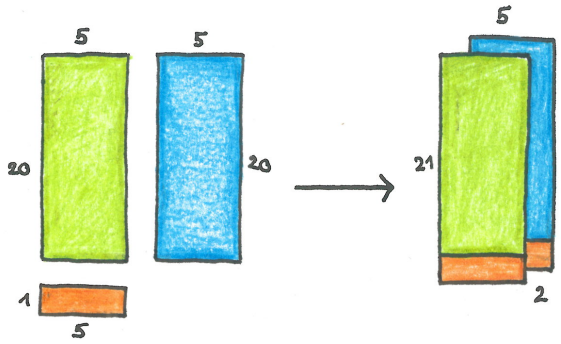


Figure 5.1: DEM (green) and mask (blue) are assembled together with angular data (red) to create data of shape $21 \times 5 \times 2$

linked, matching two samples from both classes to form an exact one to one pair is not possible.

5.1 Discarded samples

Data covers only 3 types of terrain: flat ground, staircase, and pallets. These three terrains are present in both classes. It is beneficial to the learning if all samples from other terrains are removed.

Dataset	ground	stairs	pallet	all	discarded
Simulator data (X)	41633	102141	77502	221276	213259
RealSense data (Y)	538	219	301	1058	1320

Table 5.1: Number of training images in each dataset

In this dataset, discarded data are mostly unrealistic scenarios in X, where two staircases are too close to each other, and noisy data from Y without a corresponding image in X. This leads to 221276 training images from X, and 1058 training images from Y, see Table 5.1. Unfortunately, getting more Y samples is difficult, because data must be organized manually.

5.2 Pairing data samples

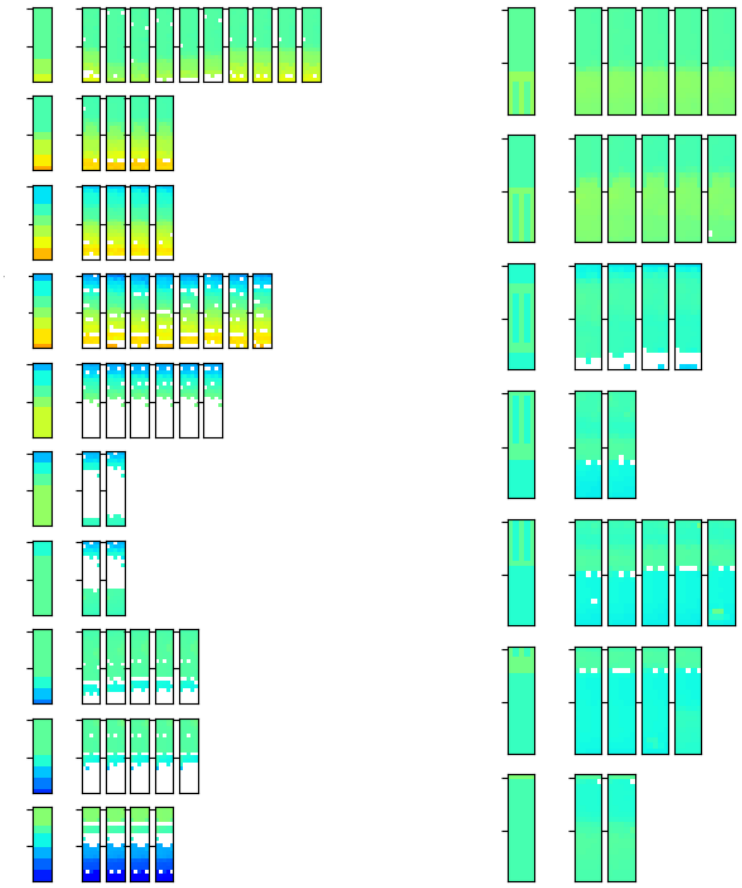
Recording exact one to one pairs is not only impossible, it is also not desirable. Note that one simulator image from X can correspond to multiple images

from Y, because RealSense DEMs are sampled at a higher frequency and because the `NaN` mask depends on previous DEMs. This is clear from the example on driving over staircase edge (frames 16-18 in Figure 2.6), where RealSense DEM changes, but robot is not moving forward, meaning the terrain DEM in simulator is not changing. Choosing only one RealSense DEM as the ultimate right pair of such simulator DEM would remove valuable data from training.

We propose matching one simulated sample x to multiple samples y_i. We consider one key DEM from X and a set of matching DEMs from Y a *pair*.

In order to get the pairs, we recorded data while stopping at positions that could be measured in the simulator, eg. measuring distance from obstacle by an extended flipper. The same situation was recreated in simulated environment, and matches were found manually. In X, all duplicities were removed, leaving only unique DEMs as keys for further pairing. Then Y was displayed and groups of Y samples were matched to each X key.

Paired data is split to two equal parts for training and validation. Figure 5.2 shows selected samples of pairs for stairs and pallet data.



(a) : Paired stairs data **(b) :** Paired pallet data

Figure 5.2: Samples of paired data

Chapter 6

Proposed solution

Using Cycle GANs without any additions, in their original form from [8], does not perform well on RealSense data. This is likely due to the fact that there are too many blind spots in the data, which makes re-creation of the actual terrain model difficult. Since the main loss in Cycle GANs is cycle loss (difference between x and $\tilde{x} = G_X(G_Y(x))$), the network prefers creating matching data x and $\tilde{x}$ to creating realistic images $G_Y(x)$ and $G_X(y)$. This is delayed by increasing weight of the Discriminator's loss, but the results still drift to a state where Cycle loss is successfully minimized and created data do not make any sense. In best cases, the DEM can be considered correct, but the `NaN` mask does not match.

This can be seen on the following models. On Figure 6.1, cycle loss is minimized, but image $\tilde{y}$ is nowhere near the desired result. It does not match the terrain and it does not resemble samples from Y.

On Figure 6.2, the errors are very similar. In the forward cycle ($X \to Y \to X$), the created DEM does not match the terrain (pallet). In the backwards cycle, the created DEM is too various to match any terrain, but still, the cycle loss is very low.

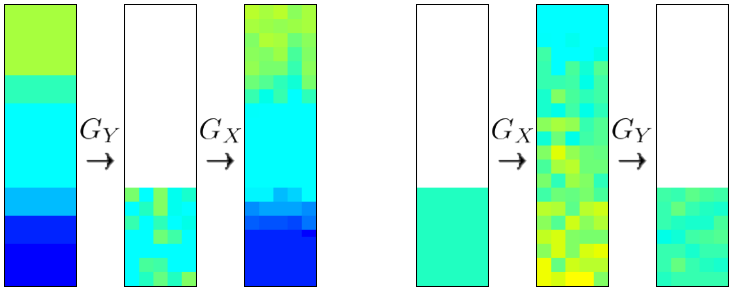


Figure 6.1: DEMs from a forward cycle $[x] \rightarrow [\tilde{y} = G_Y(x)] \rightarrow [\tilde{x} = G_X(\tilde{y})]$ and backward cycle $[y] \rightarrow [\tilde{x} = G_X(y)] \rightarrow [\tilde{y} = G_Y(\tilde{x})]$ in a badly trained model. Images x and $\tilde{x}$ in the left cycle and images y and $\tilde{y}$ in the right cycle are very similar, which means the model minimizes cycle loss well. However, middle images in the cycles do not correspond to the terrain, and are poor imitations of the opposite class.

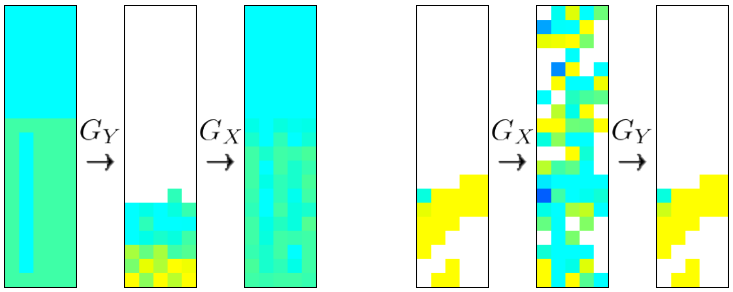


Figure 6.2: DEMs from a forward cycle $[x] \rightarrow [\tilde{y} = G_Y(x)] \rightarrow [\tilde{x} = G_X(\tilde{y})]$ and backward cycle $[y] \rightarrow [\tilde{x} = G_X(y)] \rightarrow [\tilde{y} = G_Y(\tilde{x})]$ in a model which minimizes cycle loss but is useless, because terrain in DEMs does not match.

6.1 Evaluating models

To compute model accuracy, validation data are used. Validation data contain pairs of one simulator DEM and several DEMs from RealSense for each terrain. On a pair (x, y_i), error is computed as $\min_i(\sum (y_i - G_Y(x))^2)$. Errors are computed on all pairs from validation set. There is no need to measure error of G_X, as only G_Y will be used in the training process to modify the terrain.

6.2 Introducing paired data

To force the Generator to create better images, we add paired data and introduce two paired errors. Training on paired data runs after every training step of cyclic GAN training. Errors are computed differently for each Generator,

but both of them have weight $\frac{1}{100}$ in the learning process, to have similar initial values as other losses.

On G_X, which creates simulator DEMs from RealSense, there is always a matching pair, y and x. From one RealSense DEM, only one simulator DEM can be created[1]. The error is computed as a squared difference across all pixels of input x and $\tilde{x} = G_X(y)$: G_X loss $= \sum (x - \tilde{x})^2$. Since the norm is computed equally on both channels of data, errors in DEM, where values range approximately between -0.3 and 0.3, have much smaller impact than the `NaN` mask. This is actually desirable, because `NaN` mask is crucial to the data.

Error on G_Y is more complex. One simulator DEM x can generate several different RealSense DEMs y_i. This is clear from Figure 5.2. To tackle this, the error of $\tilde{y}$ is computed against all possible y_i, and minimal possible error is used: G_Y loss $= \min_i(\sum (y_i - \tilde{y})^2)$. Additionally, a mask is applied to each of the correct options y_i before computing the error, so only valid cells are included in the error.

6.3 Cleaning pallet data

Introducing matching pairs helps the overall result, but since the pairs have higher priority, unwanted artifacts are highlighted. The most visible artifact is the stripy pattern in pallet data, that does not appear in RealSense data. Pallet DEMs are generated correctly, but DEMs without stripes, such as stairs, do not produce a realistic RealSense DEM, see Figure 6.3.

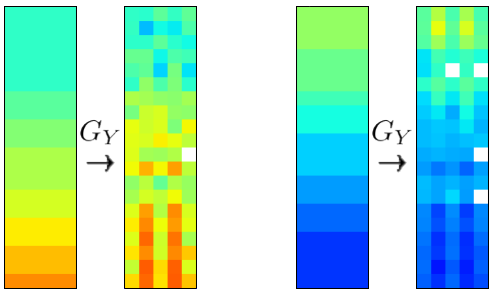


Figure 6.3: DEMs processed by a G_Y that propagates pallet artifacts

Since simulator DEMs are all made artificially, it is easy to modify the

[1]This is not generally true, because an (almost) empty DEM where all values are `NaN` can correspond to any simulator terrain. However, in our dataset, empty images do not appear and DEMs with missing cells can only be fixed in one way.

data to hide the long gaps between individual planks. New pallet data have no stripes, as if the pallet was solid. Figure 6.4 shows examples of pallet transformations.

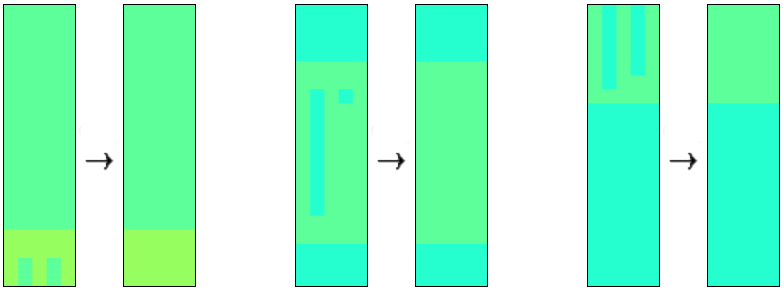


Figure 6.4: Examples of how pallet data was modified

6.4 Removing large empty spaces

Last change to the data handles a rare situation in RealSense data, that happens only on robot startup. When the robot is started, it has no information about the terrain it is currently standing on. Data is available only for the lower 7 rows of the DEM the RealSense has in range (see Figure 2.1). All other cells are unknown. Since all recorded data start like this, there is a disproportionate amount of DEMs with large missing areas. In some training cases, this has led to all DEMs being empty, which is not desirable.

To tackle this, only few DEMs with large empty areas behind the robot are left in the training set, and other DEMs have the missing values replaced by random values from existing cells.

Chapter 7

Experiments

7.1 Comparing models

Models are evaluated by testing the ability of G_Y to generate DEMs from validation data. Validation error is computed on the outputs, and DEMs from the model are visualized. The first row contains input data x, the third row contains generated outputs $\tilde{y}$, and the middle row shows paired RealSense data y closest to the actual output.

DEM 0 is flat ground, DEMs 1 to 17 show traversing pallet, and DEMs 18 to 32 and 33 to 55 show going down and up the stairs. Each experiment is evaluated on the same data it was trained on. Performance of the models can be compared on Figure 7.4.

	Experiment \ Error	Lowest	Highest	Mean	Median
1	Lidar	0.988	1.133	1.042	1.042
2	Identity	0.005	2.607	0.548	0.130
3	Paired data	0.027	1.067	0.334	0.258
4	Fixing pallet data	0.064	1.428	0.403	0.322
5	Removing `NaNs`	0.060	1.413	0.400	0.330

Table 7.1: Validation errors on multiple models. Error is computed separately on each pair of validation data. Columns contain experiment id and name, lowest error across all pairs, highest error across all pairs, mean error across all pairs, median of all errors

The error is computed on both channels separately, to highlight their

individual impact. The mask of y is applied to the DEM of $\tilde{y}$, and values that are masked do not affect the error. Also, the error is computed before a threshold is applied to the mask. This means that a cell that seems to have the correct mask might still increase the error.

The error is computed as

$$\sum (y - G_Y(x))^2/100) \tag{7.1}$$

In other words, it is the squared distance of values from y and $G_Y(x)$, summed across all cells of both the mask and the DEM. Error on one cell is simply $(y - G_Y(x))^2$.

Let us consider a cell in the mask, where $G_Y(x)$ has a value 0.5, but y has value 1. The error on this cell is $(1 - 0.5)^2/100 = 0.0025$. If the value in y was -1, the error would be $(-1 - 0.5)^2/100 = 0.0225$.

In a DEM cell in $G_Y(x)$ with value -0.09, where y is -0.07, the error is $(-0.07 - (-0.09))^2/100 = 0.0004/100$.

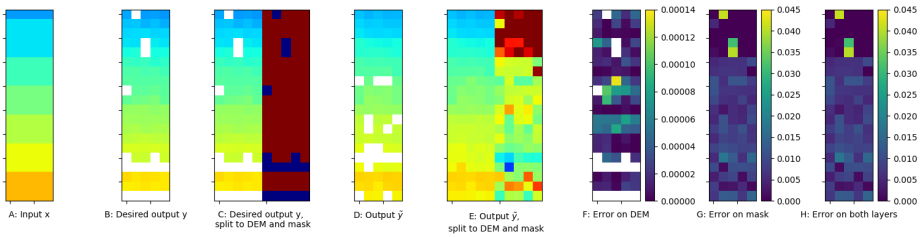


Figure 7.1: Error is computed on both channels of data using the same Equation 7.1. Input, desired output and actual output are shown on the first five images. F is the error on DEM (height) layers of C and E. G is the error on the `NaN` masks of C and E. H is the sum of F and G. The sum of all values in H gives the final error of the image: 0.7116

Figure 7.1 shows computing the error in detail. A sample data pair x and y is shown (A and B on Figure 7.1), and y is then shown with both channels side by side. Then, the generated $\tilde{y}$ is shown (D) and visualised in the same way. Error is computed on both channels separately and visualized on F, G. On the DEM channel, the error is 0.0027, and on the mask, it is 0.7089. Error on whole data is shown on H, H = F + G. The error on this sample is 0.7116.

7.1.1 Original settings

The original solution from [6] uses parameters that were optimized for Lidar data. When run, the mean error on validation data is 1.042. It is clear from Figure 7.2 that the model creates too many `NaN`s, does not understand terrain well and uses the same mask for all types of terrain.

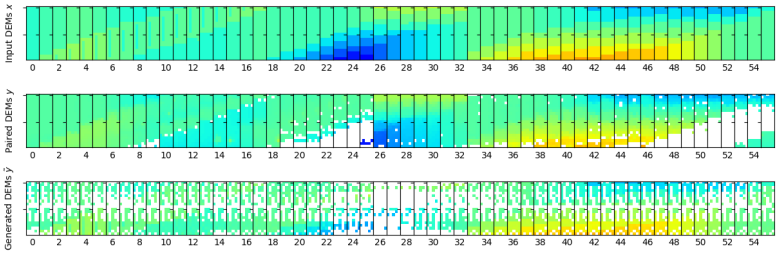


Figure 7.2: Results of lidar model

7.1.2 Identity

Identity model serves as the baseline for comparison. In this model, the Generator simply returns the input without modifying it in any way. Mean error on identity model is 0.548, which is better than the previous model. DEMs from this model can be seen on Figure 7.3

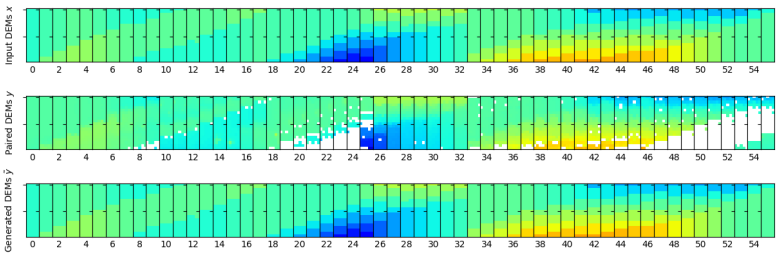


Figure 7.3: Results of identity model

Identity model has low ϵ_L and median, because RealSense performs well on many of the terrains from validation data. DEMs from these terrains do not have a lot of noise and empty cells, so they are very similar to the original simulator DEM.

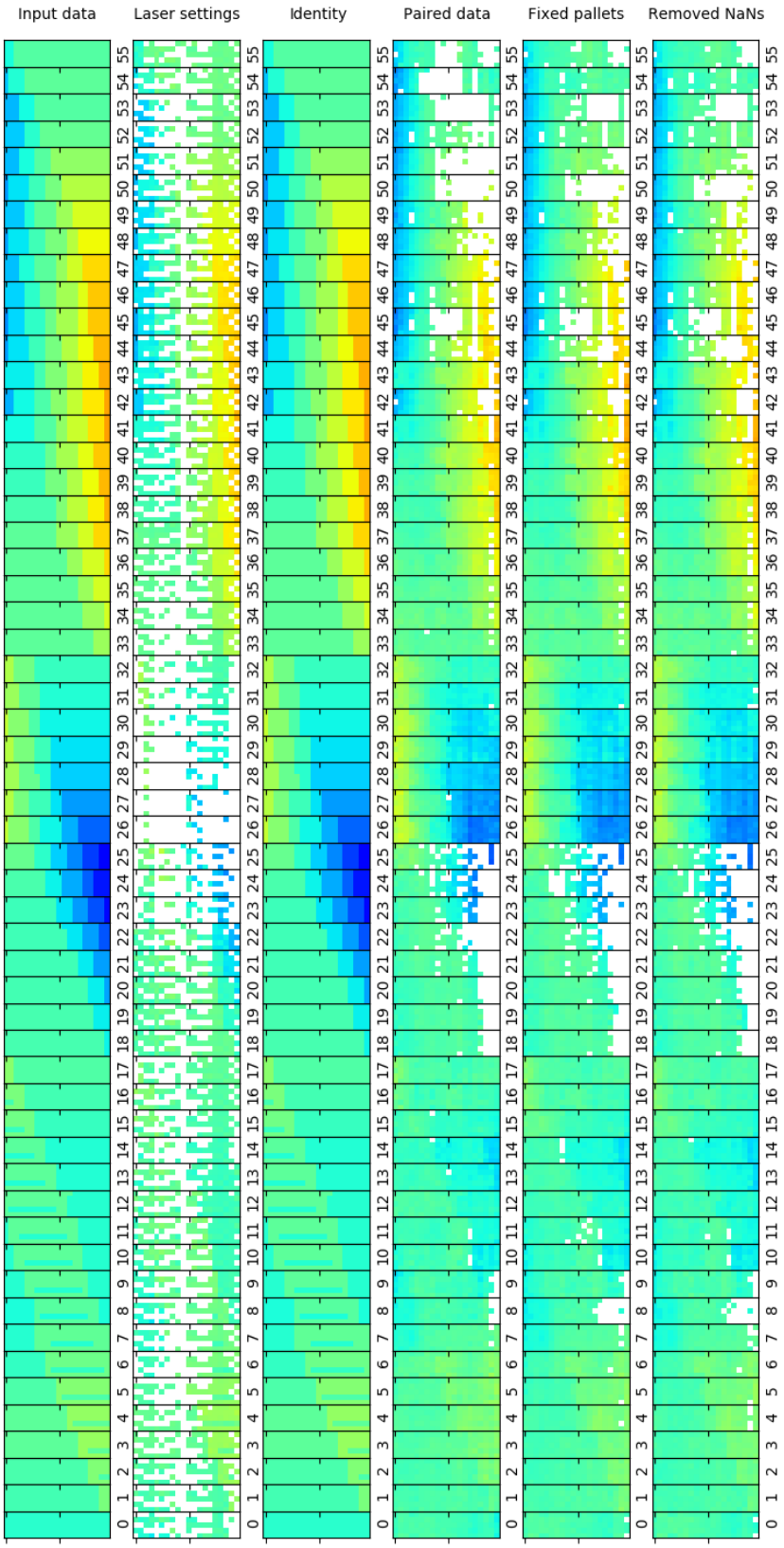


Figure 7.4: Comparison of all presented models

7.1.3 After adding paired data

Paired data improve the performance significantly, lowering the error to 0.334. The first attempt at adding paired data created artifacts, as shown in section 6.3. This led to adjusting data and creating the following two models, but after slightly modifying learning parameters, artifacts disappeared. This is the model that has the lowest error in current settings.

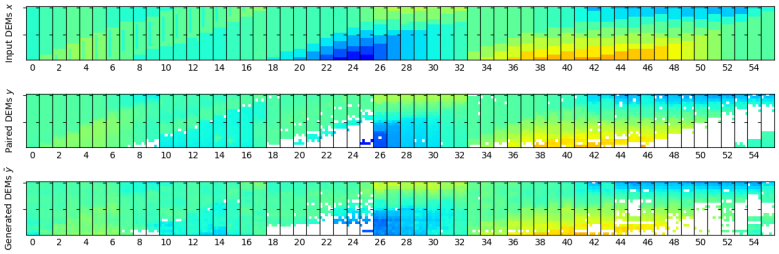


Figure 7.5: Results of model with paired data

7.1.4 Fixing holes in pallet data

Modifying pallet increases the error to 0.403. This is probably because pallets with stripes are easily recognized by the network, and help it distinguish between pallets and short stairs. Clearly, the mask of 8th sample (pallet) in Figure 7.6 matches a staircase mask, such as the one of RealSense sample 20.

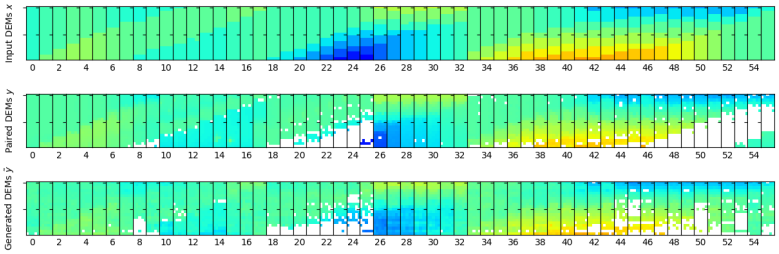


Figure 7.6: Results of model where pallet data was fixed

7.1.5 Fixing samples with NaNs

Fixing NaNs in a dataset with hidden pallet stripes does not affect the error significantly, it only lowers it to 0.400. However, with data samples with large patches of NaNs removed, the training advances faster and Generator does not drift to generating images with an all over NaN mask, as seen on Figure 6.1. Results in Figure 7.7 are very similar to results in Figure 7.6.

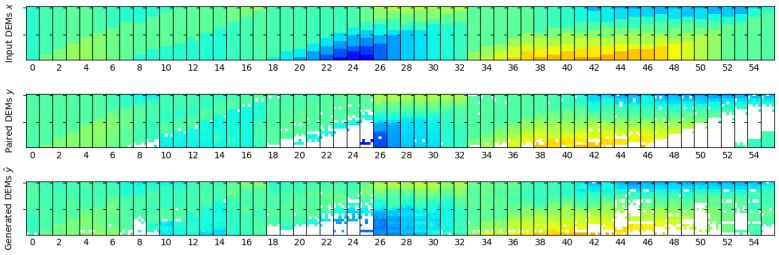


Figure 7.7: Results of model with fixed NaNs

7.2 Using trained G_Y model in simulated learning

Policy for controlling flippers [2] is trained using the model from subsection 7.1.3 to simulate RealSense. This is the model with the best performance on our data, and DEMs created in simulation using this model look very realistic.

Figure 7.8 shows robot standing on DEMs that were recorded from the RealSense. The robot approaches a staircase, climbs up, and then descends. A similar situation was recreated in simulation with our virtual sensor. This is shown on Figure 7.9. The DEMs from virtual sensor look very similar to the RealSense DEMs.

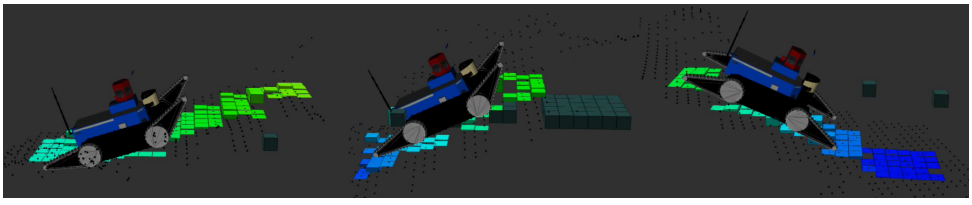


Figure 7.8: RealSense DEMs recorded while the robot was driving on stairs. DEM is visualized as color cubes around the robot. Bright colored cubes show height, dark green cubes mark missing data

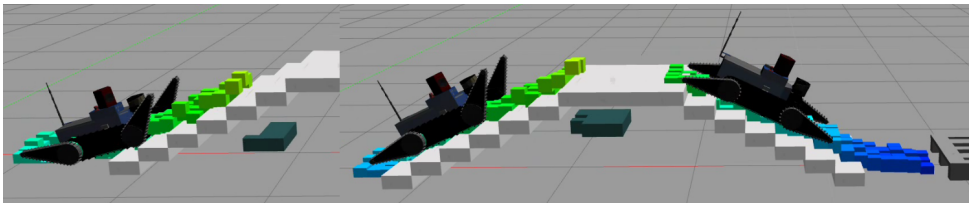


Figure 7.9: DEMs recorded in simulation using the virtual sensor. DEM is visualized as color cubes around the robot. Bright colored cubes show height, dark green cubes mark missing data

There are 8 terrain structures that the policy is tested on: flat ground, pallet, 3 stages of driving up the stairs (approach the staircase, drive on it, and leave it), and 3 stages of driving down the stairs. The trained policy performs well on 6 of those terrain structures. Approaching a staircase and driving up does not work, as the robot does not extend rear flippers, raises both front flippers, and cannot move further (see Figure 7.10). On Figure 7.11 the robot is driving over a pallet.

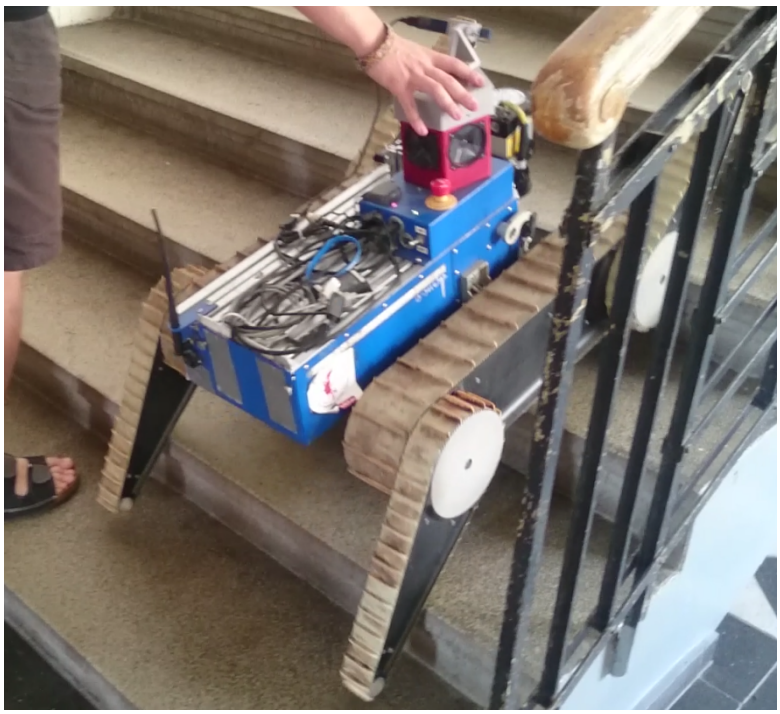


Figure 7.10: Robot struggles to climb up the stairs. This is the same behavior that was observed using this behavior in simulation.

Another of the tested policies can drive up the stairs, but it cannot descend. All of the policies were able to drive over a pallet.

Behavior of the trained policies is the same in both simulation and real world experiments. This shows that the virtual sensor in simulation provides

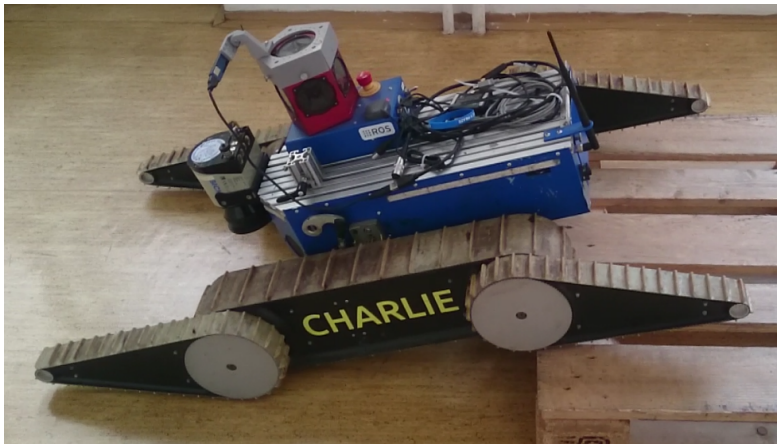


Figure 7.11: Robot descends from a pallet.

data that are very similar to real world data.

Chapter 8

Conclusion

We have developed a virtual sensor that is able to generate realistic RealSense data in a simulation environment. This sensor takes geometry-based DEM of terrain as an input, and creates a realistic DEM of the terrain. Cycle GANs are used to train this sensor, and adding paired data further improve the performance.

We propose pairing one simulated DEM to multiple RealSense DEMs to tackle unbalanced number of samples as well as varying masks of visible terrain in RealSense data.

The developed sensor performs well on validation data, and DEMs look realistic on unpaired input data, where accuracy can not be expressed in numbers.

The sensor is used to train flipper control policy, and this policy can guide the robot over 6 of 8 terrain structures.

Bibliography

[1] "Realsense in ros." http://wiki.ros.org/RealSense. Accessed: 2017-10-30.

[2] M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda, "Controlling robot morphology from incomplete measurements," *IEEE Transactions on Industrial Electronics*, vol. 64, pp. 1773–1782, Feb 2017.

[3] M. Pecka, K. Zimmermann, and T. Svoboda, "Fast simulation of vehicles with non-deformable tracks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6414–6419, Sept 2017.

[4] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[5] D. Hollmannová, "Realsense and elevation mapping for terrain mapping on tradr robot."

[6] M. Pecka, K. Zimmermann, M. Petrlík, and T. Svoboda, "Data-driven policy transfer with imprecise perception simulation," *arXiv preprint arXiv:1804.01953*, 2018.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint arXiv:1703.10593*, 2017.

Appendix: RealSense and Elevation mapping for terrain mapping on TRADR robot

For the sake of completeness, we include the unpublished technical report as an appendix to this thesis. This report was written in January 2018 by Dita Hollmannová and it describes in detail using the ROS package Elevation Mapping and the RealSense sensor this thesis is based on.

Appendix A

Introduction

In the TRADR project, a sophisticated laser sensor is used for terrain mapping. Although the sensor is accurate, it takes almost 3 seconds for it to acquire a scan of surrounding terrain. Due to this, it is difficult to use it in time critical applications, for example, autonomous driving and flipper control.

In this project, I will focus on using a different sensor, the RealSense by Intel, to map terrain in front of the robot. Obtained data will be processed by an existing ROS package, `Elevation Mapping`[2]. In a series of experiments, I will compare data from both sensors.

Appendix B

Used software and hardware

B.1 The TRADR robot

TRADR is a robot developed under the TRADR project [3]. It uses a laser sensor and omnicamera for orientation. For the purposes of this project, a RealSense R200 sensor is mounted on the robot, as seen on Figure B.1.

RealSense [4] is an infrared assisted stereovision 3D imaging system with color camera developed by Intel[1]. It publishes 3D points with color information in PointCloud2 format. In current setup, the sensor publishes 172000 points with frequency 6Hz, with the `indoor` setting.

B.2 Elevation Mapping

Elevation mapping [2] is a package that aims to provide a fast changing local map of terrain arround the robot, as seen on Figure B.2. It requires several inputs including 3D data in PointCloud2 format and odometry.

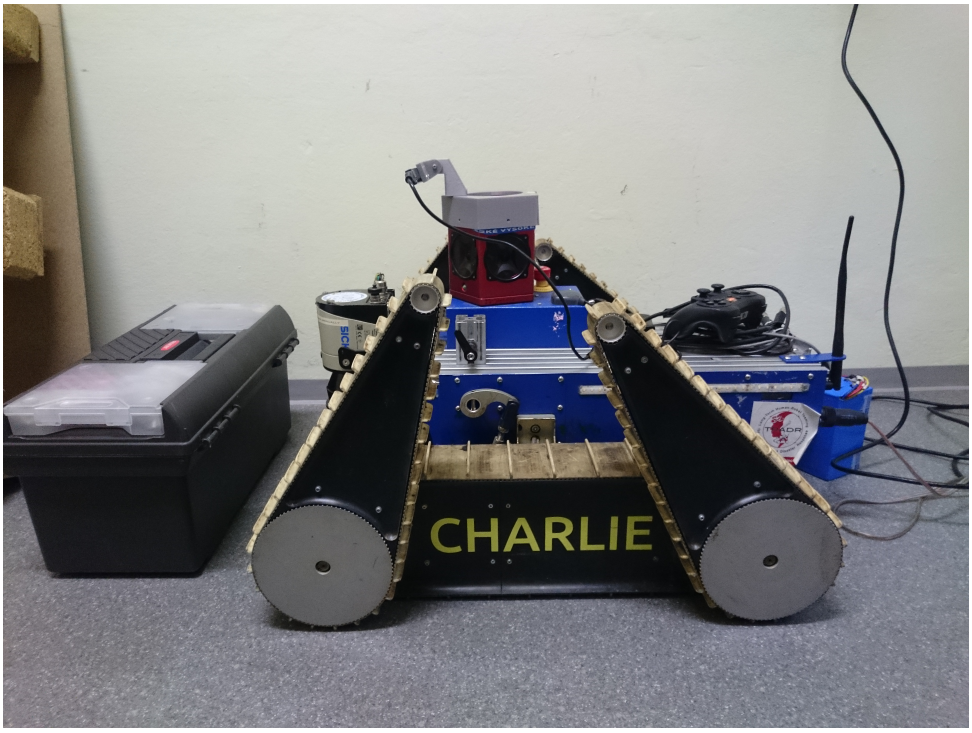


Figure B.1: TRADR robot that was used to record all the experiments

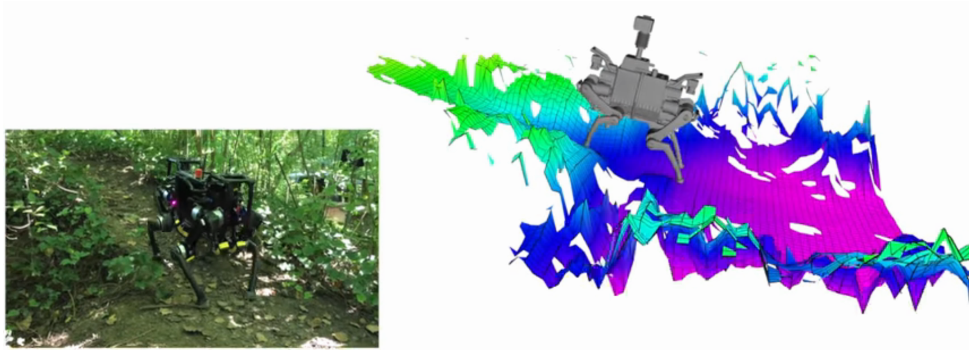


Figure B.2: Elevation mapping running on ANYmal robot of ETH-Zurich [5]

B.3 Grid map representation

Elevation mapping uses the GridMap [6] format to save terrain data. Grid map consists of a grid of cells, each containing its height value.

In the following experiments, GridMap is used to visualise Elevation mapping data, and Octomap is used to visualise Laser data. This is due to the fact that laser data cannot be visualised with GridMap, as they contain both ceiling and walls, not just a surface.

B.4 Optimal parameters

Elevation mapping provides several configuration parameters. The obvious parameters describe topics to follow and `tf` frames with important transforms.

```
/elevation_mapping/point_cloud_topic:
    "/camera/depth/points_min"
/elevation_mapping/map_frame_id: "odom"
/elevation_mapping/robot_base_frame_id: "base_link"
/elevation_mapping/robot_pose_with_covariance_topic: "/pose"
/elevation_mapping/robot_pose_cache_size: 200
/elevation_mapping/track_point_frame_id: "base_link"
/elevation_mapping/track_point_x: 0.0
/elevation_mapping/track_point_y: 0.0
/elevation_mapping/track_point_z: 0.0
```

Moreover, map properties can be changed. This includes

1. map size: `length_in_x`, `length_in_y` are set to 4 meters, not to be limited by the map size, but rather by sensor range
2. robot position in map (`position_x`, `position_y`)
3. grid resolution: in m^2, higher values give large inaccurate patches of terrain, and lower values provide unnecessary details and slow the computation significantly.

```
/elevation_mapping/length_in_x: 4
/elevation_mapping/length_in_y: 4
/elevation_mapping/position_x: 0.0
/elevation_mapping/position_y: 0.0
/elevation_mapping/resolution: 0.05
/elevation_mapping/min_variance: 0.000009
/elevation_mapping/max_variance: 0.1
/elevation_mapping/mahalanobis_distance_threshold: 2.5
/elevation_mapping/multi_height_noise: 0.000009
```

Last section of parameters specifies sensor-related configuration. Rates are set to have the map published often enough, but not to request too many computations. Other parameters come directly from the RealSense config provided by Elevation Mapping. Note that the parameters are provided for a slightly different sensor model, the ZR300, instead of R200.

```
/elevation_mapping/relocate_rate: 2.0
/elevation_mapping/visibility_cleanup_rate: 3.0
/elevation_mapping/fused_map_publishing_rate: 1.0
/elevation_mapping/scanning_duration: 0.17
/elevation_mapping/sensor_processor/type: structured_light
/elevation_mapping/sensor_processor/sensor_cutoff_min_depth:
    0.35
/elevation_mapping/sensor_processor/sensor_cutoff_max_depth:
    3.0
/elevation_mapping/sensor_processor/lateral_factor:
    0.00220941
/elevation_mapping/sensor_processor/normal_factor_a:
    0.00241809
/elevation_mapping/sensor_processor/normal_factor_b:
    0.00662547
/elevation_mapping/sensor_processor/normal_factor_c:
    0.77199589
/elevation_mapping/sensor_processor/normal_factor_d: 0
/elevation_mapping/sensor_processor/normal_factor_e: 1
```

B.5 Issues

Setting up the package properly was complicated. Compiled program would crash due to a compatibility issue in PCL [7], and there was no dataset to test the installation against, which led me to think that my data was corrupt, not the compiled program. After fixing this, it was necessary to convert TRADR's position data from Odometry to PoseWithCovarianceStamped. This was done by creating a simple translating node in Python.

Also, one of the `tf` frames, `sensor`, can't be renamed in a config file like other parameters, therefore it has to be transformed every time via `rosrun tf static_transform_publisher 0 0 0 0 0 0 1 camera_depth_optical_frame sensor 1`.

However, the greatest issue is the one that couldn't be solved during this project. For testing purposes, Elevation Mapping is setup correctly on a Ubuntu 14 notebook with ROS Indigo, but it crashes when the same installation is used on the robot. The package seems to work fine, but it dies with miscellaneous errors when another node attempts to read the published GridMaps. A workaround is to have the notebook connected to

the ROSmaster with Elevation Mapping running externally, but the data transfered is too large for WiFi and maps aren't published in time. This means that to run Elevation Mapping on the robot successfully, notebook must be mounted on the robot, connected to it by an ethernet cable, or someone must walk behind the robot holding the notebook. This is certainly not the desired result and until an on-robot installation is succesful, Elevation Mapping can not be used comfortably.

Appendix C

Experiments

For comparing laser and realsense data, I have recorded both sensors simultaneously in 6 environments. The environments were chosen to represent common scenarios or scenes in which the laser didn't perform well. The aim was to check whether RealSense would really be better as I expected, or if other issues would show up. All experiments were recorded on December 22nd, 2017, in CTU-FEL building on Karlovo náměstí, and the robot was driven by a human operator.

In each experiment, two images are shown. First one is an overview of the experiment, as seen by the omnicamera. This is just to better show how the area looks like, the data wasn't used for the experiments. The second image shows the two resulting maps merged to one image.

C.1 Driving through doorway

The first screnario was simply the robot driving through two rooms into the hallway. No moving objects were present in front of the robot. On Figure C.1, the robot is in the second room. It has driven through the first door from the left (the one with a person in it) and is facing the third door from the left, which leads to the hallway.

Both sensors performed well in this test. Figure C.2 shows both maps

Figure C.1: Omnicam image from first experiment with timestamp

fused together. Clearly, elevation mapping detected all obstacles and walls and marked them by slightly raised surface (shades of yellow and green, compared to solid purple floor). Laser sensor also worked flawlessly. Due to the complicated nature of the room, a floor map generated by octomap was added to the image to make room shape apparent.

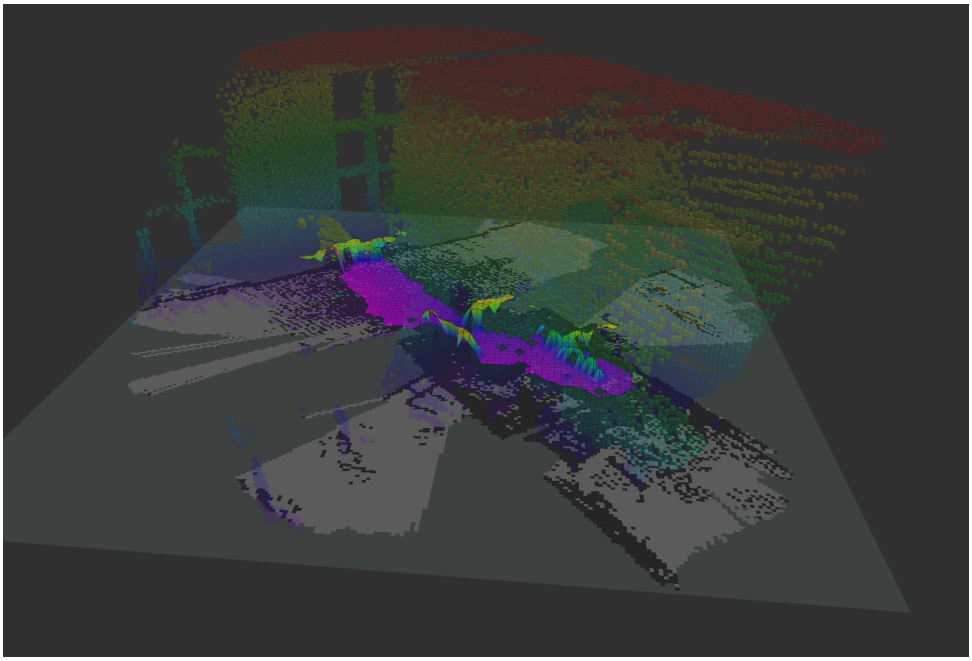


Figure C.2: Maps of the first experiment

C.2 Turning around

In this setting, the robot stays at one spot in the middle of the hallway while it makes a 360 degree turn clockwise. Next to the robot, a toolbox is placed, and two people stand nearby. Laser can have issues if the robot turns while scanning - some sections of the terrain may not get scanned at all. RealSense scans are instant, so it should provide complete data.

Figure C.3: Omnicam image from second experiment

As seen on Figure C.4, Realsense with Elevation Mapping provided a full scan of the area, while the laser has no data for two sections of the map. One of the gaps might be caused by the laser being blinded by the box, but the other occurred where no obstacles were present.

On Figure C.5, the green rectangle marks one of the sections that the laser sensor didn't map. Surprisingly, not only did the laser miss the area, it also recorded an obstacle where there wasn't any. The obstacle is the cluster of points highlighted by the red rectangle. The missing points in the lower left corner might be caused by the sensor skipping the area while turning, or by being blinded by the obstacle (box). There is a similar cluster of points above the missed area, but it seems too high to be the box.

RealSense certainly performed better, but it also had issues. All near obstacles were mapped: the wall on the right, the two people and the box, all marked by green rectangles on Figure C.6. There was also unwanted noise in the map, marked by red rectangle. This was probably the sensor's noise, as it doesn't appear in maps that were created with different parameters, eg. smaller grid cell size.

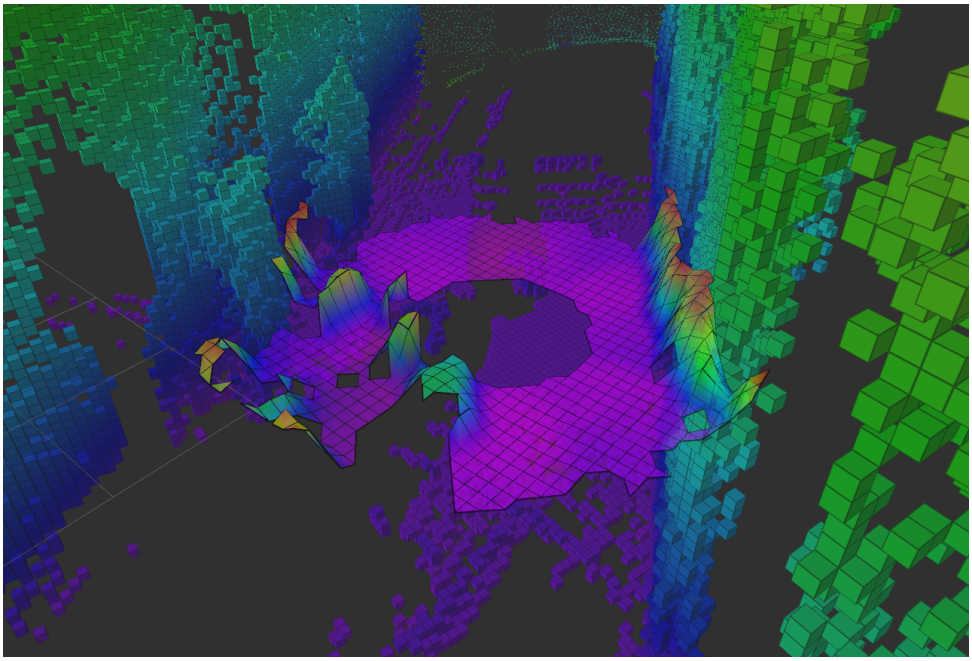


Figure C.4: Maps of the second experiment

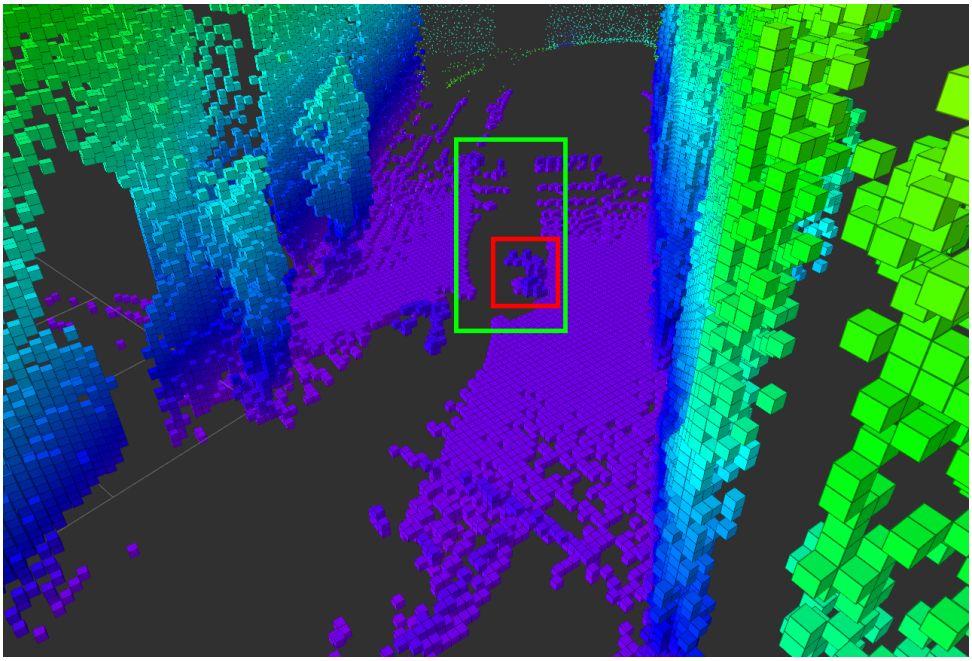


Figure C.5: Errors in laser data

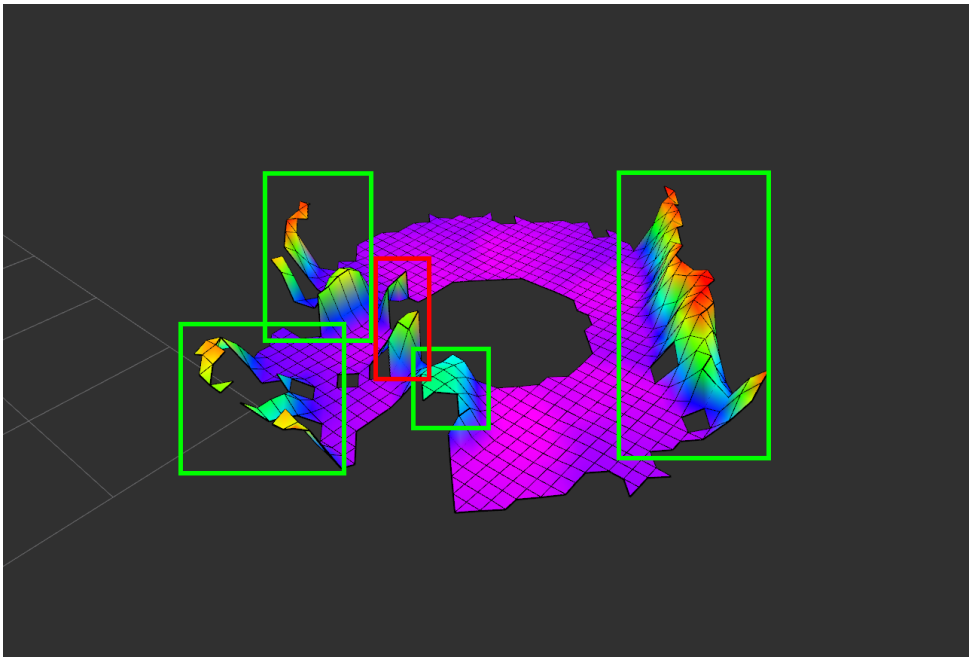


Figure C.6: Errors in RealSense data

C.3 Small obstacle

This sequence was recorded in the hallway. The robot goes straight and stops right before a small obstacle - the toolbox from Figure B.1. The purpose is to check how much would the higher placement of RealSense be an advantage. The toolbox was chosen because it is slightly higher than the laser sensor, which means it might completely block the laser's view.

Figure C.7: Omnicam image from third experiment

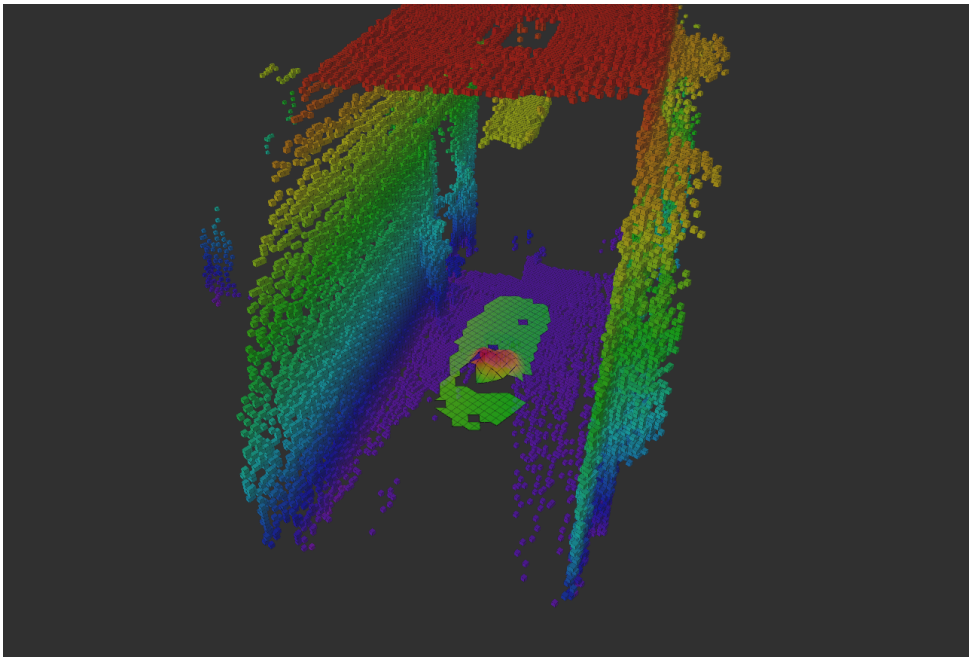


Figure C.8: Maps of the third experiment

As expected, the laser did not see behind the toolbox. In fact, the whole hallway, even a large part of the sides, remained unscanned. The RealSense, however, mapped the top of the box and a rather large part of ground behind it, leaving only a small part of unscanned area (Figure C.8).

This experiment revealed a feature of elevation mapping that has yet to be addressed - the rectangular box is shown first, an over time gets smoothened and becomes a large bump, rather than a box. This is not acceptable, since the bump seems small enough for the robot to be able to drive over it. Smoothing happens when the raw map is processed, and probably can be fixed by changing elevtion mapping parameters.

C.4 Changing environment

It is common for the robot to have moving obstacles, eg. people, around. In this experiment, a person walked in front of the robot, and when the robot stopped, walked to the side. It was designed to show whether RealSense and Elevation mapping adapt well to changing environment.

On Figure C.10, it is clear that in both maps, the person is duplicated. In

Figure C.9: Omnicam image from fourth experiment

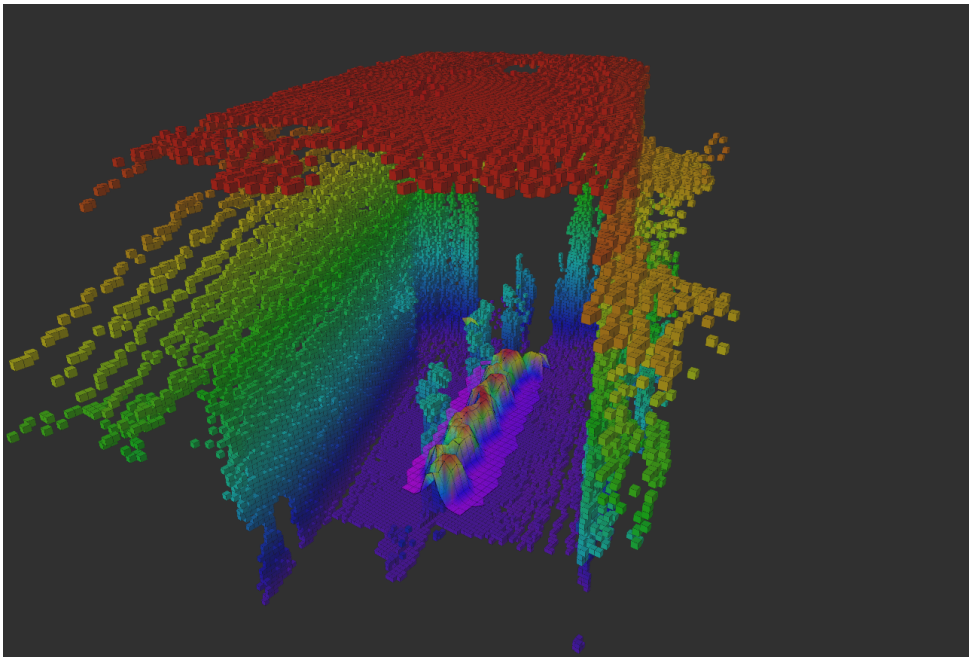


Figure C.10: Maps of the fourth experiment

case of RealSense, this is the best data it can provide, because the ground isn't fully visible at any point, as shown by Figure C.11. Ground further away is hidden behind the legs, and when the person steps back, the ground is already too close to be seen by the sensor.

Unlike RealSense, laser can observe the empty ground directly in front of the robot, as seen on Figure C.12.

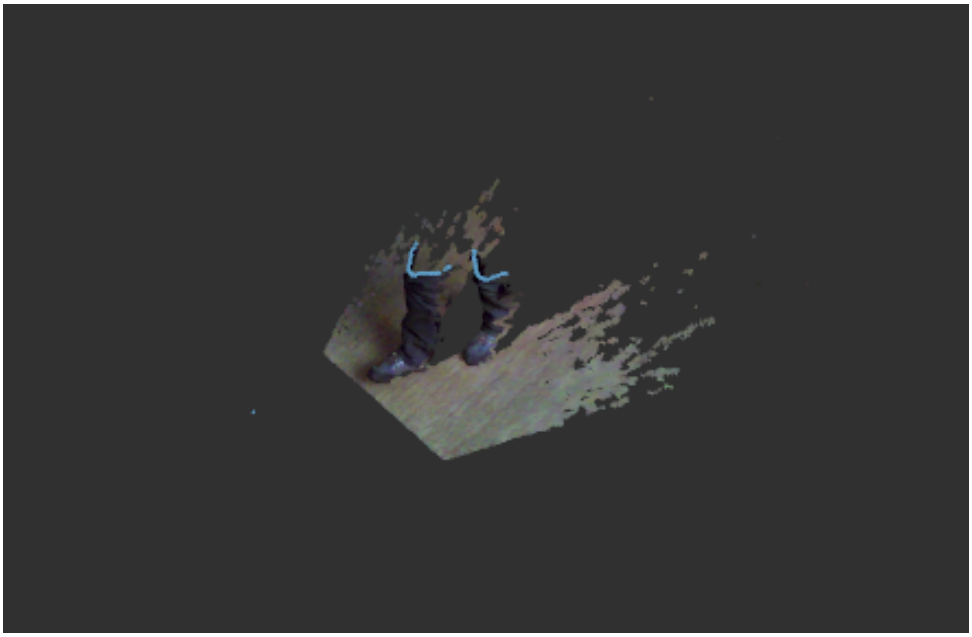


Figure C.11: Raw data (point cloud) from RealSense

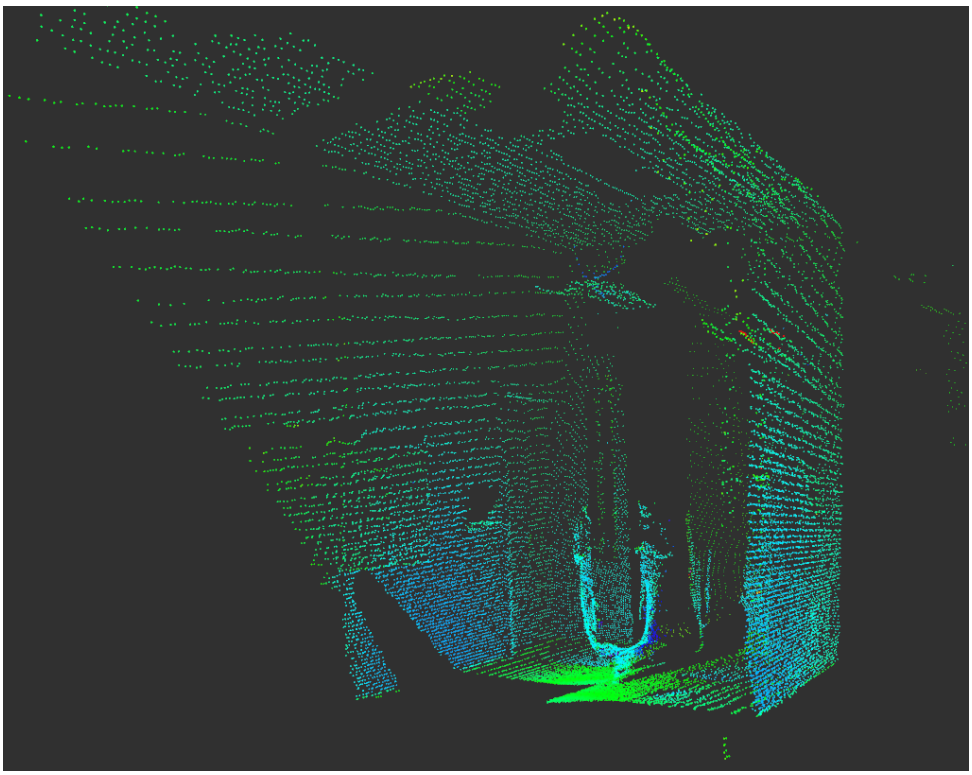


Figure C.12: Point cloud gathered from one full laser scan

C.5 Stairway

The last two experiments feature the robot going down the stairs and then back up. The results for both experiments are the same. The laser sensor mapped everything, while RealSense didn't collect any data for the upper part of the stairs.

Figure C.13: Omnicam image from fifth experiment

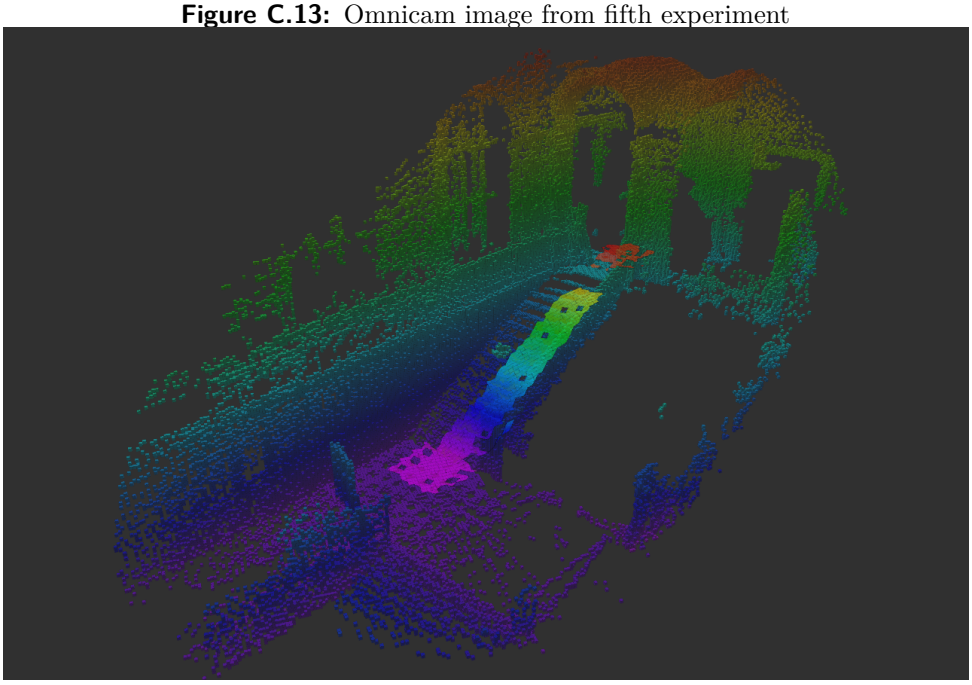


Figure C.14: Maps of the fifth experiment

This is because the angle of the stairs is similar to the angle of the sensor, therefore when the robot reaches the stairs down, there is nothing in its visibility range. Once the robot shifts over the edge, RealSense starts mapping

the stairs again. This happens with part of the stairs on the way down, and with part of the floor on the way up. On Figure C.15, missing data is marked by blue rectangles, and the edge of the stairs is highlighted by the green lines.

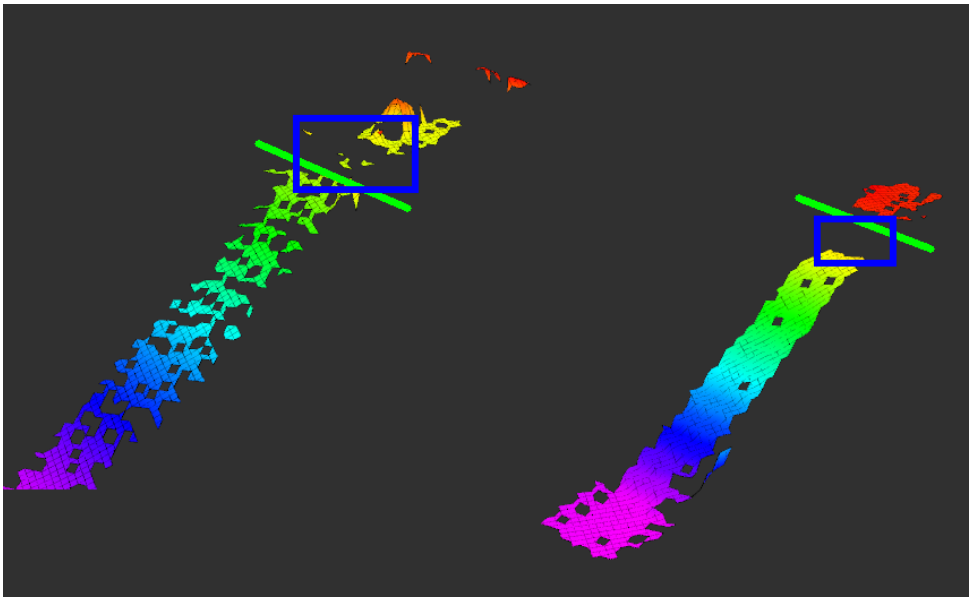


Figure C.15: Elevation map for going upstairs (left) and downstairs (right)

Also, it is worth mentioning that both sensors had trouble with black tiles in the chessboard pattern below the stairs (Figure C.16). This is probably because of reflective properties of the tiles.

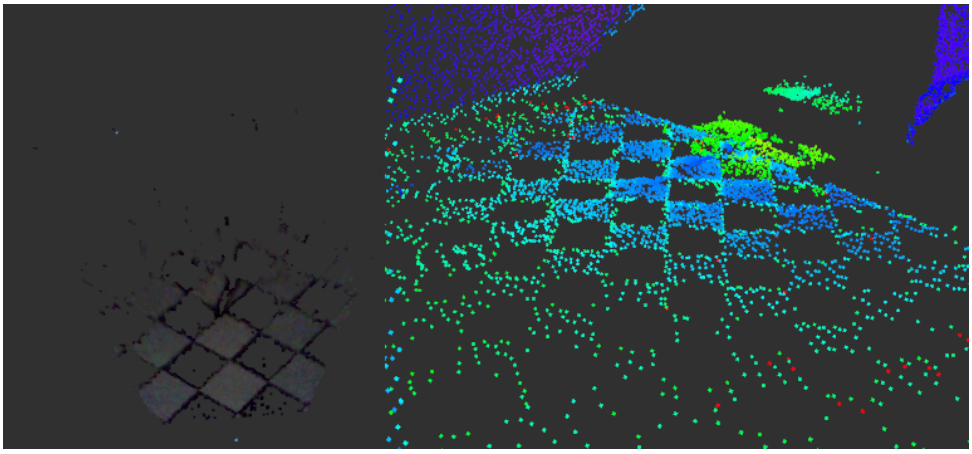


Figure C.16: Tiles as seen by RealSense (left) and laser (right)

Appendix D

Conclusion

The experiments show that RealSense with Elevation Mapping have solved all issues the laser sensor was encountering. Both sensor and mapping are faster, and there are no blindspots when the robot is turning around.

However, several new issues appeared on the RealSense and Elevation Mapping. Elevation Mapping needs to be configured to avoid smoothening of rectangular objects, and sensor position needs to be adjusted to work better on stairs and close objects. Successfully installing Elevation Mapping directly to the robot is also crucial.

Appendix E

Bibliography

[1] "Intel realsense camera r200 product datasheet." https://www.mouser.com/pdfdocs/intel_realsense_camera_r200.pdf.

[2] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[3] "Tradr project." http://www.tradr-project.eu/. Accessed: 2017-10-30.

[4] "Realsense in ros." http://wiki.ros.org/RealSense. Accessed: 2017-10-30.

[5] "Outdoor terrain mapping with anymal." https://www.youtube.com/watch?v=iVMsQPTM65M. Accessed: 2017-10-30.

[6] P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation," in *Robot Operating System (ROS) – The Complete Reference (Volume 1)* (A. Koubaa, ed.), ch. 5, Springer, 2016.

[7] "Elevation mapping issues: process has died." https://github.com/ethz-asl/elevation_mapping/issues/33. Accessed: 2017-11-07.