Tampereen ammattikorkeakoulu

Visual Autonomous Navigation in Forest Environment

Teemu Sormunen

Bachelor's Thesis
May 2020

Electrical and Automation Engineering
Intelligent Machines

ABSTRACT

Tampereen Ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Electrical and Automation Engineering
Intelligent Machines

SORMUNEN, TEEMU:
Visual Autonomous Navigation in Forest Environment

Bachelor's thesis 55 pages, appendices 3 pages
May 2020

The purpose of this study was to do a research on the state of autonomous navigation in a forest environment and to show how the same principles could be implemented in conjunction with a mobile robot. The work on building and programming a mobile robot was part of a course completed before this study, and the results are introduced here. One important goal of this thesis was also to provide the reader with an overview on some of the recent studies on autonomous navigation.

Data was gathered by studying (a relatively extensive number of) research papers on the subject. A discussion on the most relevant techniques and algorithms that are used in modern-day navigation was included in order to explain the central concepts to the reader. The abovementioned algorithms were used in developing the mobile robot.

Earlier studies indicate that the forest environment is relatively undiscovered in terms of visual navigation and mapping due to lack of practical usage. The field of study is however gaining popularity as the capability increases to create more robust ways of navigating in complex environments.

The findings indicate that in order to move forward in the domain of forest in autonomous navigation, work needs to be done on creating certain universal evaluation metrics in either real-life or in simulated environment. Moreover, a way for creating commercial applications must be found. Only after this, the quality of algorithms can be compared, and field will become more competitive. An example of resulting competitiveness can be seen in the growth explosion of machine learning and autonomous passenger vehicles.

This thesis opens up possibilities for the forest industry to become part of this growth by providing the necessary groundwork for visual navigation.

TABLE OF CONTENTS

SPECIAL WORDS

Convergence	State of convergence is reached when the iterative model is not correcting itself significantly, i.e. when optimization method is oscillating back-and-forth under predetermined threshold.
Dense method	Uses most or all pixels in an image to build a map. Requires more computational power than sparse, but provides a dense map, capable for good 3D reconstruction.
Direct method	Use pixel intensities directly. Direct method is fast, but is also susceptible to intensity changes, i.e. lighting changes.
Indirect method	Use extracted features from the image. Requires more computational power but provides better tolerance towards lighting changes.
Inlier	Inlier is a datapoint x which is correctly classified by the model. In example x belongs to model if x is less than threshold T.
Kernel	Kernel in this context is a matrix used for feature extraction from images by convolution, i.e. edges or corners. It emphasises certain patters in image patch more than others, by having weights on the matrix.
Loop closure	When a robot finds previously recognized location after going on a trajectory i.e. a loop. This loop closure can then trigger the pose graph optimisation for the trajectory to correct drift.

Navigation stack	Software architecture of navigation, which contains set of programs from processing of sensory inputs to creating a map and moving in a map. In this context it also includes the actual sensors which are responsible for collecting the data.
Outlier	Outlier is a data point *x* which is not classified correctly with the model, i.e. when *x* is greater than threshold *T*.
ROS	Robot Operating System. Provides open source libraries and tools to create a robot application**.** It was initially founded by robotics research lab Willow Garage and Stanford Artificial Intelligence Laboratory.
ROSBAG	A tool for recording and playing back ROS topics. i.e. record a sequence of point cloud messages and play back later to process the messages.
Sparse method	Sparse representation of data. Uses features from the image instead of individual pixels. Often used mainly to track camera pose and trajectory.
(Visual) descriptor	A unique feature vector which contains information of a feature that is extracted from an image. Descriptor in ideal case is invariant to transformations (i.e. geometrical and photometrical). Descriptor can also be constructed from multiple features i.e. all features from one image.

1 INTRODUCTION

This works aim is to provide knowledge for understanding state of art research studies in navigation of autonomous vehicles. A separate work was done, where the principles described in this thesis were implemented on a custom-made mobile robot. The robot was equipped with monocular depth camera, and it was the only sensor used for navigation. For this reason, this thesis focuses more on different algorithms for monocular camera, and not so much on the hardware side or traditional navigation methods such as wheel odometry, GPS or inertial measurement unit (IMU). The knowledge of the previously said traditional sensors is taken for granted when they are used in this thesis and are not explained. More about the importance of these traditional sensors can be read from article done by Thrun et al. (Thrun *et al.* 2006, 661-692). This article writes about Stanley, the first autonomous car to win DARPA 2005 autonomous vehicle grand challenge, consisting of 212 driveable kilometres, while also describing their hardware in great details.

Extreme weather conditions or robot kinematics are also not discussed. The reader is assumed to have undergraduate-level knowledge in mathematics and in computer science. In the second chapter the very essential knowledge is given to understand the behaviour of more advanced algorithms. Third chapter describes underlaying theory for navigation of the previously mentioned self-implemented mobile robot and introduces the results. Fourth chapter applies the knowledge gathered from chapter two to describe the most influential research works lately in writer's opinion. Finally, the fifth chapter is left for conclusions regarding the results and discussion on the future of the autonomous forest.

2 THEORY AND ALGORITHMS

2.1 Keypoint Detectors

Keypoint detectors are used for detecting keypoints in the image. Keypoints are areas in the image which are distinctive from background. One of the best possible keypoint is a corner, as it has unique pixel gradient in each direction, meaning that the intensity of pixels change in every direction.

Keypoints can be used in matching two images, and this matching can then be later used to build 3D reconstruction of the environment. Below are described the few most important keypoint detectors which are used in state-of-art image matching algorithms. This introduction to the algorithms is done to reduce the abstraction when talking about different Simultaneous Localization and Mapping (SLAM) approaches, and to give reader an idea why certain algorithm has its own strengths and weaknesses.

2.1.1 Scale-Invariant Feature Transform (SIFT)

SIFT is a scale-invariant algorithm, which means it can detect landmarks through different scales. SIFT is used in many feature extraction-based SLAM algorithms. To implement SIFT there are roughly four steps.

One of the key elements to get scale-invariance is to use Laplacian of Gaussian (LoG). It can be calculated by first blurring the wanted image with Gaussian kernel. In this context Gaussian kernel is a discretized 2D Gaussian filter, which is used for smoothing images. After image is smoothed, the second order derivatives are calculated over it and summed together. It can be described by the following Equation (1.)

$$L(x,y) = \nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \qquad (1.)$$

Where 2D Gaussian *f(x,y)* is described as Equation (2.). (Lowe 2004, 94-95.)

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2.)$$

where *x* is the distance from the origin in horizontal direction, and *y* is the distance from the origin in vertical direction. In Equation (2.) the σ is called standard deviation, which describes how the samples *x* and *y* are distributed around mean. With Equation (1.) we can detect pixel gradients in both x and y directions, which are good for finding edges and corners.

Calculating second order is however sensitive to the noise, and calculating derivatives is computationally expensive when calculating through multiple scales. Different scales mean using different standard deviations. Computational cost comes from the fact that LoG requires time complexity of O(2**n)* Gaussian operations, where *n* stands for amount of scales in the pyramid. The amount of calculations can be reduced to O(*n+1)* Gaussian operations with Difference of Gaussians, where *n* is amount of scales in the pyramid. Difference of Gaussian is explained below and can be seen in Figure 1. (Niki Estner 2014.)

Also because the scaling term includes σ in the denominator of Equation (2.) the second order derivative Gaussian will lose response with higher variances, leading to the conclusion that the LoG is not scale-invariant in itself. (Lowe 2004, 94.)

This can be avoided by approximating LoG by Difference of Gaussians (DoG). Difference of Gaussians is simply subtracting consecutive Gaussians from each other at different sizing scales (octaves). (Lowe 2004, 95.)

After the DoG filters are found, the images are searched for local extrema (maxima or minima) over nearest neighbours in the same scale and for a chosen amount of pixels in the next octave and the previous octave. If local extrema is found, it is marked as potential keypoint. As the pixel maxima locations are found only on the pixel level, the subpixel values are generated via Taylor expansion around the approximate keypoint. Now extremas of the subpixels can be found by derivating and equating to zero. (Lowe 2004, 95-96.)

This pipeline can be seen in Figure 1. First source image is scaled in to Gaussian pyramid, from where Difference of Gaussians is taken. After that it is possible to detect local extremas.

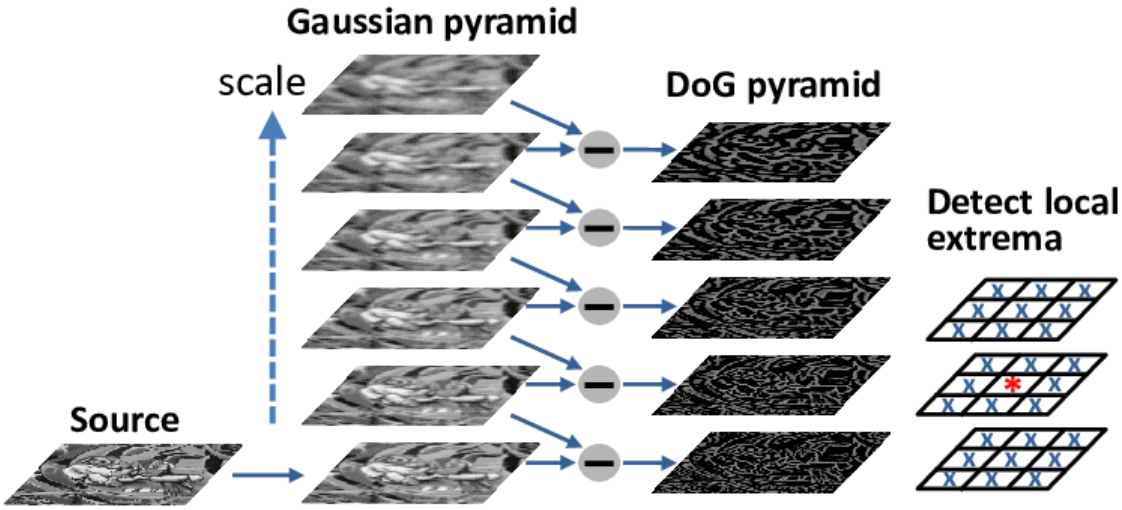

Figure 1. Pipeline of SIFT. (Wang, G., Rister & Cavallaro 2013, 759-762)

After finding the keypoints, the low-contrast and edge keypoints need to be removed. Low-contrast keypoints are filtered by thresholding the intensity level generated from Taylor series. Edge keypoints are filtered by using Hessian matrix and thresholding. (Lowe 2004, 97-99.)

Keypoints also need to be orientation invariant, so gradient magnitude and direction is calculated from the neighbourhood of every keypoint. From there orientation histogram with 36 bins covering 360 degrees is created. Orientation histogram is formed by calculating gradient directions and magnitudes. Each gradient magnitude and direction is then stored based on gradient direction to one of these bins. (Lowe 2004, 98-100.)

2.1.2 Features from Accelerated Segment Test (FAST)

Features from Accelerated Segment Test is computationally efficient corner detector for real-time applications.

First there needs to be set minimum amount N of contiguous pixels (authors used N=12 in original paper) and threshold T. Algorithm starts by picking out point of interest p which has intensity of I_p, and finds circle of 16 neighbouring points as seen in Figure 2. For point p to be classified as corner, at least N contiguous pixels need to be either over value $I_p + T$ or under value $I_p - T$. To make algorithm faster, first perpendicular pixels 1, 5, 9 and 13 are compared and inspected if three out of four pixels are matching the condition. If the case is true, then all 16 pixels are evaluated. (Rosten & Drummond 2006, 430-443.)

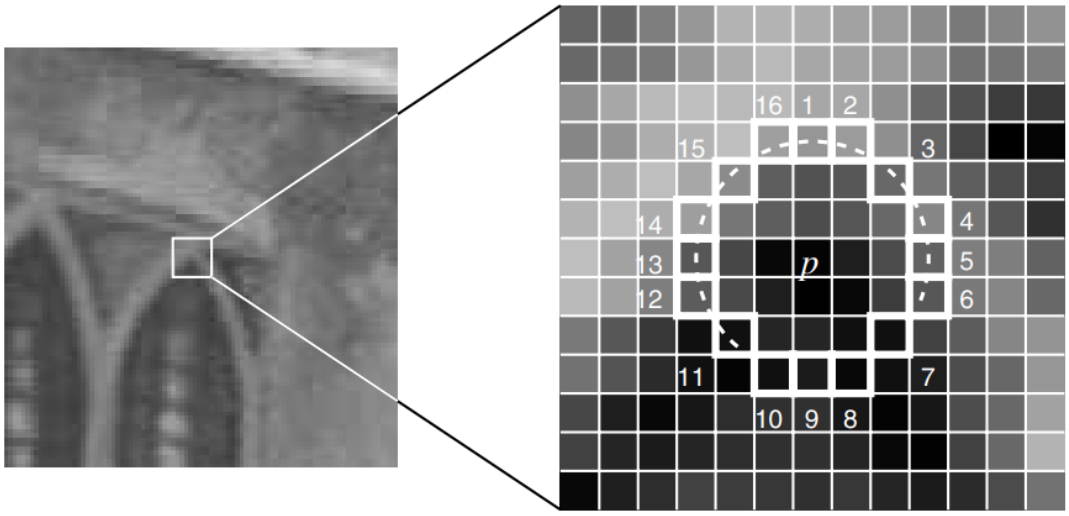

Figure 2. FAST corner detection of an image. (Rosten & Drummond 2006, 434)

This way of detecting however lacks of generalisation for $N < 12$, and multiple features are detected adjacent to one another. For this problem machine learning approach is proposed. In this approach all 16 pixels are vectorised to three vectors depending on the pixels value compared to pixel p, darker pixel P_d, similar pixel P_s and brighter pixel P_b. (Rosten & Drummond 2006, 434-435.)

From this set of pixels $\{P_d\ P_s\ P_b\}$ the pixel x that contains the most information (has the highest entropy) is chosen based on whether the candidate pixel p is a corner or not. This is done with the help of ID3 algorithm and then set of pixels $\{x_0, x_1, x_2 \ldots x_i\}$ built as decision tree. The finer implementation details is however out of the scope of this thesis. (Rosten & Drummond 2006, 434-435.)

2.1.3 Binary Robust Independent Elementary Features (BRIEF)

BRIEF is a lightweight and fast feature point descriptor, and it outperforms other fast descriptors such as SIFT in terms of speed and recognition rate. Feature point descriptors are used in many computer vision technologies, such as image retrieval and camera localization. (Calonder, Lepetit, Strecha & Fua 2010, 778-780.)

The steps for the descriptor algorithms are simple. As descriptor is searching for features on pixel level, the noise must be smoothened out. First, The Gaussian smoothing kernel is used to reduce the noise in the image. Next two random pixel values are chosen from the patch with test τ, and their intensity is compared in Equation (3.) (Calonder, Lepetit, Strecha & Fua 2010, 781.)

$$\tau(p; x, y) = \begin{cases} 1 & p(x) < p(y) \\ 0 & otherwise \end{cases} \quad (3.)$$

where *p(x)* is the intensity of the smoothed patch *p* in location *x*, and *p(y)* is the intensity of the smoothed patch *p* in location *y*. It is found that best results occur when choosing uniform random values. Now the bit array can be calculated with Equation (4.)

$$f_{n_d}(p) := \sum_{1 \le i \le n_d} 2^{i-1} \tau(p; x_i y_i) \quad . \quad (4.)$$

The best upper limit n_d for the Equation (4.) is found to be 128, 256 and 512. As bit arrays are quick and cheap to construct, BRIEF is one of the fastest state-of-art construction and matching algorithm. It however is not of scale invariant and is prone to errors with greater orientations. Other algorithms such as ORB has been designed to tackle this problem. (Calonder, Lepetit, Strecha & Fua 2010, 791.)

2.1.4 Oriented FAST and Rotated BRIEF (ORB)

Oriented FAST and Rotated BRIEF (ORB) is an alternative to SIFT as SIFT is patented algorithm. It combines the FAST and BRIEF algorithms, adding partial scale invariance and rotational invariance.

FAST is efficient for corner and edge detection, but it does not have orientation component and is not scale invariant. ORB creates multiscale pyramid, where original image is down sampled to different scales. Now ORB can detect key-points from different scales and is in this way partially scale invariant.

To achieve orientation invariance for the patch ORB uses following procedure. First moments of the patch need to be calculated with intensity centroid described in Equation (5.) (Rosin 1999, 291-307.)

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{5.}$$

where *x* and *y* are the respective *x* and *y* values, and *I* is the intensity function. This equation can be compared to the equation of mechanical moment of inertia in rigid body, but in this case with respect to the intensity. With moments acquired from Equation (5.) the centroid *C* may be found with Equation (6.).

$$C = (x,y) = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \tag{6.}$$

The centroid practically means the centre of intensity, which can be compared to centre of mass in mechanics as it is often more familiar for most people.

Object is assumed to be centred to according to the origin *O* (Rosin 1999, 294). Then vector *OC* from corners' centre of intensity *C* to the centre of the object *O* can be constructed with Equation (7.) (Rublee, Rabaud, Konolige & Bradski 2011, 2565.)

$$\theta = atan(m_{01}, m_{10}) \quad (7.)$$

This θ now provides us the orientation of the object. This can be illustrated with Figure 3, where centre of mass *C* and centre *O* are chosen to be at arbitrary locations just for illustration purposes. As tangent can only receive angles from -90 to 90 degrees, the angle between absolute horizontal line and the object can be found from Equation (7.). Now this orientation can be compared with other keypoints orientations.

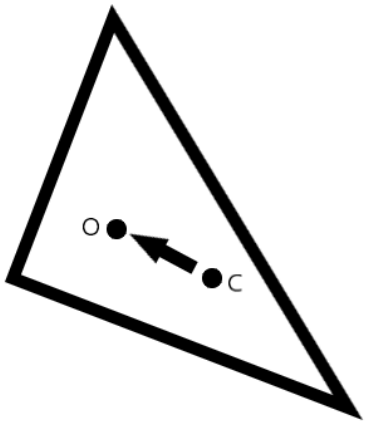

Figure 3. Determining orientation of an object.

2.2 Neural Networks

Neural networks are gaining increasingly more popularity as more data is accessible. Neural networks need to be trained in order to be successful, and the more training data accessible the better. As object classification is often done with Convolutional Neural Networks (CNN), the basic concept for how CNN works is presented here (Huang *et al.* 2017, 7310-7311.)

Some of the CNN implementations are vastly more complex than the simple architecture presented here, but they should be understandable after the fundamentals are understood.

The basic concepts of traditional neural networks are presented first, gradually moving to the more complex and more recent topics.

2.2.1 Motivation for Neural Networks

Development of neural network is inspired from the human neurology. The reason why brain has so much computing power is not completely known, but it still is known that the brain consists of billions and billions of simple similar looking units, neurons. Neuron is nothing more than unit which takes multiple inputs from different neurons and provides one output. Neuron output is active when certain amount of input signals is received. Merzenich and Jenkins proposed that neural pathways are also plastic, meaning that the neurons can either be cut out if there is no usage for the pathway, or strengthen and provide faster response times if the pathway is used multiple times. For this reason, the brains can learn based on the amount of repetitions we do and based on how much we focus on certain task. (Kriesel 2007, 19-35; Doidge 2007, 53.)

Nowadays this structure is attempted to be modelled with perceptrons and combinations of them in order to get the similar performance. One the single most important benefit but also the hardest problem of using neural networks is the capability to generalize. This generalisation comes in to play when the neural network model is given a sample to predict that it has never seen. Now based on training it should be able to classify the never seen sample, based on the ability to generalize. This ability to generalize with lowest possible amount of training samples is one of the biggest problems. For example, in image classification the ideal algorithm would give semantic meaning to the image like humans do, and the exact pixelwise precision like computers do.

2.2.2 Multilayer Perceptron (MLP)

Multilayer perceptron is a basic structure of feedforward artificial neural network.

The perceptron

Perceptron has multiple incoming connections from other perceptrons, but only one output. All these incoming connections have different weights determining the significance of a particular connection. The weighted summation x of the input signals can then be calculated with Equation (8.) by multiplying the value o_i of the

incoming signal with its weight w_i. One incoming signal for each perceptron is set to be constant value (often set as 1), called as bias. This bias perceptron is always on, therefore if all other weights fail to send signal, this bias will keep the said perceptron alive. (Kriesel 2007, 35, 45.)

$$x = \sum_{i=0} o_i w_i \tag{8.}$$

With this equation the weighted sum is forwarded to activation function. The activation function calculates the perceptron's final output signal. One of the popular activation functions is called logistic (sigmoid) function that can be seen in Equation (9.)

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9.}$$

where *x* is the sum of the perceptron inputs. With logistic function the output of the perceptron is limited between 0 and 1 as the result of exponential function in the divisor. There are multiple other popular activation functions. The choice of activation function depends on the domain where the network is used in.

The network

The multilayer perceptron consists of input units and layers of perceptrons. Input units can have any numerically interpretable value. The basic structure of the network can be seen in Figure 4. (Kriesel 2007, 38-40.)

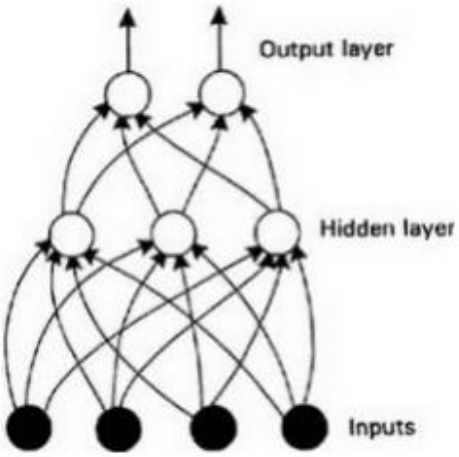

Figure 4. Two-layer net. (Kriesel 2007, 38-40)

The network is fed with different input signals (e.g. pixels values of an image) that are passed on to the hidden layer. In the hidden layer each perceptron processes the inputs, and gives output signal. This output is then passed on to the next layer, which in this case is the output layer, the predicting layer. If the problem is classification problem, then each output perceptron acts as one category, outputting a probability value for the given class as prediction. (Kriesel 2007, 38-40.)

If the network is still being trained, the predicted result can be compared against the given ground truth value. Magnitude of the difference between ground truth and predicted result can be defined with loss function. Loss function is responsible for giving magnitude to the error, so that it can eventually be minimized through optimization. Example of a popular loss function is root mean squared error. (Wang, Z. & Bovik 2009, 98-117; Kriesel 2007, 58, 86-90.)

Optimization algorithm is used for loss function minimization. The popular algorithm gradient descent is often used and will be shortly described below. Gradient descent means moving to the direction of the steepest descent in the gradient vector field. It is based on calculating the partial derivative of the loss function *Err(W)* with respect to weight w_i described in Equation (10.) (Kriesel 2007, 93.)

$$\Delta \mathrm{w}_i = -\eta \frac{\partial Err(W)}{\partial w_i} \qquad (10.)$$

After the gradient has been calculated, the adjustment of the weight Δw_i is in the direction of the negative gradient by step size η, also known as learning rate. This adjustment can be applied to weight w_i with Equation (11.).

$$w_i = w_i + \Delta w_i \tag{11.}$$

This process of doing the prediction phase, loss calculation and finally error propagation is iterated over all the training samples for wanted amount of epochs, until the network has converged to a wanted degree. One epoch in this context means that all the training data has been iterated over once. (Kriesel 2007, 54.)

In reality choosing the activation function, loss function, network structure and hyperparameter tuning is much more complicated, and also requires a lot of trial-and-error. Hyperparameter tuning is adjusting the parameters such as learning rate. For this reason, it is important to have good sense of not only theoretical understanding of the network properties itself, but also of the domain of the data which is being fed to the network. For adjusting the network to best possible performance, it is necessary to be able to interpret error metrics and behaviour of the model. (Kriesel 2007, 1-102)

2.2.3 Convolutional Neural Networks (CNN)

The amount of parameters grows rapidly when adding more layers to MLP. As in this thesis the main focus is on interpreting a video, or more precisely single images, the MLP would take every single pixel as input, and calculate optimal weights for every perceptron based on the pixel values directly. This is not something that is wanted, as the object should be recognizable from the image even if it goes through geometric or photometric changes.

For this purpose the Convolutional Neural Network (CNN) was proposed. It attempts to extract features from images, rather than use direct pixel intensities in defined areas.

The convolution

The convolutional layer lays in the heart of the CNN. Defining convolutional layer begins by picking the convolutional kernel. The convolutional kernels are size of width *w* times height *h* times depth *d*. The width and height can be of any size, but often smaller values such as 5x5 are used to reduce the computational cost. The depth should be same as the depth of input image, e.g. for RGB images on the first layer it would be 5x5x3.

The convolution operation by itself is straightforward. After choosing the kernel size, the kernel is laid over the image. Each value in the original image that is under kernel is multiplied by the kernel value in the corresponding cell. After all values underneath the kernel have been multiplied, they are all summed and sent to the feature map to the same position. Example of such operation can be seen in Figure 5, where the overlapping convolutional kernel consists of 0's and 1's positioned in the yellow area and the original image positioned in the green area. The small red numbers over the yellow rectangles are the kernel values. In the Figure 5 the feature map is the rightmost map with pink background colour. The sliding kernel goes through the original image step by step while doing the respective multiplications. (Saha 2018.)

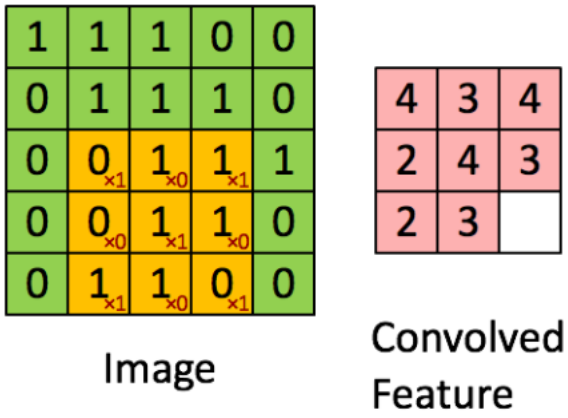

Figure 5. Simple convolution example. (Saha 2018)

The purpose of kernel is to extract certain elements from the image, such as extract vertical or horizontal edges. Each convolutional layer will have predefined number of convolutional kernels. The amount of kernels define the amount of feature maps that are produced, as one kernel produces one feature map after

doing the convolution. Now different kernels can be chosen to extract different features to separate feature maps.

The architecture

Convolutional layers can be stacked after each other as seen in Figure 6, by assigning output of the first convolutional layer to the next convolutional layer as input. In the image the activation function for convolutional layers is Rectified Linear Unit (ReLU). ReLU is popular for its simplicity, simply consisting of linearly ascending value for values higher than zero, and zero for negative values. This functions derivative is a constant resulting in quick calculation. (Hara, Saito & Shouno 2015, 1-8)

By adding multiple convolutional layers it is possible to extract more and more distinctive features of the image. After predetermined number of convolutional layers are stacked after each other, the layers will be vectorized in to one single vector by concatenating rows. The flattened vector is then used as input for Fully Connected (FC) layers.

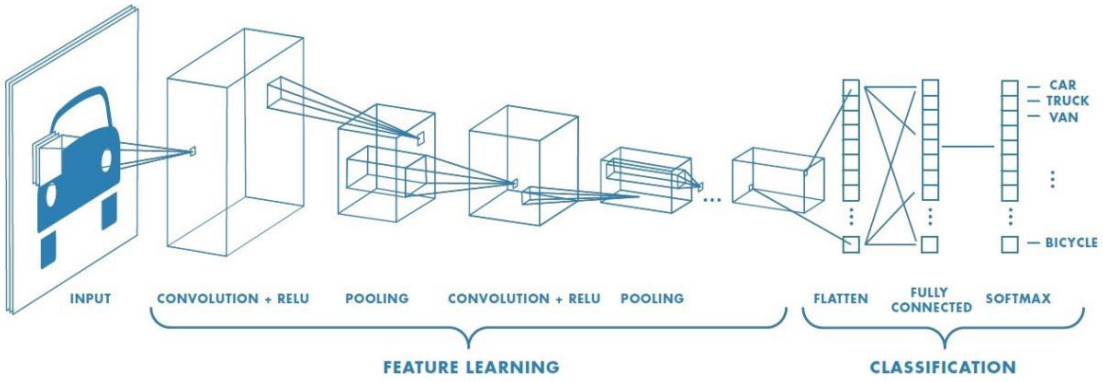

Figure 6. Basic pipeline for Convolutional Neural Network. (Saha 2018)

The fully connected layers might be in the structure of MLP or any other neural network. FC layers' purpose is to do the actual classification of the image, but now with translational invariance of the features that is achieved with the help of CNN. The CNN pools the image size down, resulting in less input units to be fed to the FC layers.

Another, perhaps better representation of the pooling and feature extraction is shown in Figure 7. The first feature map in the convolutional layer clearly extracts rough edges, second the facial structures, and the third eyes and mouth positions. After that the extraction goes to more abstract level, where it can no longer easily be interpreted how the extraction works. The dimension of image is reduced from 128x128 to 8x8, making it possible to increase amount of feature maps from 32 to 64 without significant increase in computational cost.

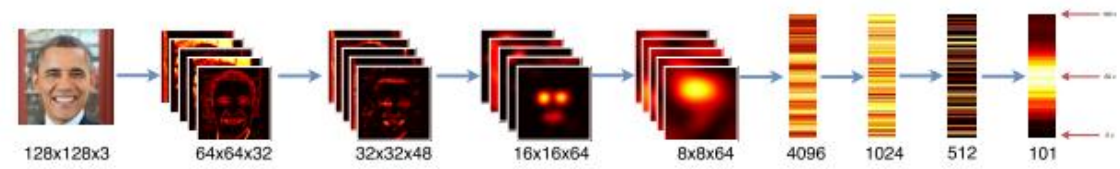

Figure 7. Feature extraction and pooling to reduce dimensions. (Huttunen 2019)

Next the last layer of convolutions is vectorized, passed to FC layers, and sent to output layer. The output layer in this case predicts how old the person in the image is. In this setting, the scale is from 0 to 100, and prediction is the brighter intensity area, getting maximum intensity at around 50 years.

It has been found that the deeper networks achieve better results in classification. Networks that are deeper than 3-4 layers are considered as deep neural networks (DNN) and belong to the popular deep learning field of study. The deep convolutional neural networks (DCNN) are introduced in many studies, so the abbreviations are used to make the text more readable. From now on it is not anymore explicitly said that we are using deep networks instead of shallow ones, as deep networks are much more common than the shallow networks. (Zhong, Ling & Wang 2019, 1-2.)

2.2.4 Transfer Learning

Training a CNN requires a lot of time and computing power. As a result of image classification competitions such as ImageNet, many of the modern network structures and their weights are released for public usage. ImageNet consist of 1 000 different classes, making it relatively generic dataset. This pre-trained network for ImageNet can be used in out-of-box manner for custom problems. The image

domain where the image classification is done is often however more specific than where ImageNet was trained. Training state-of-art DCNN from random weights requires possibly weeks of training on powerful machines. Transfer learning is used to reduce this time. (Sarkar 2018.)

In transfer learning the pre-trained weights for ImageNet are often used. These weights can be fine-tuned to a specific domain by freezing and unfreezing certain layers. One common way is to remove the fully connected layers from the end and add new layers. These new layers can be trained, while pre-trained convolutional layers are frozen. Studies have shown that it is much better to start training with pre-trained weights rather than initializing random weights, even though domain would not match directly. (Sarkar 2018.)

2.3 Light Detection and Ranging Sensor (LiDAR)

LiDAR is a distance sensor designed originally for measuring landforms and differences in landforms. It is great for assessing vertical forest structure at high spatial resolution. LiDAR sends pulses at near-infrared light (e.g. 1064 nm), invisible for human eye. (Sarkar 2018.)

LiDAR measures distance by calculating the time it takes for pulse of light to travel with following Equation (12.).

$$R = \mathrm{c}\frac{(t - \Delta t)}{2} \tag{12.}$$

where *R* is stands for measured range, *c* for speed of light, Δt for time when pulse was emitted, and *t* for the time when pulse was observed back at the originating location. (Coops 2013.)

As LiDAR signal gets further away from the emitter, it tends to disperse. This signal divergence causes non-symmetrical energy distribution over the endpoint, usually two-dimensional Gaussian distribution. Endpoint area where the

signal is expanded, is called as footprint. In general, larger diameter of the footprint can be beneficial if thin objects need to be detected, or if the forest is very dense and at least part of the signal needs to penetrate the forest canopy. Smaller diameter of the footprint on the other hand can result in more distinctive return. It is also more unlikely to receive multiple returns of the area with smaller footprint, and the surface is more likely to be homogeneous. (Rohrbach 2015.)

Popular processing algorithms for LiDAR generated data are introduced below.

2.3.1 Iterative Closest Point (ICP)

Iterative Closest Point (ICP) algorithm can be used for robot localization. ICP attempts to calculate transformation *T* to minimize the distance between two point clouds. Point clouds are often generated with LiDAR. Transformation can be calculated by setting one point cloud as fixed *M* (model point cloud), and minimizing the distance from another point cloud *S* (scene point cloud) to the model *M* point cloud. More in-depth explanation is given after the introduction to k-d trees. (Zhang 1994, 119-152.)

The ICP algorithm uses k-d trees to speed up calculations, so it first needs to be introduced briefly. Simply said, it is a binary tree in *k* dimensions. This concept is visualized in great video lecture done by Victor Lavrenko, and screen capture of the video can be seen in Figure 8. The algorithm begins by taking median value from chosen dimension. The median value is used as a root node, that will split the points evenly to smaller-than and greater-than branches. This splitting is repeated until pre-deteremined number of points for each branch is reached. The depth of the tree is at maximum $\log_2 N$, where *N* stands for the amount of samples in the tree. Now search algorithms can be run on the tree effeciently and in the time complexity of binary trees in general. (Bentley 1975, 509-517.)

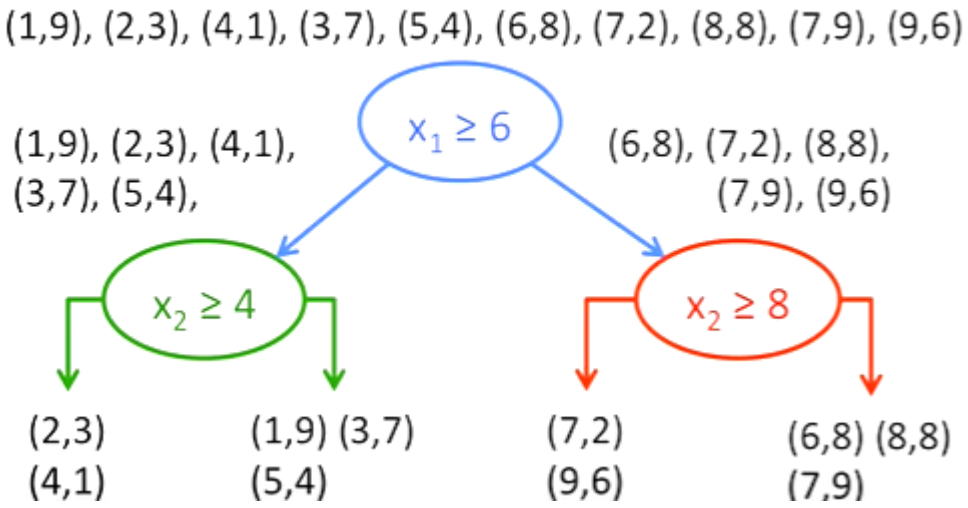

Figure 8. Visual representation of k-d tree in 2 dimensions. (Lavrenko 2015)

The simplified version of ICP can be described as following. First, the k-d tree is built for the model *S*. This is done to speed up the point search from linear to logarithmic time complexity in when finding the point pairs. Then for each point m_i in model *M,* the closest point s_i in model *S* is found by minimizing for example their least squared distance. This can also be called as minimizing the loss function. These point pairs are either kept or discarded according to the constraints described below. (Zhang 1994, 123-125.)

There are two constraints by which the pairs are discarded. As the motion between two point clouds is assumed to be small, the first constraint discards outlier pairs which have greater distance than D_{max}. Second constraint requires the angle between point pair to be smaller than the orientation of the two frames in total. For moving the inlier pairs closer to each other, any optimisation method can be used. After motion is applied, then new pairs are computed again iteratively until convergence. (Zhang 1994, 123.)

2.3.2 Normal Distribution Transform (NDT)

Normal Distribution Transform (NDT) is a point cloud matching algorithm. It creates occupancy grid, where instead of binary values there is calculated normal distribution as described in Equation (2.) in every cell, describing the probability of detection. Each cell therefore contains multiple 2D points, and the normal distribution is calculated over the sum of the points. (Biber & Straßer 2003, 2744.)

Point matching is done by mapping the scene points to the already installed NDT model with and summarizing the points according to the distributions. This calculated score is maximized by optimizing the parameters of Equation (13.) with Newton's method. (Biber & Straßer 2003, 2744.)

$$T: \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \quad (13.)$$

In Equation (13.) T is the mapping between two frames, translation is t_x and t_y, and θ is the rotation between the two frames. Newton's method in general is optimization algorithm that attempts to minimize function *f(x)* (Dennis Jr & Schnabel 1996, 86-87). In this case the score needs to be maximized, so Newton's method is used to minimize the negative score. (Biber & Straßer 2003, 2745.)

Newton's optimisation method iterative process of finding the minimum of a function, but its implementation details is out of the scope of this thesis. More precise definition is in book *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* in chapter *Newton's Method for Nonlinear Equations and Unconstrained Minimization*. (Dennis Jr & Schnabel 1996, 86-91.)

2.4 SLAM

Simultaneous Localization And Mapping (SLAM) is the process of creating a map of the environment, and localizing itself at the same time. Traditionally SLAM combines multiple different sensors, each complementing others' uncertainty. In

this thesis only concepts of visual SLAM are described. The full SLAM problem can be solved by estimating the map *m* and posterior probability for robot trajectory $x_{1:T}$ with Equation (14.)

$$p(x_{1:T}, m \mid z_{1:T}, u_{1:T}, x_0) \tag{14.}$$

where $z_{1:T}$ is a sequence of measurements from environment until step T, $u_{1:T}$ is a sequence of odometry measurements, and x_0 is the initial position of the map. This equation is not explained further in this thesis, more in-depth explanation is given in work by Grisetti, Kummerle, Stachniss and Burgard. Key takeaway is that SLAM attempts to model map and robot trajectory with the help of measurements gathered from different sources. In the case of visual SLAM, the other measurement input is the camera, LiDAR or any other visual input. (Grisetti, Kummerle, Stachniss & Burgard 2010, 31-43.)

2.4.1 Graph-based SLAM

Graph based SLAM in a nutshell models the SLAM problem with a graph. Each node in the graph represents a pose, and the edge represents constraints between the two nodes. The constraint means how each pose is relative to the other poses. If robot is moving straight line for example and the noise is pure Gaussian, then the pose constraint can be described with Equation (14.). Pose constraints can also be in the form of measure constraints. If robot gets measurement z_i from each pose x_i, then the constraint is described by the relation between each measurement z_i and pose x_i. (Grisetti, Kummerle, Stachniss & Burgard 2010, 31-43.)

These constraints can be optimized via pose graph optimization, where the goal is to minimize the error obtained from constraints. This leads to large optimization problem. Graph-based SLAMs using global pose graph optimization can have troubles if there are not enough constraints. (Labbé & Michaud 2019, 3-4.)

It is also important to note that for SLAM to be able to be run real-time, its processing time of the graph must be shorted than node acquisition cycle time (Labbé & Michaud 2019, 7). This is probably the hardest constraint for real-time

performance on many cheap computers and computationally heavy SLAM systems.

2.4.2 Real-Time Appearance-Based Mapping (RTAB-Map)

Real-Time Appearance-Based Mapping (RTAB-Map) can be used with Red-Green-Blue-Depth (RGB-D), stereo and LiDAR sensors, where RGB-D camera is simply normal RGB camera with depth sensing abilities.

RTAB-Map is a graph-based approach to SLAM, with loop closure detection and memory management. It is possible to be used for large-scale mapping, as it limits the loop closure amount. (Labbé & Michaud 2019, 416-446)

RTAB-Map has three kind of links between node transformations in the graph: neighbour, loop closure and proximity links. Neighbour links are between two consecutive poses, loop closure link is added after loop closure and proximity link is added after proximity detection of nodes. All these nodes are then used as constraints for graph optimization. Optimization is run through the whole graph after either loop closure or proximity link is added to the graph to reduce odometry drift. (Labbé & Michaud 2019, 421-423.)

RTAB-Map offers two different visual odometry approaches, Frame-To-Map (F2M) and Frame-To-Frame (F2F). The biggest different between F2M and F2F is that F2M registers the current frame against local map created from few past frames, while F2F compares consecutive frames directly. The actual feature detection algorithm by default is GoodFeaturesToTrack (GFTT) for monocular cameras, but any feature extraction algorithms can used from OpenCV library (see chapter 2.1 for popular algorithms). (Labbé & Michaud 2019, 424-425.)

OpenCV is a computer vision library which is written in C and C++. It is commonly used in computer vision groups. (Bradski & Kaehler 2008, 1-2.)

In F2M nearest neighbour distance ratio (NNDR) is used to compare features achieved from the current frame with BRIEF descriptors. Based on similarity of

the compared features the features are ranked accordingly. This creates slightly larger computational load, as descriptors have to be created and stored. (Labbé & Michaud 2019, 8-9.)

In F2F only GFTT feature detector is used to match features directly between the frames. Extra feature descriptors are not needed, making F2F faster and lighter feature detector. Using only short term feature matching however is prone to drift, resulting in slightly less accurate map. (Labbé & Michaud 2019, 424-425.) Choice of F2F or F2M needs to be decided carefully according to hardware capability.

In Figure 9 there is excellent visualization of RTAB-Map data flow pipeline. Deeper knowledge of this pipeline's components is not however needed, and only the parts where parameter tuning has improved the performance significantly are explained.

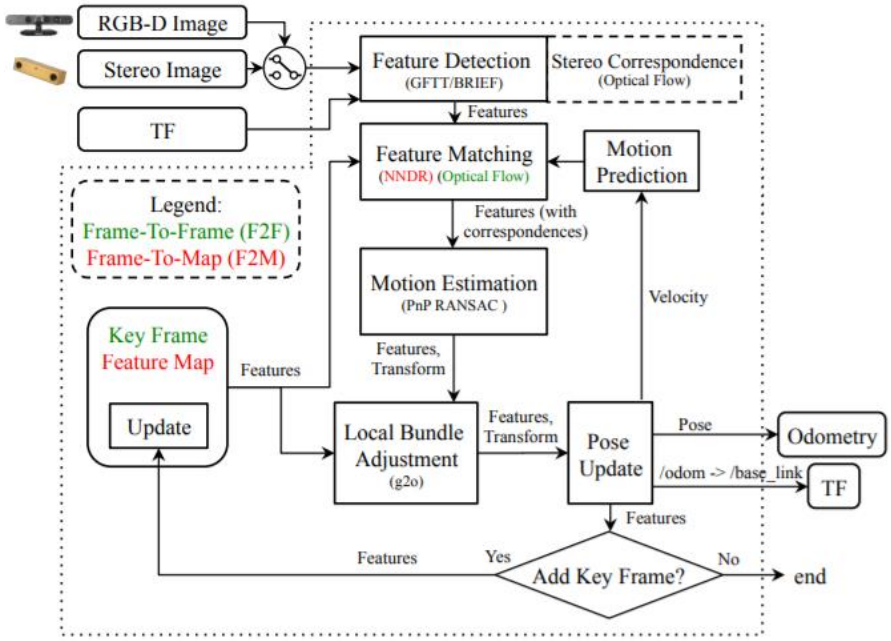

Figure 9. Data flow pipeline of RTAB-Map. (Labbé & Michaud 2019, 425)

In RTAB-Map the odometry can also be generated with different methods via application programming interface (API), and for example ORB-SLAM2 with ORB-features can be used if ORB-SLAM2's localization is set off, as RTAB-Map must handle the mapping. This provides easy to use interface for comparing other visual odometry approaches. (Labbé & Michaud 2019, 424-428.)

2.5 Object Detection and Classification

2.5.1 Motivation

Vehicle not only needs to be able to navigation without collision, it also needs to be able to classify objects in range and have sense of semantic understanding of the environment. Semantic understanding is more important when driving in urban environments as there needs to be distinction between humans and light posts, but nevertheless it is also required in forest environment. For the edge cases when animals or even humans are detected in the sensor area, the system needs to be able to act accordingly to the situation. Classification is also important in regard to knowing whether object in front is a large stump that must be avoided, or whether it is just forest vegetation which can be driven over.

CNN's have proven to be superior over conventional classification methods in most cases, such as Support Vector Machine (Grinblat, Uzal, Larese & Granitto 2016, 418-424; Ostovar *et al.* 2019, 1579; Pound *et al.* 2017, gix083), or the previously mentioned MLP (Song *et al.* 2016, 734-748). It is also proposed in the other research works that automatic feature extraction performs better than manual feature extraction (Namin *et al.* 2018, 66).

For these reasons manual feature extraction that is based on hand engineering the features with e.g. SIFT, is not considered in this chapter of object classification and the main principles of DCNNs are introduced. The DCNN detectors can be classified in to two-stage detectors, and one-stage detectors. The main difference between one-stage and two-stage detectors is that two-stage detectors have proposition step and classifying step, while single-stage detector combines them both. There is also a trade-off between accuracy and computational cost for two-stage and one-stage detectors, two-stage detectors giving more precise detections and one-stage detectors being faster. (Ostovar 2019, 40-46.)

2.5.2 Two-stage Detectors

In two-stage detectors proposition step regional proposition network (RPN) is often used. Simple definition of RPN is such that it proposes the most likely regions where object could be located. It has high recall rate, which means it rather classifies too many regions than too few. (Ren, He, Girshick & Sun 2015, 91-99.)

In general, two-stage detectors start by proposing possible regions of interest (ROIs) with RPN or sliding with approaches. These regions are then classified as objects or background. One of the state-of-art two-stage detectors is Faster Recurrent Convolutional Neural Network (Faster R-CNN). The architecture can be seen in Figure 10. It starts by extracting features with CNN, passing the feature maps to RPN, mapping the proposed regions over the feature maps and then classifying the regions. This is a modification of originally proposed R-CNN, and variations of this has been made in order to speed the computation to achieve real time performance. (Ren, He, Girshick & Sun 2015, 91-99.)

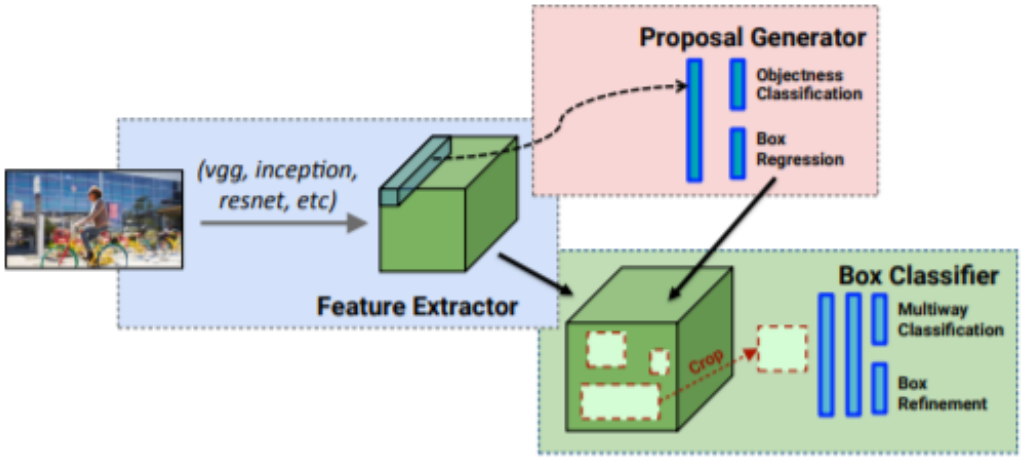

Figure 10. Faster R-CNN architecture. (Huang *et al.* 2017, 7312)

Microsoft's Common Objects in Context (COCO) is a dataset similar to ImageNet, but in this case the objects have labels according to the locations they hold in the image. It is often used for object recognition algorithm evaluation comparison. COCO dataset is used for COCO challenge, which challenges researches from all over the globe to test their algorithm performance against each other. (Lin *et al.* 2014, 740-755)

In the latest COCO challenge winner had a modification of Faster R-CNN called Cascade R-CNN. Cascade R-CNN basically addresses the problem that comes with setting intersection over union (IoU) threshold value. IoU threshold determines how much the object prediction must overlay the ground truth value to be accepted as prediction. Often IoU threshold of 0.5 is used, but it results in noisy prediction. By increasing the threshold some of the noisy predictions can be discarded, but with a cost of losing some of the detections. Cascade R-CNN tries to solve this by running the proposition network multiple times while increasing the IoU slightly every time. This helps in solidifying the true positive detections and removing the close false positives. (Cai & Vasconcelos 2018, 6154-6162.)

2.5.3 Single-stage Detectors

Single-stage detectors use only one CNN, providing the bounding boxes directly on to the image. There is no need for separate region proposals of classification, so the CNN can be optimized for detection performance end-to-end. Single-stage detectors are known of the improvement in speed compared to two-stage detectors, and below are introduced two different detectors.

You Only Look Once (YOLO)

You Only Look Once (YOLO) is one of the fastest algorithms in object recognition running on 45 frames per second. It achieves real time usage, and Fast YOLO even achieves rate of 155 frames per second. In the YOLO algorithm, the image is divided in S x S grid. Each grid cell predicts the bounding box locations if an object falls in the specific cell. (Redmon, Divvala, Girshick & Farhadi 2016, 779.)

In Figure 11 the visualization of the YOLO algorithm can be seen, and it is described more in-detail with mathematics below.

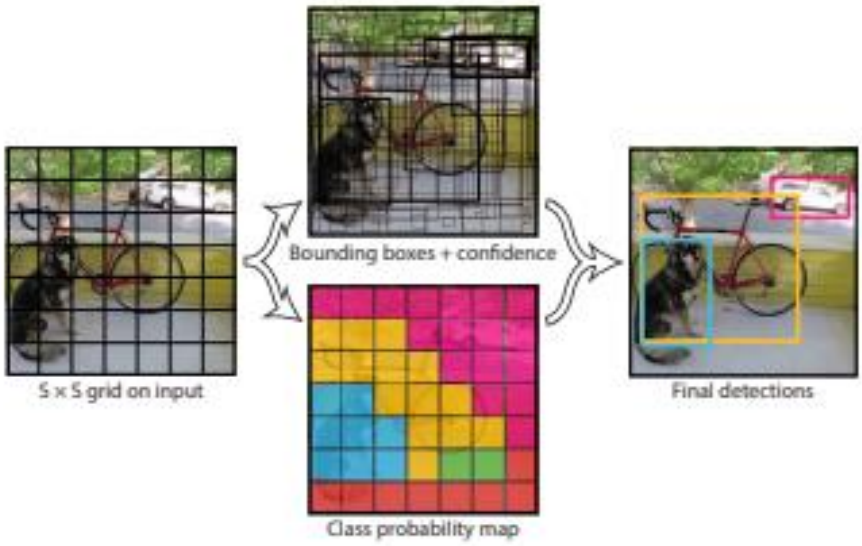

Figure 11. YOLO bounding box refinement. (Redmon, Divvala, Girshick & Farhadi 2016, 780)

The class probability C_i map as seen in Figure 11 can be calculated with Equation (15.), where probability of class C_i given object O is multiplied with the probability of object O and the intersection over union IoU. (Redmon, Divvala, Girshick & Farhadi 2016, 780.)

$$\Pr(C_i|O) \cdot \Pr(O) \cdot IoU = \Pr(C_i) \cdot IoU \qquad (15.)$$

YOLO has been improved in YOLO V2 and YOLO V3 where slight improvements were proposed. In latest YOLO V3 the base network was changed to a hybrid approach of DarkNet19 CNN with ResNet CNN, and the bounding box object recognition score was defined by logistic regression. ResNet is traditionally known as accurate but slow and large network, while DarkNet19, which was also used in Yolo V2, is extremely fast. (Redmon & Farhadi 2018, 1-6.)

RetinaNet

On the other end of single-stage detectors is RetinaNet. It was the highest performing object detector during its release (in 2018). Although it is magnitudes slower than YOLO V3, it still achieves the speed and simplicity of single-stage detectors in general but outperforms many two-stage detectors. The RetinaNet addresses challenge of training class imbalances. As one-stage detectors iterate through all locations densely, they evaluate 10^4 – 10^5 locations, while finding only

few true positive detections. This weighs the training dataset highly towards easy negatives. (Lin, Goyal *et al.* 2017, 2980; Sarkar 2018.)

For this imbalance problem focal loss was designed. Focal loss weighs easy background classifications less than object classifications. For binary classification the equation can be described with normal cross entropy loss *CE(p,y)* as in Equation (16.)

$$CE(p,y) = \begin{cases} -\log(p), & y = 1 \\ -\log(1-p), & otherwise \end{cases} \quad (16.)$$

where $y \in \{0,1\}$ is the ground truth binary label indicating whether an object is inside the bounding box and $p \in [0,1]$ the predicted probability of the confidence. (Lin *et al.* 2017, 2982.)

This loss needs to be weighed with α to balance the importance of detection as seen in Equation (17.). In the Equation (17.) the *CE* is also simplified to $CE(p,y) = CE(p_t) = -\log(p_t)$ where p_t is p when $y = 1$, or $1 - p$ otherwise. (Lin *et al.* 2017, 2982.)

$$CE(p_t) = -\alpha \log(p_t) \quad (17.)$$

Now the equation still needs to be able to differentiate between easy and hard examples, so the equation for the focal loss *FL* is defined in Equation (18.)

$$FL(p_t) = -(1-p_t)^{\gamma} \log(p_t) \quad (18.)$$

where *γ* is focusing parameter. The focusing rate adjusts the rate at which easy examples are weighed down. This effect can be seen in action in Figure 12. (Lin *et al.* 2017, 2980, 2982.)

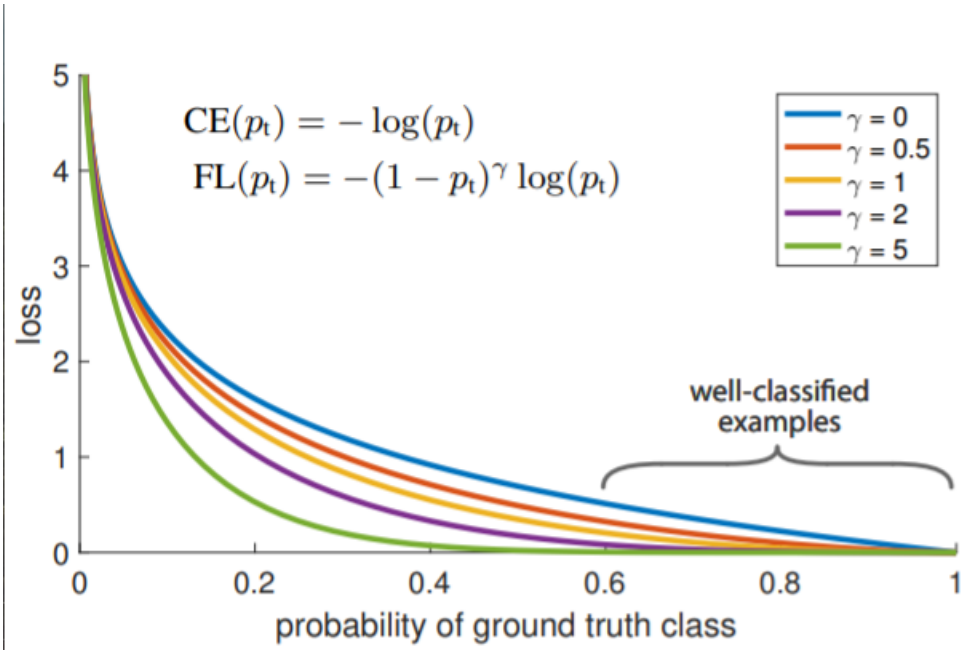

Figure 12. Focal loss visualized. (Lin *et al.* 2017, 2980)

Now the class imbalance problem has been tackled better. The base network is image pyramid of features, which basically scales the input image to different octaves, and predicts the object for each octave taking cumulative scores. This is visualized in Figure 13.

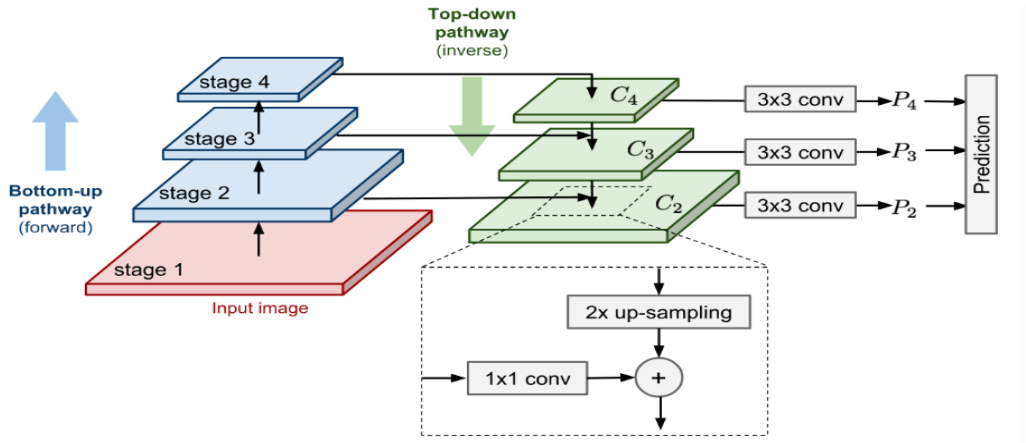

Figure 13. Featurized image pyramid architecture. (Sarkar 2018; Lin, Dollár *et al.* 2017, 2117-2125)

3 IMPLEMENTATION OF AUTONOMOUS MOBILE ROBOT

3.1 Aim of the Research

The aim in the work was to show that it is possible to implement autonomous mobile robot with a low budget. In the work there was not enough time to perform comprehensive research on how choice of planner algorithm affects the actual steering and path planning behaviour. Planner is an algorithm which is responsible for planning path from location *x* to location *y*. Good planner is able to create optimal path, which is short, smooth and robust. (Sariff & Buniyamin 2006, 1-3.)

The main focus on parameter tuning was to create as efficient map as possible with the lowest possible computational load. As the robot was only able to use visual odometry with Xbox Kinect -camera, it was important to choose efficient algorithm that would provide sufficient speed for real-time navigation.

3.2 The Hardware

3.2.1 Microsoft Kinect

Microsoft Kinect is RGB-D camera introduced by Microsoft for Xbox 360 gaming platform. Its purpose was to do motion sensing, enabling players to play with their movements. Due to its cheap price, Kinect has also been used for computer vision purposes. Its capable for producing 30 Frames-Per-Second (FPS) with resolution of 640 x 480 pixels. (Litomisky 2012, 1-4.)

In the study *Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications* (2012) Khoshelham and Elberink researched properties of Kinect for indoor mapping purposes. They did an exclusive analysis of the camera properties and came with conclusions that the mapping with Kinect should be done within 1 to 3 meters from the camera. Random error of depth measurements is increased quadratically up to 4 cm at maximum range of 5 meters, while depth resolution is also decreased quadratically. These errors happen on calibrated

camera, and are higher without calibration. For this reason Khoshelham and Elberink suggest that calibration is done for both RGB and IR cameras. (Khoshelham & Elberink 2012, 1437-1454.)

3.2.2 Mobile Robot

The mobile robot consisted of SunFounder PiCar-V mobile robot car, Xbox Kinect camera, and Raspberry Pi 3 card computer. In Figure 14 the mobile robot can be seen. The robot was fully self-built. The stand for Kinect was 3D printed.

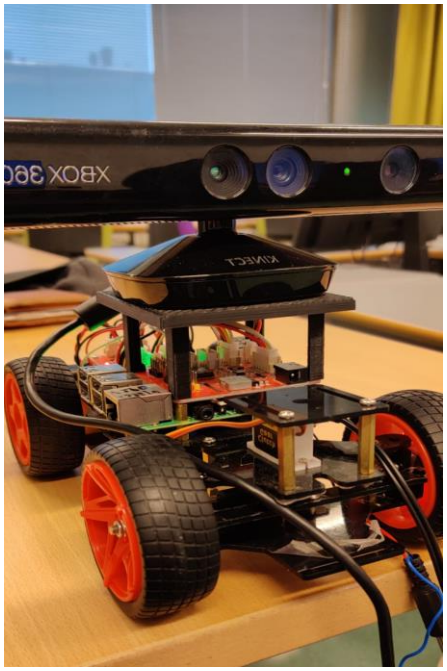

Figure 14. Mobile robot used in the project.

3.2.3 Raspberry Pi

Raspberry Pi (RPi) was in charge of receiving control commands in ROS standard format, and converting them to correct form for servo motor, and the DC motors of the robot. The actual data processing was done on HP laptop 15-n00so, which is not the most powerful, but still nonetheless has better computational capacity than RPi. The computer was not powerful enough to carry on the experiment until the very end but freezes due to unknown error when the full stack is running. Linux distribution Ubuntu was required to run ROS, so it was done as a

clean install to the laptop. Linux is an operation system that is popular due to its open source status, where everyone can contribute for the greater good. Linux is a common operating system within software developers because of its transparency and customizability.

3.3 The Software

As data processing and car control was done in separate devices, there needed to be a way to transfer the data. The control commands were transferred over wireless network to the RPi. Both devices were connected to the same ROS network, where topics and data could be published to a common IP address. This enables dynamic configuration of the wanted scripts, and transparent data management. Figure 15 depicts very rough and simplified version of how the system gathers data and provides velocity commands to the vehicle.

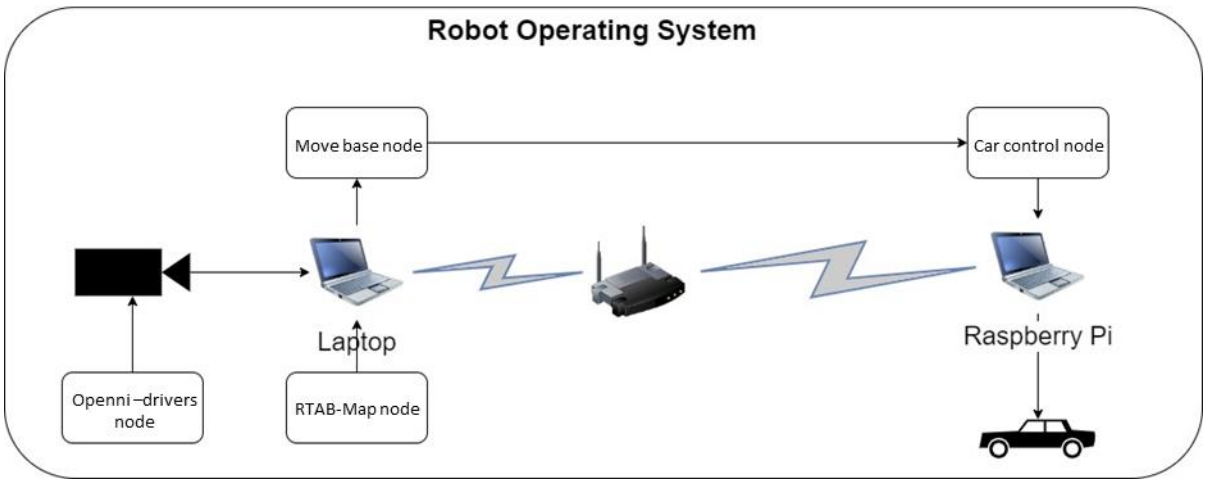

Figure 15. ROS and hardware-software architecture

After establishing the skeleton of ROS environment and good data pipelines it is relatively easy to add new sensors or add different kind of robot to be controlled.

3.4 The Obstacle Track

The obstacle track was the real platform where the performance of the mobile robot would be evaluated. In Image 1 the setting of the track can be seen.

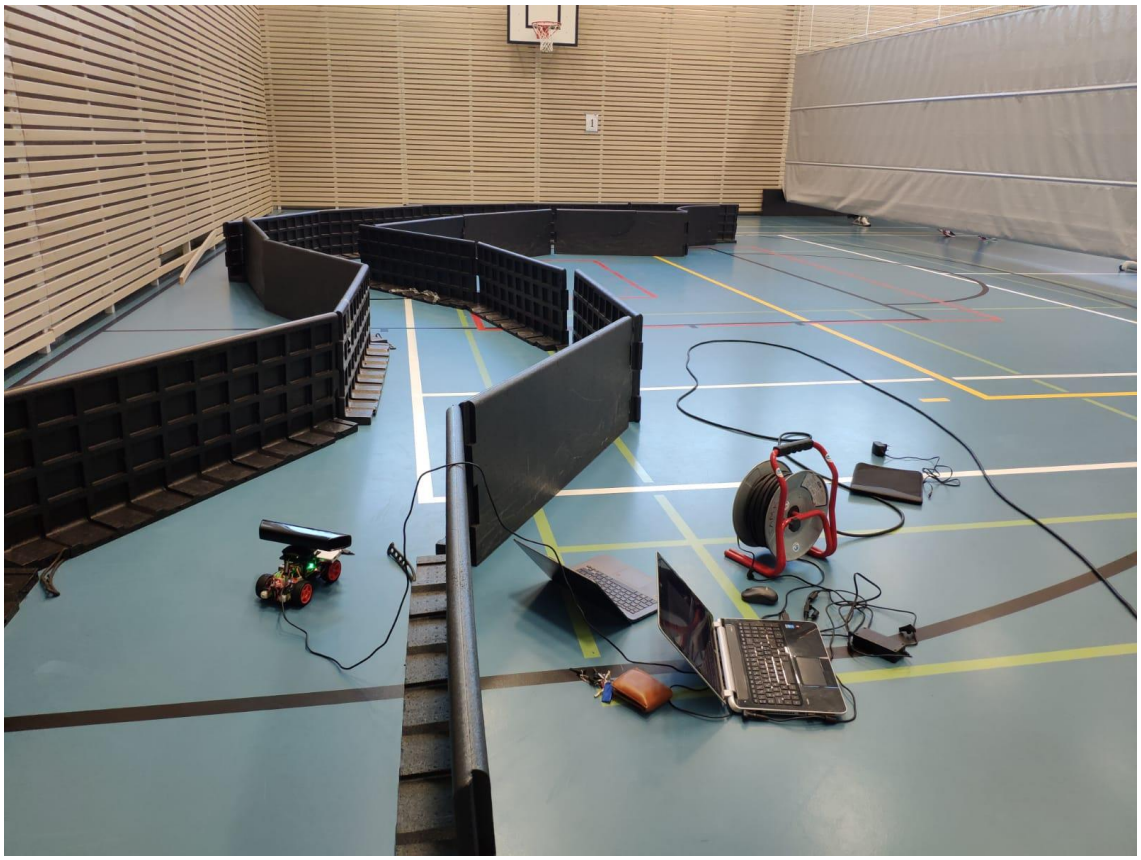

Image 1. Mobile robot evaluation track.

One field from exercising hall was rented for two hours, which created a lot of pressure to do things right and well at the first take. First the mobile robot was driven by hand through the track while mapping. Next the robot was supposed to navigate through the track autonomously. There was practically no time for exhaustive parameter tuning due to short amount of time. Personal laptop was used to debugging the RPi code as it was not flawless, and the other laptop was used for processing the data and hosting the ROS environment.

In Image 1 there can also be seen reflections on the hard floor. This made big difference in the environment where parameters had already been optimized. Previously the camera had been used with artificial lighting only. The floor is also textureless and no keypoints could be detected from it.

3.5 Performance

Sensory output of the track ride was recorded as a ROSBAG file, so the data could also be processed offline. The map that was initially generated can be seen in Figure 16. It includes a lot of noise, and is not very convenient map to be navigated in. However this was all that could be gathered, so next step was to do path planning in the map. Path planning required also some parameter tuning in order to compensate for the noisy map, and path planning node ran for around 2 meters, until the computer completely froze. Due to limited time it was no longer possible to try the navigation again.

Later the ROSBAG was replayed offline multiple times to optimize the generated map, and it was found that by adjusting parameters much better grid map could be generated. The newly generated map can be seen in Figure 17.

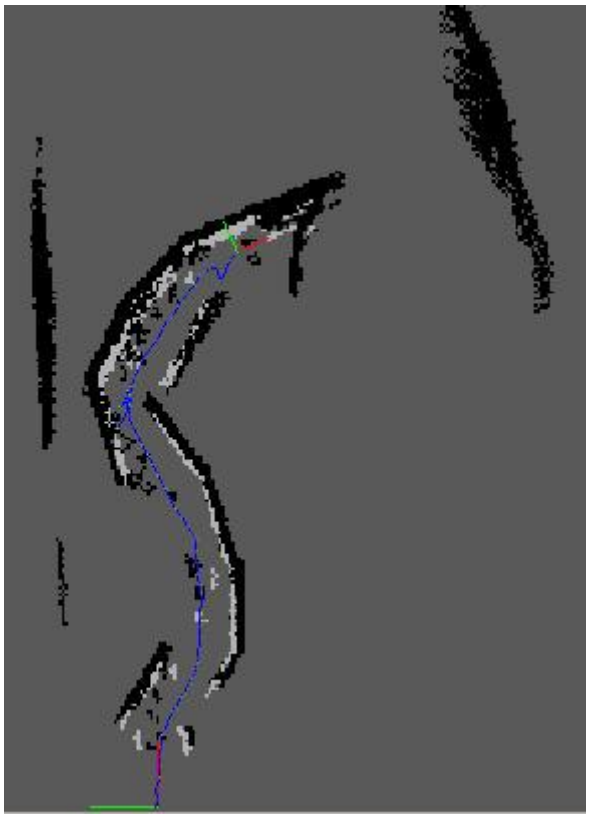

Figure 16. Initial map of the track.

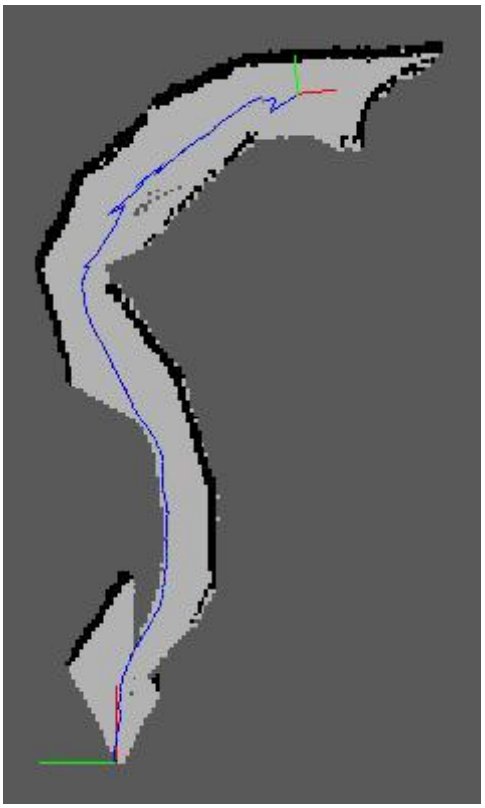

Figure 17. Optimally parametrized map.

It is also important to note that only the parameters affecting real-time performance were adjusted, and no post processing was done to achieve this map seen in Figure 17. In RTAB-Map there are plenty of post processing options, such as finding more loop closures and refining neighbour keypoints which should be used before putting a map into deployment.

Due to this freezing of the laptop, the same experiment of mapping and navigation was attempted at home, but this time it was seen that the laptop could not sustain the heavy load and kept freezing. CPU usage was monitored during the freezing, but no obvious reason was found besides the fact that path planning requires tremendous amount of memory. It seems that the freezing is simply lack of computational power to do real time image processing.

It was then concluded that this experiment should not be run further with current equipment as there is no point in trying to revive 5+ year old low-end laptop to achieve bare minimum SLAM performance.

Next the optimization improvements that lead to map in are discussed in greater detail.

3.6 Parameter Tuning

Parameter tuning means adjusting the values for local planner, global and local costmaps and for RTAB-Map. In practise this means defining the dimensions of the robot, modelling the environment, the way map is generated and what optimizers and algorithms to use. There are over hundred parameters that can be tuned, some of them requiring expert-level knowledge in specific sub-domains of computer vision or optimizer behaviour. Most tuning was done experimentally and based on the knowledge that was gathered during the project.

Parameter tuning was done initially in around 15 square meter room before going to the testing track. A ROSBAG file was recorded where the camera was walked around for 100 seconds. The bag file includes a lot of lighting changes, twists and

unnatural poses making it a difficult task for mapping. Using ROSBAG file for evaluating different parameters gives a lot of stability in testing, as it provides consistently always the same data to be evaluated against. Evaluation criteria for parameters was simply the quality of the generated map. In this case the clearness of the map was the main criteria.

More in-depth parameter tuning is divided into separate chapters below.

3.6.1 RTAB-Map Parameters

RTAB-Map's parameters are included in Appendix 1. As it was seen in chapter 3.5, the map could not be created well with the same parameters as in artificial lighting settings at home. In general *rtabmap/rtabmap*-namespace parameters are individual for each project they are not further discussed. The tunable parameters that were used in the project are listed below. All parameters and their descriptions can be found from the RTAB-Map itself.

Registration (*Reg*)

Registration parameter space contains common data point registration parameters. Data registration attempts to align two sets of data, and approaches for this are listed in Chapter 2.3.1 and 2.3.2.

Reg/Force3DoF defines whether the SLAM is generated only for 3 axis, x, y and theta. As robot position is not changing in height, this will speed up computations. *Reg/Strategy* defines the algorithm responsible for registration. By default the strategy is 0, which means it uses the cheapest visual registration algorithm.

RGB-D (*RGBD*)

RGB-D defines parameters that are used for RGB-D SLAM. *RGBD/ProximityBySpace* defines whether detections will be searched over locations near in space (in working memory). This increases computational cost and is set to *false* to provide maximum performance. *RGBD/ProximityPathMaxNeighbors* defines the maximum amount of neighbour nodes to be compared against to find valid

detections. Increasing this value increases the required computations, but also reduces the noise in the map by finding consistency.

Grid

Grid defines the occupancy grid properties. *Grid/Scan2DUnknownSpaceFilled* is used only with 2D laser scans. This defaults to *false* but is set to *false* explicitly to make distinction of the setup. *Grid/FromDepth,* if set to *true*, create occupancy grid from depth images. This is required to be *true* with RGB-D, as otherwise it is created from laser scan. *Grid/MaxGroundHeight,* maximum ground height to be considered as a ground. Prevents detecting the differences in ground as object. Set to 0.2 to increase the ground threshold, as there was many light reflections from the ground, and all objects were higher than 0.3 meters. *Grid/MaxObstacleHeight,* maximum obstacle height to be detected. Obstacles higher than this are ignored. This can be used because the environment object heights are known. *Grid/NormalsSegmentation,* if set to *true*, ground is segmented from obstacles using point normals. Set to false, as it lead to multiple false detections due to reflections. *Grid/RayTracing,* if set to *true,* fills the unknown space between the sensor and occupied cells. This is set to true to make more noise free map. *Grid/3D,* if set to *true,* creates 3D occupancy grid. 3D occupancy grid provides better accuracy than 2D occupancy grid according to the author, and did not bring drop framerate per second level according to tests.

3.6.2 Path Planner Parameters

As in Appendix 2 can be seen the Dynamic Window Approach (DWA) planner was used instead of usual Trajectory Rollout as it provided to be more functional. Planner's work is to generate optimal path from point A to point B. Planner also must take dynamic environment into consideration when making a plan for the path, such as moving obstacles. Both planners work by forward simulating the future velocities and positions. DWA simulates only for one simulation step while Trajectory Rollout simulates for the whole future forward simulation duration. This leads to lower memory usage when using DWA.

DWAPlannerRow/controller_frequency defines frequency at which this controlled will be called in Hz. This requires a lot of computational power and defaults to 20 Hz. It is set explicitly to 10 Hz, as the computer was not capable of updating the controller at rate of 20 Hz.

3.6.3 Sensor Configurations, Costmaps and Robot Dimensions

Finally, Appendix 3 includes parameters for configuring the sensors, robot dimensions and costmaps. The parameters in this Appendix are responsible for navigating successfully in a map.

Costmaps

Costmaps have mostly similar parameters, and they are described in general marked under *costmap* namespace. *costmap/update_frequency* determines the frequency map needs to be updated. This requires a lot of computational power and was the to 0.5 Hz. *costmap/publish_frequency* determines the frequency for publishing the map information. This results in slow and unsmooth behaviour of the mobile robot, as it is defining the steering only every 2 seconds. This should be synchronous with *planner/control_frequency*, but in this case it was not possible. This resulted in unexpected behaviour. *costmap/static_map,* if set to *true,* static map is used. Should be set to *true* when area is static and its mapped beforehand. In this case map was generated during the ride. *costmap/rolling_window* determines whether only pre-determined area of the map is generated around the robot. *costmap/resolution* determines the resolution of the map in meters/cell.

Robot dimensions

Robot dimensions have to be declared to make sure correct path planning trajectory is calculated. One important takeaway point for this project is that the robot must be treated as circular, as minimum Kinect sensing range is roughly 50cm, and obstacles must not be closer than that.

4 THE STATE-OF-ART METHODS

In this chapter the current state of navigation and object detection in forest environment is reviewed. Often the approaches are hard to be compared against each other. Problem in evaluating accuracies and errors in forest environments come from unreliable ground truth estimation. Some ways would be to map the forest with expensive and heavy laser equipment, but still in that case there would be always introduced small errors, as forest is changing constantly. Also, the domains in which accuracies are presented in research studies differ, and it is nearly impossible to set one metric to measure the superiority of one algorithm against another in forest environment.

The area of visual SLAM with monocular cameras is studied particularly heavily, since robot equipped with a camera is relatively cheap in hardware and in computational cost.

4.1 Monocular Camera SLAM

In study conducted by Garforth and Webb, the state-of-art algorithms Large Scale Direct Monocular SLAM (LSD-SLAM), ORB-SLAM2, Direct Sparse Odometry (DSO) and Semi-Direct Visual Odometry (SVO) were evaluated in forest environment. The datasets were from a view of a mobile robot. It was found, that only ORB-SLAM2 and DSO were able to run successfully, and on only some of the easiest tasks. This suggests that SLAM as a stand-alone navigation algorithm is not viable choice in forest environment. One of the main problems were change of lighting, and lack of loop closures. (Garforth & Webb 2019, 1794-1800.)

4.2 LiDAR SLAM

The combination of IMU + LiDAR SLAM was evaluated against the traditional IMU + GNSS approach in study from Tang et al. in 2015. It was performed on data from a ground travelling robot in Southern Finland borealis forest. In the study Improved Maximum Likelihood -algorithm for LiDAR SLAM was used. It

was seen that in mature forest the IMU + LiDAR SLAM outperformed by roughly 38% better than IMU + GNSS. However, in open forest areas, due to the lack of keypoints, IMU + LiDAR SLAM failed to outperform IMU + GNSS. (Tang *et al.* 2015, 4588-4606.) For these reasons the study supports the idea of sensor fusion and combined algorithms.

This sensor fusion between GNSS, IMU and LiDAR was done by Babin et al. in 2019. Their dataset was gathered in subarctic conditions in Forêt Montmorency research forest. Aim of this study was to create 3D map of snowy environment. (Babin *et al.* 2019, 1-14.)

In snowy environment it is difficult to extract features for SLAM, and for this reason ICP algorithm for LiDAR is used. The ICP however is susceptible for global drift. This would often be compensated by detecting loop closures and doing a pose graph optimisation. However, in this environment loop closure is not possible as route is fairly straight trajectory, a new approach needs to be implemented. (Babin *et al.* 2019, 1-14.)

GNSS is great for global accuracy in areas with low occlusions, and IMU is great for knowing the changes in pose. As ICP's goal is to minimize a loss function, GNSS and IMU measurements can be fused to the function of translation T between consecutive point clouds to reduce the global drift of the ICP. This in practise means adding known imaginary points to the point cloud to correct the drift. In this study adding GNSS and IMU to the loss function is called as adding penalty. (Babin *et al.* 2019, 1-14.)

Great example of the drift can be seen in Figure 18 where the points inside red circle and pointed by arrow are supposed to be the same. Without GNSS and IMU penalty the drift is large, resulting in misalignment of the end and start points. (Babin *et al.* 2019, 11.)

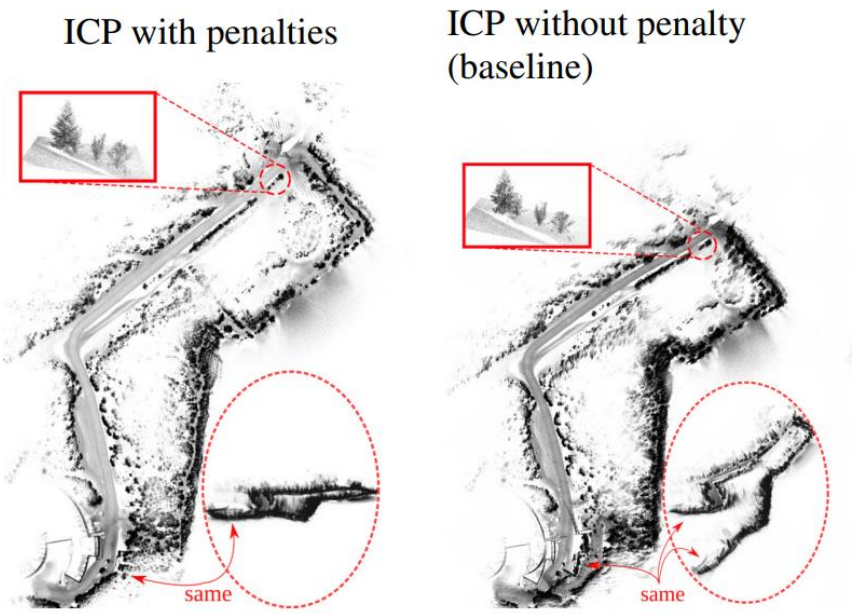

Figure 18. ICP-generated maps with penalty and without penalty (Babin *et al.* 2019, 11)

The latter research also suggests that sensor fusion is an important part of the navigation stack.

4.3 Deep Learning

Traditional SLAM requires tuning of the parameters manually or picking the correct feature extraction algorithm for the specific domain. Strength in deep learning is the ability to generalize, and this leads to good performance under different weather conditions or in unseen environments.

Unmanned aerial vehicles (UAV) have been used for deep learning research in forests, as they are fast and light for things such as forest mapping or search-and-rescue. Trail navigation with the help of DNN was researched by Smolyanskiy, Kamenev, Smith and Birchfield in 2017. They used lightweight Jetson TX1 computer for controlling a micro aerial vehicle (MAV) to follow a trail in forest for 1 kilometer autonomously. It uses modified ResNet-18 DNN architecture, which they have named as TrailNet. TrailNet outputs 3 orientations (left, straight, right) with respect to the trial, and 3 lateral offsets (shifted left, centered, shifted right). It used the principles of transfer learning by training orientation with separate

dataset from hiking dataset, and fine tuning the weights with custom-made dataset from different types of trails. The MAV found YOLO to be best for object recognition and DSO for dynamic object detection and avoidance. This navigation stack provided full autonomy for the whole trail. (Smolyanskiy, Kamenev, Smith & Birchfield 2017, 4241-4247.)

Study also worth mentioning in field of deep learning is by by Maciel-Pearson et al in 2019. In the study it was shown in simulation environment, how deep reinforcement learning could be used under forest canopy. They successfully ran their Extended Double Deep Q-Network (Extended DDQN) under snowy, dusty and foggy weather condition beating the previous DDQN approaches. It claims to also perform under unseen environments. In the study it was trained in dense forest, but also performed well in savanna and farm field. (Maciel-Pearson *et al.* 2019.)

The latter study provides exciting results as robot should be able to perform in unseen environments without specific re-training for every environment separately. The experiment was however only carried out in simulated world and needs a lot more research.

5 CONCLUSIONS

In this thesis the most important algorithms in navigation and their applications in real world were introduced. Also the implementation of autonomous mobile robot was introduced. A lot of research especially with new algorithms is done in simulations, which is quite natural as real world brings plenty of challenges. These same challenges were faced when the mobile robot was built, and mostly these challenges can be solved with money. In simulation there is no need to make financial investments, which leads it being first way of testing out new things. This distinction between application in simulation and real life must be recognized. Especially in forestry there must be clear roadmap and proof of concept on how to monetize the proposed approach before investing in expensive hardware.

Based on studies included in this thesis, it seems that the forest is gaining most of its popularity from drone or handheld mapping in order to gain sense of forest health and condition. Some studies, as also described in this thesis, supports the idea of autonomous navigation in forest for search and rescue and some studies just simply focus on the improving the quality of autonomous navigation. The effort put to the area of purely improving the quality of autonomous navigation in the forest is however not on comparable level with navigation in urban areas.

In Umeå University the idea of autonomous navigation in forest was researched thoroughly in around years 2004-2011, and they proposed quite extraordinary ways for many problems (Hellström & Ringdahl 2004; Hellström, Johansson & Ringdahl 2006, 603-614; Ali, Georgsson & Hellstrom 2008, 560-565). This research however was discontinued. Hellström comments that there was a lack of interest from industry, as the market is too narrow and return on investment is small.

One thing that could attract more people in researching this problem could be the creation of some universal evaluation metrics. This would enable the comparison of different algorithms in order to make conclusions about its performance. Currently there are no good ground truth datasets that could be used as the benchmarking dataset. This phenomena of the competition has been seen in machine learning since there are plenty of benchmarking datasets such as ImageNet,

MNIST, Fashion-MNIST etc. Even if new benchmarking datasets would be introduced, it is unclear whether it would attract more people to tackle the problems. The SLAM community in itself is quite small consisting of researchers, and there is no clear pathway yet to dive into customer market as an entrepreneur or small company.

In the writer's opinion machine learning has shown promising results so far in very different domains, and if from simulation domain the learned knowledge could be transferred into real world it could open new possibilities. It all comes down to how well we are able to continue in creating more intelligent systems in general, and how many people are willing to integrate those systems in to forest environments. Currently it is unlikely that the autonomous vehicles will be entering forest in near future based on the studies, but in general the forest mapping and forest health management are gaining financial interest. With breakthroughs often comes the money.

CITATIONS

Ali, W., Georgsson, F. & Hellstrom, T. 2008. Visual tree detection for autonomous navigation in forest environment. IEEE, pp. 560.

Babin, P., Dandurand, P., Kubelka, V., Giguere, P. & Pomerleau, F. 2019. Large-scale 3D Mapping of Sub-arctic Forests. arXiv preprint arXiv:1904.07814. pp. 1-14.

Bentley, J.L. 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM. 18, pp. 509-517.

Biber, P. & Straßer, W. 2003. The normal distributions transform: A new approach to laser scan matching. IEEE, pp. 2743.

Cai, Z. & Vasconcelos, N. 2018. Cascade r-cnn: Delving into high quality object detection. pp. 6154.

Calonder, M., Lepetit, V., Strecha, C. & Fua, P. 2010. Brief: Binary robust independent elementary features. Springer, pp. 778.

Coops, N. 2013 Dr. Nicholas Coops: Introduction to LiDAR Technology Youtube-video. https://www.youtube.com/watch?v=HfV7jJgrw4Q.

Dennis Jr, J.,E. & Schnabel, R.B. 1996. Numerical methods for unconstrained optimization and nonlinear equations. Siam.

Doidge, N. 2007. The brain that changes itself: Stories of personal triumph from the frontiers of brain science. Penguin.

Garforth, J. & Webb, B. 2019. Visual appearance analysis of forest scenes for monocular slam. IEEE, pp. 1794.

Grinblat, G.L., Uzal, L.C., Larese, M. & Granitto, P.M. 2016. Deep learning for plant identification using vein morphological patterns. Computers and Electronics in Agriculture. 127, pp. 418-424.

Grisetti, G., Kummerle, R., Stachniss, C. & Burgard, W. 2010. A tutorial on graph-based SLAM. IEEE Intelligent Transportation Systems Magazine. 2, pp. 31-43.

Hellström, T., Johansson, T. & Ringdahl, O. 2006. Development of an autonomous forest machine for path tracking. Springer, pp. 603.

Hellström, T. & Ringdahl, O. 2004. Follow The Past: a Path Tracking Algorithm for Autonomous Forest Vehicles.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y. & Guadarrama, S. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. pp. 7310.

Huttunen, H. 2019. Pattern Recognition and Machine Learning - Slide set 5: Neural Networks and Deep Learning. Accessed: 15.01.2020. http://www.cs.tut.fi/courses/SGN-41007/slides/Lecture5.pdf.

Kriesel, D. 2007. A Brief Introduction to Neural Networks.

Labbé, M. & Michaud, F. 2019. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. Journal of Field Robotics. 36, pp. 416-446.

Lavrenko, V. 2015 kNN.15 K-d tree algorithm Youtube-video. Cited in 25.01.2020. https://www.youtube.com/watch?v=Y4ZgLlDfKDg.

Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B. & Belongie, S. 2017. Feature pyramid networks for object detection. pp. 2117.

Lin, T., Goyal, P., Girshick, R., He, K. & Dollár, P. 2017. Focal loss for dense object detection. pp. 2980.

Lowe, D.G. 2004. Distinctive image features from scale-invariant keypoints. International journal of computer vision. 60, pp. 91-110.

Maciel-Pearson, B.G., Marchegiani, L., Akcay, S., Atapour-Abarghouei, A., Garforth, J. & Breckon, T.P. 2019. Online Deep Reinforcement Learning for Autonomous UAV Navigation and Exploration of Outdoor Environments. arXiv preprint arXiv:1912.05684.

Matthys, D. 2001. LoG filter. Accessed: Feb 2, 2020. https://academic.mu.edu/phys/matthysd/web226/Lab02.htm.

Namin, S.T., Esmaeilzadeh, M., Najafi, M., Brown, T.B. & Borevitz, J.O. 2018. Deep phenotyping: deep learning for temporal phenotype/genotype classification. Plant methods. 14, pp. 66.

Ostovar, A. (2019) Object Detection and Recognition in Unstructured Outdoor Environments. Umeå University.

Ostovar, A., Talbot, B., Puliti, S., Astrup, R. & Ringdahl, O. 2019. Detection and classification of Root and Butt-Rot (RBR) in stumps of Norway Spruce using RGB images and machine learning. Sensors. 19, pp. 1579.

Pierzchała, M., Giguère, P. & Astrup, R. 2018. Mapping forests using an unmanned ground vehicle with 3D LiDAR and graph-SLAM. Computers and Electronics in Agriculture. 145, pp. 217-225.

Pound, M.P., Atkinson, J.A., Townsend, A.J., Wilson, M.H., Griffiths, M., Jackson, A.S., Bulat, A., Tzimiropoulos, G., Wells, D.M. & Murchie, E.H. 2017. Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. Gigascience. 6, pp. gix083.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. 2016. You only look once: Unified, real-time object detection. pp. 779.

Redmon, J. & Farhadi, A. 2018. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767. pp. 1-6.

Ren, S., He, K., Girshick, R. & Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. pp. 91.

Rohrbach, F. 2015. LiDAR Footprint Diameter. Accessed: Feb 2, 2020. https://felix.rohrba.ch/en/2015/lidar-footprint-diameter/.

Rosin, P.L. 1999. Measuring corner properties. Computer Vision and Image Understanding. 73, pp. 291-307.

Rosten, E. & Drummond, T. 2006. Machine learning for high-speed corner detection. Springer, pp. 430.

Rublee, E., Rabaud, V., Konolige, K. & Bradski, G. 2011. ORB: An efficient alternative to SIFT or SURF. Ieee, pp. 2564.

Saha, S. 2018. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Accessed: Feb 2, 2020. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Sarkar, D. 2018. A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning. Accessed: Feb 4, 2020. https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

Smolyanskiy, N., Kamenev, A., Smith, J. & Birchfield, S. 2017. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. IEEE, pp. 4241.

Song, X., Zhang, G., Liu, F., Li, D., Zhao, Y. & Yang, J. 2016. Modeling spatio-temporal distribution of soil moisture by deep learning-based cellular automata model. Journal of Arid Land. 8, pp. 734-748.

Tang, J., Chen, Y., Kukko, A., Kaartinen, H., Jaakkola, A., Khoramshahi, E., Hakala, T., Hyyppä, J., Holopainen, M. & Hyyppä, H. 2015. SLAM-aided stem mapping for forest inventory with small-footprint mobile LiDAR. Forests. 6, pp. 4588-4606.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M. & Hoffmann, G. 2006. Stanley: The robot that won the DARPA Grand Challenge. Journal of field Robotics. 23, pp. 661-692.

Zhang, Z. 1994. Iterative point matching for registration of free-form curves and surfaces. International journal of computer vision. 13, pp. 119-152.

Zucconi, A. 2015. Understanding the Gaussian distribution. Accessed: Feb 2, 2020. https://www.alanzucconi.com/2015/09/09/understanding-the-gaussian-distribution/.

APPENDICES

Appendix 1. RTAB-Map Parameters

```
rtabmap:
  rtabmap:
    frame_id: camera_link
    map_frame_id: map
    odom_frame_id: camera_link

    subscribe_depth: false
    subscribe_rgb: false
    subscribe_rgbd: true
    subscribe_scan: true

    publish_tf: false

    Reg:
      Strategy: 0
      Force3DoF: true                   # Use 3 DOF (2D)

    RGBD:
      ProximityBySpace: false
      ProximityPathMaxNeighbors: 10

    Grid:
      Scan2dUnknownSpaceFilled: false
      FromDepth: true
      MaxGroundHeight: 0.2
      MaxObstacleHeight: 0.5
      NormalsSegmentation: false        # Use normal to segment ground
      RayTracing: true                  # Fill space between obstacles
      3D: true
```

Appendix 2. Planner Parameters

```
DWAPlannerROS:
  # Speed limits
  max_vel_x: 1.0
  min_vel_x: 0.01
  max_vel_theta: 1
  path_distance_bias: 1.0
  goal_distance_bias: 0.8
  occdist_scale: 0.01

  # Acceleration limits
  acc_lim_theta: 3.2
  acc_lim_x: 2.5
  acc_lim_y: 2.5

  # Goal tolerance
  xy_goal_tolerance: 0.2
  yaw_goal_tolerance: 0.2

  controller_frequency: 10
  ho-lo-nomic_robot: false  # Is robot able to move in all directions
freely?
```

Appendix 3. Costmap and Sensor Parameters

```
# Common parameters to be used in local and global costmaps

obstacle_range: 2.5     # Max distance for obstacle to put in costmap
ray-trace_range: 0.2    # How much clear space for tries to achieve in
front of itself
# foot-print: [[x0, y0], [x1, y1], ... [xn, yn]] # Dimensions of the
robot
robot_radius: 0.2 # Treat robot as circular
inflation_radius: 0.05  # Treat all obstacles further than x-meters as
equal

# Set navigation to accept unknown goals
NavfnROS:
  allow_unknown: true

observation_sources: point_cloud_sensor

point_cloud_sensor: {
  sensor_frame: camera_link,
  data_type: PointCloud2,
  topic: camera/depth_registered/points,
  marking: true,
  clearing: true}

global_costmap:
  global_frame: map  # Frame where costmap is run in
  robot_base_frame: camera_link # What frame costmap is referenced on
  update_frequency: 0.5 # Frequency in Hz
  publish_frequency: 0.5
  static_map: false  # true - using existing map or map server
  rolling_window: true  # Not static map
  point_cloud_sensor/topic: /camera/depth_registered/points
  transform_tolerance: 2.0

local_costmap:
  global_frame: map  # odom frame (our robots frame :D )
  robot_base_frame: camera_link # Well, base frame
  update_frequency: 0.5
  pub-lish_frequency: 0.5  # Hz, how often costmap publishes
visualisation info
  static_map: false
  roll-ing_window: true  # Costmap center will stay middle of the
robot as moving
  width: 4.0         # Width of the costmap (meters/cell)
  height: 4.0        # Height of the costmap (meters/cell)
  resolution: 0.0005  # Resolution (meters/cell)

  point_cloud_sensor: {
    expected_update_rate: 0.5}
```