TÉCNICO LISBOA

A Framework for Mobile Robots Localization and Mapping

João Santana Tavares de Castro

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisors: Prof. Carlos Baptista Cardeira
Prof. Paulo Jorge Coelho Ramalho Oliveira

Examination Committee

Chairperson: Prof. Carlos Frederico Neves Bettencourt da Silva
Supervisor: Prof. Carlos Baptista Cardeira
Member of the Committee: Prof. José António Da Cruz Pinto Gaspar

June 2020

Acknowledgments

This thesis was researched and written with the support and orientation provided by my supervisors, Prof. Carlos Baptista Cardeira and Prof. Paulo Jorge Coelho Ramalho Oliveira, and the help of the laboratory staff, Luís Jorge Bronze Raposeiro and Christo Camilo.

I would also like to thank my family and friends for all the support provided throughout my years of study and the process of writing this thesis.

Uma Arquitetura para Localização e Mapeamento de Robôs Móveis

Resumo

Um dos aspetos mais importantes de um robô móvel é a capacidade de se localizar num ambiente desconhecido. Para isso, é necessário produzir um mapa do ambiente e, ao mesmo tempo, localizar-se dentro desse mapa. Este problema chama-se localização e mapeamento simultâneo (SLAM). Nesta tese foi criada uma arquitetura que usa um robô móvel instrumentado de forma a ter a capacidade de se localizar e produzir um mapa do ambiente envolvente. O robô móvel é um pequeno carro controlado remotamente e equipado com um Raspberry Pi, um radar a laser, sensores de odometria e marcadores usados por um sistema de posição.

Foram implementados três algoritmos de localização e mapeamento concorrente (CLM) que tentam localizar o carro e produzir um mapa do ambiente envolvente. Para desempenhar estas funções, utilizam três algoritmos de registo e um modelo do carro, todos também implementados nesta tese.

Também foi desenvolvido um conjunto de blocos Simulink usados para implementar um dos algoritmos de CLM e um controlador. Estes blocos foram desenvolvidos para trabalhar com outros dois carros similares.

Palavras-chave: localização e mapeamento concorrente, algoritmos de registo, robôs móveis, instrumentação

Abstract

One of the most important features of a mobile robot is the capability to localize itself in an unmapped environment. To do it, it is necessary to produce a map of the environment while at the same time localizing itself inside that map. This problem is called simultaneous localization and mapping (SLAM). In this thesis, a framework was constructed using a mobile robot that was instrumented to give it capabilities of autonomous self-localization and mapping of the surrounding environment. The mobile robot is a remote-controlled (RC) small toy car equipped with a Raspberry Pi, a laser scanner, odometry sensors, and markers to allow a positioning system to determine the car's pose.

Three Concurrent Localization and Mapping (CLM) algorithms were implemented that attempt to localize the car and produce a map of the surrounding environment. These make use of three registration algorithms and a model of the car, all of which were implemented in this thesis.

A Simulink software suite was also developed that was used to implement one of the CLM algorithms and a controller. This software suite was also extended to work with two other similar cars.

Keywords: concurrent localization and mapping, registration algorithms, mobile robotics, instrumentation

Contents

List of Tables

List of Figures

Nomenclature

Greek symbols

α	Slip angle of tire.
β	Angle between a surface and the surface normal to a point's ray.
δ	Steering angle of the car's front wheels.
Γ	Matrix used to find ICP's transformation.
γ	Normal confidence value.
γ_c	Curvature of the surface around a point.
γ_e	Normal angular error component of the normal confidence value.
γ_f	Flatness component of the normal confidence value.
Λ	Second matrix of SVD decomposition.
λ	Damping factor in the Levenberg–Marquardt algorithm.
ν	Reduction parameter for the damping factor in the Levenberg–Marquardt algorithm.
Ω	Error vector scaling matrix.
ψ	Orientation of the car.
σ	Standard deviation.
θ	Polar angle.

Roman symbols

$\bar{p}$	Mean of a set of points from the data point cloud.
$\bar{q}$	Mean of a set of points from the model point cloud.
Δt	Sample time.
$\dot{x}$	Car's velocity along the x axis.
$\dot{x}_i^b$	Car's initial velocity in it's own reference frame.

$\dot{y}$ Car's velocity along the y axis.

$\hat{t}^a$ Linearization point of the measurement function.

$\hat{y}$ State-space estimated output vector.

$\hat{y}_e$ Encoder's output.

$[C]$ Diagonal concatenation of covariance matrices.

$[t^a]$ Concatenation of absolute poses.

$[t^e]$ Concatenation of estimated relative poses from the absolute poses.

$[t^r]$ Concatenation of relative poses from a registration algorithm.

$\vec{n}$ Surface normal around a point.

a Distance between the front axis and the car's center of mass.

A, B, C, D, F, G State-space matrices.

b Distance between the rear axis and the car's center of mass.

C Covariance matrix.

C_α Tire cornering stiffness.

$C_{\vec{n}}$ Normal scaling factor.

$C_{c\theta}$ Circle trust-region angle difference.

C_{cc} Circle trust-region radius.

C_{d0} Car's constant drag coefficient.

C_{d1} Car's linear drag coefficient.

C_{e0} Car's constant engine coefficient.

C_{e1} Car's linear engine coefficient.

$C_{e\theta}$ Ellipse trust-region angle difference.

C_{enc} Encoder linear coefficient.

C_{ex} Ellipse trust-region x radius.

C_{ey} Ellipse trust-region y radius.

d Generalized distance between two points.

D_{max} Largest eigenvalue.

D_{min} Smallest eigenvalue.

e	Error vector.
F_d	Drag forces acting on the car.
F_e	Force produced by the engine on the wheels.
F_y^F	Side force produced by the car's front tires.
F_y^R	Side force produced by the car's rear tires.
g	Gradient vector.
H	Hessian matrix.
I	Car's rotational inertia.
I	Integral image.
J	Jacobian matrix.
K	Third matrix of SVD decomposition.
L	Kalman filter gain.
L_c	Correction matrix of the Kalman filter.
L_p	Prediction matrix of the Kalman filter.
M	Set of points from the model point cloud.
m	Car's mass.
N	Set of points from the data point cloud.
n	Number of points in a set.
P	State variables covariance matrix.
p	Point from the data point cloud.
Q	Matrix from the linearization of $[t^e]$.
Q	Process noise covariance matrix.
q	Point from the model point cloud.
R	Neighborhood radius.
R	Outputs covariance matrix.
R	Rotation matrix of coordinate transformation.
r	Polar radius or depth.

R^i_{i+1} Rotation matrix that transforms points from the reference frame of the node $i+1$ to the reference frame of the node i.

S Error vector diagonal scaling matrix.

S Innovation covariance matrix.

T Translation vector of coordinate transformation.

t Vector of coordinate transformation parameters.

t^a Absolute pose.

t^e Estimated relative pose from the absolute poses.

$T^i_{i\to i+1}$ Translation vector that transforms points from the reference frame of the node $i+1$ to the reference frame of the node i.

t^m Relative pose from the model of the car.

t^r Relative pose from a registration algorithm.

t_θ Coordinate transformation parameter that describes the rotation.

t_x Coordinate transformation parameter that describes translation along the x axis.

t_y Coordinate transformation parameter that describes translation along the y axis.

u Index of first point of neighborhood.

u State-space input vector.

u_d Car's input for the steering wheels servo motor.

u_e Car's input for the driving motor.

V Coordinate transformation matrix that transforms a vector's coordinates from the normal-tangential reference frame to the point cloud's reference frame.

v Index of last point of neighborhood.

W Score to be optimized.

w Weigh given to associated pair of points.

x Position along the x axis.

x State-space variables vector.

x^b_f Stopping distance

y Position along the y axis.

Z First matrix of SVD decomposition.

Subscripts

i, j, k Computational indexes.

x, y, z Cartesian components.

Superscripts

$\dot{}$ Derivative with respect to time.

a Absolute pose.

b In the car body's reference frame.

d In the data cloud's reference frame or with respect to the data cloud.

e Estimated pose.

e In the earth's reference frame.

m In the model cloud's reference frame or with respect to the data cloud.

r Relative pose from a registration algorithm.

T Transpose.

Glossary

CLM	Concurrent Localization and Mapping
ESC	Electronic Speed Control
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
LED	Light-emitting diode
LIDAR	Light detection and ranging
NDT	Normal Distributions Transform
NICP	Normal Iterative Closest Point
PCB	Printed circuit board
PWM	Pulse Width Modulation
RC	Remote-controlled
SLAM	Simultaneous Localization and Mapping
SVD	Single Value Decomposition
UDP	User Datagram Protocol

Chapter 1

Introduction

1.1 Objectives

The localization of a body and simultaneous mapping of the environment, usually called simultaneous localization and mapping (SLAM), is a problem encountered in multiple domains, such as: surgical robots [1], modeling of real-world objects [2], self-driving cars [3], autonomous industrial robots [4], virtual reality headsets, and mobile robotics [5]. The latter is the focus of this thesis, specifically small mobile robotics. An example of this field is the autonomous home vacuum cleaners that need to produce a map of a room to clean it properly.

With the increasing importance of autonomous robots this problem has been heavily researched in the last decades and is already a mature subject with several solutions, [6, 7]. There are also commercial products available using SLAM technology including the previously mentioned virtual reality headsets and home vacuum cleaners. However, there is still a lot of research being done to improve the solutions available.

The main contribution of this thesis is a framework to support autonomous navigation and cooperation among robots with ground truth validation. The robots used are a set of small-scale RC car models equipped with multiple sensors. A diagram representing the framework built is shown in figure 1.1. Its components are the vehicle's actuators, a positioning system that measures the vehicle's position and orientation, an encoder that measures the vehicle's speed, an Inertial Measurement Unit (IMU) that measures the linear accelerations and angular velocities and a laser scanner that produces point clouds of the environment. Moreover, the system is distributed, allowing several mobile robots to coexist in the same environment, paving the way for the development of cooperative or collaborative strategies among robots.

Three applications of the architecture are also proposed in the form of Concurrent Localization and Mapping (CLM) algorithms. The SLAM problem is solved in [8, 9] assuming that landmarks exist in the environment, whose true location is assumed time invariant. Concurrent Localization and Mapping (CLM) algorithms however, do not require the existence of specific landmarks, thus not requiring structuring the environment.

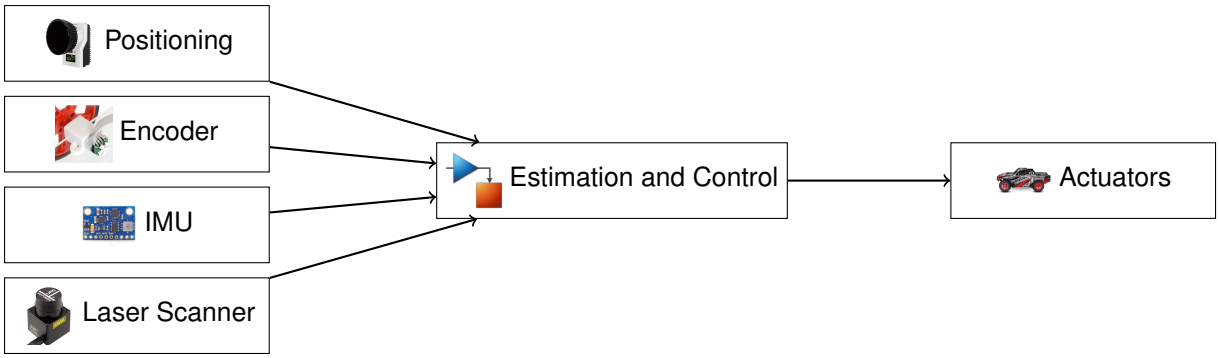


Figure 1.1: Diagram of the architecture implemented.

1.2 Thesis Outline

Following this first chapter with an introduction, in chapter 2, three registration algorithms are detailed, each with a different approach to finding the coordinate transformation between the point clouds. The algorithms are Normal Distributions Transform (NDT), Iterative Closest Point (ICP), and Normal Iterative Closest Point (NICP). Chapter 2 ends with a description of three different CLM algorithms.

In chapter 3, one of the registration algorithms is modified including a parametric study. Also, one of the CLM algorithms is further detailed. A comparison between the three registration algorithms is performed and one is chosen for use in the CLM algorithm. The chapter ends with tests of the CLM algorithms performed in a simulated environment.

In chapter 4, the architecture used and implemented is described and characterized. A state observer for the car is designed and the software developed for this work is described, including low-level software for the car and the CLM and registration algorithms for offline usage.

In chapter 5, three examples of applications are shown where the architecture is validated.

In chapter 6, conclusions taken from the work performed are shown.

Chapter 2

Background

Most CLM algorithms require the use of a registration algorithm. An introduction to registration algorithms is provided in section 2.1. In sections 2.2, 2.3 and 2.4, the three registration algorithms, NDT, ICP, and NICP, respectively, are described. Section 2.5 is used to detail three different CLM algorithms.

2.1 Registration Introduction

In registration algorithms, there are two point clouds: the model cloud and the data cloud. These algorithms try to find a coordinate transformation for the data point cloud that makes it best fit the model point cloud, meaning, a coordinate transformation that makes all the points in the data point cloud project onto their real-world location in the model cloud reference frame. This means that, if two points in each of the point clouds are sampled from the same point in the real world, they should end up with the same coordinates after the data points are transformed by the coordinate transformation. The coordinate transformation is modeled by two parameters, the rotation matrix R and the translation vector T, and is defined by the following expression:

$$p^m = Rp^d + T \tag{2.1}$$

In this expression, the superscript denotes the reference frame of the mathematical object, that is, p^d is the vector of coordinates of the point p in the data cloud reference frame. After the coordinate transformation, the vector of coordinates p^m, that describes the same point but in the model cloud reference frame, is obtained.

There are various approaches to finding the desired coordinate transformation. Some registration algorithms try to model the environment with geometric primitives and then align the other point cloud with this model. This method can be very complex and sensitive to noise as described in [10] so it is not used in this thesis. Other algorithms try to model the environment with probability distributions. One such case is the Normal Distributions Transform (NDT), described in section 2.2 and introduced in [11].

Another approach is to try to directly match points on each point cloud, that is, try to find pairs of points that correspond to the same place in the real world and then find the coordinate transformation between them. This type of algorithms can be further divided into two: feature matching and deep

matching. Feature matching algorithms try to find features and then match them, while deep matching algorithms try to match all, or most, of the points in the clouds. Iterative Closest Point (ICP) is an algorithm that can be implemented using either of these methods and is described in detail in [12]. In section 2.3, one of the deep matching variants of ICP, described in [13], is explained. A variant of ICP that takes into account the surface properties around each point is the Normal Iterative Closest Point (NICP) which was introduced in [14] and is described in section 2.4.

In this thesis, the point clouds are in two dimensions, meaning, the previously mentioned R is a 2×2 matrix described by one parameter, t_θ, and T is a 2×1 vector described by two parameters, t_x and t_y. The definition of R and T according to these parameters can be found in equations 2.2 and 2.3 respectively.

$$R = \begin{bmatrix} \cos t_\theta & -\sin t_\theta \\ \sin t_\theta & \cos t_\theta \end{bmatrix} \tag{2.2}$$

$$T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{2.3}$$

The registration algorithms estimate the following vector of parameters:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_\theta \end{bmatrix} \tag{2.4}$$

2.2 Normal Distributions Transform

Normal Distributions Transform (NDT) is an algorithm that consists in constructing a probability density map of the model point cloud, section 2.2.1, to which the data point cloud is aligned by maximizing a score criterion using a Newton method, section 2.2.2. It was introduced in [11] for the 2D case and extended for the 3D case in [15].

2.2.1 Probability Density Map

The construction of the probability density map starts with subdividing the two-dimensional space where the point cloud is into sub-cells. As in [11], each combination of four adjacent sub-cells forming a square is called a cell. Thus, each sub-cell, apart from the border ones which cannot contain any point, belongs to four different cells. Each point of the point cloud is then attributed to the sub-cell that contains it, meaning that every point belongs to four different cells.

Afterward, the distribution of points inside each cell is fitted to a two-dimensional normal distribution. To do that, the mean and covariance matrix is calculated according to equations 2.6 and 2.5 respectively

where M_i is the set of points q that belong to the cell with index i.

$$C_i = \sum_{q_j \in M_i} (q_j - \bar{q}_i)(q_j - \bar{q}_i)^T \tag{2.5}$$

$$\bar{q}_i = \frac{1}{n} \sum_{q_j \in M_i} q_j \tag{2.6}$$

The eigendecomposition of the covariance matrix C is then taken and used both to make sure the ratio between the smallest and largest eigenvalue is bigger than 0.001 and to calculate the inverse of the covariance matrix.

2.2.2 Optimization Algorithm

The score that is maximized is the sum of the probability density functions of each cell evaluated for each projected data point inside the cell. This corresponds to the expression in equation 2.7 where N_i is the set of points p whose projection is inside cell i and e_{ij} is the Euclidean distance between the projected data point and the mean point of the probability density function of the cell.

$$W = \sum_{\forall N_i} \sum_{p_j \in N_i} \exp\left(\frac{-e_{ij}^T C_i^{-1} e_{ij}}{2}\right) \tag{2.7}$$

$$e_{ij} = (Rp_j + T) - \bar{q}_i \tag{2.8}$$

To maximize the score a Newton method is used on the derivative, meaning the following system of equations is solved in each iteration:

$$H\Delta t = g \tag{2.9}$$

Here, g is the gradient defined in equation 2.10 and H is the Hessian matrix defined in equation 2.12. The expressions are the same as in [11] but in matrix form due to the much faster computation.

$$g = \sum_{\forall N_i} \sum_{p_j \in N_i} e_{ij}^T C_i^{-1} J \exp\left(\frac{-e_{ij}^T C_i^{-1} e_{ij}}{2}\right) \tag{2.10}$$

$$J = \frac{\partial e_{ij}}{\partial t} = \begin{bmatrix} 1 & 0 & -p_{j,x}\sin t_\theta - p_{j,y}\cos t_\theta \\ 0 & 1 & p_{j,x}\cos t_\theta - p_{j,y}\sin t_\theta \end{bmatrix} \tag{2.11}$$

$$H = \sum_{\forall N_i} \sum_{p_j \in N_i} \exp\left(\frac{-e_{ij}^T C_i^{-1} e_{ij}}{2}\right) \left(\left(e_{ij}^T C_i^{-1} J\right)^T \left(e_{ij}^T C_i^{-1} J\right) - \left(J^T C_i^{-1} J\right) - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(e_{ij}^T C_i^{-1} \frac{\partial e_{ij}^3}{\partial t_\theta} \right) \right) \tag{2.12}$$

$$\frac{\partial e_{ij}^3}{\partial t_\theta} = \begin{bmatrix} -p_{j,x}\cos t_\theta + p_{j,y}\sin t_\theta \\ -p_{j,x}\sin t_\theta - p_{j,y}\cos t_\theta \end{bmatrix} \tag{2.13}$$

The algorithm is stopped if the sum of the elements of the parameters vector t squared is below a threshold.

2.3 Iterative Closest Point

Iterative Closest Point (ICP) is an iterative registration algorithm, described in detail in [12], which consists of the following general steps:

1. Build association pairs with one model point and one data point.

2. Filter out outliers, that is, detect pairs of points that are wrongly associated so they can be ignored in the next step.

3. Find a rotation matrix and a translation vector that best transforms the data points into the corresponding model points.

4. Go back to the beginning unless a stopping criterion has been met.

2.3.1 Association

The association method constructs multiple pairs of points. Each pair contains one point from each point cloud, $\{q_i, p_i\}$. There can be any number of pairs and a point can belong to multiple pairs. Regardless, the goal of the method is to pair points that correspond to the same place in the real world. Two association methods are studied in this thesis.

The first association method consists of finding the model point that has the smallest Euclidean distance to the projection of each data point. This means the number of pairs is going to be the same as the number of data points and a model point can belong to multiple pairs. The projection of the data points is performed with equation 2.1 where R and T are the ones calculated in the previous iteration.

The naive approach of calculating the distance to all model points for each data point and choosing the smallest is very slow. To decrease the computational cost, a Delaunay triangulation of the model point cloud is performed during the initialization of the registration algorithm and is then used to find the aforementioned closest model point for every projected data point.

The second association method consists in re-projecting the projected data points back into the initial structure without the depth information. In a two-dimensional point cloud, this means re-projecting the points back into a 1D structure with bins for each angle interval. When multiple points project back into the same bin a depth buffer is used to choose the closest point. This method is talked about briefly for the 3D case in [14].

In conclusion, the association method is an important part of the algorithm since, other than its importance for an accurate and fast convergence, it is also usually the slowest part of the algorithm. From the two methods studied the first one is slower but much more accurate.

2.3.2 Filter Outliers

The filtering of outliers is performed by adding a weight, w_i, to each pair of points, a function of the Euclidean distance between the points of the pair: the larger the distance, the more likely it is that the two points are wrongly associated, meaning the pair should be given a lower weight. The Euclidean distance between the data and model point of each pair is given by the following expression:

$$d_i = \|(Rp_i + T) - q_i\| \tag{2.14}$$

In general, a weight function should return a value between 0 and 1. The lower the distance d_i between the points, the closer to 1 the weight w_i of the pair should be. The weight functions implemented are the ones discussed in [13].

2.3.3 Find Transformation

When finding the coordinate transformation appropriate to the pairs of weighted matched points, the objective function that is minimized is the following:

$$W = \sum_{i=0}^{n} w_i e_i^T e_i \tag{2.15}$$

Here, e_i is the error function defined by the following expression:

$$e_i = (Rp_i + T) - q_i \tag{2.16}$$

In [13], a closed form solution for the previous objective function is described which involves doing a Singular Value Decomposition (SVD) of a 2×2 matrix. This solution was first presented in [16] for the unweighted objective function.

The first step is to find the R matrix. This requires calculating the Γ matrix, with dimensions 2×2, using equation 2.17 and then calculating its SVD decomposition, $\Gamma = Z\Lambda K^T$. From these three matrices, R and then T are calculated according to equations 2.18 and 2.19.

$$\Gamma = \left(\sum_{i=1}^{N} w_i p_i q_i^T\right) - \left(\sum_{i=1}^{N} w_i\right) \bar{p}\bar{q}^T \tag{2.17}$$

$$R = KZ^T \tag{2.18}$$

$$T = \bar{q} - R\bar{p} \tag{2.19}$$

The points $\bar{p}$ and $\bar{q}$ are the weighted average of the data and model point clouds respectively and are

calculated using equation 2.20.

$$\bar{p} = \frac{\sum_{i=1}^{N} w_i p_i}{\sum_{i=1}^{N} w_i} \tag{2.20}$$

2.3.4 Stopping Criteria

The algorithm is stopped when the mean square error difference from the previous iteration is below a threshold. In this case, the mean square error is the mean of the square of the distances between each projected data point and the associated model point.

2.4 Normal Iterative Closest Point

Normal Iterative Closest Point (NICP), introduced in [14] for the 3D case, is a variation of ICP that uses surface normals to improve its accuracy. It does so by trying to align the surface normals around corresponding points and only penalizing the distance along the surface normal, which results in surfaces being allowed to slip along themselves. The normals are calculated from covariance matrices as shown in [17, 18] and described in section 2.4.1.

The association methods are the same as ICP's, discussed in section 2.3.1. The outlier filtering is performed by completely discarding outliers instead of using weights but the biggest change is in the transformation optimization step. Since the calculation of normals is necessary for this algorithm, the next section is dedicated to the estimation of the surface normal around each point in both point clouds.

2.4.1 Calculating Normals

The estimation of the surface normal around a point starts with choosing the nearby points over which the surface information is going to be calculated. This means choosing the number of points, on each side, over which the surface normal is going to be calculated, called the neighborhood radius. This value is different for each point and depends on two things: the sensor noise and the surface boundaries. The larger the noise, the more points should be considered to maintain the normal estimation accuracy. The surface boundaries are crucial, since calculating normals with points from different surfaces, results in very large errors.

Afterward, the covariance matrix is calculated with the points inside the neighborhood. The covariance matrix is used to extract the surface normal, the surface curvature and the coordinate transformation matrix. This transformation matrix is used to transform a vector from the reference frame formed by the surface normal and tangent vectors to the point cloud's reference frame.

2.4.1.1 Covariance Matrix

The expression used to compute the covariance matrix of the coordinates of a set of points N_i is in equation 2.21.

$$C_i = \frac{1}{n} \sum_{p_j \in N_i} (p_j - \bar{p}_i)(p_j - \bar{p}_i)^T \tag{2.21}$$

$$\bar{p}_i = \frac{1}{n} \sum_{p_j \in N_i} p_j \tag{2.22}$$

In this case, the set N_i contains the consecutive points that are inside the previously calculated neighborhood. This means that the summation is performed between the points of index $u = i - R(i)$ and $v = i + R(i)$ where $R(i)$ is the neighborhood radius. Taking this into account and rearranging equation 2.21, the following expression is obtained:

$$C_i = \frac{1}{v-u+1} \left(\sum_{j=u}^{v} p_j p_j^T - \frac{1}{v-u+1} \sum_{j=u}^{v} p_j \sum_{j=u}^{v} p_j^T \right) \tag{2.23}$$

Showing the elements of the matrices and abbreviating $\sum_{j=u}^{v}$ to $\sum$:

$$C_i = \frac{1}{v-u+1} \left(\begin{bmatrix} \sum p_x p_x & \sum p_x p_y \\ \sum p_x p_y & \sum p_y p_y \end{bmatrix} - \frac{1}{v-u+1} \begin{bmatrix} \sum p_x \\ \sum p_y \end{bmatrix} \begin{bmatrix} \sum p_x & \sum p_y \end{bmatrix} \right) \tag{2.24}$$

To calculate the five different summations needed for each point, five integral images are constructed that allow the computation of each summation in constant time, independently of the neighborhood radius. The five integral images are calculated in the following iterative way:

$$I_{i+1,x} = I_{i,x} + p_{i,x} \tag{2.25}$$

$$I_{i+1,y} = I_{i,y} + p_{i,y} \tag{2.26}$$

$$I_{i+1,xx} = I_{i,xx} + p_{i,x} p_{i,x} \tag{2.27}$$

$$I_{i+1,yy} = I_{i,yy} + p_{i,y} p_{i,y} \tag{2.28}$$

$$I_{i+1,xy} = I_{i,xy} + p_{i,x} p_{i,y} \tag{2.29}$$

Calculating the value of each summation from the integral images is done according to equation 2.30. This equation applies not only to the x integral image but to the others which are not shown for brevity.

$$\sum_{j=u}^{v} p_{j,x} = I_{v+1,x} - I_{u,x} \tag{2.30}$$

2.4.1.2 Eigendecomposition

From the eigendecomposition of the covariance matrix C, the eigenvectors and the corresponding eigenvalues are obtained. These can be used to extract the three previously mentioned characteristics: normal vector, curvature, and coordinate transformation matrix.

The normal vector is the eigenvector which corresponds to the smallest eigenvalue because it is the direction along which there is the least variance.

The curvature is a positive number that represents how well the surface is approximated by a plane. It is computed with the following expression where D_{min} and D_{max} refer to the smallest and largest eigenvalue respectively.

$$\gamma_c = \frac{|D_{min}|}{|D_{min}| + |D_{max}|} \tag{2.31}$$

The coordinate transformation matrix, V, contains the two eigenvectors in its columns and, when left multiplied, transforms a vector's coordinates from the normal-tangential reference frame to the point cloud's reference frame. The transpose, V^T, does the opposite: transforms a vector's coordinates from the point cloud's reference frame to the normal-tangential reference frame.

2.4.2 Filter Outliers

As in [14], a pair of points is discarded if:

- The Euclidean distance between q_i and the projection of p_i is larger than a threshold
- p_i or q_i do not have a well defined normal
- There is a large difference between both normals, $\vec{n}_i^d$ and $\vec{n}_i^m$
- There is a large difference between the curvature of p_i and q_i

2.4.3 Find Transformation

2.4.3.1 Problem definition

The objective function solved for ICP, equation 2.15, is changed to add the matrix Ω that scales the components of the error vector. The error vector is also changed to add the normal alignment error in the form of a difference of angles. Thus, the NICP objective function becomes the following:

$$W = \sum_{i=0}^{n} w_i e_i^T \Omega_i e_i \tag{2.32}$$

where:

$$e_i = \begin{bmatrix} (Rp_i + T) - q_i \\ \left(\operatorname{atan} \frac{\vec{n}_{i,y}^d}{\vec{n}_{i,x}^d} + t_\theta\right) - \operatorname{atan} \frac{\vec{n}_{i,y}^m}{\vec{n}_{i,x}^m} \end{bmatrix} \tag{2.33}$$

The Ω_i matrix is subdivided into two matrices:

$$\Omega_i = \begin{bmatrix} \Omega_i^s & 0 \\ 0 & \Omega_i^n \end{bmatrix} \tag{2.34}$$

Here, Ω_i^s is a 2×2 matrix that scales the first two components of the error vector to only penalize the distance along the surface normal. Ω_i^n is a scalar that determines the importance of aligning the normals.

To calculate Ω_i^s, the eigenvectors matrix V of the model point, together with a diagonal scaling matrix S, is used in the following expression:

$$\Omega_i^s = V_i S_i V_i^T \tag{2.35}$$

This matrix changes the error vector from the model point cloud reference frame to the reference frame defined by the surface normal and surface tangent, then scales the surface normal and surface tangent components of the vector and finally changes it back to the model point cloud reference frame.

The matrix S has two values, S_n and S_t, that scale the normal and tangent components of the error vector respectively. The position of S_n and S_t in the diagonal depends on which eigenvector corresponds to the surface normal and which corresponds to the surface tangent. The value of S_n and S_t are the inverse of the corresponding eigenvalues D_{min} and D_{max} respectively. The value of Ω_i^n is set to one.

2.4.3.2 Solution

The solution to this minimization problem is found using the Levenberg–Marquardt algorithm. As such, the minimization function is linearized around $t = \begin{bmatrix} t_x & t_y & t_\theta \end{bmatrix}^T$ and the following linear system is solved:

$$(H + \lambda I)\, \Delta t = g \tag{2.36}$$

H is an approximation of the Hessian matrix, g is the gradient and λ is the damping factor. The matrices H and g are defined in equations 2.37 and 2.38 respectively.

$$H = \sum_{i=0}^{n} J_i^T \Omega_i J_i \tag{2.37}$$

$$g = \sum_{i=0}^{n} J_i^T \Omega_i e_i \tag{2.38}$$

J_i is the Jacobian matrix of each pair of points. Each entry (j, k) of the Jacobian matrix is defined by the following expression:

$$J_i^{j,k} = \frac{\partial e_i^j}{\partial \Delta t^k} \tag{2.39}$$

After the linear system is solved, t is updated with $t = t - \Delta t$. Multiple iterations are performed until Δt is small enough.

2.5 Concurrent Localization And Mapping

Concurrent Localization and Mapping (CLM) consists in determining the position and orientation of a body in an unknown environment while at the same time producing a map of it. To do that, the trajectory of the body is discretized into points in time, called nodes in this thesis, whose position and orientation, called pose, is determined. Since there is no pre-made map of the environment, the pose of each node is calculated relative to the first node.

The pose can be calculated from the addition of the relative poses between consecutive points, described in section 2.5.1 as well as in [11]. When the CLM is performed in this way, small errors in the calculation of the relative pose between two nodes inevitably grow into large errors in the pose estimates of the subsequent nodes, called dead-reckoning error. This results in the CLM algorithm giving very different poses for the body when it goes through the same place in the real world multiple times.

To solve the problem of the dead-reckoning error, it is possible to construct a network of relative poses by directly computing the relative pose between two distant nodes with similar poses, that is, when the body goes through the same place in the real world twice. This direct relative pose can be computed by using a registration algorithm on two point clouds of the surrounding environment produced by a laser scanner, a depth camera or a normal camera. The network of relative poses can be solved to find the best estimate of the current and past absolute poses. Note that, in this solution, the estimate of the absolute pose depends not only on the pose relative to the previous node but on the pose relative to the subsequent nodes. Humans locate themselves in a similar way, being able to correct their previously estimated position when they arrive at a place they have been before. This construction and solution of the network of relative poses is presented in [19] for the 2D case and in [20] for the 3D case. It is also detailed in section 2.5.2.

Another way of solving this problem is by using measurements from a positioning system. This data can be combined with odometry data to produce an estimate of the trajectory that is resilient to missed data from the positioning system. This is performed in section 2.5.3.

2.5.1 Sequential Map Construction

In this algorithm, the absolute poses are calculated by adding up the relative poses between consecutive nodes. Equation 2.1, which describes the projection of a point onto another reference frame, also describes the relative pose between two nodes. Since this section involves computations with multiple nodes, the superscripts m and d are replaced with numbers denoting the index of the node whose reference frame the mathematical object is defined in. Also, the rotation matrices and translation vectors are numbered. The rotation matrix R^i_{i+1} and translation vector $T^i_{i\to i+1}$ are used to project points from the reference frame of the node $i+1$ to the reference frame of the node i. Equation 2.1 is rewritten with this notation in the following equation:

$$p^i = R^i_{i+1} p^{i+1} + T^i_{i\to i+1} \tag{2.40}$$

Recursively replacing the expression for p^{i+1} in the expression for p^i, the expression for the projection of a point onto a reference frame of any previous node can be derived. This expression is shown in equation 2.41 with the generalized rotation matrix and translation vector being defined in equation 2.42 and 2.43 respectively.

$$p^i = R^i_j p^j + T^i_{i\to j} \tag{2.41}$$

$$R^i_j = \prod_{k=i}^{j-1} R^k_{k+1} \tag{2.42}$$

$$T^i_{i\to j} = \sum_{k=i}^{j-1} R^i_k T^k_{k\to k+1} \tag{2.43}$$

The position of the body in a node, in the node's frame of reference, is the origin. Thus, the position of the body in a previous node's frame of reference can be obtained by replacing p^j in equation 2.41 with a zero vector.

2.5.2 Network of Relative Poses

As mentioned before, this algorithm, used is in [19], makes use of a network of relative poses. Each node of the network is connected to at least one other node. A connection between nodes exists if there is a measured relative pose, usually from registration algorithms, between the two nodes. From these relative poses, the network is optimized to find the pose of each node in the network. The optimization is performed by minimizing the following criterion:

$$W = \sum_{\{i,j\}\in P} \left(t^a_i - t^a_j - t^r_{ij}\right)^T C^{-1}_{ij} \left(t^a_i - t^a_j - t^r_{ij}\right) \tag{2.44}$$

Here, t^a_i is a vector defining the absolute pose of node i and t^r_{ij} is the measured relative pose between the nodes i and j, that is, the measured value of $t^a_i - t^a_j$. C_{ij} is the covariance matrix of the relative pose measurements and the summation is performed over the set P that contains all pairs of connected nodes.

This criterion can also be expressed in matrix form. To do that, the absolute poses t^a are concatenated into a single column vector, $[t^a]$, and the estimates of the relative poses $t^e_{ij} = t^a_i - t^a_j$ are replaced with $[t^e] = Q\,[t^a]$. The measured relative poses t^r are also concatenated into a single column vector, $[t^r]$. The matrix Q is then used to, when left multiplied with $[t^a]$, give the estimate of the measured relative poses. The covariance matrices C are also concatenated diagonally into a single matrix, $[C]$. The resulting expression for the criterion in matrix form is given by the following equation:

$$W = \left([t^r] - Q\,[t^a]\right)^T [C]^{-1} \left([t^r] - Q\,[t^a]\right) \tag{2.45}$$

The solution of $[t^a]$ that minimizes W is given by:

$$[t^a] = \left(Q^T\,[C]^{-1}\,Q\right)^{-1} Q^T\,[C]^{-1}\,[t^r] \tag{2.46}$$

What is left is the construction of the matrices $[t^r]$, $[C]$ and Q. In this thesis' case, the measured relative poses $[t^r]$ come from the registration algorithms. The matrix $[C]$, that contains the covariance matrices, is ignored, since the covariances of the registration algorithms are not estimated. This means it is set as the identity matrix with the same size as $[C]$. To calculate the matrix Q, it is necessary to describe the non-linear measurement function t^e_{ij}, which describes the relative pose returned by the sensors as a function of the absolute poses t^a_i and t^a_j. Since the sensor used is the registration algorithm, the measurement function is:

$$t^e_{ij}\left(t^a_i, t^a_j\right) = \begin{bmatrix} R(t^a_{j,\theta})^T \begin{bmatrix} t^a_{i,x} - t^a_{j,x} \\ t^a_{i,y} - t^a_{j,y} \end{bmatrix} \\ t^a_{i,\theta} - t^a_{j,\theta} \end{bmatrix} \tag{2.47}$$

Given the non-linearity introduced by the rotation matrix, it is necessary to linearize this expression around the current estimate of the absolute poses, $\hat{t}^a$. This estimate comes from the last time this algorithm was solved for $[t^a]$ or, for the most recent nodes, from the sequential adding of the relative poses as was done in section 2.5.1. As such, the first-order Taylor series of the measurement function in equation 2.47 is the following:

$$t^e_{ij}\left(t^a_i, t^a_j\right) = t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right) + \frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_i}\left(t^a_i - \hat{t^a_i}\right) + \frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_j}\left(t^a_j - \hat{t^a_j}\right) \tag{2.48}$$

Here, the derivatives $\frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_i}$ and $\frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_j}$ are the following:

$$\frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_i} = \begin{bmatrix} \cos \hat{t}^a_{j,\theta} & \sin \hat{t}^a_{j,\theta} & 0 \\ -\sin \hat{t}^a_{j,\theta} & \cos \hat{t}^a_{j,\theta} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.49}$$

$$\frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_j} = \begin{bmatrix} -\cos \hat{t}^a_{j,\theta} & -\sin \hat{t}^a_{j,\theta} & -\sin \hat{t}^a_{j,\theta}\left(\hat{t}^a_{i,x} - \hat{t}^a_{j,x}\right) + \cos \hat{t}^a_{j,\theta}\left(\hat{t}^a_{i,y} - \hat{t}^a_{j,y}\right) \\ \sin \hat{t}^a_{j,\theta} & -\cos \hat{t}^a_{j,\theta} & -\cos \hat{t}^a_{j,\theta}\left(\hat{t}^a_{i,x} - \hat{t}^a_{j,x}\right) - \sin \hat{t}^a_{j,\theta}\left(\hat{t}^a_{i,y} - \hat{t}^a_{j,y}\right) \\ 0 & 0 & -1 \end{bmatrix} \tag{2.50}$$

To perform the linearization, t^r_{ij} is replaced with $t^r_{ij} - t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)$ and the matrix Q is constructed so that, in each group of three lines that corresponds to each measurement, the sub-matrix $\frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_i}$ is added to the columns corresponding to t^a_i. The same is performed for the sub-matrix $\frac{\partial t^e_{ij}\left(\hat{t}^a_i, \hat{t}^a_j\right)}{\partial t^a_j}$ which is added to the columns corresponding to t^a_j. The resulting solution of $[t^a]$ from equation 2.46 will be $[t^a] - \left[\hat{t}^a\right]$ instead. To obtain the final solution, $[t^a]$, the vector of estimated absolute poses $\left[\hat{t}^a\right]$ is added.

2.5.3 Positioning System

To combine the odometry data with data from a positioning system, a Kalman filter was developed, with the following model equation:

$$x_k = Fx_{k-1} + Gu_{k-1} \tag{2.51}$$

x is the state variables vector, defined in equation 2.53, and u is the system inputs vector, defined in equation 2.55. F and G are the state-space matrices defined in equations 2.52 and 2.54 respectively.

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.52) \quad x = \begin{bmatrix} x^e \\ y^e \\ \psi \\ \dot{x}^e \\ \dot{y}^e \\ \dot{\psi} \end{bmatrix} \quad (2.53) \quad G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \cos\psi & 0 \\ \sin\psi & 0 \\ 0 & 1 \end{bmatrix} \quad (2.54) \quad u = \begin{bmatrix} \dot{x}^b \\ \dot{\psi} \end{bmatrix} \quad (2.55)$$

In these equations, Δt is the sample time, x^e, y^e and ψ are the pose in the earth's reference frame and $\dot{x}^e$, $\dot{y}^e$ and $\dot{\psi}$ are its derivatives. The input values come from the body's state observer which uses data from the internal sensors to fully observe those state variables.

Three of these state variables are observed from the positioning system: x^e, y^e and ψ. As such, its output equation is in equation 2.56 where C is defined in equation 2.57 and D is a zero matrix with the appropriate size.

$$\hat{y} = Cx + Du \tag{2.56}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.57}$$

The Kalman filter of this state-space system is the iterative process with the following equations:

$$x_{k|k-1} = Fx_{k-1|k-1} + Gu_{k-1} \tag{2.58}$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q \tag{2.59}$$

$$S_k = CP_{k|k-1}C^T + R \tag{2.60}$$

$$L_k = P_{k|k-1}C^T S_k^{-1} \tag{2.61}$$

$$x_{k|k} = x_{k|k-1} + L_k\left(\hat{y}_k - Cx_{k|k-1}\right) \tag{2.62}$$

$$P_{k|k} = \left(I - L_k C\right) P_{k|k-1} \tag{2.63}$$

In these equations, P, S, and L are, respectively, the state variables covariance matrix, the innovation covariance matrix, and the Kalman filter gain. The matrices Q and R are constants that represent the process covariance and the outputs covariance. When there is no observation, that is, when the positioning system fails, the values of matrix L are set to zero since the innovation covariance is infinite.

Chapter 3

Methodology

In this chapter, the changes performed to the Normal Iterative Closest Point (NICP) are described in section 3.1. In section 3.2 the implementation of the network-based CLM algorithm is described in more detail. In section 3.3 the multiple registration algorithms implemented are compared. The results of the sequential and network-based CLM algorithms in a simulated environment are shown in section 3.4.

3.1 Normal Iterative Closest Point

In this section, the changes made to NICP are described. The computation of the neighborhood radius is described in section 3.1.1. A new metric, the normal confidence value, is described in section 3.1.2. The final normal calculation algorithm is validated in section 3.1.3. In section 3.1.4, the changed outlier filtering is described and, in section 3.1.5, there is a description of the slightly changed optimization algorithm that finds the coordinate transformation, taking into account these new parameters.

3.1.1 Neighborhood Radius

As mentioned before, in section 2.4.1, the neighborhood radius depends on the sensor noise and the surface boundaries. Because of this, two neighborhood radii are calculated and the final neighborhood radius is the minima of both. These two neighborhood radii are the noise neighborhood radius and the surface boundaries neighborhood radius.

Before constructing these neighborhood radii, some assumptions have to be made about the sensor used to sample the point cloud. The sensor is assumed to sample points of the environment in polar coordinates, that is, it measures the depth r and angle θ of each point. The polar coordinates are transformed into cartesian coordinates with the following expression:

$$p = r \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{3.1}$$

3.1.1.1 Noise Neighborhood Radius

To construct the noise neighborhood radius, the noise in the estimation of the surface normal is needed. As such, the noise of the sensor that produces the point cloud and the relationship between its noise and the surface normal estimation angular error are needed. The two values returned by the sensor, depth r and angle θ from equation 3.1, are assumed to have different noise characteristics. The depth value is assumed to follow a normal distribution with mean value equal to the real value and standard deviation proportional to the real value. The angle value is assumed to have no noise.

To characterize the surface normal estimation angular error, multiple point clouds sampled from a simulated environment are used to estimate the surface normal using the method described in section 2.4.1. The simulated environment from where the point clouds are sampled has a single infinite planar surface. This planar surface has two free parameters: the angle β and the depth r. The angle β is the angle the surface makes with a surface normal to the considered point's ray. The point's ray is the line that passes through it and starts at the origin. The depth r is the depth of the considered point, that is, the distance from the considered point to the origin. The normal calculation has one free parameter: the neighborhood radius R. Multiple sets of values of β, r and R are chosen and, for each combination, multiple simulated point clouds are created and the normal angle standard deviation is calculated. Figure 3.1 shows that the standard deviation does not depend on the depth r.

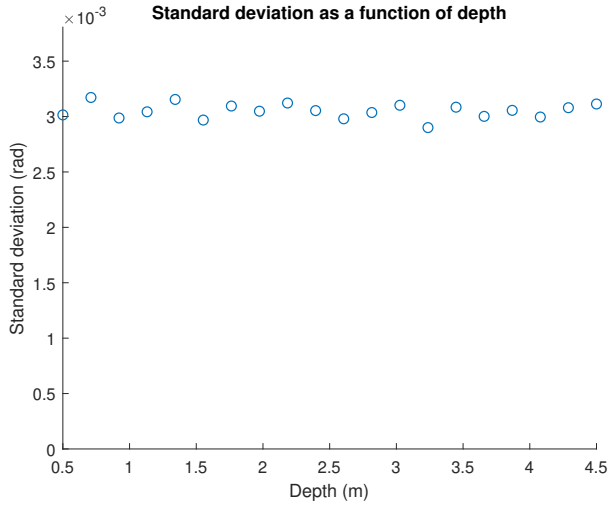


Figure 3.1: Normal Angular Standard Deviation as a function of depth r.

The standard deviation as a function of β is plotted in figure 3.2 for a neighborhood radius of 40. The value of β can only be estimated from the normals themselves increasing the complexity and cost of calculating them. Thus, the effect of β on the normals estimation error is ignored and the subsequent study of the effect of the neighborhood radius on the error is performed with an β equal to zero.

Figure 3.3 shows the inverse of the standard deviation as a function of the neighborhood radius R.

Given the independence of the normal error from the distance r, the noise neighborhood radius is a constant R_{max} equal to 8.

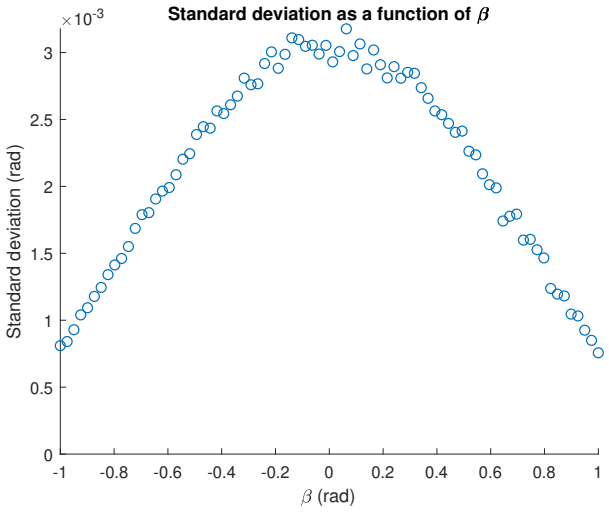


Figure 3.2: Normal Angular Standard Deviation as a function of β.

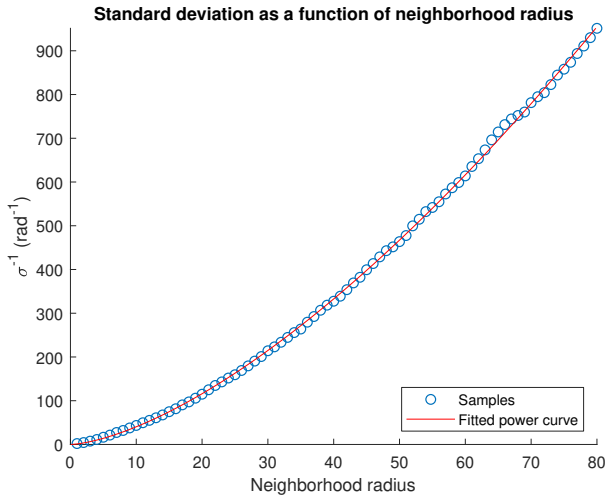


Figure 3.3: Normal Angular Standard Deviation as a function of the neighborhood radius R.

3.1.1.2 Surface Boundaries Neighborhood Radius

The computation of the surface boundary neighborhood radius starts with identifying which points correspond to a surface boundary. As in [17] and [21], the indicator of surface boundary used is a large difference in depth from either adjacent point. In this thesis, the threshold chosen is the difference in depth that an adjacent point has if both it and the point being analyzed are sampled from a planar surface that makes an angle of β with a surface perpendicular to the considered point's ray. As mentioned before, the point's ray is the line that passes through it and starts at the origin. A diagram showing this geometry is shown in figure 3.4 with the solution, which depends on the angle β and the angle interval $\Delta\theta$ between the two adjacent rays, given in equation 3.2.

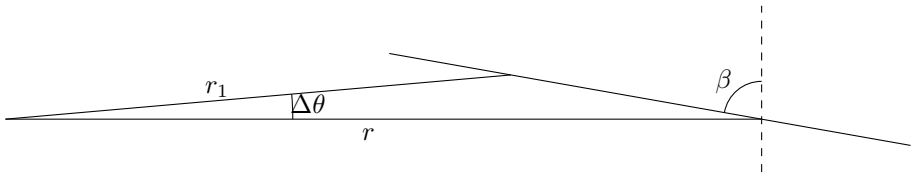


Figure 3.4: Drawing used to deduce the threshold for the surface boundary.

$$r - r_1 = r\left(1 - \frac{\cos\beta}{\cos\left(\beta - \Delta\theta\right)}\right) \tag{3.2}$$

The chosen value of β is 80°. From the surface boundary binary map, a distance transform is taken that gives the surface boundary neighborhood radius map.

3.1.2 Normal Confidence Value

The normal confidence value is a number between 0 and 1 that reflects how good the normal estimation is. It depends on the previously studied surface curvature and normal angular error. The curvature component of the normal confidence value is given by:

$$\gamma_f = min\left(\frac{\log^2\gamma_c}{40}, 1\right) \tag{3.3}$$

The normal angular error component of the normal confidence value is equal to the ratio between the minimum normal angular error standard deviation and the standard deviation of the considered point. These standard deviations are a function of the neighborhood radius R and, as such, are calculated according to the following expression:

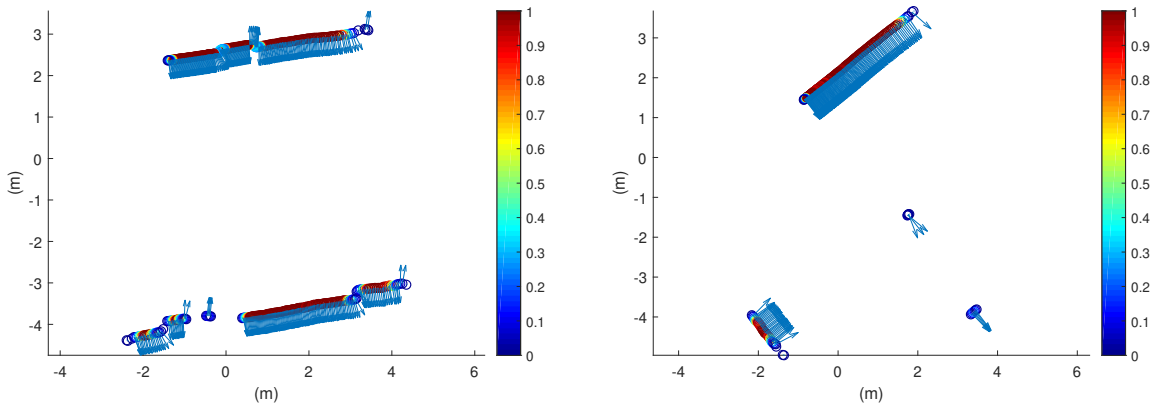
$$\gamma_e = \frac{\sigma_{min}}{\sigma} = \frac{\sigma_{min}\left(R_{max}\right)}{\sigma\left(R\right)} \tag{3.4}$$

The final value for the normal confidence value, γ, is equal to the product of the two previous expressions:

$$\gamma = \gamma_f\gamma_e \tag{3.5}$$

3.1.3 Validation

To validate the normal calculation algorithm, it was applied to two point clouds produced by a real sensor. The results can be seen in figure 3.5 where the arrows represent the normals while the color gradient represents the normal confidence value.



(a) Random scan 1.

(b) Random scan 2.

Figure 3.5: Validation of normal calculation.

The algorithm works as desired since the normal vectors returned by the algorithm seem to be precise and the normal confidence value is closer to zero for points near the surface edges where there is less information to accurately estimate the normals.

3.1.4 Filter Outliers

The filtering of outliers is performed with weights. The same weight functions mentioned in section 2.3.2 are used but the distance between points is generalized to include the misalignment of normals. The generalized distance is given by the following expression:

$$d_i = \|(Rp_i + T) - q_i\| + C_{\vec{n}} \left\|\gamma_i^d R\vec{n}_i^d - \gamma_i^m \vec{n}_i^m\right\| \tag{3.6}$$

In this expression, $\vec{n}_i^d$ and $\vec{n}_i^m$ are the normals of the data and model point respectively, γ_i^d and γ_i^m are the normal confidence values of the data and model point respectively and $C_{\vec{n}}$ is a constant scaling factor that weighs the importance of the normals being aligned.

3.1.5 Find Transformation

There are three main differences to the optimization algorithm described in section 2.4.3:

- The third element of the error vector, the difference in angles of the normals, is approximated to the cross product between them
- The values of S_n and S_t are 1 and $10^{-2\gamma^m}$ respectively.
- The value of Ω_i^n is the product of the normal confidence values of both matched points.

Regarding the optimization algorithm itself, the identity matrix in the linear system that is solved is changed to diag (H), a diagonal matrix with the same values as the ones in matrix H's diagonal. Thus, the final linear system solved is the following:

$$(H + \lambda \text{diag}\,(H))\,\Delta t = g \tag{3.7}$$

As for the damping parameter λ, it starts with a constant λ_0 and, after each iteration, is divided by ν^k, where ν is a constant and k is varied until a score, W, lower than the previous iteration is achieved. First, an attempt is made with k equal to 1, then equal to 0, then 2, and then it is incremented by 1 until 10. If no value of k is found that decreases W, the minimization algorithm finishes earlier with the final result being the one obtained in the previous iteration.

3.2 Simultaneous Localization And Mapping

Of the three CLM algorithms described in section 2.5, two are fully described. Concerning the other one, the network-based map construction in section 2.5.2, only the solution to the network of relative poses has been described. In this section, the process of constructing the network of relative poses is described.

The network of relative poses contains two kinds of connections: between consecutive nodes and between nodes distant in time. The latter connections can also be called closing loop connections. When a new node is added, as in the sequential map construction, its pose is estimated by using the registration algorithm between its point cloud and the previous node's. This new connection between the nodes is added to the network and then, an attempt is made to make closing loop connections with previous nodes with similar poses.

The selection of the previous nodes that are used for closing loop connections starts with calculating the nodes whose pose is close to the considered node's. After this, the nodes are grouped into clusters where one cluster is a group of sequential nodes that are all close the considered node. Then, the node closest to the considered node belonging to the earliest cluster is chosen.

After registering the point clouds of both nodes, the result is verified using a trust-region shown in equations 3.10 and 3.11. If both boolean values are true, the result is inside the trust-region and a connection between the two nodes is added to the network. The trust-region is based on the estimated values of the relative pose, $t^e_{ij,x}$, $t^e_{ij,y}$ and $t^e_{ij,\theta}$, which come from the current estimates of the nodes' absolute poses:

$$t^e_{ij,\theta} = t^a_{i,\theta} - t^a_{j,\theta} \tag{3.8}$$

$$\begin{bmatrix} t^e_{ij,x} \\ t^e_{ij,y} \end{bmatrix} = R\left(t^a_{j,\theta}\right)^T \begin{bmatrix} t^a_{i,x} - t^a_{j,x} \\ t^a_{i,y} - t^a_{j,y} \end{bmatrix} \tag{3.9}$$

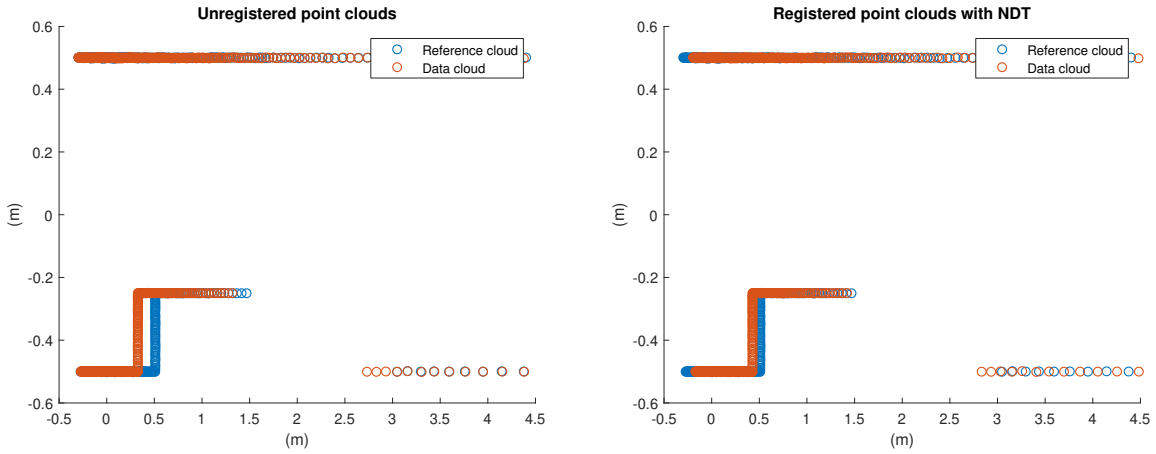
$$B_{\circ} = \left(t^r_{ij,x} - t^e_{ij,x}\right)^2 + \left(t^r_{ij,y} - t^e_{ij,y}\right)^2 <= C^2_{cc} \tag{3.10}$$

$$B_{\theta} = |t^r_{ij,\theta} - t^e_{ij,\theta}| <= C_{c\theta} \tag{3.11}$$

3.3 Registration Algorithms Analysis

The accuracy and performance of the algorithms depend heavily on the initial transformation as well as the stopping criterion. Also, the algorithms' accuracy can vary wildly between two different pairs of point clouds. This makes it difficult to make general statements comparing the algorithms. Regardless, in this section, the types of errors the registration algorithms make are studied and the most appropriate algorithm for this application is chosen.

The two simulated point clouds in figure 3.6a are registered with NDT and the results shown in figure 3.6b.



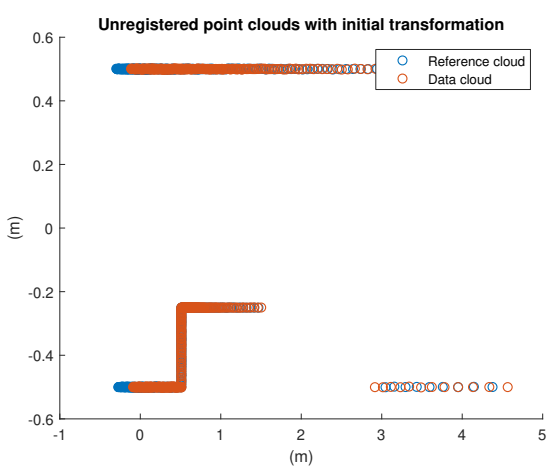
(a) Unregistered point clouds.

(b) Point clouds registered using NDT.

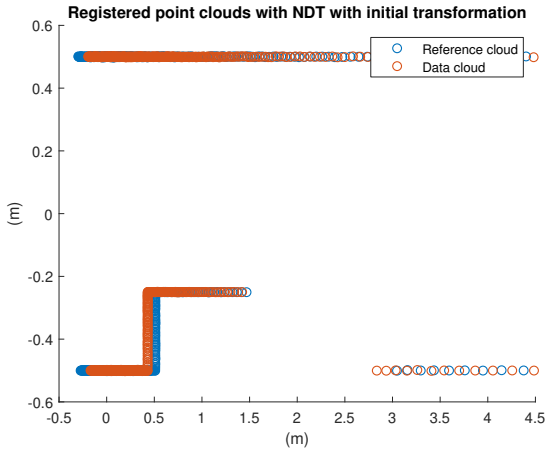
Figure 3.6: Registration of point clouds using NDT.

The failed registration happened because there is only a partial overlap between the two point clouds, meaning, some points in one point cloud don't have any real-world correspondence to a point in the other point cloud. This is not the result of convergence away from the global minima, the score W of NDT is well optimized as shown in figure 3.7, where the same point clouds are registered but the initial transformation given is the correct one. Instead of maintaining the same transformation as an ideal algorithm would, it converges away from the real solution. ICP and NICP try to solve this with outlier filtering but it has its problems.

In figure 3.8a, the same point clouds are registered using NICP without an initial transformation. It can be seen that the results are slightly worse than the registration attempt by NDT in figure 3.6b. This time, it is because few points are misaligned and the weighting algorithm incorrectly identifies them as outliers, giving them low weights and not allowing them to influence the coordinate transformation. In figure 3.8b, NICP is used to register the same point clouds but this time starting with the correct initial



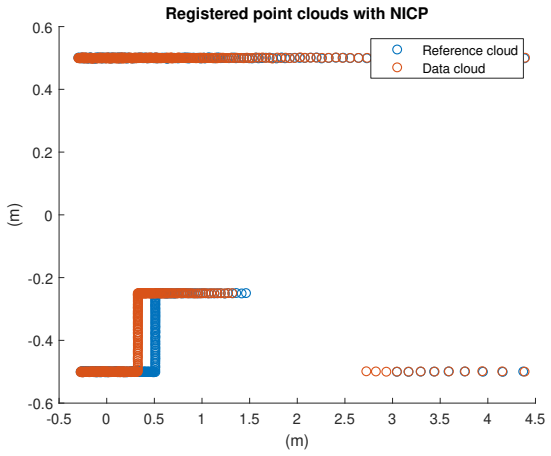
(a) Unregistered point clouds with initial transformation.



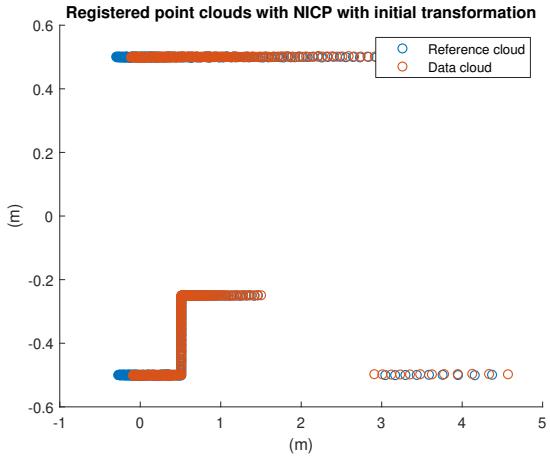
(b) Point clouds registered using NDT with initial transformation.

Figure 3.7: Registration of point cloud using NDT with initial transformation.

transformation. The algorithm correctly outputs the inputted initial transformation.



(a) Point clouds registered with NICP.



(b) Point clouds registered with NICP with initial transformation.

Figure 3.8: Point clouds registered with NICP with and without initial transformation.

In this application, there is a good estimate of the initial transformation available, thus, the advantages of an aggressive outlier filtering outweigh the disadvantages of having a weak or no outlier filtering. Besides, as shown in table 3.1, NDT is the slowest algorithm of the three.

The main difference between ICP and NICP is that ICP employs a point to point metric while NICP employs, generally, a point to plane metric. A point to point metric takes into consideration the distance between the associated points, whereas a point to plane metric considers the distance between the data point and a plane, along the plane's normal. The plane, in the case of NICP, comes from the linearization of the surface around the point in the model cloud.

A point to plane metric is more appropriate for structured environments since the surface normals can be calculated with little error. Regardless, NICP is a generalization of ICP since, when the normal confidence value is zero, it behaves as the latter. For these reasons, NICP is the registration algorithm

used for the CLM implementation.

The computational cost of each algorithm depends a lot on the stopping criterion chosen. Regardless, in this application and with good initial transformations, the average time taken by a registration of each algorithm is presented in table 3.1. These values can only be used as a rough comparison of the computational cost since they are sure to vary non-linearly between implementations and hardware.

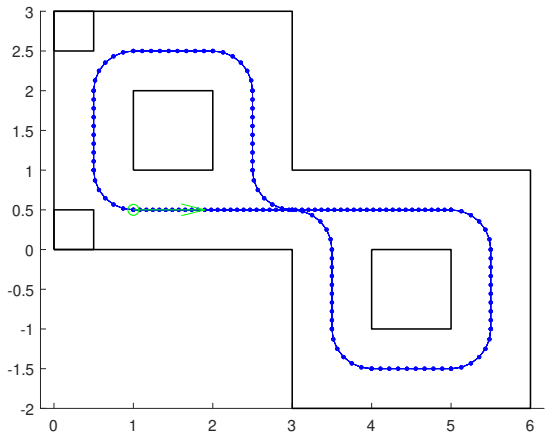
Algorithm	Time (s)
NDT	0.50
ICP	0.01
NICP	0.05

Table 3.1: Rough computational cost of the registration algorithms studied.

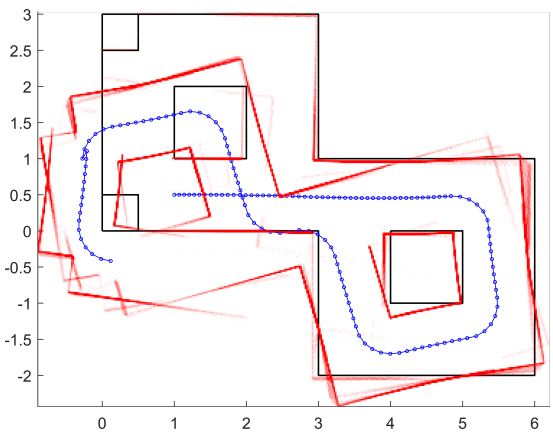
As expected, NDT is considerably slower than the other two and ICP is much faster than NICP since there is an analytical solution and there is no computation of normals. The increased computational cost of NICP, despite having only slightly better accuracy than ICP, is still considered worth it since it is fast enough on the hardware available. The normal calculation required for NICP is also important for estimating the directions in which both ICP and NICP return reliable results as shown in [22].

3.4 Simulation

In this section, the CLM algorithms described in section 3.2 are tested with the registration algorithm NICP described in section 3.1. The tests are performed in the simulated environment shown in figure 3.9a with the sensor noise characteristics mentioned in section 3.1.1, that is, normally distributed depth and no noise for the polar angle value. The result of this algorithm, without an initial transformation, is shown in figure 3.9b. Since there is no initial transformation, the registration algorithm is first executed without outlier filtering to produce an estimate of the transformation, which is then used as the initial transformation.



(a) Simulated environment and trajectory. The green circle and arrow shows the initial position and velocity.

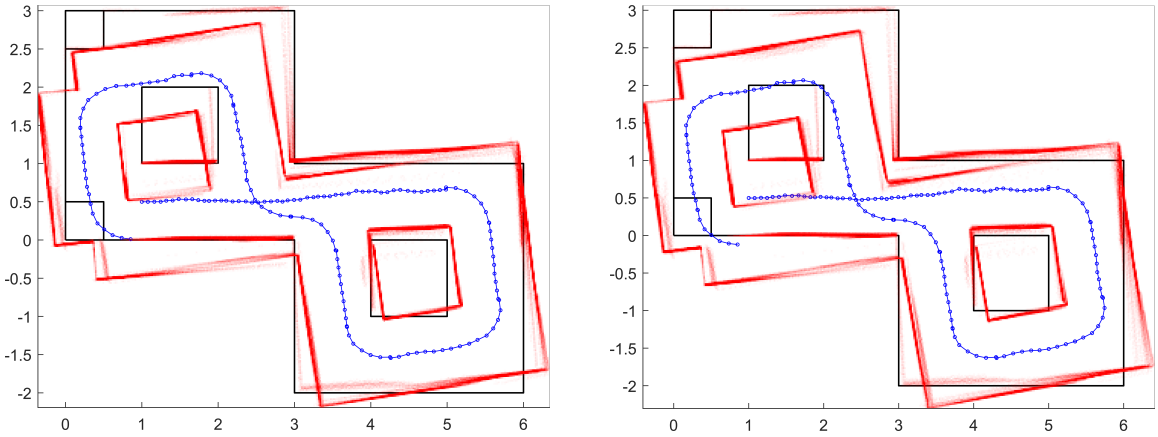


(b) Reconstructed environment with no initial transformation.

Figure 3.9: Simulated environment and its reconstruction.

The CLM algorithm is greatly improved when there is an initial transformation source. In real-world

usage, it usually comes from a vehicle model or other sensors but in this simulated environment, it is the real transformation with added noise. The result, using the same algorithm as before, but with this initial transformation can be seen in figure 3.10a. Figure 3.10b shows the CLM using only the initial transformation, without using the registration algorithm. Although the registration algorithm depends heavily on a good initial transformation it can still improve it.

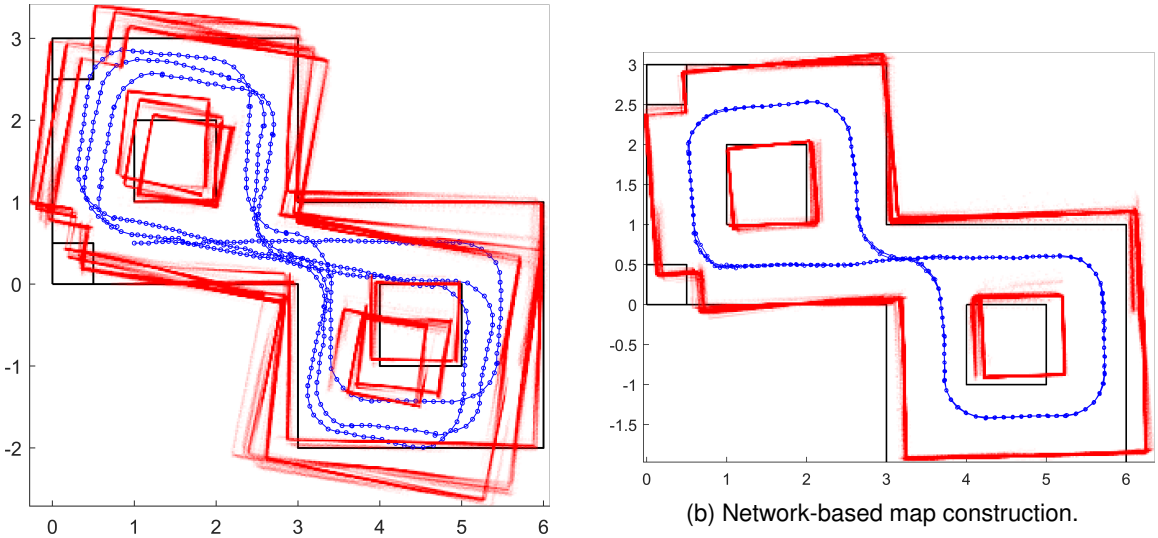


(a) Registration algorithm using initial transformation.

(b) Using only the initial transformation.

Figure 3.10: Effect of the registration algorithm on the sequential map construction.

In both these cases, there is dead-reckoning error. This error is amplified if multiple laps are performed around the same environment, as shown in figure 3.11a. The network-based CLM algorithm, described in sections 2.5 and 3.2, attempts to solve this problem. It is used on the same data, with the results being shown in figure 3.11b. Although there are still errors, there is no dead-reckoning error.



(a) Sequential map construction.

(b) Network-based map construction.

Figure 3.11: Comparison between sequential and network-based map constructions.

Figure 3.12 shows the network of relative poses used by the CLM algorithm. It can be seen that, besides the usual relative poses between consecutive point clouds, there are relative poses calculated between two point clouds distant in time.

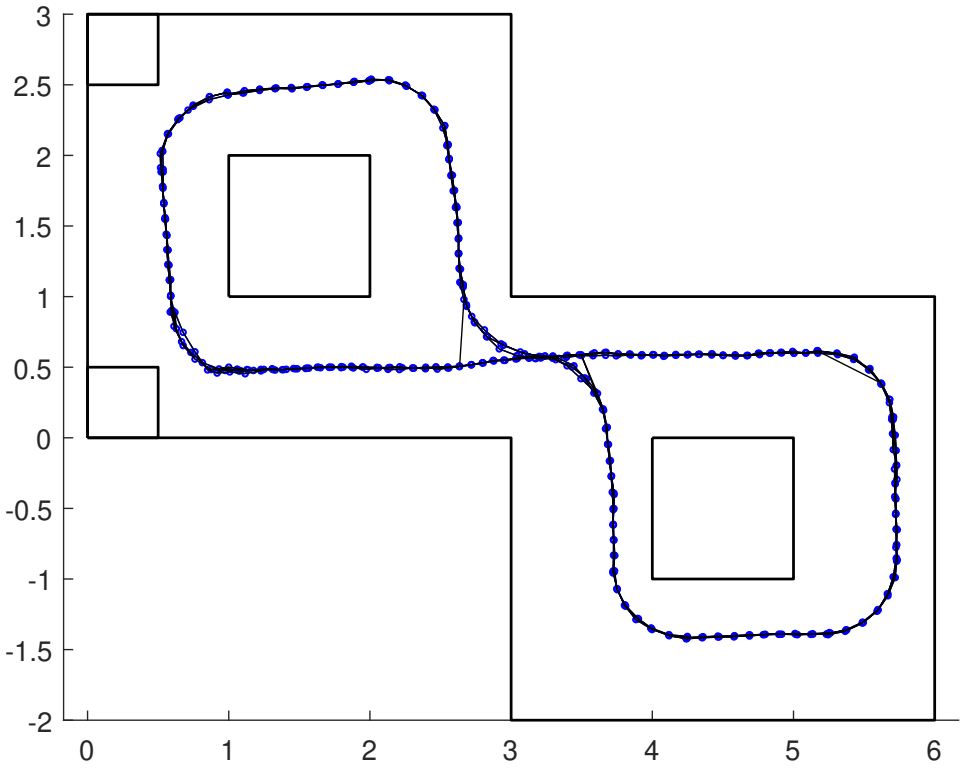
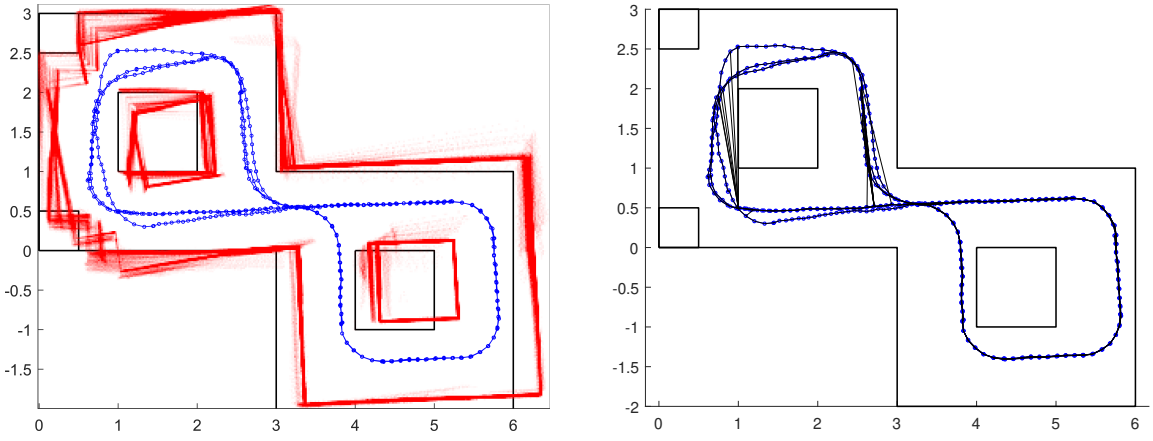


Figure 3.12: Network of relative poses used to construct the map in figure 3.11b.

Figure 3.13 shows what happens when comparisons are made between two point clouds too distant in space. Since there is little overlap between the point clouds, the registration algorithm fails and returns an erroneously relative pose which then propagates to the network solution.



(a) Map construction

(b) Connections

Figure 3.13: Errors in the network-based map construction caused by registration errors.

Chapter 4

Architecture Description

The architecture is described in this chapter. In section 4.1 the vehicle used is described and modeled. In sections 4.2 to 4.5 the sensors used are described and characterized. In section 4.6 a state observer is constructed and in 4.7 the software developed is described.

4.1 Car Model

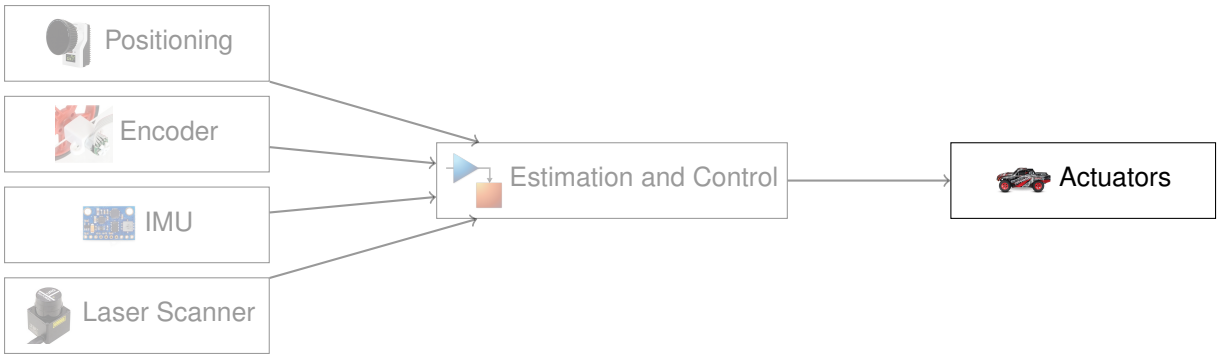


Figure 4.1: Diagram of the architecture implemented, the block discussed in this section is highlighted.

The vehicle used in this thesis is a Desert Prerunner 1/18 Scale 4WD Truck from LaTrax [23]. It has four wheels, all-wheel drive and front steering and a photo of it can be seen in figure 4.2. The equations for the model of the car, based on a bicycle model, are in equations 4.1 to 4.5 illustrated by the diagram in figure 4.3.

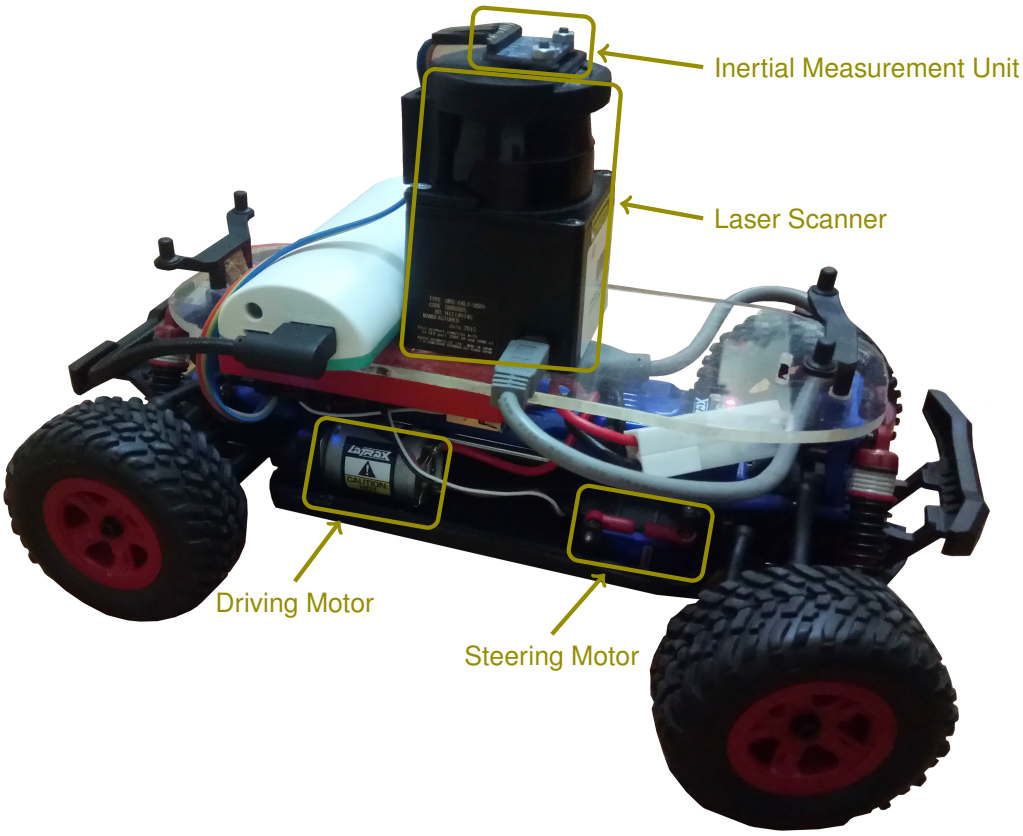


Figure 4.2: Location of some of the systems in the car.

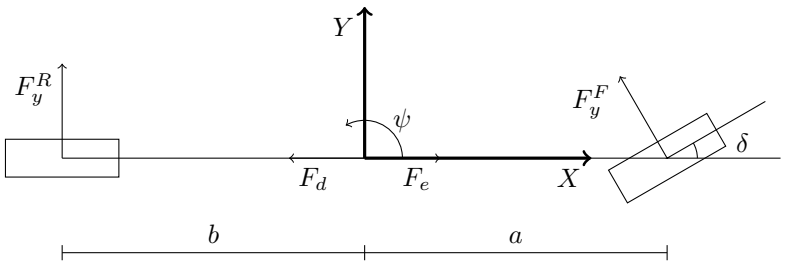


Figure 4.3: Diagram of the car.

$$\ddot{x}^b = \dot{y}^b\dot{\psi} + \frac{1}{m}\left(F_e - F_d - F_y^F \sin\delta\right) \tag{4.1}$$

$$\ddot{y}^b = -\dot{x}^b\dot{\psi} + \frac{1}{m}\left(F_y^R + F_y^F \cos\delta\right) \tag{4.2}$$

$$\ddot{\psi} = \frac{1}{I}\left(aF_y^F \cos\delta - bF_y^R\right) \tag{4.3}$$

$$\dot{x}^e = \dot{x}^b \cos\psi - \dot{y}^b \sin\psi \tag{4.4}$$

$$\dot{y}^e = \dot{y}^b \cos\psi + \dot{x}^b \sin\psi \tag{4.5}$$

$\ddot{x}^b$ and $\ddot{y}^b$ are the linear accelerations in the car's reference frame, $\ddot{\psi}$ is the angular acceleration of the car around the vertical axis and $\dot{x}^e$ and $\dot{y}^e$ are the car's velocities in the earth's frame of reference.

There are four constants: I, m, a and b. The former two, I and m, correspond, respectively, to the car's rotational inertia around the vertical axis and the car's mass. The constant a is the distance between the front wheel axis and the car's center of mass, while b is the distance between the rear wheel axis and the car's center of mass. The car is approximated to a rectangular box to calculate I, while the remaining three constants are measured directly from the car. Afterward, five variables have to be characterized: F_e, F_d, F_y^R, F_y^F and δ which are, respectively, the force produced by the engine on the wheels, the drag forces, the side force on the rear tires, the side force on the front tires and the steering angle of the front wheels.

There are two inputs to the car in the form of Pulse Width Modulation (PWM) signals. One signal controls the driving motor that is connected to all four wheels while the other signal controls the steering servo that turns the front steering wheels. Both PWM signals sent to the actuators have a duty cycle between 10% and 20%. To obtain a more practical range, the model's inputs are numbers between -1 and 1 corresponding to 10% and 20% respectively. The inputs to the model are: u_e and u_d, with values between -1 and 1, the former controls the engine force F_e and the latter controls the steering angle δ. A value of zero for u_e will stop the driving motor, while a value of zero for u_d will make the car go roughly in a straight line. The relationship between these inputs, the model and the previously mentioned forces, is detailed in the following sections.

4.1.1 Drag Force

The drag force is assumed to take the form shown in equation 4.6, where C_{d1} and C_{d0} are parameters to be estimated. The usual quadratic term is considered zero since the car is traveling at low speeds.

$$F_d = C_{d1}\dot{x}^b + C_{d0} \tag{4.6}$$

Assuming this drag force, the stopping distance is given by equation 4.7 where m and $\dot{x}_i^b$ are the car's mass and initial speed respectively.

$$x_f^b = \frac{m}{C_{d1}}\dot{x}_i^b + \frac{mC_{d0}}{C_{d1}^2}\ln\left(\frac{C_{d0}}{C_{d0} + C_{d1}\dot{x}_i^b}\right) \tag{4.7}$$

Multiple stopping distance experiments were performed, using a positioning system, and this equation was fitted to the experimental data giving an estimate of the parameters C_{d1} and C_{d0}. The fitting of equation 4.7 to the experimental data is shown in figure 4.4.

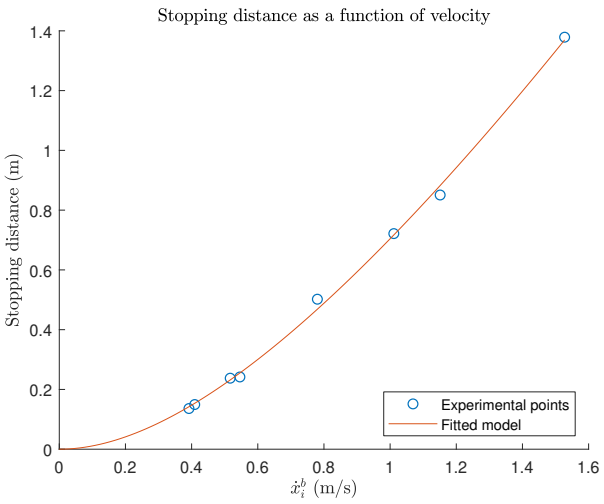


Figure 4.4: Stopping distance as a function of the car's velocity, $\dot{x}^b$.

4.1.2 Engine Force

The force transmitted by the engine to the ground is assumed to take the form of:

$$F_e = C_{e1} u_e + C_{e0} \tag{4.8}$$

From the equations 4.8 and 4.6 it can be deduced that the car's velocity in steady-state, as a function of u_e, is given by the following expression:

$$\dot{x}^b = \frac{C_{e1}}{C_{d1}} u_e + \frac{C_{e0} - C_{d0}}{C_{d1}} \tag{4.9}$$

Multiple sample points of the steady-state velocity as a function of u_e were collected inside a positioning system. These points were then fitted to a first degree polynomial that, together with equation 4.9, allows the computation of C_{e1} and C_{e0}. The collected sample points along with the fitted polynomial can be seen in figure 4.5.

4.1.3 Steering Angle

To determine the relationship between the steering input u_d and the steering angle δ, the car was driven around while the values of x^e, y^e and ψ were recorded with a positioning system. The car's steering input value u_d was also recorded. With this data, the angular velocity $\dot{\psi}$ and forward velocity $\dot{x}^b$ can be calculated. In a kinematic model the angular velocity $\dot{\psi}$ depends only on the forward velocity $\dot{x}^b$ and steering angle δ so, assuming a kinematic model, it is possible to estimate the steering angle from $\dot{\psi}$ and $\dot{x}^b$. With the values of u_d and corresponding δ, the plot in figure 4.6 was constructed. A linear regression

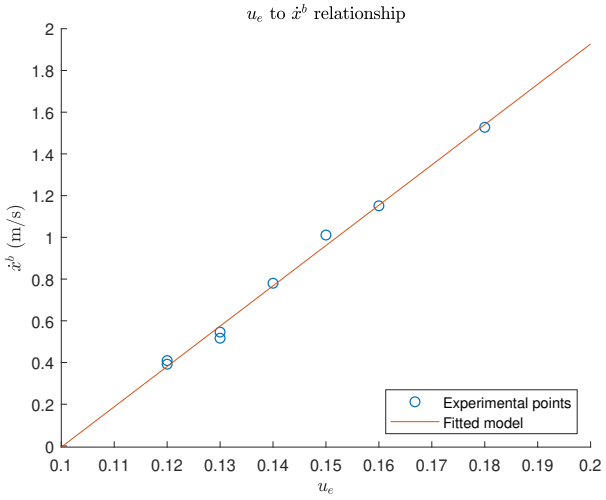


Figure 4.5: Steady-state velocity as a function of the engine input, u_e.

was performed ignoring the points with an absolute value of u_d bigger than 0.6 due to the non-linearity in those areas.

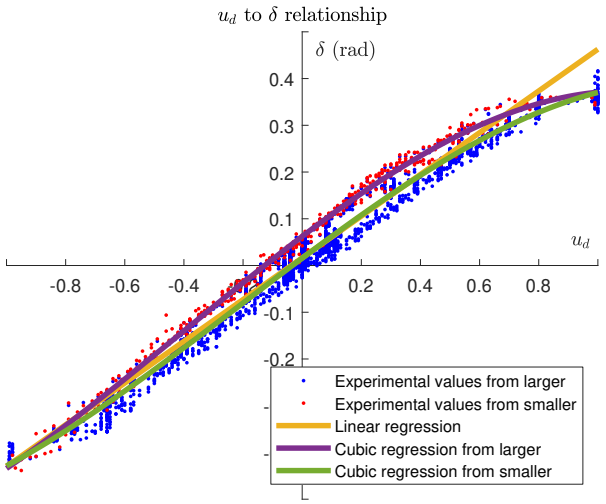


Figure 4.6: Relationship between the steering input u_d and the steering angle δ.

It can be seen that the value of δ depends on the present value of u_d as well as its past values, a behavior called hysteresis. Regardless, ignoring values of u_d far away from the origin, the relationship can be approximated by a first order polynomial.

4.1.4 Side Forces

The side forces on the wheels are assumed to take the form of equation 4.10 as in [24] and [25].

$$F_y = -C_\alpha \alpha \tag{4.10}$$

In this expression, α is the slip angle of the tire and C_α is the tire cornering stiffness. Using small-angle approximations, the front and rear slip angles, take the form shown in equations 4.11 and 4.12

respectively.

$$\alpha_F = \frac{\dot{y}^b + a\dot{\psi}}{\dot{x}^b} - \delta \tag{4.11}$$

$$\alpha_R = \frac{\dot{y}^b - b\dot{\psi}}{\dot{x}^b} \tag{4.12}$$

The radius r of a circular trajectory predicted by the car's model is fitted to experimental results to estimate the value of the tire cornering stiffness C_α. In a circular trajectory, assuming there is no side slip, the lateral forces acting on the car should be equal to the centripetal force as in equation 4.13. Also, the angular velocity is equal to the tangential velocity divided by the radius of the trajectory as in equation 4.14. The tangential velocity $\dot{x}^b$ is equal to the steady-state velocity from equation 4.9 and the steering angle δ is given by the linear regression shown in figure 4.6.

$$F_y^R + F_y^F \cos\delta = m\dot{\psi}^2 r \tag{4.13}$$

$$\dot{\psi} = \frac{\dot{x}^b}{r} \tag{4.14}$$

Solving equation 4.3 with angular acceleration $\ddot{\psi}$ equal to zero, an expression for the sideways velocity $\dot{y}^b$ is obtained. From equations 4.10, 4.11, 4.12, 4.13, 4.14, 4.9 and the steering angle δ as a function of the steering input u_d, an expression for the trajectory radius r as a function of the tire cornering stiffness C_α, the engine input u_e and the steering input u_d is obtained. The engine input u_e is replaced by the one used in each experiment and the result is the radius of the trajectory r as a function of the steering input u_d with the tire cornering stiffness C_α as a free parameter. This parameter is then fitted to the experimental data with the results shown in figure 4.7.

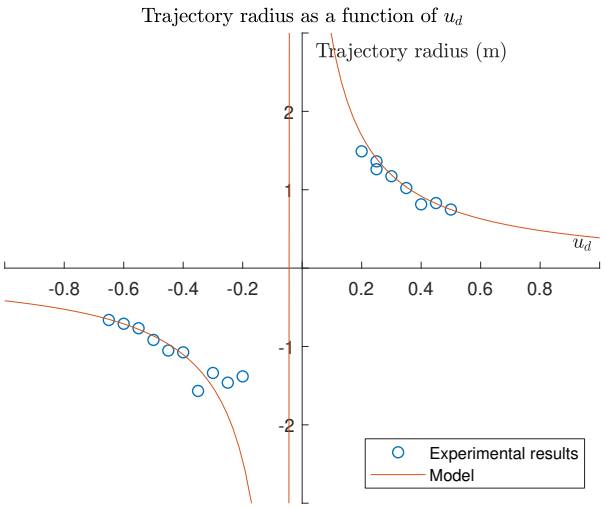


Figure 4.7: Trajectory radius r as a function of u_d with u_e equal to 0.14.

The negative radius is simply due to the direction the car is turning. From equation 4.14, a negative radius means the car is turning to the right and a positive one means the car is turning to the left.

4.1.5 State-Space Representation

The expressions for a state-space representation of a system are the following ones:

$$\dot{x} = Ax + Bu \tag{4.15}$$

$$\hat{y} = Cx + Du \tag{4.16}$$

In these expressions, the vector x is the state variables vector, u is the input vector and y is the output vector. The matrices A, B, C and D are constant matrices obtained from the linearization of the car's model around the operating point. The state variables vector is defined in the following equation:

$$x = \begin{bmatrix} \dot{x}^b \\ \dot{y}^b \\ \dot{\psi} \end{bmatrix} \tag{4.17}$$

The inputs vector has the inputs u_e and u_d. The operating point for the linearization is u_e equal to 0.14, $\dot{x}^b$ equal to the corresponding steady-state velocity and all the other state-space variables along with u_d equal to zero. This corresponds to the car driving at low speed in a straight line. The outputs vector contains the estimates of the outputs from the sensors which are described in the following sections.

The system is discretized with a sample time of $0.05\,\mathrm{s}$ which converts the equations 4.15 and 4.16 into their discrete counterparts:

$$x_{k+1} = Fx_k + Gu_k \tag{4.18}$$

$$\hat{y}_k = Cx_k + Du_k \tag{4.19}$$

4.2 Positioning System

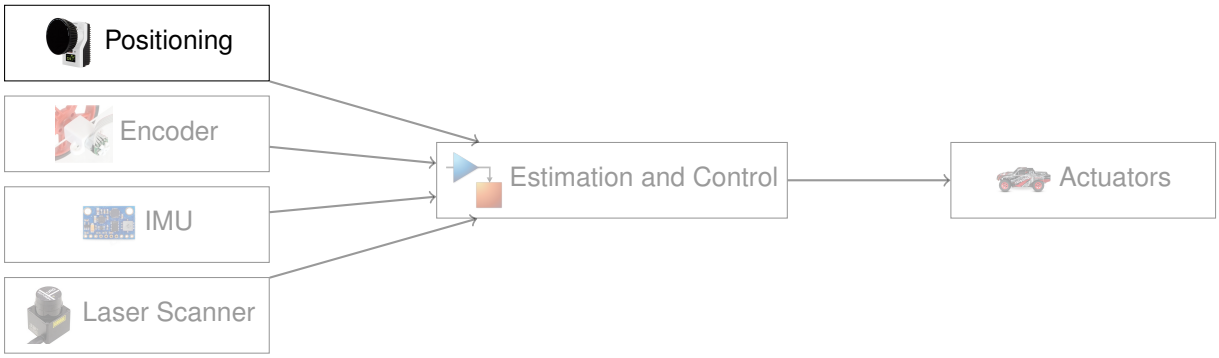
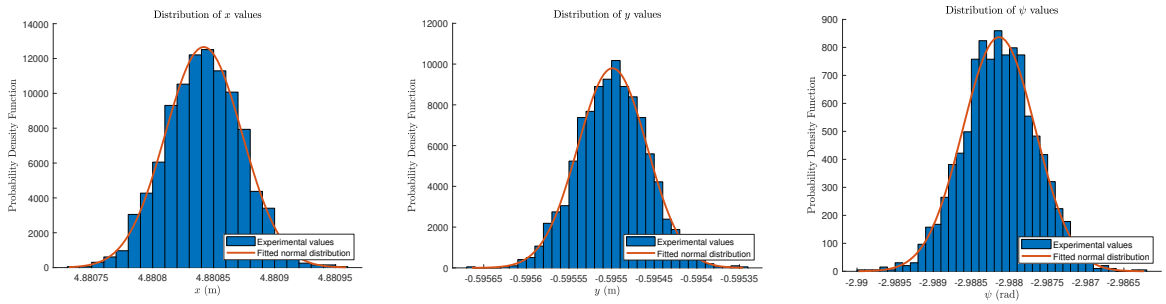


Figure 4.8: Diagram of the architecture implemented, the block discussed in this section is highlighted.

The positioning system can measure, among other things, the position and orientation of a body in three-dimensional space. In this thesis, only the position and orientation on the horizontal plane is used. The three degrees of freedom on the horizontal plane are the positions x and y and the rotation around

the z axis, ψ. To characterize the noise, these three values were measured with the car standing still. The values returned by the positioning system are shown, with a fitted normal distribution, in figure 4.9.



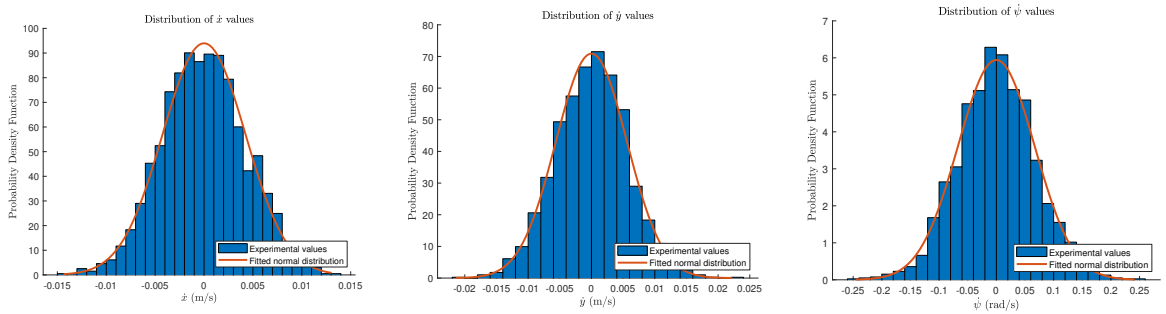
(a) x (b) y (c) ψ

Figure 4.9: Values returned by the positioning system with the car standing still.

The velocities are calculated using a finite difference between the current and previous positions. Since each position measurement is perturbed by a normally distributed noise with standard deviation σ_w, the standard deviation of each velocity measurement is given by the following expression where Δt is the sample time of the positions:

$$\sigma_{w_v} = \sqrt{2}\frac{\sigma_w}{\Delta t} \tag{4.20}$$

This can be verified by comparing the histograms in figure 4.9 with the ones in figure 4.10, which show the velocities computed with the same data, recorded with a sample time of $0.01\,\mathrm{s}$.



(a) $\dot{x}$ (b) $\dot{y}$ (c) $\dot{\psi}$

Figure 4.10: Velocity values computed from the positioning system data recorded with the car standing still.

4.3 Encoder

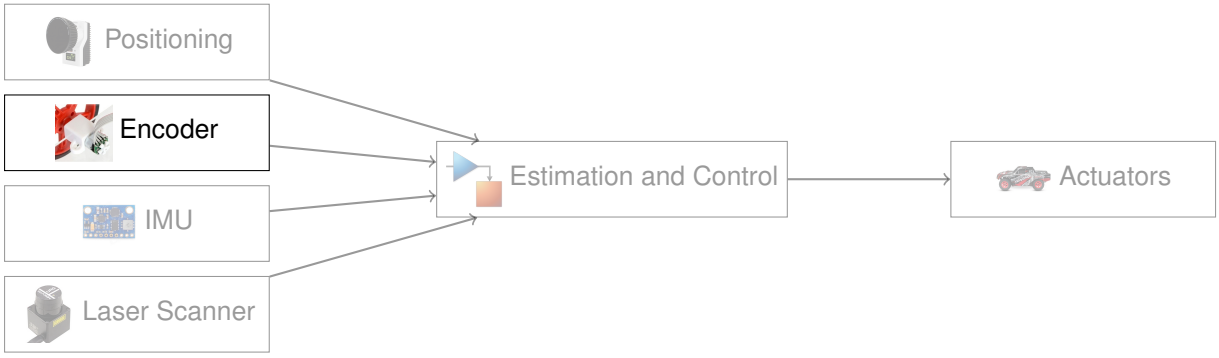


Figure 4.11: Diagram of the architecture implemented, the block discussed in this section is highlighted.

An encoder was mounted into the car, measuring the speed of the main shaft which is directly connected to the motor. To do it, a plastic part of the car was cut to expose a gear that is connected to the main shaft and receives power from the electric motor gear. A 3D printed part was designed that supports a new shaft with a gear connected to the main one. This new gear is the same as the one in the motor shaft which means it spins at the same speed. Finally, this new shaft was used to drive the encoder wheel.

When the encoder wheel spins, it interacts with two sets of a photoresistor and a Light-emitting diode (LED). When a tooth of the encoder wheel is on top of a photoresistor-LED pair it reflects the LED's light back into the photoresistor, lowering its resistance. The encoder's printed circuit board (PCB) simply contains a few traces and resistances that produce an analog electric potential signal as a function of the photoresistor's resistance. The two photoresistor-LED pairs are $90°$ out of phase, meaning the two signals can be combined to know the direction of spin of the encoder.

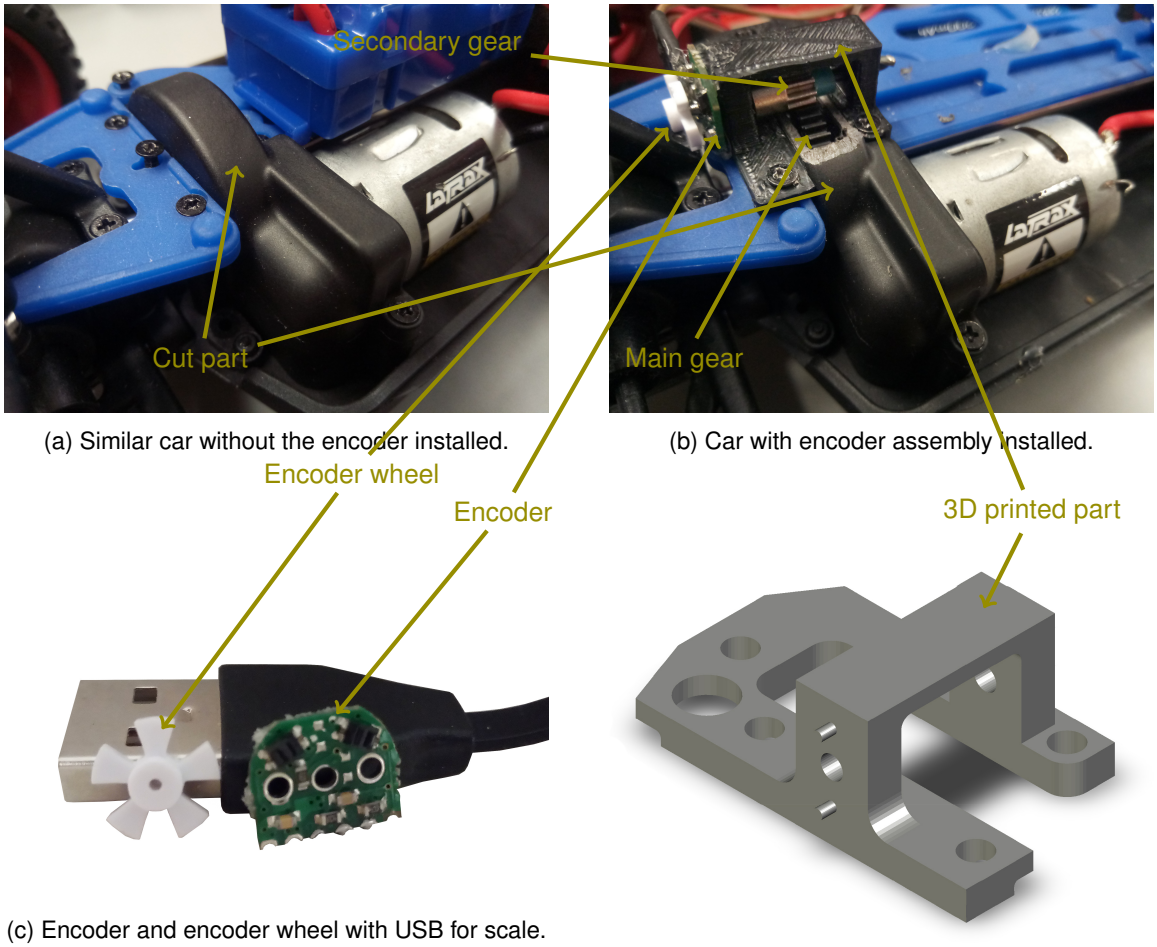
Two photographs depicting the before and after of this assembly can be seen in figures 4.12a and 4.12b. In figure 4.12c, the encoder used is shown and figure 4.12d shows the 3D model of the 3D printed part.

Multiple samples of the car running in a straight line were recorded with the positioning system, along with the number of encoder pulses per second. This data was used to calculate the relationship between the maximum velocity $\dot{x}^b$ and the maximum number of encoder pulses per second which are plotted in figure 4.13.

This data was fitted to equation 4.21 which models the expected linear relationship between the car's velocity and the number of pulses from the encoder.

$$\hat{y}_e = C_{enc}\dot{x}^b \tag{4.21}$$

To estimate the encoder variance, the car was run again with the positioning system and the difference between its velocity estimate and the estimate from the positioning system is plotted in figure 4.14.



(a) Similar car without the encoder installed.

(b) Car with encoder assembly installed.

(c) Encoder and encoder wheel with USB for scale.

(d) 3D model of the part used for the encoder installation.

Figure 4.12: Encoder assembly. The ”Cut part” is the plastic part that was cut to expose the main shaft’s gear. The ”Secondary gear” is the gear, with the same dimensions as the motor gear, which connects the main shaft’s gear to the encoder shaft. The ”Main gear” refers to the main shaft’s gear, a gear that is connected to the main shaft which transmits power to the wheels. The other three parts’ names are self-explanatory.

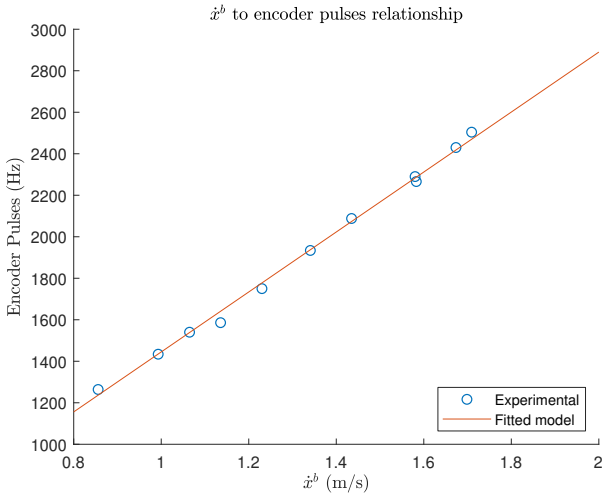


Figure 4.13: Relationship between velocity $\dot{x}^b$ and the number of encoder pulses per second.

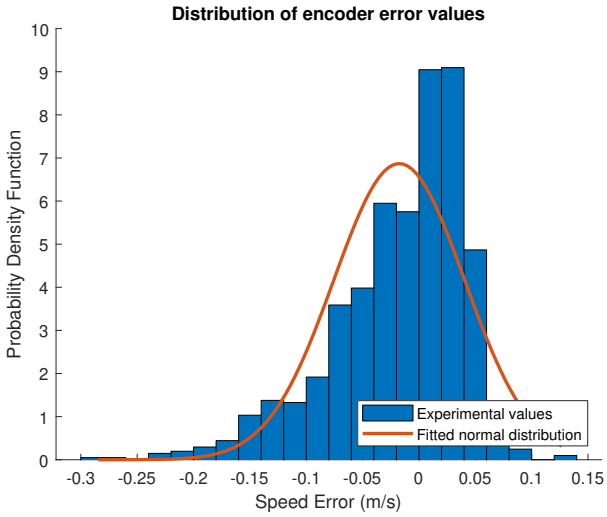


Figure 4.14: Error distribution of encoder.

4.4 Inertial Measurement Unit

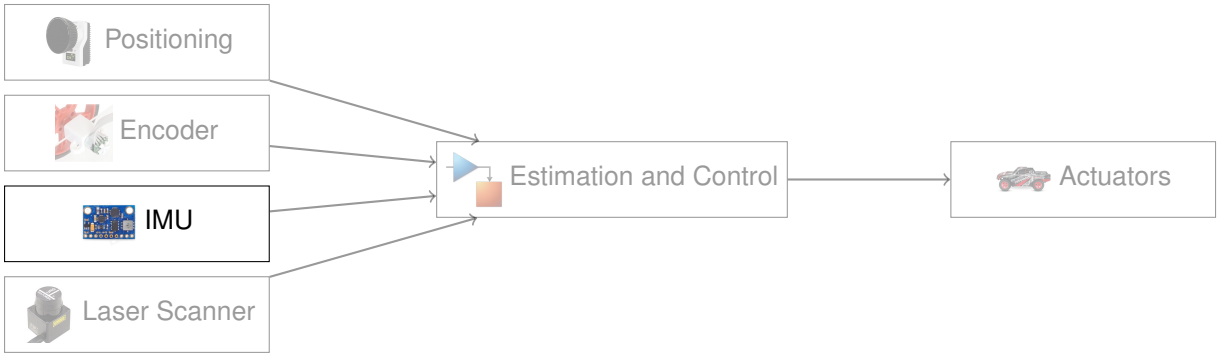
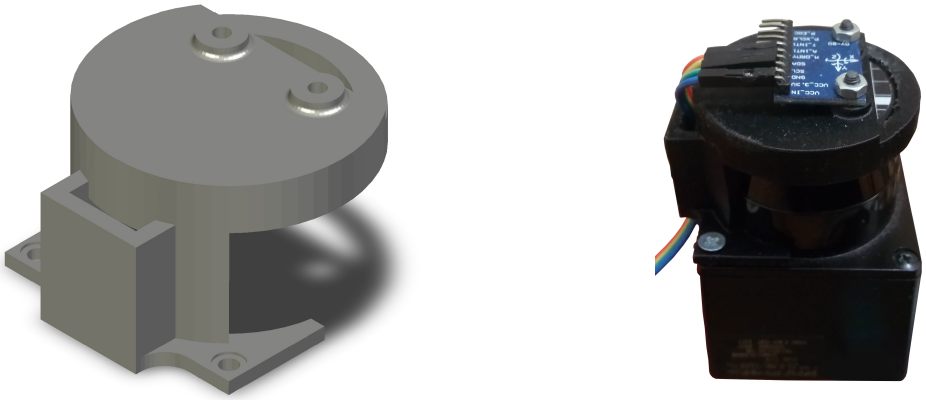


Figure 4.15: Diagram of the architecture implemented, the block discussed in this section is highlighted.

The car was received with an Inertial Measurement Unit (IMU) already installed but it was repositioned to the top of the laser scanner where it could be assembled more securely. The 3D printed part used to assemble the IMU is shown in figure 4.16.

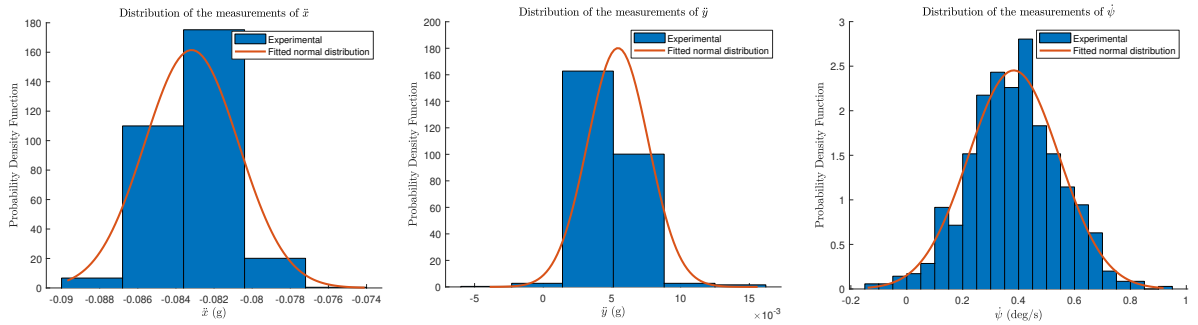


(a) 3D model of the part used to assemble the IMU.

(b) Photo of part assembled on top of the laser scanner.

Figure 4.16: IMU assembly.

The values measured with the IMU are the accelerations in the x and y direction and the angular velocity around the vertical axis. These values, $\ddot{x}^b$, $\ddot{y}^b$ and $\dot{\psi}$, follow a normal distribution as shown in figure 4.17. The small number of bins in the first two histograms is due to the limited resolution of the accelerometer.



(a) Acceleration along the x axis, $\ddot{x}$, in g's.

(b) Acceleration along the y axis, $\ddot{y}$, in g's.

(c) Angular velocity around the vertical axis, $\dot{\psi}$, in degrees per second.

Figure 4.17: Histogram of IMU measurements.

4.5 Laser Scanner

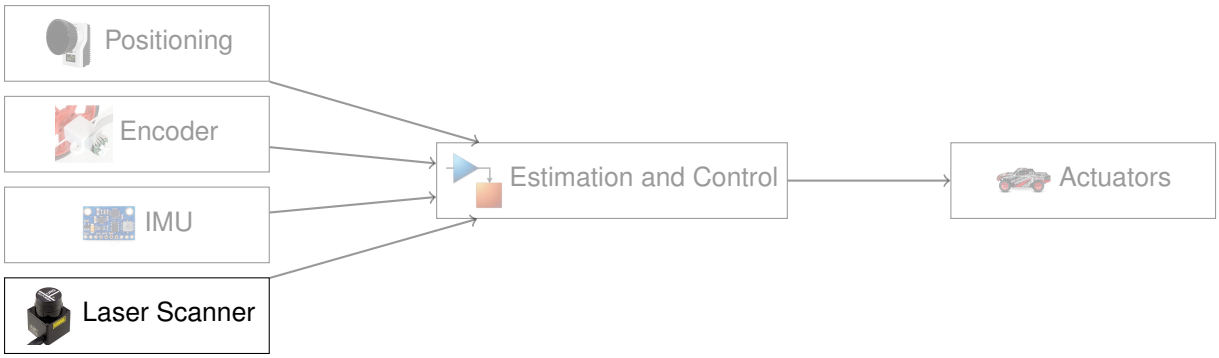


Figure 4.18: Diagram of the architecture implemented, the block discussed in this section is highlighted.

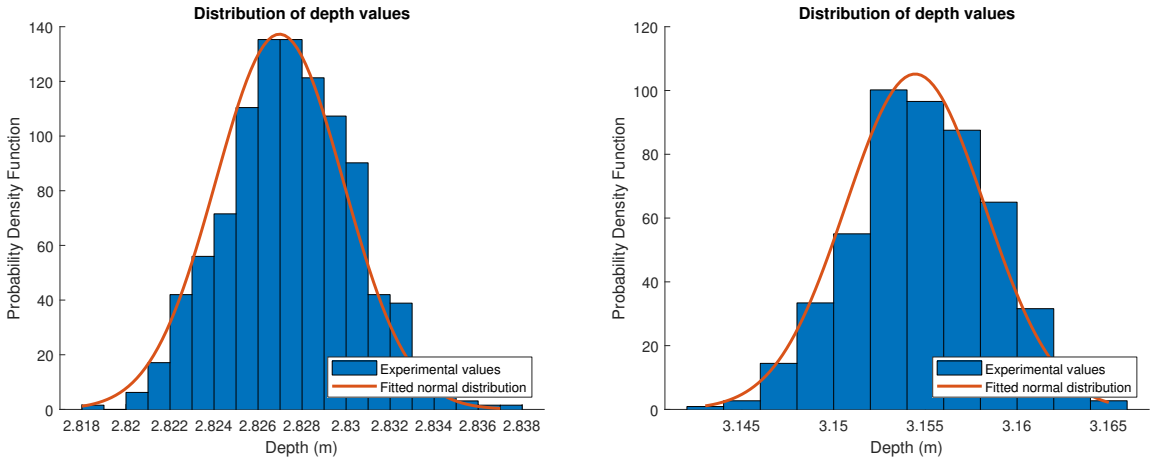
In this thesis, the point clouds are produced by a laser scanner, a device with a rotating LIDAR (light detection and ranging) device that measures the distance to the nearest surface. By spinning the LIDAR, the laser scanner is able to collect samples of multiple points in the environment. Not capturing the environment in one instant is a disadvantage when it is used in a moving body as is done in this thesis. Using estimates from a state observer it is possible to minimize this problem by correcting the point cloud to a single point in time. Regardless, every sample collected in one rotation of the LIDAR is grouped into a scan that is processed as if they were taken all at once.

The laser scanner used is the URG-04LX-UG01 from Hokuyo [26]. It is a 2D scanner, meaning the LIDAR is only spinning around one axis. It has a range of around 5 m, collects 682 samples per rotation and rotates at 10 rotations per second. Since it is a 2D scanner it is used with the LIDAR rotating around the vertical axis, meaning that every LIDAR sample is characterized by two values: the angle θ around

the vertical axis in which the sample was taken and the depth r of the nearest surface in that direction. The angle values of the samples taken in each rotation are constant and all the samples collected with the same angle value can be grouped into a ray whose depth value varies with time. The transformation of the LIDAR samples from a scan into a point cloud is simply a transformation from polar coordinates into cartesian ones:

$$p = r \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{4.22}$$

Two aspects of the laser scanner noise were characterized: the variance and the bias. Multiple sets of scans were collected in different places with the laser scanner standing still. The depth value of each ray follows a normal distribution as shown in figure 4.19 for two example rays.



(a) Random ray 1.

(b) Random ray 2.

Figure 4.19: Distribution of values of two rays.

From these sets of scans, the variance as a function of depth can be calculated. Each ray of each set of scans is fitted to a normal distribution and the standard deviation as a function of the mean value is plotted in figure 4.20. It can be seen that the standard deviation does not depend on the depth.

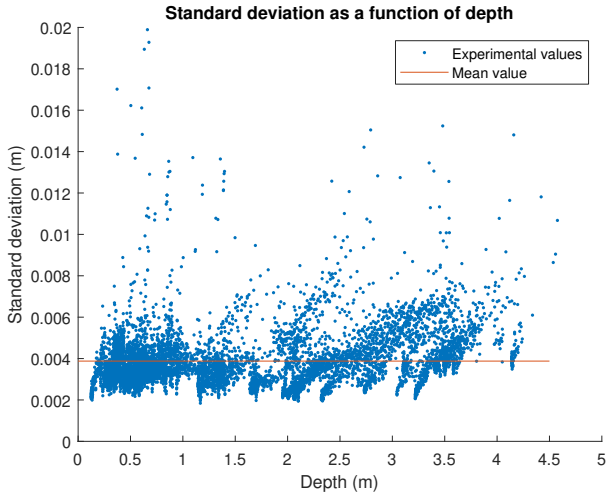


Figure 4.20: Standard deviation, σ, as a function of depth r.

The bias is modeled by placing a planar surface roughly perpendicular to the laser scanner and at various distances. Since the surface is planar, the depth values should follow equation 4.23 where r_{min} is the depth of the point closest to the laser scanner and β is the angle between the surface and the surface normal to the closest point's ray.

$$r(\theta) = \frac{r_{min}}{\cos(\theta - \beta)} \tag{4.23}$$

Multiple test runs were performed, with multiple scans each, and the average of each ray is taken. The previously mentioned planar surface is fitted to these ray averages. The difference between the ray averages and the fitted surface is then recorded as the ray bias. The data points obtained are in figure 4.21.

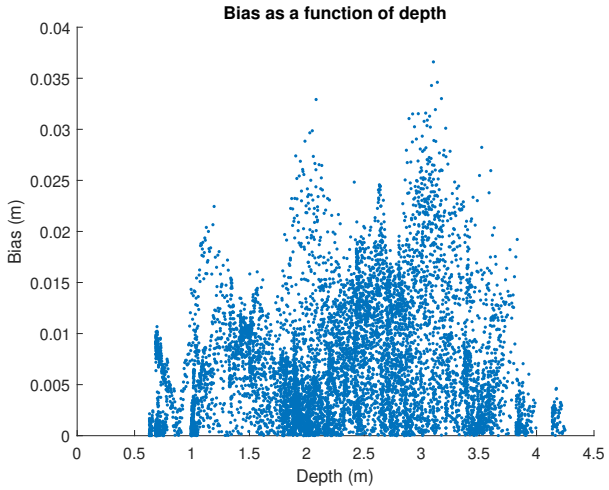
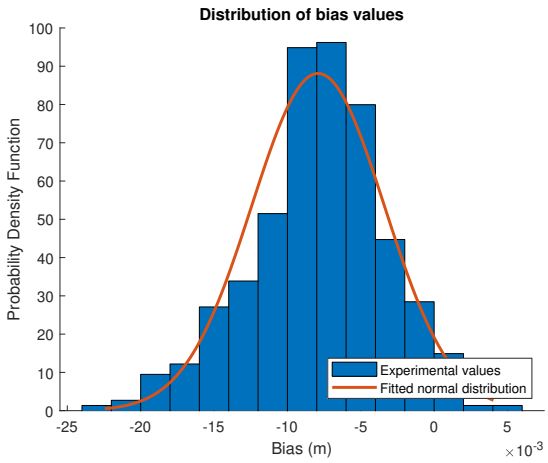


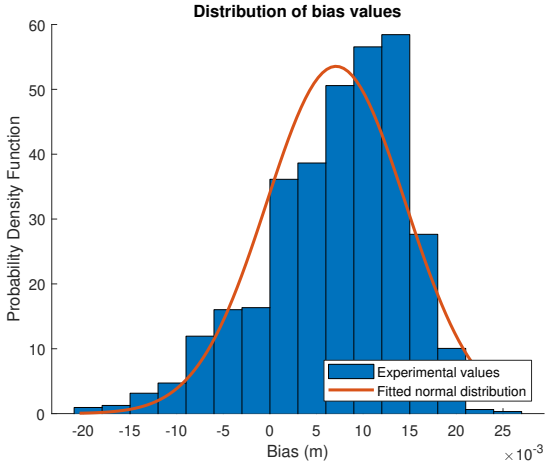
Figure 4.21: Bias as a function of depth r.

To estimate how the bias varies with depth, the previous data points are put into bins and the values in each bin are fitted to a normal distribution. Figure 4.22 shows the values and normal distribution of two random bins demonstrating that the bias values follow a normal distribution when considering points with the same depth.

The standard deviation of each bin as a function of the mean value is plotted in figure 4.23. It shows that the standard deviation of the bias is proportional to the point's depth.



(a) Bin of points with depth between 1.1 and 1.4 meters.



(b) Bin of points with depth between 2.3 and 2.6 meters.

Figure 4.22: Distribution of bias values inside two bins.

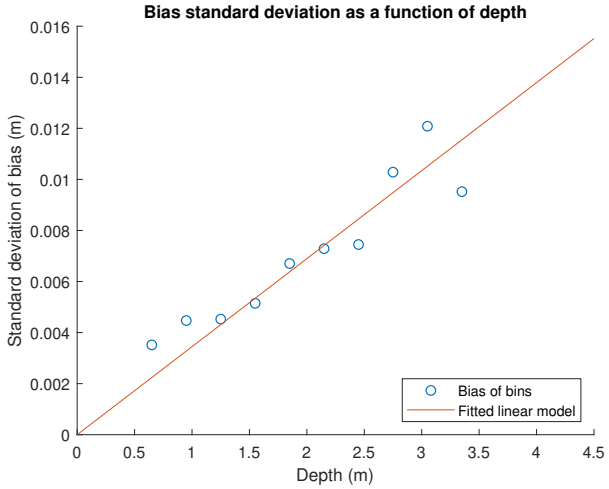


Figure 4.23: Bias standard deviation as a function of depth r.

4.6 State Observer

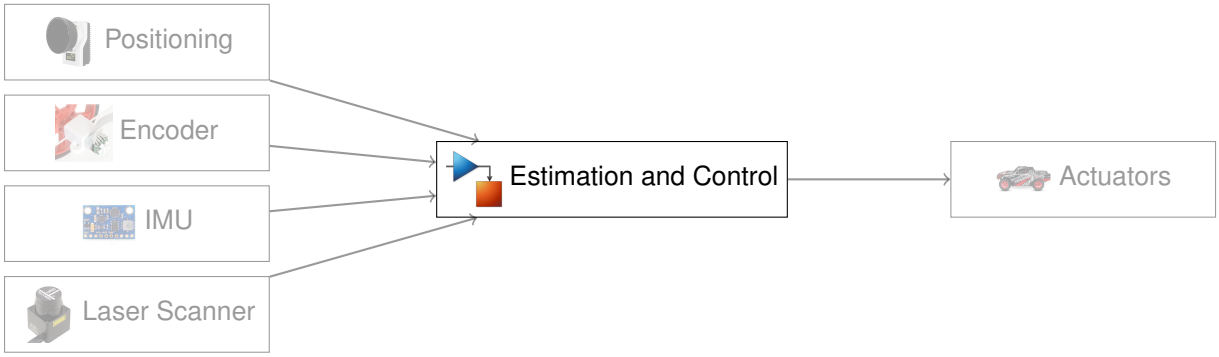


Figure 4.24: Diagram of the architecture implemented, the block discussed in this section is highlighted.

Two state observers were constructed, a Kalman filter and an observer that uses the unfiltered data from the sensors. The Kalman filter is described in section 4.6.1 and the simplified observer in section 4.6.2. Both observers are then used to estimate the relative pose between the two samples, that is, t_x^m, t_y^m and t_θ^m.

4.6.1 Kalman Filter

The equations of the Kalman filter used are the following:

$$x_{k|k-1} = Fx_{k-1|k-1} + Gu_{k-1} + L_p\left(y_{k-1} - Cx_{k-1|k-1} - Du_{k-1}\right) \tag{4.24}$$

$$x_{k|k} = x_{k|k-1} + L_c\left(y_k - Cx_{k|k-1} - Du_k\right) \tag{4.25}$$

In the first equation, the matrix L_p corrects the prediction of the model given by the first part of the expression using the difference between the real sensor outputs and the predicted outputs. In the second equation, the prediction is corrected with the new measurements from the sensors using the matrix L_c. The output vector contains the planar accelerations, the angular velocity around the vertical axis and the forward velocity. The first three outputs come from the IMU while the fourth one comes from the encoder.

The matrices L_p and L_c are computed by MATLAB and, besides the state-space matrices F, G, C, and D, they require the variance of each sensor and each input for their computation. The variances of the sensors used were the ones computed before in sections 4.2 to 4.5. The variances for the inputs were chosen manually by minimizing the mean error between the state variables estimation from the Kalman filter and from the positioning system. The manual tuning was performed because it is hard to measure and quantify the variance of the driving motor and the steering servo.

In order to estimate the coordinate transformation between both samples, t^m, equations 4.4 and 4.5 are integrated assuming all three state variables change linearly with time.

4.6.2 Simplified Observer

A simplified observer that uses the sensor data directly was also built. The value of $\dot{x}^b$ is taken by replacing the average of the current and the previous value returned by the encoder in equation 4.21 and solving for $\dot{x}^b$. The value of $\dot{\psi}$ is taken as the average of the current and previous value returned by the IMU. To compute $\dot{y}^b$, a circular trajectory and a kinematic model is assumed. In this kind of trajectory, the t_y^m between two sample steps in the reference frame of the first one is given by the following equation:

$$t_y^m = \frac{\dot{x}^b}{\dot{\psi}} (1 - \cos \psi) \tag{4.26}$$

Replacing ψ and t_y^m with $\dot{\psi}\Delta t$ and $\dot{y}^b \Delta t$ and linearizing around $\dot{\psi}$ equal to zero, the following equation for $\dot{y}^b$ as a function of the other two state variables is obtained:

$$\dot{y}^b = \dot{x}^b \frac{\Delta t}{2} \dot{\psi} \tag{4.27}$$

In this expression, Δt is the sample time of the state observer. The values of t_x^m, t_y^m and t_θ^m are the product of the sample time with the corresponding state variables.

4.6.3 Coordinate Transformation

The sample time of the laser scanner is set to $0.1\,\mathrm{s}$ which is different from the $0.05\,\mathrm{s}$ of the state observer. Because of this, there are two state observer sample steps for each laser scan. Due to the possibility that the registration algorithm will converge to a value away from the real one, a trust-region is constructed from the state observer estimate of the relative pose. If the value returned by the registration algorithm falls outside this trust-region, the result is discarded and the relative pose from the state observer is used instead. The trust-region used has the shape of an ellipse for the translation part of the transformation and a scalar interval for the orientation part. The equations for the boolean values to be evaluated are the following:

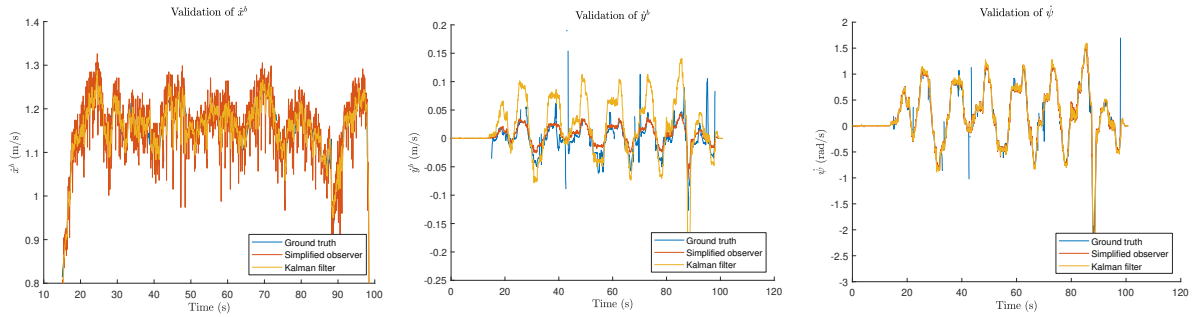
$$B_\circ = \left(\frac{t_x^r - t_x^m}{C_{ex}} \right)^2 + \left(\frac{t_y^r - t_y^m}{C_{ey}} \right)^2 <= 1 \tag{4.28}$$

$$B_\theta = |t_\theta^r - t_\theta^m| <= C_{e\theta} \tag{4.29}$$

Here, the superscripts r and m refer to the parameters estimated using the registration algorithm and the model of the car respectively. C_{ex}, C_{ey} and $C_{e\theta}$ are scalar values tuned manually.

4.6.4 Validation

To validate the state observers, the car was driven around while the values of u_e, u_d, x^e, y^e and ψ were recorded, the latter three of which with the positioning system. From the positioning system data, an accurate estimate of $\dot{x}^b$, $\dot{y}^b$ and $\dot{\psi}$ is calculated. In figure 4.25, these results are compared with the estimates from both the Kalman filter and the simplified observer.



(a) Validation of the prediction of $\dot{x}^b$. (b) Validation of the prediction of $\dot{y}^b$. (c) Validation of the prediction of $\dot{\psi}$.

Figure 4.25: Validation of the observers.

Both observers estimate the forward velocity $\dot{x}^b$ and angular velocity $\dot{\psi}$ with acceptable errors. While the Kalman filter has large errors on the estimate of $\dot{y}^b$, it has smaller errors on the estimate of $\dot{x}^b$. The errors in the estimate of $\dot{y}^b$ are more easily corrected by the registration algorithm since the surfaces are normal to this direction. Because of this, the Kalman filter was chosen as the state observer.

4.7 Software Implementation

Four pieces of software were written:

- Written in C++, runs on the car's onboard Raspberry Pi and focuses on the low-level operations such as interacting with the sensors and actuators.
- Written in Python and runs on a laptop connected to the Raspberry Pi's Wi-Fi network. The main function of this piece of software is the reception, display, and logging of the data from the car's sensors so that it can be processed offline.
- Written in MATLAB and performs CLM using the data logged by the Python software.
- Five Simulink blocks that interact with the car's sensors and actuators.

4.7.1 C++ Software

This piece of software, written in C++, is installed onto the car's onboard Raspberry Pi. It focuses on the low-level operations such as collecting data from the sensors and sending signals to the actuators. To enable other computers to interact with this piece of software, a Wi-Fi network is created by the Raspberry Pi, using freely available software. Other computers then connect to this Wi-Fi network and send signals and receive data from the car's C++ software. This piece of software can be divided into 10 parts:

Actuator Sends the signals to the actuators

Command Receiver Receives commands over UDP

Controller Implements some controllers

Encoder Interprets the data from the encoder

IMU Collects data from the IMU

Laser Scanner Collects data from the laser scanner

Positioning System Receives the data sent by the positioning system

Simulator Interprets a data log file created by the Python program and simulates it

State Observer Implements both state observers

UDP Stream Streams the data from the sensors over UDP

Two other similar cars, with a subset of the sensors available in the main car, were tested and used with this software.

4.7.1.1 Actuator

This module simply transforms the u_e and u_d values set by the Controller module into PWM signals that are then sent, using a software library, to the Raspberry Pi pins that are connected to the actuators. The PWM signal corresponding to u_e is sent to the motor's Electronic Speed Control (ESC), while the one corresponding to u_d is sent directly to the servo.

4.7.1.2 Command Receiver

This module receives and interprets User Datagram Protocol (UDP) packets sent over the network. It is used to start the UDP Stream module and receive signals that are then passed onto the Controller module.

4.7.1.3 Controller

Three controller modes are available. The first one consists in directly sending the actuator signals received by the Command Receiver to the Actuator module. The second one is a proportional controller that uses the ψ estimated by the State Observer module to maintain a ψ reference received by the Command Receiver module. The third controller implemented is one that uses the data from the Laser Scanner module to try to avoid obstacles. It locates the gap in the laser scan point cloud that is closer to the direction in front of the car and outputs the direction of that gap as a ψ reference. This ψ reference is then passed through a proportional controller similar to the one described before.

4.7.1.4 Encoder

The two analog signals generated by the encoder are connected to two Raspberry Pi pins. These pins are digital, meaning that when the signal is above a certain threshold value it is interpreted as a one and when it is below the threshold it is a zero. The digital signal interpreted by the Raspberry Pi can be seen in figure 4.26.

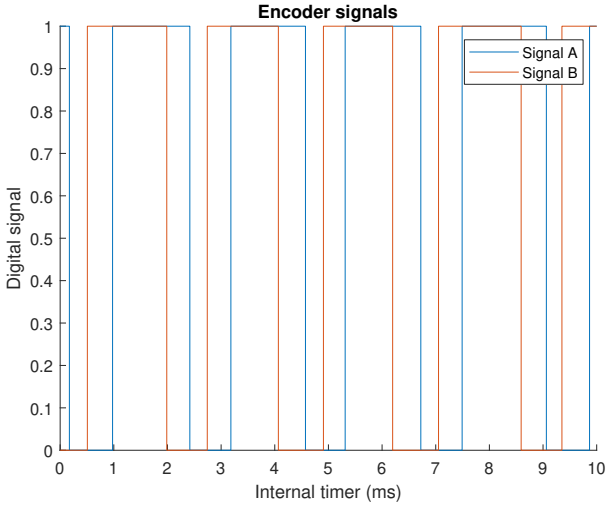


Figure 4.26: Encoder digital signals over time.

These digital signals are then interpreted by the C++ software, with an algorithm described in [27], to count the number of encoder pulses.

A bit code is given for each state with the first bit representing the state of the signal A and the second bit representing the state of the signal B. There are four possible states 00, 01, 10, and 11. The algorithm implemented needs only an integer that can work as a bit array and an array of corresponding positive, negative or zero position change. When the state changes, the bits in the bit array are moved two positions to the left and the new state is inserted in the last two positions. This state change code is then interpreted as an integer and looked up in the positions delta array. The contents of this array can be seen in table 4.1.

Bit array	Decimal integer	Position delta
0000	0	0
0001	1	1
0010	2	-1
0011	3	0
0100	4	-1
0101	5	0
0110	6	0
0111	7	1
1000	8	1
1001	9	0
1010	10	0
1011	11	-1
1100	12	0
1101	13	-1
1110	14	1
1111	15	0

Table 4.1: Positions delta array.

From this table, it can be seen that when the encoder wheel is spinning one way and the signal B is ahead of signal A, as in figure 4.26, the resulting position delta is positive. When it is spinning the other way the position delta is negative.

While this algorithm tracks the position, it can drift over time due to lost encoder pulses. Because of this, only the difference in the position since the previous sample step is used.

4.7.1.5 Inertial Measurement Unit

The connection to the Inertial Measurement Unit (IMU) is made over I^2C which is a standard protocol used to exchange data between microchips, including the Raspberry Pi. The I^2C communication is performed with the help of a software library and consists in reading and writing the appropriate registers in the IMU chip with the most important being the ones that contain the result of the acceleration, angular velocity, and magnetic strength measurements.

Two different models of IMU are supported with the software detecting which one is connected on initialization.

4.7.1.6 Laser Scanner

This module communicates with the laser scanner and receives the data from it. It makes use of a software library made available by the laser scanner manufacturer Hokuyo.

4.7.1.7 Positioning System

The positioning system used is manufactured by Qualisys who also makes available a software library, used by this module, to interpret the data sent by the system. This module can also automatically request the positioning system to start streaming the data on initialization.

4.7.1.8 Simulator

This module reads a data log file produced by the Python software and overrides the data collected by the sensors. This results in the UDP Stream module streaming the simulated data. This is useful to test the online CLM algorithm without requiring experiments to be performed live. It can also work on a desktop or laptop computer allowing the tests to be performed much more easily.

4.7.1.9 State Observer

This module implements both the Kalman filter and the simplified observer developed in sections 4.6.1 and 4.6.2 respectively. The math is performed using a matrix operations software library.

4.7.1.10 UDP Stream

This module creates and sends UDP packets. One packet is sent for each sample step and contains the data read by all the sensors along with the actuator signals.

4.7.2 Python software

The Python software runs on a computer that is connected to the car's Raspberry Pi's Wi-Fi network and has four modules. One of the modules receives, displays, and logs the data sent by the UDP Stream C++ module. The other three modules interact with each of the three C++ Controller modes mentioned in section 4.7.1.3, one Python module for each C++ controller. These latter modules receive keyboard inputs, such as pressing the arrow keys to accelerate, interpret them, and send them to the Raspberry Pi over UDP.

4.7.3 MATLAB software

Three MATLAB functions were written, each implements one of the registration algorithms studied in this thesis: NDT, ICP, and NICP. A MATLAB class was written that implements both the Kalman filter and the simplified observer designed in sections 4.6.1 and 4.6.2 respectively. Another MATLAB class was written that implements the CLM algorithms described in sections 2.5.1 and 2.5.2. A script was written that uses these pieces of software to perform offline CLM using the data logged by the Python software.

4.7.4 Simulink software

Five S-functions were developed which implement the functionality of the five C++ modules: Actuator, Encoder, IMU, Laser Scanner and Positioning System. An S-function is a type of function which follows a specific template and is used by a Simulink S-function block to run code written in other languages. The language used is C++, since it allows the reutilization of much of the C++ code, the S-functions simply call the functions of the C++ modules.

The CLM algorithm described in section 2.5.3 is implemented as a Simulink model that runs in real-time which uses the previously mentioned S-function blocks to interact with the car's sensors and actuators. A controller that makes the car follow a trajectory defined by a series of points was also developed. The block diagram in figure 4.27 shows how the state observer constructed in section 4.6.1 interfaces with the CLM algorithm and the controller.

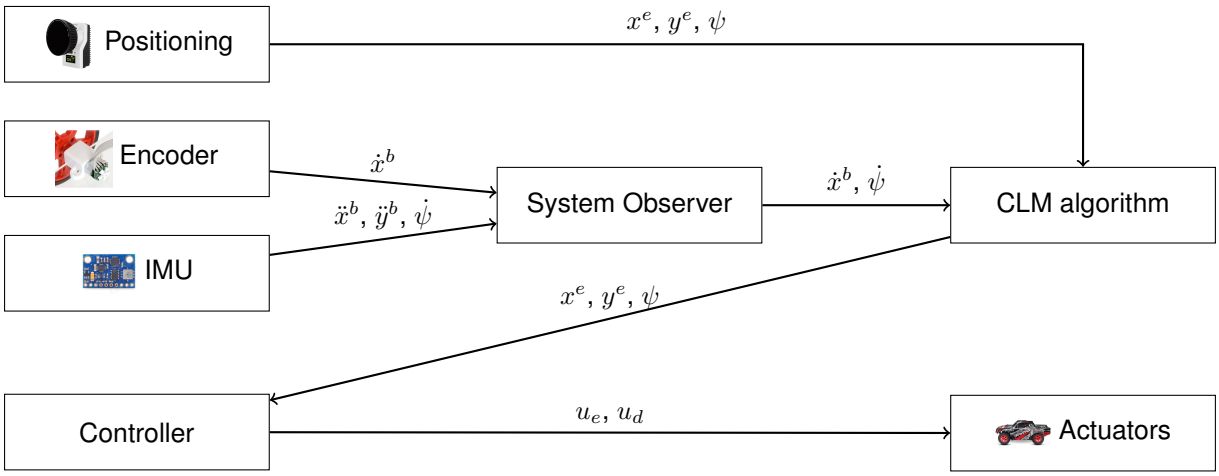


Figure 4.27: Block diagram of the CLM algorithm that uses data from the positioning system.

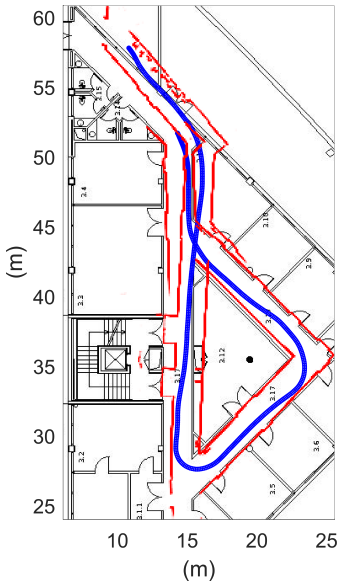
Chapter 5

Results

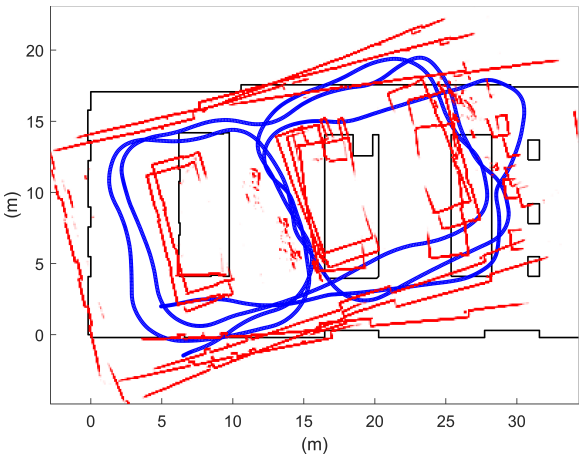
In this chapter, the results of the implemented CLM algorithms are shown. In section 5.1, the sequential map construction algorithm, in section 5.2, the network-based CLM algorithm, and in section 5.3, the CLM algorithm which uses data from a positioning system.

5.1 Sequential Map Construction

Using the algorithm of the sequential map construction described in section 2.5.1, with the registration algorithm NICP described in section 3.1 and the Kalman filter described in section 4.6.1, two different environments, shown in figure 5.1, were reconstructed.



(a) Office building



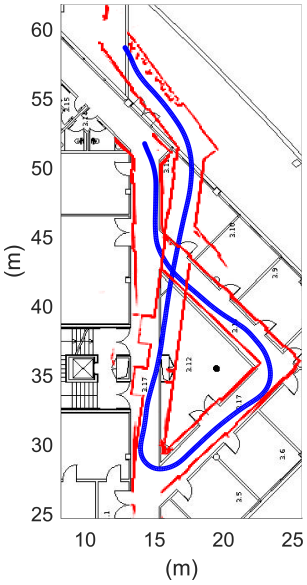
(b) Garage

Figure 5.1: Sequential CLM performed in two indoor environments.

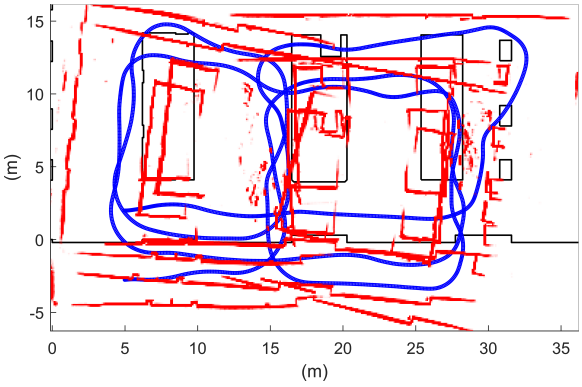
The maps show the building plants in the background, the trajectory of the car as a blue line and the obstacles detected by the laser scanner in red. The environment in figure 5.1a is a university's office building and its plant was obtained from the university's website. The environment in figure 5.1b

is a garage in the basement of a residential building and its plant was constructed from measurements performed using a measuring tape.

To determine the importance of the initial transformation provided by the state observer, the same map construction algorithm was used with the same data but using the state observer as the sole source of the relative poses. The result can be seen in figures 5.2a and 5.2b and the cumulative sum of the absolute values of the differences between the relative pose parameters can be seen in figures 5.3a and 5.3b.

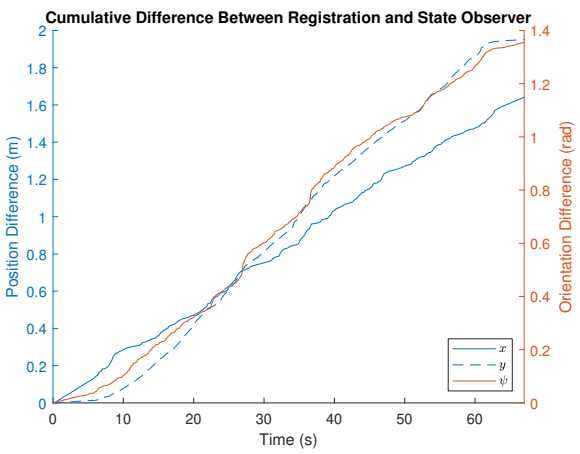


(a) Office building

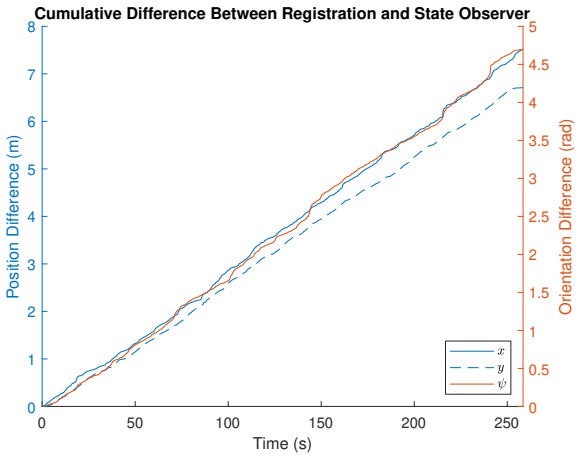


(b) Garage

Figure 5.2: Sequential CLM performed only with relative poses from the state observer.



(a) Office building

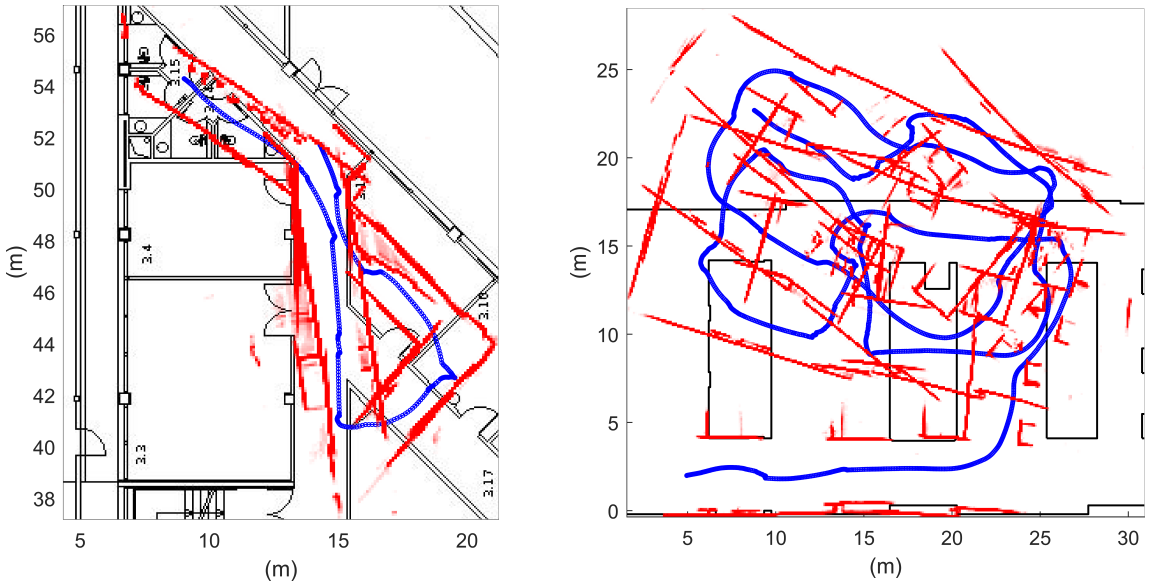


(b) Garage

Figure 5.3: Cumulative absolute difference between the relative pose parameters estimated by the registration algorithm and the initial transformation provided by the state observer.

It can be seen that the registration algorithm does not improve very much the transformation provided by the state observer. Also, the registration algorithm itself does not provide good results as can be seen in figure 5.4. Large errors happen specially when most of the normals of the point cloud are along one

direction, as in corridors.



(a) Office building

(b) Garage

Figure 5.4: Sequential CLM using the registration algorithm without the initial transformation from the state observer.

Regardless, the registration algorithm is crucial in correcting the inevitable dead-reckoning error when only the relative pose between consecutive points in time is used. This dead-reckoning error correction is what the algorithm in section 3.2, tested in the following section, aims to achieve.

A video showing the CLM algorithm constructing the map in figure 5.1b is available at: `https://www.youtube.com/watch?v=JpE5sMqOLoo`. This video shows three graphs overlaid on top of a video of the car performing the experiment.

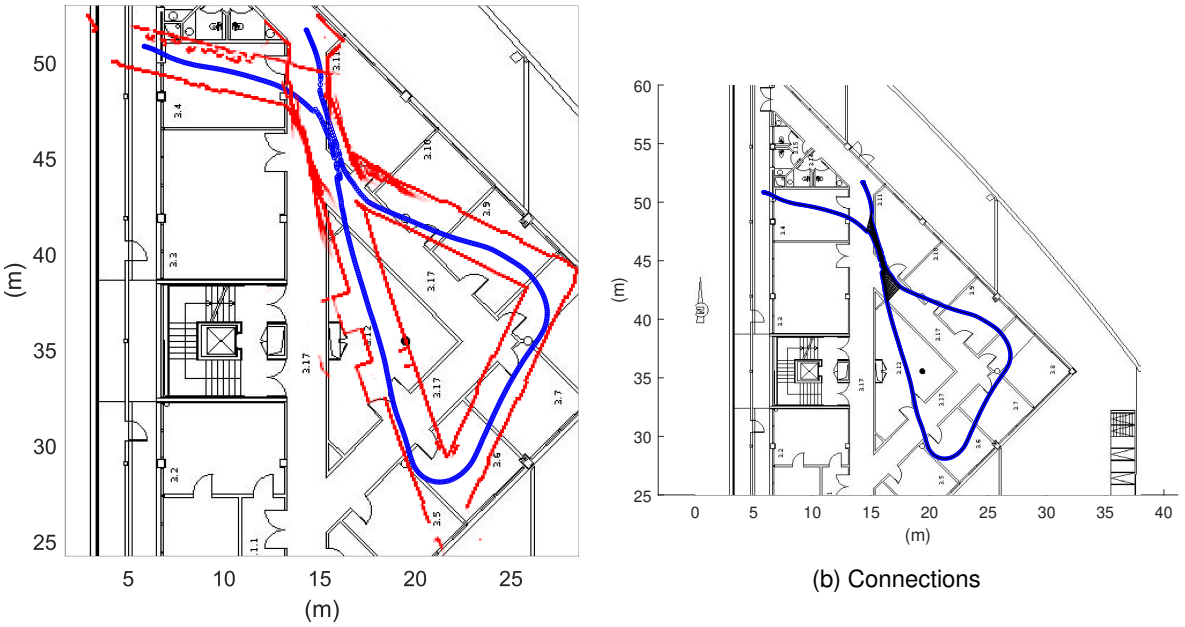
The graph in the middle left of the video shows the data from the laser scanner on a polar plot.

The graph on the bottom left shows the current estimate of the past trajectory and the map of the environment. The trajectory is shown as a blue line while the obstacles in the environment are shown in red. In the background, in black, the map of the environment collected with a measuring tape is shown.

The graph on the bottom right shows the estimates of the movement of the car from the state observer comparing them to the estimates from the registration algorithm. The previous position and orientation of the car, in the car's reference frame, is represented by the stationary small circle and arrow on the bottom. The moving small circle and arrow show the estimate of the current position and orientation, in the previous sample's reference frame. The blue ellipse and blue arrows show the trust-region of the relative pose estimate from the state observer. Although it seems like there is only one blue arrow, there are two, showing the limit to either side of the relative orientation trust interval. When the green objects turn red, the relative pose returned by the registration algorithm is outside the trust-region defined from the relative pose returned by the state observer.

5.2 Network of Relative Poses

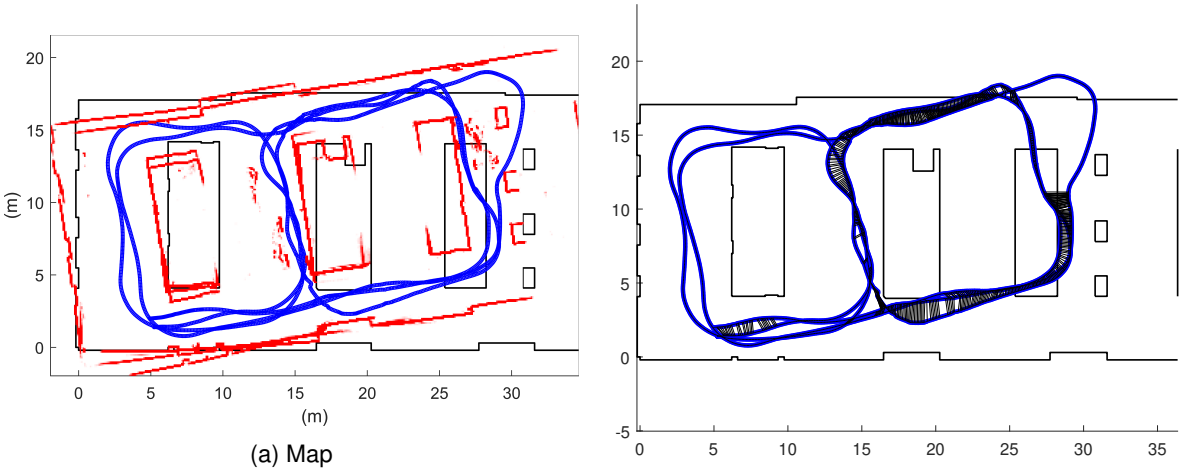
In this section, the network-based CLM algorithm described in section 3.2 is tested. Its main goal is the production of a globally consistent map, free of dead-reckoning error. The result of this algorithm, using the same data as used for the sequential map construction in figures 5.1a and 5.1b, is shown in figures 5.5a and 5.6a respectively. In figures 5.5b and 5.6b, the relative pose connections used for the network-based CLM are shown with black lines connecting the nodes.



(a) Map

(b) Connections

Figure 5.5: Network-based CLM using the data used to construct the map in figure 5.1a.



(a) Map

(b) Connections

Figure 5.6: Network-based CLM using the data used to construct the map in figure 5.1b.

The network-based CLM algorithm greatly improves the consistency of the garage map while producing large errors in the office building map. This is likely due to the same problem mentioned before where point clouds along a corridor are registered and, since all the normals point in one direction, the

relative pose along the perpendicular direction has very low accuracy.

The connections in figure 5.5b show that most of the closing loop connections in this network happen between nodes whose point clouds is of a corridor. By using the covariance matrix in 2.5.2, it is possible to ignore the relative pose along the low accuracy direction of these point clouds which would improve the network solution when there are long corridors in the environment. To do that, the characterization of the accuracy of the registration algorithm, which is shown in [22] to depend on the surface normals, would have to performed.

In the map construction of the map in figure 5.6a, there are connections between nodes whose point clouds contain surface normals pointing in different directions. These connections are what improves the consistency of the final map.

A video showing the CLM algorithm constructing the map in figure 5.6a is available at: `https://www.youtube.com/watch?v=D-ifL6qphI8`. The video is similar to the one mentioned in section 5.1 and an explanation of its graphs can be found in the same section.

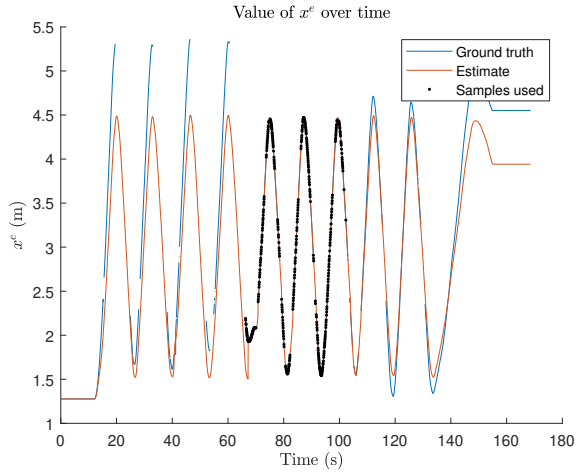
5.3 Positioning System

The CLM algorithm described in section 2.5.3 was used to estimate the trajectory of the car in real-time. The positioning system and state observer used are the ones described in section 4.2 and 4.6.1 respectively. Because the positioning system often fails to detect the car, there is no complete ground truth trajectory. Regardless, the estimated trajectory of the car is compared with the partial ground truth data available. Since the state observer from section 4.6.1 works well, the CLM algorithm takes time to drift away from the real trajectory. Because of this, a manually controlled switch used to further reduce the positioning data available for the CLM algorithm was added to the real-time Simulink model. Right after initializing the Simulink model, with the car still standing still, a few data points from the positioning system are sent to the CLM algorithm so that its initial pose is the same as the one returned by the positioning system.

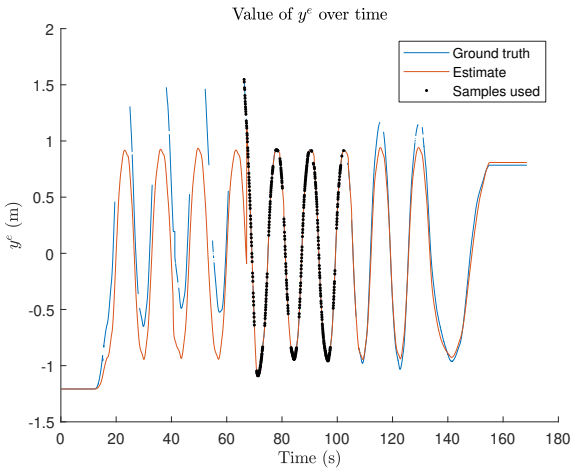
An experiment was performed where the car was driven using the controller mentioned in section 4.7.4 with a discretized circular trajectory. Figures 5.7a, 5.7b and 5.7c show the value of the absolute pose parameters over time and figure 5.7d shows the resulting trajectory.

In these figures, both the trajectory estimated by the CLM algorithm and the ground truth data available can be seen. The positioning system samples used by the CLM algorithm are represented by the black dots. It can be seen that, as expected, when there is no positioning data available, the pose estimates drift away from the real values. As soon as data from the positioning system is available it quickly corrects its estimate of the pose. The controller also performs well, driving the car in a circle according to the CLM algorithm's estimate of the car's pose.

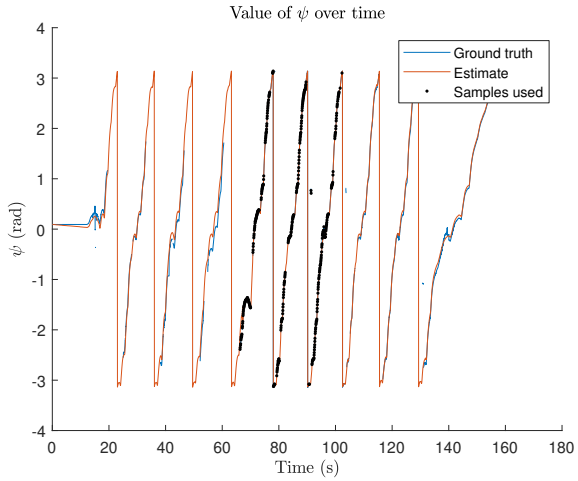
Another experiment was performed with the same controller driving the car along an elliptical trajectory. The same behavior is observed with the CLM algorithm correctly estimating the pose of the car when there is data available from the positioning system and drifting away when there is none. Both the center of the ellipse and the orientation of its axes drift over time. Near the end of this experiment, the



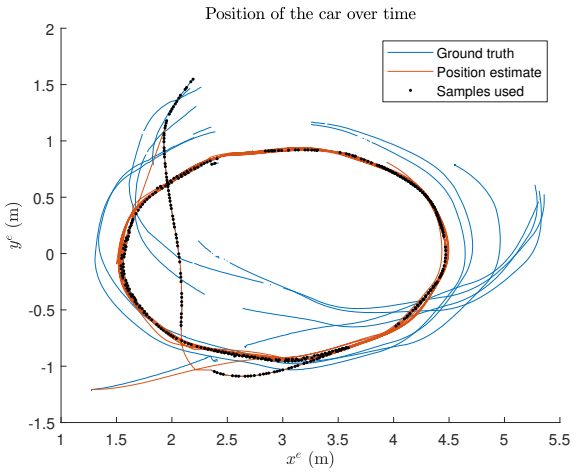
(a) Value of x^e over time.



(b) Value of y^e over time.



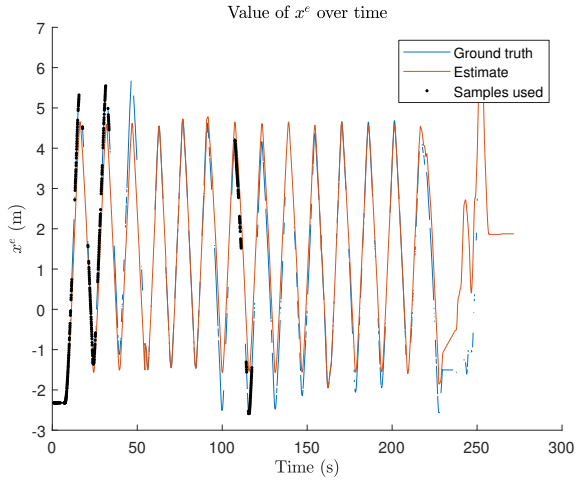
(c) Value of ψ over time.



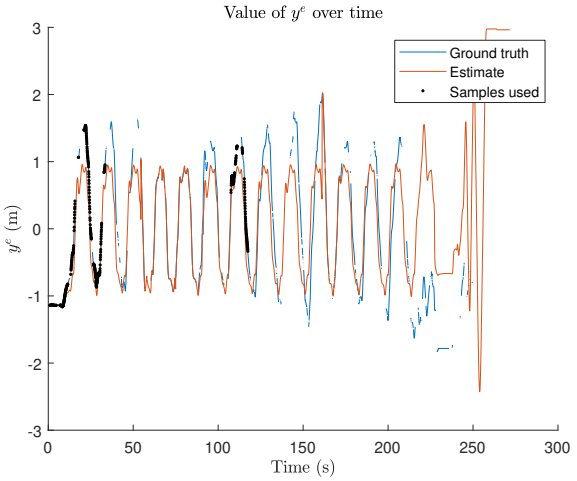
(d) Final trajectory.

Figure 5.7: Results from the first experiment performed.

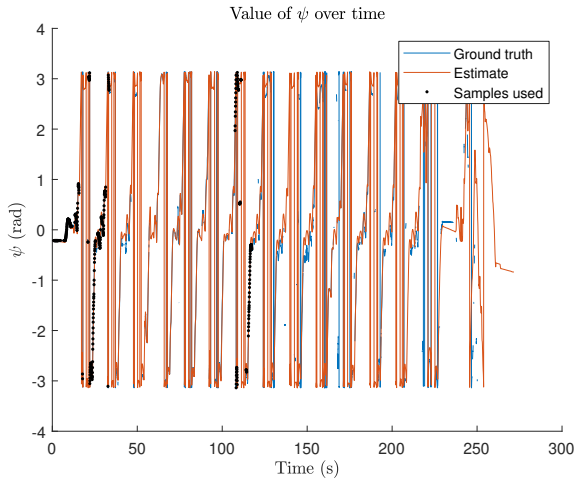
car was manually moved to a different place, while having no access to positioning data, to see how the CLM algorithm would react. The algorithm got lost and, because of this, the controller crashed the car into the surrounding environment and the experiment was ended. As before, figures 5.8a, 5.8b and 5.8c show the values of the absolute pose parameters over time and figure 5.8d shows the final trajectory.



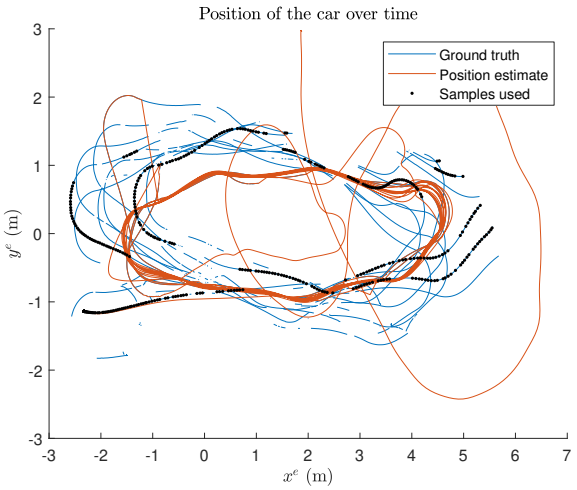
(a) Value of x^e over time.



(b) Value of y^e over time.



(c) Value of ψ over time.



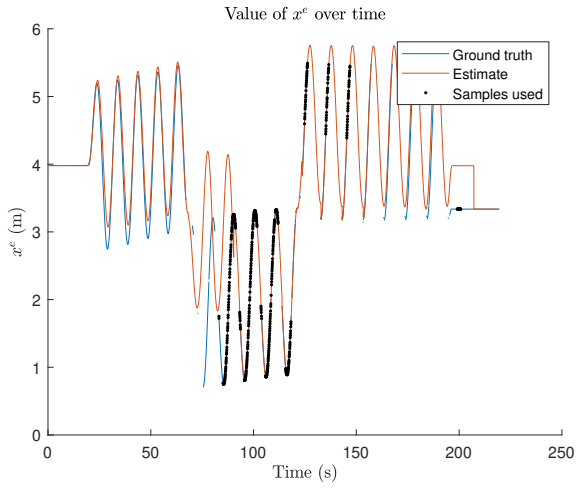
(d) Final trajectory.

Figure 5.8: Results from the second experiment performed. The car's pose is changed around 220 s.

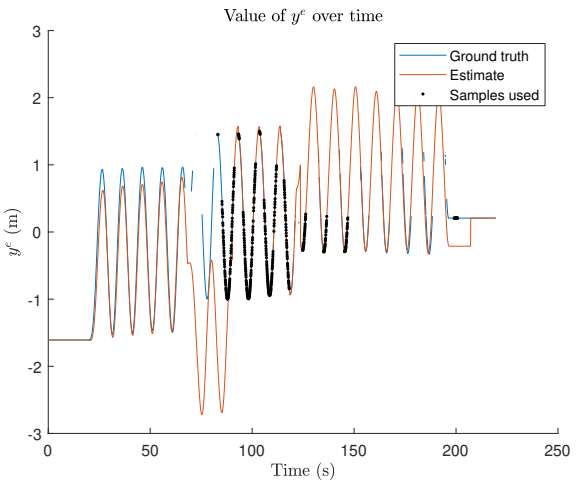
The final experiment was performed to test what happens if the car is suddenly moved, as tried in the second experiment. To do that, the car was run with fixed steering and driving inputs so that it would run around in circles. Twice during the experiment, the car was manually moved to a different location and orientation. As previously, figures 5.9a, 5.9b and 5.9c show the absolute pose values over time and figure 5.9d shows the final trajectory.

The car starts making circles and drifting away from the ground truth, this is visible in the circle centered around $[4, 0]^T$. The car is then picked up and quickly rotated $180°$. This results in the car making circles around $[2, 0]^T$ while the CLM algorithm thinks it is making circles around $[3, 2]^T$. The second time the car is picked up it is able to predict the changed pose even before having access to the positioning data. The fact that it was able to keep track of its pose could be due to the slower transition.

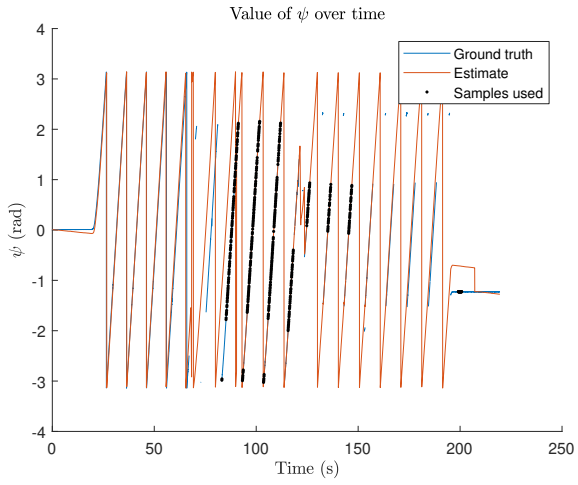
In conclusion, the algorithm provides good results, producing an estimate of the trajectory without



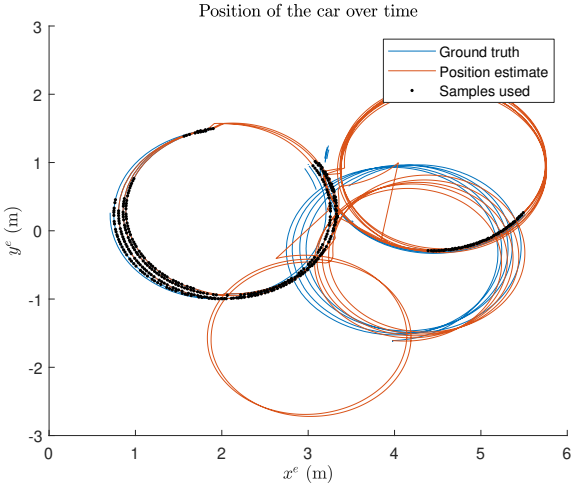
(a) Value of x^e over time.



(b) Value of y^e over time.



(c) Value of ψ over time.



(d) Final trajectory.

Figure 5.9: Results from the third experiment performed. The car's pose is changed around 70 s and 125 s.

using positioning data, while being able to correct this estimate when there is data available. The reliability of the estimate of the trajectory when there is no positioning data available is due to the Kalman filter developed in section 4.6.1 which relies, in large part, on the quality of the data from the sensors, especially the gyroscope and encoder.

A fourth experiment was conducted and a video, available at `https://www.youtube.com/watch?v=87C1F8h0Q3U`, was made from it. The experiment consists in the controller driving the car in an elliptical trajectory using the position estimate from the CLM algorithm. As the two previous videos, it shows two graphs overlaid on top of a video of the car.

The graph on left shows the trajectory of the car, in the earth's reference frame, during the last 15 seconds. The blue line shows the ground truth data from the positioning system while the orange line is the estimate of the trajectory made by the CLM algorithm. The gaps in the ground truth data from the positioning system are due to the failure of the latter in returning a position. The black dots show the positioning data used by the CLM algorithm. The graph on the right shows the behavior of the controller. The stationary circle in the bottom represents the car in the car's reference frame while the moving one represents the target position of the controller, in the car's reference frame.

Chapter 6

Conclusions

6.1 Achievements

The goal of this thesis was to develop a framework to support autonomous navigation and cooperation among robots with ground truth validation, paving the way for the development of cooperative or collaborative strategies among robots for cooperative mapping, formations and collaboration. For one robot, three different Concurrent Localization and Mapping (CLM) algorithms were implemented.

Three registration algorithms were studied and implemented: NDT, ICP, and NICP. A few modifications to the NICP registration algorithm and the network-based CLM algorithms were performed. The three registration algorithms studied were compared and NICP was chosen as the most appropriate since it makes use of the surface normals which, not only improves accuracy but can be used to estimate the accuracy of the registration. Besides, although it is considerably slower than ICP, it is much faster than NDT and, more importantly, fast enough for the hardware used.

Three CLM algorithms were also studied and implemented: sequential, network-based and positioning system based. The first two algorithms make use of the registration algorithms to estimate the trajectory of the car while the last one uses a positioning system to correct the dead-reckoning error. Some simulations of the first two algorithms were performed in an ideal environment, that is, without long corridors, and the map obtained was similar to the one in the original virtual environment.

The architecture was assembled and the car was equipped with an encoder that measures the vehicle's speed, an Inertial Measurement Unit (IMU) that measures the linear accelerations and angular velocities and a laser scanner that produces point clouds of the environment. A positioning system was also set up that was used both for deriving a model of the car and for use in one of the CLM algorithms. The sensors used were also characterized and a state observer was built which predicts the trajectory of the car with great accuracy.

Multiple pieces of software were written including low-level software that interacts with the car's hardware and MATLAB functions that perform CLM. Five Simulink blocks were also designed that can be used to interface with the car's hardware in a Simulink model.

Each of the CLM algorithms implemented was tested either online or using data collected from real-

world experiments. The sequential CLM algorithm produces a map of the environment but with the expected dead-reckoning error. The network-based CLM algorithm produces a consistent map, free of dead-reckoning error, but it is sensitive to registration errors which happen often when the environment has long corridors. The positioning system CLM algorithm works well and is able to estimate the trajectory of the car with the dead-reckoning error being corrected when data is received from the positioning system.

6.2 Future Work

In the future, work towards improving the network-based CLM algorithm, discussed in section 2.5.2, can be performed by developing an algorithm that estimates the covariance matrix of the registration algorithm such as the one in [22]. This can also be done by using the registration algorithm on point clouds produced in a virtual environment and characterizing the error. The error estimation should help to minimize the registration errors that propagate into the network solution.

A SLAM solution or a controller that uses the fleet of cars, all at the same time, can also be developed using the Simulink software suite.

Bibliography

[1] A. Guéziec, P. Kazanzides, B. Williamson, and R. H. Taylor. Anatomy-based registration of ct-scan and intraoperative x-ray images for guiding a surgical robot. *IEEE Transactions on Medical Imaging*, 17(5):715–728, 1998.

[2] C. Dorai, G. Wang, A. K. Jain, and C. Mercer. Registration and integration of multiple object views for 3d model construction. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1): 83–89, 1998.

[3] P. Lamon, S. Kolski, and R. Siegwart. The smartter-a vehicle for fully autonomous navigation and mapping in outdoor environments. In *Proceedings of CLAWAR*, 2006.

[4] F. Tâche, F. Pomerleau, G. Caprari, R. Siegwart, M. Bosse, and R. Moser. Three-dimensional localization for the magnebike inspection robot. *Journal of Field Robotics*, 28(2):180–203, 2011.

[5] S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5(3-4):253–271, 1998.

[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[7] D. F. Wolf and G. S. Sukhatme. Mobile robot simultaneous localization and mapping in dynamic environments. *Autonomous Robots*, 19(1):53–65, 2005.

[8] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[9] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[10] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[11] P. Biber and W. Straßer. The normal distributions transform: A new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 3, pages 2743–2748. IEEE, 2003.

[12] F. Pomerleau, F. Colas, and R. Siegwart. A review of point cloud registration algorithms for mobile robotics. 2015.

[13] P. Bergström and O. Edlund. Robust registration of point sets using iteratively reweighted least squares. *Computational Optimization and Applications*, 58(3):543–561, 2014.

[14] J. Serafin and G. Grisetti. Nicp: Dense normal based point cloud registration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 742–749. IEEE, 2015.

[15] M. Magnusson. *The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection*. PhD thesis, Örebro universitet, 2009.

[16] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5):698–700, 1987.

[17] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2684–2689. IEEE, 2012.

[18] F. Porikli and O. Tuzel. Fast construction of covariance matrices for arbitrary size image windows. In *2006 International Conference on Image Processing*, pages 1581–1584. IEEE, 2006.

[19] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[20] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, and J. Hertzberg. Globally consistent 3d mapping with scan matching. *Robotics and Autonomous Systems*, 56(2):130–142, 2008.

[21] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Point feature extraction on 3d range scans taking into account object boundaries. In *2011 IEEE International Conference on Robotics and Automation*, pages 2601–2608. IEEE, 2011.

[22] B.-U. Lee, C.-M. Kim, and R.-H. Park. An orientation reliability matrix for the iterative closest point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1205–1208, 2000.

[23] Latrax® desert prerunner: 1/18-scale 4wd electric truck — latrax. `https://latrax.com/products/prerunner`. Accessed: 2019-12-08.

[24] A. M. P. Antunes. Sideslip estimation of formula student prototype through gps/ins fusion. Master's thesis, Instituto Superior Técnico, 2017.

[25] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015.

[26] Hokuyo-usa :: Urg-04lx-ug01. https://www.hokuyo-usa.com/products/scanning-laser-rangefinders/urg-04lx-ug01. Accessed: 2019-12-08.

[27] Rotary encoder : How to use the keys ky-040 encoder on the arduino. https://www.best-microcontroller-projects.com/rotary-encoder.html. Accessed: 2019-11-29.