A Framework for Mobile Robots Localization and Mapping

João Castro
joao.de.castro@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

June 2020

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

1. Introduction

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. 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.

2. Background

Most CLM algorithms require the use of a registration algorithm. In this thesis, the registration algorithm used is the Normal Iterative Closest Point (NICP), introduced in [10] and described in section 2.2. The map construction algorithm used is introduced in [11] and described in section 2.3.

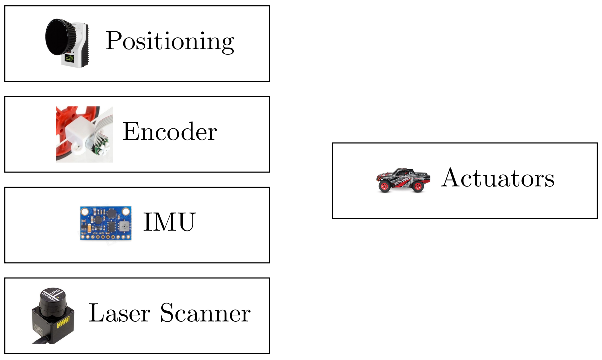


Figure 1: Diagram of the architecture implemented.

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:

$$p^m = Rp^d + T \tag{1}$$

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.

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 and 3 respectively.

$$R = \begin{bmatrix} \cos t_\theta & -\sin t_\theta \\ \sin t_\theta & \cos t_\theta \end{bmatrix} \tag{2}$$

$$T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{3}$$

The registration algorithms estimate the following vector of parameters:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_\theta \end{bmatrix} \tag{4}$$

2.2. Normal Iterative Closest Point

Normal Iterative Closest Point (NICP), introduced in [10] 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 [12] and [13].

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 a different surface results in very large errors.

The covariance matrix is calculated over the points inside the neighborhood radius. It is calculated using equation 5 in order to make it possible to use integral images.

$$\begin{aligned} C_i = \frac{1}{n}\Bigg(& \begin{bmatrix} \sum p_x p_x & \sum p_x p_y \\ \sum p_x p_y & \sum p_y p_y \end{bmatrix} - \\ & \frac{1}{n} \begin{bmatrix} \sum p_x \\ \sum p_y \end{bmatrix} \begin{bmatrix} \sum p_x & \sum p_y \end{bmatrix} \Bigg) \end{aligned} \tag{5}$$

Here, n is the number of points and p_x and p_y are the x and y coordinates of each point. These coordinates are summed over the points inside the neighborhood radius of the point i.

The eigendecomposition of matrix C_i is used to extract the surface normal, the surface curvature and the coordinate transformation matrix, which 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.

The surface normal is the eigenvector corresponding to the eigenvalue with the smallest absolute value. The surface curvature is given by: $\gamma_c = \frac{|D_{min}|}{|D_{min}|+|D_{max}|}$ where D_{min} and D_{max} are the smallest and largest eigenvalue respectively. 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.

Normal Iterative Closest Point (NICP), like Iterative Closest Point (ICP) described in detail in [14], is an iterative algorithm with 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.

The association method used consists in finding the model point that has the smallest Euclidean distance to the projection of each data point. The projection of the data points is performed with equation 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. In order to decrease the computational cost, a Delaunay triangulation of the model point cloud is performed in the initialization of the registration algorithm and is then used to find the aforementioned closest model point to every projected data point.

The filtering of outliers is performed by adding a weight w_i to each pair of points, function of the Euclidean distance between the points of the pair. The weight functions implemented return a value between 0 and 1 and are described in [15] where they are discussed in depth.

The rotation matrix and the translation vector are found by minimizing the following objective function:

$$W = \sum_{i=0}^{n} w_i e_i(T)^T \Omega_i e_i(T) \tag{6}$$

where:

$$e_i = \begin{bmatrix} (Rp_i + T) - q_i \\ \left(\operatorname{atan} \frac{\vec{n}^d_{i,y}}{\vec{n}^d_{i,x}} + t_\theta\right) - \operatorname{atan} \frac{\vec{n}^m_{i,y}}{\vec{n}^m_{i,x}} \end{bmatrix} \tag{7}$$

The Ω_i matrix is subdivided into two matrices:

$$\Omega_i = \begin{bmatrix} \Omega^s_i & 0 \\ 0 & \Omega^n_i \end{bmatrix} \tag{8}$$

Here, Ω^s_i 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^n_i is a scalar that determines the importance of aligning the normals. To calculate Ω^s_i, the eigenvectors matrix V of the model point, together with a diagonal scaling matrix S, is used in the following expression:

$$\Omega^s_i = V_i S_i V_i^T \tag{9}$$

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. This is determined from the matrix D of the eigendecomposition. 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^n_i is set to one.

The solution to this minimization problem was found using the Levenberg–Marquardt algorithm.

2.3. Simultaneous Localization And Mapping

Simultaneous localization and mapping 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 premade 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 [16]. 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 [11] for the 2D case and in [17] for the 3D case.

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.

The CLM algorithm used, based on a network of relative poses, is described in [11]. It is an optimization algorithm that minimizes the following criterion:

$$W = \sum_{\{i,j\}\in P} \left(t_i^a - t_j^a - t_{ij}^r\right)^T C_{ij}^{-1} \left(t_i^a - t_j^a - t_{ij}^r\right) \tag{10}$$

Here, t_i^a is a vector defining the absolute pose of node i and t_{ij}^r is the measured relative pose between the nodes i and j, that is, the measured value of $t_i^a - t_j^a$. 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.

The CLM algorithm used, based on data from a positioning system, is composed of a Kalman filter with the following equations:

$$x_{k|k-1} = F x_{k-1|k-1} + G u_{k-1} \tag{11}$$
$$P_{k|k-1} = F P_{k-1|k-1} F^T + Q \tag{12}$$
$$S_k = C P_{k|k-1} C^T + R \tag{13}$$
$$L_k = P_{k|k-1} C^T S_k^{-1} \tag{14}$$
$$x_{k|k} = x_{k|k-1} + L_k \left(\hat{y}_k - C x_{k|k-1}\right) \tag{15}$$
$$P_{k|k} = \left(I - L_k C\right) P_{k|k-1} \tag{16}$$

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 which 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. The inputs vector contains the velocities in earth's reference frame which come from the body's state observer. The outputs vector contains the position and orientation data from the positioning system.

3. Implementation

In this section, the five parts of the architecture shown in figure 1 are described. This is followed by a description of the software implemented.

3.1. Actuators

The vehicle used in this thesis is a Desert Prerunner 1/18 Scale 4WD Truck from LaTrax [18]. It has four wheels, all-wheel drive and front steering. The equations for the model of the car, based on a bicycle model, are in equations 17 to 21 illustrated by the diagram in figure 2.

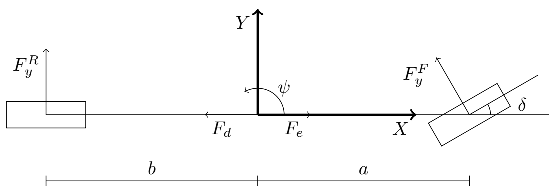


Figure 2: Diagram of the car.

$$\ddot{x}^b = \dot{y}^b \dot{\psi} + \frac{1}{m}\left(F_e - F_d - F_y^F \sin\delta\right) \tag{17}$$
$$\ddot{y}^b = -\dot{x}^b \dot{\psi} + \frac{1}{m}\left(F_y^R + F_y^F \cos\delta\right) \tag{18}$$
$$\ddot{\psi} = \frac{1}{I}\left(a F_y^F \cos\delta - b F_y^R\right) \tag{19}$$
$$\dot{x}^e = \dot{x}^b \cos\psi - \dot{y}^b \sin\psi \tag{20}$$
$$\dot{y}^e = \dot{y}^b \cos\psi + \dot{x}^b \sin\psi \tag{21}$$

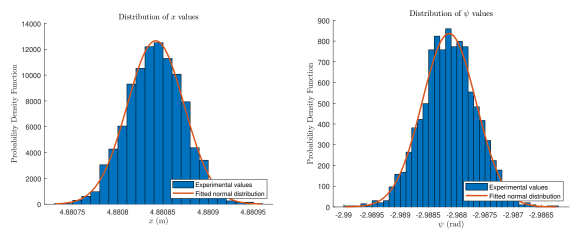
$\ddot{x}^b$ and $\ddot{y}^b$ are the linear accelerations in the car's reference frame, $\ddot{\psi}$ is the angular acceleration of the car around the vertical axis and $\dot{x}^e$ and $\dot{y}^e$ are the car's velocities in the earth's frame of reference.

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 back 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.

3.2. Positioning System

The positioning system can measure, among other things, the position and orientation 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 of x and ψ, returned by the positioning system, are shown, with a fitted normal distribution, in figure 3. The values of y follow a distribution similar to the one that the values of x follow and, as such, are not included for brevity.

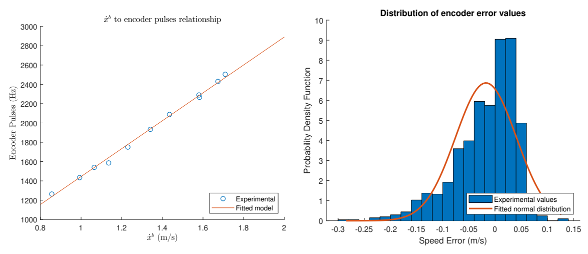


(a) x (b) ψ

Figure 3: Values returned by the positioning system with the car standing still.

3.3. Encoder

An encoder was mounted into the car, measuring the speed of rotation of the motor. The relationship between the car's velocity and the number of impulses per second can be seen in figure 4a. The distribution of encoder error values is shown in figure 4b.

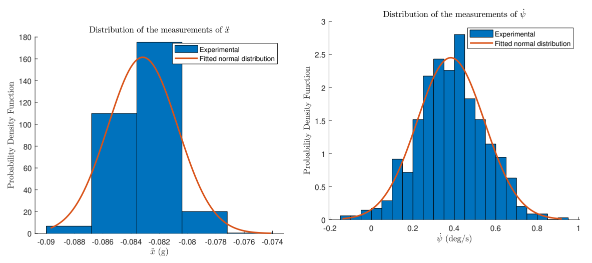


(a) Relationship between encoder and velocity. (b) Encoder error distribution.

Figure 4: Encoder characterization.

3.4. Inertial Measurement Unit

An Inertial Measurement Unit (IMU) was installed onto the car which measures 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 5 where the values of $\ddot{y}^b$ are not included due to their similarity to the values of $\ddot{x}^b$. The small number of bins in the first histogram is due to the limited resolution of the accelerometer.



(a) Acceleration along the x axis, $\ddot{x}$, in g's. (b) Angular velocity around the vertical axis, $\dot{\psi}$, in degrees per second.

Figure 5: Histogram of IMU measurements.

3.5. Laser Scanner

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 [19]. 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.

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 with a standard deviation constant across the whole depth range.

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 22 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.

$$r(\theta) = \frac{r_{min}}{\cos(\theta - \beta)} \tag{22}$$

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. To estimate how the bias varies with depth these data points are put into bins and the values in each bin are fitted to a normal distribution. The standard deviation of each bin as a function of the mean value is plotted in figure 6. It shows that the standard deviation of the bias is proportional to the point's depth.

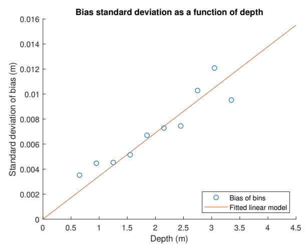


Figure 6: Bias standard deviation as a function of depth r.

3.6. State Observer

The state observer constructed is a Kalman filter whose state variables are then used to estimate the relative pose between two consecutive samples, that is, t_x^m, t_y^m and t_θ^m. The general equation of the Kalman filter used is the following:

$$\begin{aligned} x_{k|k-1} =& F x_{k-1|k-1} + G u_{k-1} + \\ & L_p \left(y_{k-1} - C x_{k-1|k-1} - D u_{k-1} \right) \end{aligned} \tag{23}$$

$$x_{k|k} = x_{k|k-1} + L_c \left(y_k - C x_{k|k-1} - D u_k \right) \tag{24}$$

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 3.2 to 3.5. 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 20 and 21 are integrated assuming all three state variables change linearly with time. 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:

$$B_\circ = \left(\frac{t_x^r - t_x^m}{C_{ex}} \right)^2 + \left(\frac{t_y^r - t_y^m}{C_{ey}} \right)^2 <= 1 \tag{25}$$

$$B_\theta = \left| t_\theta^r - t_\theta^m \right| <= C_{e\theta} \tag{26}$$

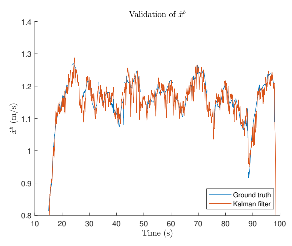
Here, the superscripts r and m refer to the parameters estimated using the registration algorithm and the model of the car respectively. C_{ex}, C_{ey} and $C_{e\theta}$ are scalar values tuned manually.

To validate the Kalman filter, 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 7, these results are compared with the estimates from both the Kalman filter and the simplified observer. The forward velocity $\dot{x}^b$ and angular velocity $\dot{\psi}$ have acceptable errors while the sideways velocity has a relatively large error but it is more easily corrected by the registration algorithm since the surfaces are normal to this direction.

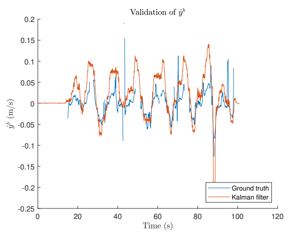
3.7. Software Implementation

Four pieces of software were written:

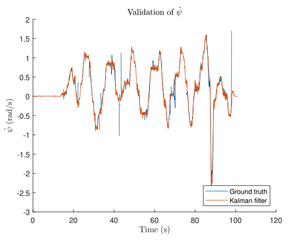
- Written in C++ and runs on the car's onboard Raspberry Pi.
- Written in Python and runs on a laptop connected to the Raspberry Pi Wi-Fi network.
- Written in MATLAB and performs offline CLM.
- Five Simulink blocks that interact with the car's sensors and actuators.



(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 7: Validation of the state observer.

The first 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 a freely available software. Other computers then connect to this Wi-Fi network and send signals and receive data from the car's C++ software.

The Python software written 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 C++ software. The other three modules interact with each of the three C++ Controllers. The first controller consists in directly controlling the car using the laptop's arrow keys. The second one is a proportional controller that uses the ψ estimated by the State Observer module to maintain a ψ reference that can be changed by the laptop's arrow keys. 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.

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 the state observer described in section 3.6. Another MATLAB class was written that implements the first two CLM algorithms described in section 2.3. A script was written that uses these pieces of software to perform offline CLM using the data logged by the Python 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 follow 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 code C++ code: the S-functions simply call the functions of the C++ modules.

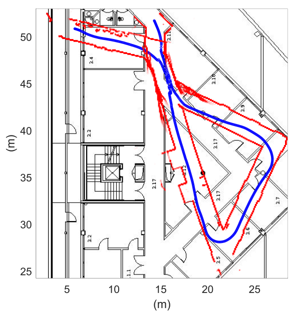
The positioning based CLM algorithm described in section 2.3 is implemented as a Simulink model that runs in real-time which uses the previously mentioned S-function blocks to interact with the car's sensors and actuators. A controller that makes the car follow a trajectory defined by a series of points was also developed.

4. Results

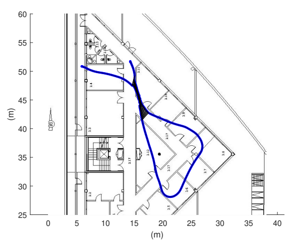
The results of the network-based and the positioning system based CLM algorithms, described in section 2.3, are shown in section 4.1 and 4.2 respectively.

4.1. Network of Relative Poses

In this section, the network-based CLM algorithm described in section 2.3 is tested. The result of this algorithm is shown in figures 8a and 9a. In figures 8b and 9b, the relative poses connections used for the network-based CLM, are shown with black lines connecting the nodes. A video showing the CLM algorithm constructing the map in figure 9a is available at: `https://www.youtube.com/watch?v=D-ifL6qphI8`.



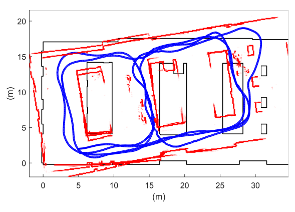
(a) Map



(b) Connections

Figure 8: Network-based CLM inside an office building.

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.



(a) Map

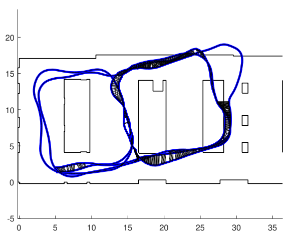

(b) Connections

Figure 9: Network-based CLM inside a residential building's garage.

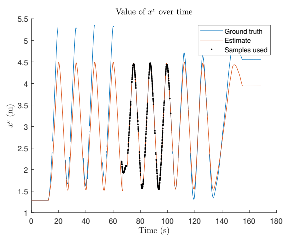
The environment in figure 8 is an office building in an university and its plant was obtained from the university's website. The environment in figure 9 is a garage in the basement of a residential building and its plant was constructed from measurements performed using a measuring tape.

The network-based CLM algorithm produces a consistent map of the garage while producing large errors in the office building map. This is likely due to the low accuracy of the registration of point clouds along a corridor since all the normals point in one direction. The connections in figure 8b show that most of the closing loop connections in this network happen between nodes whose point clouds are of a corridor. By using the covariance matrix in 2.3, 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 [20] to depend on the surface normals, would have to performed.

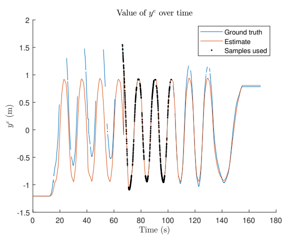
4.2. Positioning System

The CLM algorithm based on positioning data, described in section 2.3, 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 3.2 and 3.6 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 3.6 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.

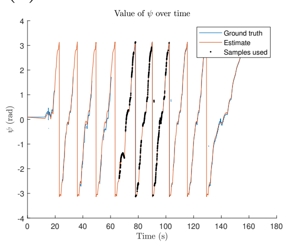
An experiment was performed where the car was driven using the controller mentioned in section 3.7 with a discretized circular trajectory. Figures 10a, 10b and 10c show the value of the absolute pose parameters over time and figure 10d shows the resulting trajectory.



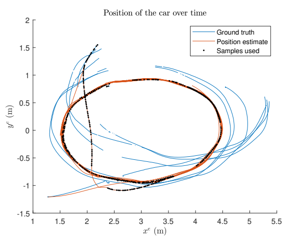
(a) Value of x^e over time.



(b) Value of y^e over time.



(c) Value of ψ over time.

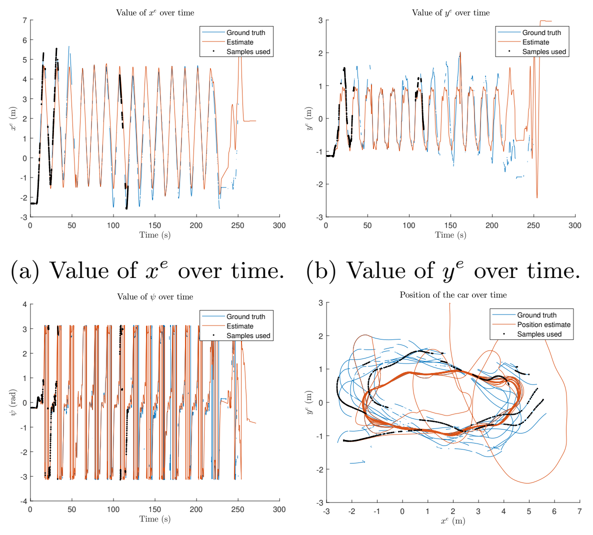


(d) Final trajectory.

Figure 10: Results from the first experiment performed.

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 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 11a, 11b and 11c show the values of the absolute pose parameters over time and figure 11d 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 11: Results from the second experiment performed. The car's pose is changed around 220 s.

Another experiment was conducted and a video, available at `https://www.youtube.com/watch?v=87C1F8h0Q3U`, was made from it. In conclusion, the algorithm provides good results, producing an estimate of the trajectory without 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 3.6 which relies, in large part, on the quality of the data from the sensors, especially the gyroscope and encoder.

5. Conclusions

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.

In the future, work towards improving the network-based CLM algorithm, discussed in section 2.3, can be performed by developing an algorithm that estimates the covariance matrix of the registration algorithm such as the one in [20]. 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.

Acknowledgements

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.

References

[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] 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.

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

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

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

[20] 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.