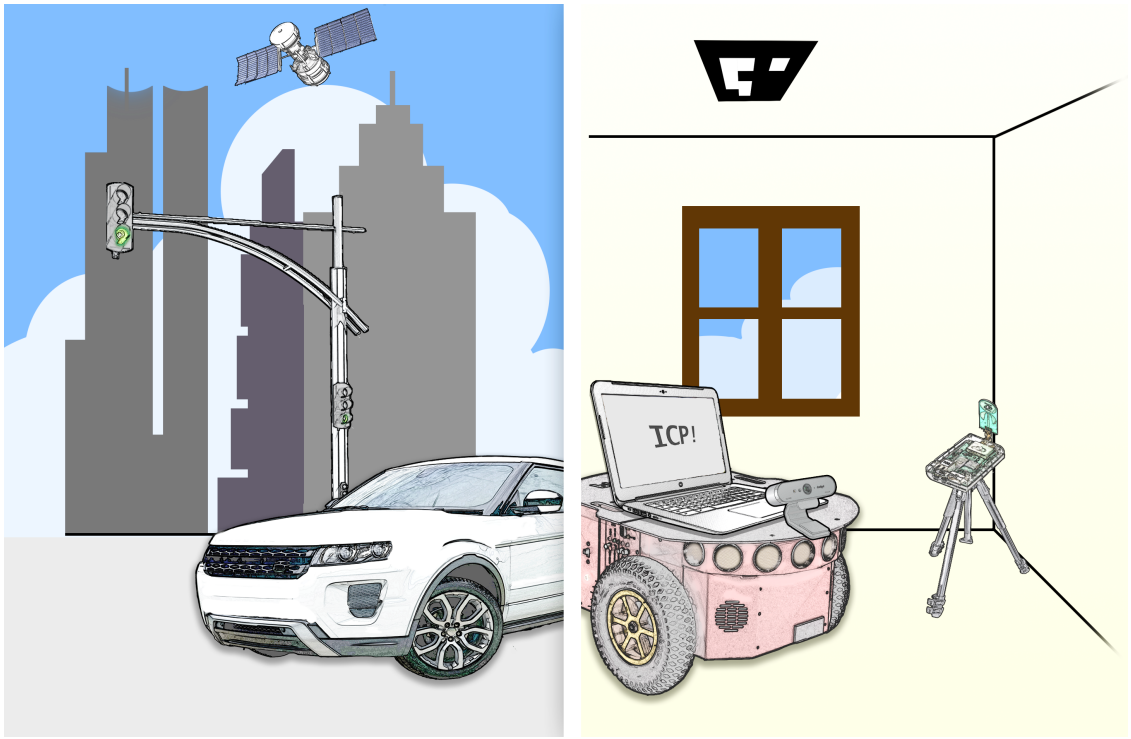
CHALMERS
UNIVERSITY OF TECHNOLOGY



Implicit Cooperative Positioning (ICP)

Implementation and Evaluation of ICP on a Semi-Autonomous System

Master's thesis in Systems, Control and Mechatronics

TOMASZ PROCZKOWSKI and NITHIN SYRIAC KURIEN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017

Implicit Cooperative Positioning (ICP)

Implementation and Evaluation of ICP on a Semi-Autonomous System

TOMASZ PROCZKOWSKI
NITHIN SYRIAC KURIEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
MPSYS
Chalmers University of Technology
Gothenburg, Sweden 2017

Implicit Cooperative Positioning (ICP)
Implementation and Evaluation of ICP on a Semi-Autonomous System

Supervisor: Markus Fröhle & Christopher Lindberg, Electrical Engineering
Supervisor: Thomas Petig, Computer Science and Engineering
Examiner: Henk Wymeersch, Electrical Engineering

Master's Thesis 2017
Department of Electrical Engineering
MPSYS
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration showing the analogy between the real world scenario and the ICP implementation.

Typeset in LaTeX
Gothenburg, Sweden 2017

Abstract

Autonomous vehicles are becoming extremely popular in the automotive field and the positioning of these vehicles is very vital. Getting position measurements with very low variance from noisy measurements is very important otherwise there could be some disastrous consequences to the pedestrians and environment. Our aim is to tackle this problem by investigating the performance of Implicit Cooperative Positioning (ICP), a cooperative positioning system aimed for autonomous vehicles. According to this algorithm vehicles track objects (features) and use this information between other vehicles to position the feature better than use this information to position themselves better. ICP was implemented on a semi-autonomous system to check the functioning in a real life scenario. The various aspects of ICP including belief propagation and message passing are dealt with. Simulations were carried out to test various scenarios where the vehicle position measurements were altered with noise, later position estimate error comparisons were made to the Kalman filter. The complete implementation and integration of the robot system including the robot framework and the components involved, e.g. Kalman filtering, Linear Quadratic Regulator were done.

Our results does show the viability of such a positioning system in a real world system. Sharing information between multiple nodes in the system ensures better estimation by reduction in measurement variance. The error in position is greatly reduced for really noisy measurements. The feedback of the ICP position estimates for the control of the robot in case of noisy measurements from UWB was done and successfully implemented on the robot system. The thesis mainly handles the ICP algorithm and does not look into the communication protocols or any feature recognition and tracking algorithms. It is assumed such technologies are already in place if the ICP is to be implemented.

Keywords: ICP, Cooperative positioning, Cramer-Rao bound, Kalman filter, LQR, Pioneer-D3X

Acknowledgements

We would like to thank our supervisors Markus, Christopher and Thomas for their undue and selfless support for our thesis. They always had an opinion or solution to our questions and problems. They were always prompt with giving feedback and were truly a joy to work with. We are grateful to have such dedicated and helpful supervisors. We would like to also thank our examiner Henk Wymeersch for giving us an opportunity to work with this thesis and for his valuable feedback and comments. We would also like to thank our dear booyas (David Gardtman & Albin Casparsson) for making our thesis enjoyable, without them our thesis would have been just some chore to do. Thanks to them we had our occasional bout of laughter. We also thank them for their valuable suggestions and advice regarding our thesis and other random stuff about life. Lastly we would like to thank our dearest families, friends and beloved ones for their motivation and support without which we would have a tough time completing our thesis.

Tomasz Proczkowski & Nithin Syriac Kurien, Gothenburg, June 2017

Contents

List of Figures

1
Introduction

This chapter describes the background of problems related to positioning of autonomous vehicles, its possible and existent solutions and also the idea behind and the uniqueness of the implicit cooperative positioning. Furthermore it specifies the scope and limitations of the thesis described in this report.

1.1 Background

Currently there is a lot of interest in the field of autonomous vehicles and robots. Autonomous vehicle is the future of the automobile industry. Also the use of autonomous robots is constantly increasing in a variety of tasks e.g. warehouse logistic. In both cases one of main hurdles is the exact positioning of the agent [1]. Accurate positioning is often crucial for providing proper and safe functioning of the systems. Especially in case of autonomous vehicles, inaccurate positioning may have drastic and life threatening consequences. Thus the task of making the positioning system of autonomous vehicles accurate and reliable would be really crucial, keeping the safety of the passengers and the environment around the vehicle in mind.

For outdoor applications, thanks to the worldwide coverage and good accuracy, time of arrival (TOA) based global navigation satellite systems (GNSS) such as GPS, GLONASS, etc. are widely used for localisation [2]. For indoor purposes, where GNSS suffer form significant signal loss the Ultra Wide Band (UWB) radio beacons gained big popularity. These can be used in a Round-trip Time of Arrival (RTOA) based system, which in an energy efficient way that provides estimates with just few centimetre accuracy [3]. However, because of i.a. noise in the TOA measurements, shadowing and multiple path propagation, these systems suffer from uncertainties [4, 5, 6].

1.2 Related work

To alleviate the negative effects of the measurements with low accuracy many companies and academic institutions are constantly striving to develop new algorithms to improve the positioning. For example applying Real Time Kinematics as in [7] may increase the GNSS measurement accuracy down to a centimetre level, although requires high cost equipment and a reference station located at exactly known coordinates. Many cooperative localisation solutions have been developed too. These

rely on vehicles sharing their relative position to each other and use this information to correct the GNSS or UWB measurements [8, 9]. This concept may strongly improve the localisation increasing its accuracy and enlarging the coverage. Also the combination of cooperative localisation and distributed target tracking [10], cooperative self localisation and tracking (CoSLAT) has been developed [11, 12]. These use ranging to estimate the position of targets which further can help in estimation of position of the agents themselves. Although accurate, they often require radio anchors with predefined position and accurate ranging between the nodes and the targets. The CoSLAT algorithm is though using recognition of only one target per node and the ranging is not contained within any of existing and popular standards [13].

1.3 Purpose

The natural reasoning is that having more sensors on the vehicle would increase the amount of information about where the vehicle is located. One sensor that comes to mind in an autonomous vehicle would be the camera. A camera would be required to sense the environment and this could be used to see objects and position the vehicle with respect to this object. What if this could be made better by cooperating measurements between numerous cars in the vicinity. This indirectly unlocks the potential of adding more sensors to a given vehicle without installing any physical components.

The cooperative positioning system hence needs a robust algorithm which will help the vehicle with bad GNSS measurements to get better measurements. Also by cooperating with many vehicles and sensing more objects using the camera one should be able to acquire much more accurate estimates for all vehicles as each vehicles share their measurements with other vehicles. Given the advancements in communication protocols for vehicle-to-everything (V2X)[14, 15] and the current development in 5G mainly pushing for the implementation of the "internet of things(IoT)"[16], such kind of vehicle cooperation would be more practical in the near future. Our aim is to achieve such a system which will carry out the cooperative positioning and also test its viability in an actual test scenario using robots.

A new idea of so called implicit cooperative positioning emerged [13]. This algorithm is using the information from easily accessible on-board sensors, as cameras and radars, to sense the environment and to see different objects (features) and position them with respect to the vehicle. Sharing the information about each vehicles beliefs of the features' position via vehicle-to-vehicle (V2V) links would allow to cooperatively estimate the features' coordinates. Further on each vehicle could use this information to improve its own position acquired using internal sensors as GNSS receivers or UWB beacons.

1.4 Scope and limitations

In this report the implementation in an real world application and further evaluation of the ICP algorithm will be presented. Differential drive robots [17] will be used for as the nodes. Positioning will be done using RTOA based UWB system and the relative position of the features will be measured using Gulliview system [18].

The questions to be investigated are e.g.

- Can the estimation performance be improved by incorporating feature measurements?
- If yes, how much can the estimation error be reduced?
- When is it most advantageous to use ICP?
- Are there any situations when ICP might not work as expected?

Our focus will be mainly related to working on the algorithm focusing with the positioning of the vehicles given that a system for tracking and recognising features is already in place. Also the intricacies of the protocols for communication between the robots will not be looked into or followed. Therefore the algorithm will be implemented in a centralised fashion, giving each vehicle access to the information needed instantly and exactly.

2

Theory

This chapter describes the theory needed to understand the solution to the problem formulated in the previous chapter. Topics explained are i.a. the system model, trilateration, Bayesian estimators, Cramer-Rao bound and linear quadratic regulator.

2.1 System model

As described in Section 1.4, the system contains of a set of agents (let us call them vehicles), denoted $\mathcal{V}$ and a set of features, $\mathcal{F}$. Each vehicle, $i \in \mathcal{V}$, has its corresponding state vector containing the position and velocity states in x and y directions at time t, $\boldsymbol{x}_{i,t} = [p_{\mathrm{x}i,t}, p_{\mathrm{y}i,t}, v_{\mathrm{x}i,t}, v_{\mathrm{y}i,t}]^{\mathrm{T}}$, and each feature, $k \in \mathcal{F}$, its x and y position at time t as state vector $\boldsymbol{f}_{k,t} = [f_{\mathrm{x}i,t}, f_{\mathrm{y}i,t}]^{\mathrm{T}}$. The position measurement (say GNSS measurement) of a vehicle i at the time t will be denoted as $\boldsymbol{\rho}_{i,t}^{\mathrm{G}}$ and the V2F measurement between vehicle i and feature k at time t will be denoted as $\boldsymbol{\rho}_{i,k,t}^{\mathrm{V2F}}$. Time t denotes the current discrete time step, while time $t-1$ denotes the previous discrete time step. Figure 2.1 depicts a system of two vehicles and two features and the associated measurements.

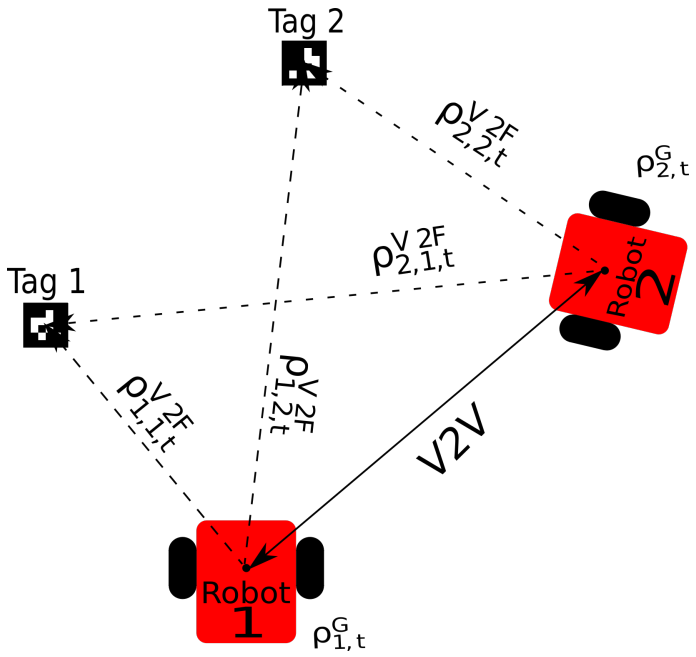


Figure 2.1: The system containing of two robots/vehicles and two features with specified GNSS-like measurements $\boldsymbol{\rho}_{i,t}^{\mathrm{G}}$, V2F measurements $\boldsymbol{\rho}_{i,t}^{\mathrm{V2F}}$ and the V2V communication link

2.1.1 Motion model

The relation between the vehicle state in time step $t-1$, $\boldsymbol{x}_{i,t-1}$, and the current one, $\boldsymbol{x}_{i,t}$, can be described by the linear motion model of the vehicle.
The state in time t is

$$\boldsymbol{x}_{i,t} = A\boldsymbol{x}_{i,t-1} + B\boldsymbol{u}_{i,t-1} + \boldsymbol{w}_{i,t-1}, \tag{2.1}$$

where the matrix A represents the system matrix, B is the input matrix and the process noise $\boldsymbol{w}_{i,t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{Q}_{i,t})$.
For the purpose of evaluation and testing of the ICP algorithm a simple constant velocity (CV) motion model has been chosen. One advantage of a CV model is the simplicity, since it is linear and does not require successive linearisation which would increase the complexity of the algorithm. It is also pretty accurate model to use with differential drive robots, where the velocity vector is unlikely to change a lot from one iteration to another. The system and input matrices are

$$A_{\mathrm{CV}} = \begin{bmatrix} 1 & 0 & T_s & 0 \\ 0 & 1 & 0 & T_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \text{and} \qquad B_{\mathrm{CV}} = \begin{bmatrix} \frac{T_s^2}{2} & 0 \\ 0 & \frac{T_s^2}{2} \\ T_s & 0 \\ 0 & T_s \end{bmatrix}, \tag{2.2}$$

where the T_s is the sampling time.

The input to the vehicle proposed by [13] is defined as the vehicles acceleration acquired either using sensors or control input to the vehicle, $\boldsymbol{u}_{i,t-1} = [a_{\mathrm{x}i,t} \;\; a_{\mathrm{y}i,t}]^{\mathrm{T}}$. Using acceleration vector as the input to the model is not necessarily going to be consistent throughout the project. Since the acceleration of the vehicle is not straight forward to access, the effect of the input to the model will be neglected. Only the process noise will be present for the prediction step, allowing the changes in the velocity states to happen. The algorithm will in this case work without any information from the vehicle, using only its own observations and prior knowledge of the vehicle state. The function of the algorithm should still be satisfactory for acquiring useful results.

Without using the input to the system an assumption is implicitly done, that the process noise $\boldsymbol{w}_{i,t}$ (which is a Gaussian noise) is responsible for all changes of the velocities in the system. For the optimal behaviour of the algorithm the value of the process noise covariance in the algorithm needs to be matched with the covariance value of the actual trajectory. According to [19, p. 60] we therefore set the process noise covariance in the algorithm as

$$\boldsymbol{Q}_{i,t} = \begin{bmatrix} \frac{\boldsymbol{q}_{\mathrm{x}}^c T_s^3}{3} & 0 & \frac{\boldsymbol{q}_{\mathrm{x}}^c T_s^2}{2} & 0 \\ 0 & \frac{\boldsymbol{q}_{\mathrm{y}}^c T_s^3}{3} & 0 & \frac{\boldsymbol{q}_{\mathrm{y}}^c T_s^2}{2} \\ \frac{\boldsymbol{q}_{\mathrm{x}}^c T_s^2}{2} & 0 & \boldsymbol{q}_{\mathrm{x}}^c T_s & 0 \\ 0 & \frac{\boldsymbol{q}_{\mathrm{y}}^c T_s^2}{2} & 0 & \boldsymbol{q}_{\mathrm{y}}^c T_s \end{bmatrix}, \tag{2.3}$$

where $\boldsymbol{q}_{\mathrm{x}}^c$ and $\boldsymbol{q}_{\mathrm{y}}^c$ are the continuous time variances of the velocity state of the actual trajectory of the vehicle in x and y directions.

This does not hold in the real process where the actual movement of the vehicles follows a specified trajectory. However for the filter to work satisfyingly the $\boldsymbol{q}^c$ values can be later tuned to best match the motion of the vehicle.

Also, the constant velocity model will be used for the positioning algorithm only. A more thorough model of the differential drive robot, the robot twist model, is derived for control purposes in Section 2.1.2.

2.1.2 Robot twist model

The differential robot's motion model needs to be modelled. To represent the dynamics of the system would be useful for the filtering and control algorithms that are to be implemented later. As the inputs to the robot are linear and angular velocity and that to with respect to the robot frame we need to create transformations from the robot frame to the world frame. Let us assume that the linear velocity is described as V and angular velocity as Z then we can make many deductions, if the robot was to make a circular motion forward as shown in the Figure 2.2

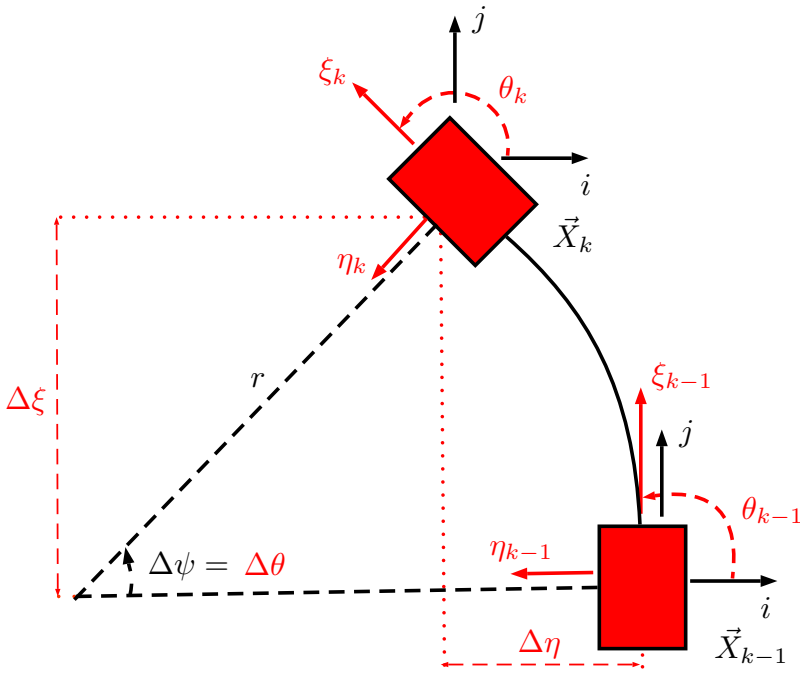


Figure 2.2: Movement of robot during a time step

Here $\xi - \eta$ represents the robot/inertial frame and $i - j$ the world frame, the colour red in Figure 2.2 is used to denote the robot coordinate space while the black denotes the world space. For a starting point of the model the model in [20] is simplified and used. If $\vec{X}_{k-1}$ is the present state then the prediction of the state of the robot after the time step can be given by

$$\vec{\hat{X}}^{\xi\eta}_{k|k-1} = \begin{bmatrix} \hat{\xi}_{k|k-1} \\ \hat{\eta}_{k|k-1} \\ \hat{\psi}_{k|k-1} \end{bmatrix} = \begin{bmatrix} \frac{V_{k-1}}{Z_{k-1}} \sin Z_{k-1}\Delta t \\ \frac{V_{k-1}}{Z_{k-1}}(1 - \cos Z_{k-1})\Delta t \\ Z_{k-1}\Delta t \end{bmatrix}. \tag{2.4}$$

The model can be transformed using trigonometric identities to give us a model in the $i - j$ coordinate space is given as

$$\vec{\hat{X}}^{ij}_{k|k-1} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\theta}_{k|k-1} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k-1|k-1} + \frac{V_{k-1}}{Z_{k-1}}[\sin(\hat{\theta}_{k-1} + Z_{k-1}\Delta t) - \sin\hat{\theta}_{k-1}] \\ \hat{y}_{k-1|k-1} + \frac{V_{k-1}}{Z_{k-1}}[\cos\hat{\theta}_{k-1} - \cos(\hat{\theta}_{k-1} + Z_{k-1}\Delta t)] \\ \hat{\theta}_{k-1} + Z_{k-1}\Delta t \end{bmatrix}. \tag{2.5}$$

We need to linearise the model as the LQR and Kalman filtering algorithms work on linearised models. The linearised model can be later used for evaluating the controller gain matrix K and this is to be done at every time step to have a good approximation of the model. For the purpose of linerisation we need to find the Jacobian matrices. To do this the equations are partially derivated with respect to the states and then the input to complete the linear differential form of

$$\Delta\hat{X}_{k|k-1} = A_{k-1}\Delta\hat{X}_{k-1|k-1} + B_{k-1}\Delta u_{k-1}, \tag{2.6}$$

where A and B are found by the following method; assume that the equations for the states in $\vec{\hat{X}}^{ij}_{k|k-1}$ in symbolic form can be represented such that

$$\vec{\hat{X}}^{ij}_{k|k-1} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\theta}_{k|k-1} \end{bmatrix} = \begin{bmatrix} f_1(\hat{x}_{k|k-1}, \hat{\theta}_{k|k-1}, V_{k-1}, Z_{k-1}) \\ f_2(\hat{y}_{k|k-1}, \hat{\theta}_{k|k-1}, V_{k-1}, Z_{k-1}) \\ f_3(\hat{\theta}_{k|k-1}, Z_{k-1}) \end{bmatrix}, \tag{2.7}$$

then the matrices A and B are found by

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial \hat{x}_{k|k-1}} & \frac{\partial f_1}{\partial \hat{y}_{k|k-1}} & \frac{\partial f_1}{\partial \hat{\theta}_{k|k-1}} \\ \frac{\partial f_2}{\partial \hat{x}_{k|k-1}} & \frac{\partial f_2}{\partial \hat{y}_{k|k-1}} & \frac{\partial f_2}{\partial \hat{\theta}_{k|k-1}} \\ \frac{\partial f_3}{\partial \hat{x}_{k|k-1}} & \frac{\partial f_3}{\partial \hat{y}_{k|k-1}} & \frac{\partial f_3}{\partial \hat{\theta}_{k|k-1}} \end{bmatrix}, B = \begin{bmatrix} \frac{\partial f_1}{\partial V_{k-1}} & \frac{\partial f_1}{\partial Z_{k-1}} \\ \frac{\partial f_2}{\partial V_{k-1}} & \frac{\partial f_2}{\partial Z_{k-1}} \\ \frac{\partial f_3}{\partial V_{k-1}} & \frac{\partial f_3}{\partial Z_{k-1}} \end{bmatrix}. \tag{2.8}$$

Using the relation (2.8) on the model described in (2.5) we get our linearised matrices as

$$A = \begin{bmatrix} 1 & 0 & \frac{V_{k-1}}{Z_{k-1}}(\cos(Z_{k-1}\Delta t + \hat{\theta}_{k|k-1}) - \cos\hat{\theta}_{k|k-1}) \\ 0 & 1 & \frac{V_{k-1}}{Z_{k-1}}(\sin(Z_{k-1}\Delta t + \hat{\theta}_{k|k-1}) - \sin\hat{\theta}_{k|k-1}) \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.9}$$

$$B = \begin{bmatrix} \frac{-\sin(Z_{k-1}\Delta t + \hat{\theta}_{k|k-1}) + \sin\hat{\theta}_{k|k-1}}{Z_{k-1}} & a \\ \frac{\cos(Z_{k-1}\Delta t + \hat{\theta}_{k|k-1}) - \cos\hat{\theta}_{k|k-1}}{Z_{k-1}} & b \\ 0 & \Delta t \end{bmatrix}, \tag{2.10}$$

where

$$a = \frac{\Delta t V_{k-1}\cos(Z_{k-1}\Delta t + \hat{\theta}_{k|k-1}) + \hat{x}_{k|k-1}}{Z_{k-1}} - \frac{Z_{k-1}\hat{x}_{k|k-1} - V_{k-1}\sin(\hat{\theta}_{k|k-1}) + V_{k-1}\sin(Z_{k-1}\Delta t\hat{\theta}_{k|k-1})}{Z^2_{k-1}},$$

$$b = \frac{\Delta t V_{k-1}\sin(Z_{k-1}\Delta t + \hat{\theta}_{k|k-1}) + \hat{y}_{k|k-1}}{Z_{k-1}} - \frac{Z_{k-1}\hat{y}_{k|k-1} + V_{k-1}\cos(\hat{\theta}_{k|k-1}) - V_{k-1}\cos(Z_{k-1}\Delta t\hat{\theta}_{k|k-1})}{Z^2_{k-1}}.$$

The motion model Sections described above is only a part of the solution to a common model based problem. The motion model is most of the times preceded by a measurement model, which shows how the states of the model can be described by the measurements obtained this is dealt with in the Section 2.1.3.

2.1.3 Measurement models

The measurement $\boldsymbol{\rho}$ can be described as a sum of a linear function dependent on some states and a zero-mean Gaussian noise $\boldsymbol{n}$ with covariance R and relates to the vehicle state $\boldsymbol{x}$ by

$$\boldsymbol{\rho} = H\boldsymbol{x} + \boldsymbol{n}, \tag{2.11}$$

where the H is the observability matrix that transforms the state of interest into the pure measurement value.

In the ICP algorithm there are two measurements to consider: The first one, $\boldsymbol{\rho}_{i,t}^{\mathrm{G}}$ is the position measurement (GNSS or alike) of the vehicle. The measurement is assumed to be normally distributed around the actual position of the vehicle as follows

$$\begin{aligned}\boldsymbol{\rho}_{i,t}^{\mathrm{G}} =& [p_{\mathrm{x}i,t}\ p_{\mathrm{y}i,t}]^{\mathrm{T}} + \boldsymbol{n}_{i,t}^{\mathrm{G}} && (2.12)\\ =& H_{\mathrm{G}}\boldsymbol{x}_{i,t} + \boldsymbol{n}_{i,t}^{\mathrm{G}}. && (2.13)\end{aligned}$$

Since the vehicle state vector contains the position of the vehicle, observability matrix is set to directly choose the first two values of the state vector,

$$H_{\mathrm{G}} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}. \tag{2.14}$$

The covariance value of the measurement noise $R_{i,t}^{\mathrm{G}}$ is in this case set accordingly to the accuracy of the position sensor.

The second measurement is the vehicle to feature (V2F) measurement. It is given by the position difference between the feature position and the vehicle position with additive zero-mean Gaussian noise

$$\begin{aligned}\boldsymbol{\rho}_{i,k,t}^{\mathrm{V2F}} =& [f_{xi,t}\ f_{yi,t}]^{\mathrm{T}} - [p_{xi,t}\ p_{yi,t}]^{\mathrm{T}} + \boldsymbol{n}_{i,k,t}^{\mathrm{V2F}} && (2.15)\\ =& H_1\boldsymbol{f}_{k,t} + H_2\boldsymbol{x}_{i,t} + \boldsymbol{n}_{i,k,t}^{\mathrm{V2F}}, && (2.16)\end{aligned}$$

where

$$H_1 = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}, \qquad H_2 = \begin{bmatrix}-1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\end{bmatrix}, \tag{2.17}$$

and

$$\boldsymbol{n}_{i,k,t}^{\mathrm{V2F}} \sim \mathcal{N}(0, R_{i,k,t}^{\mathrm{V2F}}). \tag{2.18}$$

2.1.4 Homogeneous transformation matrix

The robot, even though is one unified object, has many components on it like the camera and the UWB beacon. Each components on the robot has it's own location with respect to the location of the robot itself. When the robot moves the components on the robot are therefore displaced with regards to the world coordinate system based on the robot's orientation and displacement. A transformation is needed to convert the location of each of the robot's components from it's own coordinate system to the world-coordinate system. The Figure 2.3 shows the robot at point $(0, 0)$ being displaced by distance X_t in the X-Axis and Y_t in the Y-Axis with a rotation of Θ.

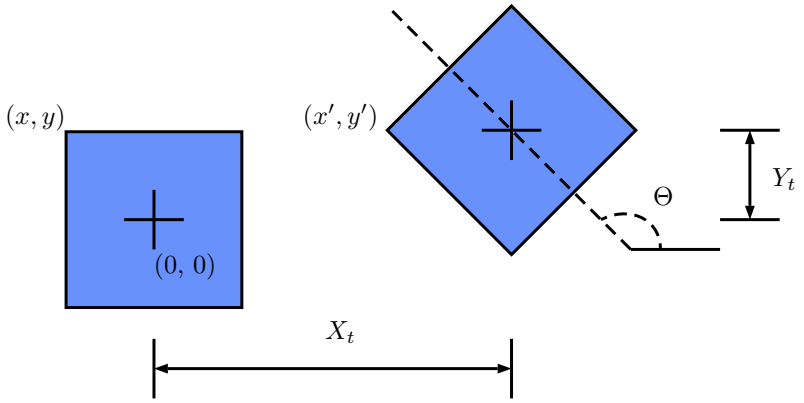


Figure 2.3: Transformation of point (x, y) to (x', y') after robot translation and rotation

Assume that the top-left corner of the robot is (x, y) and after the translation and rotation the corner is located at (x', y'). We need to find the new location of the corner given the translation and rotation of the robot which currently is located at (x_t, y_t). Using simple geometric computation we can see that new location of the corner is at

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta + X_t \\ x\sin\theta + y\cos\theta + Y_t \end{bmatrix}, \tag{2.19}$$

this can be rearranged and written in matrix form as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{2.20}$$

where

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & x_t \\ \sin\theta & \cos\theta & y_t \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.21}$$

The matrix T is called the homogeneous transformation matrix which is very commonly used to represent such transformations in many robotic applications. More about this transformation and other type of transformations can be read about in [21].

2.2 Trilateration

Trilateration is the method of finding the absolute or relative location/position of a point using distance measurements. Trilateration for positioning using a positioning system is carried out in three steps as can be seen in Figure 2.4.

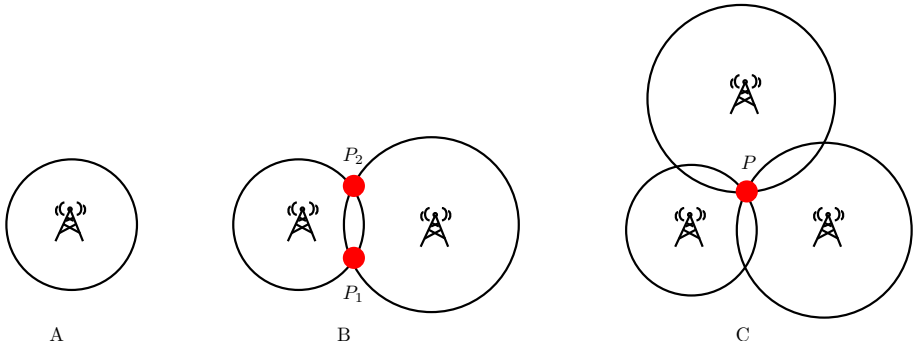


Figure 2.4: Trilateration position uncertainty in 2-D with A) One transceiver, B) Two transceivers and C) Three transceivers

Carrying out trilateration normally involves a transmitter/receiver pair. A transmitter is used to ping a receiver to find the distance between the transmitter and the receiver using some TOA algorithm. In the case of GNSS systems the message is sent from the satellite to the device and based on the time-stamp of the message received the receiver does the computation on how far away the transmitter and receiver are. Most indoor positioning system have no global time system like in a GNSS system. To make the indoor positioning viable a transceiver has to initiate the ping to another transceiver which then sends back a ping to the initial transceiver then the first transceiver checks the delay between the sending and receiving ping to decide the distance between the transceivers. This step is commonly called ranging and it is the first step to the trilateration problem.

Given the ranging distance the position of a target transceiver is to be found using transceivers at a known position. The target transceiver (the transceiver for which we are interested in finding the position) could be anywhere around the sphere (if elevation is needed) or circle (if elevation is not needed) around the anchor transceiver (the transceiver which is anchored at known positions), assuming that the transmitter propagates equally well in all directions. For the second setup with two anchor transceivers using the known distance between the target transceiver and the two anchor transceivers, we can find the region of intersection (a circle) where the target transceiver could lie, by using the spheres of radiation at set distances to the target transceiver, propagating from the anchor transceivers. If we are only concerned with the 2-D case the point where the target transceiver is located is reduced to two possible points. Another distance measurement would limit the possible position of a target transceiver to two points for 3-D and only one in 2-D case (or constrained 3-D case). To get the location of the target transceiver we need at least four transmitters for 3-D, and three transmitters for 2-D, where the intersection of the spheres created by the radiation with radius equal to the distance to the target transceiver returns just one point. So by changing the anchor transceiver count from one to four the possibility of finding the target transceiver location in 3-D has been changed from a sphere to a circle to two points to just one point.

Using more anchor transceivers than the minimum required improves the accuracy of the positioning especially when the ranging measurements are prone to noise. It

is common in GNSS systems to use more than four satellites to give more accurate position estimates.

2.3 Bayesian estimation

There are many ways to estimate a state vector of a system given some additional information like measurement values. Some of them, like non-parametric estimators, use a set of particles to represent probability of certain states and how they affect each other. Also Least Mean Square filters are used for this purpose. In this project however, parametric Bayesian estimator will be used. Here we describe all states of the system in a common vector $\boldsymbol{\theta}$ and all the measurements in vector $\boldsymbol{\rho}$.

2.3.1 Finding the posterior

Bayesian estimators are minimum mean square error (MMSE) estimators and they aim to find the estimated state as

$$\hat{\boldsymbol{\theta}}_t^{\text{MMSE}} = \int \boldsymbol{\theta}_t p(\boldsymbol{\theta}_t|\boldsymbol{\rho}_{1:t})\mathrm{d}\boldsymbol{\theta}_t, \tag{2.22}$$

which is the expected value of the combined state given all the observed values, or simply the expected value of the posterior probability density function of the state $\boldsymbol{\theta}$ [13]. Finding the posterior is therefore an important task in state estimation. Since the process is assumed to fulfil the Markov property (the states at time t are dependent only on states in time step $t-1$), and the prior $p(\boldsymbol{\theta}_0|\boldsymbol{\rho}_0)$ is assumed to be known, the problem can be solved recursively. Usually one step in the recursion is split in two stages: prediction and update [22].

The prediction describes the probability of a state in time t given the information about the state in time $t-1$

$$p(\boldsymbol{\theta}_t|\boldsymbol{\rho}_{1:t-1}) = \int p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})p(\boldsymbol{\theta}_{t-1}|\boldsymbol{\rho}_{1:t-1})\mathrm{d}\boldsymbol{\theta}_{t-1}, \tag{2.23}$$

where the $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$ is defined by the system dynamics and $p(\boldsymbol{\theta}_{t-1}|\boldsymbol{\rho}_{1:t-1})$ is the posterior of the previous time iteration.

The update step can be done as follows

$$p(\boldsymbol{\theta}_t|\boldsymbol{\rho}_{1:t}) \propto p(\boldsymbol{\rho}_t|\boldsymbol{\theta}_t)p(\boldsymbol{\theta}_t|\boldsymbol{\rho}_{1:t-1}), \tag{2.24}$$

where the term $p(\boldsymbol{\rho}_t|\boldsymbol{\theta}_t)$ is the likelihood function of all the measurements in the system and $p(\boldsymbol{\theta}_t|\boldsymbol{\rho}_{1:t-1})$ is the output from the prediction step.

The equations (2.23) and (2.24) together gives the posterior needed to find the estimate of the state of interest. However, often the terms in these equations must be further factorised, depending on the number of different measurements and how the system is built up. The full factorisation of the posterior for the system (described in Section 2.1) is shown in Section 3.1. This factorisation can also be described in

form of a factor graph, representing the dependencies (represented as edges in the graph) of different functions (represented as rectangular nodes) and states (circular nodes). Belief propagation (or the Sum-product algorithm) [23, Ch. 4] is used in this project to, given the dependencies in the graph, estimate the posterior probability density function of the system. Since all distributions are assumed to be Gaussian in the graph, belief propagation will consist of a set of simple message passing rules.

2.3.2 Some simple message passing rules

Given the problem formulation it is desirable to estimate the posterior of the vehicle and feature states given the GNSS and V2F measurements. To obtain this posterior a belief propagation algorithm is performed on a factor graph resulting from the factorisation of the posterior [23, ch. 4]. However, to understand the derivation of the algorithm equations an understanding of how messages (representing the different probabilities in the graph) are propagated through different types of functions in the model and passed around in the graph.

The cases needed for understanding the ICP derivations are message passing through a linear mapping, an addition and how multiple messages are together used for estimating a belief of a state. Also the initialisation of a message will be considered. Note that given the assumptions made in this project only Gaussian messages will be present in the message passing algorithm. Further on E will denote the edge concerned in calculations along which the message is passed, and f will denote the node representing a function.

In case of initialisation of the message (that is a single outgoing edge from a node representing a probability density function), the message will be equal to the probability function of the node. Hence if $f(\boldsymbol{e}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{C})$ then the outgoing message is given by

$$m_{f\to E} = \mathcal{N}(\boldsymbol{\mu}, C). \tag{2.25}$$

In case of a message passing through a linear map as $\boldsymbol{e}_2 = A\boldsymbol{e}_1$, where $\boldsymbol{e}_1$ is the input to the function and $\boldsymbol{e}_2$ is the output, and $\boldsymbol{A}$ is a square and invertible matrix there are two different calculations to mention: The first one is when the message is passed in the direction of the function, that is where the known message is the message from edge E_1 to the function, denoted $m_{E_1\to f} = \mathcal{N}(\boldsymbol{\mu}_1, C_1)$, then the message of interest is

$$m_{f\to E_2} = \mathcal{N}(A\boldsymbol{\mu}_1, AC_1A^{\mathrm{T}}), \tag{2.26}$$

and the second, when the message is passed in direction opposite to the direction of the function. Message known is then denoted $m_{E_2\to f} = \mathcal{N}(\boldsymbol{m}_2, C_2)$, and message of interest can be calculated as follows

$$m_{f\to E_1} = \mathcal{N}(A^{-1}\boldsymbol{\mu}_2, A^{-1}C_2(A^{-1})^{\mathrm{T}}). \tag{2.27}$$

In case of an addition, where $\boldsymbol{e}_3 = \boldsymbol{e}_1 + \boldsymbol{e}_2$ and the incoming messages are $m_{E_1\to f} = \mathcal{N}(\boldsymbol{\mu}_1, C_1)$ and $m_{E_2\to f} = \mathcal{N}(\boldsymbol{\mu}_2, C_2)$, the outgoing message is

$$m_{f\to E_3} = \mathcal{N}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2, C_1 + C_2). \tag{2.28}$$

However, when message is passed in opposite direction the message can be computed

$$m_{f\to E_1} = \mathcal{N}(\boldsymbol{\mu}_3 - \boldsymbol{\mu}_2, C_3 + C_2). \tag{2.29}$$

Other important part of message passing is determining the belief of state $\boldsymbol{x}$ given all incoming messages $m_{E_1\to\boldsymbol{x}} = \mathcal{N}(\boldsymbol{\mu}_1, C_1), \ldots, m_{E_J\to\boldsymbol{x}} = \mathcal{N}(\boldsymbol{\mu}_J, C_J)$. The belief is simply the product of the messages and can be calculated as follows

$$b(\boldsymbol{x}) = \prod_{j=1}^{J} \mathcal{N}(\boldsymbol{\mu}_j, C_j) \propto \mathcal{N}(\boldsymbol{\mu}_f, C_f), \tag{2.30}$$

where

$$C_f = \left(\sum_{j=1}^{J} C_j^{-1}\right)^{-1} \qquad \text{and} \qquad \boldsymbol{\mu}_f = C_f \left(\sum_{j=1}^{J} C_j^{-1}\boldsymbol{\mu}_j\right). \tag{2.31}$$

These rules will be used later in Section 3.1 for derivation and explanation of the ICP algorithm.

2.4 Extended Kalman filter

The Kalman filter [19] is a Gaussian estimator for the states of a dynamical system. The algorithm mainly uses Bayesian statistics for estimating the probability distribution of the state variables over each time frame. The Kalman filter consists of two steps: (i) the prediction step and (ii) the update step. In the prediction step a linear model which describes the dynamics of the system is used to compute the states at the next time step. The estimate covariance are also updated depending on the process noise. These represent the prior before the measurement update. In the measurement update step the measurements are obtained and depending on the covariance of the measurements, the probability distribution are combined to update the states and the estimate covariance. A lower process noise indicates the filter to trust the model more whereas a lower measurement noise indicates the filter to trust the measurement more. The Kalman filter also have the added advantage of estimating states for which we do not have measurements but which could be observable [24, Ch. 7]. For a normally distributed variable $\boldsymbol{x}$ the conditional probability $p(\boldsymbol{x}_{k-1}|\boldsymbol{y}_{k-1})$ is described by the probability density function (PDF),

$$p(\boldsymbol{x}_{k-1}|\boldsymbol{y}_{1:k-1}) = \mathcal{N}(\hat{\boldsymbol{x}}_{k-1|k-1}, P_{k-1|k-1}). \tag{2.32}$$

The Kalman equations as stated in [19, p. 37] are summarised by two steps. The first being the prediction step written as

$$\hat{\boldsymbol{x}}_{k|k-1} = A_{k-1}\hat{\boldsymbol{x}}_{k-1|k-1}, \tag{2.33}$$

$$P_{k|k-1} = A_{k-1}P_{k-1|k-1}A_{k-1}^T + Q_{k-1}, \tag{2.34}$$

and the update step written as

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + K_k v_k, \tag{2.35}$$

$$P_{k|k} = P_{k|k-1} - K_k S_k K_k^T. \tag{2.36}$$

The Kalman gain K_k, innovation v_k and the innovation covariance S_k are obtained by:

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{2.37}$$

$$v_k = \boldsymbol{y}_k - H_k \hat{\boldsymbol{x}}_{k|k-1} \tag{2.38}$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k. \tag{2.39}$$

Where $\hat{\boldsymbol{x}}_{k|k-1}$ represents the prediction of states for next time step, $P_{k|k-1}$ is the error covariance estimate, Q represents the process noise. $\hat{\boldsymbol{x}}_{k|k}$ and $P_{k|k}$ is the updated (posterior) state and updated (posterior) covariance estimates respectively. H_k represents the observability matrix. The posterior state vector $\hat{\boldsymbol{x}}_{k|k}$ is the filtered state estimates.
The Kalman filter mentioned above is useful for the linear motion and measurement models cases. For non linear motion and/or measurement model we need to approximate the non linear model to a linear one using the Jacobian. This type of Kalman filtering is called the extended Kalman filter (EKF). Assume the non-linear model is described by

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}) + q_{k-1}. \tag{2.40}$$

Then the linearised model is represented by

$$\boldsymbol{x}_k \approx f(\hat{\boldsymbol{x}}_{k-1|k-1}) + f'(\hat{\boldsymbol{x}}_{k-1|k-1})(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1|k-1}) + q_{k-1}. \tag{2.41}$$

Similarly the measurement model if non-linear can be described as

$$\boldsymbol{y}_k = h(\boldsymbol{x}_k) + r_k, \tag{2.42}$$

with the linear measurement model written as

$$\boldsymbol{y}_k \approx h(\hat{\boldsymbol{x}}_{k|k-1}) + h'(\hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1}) + r_{k-1}, \tag{2.43}$$

where $f'(\hat{\boldsymbol{x}}_{k-1|k-1})$ and $h'(\hat{\boldsymbol{x}}_{k|k-1})$ are the jacobians given by

$$[f'(\hat{\boldsymbol{x}}_{k-1|k-1})]_{ij} = \frac{\partial g_i(\boldsymbol{x})}{\partial x_j}, \ [h'(\hat{\boldsymbol{x}}_{k|k-1})]_{ij} = \frac{\partial h_i(\boldsymbol{x})}{\partial x_j}. \tag{2.44}$$

The EKF equations as seen in [19, p. 69-70] thus can be written as:
The prediction step

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}), \tag{2.45}$$

$$P_{k|k-1} = f'(\hat{\boldsymbol{x}}_{k-1|k-1}) P_{k-1|k-1} f'(\hat{\boldsymbol{x}}_{k-1|k-1})^T + Q_{k-1}. \tag{2.46}$$

The update step

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + K_k v_k, \tag{2.47}$$

$$P_{k|k} = P_{k|k-1} - K_k S_k K_k^T. \tag{2.48}$$

The Kalman gain K_k, innovation v_k and the innovation covariance S_k are obtained by:

$$K_k = P_{k|k-1} h'(\hat{\boldsymbol{x}}_{k|k-1})^T S_k^{-1}, \tag{2.49}$$

$$v_k = \boldsymbol{y}_k - h(\hat{\boldsymbol{x}}_{k|k-1}), \tag{2.50}$$

$$S_k = h'(\hat{\boldsymbol{x}}_{k|k-1}) P_{k|k-1} h'(\hat{\boldsymbol{x}}_{k|k-1})^T + R_k. \tag{2.51}$$

The EKF hence can be used to estimate non-linear states as described above this will be later helpful in our implementation later in the project.

2.5 Cramer-Rao bound

Most estimators have a variance in the estimates, being able to place a lower bound on the variance of any unbiased estimator can be extremely useful. At the best case it allows us to assert that an estimator is the minimum variance unbiased (MVU) estimator. Many such bounds do exists but the Cramer-Rao bound (CRB) is the easiest to determine.
Since all the information for the CRB is obtained from the observed data and the PDF for that data, the accuracy of the estimation directly depends on the PDF. So if the influence of the unknown parameter on the PDF is more, the better we should be able to estimate it.
A simple estimator model as stated in [25, p. 28] can be written as

$$x[0] = A + w[0], \tag{2.52}$$

where $w[0] \sim \mathcal{N}(0, \sigma^2)$ and A is the parameter to be estimated. We can expect the estimate to be good if the variance σ^2 is small. For the ideal case the estimate is the best if σ^2 is 0, which is the best unbiased estimator where the estimate then becomes $\hat{A} = x[0]$. The estimator accuracy will hence increase as σ^2 decreases.
As A is a deterministic value, the distribution of $x[0]$ will be normally distributed such that $A \sim \mathcal{N}(A, \sigma^2)$ the PDF of $x[0]$ can be written as

$$p_i(x[0]; A) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left[-\frac{1}{2\sigma_i^2}(x[0] - A)^2\right]. \tag{2.53}$$

When the PDF is viewed as a function of the unknown parameter it is termed as the likelihood function. The sharpness of the likelihood function determines the accuracy. To quantify and measure this, the negative of the second derivative of the logarithm of the likelihood function is taken. This quantity gives us the curvature of the log-likelihood function. To demonstrate this we will take the natural logarithm of (2.53)

$$\ln p_i(x[0]; A) = -\ln\sqrt{2\pi\sigma_i^2} - \frac{1}{2\sigma_i^2}(x[0] - A)^2. \tag{2.54}$$

The first derivative is

$$\frac{\partial \ln p_i(x[0]; A)}{\partial A} = -\frac{1}{\sigma_i^2}(x[0] - A) \tag{2.55}$$

and the negative of the second derivative becomes

$$-\frac{\partial^2 \ln p_i(x[0];A)}{\partial A^2} = \frac{1}{\sigma_i^2}. \tag{2.56}$$

The curvature of the PDF will increase as σ^2 decreases. Since for our simple estimator case we know that $\hat{A} = x[0]$ has variance σ^2. We can write the variance of the estimator as

$$\mathrm{Var}(\hat{A}) = \frac{1}{-\frac{\partial^2 \ln p_i(x[0];A)}{\partial A^2}}. \tag{2.57}$$

In this simple example the second derivative does not depend on $x[0]$ but in the general case it would. For a general case, assume θ is the unknown parameters to be found from the measurements x, such that θ is distributed according to a PDF $f(x;\theta)$. If this is the case then the variance of the unbiased estimator $\hat{\theta}$ of θ is then bounded by the inverse of the Fisher information matrix (FIM) $I(\theta)$ as mentioned in [25, p. 30] i.e

$$\mathrm{Var}(\hat{\theta}) \geq \frac{1}{I(\theta)}. \tag{2.58}$$

The FIM is defined by the second derivative of the negative log likelihood function of the probability density function (pdf). As given by (2.59)

$$I(\theta) = -E\left[\frac{\partial^2 \ln(f(x;\theta))}{\partial \theta^2}\right]. \tag{2.59}$$

It should be noted that no unbiased estimator can exist whose variance is lower than σ^2. Hence the CRLB is the lowest possible bound an estimator can ideally reach. We would like to find out the CRB of the positioning system using the TOA system. The sections below will cover the derivation of the CRB for the distance measurement and then use that information to find the CRB for the position.

2.5.1 Cramer-Rao bound for ranging

Before we can find the CRB for the complete trilateration setup we need to find the CRB for the simple ranging scenario. For electromagnetic radiation propagation we know that the height of the transmitting and receiving antenna is crucial as there would be losses due to the fresnel zone. The minimum length of the antenna required to transmit without the influence of the Fresnel effect as mentioned in [26, p. 16] can be written as

$$H\ [m] = 8.656\sqrt{\frac{D\ [\mathrm{km}]}{f\ [\mathrm{GHz}]}}, \tag{2.60}$$

where H is the height of the antenna needed, D is the distance between the transmitter and receiver and f the frequency of transmission. As the test setup is set up in an area between a rectangle of dimensions of 3 m length and 3.5 m breadth the maximum distance possible between the beacons is 4.6 m . The frequency of transmission for the UWB is 2.2 GHz using this the height required for transmission was found to be approximately 40 cm. For the future experiments the antenna heights

were taken as 80 cm.
As mentioned in [27] the variance of CRB for ranging can be found by

$$\frac{1}{\text{Var}} = \frac{c^2}{8\pi^2 P_{\text{SNR}} B^2}. \tag{2.61}$$

The SNR is calculated by finding the power received and power lost using the equations (assuming no fading). The power received at distance d from the transmitter is found by

$$P_r = P_t k \frac{D_0^\gamma}{D_d^\gamma}, \tag{2.62}$$

where P_t is the power transmitted by the transmitter antenna which is $3.981 \cdot 10^{-5}$ W, D_0 is the reference distance 1 m, k is $10^{-L_0/10}$ where L_0 is the power received at the reference distance, k is very close to 1. D_d is the distance between the beacons and γ is the path loss exponent which is taken as 2 assuming vacuum. The noise power hence can be calculated by

$$P_n = 4 k_b T B. \tag{2.63}$$

Where k_b is the Boltzman constant, T is the atmospheric temperature which is about 293 K and B is the bandwidth of transmission which is 2.2 GHz

$$P_{\text{SNR}} = \frac{P_r}{P_n}. \tag{2.64}$$

The CRB for the distance estimate was plotted against distance between the transmitter and receiver, the resulting plot is shown in Figure 4.1.

2.5.2 Cramer-Rao bound for positioning

For finding the CRB for positioning, the net 2D covariance of a given position is found by summing up the influence of the variance of the multiple pivot anchors at that point as described in detail in [28], i.e the CRB is givn by

$$\text{Var} = J_c^{-1}, \tag{2.65}$$

where J_c Represents the FIM which can be found by

$$J_c = \sum_{i=1}^{n} J_{rn} \lambda(D_n). \tag{2.66}$$

The quantity $\lambda(D)$ is the variance of ranging given the distance between the beacons D which can be found using (2.61) - (2.64), and n denotes the total number of anchor beacons. The transformation matrix J_r denotes the influence of the variance due to ranging on the position. The quantity J_r is dependent on the angle ϕ. The angle ϕ is the angle made by the line joining the two beacons with respect to the positive x-axis. The transformation matrix is given by

$$J_r = \begin{bmatrix} \cos^2(\phi) & \cos(\phi)\sin(\phi) \\ \cos(\phi)\sin(\phi) & \sin^2(\phi) \end{bmatrix}. \tag{2.67}$$

As the CRB for positioning is dependent on the configuration of the setup and the total number of beacons, a test setup as shown in Figure 2.5 was put up. Later more tests were carried out by increasing the beacon count and changing the configuration of the beacons.

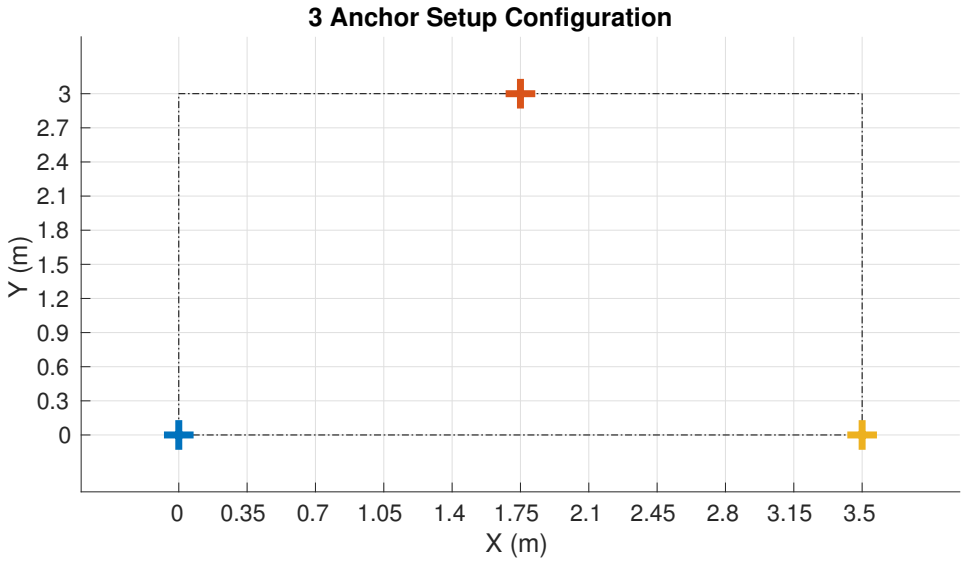


Figure 2.5: Test configuration for finding CRB for positioning with anchor beacons represented as crosses.

For the purpose of illustration and making the CRB visually obvious the 300σ ellipses at various points in the test setup were evaluated and plotted. The plot is shown in Figure 4.2. The CRB for other kind of configurations were also found which can be seen from the Figures 4.3 - 4.5.

2.6 Linear quadratic regulator

There are many type of controllers like the PID controllers or more complex ones like the model predictive control (MPC). While the PID controller just tries to minimise the error in a simple way the MPC controller solves an optimisation problem which tries to find the control input between set constraints. The linear quadratic regulator (LQR) on the other hand is somewhere in between the PID and a MPC controller. The LQR is a an optimal controller used for dynamical systems that can be described by linear differential equations. It has the advantage of being not so computationally complex yet being very robust at control. The LQR also easier to tune when compared to a PID Controller. The LQR minimises a cost function based on the weight on the system states and input, which is predetermined by the control designer. The infinite-horizon, discrete time LQR are described in (2.68)-(2.72).
For a discrete-time linear system described by [29]

$$\boldsymbol{x}_{k+1} = A_{\mathrm{LQR}}\boldsymbol{x}_k + B_{\mathrm{LQR}}\boldsymbol{u}_k, \tag{2.68}$$

where $\boldsymbol{x}$ is the vector of states, $\boldsymbol{u}$ is the vector of input and k is the time step. With

a cost function J defined as

$$J = \sum_{k=0}^{\infty} \left(\boldsymbol{x}_k^T Q \boldsymbol{x}_k + \boldsymbol{u}_k^T R \boldsymbol{u}_k + 2\boldsymbol{x}_k^T N \boldsymbol{u}_k \right), \tag{2.69}$$

where Q is the weight matrix for the states which has dimensions $n \times n$, n is the size of the state vector. The weight matrix R for the inputs has dimensions $m \times m$, m is the number of input. N is the weight matrix for input and states combined, has dimensions $n \times m$. The optimal control sequence minimising the cost function is given by

$$\boldsymbol{u}_k = -K\boldsymbol{x}_k, \tag{2.70}$$

where the controller gain K is given by

$$K = (R + B_{\mathrm{LQR}}^{\mathrm{T}} P B_{\mathrm{LQR}})^{-1} (B_{\mathrm{LQR}}^T P A_{\mathrm{LQR}} + N^{\mathrm{T}}). \tag{2.71}$$

P is the unique positive definite solution to the discrete time algebraic Riccati equation (DARE) which can be evaluated by

$$P = A_{\mathrm{LQR}}^T P A_{\mathrm{LQR}} - (A_{\mathrm{LQR}}^T P B_{\mathrm{LQR}} + N) \left(R + B_{\mathrm{LQR}}^T P B_{\mathrm{LQR}} \right)^{-1} (B_{\mathrm{LQR}}^T P A_{\mathrm{LQR}} + N^{\mathrm{T}}) + Q. \tag{2.72}$$

The infinite horizon problem can be solved by the finite horizon problem by recursively solving the finite horizon problem until it converges. For getting reference tracking the controller gain is multiplied to the difference between the present state and the reference state and this gives us the control input which is feedback to the system.

2.7 Robot Operating System

Robot Operating System (ROS) [30] is a flexible platform/framework for writing robot software. It is a collection of tools and libraries that are geared towards simplifying the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. ROS was first developed by the Stanford Artificial Intelligence Laboratory in 2007. From then on it has been developed as an open source source platform. ROS is extensively used for robotics projects owing to it's robustness, modularity and enormous community driven libraries which can be easily incorporated without fiddling too much with the underlying architecture of software or the hardware. ROS system mainly consists of a ROSCORE which is the master node through which all information passes to the various programs. ROSCORE is responsible for maintaining the flow of data, scheduling and routing of the various intricate tasks for the functioning of the system. A ROS system can be programmed in two ways, one being the publisher-subscriber or the service-client workflow. ROS system is mainly comprised of nodes which are programs intended to do a particular task with the information that is transferred via ROSCORE. The nodes for a publisher-subscriber can be broadly classified into two groups, a publisher and a subscriber. A publisher node keeps publishing some kind of data continuously whereas the subscriber will continuously access the data posted by the

subscriber. The publisher will keep posting the data to a pre-configured channel which are called as topics in ROS terminology. The subscribers will access the data from the topics. The data sent to the topics are called messages in ROS terminology which are wrapped based on a pre-defined or a user-defined data structure. The basic block diagram for a ROS system is shown in Figure 2.6

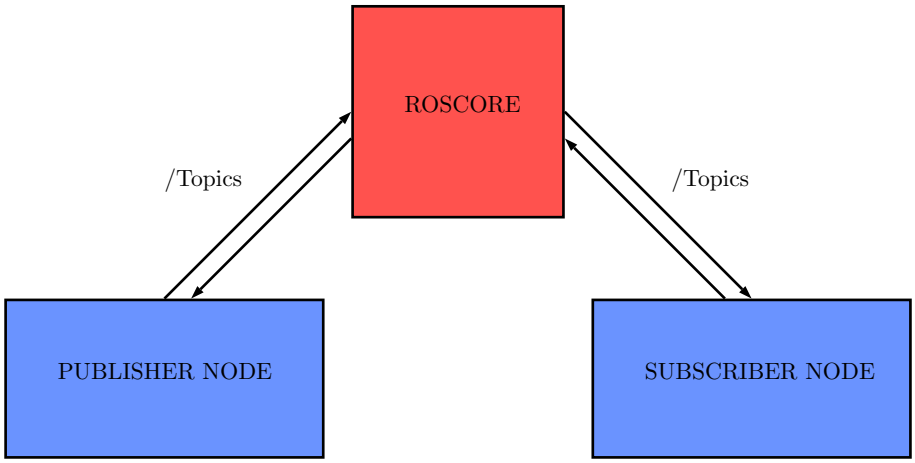


Figure 2.6: Publisher and Subscriber nodes in a ROS system

The service-client work-flow on the other hand does not continuously stream data as in the case of the Publisher-Subscriber but only does it when the data is requested by the client node. So the client node will request the message by initiating the call to the service node via the ROSCORE and wait for the response. The service will get the request for the message and will acknowledge it and then send the data to the client. The Pioneer-D3X robots have a publisher-subscriber node called ROSARIA which helps the robot to inform subscribers about it's sensor states and also accept the input to the wheel motors as linear and angular velocity from other publishers.

3
Methods

This chapter gives a description of the algorithm and how it was implemented in a real world scenario using the theory presented in the previous chapter.

3.1 ICP

The purpose of ICP is to find the MMSE estimate of the vehicle state (hence also position) of the vehicle at a certain time step, given all the information contained in the GNSS measurements of the vehicles and all the V2F measurements [13]. That is, the objective of ICP is to find the posterior distribution of the vehicle state $\boldsymbol{x}_{i,1:t}$ given the measurements $\boldsymbol{\rho}^{\mathrm{G}}_{i,1:t}$ and $\boldsymbol{\rho}^{\mathrm{V2F}}_{i,k,1:t}$. This posterior can be denoted as $p(\boldsymbol{x}_{i,t}|\boldsymbol{\rho}^{\mathrm{G}}_{i,1:t}, \boldsymbol{\rho}^{\mathrm{V2F}}_{i,k,1:t})$. Since all the vehicle and feature states are assumed independent of each other and are as well fulfilling the Markov criterion, this distribution can be factorised, splitting the calculations over every time step (iterative filtering) and making it possible for the GNSS and V2F updates to be done separately.
The factorisation of the posterior for each time iteration is done as follows

$$p(\boldsymbol{\theta}_t|\boldsymbol{\rho}_{1:t}) \propto \prod_{i\in\mathcal{V}} \Big(p(\boldsymbol{\rho}^{\mathrm{G}}_{i,t}|\boldsymbol{x}_{i,t}) \int p(\boldsymbol{x}_{i,t}|\boldsymbol{x}_{i,t-1}) p(\boldsymbol{x}_{i,t-1}|\boldsymbol{\rho}_{1:t-1}) d\boldsymbol{x}_{i,t-1} \times \prod_{k\in\mathcal{F}_{i,t}} p(\boldsymbol{\rho}^{\mathrm{V2F}}_{i,k,t}|\boldsymbol{f}_k, \boldsymbol{x}_{i,t}) \prod_{k\in\mathcal{F}} p(\boldsymbol{f}_k|\boldsymbol{\rho}_{1:t-1}) \quad (3.1)$$

where the $\boldsymbol{\theta}$ is the full state vector of the system containing both vehicle state and the feature state, and the $\boldsymbol{\rho}_t$ is the combined measurement vector with both GNSS and V2F measurements.
Here one can see that many of the probability functions may be calculated using the knowledge about the system model from Section 2.1 and applying the belief propagation rules from Section 2.3.2:

- Motion model can be used to describe the probability of the vehicle state $\boldsymbol{x}_{i,t}$ considering the vehicle state in previous time iteration $\boldsymbol{x}_{i,t-1}$, i.e. $p(\boldsymbol{x}_{i,t}|\boldsymbol{x}_{i,t-1})$
- $p(\boldsymbol{\rho}^{\mathrm{G}}_{i,t}|\boldsymbol{x}_{i,t})$ is the likelihood of the GNSS measurement given the current vehicle state and can be described using the GNSS measurement model,
- $p(\boldsymbol{\rho}^{\mathrm{V2F}}_{i,k,t}|\boldsymbol{f}_k, \boldsymbol{x}_{i,t})$ is the V2F measurement likelihood that can be described using the V2F measurement model.

probability functions $p(\boldsymbol{x}_{i,t-1}|\boldsymbol{\rho}_{1:t-1})$ and $p(\boldsymbol{f}_k|\boldsymbol{\rho}_{1:t-1})$ are the priors of a time iteration of the algorithm, which simply are the outcome of the previous iteration.

3.1.1 Belief propagation in the factor graph

The factorisation in (3.1) can be presented in a factor graph illustrated in Figure 3.1 depicting all the influences and all the passed information present in the system.

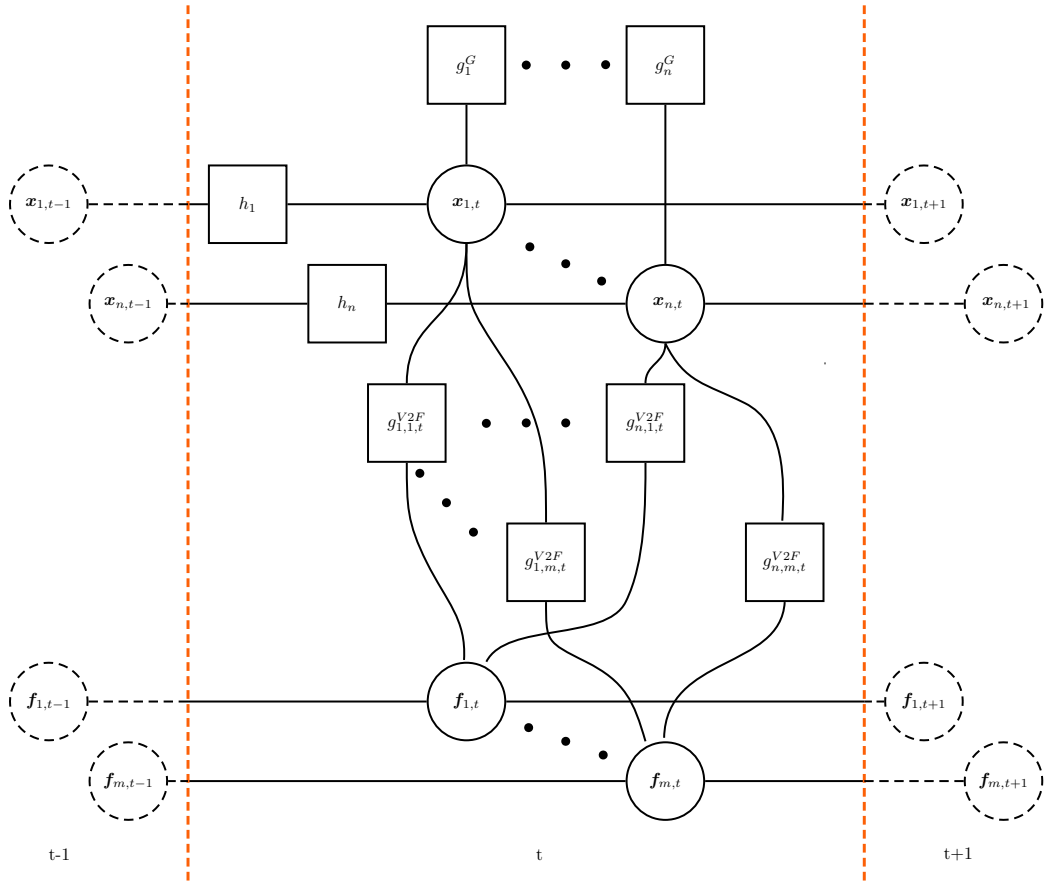


Figure 3.1: Factor graph representation of the system where g_i^{G} represents the GNSS measurement function, h_i the motion model of the vehicle and $g_{i,k,t}^{\mathrm{V2f}}$ the V2F measurement. $\boldsymbol{x}_{i,t}$ and $\boldsymbol{f}_{k,t}$ are the vehicle state and the feature state at time t.

3.1.1.1 Prediction message

The effect of the state $\boldsymbol{x}_{i,t-1}$ on the state $\boldsymbol{x}_{i,t}$ is described by the motion model of the vehicle i and passed to the state node using the prediction message $m_{h_i \to x_i}$ using the rules described in Section 2.3.2.
The motion model used is the CV model described in Section 2.1.1

$$\boldsymbol{x}_{i,t} = A_{\mathrm{CV}}\boldsymbol{x}_{i,t-1} + B_{\mathrm{CV}}\boldsymbol{u}_{i,t-1} + \boldsymbol{w}_{i,t-1}. \tag{3.2}$$

The prediction message to the vehicle state $\boldsymbol{x}_{i,t}$ can then be calculated as follows

$$m_{h_i \to x_i} = \int h_i b(\boldsymbol{x}_{i,t-1}^{N_{mp}}) \mathrm{d}x_{i,t-1} \tag{3.3}$$

$$= \mathcal{N}(A_{\mathrm{CV}}\boldsymbol{\mu}_{i,t-1}^{N_{mp}} + B_{\mathrm{CV}}\boldsymbol{u}_{i,t-1}^{N_{mp}},\ A_{\mathrm{CV}}C_{i,t-1}^{N_{mp}}A_{\mathrm{CV}}^T + Q_{i,t-1}), \tag{3.4}$$

where the $b(\boldsymbol{x}_{i,t}^{N_{mp}}) = \mathcal{N}(\boldsymbol{\mu}_{i,t-1}^{N_{mp}}, C_{i,t-1}^{N_{mp}})$ is the final belief of the vehicle i's state from the previous time iteration, and where $h_i = p(\boldsymbol{x}_{i,t}|\boldsymbol{x}_{i,t-1})$ describes how the current state $\boldsymbol{x}_{i,t}$ is affected by the state in previous time instance $\boldsymbol{x}_{i,t-1}$.

3.1.1.2 GNSS measurement update

The factorisation of the GNSS measurement can be depicted using the GNSS measurement model from Section 2.1.3 as shown below

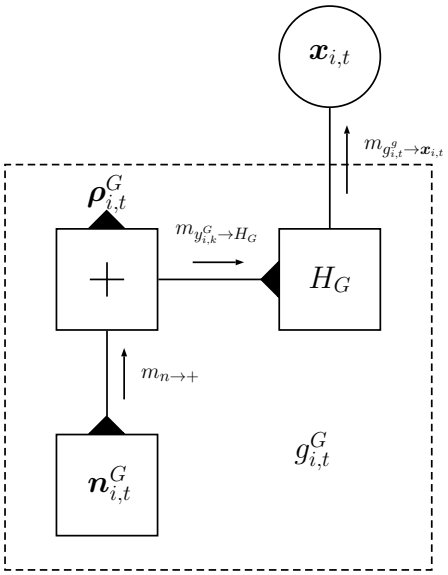


Figure 3.2: The factor graph representing the effect of the vehicle state $\boldsymbol{x}_{i,t}$ and the measurement noise $\boldsymbol{n}_{i,t}^{\mathrm{G}}$ on the GNSS measurement $\boldsymbol{\rho}_{i,t}^{\mathrm{G}}$, as described in Section 2.1.3

Using this factor graph Figure 3.2, the GNSS update can be performed on the prediction message, forming the initial belief $b(\boldsymbol{x}_{i,t}^0)$ of the vehicle state in the message passing algorithm will boil down to the usual Kalman filter update step equations. Thus the belief of the vehicle can be updated according to following [23, pp. 120-123]

$$b(\boldsymbol{x}_{i,t}^0) = \mathcal{N}(\boldsymbol{\mu}_{i,t}^0, C_{i,t}^0), \tag{3.5}$$

where

$$\boldsymbol{\mu}_{i,t}^0 = \boldsymbol{\mu}_{h_i\to x_i} + K(\boldsymbol{\rho}_{i,t}^{\mathrm{G}} - H_{\mathrm{G}}\boldsymbol{\mu}_{h_i\to x_i}), \tag{3.6}$$

$$C_{i,t}^0 = (\boldsymbol{I} - KH_{\mathrm{G}})C_{h_i\to x_i}, \tag{3.7}$$

and where

$$K = C_{h_i\to x_i}(H_{\mathrm{G}})^T(H_{\mathrm{G}}C_{h_i\to x_i}(H_{\mathrm{G}})^T + R_{i,t}^{\mathrm{G}})^{-1}. \tag{3.8}$$

Here, (3.6) updates the expected value of the prediction message $\boldsymbol{\mu}_{h_i\to x_i}$ with the sum of a Kalman gain K and the innovation term $\boldsymbol{\rho}_{i,t}^{\mathrm{G}} - H^{\mathrm{G}}\boldsymbol{\mu}_{h_i\to x_i}$. The innovation is the difference between the measurement value and the predicted position. Kalman gain decides according to the covariance matrices, how much of the innovation should be

used for update, that is whether the measurement or the prediction should be more trusted. The higher the measurement covariance (lower measurement accuracy) compared to the covariance of predicted state, the more the prediction will affect the updated belief of the state, and vice versa.

For the optimal function of the Kalman filter the measurement noise needs to be normally distributed with zero mean. Since there are non-linearities in the TOA-based position measurements this is not the case in the real case scenario. The Monte-Carlo approximation can be applied on a set of measurements to approximate the measurement noise as Gaussian, which should be sufficient for good functioning of the filter.

3.1.1.3 Feature measurement update

The measurement likelihood function $g_{i,k,t}$ can be then depicted (using the V2F measurement model from Section 2.1.3) in the factor graph presented below

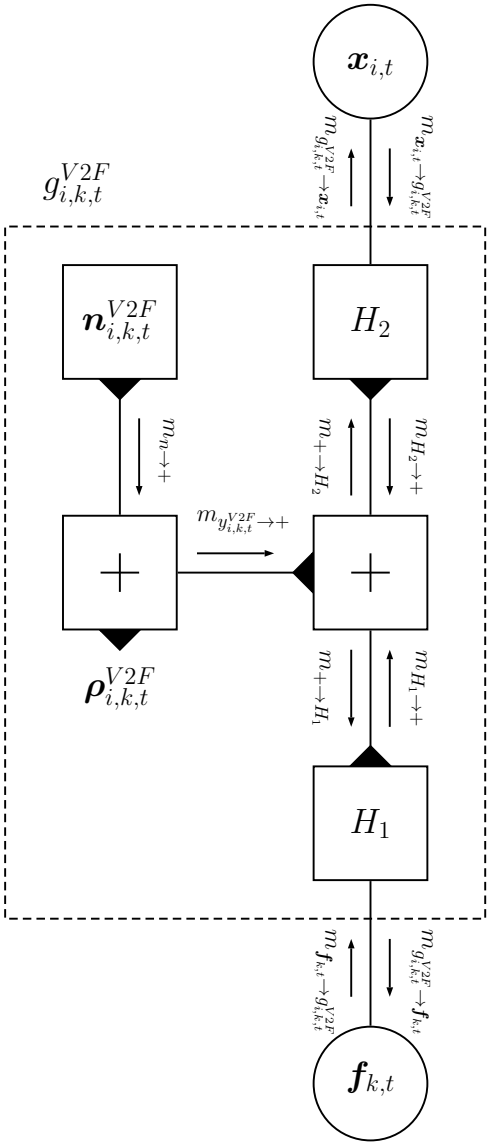


Figure 3.3: The factor graph representation of the effect of the feature state $\boldsymbol{f}_{k,t}$, the vehicle state $\boldsymbol{x}_{i,t}$ and the measurement noise $\boldsymbol{n}^{\mathrm{V2F}}_{i,k,t}$ on the V2F measurement $\boldsymbol{\rho}^{\mathrm{V2F}}_{i,k,t}$. Matrices H_1 and H_2 are the observability matrices. See Section 2.1.3 for further explanation.

Following the conventions of message passing from Section 2.3.2 from state $\boldsymbol{x}_{i,t}$ across the $g_{i,k,t}$ to the state $\boldsymbol{f}_{k,t}$ results in the expressions for the individual messages.
The message $m_{x_i\to g_{i,k}}$ is firstly passed through the matrix H_2 and the outgoing message becomes

$$m_{H_1\to+} = \mathcal{N}(H_1\boldsymbol{\mu}^{n-1}_{f_k\to g_{i,k}},\, H_1 C^{n}_{f_k\to g_{i,k}} H_1^T), \tag{3.9}$$

Note here that n is the iteration index of the loop in the message passing algorithm. It will be further discussed in Section 3.1.2. The message from the measurement node $y^{\mathrm{V2F}}_{i,k}$ is representing the likelihood function of the V2F

$$m_{y^{\mathrm{V2F}}_{i,k,t}\to+} = \mathcal{N}(\boldsymbol{\rho}^{\mathrm{V2F}}_{i.k},\, R^{\mathrm{V2F}}). \tag{3.10}$$

Further on the two above messages are merged

$$m_{+\to H_2} = \mathcal{N}(\boldsymbol{\rho}_{i,k}^{\text{V2F}} - H_1 \boldsymbol{\mu}_{f_k \to g_{i,k}}^n,\, R^{\text{V2F}} + H_1 C_{f_k \to g_{i,k}}^n H_1^T). \tag{3.11}$$

The outgoing messages from the measurement $g_{i,k}$ are not possible to calculate as the observability matrices H_1 and H_2 does not necessarily need to be square, and therefore not invertible. However, it turns out that the product of these messages is possible to calculate exactly. Since the beliefs of the vehicle states $\boldsymbol{x}_i$ and feature states $\boldsymbol{f}_k$ and their outgoing messages can all be calculated using the product of their incoming messages, the product is the only thing needed.
According to (2.30) the information sufficient for calculating the product is therefore following two expressions

$${C_{g_{i,k} \to x_i}^n}^{-1} = H_2^T (R^{\text{V2F}} + H_1 C_{f_k \to g_{i,k}}^n H_1^T)^{-1} H_1, \tag{3.12}$$

and

$$\begin{aligned} {C_{g_{i,k} \to x_i}^n}^{-1} \boldsymbol{\mu}_{g_{i,k} \to x_i}^n =& H_2^T (R^{\text{V2F}} + H_1 C_{f_k \to g_{i,k}}^n H_1^T)^{-1} H_2 H_2^{-1} (\boldsymbol{\rho}_k^{feat} - H_1 \boldsymbol{\mu}_{f_k \to g_{i,k}}^n) && (3.13) \\ =& H_2^T (R^{\text{V2F}} + H_1 C_{f_k \to g_{i,k}}^n H_1^T)^{-1} (\boldsymbol{\rho}_k^{feat} - H_1 \boldsymbol{\mu}_{f_k \to g_{i,k}}^n), && (3.14) \end{aligned}$$

As can be seen in (3.14) by multiplying the inverse of the covariance with the expected value of the message the inverse of the non-square matrix H_2 is eliminated. That is the reason to why the product of the messages is possible to calculate, while the expected value itself is not.
In similar manner the message passing from the feature state to the vehicle state can be done resulting in:

$${C_{g_{i,k} \to f_k}^n}^{-1} = H_1^T (R^{\text{V2F}} + H_2 C_{x_i \to g_{i,k}}^{n-1} H_2^T)^{-1} H_1, \tag{3.15}$$

and

$${C_{g_{i,k} \to f_k}^n}^{-1} \boldsymbol{\mu}_{g_{i,k} \to f_k}^n = H_1^T (R^{\text{V2F}} + H_2 C_{x_i \to g_{i,k}}^{n-1} H_2^T)^{-1} (\boldsymbol{\rho}_k^{feat} - H_2 \boldsymbol{\mu}_{x_i \to g_{i,k}}^{n-1}). \tag{3.16}$$

In this case, since the H_1 is an invertible, the actual expected value of the message can be calculated. The calculations are done in the same fashion though, to make the algorithm more consistent and allow for changes in the measurement model.

3.1.2 Message passing algorithm

After the initial belief $b(\boldsymbol{x}_{i,t}^0)$ is calculated for each vehicle, the information contained in the feature measurements should be incorporated in the vehicle beliefs. However, since there exist loops in the factor graph, the usual sum-product message passing algorithm is no longer sufficient to calculate the beliefs of the states of interest. It needs therefore be expanded by applying so called loopy belief propagation. The idea is to iteratively pass the messages around in the loop, letting them affect messages starting other loops in the end of each iteration.

Firstly, using the message incoming to $g_{i,k}$ from the vehicle state $\boldsymbol{x}_i$ from the previous iteration of the message passing algorithm, $\boldsymbol{\mu}^{n-1}_{x_i \to g_{i,k}}$, the terms of outgoing message from $g_{i,k}$ are calculated according to equations (3.15) and (3.16). Notice here that at the initial iteration $n = 1$ the messages $\boldsymbol{\mu}^{n-1}_{x_{i,t} \to g_{i,k,t}}$ has not yet been calculated. Therefore for the calculations of (3.15) and (3.16) the value is simply set $\boldsymbol{\mu}^{0}_{x_{i,t} \to g_{i,k,t}} = b(\boldsymbol{x}^0_{i,t})$.
This is done in parallel by each vehicle for every of vehicle's visible features. Since the algorithm is implemented here in a pseudo-decentralised fashion, the terms acquired from (3.15) and (3.16), can be accessed directly from all vehicles (there is no V2V communication needed) and therefore consensus value $u_{k,t,n}$ can be found directly by multiplying all incoming messages to the feature state $\boldsymbol{f}_k$.
Thus the consensus over belief of feature's k state at time t and iteration n is given by

$$u^u_{k,t} = \mathcal{N}(\boldsymbol{\mu}^{\text{cons}}_{k,t,n}, \boldsymbol{C}^{\text{cons}}_{k,t,n}), \tag{3.17}$$

where

$$\boldsymbol{C}^{cons}_{k,t,n} = \left(\sum_{i \in \mathcal{V}} {C^n_{g_{i,k} \to f_k}}^{-1} \right)^{-1}, \tag{3.18}$$

and

$$\boldsymbol{\mu}^{cons}_{k,t,n} = \boldsymbol{C}^{\text{cons}}_{k,t,n} \left(\sum_{i \in \mathcal{V}} {C^n_{g_{i,k} \to f_k}}^{-1} \boldsymbol{\mu}^n_{g_i \to x_i} \right). \tag{3.19}$$

Continuing, each vehicle calculates for all features the current message from the all the feature state $f_{k,t}$ to the measurement function $g_{i,k,t}$ according to following equation

$$\boldsymbol{\mu}^n_{f_{k,t} \to g_{i,k,t}} = b(\boldsymbol{f}_{k,t-1}) \prod_{l \in \mathcal{V} \setminus \{i\}} \boldsymbol{\mu}^n_{\boldsymbol{x}_{l,t} \to \boldsymbol{f}_{k,t}}. \tag{3.20}$$

Since (3.20) contain a product of all incoming messages to a feature except the message from vehicle i, this equation can be simplified by using the consensus value. Rewriting gives

$$\boldsymbol{\mu}^n_{f_{k,t} \to g_{i,k,t}} = b(\boldsymbol{f}_{k,t-1}) \frac{u^n_{k,t}}{\boldsymbol{\mu}^n_{\boldsymbol{x}_{i,t} \to \boldsymbol{f}_{k,t}}}, \tag{3.21}$$

where the division of Gaussians is done as in the product in (2.30) but using subtraction of the terms of distribution in the denominator instead of addition.
Further on the terms describing message $m^n_{g_{i,k,t} \to \boldsymbol{x}_{i,t}}$ are obtained using (3.12) and (3.14).
The outgoing messages from the vehicle state can then be calculated for use in next iteration as follows

$$\boldsymbol{\mu}^n_{x_{i,t} \to g_{i,k,t}} = b(\boldsymbol{x}^0_{i,t}) \prod_{j \in \mathcal{F} \setminus \{k\}} \boldsymbol{\mu}^n_{\boldsymbol{f}_{j,t} \to \boldsymbol{x}_{i,t}}. \tag{3.22}$$

After all the iterations in message passing algorithm are done the vehicles' and the features' states can be calculated as in (2.30). These beliefs will be then

$$b(\boldsymbol{f}_{k,t}) = u_{k,t}^{n} b(\boldsymbol{f}_{k,t-1}), \tag{3.23}$$

and

$$b(\boldsymbol{x}_{i,t}) = b(\boldsymbol{x}_{i,t}^{0}) \prod_{k \in \mathcal{F}} \boldsymbol{\mu}_{\boldsymbol{f}_{k,t} \to \boldsymbol{x}_{i,t}}^{n}. \tag{3.24}$$

Here a new time iteration of the algorithm can be started.

3.2 Hardware and ROS setup

This Section will discuss the setup used to carry out the experiments, the hardware and the framework used to develop the robot software. The various aspects of how the link to an actual real life scenario is achieved by the experiment will be portrayed.

3.2.1 Real scenario - experimental setup analogy

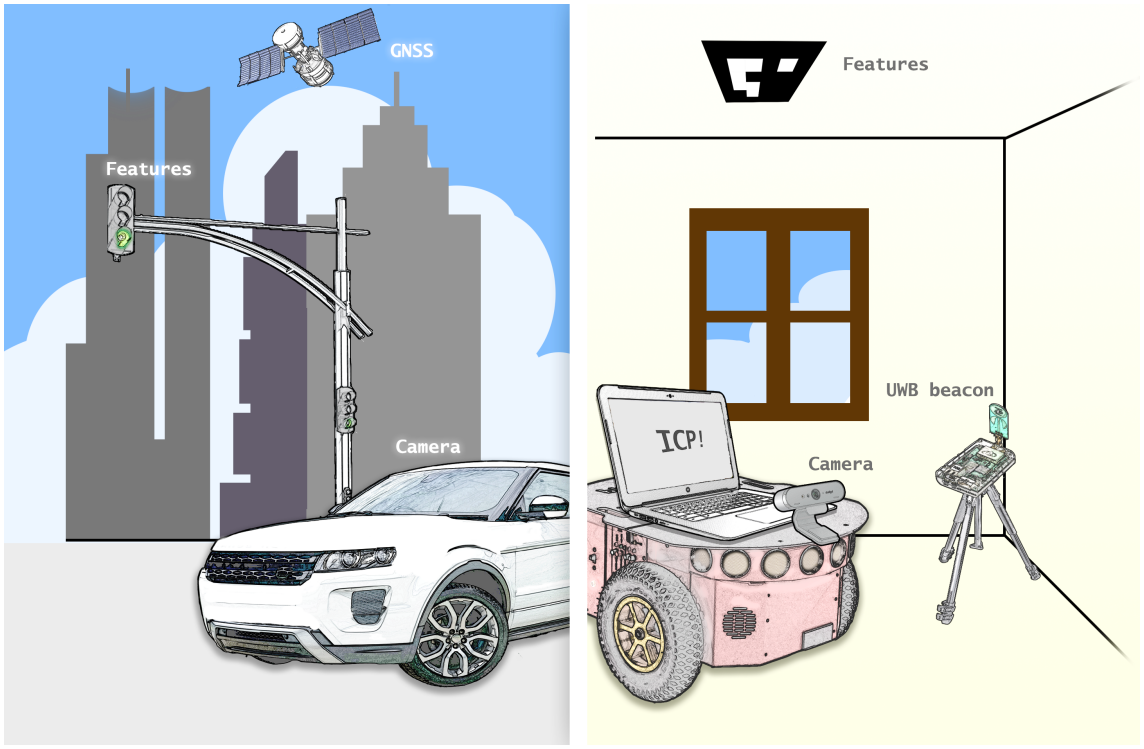


Figure 3.4: Illustration showing the real scenario and experimental setup analogy

The ICP is expected to be implemented on autonomous vehicles. The measurements for the vehicle would be obtained from a global positioning system like the GNSS. The relative distance to features measurements would be obtained form a camera placed in/on the vehicle. The features would be famous landmarks, prominent traffic control apparatuses and accessories like traffic lights or islands. For our depiction of the actual scenario we will have the Pioneer-D3X robots [17] representing the autonomous vehicles. The Time-Domain UWB [3] sensor beacons are used to simulate the GNSS as for both systems we would have to do some sort of trilateration and use radio waves. For the feature measurements we will use a web-cam [31] which can be compared to a camera in/on the autonomous vehicle. The features would be AprilTags [32] which are QR like tags placed on the ceiling. These tags will have a specific id which represents that feature number. The analogy of the real world scenario with our experiment is illustrated in Figure 3.4. The physical experimental setup used to test the functioning and implementation of the ICP algorithm is illustrated in Figure 3.5.

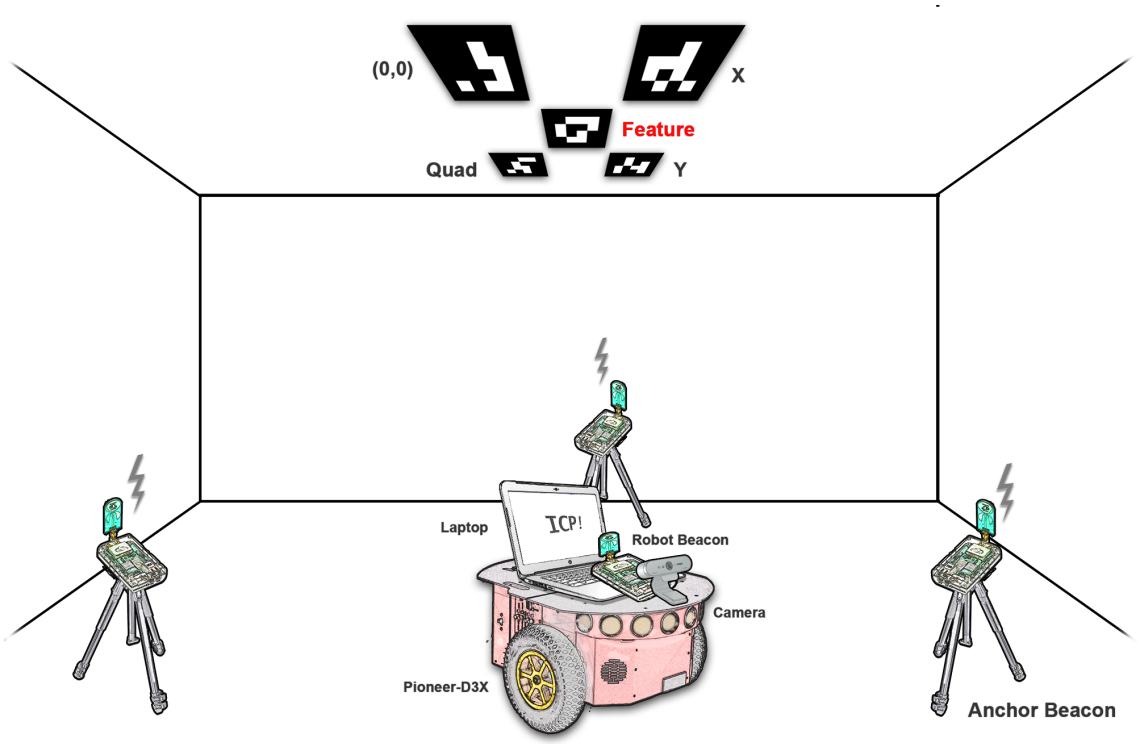


Figure 3.5: Test setup used for carrying out the robot experiments

3.2.2 ROS setup

As discussed earlier ROS is a very robust system to program robot software. For this particular reason ROS was used for setting up the robot system. The tests use two robots and each robot has a laptop. The robot laptop needs to be connected to the robot, the sensors and ROSCORE, the setup is shown in Figure 3.6. The UWB is connected to the laptop on the robot using the Ethernet connection, whereas the camera and robot itself is connected to the laptop on the robot using USB. The robots (laptops) are connected to the ROSCORE on the master computer using Wi-Fi. The nodes running on the laptops individually are the ROSARIA for the robot, Gulliview for the feature and robot orientation measurements, the service client for the UWB positioning system for the robot. The nodes running on the master computer where the ROSCORE is running is the robot controllers and the ICP node. The master computer receives the sensor measurements from all the robots, calculates the ICP estimates and computes the control action and sends the controller information to the robots to execute.

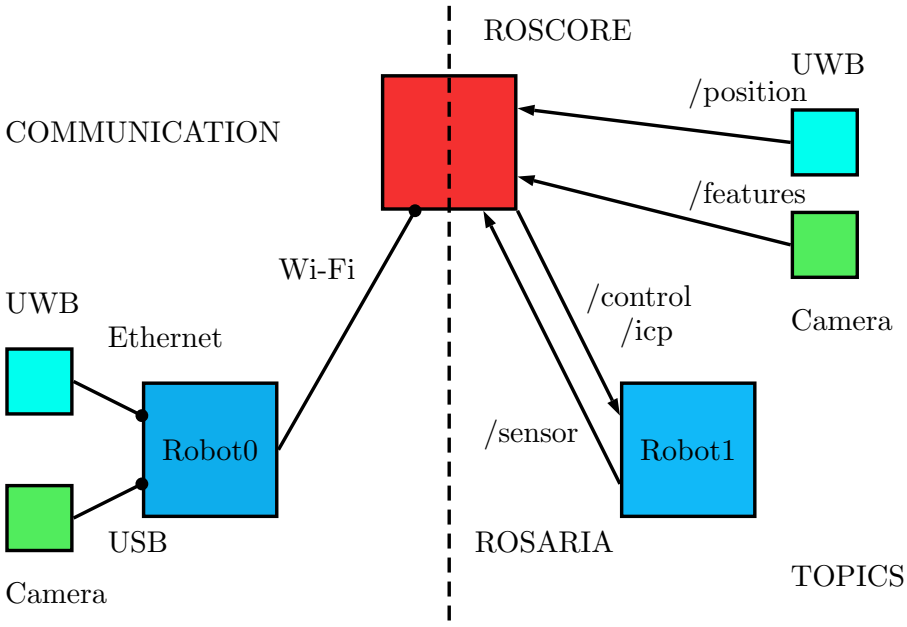


Figure 3.6: Communication and topic flow between the component and node interfaces

3.2.3 Reused ROS nodes

Some of the desired functionalities for the robot have already been implemented as ROS nodes either by the hardware providers or by efforts of other people. These nodes will be discussed in this Section.

3.2.3.1 ROSARIA

The Pioneer-D3X robots can interface with ROS using the ROS node called ROSARIA, this can be used to interface to most robots created by Adept MobileRobots. ROSARIA has the desired functions which help make use of the ARIA libraries for the control of the robots and also reading the sensor measurements. ROSARIA contains topics like *pose* which publish the state of the various sensors and topics like *cmd_vel* which act as input to the controller on the robot. For the robot we will mainly depend upon the Gulliview to get the orientation but there will be cases where the robot moves such that it does not see any tags. In such scenario we will use the encoder data in the robots through ROSARIA to do the update on orientation. More about ROSARIA and it's implementation can be read in [33]

3.2.3.2 Gulliview

As discussed we need a program to get the feature measurements (relative distance between the robot and the feature). Gulliview is a smart program which uses special tags called AprilTags placed at known locations to find the location of an unknown tag with respect to the tags placed. For doing this four tags are set up on the ceiling to define the coordinate system. The distance between the placed tags are measured. The Gulliview is then fed with this information and then the position of

the tags are converted to a desired unit of our own. After setting this up if we bring a fifth tag with an id other than the ones used for defining the coordinate axis, the Gulliview would then return the position of the fifth tag with respect to the four tags placed. The problem we are dealing with, which is finding the relative position of the robot with respect to the feature is achieved using the existing functionality of the Gulliview with a system which measures the camera position with respect to the tags placed on the ceiling. For achieving this the coordinate axis is set up on the roof and the unknown tag which represents the feature is also placed on the roof. The Gulliview program is used to find the position of the unknown tag/feature with respect to the coordinate axis and the code of the Gulliview was modified using the OpenCV libraries to return the camera position of the robot. The camera matrix for the camera is used to specify the perspective and barrel distortion effects and then the known information of the ceiling tags is used to do the computation of camera position real-time using the *SolvePnp* function. The camera position is then subtracted from the feature position to give the relative feature measurement. The Rotation Matrix obtained from the *SolvePnp* also helps to find the orientation of the robot which is also used for controlling the robot. The example of a tag family used for the Gulliview can be seen in Figure 3.7

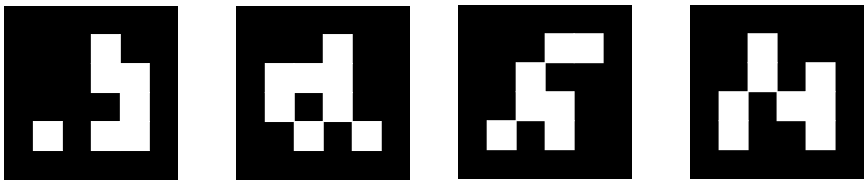


Figure 3.7: The tags defining the coordinate axis from the AprilTag family of 16h6

The implementation of the default system and the default code for the gulliview can be got from the repository [18].

3.2.3.3 Robot position service

This node was developed by an earlier group for their project [20]. The UWB setup uses three anchor beacons and a beacon on top of each robot. To get the position of the robot, the beacon on the top of the robot should ping the three anchor beacons separately. Each ping to the anchor beacons help to get the distance of the robot beacon from the anchors individually. The separate distance measurements is then used for trilateration to find the position. The pings to the beacons can only be done one at a time and to get the position of the robot we need to do three pings one after the other. This means one of the robot has to finish the series of three pings without interruption from the other robot. To ensure this the service-client approach was used, so that the client requested for the position. The request then would be initiated and serviced blocking other services in the meanwhile, this would mean that both robots would not randomly try pinging the beacons. Thus one robot could complete the series of the pings it required to get the position without errors. This kind of service routine can be easily done in ROS. The robots hence

have a service of their own which will initiate the positioning. The position service for one of the robots would be called by the ICP node depending on whether the other service is called or not.

3.2.4 Developed ROS nodes

This Section will describe the nodes discussed were developed from the ground up to complete the intended purpose.

3.2.4.1 ICP node

Once the robot and feature measurements are obtained from the positioning service and the Gulliview node of all the robots, we just need to carry out the ICP algorithm on the data to give the updated position of the robot. The algorithm is explained in Section 3.1. This updated position measurements can now be sent to the controller for computing the control inputs to the robots. For testing the robustness and efficiency of the algorithm we need to add different noise levels to the measurements and test the runs using the ICP estimate feedback to controllers. To achieve this a measurement mixer is used so that we can artificially add Gaussian noise to the measurements to simulate faulty measurements. The ICP node will ultimately publish the actual/simulated measurements, the ICP estimate and orientation of each robot to the robot controllers.

3.2.4.2 Robot controller node

After the execution of the ICP algorithm we would have better robot position estimates which can be feed back to the controllers. The controllers are a LQR controller as explained in Section 3.4.2. The controller has two parts, one being the state estimators which is the Kalman filter and second the LQR. The nodes also have an option to toggle between using the actual/simulated measurements or the ICP estimate for checking the performance of the controller using both measurements.

3.2.5 ROS node integration

The graphical representation of the final integration of the complete robot system nodes for two robots is shown in Figure 3.8. The ROS program called *rqt_graph* [34] was used to display the topic exchange between the various nodes. The *get_coord _serverN* is the service for finding the position for robot N. The node *gulliviewN* is the node responsible for the feature measurement and orientation for the robot N, this is done through the */positionN* topic. The node *RobotN* is the ROSARIA node on the robot N which uses the topics $/RobotN/cmd_{vel}$ and */RobotN/pose* to receive the control input values from the controller and send the robot orientation information from encoders to the ICP node respectively. The nodes described above are running on the robots itself. The remaining nodes mainly the ICP_Node and the $Robot\_ControllerN$ are running on the central computer which runs ROSCORE. The ICP_Node takes the feature measurements from the Gulliview and calls the service for the positioning to get the robots position. After the measurements are

done the ICP algorithm is run and the ICP belief of robots are sent to the controller nodes. The *Robot_ControllerN* node takes the measurements, computes the reference trajectory the robot is to move and sends the desired control inputs to the robots.

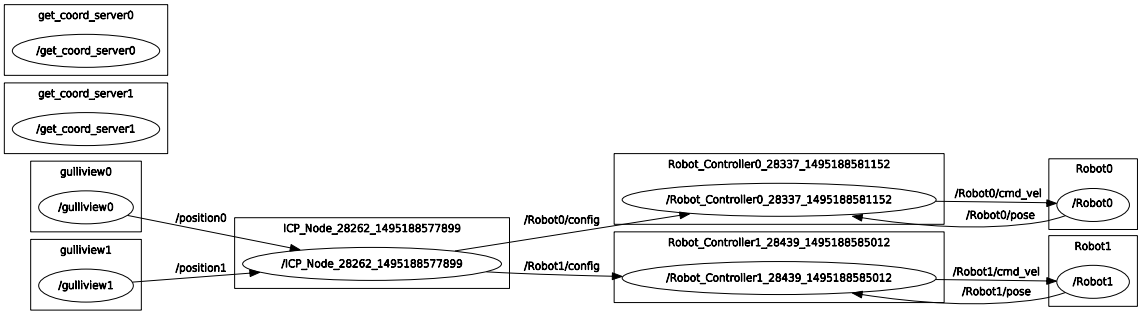


Figure 3.8: *rqt_graph* showing the various ROS nodes and the flow of topics between them

3.3 State estimator using Kalman filter

For the present robot systems we need an efficient way for finding the state estimates from the inaccurate measurements, also the robot presently relies on the camera, the Gulliview and the tags on the ceiling to get it's orientation. If the tags are not visible there is no way of getting the orientation. There would be problems controlling the robot, to alleviate this issue the wheel encoder of the robot is used to return the orientation when the tags is not visible. The problem of the wheel encoder though is that it is liable to drift if it runs for a long time due to slip. So the orientation for the robot is currently got from the combination of camera and the wheel encoders. The orientation is mainly got from the camera, whenever the tags are visible, but when the tags are not visible the increments from the wheel encoder at every time step are taken to do the update of the angle until the tags are visible again.
As the sensors can give inaccurate readings a filtering solution as explained in Section 2.4 was needed. As the dynamics of our system has already been described by the twist robot model in (2.9) - (2.10) we just need to use these matrices to implement the EKF. The prediction step for the Kalman is done using the system model whereas the update step is done in three stages. These update steps is part of the sensor fusion employed. There is some outlier detector for the measurements which checks the previous filtered value and the present measurement value and if it is below a certain threshold the update is done, else the prediction is used for that time step as the measurement obtained is probably an outlier. The first update step is for the position using the position reading from the UWB. The second update step is for orientation form the camera and encoder combination whereas the third update is also for the orientation using a generated measurement based on the previous filtered position and the present UWB position measurement which is not an outlier. The measurement is generated by

$$\hat{\theta}_{k|k-1} = \tan^{-1}\left(\frac{\hat{y}_{k-1} - y_{k|k-1}}{\hat{x}_{k-1} - x_{k|k-1}}\right). \tag{3.25}$$

To find the variance R of the noise for each sensor close to 1000 samples of measurements were taken and the covariance were calculated. For the UWB positioning system the covariance were found as

$$R_{\text{UWB}} = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy}^2 \end{bmatrix} = \begin{bmatrix} 6.18 \times 10^{-4} & 1.75 \times 10^{-4} \\ 1.75 \times 10^{-4} & 4.43 \times 10^{-4} \end{bmatrix}. \tag{3.26}$$

The variance for the orientation using the camera was found to be

$$R_{\theta 1} = \sigma_{\theta 1}^2 = 2.5 \times 10^{-8} \text{ rad}^2. \tag{3.27}$$

As the variance of the camera is greater than that of the wheel encoder and both sensor reading represent one measurement model, we can take the variance of the combined reading to be that of the camera.
The variance for the orientation using the generated measurement model described in (3.25) was found to be

$$R_{\theta 2} = \sigma_{\theta 2}^2 = 1.0 \text{ rad}^2. \tag{3.28}$$

It should be noted that this variance calculation was done with the robot stationary hence the reading is not that accurate as the position estimates fluctuate a lot giving false readings. This is also observed when the robot is stationary but making a rotation around it's Z-Axis. On the contrary if the robot was made to move linearly such that there is a change in position, the variance of the orientation readings got from this measurement model was really low and of the order of 10^{-4}, depending on if the robot was perfectly moving a straight line without any rotations. We will not dwell too deep into the fine tuning of this measurement model but will assume the worst case. This is because we get measurements with very low variance from the Gulliview, so we just need the measurement model to guide the update in the general direction in case the Gulliview is not functional due to the disappearance of the tags.
As the noise in the model is subjected to the previous input. We would want to assume that the controller will work really well such that trajectory which is created with the robot having constant linear and angular velocity will be followed perfectly. Hence can assume a small process noise and take it as an arbitrary value for experimentation for further tuning

$$Q = \begin{bmatrix} 8.0 \times 10^{-7} & 0 & 0 \\ 0 & 8.0 \times 10^{-7} & 0 \\ 0 & 0 & 1.0 \times 10^{-7} \end{bmatrix}. \tag{3.29}$$

3.4 Controller for Pioneer-D3X robots

In this Section we will describe the various aspects of the controller needed for the Pioneer-D3X. We will look into the different modes of operation of the trajectory planner and the control algorithm employed. The components of the controller and the way they are combined will also be discussed. A brief description of angle discontinuity will also be summarised.

3.4.1 Trajectory planning

For the purpose of controlling the robot through the desired trajectory with the desired angular and linear velocity we need to plan a trajectory. For planning the trajectory we need to decide the least possible sampling time. The Pioneer-D3X while using the UWB beacons have minimum sampling time of 0.1 s to get the position estimate, so for the trajectory a smaller sampling time of 0.05 s was chosen. The default trajectory desired to move for the robot is an elliptical trajectory, so that the robot could make rotations around the features when implementing the ICP algorithm. That way we can be certain that the robot see's the tags most of the time as the tags are really important for the ICP algorithm. As the robot is not positioned precisely at the trajectory every time the robot runs would be executed, we need to device a trajectory re-router which would help the robot merge into the elliptical trajectory no matter where the robot is.
This problem is handled in two ways, if the robot is positioned inside the elliptical trajectory the least distance possible is calculated to the elliptical trajectory. Once the least possible distance is calculated the robot is rotated to face that point then the robot moves forward. Once the robot reaches the desired point the robot is rotated to match the orientation of the trajectory then the robot starts the rotations. If the robot is outside the elliptical trajectory the robot tries to find the tangent to the elliptical trajectory and then selects the one which is in the direction of rotation of the robot. Once the tangent is selected the robot will rotate to match the tangent orientation and then the robot will move forward and merge into the elliptical trajectory. When the controller is started the location of the robot is checked and then a suitable re-routing is done and the total trajectory is returned for the controller to follow. The trajectory has three states, x, y, θ. Depending on when the measurement arrives the trajectory is checked on where the robot should be at that particular time and the three states for that time is returned to the controller.

3.4.2 LQR for Pioneer-D3X robots

For the purpose of controlling the robot we need a robust system which would be capable of handling the quick fluctuations of sensor measurements and still maintain the desired smooth reference trajectory without much error. The solution for this is using a LQR controller, as explained in Section 2.6, with the states estimated from a Kalman filter. The resulting controller would be robust enough to control systems disturbed by white additive noise optimally. The block diagram for the LQR Controller using a Kalman estimator is shown in Figure 3.9

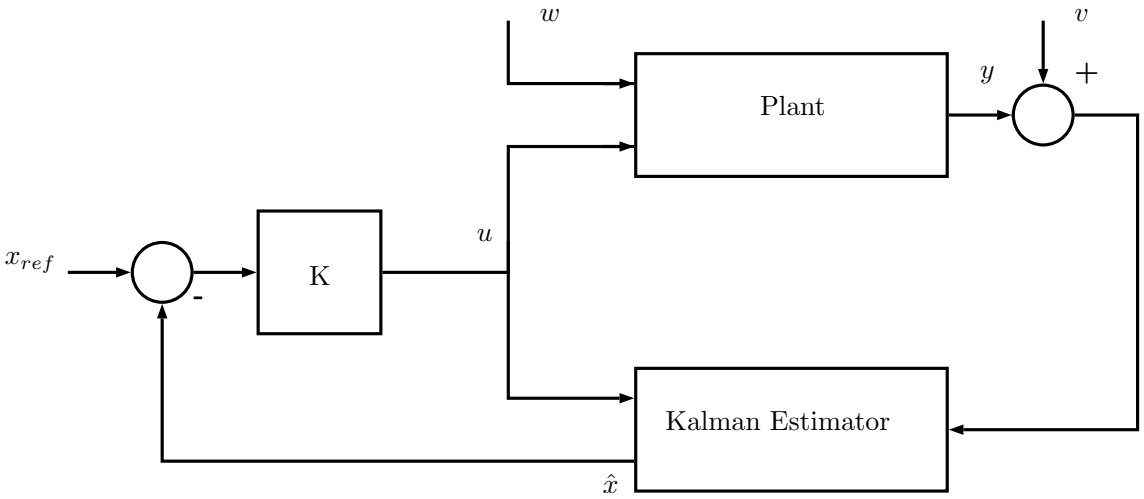


Figure 3.9: Block diagram showing the various components of the robot controller

The block K represents the controller gain from the LQR controller. The quantities w and v represents the process and measurement noise respectively, $\hat{x}$ is the estimated states from the Kalman Filter, x_{ref} is the reference trajectory the robot is expected to move in. We have already implemented the Kalman estimator in Section 3.3 and the trajectory.
From the Section 2.1.2 the dynamics of the system have already been defined which are given by (2.9) - (2.10) the weights for the states Q and the inputs R matrices were chosen as (these were finally concluded by running test runs)

$$Q = \begin{bmatrix} 300 & 0 & 0 \\ 0 & 300 & 0 \\ 0 & 0 & 200 \end{bmatrix}, R = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}. \tag{3.30}$$

Simulations were run using the controller parameters in MobileSim [35] for the Pioneer-D3X robot model. For the simulations state estimators was not needed as the simulation returned precise robot states without any noise. The scripting language used was Python and the results of the simulation are shown in Figures 4.7 - 4.8. For the actual run on the robot the Kalman filter as described in Section 3.3 was used, as the robot sensor measurements, unlike in the simulation, are prone to noise. The complete LQR controller as seen in Figure 3.9 was implemented and run on the robot which can be seen in Figure 4.9.

3.4.3 Angle discontinuity

During the simulations and actual run on the robots we do come across the problem of angle discontinuity. For our experiments we take $\theta \in [0, 2\pi]$. Even the angles obtained from the Gulliview is mapped from $\theta \in [-\pi, \pi]$ to $\theta \in [0, 2\pi]$. When the angle switches between 0 to 2π and vice versa there is a big discontinuity. We need to keep track of such discontinuities as when the robot is following the trajectory and it is crossing this discontinuity for e.g. the robot's orientation is close to 2π rad but the reference trajectory is 0 rad the controller will see this as a big difference in values even though the angle is really close. Similarly if the robot's orientation

is 0 rad and the reference trajectory is 2π rad we would have a similar issue. So a check is done to avoid such kind of discontinuities. The check would try to find such discontinuities and bring back the error difference to the actual one instead of the perceived discontinuity.

3.5 Implementation of ICP on Pioneer-D3X robots

The implementation of ICP is converted to a ROS node which access the camera of both robots to get the feature measurement and keeps pinging the UWB beacons to get the robots' positions. The ICP node then does it's estimate by running the complete set of iterations through the algorithm to compute the various messages to update the belief of features and vehicles. The ICP estimates are then sent to the controller which will carry out the control operations of trajectory planing and controller feedback back to the robots. For the purpose of simulations there is also the possibility of adding noise to the actual measurements using a mixer so that we get some simulated measurements which depict a bad reception scenario. The ICP ROS node has the functionality of activating and deactivating the ICP estimates so that the effect of switching on/off the ICP can be evaluated real-time. When the ICP is deactivated the ICP will only evaluate the Kalman estimate from the linear velocity model with measurement update without taking the feature information.
For the purpose of comparing the performance of the controller with just the Kalman filtering and the ICP algorithm we need a common test platform hence it would be better to use the common prediction model for the Kalman filter which is described in Section 3.1.1.1. Hence we will be using the CV model instead of the robot twist model in both cases when we have to make the runs to compare the performance of the two estimator algorithms. The reason for not using the twist model for the ICP case is because the ICP algorithm uses the constant velocity model for the prediction and using the estimate of the ICP into the prediction of the twist model would mean we are doing two predictions which could be avoided by using just the constant velocity. We in the process of choosing the constant velocity model will sacrifice the performance of the twist model (the twist model is a better model which closely represents the true robot model when compared to the CV model). Our aim being the comparison of the traditional Kalman filtering we do not mind the reduction in performance as the drop will be the same for both scenarios. The robot twist model computations are still done as they are used to predict the orientation the position estimates though will come from the velocity model.
For running the complete ICP setup on the robots and testing the effectiveness of the controller using the estimates got form the ICP, a noise was added to the position estimates got from the UWB for one of the robots which we want to move with $\sigma = 0.1$ m^2 on both x and y (this robot will represent the vehicle with faulty GNSS). For the test a robot was kept stationary observing the tags (this robot represents the vehicle with good GNSS reception). The faulty robot (the robot with noise added to the UWB position estimates) is made to circle around the tags on the ceiling where one of the tags is the feature. Two scenarios were tested where the ICP was either turned on or off while feeding the ICP/Kalman estimates to the LQR controller. The run using the ICP estimate as feedback to controller can

be seen in Figure 4.18 and the run with ICP deactivated with Kalman estimate as controller feedback is shown in Figure 4.19.
As explained in Section 2.1.4 not all components of the robot are placed exactly one over the other, which means that the position of each components of the robot does not coincide. We have the similar problem where the UWB beacon is not placed where the camera is. We thus need to correct for the camera's position so that is coincides with the beacon's position. This would help us get the feature measurements with respect to the beacon position as we want it to be instead of with respect to the camera. Accounting for this transformation hence helps to get the precise feature measurement, otherwise we will get strange offsets in the measurement during the robot runs due to the relative position between the camera and beacon position.

4

Results

This chapter will show the various results and plots obtained by carrying out the simulations and test runs proposed in the previous chapters. The inferences made by observing the results will also be discussed in detail in this chapter.

4.1 CRB results for ranging and position

The equations used to find CRB for the distance estimation in Section 2.5.1 was written in a Matlab script and simulations for the CRB with increasing distance was done as seen in Figure 4.1.

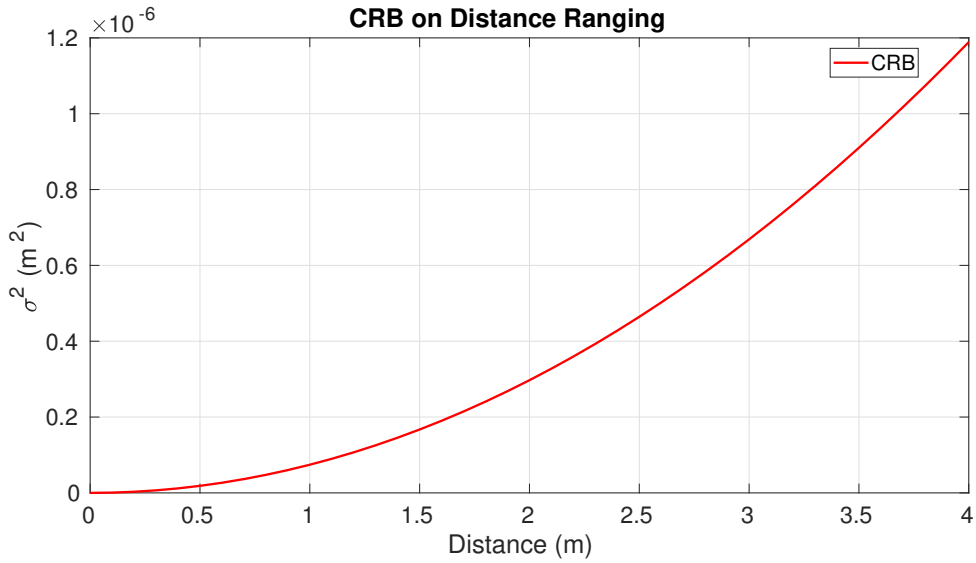


Figure 4.1: CRB for Ranging, shows the change in variance with respect to the distance between two pinging beacons

We can see from the plot that the CRB increases gradually as the distance between the beacons increases. This is expected as the radio signal strength reduces as we go away from the transmitter hence we can expect this kind of behaviour. For the purpose of finding the CRB on position estimates we will use the CRB of distance estimates used in Section 2.5.1. The transformations on the CRB for position is done using the equations described in Section 2.5.2. As before a Matlab script was created and simulations for the CRB were run depending on different anchor beacon configurations as shown later. For the basic test case scenario shown in

Figure 4.2 we can see how the the covariance curves for the variance in position changes throughout the test region.

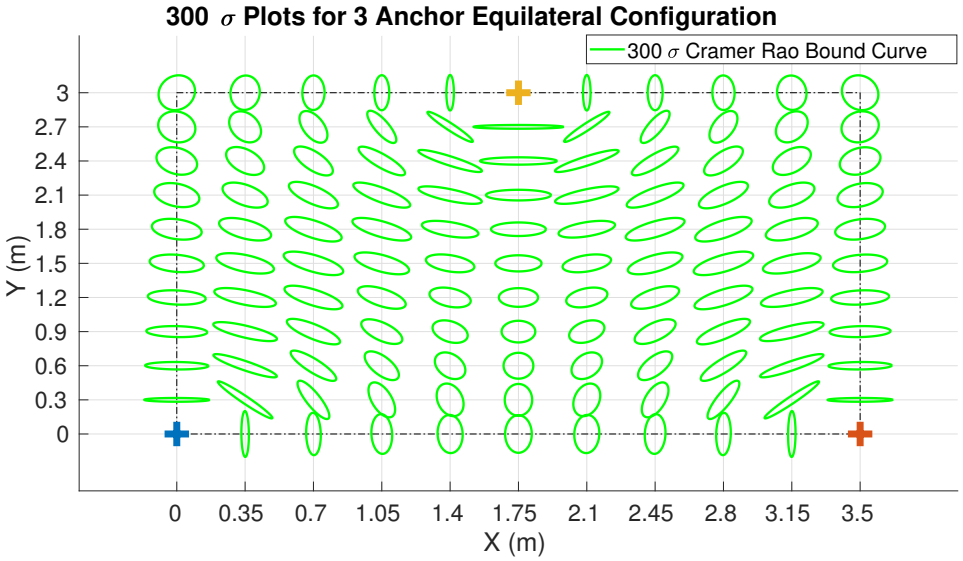


Figure 4.2: Plot showing the CRB curves for positioning with the beacons in the default setup with three anchor beacons

It can be seen that the region in the centre of the equilateral triangle has a more rounded curve which is attributed to this region being equidistant from the anchor beacons. Moving closer to one of the anchor beacons shows skewness in one of the dimensions. The influence of increasing the anchor beacon by one can be seen in Figure 4.3. Here we can see how the covariance curves in the central region becomes more round and with smaller covariances when compared to the three anchor beacon case.

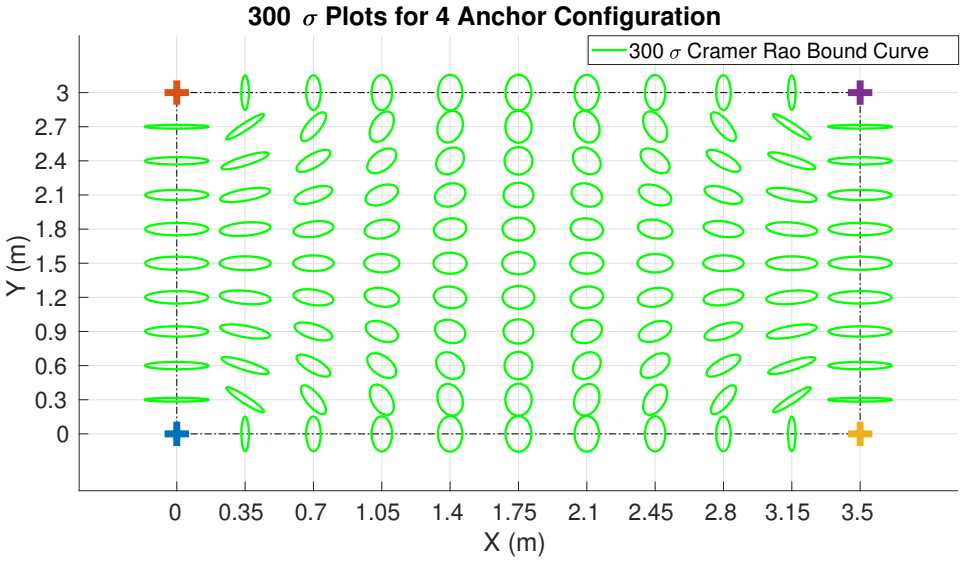


Figure 4.3: Plot showing the CRB curves for positioning with four anchor beacons

The beacon is then rearranged to a different setup as shown in Figure 4.4 here we can see how changing the configuration gives us really small covariances close to the centre of the triangle due to larger concentration of anchor beacons in this region.

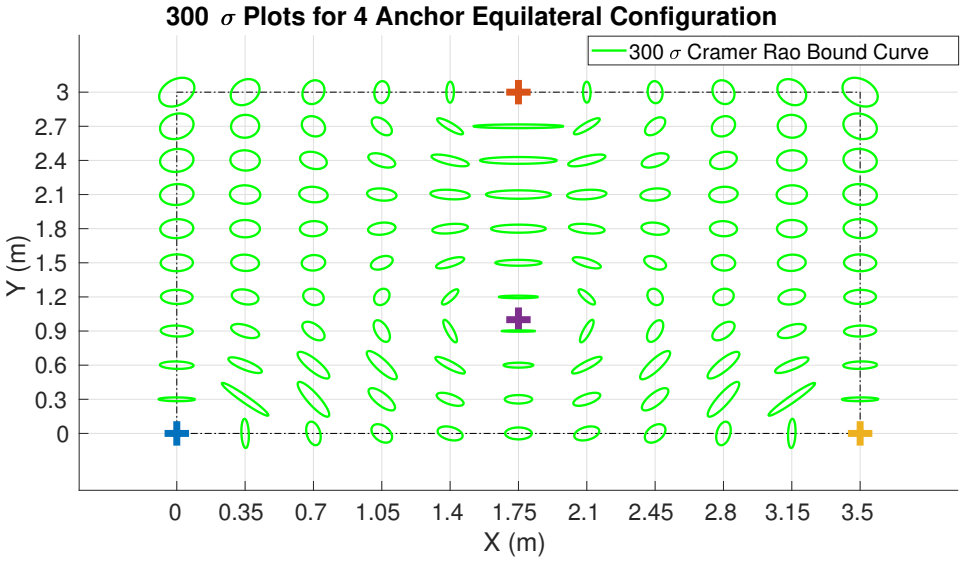


Figure 4.4: Plot showing the CRB curves for positioning with four anchors in an equilateral triangle setup

The anchor beacon count is then increased to five and arranged as shown in Figure 4.5 here we see more decrease in the covariance curves indicating better measurement estimates. From all the configuration we can see how adding more anchor beacons contribute to lesser CRB showing that the measurements would be more accurate with the introduction of more beacons. Also depending on the position where the measurement was taken the cross-covariance between the x and y position would change. Having asymmetric configurations also had an influence where the covariances were greater where the beacons were lesser in number. As our aim is to improve the position estimates using the ICP we will not try to improve it using more beacons. We will thus stick to the minimum anchor beacon setup needed which is three and the configuration as shown in the Figure 4.2.

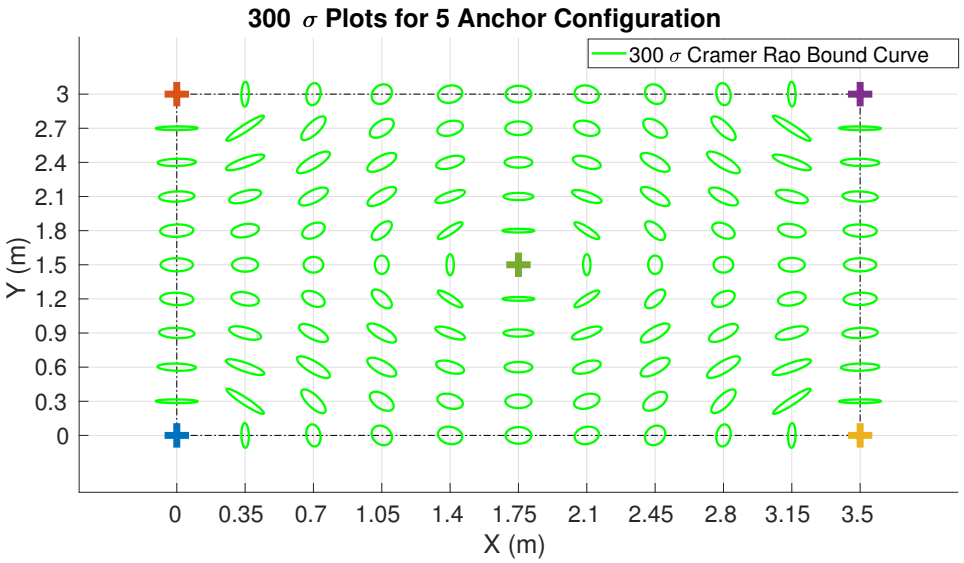


Figure 4.5: Plot showing the CRB curves for positioning with five anchor beacons

Tests were carried out by finding the position estimates using the trilateration method using the distance of the point from three anchor points. The tests were carried out from four points using the afore mentioned configuration. 1000 samples were taken from each position and the covariance matrix was calculated using these samples. Three of such runs were done and the average was taken to finally decide the covariance matrix for a given position. The variance matrix obtained at the four points and the CRB for the corresponding positions are plotted in the Figure 4.6. For the purpose of illustration the 600σ of CRB is plotted with the 6σ of the measurements obtained.
It can be seen from Figure 4.6 that the CRB is close to 10^2 times smaller than the actual covariances obtained form the UWB trilateration algorithm. This is expected as the CRB represents the least possible variance that can be achieved by an estimator. Also to do this calculation various assumptions of parameters were made e.g the medium is vacuum and so on these assumptions are not completely valid. The CRB calculated is for an almost ideal case and the test setup is not so.
What should be noted from the plots are that the covariances follow the general shape of the CRB. This is a good indication that the position estimates from the experiment, even though are very large compared to the CRB are still reliable.

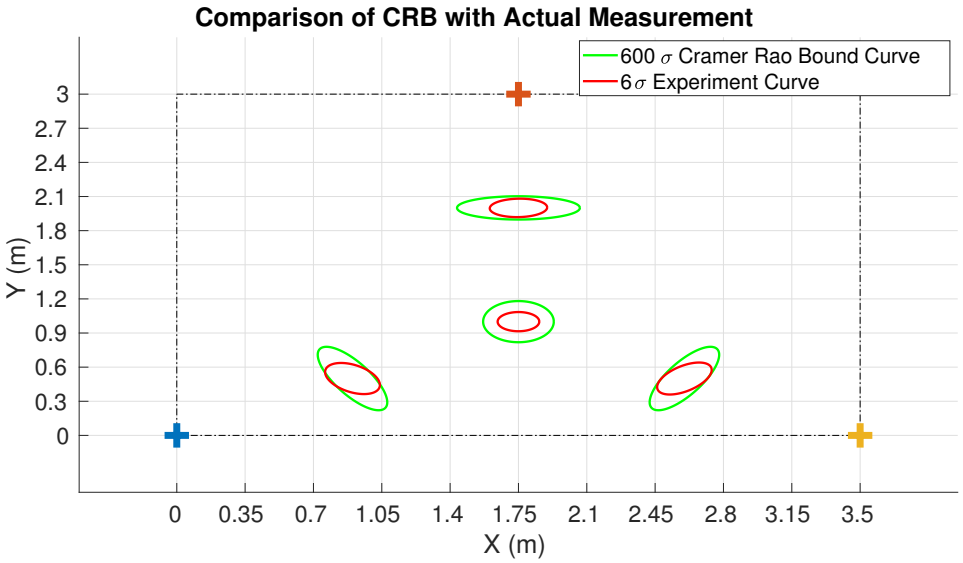


Figure 4.6: Comparison of CRB with actual measurement covariance

4.2 Plots of LQR

This Section will show the results, plots and the discussion of these from carrying out the LQR simulations. The simulations will cover the controller behaviour without any ICP implementation. The results from the software simulator and the actual robot run will be discused.

4.2.1 Simulation of LQR using MobileSim

The functionality of the trajectory planning and LQR controller is shown in Figures 4.7 - 4.8 the simulations were carried out on a ROS simulation software called MobileSim. As mentioned earlier in Section 3.4.2 we did not use any state estimators as the simulation returned perfect measurement values of states for the robot. In the first case the robot is placed at the origin, the desired trajectory to follow is a circular trajectory with radius 1 m with centre at point (1.6, 1.1). We can infer from the plot that the trajectory planner checks the initial position of the robot and tries to decide if the robot is inside the circle or not, as the robot is outside the circle it tries to find the tangent to the circle and creates the path. The trajectory is generated assuming the robot can move with linear velocity of 0.1 m/s and angular velocity of 0.1 rad/s.
For the second case the circular trajectory has the same radius of 1 m with the centre at the origin the robot is also placed at the origin. When the trajectory planner is called it realises that the robot is inside the trajectory. It finds the closest point on the circular trajectory and then creates the θ reference trajectory to face that point and then creates the position reference to move to that point.
We can infer from the plots that the simulations gave accurate reference tracking on all the reference states of x, y and θ. The Trajectory planner seems to do its intended purpose in both cases where the robot starts from the middle and when the

robot starts from the outside of the circular trajectory. The simulations do perform as expected. Now with the controller working as intended in the simulations we can proceed to implement it on the actual robot.

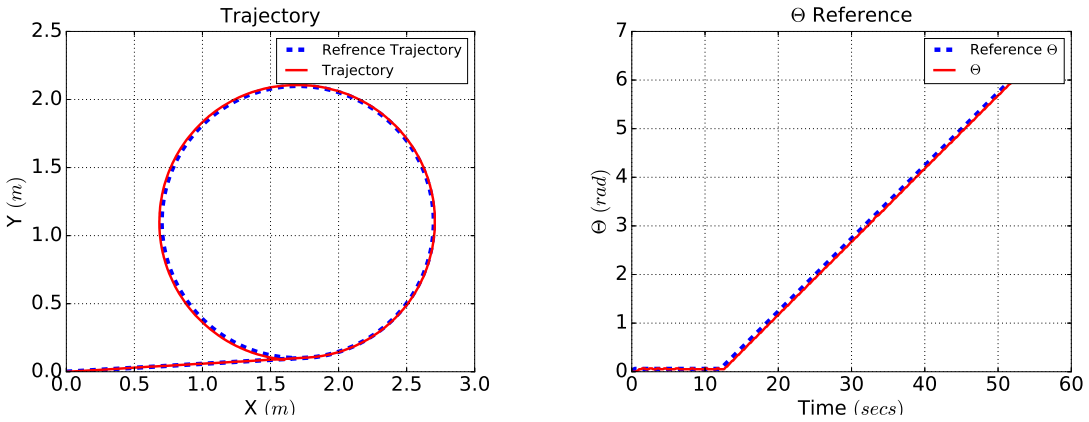


Figure 4.7: Trajectory tracking simulation with start from outside circular trajectory

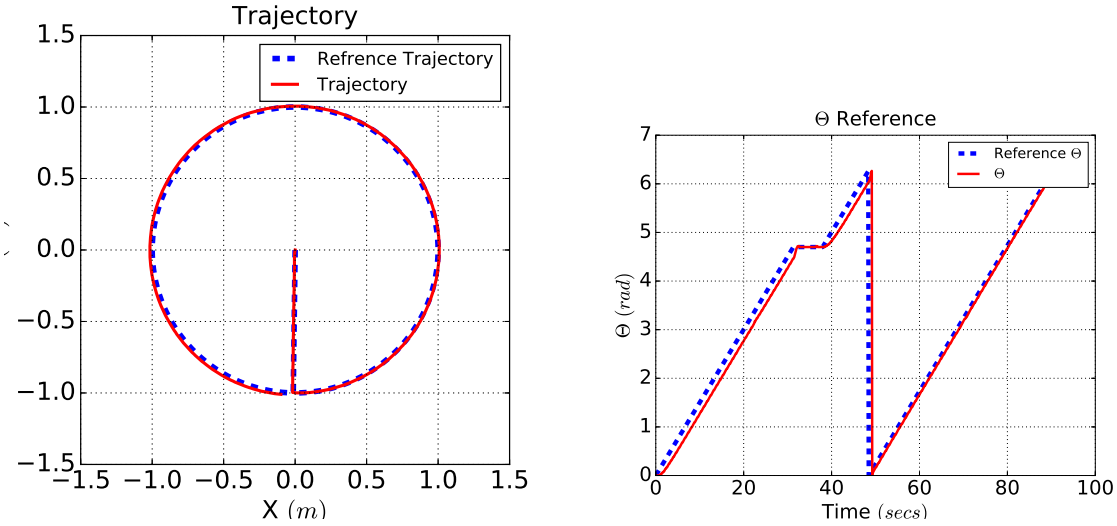


Figure 4.8: Trajectory tracking simulation with start from inside circular trajectory

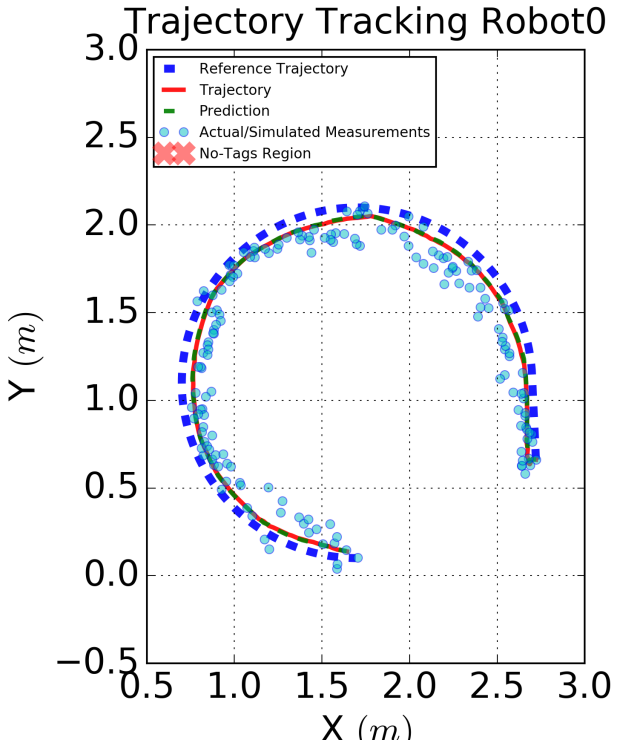
4.2.2 Trajectory tracking on Pioneer-D3X robots

The first few tests for the robot using the controller from the simulations were not successful as unlike in the simulations the measurements in the actual scenario kept fluctuating due to noise levels and irregular position ping timings. The uncertainties in the measurement caused the robot to move in sharp steps with very poor trajectory tracking. The need for filtering is quite obvious form the initial test runs. The Kalman filter with the robot twist model was implemented to filter the sensor measurements and get better estimates on the position and the orientation. Multiple runs were executed to tune the parameters of the LQR. Finally a good enough

weights were found as

$$Q = \begin{bmatrix} 375 & 0 & 0 \\ 0 & 375 & 0 \\ 0 & 0 & 180 \end{bmatrix}, \; R = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}. \tag{4.1}$$

For the final tests the previous problems faced during the run of the robot using the LQR controller has been alleviated with the introduction of the Kalman filtering and sensor fusion, the robot now follows the trajectory smoothly without sudden jerks, the trajectory followed is a bit offseted from the actual trajectory this is partly because of lower weights on the controller than the previous case. The gain could be further increased for compensating for the offset but due to the irregular position estimate got due to delayed pings between the beacons, the performance is not stable over multiple runs. Hence with keeping the stability of performance of the controller in mind over multiple runs the present weights are satisfactory. Other than the slight offset form the actual trajectory we can observe appropriate tracking on the position and the orientation as can be seen from Figure 4.9



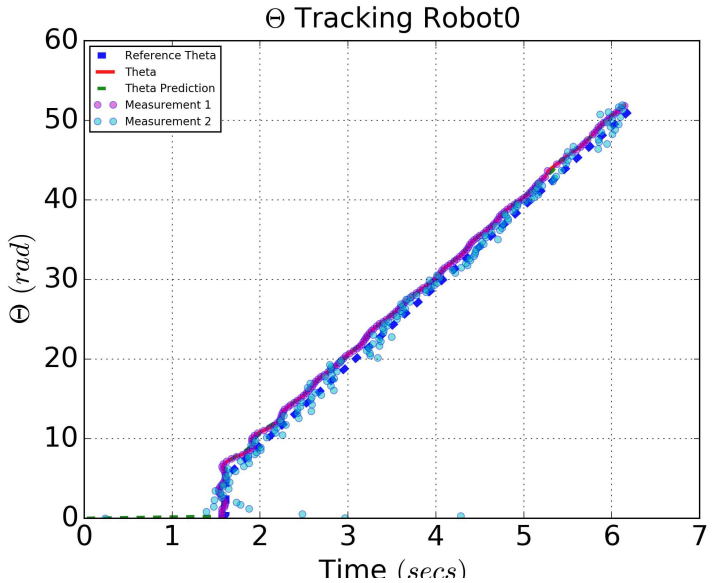


Figure 4.9: Trajectory tracking on actual robot with Kalman filtering and sensor fusion

4.3 Performance evaluation of ICP using synthetic measurements

In this subsection the results generated during simulations are presented. Firstly the error graphs generated from multiple runs of the algorithms on a randomly generated trajectory are presented. Later, the trajectory is fixed to examine the behaviour of the filter in 2-D plane.

4.3.1 Randomly generated trajectory

To verify the performance of the algorithm error plots are generated during the simulations. The algorithm is run multiple times, each time on a newly generated random trajectory and measurement generated around this trajectory. The errors of vehicle position are found by subtracting the estimated position vectors with the true trajectory. Further on the errors are squared and the x and y terms are summed together to represent the overall accuracy. The mean over all runs is then taken and the square root of it is presented in a graph. It is important to mention that the covariance matrices are matched with the actual covariance values of the trajectory and the measurement noises.
Firstly the most interesting case where first vehicle's position measurement is good ($\sigma_1 = 1$ m), while the second ones is poor ($\sigma_2 = 4$ m) is examined. The noise intensity of the V2F measurement is set to a value of $\sigma_{\text{V2F}} = 1$ m. This case represents the scenario where one vehicle enters e.g. an urban canyon and the GNSS measurements are affected by shadowing and multipath propagation. It is where the help from other vehicles which not yet suffer from these problems is most valuable. The case with two vehicles and one to four features is examined.

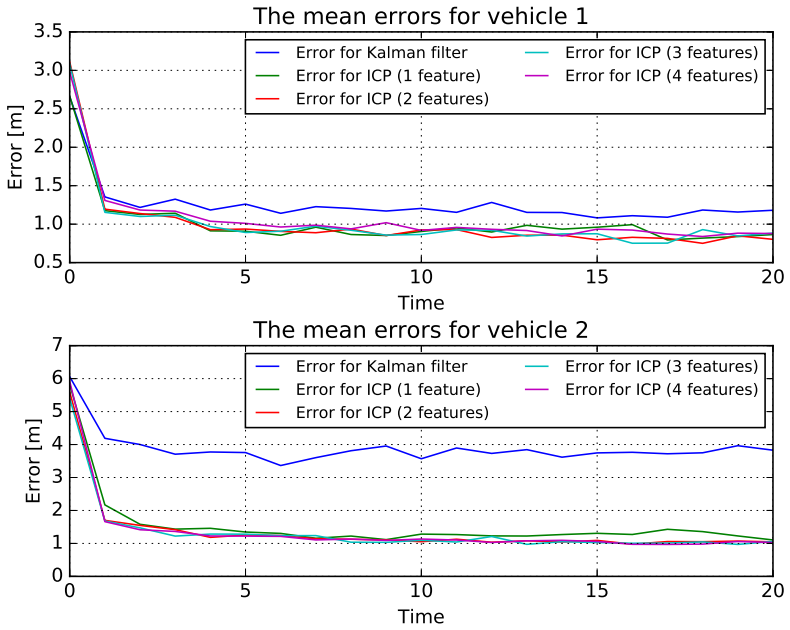


Figure 4.10: The plot of the estimation error acquired from 100 runs, plotted over time (s), $\sigma_1 = 1$ m, $\sigma_2 = 4$ m, $\sigma_{\text{V2F}} = 1$ m

In this case it is obvious that the ICP improves the position estimates of the vehicles. The Kalman filter relying on the GNSS measurements and being decoupled for the two vehicles result in an error of approximately 1.3 m for the vehicle 1 and approximately 4 m for vehicle 2. The error of the ICP converges instead to 1 m for vehicle 1 and somewhat above 1 m for vehicle 2 for all features. This behaviour is expected. Since the V2F measurements are good the positioning of the vehicles with respect to each other will be accurate. This leads to the belief errors being close to each other for both vehicles. It is also logical that the vehicle with bad GNSS measurement will gain the most from the ICP algorithm since its belief is much more uncertain and the other vehicle will be able to affect its belief stronger. Even the first vehicle will benefit from the algorithm since there is some poor, but still helpful information being passed from the vehicle 2. Since the feature measurement are accurate the increased number of features will not lead to much gain in the certainty of the estimator.

Following the same case but increasing the V2F measurement covariance to $\sigma_{\text{V2F}} = 5$ m Figure 4.11 is generated.

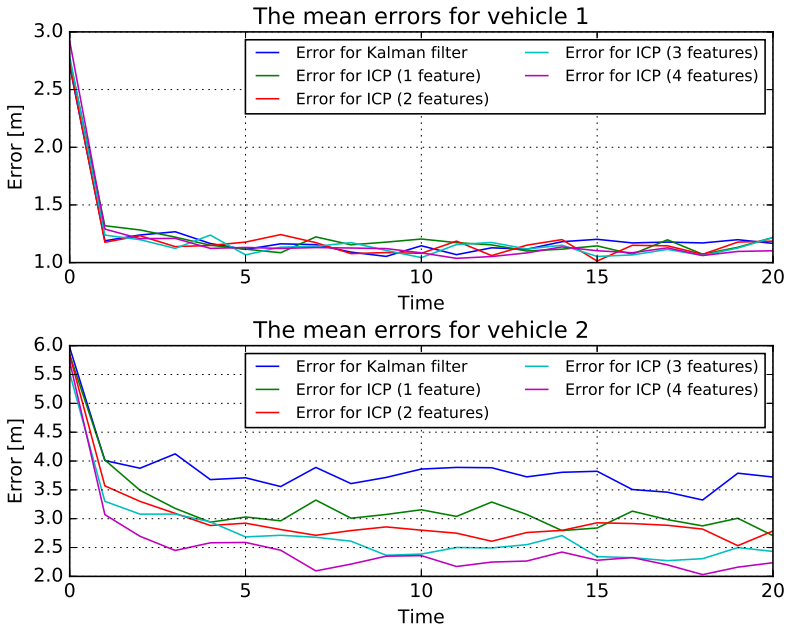


Figure 4.11: The plot of the estimation error acquired from 100 runs, plotted over time (s), $\sigma_1 = 1$ m, $\sigma_2 = 4$ m, $\sigma_{\text{V2F}} = 5$ m

As it can be seen in Figure 4.11 the ICP algorithm will not improve the position estimation of vehicle 1 noticeably. This is because of the uncertainty of the belief of the second vehicle combined with the uncertainty of the feature measurement will make the information passed have a limited accuracy and will therefore be of lesser value for the algorithm. This will result in ICP output being very alike the pure Kalman filtered output. However the error for the second vehicle can be decreased significantly. Here, unlike the previous case, the effect of bigger feature count can be seen clearly. Since only a feeble conclusion can be done by the algorithm about the other vehicles position using only one feature, the increase in the number of feature measurements will be of great help.

Another interesting case can be done by examining the effect of the ICP on two vehicles both having a poor position measurement. This can be seen in both Figure 4.12 and 4.13 presenting the estimation error of ICP with V2F measurements of low respective high variance.

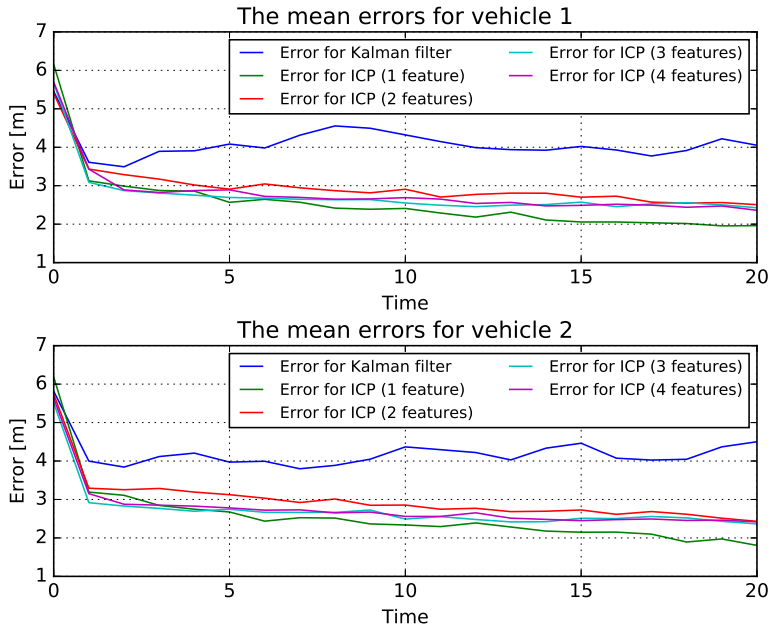


Figure 4.12: The plot of the estimation error acquired from 100 runs, plotted over time (s), $\sigma_1 = 4$ m, $\sigma_2 = 4$ m, $\sigma_{\text{V2F}} = 1$ m

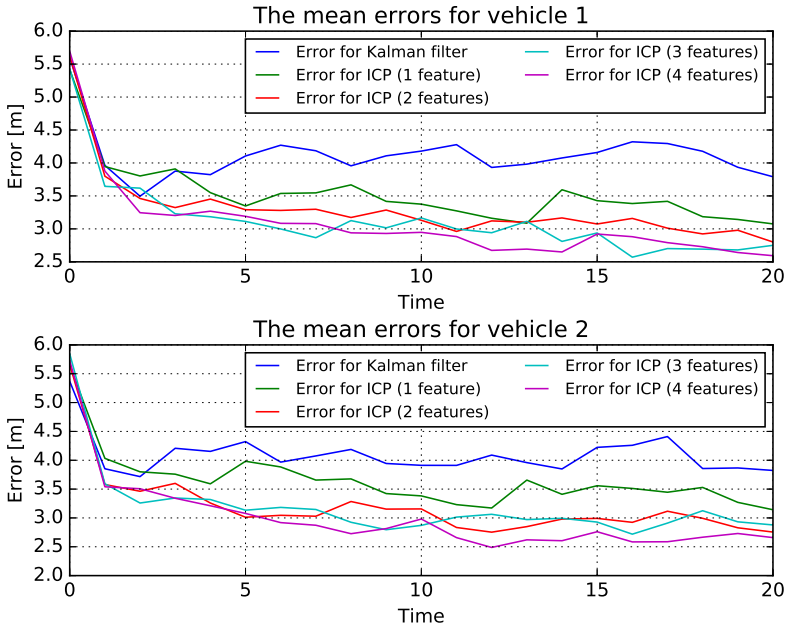


Figure 4.13: The plot of the estimation error acquired from 100 runs, plotted over time (s), $\sigma_1 = 4$ m, $\sigma_2 = 4$ m, $\sigma_{\text{V2F}} = 5$ m

Figure 4.13 clearly shows that even for two vehicles with bad measurements ($\sigma_1 = 4$ m) and bad feature measurements ($\sigma_{\text{V2F}} = 5$ m) the ICP algorithm increases the accuracy significantly. The error for the Kalman filter converges to a value of 4 m

while the ICP with four features converges to 2.5 m. It can also be noticed that less feature measurements in the ICP results in a smaller improvement of accuracy. This because of the low confidence of the V2F measurements and thereby more information needed to improve the estimates.

Not exactly the same behaviour is presented in Figure 4.12. ICP, as expected, increases the accuracy but the feature count does not affect it as it does in Figure 4.13. First of all, since the accuracy of the feature measurement is very good compared to the vehicles, only one feature measurement will provide a confident correction of the estimate. Further increase in the feature count may actually result in overconfidence resulting in feature estimates converging to a wrong value and lead to an offset in the vehicles position estimate.

4.3.2 Fixed trajectory

Next step of verification is applying the algorithm on measurement generated around a fixed trajectory of a certain shape. The measurement noise is now also scaled more appropriately to match the setup with the UWB beacons and Gulliview described earlier. The good measurement will therefore be position measurement with standard deviation of 1 cm and the bad one with 10 cm. The number of features is set to one.

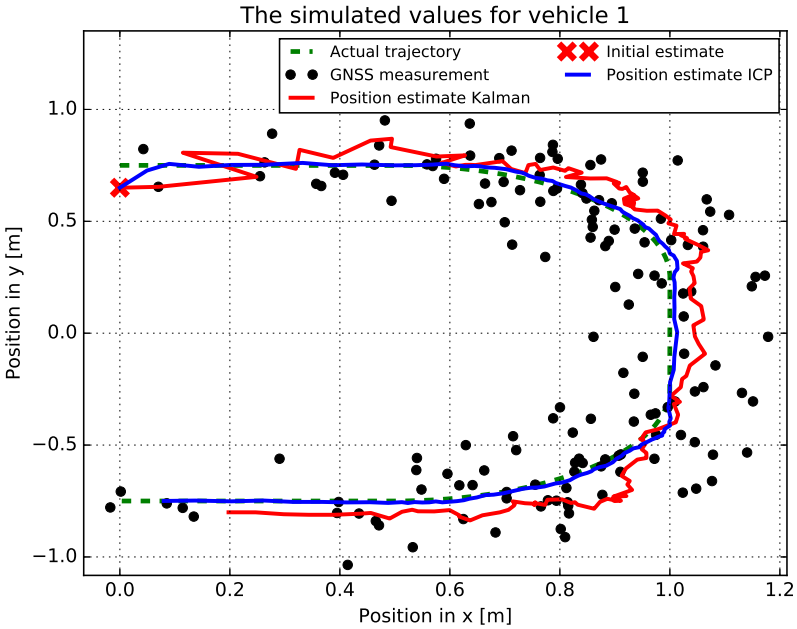


Figure 4.14: The ICP and Kalman estimates and the actual trajectory of vehicle 1, $\sigma_1 = 0.1$ m, $\sigma_2 = 0.01$ m, $\sigma_{\text{V2F}} = 0.01$ m

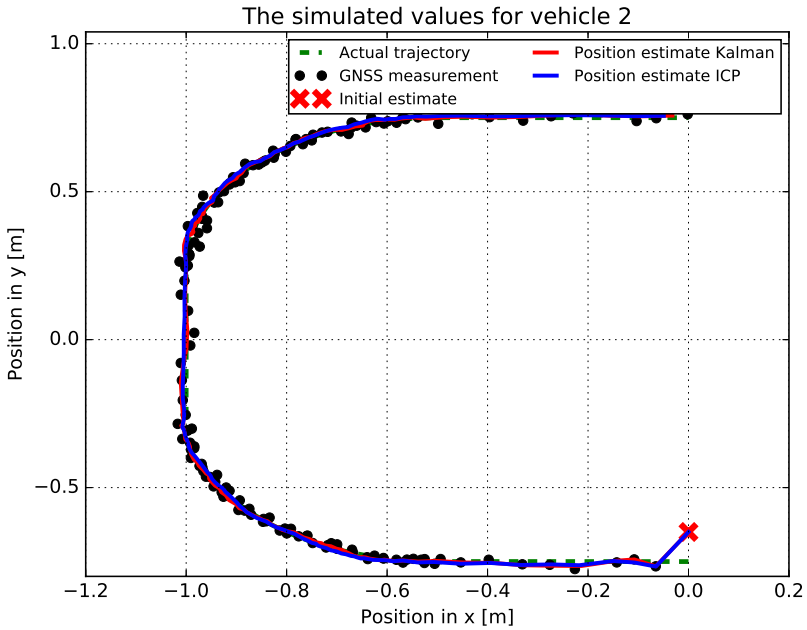


Figure 4.15: The ICP and Kalman estimates and the actual trajectory of vehicle 2, $\sigma_1 = 0.1$ m, $\sigma_2 = 0.01$ m, $\sigma_{\text{V2F}} = 0.01$ m

The above Figures show clearly the improvement in the estimations of the vehicle 1 using ICP (seen also in Figure 4.10). The red line representing the Kalman estimate, despite tuning, has a visible offset from the real trajectory during the whole simulation. For the vehicle 1 the trajectory's ICP estimate, as before, is very close to the Kalman estimate.

The earlier results shown in Figure 4.12 are also be seen while examining the trajectories, as in Figures 4.16 and 4.17. That is even when both vehicles have poor position readings, a good feature measurement can improve their positioning significantly. It is obvious that the ICP estimates represents the true trajectory much better then the Kalman filter itself. This affect is not as big as in previous case. This is because of none of the vehicles is really certain about its position. It can also be seen that it takes longer for the vehicle 1 to converge to the actual trajectory. This probably again because of insufficient information passed from vehicle 2. Also a bigger error can be noticed around the curves of the trajectory. This is because the initial beliefs in the ICP algorithm (since the CV model) will take time to react when some acceleration is present. That will lead to communicating faulty beliefs to the other vehicle and in such a way accumulating the error.

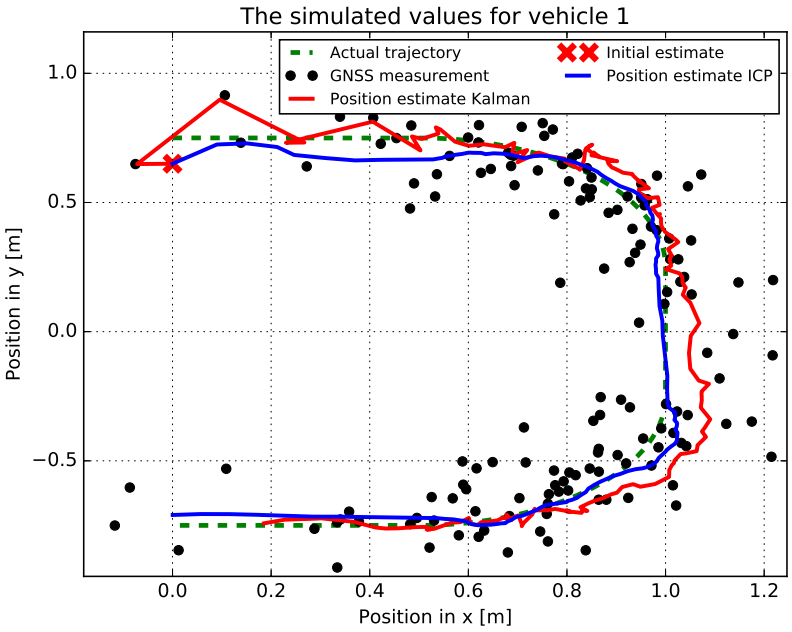


Figure 4.16: The ICP and Kalman estimates and the actual trajectory of vehicle 1, $\sigma_1 = 0.1$ m, $\sigma_2 = 0.1$ m, $\sigma_{\text{V2F}} = 0.01$ m

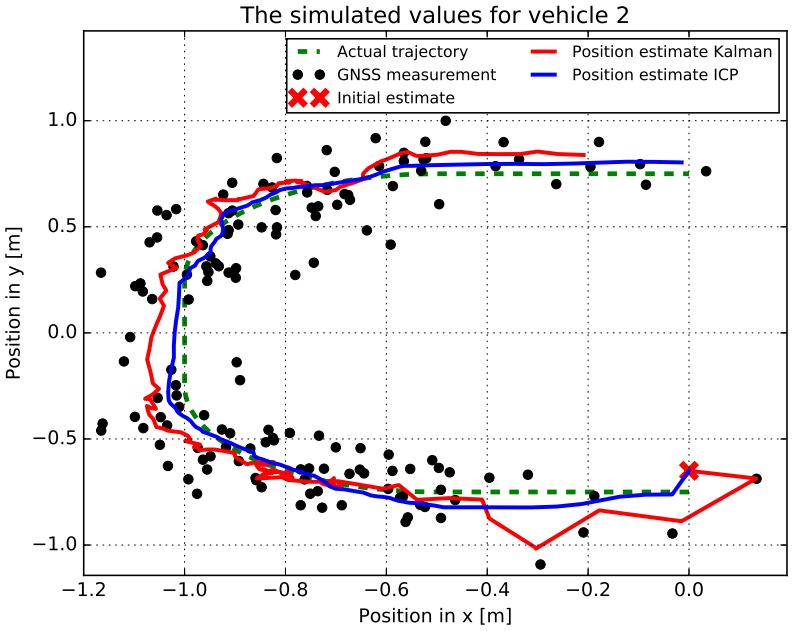


Figure 4.17: The ICP and Kalman estimates and the actual trajectory of vehicle 2, $\sigma_1 = 0.1$ m, $\sigma_2 = 0.1$ m, $\sigma_{\text{V2F}} = 0.01$ m

4.4 Evaluation of ICP on Pioneer-D3X robots

For the implementation of ICP on the robot, as discussed before in Section 3.5, we would use the constant velocity model instead of the robot twist model for the Kalman filter. As the model is different form the previously implemented controller we need to re-tune the controller parameters as the runs using the previous controller weights was not functional. The robot twist model is a better representation of the robot than the CV model hence this is to be expected. During the run using the CV model it was observed that the robot was overextending the trajectory every time causing the robot to vigorously move forward and backwards. Extensive runs were done using the CV model until a good enough weights were found as

$$Q = \begin{bmatrix} 30 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 100 \end{bmatrix}, R = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}. \tag{4.2}$$

Whereas the covariances on the process noise for the CV model for the robots were kept as:

$$P = \begin{bmatrix} 10^{-4} & 0 \\ 0 & 10^{-4} \end{bmatrix}. \tag{4.3}$$

We intend to test the working of ICP with the Kalman filter so the two scenarios were tested out. The test scenario were setup as explained in Section 3.5, Figure 4.18 shows the ICP estimates got form the ICP node fed into the controller. As can be seen in the plots the simulated measurements for position is spread all around the reference trajectory (the noise added to both the x and y estimates of the UWB measurement is $\sigma^2 = 0.1$ m^2). The noisy measurements when fed through the ICP gives really good estimates of the robot position and the controller as expected, performs really well. We can see that the robot follow the trajectory precisely with very less error.

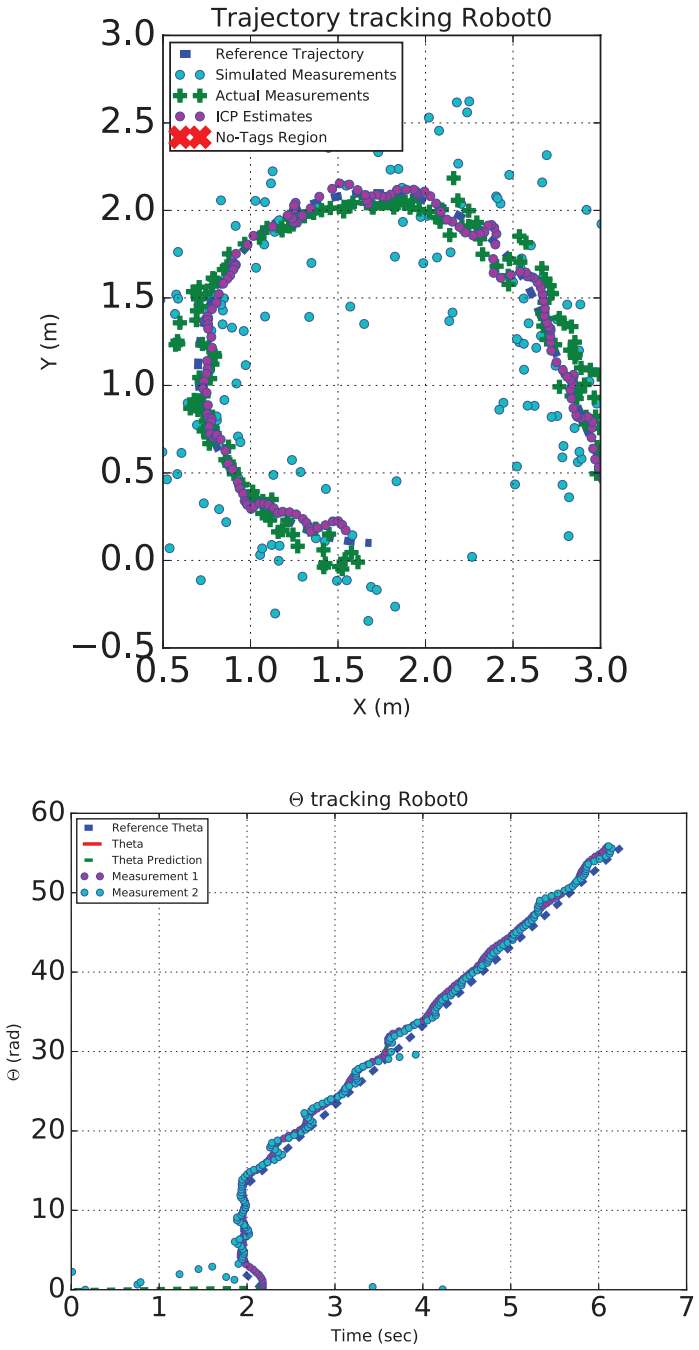


Figure 4.18: ICP measurement feedback to controller with noise on UWB measurements, $\sigma_x^2 = 0.1\ m^2$, $\sigma_y^2 = 0.1\ m^2$. The top figure shows the reference tracking of the position while the bottom shows the tracking of the orientation

For the second scenario the robot controller is fed with the Kalman estimates with the ICP deactivated. It can be inferred from Figure 4.19 that the estimates is too far away from the actual position and keeps jumping a lot around the actual position as the robot moves this causes the robot to move in a very choppy manner. The robot sometimes has to accelerate and sometimes decelerate to reach the position estimates.

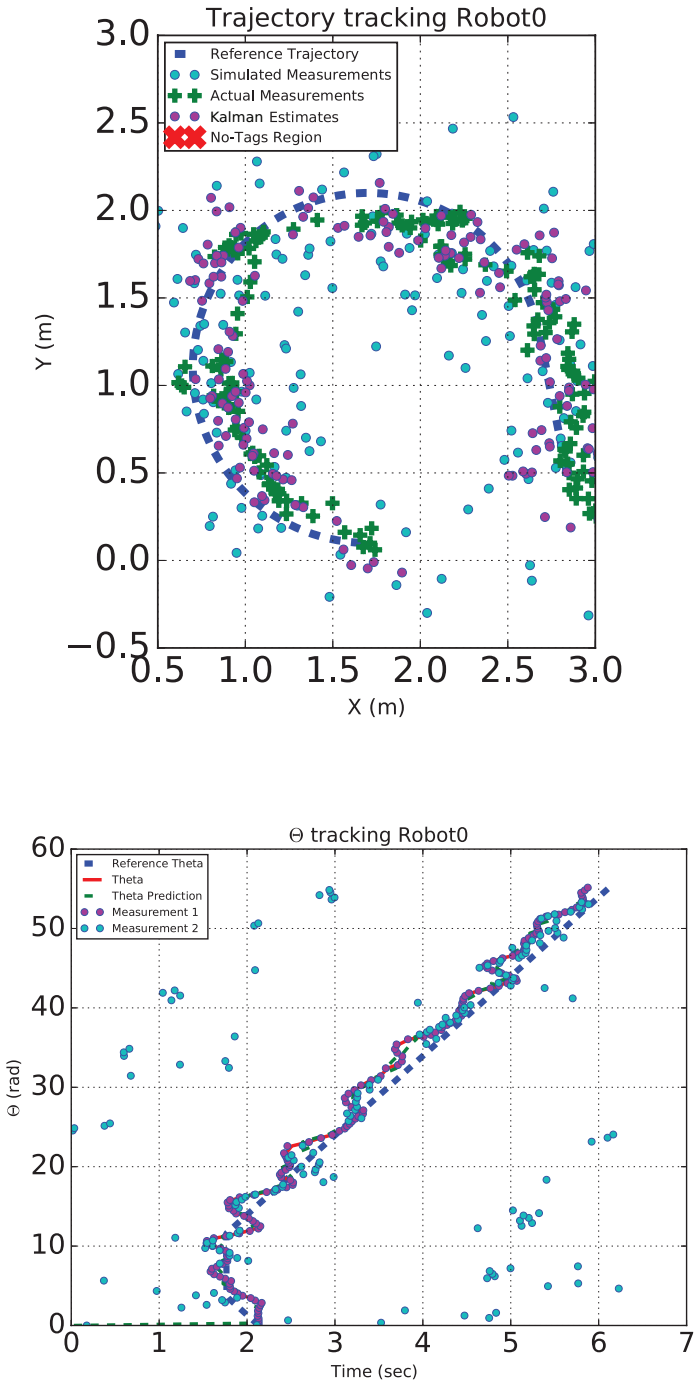


Figure 4.19: Kalman Estimate feedback to controller with noise on UWB measurements, $\sigma_x^2 = 0.1\ m^2$, $\sigma_y^2 = 0.1\ m^2$. The top figure shows the reference tracking of the position while the bottom shows the tracking of the orientation

We can see from the scenarios that the ICP gives really good estimates of the position which when fed to the controller gives really good controller behaviour. The importance of the ICP hence can be seen if it were to be ported to a real scale autonomous car application. In Figure 4.20 the error in distance generated while feeding back the ICP estimates and the Kalman estimates in the above test scenarios

can be seen. We notice the error in the controller tracking with ICP estimates as feedback is less than half of what it is while using Kalman estimates as feedback. In our test demo we have used only two robots and one feature. Using multiple vehicles and features will improve the algorithms efficiency and reliability by a substantial amount.

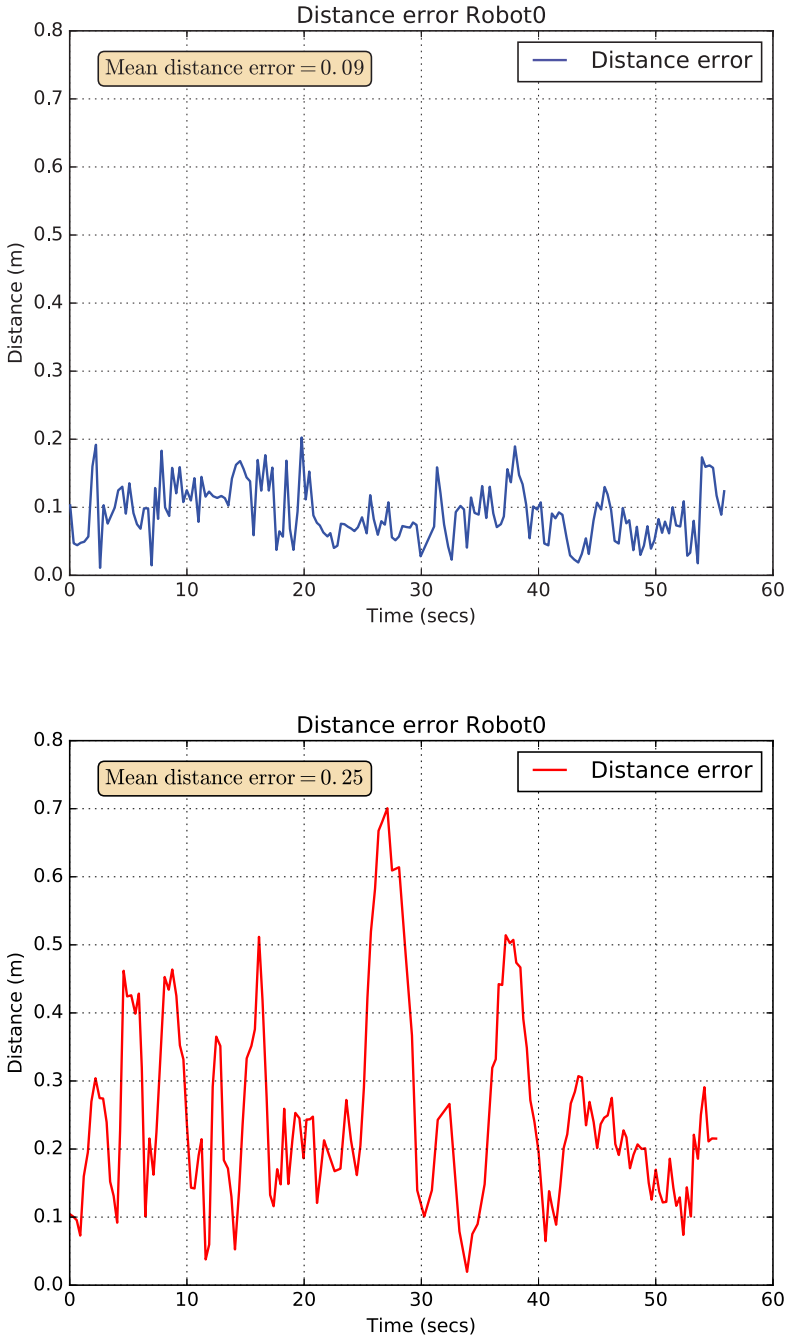


Figure 4.20: Error in distance from actual trajectory after control by using A) ICP estimate feedback (Top) and B) Kalman estimate feedback (Bottom), $\sigma_x^2 = 0.1$ m^2, $\sigma_y^2 = 0.1$ m^2

5

Conclusion

According to the results presented in the previous chapter some conclusions can be drawn. These are presented hereafter. Also some possible improvements of the ICP and its evaluation are proposed.

5.1 Evaluation

Conclusion can be done that using Implicit Cooperative Positioning algorithm can improve the accuracy of positioning significantly. Simulations show that cooperation made by agreeing on feature position by multiple vehicles increase the amount of information available for a vehicle about its own state. Compared to a Kalman filter, if tuned properly, ICP should always improve the positioning, independent of the noise levels present in the different measurement. The experiments on robots also show clear improvement compared to only the Kalman filter.

The relative gain of accuracy using ICP is hard to determine since it is dependent on the noise levels of the measurements and the number of vehicles and features present in the system. However, according to the simulations, for two vehicles with the same magnitude of noise on GNSS measurements, the error of position estimates can be reduced approximately by 50 % if the V2F measurement is very accurate. Also, if ICP is applied for multiple vehicles, where one of them has a very small error on GNSS measurement and the V2F measurements are very accurate, the estimates of the vehicles with bad GNSS positioning will converge to the estimates of the first vehicle. This can lead to a great improvement of positioning accuracy, and this is probably the case where ICP is most beneficial to use.

Only negative aspect was noticed during simulations where many features with a low noise intensity on V2F measurements were used for running the simulation. In some cases the feature beliefs converged to and stabilised around wrong values. This probably due to overconfidence on feature beliefs. The affect of the overconfidence was reduced by applying a process noise on feature belief between time iterations, letting the algorithm assume feature can change its position and hence letting the previous value of the state have less effect on the new feature belief.

5.2 Future work and improvements

The system that has been implemented is not completely decentralised, and therefore is not completely representing the system the algorithm is meant for. To fully test the system of truly independent vehicles running ICP, the system should be decentralised. The algorithm should then be run on separate machines connected by V2V links and running consensus over the feature beliefs using the information available from other cars. This problem has been left out from this thesis outline due to the time limitations but should be considered in future work.

There are still many possible improvements that can be done to make the ICP applied in a real case scenario to work better. One of these can be to implement and use a more thorough motion model. Incorporating the input vector into the prediction step in the algorithm would increase the accuracy of the prediction step, improving the estimation. Also the choice of states to the state vector has an effect on the estimation. In systems like differential dive robots or modern cars the input to the system might strongly affect the knowledge of the future behaviour of the system. Also in many modern applications, very accurate sensors as IMUs (Inertial Measurement Units) are incorporated. These can give readings accurate enough to assume being input, or, if not, modifying the state vector can make it possible to use these measurements for updating the belief of the state, giving additional information that could be used directly or in next iteration's prediction. Here for example, it is easy to imagine an accurate orientation measurement would increase the certainty of the vehicle moving in a specific direction.

A possibility that has not been considered in this project is the recognition and measuring the relative position of some moving features. This could make it possible to increase feature count and further improve the positioning. This step would require modelling of the feature, determining it properties such as possible velocity ranges etc. Given this model a prediction step could be implemented, helping to determine the feature state given the previous estimated state. This of course could affect the positioning in a negative way, reducing the amount of useful information for estimation of the vehicle position itself. Also, a badly tuned prediction step could lead to confidence mismatch and degeneration of the localisation performance. Since the diversity of objects possible to recognise as features, it is easy to believe the tuning of the feature behaviour would be a complicated task. Using moving features in the decentralised algorithm would result in timing issues to be solved too. However, simulation showed that in some cases when over-confidence might occur, adding some uncertainty to the feature belief may improve the outcome of the filter. In other words, assuming a feature is moving when it actually is still might sometimes have positive influence.

Further on, we believe combining ICP with other existing positioning methods as for example the CoSLAT algorithm [11, 12] might further improve the function of the estimator. In Bayesian filtering many different sources of information may be combined to form an accurate estimate of the state.

Bibliography

[1] S. E. Shladover and S.-K. Tan, "Analysis of vehicle positioning accuracy requirements for communication-based cooperative collision waring," *Journal of Intelligent Transportation System*, vol. 10, no. 3, pp. 131–140, 2006.

[2] E. Kaplan and C. Hegarty, *Understanding GPS: principles and applications*. Artech house, 2006, ch. 1.

[3] "PlusON 330 Datasheet," Accessed: 2017-01-18. [Online]. Available: http://www.timedomain.com/wp/wp-content/uploads/2015/11/320-0330A-P330-Data-Sheet-PRELIMINARY.pdf

[4] E. Kaplan and C. Hegarty, *Understanding GPS: principles and applications*. Artech house, 2006, ch. 6, 7.

[5] N. Patwari, J. N. Ash, S. Kyperountas, A. O. Hero, R. L. Moses, and N. S. Correal, "Locating the nodes: cooperative localization in wireless sensor networks," *IEEE Signal processing magazine*, vol. 22, no. 4, pp. 54–69, 2005.

[6] Y. Wang, G. Leus, and A. J. van der Veen, "Cramer-rao bound for range estimation," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, April 2009, pp. 3301–3304.

[7] A. El-Mowafy and M. Al-Musawa, "Machine automation using rtk gps positioning," in *2009 6th International Symposium on Mechatronics and its Applications*, March 2009, pp. 1–6.

[8] H. Wymeersch, J. Lien, and M. Z. Win, "Cooperative localization in wireless networks," *Proceedings of the IEEE*, vol. 97, no. 2, pp. 427–450, 2009.

[9] A. T. Ihler, J. W. Fisher, R. L. Moses, and A. S. Willsky, "Nonparametric belief propagation for self-localization of sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 809–819, April 2005.

[10] J. Liu, M. Chu, and J. E. Reich, "Multitarget tracking in distributed sensor networks," *IEEE Signal Processing Magazine*, vol. 24, no. 3, pp. 36–46, 2007.

[11] F. Meyer, E. Riegler, O. Hlinka, and F. Hlawatsch, "Simultaneous distributed sensor self-localization and target tracking using belief propagation and likelihood consensus," in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*. IEEE, 2012, pp. 1212–1216.

[12] F. Meyer, F. Hlawatsch, and H. Wymeersch, "Cooperative simultaneous localization and tracking (coslat) with reduced complexity and communication," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 4484–4488.

[13] G. Soatti, N. Garcia, H. Wymeersch, M. Nicoli, B. Denis, and R. Raulefs, "Implicit cooperative positioning."

[14] Z. H. Mir and F. Filali, "Lte and ieee 802.11 p for vehicular networking: a performance evaluation," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, p. 89, 2014.

[15] A. Festag, "Standards for vehicular communication—from ieee 802.11 p to 5g," *e & i Elektrotechnik und Informationstechnik*, vol. 132, no. 7, pp. 409–416, 2015.

[16] F. Mattern and C. Floerkemeier, "From the internet of computers to the internet of things," in *From active data management to event-based systems and more*. Springer, 2010, pp. 242–259.

[17] Mobile Robots, "Pioneer Robot Datasheet," Accessed: 2017-01-18. [Online]. Available: http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx

[18] Edwin Olson, "Gulliview," Accessed: 2017-03-10. [Online]. Available: https://bitbucket.org/thpe/visionlocalization

[19] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, vol. 3.

[20] O. BERONIUS, E. LUNDÉN, M. MALMQUIST, and A. ROHLIN, "Coordination of robots via a wireless network," 2016.

[21] H. Pottmann and J. Wallner, *Computational line geometry*. Springer Science & Business Media, 2009.

[22] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, Feb 2002.

[23] H. Wymeersch, *Iterative Receiver Design*. Cambridge University Press, 2007.

[24] K. J. Aström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[25] S. M. Kay, *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory (v. 1)*. Prentice Hall, 1993.

[26] R. L. Freeman, *Radio system design for telecommunication*. John Wiley & Sons, 2006, vol. 98.

[27] S. Gezici, Z. Tian, G. B. Giannakis, H. Kobayashi, A. F. Molisch, H. V. Poor, and Z. Sahinoglu, "Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks," *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 70–84, July 2005.

[28] Y. Shen and M. Z. Win, "Fundamental limits of wideband localization—part i: A general framework," *IEEE Transactions on Information Theory*, vol. 56, no. 10, pp. 4956–4980, 2010.

[29] G. C. Chow, *Analysis and Control of Dynamic Economic Systems*. Krieger Pub Co, 1986.

[30] "ROS," Accessed: 2017-02-16. [Online]. Available: http://www.ros.org/

[31] Logitech, "Logitech Webcam C930e," Accessed: 2017-01-18. [Online]. Available: http://www.logitech.com/assets/64665/c930edatasheet.ENG.pdf

[32] "AprilTags," Accessed: 2017-04-20. [Online]. Available: https://april.eecs.umich.edu/

[33] "ROSARIA," Accessed: 2017-02-16. [Online]. Available: http://wiki.ros.org/ROSARIA

[34] Dirk Thomas, “rqt_graph,” Accessed: 2017-05-15. [Online]. Available: http://wiki.ros.org/rqt_graph
[35] Omron, “MobileSim,” Accessed: 2017-03-18. [Online]. Available: http://www.mobilerobots.com/Software/MobileSim.aspx