Ing. Michael Geigl, BSc

Analysis and calibration of time inaccuracies in sensor data affecting a mobile robot's localization system.

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisors

Gerald Steinbauer, Assoc.Prof. Dipl.-Ing. Dr.techn.
Institute of Software Technology

Graz, January 2021

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date, Signature

Abstract

Nowadays mobile robots integrate many different sensors into the data fusion process to increase safety and stability. An important requirement for the input data streams for the fusion process is that they are synchronized in time.

The different sensors use different sampling periods and communication interfaces. In most mobile robots the central processing unit builds upon general-purpose PCs and operating systems. All of these given components introduce time inaccuracy which conflicts with the synchronized time requirement.

This thesis shows concepts to correct this time inaccuracies and to estimate the delay between two sensor data streams using the cross-correlation function. A key property of this concept is the low computation cost which enables the execution of the delay estimations during the normal workload of a mobile robot.

The concept was evaluated using three real shuttles and the results show that in most cases there is a significant asynchronicity between the sensors. With the knowledge of the delay, it is possible to correct it at the timestamping procedure.

Furthermore, the effect of this asynchronicity on the localization system of a mobile robot is investigated.

Kurzfassung

Aktuelle mobile Roboter inkludieren immer mehrere verschiedene Sensoren, um die Stabilität und die Sicherheit des Systems zu erhöhen. Dabei werden die Datenströme der verschiedenen Sensoren miteinander fusioniert. Um ein sinnvolles Ergebnis durch die Fusionierung zu erhalten ist es notwendig, dass die Datenströme zeitlich synchronisiert sind.

Die verschiedenen Sensoren benützen unterschiedlichste Messperioden und Kommunikationsschnittstellen. Zusätzlich läuft die zentrale Recheneinheit eines gewöhnlichen Roboters auf herkömmlichen Betriebssystemen. Diese Eigenschaften bringen Fehler in die zeitliche Betrachtung von den Sensordaten mit ein und stören die geforderte zeitliche Synchronisation.

Diese Masterarbeit beschreibt ein Konzept zur Korrektur dieser Fehler und eine Methode zur Abschätzung des zeitlichen Versatzes zwischen zwei Sensordatenströmen basierend auf der Kreuzkorrelationsfunktion. Diese Methode ermöglicht es, den Zeitversatz zwischen verschiedensten Sensortypen zu bestimmen. Außerdem ist sie dafür konzipiert während des alltäglichen Arbeitsablaufs eines mobilen Roboters zu funktionieren, um kontinuierlich Abschätzungen ohne zusätzliche Gegebenheiten durchführen zu können.

Das Konzept wurde auf drei unterschiedlichen realen mobilen Robotern getestet. Die Ergebnisse zeigen einen signifikanten Zeitversatz zwischen den Sensoren. Durch die Messung dieses Versatzes kann dieser in der Weiterverarbeitung der Daten berücksichtigt werden.

Zusätzlich wurden in dieser Arbeit auch die Auswirkungen dieses Zeitversatzes auf den Lokalisierungsprozess eines mobilen Roboters untersucht.

Acknowledgements

I want to thank the incubedIT company that they came up with the idea for this thesis and their support during the work on the implementation and evaluation of it. Special thanks go to Michael Reip and Lukas Reisinger for their constructive advises and results discussions. Furthermore, I want to thank Gerald Steinbauer for the scientific input and the given feedback.

Contents

List of Figures

List of Tables

Listings

List of Algorithms

1 Introduction

In this chapter, the topic, goals and challenges of this thesis are presented. The conclusion of the chapter is an outlook on the thesis.

1.1 Motivation

Nowadays, more and more autonomous robots appear in our everyday life. Autonomous driving cars (Figure 1.1a), mobile shuttle robots, which deliver packages in the same area where people and dynamic obstacles are active, vacuum cleaning robots (Figure 1.1b), exploration robots, which are used in hazardous areas, or drones are only a few examples.

(a) Self driving car. (b) Vaccum cleaning robot.

Figure 1.1: Mobile robots in everyday life.

A key ability, all of them have in common, is that they localize themselves without any additional infrastructures such as magnetic tapes on the floor or some kind of wireless beacons in the space. Their localization only relies on the data observed from their own intrinsic (wheel encoders, imu,..) or extrinsic (lidar, camera, ultrasonic,..) sensors. This allows the robot to be more flexible and reduces installation costs and time.

In general, autonomous mobile robots combine different types of sensors to estimate their internal state or a state of the environment. The reason for this is that every sensor type has its own advantages and disadvantages which makes it impractical to only rely on one sensor for several tasks. Therefore, different sensor types are combined to overcome one sensor's disadvantages. For example, an encoder sensor and a laser scanner are combined for the localization task. The encoder sensor can measure driven

distances even in quiet dynamic scenarios but has the disadvantage that slippage and other errors sum up over time and let the estimated robot pose drift away from its ground truth pose. Whereas the laser scanner is not affected by a drift because it always measures the absolute distances to its environment. But a laser scanner gets disturbed by unmapped obstacles in its environment. Moreover, to due its lower measurement frequency, the laser scanner has a lower performance at higher dynamical movements of the robot.

The example shows that this kind of sensor fusion can be very beneficial but there is one very important condition to be fulfilled. The sensors need to be synchronized in time. In a worst-case scenario, where the synchronization is very bad, the fused sensor data can be worse than using the data of one sensor on its own. This condition is even more important the faster the robot moves. To illustrate this, let us assume a robot moves with 0.1m/s and the sensors are off by a time delay of 100ms. This results in a position difference of 1cm whereas if the robot moves with 10m/s the position difference is already 1m. And the velocity of mobile robots today can be roughly quantified with 2m/s at indoor application and up to 35m/s at autonomous driving cars.

1.2 Challenges and goals

The major goal of this thesis is to determine the delay between the sensors to be able to synchronize them and fulfill the important condition for sensor fusion.

To make the task more challenging, this estimation should be applicable to many different types of sensors with even more divergent sampling rates and communication interfaces. And, the estimation should be executed during the normal application workload of the mobile robot which forces the concept to take care of the computational costs.

A second goal of the thesis is to show how the delay between two sensors affects the localization of a mobile robot. It is expected that it confirmes the importance of the synchronization property of the input data.

1.3 Thesis Outline

The two next chapters in this thesis introduce the problem and some basics of robotics, which are used throughout the whole thesis. This is followed up by a discussion about related research and the explanation of the theoretical concept of the selected method. The thesis continues with the implementation chapter, where all details are included on how the concept was realized on real mobile robots, and the evaluation chapter, where the test results are presented. Finally, the work ends with a conclusion and an outlook about future work.

2 Problem statement

In this chapter, the initial situation of the target application and the problem are introduced.

2.1 Initial situation

The sensor time synchronization is an essential topic for a big variety of robots and an even bigger variety of multi-sensor setups on such robots. To keep the scope of this thesis in an appropriate size the focus lies on the time synchronization of mid-size indoor shuttles like the one in Figure 2.1. These mobile robots are designed for delivering packages from one station to another inside a warehouse with dynamical obstacles and no localization helping infrastructure. Nevertheless, the concepts and theories are also valid for similar robots and sensor setups.

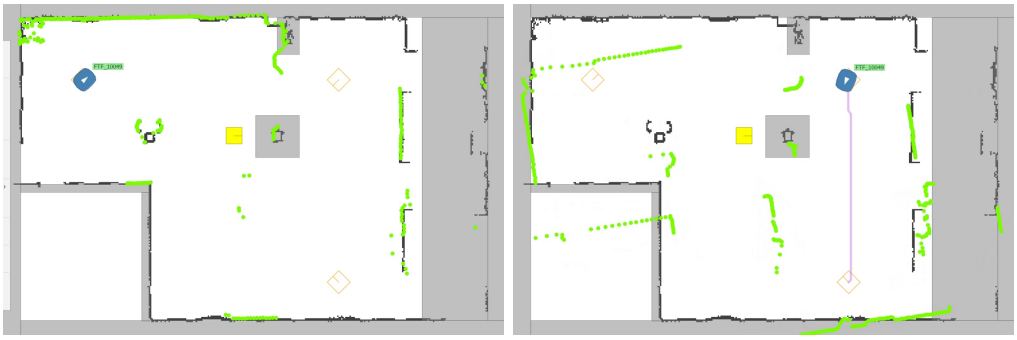
Figure 2.1: A mobile robot as a box deliviering shuttle

Such shuttle robots are typically equipped with a differential drive including encoder sensors that delivers odometry data and a Light Detection and Ranging (LiDAR) laser scanner as an extrinsic sensor. This combination is preferred because it is more robust and computational less expensive than other systems like a 3D scanner or a camera.

The sensors are kept simple to make the setup process easier. They often only need to be supplied with power and already deliver periodically sensor data at the communication interface. Usually, they do not offer any kind of clock synchronization neither in hardware or software. The data receives its timestamp upon arrival at the host PC using the PC's local clock. The data from different sensors are then aligned in time upon these timestamps and get fused in subsequent processes.

The shuttles already comprises a full-stack localization where this sensor data is implicitly fused within a particle filter to get a pose estimation. This means that this localization stack is fixed and should not be changed.

Nevertheless, errors from time delays in the sensor data could be observed at the Graphical User Interface (GUI) of the shuttle. The laser scanner data gets transferred into the map coordinate system depending on the estimated robot pose from the localization and the endpoints are shown in the GUI. If the robot stands the points are perfectly aligned with walls and obstacles in the map (see Figure 2.2a). If the robot starts accelerating or decelerating the laser endpoints are shifted from the walls and obstacles (see Figure 2.2b). This seems to be mainly caused by time delay issues because it is only observable during the movement of the robot.



(a) Robot standing (b) Robot accelerating

Figure 2.2: Laser scan end points (green dots) projected into the map at the GUI.

There were no investigations made so far in how much delay exists between the sensor data and how they could be synchronized. Thus, there exists no prior knowledge about the nature of the delay and its effect on the sensor fusion or the localization process.

2.2 Problem discribtion

To understand the problem with the time synchronization of sensor data, it is necessary to know the data flow from the time where it is measured until the data receives a timestamp. The shuttles which are mentioned in the previous section use, in general, sensors with an internal clock that triggers new measurements at a constant frequency. When the measurement is done the sensor internally post-processes it and sends it via serial or Ethernet-based communication to the shuttles central processing unit. This is a

PC with a generic operating system that has its own clock and which processes incoming data as soon as resources are free with no guaranteed maximum delay time. When the data gets processed in the PC it receives a timestamp of the local PC time which is further associated with the time point where the measurement was taken.

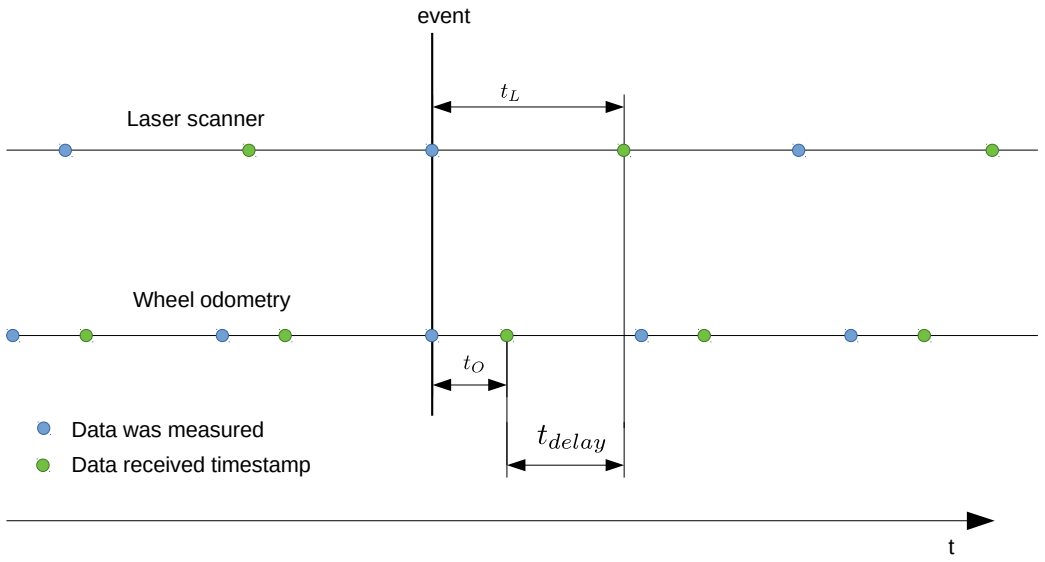


Figure 2.3: Example of measuring and timestamping behaviour from two different sensors

Obviously, the time point which is associated with the measurement is delayed from the real-time point where the measurement was taken. The delay between the real-time point and the associated time point of the laser scanner data is called t_L and respectively for the wheel odometry t_O. This values are illustrated in Figure 2.3. These sensor delays are affected by a major constant delay but also by jitter and possible data jams [Mai+11]. The constant delay depends on the sensor, the communication method and the startup sequence. The jitter is mainly caused by the operating system of the PC and its scheduling behavior of the software system. These sensor delays also differ from one sensor to another, even when two sensors used are from the same type. And these sensor delays are unknown in general.

Not knowing these sensor delays is a problem for every kind of sensor fusion. It is not clear at which exact point in time the data was taken and thus, it is very unlikely that it is fused with another sensor data at the exact same real measurement time point. The only way to measure these sensor delays is to use external hardware, but this is not convenient for the desired application.

Nevertheless, the sensor fusion process is not sensitive to the individual sensor delays but more to the delays between sensors which is named t_{delay} and shown in the Figure 2.3. If all sensors would have the same delay ($t_L = t_O$) the measurements would be still

aligned in time to each other. It would only produce a latency to the real-time which is not that critical for the localization process in the desired application. As a result, the problem is to find out the time delay between two sensor data streams and correct the timestamps such that the streams are aligned in time.

In addition to the problem of not knowing t_{delay}, it is also a problem that it is not known how such a delay affects the localization. Maybe the localization is not affected by small delays of the sensor data, but maybe the performance of the localization would increase dramatically if the sensor's data are synchronized. Most likely, the truth will be somewhere in between these two extremes and should be found out.

2.3 Contribution

Two major goals should be achieved with this thesis. The first one is to estimate the delay and align all sensor data streams to each other in time. This delay estimation needs to be computationally inexpensive, such that it could be performed regularly on the shuttle itself, work without any additional hardware and apply to many different sensors and robots.

The second major goal is to quantify the effect of the time delay on the localization process. This would allow to control one error source of the localization and therefore be able to improve it.

3 Prerequisites

In this chapter, all the necessary background information needed to understand the continuing thesis are presented. First of all, a common software framework for robot systems the so-called Robot Operating System (ROS) is introduced. The subsequent section, which is called Range Flow 2D Odometry, describes a method to extract odometry information from laser scanner data. This is then followed by a section about a popular robot localization algorithm. Finally, this chapter finishes with a robust method for time-delay estimations between two signals.

3.1 Robot Operating System

The Robot Operating System (ROS) is one of the most popular frameworks for developing robots today. It originates from a project of the Willow Garage and the Stanford University back in 2007 [Qui+09]. At that time each robot framework was specialized for its application and each new robot needed the development of a whole new framework again. This was impractical and these two groups tried to change it by establishing a framework that can be easily applied to many different robot platforms and make it possible to reuse software modules. They came up with five design goals for ROS: peer-to-peer, tools-based, multi-lingual, thin, free and open-source.

With peer-to-peer, the goal was to enable a structure of independent processes that directly communicate with each other. Independent processes were important to make the software more modular such that each separatable processing step can be kept within one module and can run on its own. Direct communication between these modules should keep the network traffic low in comparison to a communication structure with one dedicated master.

The tools-based goal aimed to provide useful tools like visualizations, auto-generated documentation, bandwidth measurement, etc. but at the same time keep the complexity low. Thus, these tools are also separately implemented in modules instead of all in one master module.

The multi-lingual goal was important because in different topics of the robotic different languages were preferred and with this goal, the already established codes could be used with minimal adaptions to the ROS framework.

This reusability of other code is also expressed in the thin goal, which forces the driver and other software implementation to put the complexity into stand-alone libraries.

And last but not least the framework should be open-source to make collaborating work easier.

These five goals are still present and the core ROS package gets developed from the community-driven Open Source Robotics Foundation. Besides this core ROS package,

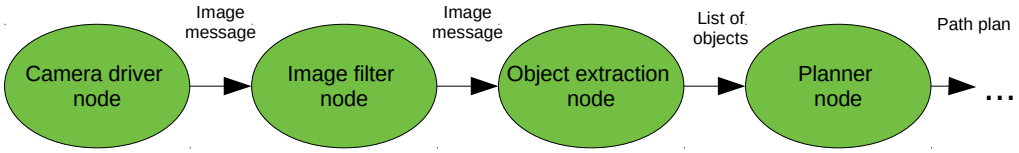


Figure 3.1: Example of a sensor pipeline in ROS

many additional packages for drivers, navigation, localization, mapping, etc. are already provided by the community and are ready to use which raised the popularity of this framework.

This settles the background about ROS. The continuing sections will explain how ROS exactly works internally.

3.1.1 Nodes and communication

The ROS framework is modular and puts one processing step into one module. Such a module is called **node** in ROS and each node can run on its own. Typically, a robot consists of many nodes running in parallel. Already a sensor pipeline is generally split into different nodes like it is shown in Figure 3.1.

The nodes communicate with each other via **messages**. Messages define the structure of the data which is exchanged within the message and there are many predefined messages available in the ROS library. One of them is the laser scan message which is listed, as an example, in Listing 3.1.

```
1  # Single scan from a planar laser range-finder
2  Header header           # timestamp in the header is the acquisition
3                          # time of the first ray in the scan.
4
5  float32 angle_min       # start angle of the scan [rad]
6  float32 angle_max       # end angle of the scan [rad]
7  float32 angle_increment # angular distance between measurements [rad]
8
9  float32 time_increment  # time between measurements [seconds]
10 float32 scan_time       # time between scans [seconds]
11
12 float32 range_min       # minimum range value [m]
13 float32 range_max       # maximum range value [m]
14
15 float32[] ranges        # range data [m]
16 float32[] intensities   # intensity data [device-specific units]
```

Listing 3.1: Definition of the sensor_msgs/LaserScan.msg [1]

[1] http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html, accessed 17.05.2021

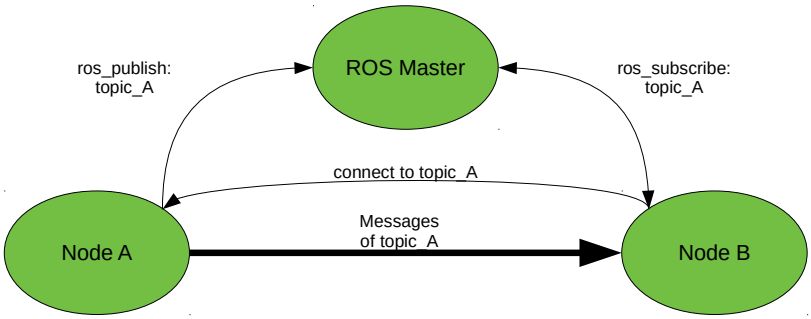


Figure 3.2: Example of ros publish and subscribe mechanism.

The header field of this message is not a primitive datatype like an integer, a float, or a char. Instead, it is a standard message again which consists of useful metadata about the current message and its definition is shown in Listing 3.2. The stamp field of the datatype time has an important role in this thesis because it stores the time point when the data of the current message was recorded.

```
1 uint32 seq       #sequence ID: consecutively increasing ID
2 time stamp       #Two-integer timestamp that is expressed as
3                  # stamp.sec and stamp.nsec
4 string frame_id #Frame this data is associated with
```

Listing 3.2: Definition of the header.msg [2]

Nodes are not aware of each other in a ROS Framework. To enable to communicate with other nodes the **publisher** and **subscriper** paradigm is used. A node that wants to provide data to other nodes publishes this data onto a defined **topic** which is identified by its name. If another node wants to use this data it subscribes to this specific topic.

To coordinate the nodes, publisher and subscribers, all nodes register during the start-up process themselves at the **rosmaster**. They tell the master which topics they publish and on which topics they want to subscribe. For the topics they subscribe to, the master tells them which node publishes these data and therefore the subscribing node can directly communicate with the publishing node such that the data traffic does not need to travel through the master. This is part of the peer-to-peer design goal of ROS and it is illustrated in Figure 3.2.

3.1.2 Transformations

Typical robot systems consist of many separate right-handed coordinate systems to make calculations and data processing steps easier. An example of such a coordinate system is a laser scan sensor coordinated system where the origin lies at the center of the rotating mirror. Another example is the robot coordinate system of a differential-drive mobile

[2] http://docs.ros.org/en/noetic/api/std_msgs/html/msg/Header.html, accessed 17.05.2021

robot where the origin of its systems conveniently lies in the pivot point of the differential drive.

In ROS, such coordinate systems are called **frames** and each data set is rooted in a particular frame. Often it is necessary to represent data in a different frame. Thus, the data needs to be transformed into the target frame. This is handled in ROS with the **tf** library [Foo13] which keeps track of the available frames and how they are transformed. These frames and transformations are represented in a tree-like structure as can be seen in Figure 3.3. If data needs to be transformed, the node sends the data with the source frame, the target frame and the desired time point to the tf library. The library searches for the transformation with the closest timestamps for the desired time point, linear interpolates it, transforms the data and sends the data to the node.

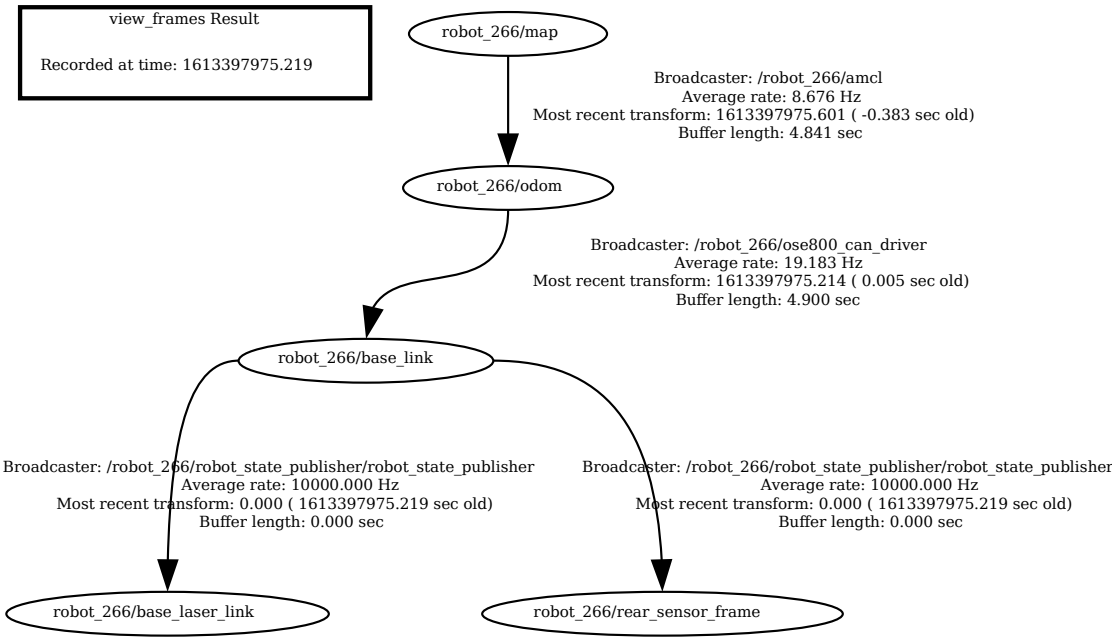


Figure 3.3: The transformation tree of a mobile robot.

3.1.3 Command line tools

The ROS package provides several useful tools that can be used via the command line. This is handy because the code does not need to be recompiled and the tools can be used right away when the nodes are already running.

One of these tools is the **rosbag**. This tool subscribes to a desired topic and stores all received messages in a bagfile on the hard drive. Later on, the rosbag tool can replay the messages from the bagfile which makes it possible to once record sensor data and test

different postprocessing steps with different parameters on the same data. This function is necessary for the location evaluation in this thesis.

3.2 Localization

Localization, in the context of mobile robots, is the process of determining the pose of the robot in its environment. In the specific case of our indoor shuttle, the pose is the x and y coordinate and the yaw orientation in the coordinate system of the environment map. In a more mathematical term, this pose is represented as a state. The state variable of the robot at the time point t is called x_t and it is a vector of the following shape.

$$x_t = \begin{pmatrix} x \\ y \\ \Theta \end{pmatrix} \tag{3.1}$$

This state can not be measured directly. Instead, it must be estimated from a stream of sensor measurements of the environment and subsequent motion commandos of the robot. A common technique for such state estimations is the Bayes Filter.

3.2.1 Bayes Filter

The Bayes Filter calculates the estimated state from the past observation and control data. In our case, the observation data are the sensor measurements with the mathematical abbreviation z and the control data are the motion commandos of the robot with the abbreviation u.

To derive the Bayes filter we start with the given inputs and the desired output. The inputs are all past observation and control data. The desired output is the probability distribution of the state variable conditioned with all past observation and control data $p(x_t|u_{1:t}, z_{1:t})$. This is also called belief $bel(x_t)$.

If the conditioned Bayers rule, Equation 3.2, is used the belief can be expressed as in Equation 3.4. Where $p(z_t|x_t, u_{1:t}, z_{1:t-1})$ represents the likelihood of the current observation at the state x_t and $p(x_t|u_{1:t}, z_{1:t-1})$ represents the prior distribution of x_t updated with the latest control data. η is just a scaling factor.

$$p(x|y, z) = \frac{p(y|x, z)p(x|z)}{p(y|z)} \tag{3.2}$$

$$\begin{aligned} bel(x_t) = p(x_t|u_{1:t}, z_{1:t}) &= \frac{p(z_t|x_t, u_{1:t}, z_{1:t-1})p(x_t|u_{1:t}, z_{1:t-1})}{p(z_t|u_{1:t}, z_{1:t-1})} && (3.3) \\ &= \eta p(z_t|x_t, u_{1:t}, z_{1:t-1})p(x_t|u_{1:t}, z_{1:t-1}) && (3.4) \end{aligned}$$

Given the total probability theorem, the updated prior distribution expands to Equation 3.5. Because the control data at time point t has no effect on the state x_{t-1}, we get the Equation 3.6. Finally, it can be observed that the right probability is the same as the belief of the state at $t - 1$.

```
1 Algorithm Bayes filter(bel(x_{t-1}),u_t,z_t):
2   for all x_t do
3     \overline{bel}(x_t) = ∫ p(x_t|u_t,x_{t-1}) bel(x_{t-1}) dx
4     bel(x_t) = η p(z_t|x_t) \overline{bel}(x_t)
5   endfor
6   return bel(x_t)
```

Listing 3.3: Bayes filter algorithm. Source: [TBF05]

$$
\begin{aligned}
p(x_t|u_{1:t}, z_{1:t-1}) &= \int p(x_t|x_{t-1}, u_{1:t}, z_{1:t-1}) p(x_{t-1}|u_{1:t}, z_{1:t-1}) dx_{t-1} && (3.5)\\
&= \int p(x_t|x_{t-1}, u_{1:t}, z_{1:t-1}) p(x_{t-1}|u_{1:t-1}, z_{1:t-1}) dx_{t-1} && (3.6)\\
&= \int p(x_t|x_{t-1}, u_{1:t}, z_{1:t-1}) bel(x_{t-1}) dx_{t-1} && (3.7)
\end{aligned}
$$

For the next step, the assumption is used that the state x_t fulfills the Markov property. A state with the Markov property is a complete state. A complete state means that all previous observations and control data is incorporated into the state x_t and previous data, therefore, does not deliver any further information. This leads to conditional independence in mathematical terms which results in Equation 3.8 and 3.9.

$$
\begin{aligned}
p(z_t|x_t, u_{1:t}, z_{1:t-1}) &= p(z_t|x_t) && (3.8)\\
p(x_t|x_{t-1}, u_{1:t}, z_{1:t-1}) &= p(x_t|x_{t-1}, u_t, u_{1:t-1}, z_{1:t-1}) = p(x_t|x_{t-1}, u_t) && (3.9)
\end{aligned}
$$

Compining all together results in Equation 3.10 where $p(z_t|x_t)$ represents the probability of the current measurement z_t at the current state, $p(x_t|x_{t-1}, u_t)$ the transition probability from the state x_{t-1} to x_t given the control data u_t and $bel(x_{t-1})$ the belief of the state at the timepoint $t-1$. This equation is the Bayes filter and it can be observed that the belief of the current state depents on the belief of the last state, which makes it an recursive filter. It further depents on the current observation and the current control data.

$$
bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1} \qquad (3.10)
$$

Listing 3.3 shows the general Bayes filter algorithm where the Equation 3.10 is split into two steps. In Line number 3 the control update step is calculated where the last state gets updated with the new control data. Line number 4 shows the correction step, where the new belief is corrected given by the current observation.

Calculating the belief exactly with the Bayes filter is only applicable in very specialized and simple cases. Most applications use an approximation of the posterior belief to

```
1  Algorithm Particle filter(X_{t-1}, u_t, z_t):
2    X̄_t = X_t = 0
3    for m = 1 to M do
4      sample x_t^[m] ~ p(x_t | u_t, x_{t-1}^[m])
5      w_t^[m] = p(z_t | x_t^[m])
6      X̄_t = X̄_t + ⟨x_t^[m], w_t^[m]⟩
7    endfor
8    for m = 1 to M do
9      draw i with probability ∝ w_t^[i]
10     add x_t^[i] to X_t
11   endfor
12   return X_t
```

Listing 3.4: The Particle filter algorithm. Source: [TBF05]

make it easier to implement, useable for more complicated posterior distributions and computationally more efficient. Unfortunately, an approximation always comes with a certain loss of accuracy. Therefore, a good trade-off needs to be found.

3.2.2 Particle filter

One approximation method of the Bayes filter is the Particle filter. A particle is a sample state in the state space. The Particle filter uses a set of these particles, which is called X_t and the filter calculates a weight w of each particle depending on how good the current observation fits this particle. The better it fits the higher the weight. These weighted particles represent the belief about the current state.

The Listing 3.4 shows the Particle filter algorithm. It gets the last set of particles X_{t-1}, the current control data u_t and the current observation data z_t as an input. The variable M stands for the number of the particles in the particle sets and lies typically between several hundred and a few thousand. One explicit particle is donated with $x_t^{[m]}$ in the particle set with $1 < m < M$ and its corresponding weight is donated with $w_t^{[m]}$. The particle sets get processed in the algorithm in the steps of update, weighting and resampling.

The first step is implemented in Line 4 and is called the update step. This is similar to the control update step from the Bayes filter. In this step a new particle $x_t^{[m]}$ is drawn from the transition probability $p(x_t|u_t, x_{t-1}^{[m]})$ with respect to the prior sample $x_{t-1}^{[m]}$ and the current control data u_t.

The weighting step is implemented in Line 5 of the Particle filter algorithm. It gives each particle a weight representing how well the state of the particle fits with the current observation. The weighted particles after this step represent the posterior belief $bel(x_t)$ of our state.

The final step of the algorithm is the resampling step in Lines 8 to 11. It uses the previously calculated sample set and draws new samples at places where the weights of

the samples were high. This is beneficial because the particles are now focused around the belief and particles with very small weights in very unlikely regions are removed.

In order to implement this filter in a real application, the transition probability $p(x_t|u_t, x_{t-1})$ and the observation probability $p(z_t|x_T)$ need to be known. These probabilities model the real-world motion and measurement process and are explained in the next two subsections.

3.2.3 Motion Model

The control data u_t is affected by uncertainty because of measurement errors etc. The motion model incooperates this uncertainties and outputs a posterior distribution $p(x_t|u_t, x_{t-1})$ of x_t when the control command u_t was applied on the prior state x_{t-1}.

The control data u_t on a differential drive robot conveniently consists of the translational v_t and rotational w_t velocity. These velocities are applied to the robot and the robot ideally moves immediately and exactly with these velocities. This is not possible because of the dynamics of the robot. To suppress this error, it is common to use odometry data given from wheel encoders.

The odometry delivers periodically pose data in the robot frame which is stated with the bar above the laters. The key idea is that the difference between the two poses in the robot frame is similar to the difference of the poses of the robot in the global frame.

The motion between two subsequent odometry poses $\langle \bar{x}, \bar{y}, \bar{\Theta} \rangle$ and $\langle \bar{x}', \bar{y}', \bar{\Theta}' \rangle$ is splited into three premitive motions. First, a in place rotation with the angle δ_{rot1}, followed by a translation of δ_{trans} and concluded with a in place rotation with angle δ_{rot2}. This motion seperation is shown in Figure 3.4 and the calculation of the delta values from the odometry poses is implemented in line 2 to 4 in the Listing 3.5.

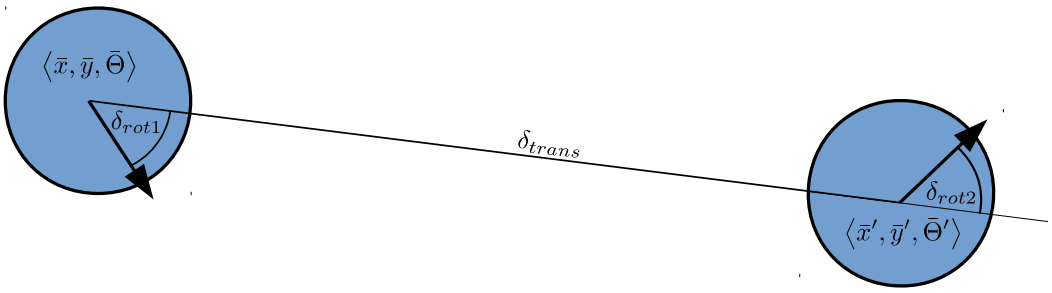


Figure 3.4: The robot motion splited into three premitive motions.

This odometry data is more accurate than the control data but is still affected by errors from slippage and drifts. To apply this uncertainty each primitive motion is added with zero centered Gaussian noise where the $\alpha_1 - \alpha_4$ values affect the standard deviation of the distribution. The sampling of the normal distribution is implemented as it is shown in Listing 3.6. The α values need to be selected appropriately for each application.

```
1  Algorithm sample motion model odometry(u_t, x_{t-1}):
2    δ_rot1 = atan2(ȳ' − ȳ, x̄' − x̄) − Θ̄
3    δ_trans = √((x̄ − x̄')² + (ȳ − ȳ')²)
4    δ_rot2 = Θ̄' − Θ̄ − δ_rot1
5
6    δ̂_rot1 = δ_rot1 − sample(α1 δ_rot1 + α2 δ_trans)
7    δ̂_trans = δ_trans − sample(α3 δ_trans + α4(δ_rot1 + δ_rot2))
8    δ̂_rot2 = δ_rot2 − sample(α1 δ_rot2 + α2 δ_trans)
9
10   x' = x + δ̂_trans cos(Θ + δ̂_rot1)
11   y' = y + δ̂_trans sin(Θ + δ̂_rot1)
12   Θ' = Θ + δ̂_rot1 + δ̂_rot2
13   return x_t = (x', y', Θ')^T
```

Listing 3.5: The sampling odometry motion model. Source: [TBF05]

```
1  Algorithm sample normal distribution(b):
2    return b/6 Σ_{i=1}^{12} rand(−1, 1)
```

Listing 3.6: The sampling of a normal distribution. Source: [TBF05]

3.2.4 Sensor Model

For the observation data we also need to model the probability distribution $p(z_t|x_t)$ which reflects the likelihood of an observation. A difference between a general Particle filter and the application in a mobile robot is that this probability not only depends on the last robot state x_t but also a given map m of the static environment of the robot. Therefore, we are looking for the propability distribution $p(z_t|x_t, m)$.

Our application uses a LIDAR sensor that sends out laser beams and measures the time until the reflection is back at the sensor. Multiplying this time with the speed of light results in a distance value. The laser sensor is rotating so that a distance value is taken from the environment at different angles. All distance values from the most left to the most right measurement angle together are called a scan and represent the current observation z_t.

In the real-world such a distance measurement most likely does not give the exact distance value from the robot pose to the environment presented in the map. There exist four common errors affecting such measurements.

The first one is measurement noise. Given through the discretization, internal sensor state changes (temperature,..) and air pollution the expected value can slightly vary from the real value. This can be represented with a normal distribution $p_{exp}(z_t|x_t, m)$ centered at the expected value z_t^{k*} and a standard deviation which depends on the sensor type (Figure 3.5a). The k at the expected value z_t^{k*} stands for the k-th measurement in a scan and the star marks that this value is the expected value of the scan at the pose

x_t extracted from the map via ray casting.

The second error is a maximum distance value. This appears if the out sent beam is not reflected and does not reach back to the sensor. It happens for example on smooth surfaces and small angles between the light beam and the surface or at very long corridors. This error can be modeled as uniform distribution at the maximum distance value of the sensor z_{max} and is donated with $p_{max}(z_t|x_t, m)$ (Figure 3.5b).

The third source of error is that the environment is not only static. There are dynamic and static obstacles around the robot that are not included in the map. These measurements are called unexpected measurements and can be model with an exponential distribution $p_{unexp}(z_t|x_t, m)$ because small distances for unexpected values are much more likely than bigger ones (Figure 3.5c).

The last error type is random measurements. Some measurement values of the sensor are pure random values which are triggered because of inferences from other light sources or reflections from past out sent laser beams. This error can be model as a uniform distribution $p_{rand}(z_t|x_t, m)$ over the whole measurable distances of the sensor. Figure 3.5d shows this probability distribution.

This beam-based model makes it necessary to use ray casting for each laser beam to find out the expected value z_t^{k*}. Ray casting for k values, k typically is in the size of 180 to 720, is computationally expensive and not always possible in real-time. Because of this, the likelihood field method is preferred.

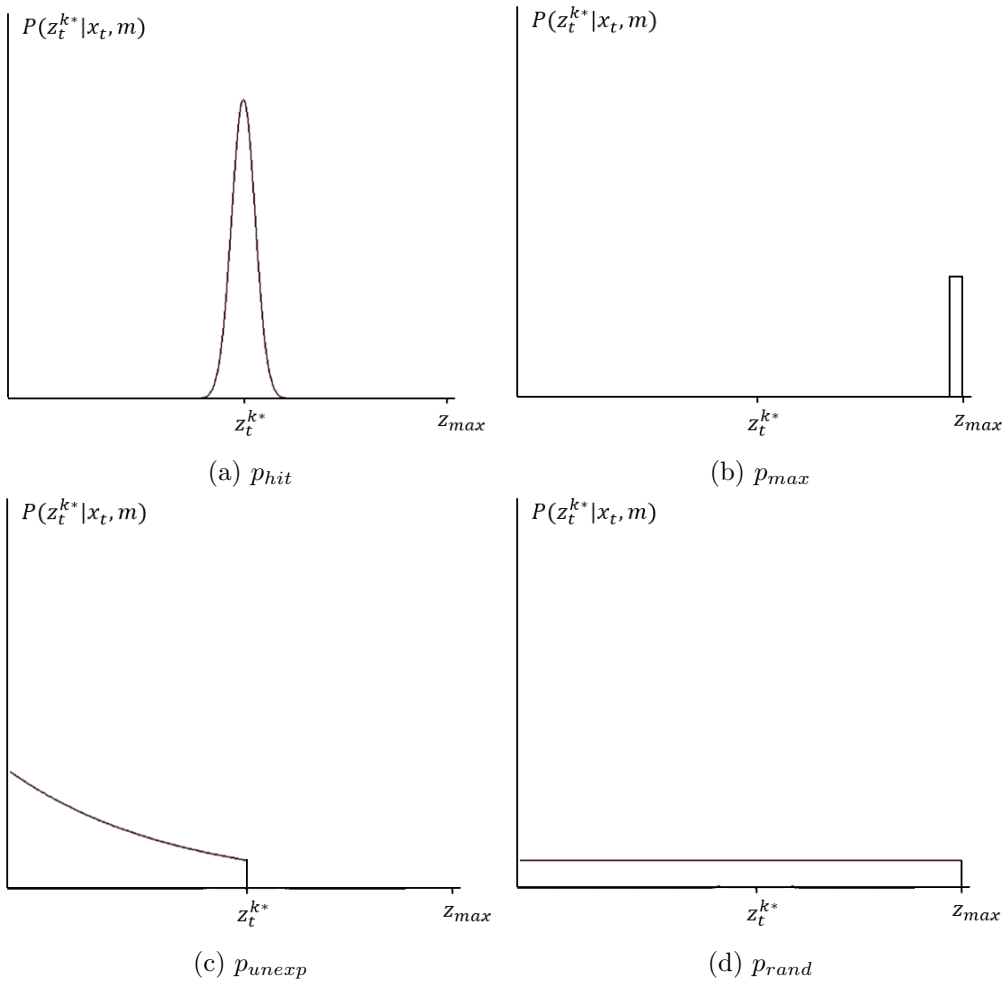
The likelihood field method inverts the process of the p_{hit} distribution. Instead of calculating the probability from the distance of the measurement and the expected distance, it computes a likelihood map where walls and objects in the map represent a high probability of a hit. For areas in the map with no walls, the probability of a hit decreases with respect to the distance to the closest wall. The likelihood map can be precalculated and for the real-time operation, the endpoint of each sensor measurement z_t^k only needs to be transformed in the map frame and the probability is simply a lookup in the likelihood map. The transformation calculation is shown in the Listing 3.7 Line 5 and 6. Line 7 shows the distance calculation to the nearest neighbor in the map which can be exchanged with a lookup if the likelihood map is precalculated.

The cases of maximum and random measurements are still modeled as uniform distributions and are included in the algorithm in Line 9. The short measurements are ignored because they can somewhat be seen as random measurements.

Of course, this is a simplified model with no sound mathematical derivation and it has some drawbacks like by only using the endpoint of a beam it could go through walls. Nevertheless, it is very popular and delivers good results in real applications.

3.2.5 Adaptive Monte Carlo Localization

Combining a particle filter with a motion model and a sensor model from the above section to use it for the localization of a mobile robot leads to the Monte Carlo Localization (MCL). The algorithm of it is shown in the Listing 3.8. It is very popular because it solves the global localization problem, is simple to implement, non-parametric and computational moderate expensive.



(a) p_{hit} (b) p_{max}

(c) p_{unexp} (d) p_{rand}

Figure 3.5: Separate probability distributions of the Sensor Model (Source: [TBF05]).

The Adaptive MCL is an improvement of the MCL because it adapts the number of particles M used in the Particle filter during the run time. This adaptation is useful because the number of particles is a trade-off between the accuracy of the localization and the computational expense. The more particles are used the more accurate the localization is but also the computational expanse rises. Increasing the number of particles only happens if the localization is uncertain about its belief which can be derived from the variation of particles or the variation of the particle weights before the resampling step. If the algorithm is certain about its belief the number of particles can be reduced because they are all packed together at the current belief which does not improve the accuracy anymore.

A further reason why the AMCL is very popular is that with the small addition of inserting a few random samples during the resampling step, the algorithm can solve the kidnapped robot problem. This is not only useful for small robots which can easily be

```
1  Algorithm likelihood field range finder model(z_t, x_t, m):
2    q = 1
3    for all k do
4      if z_t^k ≠ z_max
5        x_{z_t^k} = x + x_{k,sens} cosΘ − y_{k,sens} sinΘ + z_t^k cos(Θ + Θ_{k,sens})
6        y_{z_t^k} = y + y_{k,sens} cosΘ + x_{k,sens} sinΘ + z_t^k sin(Θ + Θ_{k,sens})
7        dist^2 = min { (x_{z_t^k} − x')^2 + (y_{z_t^k} − y')^2 | ⟨x', y'⟩  occupied in m }
8
9        q = q ( z_hit · prob(dist^2, σ_hit) + z_random / z_max )
10   return q
```

Listing 3.7: The likelihood field range finder model. Source: [TBF05]

```
1  Algorithm MCL(X_{t-1}, u_t, z_t, m):
2    X̄_t = X_t = 0
3    for m = 1 to M do
4      x_t^[m] = sample_motion_model_odometry(u_t, x_{t-1}^[m])
5      w_t^[m] = liklehood_field_range_finder_model(z_t, x_t^[m], m)
6      X̄_t = X̄_t + ⟨x_t^[m], w_t^[m]⟩
7    endfor
8    for m = 1 to M do
9      draw i with probability ∝ w_t^[i]
10     add x_t^[i] to X_t
11   endfor
12   return X_t
```

Listing 3.8: The Monte Carlo Loaclization algorithm. Source: [TBF05]

carried around by people. It is also useful for all robots to recover from delocalization.

3.3 Cross Correlation

The cross correlation is a method from the field of signal processing. It quantifies the similarity of two signals with respect to a time displacement τ between them. If $x(t)$ and $y(t)$ are two time continious signals the cross correlation $R_{xy}(\tau)$ is defined as in Equation 3.11 [Ler12].

$$R_{xy}(\tau) = \lim_{T \to \infty} \frac{1}{2T} \int_{-T}^{T} y(t)x(t+\tau)dt \tag{3.11}$$

This correlation function reaches its maximum at the value τ where the signal $x(t+\tau)$ and $y(t)$ are most similar. To extract the time displacement t_{delay} between the most

similar signals, it is necessary to take the maximum argument function of the correlation function.

$$t_{delay} = \underset{\tau}{\operatorname{argmax}} R_{xy}(\tau) \tag{3.12}$$

This idea is the basis for many time measurement applications. One example from [Bec81] would be a flow meter where two spatial displaced sensors measure the same property but time-delayed because of the displacement. Or sensors that use the principle of time of flight where a signal is sent out and received later (Sonar, LIDAR).

A nice property of the cross-correlation function is that it is robust against white noise if $y(t)$ and $x(t)$ have similarities and the observation time T is long enough. To explain this property, let us assume $y(t)$ is the same signal as $x(t)$ but time-delayed and affected with white noise ϵ.

$$y(t) = x(t - t_1) + \epsilon \tag{3.13}$$

Placing this into the definition of the correlation would result in Equation 3.14.

$$R_{xy}(\tau) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} (x(t - t_1) + \epsilon)x(t + \tau)dt \tag{3.14}$$

Given the additivity of the integral, this can be rewritten like the following.

$$R_{xy}(\tau) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} x(t - t_1)x(t + \tau)dt + \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} \epsilon x(t + \tau)dt \tag{3.15}$$

The first integral in the Equation 3.15 will reach its maximum at $\tau = -t_1$ because it is the same signal only delayed about t_1. The second integral of the Equation 3.15 is a correlation between white noise and a signal. Since this signals have no relation, the correlation value goes to 0 especially if T goes to infinity.

The Figure 3.6 displayes the two assumed signals and Figure 3.7 shows the cross correlation function $Rxy(\tau)$ of it.

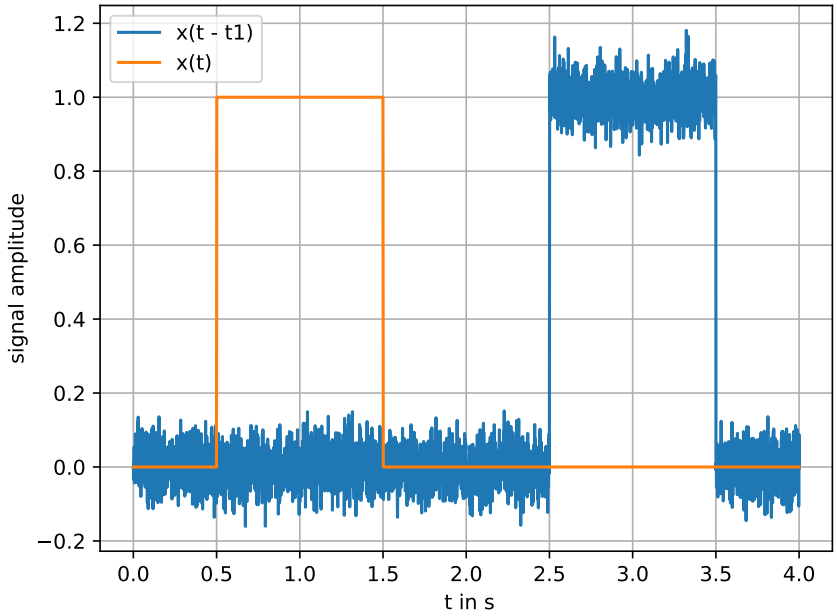


Figure 3.6: Signal x_t and its delayed and noise version y_t.

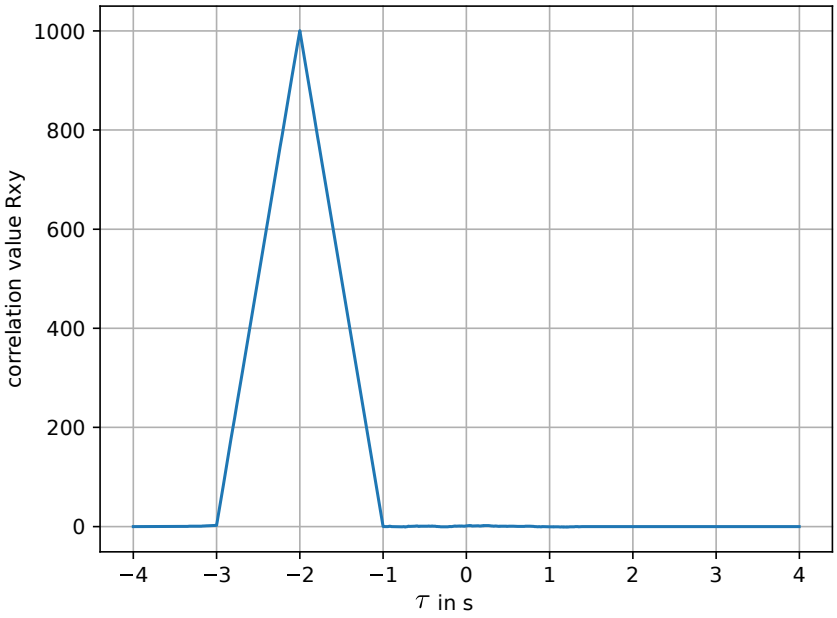


Figure 3.7: Cross correlation of x_t and y_t.

4 Related Research

The temporal alignment of sensors is a hot topic in the literature for the last decade. There are many different approaches developed to tackle this problem. The most popular ones are discussed in the upcoming section.

This chapter also covers literature research about how odometry data can be extracted from a laser scanner and which methods are popular to evaluate the accuracy of a localization system.

4.1 Estimation and calibration of time inaccuracies

The calibration of various sensor setups useable for different applications became a big research topic in the last decade. Many early references ([ALD03], [LD07], [MR06]) focused on the extrinsic and intrinsic calibration of sensors either offline or online. More recent papers state that the temporal calibration is as important as the extrinsic calibration ([Ols10], [RSF16], [ZI04], [HSK14], [KS14]).

The best way to align sensors in time would be if each sensor would run on a real-time platform, has an accurate clock for time-stamping and this clock can be synched to a common clock of the robot. A real-time platform is necessary to guarantee a maximum processing time between an event that occurred and the corresponding data receives its timestamp. An accurate clock is necessary to avoid clock drifts between synchronization points and the synchronization mechanism is necessary to get all different sensors to a common clock. Unfortunately, these sensor types are more complex and harder to set up because of this synchronization mechanism compared to the sensor types used in today's applications.

The commonly used sensors do not support any kind of hardware or software synchronization. They simply output their data to the communication interface as soon as they collected it. The timestamp is only added to the data when it is received at the host PC. This introduces delays and jitter from the measuring and communication process and led to the development of new techniques. This techniques can be split into three major approaches which are covered subsequently in the following subsections.

4.1.1 Map clarity

A simple method for a robot to generate a map is when an intrinsic sensor, for example, a wheel encoder, and an extrinsic sensor, for example, a laser scanner, are combined. The encoder delivers odometry data, or in other words, the pose of the robot in a world coordinate system. The endpoints of each laser beam in a laser scan can be transformed

into the world coordinate system given the robot pose's. These endpoints represent obstacles like walls from the environment. If the robot moves and these obstacles are stored over many subsequent laser scans it generates a map.

In the ideal case of a static environment and synchronized sensors, a straight wall should be represented as one straight line of points in the map. If a robot moves closer to a wall, the position of the robot should change the exact same amount as the distances of the laser measurement to its environment reduces. This leads to exactly the same point in the map of subsequent laser scans.

In the case that the sensors are not well aligned in time, the endpoints of the laser scanner get transformed to the map at the wrong robot pose. This is especially observable during acceleration and deceleration and leads to a blurred or unclear map. The following papers use this indication and estimate the time delay by rerunning the same measurements for different time delays and search for the clearest map.

An example of this idea can be found in [VWW18]. Here the authors used a tilting motor and a 2d laser range scanner to build a tool for 3d mapping. The laser data gets transformed with the motor encoder data from the same timestamp and in the ideal case, there would not be any motion in the generated map. They change the timestamp by an offset until they get the clearest map.

Another example is [SHN14] where three 2d laser range scanners were mounted on top of a rotating plated. Again with the movement of the 2d laser scanners, it should enable to get a 3d map from the surrounding. They introduce the Rnyi Quadratic Entropy metrics for the map clarity and use it as a cost function in an optimization problem.

Concluding this approach to estimate the timestamp offset, all use a still standing mechanic where a motor is tilting or rotating a laser scanner around one axis. This is a problem for the usage at a shuttle because the shuttle needs to move to get movement in the laser scan and this movement of the shuttle already introduces new error sources. Additionally, rerunning the same measurement with many different delay values is computationally expensive and this method only works for a combination of a laser scanner with odometry but we search for a more general approach to be able to also synchronize IMUs or cameras.

4.1.2 Delay as a state variable

The calibration of the spatial alignment between two sensors is often solved via a batch-optimization framework ([Fle+11], [QP04]). The methods in this subsection use such optimization frameworks and introduce the time delay as an additional state variable by assuming that the time delay is constant.

The method described in [FRS13] calibrates an IMU and a camera temporally and spatially at the same time. The camera and IMU are rigidly connected and waved in front of a checkerboard to be able to extract the ego-motion of the camera from the image stream. The calibration method uses basis functions for the time-varying states

which enable the usage of the maximum likelihood estimation framework to extract the desired parameters.

Another approach is described in [QS18] where they use features in the environment and the optimization framework of Simultaneous Localization and Mapping (SLAM). This has the advantage compared to the method of [FRS13] that it does not need a checkerboard and can be done online.

A drawback of both mentioned methods is that they are developed for calibrating a combination of a camera and an IMU. But our target application does not use a camera. It uses an IMU, a laser scanner and wheel encoders.

The approach of [Reh+14] solves this problem for the laser scanner and incorporates it by detecting planes in the laser scan and extracting the trajectories of the movement. But it still is not useable for encoder values and it needs a well-known environment with several planes which are not given in our target application where the shuttles are used in many different environments.

A further disadvantage of these methods is that they calibrate the uncorrelated properties of the time delay and the spacial sensor displacement at the same time. This unnecessarily increases the search space for the optimization algorithm which can negatively affect the result [Mai+11]. In general it is simpler and more accurate to calibrate the time delay and the frame transformations in two separate steps.

Finally, the assumption that the time delay is constant for a short period of time seems feasiable but it is not over a longer period of time (e.g. days, months). Factors like CPU usage at the host PC, different temperature behavior, bus traffic or startup sequences will affect the time delay. Therefore, it should be possible to online calibrate it during the normal application workload without the need for special hardware or driving routines. This renders the methods using the delay as a state variable infeasiable.

4.1.3 Stream comparison

The basic idea behind the following methods is that the data of different sensor types are transformed into a common representation related by time. For example, the angular velocity of a robot is extracted from an odometry sensor and an IMU sensor. These velocity data streams then can be compared because if the sensors are rigidly connected the data streams of the different sensors should have a similar shape but with an offset in time.

A simple approach using this idea is explained in [ZI04]. In this paper, they detected events in two different sensors when the robot's state changes from still standing to moving. Every event then has a timestamp from each sensor and by subtracting them they receive a guess for the true delay between the sensors. It is only a guess because the sensors have different sampling periods and therefore this delay can lie inside an interval of the sum of both sampling periods. Incorporating many of these events reduces this interval and converges it to the true delay between the two sensors.

There are two drawbacks of this method concerning our application. The event detection from standing to moving in an IMU sensor is not straightforward given to noise and there are at least fifty events necessary such that this method converges. Collecting fifty stop-start events can take some time in the target application.

The Time Delayed Iterative Closest Point (TD-ICP) method is described in [KS14]. To use this method, the orientation velocity of each sensor gets extracted in a preprocessing step. This delivers a curve in the three-dimensional orientation space of pitch, yaw, and roll for each sensor and the curves get aligned via the ICP method but also incorporate the time delay. The conversion into orientation velocity is advantageous because it does not need to know the spatial transformation between the sensors. Therefore, this method can be used as the first step of calibration. The TD-ICP additionally delivers the orientational displacement of the sensor. With the time delay and the orientational displacement given, the calibration of the translational displacement is much simpler.

Nevertheless, the TD-ICP is computationally expensive which makes it impractical for our application where the calibration should be performed on the local host PC of the shuttle. Furthermore, this method was invented for robots that can rotate freely in the 3D space but the target shuttles are only possible to rotate around the z-axis because they are moving on an x-y plane. This makes this method an overkill for our application.

The next method again uses the rotational velocities from different sensors of a robot to use the advantage of their independence of the spatial transformation. But instead of using the three rotational velocities on their own, they are combined to one absolute rotational velocity in the papers [Mai+11] and [HSK14]. This delivers a simple data sequence over time for each sensor. The delay between two sequences is then estimated by using the cross-correlation function. To be more precise, the maximum argument of the cross-correlation function delivers the time delay between the two sequences.

The cross-correlation is used for decades in the field of sonar measurements where the time delay between an out sent signal and a similar later received signal is measured. The cross-correlation is popular in this field because of its robustness against noise (see explanation at section 3.3) and its simplicity. This simplicity reflects in a low computation complexity and therefore enables an online calibration at the host PC.

A further advantage of this method concerning our application is that the used velocity streams are not very restricted which allows the shuttle to use sensor measurements during its normal application workflow. It only needs to record several seconds of maneuvering where the velocity of the shuttle changes. This is given in the target application because the shuttle needs to accelerate and decelerate at the stations it is driving to and to avoid obstacles.

To sum up, the method with rotational velocities and the cross-correlation is independent of the spatial transformation of the sensor, has a low computational complexity which makes it useable for online calibration, is robust against noise and ocan work with data already available in our application. This makes it convenient for the target application and is therefore used in this thesis.

4.2 Laser odometry

To be able to use the velocity stream comparison method, the velocity streams from each sensor need to be extracted. For the encoder sensors of a mobile robot it is straightforward because, if the motion model from Section 3.2.3 is used, they deliver odometry data. The extraction of the odometry data from a laser scanner sequence is more advanced. The basic idea behind laser odometry is to compare subsequent laser scans and extract the ego-motion of the laser scanner. There are two popular methods available in the literature.

The first method for this comparison is the iterative closed point (ICP) algorithm which is described in [BM92]. To use this method the laser sensor data from two subsequent scans needs to be transformed into two point clouds. Then the algorithm aligns the point clouds in a plane (or space in the 3D case) such that the distance between point correspondences is minimized. The output of the algorithm is the necessary transformation for this alignment. The sequence of these transformations represents the ego-motion of the laser scan sensor.

This ICP algorithm is computationally expensive especially if many scan points are used. An improved version of this algorithm was invented by Censi [Cen08] where the author uses a different metric for the alignment process. This algorithm is widely used for registration tasks and is implemented in the laser_scan_matcher ROS package. It achieves an acceptable accuracy for the odometry with around 5% RMSE [SJB02] but it is still not optimal concerning the computational expense.

The second method uses a range flow-based approach to calculate the ego-motion of a laser scanner. It was first mentioned in the paper [HS81] and the basic idea behind this algorithm is that the laser scanner ego-motion can be extracted from the motion of scan points. This range flow-based approach got optimized in the paper [SJB02] for the application of odometry estimation of a mobile robot in an environment with dynamical obstacles. This paper additionally presents data from the comparison with the ICP-method and shows that it is with 1% RMSE more accurate and at the same time nearly 20 times faster. This makes it applicable for our application and is therefore used in this thesis.

4.3 Accuracy of a localization

After the time delays between the sensors are calibrated, the effect on the localization process should be evaluated. In the literature ([Röw+12], [ZS18]) there are three methods described to quantify the accuracy of the localization. All of them need a ground truth from a motion capturing system (Mocap).

In the first method, several positions in the room are used as end positions to which the robot should drive to. The robot moves from one position to another and stops

at each position for several seconds. This enables to quantify how accurately a robot can locate itself at certain positions and the stopping compensates timing misalignments between the robot and the Mocap system. It is a simple and stable method but we also want to know how good the localization of the robot works during movement because the highest errors were observable during the movement of the robot.

The second method uses the relative position error. With it, the difference between two subsequent positions is compared from the robot's localization and the Mocap. It is beneficial because the exact spatial transformation between the robot and the Mocap frame is not needed and it is able to deliver quantities during the movement of the robot. Another advantage is that it does not integrate errors over time which makes the errors at each time point weighted equally. Nevertheless, the quantity this method delivers is not feasible for what it means in absolute values. It is only useful for comparison within the same experiment but not over many different experiments or robots.

The last method is the absolute position error. It has the drawback that the spatial transformation between the robot and the Mocap system needs to be given but then it delivers an absolute error value of the accuracy which can be compared over many experiments and robots. And in the target application, we want to compare different robots which leads to the conclusion that the absolute position error should be used.

5 Concept

The previous chapter explained why the estimation of the delay should be done with the velocity stream comparison method. This chapter introduces this method in more detail and explains all necessary preprocessing steps. But before we present the details a course overview is given for orientation purposes.

5.1 Overall concept

As the name of the velocity stream comparison method suggests, the method can compare two velocity streams with each other and extracts the time delay between them. Therefore, this method needs to be applied to each pair of sensors in a robot. It is convenient to compare all sensors of a robot against one sensor whose time is assumed to be the true time the robot is working on. We selected to wheel odometry sensor for this true time and a laser scanner as the second sensor to explain the concept of the time delay estimation. This is then applied to all sensors of the robot in the next chapter.

The sensor data will go trough three separate processing steps in our concept. The first step is used to correct the timestamps by eliminating jitter and buffering effects from the communication process. This is shown in Figure 5.1 with the yellow boxes. The second step is the conversion of each sensor data to a velocity stream. This step is individual for each sensor type because each sensor type delivers a different kind of data. It is represented in Figure 5.1 with the teal and blue boxes respectively. The final step is represented by the purple box and is the velocity stream comparison step. In it the two separate streams are compared over time and, with the help of the cross-correlation, the time delay between the wheel odometry and the laser scanner t_{wl} gets extracted. The following sections describe the separate processing steps in more detail.

5.2 Timestamp correction

The quality of time delay estimations based on timestamped data is sensitive to the correctness of the timestamps. The best case would be if the sensor itself adds a timestamp from a systemwide synchronized clock to the data when it finished the measurement. Unfortunately, the typical sensors in the target application do not provide this feature.

The majority of sensors send out the data when they finished the measurement and the timestamp is only added by the host PC which further processes the data. Inbetween the sensor and the host PC lie different communication and preprocessing systems like it is shown in Figure 5.2. This is only one example. It could also be that the sensor is

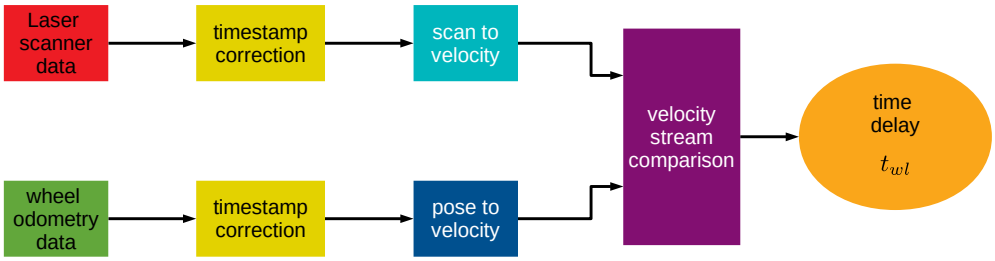


Figure 5.1: Overview of the delay estimation concept.

directly connected to the host PC via a serial communication or the host PC has access to the CAN bus. Nevertheless, each communication and preprocessing step introduces delays.

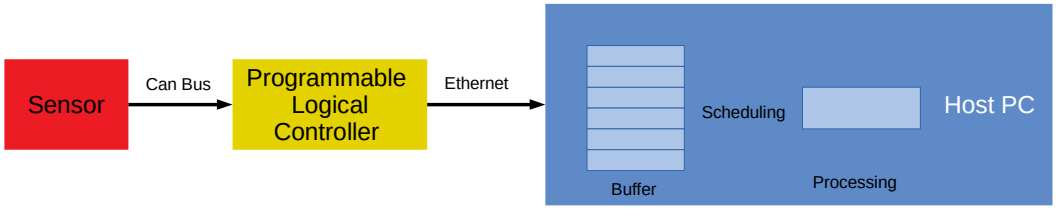


Figure 5.2: Sensor pipeline from measurement until it receives a timestamp.

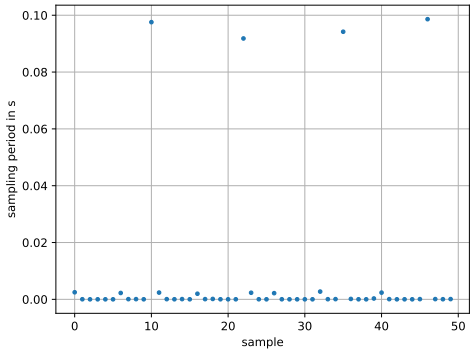
If this sensor would be the only one in the robot system, has the communication system to the host PC for its own and the host PC has nothing else to do then processing the data from this one sensor, the delay between the sensor measurement and the timestamping at the host PC would be constant. This constant delay would make problems if it exceeds several hundred milliseconds because of latency issues. In the normal case, this constant delay stays below some hundred milliseconds and can be handled with the time delay estimation between two sensors in the next section.

Nevertheless, a robot with only one sensor and no other tasks to execute would be kind of useless. Multiple sensors typically share a common communication system which can lead to a buffering behavior if the communication system is blocked at the moment the sensor wants to send data. An extreme example of the buffering effect is shown in Figure 5.3a because the timestamp period, the interval between two consecutive timestamps, is zero when the communication system was free and it was higher than the sensors sampling period during the time the communication system was blocked.

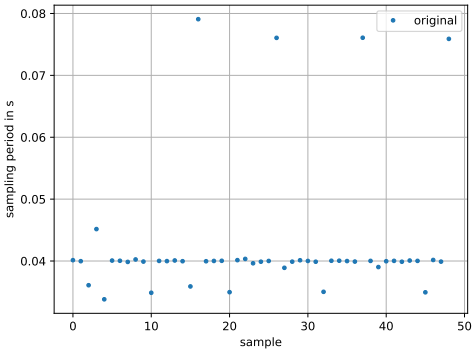
Another problem is that data packages could get lost during the communication. The effect of a package loss on the timestamp period is that the period of a certain sample

is a multiple, number of lost packages, of the sensor sampling period. This is shown in Figure 5.3b.

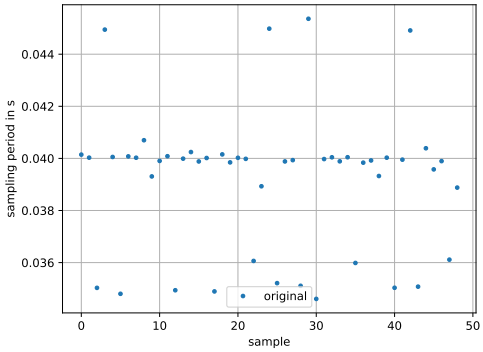
Also, many different tasks will be processed by the host PC which inevitably leads to the prioritized scheduling behavior of tasks. This scheduling introduces jitter in the timestamp periods and is shown in Figure 5.3c.



(a) Timestamp periods affected by buffering. Sampling period: 0.00833



(b) Timestamp periods affected by sample loss. Sampling period: 0.0392



(c) Timestamp periods affected by jittering. Sampling period: 0.0392

Figure 5.3: Errors affecting the timestamp periods of the sensor data.

These inaccuracies in the timestamps can and need to be corrected to achieve the best results with the velocity stream comparison. The basic idea behind the timestamp correction is that the sensors execute their measurements at a constant time period. Thus, the timestamps at the host PC should also have a constant time period.

To achieve this, three steps need to be done. The first step is to counteract the package loss. This can only sufficiently be solved if the sensor or the sender at the communication system adds an ID, in most cases a counter, to the data package. If no counter is included, a preknowledge about the expected sampling period is necessary.

With this knowledge, it is possible to estimate how many packages are lost since the last data was received if the data is not affected by buffering issues. Luckily, all our sensors provide such a counter and make it unneeded to further dive into this estimation.

With the knowledge of how many packages got lost, it is possible to calculate the number of measurement periods since the last data package was received by subtracting the last message number from the current one. This is implemented in the first part, line numbers 1-5, of the constant timestamper Algorithm 1.

The next step is to extract the sensor sampling period. The sampling period δ_p can be estimated from current timestamp $t_{current}$, the last timestamp t_{last} and the number of periods $n_{periods}$ with $\delta_{current} = {}^{(t_{currentRaw} - t_{lastRaw})}/_{n_{periods}}$. Putting several subsequent sampling period estimations into a simple moving average filter with an appropiate filter window size will extract a rather good true sampling period δ. This period calculation and moving average filtering is implemented in the line numbers 6-16.

The final step is to calculate the new timestamp t_{new} for the current data package. This is done by adding the average sampling period δ to the last used timestamp t_{last} with $t_{new} = t_{last} + \delta \cdot n_{periods}$. But this is only allowed if the moving average filter is already converged which means that the averaged sampling period δ is close to constant. This is implemented in line numbers 17-23 of the constant timestamper algorithm.

5.3 Data conversion

To make the data from different sensors of a mobile robot comparable, the same characteristic needs to be extracted from it. The selected common characteristic is the translational v and rotational ω velocity of the robot. This extraction is different for each kind of sensor and will be explained for the laser scanner, the wheel odometry, the IMU and the motion capturing system in the subsequent sections.

5.3.1 Laser scanner

The basic idea behind the extraction of the robot velocity from a laser scanner is to compare subsequent laser scans and extract the ego-motion from the laser scanner given the assumption that the scanned environment is static and rigid. The ego-motion of the scanner can then be simply converted into the motion of the robot by applying the static transformation between the scanner and the robot coordinate system.

The Range Flow 2D Odometry (RF2O) is such a tool to extract the ego-motion of the laser scanner. Its extraction method is based on the evaluation of the velocities of scan points in subsequent laser scans. Figure 5.4 illustrates the motion of a scan point between two laser scans.

The velocity of this scan point V_P can be obtained from the motion of the laser scanner as:

$$V_P = -w \cdot r \begin{bmatrix} -sin\Theta \\ cos\Theta \end{bmatrix} - \begin{bmatrix} u \\ v \end{bmatrix} \tag{5.1}$$

Algorithm 1: Constant timestamper

```
   Data: current_message_number, raw_timestamp
   Result: new_timestamp
 1 number_of_periods = current_message_number - last_message_number;
 2 if checkNumberOfPeriodsNotPlausible( number_of_periods) then
 3 |   return raw_timestamp;
 4 end
 5 last_message_number = current_message_number;
 6 period = (raw_timestamp - last_raw_timestamp) / number_of_periods;
 7 last_raw_timestamp = raw_timestamp;
 8 for number_of_periods do
 9 |   moving_average_sum += period - moving_average_data[current_index];
10 |   current_index++;
11 |   moving_average_data[current_index] = period;
12 |   if current_index ≥ filter_size then
13 |   |   current_index = 0;
14 |   end
15 end
16 average_period = moving_average_sum / filter_size;
17 if isFilterConverged( average_period ) then
18 |   new_timestamp = last_timestamp + number_of_periods * average_period;
19 else
20 |   new_timestamp = raw_timestamp;
21 end
22 last_timestamp = new_timestamp;
23 return new_timestamp;
```

where w is the rotational velocity and u and v are the translational velocities of the laser scanner. r represents the range value and Θ represents the angle of one laser scan in the sensors coordinate system.

A second way to express the velocity V_P is to take the derivative of the Point P and use $\dot{r}$ and $\dot{\Theta}$ as the points polar velocity.

$$V_P = \frac{dP}{dt} = \begin{bmatrix} cos\Theta & -rsin\Theta \\ sin\Theta & rcos\Theta \end{bmatrix} \cdot \begin{bmatrix} \dot{r} \\ \dot{\Theta} \end{bmatrix} \tag{5.2}$$

Combining the Equations 5.2 and 5.1 results in the following:

$$0 = \begin{bmatrix} cos\Theta & -rsin\Theta \\ sin\Theta & rcos\Theta \end{bmatrix} \cdot \begin{bmatrix} \dot{r} \\ \dot{\Theta} \end{bmatrix} + w \cdot r \begin{bmatrix} -sin\Theta \\ cos\Theta \end{bmatrix} + \begin{bmatrix} u \\ v \end{bmatrix} \tag{5.3}$$

The Equation 5.3 consists of two unknowns from the scan point ($\dot{r}$,$\dot{\Theta}$) and three unknowns from the sensor motion (u,v,w). With the assumption of the static environment

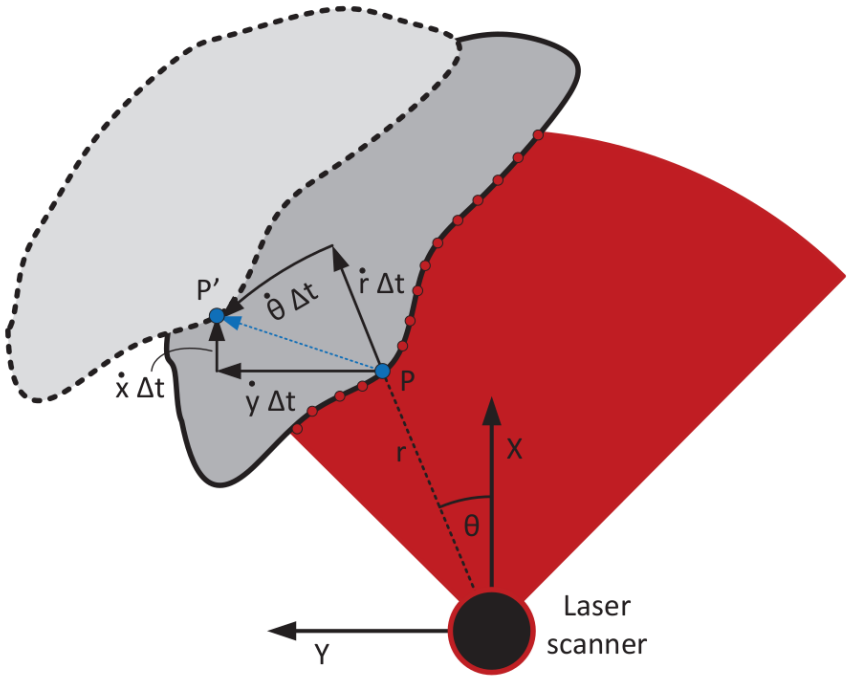


Figure 5.4: Motion of a scan point P from two subsequent scans (Source:[JMG16]).

the unknowns from the scan point are dependend via the velocity constraint Equation (5.4). The derivation of this equation is explained in [SJB02].

$$\dot{r} = r_\Theta \dot{\Theta} + r_t \tag{5.4}$$

r_Θ and r_t are the partial derivatives of the range function $R(t, \Theta)$ which can be observed from a sequence of scan data.

Including this constraint in Equations 5.3 delivers the final Equation 5.5

$$0 = \begin{bmatrix} cos\Theta & -rsin\Theta \\ sin\Theta & rcos\Theta \end{bmatrix} \cdot \begin{bmatrix} r_\Theta \dot{\Theta} \\ \dot{\Theta} \end{bmatrix} + \begin{bmatrix} cos\Theta & -rsin\Theta \\ sin\Theta & rcos\Theta \end{bmatrix} \cdot \begin{bmatrix} r_t \\ 0 \end{bmatrix} + w \cdot r \begin{bmatrix} -sin\Theta \\ cos\Theta \end{bmatrix} + \begin{bmatrix} u \\ v \end{bmatrix} \tag{5.5}$$

With the Equation 5.5 each scan point introduces two equations but only one new unknown ($\dot{\Theta}$). Therefore, if three scan points are used, all unknowns can be determined and the sequence of w, v and u delivers the desired laser scanner velocity.

Of course, relying the calculation only on three scan points, which are subject to measurement noise and quantization errors, is not sufficient for a useful result. But laser scanners typically deliver 100+ scan points for each scan and the results of the Equation 5.5 are combined in a least-squares problem. This can be solved with an appropriate computational expense and provides a stable result in moderate environments.

5.3.2 Wheel odometry

The wheel encoder sensors of a robot deliver counter values of how many encoder ticks have occured since the last data request. These counter ticks get converted in a pre-processing step into wheel velocities. These wheel velocities are then sent to the host PC.

To convert the wheel velocity values from the sensors into the robot's translational and rotational velocity, it is necessary to know the kind of locomotion the robot uses. Many robots, and our target application as well, use a differential drive locomotion because of its simplicity. A differential drive robot consists of a left and a right wheel as it is shown in Figure 5.5. The center of the rotation of the robot is exactly at the half distance between the two wheels, which is called wheelbase b. Conveniently, the origin of the robot frame is defined at this center point of the rotation and the orientation of the robot frame is normal to the wheel axis.

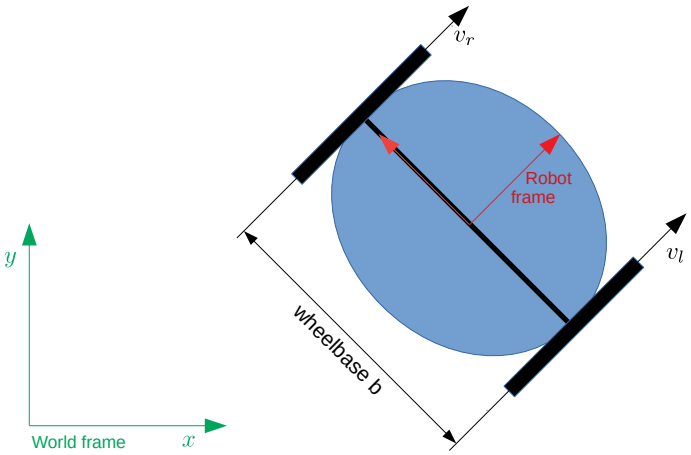


Figure 5.5: Locomotion of a differential drive mobile robot.

In this case the translational velocity $v_{wo,i}$ and the rotational velocity $\omega_{wo,i}$ of the robot from the wheel odometry sensor can be calculated with the Equations 5.6 and 5.7. The index i stands for the current descrete value.

$$v_{wo,i} = \frac{v_{l,i} + v_{r,i}}{2} \tag{5.6}$$

$$\omega_{wo,i} = \frac{v_{r,i} - v_{l,i}}{b} \tag{5.7}$$

5.3.3 Inertial Measurement Unit

An Inertial Measurement Unit (IMU) delivers the acceleration data along the x, y, z axes at its own frame (a_x^{IMU}, a_y^{IMU}, a_z^{IMU}) and the rotational velocities around these axes ($\dot{\phi}$, $\dot{\psi}$, $\dot{\Theta}$).

A convenient way to place such an IMU sensor on a robot is to align the frame of the sensor with the frame of the robot. This makes it possible that no static transformation is necessary and the rotational velocity of the robot from the IMU sensor ω_{IMU} can directly be used from $\omega_{IMU} = \dot{\Theta}$.

Because the frames of the IMU and the robot are aligned, the robot's translational velocity from the IMU sensor v_{IMU} is the velocity of the IMU sensor along its x-axis v_x^{IMU}. The v_x^{IMU} can be calculated from the acceleration data a_x^{IMU} by integrating it over time. This integration needs to be approximated because the computers do not work in continuous time. They work in discrete time and the sensor data is delivered subsequently. Therefore, the current velocity value can be calculated with the following equation where δ represents the sampling period of the sensor.

$$v_{IMU,i} = v_{IMU,i-1} + a_x^{IMU} \cdot \delta \tag{5.8}$$

Of course, this approximation of the integral introduces errors that sum up over time. Nevertheless, we only need velocity snippets of short periods of less than 30 seconds such that this error can be neglected.

5.3.4 Motion capturing system

The motion capturing system (Mocap) uses reflecting markers on the mobile robot to capture its motion. These markers can be combined to a rigid body in the Mocap system and each rigid body has a definable pivot point. This pivot point is set to the center of the robot again. With this, the Mocap system delivers straight away the robot pose. Although these pose values are in the Mocap reference frame, the absolute translational v_{mc} and the rotational ω_{mc} velocity of the robot from the Mocap system can be directly calculated. To explain this behavior we need to take a look at how the velocities are calculated.

The translational velocity v_{mc} is calculated from the Euclidean distance between two subsequent poses ($[x_i \quad y_i]^T$, $[x_{i-1} \quad y_{i-1}]^T$) and the time difference between the timestamps of the two poses (t_i, t_{i-1}) with the Equation 5.9. This euclidean distance is independent of the frame. If we think of the robot velocity as a vector in both different frames, the absolute translational velocity v_{mc} represents the length of this vector and this length is the same in both frames.

$$v_{mc,i} = \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{t_i - t_{i-1}} \tag{5.9}$$

The same is valid for the rotational velocity of the robot from the Mocap system ω_{mc} because also in this calculation only the difference between two consecutive angles θ is of interest and not the value of the angle itself.

$$\omega_{mc,i} = \theta_i - \theta_{i-1} \tag{5.10}$$

5.4 Estimation of the delay

As the velocity streams, v and ω of each sensor are now available, the time delay between two velocity streams should be estimated. To formulated this more precisely, the translational velocity streams $v_{wo}(t)$ and $v_{ls}(t)$ are similar and time-shifted by the delay $\delta_{wo,ls}$ such that $v_{wo}(t) \sim v_{ls}(t - \delta_{wo,ls})$. Therefore, we search for a $\delta_{wo,ls}$ that maximizes the similarity between the two velocity streams. This appears straightforward in the continuous-time domain with analytically expressible signals. But to be able to implement it on a real shuttle some things need to be taken care of.

First of all, the velocity streams are in discrete-time and continue from the start of the robot until it is switched off. An example of typical velocity streams of shuttle robots is shown in Figure 5.6 where it drove from one station to another several times. The values of these velocity streams need to be stored to be able to apply the correlation function on it. It is neither possible to store the whole velocity stream from each sensor in a robot nor it is needed. Segments from the whole stream of a few seconds are already sufficient for the correlation function. It is only important that the selected segment includes a change of the velocity otherwise the delay can not be estimated.

The concept for this segmentation is to extract such a typical acceleration and deceleration "peak" from the velocity streams like is shown in the Figure between 30 and 50 seconds. To do so a threshold is selected and if the current velocity value is higher than this threshold the recording begins and if the velocity drops below this threshold again the recording stops. The threshold should be selected that it is not already triggered from the noise around zero velocity. To still include the transition from standing still to moving one second before and after the recording is also stored and used in the following steps. This is still possible to implement because the number of samples for the one second before the start of the recording can always be stored in a ring buffer and copied to the recording when the start was triggered. An example of such a segmented peak is shown in Figure 5.7.

The next problem that occurs on real shuttles is that the temporal resolution of the sensors, in the range of one to 100 Hz, is low compared to the delay values between the sensors which we want to estimate. The low temporal resolution causes problems with the cross-correlation because it can only shift the input signals by one sampling period and not by a fraction of it. This would make the correlation for a laser scanner with a sampling period of 120ms and a delay to wheel odometry of only 10ms useless. The values are taken from real examples. Additionally, the different sampling rates of the sensors make a problem because the correlation function does not incorporate the time point when a sensor value was taken. It simply assumes that the values of each input stream were taken at the same constant sampling rate.

To solve these problems, the segmented velocity streams need to be interpolated in time. The concept generates a new time vector for the selected segment with a constant sampling period of one millisecond and linearly interpolates the velocity streams to this time vector. One millisecond should, on the one hand, be accurate enough because a

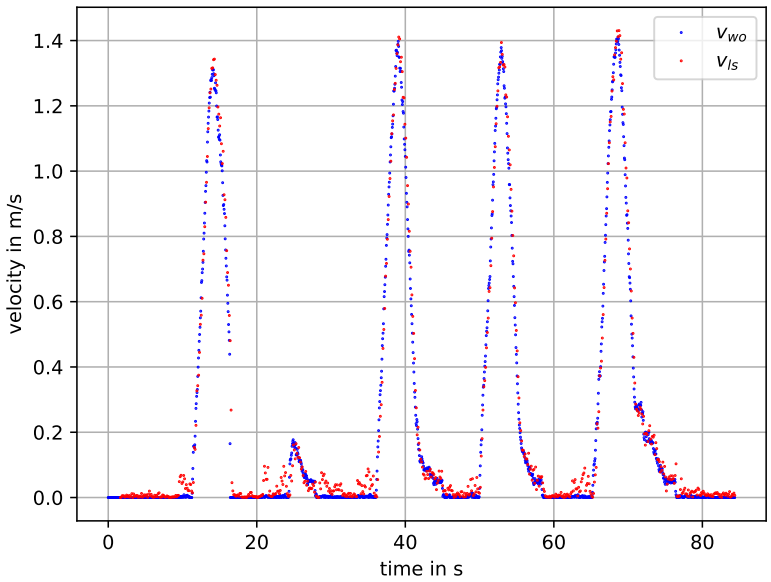


Figure 5.6: Translational velocity streams over a period of 80 seconds of a wheel odometry and a laser scanner from typical robot movement with the sensors orignial sampling period.

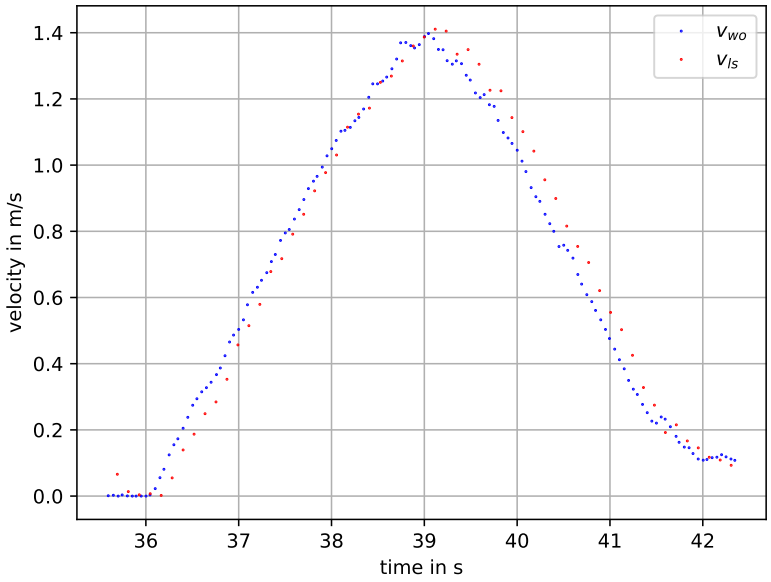


Figure 5.7: One segmented peak of the streams.

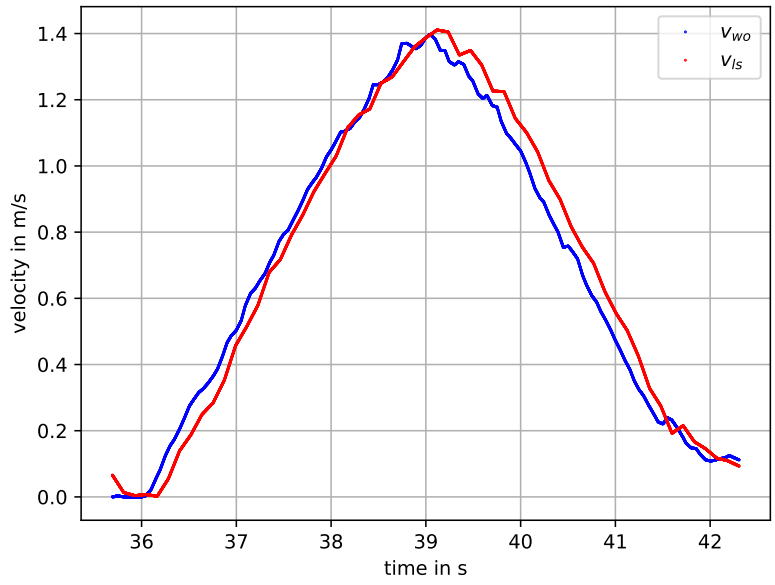


Figure 5.8: Interpolated velocity streams from a wheel odometry and a laser scanner.

delay value of several tens of milliseconds is expected and, on the other hand, be computational and storage vise inexpensive enough to execute it online on the shuttle. An example of such a interpolated segmented is shown in Figure 5.8.

With these problems solved, only the cross-correlation function 3.11 needs to be transformed into the discrete-time domain and adapted for a finite number of samples. For this case the cross-correlation is defined as in Equation 5.11 with y_1 and y_2 representing the input signals, N the number of samples in the input signal and m the number of samples the signal is shifted.

$$R_{y_1,y_2}[m] := \begin{cases} \frac{1}{N}\sum_{n=0}^{N-m-1} y_1[n]y_1[n+m] & for \quad m \geq 0 \\ \frac{1}{N}\sum_{n=-m}^{N-1} y_1[n]y_1[n+m] & for \quad m < 0 \end{cases} \tag{5.11}$$

$R_{y_1,y_2}[m]$ is the correlation function, or similarity of the signals y_1 and y_2, over the displacement variable m. This function seems to be computationally expensive because of the summation of the signal values for each displacement value of m. Luckily, it is not expensive in our application because we only use short periods of velocity streams. Additionally, the correlation function is similar to the convolution function and the calculation of this is highly optimized in the used scipy package with the usage of a Fast Fourier transform.

To extract the time value of the delay we again need to use the argmax function which delivers the number of samples for the best shift. This needs to be multiplied with the

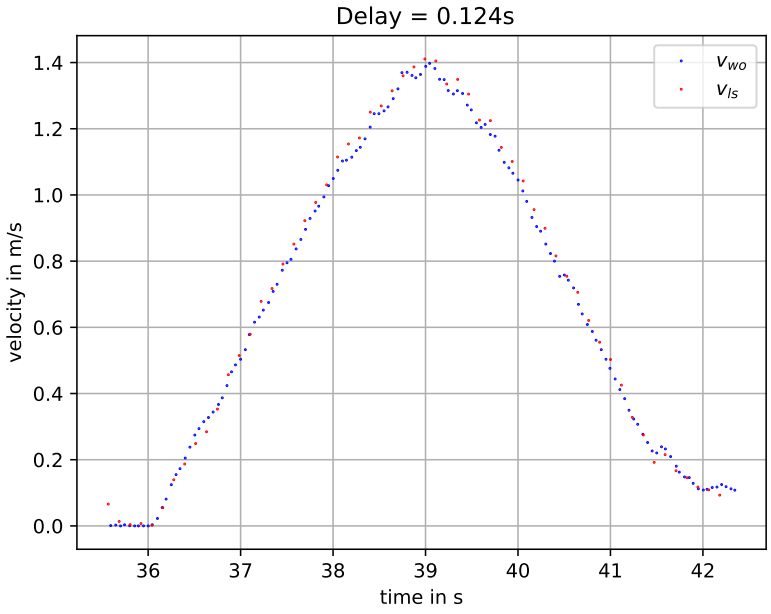


Figure 5.9: Aligned velocity streams of a wheel odometry and a laser scanner with there original sampling period and the delay value subtracted from the laser scanner timestamps.

sampling period, in our case the selected one millisecond from the interpolation step, to get the time delay δ between the two signals as a time value.

$$\delta = \Delta_{period} \cdot \underset{m}{\operatorname{argmax}} R_{y_1,y_2}[m] \tag{5.12}$$

To aligne the velocity streams v_{wo} and v_{ls} the time delay $\delta_{wo,ls}$ needs to be subtracted from the time vector of the laser scanner with $t_{ls,aligned} = t_{ls} - \delta_{wo,ls}$. An example of such an alignement result is shown in Figure 5.9 where the $\delta_{wo,ls}$ was estimated with 124ms.

6 Implementation

The concept proposed in the last chapter got implemented on real hardware. This hardware is introduced in this chapter and some details of the, for the concept relevant, components are explained. Additionally, the software modules used to realize the concept are explained in the second part of this chapter.

6.1 Hardware

The used hardware in this thesis was provided by the company incubedIT [1]. incubedIT develops a Fleet Management System (FMS) and the software framework for autonomous mobile shuttle robots. To test this software framework, the company has an in-house laboratory with several real mobile robots from their partners. All of these shuttles are given a name and we used the shuttles **Sarah**, **Valentina** and **Marie** to implement and evaluate the concept of this thesis.

Sarah, **Valentina** and **Marie** are depicted in Figure 6.1. They are very similar in their design but differentiate in a few hardware components. A similar design is obvious because they fulfill the same application, which is transporting one box from one station to another.

(a) **Sarah**. (b) **Valentina**. (c) **Marie**.

Figure 6.1: Mobile robots used to implement the concept.

[1] https://www.incubedit.com/, accessed 06.04.2021

Shuttle Name	**Sarah**
Processing Unit	TANK-700-QM67
Processor	i5-2510E
Ram	4gb
Laser scanner	Sick S300
Communication Laser scanner to PC	Serial (RS422)
Motor	Brushless Synchron Servomotor
Encoder	Resolver
PLC	Beckhoff CX9010
Communication PLC to PC	Ethercat

Table 6.1: Hardware specification of the **Sarah** shuttle.

Shuttle Name	**Valentina**
Processing Unit	TANK-700-QM67
Processor	i5-2510E
Ram	8gb
Laser scanner	Sick S300
Communication Laser scanner to PC	Serial (RS422)
Motor	450W Brushless dc motors
Encoder	Incremental encoder
PLC	B&R x20
Communication Encoder to PLC	Can
Communication PLC to PC	Ethernet (UDP)

Table 6.2: Hardware specification of the **Valentina** shuttle.

The common parts of the three shuttles are that they use a differential drive locomotion and four castor wheels, one in each corner, for stabilization. All of them also use wheel encoders and a laser scanner sensor which is mounted in the front center of the robot, a few centimeters above the ground. Another common feature of them is the central processing unit which is always a Tank-700 PC with an intel i5 processor. This common features are also listed in the hardware specifications of each shuttle in the Tables 6.1, 6.2 and 6.3.

6.1.1 Laser scanner

It can be seen in the tables 6.1, 6.2 and 6.3 that all of the three shuttles use a Sick S300 laser scanner via serial interfaces. This scanner, which is shown in Figure 6.2a, works with a rotating mirror to deflect the laser beam at different angles from its center point. The S300 makes a distance measurement at each half degree over a field of view from -135 to +135 degrees. Such a scan is done every 40ms and the distance values are then sent via the serial interface to the host PC. Unfortunately, the serial port at the host PC is only able to cope with a baud rate of 115.2k. With this baud rate the transfer of one scan takes approximately 105ms to complete which is longer than the measurement of one scan needs. Thus, only each third laser scan can be transmitted and used in the

Shuttle Name	**Marie**
Processing Unit	TANK-700-QM67
Processor	i5-2510E
Ram	4gb
Laser scanner	Sick S300
Communication Laser scanner to PC	Serial (RS422)
Laser scanner 2	Sick Microscan3
Communication Laser scanner 2 to PC	Ethernet (UDP)
Motor	450W Brushless dc motors
Encoder	Incremental encoder
PLC	Wago
Communication Encoder to PC	Can
Communication PLC to PC	Can

Table 6.3: Hardware specification of the **Marie** shuttle.

host PC which results in an effective measurement period of the S300 laser scanner of 120ms.

The incubedIT had also a new version of this Sick laser scanner available which is called Microscan3 and is shown in Figure 6.2b. The measurement principle is the same as the one at the S300 but it offers a higher angular resolution, up to 0.1 degrees, a wider view angle, some improved security features and a faster data interface. The Microscan3 uses an Ethernet interface and the UDP protocol which enables it to send each laser scan to the host PC. Thus the effective measurement period of the Microscan3 is 40ms. To test this scanner we attached it right above the S300 laser scanner to the **Marie** shuttle.

(a) S300. [2] (b) Microscan3. [3]

Figure 6.2: Sick laser scanners.

[2]https://www.sick.com/at/de/optoelektronische-schutzeinrichtungen/sicherheits-laserscanner/s300-standard/c/g187239, accessed 28.03.2021

[3]https://www.sick.com/at/de/optoelektronische-schutzeinrichtungen/sicherheits-laserscanner/microscan3/c/g295657, accessed 01.04.2021

6.1.2 Communication / PLC / Encoder

A bigger difference between the shuttle can be recognized in terms of the used PLCs and wheel encoder sensors. The **Sarah** shuttle uses synchro servomotors where the resolver sensors are connected to the encoders of the Beckhoff PLC. The PLC collects the encoder values, calculates the wheel rotation speed and sends it to the host PC via an Ethercat interface.

Marie and **Valentina** instead use DC motors with incremental sensors and standalone encoder cards. But there is a difference between **Marie** and **Valentina**. At the **Marie** shuttle, the encoders and the host PC are directly connected to the CAN bus. At the **Valentina** shuttle, the encoders are connected via the CAN bus to the PLC and the PLC again calculates the wheel velocities and sends them via the Ethernet and the UDP protocol to the host PC with 20Hz.

Of course, the shuttles consist of more parts than explained here, but they are not relevant for this thesis and so they are kept aside.

6.1.3 Motion capturing system

The laboratory at the incubedIT also provides a motion capturing system (Mocap) to track the motion of rigid bodies inside a certain area. The Mocap system is from the company Optitrack [4] and consists of eight Flex13 cameras, which are connected via 2 Optihubs to the Mocap-PC. The cameras emit infrared light which gets reflected from the small ball-like markers which are attached to the shuttles. Each shuttle got equipped with 5 such markers to make the tracking more stable. The Mocap-PC has an intel i7-9700 processor, 16Gb Ram and an Nvidia Quadro P620 to be sure that it has enough processing power for the pose tracking because the cameras deliver pictures at a rate of 120Hz.

6.2 Software

incubedIT offers a full software framework for mobile shuttle robot tasks like logistics transports. The implementation of this thesis should make use of the framework as far as possible and keep the changes at a minimum. Therefore, the most driver components of the shuttles were already given and all additional elements of the concept were implemented in a compatible way.

As it can be seen in Figure 6.3, the implementation of the concept is done on two separate PCs, Mocap and host PC of the shuttle, which are connected via Ethernet. To structure the explanation, the software of each PC is explained in the subsequent sections.

[4] https://optitrack.com/, accessed 06.04.2021

6.2.1 Mocap-PC

The Mocap-PC has the task to extract the pose information of the robot from the sequence of camera images and convert the pose information into a ROS message. The pose information extraction is made by the Motive software which only runs on a Windows operating system. This is a problem because ROS does not run on Windows operating systems.

It is possible to use a separate Linux PC connected to the network to convert the data from the Motive software to ROS messages. This was tested and showed poor timing behaviors because of the big data traffic on the network. To get the best timing behavior it is more convenient to install a virtual machine on the same PC with a Linux OS. This virtual machine runs the Mocap ROS node which converts the Motive data to a ROS message and adds the current wall time as a timestamp to it. This message is then transferred via Ethernet to the shuttle PC. Network problems do not affect the timing behavior of the data anymore because the timestamp has already been attached.

Operating System	Windows 10	
Tracking Software	Optitrack Motive 1.10.0	
Virtual Computer	Oracle VirtualBox 6.1	
OS on virtual computer	Ubuntu 18.04.3 LTS	
ROS Version	Melodic	http://wiki.ros.org/melodic
Time synchronization	chrony	https://chrony.tuxfamily.org/
Mocap ROS Node	mocap_optitrack	https://github.com/ros-drivers/mocap_optitrack

Table 6.4: Used software packages on the Mocap-PC.

The timestamp in the Mocap node is the wall time of the OS. To get useful timestamps this wall time needs to be synchronized at all PCs used in a ROS network. This is done with the chrony package on a Ubuntu system. This package uses the NTP protocol and synchronizes the wall time of the PC to a parameterized time server. At incubedIT, all host PCs of the shuttles are connected to one server which runs the FMS and a timeserver. All shuttles synchronize their time to this timeserver and therefore also the chrony at the Mocap-PC got configured to synchronize to this timeserver.

6.2.2 Host PC

The host PC of the shuttle is placed inside the shuttle and is the central processing unit. At this host PC, all sensor data and commands from the FMS come together and are processed to calculate and send out new commands to the actuators of the shuttle. Because all sensor data is available on this PC, it makes sense to implement the concept on it.

The sensor data arrive at the host PC through different communication interfaces using the respective driver node. These nodes are already given from the company where they use the software packages listed in Table 6.5. The drivers for the wheel odometry of the **Marie** and **Valentina** shuttle were implemented by incubedIT.

Operating System	Ubuntu 18.04.3 LTS	
ROS Version	Melodic	http://wiki.ros.org/melodic
Time synchronization	chrony	https://chrony.tuxfamily.org/
Sick S300 driver	Care-O-bot (cob)	http://wiki.ros.org/cob_sick_s300
Sick Microscan driver	Sick Saftey Scanner	http://wiki.ros.org/sick_safetyscanners
Laser to velocity conversion	Range Flow 2D Odometry	http://wiki.ros.org/rf2o
Beckhoff PLC communication	libAds	https://github.com/gass/libads

Table 6.5: Used software packages of the shuttle.

```
1  std_msgs/Header header
2  string child_frame_id
3  geometry_msgs/PoseWithCovariance pose
4  geometry_msgs/TwistWithCovariance twist
```

Listing 6.1: Definition of the nav_msgs/Odometry Message. [5]

After the driver nodes, the sensor data continuous as ROS messages to the constant timestamper node in Figure 6.3. This constant timestamper would be needed to be a separate post-processing node if the drive node's source code should not be changed. In our implementation, we had access to the source code of each driver node and integrated this constant timestamper there to avoid the implementation of a new node for each type of sensor data.

The sensor data with the corrected timestamps then are transfered to the data conversion step which outputs the robot velocity observed from the sensor data. This robot velocity should be represented with the same ROS message type for all sensors to make the comparison easier. The wheel odometry driver node and the RF2O laser conversion node output a nav_msgs/Odometry with the definition shown in Listing 6.1. This includes the timestamp in the header and the robot translation and rotational velocity in the twist data. Therefore, the nav_msgs/Odometry message was selected as the common velocity data representation and only one node needed to be implemented by our self which converts the geometry_msgs/PoseStamped from the Mocap node to the desired nav_msgs/Odometry using the Equations 5.9 and 5.10.

Please note that in the original implementation of the RF2O node, the sensor data is buffered and the odometry data is only calculated and published at a certain period which can be selected with a parameter. We changed this behavior in our implementation such that the odometry data is calculated for each new laser scan data and copies the timestamp from the laser scan data to the output nav_msgs/Odometry message.

Because all sensor data is represented in the same message type the correlation node

[5] http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html, accessed 17.05.2021

needs to be implemented only once and can be used for each pair of sensors. The correlation node is implemented in Python which makes it possible to use the SciPy package [6]. This package is designed and optimized for signal processing and provides a function for the interpolation step and the cross-correlation. The node continuously segments the input data, uses the SciPy functions for the interpolation and correlation and outputs the estimated delay in the log file.

[6] https://www.scipy.org/, accessed 07.04.2021

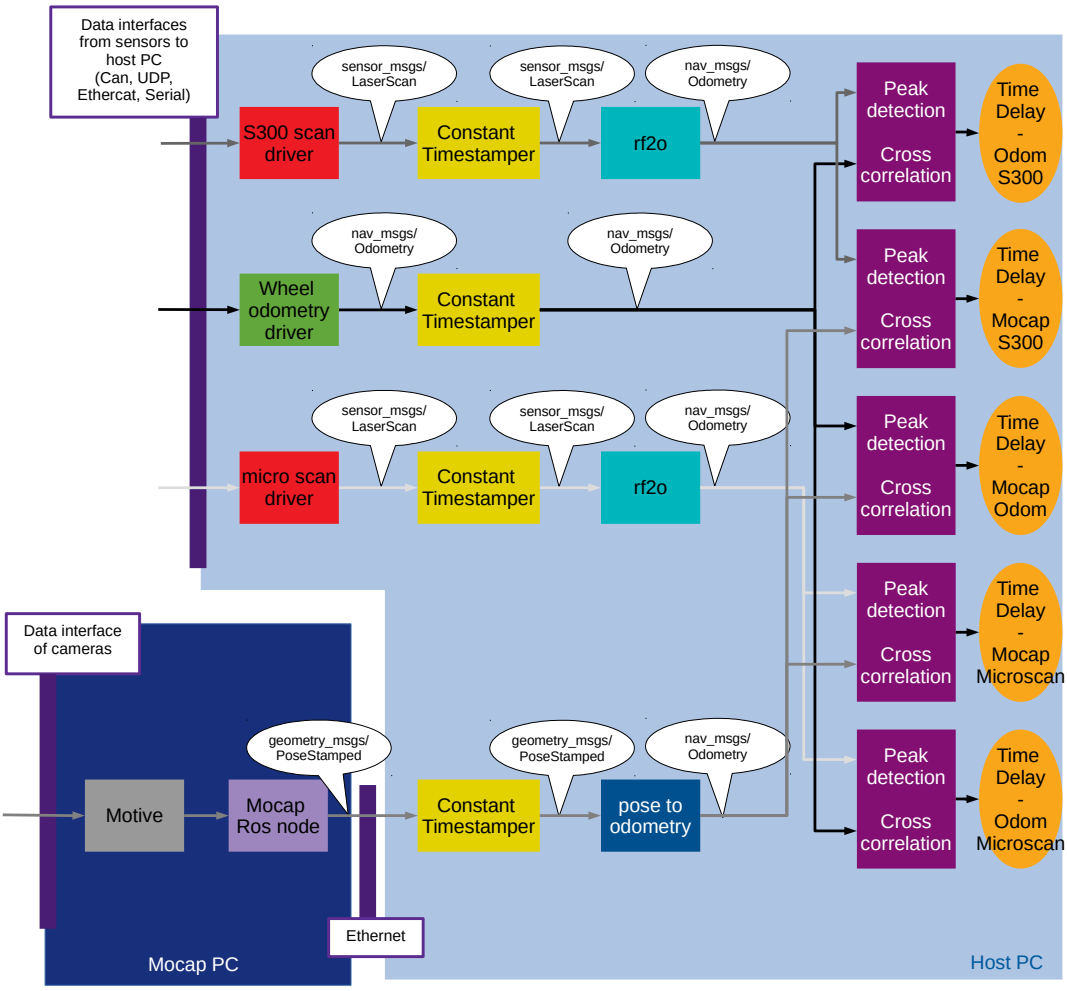


Figure 6.3: Overview of the software components with the used ROS messages.

7 Evaluation

This chapter starts with the presentation of the test environment. It continues with the evaluations of the sensor data timestamps and the results from the estimation of the delay between the different sensor data streams. The chapter concludes with the results from tests of how the delay value affects the localization performance of a shuttle.

7.1 Environment

The test of our concept should be executed in an area that is similar to the shuttles application area. This is given in the incubedIT laboratory because they use the same floor and similar obstacles as in typical application areas. The used test area can be seen in Figure 7.1.

The laboratory also includes a Mocap system which is needed for testing the effect of the timestamp correction on the localization quality of the shuttle. In a real application, the shuttle drives back and forth between different stations. Therefore, we set up two stations in the area of the Mocap system and let the shuttle drive continuously between points A and B which are shown in Figure 7.1. Boxes were placed on the direct path between the two points such that the shuttle has to drive around them. This increases the driving distance and includes rotation in the movement to enable the rotation velocity stream comparison in the delay estimation step.

To be able to develop the correlation node and repeat the same estimations several times, the sensor messages of each shuttle were recorded in a bagfile for a period of about 2 minutes. During this period the shuttle drove 4 to 6 times back and forth between the stations which includes enough acceleration and deceleration phases for the delay estimation. The delay estimation and the localization evaluation were made on the local laptop running the same ROS nodes as a real shuttle and used the sensor data from the bagfile as input.

7.2 Sensor Timestamping

The delay estimation relies on the timestamps of the sensor data. Thus, the timestamp quality itself was evaluated for each sensor used in this thesis before the delay estimations were made. These evaluations are presented in the subsequent subsections for each sensor.

Figure 7.1: Testing area with the path of the shuttle and the obstacles.

7.2.1 Sick S300

As described in Section 6.1.1, all shuttles under test use the Sick S300 laser scanner connected via a serial interface which results in a nominal sampling period of 120ms. This means that in the ideal case, where no timing inaccuracies are introduced, the period between the timestamps of two subsequent laser scan messages should be constant at 120ms.

Unfortunately, this is not the case as we can see with the blue dots in Figure 7.2. It shows the original timestamp periods from the laser scan messages which are influenced by timing jitter and which are centered at around 118.2ms instead of the expected 120ms.

The jitter of about +-2ms is mainly caused by the generic operation system of the host PC and its scheduling behavior because the sensor sends out its data as soon as it has finished its measurement to the exclusively used serial interface. To counteract this jitter the constant timestamper Algorithm 1 with a window size of 25 was implemented. The result is shown with the orange dots in the same Figure. This corrected timestamp period is close to constant for this short test period of 250 samples and is therefore well prepared for the further evaluations.

To find explanations why the average period is, with a value of 118.2ms, lower than the expected 120ms we recorded the laser scan messages from different shuttles for 30 minutes and plotted them in Figure 7.3. As it can be observed, all of them show similar behavior. Their moving average corrected periods drift between 117.95 and

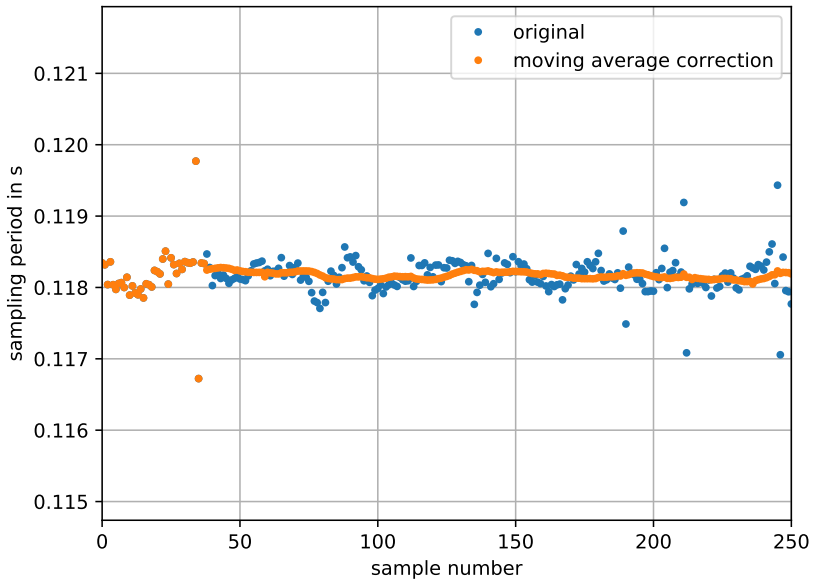
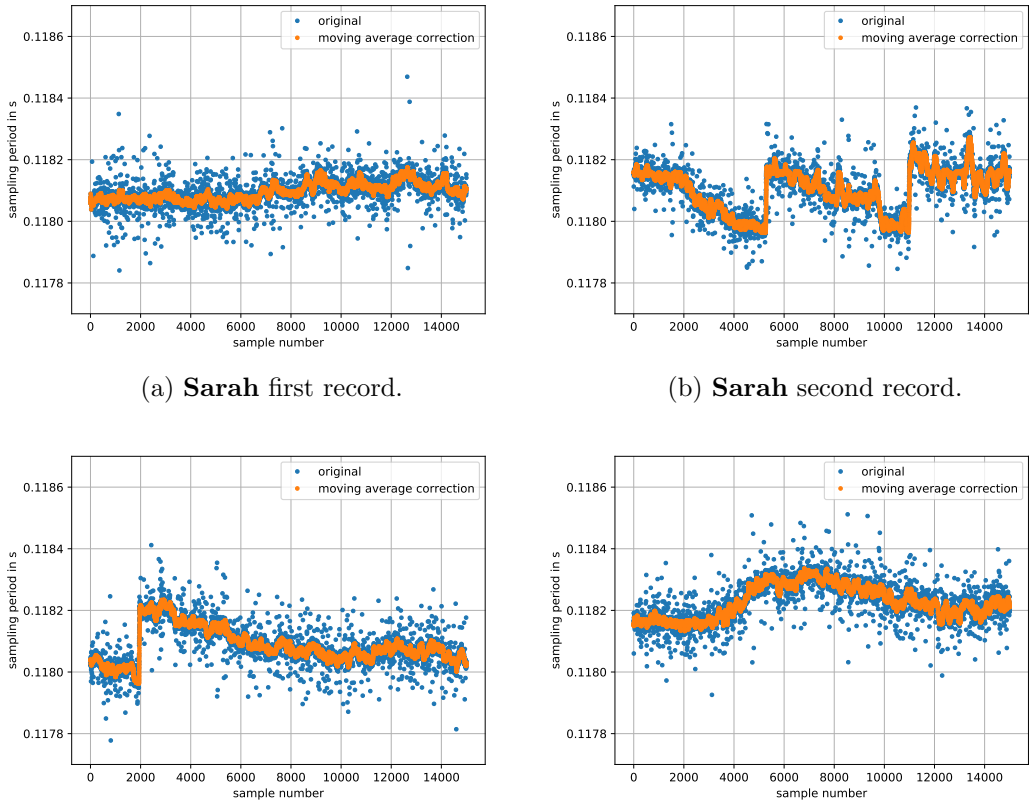


Figure 7.2: Sick S300 timestamping behavior.

118.3 milliseconds. The measurement periods of two sensors were also measured with an oscilloscope to eliminated additional error sources. But the results show similar behavior with periods between 117.6 and 118.5 milliseconds.

The response of the vendor company Sick, after we presented them the results, confirmed that the measurements were right and that this behavior is within the limits of the specification of the sensor. The average period is lower than 120ms because the sensor needs to ensure that he finished a scan every 40ms. The motor which rotates the mirror inside the laser scanner has a rotation velocity control divergence of a maximum of +-2 percent. Therefore, Sick sets the control target of this motor to 39.2ms per rotation such that even if the motor controller is on the limit of -2 percent rotation speed a scan every 40ms can be ensured. And because the serial interface still can only transmit each third laser scan the target sampling period is $39.2 \cdot 3 = 117.6$ms which corresponds with our measurements.

To verify the observations for the Sick S300 laser scanner, the velocity stream from the RF2O package once with the original and once with the corrected timestamps is shown in Figure 7.4a. The velocity stream with the original timestamps is already smooth because the jitter of the timestamps compared to the sampling period is small. Because of the exclusive use of the serial interface, no lost messages and buffering influences the timestamp. To see the difference between the original timestamping and the corrected one it needs to be zoomed in which is shown in Figure 7.4b.



(a) **Sarah** first record.

(b) **Sarah** second record.

(c) **Valentina** record.

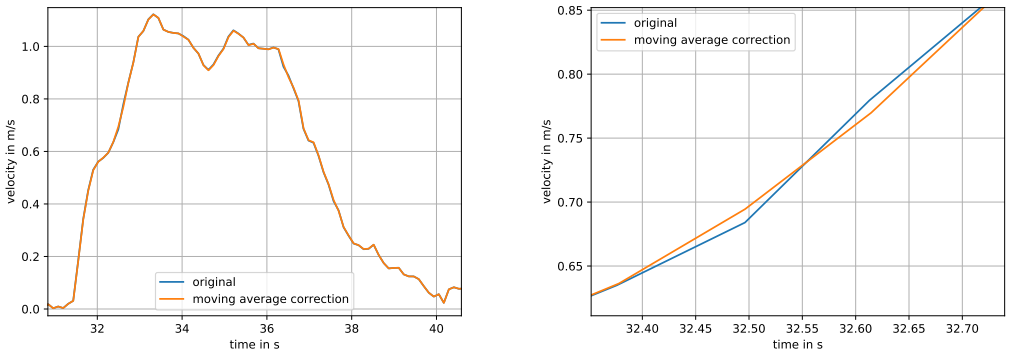
(d) **Marie** record.

Figure 7.3: Sick s300 30 minutes timestamp drift behavior.

7.2.2 Sick Microscan3

The Sick Microscan3 laser scanner uses similar hardware for the distance measurements as the S300 and needs also 39.2ms for one scan. The big difference between the two scanners is the data interface. The Microscan3 uses the UDP protocol via an Ethernet connection.

This interface offers two new features. The first one is that the laser scanner adds a timestamp from its internal clock to the sensor data. This is beneficial in theory because now all time inaccuracies introduced by the communication process and the scheduling of the host PC are eliminated. As experienced from the S300, we would expect that the sampling period of the Microscan3 is rather constant at a value of 39.2ms. Unfortunately, this is not case as it can be seen in Figure 7.5. The timestamp periods from the laser scanner, orange dots in the Figure, show four discrete values of 36, 38, 40 and 42ms. This leads to the interpretation that the data timestamping and sending part in the sensor is triggered every 2ms and is not synchronized with the measurement trigger. This would explain the behavior shown in the figure because most times the period is 38 or 40ms



(a) Original vs. corrected timestamp.

(b) Zoomed into detail.

Figure 7.4: Effect of the Sick S300 timestamp correction on the velocity stream.

which are the closest possible periods to the expected period of 39.2ms. And because the measurement and the sending parts are not synchronized it sometimes happens that the new measurement data is already available after 36ms but in the next cycle the sending part needs to wait longer which is represented with a 42ms period. We again requested an explaination from SICK and they confirmed that the timestamp is added by the application software in the sensor and that this software is not synchronized with the measurement trigger.

The second new feature of the Microscan3 is that it can synchronize its internal clock to a selectable timeserver via the Ethernet interface and the Simple Network Time Protocol (SNTP). This is very important if the timestamps from the laser scanner should be used because without synchronization the clock of the host PC and the clock of the laser scanner would drift apart. This can be seen in Figure 7.6a where the timestamp from the host PC is subtracted from the timestamp of the laser scanner. After 4000 samples, which are around 160 seconds, the clocks drifted 7.5ms apart.

In contrast to that, Figure 7.6b shows that the synchronization works and the clocks do not drift apart. But this Figure uncovers another problem. The synchronization introduces jumps in the timestamps because only the SNTP is used. The SNTP protocol corrects time differences at once at some occasions. The NTP protocol instead would correct an offset by changing the clock rate which would result in a smooth transition. But the sensor does not support the NTP protocol.

The new features with the jumping and the discrete values of the timestamp periods provide more disadvantages than advantages which leads to the conclusion to use again the timestamp of the host PC and the constant timestamper algorithm.

Figure 7.7 shows the timestamp periods behavior of the Microscan3 when the host PC timestamps are used. These original periods, blue dots, also show discrete values which leads to the conclusion that the sensor data is sent out by the scanner to Ethernet at these different periods. But the laser scanner itself definitely does not change its scanning

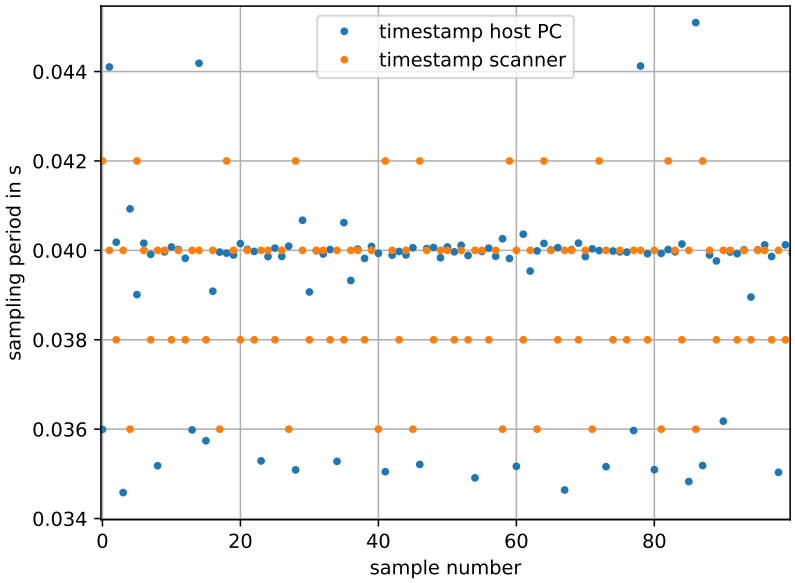
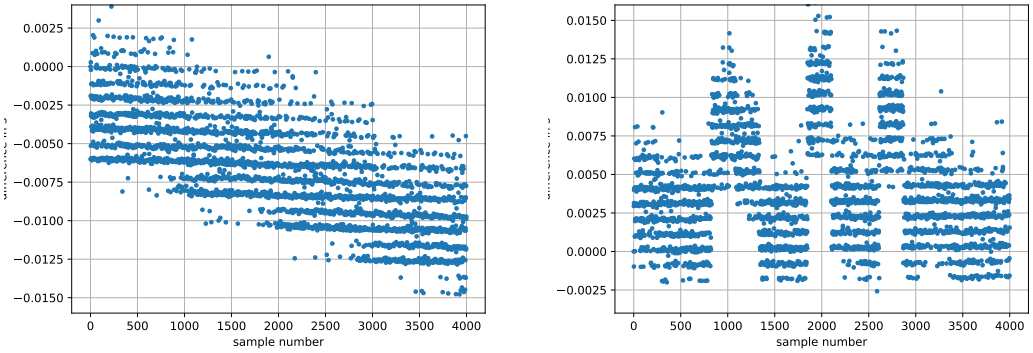


Figure 7.5: Host PC and Sick Microscan3 timestamp.



(a) No synchronization.

(b) Synchronization active.

Figure 7.6: Difference of the host PC and the laser scanner timestamp.

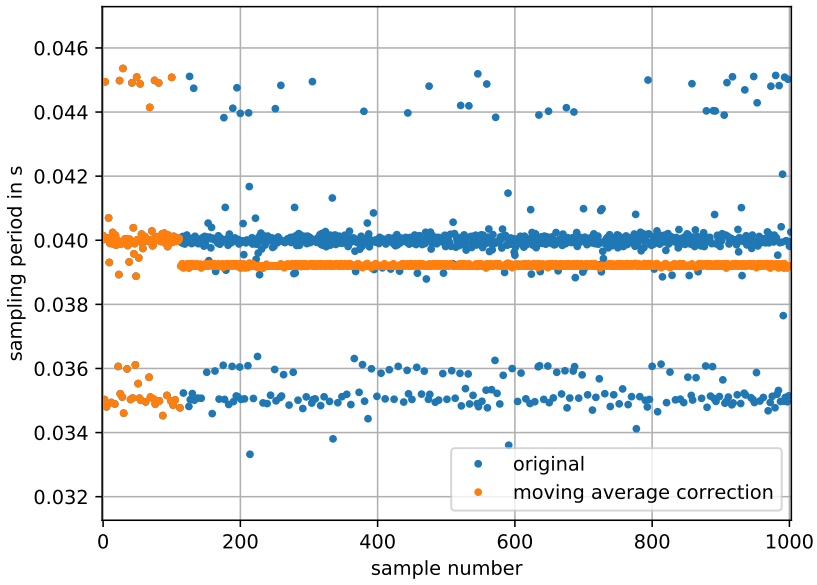


Figure 7.7: Sampling period of the Sick Microscan3 with original and corrected timestampes.

period that quickly and that much. Thus the sampling period is not trustworthy and needs to be corrected to match better with the underlying physical process.

The result of this timestamp correction with a window size of 75 is shown with the orange dots. After the settling time, the timestamp periods are close to the constant value of 39.2ms which we expected. A confirmation that this timestamp correction makes sense can be seen in Figure 7.8 because it affects the velocity stream drastically. With the original timestamps, the velocity stream has jumps at each data point where the timestamp period was lower or higher than the expected value. These jumps are not feasible for the robot because its acceleration and deceleration capabilities are limited. The velocity stream with the corrected timestamps, the orange line in the Figure, is much smoother and correlates much better with the truth velocity profile of the shuttles.

7.2.3 Wheel odometry

Now that we finished the exploration on the used laser scanner sensors, we continue with the next sensor type used by the shuttles, the wheel odometry. As mentioned in 6.1.2, the three shuttles we used to test the concept use different wheel odometry sensors or communication paths until the data receives its timestamp. Therefore, we take a look at the timestamping behavior of the wheel odometry on each shuttle separately and start with the **Sarah** shuttle.

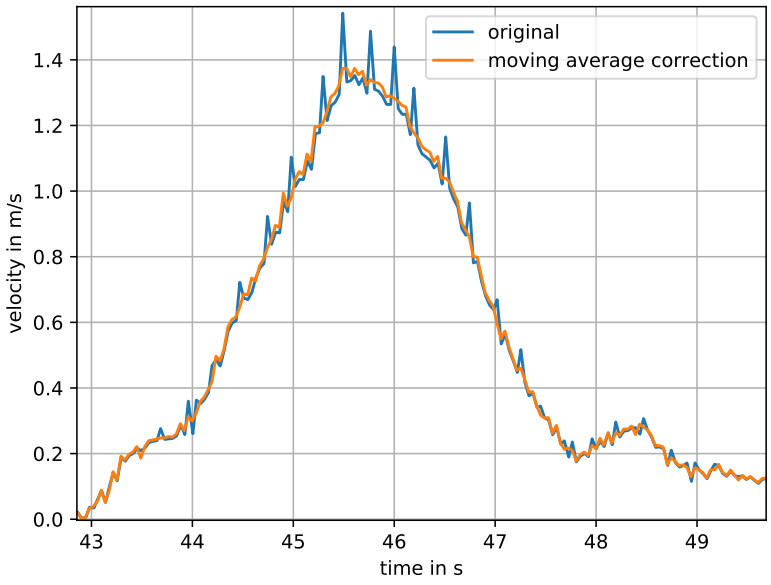


Figure 7.8: Effect of the Sick Microscan3 timestamp correction on the velocity stream.

At the **Sarah** shuttle the encoders are connected to the PLC and the PLC communicates via EtherCAT with the host PC. The implementation of the PLC is provided by the Partners of incubedIT and no insights into it were given. The only observation we were able to make at the **Sarah** shuttle was that the interface driver sends a request to the PLC and receives the wheel velocity values as a response to it. These are two communication paths that introduce time inaccuracies. It is a big factor of the cause for the high timestamp period jitter of about +-18ms which can be seen in the Figure 7.9. To correct this jitter, the constant timestamper was used. This time with a window size of 60 do use samples from the last 3 seconds at an expected sampling period of 0.05s. The timestamp correction with this window size outputs a constant sampling period as it can be seen with the orange dots in Figure 7.9.

The result on the velocity streams shown in Figure 7.10 were surprising. The timestamp correction made the velocity stream worse and introduced discontinuities. This proves the assumption wrong that the PLC collects the wheel velocity values at a fixed sampling period. It is more likely that the PLC grabs the encoder values and calculates the wheel velocities when it received the request from the host PC. Therefore, the jitter in the timestamping period reflects the true measurement period better which results in a much smoother velocity stream. Because of this finding, the constant timestamper was removed from the wheel odometry processing of the **Sarah** shuttle.

The **Valentina** shuttle communicates with its PLC via Ethernet and the UDP protocol. The messages sent from the PLC include a timestamp from the local PLC clock

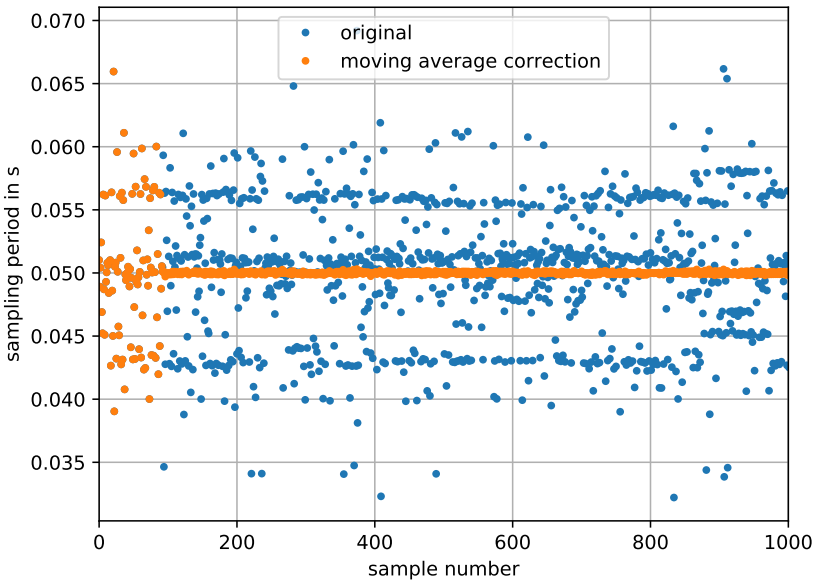


Figure 7.9: Sampling period of the **Sarah** wheel odometry with original and corrected timestampes.

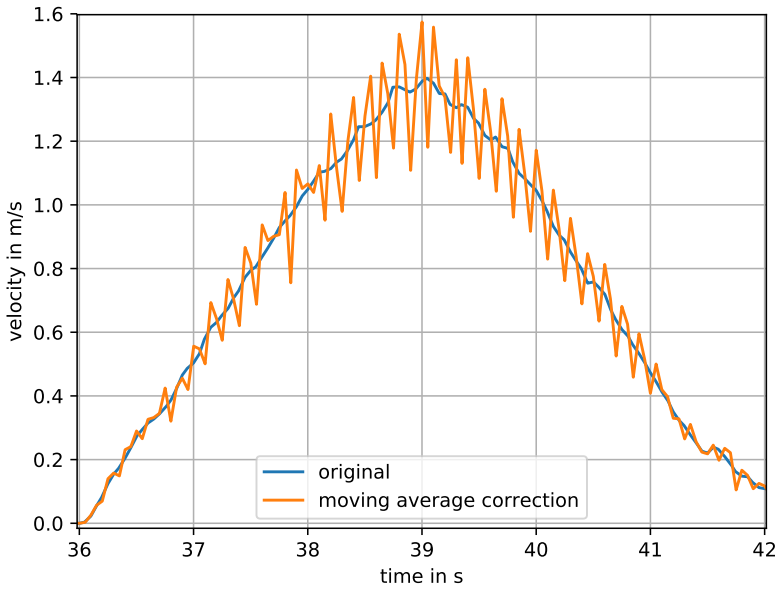


Figure 7.10: Effect of the **Sarah** wheel odometry timestamp correction on the velocity stream.

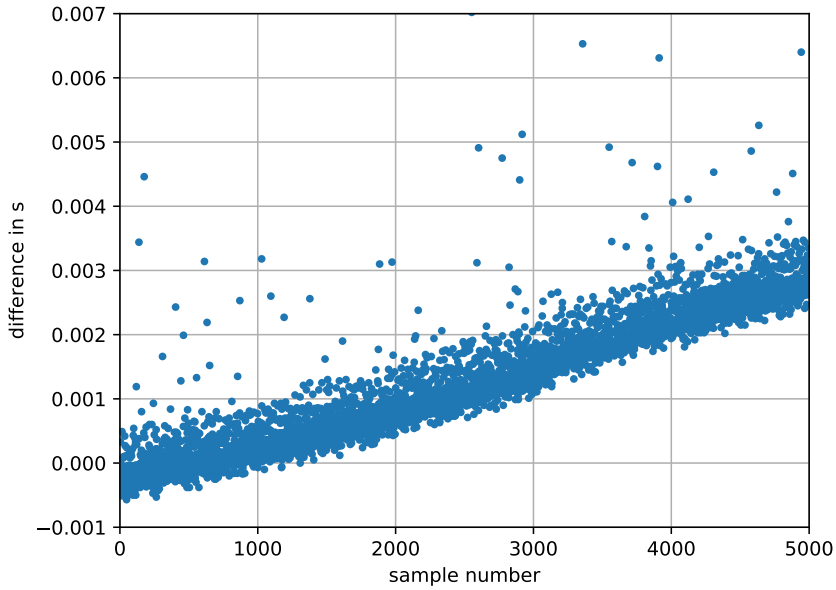


Figure 7.11: Difference of the host PC and the PLC timestamps of the **Valentina** shuttle

when the measurement was taken. To see if this timestamp is beneficial for our application, we subtract the host PC timestamp and the PLC timestamp from each other and plotted the course in Figure 7.11 for about 4 minutes (5000 samples at 20Hz). As it can be observed, the clocks drift apart by about 3ms after 4 minutes. This would not be a problem for our velocity stream calculation because there, only the timestamp period and not the absolute timestamp value is important. These timestamp period values differ only marginal which can be neglected for the velocity stream. But the wheel odometry data in the robot is used for many different tasks and this drift would make a problem at some of these tasks. Additionally, the PLC does not offer any mechanism to synchronize the clocks. Therefore, the timestamps from the host PC are used.

During the start-up process of the **Valentina** shuttle, the host PC tells the PLC at which period he wants to receive the wheel odometry values. This parameter is set to 20Hz or 50ms. This leads to the assumption that the PLC takes the encoder value exactly at every 50ms and sends them in a UDP package to the host PC without any request procedure like it was the case at the **Sarah** shuttle. Nevertheless, the plot of the timestamp periods in Figure 7.12 shows a different behavior. The host PC timestamp periods are placed at the two values 0.052 and 0.056ms with some additional jitter around it. This lets one conclude that the PLC seems to have a period cycle of 4ms because 52ms and 56ms are dividable by 4ms with an integer number and 50ms is not. So the PLC provides the wheel odometry data every 52ms instead of the desired 50ms and every now and then another cycle is needed to send the wheel odometry which are the samples

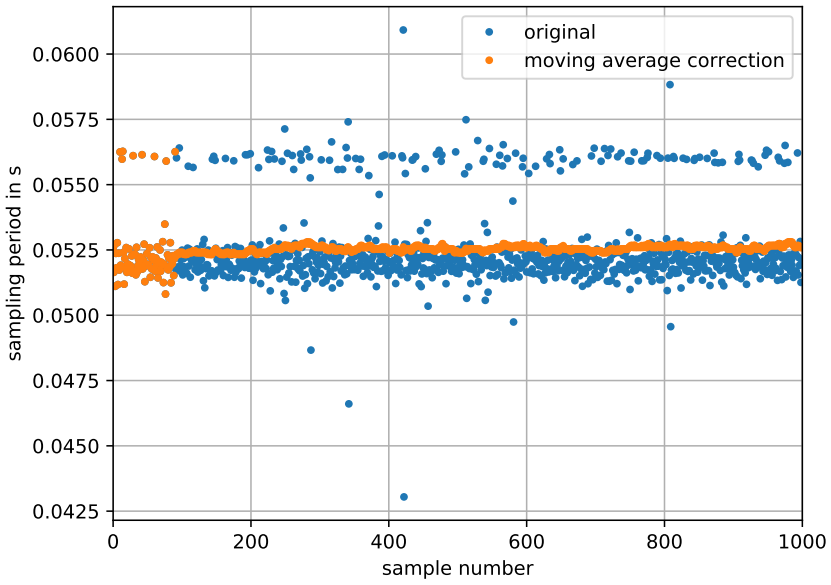


Figure 7.12: Sampling period of the **Valentina** wheel odometry with original and corrected timestampes.

at 56ms period. We again were not able to investigate the PLC implementation to check if our conclusion is correct but we contacted the partner company who was responsible for the PLC implementation. It confirmed our conclusion of the 4ms PLC cycle time for the UDP interface processing and added that there is a priority list of UDP packages to send to the host PC. If a package with higher priority is ready to send at the same time when the wheel odometry package should be sent, the wheel odometry package gets postponed until the next PLC UDP interface cycle. This explains why some timestamp periods are at 56ms. The jitter around these two period values can be explained with the Ethernet communication and the processing behavior of the host PC.

To counteract this jitter and because we concluded that the PLC internally uses a faster period for the encoder value measurement, we applied the constant timestamper with a window size of 60 again. But Figure 7.13 proves that this conclusion is wrong. The PLC only takes the wheel encoder values when its UDP cycle is ready for the wheel odometry package. This means that the sampling period changes between 52ms and 56ms which makes the constant timestamper useless and leads to jumps in the velocity stream. Luckily, the jitter introduced by the Ethernet is low compared to the sampling period which keeps the error in the velocity stream at a neglectable value and results in a smooth velocity stream.

At the **Marie** shuttle, the encoder is directly connected to the host PC via a CAN bus. Unlike the other two-wheel odometry, the timestamp periods of the **Marie** shuttle

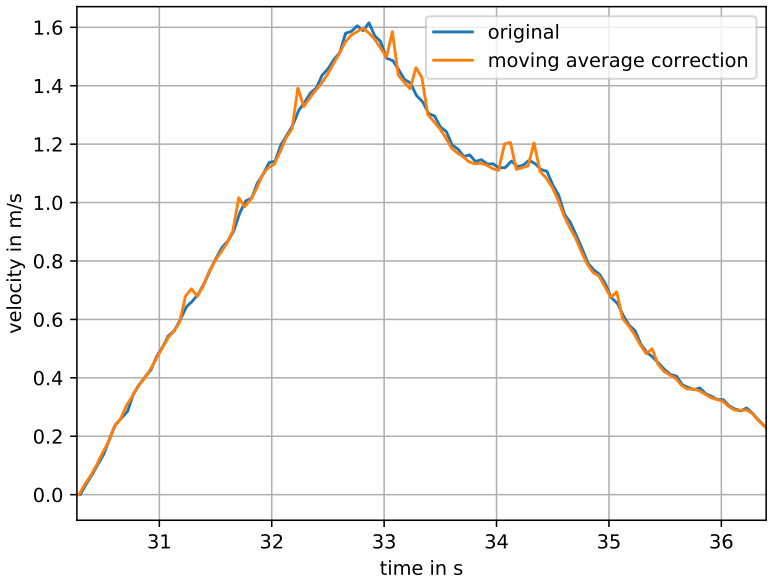


Figure 7.13: Effect of the **Valentina** wheel odometry timestamp correction on the velocity stream.

are constant at exactly 50ms and only some jitter from the communication process is superimposed. Additionally, the jitter keeps small compared to the expected timestamp period which can be seen in Figure 7.14. The constant sampling period and the small jitter are ideal to use the constant timestamp correction which is shown with the orange dots in the same Figure.

Because the jitter is small compared to the expected timestamp period the effect of it on the velocity stream is also small. The velocity plot in Figure 7.15 needed to be zoomed in to see a difference between the velocity stream calculated with the original timestamps and the velocity stream calculated with the corrected timestamps. Only at a few areas, where the original timestamp period is of from the average period, the streams separat from each other. It is not clear if the timestamp correction is beneficial in this case because no ground truth of the velocity stream is given and both are very similar. Because they are very similar we decided to continue without the timestamp correction for the wheel odometry of the **Marie** shuttle to keep it simpler.

7.2.4 Mocap

The last sensor for which we evaluated the timestamps was the Mocap system. As it was mention in Section 6.2.1 that the conversion node from the Motive data to a ROS message can run on any PC in the same network as the Mocap-PC. In the standard setup of incubedIT this node was either started at the shuttles host PC or on the laptop

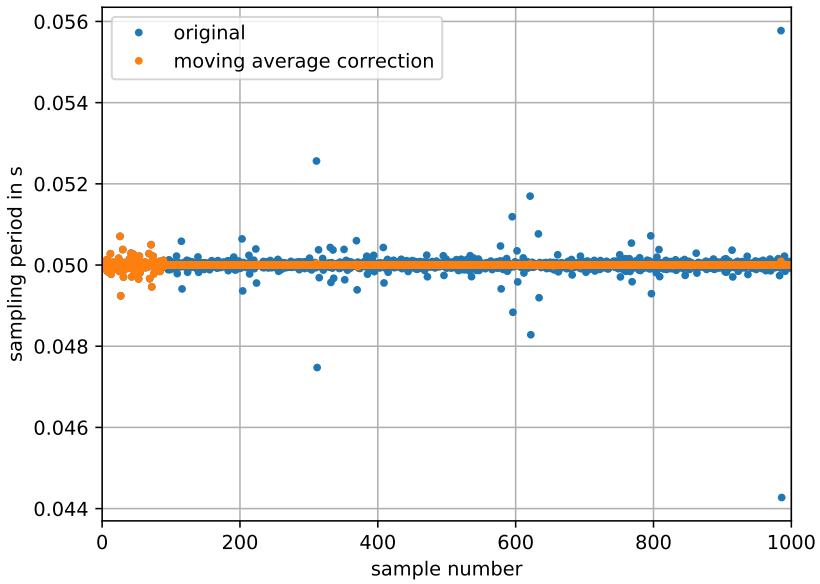


Figure 7.14: Sampling period of the **Marie** wheel odometry with original and corrected timestampes.

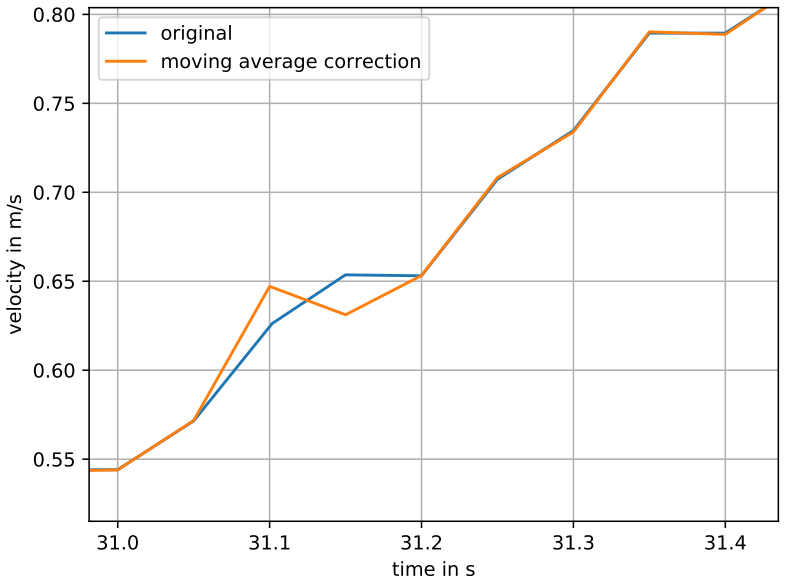


Figure 7.15: Effect of the **Marie** wheel odometry timestamp correction on the velocity stream.

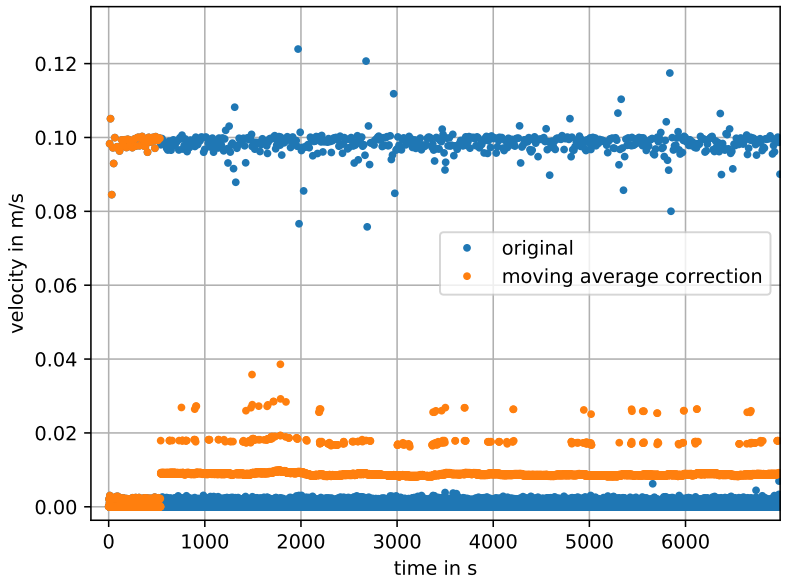


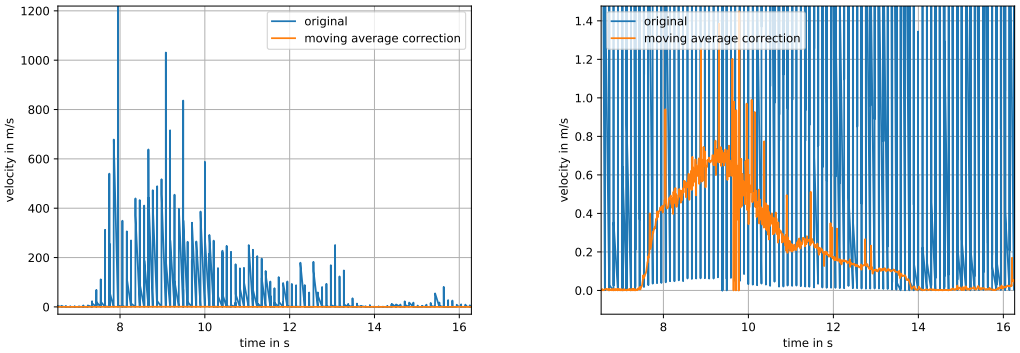
Figure 7.16: Timestamp periods of the Mocap node running on a Laptop with the original and corrected timestampes.

in the same network. This setup showed a very bad timing behavior as can be seen in Figure 7.16. The timestamping period changes between close to zero and 10ms at an expected sampling period of 8.333ms (120Hz) which indicates that somewhere in the network buffering happened. A sampling period of nearly zero leads to very high velocity values which are shown in Figure 7.17a. The maximum velocity of the shuttles is 2 meters per second so the values in the Figure are not feasiable.

The constant timestamper was used to try to get useful velocity streams again. But these errors were too big to correct them. The timestamp periods after the constant timestamper are not a constant value and still jump between certain values. At least, the constant timestamper corrects the zero timestamping periods which is definitely an improvement in the velocity stream. It can be seen in Figure 7.17b but it still includes too many jumps and is therefore not useful.

Placing the Mocap node on the same PC as the Motive software improved the timing behavior dramatically. The data does not need to travel through the network anymore which removed the buffering behavior as it can be observed in Figure 7.18 where no zero timestamp periods are present anymore. Nevertheless, the jitter introduced from the use of no real-time operating system with +-2ms is still high compared to the expected period of 8.333ms. But the constant timestamper correction can handle this and outputs a constant timestamp period again which is shown in Figure 7.18 with the orange dots.

The effect on the velocity stream of this timestamp correction can be observed in



(a) Velocity stream of the original timestamps with its unfeasible high values.

(b) Scaled Y axis to see the velocity stream of the corrected timestamps.

Figure 7.17: Velocity stream of the Mocap node running on a Laptop.

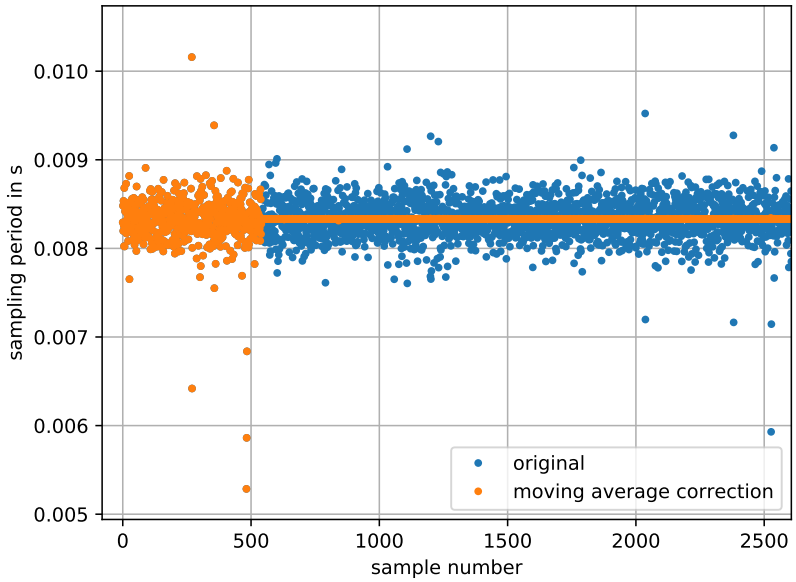


Figure 7.18: Sampling period of the Mocap node running on the Mocap-PC with the original and corrected timestampes.

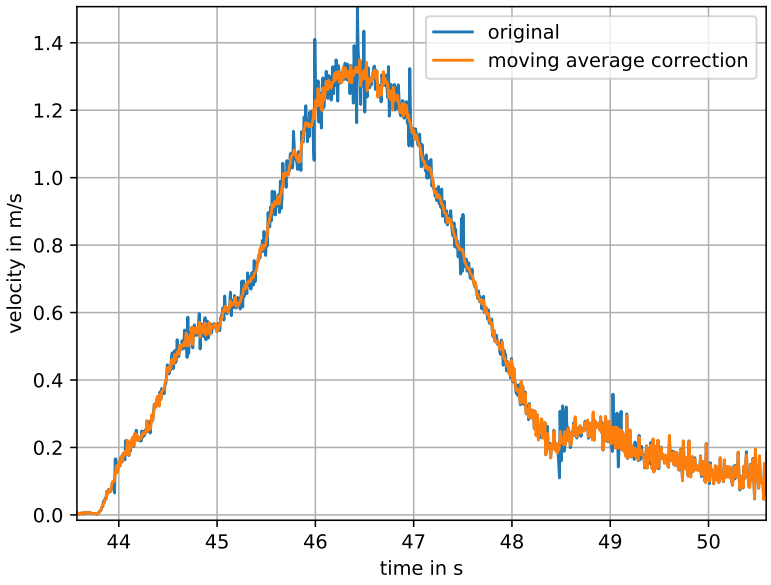


Figure 7.19: Velocity stream of the Mocap node running on the Mocap-PC.

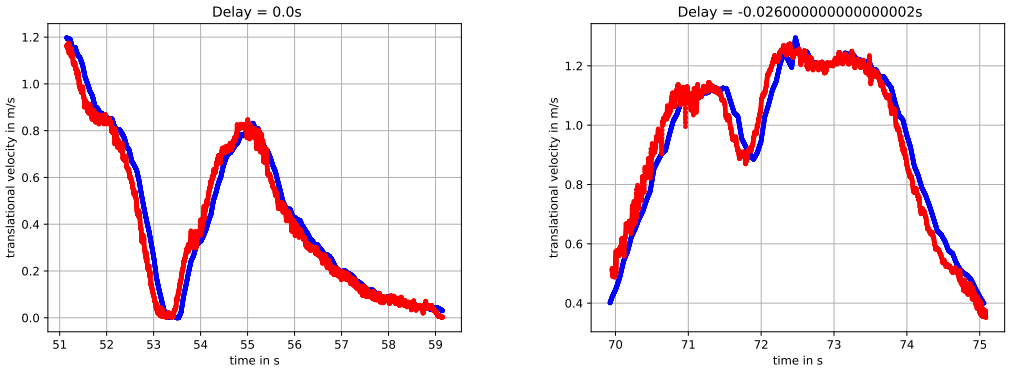
Figure 7.19. The velocity stream calculated with the original timestamp has much higher jumps in it. The timestamp corrected velocity stream is much smoother and therefore closer to the real movement of the robot.

Compared to the velocity streams of the other sensors it is still not that smooth. The period is already constant after the timestamp correction so the discontinuity can not be caused by timing inaccuracies. The other values which are needed for the velocity calculation are the position values. Actually, these position values from the Mocap system jump such that it results in the jumps in the velocity streams. But they can be interpreted as white noise which should not make any problem for the correlation function (see Section 3.3) used in the upcoming section.

7.3 Estimation of the Delay

At this point, the timestamps of the sensor data are corrected sufficiently and velocity streams of each sensor are available. Now it is possible to estimate the delay between two sensors by using the concept introduced in Section 5.4.

No experience was available of how the cross correlation function behaves in different situations. This made it necessary to start with some trial and error iterations until a useful setup was found. Some lessons we have learned from it are presented in the next subsection. The subsequent subsections present the estimation results from the different shuttles and a comparison to a different estimation method the so called minimum cells test.



(a) Bad peak extraction.

(b) Start and end at high values.

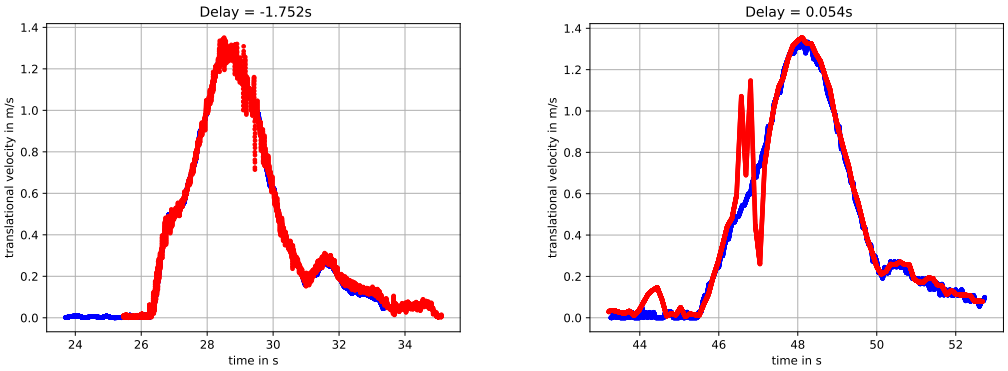
Figure 7.20: Bad correlation results.

7.3.1 Behavior and Errors of the Correlation Estimation

One of the first findings in relation to the behavior of the correlation function was that it works best if the signal shape starts and ends with values close to zero and the highest values occur around the middle of the signal. It seems like that the low values at the edges should be at least as long as the expected time delay which would allow the correlation to shift the signal in time. Two negative examples are shown in Figure 7.20 where the estimated delay was applied to the signals but they still do not lie on top of each other. Hence, the delay was estimated incorrectly. In both situations, the edges of the signals are not close to zero. If the signal is now shifted, the overlapping part in time of the signals gets smaller which reduces the correlation value. If the correlation value of the signal at the true correspondence is not high enough to counteract this overlapping reduction, the correlation function will not have the maximum value at the true correspondence. This behavior is caused by the finite number of samples from the signal, is known in the field of signal processing and can be corrected with window functions.

If the signals have close to zero values at the edges, the reduction of the correlation value given by the shifting is minimal and does not change the location of the maximum value of the correlation function. A good example that this also works with big delay values (e.g. 2 seconds) compared to the duration of the signals (e.g. 10 seconds) is shown in Figure 7.21a. The correlation works well in this situation because on either side of the main peak the values are small for a duration that is longer than the real delay.

Another finding confirmed the theory from Section 3.3 that the correlation function is robust against white noise. The noise from the Mocap velocity streams did not affect the correlation value. Even more aggressive errors from a wrong data conversion in the RF2O package could not lead the correlation to a wrong delay estimation. This example is shown in Figure 7.21b.



(a) Big delay.

(b) Velocity calculation error.

Figure 7.21: Good correlation results.

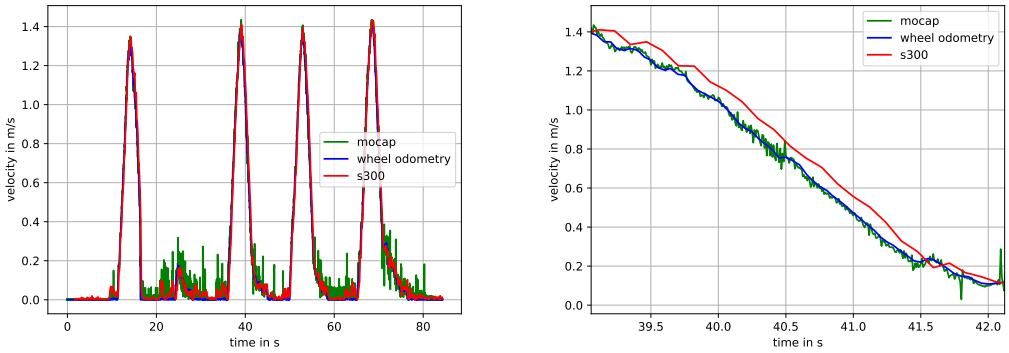
Concluding from these findings, the leftover noise in the velocity streams is neglectable. More important is to extract good peaks from the velocity streams with a high peak in the center and low values at the edges.

7.3.2 Sarah

The first shuttle, the delay estimation was tested on, was **Sarah**. The sensor messages were recorded and the timestamp correction and data conversions were applied. The results are translational and rotational velocity streams which are shown in Figure 7.22a and 7.23a. The four peaks in the first Figure show that the shuttle drove four times from one station to another. Between the transfers, the shuttle made in-place rotations at the stations to stop in the desired direction of the station and a second one to start in the forward direction for the next transfer. These in-place rotations are shown in the later figure in areas where the translation velocity is close to zero.

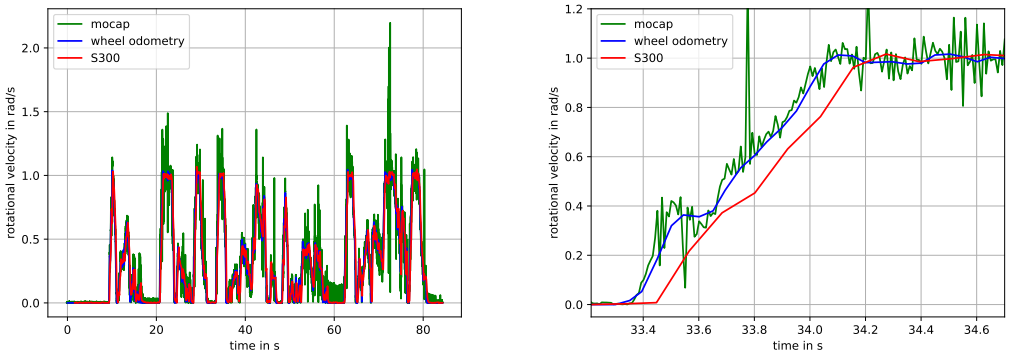
The peaks in the rotation velocity are not that clear as at the translational velocity. The maxima of the peaks are not that much higher compared to the low values besides the peaks as it is the case at the translational velocity. This makes them harder to extract and does not give the needed low values at the edges for a good correlation. Therefore, the delay estimations were only continued with the translational velocities. To also get useful rotational velocities, the driven path of the shuttle needs to be changed. Instead of driving between stations the shuttle should only rotate in-place stop for a while and rotate in-place again.

Figure 7.22b shows the velocity streams from the different sensors for one deceleration period. It can be observed that the Mocap and wheel odometry velocities more or less lie on top of each other. The expected delay value between them is therefore close to zero. In contrast to that is the S300 laser scanner. It seems to be delayed for about 150ms.



(a) Full test period. (b) Zoomed.

Figure 7.22: Translational velocity stream of the different sensors on the **Sarah** shuttle.



(a) Full test period. (b) Zoomed.

Figure 7.23: Rotational velocity stream of the different sensors on the **Sarah** shuttle.

The values in Table 7.1 represent the delay between the first sensor and the second sensor named in the first column. This means if the value is positive the first sensor is earlier in time than the second one. If the value is negative the second sensor is earlier in time than the first one.

The delay between the sensors was estimated at each peak in the velocity stream and the mean value got calculated. These mean delay values are also represented in Figure 7.24 to make them more intuitively.

The results confirm the expectations. The Mocap velocity stream is the earliest, followed by the wheel odometry with just a 12ms delay and the S300 laser scanner with a bigger delay of 136ms.

An indication that the estimation makes sense is the triangular relationship between the sensor delays. If we sum up the delay between the Mocap and the wheel odometry

Peak Nr	1	2	3	4	Mean
wheel odometry to S300	134	127	110	111	121
wheel odometry to Mocap	11	-21	-19	-19	-12
S300 to Mocap	-129	-148	-132	-134	-136

Table 7.1: Delay values given in ms between the different sensors of the **Sarah** shuttle.

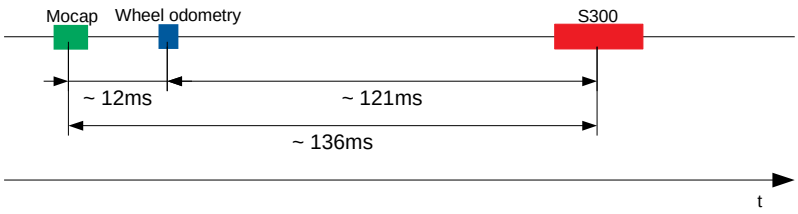


Figure 7.24: Overview of delay values between the sensors of the **Sarah** shuttle.

and the delay between the wheel odometry and the S300 laser scanner we receive, with a little error, the delay between the Mocap and the S300 laser scanner. This makes the estimation consistent and plausible although no ground truth is available.

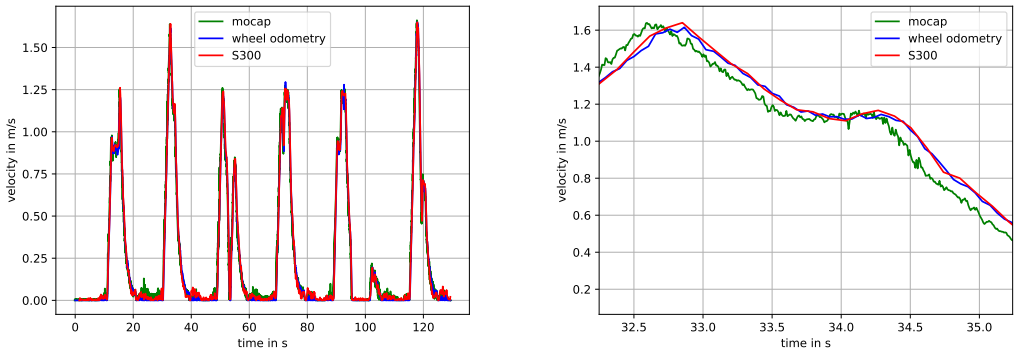
The delay value between the wheel odometry and the S300 laser scanner seems also reasonable considering the used hardware. The serial interface of the S300 scanner needs about 110ms to transfer a data package. Adding some processing and scheduling time makes the 121ms plausible.

7.3.3 Valentina

At the test run of the **Valentina** shuttle, six cycles of driving from one station to another got recorded and the translational velocity of it can be seen in Figure 7.25a. A big difference compared to the **Sarah** shuttle can be observed in Figure 7.25b. The wheel odometry is nearly time synchronized with the S300 laser scanner. This means that the wheel odometry communication path and processing is much slower at the **Valentina** than at the **Sarah** shuttle since they use the same S300 laser scanner via the same interface.

This is confirmed with the results from the delay estimation. The wheel odometry data needs 11ms more from the detection in the sensor until it receives a timestamp than the S300 laser scanner data.

The delay between the Mocap system and the S300 laser scanner kept at a similar value compared to the test with **Sarah**. Whereas, the delay between the Mocap system and the wheel odometry changed. This was obvious because otherwise the triangle relationship could not be fulfilled anymore.



(a) Full test period.

(b) Zoomed.

Figure 7.25: Translational velocity stream of the different sensors on the **Valentina** shuttle.

Peak Nr	1	2	3	4	5	6	Mean
wheel odometry to S300	-10	-18	-2	-14	-15	-6	-11
wheel odometry to Mocap	-148	-147	-128	-137	-138	-137	-139
S300 to Mocap	-133	-129	-131	-123	-124	-131	-129

Table 7.2: Delay values given in ms between the different sensors of the **Valentina** shuttle.

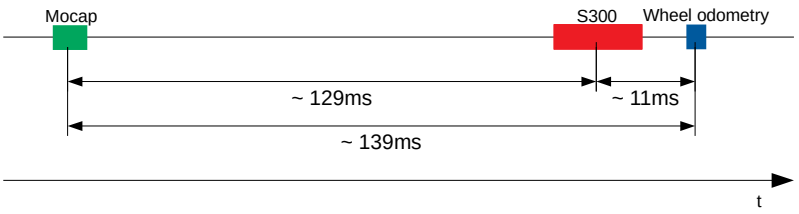
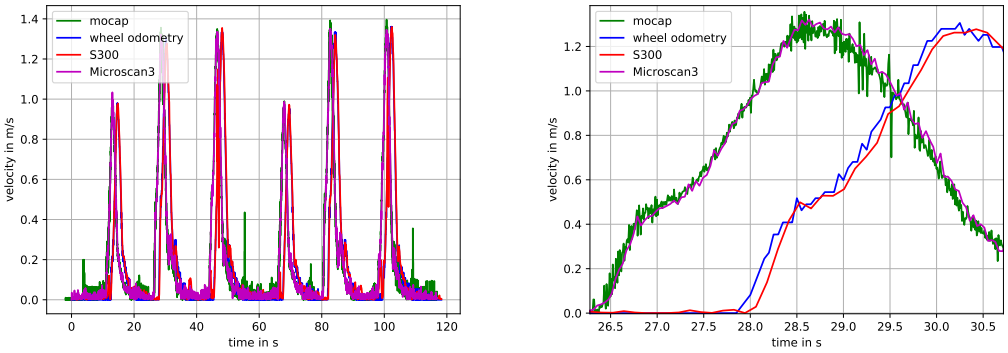


Figure 7.26: Overview of delay values between the sensors of the **Valentina** shuttle.



(a) Full test period.

(b) Zoomed.

Figure 7.27: Translational velocity stream of the different sensors on the **Marie** shuttle.

Peak Nr	1	2	3	4	5	6	Mean
wheel odometry to S300	45	68	54	44	60	43	52
wheel odometry to Mocap	-1581	-1563	-1577	-1581	-1573	-1582	-1576
S300 to Mocap	-1630	-1632	-1638	-1648	-1632	-1634	-1636
wheel odometry to Microscan3	-1546	-1521	-1546	-1555	-1522	-1550	-1540
Microscan3 to Mocap	-26	-33	-14	-22	-40	-26	-27
S300 to Microscan3	-1591	-1594	-1592	-1607	-1602	-1588	-1596

Table 7.3: Delay values given in ms between the different sensors of the **Marie** shuttle.

7.3.4 Marie first try

The first test run with the **Marie** shuttle consists also of six driving cycles. But the velocity streams surprise with a delay of the wheel odometry and the S300 laser scanner to the Mocap system and the Microscan3 of about one and a half-second. This is shown in Figure 7.27b. It is not clear why this happened. A guess is that given through the higher sampling rate of the Microscan3 sensor, the higher CPU usage delays the processing of lower frequent tasks.

The estimation values in Table 7.3 show that the delay between the wheel odometry and the Microscan3 laser scanner is 1540ms. This is a huge value that will lead to problems in subsequent processes of the robot. The delay between the wheel odometry and the S300 laser scanner seems to be in the same order as at the other two shuttles. If we again use the S300 as a reference, the wheel odometry directly taken from the CAN bus is faster than the wheel odometry from the **Valentina** shuttle but slower than the one from the **Sarah** shuttle.

Although the values of 1.5 seconds seemed to be wrong, the triangle relationship of all sensor combinations works again. This is shown in Figure 7.28 and it means that the values are consistent.

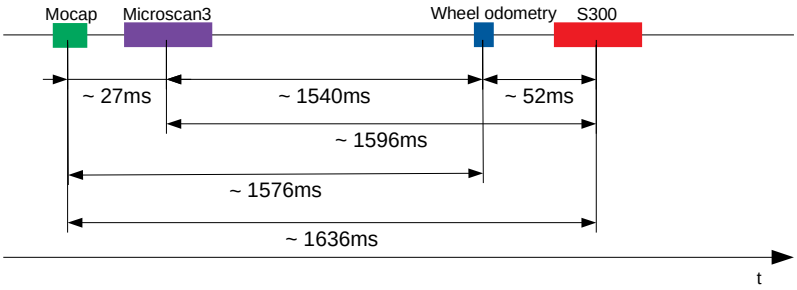
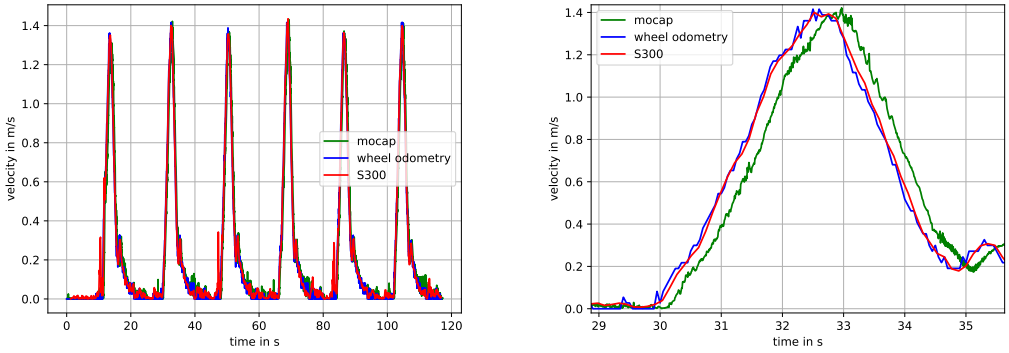


Figure 7.28: Overview of delay values between the sensors of the **Marie** shuttle.



(a) Full test period.

(b) Zoomed.

Figure 7.29: Translational velocity stream of the different sensors on the **Marie** shuttle at the second try.

7.3.5 Marie second try

Because the values of the first **Marie** test run differed significantly from the other shuttles, the test run was repeated with **Marie** and the Microscan3 sensor was removed. This test run had again 6 cycles and showed that the big delays vanish (7.29b). This makes the assumption that the Microscan3 and the higher CPU usage were causing the delay in the first test run plausible.

But also this test run included a surprise. The Mocap system was the last this time which differs from the others where it was always the earliest. Explanations could be that the clock synchronization was off by that delay or the CPU usage at the Mocap-PC was higher than usual. Unfortunately, these properties were not recorded so it is not possible to clearly state the cause of it.

There had to be some troubles at the Mocap-PC at the beginning of the recording because, as it can be seen in Figure 7.30, the velocity stream from the Mocap system was not smooth at the first peak. The problem disappeared until the next peak but made it necessary to ignore the estimations from the first peak. This is the reason why

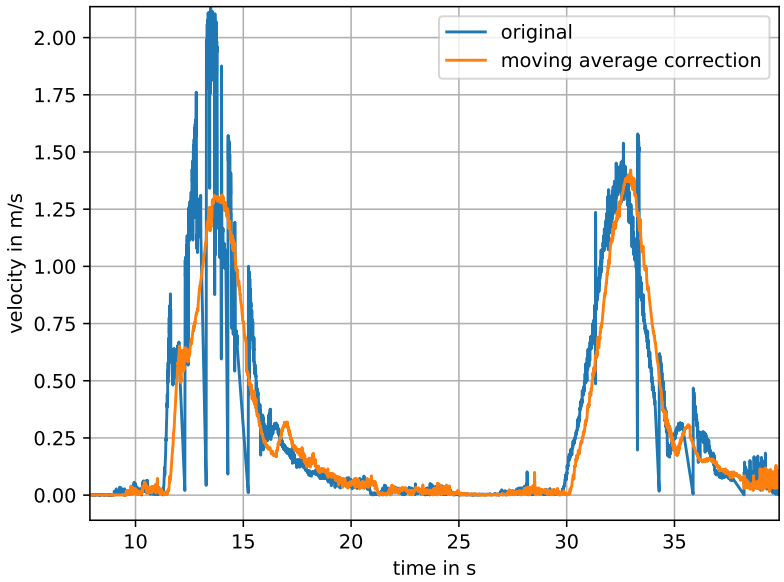


Figure 7.30: Problems causing the Mocap velocity to be useless.

Peak Nr	1	2	3	4	5	6	Mean
wheel odometry to S300	69	38	66	33	66	41	52
wheel odometry to Mocap	(345)	264	273	267	281	270	271
S300 to Mocap	(285)	224	210	235	213	229	222

Table 7.4: Delay values in ms between the different sensors of the **Marie** shuttle at the second try.

all estimations, including the Mocap velocity stream, at the first peak, are placed inside brackets in Table 7.4 and are not incorporated in the mean value.

The triangle relationship also fits for this test run and the delay between the wheel odometry and the S300 laser scanner kept the same. This looks as if they were affected by the Micoscan3 delay in the first test run by the same amount.

7.3.6 Minimum Cells test

The results from the correlation estimations seem plausible because of the triangle relationship and the knowledge about the used hardware. Nevertheless, it is still not 100 percent clear if these values are correct. There is no ground truth available but we like to double check it against another method to see if it leads to the same results.

The second method we used is a trivial version of the map clarity method mentioned in

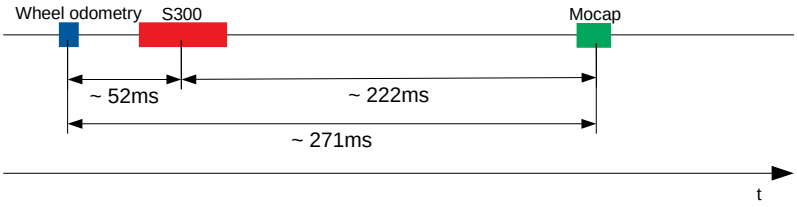


Figure 7.31: Overview of delay values between the sensors of the **Marie** shuttle at the second try.

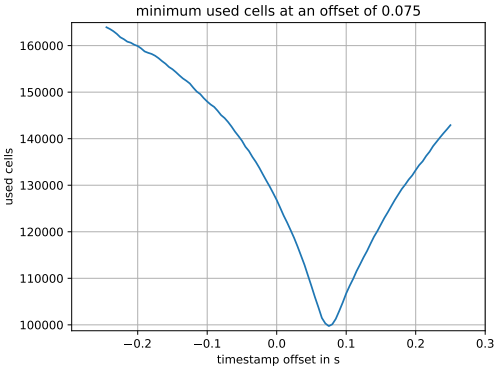
Section 4.1.1 and is called minimum cells. The S300 laser scanner and the transformation from the wheel odometry data was recorded in a bagfile from the test run. The bagfile was replayed and the endpoints of each laser scan get transformed into the map frame. The map is represented as a grid map with a grid resolution of 10 by 10 centimeters. The cells in this grid where the endpoints are placed after the transformation are marked as used cells. These used cells are collected over the entire period of the bagfile. After one test is finished, the number of used cells gets extracted. These steps are repeated with the same bagfile for different subtracted delay values at the S300 laser scanner data.

The key assumption behind this method is that in a static environment if the laser scanner and wheel odometry data are aligned in time the number of used cells has its minimum. At the minimum of used cells, the endpoints of the laser scanner data hit the same cells most often. If it is not the minimum, the endpoints hit nearby cells given through sensors misalignment leading to a kind of blur effect which is also shown in Figure 2.2b.

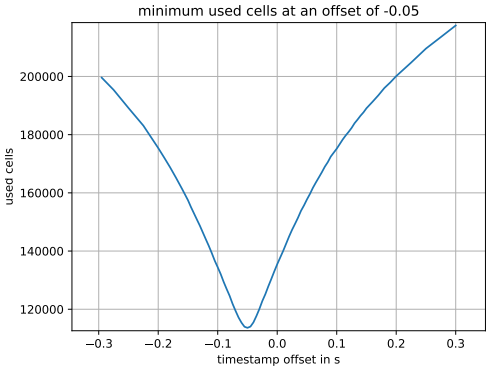
The advantage of this method is that it is very simple to implement in such a mobile robot framework where the transformation between frames for sensor data is already available. The disadvantages of this method are that it incorporates the odometry error, it is only useable for a sensor combination of a laser scanner and wheel odometry and the test run needs to be repeated for every delay value. The delay values were selected in 5ms steps from -300ms to +300ms to keep the processing time at a moderate level.

Figure 7.32 shows how the number of used cells changes if the offset between the wheel odometry and the S300 laser scanner varies. At each test run, a clear minimum is observable. The offsets at each minimum and the results from the correlation estimation method are placed in Table 7.5. This makes a comparison easier.

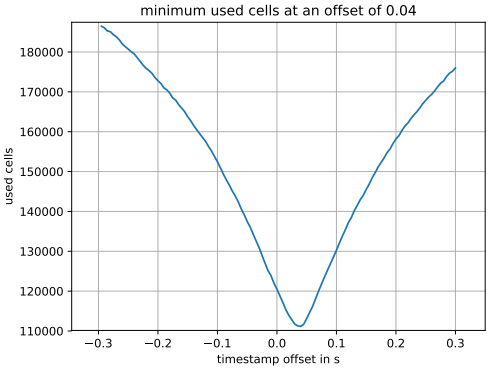
The results of both methods are in the same range. This is a good indication that the estimated values reflect the true value because for two completely different methods it is quite unlikely to deliver the same results if they do not share a common ground truth. Nevertheless, they differ more than expected. For all tests besides **Valentina**s, it seems as the correlation method adds some additional delay to the S300 laser scanner data. The velocity streams always lie on top of each other if the estimated delay from the correlation is corrected similarly as it is shown in Figure 7.21. This means that the



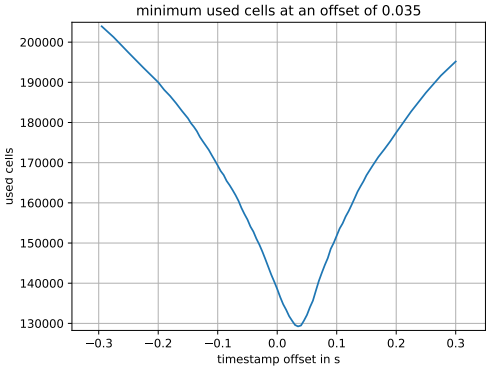
(a) **Sarah**.



(b) Valentina.



(c) **Marie** first try.



(d) **Marie** second try.

Figure 7.32: Cell count variation over time offset between wheel odometry and S300 laser scanner.

	correlation	minimum cells
Sarah	121	75
Valentina	-11	-5
Marie 1st	52	40
Marie 2st	52	35

Table 7.5: Delay value comparisons in ms between the correlation and the minimum cells method.

delay between the velocity streams is estimated correctly but it does not mean that it is the same delay as the delay between the raw sensor data.

The two steps between the raw sensor data and the velocity streams are the timestamp correction and the data conversion. The effect of the timestamp correction on the estimated delay value from the correlation was tested by changing the window size of the average moving filter in the correction step. The results from tests with window sizes up to three times higher than the actually used one never differed more than 5ms from the tests with no timestamp correction. This concludes that the timestamp correction is the cause of a small portion of the approximatly 20ms difference between the correlation and minimum cells results.

The rest of the difference could only be caused by the data conversion step. How the data conversion step changes the timing of the data is not known but it at least needs two subsequent laser scans to output a velocity value for the current scan timestamp. It somehow incorporates past data which could explain an additional delay.

But this does not explain why the value from the correlation method is lower at the **Valentina** test run than the one from the minimum cells method. The only explanation which is left is that the odometry error in the minimum cells test over the test run period of about 2 minutes is casing it.

7.4 Impact of the delay on the localization

The second goal of this thesis is to show how the localization of a mobile robot is affected by a delay between laser scanner and wheel odometry data. To show this, the setup was kept simple by using the standard AMCL implementation for ROS[1]. The S300 laser scanner data, the wheel odometry and the map topic where recorded to a bagfile from the run with the shuttle. A delay value in the range from -0.3s to 0.3s every 0.005s is subtracted from the timestamp of the laser scanner data and the bagfile was then used as the input for the standard AMCL node. This AMCL node outputs the position of the mobile robot in the map frame. The subsequent positions from each test run are recorded. The same test run is repeated with the different delay values. This results in different robot position sequences for every tested delay value.

These position sequences get compared to the position sequences received from the

[1] http://wiki.ros.org/amcl, accessed 20.04.2021

Mocap systems which provides the ground truth. To compare the absolute trajectory error between these position sequences both need to be in the same frame. Therefore, the Mocap positions need to be transformed to the map frame as well. The position values from the test run with a zero delay value were used to find the correct transform by trying different transformation parameters and evaluating the error between the position values. The transformation with the smallest error was then used for further tests.

To compare the positions, the temporal corresponding Mocap position for each position from the AMCL is searched and linear interpolated. The Euclidean distances between each position pair is calculated and the mean value of this position errors over a test run is calculated. This gives a mean position error value for each timestamp delay value which is presented in the following Figures 7.33, 7.35, 7.37 and 7.39 as position error.

The recording of the Mocap system, the randomness of the AMCL, the transformation between frames and the linear interpolation of the Mocap positions to get the temporal corresponding position to introduce errors where we do not know how they will affect the position error value. Therefore, the weight variance of the particles before the resampling step is recorded as well to get a second quantitive value from the behavior of the AMCL. The weight variance was selected because it represents the certainty of the belief as it is described in Section 3.2.5. Again, the mean value of the weight variances over each test run was calculated and is plotted over the corresponding delay values for each shuttle in the Figures 7.34, 7.36, 7.38 and 7.40.

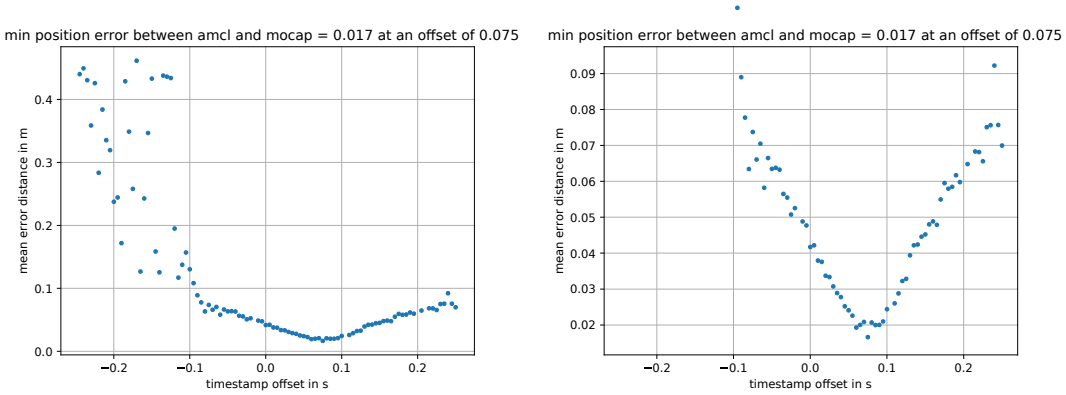
7.4.1 Sarah

Figure 7.33a show a minimum mean position error at a delay value of 75ms and a nearly linear increase in the error as long as the delay value keeps within a range of +-200ms of the offset at the minimum error. If the delay is further off from the optimal delay, the AMCL seems to delocalize which results in a big mean position error of more than 10cm.

But also small differences in the delay value can lead to significant position errors. From Figure 7.33a it can be observed that the mobile robot, in its standard setup with an offset of 0, has a mean position error of 4.1cm whereas this error reduces to 1.7cm when the sensors are correctly aligned. This is a reduction by half of the error value.

The location of the minimum, at an offset of 75ms, corresponds with the result from the minimum cells test. This strengthens the assumption that the minimum cell tests are closer to the real delay value between the sensors and that the result of the correlation estimation method differs for some reason.

Figure 7.34 represents how the mean particle weight variance has changed with the different delay values. The location of the minimum again matches with the results from the position error and the minimum cells test because a low variance in the weights of the particles is associated with the fact that all particles are close together and reach nearly the same weight. If the particles are concentrated in a small area it is a hint that the filter's belief is very certain that the true position is in this certain area.



(a) Position error.

(b) Zoomed.

Figure 7.33: AMCL position performance for different wheel odometry to laser scanner offsets using sensor data from **Sarah**.

The certainty of the filter decreases if the particles start to spread around the true position. This leads to particles that fit better or worse to the received laser sensor data and results in more varying weights of the particles. This can be observed in the figure with the increasing mean weight variance on both sides close to the local minima.

Of course, if the filter delocalizes and the particles are spread randomly over the whole map, the variance in the particle weights decreases again because the laser sensor data fits equally bad to every particle position. This can be observed in Figure 7.34 for the weight variance at offset values below -0.1s where the AMCL delocalizes which we already saw from the position error plot.

7.4.2 Valentina

The shape of the position error plots in Figure 7.35 are similar to those from the **Sarah** shuttle. There is a global minimum. The position error increases linear right next to this minimum and becames big if the AMCL delocalizes. The minimum mean position error is 3.9cm higher than the one at the **Sarah** shuttle which shows that the **Sarah** shuttle is better able to localize. It is not clear why the localization is worse but it could be caused by the different wheel odometry sensors.

What is confusing and where we do not have an explanation for, is the location of the minimum value in these plots. The -100ms does not correspond to the minimum cells test with -5ms or the delay estimation with the correlation method of -11ms. But it seems that the AMCL and Mocap positions would fit the best at this -100ms.

Additionally, the location of the minimum weight variance at a value of -45ms differ from the other tests as well. It is much closer to the delay values from the delay estimation methods but is still off by more than 30ms. This leaves the question of why the position error does not correspond with the weight variance anymore and which delay value is now the correct one (-5ms, -11ms, -45ms, -100ms). Unfortunately, we did not

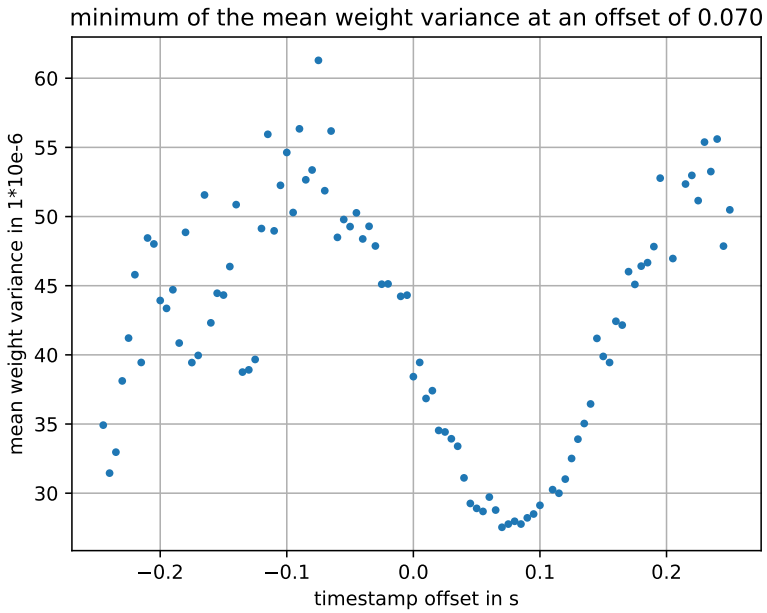
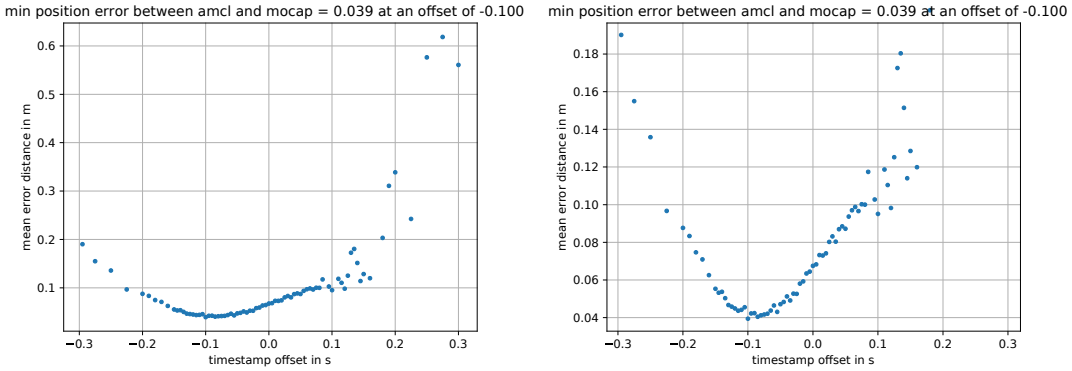


Figure 7.34: Weights variance for different wheel odometry to laser scanner offsets using sensor data from **Sarah**.



(a) Position error.

(b) Zoomed.

Figure 7.35: AMCL position performance for different wheel odometry to laser scanner offsets using sensor data from **Valentina**.

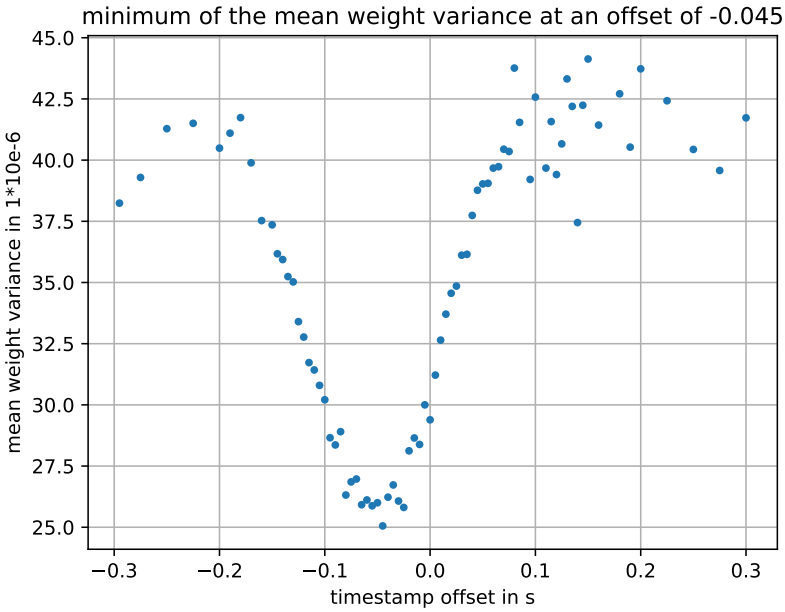


Figure 7.36: Weights variance for different wheel odometry to laser scanner offsets using sensor data from **Valentina**.

found plausible answers to these questions yet.

7.4.3 Marie's first and second try

With the Figure 7.37, the confusion about the positions error behavior concerning the timestamp delay value increased. The results presented in the plot are not reasonable. The only explanation for us is that the big delay of 1.5s between the Mocap and the wheel odometry and S300 laser scanner is not correctly handled by our location test framework and it makes the position comparison useless at this try.

At least Figure 7.39 shows somehow a similar shape to the test of the other two shuttles. But its location of the minimum does not correlate to any other estimation and the fact that all position error values are above 10cm does not speak for a good localization. Especially, if we compare it with the other shuttles where it starts to delocalize at such high position error values.

The weight variance is independent of the delay between the Mocap system and the robot sensors used for the localization. Therefore, the results from the mean weight variance plots in Figure 7.38 and 7.40 to not correlate with the results from the position error evaluation. Nevertheless, they do correlate with the minimum cell tests because the locations of the minima are close to the ones at that tests. This again confirmes that the true delay value will be somewhere around the results of the minimum cells tests and the location of the minimum weight variance. The estimation of the delay with the

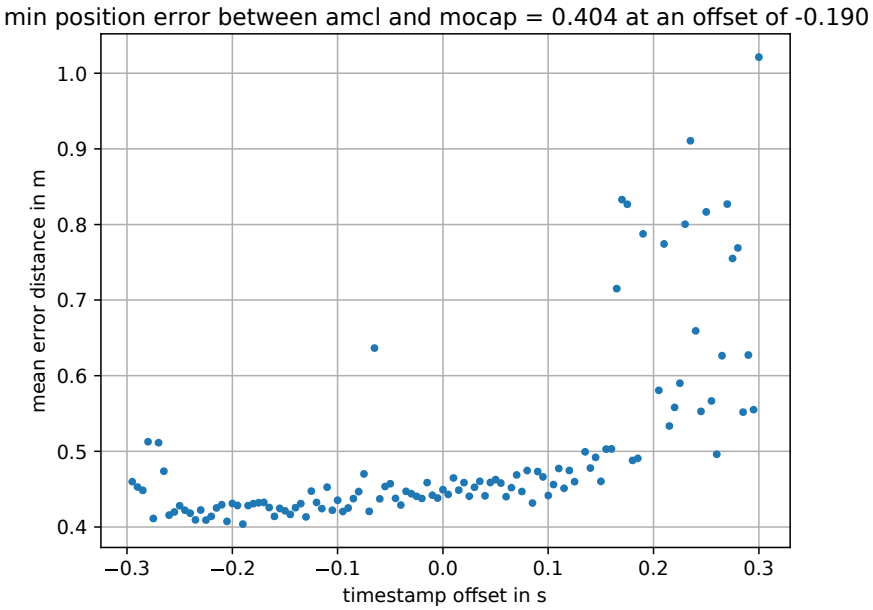


Figure 7.37: AMCL position performance for different wheel odometry to laser scanner offsets using sensor data from **Marie**'s first try.

correlation method is affected by an offset to these values.

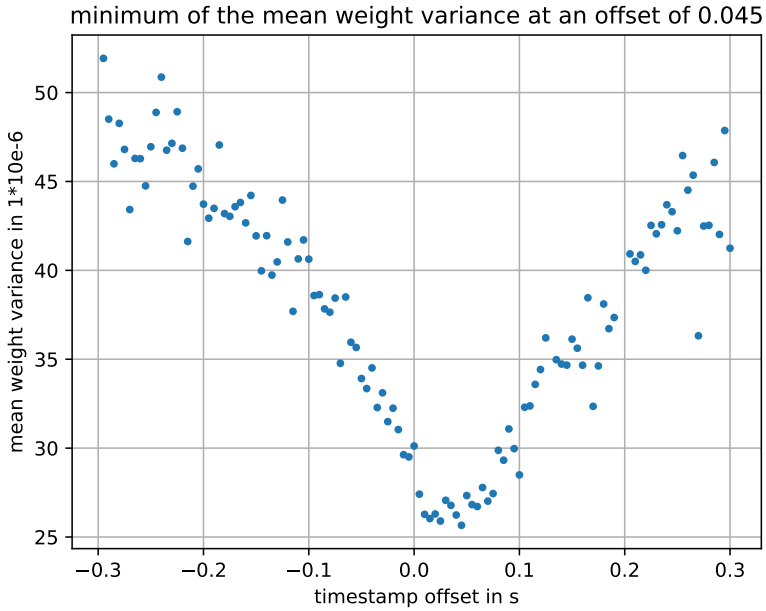
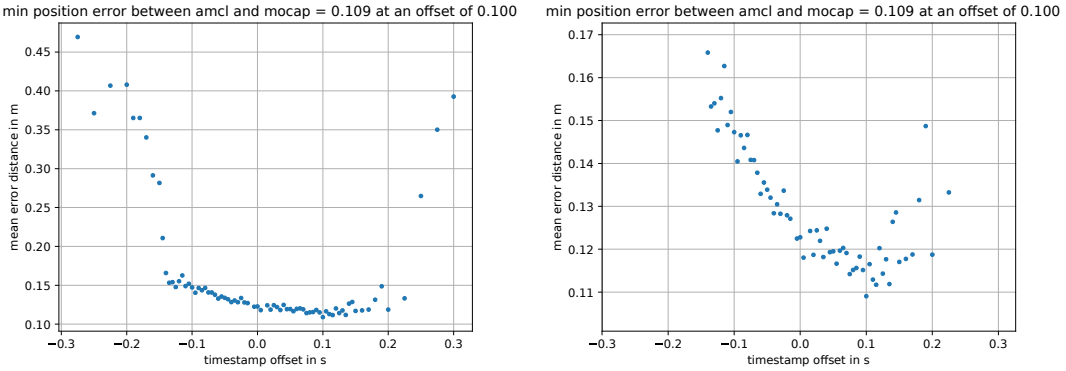


Figure 7.38: Weights variance for different wheel odometry to laser scanner offsets using sensor data from **Marie**'s first try.



(a) Position error.

(b) Zoomed.

Figure 7.39: AMCL position performance for different wheel odometry to laser scanner offsets using sensor data from **Marie**'s second try.

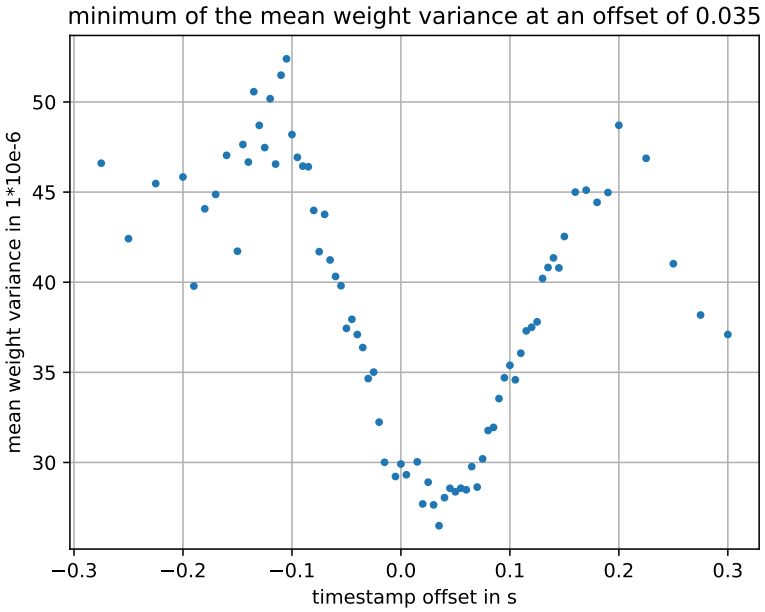


Figure 7.40: Weights variance for different wheel odometry to laser scanner offsets using sensor data from **Marie**'s second try.

8 Conclusion

In this thesis, we present an approach of using the cross-correlation function to estimate the delay between data streams of different sensors. Sensor data streams from real mobile robot shuttles were recorded after they were received at the host PC and timestamped with its clock. The timestamping properties of them got evaluated and corrected. In the next step, the data of all streams got transformed into a common representation. In the final step, the streams in the common representation got compared with the cross-correlation function to extract the value of the delay between them.

The evaluation of the data timestamps revealed that every sensor and communication path has a different jitter in the sampling periods. Correcting this jitter for sensors like a laser scanner, where it is known that the underlying physical process uses a close to constant sampling period, is beneficial. For other sensors like the wheel odometry, the timestamp correction proofed to be counterproductive because the jitter was mainly caused by the jitter in the triggering of the measurement and not by the communication process between the sensor and the host PC. For such sensors, it is not possible to correct the jitter from the communication process but the error introduced by it remains small.

The mobile robot's velocity was selected as the common representation for further investigations because some sensors provide it right away and the transformation of the other sensors was feasible with moderate complexity. This allows the proposed approach to estimate the delay between different sensors during normal operation of the mobile robot which was one key aspect to achieve.

This thesis further presents the realization of three different real mobile robots and the results from the delay estimations. The results show that the delays between different sensors vary from nearly 0 to up to 1700ms. In the context of mobile robots which move with a velocity of up to 2 meters per second, this is a huge amount and needs to be corrected. The results also show that this delay is not constant because it changed on the same mobile robot between two test runs. A cause for this change is the used general-purpose operating system and its scheduling behavior especially during the start-up period of the shuttle.

Additionally, the results from this approach got compared to the results from the minimum cells method. The results were in a similar range but not the same. Given the absence of the ground truth of the real delay value, it is hard to tell which values are the correct ones and what is the cause of this difference.

The last part of the thesis shows how a widely used localization algorithm (AMCL) is affected by the delay value between the sensor data. The correction of the delay between the sensors can result in an improvement of the localization of several cm for the mean

position error. It also showed that, if the delay between the two sensors exceeds +-200ms the AMCL is likely to delocalize which would be the worst-case for a mobile robot.

Furthermore, the evaluation of the localization showed that while the particle weight variance performed well at all different shuttles the absolute position error did not. For the **Sarah** and **Valentina** shuttle, it was plausible and correspondent with the other delay estimations. In contrast to the **Marie** shuttle, where the performance of the absolute position error is not simple to explain.

9 Future work

The results in the evaluation show that the sensor data time alignment is important for the localization of a mobile robot and that the concept of delay estimation with the cross-correlation function works. Nevertheless, there is room for improvement and research to answer some questions raised during the evaluation process.

A big advantage would be if a ground truth of the delay value is available. This will give more insights into why the minimum cells test and the correlation delay estimation do not deliver the same result. Though, getting this ground truth needs specialized hardware.

A next step in the development of the delay correction should be to observe the long-term and restart behavior of the delays between sensors. How do they change after a restart and after several hours or days at a normal workload of a mobile robot shuttle? A hint here is to record the CPU usage as well because the try of one particular shuttle showed indications that this influences the delay value.

Finally, the mean position error evaluation should be reconsidered. The result from the localization evaluation of the first **Marie** test run was not explainable. A possible explanation is, that there went something wrong with the incorporation of the time delay between the Mocap system and the AMCL output.

Bibliography

[ALD03] João Alves, J. Lobo, and J. Dias. "Camera-Inertial Sensor modelling and alignment for Visual Navigation". In: 2003 (cit. on p. 43).

[Bec81] M S Beck. "Correlation in instruments: cross correlation flowmeters". In: *Journal of Physics E: Scientific Instruments* 14.1 (Jan. 1981), pp. 7–19. DOI: 10.1088/0022-3735/14/1/001. URL: https://doi.org/10.1088/0022-3735/14/1/001 (cit. on p. 41).

[BM92] P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791 (cit. on p. 47).

[Cen08] A. Censi. "An ICP variant using a point-to-line metric". In: *2008 IEEE International Conference on Robotics and Automation*. 2008, pp. 19–25. DOI: 10.1109/ROBOT.2008.4543181 (cit. on p. 47).

[Fle+11] M. Fleps et al. "Optimization based IMU camera calibration". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 3297–3304. DOI: 10.1109/IROS.2011.6095067 (cit. on p. 44).

[Foo13] T. Foote. "tf: The transform library". In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373 (cit. on p. 32).

[FRS13] P. Furgale, J. Rehder, and R. Siegwart. "Unified temporal and spatial calibration for multi-sensor systems". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 1280–1286 (cit. on pp. 44, 45).

[HS81] Berthold K.P. Horn and Brian G. Schunck. "Determining optical flow". In: *Artificial Intelligence* 17.1 (1981), pp. 185–203. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. URL: https://www.sciencedirect.com/science/article/pii/0004370281900242 (cit. on p. 47).

[HSK14] Manuel Huber, Michael Schlegel, and Gudrun Klinker. "Application of Time-Delay Estimation to Mixed Reality Multisensor Tracking". In: *Journal of Virtual Reality and Broadcasting*. Vol. 11. 3. 2014. URL: urn:nbn:de:0009-6-38778 (cit. on pp. 43, 46).

[JMG16] Mariano Jaimez, Javier Monroy, and Javier Gonzalez-Jimenez. “Planar Odometry from a Radial Laser Scanner. A Range Flow-based Approach”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm, Sweden, 2016, pp. 4479–4485. DOI: 10.1109/ICRA.2016.7487647. URL: http://mapir.isa.uma.es/mapirwebsite/index.php/mapir-downloads/papers/217 (cit. on p. 54).

[KS14] Jonathan Kelly and Gaurav S. Sukhatme. “A General Framework for Temporal Calibration of Multiple Proprioceptive and Exteroceptive Sensors”. In: *Experimental Robotics: The 12th International Symposium on Experimental Robotics*. Ed. by Oussama Khatib, Vijay Kumar, and Gaurav Sukhatme. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 195–209. ISBN: 978-3-642-28572-1. DOI: 10.1007/978-3-642-28572-1_14. URL: https://doi.org/10.1007/978-3-642-28572-1_14 (cit. on pp. 43, 46).

[LD07] Jorge Lobo and Jorge Dias. “Relative Pose Calibration Between Visual and Inertial Sensors”. In: *I. J. Robotic Res.* 26 (June 2007), pp. 561–575. DOI: 10.1177/0278364907079276 (cit. on p. 43).

[Ler12] Reinhard Lerch. *Elektrische Messtechnik*. Springer-Verlag Berlin Heidelberg, 2012. ISBN: 978-3-642-22609-0 (cit. on p. 40).

[Mai+11] E. Mair et al. “Spatio-temporal initialization for IMU to camera registration”. In: *2011 IEEE International Conference on Robotics and Biomimetics*. 2011, pp. 557–564 (cit. on pp. 27, 45, 46).

[MR06] Christopher Mei and Patrick Rives. “Calibration between a Central Catadioptric Camera and a Laser Range Finder for Robotic Applications.” In: vol. 2006. May 2006, pp. 532–537. DOI: 10.1109/ROBOT.2006.1641765 (cit. on p. 43).

[Ols10] E. Olson. “A passive solution to the sensor synchronization problem”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 1059–1064. DOI: 10.1109/IROS.2010.5650579 (cit. on p. 43).

[QP04] Qilong Zhang and R. Pless. “Extrinsic calibration of a camera and laser range finder (improves camera calibration)”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2301–2306 vol.3. DOI: 10.1109/IROS.2004.1389752 (cit. on p. 44).

[QS18] T. Qin and S. Shen. “Online Temporal Calibration for Monocular Visual-Inertial Systems”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 3662–3669 (cit. on p. 45).

[Qui+09] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: vol. 3. Jan. 2009 (cit. on p. 29).

[Reh+14] J. Rehder et al. “Spatio-temporal laser to visual/inertial calibration with applications to hand-held, large scale scanning”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 459–465 (cit. on p. 45).

[Röw+12] J. Röwekämper et al. “On the position accuracy of mobile robot localization based on particle filters combined with scan matching”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 3158–3164 (cit. on p. 47).

[RSF16] J. Rehder, R. Siegwart, and P. Furgale. “A General Approach to Spatiotemporal Calibration in Multisensor Systems”. In: *IEEE Transactions on Robotics* 32.2 (2016), pp. 383–398. DOI: 10.1109/TRO.2016.2529645 (cit. on p. 43).

[SHN14] Mark Sheehan, Alastair Harrison, and Paul Newman. “Automatic Self - calibration of a Full Field-of-View 3D n-Laser Scanner”. In: *Experimental Robotics: The 12th International Symposium on Experimental Robotics*. Ed. by Oussama Khatib, Vijay Kumar, and Gaurav Sukhatme. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 165–178. ISBN: 978-3-642-28572-1. DOI: 10.1007/978-3-642-28572-1_12. URL: https://doi.org/10.1007/978-3-642-28572-1_12 (cit. on p. 44).

[SJB02] Hagen Spies, Bernd Jähne, and John L. Barron. “Range Flow Estimation”. In: *Computer Vision and Image Understanding* 85.3 (2002), pp. 209–231. ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.2002.0970. URL: https://www.sciencedirect.com/science/article/pii/S1077314202909707 (cit. on pp. 47, 54).

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005. ISBN: 0262201623 9780262201629 (cit. on pp. 19, 34, 35, 37, 39, 40).

[VWW18] R. Voges, C.S. Wieghardt, and B. Wagner. “Finding Timestamp Offsets for a Multi-Sensor System Using Sensor Observations”. In: *Photogrammetric Engineering & Remote Sensing*. Vol. 84. 2018, pp. 357–366 (cit. on p. 44).

[ZI04] M. Zaman and J. Illingworth. “Interval-based time synchronisation of sensor data in a mobile robot”. In: *Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004.* 2004, pp. 463–468 (cit. on pp. 43, 45).

[ZS18] Z. Zhang and D. Scaramuzza. “A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 7244–7251. DOI: 10.1109/IROS.2018.8593941 (cit. on p. 47).