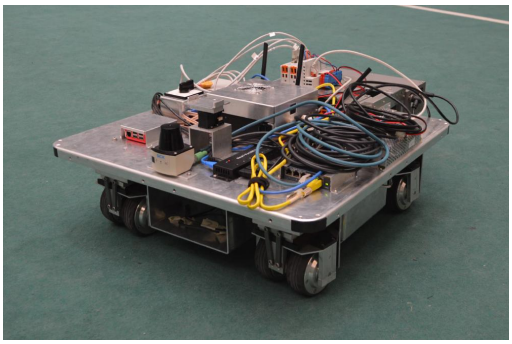
Department of Mechanical Engineering
Control Systems Technology group
CST2019.011

Odometry and localization estimation of a mobile robot with multiple swivel wheels

Bachelor final project

J.P.M. Verhagen



Supervisor:
Prof.dr.ir. Herman Bruyninckx

University coach:
Dr.ir. César López

Company coach:
Ir. Willem-Jan Lamers

Eindhoven, January 2019

Abstract

This report aims to provide details on the development and implementation of an odometry and localization estimator. To desirably estimate the odometry and the positioning of the Robotic Pod (RoPod) as a supplement to usage of external landmarks, a Kalman filter framework has been designed to replace the currently used direct encoder mapping. The framework is able to provide estimates based on a fusion of wheel encoder-, pivot encoder-, gyroscope-, and accelerometer data. The Kalman filter framework consists of two and a half Linear Kalman Filters. It contains separate estimation of orientation and rotational velocity and acceleration using the linear measurement mapping of the gyroscopes, and position and lateral velocity and acceleration using the now linear measurement mapping of the wheel encoders and accelerometer. It contains an additional 'update' stage for the orientation once the first 'update' stage has been completed. This allows the orientation state to be improved upon by measurements that use a non-linear state mapping while still using computationally efficient and mathematically desirable Linear Kalman Filters. Both Kalman filters employ a constant acceleration model and wheel slip is detected by covariance inflation based on the residual in the Kalman equations.

The Kalman filter framework is able to outperform the encoder mapping and provides great accuracy on different floor types. With implementation of this Kalman filter framework, the RoPod is able to localize a worst-case scenario (no external references can be used) for a considerable distance.

An additional sensor, the Accerion Jupiter (v1), which claims to be able to take on the odometry and positional estimation by itself, has been tested on different floor types as well. Although the Jupiter is able to outperform the Kalman filter framework on most surfaces, it is also subjected to considerable drift. Furthermore, the accuracy of the Jupiter sensor was found to be strongly dependent on the RoPod's velocity. This is especially the case on linoleum, a smooth surface which often occurs in hospitals. The designed Kalman filter framework is significantly less affected by varying velocities.

Contents

List of Symbols

Symbol	Quantity	Unit	Abbreviation
A	State-transition model matrix	N.A.	N.A.
a_x	Acceleration in the x-axis of the IMU	meters per second squared	m/s^2
a_y	Acceleration in the y-axis of the IMU	meters per second squared	m/s^2
B	Control input model matrix	N.A.	N.A.
d_w	Distance between the two wheels on a pivot	meters	m
F	Multiplication factor	[-]	[-]
g_z	Rotational velocity around z-axis (yaw)	radians per second	rad/s
H	Matrix mapping state into measurement space	N.A.	N.A.
I	Identity matrix	N.A.	N.A.
K	Kalman gain matrix	N.A.	N.A.
M	Frame transformation matrix	N.A.	N.A.
O	Origin of Cartesian frame	N.A.	N.A.
P	Covariance matrix of state estimate	N.A.	N.A.
p	Position vector	N.A.	N.A.
Q	Process covariance matrix	N.A.	N.A.
R	Measurement covariance matrix	N.A.	N.A.
S	Skew-symmetric angular derivative matrix	N.A.	N.A.
s_w	Offset pivot point and wheel axis	meters	m
s_{ximu}	Offset wheel axis and IMU in x-direction	meters	m
s_{yimu}	Offset wheel axis and IMU in y-direction	meters	m
T	Frame translation matrix	N.A.	N.A.
T_s	Sample time	seconds	s
u	Control vector	N.A.	N.A.
V	Measurement mapping Jacobian	N.A.	N.A.
W	State transition Jacobian	N.A.	N.A.
X	x-axis of Cartesian frame	N.A.	N.A.
x	State vector	N.A.	N.A.
x	Position in x-direction in global frame	meters	m
x_r	Position in x-direction in RoPod frame	meters	m
Y	y-axis of Cartesian frame	N.A.	N.A.
y	Position in y-direction in global frame	meters	m
y_r	Position in y-direction in RoPod frame	meters	m
Z	Frame rotation matrix	N.A.	N.A.
z	Measurement array	N.A.	N.A.
γ	IMU orientation offset angle	radians	rad
δ	Pivot angle w.r.t. RoPod frame	radians	rad
θ	Orientation of the RoPod	radians	rad
λ	Drift percentage	[-]	[-]
σ	Standard deviation	[-]	[-]
ϕ	Lateral wheel velocity	meters per second	m/s

Subscript/Superscript	Interpretation
$(\cdot)_0$	Variable at time step 0
$(\cdot)_i$	Pivot index for the four pivots, $i = 1,2,3,4$
$(\cdot)_{pi}$	Properties related to pivot frame i
$(\cdot)_r$	Properties related to RoPod frame
$(\cdot)_g$	Properties related to global frame
$(\cdot)_k$	Variable at time step k
$(\cdot)_l$	Variable related to left wheel
$(\cdot)_r$	Variable related to right wheel
$(\cdot)^-$	*A priori* estimate

Accents	Interpretation
$\dot{(\cdot)}$	First derivative of variable w.r.t. time
$\ddot{(\cdot)}$	Second derivative of variable w.r.t. time
$\hat{(\cdot)}$	Estimate of variable
$\bar{(\cdot)}$	Mean of variable

1. Introduction

The Robotic Pod (RoPod) project aims to provide 'ultra-wide, ultra-flexible, and cost-effective robotic pods' for handling legacy in logistics. Specifically in hospitals, the RoPod could for example assist nurses with moving beds and moving carts around the hospital from storage locations to the room of a patient. To facilitate these complex goals, the robot will constantly have to know its location and velocity with respect to the environment as well as possible obstructions on its path. The RoPod will use global feature detection methods to obtain information about this.

However, computing odometry and localization based on direct mapping of local motion sensors (which is the current approach by an inverse mapping of wheel- and pivot encoder data) and feature detection is a fault sensitive approach. Direct mapping of local motion sensor can cause estimations which are subjected to noise or drift due to noise or drift from the measurement data. Feature detection not only does not directly compute the RoPod's odometry, its localization can be subjected to errors due to the fact that an object or person might block a so-called reference point, the odometry needs to be computed in a hallway without any reference points or the uncertainty in the global localization is too high. These factors require a more continuous and robust odometry and positioning estimation which when available can be fused with the global reference detection. This improved odometry and positioning estimate should be computed from sensor data that is normally continuously available. Since both encoder- and IMU-data is continuously available, the RoPod essentially obtains an overlap of its odometry estimation. If a trust can be applied to these individual estimations, an improved estimation can be obtained.

To this end, an algorithm will be made that implements the 'belief' or 'trust' in a measurement and computes the best estimate of the RoPod's odometry by combining these measurements relative to their 'trust'. This estimation will then be used in the feedback control loop which allows the RoPod to follow its predetermined- or real-time computed path more accurately when sensors that rely on external landmarks cannot be used.

It can realistically never be expected that perfect or even near-perfect odometry estimation and positioning is obtained with the available sensors. It was reasoned that, in the worst case scenario, no features could be detected in two consecutive hallways and no features could be detected at a crossing. If the RoPod is able to navigate to the second crossing without recalibration using environment features, its 'offline' navigation would be sufficiently robust. It was therefore reasoned that sufficient accuracy was met if the RoPod is able to navigate a 5 meter long hallway, a sharp 90 degree turn followed by another 5 meter long hallway. An absolute positioning difference of 10 centimeters would suffice given the size of the RoPod and the width of the hallway. This results in a relative drift of 1%

2. State estimation and Kalman filtering

This chapter introduces theoretical information on robot odometry estimation (state estimation) and so-called 'sensor fusion' which will be used as means of estimating the odometry and localization. It also introduces some approaches to the problem definition that have shown to be successful in other comparable situations.

2.1 State estimation

State estimation is, as the name suggests, estimating the state of a system based on previous state(s), control input(s), and previous and/or current measurement(s). In most systems, the state or part of the state of the system can not be directly measured (e.g. the rotational velocity of a car driving on a bendy road) and instead should be obtained from the above-mentioned information. Directly transferring the measurement data to state estimation is extremely susceptible to inaccuracies in the data (be it sensor noise, -drift, -bias, or measurement jumps) and therefore provides a poor estimation of the state, especially when computationally integrating the signal where inaccuracies get amplified.

The problems that occurs from directly transferring measurement data can be solved by using an algorithm that combines measurements from different sensors and, using the 'belief' or 'trust' in each of these measurements, provides an improved estimate of the state. Since the sensor data has a certain variance, related to its accuracy, combining two sensors that measure the same value results in two overlapping distributions. Multiplying these distributions results in a more accurate (lower variance) estimate of measured state [1]. This concept is shown in the figures below:

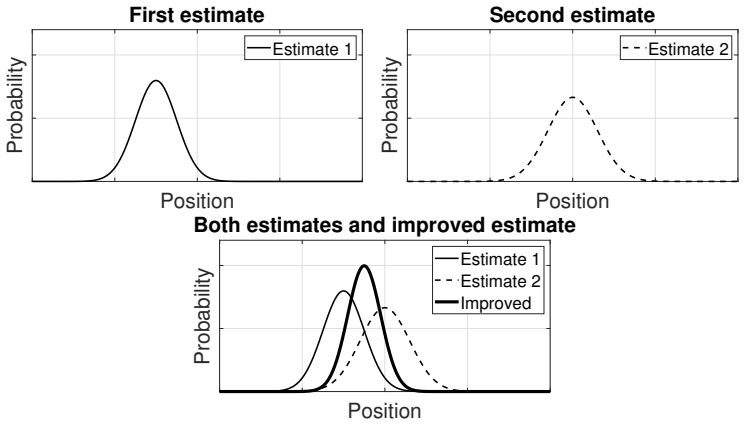


Figure 2.1: Initial and secondary estimate resulting in improved joint estimate

Since information on the odometry is essential for a properly working autonomous robot, many robots use these principles in their state estimation and some have proven to be more successful in certain applications than others. Below, the three filter types that have proven to be most successful and are most widely applied in comparable situations are discussed.

2.1.1 The Linear Kalman Filter

Probably the most common method in state estimation for robotics is the Linear Kalman Filter (LKF) [2]. This algorithm predicts an *a priori* state based on the previous state and the control input. The *a priori* state estimation is then improved upon by a difference in measurement data and expected measurement data. This expected measurement data is obtained from linear mapping of the *a priori* state to the measurement space by solving the forward kinematics. The Kalman gain, which is calculated for each new state estimate, determines how each of the measurements is to be trusted to be incorporated in the state update. The result is an *a posteriori* state estimate which is, after one cycle of the algorithm, the best possible estimate. Intuitively, this follows from the fact that when a measurement of a sensor output differs significantly from the estimated sensor output, it should be incorporated more into the *a posteriori* state (indicating that the state transition from *a priori* to *a posteriori* was insufficient). However if the trust in the accuracy of this measurement is low it should be incorporated less into the *a posteriori* state.

This algorithm assumes linear stochastic difference equations for both state transition and state-to-measurement mapping. It also assumes (and demands) that all errors are Gaussian, meaning a measured value becomes normally distributed when a sufficient amount of measurements are available [3]. In most situations this assumption is permitted since assumptions regarding white noise over measurement data are often sufficient to describe the measurement's characteristics. The recursive operation of the LKF consists of a Time Predict (or simply Predict) stage and a Measurement Update (or Correct) stage [2]. The equations are given below:

Predict

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{2.1}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.2}$$

Correct

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.3}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.4}$$

$$P_k = (I - K_k H)P_k^- \tag{2.5}$$

Superscript minus indicates an *a priori* estimate, A is the state transition matrix, B is the control update matrix, P is the state covariance matrix, Q is the process noise covariance, K is the Kalman gain matrix, H is the matrix mapping state to measurement space, and R is the measurement noise covariance.

As seen in the equations, the LKF requires initial values for the state and the state covariance matrix. It also requires the experimentally obtained process noise- and measurement noise covariance matrices. The recursive LKF algorithm, as directly implemented in applications is given below:

Algorithm 1 Kalman Filter algorithm

Input: state $\hat{x}_0$ and initial covariance matrices Q, P_0 and R
Output: state $\hat{x}_k$, improved covariance matrix P_k

1: $\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$
2: $P_k^- = AP_{k-1}A^T + Q$
3: $K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
4: $\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
5: $P_k = (I - K_k H)P_k^-$
6: return $\hat{x}_k$ and P_k as $\hat{x}_{k-1}$ and P_{k-1}

2.1.2 The Extended Kalman Filter

The assumption of linearity of the state transition and measurement mapping in the Linear Kalman Filter is not always applicable to real life applications. A Kalman filter type that allows non-linearity is the Extended Kalman Filter (EKF) where the nonlinear state transition and state-to-measurement mapping matrices will be linearized around the current estimate [2]. The EKF equations, being similar to those of the LKF, are given below:

Predict

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \tag{2.6}$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \tag{2.7}$$

Correct

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \tag{2.8}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-)) \tag{2.9}$$

$$P_k = (I - K_k H_k)P_k^- \tag{2.10}$$

Here A and W are Jacobian matrix of partial derivatives of f with respect to the state x and process noise w respectively. H and V are Jacobian matrix of partial derivatives of h with respect to the state x and measurement noise v respectively as seen in the equations below:

$$A_k = \frac{\partial f_k}{\partial x_k}(\hat{x}_{k-1}, u_{k-1}, 0) \tag{2.11}$$

$$W_k = \frac{\partial f_k}{\partial u_k}(\hat{x}_{k-1}, u_{k-1}, 0) \tag{2.12}$$

$$H_k = \frac{\partial h_k}{\partial x_k}(\tilde{x}_k) \tag{2.13}$$

$$V_k = \frac{\partial h_k}{\partial u_k}(\tilde{x}_k) \tag{2.14}$$

Again as seen in the equations, the EKF requires initial values for the state and the state covariance matrix. It also requires the experimentally obtained process noise- and measurement noise covariance matrices. The recursive EKF algorithm, as directly implemented in applications is given below:

Algorithm 2 Extended Kalman Filter algorithm

Input: state $\hat{x}_0$ and initial covariance matrices Q, P_0 and R
Output: state $\hat{x}_k$, improved covariance matrix P_k

1: $\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0)$
2: $P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T$
3: $K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$
4: $\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-))$
5: $P_k = (I - K_k H_k) P_k^-$
6: return $\hat{x}_k$ and P_k as $\hat{x}_{k-1}$ and P_{k-1}

It is important to mention that the EKF algorithm, while allowing non-linearity in state transition and measurement mapping, still assumes/demands Gaussian errors. A drawback of using the EKF compared to the normal Kalman Filter is that the now linearized transformations are only reliable if the error propagation is sufficiently approximated by a linear function as well. If this condition does not hold, divergence of the state estimate can be expected.

2.1.3 The Unscented Kalman Filter

While the EKF allows non-linear models (predict and update functions f and h), problems occur when these models are highly non-linear. Since the covariance is propagated through linearization of the non-linear model, the linearization that takes place in a highly non-linear model is not descriptive of the situation at hand and is therefore a bad estimate.

The Unscented Kalman Filter (UKF) extends on the limitations of the EKF by using a deterministic sampling technique. By picking a set of sample points (so-called sigma points) around the mean and propagating them through the highly non-linear functions f and h. Depending on the value of these sigma points after propagation, a new mean and covariance estimate, as required by the Kalman Filter algorithm, can be formed [4].

Even though the UKF algorithm can create a more accurate estimate of the mean and covariance for models with high non-linearity, the problem at hand was reasoned to be only slightly non-linear due to the mapping of pivot angles with sines and cosines. Also due to the high sampling rate of 1000 Hz it was expected that the model appears to behave linearly on the timescale at hand. Validity of this assumption will be discussed in the discussion of this report. Since the UKF was initially not seen as necessary and is not used in the project it will not be further elaborated upon.

2.2 Conclusion

This chapter has introduced multiple approaches for state estimation with possibilities of application in robotics. Limiting the possible approaches to the discrete Linear Kalman Filter and the discrete Extended Kalman Filter gives the following table showing their respective advantages and disadvantages. When the system is better understood, a choice on which approach suits the system best will be made based on other similar applications and these advantages and disadvantages.

Advantages	**Disadvantages**
Best estimator for linear systems	Assumes linear systems
Computationally more efficient	Risk of numerical instability for non-identity covariance matrices

Table 2.1: Advantages and disadvantages of the Linear Kalman Filter (LKF)

Advantages	**Disadvantages**
Allows for non-linearity in estimated process	Still not robust for highly non-linear systems
Allows for non-linearity in measurement mapping	Loses mathematical optimality

Table 2.2: Advantages and disadvantages of the Extended Kalman Filter (EKF)

The main advantage for a Kalman filter in any shape or form is that there is no requirement of any significant memory as opposed to for example Monte-Carlo estimation [5]. The properties of the estimation are computed at every iteration and the history of the estimation is incorporated in the covariance matrix instead of being saved on memory. Additionally, the Kalman filter is able to take the variance of the initial estimate and the variance of the estimation error into account.

A disadvantage of the Kalman filter is that, often in real-life applications, the covariance matrices for R and Q will have to be estimated. Although an elaborate 'training' of Kalman filters is a possibility [6] [7], they are often tuned manually by analyzing and comparing the state estimation to a ground truth when changing a scaling factor. This often results in sub-optimal predictions. To account for this, an optimization has taken place which will be described in chapter 5. Kalman filters also run the risk of becoming unstable due to non-modeled behaviour or undetected model errors. When using a manually tuned Kalman filter in any shape or form, one should generally perform a robustness analysis. This analysis is shown in chapter 7.

3. RoPod Conventions and Description

To properly understand notations in subsequent chapters, it is important to outline the conventions that are used with the RoPod platform. This chapter aims to create a clear image regarding these conventions.

3.1 RoPod on frame level

The RoPod consists of a flat platform with four pivot points located at each corner. The RoPod's platform has its own Cartesian coordinate system consisting of O_r, X_r, and Y_r which makes an angle θ with respect to the global reference frame. With respect to the forward moving direction, the pivot points are numbered as; 1 at front-left, 2 at back-left, 3 at back-right, and 4 at front-right. Each individual pivot point has its own Cartesian coordinate system consisting of O_p, X_p, and Y_p which makes an angle δ_i (where the subscript is the number assigned to the pivot) with respect to the RoPod's platform frame. Figure. 3.1 shows this layout.

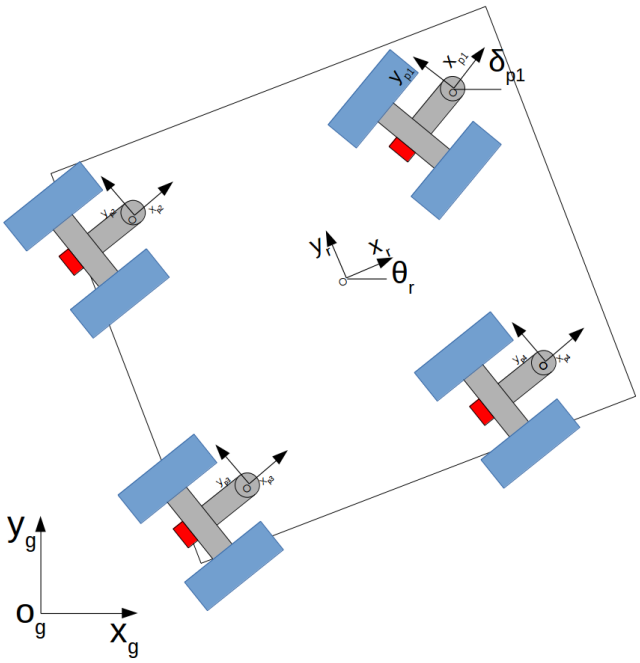


Figure 3.1: Schematic representation of the robotic platform and its relevant axis

3.2 RoPod on pivot level

As previously mentioned, the RoPod has four pivot points. Each pivot point contains two wheels with their axis at offset s_w. The distance between the wheels is d_w. The IMU, which is located on a PCB right behind the wheels, has an additional offset in the x-direction of the pivot frame of s_{ximu} and an offset to the pivot point in the y-direction of the pivot frame of s_{yimu}. The pivot points contain multiple additional Cartesian coordinate frames. The center of the IMU has a frame consisting of $O_{p_i imu}$, $X_{p_i imu}$, and $Y_{p_i imu}$ with previously mentioned offsets $s_w + s_{ximu}$ and s_{yimu}. Each wheel has its own frame at the contact point between the wheel and the ground consisting of $O_{p_i w_i}$, $X_{p_i w_i}$, and $Y_{p_i w_i}$ with previously mentioned offsets s_w and $\frac{d_w}{2}$. Both the wheel frames of each pivot point in the same direction. This will be used in subsequent chapters. Figure. 3.2 shows these conventions for an arbitrary pivot point.

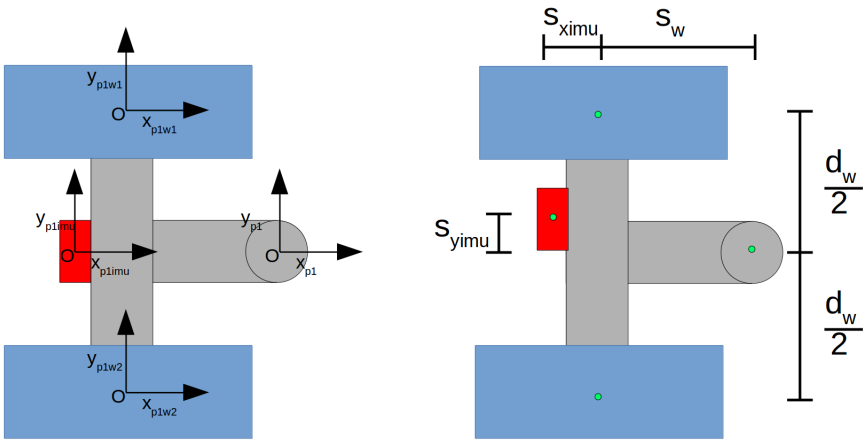


Figure 3.2: Schematic representation of a pivot point and its relevant axis

4. Filter Architecture

As indicated in chapter 2, the system at hand is non-linear which would indicate usage of a single Extended Kalman Filter which is able to accurately estimate the state for slightly non-linear systems. This does however come with an increase in computational complexity but most importantly results in a mathematically sub-optimal state estimation. Instead it was chosen to design a sensor fusion algorithm that first estimates the non-linear states that appear in the measurement mapping which can then be assumed ground truth in an algorithm which uses this part of the state in the estimation of the states which requires these non-linear appearing states. This results in mathematically optimal estimation but does come with the trade-off that measurements which contain these non-linear states can not be used for the estimation of that very same state. This approach has also shown great potential in other robotic odometry estimation cases [11].

Initially, the state which can be mapped to the measurements in a linear manner is limited to $\dot{\theta}$ using the gyroscopes. It was chosen to first estimate the entire orientation state $[\theta \ \dot{\theta} \ \ddot{\theta}]^T$ in a Linear Kalman filter which is then used in the second Linear Kalman filter which estimates the positional states $[x \ \dot{x} \ \ddot{x} \ y \ \dot{y} \ \ddot{y} \ \dot{x}_r \ \ddot{x}_r \ \dot{y}_r \ \ddot{y}_r]^T$. The image below shows the architecture of the initial sensor fusion algorithm including measurement data pre-processing.

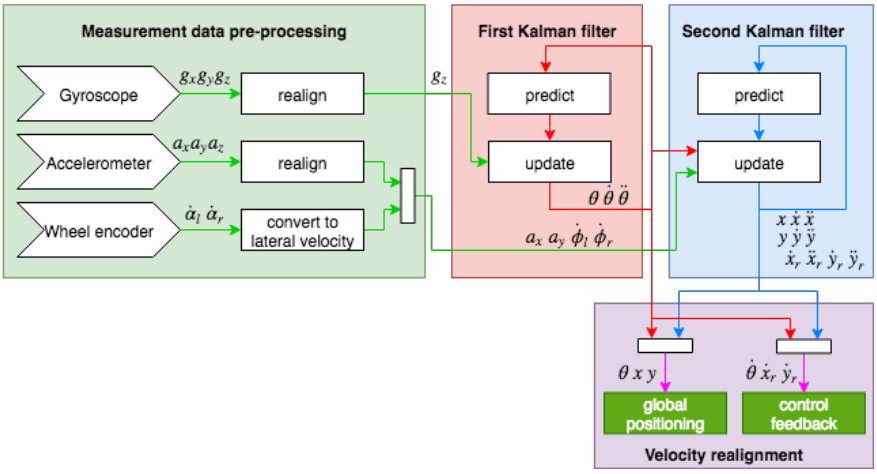


Figure 4.1: Schematic representation of the sensor fusion architecture

Where g_z indicates the rotational velocity of the IMU around the z-axis, a_x and a_y represent the acceleration of the IMU in the x- and y-axis respectively and $\dot{\phi}_l$ and $\dot{\phi}_r$ represent the lateral velocities at the right and left wheel at each pivot. Subsequent sections in this chapter describe the orientation- location Kalman filter in more detail. An improvement upon this two-phase architecture will be described in section 4.3. chapter 5 describes the measurement data pre-processing in more detail.

4.1 First stage Kalman filter

As mentioned previously, the first Kalman filter is a linear assumption algorithm that estimates the orientation state $x = [\theta \ \dot{\theta} \ \ddot{\theta}]^T$ using the gyroscopes around the z-axis. This section describes the assumption that are made regarding pivot angle δ_i and pivot angle velocity $\dot{\delta}_i$, which will be used in the measurement mapping, as well as the state transition, measurement model and precise workings of the first Kalman filter algorithm.

4.1.1 Assumptions regarding pivot angles

The orientation state uses exclusively measurements from the gyroscopes. Since these are located at pivot level it becomes clear that the pivot angle velocity will be apparent in this mapping. The measurement data from δ_i, $\dot{\delta}_i$, and $\ddot{\delta}_i$ is regarded as ground truth in the entire Kalman filter framework.

It is not far fetched that these pivot measurements regarded as ground truth. This has multiple reasons:

1. δ is measured by means of encoders that only contain discretization noise
2. Slip of the wheels in its direction tangential to its movement is not likely to occur since rotational velocities of the ROPOD are relatively low.
3. Due to the weight of the robot and the relative flat surfaces that the robot will drive on, sudden change in δ due to the wheel not being in contact with the surface is not likely to occur.

In the pre-processing stage only a glitch removal will be added to pivot angle and pivot angle velocity measurements. This will be elaborated in chapter 5.

4.1.2 First stage Kalman Filter architecture

The orientation state will be estimated using the gyroscopes that are attached to the wheel base frame of the pivot points. The state is given as

$$x_{kf1} = \begin{bmatrix} \theta & \dot{\theta} & \ddot{\theta} \end{bmatrix}^T \tag{4.1}$$

Since the high sample time ($T_s = 1000Hz$) ensures that the acceleration does not change drastically between time steps, a constant acceleration model offers an excellent prediction of the state at the next time step. The *a priori* estimate could be improved by incorporating a control update matrix B which could model how input voltages can be related to wheel forces which, using complex kinematic model, could predict the acceleration at the next time step. The high sample time makes this unnecessary. The state transition matrix of x_{kf1} is therefore simply given as

$$\hat{x}_k^- = A\hat{x}_{k-1} = \begin{bmatrix} 1 & T_s & \frac{T_s^2}{2} \\ 0 & 1 & T_s \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} \tag{4.2}$$

Based on the current state and the pivot velocity $\dot{\delta}_i$, which is regarded as ground truth, an estimate of the measurement of the gyroscope can be computed using

$$g_{z,i} = \dot{\theta} + \dot{\delta}_i \tag{4.3}$$

Where $g_{z,i}$ is the rotational velocity of the IMU around the z axis. Since the robot contains four pivot points, four estimates of gyroscopic rotation can be computed. Applying Equation 4.3 to the four smart wheels results a description of measurement space and measurement mapping

$$z_{kf1} = \begin{bmatrix} g_{z,1} & g_{z,2} & g_{z,3} & g_{z,4} \end{bmatrix}^T \tag{4.4}$$

$$\hat{z}_k^- = H\hat{x}_k^- + D\dot{\delta}_i = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\delta}_1 \\ \dot{\delta}_2 \\ \dot{\delta}_3 \\ \dot{\delta}_4 \end{bmatrix} \tag{4.5}$$

An example of a measured and predicted gyroscopic rotation is given below. The prediction follows the measured signal accurately meaning the measurement mapping is valid. Note however that the prediction is noisier than the measured signal. This is likely caused by the implementation of the pivot velocity in the equation which is a discrete derivative of the measured pivot encoder signal.

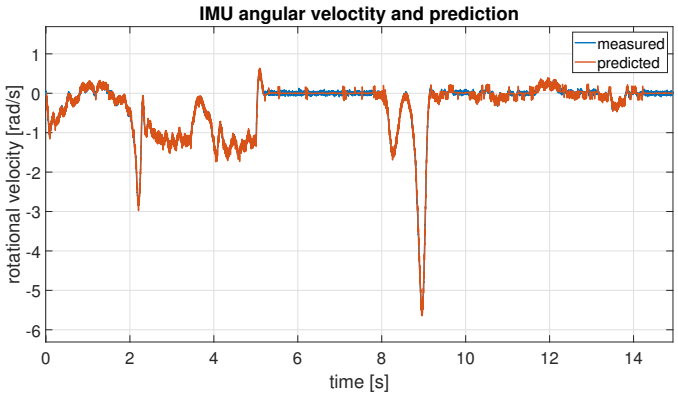


Figure 4.2: Measured and predicted gyroscopic rotation

Using the functions of a common Kalman filter it is now possible to improve the state of θ separately from the state estimation of x and y. The complete algorithm for determining $x_{kf1} = \begin{bmatrix} \theta & \dot{\theta} & \ddot{\theta} \end{bmatrix}^T$ is given in the table below:

Algorithm 3 First stage KF

Input: state $\hat{x}_{k-1}$, pivot angles δ_i, initial covariance matrices Q, P and R, measurements $g_{z,i}$
Output: state $\hat{x}_k$, improved covariance matrices Q, P and R

1: $\hat{x}_k^- = A\hat{x}_{k-1}$
2: $P_k^- = AP_{k-1}A^T + Q$
3: $K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
4: $\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
5: $P_k = (I - K_k H)P_k^-$
6: return $\hat{x}_k$ and P_k as $\hat{x}_{k-1}$ and P_{k-1}

The results of $\hat{x}_k$ are then used in the state estimation of x and y before the algorithm continues. It is now of great importance to find expressions for the covariance matrices Q, and R. This will be described chapter 5.

4.2 Second stage Kalman Filter

The second Kalman Filter is a linear assumption algorithm that estimates the global and local orientation state $x_{kf2} = [x\ \dot{x}\ \ddot{x}\ y\ \dot{y}\ \ddot{y}\ \dot{x}_r\ \ddot{x}_r\ \dot{y}_r\ \ddot{y}_r]^T$. The first six state parameters are used for the global positioning, the last four state parameters are used for determining the local RoPod platform velocity used in the path planning. It was researched that while incorporation of this state is computationally more expensive, it removes the need of mapping global velocities with angle θ which runs the risk of drifting as θ drifts. It was also seen that obtaining global positioning from local velocities is less accurate than incorporating a global state. This chapter describes the usage of the previously estimated state $x_{kf1} = [\theta\ \dot{\theta}\ \ddot{\theta}]^T$, the state transition and measurement models.

4.2.1 Measurement models

To obtain an estimate of the measurements based on the *a priori* state estimate, a mapping that describes the expected measurements of both the wheel encoders and IMU data based on the current state is necessary. By using translation matrices the expected movement of, for example, the IMU within the IMU frame can be expressed as an equation with the variables being part of the *a priori* state. The next sections describe how these measurement models can be obtained.

IMU measurement model

The following formulae describe the translation matrices from IMU, pivot, and robot (subscript *I, P*, and *R* respectively) to its respective overlaying frame (*P, R*, and the global frame *G* respectively).

$$M_I^P = T_I^P = \begin{bmatrix} 1 & 0 & -(s_w + s_{ximu}) \\ 0 & 1 & s_{yimu} \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

$$M_P^R = T_P^R R_P^R = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{(\delta)} & -s_{(\delta)} & 0 \\ s_{(\delta)} & c_{(\delta)} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

$$M_R^G = T_R^G R_R^G = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{(\theta)} & -s_{(\theta)} & 0 \\ s_{(\theta)} & c_{(\theta)} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

Here, s_w is the distance of the pivot centre to the wheel axis, s_{ximu} is the offset of the IMU with respect to the wheel axis in the x-direction, and s_{yimu} is the offset of the IMU with respect to the centre of the wheel axis in the y-direction. Multiplication of these translation matrices gives the homogeneous translation matrix from the global frame G to the IMU frame as seen below:

$$M_I^G = M_R^G M_P^R M_I^P = \begin{bmatrix} c_{(\delta+\theta)} & -s_{yimu}s_{(\delta+\theta)} & x - (s_w + s_{ximu})c_{(\delta+\theta)} + x_r c_{(\theta)} - y_r s_{(\theta)} \\ s_{(\delta+\theta)} & s_{yimu}c_{(\delta+\theta)} & y - (s_w + s_{ximu})s_{(\delta+\theta)} + y_r c_{(\theta)} + x_r s_{(\theta)} \\ 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

Since only the center of the IMU with respect to the global frame are of interest and since the center of the IMU in its own reference frame can be expressed as $p_i^i = [0\ 0\ 1]^T$ the homogeneous translation matrix describing the center of the IMU with respect to the global reference frame can be multiplied by the translation matrix in Equation. 4.9 and a vector describing the coordinates of the center of the IMU with respect to the global coordinate frame can be obtained. The third dimension (height) is in this case of no importance causing the three by

one vector to become a two by one vector with only the 'x' and 'y' coordinates as described below:

$$p_I^G = \begin{bmatrix} x-(s_w+s_{ximu})c_{(\delta+\theta)} + x_r c_{(\theta)} - y_r s_{(\theta)} \\ y-(s_w+s_{ximu})s_{(\delta+\theta)} + y_r c_{(\theta)} + x_r s_{(\theta)} \end{bmatrix} \tag{4.10}$$

Rearranging this leads to an expression that, when differentiated with respect to time, gives an expression for velocity or acceleration of the center of the IMU in the global frame:

$$\begin{bmatrix} x_I^G \\ y_I^G \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c_{(\delta+\theta)} & -s_{(\delta+\theta)} \\ s_{(\delta+\theta)} & c_{(\delta+\theta)} \end{bmatrix} \begin{bmatrix} -(s_w+s_{ximu}) \\ 0 \end{bmatrix} + \begin{bmatrix} c_{(\theta)} & -s_{(\theta)} \\ s_{(\theta)} & c_{(\theta)} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix} \tag{4.11}$$

From now on, the rotational matrix with $(\delta+\theta)$ and (θ) will be described by $R_{(\delta+\theta)}$ and $R_{(\theta)}$ respectively.

$$R_{(\delta+\theta)} = \begin{bmatrix} c_{(\delta+\theta)} & -s_{(\delta+\theta)} \\ s_{(\delta+\theta)} & c_{(\delta+\theta)} \end{bmatrix} \tag{4.12}$$

$$R_{(\delta)} = \begin{bmatrix} c_{(\theta)} & -s_{(\theta)} \\ s_{(\theta)} & c_{(\theta)} \end{bmatrix} \tag{4.13}$$

Taking the first and second derivative of Equation. 4.11 with respect to time results in the following expression for the velocity in the x and y direction of the IMU with respect to the global frame.

$$\begin{aligned} \begin{bmatrix} \ddot{x}_I^G \\ \ddot{y}_I^G \end{bmatrix} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} + (S(\ddot{\delta+\theta})R_{(\delta+\theta)} + S(\dot{\delta+\theta})S(\dot{\delta+\theta})R_{(\delta+\theta)})) \begin{bmatrix} -(s_w+s_{ximu}) \\ 0 \end{bmatrix} \\ + (S(\ddot{\theta})R_{(\theta)} + S(\dot{\theta})S(\dot{\theta})R_{(\theta)}) \begin{bmatrix} x_r \\ y_r \end{bmatrix} \end{aligned} \tag{4.14}$$

Where S matrices are skew-symmetric derivative matrices shown in Appendix A. If Equation 4.14 is mapped from the global frame the the IMU frame, an expression that gives the velocity of the IMU given a certain state is obtained.

$$\begin{bmatrix} \ddot{x}_I^I \\ \ddot{y}_I^I \end{bmatrix} = R_G^I \begin{bmatrix} \ddot{x}_I^G \\ \ddot{y}_I^G \end{bmatrix} = \begin{bmatrix} -c_{(\delta+\theta)} & s_{(\delta+\theta)} \\ -s_{(\delta+\theta)} & -c_{(\delta+\theta)} \end{bmatrix} \begin{bmatrix} \ddot{x}_I^G \\ \ddot{y}_I^G \end{bmatrix} \tag{4.15}$$

Subscript I and superscript I indicates that this expression computes the acceleration of the IMU frame with respect to the IMU frame itself. Notice that this expression can give an estimate of the expected measurement results of the IMU given a certain global state. To obtain measurement mapping based on the local state $[\dot{x}_r \;\; \ddot{x}_r \;\; \dot{y}_r \;\; \ddot{y}_r]^T$, angle θ should be set to zero. The remaining expressions of global velocity then automatically can be referred to as local platform velocities. This obtains expressions of measurement mapping that is not muddied by θ which can be subjected to drift. These expressions will later be used in the Kalman Filter algorithm to compare to the actual measurement readings of the IMU to indicate the accuracy of these measurements. An example of a measured and predicted IMU acceleration is shown in Figure 4.3. Although the prediction shows expected results (the set of measurements is jerk-like movement), the excessive noise in the measurement signal makes the unfiltered or unprocessed IMU data less useful in the Kalman filter algorithm.

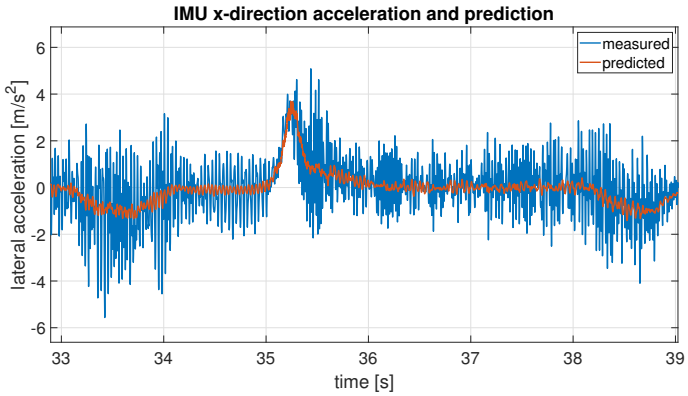


Figure 4.3: Measured and predicted IMU acceleration

An attempt was made at using the integration of the measured signal. The excessive noise made the integration extremely susceptible to the always present integration drift. Resetting the integration to zero at certain times (e.g. both wheel velocities are zero) proved to be a fragile approach. It can be concluded that the unprocessed or unfiltered IMU data will initially have a small influence on the odometry estimation compared to the other measurement sources (gyroscope and encoders). However, subsection 5.1.2 will show that in order to implement a sufficient slip detection, which is necessary to not let the stage be influenced by slipping wheels, the IMU data will have to be preprocessed to make it more viable in the Kalman filter algorithm.

Encoder measurement model

The process of obtaining a measurement model for the wheel encoders from global and local state is similar to that of the IMU in subsection 4.2.1. The only differences are that the equation is only differentiated once with respect to time and that the center of the wheel has a different offset than the IMU. There is similarity in the derivation for the left- and the right wheel. The complete derivation of the left wheel is shown in Appendix A.

An example of a measured and predicted wheel lateral velocity based on the global state is shown in Figure 4.4. The prediction follows the measured signal accurately meaning the measurement mapping is valid and can be used to a larger extend than the prediction of the IMU acceleration.

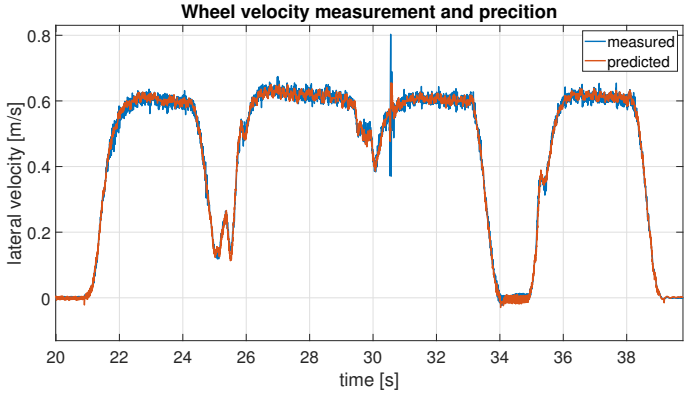


Figure 4.4: Measured and predicted wheel velocity

4.2.2 Second stage Kalman Filter architecture

Now that the estimation of angle θ is sufficiently accurate, the other states of the robot frame with respect to the global frame and the local velocities of the robot frame need to be estimated. For this, the previously introduced second Kalman filter will be used.

Since it is safe to assume that the high sample time ($T_s = \frac{1}{1000}\,s$) makes a constant acceleration state transition permissible, an already linear state transition matrix will be accompanied by a nonlinear state mapping matrix. However, since $\dot{\theta}$ and $\dot{\delta}_i$ are implemented to the state and are, after the first Kalman filter, regarded as the truth the mapping matrix of the state to the measurement space linearizes for every time instant k as will be shown further on. The formulae describing the second state Kalman filter are shown below and are mathematically similar to those of the first Kalman filter.

Algorithm 4 Second stage KF

Input: state $\hat{x}_{k-1}$, pivot angles δ_i, KF1 state $\begin{bmatrix}\theta & \dot{\theta} & \ddot{\theta}\end{bmatrix}$, initial covariance matrices Q, P and R, measurements $\begin{bmatrix}\dot{\phi}_{l,i} & \dot{\phi}_{r,i} & a_{x,i} & a_{y,i}\end{bmatrix}$
Output: state $\hat{x}_k$, improved covariance matrices Q, P and R

1: $\hat{x}_k^- = A\hat{x}_{k-1}$
2: $P_k^- = AP_{k-1}A^T + Q$
3: $K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
4: $\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
5: $P_k = (I - K_k H)P_k^-$
6: return $\hat{x}_k$ and P_k as $\hat{x}_{k-1}$ and P_{k-1}

Since the previous section, the estimation of $x_\theta = [\theta\ \dot{\theta}\ \ddot{\theta}]$, has been computed, both the angles θ and δ can now be considered as ground truth. The state space for the second Kalman filter is

$$x_{xy} = \begin{bmatrix} x & \dot{x} & \ddot{x} & y & \dot{y} & \ddot{y} & \dot{x}_r & \ddot{x}_r & \dot{y}_r & \ddot{y}_r \end{bmatrix}^T \tag{4.16}$$

Since the sample time is still high ($T_s = 1000Hz$) a constant acceleration state transition is still permissible as described in subsection 4.1.2. The state transition matrix of x_{xy} is therefore equal to

$$\hat{x}_k^- = A\hat{x}_{k-1} = \begin{bmatrix} 1 & T_s & \frac{T_s^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & T_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T_s & \frac{T_s^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T_s & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & Ts & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & Ts \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ y \\ \dot{y} \\ \ddot{y} \\ \dot{x}_r \\ \ddot{x}_r \\ \dot{y}_r \\ \ddot{y}_r \end{bmatrix} \tag{4.17}$$

The measurement space for the second Kalman filter looks as follows:

$$z_{xy} = \begin{bmatrix} \dot{\phi}_{l,i} & \dot{\phi}_{r,i} & a_{x,i} & a_{y,i} & \dots \end{bmatrix}^T \tag{4.18}$$

Where $\dot{\phi}_{l,i}$ is the lateral velocity of the left wheel of pivot i and $a_{x,i}$ is the acceleration of the IMU in the x-direction of pivot i. Although Equation 4.15 and Equation A.7 show that

measurement mappings are functions of $\dot{x}$, $\ddot{x}$, $\dot{y}$, $\ddot{y}$, $\dot{x}_r$, $\ddot{x}_r$, $\dot{y}_r$, $\ddot{y}_r$, θ, $\dot{\theta}$, $\ddot{\theta}$ δ_i, $\dot{\delta}_i$, and $\ddot{\delta}$, the measurement mapping matrix H does not need to contain every term of these equations. Only those that are linearly related to variables that are implemented into the state. The H matrix, simplified by using function statements, is due to its large size shown in Appendix A. Since the entire measurement space only contains velocity and acceleration measurements ($\dot{\phi}_{l,i}$, $\dot{\phi}_{r,i}$, $a_{x,i}$, and $a_{y,i}$), the shape of the matrix is only logical. Velocities of the current time step are only dependent on other velocities of that same time step and accelerations of the current time step are only dependent on other accelerations of that same time step. This is the case for both the global state measurement mapping and the local state measurement mapping.

4.3 Improving the orientation Kalman filter

The initial Kalman filter architecture as seen in Figure 4.1 was made due to the limitations of the non-linearity of θ in the mapping of the wheel encoders and IMU. This limitation results in an estimate of $x = [\theta \ \dot{\theta} \ \ddot{\theta}]^T$ that is not able to incorporate measurements of the wheel encoders and IMU accelerations although they contain valuable information on this state. This begs the question if the orientation state could not be further improved by another LKF that uses the *posteriori* estimate of θ (since this appears non-linearly in the measurement mapping) and regards $\dot{\theta}$ and $\ddot{\theta}$ as updateable states (using the *a priori* estimate from the first Kalman filter). This possibility can be seen when the solution of Equation A.7 is worked out

$$\dot{\phi}_{l,i,est} = -\frac{d_w\dot{\delta}_i}{2} + \dot{x}\cos(\delta_i+\theta) + \dot{y}\sin(\delta_i+\theta) - \underline{\dot{\theta}(\frac{d_w}{2} + y_{r,i}\cos(\delta_i) - x_{r,i}\sin(\delta_i))} \tag{4.19}$$

Notice that in this equation not only $\dot{x}$ and $\dot{y}$, which are used in the second Kalman filter, appear linearly but also $\dot{\theta}$ (underlined). This allows, now that θ has been sufficiently estimated in the first Kalman filter, the wheel velocities to improve the orientation state.

Similarly, a written out Equation 4.14 shows a linear appearing $\ddot{\theta}$ which allows the IMU data to improve the orientation state as well.

$$a_{x,i,est} = \ddot{x}\cos(\delta_i+\theta) + \ddot{y}\sin(\delta_i+\theta) + s_{ix}(\dot{\delta}_i^2+\dot{\theta}^2) - \dot{\theta}^2 x_{r,i}\cos(\delta_i) - \dot{\theta}^2 y_{r,i}\sin(\delta_i) + 2s_{ix}\dot{\delta}_i\dot{\theta} + \underline{\ddot{\theta}(x_{r,i}\sin(\delta_i) - y_{r,i}\cos(\delta_i))} \tag{4.20}$$

Here $\dot{\theta}$ still does not occur linearly in the measurement mapping of the IMU so it can still not be implemented in a linear Kalman filter. For this $\dot{\theta}$ the *posteriori* estimate from the first Kalman filter will be used. The incorporation of Equation 4.19 and Equation 4.20 for the left- and right wheel and the x- and y-acceleration results in a $[20 \times 3]$ measurement mapping matrix H_{impr} (subscript *improved*) which is shown in Appendix A. It is now able to incorporate every possible measurement into the prediction of the orientation state with the measurement space as

$$z_{impr} = \begin{bmatrix} g_{z,i} & \dot{\phi}_{l,i} & \dot{\phi}_{r,i} & a_{x,i} & a_{y,i} & \dots \end{bmatrix}^T \tag{4.21}$$

The schematically shown architecture of the initial sensor fusion algorithm in Figure 4.1 changes accordingly to Figure 4.5.

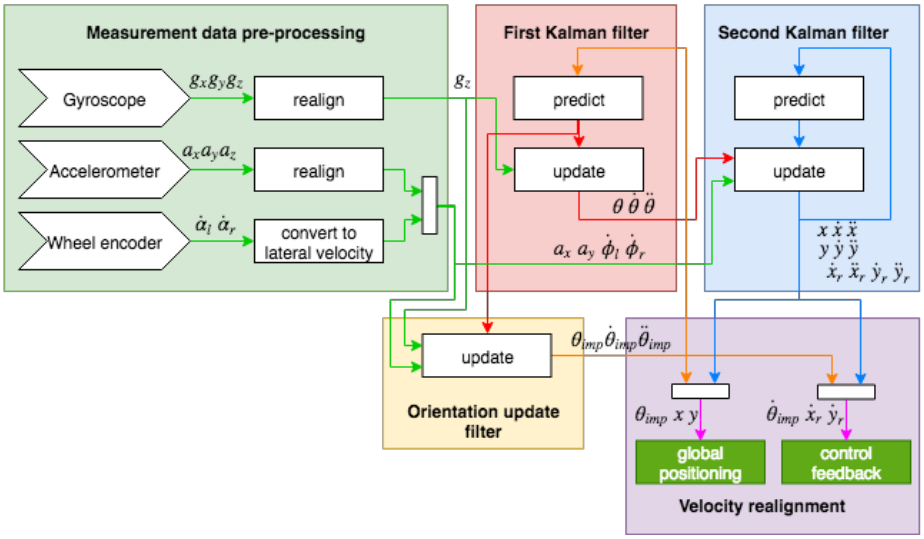


Figure 4.5: Schematic representation of the improved sensor fusion architecture

This sensor fusion design, compared to the initial sensor fusion architecture, has a higher computational costs but improves the estimation of the orientation as will be shown in chapter 7. This elaborate application of the LKF was not noticed in existing literature which would indicate that the trade-off of optimality and computational cost between a two-and-a-half LKF and an EKF could be an interesting research subject. This two-and-a-half LKF is named this way since it contains two *predict* stages (the first and the second Kalman filter) and three *update* stages (the first and the second Kalman filter and the orientation improvement stage).

5. Sensor and Process Identification

To accurately predict measurements and update the state, accurate measurement results are required. This chapter describes how sensor information is preprocessed before usage in the Kalman filter algorithm. As seen from the Kalman filter equations, covariance matrices of P (*a priori* state belief covariance), Q (state transition covariance), and R (measurement covariance) have to be constructed and initialized. These matrices give a representation for the trust in a certain variable and are vital to the accuracy of the algorithm. The second part of this chapter will look in depth at the construction of these matrices.

5.1 Measurement data pre-processing

This part of the chapter describes the pre-processing that the raw measurement data undergoes before being used by the one of the two Kalman filter algorithms.

5.1.1 First Kalman filter measurement data

Gyroscopic offset and drift

The first Kalman filter, estimating $x_\theta = [\theta \ \dot{\theta} \ \ddot{\theta}]^T$, only uses a gyroscopic measurement around the z-axis. A bias has first been determined based on static measurements. Although this would indicate a direct implementation of the measured signal the IMU, which measures the angular gyroscopic velocities, is placed on a vertical PCB right behind the wheels. This causes the axis of the IMU to not align with the axis of the pivot point (which should point in the same direction). To align these axis, measurements where performed where pure rotations of the pivot points where taking place. Although only the realigned z-axis of the IMU is supposed to be able to measure this movement, the IMU reads values around all of their axis as can be seen in Figure 5.1
Close examination of the unaligned accelerometer indicates that the unaligned x-axis is sufficiently zero and therefore has a negligible share in the total gyroscopic measurement around the aligned z-axis, the unaligned z-axis shows a small deviation from the zero line. Using first a rough realignment (y-axis unaligned becomes z-axis roughly aligned etc.) the angle γ that indicates the angular offset of the finely aligned z-axis to the roughly aligned z-axis of the IMU can be determined via the following formula.

$$\gamma = \text{mean}\left(atan\left(\frac{g_{y,i}}{g_{z,i}}\right)\right) \tag{5.1}$$

Meaning the angle γ is first calculated for every iteration and then averaged to obtain the best estimate of γ. This is done for all four gyroscopes.

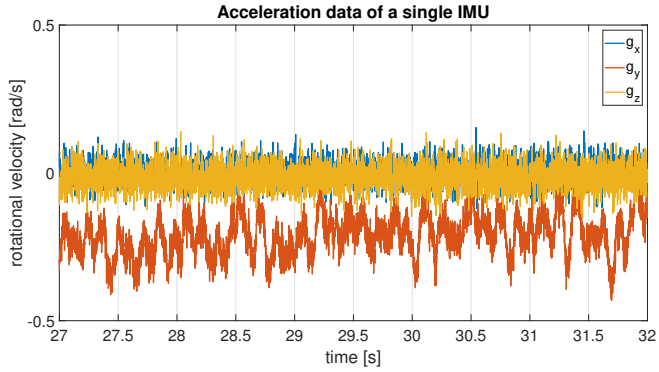


Figure 5.1: Unaligned gyroscope measurements of the IMU

Whereas gyroscopes are often susceptible to significant drift, close examination of static measurements shows otherwise. Figure 5.2 indicates that although the measurements are noisy (with 2σ at roughly 0.045 rad/s), the mean remains static over a time frame of almost 5 minutes. Figure 5.3 shows that the noise approaches a Gaussian distribution allowing it for application in a Kalman filter framework. The Gaussian shows increased power density at specific rate of turn measurements. The cause of this is unknown but even this increased power density is Gaussian shaped therefore not interfering with the workings of the Kalman filter equations.

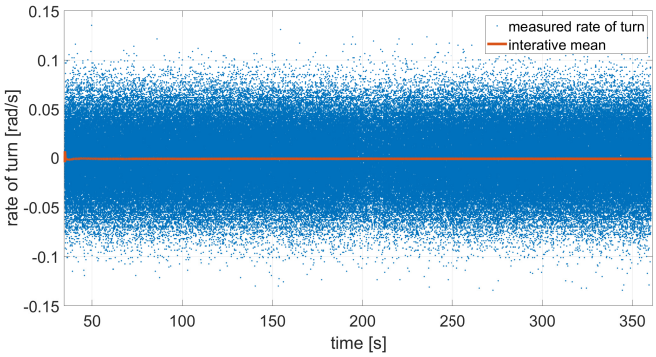


Figure 5.2: Measured static noise and iteratively computed mean

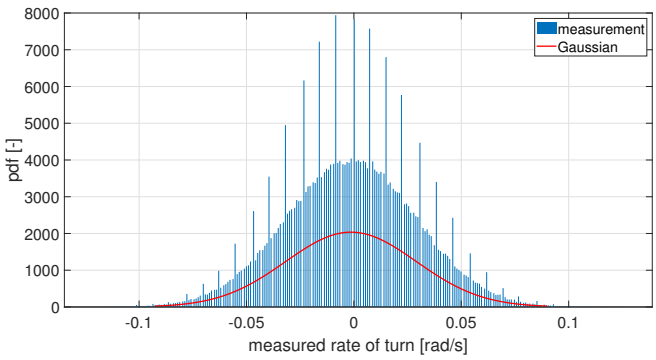


Figure 5.3: Static noise characteristics of the gyroscope

Pivot encoder velocity glitches

Since the pivot angular velocity $\dot{\delta}_i$ is regarded as ground truth as mentioned earlier, only a glitch remover has been applied to this signal. There is hardly any noise due to vibration apparent in the measurement data of the pivot encoders indicating that low-pass filtering is not required. Since rotational velocities of the pivot is not actuated, the glitch remover merely checks whether the difference between $\dot{\delta}_{i\{k+1\}}$ and $\dot{\delta}_{i\{k\}}$ is permissible. If this is not the case, and the value of $\dot{\delta}_{i\{k+1\}}$ is either much larger or much smaller, a glitch has occurred and the value of $\dot{\delta}_{i\{k\}}$ gets chosen instead.

5.1.2 Second Kalman filter measurement data

The second Kalman filter, estimating $x_{xy} = [x \ \dot{x} \ \ddot{x} \ y \ \dot{y} \ \ddot{y}]^T$, uses measurement data from the accelerometer in local x- and y-direction and rotational velocity of the left and right wheel.

IMU offset and drift

The local axis of the accelerometer correspond to the local axis of the gyroscope (they are located on the same IMU chip) and can therefore be realigned with the same angle γ obtained in Equation 5.1 [12]. Since the acceleration in the x- and y-direction should be zero during a static measurement, a bias of each component can be determined as well. This is done for all four pivot points. Figure 5.4 shows, similarly to Figure 5.2, that the measurements are noisy (with 2σ at roughly 0.052 m/s^2) and the mean remains static over a time frame of almost 5 minutes. Similarly, Figure 5.5 shows that the noise approaches a Gaussian distribution as well also allowing this signal to be used in a Kalman filter framework. The figures are limited to the realigned x-direction but the y-direction shows similar results.

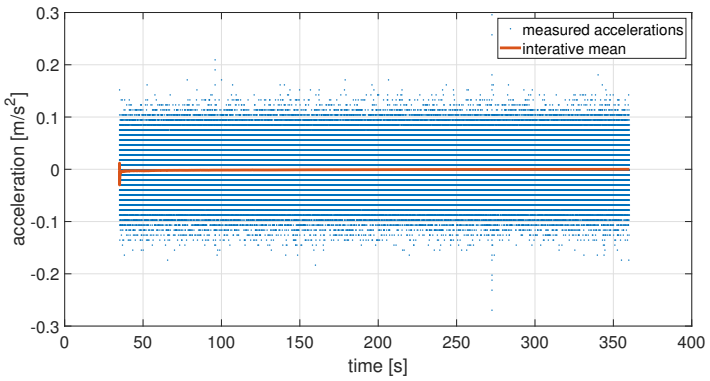


Figure 5.4: Measured static noise and iteratively computed mean in x

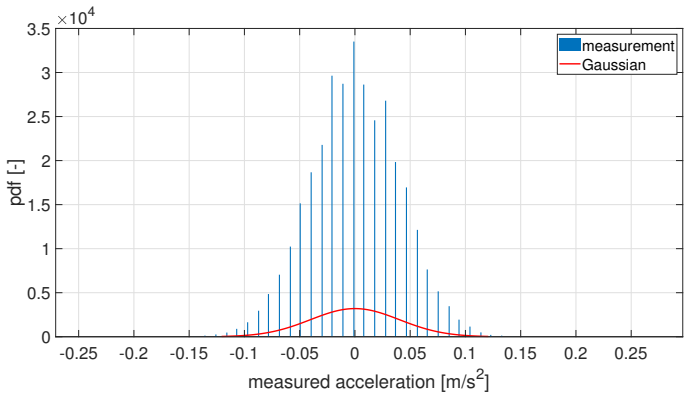


Figure 5.5: Static noise characteristics of the IMU in x

From Figure 4.3 and the figures above it can be seen that the IMU measurements, compared to encoder wheel velocities and gyroscopic data, contains excessive noise limiting the extent to which these signals can be used in the Kalman filter. Although limiting this Gaussian distributed noise as seen in Figure 5.5 would theoretically not necessarily improve the state estimation it will allow for a more elaborate and rigid approach for slip detection. To this end, the frequency distribution of the IMU signals while driving with a constant velocity on a straight line were analyzed showing potential for low-pass and notch filtering as can be seen in the figure below showing the single-sided amplitude spectrum of the x-acceleration respectively. In the rest of this section the y-acceleration is subjected to the same pre-processing with only slight changes which is shown in Appendix A

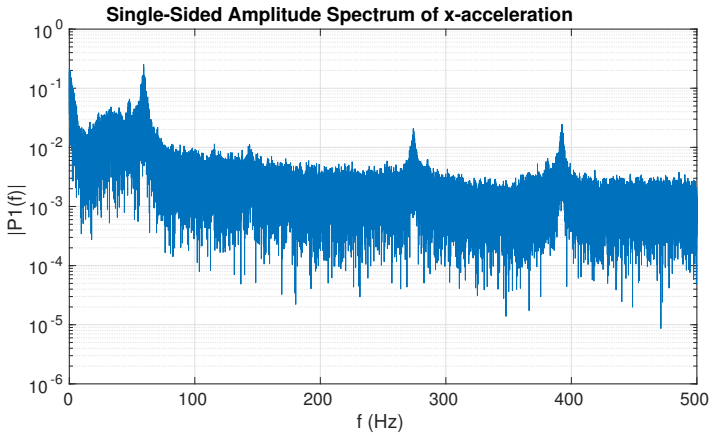


Figure 5.6: Single sided frequency density x

Figure 5.6 shows a significant peak at 60 Hz and less significant peaks at 270 and 390 Hz. Different pivots show similar results. The peak at 60 Hz, only showing in the x-direction is likely caused by the eigenfrequency of the PCB on which the IMU is mounted. The PCB is mounted parallel to the wheel's axis causing only the x-direction to be subjected to this eigenfrequency. As filter type, an aggressive IIR low-pass filter was chosen. This filter type, compared to the more well-known FIR low-pass filter, has both the properties of a low-pass- and a notch-filter. The filter's magnitude and phase response for the x-direction is shown below.

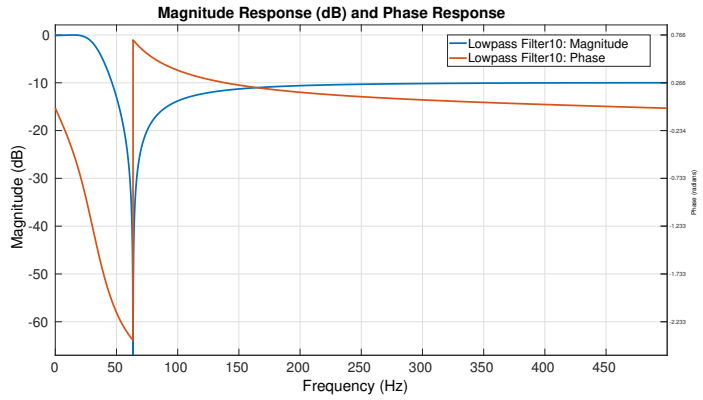


Figure 5.7: IIR magnitude and phase response of measurements in the x-direction

These figures show the stop-band edge frequency at 60 Hz according to the peaks in the frequency density of the measurements in the x-direction as shown in Figure 5.6. The filtered signal is shown in the figure below showing diminished eigenfrequency peaks and white noise.

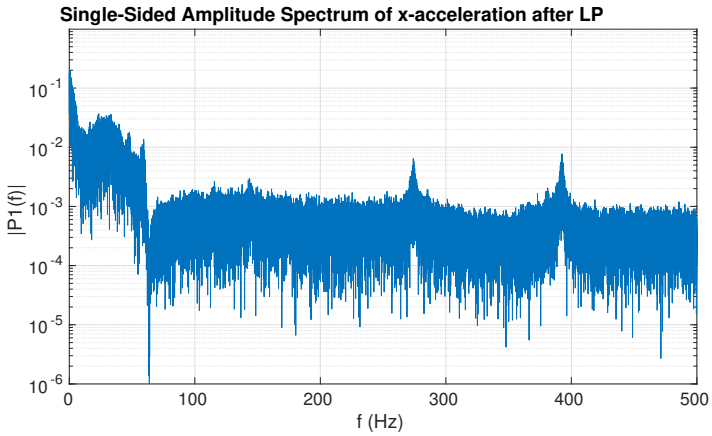


Figure 5.8: Single sided frequency density of x-direction IMU measurements after LPF

This allows the Kalman filter to assign more trust to the IMU measurement and therefore take these measurements more into account with respect to the wheel encoders. As previously mentioned, this approach does not only increase the accuracy of the state prediction, it also allows for a more rigid approach to slip detection as will be explained in 5.2.2.
For now it suffices to indicate that the prediction and the measurement of the IMU acceleration in the x-direction are still valid and only contain less noise as opposed to the IMU prediction and measurement in Figure 4.3. This is shown in the figure below.

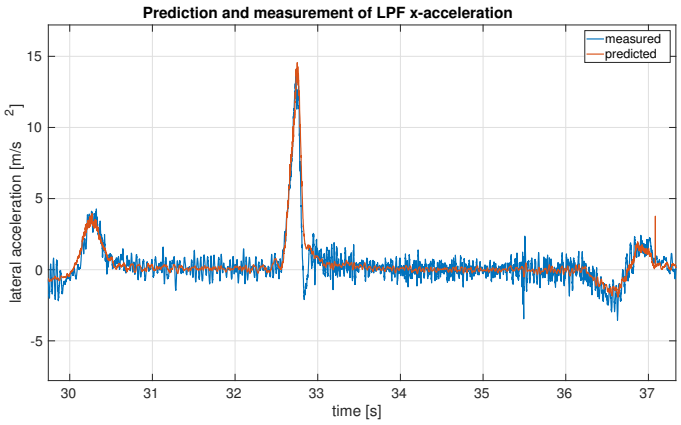


Figure 5.9: Measured and predicted IMU acceleration after LPF

Wheel encoder velocity glitches

The rotational velocities of the left and right wheel are first checked by a glitch removal function. This glitch removal function, just like the pivot angular velocity glitch removal function, checks whether the difference between $\dot{\phi}_{i\{k+1\}}$ and $\dot{\phi}_{i\{k\}}$ is permissible. If possible glitches are removed, the angular velocity gets multiplied by the wheel radius to obtain a lateral velocity. This lateral velocity is used for the measurement mapping as described in Equation 4.18. Figure 5.10 shows an example of a rotational velocity of a wheel containing glitches.

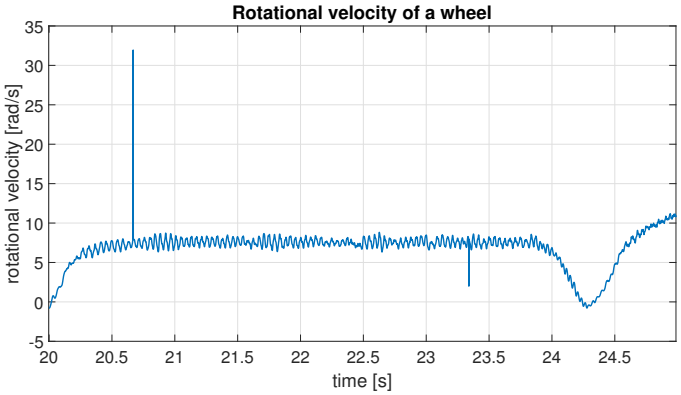


Figure 5.10: Rotational velocity of a wheel with two apparent glitches

5.1.3 Improved orientation Kalman filter

Since the Kalman filter *update* sequence of the improvement of the orientation state uses all the aforementioned measurements, no additional pre-processing is required.

5.2 Covariance construction and initialization

This part of the chapter describes how the previously mentioned Q and R matrices are constructed to accurately describe the processing and to ensure performance of the Kalman filter algorithm. Q is the process noise covariance and assigns a trust to the linear state transition models for both Kalman filters in Equation 4.2 and Equation 4.17. R is the measurement noise covariance and assigns a trust (related to the accuracy) of the measurements for both Kalman filters.

5.2.1 First Kalman filter covariance matrices

Process noise covariance Q

Due to the previously mentioned constant acceleration model and states with sample time dependency, a straightforward process noise covariance matrix was chosen which has shown promising results in similar odometry estimation cases [11].

$$Q = \begin{bmatrix} \frac{T_s^4}{4} & \frac{T_s^3}{2} & \frac{T_s^2}{2} \\ \frac{T_s^3}{2} & T_s^2 & T_s \\ \frac{T_s^2}{2} & T_s & 1 \end{bmatrix} Q_{mult} \tag{5.2}$$

The matrix gets multiplied scaling factor Q_{mult}. Since the state estimation depends on the relative covariance in the process noise- and measurement sensitivity matrices, only a single scaling factor is required. This scaling factor is optimized by minimizing an error computed as the difference between Kalman filter estimate and supposed ground truth divided by the driven distance, averaged for multiple instances. More details in subsequent sections.

Measurement sensitivity covariance R

In the case of estimating state $x = [\theta \ \dot{\theta} \ \ddot{\theta}]^T$, the measured variables $z_\theta = [g_{z,1} \ g_{z,2} \ g_{z,3} \ g_{z,4}]^T$, can be said to depend on one another since a geometry with roughly equal orientations of the pivot points is far more likely to occur then an orientation where, for example, pivot point 1 and pivot point 2 are pointing in opposite directions. Using the standard sample covariance formula in Equation 5.3, it becomes possible to calculate the covariance of the measured variables.

$$\sigma_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \tag{5.3}$$

Based on multiple data sets of static measurements, the measurement sensitivity covariance matrix R would contain non-zero elements on its diagonal as well as outside of its diagonal due to the previously mentioned dependency of measurements. However, these influences were shown to be negligible (covariance a factor 100 lower). Due to the relatively small influence on performance and high computational cost, the covariance matrix R for the first Kalman filter is finally computed as:

$$R = \begin{bmatrix} \sigma_{g1}^2 & 0 & 0 & 0 \\ 0 & \sigma_{g2}^2 & 0 & 0 \\ 0 & 0 & \sigma_{g3}^2 & 0 \\ 0 & 0 & 0 & \sigma_{g4}^2 \end{bmatrix} \tag{5.4}$$

5.2.2 Second Kalman filter covariance matrices

Process noise covariance Q

The second Kalman filter assumes the same constant acceleration model as apparent in the first Kalman filter. This allows for a similar approach to the process noise covariance matrix Q.

$$Q = \begin{bmatrix} \frac{T_s^4}{4} & \frac{T_s^3}{2} & \frac{T_s^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{T_s^3}{2} & T_s^2 & T_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{T_s^2}{2} & T_s & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{T_s^4}{4} & \frac{T_s^3}{2} & \frac{T_s^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{T_s^3}{2} & T_s^2 & T_s & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{T_s^2}{2} & T_s & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & T_s^2 & T_s & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & T_s & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & T_s^2 & T_s \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & T_s & 1 \end{bmatrix} \tag{5.5}$$

Measurement sensitivity covariance R

For estimation of the state $x = [x\ \dot{x}\ \ddot{x}\ y\ \dot{y}\ \ddot{y}\ \dot{x}_r\ \ddot{x}_r\ \dot{y}_r\ \ddot{y}_r]^T$, measurements from both encoders as well as IMUs are used, as seen in Equation 4.18. This causes problems when trying to assign covariances to both of these sources.

As mentioned in Equation 5.2.1 the covariance of noisy IMU or gyroscopic measurements can be characterized by the noise distribution during static measurements. However encoders only contain glitches (which are filtered out in the measurement data pre-processing as described in section 5.1) but no noise. If the covariance were to be determined computationally an extensive tire-slip model would have to be created that relates actual lateral movement to the measured lateral movement, regards the error as noise, and calculates the covariance of this noise. Creating this extensive tire-slip model can be avoided by first assigning static covariances between both wheels and acceleration in x and y and then creating a slip detection method that takes the static wheel velocity covariance as initial no-slip covariance and inflates when slip is detected.

The covariance for all the left- and right wheels are generally expected to be equal. Slip requires a different and individually assigned covariance since wheels that are subjected to slip should be neglected by the Kalman filter. An example of the wheel slip is shown in Figure 5.11 which shows slip occurring on one of the left wheels during acceleration as well as during deceleration giving the indication that it was caused by uneven weight distribution on the left- and right wheel.

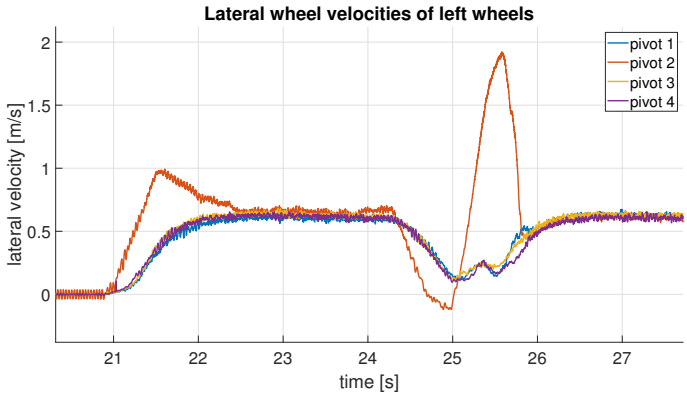


Figure 5.11: Excessive slip on the left wheel of the second pivot

This slip can be detected in a multitude of ways:

1. Using the integration of the IMU acceleration measurements in the x-direction and the gyroscopic rotational velocity, compute an additional estimate of the lateral wheel velocities and compare this to the measured lateral wheel velocities on that specific pivot. A significant difference in prediction and measurements indicates slip.

2. Integrate the Jupiter sensor (more in chapter 6) which provides accurate platform velocities. Using the pivot angle these can be translated to expected wheel velocities. Outliers again can be said to undergo slip.

3. Average the wheel velocities of either the set of left wheels or the set of right wheels. Outliers of this average velocity can be said to undergo slip.

4. Use the computed residual of the wheel velocities from from the Kalman filter equations. A significant residual indicates slip.

Although an attempt was made at the first option, integrating the acceleration data proved useless as sudden jumps in acceleration caused shifts in velocity. The second option would be great in future applications but was not possible due to lack of a complete integration of the Jupiter sensor into Simulink. This option would also require an additional slip detection since the Jupiter, if it were to be used by the RoPod project, might get turned off when the RoPod is on a low power mode or it might not be used by the RoPod project at all. The third option was initially chosen where the covariance of a wheel encoder gets determined by the difference between its own wheel velocity and those side's wheel velocities average.

This method has severe limits on its ability to detect slip when it occurs on multiple left or right wheels (i.e. if two wheels out of four are subjected to similar slip, which two wheels should the algorithm trust more) and is less robust for slip detection when the RoPod is making turns where none of the wheel velocities are supposedly equal. It might even be possible that individual wheel velocities all differ a lot causing the algorithm to assign slip to wheels that are not actually subjected to slip.

The fourth option was therefore chosen as final slip detection method. Although using the residual to determine the covariance of the measurements, which is already calculated in the second Kalman filter, seems counter-intuitive to the workings of the Kalman filter (i.e. a large residual indicates a bad *a priori* estimate so this residual should have more impact on the *a posteriori* estimate and not the other way around) there is some nuance which makes this method valid. First of all it is important that the wheel velocity measurements do not

solely or even predominantly determine the state estimate. This would result in the counter-intuitive case where one tries to estimate accuracy of a measurement based on the measurement and the state that is determined by that very same measurement. Secondly, the covariance should only 'blow up' if the residual is unexpectedly high with respect to residuals after sudden movement (sharp angles) which should certainly be taken into account by the algorithm. The first requirement for this method is met by the alteration of the measurement data of the IMU accelerations in subsection 5.1.2. The second requirement was met by analyzing measurement data of sudden movements and determining when the covariance should blow up when slip occurs and when it should remain low when there is a significant state transition taking place that was not taken into account by the *a priori* estimate. It was determined that residuals exceeding 0.15 m/s should be considered as artifacts of slip. The covariance for the wheels is computed as

$$R_{dyn,l,i} = R_{stat} + |\frac{20}{3}(\dot{\phi}_{l,i} - \dot{\hat{\phi}}_{l,i})|^4 \tag{5.6}$$

The relative covariances between the x- and the y acceleration were experimentally determined by driving with a constant velocity on a straight line. Analyzing static measurements is here not sufficient due to the extreme pivot wiggle and ground inconsistencies which have detrimental effect on the measured signal of the sensor. Note however that the noise originating from static IMU measurements is here added with measurement inconsistencies and is not purely noise anymore. It was however noted that the Kalman filter was not able to properly calculate and predict these inconsistencies based on the current state (as would be the nature of the Kalman filter). Incorporating these inconsistencies was therefore seen as a necessary approach.

Together, the relative covariances for one pivot point has the following form

$$R_{wheel\,i} = \begin{bmatrix} R_{dyn,l,i}F_w & 0 & 0 & 0 \\ 0 & R_{dyn,r,i}F_w & 0 & 0 \\ 0 & 0 & 1F_i & 0 \\ 0 & 0 & 0 & 2F_i \end{bmatrix} \tag{5.7}$$

Where F_w and F_i are the scaling factors that will be used in the optimization for the wheels and the IMU data respectively. Since the platform has four pivot points, the entire covariance matrix R looks as follows

$$R_{global} = R_{local} = \begin{bmatrix} R_{wheel\,1} & zeros[4\times4] & zeros[4\times4] & zeros[4\times4] \\ zeros[4\times4] & R_{wheel\,2} & zeros[4\times4] & zeros[4\times4] \\ zeros[4\times4] & zeros[4\times4] & R_{wheel\,3} & zeros[4\times4] \\ zeros[4\times4] & zeros[4\times4] & zeros[4\times4] & R_{wheel\,4} \end{bmatrix} \tag{5.8}$$

$$R = \begin{bmatrix} R_{global} & zeros[16\times16] \\ zeros[16\times16] & R_{local} \end{bmatrix} \tag{5.9}$$

The scaling factors F_w and F_i were obtained by assigning a quality to an estimation with certain F_w and F_i and comparing this to an estimation with different scaling values. A value for the quality of the estimation was obtained by usage of a high accuracy electro-optical sensor (the Accerion Jupiter, more information in chapter 6) which can, on certain floor types be regarded as ground truth. The quality is then computed as

$$P = \sum_{i=1}^{N-1} \frac{\sqrt{(x_{ju_i} - x_{kf_i})^2 + (y_{ju_i} - y_{kf_i})^2}}{d_{kf} + \sqrt{(x_{kf_{i+1}} - x_{kf_i})^2 + (y_{kf_{i+1}} - y_{kf_i})^2}} \tag{5.10}$$

Which is therefore the absolute positioning difference between the Jupiter sensor (subscript ju) and the Kalman filter (subscript kf) divided by the traveled distance of the Kalman filter estimation. This performance factor P is minimized in the optimization for driving on short carpet (the robot soccer field). The method could be done for multiple surfaces resulting in covariance matrices that are not only variable by slip detection and nature of the movement but also by the surface the RoPod is driving on. In the feature map that is used for localization, properties of the floor type could be included which will vary the covariance matrices when necessary.

5.2.3 Improved orientation Kalman filter covariance matrices

Process noise covariance Q

Since the improved orientation Kalman filter only consists of an *update* stage, its process noise covariance Q is that of the first Kalman filter algorithm shown in Equation 5.2.

Measurement sensitivity covariance R

The improved orientation Kalman filter uses all measurements that are described in subsection 5.2.1 and subsection 5.2.2. This results in covariances in the first and second Kalman filter that are already computed but not yet related to one another, i.e. the first Kalman filter uses gyroscopes having ideal covariances relative to each other. The second Kalman filter uses wheel encoders and IMU accelerations having ideal covariances relative to each other. The covariance between measurements from the first- and second Kalman filter are not yet related to each other).

Combining Equation 5.4 and Equation 5.7 and allowing for relative differences between these covariances from different Kalman filters results in the following expression of the covariance for all the measurement data obtained from a single pivot point

$$R_{wheeli} = \begin{bmatrix} \sigma^2_{gz,i} F_{kf1} & 0 & 0 & 0 & 0 \\ 0 & R_{dyn,l,i} F_w F_{kf2} & 0 & 0 & 0 \\ 0 & 0 & R_{dyn,l,i} F_w F_{kf2} & 0 & 0 \\ 0 & 0 & 0 & 1F_i F_{kf2} & 0 \\ 0 & 0 & 0 & 0 & 2F_i F_{kf2} \end{bmatrix} \tag{5.11}$$

Where F_w and F_i are scaling factors that are used in constructing the relative covariance between wheel encoders and IMU data in the second Kalman filter (obtained from its optimization) and F_{kf1} and F_{kf2} are scaling factors that are used in constructing the relative covariance between the gyroscopes and the wheel encoders and IMU data in the first- and second Kalman filter respectively. Iteratively checking the difference in absolute position for paths with identical starting and finishing locations allows to minimize this distance using these scaling factors.

6. Accerion Jupiter Sensor

The company Accerion has developed an indoor positioning sensor capable of determining the position of a mobile robot with respect to its environment. This Jupiter sensor, which is shown in Figure 6.1 is an electro-optical system which applies two different concepts to obtain positional information of a robot. One is relative positioning similar to wheel encoder or IMU position accumulation. Its working is similar to that of an optical mouse. Although the accuracy is high (an absolute positioning error of 0.1% to 1% best- and worst case scenario respectively) it is still susceptible to drift [13]. The other concept is absolute positioning. The robot is placed on a designated piece of flooring, scans/maps this part of the floor and is able to recognize it when the sensor moves above these areas. When an area is recognized, it is used as an absolute reference point to correct the accumulated drift from the relative positioning.

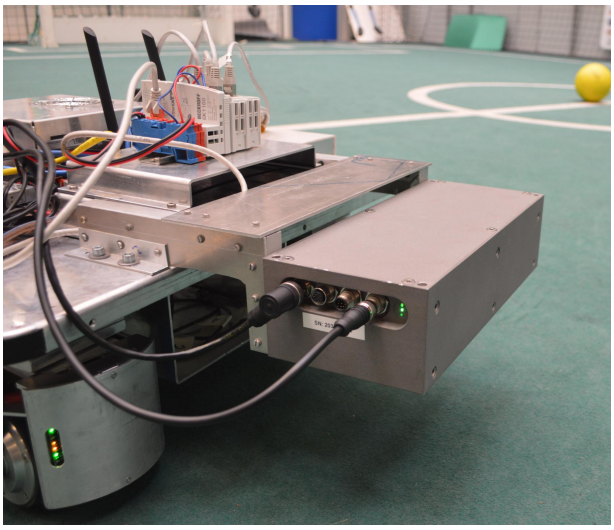


Figure 6.1: Accerion Jupiter sensor (V1)

Using multiple tactically placed mapped areas in a static environment, the Accerion Jupiter sensor can manage the entire positioning by itself. The sensor could prove use full for the RoPod project in that it can determine odometry of the platform and position relative to its environment with impressive accuracy. The sensor is however designed for industrial purposes where robotic platforms drive on clean and flat concrete surfaces. The application of the RoPod robotic platform is not limited to concrete flooring. It is therefore essential that the sensor is tested on different types of flooring that occur in hospitals. This chapter introduces the measurement plan for the Accerion Jupiter and provides an introduction of its results. The subsequent chapter relate these results to results obtained by the Kalman filter algorithm.

6.1 Absolute positioning experiments

The goal of the RoPod project is to develop a platform which can assist in hospitals (the project's main focus point). To this end, accurate positioning is required on multiple different floor types. For experimentally determining the accuracy of the Kalman filter algorithm and the Accerion Jupiter sensor these floor types where chosen to be

- Concrete (cleaned and uncleaned)
- Rough surface tiling
- Linoleum
- Short pile carpet

Images of these different floor types and their location throughout Gemini are shown in Appendix C. On these different floor types, similar tests where, due to problems with the RoPod during the course of this project, the sensor was attached to a cart and the RoPod was placed on this cart to provide power to the sensor. The setup is shown in Appendix C. This cart was then pushed to drive two consecutive square shapes (rotating the platform 90 degrees at every corner) and finishing at the same location as where the experiments started. Although one can expect to have a certain error in positioning the cart at the exact starting location, the traversed path was chosen sufficiently large to account for this error. In Appendix C it can be seen that the wheels of the cart are significantly larger than the wheels of the RoPod and also have more suspension (thicker rubber layer) which enables the cart to drive much smoother than the RoPod. It is therefore important to note that the results of the experiments are only to be compared with one another and not with performance measures of the RoPod itself.

The experiments were conducted multiple times to create an average performance value for the Jupiter sensor. This performance value is computed as

$$\lambda = \frac{\text{Absolute positioning difference}}{\text{Distance travelled}} 100\% \tag{6.1}$$

Resulting in a similar absolute positioning error calculation that was provided by Accerion for concrete surfaces. Equation 6.1 also shows that the longer the measured path is, the smaller the influence of the final re-positioning error. The average path length is roughly 80 meters. Assuming a re-positioning error of 1 centimeter (a realistic estimate of this error) this results in an addition to the performance value of 0.000125 which, as can be seen in Table 6.1, is of negligible influence. The results of the experiments are summarized and shown in the table below

Surface	**Performance value**
Concrete (uncleaned)	0.4819 %
Concrete (cleaned)	0.6387 %
Rough tiling	0.5423 %
Linoleum	1.2258 %
Short carpet	0.1624 %

Table 6.1: Table with average drift percentages of the Accerion Jupiter (v1) on different floor types

From this table, the following conclusions about the accuracy of the Accerion Jupiter sensor on different floor types can be made;

- Uncleaned concrete provides a better surface for the sensor compared to uncleaned concrete. This is likely due to dust and other particles that give more relief for the sensor to detect, allowing to estimate the position and velocity more accurately.
- Rough tiling with considerable grooves shows impressive performance, indicating that slight differences in sensor mounting height (and therefore vertical sensor vibrations) are of relatively small effect on the accuracy of the sensor.
- Linoleum, an often-used flooring type in hospitals and other professional settings shows, at least to other surfaces, significant drift.
- Short carpet (which is an extremely rough surface) shows the best performance which is likely due to the significant relief that the individual hairs of the carpet provide. When applying the absolute positioning method to this surface, it might not be able to detect surfaces due to changes that occur when the surface is being driven over (hairs of the carpet changing orientation).

Due to the extend of the decrease in accuracy on linoleum, tests were conducted which tried to find an answer to this difference. Sensor installation height proved to not have an influence as long as it was installed within the permissible range (install at 150 mm +/- 20 mm) [13]. It was however discovered that, not unsurprisingly, the velocity of the platform had influence on the accuracy of the sensor. The surprising part was the extend to which this velocity plays a role in the sensor's accuracy. A table of drift percentages for different velocities on linoleum is shown below.

Average velocity	**Performance value**
0.7459 m/s	35.0462%
0.5021 m/s	2.8121%
0.2981 m/s	1.2258%

Table 6.2: Table with average drift percentages of the Accerion Jupiter (v1) for different velocities on linoleum

Figure 6.2 shows the paths that were determined by the Accerion Jupiter sensor for the fast, intermediate, and slow movement of the testing platform. All of the paths in the figures had identical starting and finishing location. These figures clearly show the increase in accuracy when platform velocity decreases.

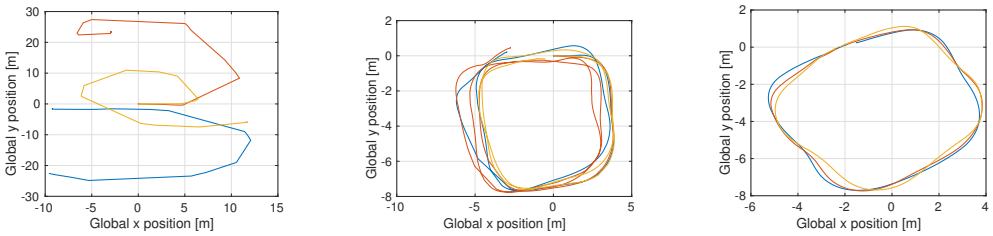


Figure 6.2: Global positioning estimate based on fast, intermediate, and slow movement

It should be noted that an average velocity of 0.7459 m/s is not unrealistic and could occur in many robotics applications. It should therefore be stated that accuracy of the Jupiter sensor might show significant drift at higher velocities. This decrease in accuracy was especially apparent on linoleum (which also happens to be the smoothest surface). This should be researched further.

In comparison, Figure 6.3 shows the measurements of the Jupiter on short carpet which is, for the experiment setup described in Appendix C, the most accurately mapped surface.

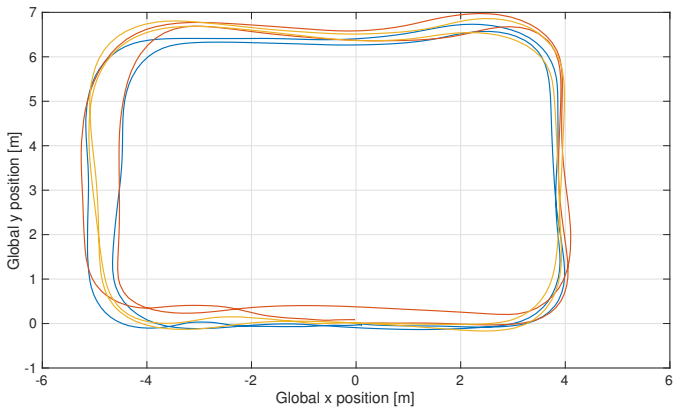


Figure 6.3: Global positioning estimate of the Jupiter sensor on short carpet

Overall, the Accerion Jupiter sensor shows great performance on concrete (for which the sensor is originally designed) but it prefers uncleaned- over cleaned which is not likely to occur in hospitals. Rough tiling shows results comparable to concrete although significant grooves were present which were suspected to decrease the accuracy of the position estimate. The soft tires of the cart on which the sensor was attached is able to soften these bumps resulting in less vibration in the sensor's installation height. Short carpet shows unexpectedly accurate results and linoleum, of which most of the flooring in hospitals consists, shows lackluster results indicating that to make the Jupiter sensor work in applications such as the RoPod project its processing algorithms should be made more robust to smoother floor types and higher velocities on these floor types.

7. Results

This chapter describes the performance of the two-and-a-half LKF framework compared to the performance of the existing encoder mapping and to that of the Accerion Jupiter sensor. Both the LKF framework- and encoder mapping localization and odometry estimates are obtained from offline post-processing of the data. Although the LKF framework has to be implemented in the RoPod software to compute odometry and localization in real-time, this was not tested due to lack of time. Instead, the post-processing run-times were measured indicating that, before the code is compiled to *C*, which runs more efficiently, the computation takes 98% of real-time. This indicates that when the code is running more efficiently in *C*, the implementation is able to convey the same results. The Jupiter sensor data is obtained from the ROS node in Simulink and directly plotted and compared without any post-processing. Similarly to chapter 6 these methods have been tested on multiple different surfaces. The entire framework both predicts global positioning and odometry. The performance of both these estimations will be highlighted and compared. Since the Accerion Jupiter sensor has already been tested by itself in chapter 6 where performance values are obtained for different floor types, the performance of the two-and-a-half LKF framework and of the existing encoder mapping can, on certain floor-types be compared to the near ground truth estimation of the Accerion Jupiter sensor. These floor types are limited to; concrete, rough tiling, and short carpet, with short carpet clearly having the best accuracy.

Throughout this chapter, the following color conventions are used:

- Blue: Old estimate from direct encoder mapping
- Orange: New estimate from two-and-a-half LKF framework
- Yellow: Accerion Jupiter estimate

To keep consistency within this chapter, a specific use-case is chosen which will be plotted unless specifically stated otherwise. This use-case consists of a rough driven rectangle on short carpet. During this run, excessive slip occurred on one of the wheels causing the encoder mapping to drift extensively. Since the LKF framework is designed to account for- and be robust to this slip, this run is of great interest. The Jupiter sensor has proven itself on short carpet in chapter 6 resulting in an excellent reference path. The rectangular consists, due to the current encoder mapping control strategy, of bended paths on the supposedly straight parts. This requires small steering corrections which will also occur in real applications. This use-case is however only to be consistent in figures. Tables are given where average performance values are noted. These performance values are calculated from multiple representative and similar runs.

7.1 Comparison of orientation estimation

Just as in chapter 6, the orientation has been rigorously tested on both concrete, rough surface tiling, linoleum, and short carpet. In chapter 1 the problem statement was defined whereby it is desired that the RoPod is able to drive 5 meters, make a sharp 90 degree turn, and drive another 5 meters entirely on its local navigation method for all these surfaces. The RoPod's precise starting and finishing location are therefore known. For assigning a performance value to the orientation estimation, this method would prove to be less robust (i.e. a 90 degree turn is made so small differences between starting and finishing orientation are of a relatively large influence). It was therefore decided to also drive a single large size rectangular on all surfaces. Hereby, the starting and finishing orientation are identical and a 360 degree turn has been completed resulting in lesser influence of differences in starting and finishing orientation.

Results of the first run (whereby excessive slip occurs on one of the wheels) is shown below. Notice that the encoder mapping, due the slip, reports unrealistic orientations. Both the LKF framework and the Jupiter show great results.

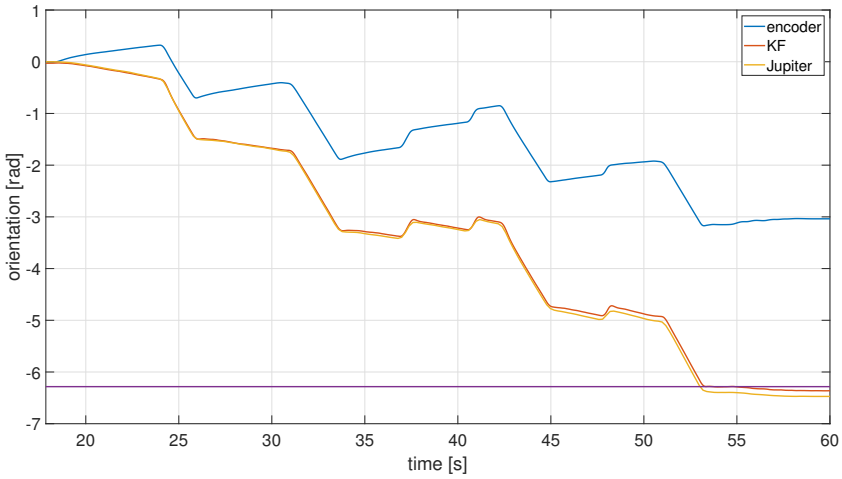


Figure 7.1: Estimation of orientation for a driven rectangular on short carpet

In this figure the purple line indicates -2π which is the orientation of the RoPod at its finishing location. Also notice that the LKF framework and the Jupiter show only small divergence with a distance driven of roughly 30 meters. Since the Jupiter sensor showed near ground-truth accuracy on short carpet, as shown in chapter 6, it can be assumed that the difference between the estimate of the Jupiter sensor and the -2π line is due to differences in beginning and finishing orientation and not due to inaccuracies of the Jupiter sensor. It is therefore not necessarily the case that the Kalman filter framework provides better results than the Jupiter sensor. During the experiments it was noted what the difference in orientation and location was with respect to the expected finishing location. This difference is taken into account in the analysis of the experiments.

Calculating the performance of the different methods for the different floor types results in the following table of drift percentages:

Surface	**Encoder**	**KF**	**Jupiter**
Concrete (uncleaned)	0.2176%	0.0592%	0.0368%
Rough tiling	0.1551%	0.0147%	0.0196%
Linoleum	0.3095%	0.0364%	0.0189%
Short carpet	0.2449%	0.0127%	0.0257%
Average	0.2318%	0.0308%	0.0253%

Table 7.1: Table with average drift percentages for the orientation on different floor types

The drift percentages in this table are computed by

$$\lambda = \frac{\textit{Absolute difference starting and finishing location}}{\textit{Distance travelled}} 100\% \tag{7.1}$$

This table shows that the encoder mapping is far less accurate compared to the other methods caused by excessive drift due to slip, as also shown in Figure 7.1 making it unsuitable for orientation estimation. Both the LKF framework and the Jupiter sensor show similarly accurate results for all floor types. This indicates that the LKF framework greatly outperforms the encoder mapping and is on par with the Jupiter sensor. On average, the Jupiter sensor still outperforms the LKF framework. This is even the case on linoleum which in chapter 6 showed lackluster results. The sudden increase in accuracy on linoleum has multiple reasons:

- The surface on which the experiment was conducted was less littered with dust and other particles during the experiments of chapter 6, causing the sensor to be able to map the surface less accurately.
- The difference in height caused by the difference in measurement setup, while not proving important for other surfaces, caused inaccuracies in the measurements of chapter 6.
- The average velocity of the experiments conducted by the RoPod are lower causing an increase in accuracy as seen in Table 6.2.

As will be shown in subsequent chapters, the accuracy of the Jupiter sensor and LKF framework lie close to each other for every surface.

7.1.1 Conclusion

The orientation estimate of the two-and-a-half LKF framework outperforms the current encoder mapping method and is on par with the near ground-truth estimate of the Jupiter sensor. This can be attributed to i) the slip detection method apparent in the framework and ii) the fusion of not only encoder data but also accurate gyroscope and less accurate accelerometer data. Finally, it can be said that given the time scale in Figure 7.1, the orientation of the RoPod is sufficiently estimated to navigate '*worst case scenarios*'. It is now up to the global positioning- and local RoPod velocity estimate to further substantiate this claim.

7.2 Comparison of localization estimation

An accurate estimation of global localization is important when feature detection is not available or is insufficient. This global localization is directly related to the previously stated problem statement. Analyzing the experiments for 5 meter, 90 degrees, 5 meter paths allows to validate this statement and see whether or not the RoPod, with the two-and-a-half LKF framework, is able to navigate on its own to the extend that would be required in a '*worst case scenario*'. As mentioned for consistency in this chapter, the use-case (where excessive slip occurs on one of the wheels) is shown below. Similarly to Figure 7.1, the encoder mapping drifts significantly while the global positioning estimates of the Kalman filter and the Jupiter sensor remain close to one-another. Since the driven path has the same starting and finishing location it is clear that both of these estimates outperform the encoder mapping.

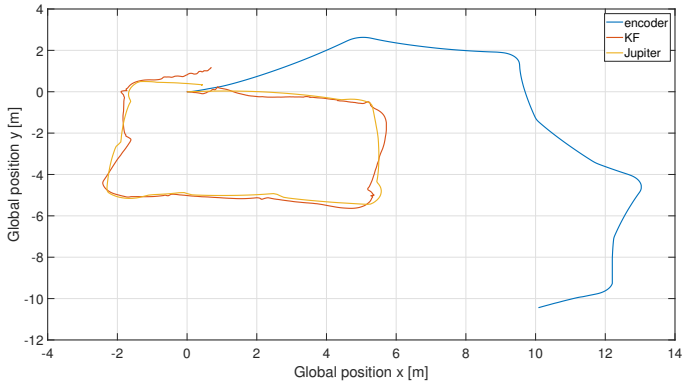


Figure 7.2: Estimation of positioning for a driven rectangular on short carpet

This figure and Figure 7.3 also indicate that compared to both the encoder mapping and the Jupiter sensor, the global positioning estimate of the LKF framework is more susceptible to small steering corrections (which occur at the end of the driven path to align the final locations with the starting location). The reason for these inaccuracies is expected to be the linearity assumption that is apparently not permissible at high angular velocity of the pivots.

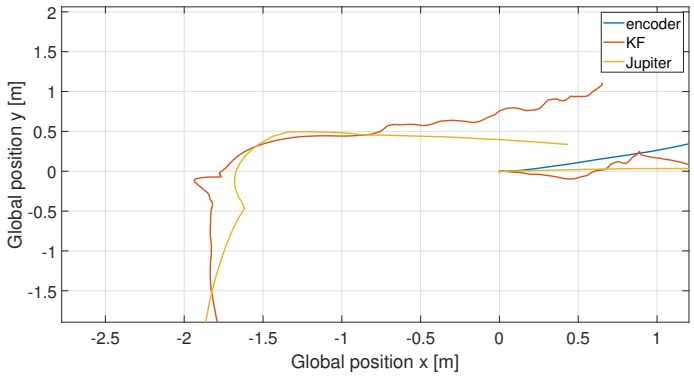


Figure 7.3: Estimation of positioning for a driven rectangular on short carpet zoomed in on the last corner with respect to Figure 7.2

A measurement of a path on tiling, where similarly to the path of Figure 7.2 and Figure 7.3 a rectangular has been driven with identical start- and finishing location is shown below:

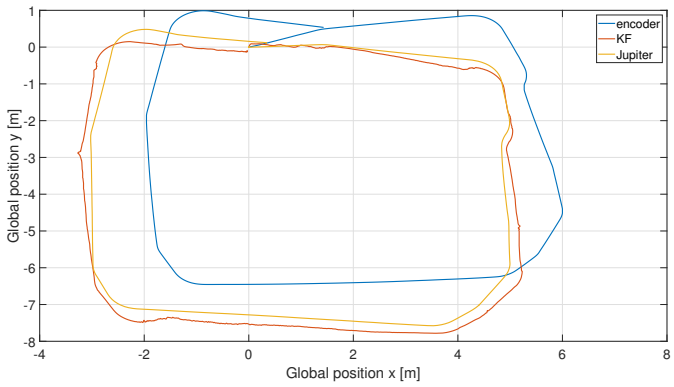


Figure 7.4: Estimation of positioning for a driven rectangular on tiling

Notice that this figure shows increased accuracy of the algorithm. The driven path also contained less wheel slip compared to the path driven on short carpet which not only results in a more accurate positioning estimate of the encoder mapping but also of a more accurate positioning estimate of the LKF framework indicating that the current detection and covariance scaling for wheel slip has room for improvement.

Calculating the performance of the different methods for the different floor types results in the following table of drift percentages:

Surface	**Encoder**	**KF**	**Jupiter**
Concrete (uncleaned)	13.8548%	3.1470%	1.5467%
Rough tiling	10.8289%	1.2751%	2.5367%
Linoleum	11.5042%	3.5036%	1.1850%
Short carpet	25.7484%	2.1890%	1.3327%
Average	15.4841%	2.5287%	1.6503%

Table 7.2: Table with average drift percentages for the positioning estimation on different floor types

From this table it can be concluded that both the two-and-a-half LKF framework and the Jupiter sensor vastly outperform the encoder mapping on all surfaces. More interesting are the differences between the LKF framework and the Jupiter sensor itself. Firstly it can be noted that both methods still have considerable errors and do not actually meet the requirement of 1% maximum error which was set in chapter 1. The Algorithm has difficulties with uncleaned concrete, likely due to side-wards wheel slip which might occur due to high acceleration of the pivot angles onto slippery surfaces such as dust and sand (which is apparent on the uncleaned concrete). Unexpectedly however, the LKF framework almost meets the requirement of 1% on rough tiling. This is unexpected due to the deep grooves which were expected to negatively affect the IMU data (unevenly driving over a groove even momentarily affects the gyroscope measurements). Linoleum is for the LKF framework the least accurate surface. This is likely related to the fact that it is also the smoothest surface. This could cause excessive wheel slip on more than one or two wheels which, since the residual method is used for slip detection, could negatively affect the state estimation. Also, since the weight of

the RoPod is not equally distributed, the smoothest surface has the highest chance of sideways wheel slip.

As for the Jupiter, its performance is consistently lower than its presumed worst-case scenario performance given by Accerion. This might have multiple reasons;

- The RoPod is too much affected by bumps and other inconsistencies causing the sensor to be subjected to continuous differences in height during its path.
- The RoPod being too much affected by bumps and other inconsistencies could also lead to minuscule accelerations that were not apparent in Accerion's testing situation.
- The acceleration profile of the RoPod is inefficient for the sensor and not comparable to Accerion's testing situation.

It should however be stated that the testing situation at which the experiments in this report were conducted are representative for the future application of the RoPod. Furthermore, relating the results of the experiments in this chapter to the results of chapter 6 appears to show contradicting information. It appears to be that the experimental setup (installation height, velocity profile, cart suspension) plays a significant role in the accuracy of the sensor. The results in chapter 6 can be considered as a best-case scenario compared to the results in this chapter. The RoPod appears to limit the capabilities of the Jupiter sensor.

7.2.1 Conclusion

The accuracy of global positioning is vastly improved by the usage of the LKF framework with respect to the usage of the encoder mapping. Drift is taken care of in the algorithm but when too much drift is appearing on a significant amount of wheels, it has artifacts in the state estimation. Although the Jupiter sensor outperforms the LKF framework on all surfaces except rough tiling it is still subjected to considerable drift and performs worse than the drift values given by Accerion (0.1% to 1%). Both global positioning estimation methods do not meet the 1% drift requirement set in chapter 1 so to properly navigate its environment in the described worst-case scenario, additional sensors or improvement of the LKF framework is required. An excellent addition could be an IMU on platform level. This will be elaborated in chapter 9.

7.3 Comparison of odometry estimation

To let the RoPod follow its predetermined or real-time computed path more accurately, the local velocities of the RoPod platform are required. These velocities are used in the path planning where a local x velocity $\dot{x}_r$, a local y velocity $\dot{y}_r$, and a rotational velocity are required. As stated in chapter 4, these properties are incorporated into the state of the second LKF. The local velocity $\dot{x}_r$ for the previously mentioned use-case is shown below.

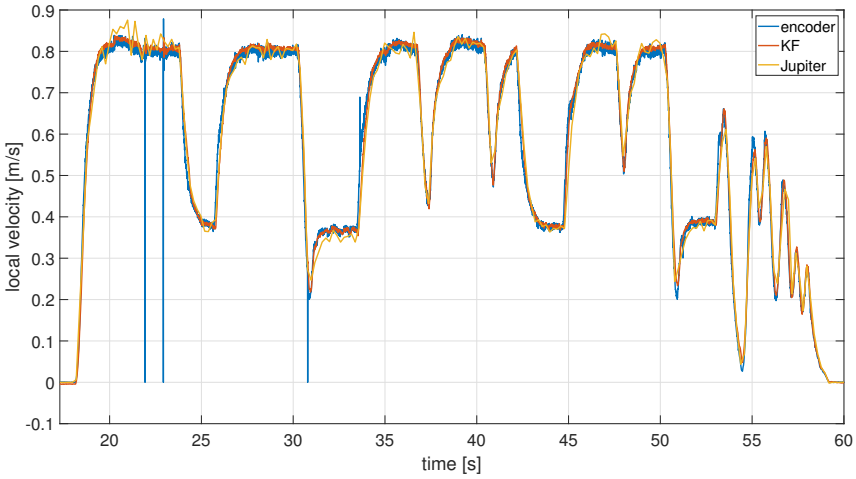


Figure 7.5: Estimation of local velocity for a driven rectangular on short carpet

At first sight the estimation according to the encoder mapping, the LKF framework, and the Jupiter sensor show identical results (other than glitch removal by the LKF framework). This is counter intuitive due to the excessive slip that occurs on the wheels. One would expect to see artifacts of this drift in the local platform velocities. Only when a part of the path is highlighted do the differences in platform velocity become more clear. This is shown in Figure 7.6 where the part from 53 seconds until 59 seconds is expanded.

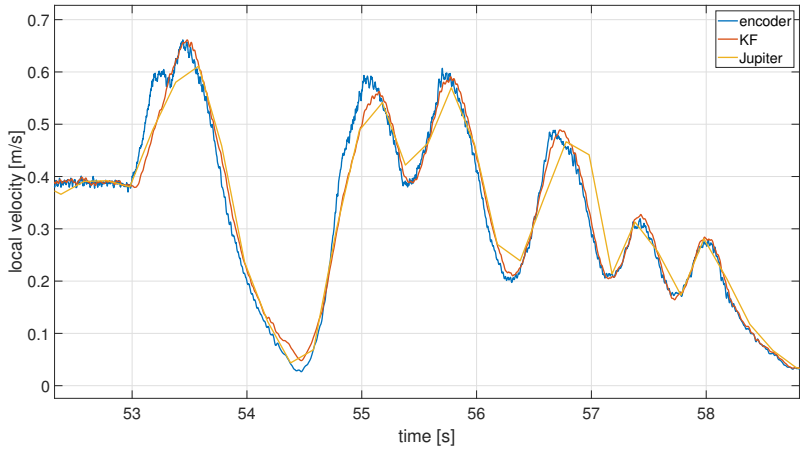


Figure 7.6: Estimation of local velocity for a driven rectangular on short carpet

This figure shows that, generally speaking, the LKF framework follows the Jupiter sensor more accurately compared to the encoder mapping. Especially the bump in local velocity which occurs in the encoder mapping at roughly 53 seconds is neither visible in the LKF framework or the Jupiter's local velocity. The flattened tops and bottoms from the Jupiter sensor occur due to the significantly lower sample time of 0.2 seconds as opposed to 0.001 seconds from the encoder mapping and LKF framework. Nevertheless, the LKF framework shows a better estimate of the local velocity of the platform indicating that incorporation of the LKF framework allows the RoPod to follow its predetermined of real-time computed path more accurately. This increased performance of odometry estimation can, due to the low sample time of the Jupiter and the requirement of manually aligning the Jupiter data, not be quantized. However analysis of the data for different floor types consistently show better performance by the LKF framework as opposed to the encoder mapping as also shown below

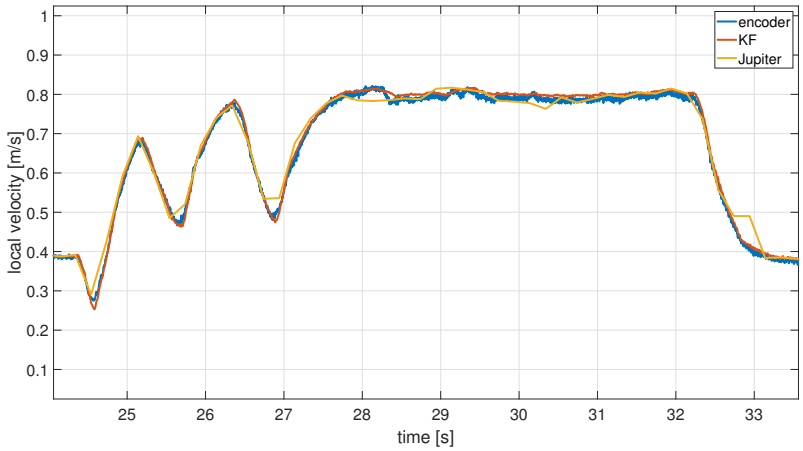


Figure 7.7: Estimation of local velocity for a driven rectangular on linoleum

Notice that although differences in estimation on linoleum are smaller than on short carpet, both Figure 7.6 and Figure 7.7 show that the LKF framework is able to smoothen the estimation of the odometry. This is also an important property of the Kalman filter which helps in creating a more robust approach to trajectory tracking.

8. Conclusion

This report has described the approach to odometry and localization estimation of the RoPod. It describes the algorithm's background theory (chapter 2), its architecture (chapter 4), its sensor and process identification (chapter 5), its implementation (Appendix B) and its performance compared to the currently used encoder mapping and the Accerion Jupiter sensor (chapter 7).

The designed LKF framework is able to fuse multiple streams of measurements to improve estimation of the state of the RoPod. With respect to the encoder mapping which only uses wheel and pivot encoder velocities, the LKF framework uses

1. Wheel velocity measurements
2. Pivot angle and velocity measurements
3. Accelerometer measurements in the driven plane
4. Gyroscopic rotational velocity around the axis perpendicular to the driven plane

the LKF framework consists of an *predict* and *update* phase for estimating the orientation state $x_\theta = [\theta \ \dot{\theta} \ \ddot{\theta}]^T$, an *predict* and *update* phase for estimating the positional state $x_{xy} = [x \ \dot{x} \ \ddot{x} \ y\dot{y} \ \ddot{y} \ \dot{x}_r \ \ddot{x}_r \ \dot{y}_r \ \ddot{y}_r]^T$, and a third *update* phase for increasing the accuracy of the orientation state x_θ.

The LKF framework, the encoder mapping, and the Jupiter sensor have been tested on different surfaces. On average, the LKF framework has an error for orientation of 0.031% compared to an error of 0.232% and 0.025% for the encoder mapping and Jupiter sensor respectively. The LKF framework has an error for global positioning of 2.529% compared to an error of 15.484% and 1.650% for the encoder mapping and Jupiter sensor respectively.

It can be concluded from these results and the results in chapter 7 that while the LKF framework outperforms the encoder mapping, it is not yet on-par with the Accerion Jupiter sensor. The LKF framework is however capable of global positioning based solely on local data (no feature detection) in a '*worst case scenario*' described in chapter 1 indicating that the two-and-a-half LKF framework can be implemented in the RoPod software and is able to achieve desirable performance.

Similarly in chapter 7 it has been shown that, compared to the encoder mapping, the LKF framework is able to provide an improved local RoPod platform velocity. This conclusion is drawn based on the assumption that the velocity data from the Jupiter can be regarded as a ground truth. Analyzing the performance of the Jupiter sensor on different floor types, this is a valid assumption. It is therefore that the LKF framework, after implementation, is able to provide better path following capabilities.

9. Discussion

Although the two-and-a-half LKF framework is able to achieve desirable performance, there are some improvements that could be made that could soften the unrealistic influences of sharp steering on the prediction of positioning, could improve the estimation on either the global positioning or the local RoPod velocities, or could make the current framework more robust.

1) IMU on platform level:
Currently the RoPod has 4 IMUs which are located on pivot level. This, together with the fact that pivot angles and velocities are regarded as ground truth, means that unmodeled inaccuracies occur in these IMU measurements (especially the acceleration in the y-axis). Adding an extra IMU would increase IMU costs by only 25% but will be expected to improve the predict of the orientation and the local RoPod accelerations much more accurately. It contains a direct measurement of $\dot{\theta}$ and measurements of $\ddot{x}_r$ and $\ddot{y}_r$ that are only affected by the improved estimate of the angular velocity and acceleration $\dot{\theta}$ and $\ddot{\theta}$.

2) Improve kinematic parameters:
The kinematic parameters of the RoPod which are currently used are; pivot offset with respect to the centre of the RoPod, wheel offset with respect to the pivot centre, IMU offset with respect to the pivot centre and wheel radius. Although these parameters were provided, some doubt has been cast on the accuracy of these parameters during the completion of this project. It is expected that, when a new iteration of the RoPod has been constructed that the accuracy of the kinematic parameters could be improved resulting in improvement of the accuracy of the LKF framework.

3) Improve IMU pre-processing:
Currently the IMU sensor data pre-processing only contains bias compensation and axis realignment. Some doubts were cast (by Ir. W. Houtman and A. Ketels, designer of the SmartWheels) on the nature of the excessive noise apparent in the accelerometer. Although more elaborate IMU realignment methods exist, these are mostly reserved for three dimensional situations (e.g. drones) where a complementary filter could be used. Some more research into IMU realignment could provide useful information and allows better usage of this data which is in other similar applications far more useful.

4) Fix the PCB:
Currently the PCB on which the IMU is located is connected parallel to the wheel axis in a manner similar to a cantilever. This results in considerable vibrations in the x-direction. Although this eigenfrequency peak has been filtered in the data pre-processing, it would be desirable to limit the influence of the vibration altogether by better fixing the PCB to the SmartWheel. An example could be to not only fix it at the top (the cantilever side) but also at the bottom near the axis of the wheels.

5) Research non-linearity in the model:
As mentioned in chapter 2 an assumption of linearity on the time-scale at hand has been made. This assumption has been made to allow the two-and-a-half LKF framework. It was however found out that the pivot velocities (which are causing the non-linearity) may rotate and accelerate rather fast, even on the time-scale at hand. It is therefore an interesting research subject if usage of a single EKF would be more accurate and computationally efficient or at least obtain a similarly accurate estimation using less computation power. Since the LKF framework works best on straight movements but shows inaccuracies during corner taking, the linearity assumption is thought to be a bottleneck of the accuracy the global positioning estimation. It should be noted though that by definition the EKF is an approximation of the solution while the LKF is the exact solution (in theory).

6) Improve suspension of the RoPod:
The RoPod currently consists of non-symmetric suspension. On the left side of the RoPod the suspension consists of a rotating cantilever fixed in all DoFs but the rotation itself at the centre of the RoPod. On the right side of the RoPod the suspension consists of a beam fixed in all DoFs. This results in an uneven weight and load distribution on the pivots. This uneven distribution has its effect on the accuracy of the LKF framework. If the suspension where to be made symmetric and the weight distribution in general could be improved, the LKF framework is expected to achieve better positioning and odometry estimates.

Implementation of these recommendations could increase the robustness and accuracy of the global positioning and odometry estimation.

Bibliography

[1] Borenstein J., Everett H., Feng L.,& Wehe D. (1997), *Mobile Robot Positioning Sensors and Techniques*, Journal of Robotic Systems, Special Issue on Mobile Robotics, 14(4), pp 231-249. 2

[2] Welch G.,& Bishop G. (2006), *An introduction to the Kalman Filter*, In Practice Journal, 7(1), pp 1-16. 3, 4

[3] Thrun S.,& Burgard W. (2000), *Probabilistic Robotics*. 3

[4] Awasthi V., & Raj K. (2011), *A Comparison of Kalman Filter and Extended Kalman Filter in State Estimation*, International Journal of Electronics Engineering, 3(1), pp 67-71. 5

[5] Sabatini A. (2011), *Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation*, Sensors, 11(10), pp 9182-9206. 6

[6] Abbeel P., Coates A., Montemerlo M., Ng A.Y.,& Thrun S. (2005), *Discriminative Training of Kalman Filters*, Conference: Robotics: Science and Systems I, Stanford University, Department of Computer Sciences. 6

[7] Rajamani M. (2007), *Data-based Techniques to Improve State Estimation in Model Predictive Control*, pp 235. 6

[8] Thrun S., Burgard W., & Fox D., *Lecture slides: Probabilistic Robotics Bayes Filter Implementations*.

[9] Fox D., Burgard W., Kruppa H.,& Thrun S. (1999), *Collaborative Multi-Robot Localization*, Proc. of the German Conference on Artificial Intelligence, (KI), pp 12.

[10] Hol J. (2011), *Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS*, Linköping studies in science and technology. Dissertations, 1368, pp 165.

[11] Kon J. (2018), *Planar pose and velocity estimation of a soccer robot*, Bachelor end project, Eindhoven University of Technology, Department of Mechanical Engineering. 9, 24

[12] Bosch (2014), *BMI055 Small, versatile 6DoF sensor module Data sheet* 20

[13] Accerion (2018), *Jupiter installation and operation manual* 29, 31, 51

[14] Accerion (2018), *Accerion - Customer Downloads* (password protected) 48

A. Equations

A.1 Wheel encoder measurement mapping

Mapping of the current state to the expected measurements of the lateral wheel velocity begin by considering the matrix transformations from global level towards wheel level as shown in chapter 3.

$$M_{WL}^{P} = T_{I}^{P} = \begin{bmatrix} 1 & 0 & -s_w \\ 0 & 1 & \frac{d_w}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

$$M_{P}^{R} = T_{P}^{R} R_{P}^{R} = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{(\delta)} & s_{(\delta)} & 0 \\ -s_{(\delta)} & c_{(\delta)} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

$$M_{R}^{G} = T_{R}^{G} R_{R}^{G} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{(\theta)} & s_{(\theta)} & 0 \\ -s_{(\theta)} & c_{(\theta)} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

Where s_w, d_w, x_r, and y_r follow from the conventions described in Chapter 3. x and y describe the 2D location of the robot frame with respect to the global frame. Subscript WL indicates the left wheel of a pivot point. Multiplying these translation matrices in specific order results in an expression that describes the x and y coordinates of wheel 1 in the global frame:

$$p_{WL}^{G} = \begin{bmatrix} x - s_w c_{(\delta+\theta)} - \frac{d_w}{2} s_{(\delta+\theta)} + x_r c_{(\theta)} - y_r s_{(\theta)} \\ y + \frac{d_w}{2} c_{(\delta+\theta)} - s_w s_{(\delta+\theta)} + y_r c_{(\theta)} + x_r s_{(\theta)} \end{bmatrix} \tag{A.4}$$

Rearranging these equations shows us the relationships with respect to the rotational matrices:

$$\begin{bmatrix} x_{WL}^{G} \\ y_{WL}^{G} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c_{(\delta+\theta)} & s_{(\delta+\theta)} \\ -s_{(\delta+\theta)} & c_{(\delta+\theta)} \end{bmatrix} \begin{bmatrix} -s_w \\ \frac{d_w}{2} \end{bmatrix} + \begin{bmatrix} c_{(\theta)} & s_{(\theta)} \\ -s_{(\theta)} & c_{(\theta)} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix} \tag{A.5}$$

Differentiating once with respect to time gives us the velocity of the center of the wheel in the global frame:

$$\begin{bmatrix} \dot{x}_{WL}^{G} \\ \dot{y}_{WL}^{G} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + S(\dot{\delta+\theta}) R_{(\delta+\theta)} \begin{bmatrix} -s_w \\ \frac{d_w}{2} \end{bmatrix} + S(\dot{\theta}) R_{(\theta)} \begin{bmatrix} x_r \\ y_r \end{bmatrix} \tag{A.6}$$

Mapping this to the frame of the center of the wheel gives us the expected velocity of the centre of the wheel as function of the state (again, assuming δ is ground truth):

$$\begin{bmatrix} \dot{x}_{WL}^{WL} \\ \dot{y}_{WL}^{WL} \end{bmatrix} = R_{G}^{WL} \begin{bmatrix} \dot{x}_{WL}^{G} \\ \dot{y}_{WL}^{G} \end{bmatrix} = \begin{bmatrix} -c_{(\delta+\theta)} & -s_{(\delta+\theta)} \\ s_{(\delta+\theta)} & -c_{(\delta+\theta)} \end{bmatrix} \begin{bmatrix} \dot{x}_{WL}^{G} \\ \dot{y}_{WL}^{G} \end{bmatrix} \tag{A.7}$$

Indicating the x- and y directional velocities of the wheel frame as expressed in the wheel frame. Obtaining the same equations for the second wheel (WR) only requires that the transition matrix T_{WL}^{P} in Equation A.1 changes its sign of d_w. Note however that the wheel encoder

of the left- and the right wheel are rotated 180 degrees but that the frame of of the left- and right wheel is not as can be seen in Figure 3.2. To accommodate for this, the sign of the measurements should be reversed. This can be done in the pre-processing stage of the filter. Using the radius of each wheel, the rotational velocity that is obtained from the encoder measurements can be translated to translational velocity.

To compute the measurement mapping of the local velocity and local acceleration state, one should set the angle θ to zero. The resulting $\dot{x}$ and $\dot{y}$ are then automatically $\dot{x}_r$ and $\dot{y}_r$. Both the measurement mapping of global state and the measurement mapping of local state is implemented in the computations of the two-and-a-half phase LKF framework. This increases computational costs but is required for proper operation of the algorithm.

A.2 Data pre-processing

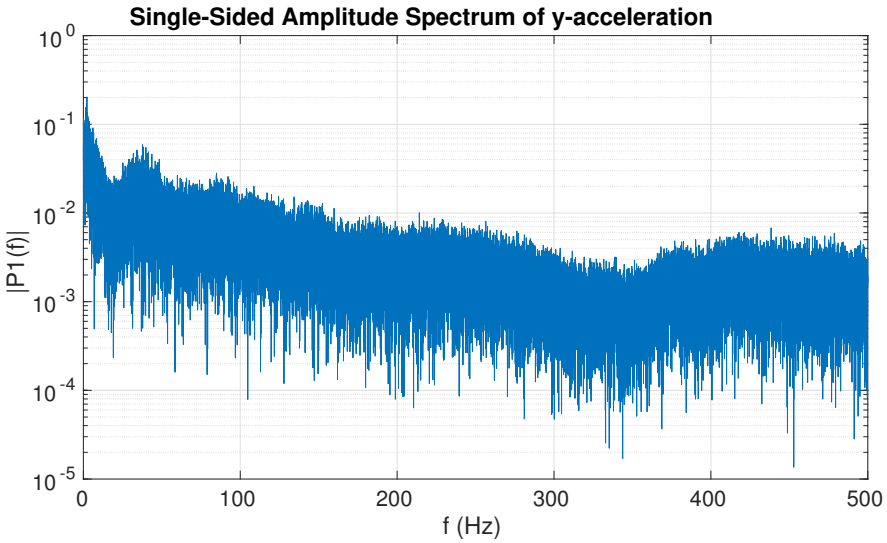


Figure A.1: Single sided frequency density y acceleration

Figure A.1 shows a less significant peak at 40 Hz.

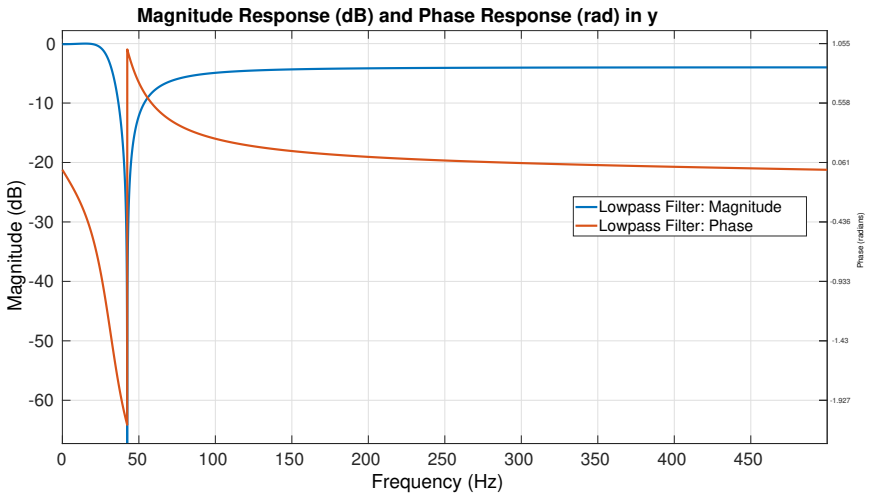


Figure A.2: IIR magnitude and phase response y

and 40 Hz according to the peaks in the frequency density of the x- and y-measurements as shown in Figure 5.6 and Figure A.1 respectively. Figure A.3 shows the frequency density of the y-acceleration after it has been filtered by the IIR low-pass filter shown in Figure A.2

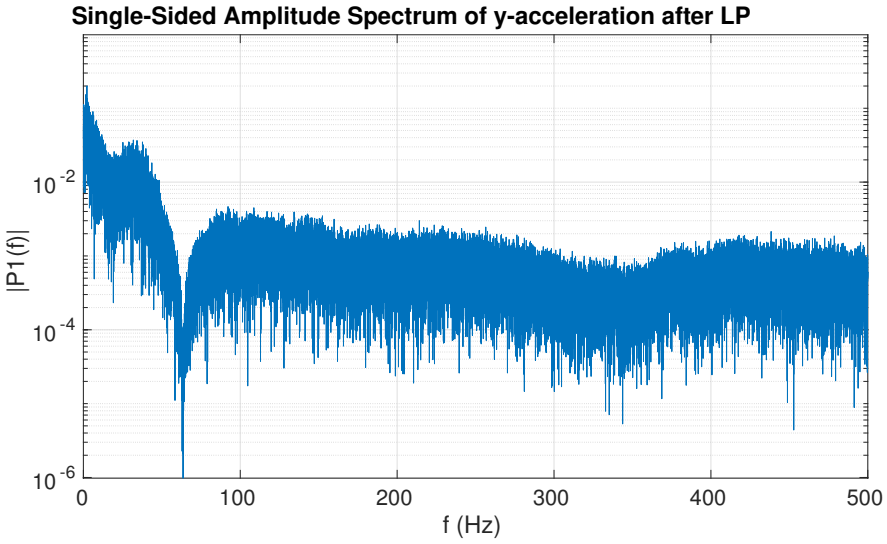


Figure A.3: Single sided frequency density y after the IIR LPF

B. Implementation

The algorithm has been implemented in a hybrid of MATLAB and Simulink. Since the algorithm has only been applied to the data in post-processing of data (the algorithm has not been run in real-time on the RoPod), showing results requires first that the RoPod drives around. The first part will elaborate on the method of obtaining measurements.

To obtain measurements of both the RoPod's sensors and the Accerion Jupiter sensor, the following steps should be taken;

1. Install the ROS node from Accerion on your Ubuntu 16.04 installation (this could later be done on the RoPod's computer but is currently done on the laptop) [14].
2. Connect the Jupiter sensor to the RoPod using the aluminum structure
3. Turn on the RoPod
4. Connect to the RoPod using *ssh* and enter the password
5. On the laptop, open the Simulink file '*running_jupiter*'
6. Plug the ethernet cable from the Jupiter into the laptop and enter the following commands in a terminal on the laptop:
    - In catkin workspace: '*cd devel*' followed by '. *./setup.bash*'
    - In devel folder: '*cd* .. followed by '*catkin_make*'
    - In catkin workspace: '*roslaunch Jupiter.jupiter_node*
7. Open the Accerion Jupiter configuration GUI [14]
8. Reset the position of the Jupiter sensor in the second tab of the GUI
9. Run the Simulink file, which uses a ROS node to communicate with the Jupiter sensor
10. Run an executable on the RoPod (either with a pre-programmed path or with controller path mapping)

To then pre-process the measurement data of both the RoPod's sensors and the Accerion Jupiter, the all the obtained *.mat* files should be copied to the folder with the Simulink model *datarun.slx*. This includes the following required files:

- *all_wheel_data_out.mat*: Data from the RoPod's sensor. Arrangement of this data can be seen in '*ec_structdata.txt*'
- *p_p_o_w.mat*: Orientation of the Jupiter sensor (θ)
- *p_p_p_x.mat*: Global x position of the Jupiter sensor (x)
- *p_p_p_y.mat*: Global y position of the Jupiter sensor (y)
- *t_t_a_z.mat*: Rotational velocity of the Jupiter sensor ($\dot{\theta}$)
- *t_t_l_x.mat*: Global x velocity of the Jupiter sensor ($\dot{x}$)
- *t_t_l_y.mat*: Global y velocity of the Jupiter sensor ($\dot{y}$)

After running the MATLAB file *datarun.slx* multiple *.mat* files are created which contain the results of the data post-processing. Often, some manual realignment regarding the Jupiter sensor is required. To visualize the results one can run *plotting_final_results.m* which creates figures of local velocity in x, global positioning and angle θ for the encoder mapping, Kalman filter, and the Jupiter sensor.

C. Elaborate results

C.1 Floor types

All the experiments have been conducted on four different floor types. Namely, short haired carpet (the Robot soccer field), Concrete (Gemini-North -1 around the elevator shaft), Linoleum (Gemini-North 1 around the elevator shaft), and Rough surface tiling (Gemini-North 0 around the elevator shaft). Images of these floor types to give and indication of what the surface and measurement area looks like are shown below.

Figure C.1: Short haired carpet (L) and concrete (R)

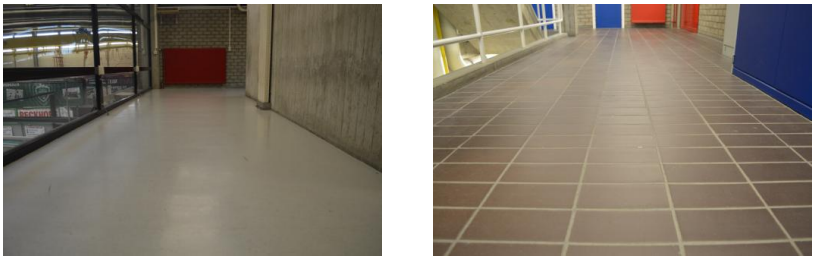


Figure C.2: Linoleum (L) and rough surface tiling (R)

Although the orientation and global localization measurements have been conducted by attaching the Jupiter sensor to the RoPod, the results from chapter 6 are obtained from a different measurement setup. Due to the consistent breaking down of the RoPod (problems occurred with installation of pivot points, multiple pivots were replaced during the course of this project), measurements for the accuracy of the Jupiter sensor on different floor types was initially done by attaching the sensor to the blue cart shown below. This has its influence on the accuracy of the results and the relationship that can be drawn from results of attaching the sensor to the RoPod and results of attaching the sensor to the blue cart. It was however ensured that the installation height for the sensor was in both cases within the marge specified by Accerion [13].

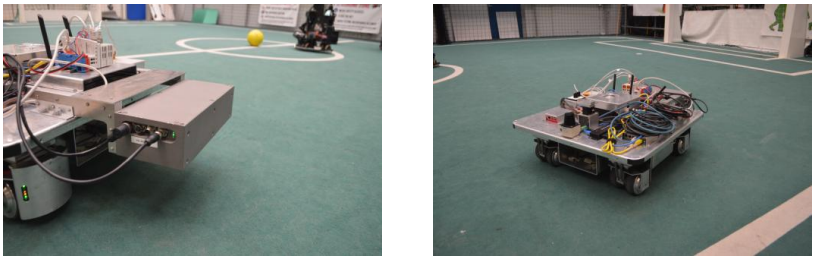


Figure C.3: Experiment setup for orientation and positioning experiments

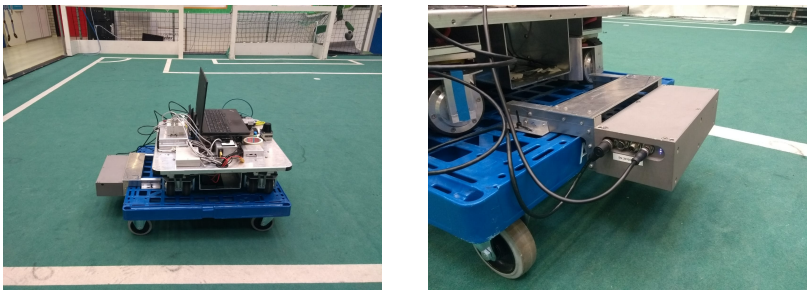


Figure C.4: Experiment setup for Accerion Jupiter accuracy in chapter 6

C.2 Orientation accuracy

This section describes results of the orientation estimate of the encoder mapping, the Kalman filter framework, and the Accerion Jupiter sensor. *Angle driven* indicates the supposed angle that should be measured. The first three columns show the measured angle. The second three columns show the percentile drift in orientation.

This is done for different floor types.

Carpet	Angle driven	Offset	Encoder	KF	Jupiter
	$\pi/2$	0	1.6425	1.542	1.564
	$\pi/2$	0	2.513	1.588	1.68
	$\pi/2$	0	1.6014	1.573	1.596
	$\pi/2$	0	1.7061	1.588	1.599
	2	0	3.038	6.37	6.472
	2π	0	7.55	6.4149	6.386

Encoder	KF	Jupiter
0.0456	0.01833	0.00433
0.5998	0.01095	0.06952
0.0195	0.0014	0.01605
0.0861	0.01095	0.01796
0.5165	0.01382	0.03005
0.2016	0.02097	0.01636

Concrete	Angle driven	Offset	Encoder	KF	Jupiter
	$\pi/2$	0	1.782	1.481	1.496
	$\pi/2$	0	1.639	1.465	1.473
	$\pi/2$	0.03491	0.5484	1.4518	1.516
	$\pi/2$	0	0.9284	1.3784	1.502
	$\pi/2$	0	1.7685	1.4884	1.6
	2π	0	5.8398	6.7982	6.503
	2π	0	7.164	6.2144	6.411
	2π	0	7.3304	6.316	6.486

Encoder	KF	Jupiter
0.1345	0.05717	0.04762
0.0434	0.06735	0.06226
0.6509	0.07576	0.03488
0.4090	0.12252	0.0438
0.1259	0.05244	0.01859
0.0706	0.08197	0.03498
0.1402	0.01095	0.02034
0.1667	0.00522	0.03228

Linoleum	Angle driven	Offset	Encoder	KF	Jupiter
	$\pi/2$	0	1.5286	1.4795	1.553
	$\pi/2$	0	1.5875	1.5847	1.469
	$\pi/2$	0	0.2724	1.7019	1.577
	$\pi/2$	0	0.3712	1.628	1.563
	2π	0	6.9782	6.3787	6.366
	2π	0	5.5364	6.1812	6.189

Encoder	KF	Jupiter
0.0269	0.0581	0.01133
0.0106	0.00885	0.06481
0.8266	0.08346	0.00395
0.7637	0.03642	0.00496
0.1106	0.0152	0.01318
0.1189	0.01623	0.01499

Tiling	Angle driven	Offset	Encoder	KF	Jupiter
	$\pi/2$	0	1.562	1.5723	1.598
	$\pi/2$	0	2.4112	1.5226	1.534
	$\pi/2$	0	1.5787	1.536	1.553
	2π	0	6.753	6.251	6.448

Encoder	KF	Jupiter
0.0056	0.00096	0.01732
0.5350	0.03068	0.02343
0.0050	0.02215	0.01133
0.0748	0.00512	0.02623

C.3 Global positioning accuracy

This section describes results of the global positioning estimate of the encoder mapping, the Kalman filter framework, and the Accerion Jupiter sensor. *w* or *q* indicates whether a whole rectangular was driven or only a quarter (as by the requirement from chapter 1) which is taken into account when computing the average. The three columns indicate the performance values of the encoder mapping, Kalman filter framework and the Accerion Jupiter sensor respectively.

This is done for different floor types.

Carpet	Path	Encoder	KF	Jupiter
	w	12.2354	0.4226	1.0952
	w	9.7198	3.3514	1.0471
	w	55.2901	2.793	1.8559
Average		25.7484	2.1890	1.3327

Concrete	Path	Encoder	KF	Jupiter
	w	7.0998	1.4144	1.1161
	w	12.7075	7.8364	1.693
	w	13.4687	0.6869	1.2514
	w	15.3883	0.9178	1.6958
Average		13.8548	3.1470	1.5467

Linoleum	Path	Encoder	KF	Jupiter
	w	8.2597	6.046	1.3229
	w	8.2597	1.521	1.3229
	w	17.9933	2.9439	0.9091
Average		11.5042	3.5036	1.1850

Tiles	Path	Encoder	KF	Jupiter
	q	26.3716	1.9897	7.3325
	q	17.5931	3.9055	3.0177
	w	5.2522	0.4389	1.2175
Average		10.8289	1.2751	2.5367