Mapping and Localization of a Rail-guided Robot

S.H. (Steven) Gies

BSc Report

Committee:
Dr.ir. M. Fumagalli
D.J. Borgerink, MSc
Dr.ir. J.F. Broenink

July 2015

019RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

CONTENTS

PREFACE

The following paper covers the report of a Bachelor's thesis that was carried out at the end of a study in Electrical Engineering at the University of Twente. The presented work took place at the chair of Robotics and Mechatronics (RaM), which takes part in the interregional project RoboShip: a maritime subproject of the INTERREG IVA project SmartBot in which multi-sensor robot platforms are developed for applications in various sectors. The RoboShip subproject aims at automation of the inspection of ballast water tanks in ships, a dangerous and unhealthy task currently being done by humans.

At the start of this thesis in April 2015 a prototype of the robot was already available, including sensing equipment and an inspection arm. One of the things that had to be worked on in order to improve autonomous behavior was the self-awareness of the robot's location in its working environment. Finding a solution to this problem formed the assignment of this thesis. After following a course on optimal estimation in dynamic systems and researching on how mapping and localization of similar robots is performed, an algorithm was created that uses a model of the robot's kinematics as well as control input data and sensor measurements to determine the robot's actual position in both 1D and 3D space.

Mapping and Localization of a Rail-guided Robot

S.H. Gies*, D.J. Borgerink[a,b]MSc

July 22, 2015

Abstract— **A rail-guided robot, part of the SmartBot-RoboShip project, will be used to inspect ballast water tanks in various ships, meaning that it should be capable of exploring different tank layouts. In this research project an algorithm based on the extended Kalman filter is developed for determining its position on a self-learned 1D map of the rail using a basic model of the robot's kinematics. This map is then extended by considering the rail in 3D space.**
Reference points used for both mapping and localization are formed by the detection of standard turns of 90 degrees as well as characteristic magnetic field peaks that are caused by the complex architecture of the metal tank walls. As the robot is led by a rail, it is incapable of steering. However, by taking into account the standardized rail specifications while processing odometry data and IMU measurements, the robot is well able to estimate its system parameters. The resulting algorithm reconstructs a 3D map of the rail in which the robot can localize itself, and correct its coordinates using self-created landmarks whenever these are reobserved.

I. INTRODUCTION

The bottom sections of large ships usually contain compartments called ballast water tanks. By filling these tanks with sea water the ship's center of gravity is lowered, improving stability. The inside features a complex system of metal walls which strengthens the outer walls. As these walls are subject to damage and corrosion caused by the salty sea water, they have to be inspected frequently, and if needed, repaired. Currently, this work is still done by humans. Considering the unhealthy and dangerous working environment it would be a good alternative to automate this process in such a way that workers do not have to enter the tanks anymore. The SmartBot-Roboship project [1] aims at solving this problem.

During earlier research [2] a rail-guided robot, equipped with a long arm with inspection tools, was found to be well-suited for this application. However, in order to navigate autonomously in an unknown environment, the robot should still be able to determine its own location and environment during operation. Using this knowledge the robot can move to specific positions or coordinates based on user-defined commands. Also when combining the on-rail position with the relative position of the tip of the inspection arm, the corresponding location of each inspection measurement can be determined and registered. Furthermore, despite being guided by a rail, the robot still has to take care of obstacle avoidance due to its long arm and the complex tank walls. The extension towards mapping and localization in 3D space would create a basis for this problem.

Previously the localization of the robot has been tested by the detection of RFID tags which were attached to the rail. However, this additional complexity is undesired, especially when implementing very long rails. This assignment therefore investigates the possibility of mapping and localization of the robot using its kinematics and assuming the rail is in its bare form.

A. Goal

Aim of this assignment is the development, testing and verification of an algorithm that uses sensor data and user commands both to reconstruct a map of the rail in 3D space as well as to determine the current position of the robot on this map. This algorithm should furthermore be able to create identifiable reference points. After recognizing these points during later encounters the estimate of the robot's current position and the location of the reference point should be updated.

B. Outline

This paper is structured as follows. Section II sketches the situation that is being dealt with; relevant information and assumptions of both the robot and the rail are given, as well as the available sensors. Using this knowledge the kinematics of the robot have been modelled. In Section III the main outline of the algorithm that is used for localization and mapping is discussed. It starts with the design of a filter for fusing the modelled behavior with sensor data and user control inputs. Thereafter the principle that is used to detect and handle reference points will be explained. Section IV shows the performed tests on this algorithm and discusses its results. Final conclusions of the assignment are presented in Section V, and based on these, recommendations are given in Section VI.

II. MODELLING

Before starting the design of an algorithm the available instruments, operating environment and underlying physics of this problem are analyzed. This section discusses these topics one by one before heading towards the design and implementation.

*Corresponding author: s.h.gies@student.utwente.nl
[a]Robotics and Mechatronics, University of Twente.
[b]INCAS^3, Assen, the Netherlands.

A. Robot specifications

In Appendix A a top view of the robot on a straight rail segment is shown. It is driven by two pairs of opposing Faulhaber 2657W024CR motors in combination with Platine MCDC3003 PRS motion controllers. These accept either a target velocity in rpm or a target position in encoder increments as control input from the robot's main computer. Next to that, they are able to determine the actual velocities and positions of each individual wheel and return these as measurement data. For more accurate control, the motors are connected to the driving wheels via n:1 gearheads with $n = 23$. The wheel radii r were specified as 20 *mm*, but due to wearage and deformation (caused by the tension at which the wheels have been clamped onto the rail), the radii were later calibrated and estimated at 17.5 *mm* (further elaboration on this in the parameter estimations of Appendix B). As a result the velocity of each wheel is determined with a resolution of 1/n rpm or $2\pi r/(n{\cdot}60) \approx 80\ \mu$m/s. The incremental position values are read out using additional magnetic encoders with 1024 pulses per revolution and quadrature signal output, and are thus determined with a resolution of $2\pi r/(n{\cdot}1024{\cdot}4) \approx 1.2\ \mu$m per increment.

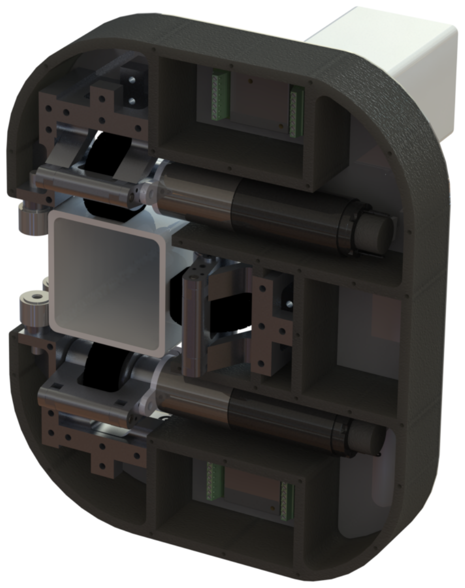


Fig. 1. Section of one of the driving modules clamped around a straight rail segment.

A CAD drawing of a section of one of the driving modules clamped around a hollow rail segment can be seen in Figure 1, showing its pair of actuated wheels (top and bottom). Additionally the module contains a similar sized wheel at one side (right), and smaller ones at the other side (left) with a gap inbetween for passing the anchor points that fix the rail to the tank walls. These have not been actuated.

In this assignment only the movements of the front driving module is considered. An Xsens MTI-300-2A5G4 inertial measurement unit (IMU) is mounted inbetween these two wheels at a height of 12 cm above the rail surface. This IMU is able to measure a large variety of quantities, in particular accelerations and gyrations. Table I shows an overview of the most important parameters with the sampling frequencies f_s at which the measurements have been logged. Since the IMU is attached to the top of the motor module its translational velocity will not always equal that of the wheels during a corner due to the difference in turning radius. However, the entire module and everything that is solidly attached to it will experience the same angular velocities. The correctly matching of translational and angular velocity measurements will prove to be an important aspect in the further derivation of other kinematic parameters. Therefore, whenever positions and coordinates are mentioned in this paper, the center of the rail underneath the IMU is defined as the center point of the robot, since then in any situation both the angular and translational velocity measurements correspond with those of the center point.

TABLE I
OVERVIEW OF THE MOST IMPORTANT AVAILABLE PARAMETERS.

Source	Parameter	Description	f_s (Hz)
Joystick	u	Control input, ranging from -2000 to 2000	∼30
Odometry	Pos	Actual position, available per wheel	∼2.5
	Vel	Actual velocity, available per wheel	∼2.5
Xsens	$Roll$ $Pitch$ Yaw	Euler angles (deg)	400
	a_x a_y a_z	Acceleration components (m/s^2)	400
	Gyr_x Gyr_y Gyr_z	Angular velocities (rad/s)	400
	Mag_x Mag_y Mag_z	Magnetic field components (arbitrary units ≈40uT)	100

B. Rail specifications

The rail that is to be mounted inside the ballast water tanks has a very basic structure and therefore it has several characteristics that are useful for this problem. The following assumptions have been made:

- The rail profile is square-formed everywhere with sides of 60 mm.
- The rail does not rotate around its central axis. As a consequence, both the rail and the robot do not 'roll'.
- Corners have a constant radius of 40 cm and are limited to turns of 90 degrees.

As a result of the previous assumptions, on any straight rail segment, orientation angles of both the rail and the robot are always multiples of 90 degrees. For a clearer impression a

3D view of an experimental tank including a rail is shown in Appendix D1.

C. Kinematics

Now, by using the velocity measurements and rail specifications, the kinematics can be modelled. Figure 2 shows the front wheels of the robot entering a turn of 90 degrees.

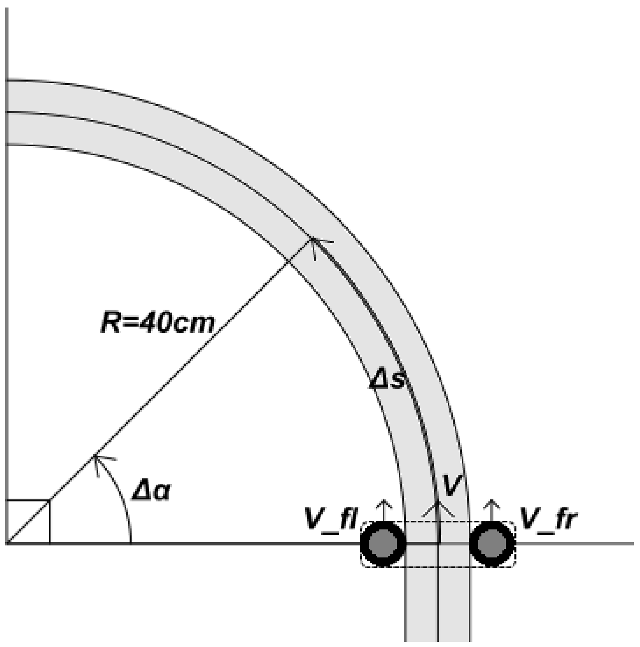


Fig. 2. The robot's kinematics during circular motion.

At any time, the on-rail velocity V is given by the average of the individual velocities V_{fl} and V_{fr} of the wheels. Once the robot enters the corner, it will follow a circular motion due the constant radius R of the rail. During such turn it will rotate around its center by $\Delta\alpha$ radians while travelling a distance of Δs meters, where $\Delta s = R\Delta\alpha$. The curvature κ of the rail can then be calculated as:

$$\kappa = \frac{1}{R} = \frac{\Delta\alpha}{\Delta s} = \frac{\omega}{V}$$

provided $V \neq 0$ m/s. This holds for both ω_y and ω_z, the angular velocities around the robot's y- and z-axis respectively.

One of the biggest problems in modelling the behavior of the robot is the fact that the robot cannot steer, in contrast to many other robots that make use of odometry. While regular odometry models assume that the robot's angular motion can be determined by driving its wheels at different velocities [3][4][5], in this situation the steering behaviour is fully determined by the rail and V. The robot itself can only drive forwards or backwards and therefore only one velocity control input will be used for all four wheels. Furthermore, determining angular velocity by using the difference in measured wheel velocities is difficult as well. This has two main reasons. First off, the difference in wheel velocities is very small. Assuming a constant $V = \omega \cdot R$ along the corner, the individual wheel velocities are given by

$$V_{fl} = \omega \cdot R_{fl}$$

and

$$V_{fr} = \omega \cdot R_{fr}$$

where $R_{fl} = 0.40 - 0.030 = 0.37$ m and $R_{fr} = 0.40 + 0.030 = 0.43$ m, the radii of the points where the wheels make contact with the rail. Since ω is equal for both wheels, the expected velocity of the right wheel is only $R_{fr}/R_{fl} \approx 1.16$ times that of the left wheel. Especially when considering the low speed of the robot this difference is easily dominated by measurement noise. Secondly, the rail can curve in four possible directions (vertically: up/down, and horizontally: left/right). This method of measuring rotation is only applicable to two out of four possible curvature directions; only those where the driving wheels are in the same plane as the curved rail as drawn in Figure 2.
Because of the reasons mentioned above, in this assignment the detection of rotations will purely rely on combining V with IMU measurements.

III. IMPLEMENTATION

Since the measurement data consists of discrete time samples that have to be fused immediately, a discrete state space model will be created in MATLAB. The next step will then be to actively filter incoming measurements in order to suppress noise, as localization and mapping heavily rely on these measurements. An extended Kalman filter will be used for this. Once clean and reliable states are available during operation the detection of and correction according to reference points can be implemented.

A. State space model

First, a state vector $\bar{x}$ is created as the algorithm's basis containing the most important system parameters:

$$\bar{x}(k) = \begin{bmatrix} D(k) \\ V(k) \\ x(k) \\ y(k) \\ z(k) \\ \theta(k) \\ \psi(k) \\ \omega_y(k) \\ \omega_z(k) \\ \kappa_h(k) \\ \kappa_v(k) \end{bmatrix} \tag{1}$$

with D in m the one-dimensional distance travelled along the rail and $V = \Delta D/\Delta t$ in m/s the on-rail velocity. x, y, z in m are the Eucledian world frame coordinates, θ, ψ in rad

the Euler angles Yaw and Pitch of the robot's orientation, ω_y, ω_z in rad/s the angular velocities around the robot's Y-axis and Z-axis, and κ_h, κ_v in m^{-1} the curvatures in the world's horizontal and vertical plane.
The Euler angle ϕ (Roll) and ω_x are not included as states as they were assumed constants. However, as the robot can be mounted onto the rail at different Roll angles, it is necessary to distinguish between the robot's coordinate frame and the world's fixed coordinate frame: Angles and angular velocities are measured with respect to the robot's coordinate frame whereas coordinates and curvatures are based on the world's fixed frame.

B. Parameter estimation

In order to determine accurate estimates of the robot's states, the measurements need to be filtered. This is done by a process of prediction and correction. At any discrete point in time k, the next system state $\bar{x}(k+1)$ can be predicted by means of a state transition function $f(\bar{x}(k),u)$. Using (1) gives:

$$\hat{x}(k+1) = f(\hat{x}(k),u(k)) + \bar{w}(k)$$

$$= \begin{bmatrix} D(k) + V(k)\cdot\Delta t \\ \frac{2\pi r}{23\cdot 60}\cdot u(k) \\ x(k) - V(k)\cos\psi(k)\sin\theta(k)\cdot\Delta t \\ y(k) + V(k)\cos\psi(k)\cos\theta(k)\cdot\Delta t \\ z(k) + V(k)\sin\psi(k)\cdot\Delta t \\ \theta(k) + \omega_y(k)\cdot\Delta t \\ \psi(k) + \omega_z\cdot\Delta t \\ V(k)\kappa_h(k) \\ V(k)\kappa_v(k) \\ \kappa_h(k) \\ \kappa_v(k) \end{bmatrix} + \bar{w}(k)$$

where $u(k)$ represents the velocity control input in rpm·23 as received by the motion controllers and $\bar{w}(k)$ is assumed to be zero mean white Gaussian process noise with covariance matrix $Q(k)$. The hat on $\hat{x}(k+1)$ indicates that this is an estimate of the real state vector. Δt is the time in seconds between discrete timesteps and is set to 2.5 ms; the sampling time of the Xsens' angular velocity measurements.

Calculation of the new coordinates $(x,y,z)(k+1)$ requires decomposition of $V(k)$ using the robot's orientation. In the case of the experimental tank, the robot is assumed to be mounted onto the rail at initial Euler angles $(Yaw_0, Pitch_0, Roll_0) = (\theta_0, \psi_0, \phi_0) = (0, 0, -\pi/2)$ radians and departing in the y-direction.

Since the rail consists of either straight or constant-curvature segments, the next curvature values are nearly always expected to equal their previous values, and so they are predicted as $\hat{\kappa}(k+1) = \hat{\kappa}(k)$. Conversions from $\kappa = 0\ \mathrm{m}^{-1}$ to $\kappa = 1/0.40 = 2.5\ \mathrm{m}^{-1}$ and vice versa will thus have to be found through measurements. Therefore a measurement function $h(\bar{x}(k))$ is introduced which returns the measurements $\bar{z}(k)$ of the motion controllers and the IMU:

$$\bar{z}(k) = \begin{bmatrix} Vel(k) \\ Gyr_y(k) \\ Gyr_z(k) \end{bmatrix} = h(\bar{x}(k)) + \bar{v}(k)$$

$$= \begin{bmatrix} \frac{23\cdot 60}{2\pi r}\cdot V(k) \\ -\omega_y(k) \\ \omega_z(k) \end{bmatrix} + \bar{v}(k)$$

where $Vel(k)$ is the average of the actual velocities of the front wheel pair as returned by the motion controllers and $Gyr_y(k)$ and $Gyr_z(k)$ are the gyration data in rad/s as measured by the IMU. $\bar{v}(k)$ is assumed to be zero mean white Gaussian measurement noise with covariance matrix $R(k)$. Note that the position measurements of the motion controllers are not included. The reason for this is that once a known reference point is observed and an error in estimated position is detected, the new position measurements will remain biased with this error.

For the larger part of the system, the propagation from $\bar{x}(k)$ to $\bar{x}(k+1)$ is a linear relation and thus a discrete Kalman filter [6] would form an option for online parameter estimation. However, due to the non-linear relationship between the Euler angles and the real world coordinates x,y,z an extended Kalman filter (EKF) is required. This filter estimates the system's states during each timestep by finding a weighted average of predictions and measurements while considering the modelled uncertainties that were caused by the presence of $\bar{w}(k)$ and $\bar{v}(k)$. As an addition to the regular Kalman filter, the EKF linearizes around an estimate of the current states' means and covariances. In Appendix C a flow chart is drawn showing the process of the EKF during each timestep.

Besides the use of an EKF some additional data processing steps are taken. Most important, in line with the rail characteristics discussed earlier, when following straight rail segments, the estimates of θ and ψ can be rounded towards the nearest multiple of $\pi/2$. In order to decide whether the robot finds itself in a corner or not the curvature estimates are compared to a threshold curvature $\kappa_{th} = 0.5\mathrm{m}^{-1}$ during each timestep. This rounding is also the reason why direct Euler angle measurements from the IMU are not included in $\bar{z}$; while Euler angle measurements from the IMU are prone to drifting over time, the pre-knowledge about possible angles should eliminate these errors. This step and the other additional processing steps are listed in Table II.

C. Estimation of noise factors

The covariance matrices $Q(k)$ and $R(k)$ form an important contribution to the correct working of the estimator. In this assignment, they have been modelled as time constant diagonal matrices with the squared standard deviations $\sigma_{x_1}^2 \ldots \sigma_{x_N}^2$ for each of the N states and $\sigma_{z_1}^2 \ldots \sigma_{z_M}^2$ for each of the M

measurements on the diagonals:

$$Q = \begin{pmatrix} \sigma_D^2 & 0 & \cdots & 0 \\ 0 & \sigma_V^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{\kappa_v}^2 \end{pmatrix}$$

$$R = \begin{pmatrix} \sigma_{Vel}^2 & 0 & 0 \\ 0 & \sigma_{Gyr_y}^2 & 0 \\ 0 & 0 & \sigma_{Gyr_z}^2 \end{pmatrix}$$

These standard deviations have been determined by either testing or tuning and are listed in the overview of estimated parameters in Appendix B.

D. Landmark detection

Besides mapping of the rail the robot has to be able to localize itself during operation. To do this, reference points in the form of landmarks along the rail can be created and remembered. This is done by saving the positions at which so-called 'corner edges' occur; the transitions from straight rail segments to corners or vice versa. During each timestep the last set of curvature estimates is analyzed in order to detect a transition. Analyzing multiple estimates prevents that false curvature spikes will be considered as real transitions.

Once a transition is detected the current state estimate will be compared with a matrix Λ of previously encountered landmarks. Each landmark λ_i is not only characterized by its 1D position, but also by its 3D coordinates, Euler angles and corner type. Finding a matching landmark within the range of 30 cm results in updating both the current state and the registered landmark whereas no match results in registering a new landmark by adding a new column to Λ.

Since the location of a newly detected landmark always has an error, the updating of the current state and registered landmark depends on the number of encounters. Each new landmark is initialized with a number of times encountered equal to 1. Everytime the landmark is reobserved, this weight is incremented. The updated position and coordinates of the robot and landmark are then calculated by the weighted average of the registered landmark and the new estimate (weight 1).

Another aspect of the environment that the robot could use as reference is the magnetic field, of which the components B_x, B_y and B_z are measured by the IMU. Due to the complex architecture of the tank walls and the passing through so-called manholes, a typical magnetic field pattern is expected to be measured along the rail. This is caused by the metal, which 'shapes' the magnetic field, especially close to the surface. Local minima and maxima in one or more of the magnetic field components could then be used as landmarks.

A flow chart of the algorithm cycle consisting of landmark detection and EKF steps is shown in Figure 3.

TABLE II

ADDITIONAL PROCESSING STEPS DURING EACH ALGORITHM CYCLE.

Code	Explanation
if $abs(Gyr_k) > 0.5$ then $Gyr_k = Gyr_{k-1}$	The measured angular velocity is exceptionally high or low and thus not reliable.
if $abs(Vel_k) > 1.1 \cdot MaxVel$ then $Vel_k = 0$	The measured velocity (of one of the wheels) exceeds the maximum input velocity by at least 10 percent; slipping has occurred.
if $abs(\hat{\kappa}_{h,k}) < \kappa_{th}$ then $\hat{\theta}_k = round(\hat{\theta}_k \cdot \frac{2}{\pi}) \cdot \frac{\pi}{2}$	The robot is not in a horizontal corner; Pitch is a multiple of $\pi/2$ radians.
if $abs(\hat{\kappa}_{v,k}) < \kappa_{th}$ then $\hat{\psi}_k = round(\hat{\psi}_k \cdot \frac{2}{\pi}) \cdot \frac{\pi}{2}$	The robot is not in a vertical corner; Yaw is a multiple of $\pi/2$ radians.
if $abs(u_k) < 0.2 \cdot MaxVel$ then $\hat{\kappa}_{h,k} = \hat{\kappa}_{h,k-10}$ $\hat{\kappa}_{v,k} = \hat{\kappa}_{v,k-10}$	The robot moves very slowly or stops; curvature estimates are unreliable.

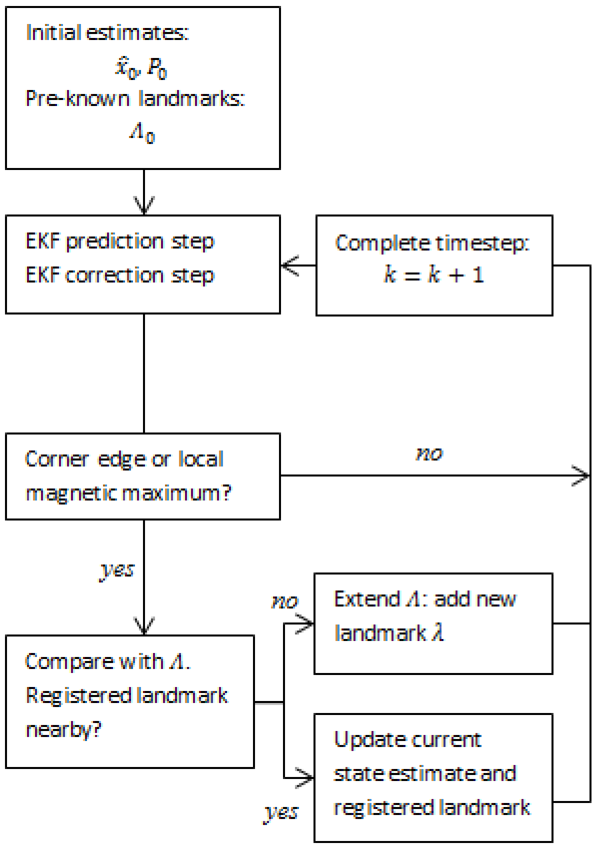

Initial estimates:
$\hat{x}_0, P_0$
Pre-known landmarks:
Λ_0
EKF prediction step
EKF correction step
Complete timestep:
$k = k + 1$
Corner edge or local magnetic maximum?
no
yes
Compare with Λ. Registered landmark nearby?
no
Extend Λ: add new landmark λ
yes
Update current state estimate and registered landmark

Fig. 3. Flow chart overview of the full algorithm's process.

IV. EXPERIMENTAL VALIDATION

Now that the algorithm has been created its correct working has to be verified. Being guided through a ballast tank by a rail that has the same standardized properties as discussed earlier the algorithm should accurately reconstruct the path of the robot. Also, landmarks should be detected and in case of multiple encounters a correction of position D and coordinates x, y, z should be made.

A. Test setup

The algorithm has been tested by using the measurements logged by the robot while travelling through the two-layer experimental tank. Logging of the odometry and joystick data is done by the robot's on-board computer while the IMU measurements are transmitted over a wireless network, and logged on an external computer using Xsens' *MTManager*. After logging, all data have been imported and synchronized in MATLAB.

The dimensions of the rail in this tank are known and both a top view and a 3D view of it are shown in Appendix D. The robot departs at a Roll angle $\phi = -\frac{\pi}{2}$. During the test the velocity control input u is set as high and constant as possible.

Four subexperiments have been carried out:

1) The robot is mounted onto the rail and moves forward through the tank until it reaches the end. During this experiment, the robot should be able to detect landmarks and register them.
2) At the end of the rail, the robot changes direction and travels back until it has reached the start of the rail again. On the way back, the robot should be able to recognize the landmarks that have been registered earlier, and update its own position accordingly. Also, since the estimates of the landmark locations of subexperiment 1 are not certain, the locations of the landmarks should be updated.
3) Subexperiment 1 is repeated, but now with pre-knowledge of real landmarks; the locations of the real corner edges are given as initial landmarks Λ_0 with weights of 10^{10}. Because of these large weights the algorithm should be able to update the robot's location to its true location whenever corner edges are detected. An extra requirement in this case is that there is no large drift in position estimates because then they will not match with the positions of the initial landmarks anymore, and the algorithm might confuse a corner edge with a different initial landmark.
4) For comparison, Subexperiment 2 is also repeated with pre-knowledge of real landmarks.

B. Results and Discussion

1) Path estimation: In Appendix E1, the raw measurements of Subexperiment 1 are shown. The vibrations caused by following the rail resulted in quite noisy angular velocity measurements, but the changes during corners are still visible. After applying the algorithm a path was reconstructed. A top view of the path that was estimated while travelling from start to end is shown in Figure 4. In Appendix E2 and E3, the other estimated paths for both cases, with and without pre-knowledge of real landmarks, are shown.

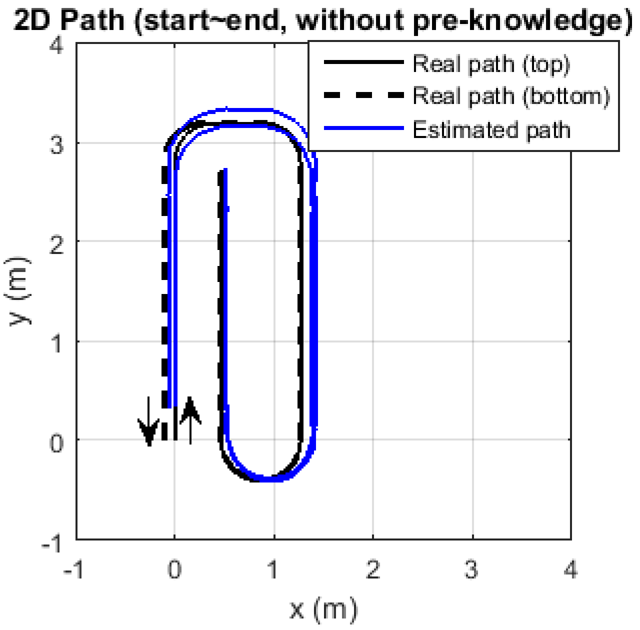


Fig. 4. Top view of the path estimated while travelling through the tank.

The estimated path of Figure 4 represents the same rail segments as in reality, but does have some spatial errors. Despite locations of corner edges being estimated quite accurately, the filter tends to underestimate the absolute curvature causing the estimated coordinates to drift off. This effect can be confirmed by zooming in on the curvature estimates of one of the horizontal corners, as shown in Figure 5. The underestimation is caused by the EKF which introduces a characteristic low-pass filter response to the sudden step in curvature from 0 m^{-1} to -2.5 m^{-1}. This problem can not be solved directly by rounding the curvature estimates to multiples of 2.5 though; switching back and forth between these values would then require the filter to heavily rely on the rather noisy gyroscope measurements, which will in turn cause false corner detections.

2) Landmark detection: Appendix E4 shows the landmarks that were detected by the algorithm during Subexperiment 1. Out of the 20 corner edges that exist on the path, the algorithm has detected 15 correctly. However, 4 of the missing edges can hardly be detected as they connect two consecutive turns with minimal spacing inbetween. Slight errors exist between these first estimates of the landmark positions and the real landmark locations. An overview of these spatial errors at corner edges is shown

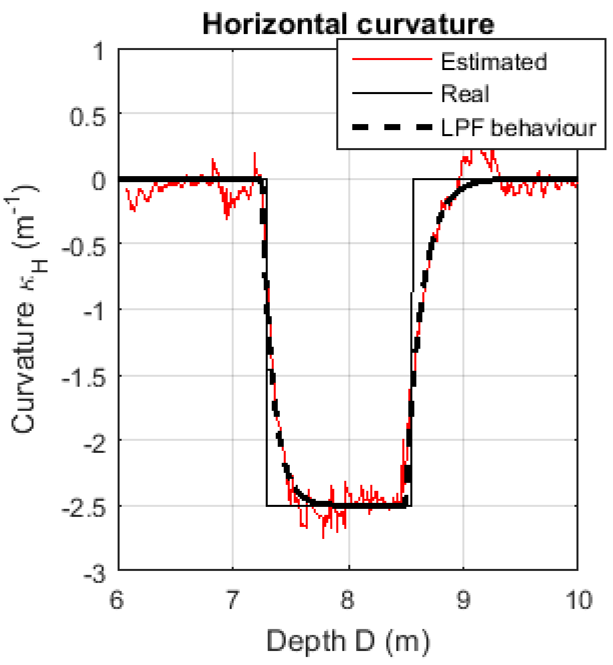


Fig. 5. Curvature estimates around one of the horizontal corners.

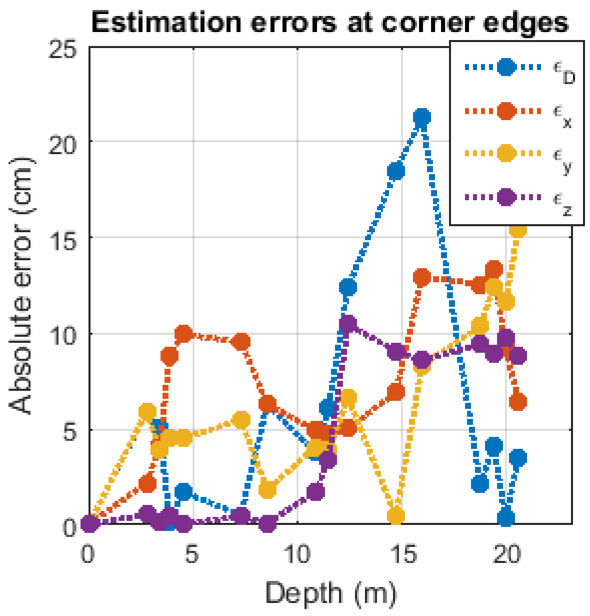


Fig. 6. Spatial errors at corner edges detected during Subexperiment 1.

in Figure 6. The mean landmark position error $\varepsilon_{D,mean}$ is found to be 6.11 cm, with a maximum of $\varepsilon_{D,max}$ = 21.27 cm. During another short test run (results not shown) 8 out of 11 corner edges were detected correcly, with $\varepsilon_{D,mean}$ = 5.47 cm and $\varepsilon_{D,max}$ 11.75 cm. As expected, as a result of drift the magnitude of the errors gradually increases over time.

On the way back, the algorithm correctly links reobserved corner edges to earlier observed ones. As can be observed in both the 2D and the 3D plots, during the double horizontal corner in the bottom layer the IMU got disconnected, causing the estimated path to drift off. However, due to the detection of the next vertical corners, the correct coordinates were restored.

The local maxima in the magnetic field data, shown in Appendix E4, were found using MATLAB's function *findpeaks*. This is a detection of peaks with respect to (discrete) time k. It turned out to be difficult to properly determine these peaks with respect to the distance since D is not monotonically increasing; the robot can move in both positive and negative direction.
Furthermore, most of the detected local maxima seem to appear during corners, where the angular movement of the robot results in the individual magnetic field components following a sinusoidal behaviour. Around these positions the robot already makes use of corner edges as landmarks. Finally, it has to be noted that the measured magnitudes of the field varied per experiment. Therefore the local maxima of the magnetic field were not registered as landmarks for localization purposes during this assignment.

V. CONCLUSIONS

The proposed algorithm is able to deal with the non-linearities of converting angles and velocities to positions in Cartesian coordinates. However, estimating the discrete-valued curvature turned out to be more troublesome as the combination of the highly non-linear process of switching between discretized states and the fact that the robot cannot predict its own angular motion well resulted in unrealistic corner estimates. The incorrect coordinates of the end of the corner in turn caused a drift in estimated coordinates for the rest of the estimated path.

The algorithm correctly detects most corner edges and is able to use these as reference points for later corrections of position and coordinates. Even when the estimated location of the robot drifts far off due to the estimation of an erronous corner, by observing landmarks and identifying them with previously learned ones the location can still be improved. Corner edges between consecutive turns are hard to detect however, especially when they are seperated with minimal spacing inbetween.

While travelling through the tank the expected characteristic magnetic field pattern was measured. Despite the uncertainty in the mean of the measurements, from the analysis of these patterns it can be concluded that linking local minima and maxima to specific positions could be used as an improvement of the localization process.

VI. RECOMMENDATIONS

One of the largest problem of the algorithm is the incorrect estimation of curvatures. For the sake of better estimation a different estimator that is able to deal with stronger non-linearities, such as a particle filter [6][7], is required. This could fix the underestimation of curvature estimates and as a result it could reduce drift in the estimated coordinates.

Besides the step towards a filter that is able to cope better with non-linearities than the EKF, several other improvements can be made to the algorithm. For example, the landmarks could be added to the state vector as is done with many vision-based localization and mapping algorithms [8]. This would enable the filter to also keep track of the landmarks' parameters by means of a covariance matrix, as well as the covariance between the landmark's parameters and the robot's state.

The positions are currently being estimated by integrating velocity data, which is prone to drift due to the low sampling frequency of the odometry measurements. Including the position measurements in the $\bar{z}$ would certainly allow for an increase of the accuracy of position estimates. This however requires keeping track and updating of the position error as found by reobserved landmarks, and then subtracting this error from all incoming position measurements.

REFERENCES

[1] SmartBot-RoboShip website, http://www.smartbot.eu/en/roboship/

[2] L. Christensen, N. Fischer, S. Kroffke, J. Lemburg and R. Ahlers. Cost Effective Autonomous Robots for Ballast Water Tank Inspection, Journal of Ship Production and Design 08/2011; 27(3):127-136.

[3] J. Borenstein, H.R. Everett, and L. Feng. Where Am I? Mobile robot positioning: Sensors and techniques. Journal of Robotic Systems 14,4 (1997), 231249.

[4] T. Larsen, K. Hansen, N. Andersen and O. Ravn. Design of Kalman Filters for Mobile Robots; Evaluation of the Kinematic and Odometric Approach, Proceedings of the 1999 IEEE, International Conference on Control Applications, August 22-27, 1999.

[5] A. Surrcio, U. Nunes and R. Arajo. Fusion of Odometry with Magnetic Sensors Using Kalman Filters and Augmented System Models for Mobile Robot Navigation, IEEE ISIE 2005, June 20-23, 2005.

[6] F. van der Heijden, R.P.W. Duin, D. de Ridder and D.M.J. Tax. Classification, Parameter Estimation and State Estimation - An Engineering Approach using MATLAB, 2004 John Wiley and Sons, Ltd.

[7] S. Thrun. Particle Filters in Robotics, in Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI), 2002.

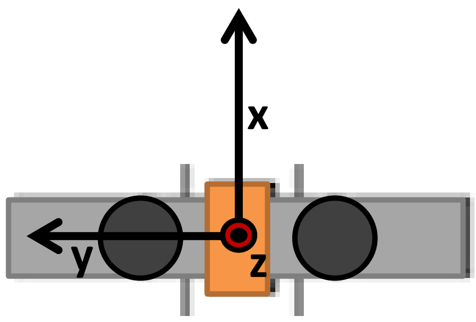
[8] N. Muhammad, D. Fofi, S. Ainouz. Current state of the art of vision based SLAM, SPIE 7251, Image Processing: Machine Vision Applications II, 72510F (3 February 2009).

Appendix

Contents

A. Robot Top View

Figure A1 shows a top view of the SmartBot-RoboShip robot, mounted on a straight rail segment. Indicated are the different modules, the robot's coordinate frame and the robot's center point underneath the IMU.



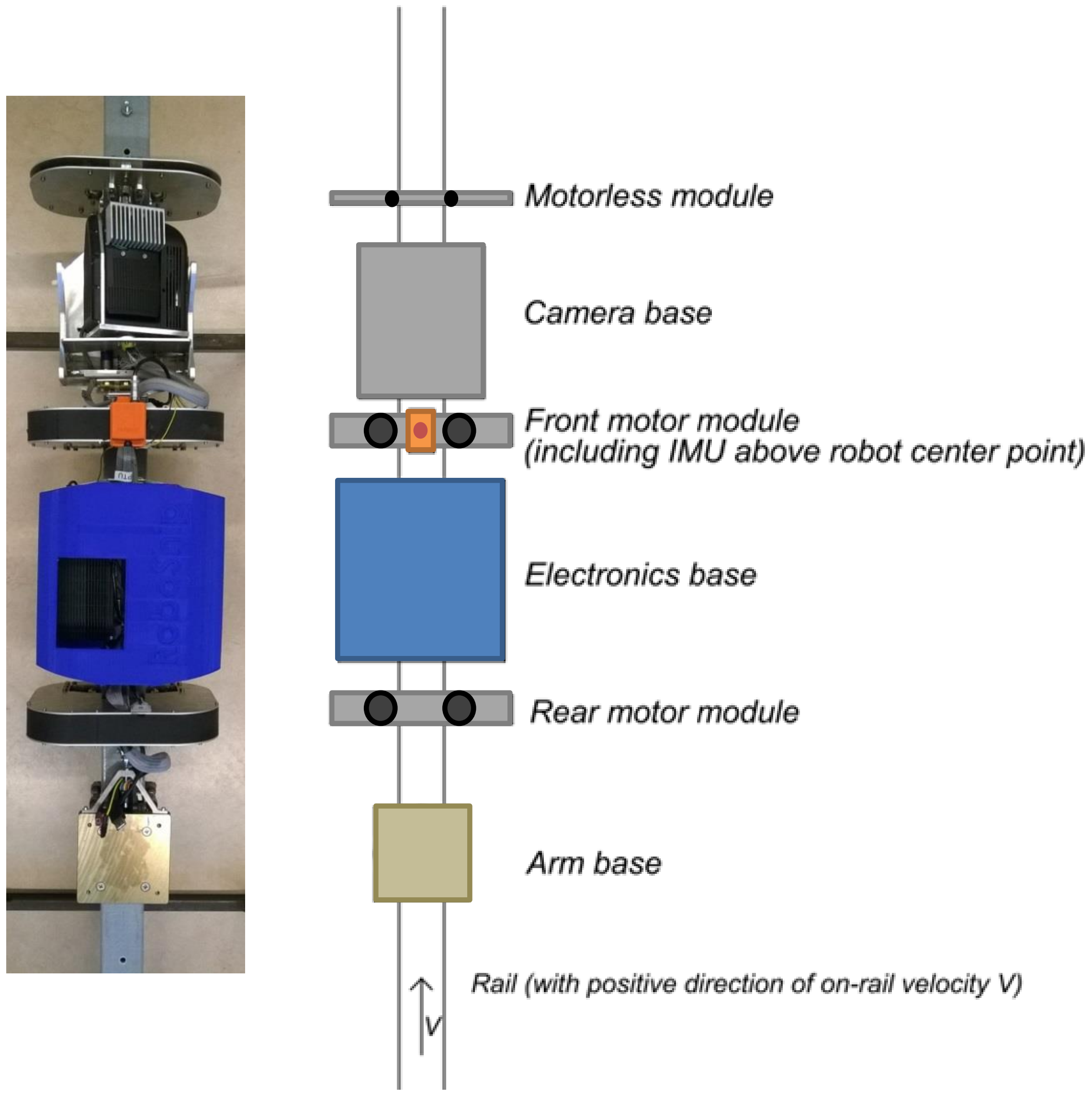

Motorless module
Camera base
Front motor module
(including IMU above robot center point)
Electronics base
Rear motor module
Arm base
Rail (with positive direction of on-rail velocity V)
V

Figure A1: Simplified top view of the robot.

B. Estimated Parameters

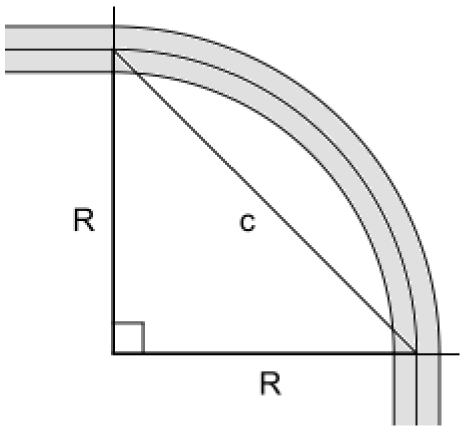
Table B1 presents a list of the estimated parameters, such as the standard deviations used to model process and measurement noise, together with their test method and causes.

Table B1: List of estimated parameters

	Parameter	Value	Unit	Test method	Cause of uncertainty
Process noise	σ_D	$\boldsymbol{\sigma_V \cdot \Delta t}$	m	-	σ_V.
	σ_V	**0.000293**	m/s	Assumed equal to σ_{Vel} but converted to m/s.	Wheel radius irregularities, non-smooth rail, wheel slipping.
	$\sigma_x, \sigma_y, \sigma_z$	$\boldsymbol{\sigma_V \cdot \Delta t}$	m	Assumed noise contribution by inaccuracies in θ and ψ negligible compared to noise contribution by innaccuracies in V.	σ_D, σ_θ and σ_ψ.
	σ_θ	$\boldsymbol{\sigma_{\omega_y} \cdot \Delta t}$	rad	-	$\sigma_{\omega y}$.
	σ_ψ	$\boldsymbol{\sigma_{\omega_z} \cdot \Delta t}$	rad	-	$\sigma_{\omega z}$.
	$\sigma_{\omega y}$	**0.05**	rad/s	Set by tuning for optimal filtering results.	Non-smooth rail.
	$\sigma_{\omega z}$	**0.05**	rad/s	Set by tuning for optimal filtering results.	
	$\sigma_{\kappa_h}, \sigma_{\kappa_v}$	**0.0105**	m^{-1}	Determine curvature κ of corners in test tank assuming constant radius, see Figure B1. Calculate sigma of determined curvatures.	Rail deformation.

Measurement noise	σ_{Vel}	**7.14**	rpm·23	Driving with constant u, use MATLAB to calculate standard deviation of Vel measurements.	Pulse skipping, wheel slipping.
	σ_{Gyr_y}	**0.5**	rad/s	Set by tuning for optimal filtering results.	Sensor noise, vibration.
	σ_{Gyr_z}	**0.5**	rad/s	Set by tuning for optimal filtering results.	Sensor noise, vibration.

Other	r_{eff}	**0.0175**	m	Measure the real on-rail distance L travelled by the robot after k wheel rotations, then determine the average of the effective wheel radii using $r_{eff} = \frac{L}{2\pi k}$.	Deformation of wheels by spring forces, wearing.



$$c^2 = 2R^2$$
$$\rightarrow \kappa = \frac{1}{R} = \frac{c}{\sqrt{2}}$$

Figure B1: Determining curvature by measuring the direct distance between a corner's endpoints

C. Extended Kalman Filter

Figure C1 sketches an overview of the calculations performed by the EKF process. During each cycle, the states are first predicted by using the state equations and then corrected using measurements $\vec{z}_k$. A regular discrete Kalman filter is based on the assumption that the states $\vec{x}_k$ of a system are propagated in discrete time by multiplication with a linear state transition matrix F_k and that (part of) the states are measured through multiplication with a linear measurement matrix H_k. Both are assumed to be corrupted by zero mean white Gaussian noise, with the result that the expectation or mean of every state $E[x_k] = \mu_{x_k}$ can be used as an estimate $\hat{x}_k$ of the real state. According to linear system theory, an expectation vector $\vec{\mu}_x$ and its covariance matrix P propagate through a linear system A as:

$$\vec{\mu}_{x,out} = A\vec{\mu}_{x,in}$$

$$P_{out} = AP_{in}A^T + N$$

...where N is a matrix containing the noise components. Additionally, the EKF takes care of non-linear state and measurement equations $f(\vec{x}_k, u)$ and $h(\vec{x}_k)$ by linearizing them around the current mean $\vec{\mu}_{x,k}$. This enables the use of above equations for a non-linear system.

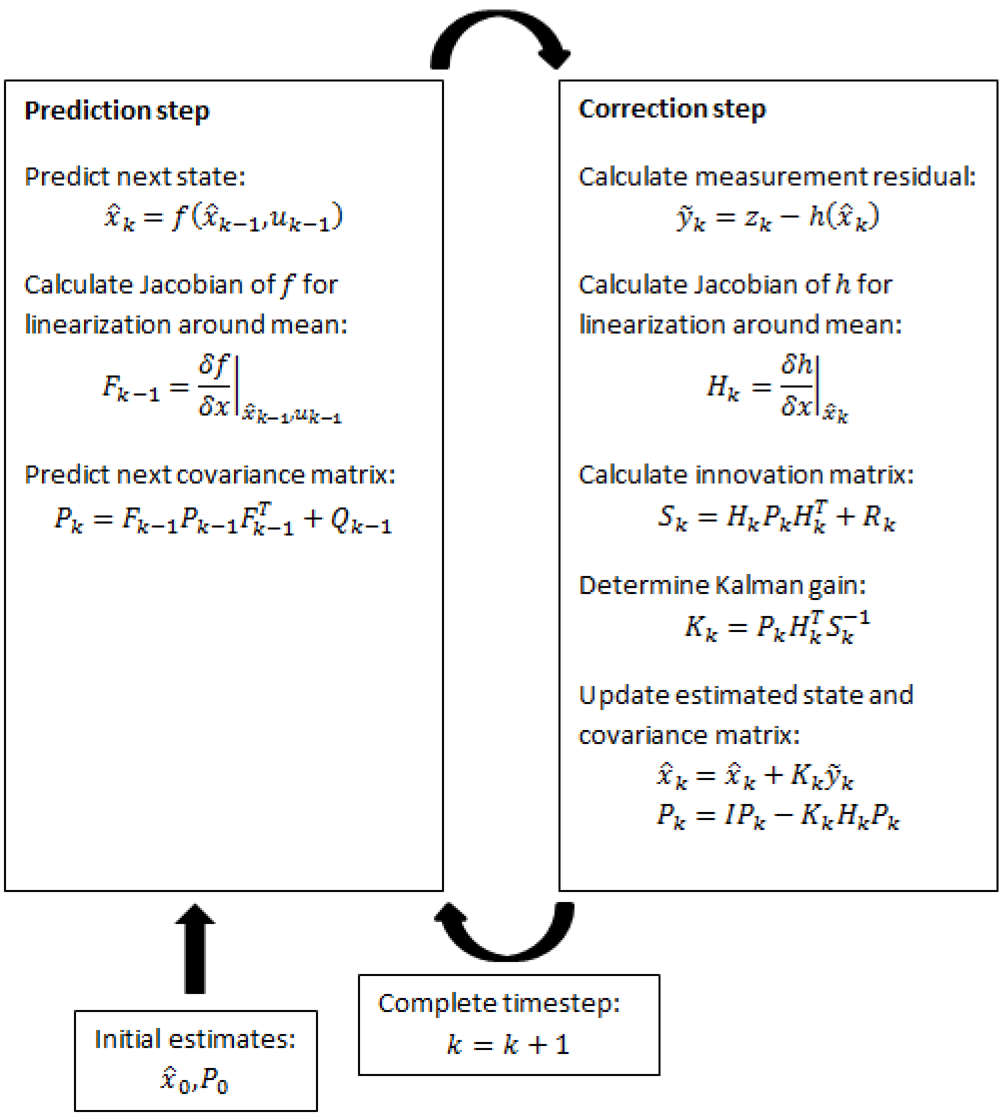

Prediction step
Predict next state:
$\hat{x}_k = f(\hat{x}_{k-1}, u_{k-1})$
Calculate Jacobian of f for linearization around mean:
$F_{k-1} = \left.\frac{\delta f}{\delta x}\right|_{\hat{x}_{k-1}, u_{k-1}}$
Predict next covariance matrix:
$P_k = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1}$
Correction step
Calculate measurement residual:
$\tilde{y}_k = z_k - h(\hat{x}_k)$
Calculate Jacobian of h for linearization around mean:
$H_k = \left.\frac{\delta h}{\delta x}\right|_{\hat{x}_k}$
Calculate innovation matrix:
$S_k = H_k P_k H_k^T + R_k$
Determine Kalman gain:
$K_k = P_k H_k^T S_k^{-1}$
Update estimated state and covariance matrix:
$\hat{x}_k = \hat{x}_k + K_k\tilde{y}_k$
$P_k = IP_k - K_k H_k P_k$
Initial estimates:
$\hat{x}_0, P_0$
Complete timestep:
$k = k + 1$

Figure C1: Flow chart of the EKF process. Source: http://en.wikipedia.org/wiki/Extended_Kalman_filter

D. Experimental Tank

D1. 3D View

Figure D1 shows a CAD drawing of the experimental tank, consisting of the rail mounted inside a complex metal construction. Well visible are the passages through the tank's manholes.

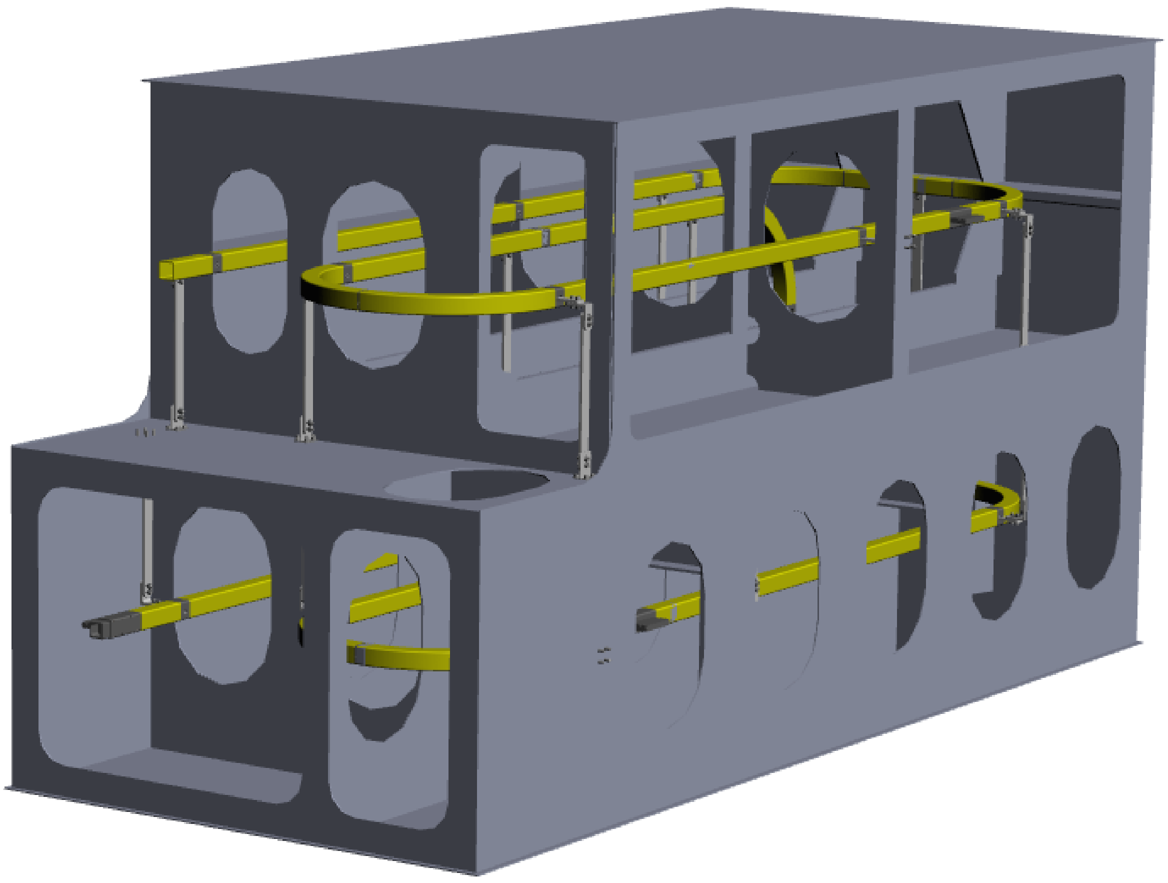


Figure D1: CAD drawing of the experimental tank used for testing.

D2. Top View

Figure D2 represents a simplified top view of the rail that was mounted into the test tank. Indicated are the locations of possible reference points such as corner edges manhole passages.

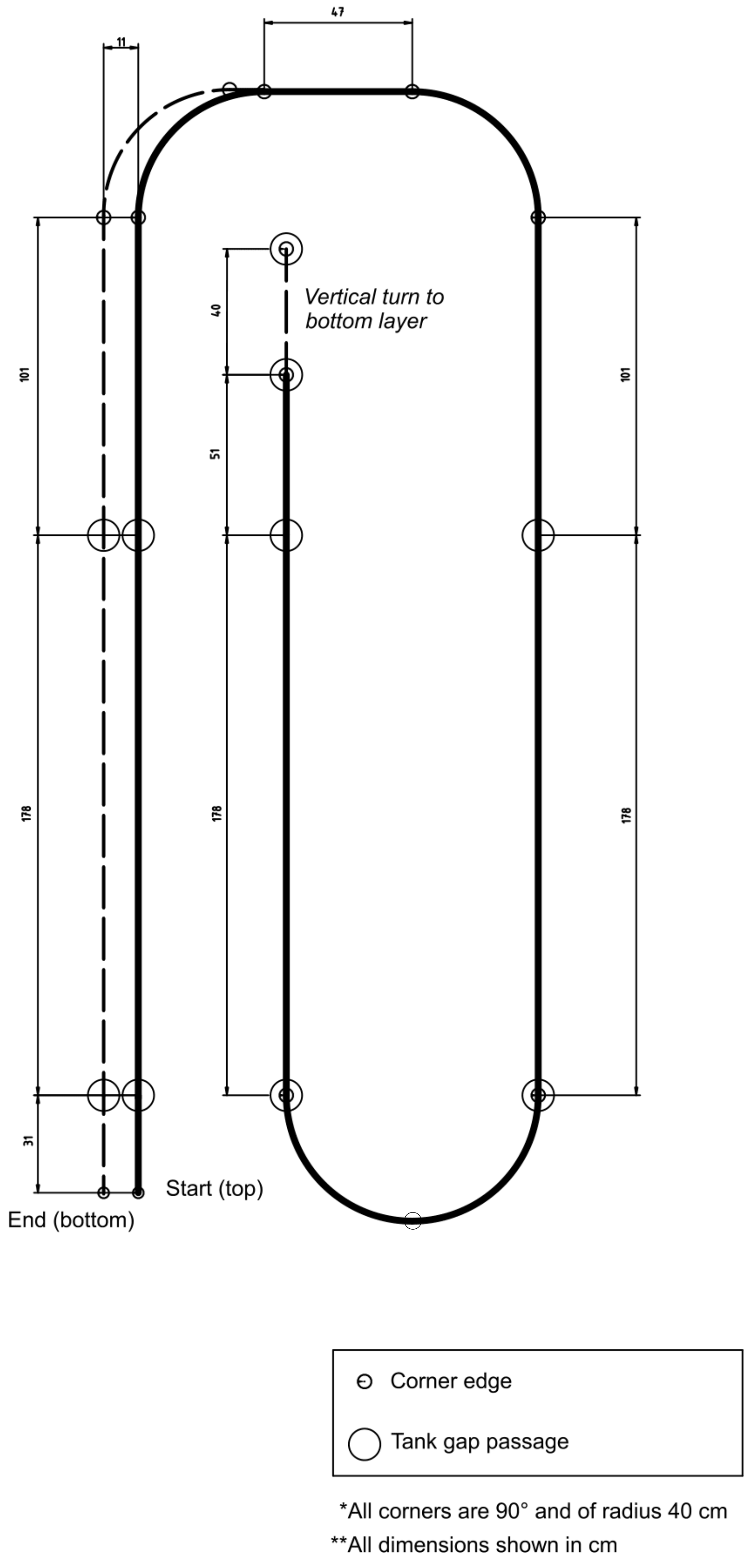

47
11
Vertical turn to bottom layer
40
51
101
101
178
178
178
31
Start (top)
End (bottom)
Corner edge
Tank gap passage
*All corners are 90° and of radius 40 cm
**All dimensions shown in cm

Figure D2: Top view of the experimental tank including important features

E. Test Results

E1. Raw measurement data

Figure E1 shows the measurement data logged during the first half of the test, where the robot travelled through the experimental tank from start to end of the rail. The angular velocity measurements show large amounts of noise but the existence of corners can be seen.

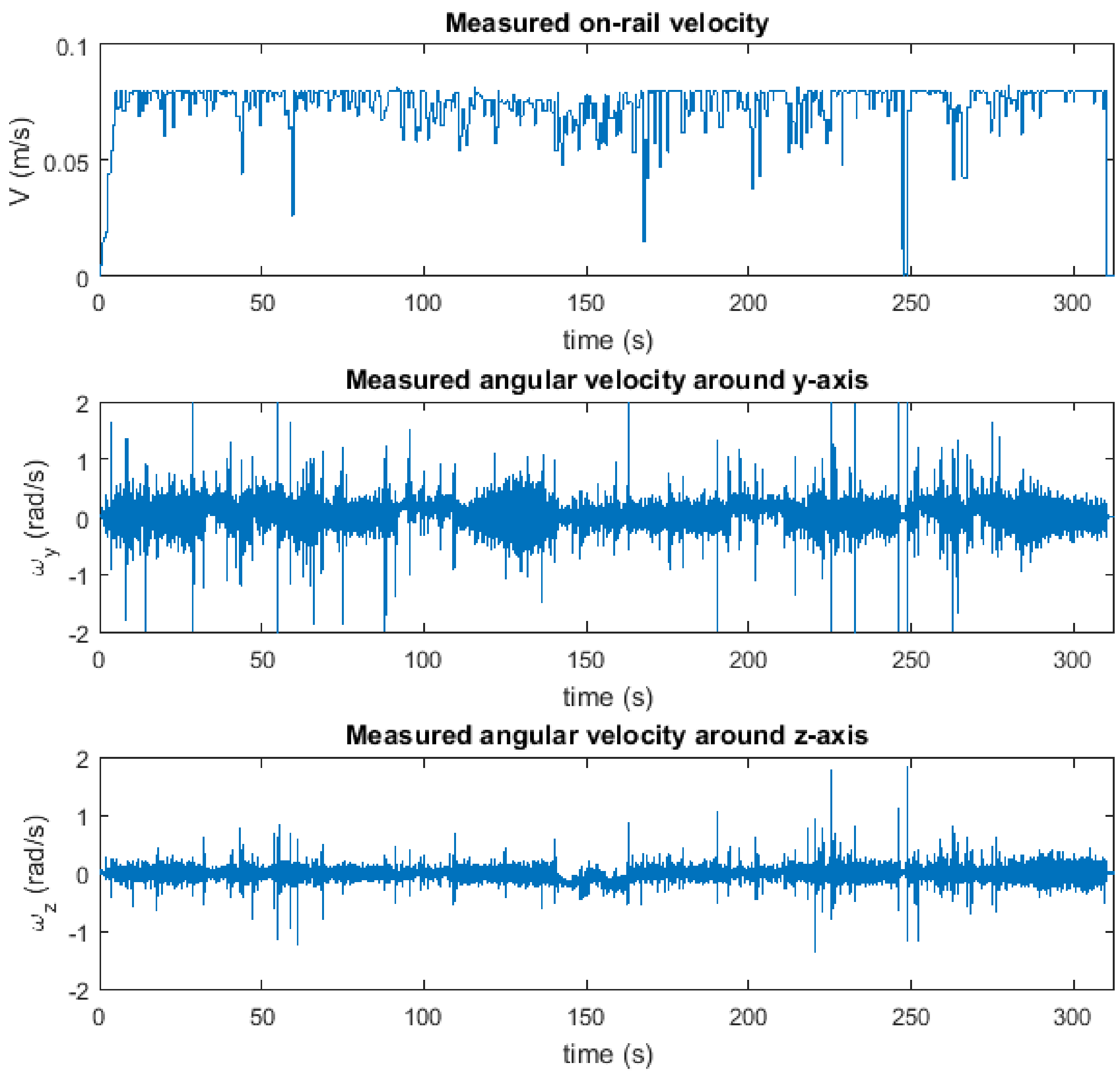


Figure E1: Raw measurement data logged during the first half of the experiment.

E2. Estimated paths without pre-knowledge of real landmark locations

The four plots of Figure E2 show a 2D top view and a 3D view of the estimated paths, where the robot has explored the tank without any pre-knowledge about corners. The top plots were created while travelling from start until end of the rail. The bottom plots were created on the way back.

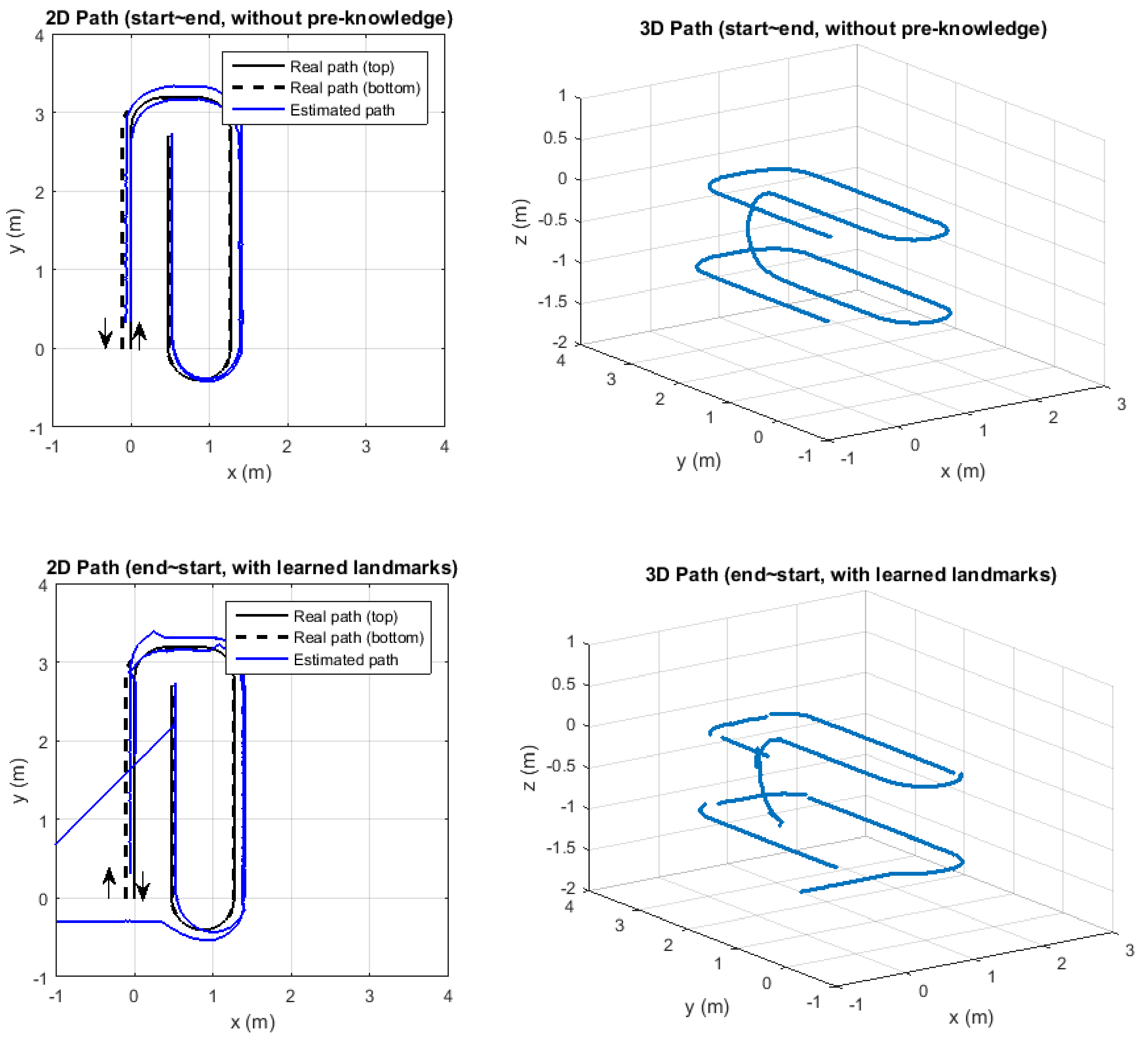


Figure E2: Estimated paths without pre-knowledge of real landmark locations.

E3. Estimated paths with pre-knowledge of real landmark locations

The four plots of Figure E3 show a 2D top view and a 3D view of the estimated paths, where the robot has explored the tank with pre-knowledge; the locations and coordinates of the corner edges were given as initial landmarks Λ_0. The top plots were created while travelling from start until end of the rail. The bottom plots were created on the way back.

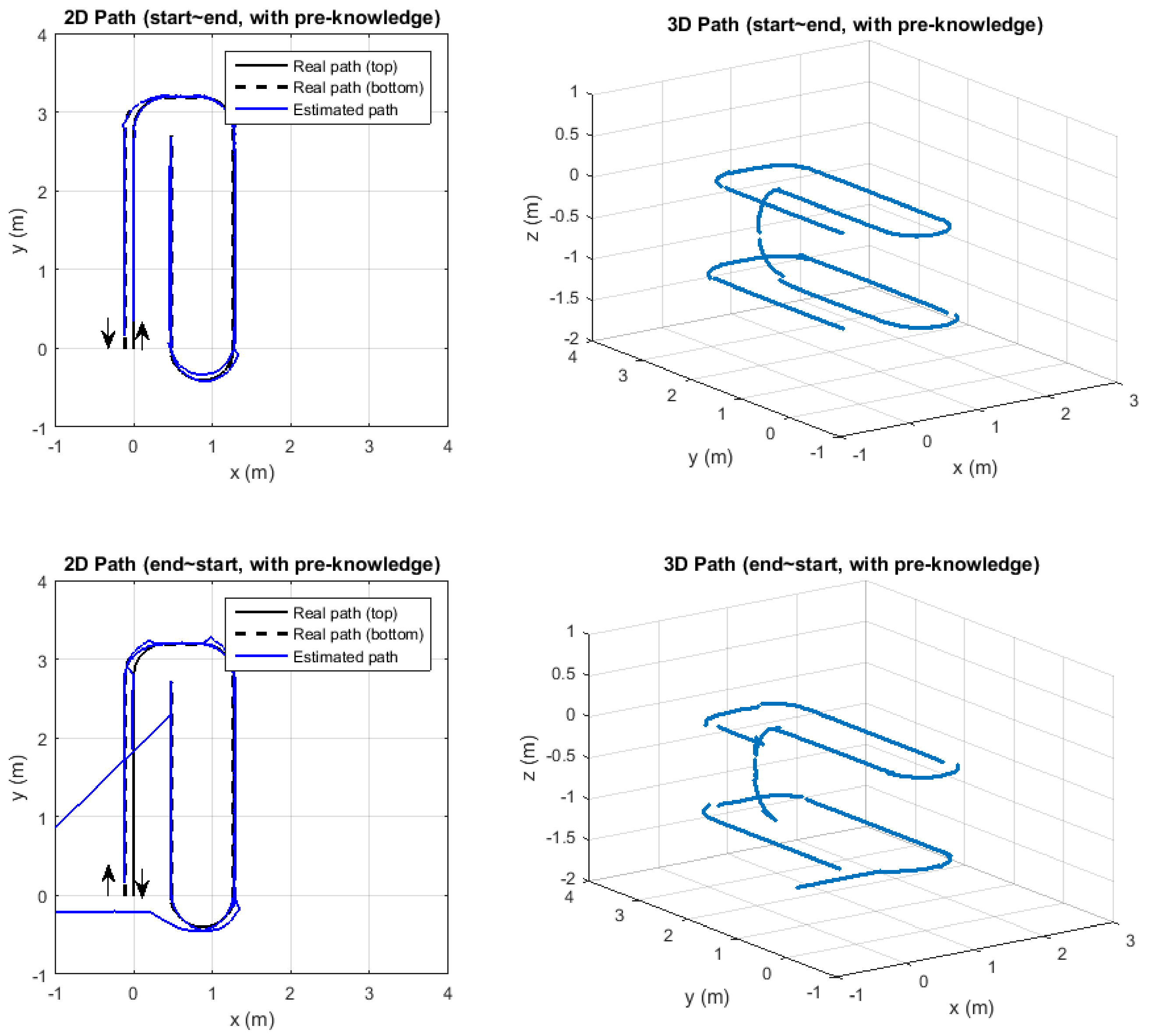


Figure E3: Estimated paths with pre-knowledge of real landmark locations.

E4. Landmark Detection

Figure E4 shows an overview of the landmarks detected by the algorithm along its path through the tank from start until end.

The first subplot indicates the locations and types of gathered corner edges. A type 1 corner corresponds to a horizontal turn whereas a type 2 corner corresponds to a vertical turn. The landmark type also contains a sign; if the robot is driving with a positive velocity (forward), it will mark the start of a corner as positive and the end of a corner as negative.

The remaining subplots show the discrete time patterns of the three magnetic field components B_x, B_y and B_z. Markers are placed wherever local maxima were detected.

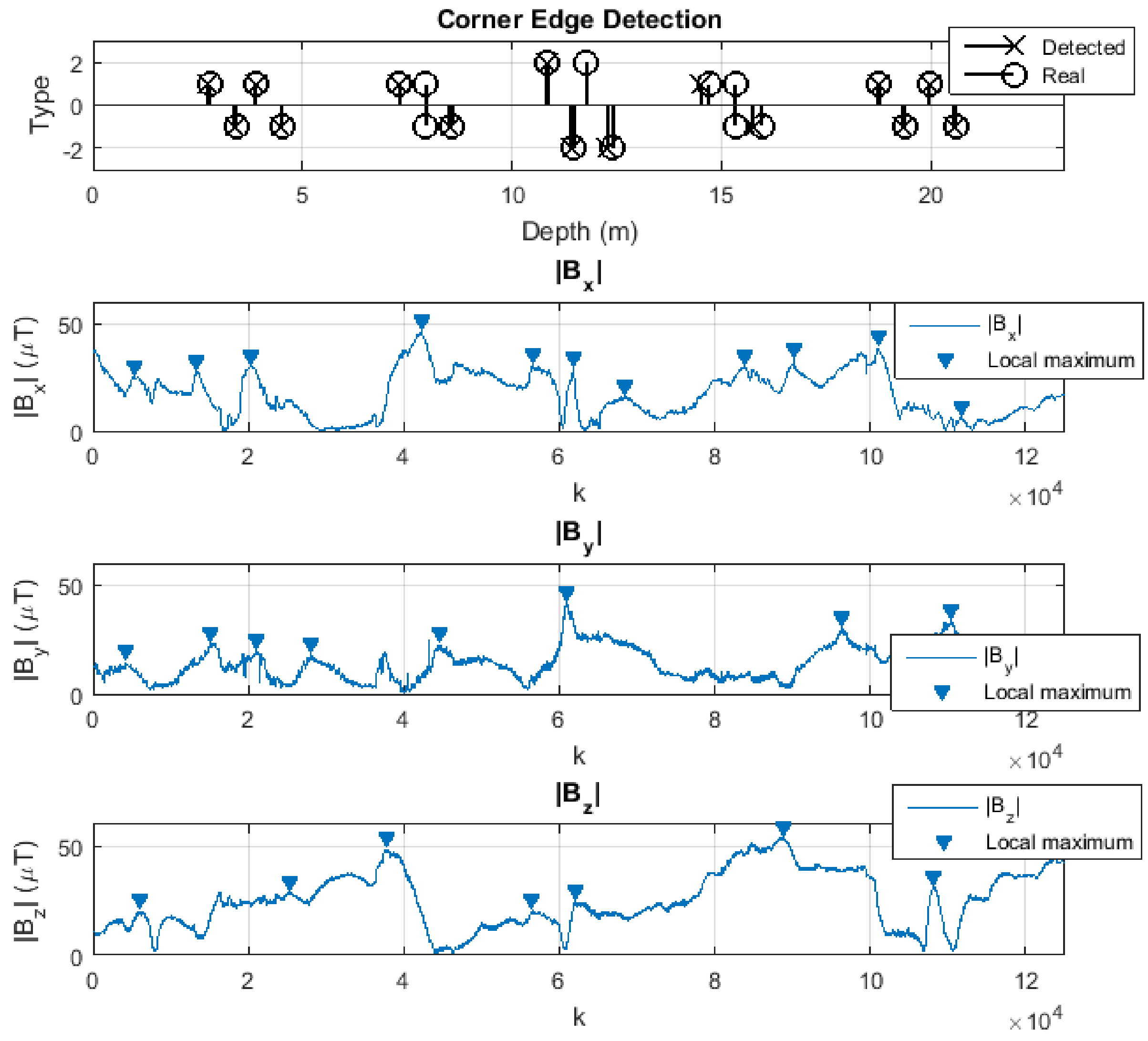


Figure E4: Detected landmarks during the first half of the experiment.

F. MATLAB Code

Part of the used MATLAB code is listed here. F1 shows the main function in which initializing, the actual filtering and the plotting takes place. Functions called during the loop are *ekf.m* (not listed) for the EKF prediction and correction steps and *DetectCorner3D.m* (see F2) for the detection of landmarks in the form of corner edges.

F1. main.m

```
%% Initializing
%N=125000;                              %total dynamic steps (start~end)
N=250000;                               %total dynamic steps (start~end~start)
n=11;                                   %number of states
m=3;                                    %number of measurements
if ~exist('CornerEdges')                %Check for Lambda, the matrix of landmarks
    CornerEdges(1:7,1) = zeros;
end
load 'RealCornerEdges.mat'
%CornerEdges = RealCornerEdges;

kappaStandard = 2.5;                    %Curvature of standardized corner segments
kappaTH = 0.5;                          %curvature threshold
VelErrorFactor = 0.5;                   %Velocity error caused by wrong encoder configurations
d = 2*0.0175;                           %effective wheel diameter
MaxVel = 0.1*23*60/2/pi/(d/2)/VelErrorFactor;           %Maximum on-rail velocity

kOdo = 1;                               %Odometry data index
kJoy = 1;                               %Joystick/user input data index
kMag = 1;                               %Magnetic field data index
dt = 1/400;                             %Timestep in seconds

q = [0.000293*dt 0.000293 0.000293*dt 0.000293*dt 0.000293*dt 0.05*dt 0.05*dt 0.05 0.05 0.0105
0.0105]; %stdevs of process
r = [7.14 0.5 0.5];                     %stdevs of measurements
Q=diag(q).^2;                           % covariance of process
R=diag(r).^2;                           % covariance of measurements

x0 = [0.35;0;0;0.35;0;0;0;0;0;0;0];     % initial state
p = [0.1 0.001 0.01 0.1 0.01 1 1 0.001 0.001 0.001 0.001];             %stdevs of initial
state
P = diag(p).^2;                         % initial state covariance

x=x0;
xV = zeros(n,N);                        %allocate memory for state estimates
sV = zeros(n,N);                        %process predictions
zV = zeros(m,N);                        %measurements
MagV = zeros(4,N);                      %magnetic field measurements

%% Equations
f=@(x)[x(1)+x(2)*dt;                          %1,Position/depth  non-linear state equations
    0;                                        %2,V
    x(3)+sin(-x(6))*x(2)*dt;                  %3,x  -\
    x(4)+cos(x(7))*cos(x(6))*x(2)*dt;         %4,y    > Note: assumed robot is mounted on rail
    x(5)+sin(x(7))*x(2)*dt;                   %5,z  -/        at -90 deg Roll
    x(6)+x(8)*dt;                             %6,theta/Pitch
    x(7)+x(9)*dt;                             %7,psi/Yaw
    x(2)*x(10);                               %8,Gyr_Y
    x(2)*x(11);                               %9,Gyr_Z
    x(10);                                    %10,kappa_horizontal
    x(11)];                                   %11,kappa_vertical

g=@(x,u)[x(1);                                % non-linear state equations with user input
    2*pi*d/2*u/60/23*VelErrorFactor;
    x(3);
    x(4);
    x(5);
    x(6);
    x(7);
    x(8);
    x(9);
    x(10);
    x(11)];

h=@(x)[x(2)*23*60/2/pi/(d/2)/VelErrorFactor;  %Vel
```

```
    -x(8);                                    %Gyr_Y
    x(9)];                                    %Gyr_Z


%% Extended Kalman
for k = 1:N
  while t_vel_fl(kOdo+1)<SampleTimeFineI(k)/1e4
      kOdo = kOdo+1;
  end
  while t_u(kJoy+1)<SampleTimeFineI(k)/1e4
      kJoy = kJoy+1;
  end
  while SampleTimeFineM(kMag)<SampleTimeFineI(k)
      kMag = kMag+1;
  end
  if abs(Gyr_Y(k))>0.5                       %Gyration measurement invalid, too high/low
      Gyr_Y(k)=Gyr_Y(k-1);
  end
  if abs(Gyr_Z(k))>0.5                       %Gyration measurement invalid, too high/low
      Gyr_Z(k)=Gyr_Z(k-1);
  end
  if vel_fl(kOdo)>1.1*MaxSpeed*23*60/2/pi/(d/2)/VelErrorFactor  %Slipping detected
      vel_fl(kOdo) = 0;
  end
  if vel_fr(kOdo)>1.1*MaxSpeed*23*60/2/pi/(d/2)/VelErrorFactor  %Slipping detected
      vel_fr(kOdo) = 0;
  end
  z = [(vel_fl(kOdo)+vel_fr(kOdo))/2; Gyr_Y(k); Gyr_Z(k)];         % measurements
  zV(:,k) = z;                                % save measurement
  xV(:,k) = x;                                % save estimate
  [xV,P,CornerEdges] = DetectCorner3D(xV,P,CornerEdges,k,1,2,3,10,11,6);
  [x,P,K] = ekf(f,xV(:,k),g,u(kJoy),P,h,z,Q,R);            % ekf

  if k>10
      if abs(z(1))<0.1*MaxVel || abs(u(kJoy))<0.1*MaxVel %V expected to be very low;
          x(10)=xV(10,k-10);                          %unreliable curvature estimates
          x(11)=xV(11,k-10);
      end
  end

  if abs(x(10))<kappaTH                       %if not in horizontal corner
      x(6) = round(x(6)/pi*2)*pi/2;           %Pitch multiple of 90 deg
      P(6,6) = 0.01;
  end
  if abs(x(11))<kappaTH                       %if not in vertical corner
       x(7) = round(x(7)/pi*2)*pi/2;          %Yaw multiple of 90 deg
       P(7,7) = 0.01;
  end
  xV(:,k)=x;                                  % save corrected estimate
  s = f(xV(:,k));                             % prediction using process
  s = g(s,u(kJoy))                            % prediction using process
  sV(:,k) = s;                                % save prediction

     %Xsens measures magnetic field in arbitrary units ~40uT/a.u.
     %Mag = 40*([abs(Mag_X(kMag)); abs(Mag_Y(kMag)); abs(Mag_Z(kMag)); mean([abs(Mag_X(kMag))
abs(Mag_Y(kMag)) abs(Mag_Z(kMag))])]);

  if kMag>10                                  %avoid checking index<1
     %Xsens measures magnetic field in arbitrary units ~40uT/a.u.
     Mag = 40*([mean(abs(Mag_X(kMag-10:kMag))); mean(abs(Mag_Y(kMag-10:kMag)));
mean(abs(Mag_Z(kMag-10:kMag))); mean([mean(abs(Mag_X(kMag-10:kMag))) mean(abs((Mag_Y(kMag-
10:kMag)))) mean(abs(Mag_Z(kMag-10:kMag)))])]);
  else
     Mag = 40*([mean(abs(Mag_X(kMag))); mean(abs(Mag_Y(kMag))); mean(abs(Mag_Z(kMag)));
mean([mean(abs(Mag_X(kMag))) mean(abs((Mag_Y(kMag)))) mean(abs(Mag_Z(kMag)))])]);
  end

%% Plotting
(not included)
```

F2. DetectCorner3D.m

```
function [xV,P,CornerEdges] = 
DetectCorner3D(xV,P,CornerEdges,k,indexPos,indexVel,indexX,indexkH,indexkV,indexTheta)
%% Detection of and state correction according to landmarks
% Every landmark is saved as a column in matrix CornerEdges as:
% [Edge type; D; x; y; z; psi; times encountered]

if ~CornerEdges(1:7,1);                                     %no landmarks yet (initially 
CornerEdges = zeroes)
    LandmarkIndex = 1;                                      %overwrite first column
else
    LandmarkIndex = size(CornerEdges,2)+1;                  %create new column
end



if k>41  %Comparing means over last .1 seconds. Avoid checking index<1
  if  abs(mean(xV(indexkH,k-20:k)))>1.25 && abs(mean(xV(indexkH,k-41:k-21)))<1.25
      EdgeType = sign(xV(indexVel,k))*1;                  %Entering horizontal corner
      if  ~any(abs(CornerEdges(2,:)-xV(indexPos,k-2))<0.4)  %New landmark detected
          CornerEdges(1:7,LandmarkIndex)=[EdgeType; xV(indexPos,k-2); xV(indexX,k-2); 
xV(indexX+1,k-2); xV(indexX+2,k-2); round(xV(indexTheta,k-2)/pi*2)*pi/2; 1];     %Register 
characteristics
      else                                                  %Registered landmark nearby
          [xV,P,CornerEdges] = 
AdjustStateToLandmark(xV,P,k,CornerEdges,indexPos,indexX,indexTheta,EdgeType);       %Update 
state and landmark
      end


  elseif abs(mean(xV(indexkH,k-20:k)))<1.5 && abs(mean(xV(indexkH,k-41:k-21)))>1.5
      EdgeType = -sign(xV(indexVel,k))*1;                  %Leaving horizontal corner
      xV(6,k) = round(xV(6,k)/pi*2)*pi/2;             %Leaving corner, so Pitch multiple of 90 
deg
      P(6,6) = 0.01;                                       %update Yaw variance
      if  ~any(abs(CornerEdges(2,:)-xV(indexPos,k-2))<0.4)   %New landmark detected
          CornerEdges(1:7,LandmarkIndex)=[EdgeType; xV(indexPos,k-2); xV(indexX,k-2); 
xV(indexX+1,k-2); xV(indexX+2,k-2); round(xV(indexTheta,k-2)/pi*2)*pi/2; 1];     %Register 
characteristics
      else                                                   %Registered landmark nearby
          [xV,P,CornerEdges] = 
AdjustStateToLandmark(xV,P,k,CornerEdges,indexPos,indexX,indexTheta,EdgeType);       %Update 
state and landmark
      end


  elseif abs(mean(xV(indexkV,k-20:k)))>1.0 && abs(mean(xV(indexkV,k-41:k-21)))<1.0
      EdgeType = sign(xV(indexVel,k))*2;                    %Entering vertical corner
      if  ~any(abs(CornerEdges(2,:)-xV(indexPos,k-2))<0.4)    %New landmark detected
          CornerEdges(1:7,LandmarkIndex)=[EdgeType; xV(indexPos,k-2); xV(indexX,k-2); 
xV(indexX+1,k-2); xV(indexX+2,k-2); round(xV(indexTheta,k-2)/pi*2)*pi/2; 1];     %Register 
characteristics
      else                                                    %Registered landmark nearby
          [xV,P,CornerEdges] = 
AdjustStateToLandmark(xV,P,k,CornerEdges,indexPos,indexX,indexTheta,EdgeType);       %Update 
state and landmark
      end


  elseif abs(mean(xV(indexkV,k-20:k)))<1.25 && abs(mean(xV(indexkV,k-41:k-21)))>1.25
      EdgeType = -sign(xV(indexVel,k))*2;                   %Leaving vertical corner
      xV(7,k) = round(xV(7,k)/pi*2)*pi/2;             %Leaving corner, so Yaw multiple of 90 
deg
      P(7,7) = 0.01;                                      %update Yaw variance
      if  ~any(abs(CornerEdges(2,:)-xV(indexPos,k-2))<0.4)    %New landmark detected
          CornerEdges(1:7,LandmarkIndex)=[EdgeType; xV(indexPos,k-2); xV(indexX,k-2); 
xV(indexX+1,k-2); xV(indexX+2,k-2); round(xV(indexTheta,k-2)/pi*2)*pi/2; 1];     %Register 
characteristics
      else                                                    %Registered landmark nearby
          [xV,P,CornerEdges] = 
AdjustStateToLandmark(xV,P,k,CornerEdges,indexPos,indexX,indexTheta,EdgeType);       %Update 
state and landmark
      end
  else
      return;
  end
```

```
else
  return;
end
end


function [xV,P,CornerEdges] =
AdjustStateToLandmark(xV,P,k,CornerEdges,indexPos,indexX,indexTheta,EdgeType)
for LandmarkIndex = 1:1:size(CornerEdges,2)                   %Loop through registered
landmarks
    if EdgeType == -2 || EdgeType == 2                          %Define confidential range of
vertical turns
        Range = 0.3;
    else                                                         %...and of horizontal turns
        Range = 0.3;
    end
    if(abs(CornerEdges(2,LandmarkIndex)-xV(indexPos,k-2))<Range &&
CornerEdges(1,LandmarkIndex)==EdgeType)         %If identical landmark within range found
        CornerEdges(7,LandmarkIndex) = CornerEdges(7,LandmarkIndex) +1; %Increment times
encountered
        Encounters = CornerEdges(7,LandmarkIndex);
        NewLandmarkEstimate = [xV(indexPos,k-2); xV(indexX,k-2); xV(indexX+1,k-2);
xV(indexX+2,k-2)];                              %Hold new location estimate
        xV(indexPos,k) =
NewLandmarkEstimate(1)/Encounters+CornerEdges(2,LandmarkIndex)*(Encounters-1)/Encounters;
%Correct pos according to weighted average of prev. encountered locations and present
        xV(indexX:indexX+2,k) =
NewLandmarkEstimate(2:4)./Encounters+CornerEdges(3:5,LandmarkIndex).*(Encounters-
1)./Encounters;            %Correct coordinates according to weighted average of prev.
encountered locations and present
        xV(indexTheta,k) = CornerEdges(6,LandmarkIndex);            %Correct theta
        P(indexPos,indexPos) = Range^2;                             %Update variance
        P(indexX,indexX) = Range^2;                                 %Update variance
        P(indexX+1,indexX+1) = Range^2;                             %Update variance
        P(indexX+2,indexX+2) = Range^2;                             %Update variance
        return;                                 %Avoid identifying with multiple registered
landmarks, choose only the first one registered.
    end
end
end
```