SAPIENZA
UNIVERSITÀ DI ROMA

Encoders and IMU sensor fusion using the Kalman Filter

Faculty of Information Engineering, Informatics and Statistics

Degree Program in Computer and Control Engineering

Candidate

Filippo Graziano
ID number 1761694

Candidate's Tutor

Prof. Giorgio Grisetti

Academic Year 2018/2019

Encoders and IMU sensor fusion using the Kalman Filter
Honours Programme's report. Sapienza – University of Rome

This report has been typeset by LaTeX and the Sapthesis class.

Author's email: graziano.1761694@studenti.uniroma1.it

Abstract

This report completes my Bachelor's thesis[1] in which we describe the robot used for our tests and the localization techniques it is using:

- our robot uses two rotary encoders to measure the wheels rotations and calculate the odometry using Differential Drive;
- it also uses IMU observations to calculate the odometry using IMU Navigation.

In this report we will show how to use the sensors together in the update of a single global odometry and we will perform sensor fusion based on Kalman filtering.

The report is structured as follows:

- Chapter 1 introduces sensor fusion and Kalman filtering.
- Chapter 2 describes the Kalman filter algorithm.
- Chapter 3 includes a broad description of the transition and the observation model used for our robot.
- Chapter 4 shows the results given by our filter and provides some final considerations.
- The implementation details of the Kalman filter are provided in Appendix A.

The code for the Kalman Filter is available at `https://github.com/FilippoGraziano98/Acquisition_Platform`, where it is integrated with the rest of the code to manage the robot.

Contents

1 Introduction **1**

2 Kalman Filter **3**
2.1 Predict phase 5
2.2 Update phase 5

3 Kalman Filter design for our robot **7**
3.1 Predict phase 7
3.1.1 State vector 7
3.1.2 Controls vector 8
3.1.3 Transition model 9
3.2 Update phase 10
3.2.1 IMU observations 10
3.2.2 Encoders observations 11
3.2.3 Joint Update phase 15

4 Conclusions **17**

Appendix A Kalman Filter Implementation **21**
A.1 Matrix Operations 21
A.2 Parameters and Global Variables 22
A.3 Predict Phase 25
A.4 Update Phase 25

References **29**

Chapter 1

Introduction

Sensor fusion is a very important technique because almost all sensors have some limitations. It consists in combining sensory data derived from disparate sources such that the resulting information has less uncertainty than would be possible when using the single sources individually.

In our scenario we will combine different localization sensors, the encoders and the IMU. In this way each sensor will compensate for the weaknesses of the other and we will calculate a more accurate and robust odometry.

More in detail we will use Kalman filtering. This is an algorithm which takes measurements containing statistical noise and produces estimates of unknown variables (in our case we will estimate the position and the orientation of our robot) by updating a joint probability distribution over the variables in each timeframe.

Moreover the Kalman filter is the optimal filter for linear systems with Gaussian noise.

Chapter 2

Kalman Filter

If there is no uncertainty we can model our system as described in following picture:

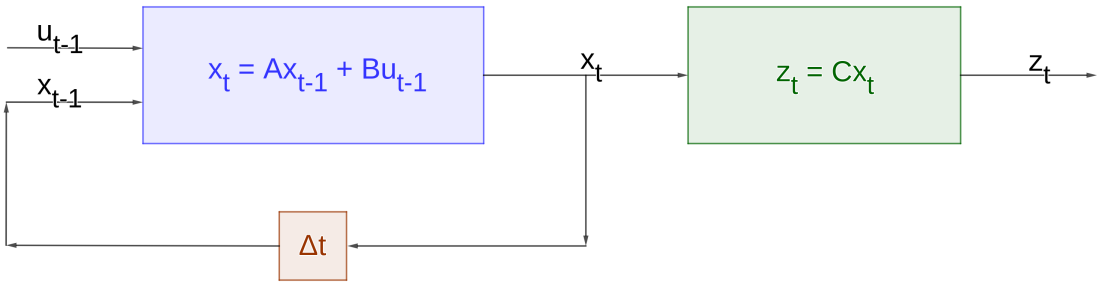

Figure 2.1. Deterministic System model.

- The state variables x_t represents out robot's position and orientation.
- The control vector u_t represents the rotational and translational accelerations our robot is subject to.
- The observation vector z_t represents the sensors measurements.

As we can see in this model there is no uncertainty, the transition from a state to another is deterministic, we assume there is always no error in the observations or in the state.

However our inputs and our observations are not known with certainty. Therefore we must modify the previous model to include uncertainty and errors.

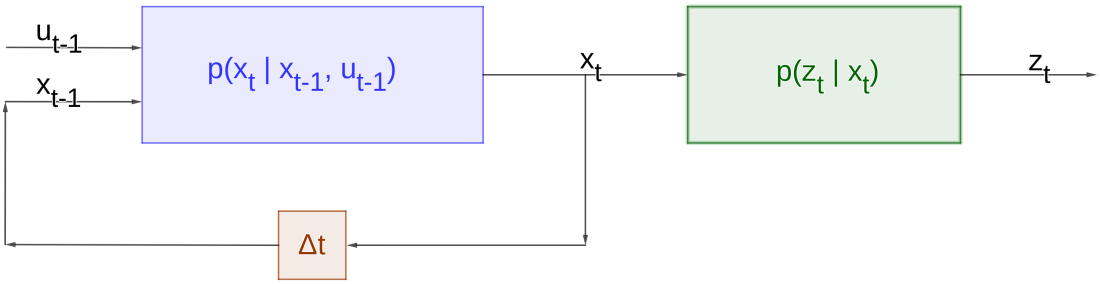

Figure 2.2. Stochastic System model.

- $p(x_t|x_{t-1}, u_{t-1})$ is the transition model, which correlates the current state with the previous state and the previous control.

- $p(z_t|x_t)$ is the observation model, which correlates the current observation with the current state.

The goal of the Kalman filter is to compute the distribution over the current state given the sequence of all controls and of all observations up to the current instant, i.e. $p(x_t|u_{0:t-1}, z_{0:t})$.

As a basic idea we will perform two successive phases at each time instant, a predict phase and an update one.

- In the predict phase we only consider the previous state and control to update the state. Therefore we will calculate the expected state in consequence to the control we gave to the system.

- In the update phase we update the state according to what we have observed and to how likely it is to have these observations from the current state.

In particular, in Kalman filtering we assume the state and the noise to be Gaussian. Since Gaussian distributions are closed under affine transformation, chain rule, marginalization and conditioning the state will always remain a Gaussian distribution. Therefore in each phase we will just have to compute the new parameters of the Gaussian.

In the following we will briefly present the Kalman filtering algorithm. For further details about Kalman Filtering we recommend the slides of the "Probabilistic Robotics" Course, held by Prof. Giorgio Grisetti at Sapienza University of Rome[2].

2.1 Predict phase

In the predict phase we update the estimated state incorporating the latest control. The transition model is an affine transformation of the state and the control:

$$x_t = A_t x_{t-1} + B_t u_{t-1} \tag{2.1}$$

The previous state x_{t-1} is a Gaussian with mean $\mu_{t-1|t-1}$ and covariance $\Sigma_{t-1|t-1}$. While the control vector u_{t-1} is a Gaussian with mean $\mu_{u,t-1}$ and covariance $\Sigma_{u,t-1}$.

After the predict phase, the estimated state $p(x_{t|t-1})$ is still a Gaussian, with the following parameters:

$$\mu_{t|t-1} = A_t \mu_{t-1|t-1} + B_t \mu_{u,t-1} \tag{2.2}$$

$$\Sigma_{t|t-1} = A_t \Sigma_{t-1|t-1} A_t^T + B_t \Sigma_{u,t-1} B_t^T \tag{2.3}$$

2.2 Update phase

In the update phase we incorporate the new observations in the state estimate, starting form the observation model:

$$z_t = C_t x_{t|t-1} \tag{2.4}$$

We want to compute the probability $p(x_t|z_t)$ given the following distributions:

- $p(z_t|x_t)$ which is a Gaussian distribution with mean $\mu_{z,t} = C_t x_{t|t-1}$ and covariance $\Sigma_{z,t}$,
- $p(x_{t|t-1})$ which is the Gaussian distribution of the estimated state after the predict phase.

In the update phase we compute the Kalman gain:

$$K_t = \Sigma_{t|t-1} C_t^T (\Sigma_{z,t} + C_t \Sigma_{t|t-1} C_t^T)^{-1} \tag{2.5}$$

Using the Kalman gain we can update the parameters of the Gaussian distribution representing the state:

$$\mu_{t|t} = \mu_{t|t-1} + K_t(z_t - \mu_{z,t}) \tag{2.6}$$

$$\Sigma_{t|t} = (I - K_t C_t)\Sigma_{t|t-1} \tag{2.7}$$

Role of the observation covariance We can rewrite the 2.5 as follows:

$$K_t = \frac{\Sigma_{t|t-1} C_t^T}{\Sigma_{z,t} + C_t \Sigma_{t|t-1} C_t^T}$$

In this form it becomes evident the role of the observation covariance $\Sigma_{z,t}$ in the update phase.

If the observation covariance (i.e. the observation uncertainty) tends to zero, then the Kalman Gain tends to:

$$K_t = \frac{\Sigma_{t|t-1} C_t^T}{C_t \Sigma_{t|t-1} C_t^T} = \frac{1}{C_t}$$

and consequently:

$$\begin{aligned}
\mu_{t|t} &= \mu_{t|t-1} + K_t(z_t - \mu_{z,t}) = \mu_{t|t-1} + \frac{1}{C_t}(z_t - C_t \mu_{t|t-1}) = \\
&= \mu_{t|t-1} + \frac{1}{C_t} z_t - \mu_{t|t-1} = C_t^{-1} z_t \\
\Sigma_{t|t} &= (I - K_t C_t)\Sigma_{t|t-1} = (I - \frac{1}{C_t} C_t)\Sigma_{t|t-1} = (I - I)\Sigma_{t|t-1} = 0
\end{aligned}$$

Therefore if the observation uncertainty tends to zero, the state estimate can be calculated only from the current observation and will have zero covariance.

If the observation covariance tends to infinity, then the Kalman Gain tends to zero and consequently:

$$K_t = \frac{\Sigma_{t|t-1} C_t^T}{\Sigma_{z,t} + C_t \Sigma_{t|t-1} C_t^T} = 0$$

$$\begin{aligned}
\mu_{t|t} &= \mu_{t|t-1} + K_t(z_t - \mu_{z,t}) = \mu_{t|t-1} \\
\Sigma_{t|t} &= (I - K_t C_t)\Sigma_{t|t-1} = \Sigma_{t|t-1}
\end{aligned}$$

Therefore if the observation uncertainty tends to infinity, the state estimate will not be updated by the observations.

Wrapping up, if the observation covariance tends to zero, the state estimate will move immediately on the observations; while if the covariance tends to infinity the state estimate will not incorporate the observations.

Chapter 3

Kalman Filter design for our robot

Since the 16 MHz CPU on the MEGA2560 board was not able to carry out all the matrix calculations required by the Kalman filter at 100 Hz, we have decided to do all the computations host-side using the 2.7 GHz CPU on our PC[1] and send the observations via UART from the robot at 100 Hz.

In order to use the Kalman filter presented in the previous chapter, we only have to decide how to model our problem. More in detail we must:

- choose how to represent the state,
- choose how to represent the controls,
- implement a transition model,
- choose how to represent the observations,
- implement an observation model.

Our implementation of the filter is shown in Appendix A.

3.1 Predict phase

3.1.1 State vector

Our state must represent all the information on the position and orientation of the robot. Therefore we have used an 8-components state:

- x representing the global displacement along the x-axis,

[1] PC stands for Portable Computer

- $\dot{x}$ representing the translational velocity along the global x-axis,
- $\ddot{x}$ representing the translational acceleration along the global x-axis,
- y representing the global displacement along the y-axis,
- $\dot{y}$ representing the translational velocity along the global y-axis,
- $\ddot{y}$ representing the translational acceleration along the global y-axis,
- θ representing the global orientation along the z-axis,
- $\dot{\theta}$ representing the rotational velocity along the global z-axis.

We have included the translational accelerations because our accelerometers measure accelerations, instead we have stopped to rotational velocities because this is what our gyroscope measures.

Since our state will be represented by a Gaussian distribution we need to initialize both its mean and its covariance. We have decided to initialize them both with zeros because we want to measure the displacement from the starting position and because there is no uncertainty at the beginning.

3.1.2 Controls vector

Since our robot is pushed by hand (having no motors), we have no knowledge about the controls vector. However we will use this vector to model noise in the transition process and to introduce uncertainty in it; therefore we will assume the control vector to be zero-mean but with non-zero covariance.

We will use the controls vector to model spikes in the translational accelerations or in the rotational velocity which could not be observed.

- $\dddot{x}$ representing a derivative of the translational acceleration along the global x-axis,
- $\dddot{y}$ representing a derivative of the translational acceleration along the global y-axis,
- $\ddot{\theta}$ representing a derivative of the rotational velocity (i.e. a rotational acceleration) along the global z-axis.

We will use the following covariance matrix:

$$\Sigma_u = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} \tag{3.1}$$

We need high control covariances because, since we are pushing the robot by hand, our process is very noisy; surely if we had engines controlling the robot's wheels the process would be more continuous and we could use lower covariances.

3.1.3 Transition model

We are now ready to implement the transition model, which will be used in the Predict phase. In order to define the affine transformation 2.1, we must define the state transition matrix A_t and the input matrix B_t.

According to how we defined the state in section 3.1.1 and to some basics of kinematics[2], we can define the following state transition matrix:

$$A = \begin{pmatrix} 1 & \Delta t & .5\Delta t^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & .5\Delta t^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

According to how we have defined the controls vector in section 3.1.2 we can

[2]Basic kinematic relationships between position, velocity and acceleration:

$x = x + \dot{x}\Delta t + .5\ddot{x}\Delta t^2$

$\dot{x} = \dot{x} + \ddot{x}\Delta t$

define the following input matrix:

$$B = \begin{pmatrix} .167\Delta t^3 & 0 & 0 \\ .5\Delta t^2 & 0 & 0 \\ \Delta t & 0 & 0 \\ 0 & .167\Delta t^3 & 0 \\ 0 & .5\Delta t^2 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & .167\Delta t^3 \\ 0 & 0 & .5\Delta t^2 \\ 0 & 0 & \Delta t \end{pmatrix} \tag{3.3}$$

We can notice how this two matrices are independent of the precise current sampling instant and they remain constant throughout all the computation.

From the equations 2.2 and 2.3, remembering that our control is zero-mean we obtain the following predict phase equations:

$$\mu_{t|t-1} = A\mu_{t-1|t-1} \tag{3.4}$$

$$\Sigma_{t|t-1} = A\Sigma_{t-1|t-1}A^T + B\Sigma_{u,t-1}B^T \tag{3.5}$$

3.2 Update phase

3.2.1 IMU observations

IMU observations vector

The accelerometer gives us measurements about the accelerations along the three axes, while the gyroscope measures rotational velocities along the three axes. Assuming that our robot moves only in 2D we will consider only:

- $\ddot{x}_z$ which is the translational acceleration along the x-axis measured by the accelerometer,
- $\ddot{y}_z$ which is the translational acceleration along the y-axis measured by the accelerometer,
- $\dot{\theta}_z$ which is the rotational velocity along the z-axis measured by the gyroscope.

We will consider these measurements as independent, therefore the IMU covariance matrix will be diagonal. Moreover since the accelerometer is very noisy and inaccurate, we will give the accelerometer observations a relevant covariance, instead

since the gyroscope is much less noisy we will give it less covariance.

$$\Sigma_{IMU} = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.000001 \end{pmatrix} \tag{3.6}$$

Let us note how the measurement error covariances were defined by trial and error since they were not provided by the IMU manufacturer.

IMU observation model

To define the observation model for the IMU observations, we must define only the observation matrix C_t to be used in the equation 2.4.

$$C_{IMU} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

3.2.2 Encoders observations

Unlike the IMU which gives us measurements for which the definition of an observation matrix is immediate, the encoders measures the wheel rotations.

Linearized Observation Matrix

We have started considering the wheels rotations as the observations to be used in Kalman filter. Since we assume the encoders to be very accurate we assumed these measurements to have small covariance.

$$\Sigma_{encoders,linearized} = \begin{pmatrix} 0.0000001 & 0 \\ 0 & 0.0000001 \end{pmatrix} \tag{3.8}$$

Let us note how these measurement error covariances were defined by trial and error since they were not provided by the encoders manufacturer.

As described in my thesis[1], the rotations of the left and right wheel (l and r respectively) are related to the parameters of the local rotation (the radius R and the angle $\Delta\theta$) by the equations (where b is the length of the wheel axis):

$$r = (R + b/2)\Delta\theta \tag{3.9}$$

$$l = (R - b/2)\Delta\theta \tag{3.10}$$

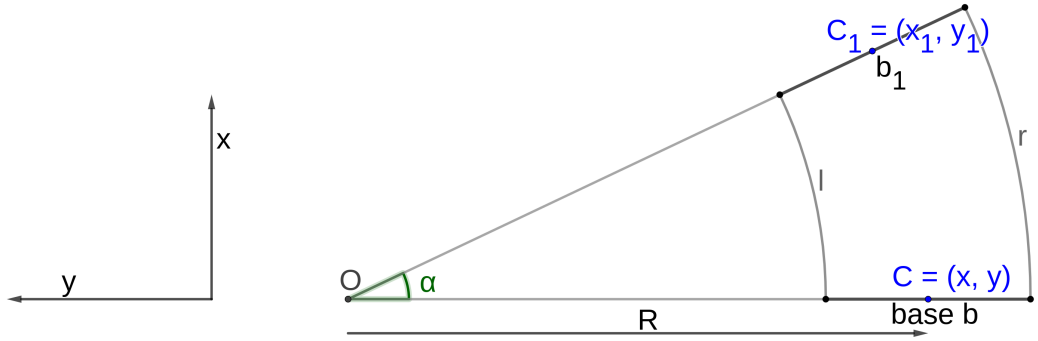

Figure 3.1. Model of an instantaneous movement through a rotation.

Moreover R is related to the local movement by the equation:

$$\Delta x_{local} = R \sin \Delta\theta \tag{3.11}$$

and the local movement can be related to the global movement using:

$$\Delta x_{local} = \Delta x \cos\theta + \Delta y \sin\theta \tag{3.12}$$

Therefore using the 3.11 and the 3.12:

$$R = \frac{\Delta x_{local}}{\sin \Delta\theta} = \frac{\Delta x \cos\theta + \Delta y \sin\theta}{\sin \Delta\theta} \tag{3.13}$$

Finally using the 3.9, the 3.10 and the 3.13:

$$r = (\frac{\Delta x \cos\theta + \Delta y \sin\theta}{\sin \Delta\theta} + b/2)\Delta\theta \tag{3.14}$$

$$l = (\frac{\Delta x \cos\theta + \Delta y \sin\theta}{\sin \Delta\theta} - b/2)\Delta\theta \tag{3.15}$$

These equations enable us to calculate the observations starting from the state, however these equations are not linear. Therefore we have linearized them using the

Taylor expansion to the first order centered in zero[3].

$$r = \cos\theta\Delta x + \sin\theta\Delta y + \frac{b}{2}\Delta\theta \tag{3.16}$$

$$l = \cos\theta\Delta x + \sin\theta\Delta y - \frac{b}{2}\Delta\theta \tag{3.17}$$

In order to use these equations in the observation matrix we must write Δx, Δy, $\Delta\theta$ as a function of the state:

$$\Delta x = \dot{x}\Delta t + \frac{1}{2}\ddot{x}\Delta t \tag{3.18}$$

$$\Delta y = \dot{y}\Delta t + \frac{1}{2}\ddot{y}\Delta t \tag{3.19}$$

$$\Delta\theta = \dot{\theta}\Delta t \tag{3.20}$$

Using the 3.16, the 3.17, the 3.18, the 3.19 and the 3.20:

$$r = (\cos\theta\Delta t)\dot{x} + (.5\cos\theta\Delta t^2)\ddot{x} + (\sin\theta\Delta t)\dot{y} + (.5\sin\theta\Delta t^2)\ddot{y} + (.5b\Delta t)\dot{\theta} \tag{3.21}$$

$$l = (\cos\theta\Delta t)\dot{x} + (.5\cos\theta\Delta t^2)\ddot{x} + (\sin\theta\Delta t)\dot{y} + (.5\sin\theta\Delta t^2)\ddot{y} - (.5b\Delta t)\dot{\theta} \tag{3.22}$$

Now we are ready to write the linearized observation matrix:

$$C_{encoders,linearized} = \begin{pmatrix} 0 & \cos\theta\Delta t & .5\cos\theta\Delta t^2 & 0 & \sin\theta\Delta t & .5\sin\theta\Delta t^2 & 0 & .5b\Delta t \\ 0 & \cos\theta\Delta t & .5\cos\theta\Delta t^2 & 0 & \sin\theta\Delta t & .5\sin\theta\Delta t^2 & 0 & -.5b\Delta t \end{pmatrix} \tag{3.23}$$

However in this matrix all the dependencies from y, $\dot{y}$, $\ddot{y}$ are functions of $\sin\theta$. Since we start with our robot in motionless state with orientation corresponding to $\theta = 0$ (and since $\sin 0 = 0$), we will have zero components in the observation matrix for all the components relative to the motion along the y-axis, therefore the update phase will not reduce the uncertainty relative to these components as it does for the ones relative to the motion along the x-axis. This leads to high uncertainty on the y-axis even if the robot is stationary and we observe no wheel rotations. This is certainly a non-desirable effect.

[3]The Taylor expansion to the first order for a function $f(x, y)$ results in the polynomial:

$$P = f(0,0) + \frac{\partial f(0,0)}{\partial x}x + \frac{\partial f(0,0)}{\partial y}y$$

Converted Measurements Observation Matrix

In order to solve the problem previously described, we have changed our approach to the encoders measurements. This time we have not considered as observations directly the wheel rotations, but we converted the wheel rotations in the local instant movement and used this movement as the observation.

As described in my thesis[1] we can calculate the local movement using the Differential Drive technique:

$$\Delta x = \frac{r+l}{2}\frac{\sin\Delta\theta}{\Delta\theta} \tag{3.24}$$

$$\Delta y = \frac{r+l}{2}\frac{1-\cos\Delta\theta}{\Delta\theta} \tag{3.25}$$

$$\Delta\theta = \frac{r-l}{b} \tag{3.26}$$

Since we assume the encoders to be very accurate, we can still believe these observations as reliable. However we must point out that the encoders are subject to errors when the wheels start to slip; in this case the highest drift will be seen in the orientation because calculating Δx, Δy we use the mean between r and l while updating $\Delta\theta$ we use the difference between the two wheels rotations. Therefore we will give a slightly higher covariance to $\Delta\theta$.

$$\Sigma_{encoders,converted} = \begin{pmatrix} 0.0000001 & 0 & 0 \\ 0 & 0.0000001 & 0 \\ 0 & 0 & 0.000001 \end{pmatrix} \tag{3.27}$$

Using the local movement $\Delta\theta$, Δx, Δy as the observation, we can define the observation matrix as follows.

$$\Delta x_{local} = \Delta x\cos\theta + \Delta y\sin\theta \tag{3.28}$$

$$\Delta y_{local} = -\Delta x\sin\theta + \Delta y\cos\theta \tag{3.29}$$

$$\Delta\theta_{local} = \Delta\theta \tag{3.30}$$

Using again the 3.18, the 3.19 and the 3.20, we can rewrite the previous equations as follows:

$$\Delta x_{local} = (\cos\theta\Delta t)\dot{x} + (.5\cos\theta\Delta t^2)\ddot{x} + (\sin\theta\Delta t)\dot{y} + (.5\sin\theta\Delta t^2)\ddot{y} \tag{3.31}$$

$$\Delta y_{local} = (-\sin\theta\Delta t)\dot{x} + (-.5\sin\theta\Delta t^2)\ddot{x} + (\cos\theta\Delta t)\dot{y} + (.5\cos\theta\Delta t^2)\ddot{y} \tag{3.32}$$

$$\Delta\theta_{local} = \dot{\theta}\Delta t \tag{3.33}$$

Finally we can write the converted measurements observation matrix:

$$C_{encoders,converted} = \begin{pmatrix} 0 & \cos\theta\Delta t & .5\cos\theta\Delta t^2 & 0 & \sin\theta\Delta t & .5\sin\theta\Delta t^2 & 0 & 0 \\ 0 & -\sin\theta\Delta t & -.5\sin\theta\Delta t^2 & 0 & \cos\theta\Delta t & .5\cos\theta\Delta t^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t \end{pmatrix} \tag{3.34}$$

3.2.3 Joint Update phase

Since we are sampling both the sensors at the same rate (100 Hz), we will consider a single observation vector:

- $\ddot{x}_z$ which is the translational acceleration along the x-axis measured by the accelerometer,
- $\ddot{y}_z$ which is the translational acceleration along the y-axis measured by the accelerometer,
- $\dot{\theta}_z$ which is the rotational velocity along the z-axis measured by the gyroscope,
- Δx_{local} which is the local displacement along the x-axis calculated from the encoders rotations,
- Δy_{local} which is the local displacement along the y-axis calculated from the encoders rotations,
- $\Delta\theta_{local}$ which is the local orientation change calculated from the encoders rotations.

Globally the observation covariance will be:

$$\Sigma_z = \begin{pmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.000001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0000001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0000001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.000001 \end{pmatrix} \tag{3.35}$$

and the observation matrix will be:

$$C_t = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \cos\theta\Delta t & .5\cos\theta\Delta t^2 & 0 & \sin\theta\Delta t & .5\sin\theta\Delta t^2 & 0 & 0 \\ 0 & -\sin\theta\Delta t & -.5\sin\theta\Delta t^2 & 0 & \cos\theta\Delta t & .5\cos\theta\Delta t^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t \end{pmatrix} \tag{3.36}$$

We can notice how this matrix depends on the current sampling instant, in particular it depends on the current orientation. Therefore we will need to recalculate the observation matrix at each iteration of the filter.

Chapter 4

Conclusions

In this report we have presented the Kalman filter we used for our robot. Now we will report an evaluation test on the accuracy af the filter with respect to the odometries calculated using sensors individually.

We drove the robot along the following trajectory starting and ending at point A and passing through the other points in the order B, C, D, E ,F.

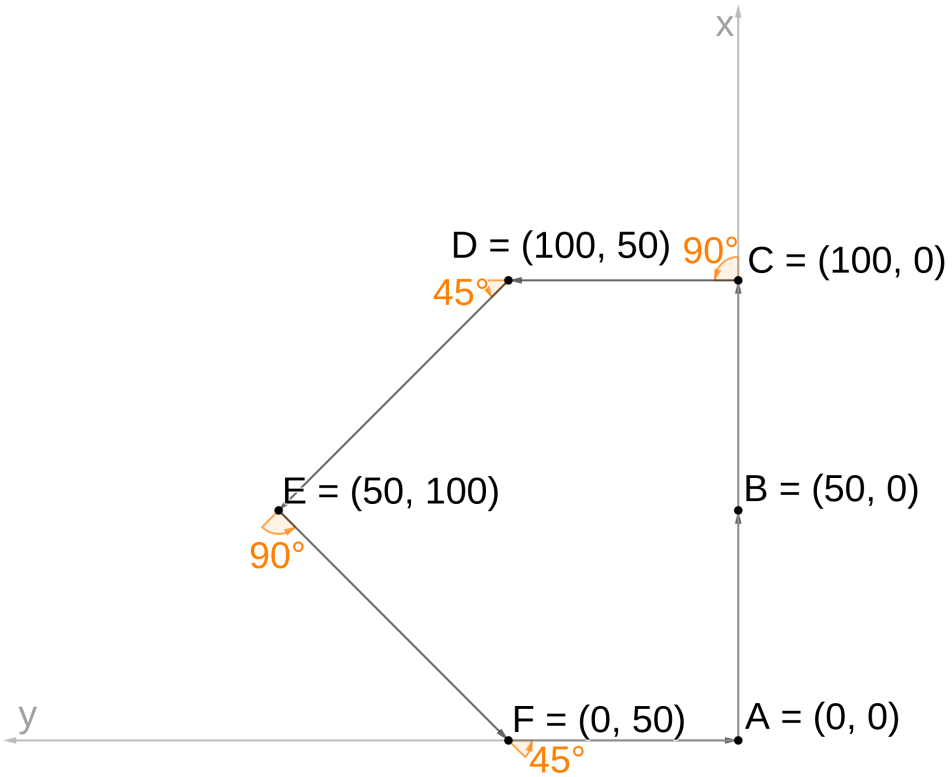

Figure 4.1. Expected trajectory for the test (distances are expressed in cm).

Test 1

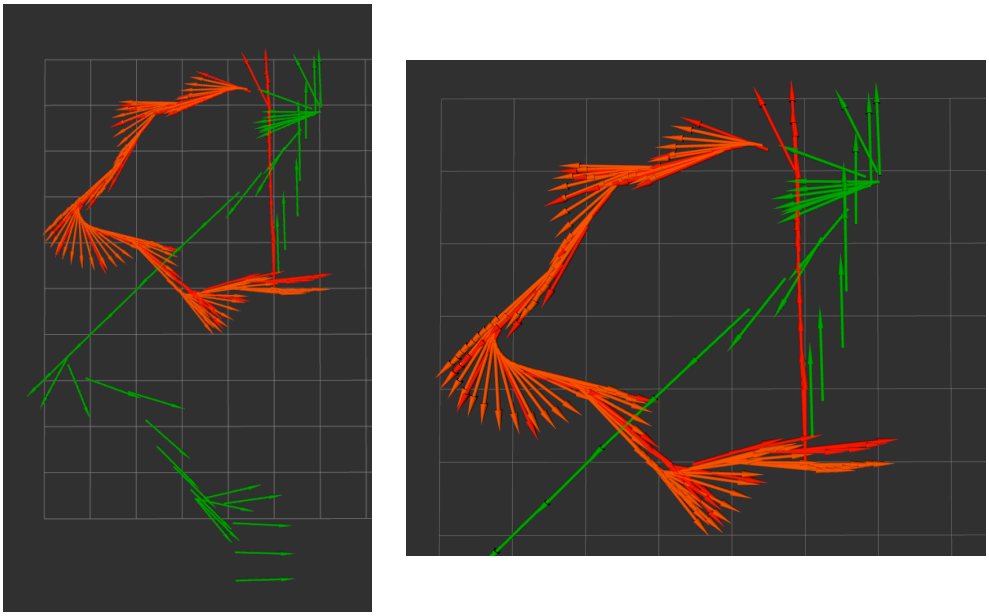

Figure 4.2. Test 1 : estimated trajectories, Kalman filter trajectory in orange, Differential Drive trajectory in red, Inertial Navigation trajectory in green (visualized using RViz).

	Expected Position	Kalman Filter	Differential Drive	Inertial Navigation
A	(0.000, 0.000, 0.000)	(0.000, 0.000, 0.000)	(0.000, 0.000, 0.000)	(0.000, 0.000, 0.000)
B	(0.500, 0.000, 0.000)	(0.501, 0.009, 0.011)	(0.498, 0.016, 0.022)	(0.679, -0.141, 0.011)
C	(1.000, 0.000, 0.000)	(1.002, 0.017, 0.010)	(0.999, 0.029, 0.011)	(0.991, -0.252, 0.010)
D	(1.000, 0.500, 1.571)	(1.012, 0.523, 1.552)	(1.001, 0.534, 1.566)	(0.960, -0.256, 1.567)
E	(0.500, 1.000, 2.356)	(0.509, 1.031, 2.325)	(0.481, 1.026, 2.349)	(-0.364, 1.117, 2.344)
F	(0.000, 0.500, 3.927)	(-0.012, 0.518, 3.840)	(0.001, 0.479, 3.915)	(-1.136, 0.433, 3.866)
A	(0.000, 0.000, 4.712)	(-0.009, 0.017, 4.721)	(0.046, -0.020, 4.807)	(-1.923, 0.205, 4.762)

Table 4.1. Test 1 : values for all the localization techniques [expressed as (x, y, theta)].

By way of example we also report the final state[1] and covariance, calculated by the Kalman filter, at the end of the path after 60 seconds of motionless.

[1]The state is expressed in the form $\begin{pmatrix} x & \dot{x} & \ddot{x} & y & \dot{y} & \ddot{y} & \theta & \dot{\theta} \end{pmatrix}$

$$x = \begin{pmatrix} -8.78e^{-3} & -1.14e^{-2} & -1.33e^{-2} & 1.67e^{-2} & 1.05e^{-2} & 1.24e^{-2} & 4.71 & -9.31e^{-4} \end{pmatrix}$$

$$\Sigma_x = \begin{pmatrix} 2.67e^{-2} & 9.80e^{-5} & -2.37e^{-6} & 0 & 0 & 0 & 0 & 0 \\ 9.80e^{-5} & 2.01e^{-4} & 2.34e^{-4} & 0 & 0 & 0 & 0 & 0 \\ -2.37e^{-6} & 2.34e^{-4} & 6.42e^{-4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.67e^{-2} & 1.00e^{-4} & 1.07e^{-8} & 0 & 0 \\ 0 & 0 & 0 & 1.00e^{-4} & 2.06e^{-4} & 2.41e^{-4} & 0 & 0 \\ 0 & 0 & 0 & 1.08e^{-8} & 2.41e^{-4} & 6.52e^{-4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.69e^{-7} & 2.51e^{-7} \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.51e^{-7} & 5.01e^{-5} \end{pmatrix}$$

Test 2

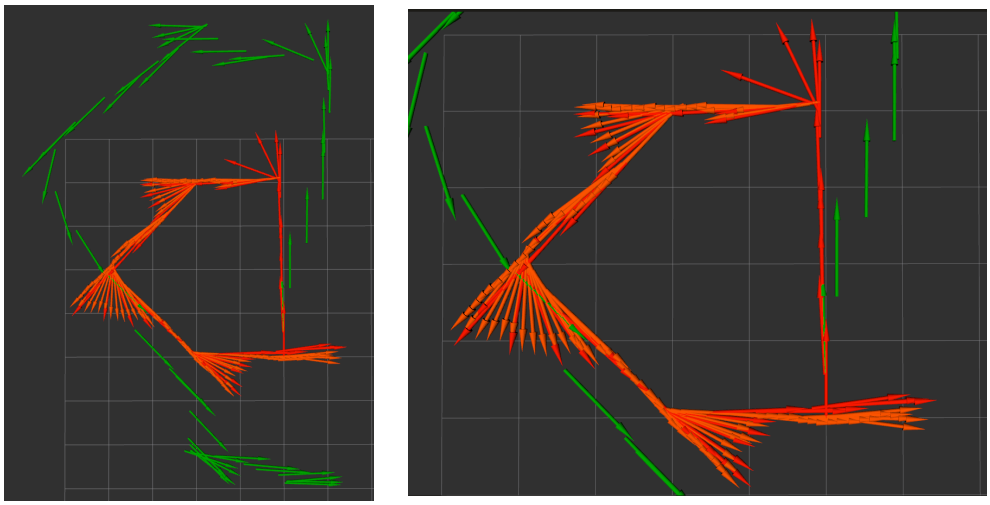

Figure 4.3. Test 2 : estimated trajectories, Kalman filter trajectory in orange, Differential Drive trajectory in red, Inertial Navigation trajectory in green (visualized using RViz).

	Expected Position	Kalman Filter	Differential Drive	Inertial Navigation
A	(0.000, 0.000, 0.000)	(0.000, 0.000, 0.000)	(0.000, 0.000, 0.000)	(0.000, 0.000, 0.000)
B	(0.500, 0.000, 0.000)	(0.499, 0.000, 0.007)	(0.498, 0.012, 0.021)	(0.973, -0.224, -0.002)
C	(1.000, 0.000, 0.000)	(1.001, 0.000, 0.006)	(0.980, 0.021, 0.076)	(1.528, -0.258, 0.056)
D	(1.000, 0.500, 1.571)	(1.010, 0.501, 1.558)	(0.992, 0.502, 1.577)	(1.896, 0.455, 1.539)
E	(0.500, 1.000, 2.356)	(0.516, 0.973, 2.348)	(0.524, 0.975, 2.367)	(-0.364, 1.117, 2.324)
F	(0.000, 0.500, 3.927)	(0.006, 0.506, 3.906)	(-0.005, 0.481, 3.969)	(-0.516, 0.531, 3.911)
A	(0.000, 0.000, 4.712)	(0.003, 0.000, 4.693)	(0.038, -0.025, 4.785)	(-0.707, -0.013, 4.691)

Table 4.2. Test 2 : values for all the localization techniques [expressed as (x, y, theta)].

By way of example we also report the final state and covariance, calculated by the Kalman filter, at the end of the path after 60 seconds of motionless.

$$x = \begin{pmatrix} 3.67e^{-3} & -3.57e^{-3} & -7.78e^{-3} & -2.38e^{-3} & 3.45e^{-3} & 7.69e^{-3} & 4.66 & 1.44e^{-4} \end{pmatrix}$$

$$\Sigma_x = \begin{pmatrix} 1.31e^{-3} & 9.60e^{-6} & -8.84e^{-7} & 0 & 0 & 0 & 0 & 0 \\ 9.60e^{-6} & 3.99e^{-5} & 8.70e^{-5} & 0 & 0 & 0 & 0 & 0 \\ -8.84e^{-7} & 8.70e^{-5} & 4.07e^{-4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.31e^{-3} & 9.60e^{6} & -8.84e^{-7} & 0 & 0 \\ 0 & 0 & 0 & 9.60e^{-6} & 3.99e^{-5} & 8.70e^{-5} & 0 & 0 \\ 0 & 0 & 0 & -8.84e^{-7} & 8.70e^{-5} & 6.52e^{-4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.31e^{-6} & 5.1e^{-9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 5.1e^{-9} & 9.81e^{-7} \end{pmatrix}$$

From these tests it emerges how the trajectory obtained with the sensor fusion (implemented through Kalman filtering) is surely far more accurate than the Inertial Navigation trajectory and it is also more accurate than the Differential Drive one. Moreover it also emerges how after a period of motionless the state is stable and the filter is very confident.

Appendix A

Kalman Filter Implementation

In this appendix we present the implementation of the Kalman filter.
Since all the Kalman filter is based on matrices, we needed to implement a few primitives to work with matrices and vectors which are presented in the first section. Then we will list the parameters and the global variables of the filter; and finally we wil show the Predict phase and the Update phase.

A.1 Matrix Operations

```
1   /* VECTORS */
2
3   void vector_print(int size, float v[]);
4
5   //adds two vectors of size len
6   void vector_add(int len, float _a[], float _b[], float dest[]);
7
8   //subtracts two vectors of size len
9   void vector_sub(int len, float _a[], float _b[], float dest[]);
10
11  /* MATRIXES */
12
13  void matrix_print(int rows, int cols, float m[][cols]);
14
15  //set to Identity matrix
16  void matrix_set_identity(int size, float m[][size]);
17
18  //calculates the transpose of matrix s
19  void matrix_transpose(int rows, int cols, float s[][cols], float d
        ↪ [][rows]);
20
```

```
21  //multiplies matrix per scalar value
22  void matrix_scalar_mul(int rows, int cols, float m[][cols], float k
        ↪ , float d[][cols]);
23
24  //adds two matrices
25  void matrix_add(int rows, int cols, float _a[][cols], float _b[][
        ↪ cols], float dest[][cols]);
26
27  //subtracts two matrices
28  void matrix_sub(int rows, int cols, float _a[][cols], float _b[][
        ↪ cols], float dest[][cols]);
29
30  //calculates the product of two square matrix
31  void square_matrix_product(int size, float _left_m[][size], float
        ↪ _right_m[][size], float dest_m[][size]);
32
33  //calculates the product of two general size matrix [left_m is r1 x
        ↪  c1, right_m is c1 x c2]
34  void matrix_product(int r1, int c1, int c2, float left_m[][c1],
        ↪ float right_m[][c2], float dest_m[][c2]);
35
36  //calculates the product of a matric and a vector
37  void matrix_vector_product(int rows, int cols, float m[][cols],
        ↪ float _v[], float dest[]);
38
39  //calculates the determinant of a matrix
40  float matrix_determinant(int size, float m[][size]);
41
42  //calculates the inverse of matrix m
43  void matrix_inverse(int size, float m[][size], float inv[][size]);
```

A.2 Parameters and Global Variables

```
1  #define KF_ODOMETRY_UPDATE_RATE 100 // Hz
2  float delta_time = 1./KF_ODOMETRY_UPDATE_RATE; //secs
3
4  //covariance parameters
5  #define PROCESS_NOISE_XY_COV 0.1 // m/s^3
6  #define PROCESS_NOISE_THETA_COV 0.5 // rad/s^2
7
8  #define OBSERV_IMU_ACCL_NOISE_COV 0.1
9  #define OBSERV_IMU_GYRO_NOISE_COV 0.000001
10
11 #define OBSERV_ENC_DXY_NOISE_COV 0.0000001
12 #define OBSERV_ENC_DTHETA_NOISE_COV 0.000001
13
14
```

```
15  //KF_STATUS
16  #define KF_STATUS_LEN 8
17
18  //KF_STATUS indexes following
19  #define KF_ODOM_X 0
20  #define KF_TRANS_VEL_X 1
21  #define KF_TRANS_ACCL_X 2
22
23  #define KF_ODOM_Y 3
24  #define KF_TRANS_VEL_Y 4
25  #define KF_TRANS_ACCL_Y 5
26
27  #define KF_ODOM_THETA 6
28  #define KF_ROT_VEL_Z 7
29
30
31  //KF_OBSERVATION
32  #define KF_OBSERVATION_LEN 6
33
34  //KF_OBSERVATION indexes following
35  #define KF_OBS_ENC_DX 0
36  #define KF_OBS_ENC_DY 1
37  #define KF_OBS_ENC_DTHETA 2
38
39  #define KF_OBS_IMU_ACCL_X 3
40  #define KF_OBS_IMU_ACCL_Y 4
41
42  #define KF_OBS_IMU_ROTV_Z 5
43
44
45
46  float dt = delta_time;
47  float dt2_2 = .5*delta_time*delta_time;
48  float dt3_6 = delta_time*delta_time*delta_time / 6.;
49
50  //TRANSITION_MATRIX
51  float TRANSITION_MATRIX[KF_STATUS_LEN][KF_STATUS_LEN] =
52    {{1.,dt,dt2_2,0.,0.,0.,0.,0.},
53    {0.,1.,dt,0.,0.,0.,0.,0.},
54    {0.,0.,1.,0.,0.,0.,0.,0.},
55    {0.,0.,0.,1.,dt,dt2_2,0.,0.},
56    {0.,0.,0.,0.,1.,dt,0.,0.},
57    {0.,0.,0.,0.,0.,1.,0.,0.},
58    {0.,0.,0.,0.,0.,0.,1.,dt},
59    {0.,0.,0.,0.,0.,0.,0.,1.}};
60
61  //TRANSITION_MATRIX_TRANSPOSE
```

```
62  float TRANSITION_MATRIX_TRANSPOSE[KF_STATUS_LEN][KF_STATUS_LEN];
63  matrix_transpose(KF_STATUS_LEN, KF_STATUS_LEN, TRANSITION_MATRIX,
        ↪ TRANSITION_MATRIX_TRANSPOSE);
64
65
66
67  //transition noise will be modelled with 3 components:
68    //[third_derivate_x, third_derivate_y, second_derivate_theta]
69  #define TRANSITION_NOISE_LEN 3 //control covariance
70
71    //PROCESS_NOISE_CONTRIBUTE_MATRIX
72  float process_noise_covariance[TRANSITION_NOISE_LEN][
        ↪ TRANSITION_NOISE_LEN] =
73      {{PROCESS_NOISE_XY_COV, 0., 0.},
74      {0.,PROCESS_NOISE_XY_COV,0.},
75      {0.,0.,PROCESS_NOISE_THETA_COV}};
76
77  float transition_noise_matrix[KF_STATUS_LEN][TRANSITION_NOISE_LEN]
        ↪ =
78      {{dt3_6,0.,0.},
79      {dt2_2,0.,0.},
80      {dt,0.,0.},
81      {0.,dt3_6,0.},
82      {0.,dt2_2,0.},
83      {0.,dt,0.},
84      {0.,0.,dt2_2},
85      {0.,0.,dt}};
86  float transition_noise_matrix_transpose[TRANSITION_NOISE_LEN][
        ↪ KF_STATUS_LEN];
87  matrix_transpose(KF_STATUS_LEN, TRANSITION_NOISE_LEN,
        ↪ transition_noise_matrix, transition_noise_matrix_transpose);
88
89  float aux_cov[KF_STATUS_LEN][TRANSITION_NOISE_LEN];
90  matrix_product(KF_STATUS_LEN, TRANSITION_NOISE_LEN,
        ↪ TRANSITION_NOISE_LEN, transition_noise_matrix,
        ↪ process_noise_covariance, aux_cov);
91  float PROCESS_NOISE_CONTRIBUTE_MATRIX[KF_STATUS_LEN][KF_STATUS_LEN
        ↪ ];
92  matrix_product(KF_STATUS_LEN, TRANSITION_NOISE_LEN, KF_STATUS_LEN,
        ↪ aux_cov, transition_noise_matrix_transpose,
        ↪ PROCESS_NOISE_CONTRIBUTE_MATRIX);
93
94
95  //OBSERVATION_COVARIANCE_MATRIX
96  float OBSERVATION_COVARIANCE_MATRIX[KF_OBSERVATION_LEN][
        ↪ KF_OBSERVATION_LEN] =
97    {{OBSERV_ENC_DXY_NOISE_COV,0.,0.,0.,0.},
```

```
98      {0.,OBSERV_ENC_DXY_NOISE_COV,0.,0.,0.,0.},
99      {0.,0.,OBSERV_ENC_DTHETA_NOISE_COV,0.,0.,0.},
100     {0.,0.,0.,OBSERV_IMU_ACCL_NOISE_COV,0.,0.},
101     {0.,0.,0.,0.,OBSERV_IMU_ACCL_NOISE_COV,0.},
102     {0.,0.,0.,0.,0.,OBSERV_IMU_GYRO_NOISE_COV}};
```

Listing A.1. Parameters of the Kalman Filter

```
1   //STATE
2   float status_mean[KF_STATUS_LEN];
3   float status_covariance[KF_STATUS_LEN][KF_STATUS_LEN];
4
5   //OBSERVATION_MATRIX
6   float observation_matrix[KF_OBSERVATION_LEN][KF_STATUS_LEN];
```

Listing A.2. Global Variables of the Kalman Filter

A.3 Predict Phase

```
1   //mean = A*mean
2   matrix_vector_product(KF_STATUS_LEN, KF_STATUS_LEN,
        ↪ TRANSITION_MATRIX, status_mean, status_mean);
3
4   //covariance = A*Cov*A_t
5   square_matrix_product(KF_STATUS_LEN, TRANSITION_MATRIX,
        ↪ status_covariance, status_covariance);
6   square_matrix_product(KF_STATUS_LEN, status_covariance,
        ↪ TRANSITION_MATRIX_TRANSPOSE, status_covariance);
7
8   //covariance += B*Cov_noise*B_t
9   matrix_add(KF_STATUS_LEN, KF_STATUS_LEN, status_covariance,
        ↪ PROCESS_NOISE_CONTRIBUTE_MATRIX, status_covariance);
```

A.4 Update Phase

```
1   //note: obs is an array of KF_OBSERVATION_LEN elements
2   float obs[KF_OBSERVATION_LEN];
3   obs[KF_OBS_ENC_DX] = sens_obs->local_dx;
4   obs[KF_OBS_ENC_DY] = sens_obs->local_dy;
5   obs[KF_OBS_ENC_DTHETA] = sens_obs->local_dtheta;
6   obs[KF_OBS_IMU_ACCL_X] = sens_obs->imu_accel_x;
7   obs[KF_OBS_IMU_ACCL_Y] = sens_obs->imu_accel_y;
8   obs[KF_OBS_IMU_ROTV_Z] = sens_obs->imu_vel_theta;
9
10
11  float sin_theta = sin(status_mean[KF_ODOM_THETA]);
12  float cos_theta = cos(status_mean[KF_ODOM_THETA]);
```

```
13
14  float dt2_2 = .5*delta_time*delta_time;
15  float dt = delta_time;
16
17  memset(&observation_matrix, 0, sizeof(observation_matrix));
18
19  //ENCODERS
20  observation_matrix[KF_OBS_ENC_DX][KF_TRANS_VEL_X] = cos_theta*dt;
21  observation_matrix[KF_OBS_ENC_DX][KF_TRANS_ACCL_X] = cos_theta*
        ↪ dt2_2;
22  observation_matrix[KF_OBS_ENC_DX][KF_TRANS_VEL_Y] = sin_theta*dt;
23  observation_matrix[KF_OBS_ENC_DX][KF_TRANS_ACCL_Y] = sin_theta*
        ↪ dt2_2;
24
25  observation_matrix[KF_OBS_ENC_DY][KF_TRANS_VEL_X] = -sin_theta*dt;
26  observation_matrix[KF_OBS_ENC_DY][KF_TRANS_ACCL_X] = -sin_theta*
        ↪ dt2_2;
27  observation_matrix[KF_OBS_ENC_DY][KF_TRANS_VEL_Y] = cos_theta*dt;
28  observation_matrix[KF_OBS_ENC_DY][KF_TRANS_ACCL_Y] = cos_theta*
        ↪ dt2_2;
29
30  observation_matrix[KF_OBS_ENC_DTHETA][KF_ROT_VEL_Z] = dt;
31
32
33  // IMU
34  //nel reference frame del robot:
35    // accel_x_local = accel_x_global * cos_theta + accel_y_global *
          ↪ sin_theta
36    // accel_y_local = -accel_x_global * sin_theta + accel_y_global *
          ↪  cos_theta
37  observation_matrix[KF_OBS_IMU_ACCL_X][KF_TRANS_ACCL_X] = cos_theta;
38  observation_matrix[KF_OBS_IMU_ACCL_X][KF_TRANS_ACCL_Y] = sin_theta;
39
40  observation_matrix[KF_OBS_IMU_ACCL_Y][KF_TRANS_ACCL_X] = -sin_theta
        ↪ ;
41  observation_matrix[KF_OBS_IMU_ACCL_Y][KF_TRANS_ACCL_Y] = cos_theta;
42
43  observation_matrix[KF_OBS_IMU_ROTV_Z][KF_ROT_VEL_Z] = 1.;
44
45  //attended observation = C*status_mean
46  float attended_obs[KF_OBSERVATION_LEN];
47  matrix_vector_product(KF_OBSERVATION_LEN, KF_STATUS_LEN,
        ↪ observation_matrix, status_mean, attended_obs);
48
49  //K = cov_status*C_T * inverse(cov_noise + C*cov_status*C_T)
50  float K_matrix[KF_STATUS_LEN][KF_OBSERVATION_LEN];
51
```

```
52  float obs_matrix_transpose[KF_STATUS_LEN][KF_OBSERVATION_LEN];
53  matrix_transpose(KF_OBSERVATION_LEN, KF_STATUS_LEN,
        ↪ observation_matrix, obs_matrix_transpose);
54
55  float aux_m_5_8[KF_OBSERVATION_LEN][KF_STATUS_LEN];
56  matrix_product(KF_OBSERVATION_LEN, KF_STATUS_LEN, KF_STATUS_LEN,
        ↪ observation_matrix, status_covariance, aux_m_5_8);
57
58  float aux_m_5_5[KF_OBSERVATION_LEN][KF_OBSERVATION_LEN];
59  matrix_product(KF_OBSERVATION_LEN, KF_STATUS_LEN,
        ↪ KF_OBSERVATION_LEN, aux_m_5_8, obs_matrix_transpose,
        ↪ aux_m_5_5);
60
61  matrix_add(KF_OBSERVATION_LEN, KF_OBSERVATION_LEN,
        ↪ OBSERVATION_COVARIANCE_MATRIX, aux_m_5_5, aux_m_5_5);
62
63  matrix_inverse(KF_OBSERVATION_LEN, aux_m_5_5, aux_m_5_5);
64
65  float aux_m_8_5[KF_STATUS_LEN][KF_OBSERVATION_LEN];
66  matrix_product(KF_STATUS_LEN, KF_OBSERVATION_LEN,
        ↪ KF_OBSERVATION_LEN, obs_matrix_transpose, aux_m_5_5,
        ↪ aux_m_8_5);
67
68  matrix_product(KF_STATUS_LEN, KF_STATUS_LEN, KF_OBSERVATION_LEN,
        ↪ status_covariance, aux_m_8_5, K_matrix);
69
70  // status_mean += K_matrix * (obs - attended_obs)
71  float delta_obs[KF_OBSERVATION_LEN];
72  vector_sub(KF_OBSERVATION_LEN, obs, attended_obs, delta_obs);
73
74  float delta_status[KF_STATUS_LEN];
75  matrix_vector_product(KF_STATUS_LEN, KF_OBSERVATION_LEN, K_matrix,
        ↪ delta_obs, delta_status);
76
77  vector_add(KF_STATUS_LEN, status_mean, delta_status, status_mean);
78
79  //status_covariance -= (K_matrix*C ) * status_covariance
80  float aux_m_8_8[KF_STATUS_LEN][KF_STATUS_LEN];
81  matrix_product(KF_STATUS_LEN, KF_OBSERVATION_LEN, KF_STATUS_LEN,
        ↪ K_matrix, observation_matrix, aux_m_8_8);
82
83  square_matrix_product(KF_STATUS_LEN, aux_m_8_8, status_covariance,
        ↪ aux_m_8_8);
84
85  matrix_sub(KF_STATUS_LEN, KF_STATUS_LEN, status_covariance,
        ↪ aux_m_8_8, status_covariance);
```

References

[1] F. Graziano, "Differential drive and inertial navigation robot", Bachelor's thesis, Sapienza University of Rome, Jul. 2019. [Online]. Available: https://github.com/FilippoGraziano98/Acquisition_Platform/blob/master/docs/thesis.pdf.

[2] G. Grisetti, *Probabilistic Robotics Course*. Sapienza University of Rome. [Online]. Available: https://gitlab.com/grisetti/probabilistic_robotics_2018_19.