

Nonlinear attitude filtering and estimators for SLAM

TPK4560 - Robotics and Automation, Specialization Project
Thesis

Thilogen Thambirajah

2021-12-20

Abstract

In this report, the MEKF and RIEKF, both of the filters being a modified version of the EKF, are compared relative to nonlinear attitude filtering, where the RIEKF proves to perform slightly better than the MEKF.

Simultaneous localization and mapping (SLAM) are used for autonomous vehicles to localize the vehicle and at the same time identify objects or obstacles in the surroundings. The EKF-based SLAM has been a big part of SLAM history, and numerous studies have concluded the EKF to be prone to inconsistency. In this report, the inconsistency of the EKF-based SLAM is also addressed. A modified version of the EKF, the IEKF consisting of a nonlinear error variable, proved to significantly improve the inconsistency, which explains the origin of the consistency issues. Another method presented for SLAM is the analytical SLAM-DUNK; A type of SLAM without linearization that managed to converge as accurately as the IEKF and thus outperform the EKF.

Contents

Abstract	i
1. Introduction	1
2. Nonlinear Attitude Filtering	2
2.1. Multiplicative Extended Kalman Filter - MEKF	2
2.1.1. Time propagation of state estimate	3
2.1.2. Error equations and linearization of error quaternions	4
2.1.3. Covariance propagation	6
2.1.4. MEKF Algorithm	7
2.2. Right Invariant Extended Kalman Filter - RIEKF	7
2.2.1. Time propagation of state estimate	8
2.2.2. Error equations and linearization of error quaternions	8
2.2.3. Covariance propagation	10
2.2.4. RIEKF Algorithm	11
3. SLAM	12
3.1. EKF SLAM	12
3.1.1. Error equations and linearization of error equations	14
3.1.2. Kalman gain and estimates	16
3.1.3. EKF SLAM Algorithm	17
3.1.4. Consistency issues of the EKF SLAM	18
3.2. Lie groups in SLAM	19
3.3. SE(2) as a Lie group	19
3.4. IEKF SLAM	20
3.4.1. Error equations and linearization of error equations	22
3.4.2. Kalman gain and estimates	24
3.4.3. IEKF SLAM Algorithm	25
3.5. Analytical SLAM	26
3.5.1. Decoupled Unlinearized Networked Kalman filter - DUNK .	27
3.5.2. Implementation of the SLAM-DUNK	27
3.5.3. SLAM-DUNK Algorithm	30

- 4. Simulation** **31**
 - 4.1. Nonlinear attitude filtering 31
 - 4.2. SLAM 33

- 5. Results & Discussions** **35**
 - 5.1. Nonlinear attitude filtering 35
 - 5.1.1. Error plots 38
 - 5.1.2. Discussion 39
 - 5.2. SLAM 40
 - 5.2.1. Discussion 44

- 6. Conclusion** **45**

- A. Code listing** **48**
 - A.1. MEKF and RIEKF implementation for nonlinear attitude filtering 48
 - A.2. EKF- and IEKF SLAM implementation 53
 - A.3. Analytical SLAM implementation 61

List of Figures

3.1. Visualization of the SLAM-DUNK with virtual vehicles and consensus [10]	27
5.1. Simulated angle rotation results for the MEKF	35
5.2. Simulated angle rotation results for the RIEKF	36
5.3. Comparison of convergence time for the rotation angle estimation .	36
5.4. Simulated bias results for the MEKF	37
5.5. Simulated bias results for the RIEKF	37
5.6. Comparison of convergence time for the bias estimation	38
5.7. Error of rotation angle estimation plotted against the time	38
5.8. Error of bias estimation plotted against the time	38
5.9. Simulated results for the EKF SLAM with confidence ellipse . . .	40
5.10. Simulated results for the IEKF SLAM with confidence ellipse . . .	41
5.11. Simulated results for the analytical SLAM-DUNK	42
5.12. Simulated results for the EKF SLAM with landmark trajectories and initial landmark positions	43
5.13. Simulated results for the IEKF SLAM with landmark trajectories and initial landmark positions	43
5.14. Simulated results for the analytical SLAM-DUNK with landmark trajectories and initial landmark positions	44

List of Tables

4.1. Nonlinear attitude filtering: Parameters and their respective values for the system initialization	32
4.2. Nonlinear attitude filtering: Parameters and their respective values for the noise initialization and coefficient matrices	32
4.3. SLAM: Parameters and their respective values for the system initialization of the EKF and IEKF	33
4.4. SLAM: Parameters and their respective values for the system initialization for the analytical SLAM-DUNK	34
4.5. SLAM: Noise initialization and covariance matrix parameters and their respective values for the EKF- and IEKF SLAM	34
4.6. SLAM: Noise initialization and covariance matrix parameters and their respective values of the analytical SLAM-DUNK	34

Chapter 1.

Introduction

This report is a literature study of various filters used in nonlinear attitude filtering and simultaneous localization and mapping (SLAM). The filters studied relative to SLAM are the Extended Kalman Filter (EKF) and the Invariant Extended Kalman Filter (IEKF), which Bonnabel presents in [1], and SLAM without linearization, also called analytical SLAM-DUNK, which Slotine presents in [10]. For the study relative to the nonlinear attitude filtering, the filters studied are the Multiplicative Extended Kalman Filter (MEKF), which F. Landis Markley, NASA Goddard Space Flight Center, presents in [9] and [6], and the Right Invariant Extended Kalman Filter (RIEKF), which Bonnabel presents in [1]. Zamani, Trumpf, and Mahony present the comparison study for nonlinear attitude filtering in [11].

This report will introduce the equations necessary to implement the filters, followed by a simple algorithm and a brief theoretical background. In addition, the consistency issues of the EKF relative to SLAM and how the RIEKF and the analytical SLAM-DUNK avoid/improve the inconsistency are explained. Then the report will present a simplified simulation of the filters created based on the papers mentioned above, where the results are compared and discussed.

Chapter 2.

Nonlinear Attitude Filtering

2.1. Multiplicative Extended Kalman Filter - MEKF

The MEKF is a modification of the Extended Kalman Filter (EKF) to estimate a three-component attitude error. A quaternion defines the error, and a unit quaternion parameterizes the global attitude [9]. The MEKF error quaternion results in an identity quaternion using an invariant output error termed left-invariant error instead of a linear error used for the EKF [9].

The presented equations and algorithm for the MEKF in this chapter are mainly based and inspired by [6] and [9].

It is noted that in this project thesis, the unit quaternion $q = \eta + \epsilon$ is equivalent to the rotation matrix $R = I + 2\eta\epsilon + \epsilon^\times\epsilon^\times$.

Since q is a unit quaternion, the kinematic equation can be written in the form of

$$\dot{q} = \frac{1}{2}q \circ \omega \quad (2.1)$$

where ω corresponds to the angular velocity of the attitude. Following Farrenkopf's model [3], the gyroscope measurement ω_m is expressed as

$$\omega_m(t) = \omega(t) + b(t) + n_1(t) \quad (2.2)$$

and the bias model as

$$\dot{b} = n_2 \quad (2.3)$$

Combining equation (2.1) and (2.2) presents the state propagation

$$\dot{q} = \frac{1}{2}q \circ (\omega_m - b - n_1) \quad (2.4)$$

$$\dot{b} = n_2 \quad (2.5)$$

2.1.1. Time propagation of state estimate

Given an angular velocity measured by a gyroscope, ω_m , the estimated angular velocity is expressed as

$$\hat{\omega} = \omega_m - \hat{b} \quad (2.6)$$

The estimate of the measurement i is expressed by the estimated rotation matrix, \hat{R} , and is given as

$$\hat{y}_i = \hat{R}^T d_i^n \quad (2.7)$$

where d_i is a known vector in the spatial frame n , e.g the earth magnetic field in NED coordinates as it is described in [2], and is not to be confused with "d" further in this text. It is noted that \hat{R} is defined by the components of \hat{q} .

The estimate of the gyroscope bias is updated by the following formula

$$\hat{b}_{k|k-1} = \hat{b}_{k-1|k-1} + h \left(d\hat{b} \right) \quad (2.8)$$

where

$$d\hat{b} = K_b \tilde{y} = P_{c,k-1|k-1}^T C_{ai}^T R_c^{-1} \tilde{y} = P_{c,k-1|k-1}^T \delta \quad (2.9)$$

The update of the state estimate is

$$\hat{q}_{k|k-1} = \hat{q}_{k-1|k-1} \circ \exp \left(0.5h \left(\hat{\omega} + P_{a,k-1|k-1} \delta \right) \right) \quad (2.10)$$

2.1.2. Error equations and linearization of error quaternions

The error equations for the MEKF is presented as

$$\tilde{q} = \hat{q}^{-1} \circ q \quad (2.11)$$

$$\tilde{b} = b - \hat{b} \quad (2.12)$$

Then the error dynamics of attitude can be described by the kinematic differential equation of the error quaternion, and is presented as

$$\dot{\tilde{q}} = \hat{q}^{-1} \circ \dot{q} + \frac{d}{dt} \left(\hat{q}^{-1} \right) \circ q \quad (2.13)$$

Following the article written by Silvère Bonnabel in 2009 [2] it is mentioned that if q depends on time, then $\dot{q}^{-1} = -q^{-1} * \dot{q} * q^{-1}$. This gives

$$\frac{d}{dt} \left(\hat{q}^{-1} \right) = -\hat{q}^{-1} \circ \dot{\hat{q}} \circ \hat{q}^{-1} \quad (2.14)$$

By combining equation (2.13) and (2.14), the differential equation of the error matrix is seen as

$$\begin{aligned} \dot{\tilde{q}} &= \hat{q}^{-1} \circ \dot{q} - \hat{q}^{-1} \circ \dot{\hat{q}} \circ \hat{q}^{-1} \circ q \\ &= \hat{q}^{-1} \circ \frac{1}{2} q \circ (\omega_m - b - n_1) - \hat{q}^{-1} \circ \frac{1}{2} \dot{\hat{q}} \circ (\omega_m - \hat{b} + K_q \tilde{y}) \circ \tilde{q} \\ &= \frac{1}{2} \tilde{q} \circ (\omega_m - (\hat{b} + \tilde{b}) - n_1) - \frac{1}{2} (\omega_m - \hat{b} + K_q \tilde{y}) \circ \tilde{q} \\ &= \tilde{\epsilon}^\times \hat{\omega} - \underbrace{\frac{1}{2} (\tilde{\epsilon}^T) (-\tilde{b} + K_q \tilde{y} + n_1)}_{\text{Scalar}} + \frac{1}{2} \tilde{\eta} (-\tilde{b} - K_q \tilde{y} + n_1) + \frac{1}{2} \tilde{\epsilon}^\times (-\tilde{b} + K_q \tilde{y} + n_1) \end{aligned} \quad (2.15)$$

From the resulting equation, the vector part of the error equation then presents the following kinematic differential equation

$$\dot{\tilde{\epsilon}} = \tilde{\epsilon}^\times \hat{\omega} + \frac{1}{2} \tilde{\eta} (-\tilde{b} - K_q \tilde{y} + n_1) + \frac{1}{2} \tilde{\epsilon}^\times (-\tilde{b} + K_q \tilde{y} + n_1) \quad (2.16)$$

It is noted that $\tilde{a} = 2\tilde{\epsilon}$ is used as the three parameter representation of the error rotation as long as $\hat{\theta} < \pi$, which is assumed to be the case for all practical implementations.

Given this information, the linearized error dynamics is expressed as

$$\begin{bmatrix} \dot{\tilde{a}} \\ \dot{\tilde{b}} \end{bmatrix} = \begin{bmatrix} -\hat{\omega}^\times & -I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{a} \\ \tilde{b} \end{bmatrix} + \begin{bmatrix} K_q \\ K_b \end{bmatrix} \tilde{y} + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \quad (2.17)$$

and the linearized measurement error equation is expressed as

$$\tilde{y}_i = C_i \begin{bmatrix} \tilde{a} \\ \tilde{b} \end{bmatrix}, \quad C_i = \begin{bmatrix} C_{ai} & 0 \end{bmatrix} = \begin{bmatrix} \hat{y}_i^\times & 0 \end{bmatrix} \quad (2.18)$$

Summarized, the A and C matrices are a result of the linearization of error quaternions and are given as

$$A = \begin{bmatrix} -\hat{\omega}^\times & -I \\ 0 & 0 \end{bmatrix} \quad (2.19)$$

$$C = \begin{bmatrix} \hat{y}_1^\times & 0 \\ \vdots & \\ \hat{y}_m^\times & 0 \end{bmatrix} \quad (2.20)$$

Further, the innovation is described as

$$\delta = \sum_{i=1} \hat{y}_i^\times R_c^{-1} (\hat{y}_i - y_i) \quad (2.21)$$

and the covariance of the innovation S is

$$S = C^T R_C^{-1} C = \sum_{i=1} \left(\hat{y}_i^\times \right)^T R_c^{-1} \hat{y}_i^\times \quad (2.22)$$

2.1.3. Covariance propagation

The covariance matrix follows [11]

$$P = \begin{bmatrix} P_a & P_c \\ P_c^T & P_b \end{bmatrix} \quad (2.23)$$

and consists of the gains P_a , P_b and P_c . These gains are updated from the following equations

$$P_{a,k|k-1} = P_{a,k-1|k-1} + h(dP_a) \quad (2.24)$$

$$P_{b,k|k-1} = P_{b,k-1|k-1} + h(dP_b) \quad (2.25)$$

$$P_{c,k|k-1} = P_{c,k-1|k-1} + h(dP_c) \quad (2.26)$$

where

$$dP_a = 2\mathbb{P} \left(P_{a,k-1|k-1} \hat{\omega}^\times - P_{c,k-1|k-1} \right) + Q_a - P_{a,k-1|k-1} S P_{a,k-1|k-1} \quad (2.27)$$

$$dP_b = Q_b - P_{c,k-1|k-1}^T S P_{c,k-1|k-1} \quad (2.28)$$

$$dP_c = -\hat{\omega}^\times P_{c,k-1|k-1} - P_{b,k-1|k-1} + Q_c - P_{a,k-1|k-1} S P_{c,k-1|k-1} \quad (2.29)$$

2.1.4. MEKF Algorithm

Algorithm 1 MEKF signal Algorithm

Initialize Q_a, Q_b, Q_c and R_C

Initialize q_0, b_0 and P_{a0}, P_{b0}, P_{c0}

loop

$$\hat{\omega} = \omega_m - \hat{b}_{k-1|k-1}$$

$$\hat{y}_i = \hat{R}^T d_i^n$$

$$S = C^T R_C^{-1} C = \sum_{i=1} \left(\hat{y}_i^\times \right)^T R_c^{-1} \hat{y}_i^\times$$

$$\delta = \sum_{i=1} \hat{y}_i^\times R_c^{-1} (\hat{y}_i - y_i)$$

$$\hat{q}_{k|k-1} = \hat{q}_{k-1|k-1} \circ \exp \left(0.5h \left(\hat{\omega} + P_{a,k-1|k-1} \delta \right) \right)$$

$$d\hat{b} = P_{c,k-1|k-1}^T \delta$$

$$\hat{b}_{k|k-1} = \hat{b}_{k-1|k-1} + h \left[d\hat{b} \right]$$

$$dP_a = 2\mathbb{P} \left(P_{a,k-1|k-1} \hat{\omega}^\times - P_{c,k-1|k-1} \right) + Q_a - P_{a,k-1|k-1} S P_{a,k-1|k-1}$$

$$dP_b = Q_b - P_{c,k-1|k-1}^T S P_{c,k-1|k-1}$$

$$dP_c = -\hat{\omega}^\times P_{c,k-1|k-1} - P_{b,k-1|k-1} + Q_c - P_{a,k-1|k-1} S P_{c,k-1|k-1}$$

$$P_{i,k|k-1} = P_{i,k-1|k-1} + h \left[dP_i \right], \quad i \in \{a, b, c\}$$

$$P = \begin{bmatrix} P_a & P_c \\ P_c^T & P_b \end{bmatrix}$$

end loop

2.2. Right Invariant Extended Kalman Filter - RIEKF

The RIEKF is another modification of the EKF and is closely related to the MEKF in that they both use an invariant output error. Unlike the MEKF, the RIEKF uses the right-invariant error, hence its name. The main benefit of this modification is that the matrices A and C are constant on a much more extensive set of trajectories [2], which may lead to better accuracy and less computational power. The presented equations and algorithm for the RIEKF in this chapter are inspired by [2].

Given that the RIEKF and the MEKF both use an invariant output error, the implementation of the RIEKF is to some degree similar to the MEKF. Therefore, the identical formulas for both filters are not included in this chapter, as they have already been specified in chapter 2.1.

The state propagation of the RIEKF is given by

$$\dot{q} = \frac{1}{2}q \circ (\omega_m - b) + n_q \circ q \quad (2.30)$$

$$\dot{b} = q^{-1} \circ n_b \circ q \quad (2.31)$$

2.2.1. Time propagation of state estimate

The estimated measurement i is

$$\hat{y}_i = \hat{q}^{-1} \circ d_i \circ \hat{q} \quad (2.32)$$

Similar to the MEKF, the estimate of the gyroscope bias is defined as

$$\hat{b}_{k|k-1} = \hat{b}_{k-1|k-1} + h \left(d\hat{b} \right) \quad (2.33)$$

but for the RIEKF, the time propagation of the bias estimate is

$$d\hat{b} = \hat{q}^{-1} \circ (K_b E) \circ \hat{q} = \hat{q}^{-1} \circ \left(P_{c,k-1|k-1}^T \delta \right) \circ \hat{q} \quad (2.34)$$

2.2.2. Error equations and linearization of error quaternions

It is noted that considering unit quaternions, the error equation for the left invariant, LIEKF, is

$$\tilde{q}_l = \hat{q}^{-1} \circ q \quad (2.35)$$

which is the same error equation used in the MEKF. Whereas the right invariant, RIEKF, the error equation is given as

$$\tilde{q}_r = q \circ \hat{q}^{-1} \quad (2.36)$$

Then the error equations for the RIEKF is

$$\tilde{q} = q \circ \hat{q}^{-1} \quad (2.37)$$

$$\tilde{b} = q \circ (b - \hat{b}) \circ q^{-1} \quad (2.38)$$

Next, the kinematic differential equation of the error quaternion is

$$\begin{aligned}
\dot{\hat{q}} &= \dot{q} \circ \hat{q}^{-1} - q \circ \hat{q}^{-1} \circ \dot{q} \circ \hat{q}^{-1} \\
&= \frac{1}{2} q \circ (\omega_m - b) \circ \hat{q}^{-1} - q \circ \hat{q}^{-1} \circ \frac{1}{2} \hat{q} \circ (\omega_m - \hat{b}) \circ \hat{q}^{-1} - q \circ \hat{q}^{-1} \circ K_q E \circ \hat{q} \circ \hat{q}^{-1} \\
&= \frac{1}{2} q \circ (\omega_m - b) \hat{q}^{-1} - \frac{1}{2} q \circ (\omega_m - \hat{b}) \hat{q}^{-1} - \tilde{q}^{-1} \circ K_q E \\
&= \underbrace{\frac{1}{2} \tilde{b}^T \tilde{\epsilon}}_{\text{Scalar}} - \frac{1}{2} \tilde{\eta} \tilde{b} - \frac{1}{2} \tilde{b}^\times \tilde{\epsilon} + \underbrace{\tilde{q}^T K_q E}_{\text{Scalar}} - \tilde{\eta} K_q E - \tilde{\epsilon}^\times K_q E
\end{aligned} \tag{2.39}$$

where the measurement estimation error E is

$$E_i = \tilde{q}^{-1} \circ (d_i + w_y) \circ \tilde{q} - d_i \tag{2.40}$$

and the vector part of the equation is then

$$\dot{\tilde{\epsilon}} = -\frac{1}{2} \tilde{\eta} \tilde{b} - \frac{1}{2} \tilde{b}^\times \tilde{\epsilon} - \tilde{\eta} K_q E - \tilde{\epsilon}^\times K_q E \tag{2.41}$$

The kinematic differential equation of the bias error is

$$\begin{aligned}
\dot{\tilde{b}} &= \dot{q} \circ (b - \hat{b}) \circ q^{-1} + q \circ (\dot{b} - \dot{\hat{b}}) \circ q^{-1} - q \circ (b - \hat{b}) \circ q^{-1} \circ \dot{q} \circ q^{-1} \\
&= \frac{1}{2} q \circ \hat{\omega} \circ (b - \hat{b}) \circ q^{-1} + q \circ \left(q^{-1} M_\omega w_\omega \circ q - \hat{q}^{-1} \circ K_\omega E \circ \hat{q} \right) \circ q^{-1} \\
&\quad - \frac{1}{2} q \circ (b - \hat{b}) \circ q^{-1} \circ q \circ \hat{\omega} \circ q^{-1} \\
&= \frac{1}{2} \left(\tilde{q} \circ \left(\hat{q} \circ \hat{\omega} \circ \hat{q}^{-1} \right) \circ \tilde{q}^{-1} \right)^\times \tilde{b} + M_\omega w - \tilde{q} \circ K_\omega E \circ \tilde{q}^{-1}
\end{aligned} \tag{2.42}$$

using the common state variable $\tilde{a} = 2\tilde{\epsilon}$ the linearization of the error equations gives

$$A = \begin{bmatrix} 0 & -I \\ 0 & \hat{\Omega}^\times \end{bmatrix} \tag{2.43}$$

$$C_i = \begin{bmatrix} d_i^\times & 0 \end{bmatrix} \tag{2.44}$$

It is also noted that C is constant and that A consists only of $\hat{\Omega}^\times$ which is not constant, as $\hat{\Omega} = \hat{q} \circ \hat{\omega} \circ \hat{q}^{-1}$.

Next, the innovation is

$$\delta = \sum_{i=1} \left(d_i^\times \right)^\top R_c^{-1} \left(d_i - \hat{q} \circ y_i \circ \hat{q}^{-1} \right) \quad (2.45)$$

where the covariance of the innovation is

$$S = C^\top R_C^{-1} C = \sum_{i=1} \left(d_i^\times \right)^\top R_c^{-1} d_i^\times \quad (2.46)$$

2.2.3. Covariance propagation

Finally the time propagation of the gains are expressed as

$$dP_a = -2\mathbb{P} \left(P_{c,k-1|k-1} \right) + Q_a - P_{a,k-1|k-1} S P_{a,k-1|k-1} \quad (2.47)$$

$$dP_b = 2\mathbb{P} \left(\hat{\Omega}^\times P_{b,k-1|k-1} \right) + Q_b - P_{c,k-1|k-1}^\top S P_{c,k-1|k-1} \quad (2.48)$$

$$dP_c = -P_{c,k-1|k-1} \hat{\Omega}^\times - P_{b,k-1|k-1} + Q_c - P_{a,k-1|k-1} S P_{c,k-1|k-1} \quad (2.49)$$

2.2.4. RIEKF Algorithm

Algorithm 2 RIEKF signal Algorithm

Initialize Q_a, Q_b, Q_c and R_C

Initialize q_0, b_0 and P_{a0}, P_{b0}, P_{c0}

loop

$$\hat{\omega} = \omega_m - \hat{b}_{k-1|k-1}$$

$$\hat{y}_i = \hat{q}^{-1} \circ d_i \circ \hat{q}$$

$$\delta = \sum_{i=1} \left(d_i^\times \right)^\top R_c^{-1} (d_i - \hat{q} \circ y_i \circ \hat{q}^{-1})$$

$$S = \sum_{i=1} \left(d_i^\times \right)^\top R_c^{-1} d_i^\times$$

$$\hat{q}_{k|k-1} = \hat{q}_{k-1|k-1} \circ \exp \left(0.5h \left(\hat{\omega} + P_{a,k-1|k-1} \delta \right) \right)$$

$$d\hat{b} = \hat{q}^{-1} \circ \left(P_{c,k-1|k-1}^\top \delta \right) \circ \hat{q}$$

$$\hat{b}_{k|k-1} = \hat{b}_{k-1|k-1} + h \left[d\hat{b} \right]$$

$$dP_a = -2\mathbb{P} \left(P_{c,k-1|k-1} \right) + Q_a - P_{a,k-1|k-1} S P_{a,k-1|k-1}$$

$$dP_b = 2\mathbb{P} \left(\hat{\Omega}^\times P_{b,k-1|k-1} \right) + Q_b - P_{c,k-1|k-1}^\top S P_{c,k-1|k-1}$$

$$dP_c = -P_{c,k-1|k-1} \hat{\Omega}^\times - P_{b,k-1|k-1} + Q_c - P_{a,k-1|k-1} S P_{c,k-1|k-1}$$

$$P_{i,k|k-1} = P_{i,k-1|k-1} + h \left[dP_i \right], \quad i \in \{a, b, c\}$$

$$P = \begin{bmatrix} P_a & P_c \\ P_c^\top & P_b \end{bmatrix}$$

end loop

Chapter 3.

SLAM

3.1. EKF SLAM

The presented EKF SLAM solution in this chapter is based on the simple SLAM problem which is the 2D unicycle model, where the solution presented is inspired by [1].

The state vector given by n landmarks is defined as

$$X = (\theta_k, x_k, p^1, \dots, p^n) \quad (3.1)$$

where θ_k and x_k is the orientation and position of the robot at time step k and p^i is the position of the landmark i . The measurement of the position of a landmark i relative to the robot in the robot frame is

$$z^i = R^T(\theta) (p^i - x) \quad (3.2)$$

where $R(\theta)$ is the rotation matrix of angle θ defined by

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \quad (3.3)$$

The model of the robot moving in a plane is defined as

$$\theta_k = \theta_{k-1} + h (\omega_k + w_k^\omega) \quad (3.4)$$

$$x_k = x_{k-1} + hR(\theta_{k-1})v_k + w_k^v \quad (3.5)$$

$$p_k^i = p_{k-1}^i \quad (3.6)$$

with $\omega_k \in \mathbb{R}$ and $v_k \in \mathbb{R}^2$ being the odeometry-based estimated change in heading and velocity, respectively, where w_k^ω and w_k^v are the associated noises. Then the covariance matrix of the noises is defined as

$$Q_k = \text{Cov} \left(\begin{bmatrix} w_k^\omega \\ w_k^v \\ 0_{2n} \end{bmatrix} \right) \in \mathbb{R}^{(3+2n) \times (3+2n)} \quad (3.7)$$

and the output noise covariance matrix as

$$R_k = \text{Cov} \left(\begin{bmatrix} V_n^1 \\ \vdots \\ V_n^K \end{bmatrix} \right) \in \mathbb{R}^{2n \times 2n} \quad (3.8)$$

where V_n is the observation noise.

The estimated measurement can be written as

$$\hat{z}^i = R^T(\hat{\theta}) (\hat{p}^i - \hat{x}) \quad (3.9)$$

and the propagation of the estimated model of the robot is then

$$\hat{\theta}_{k|k-1} = \hat{\theta}_{k-1|k-1} + h\omega_k \quad (3.10)$$

$$\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} + hR(\hat{\theta}_{k-1|k-1})v_k \quad (3.11)$$

$$\hat{p}_{k|k-1}^i = \hat{p}_{k-1|k-1}^i \quad (3.12)$$

3.1.1. Error equations and linearization of error equations

The estimation errors are given as

$$\tilde{\theta} = \theta - \hat{\theta} \quad (3.13)$$

$$\tilde{x} = x - \hat{x} \quad (3.14)$$

$$\tilde{p}^i = p^i - \hat{p}^i \quad (3.15)$$

and together with the equations (3.11) and (3.5), it gives

$$\begin{aligned} \tilde{x}_{k|k-1} &= x_k - \hat{x}_{k|k-1} \\ &= x_{k-1} + hR(\theta_{k-1})v_k - \left(\hat{x}_{k-1|k-1} + hR(\hat{\theta}_{k-1|k-1})v_k \right) \\ &= \tilde{x}_{k-1|k-1} + h \left(R(\hat{\theta}_{k-1|k-1} + \tilde{\theta}_{k-1|k-1}) - R(\hat{\theta}_{k-1|k-1}) \right) v_k \\ &\approx \tilde{x}_{k-1|k-1} + h\tilde{\theta}_{k-1|k-1} \frac{d}{d\theta} R(\theta) \Big|_{\hat{\theta}_{k-1|k-1}} v_k \\ &= \tilde{x}_{k-1|k-1} + h\tilde{\theta}_{k-1|k-1} R(\hat{\theta}_{k-1|k-1}) [1]^\times v_k \end{aligned} \quad (3.16)$$

and the error equations can then be written as

$$\tilde{\theta}_{k|k-1} = \tilde{\theta}_{k-1|k-1} \quad (3.17)$$

$$\tilde{x}_{k|k-1} = \tilde{x}_{k-1|k-1} + h\tilde{\theta}_{k-1|k-1} R(\hat{\theta}_{k-1|k-1}) [1]^\times v_k \quad (3.18)$$

$$\tilde{p}_{k|k-1}^i = \tilde{p}_{k-1|k-1}^i \quad (3.19)$$

Next, using the equations (3.2) and (3.9) the estimation error for the measurement can be written as

$$\begin{aligned} \tilde{z}^i &= z^i - \hat{z}^i \\ &= R^T(\theta) (p^i - x) - R^T(\hat{\theta}) (\hat{p}^i - \hat{x}) \\ &= \left(R^T(\hat{\theta} + \tilde{\theta}) - R^T(\hat{\theta}) \right) (\hat{p}^i - \hat{x}) + R^T(\hat{\theta} + \tilde{\theta}) (\tilde{p}^i + \tilde{x}) \\ &= \frac{d}{d\theta} R^T(\theta) \Big|_{\hat{\theta}} (\hat{p}^i - \hat{x}) \tilde{\theta} + R^T(\hat{\theta}) (\tilde{p}^i - \tilde{x}) + r \\ &= -[1]^\times R^T(\hat{\theta}) (\hat{p}^i - \hat{x}) \tilde{\theta} + R^T(\hat{\theta}) (\tilde{p}^i - \tilde{x}) + r \end{aligned} \quad (3.20)$$

with r being a higher order terms in $\tilde{\theta}$, \tilde{x} and \tilde{p}^i .

Note that for equations (3.16) and (3.20) it is used that

$$\frac{d}{d\theta} R^T(\theta) = \left(\frac{d}{d\theta} R(\theta) \right)^T = -[1]^\times R(\theta)^T \quad (3.21)$$

$$\frac{d}{d\theta} R(\theta) = [1]^\times R(\theta) = R(\theta)[1]^\times \quad (3.22)$$

Then the linearized matrices at the estimates are presented as

$$A_k = \begin{bmatrix} 1 & 0^T & 0^T \\ hR(\hat{\theta}_{k-1|k-1}) [1]^\times v_k & I & 0 \\ 0 & 0 & I \end{bmatrix} \quad (3.23)$$

$$C_k = \begin{bmatrix} -[1]^\times R(\hat{\theta}_{k|k-1})^T (\hat{p}^1 - \hat{x}) & -R^T(\hat{\theta}_{k|k-1}) & R(\hat{\theta}_{k|k-1})^T \end{bmatrix} \quad (3.24)$$

$$G_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & R(\hat{\theta}_{k-1|k-1}) & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.25)$$

It is noted that the linearized matrices presented above are defined given $n = 1$ landmark. For example, taking into consideration the cases where $n = 2$ landmarks or $n = 3$ landmarks, it is possible to observe a representation of how the matrices should look given n numbers of landmarks. The linearized matrices for these two cases are written as

For $n = 2$:

$$A_k = \begin{bmatrix} 1 & 0^T & 0^T & 0^T \\ hR(\hat{\theta}_{k-1|k-1}) [1]^\times v_k & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}$$

$$C_k = \begin{bmatrix} -[1]^\times R(\hat{\theta}_{k|k-1})^T (\hat{p}^1 - \hat{x}) & -R^T(\hat{\theta}_{k|k-1}) & R(\hat{\theta}_{k|k-1})^T & 0 \\ -[1]^\times R(\hat{\theta}_{k|k-1})^T (\hat{p}^2 - \hat{x}) & -R^T(\hat{\theta}_{k|k-1}) & 0 & R(\hat{\theta}_{k|k-1})^T \end{bmatrix}$$

$$G_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & R(\hat{\theta}_{k-1|k-1}) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

For $n = 3$:

$$A_k = \begin{bmatrix} 1 & 0^T & 0^T & 0^T & 0^T \\ hR(\hat{\theta}_{k-1|k-1})[1]^\times v_k & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}$$

$$C_k = \begin{bmatrix} -[1]^\times R(\hat{\theta}_{k|k-1})^T (\hat{p}^1 - \hat{x}) - R^T(\hat{\theta}_{k|k-1}) R(\hat{\theta}_{k|k-1})^T & 0 & 0 \\ -[1]^\times R(\hat{\theta}_{k|k-1})^T (\hat{p}^2 - \hat{x}) - R^T(\hat{\theta}_{k|k-1}) & 0 & R(\hat{\theta}_{k|k-1})^T & 0 \\ -[1]^\times R(\hat{\theta}_{k|k-1})^T (\hat{p}^3 - \hat{x}) - R^T(\hat{\theta}_{k|k-1}) & 0 & 0 & R(\hat{\theta}_{k|k-1})^T \end{bmatrix}$$

$$G_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & R(\hat{\theta}_{k-1|k-1}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3.1.2. Kalman gain and estimates

With the linearized system, the prior and posterior estimates are defined. The prior estimate of the covariance is defined as

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + G_k Q_k G_k^T \quad (3.26)$$

The Kalman gain is

$$K_{k|k} = P_{k|k-1} C_k^T S_k^{-1} \quad (3.27)$$

with S_k being

$$S_k = C_k P_{k|k-1} C_k^T + R_k \quad (3.28)$$

Then the Kalman gain is used to define the posterior estimate of the covariance

$$P_{k|k} = [I - K_k C_k] P_{k|k-1} \quad (3.29)$$

and also the posterior estimate of the model state

$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k (z_k - \hat{z}_k) \quad (3.30)$$

3.1.3. EKF SLAM Algorithm

From the formulas presented in this chapter, the algorithm for the EKF SLAM can be described by a propagation and an update.

Algorithm 3 EKF SLAM

Initialize P_0 and \hat{X}_0

loop

 Define A_k, C_k, G_k as in (3.23), (3.24), (3.25)

 Define Q_k, R_k as in (3.7), (3.8)

Propagation

$$\hat{\theta}_{k|k-1} = \hat{\theta}_{k-1|k-1} + h\omega_k$$

$$\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} + hR \left(\hat{\theta}_{k-1|k-1} \right) v_k$$

$$\hat{p}_{k|k-1}^i = \hat{p}_{k-1|k-1}^i$$

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + G_k Q_k G_k^T$$

Update

$$\hat{z}_k = h \left(\hat{X}_{k|k-1} \right)$$

$$S_k = C_k P_{k|k-1} C_k^T + R_k$$

$$K_{k|k} = P_{k|k-1} C_k^T S_k^{-1}$$

$$P_{k|k} = [I - K_k C_k] P_{k|k-1}$$

$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k (z_k - \hat{z}_k)$$

end loop

3.1.4. Consistency issues of the EKF SLAM

One of the main issues of the EKF SLAM is the inconsistency from the covariance estimate since the Kalman gain is computed from the covariance estimate. The inconsistency is a result of the updated covariance matrix $P_{n|n}$ being calculated before the update with the predicted state $\hat{X}_{n|n.1}$. This means that it does not take into account the updated state $\hat{X}_{n|n}$ even though it is supposed to represent the covariance of the updated error.

From [4] it has been proved that by observing the properties of the nonlinear SLAM system with the linearized error state, the observable subspace of the standard EKF always has a higher dimension than the observable subspace of the nonlinear SLAM system. This reduces the covariance matrix in directions of the state space, although there is no information available. Further, in [5] it has also been shown that the orientation of the robot, when the error is large, can cause the inconsistency to be a significant problem.

In the usual SLAM problem, the measurements are related to the position of the landmarks relative to the robot, where both absolute rotation of the world frame and the global position is unknown. From [1], the problem observed was due to the linearization of the EKF equations, the EKF slam indicated that the global position and rotation are both estimated by the EKF. If the EKF equations were linearized at the true trajectory, this would not be an issue, and consistency would be achieved. This is not possible as the true trajectory is unknown, and the linearization is based on the estimates. It was also shown that the condition

$$\forall k > 0, \quad -(\hat{p}_{k|k-1} - \hat{p}_{0|0}) + \sum_{i=1}^{k-1} (\hat{x}_{i|i} - \hat{x}_{i|i-1}) = 0 \quad (3.31)$$

which consists of the updated states, is the only condition that guarantees consistency, but due to noise, there is a null probability for this condition to be realized in practice. In [1], a way of improving consistency through the error variables is discussed. It is shown that since the SLAM problem is a nonlinear system and the calculation of the error variables $X - \hat{X}$ is linear, that the linearization of the error variables is the stem of the consistency issues and not the actual EKF. By choosing a preferable error variable, it is possible to significantly improve the consistency of the EKF.

3.2. Lie groups in SLAM

This section presents a short and reasonable explanation of matrix lie groups and how this can be implemented together with SLAM and the special euclidean group (SE(2)).

A matrix lie group G is a closed subgroup of the defined set $GL(n; \mathbb{R})$ where $M_n(\mathbb{R})$ is defined as the set of all $n \times n$ matrices with real entities. Since G is subgroup of $GL(n; \mathbb{R})$, it has the following properties

$$I_n \in G, \quad \forall g \in G, g^{-1} \in G, \quad \forall a, b \in G, ab \in G \quad (3.32)$$

where I_n is the identity matrix of \mathbb{R}^n [1].

3.3. SE(2) as a Lie group

Using homogenous matrices, the group is defined by

$$G = SE(2) \quad (3.33)$$

The vector form of the logarithm is

$$\zeta = \begin{bmatrix} \theta \\ \rho_x \end{bmatrix}, \quad \theta \in \mathbb{R}, \rho_x \in \mathbb{R}^2 \quad (3.34)$$

and in matrix form it is

$$\zeta^\wedge = \begin{bmatrix} \theta[1]^\times & \rho_x \\ 0^T & 0 \end{bmatrix} \quad (3.35)$$

$$\exp(\zeta^\wedge) = \begin{bmatrix} \exp(\theta[1]^\times) & E(\theta)\rho_x \\ 0^T & 1 \end{bmatrix} \quad (3.36)$$

$$E(\theta) = \begin{bmatrix} \frac{\sin \theta}{\theta} & -\frac{1-\cos \theta}{\sin \theta} \\ \frac{1-\cos \theta}{\theta} & \frac{\sin \theta}{\theta} \end{bmatrix} \quad (3.37)$$

$$Ad(\zeta) = \begin{bmatrix} 1 & 0^T \\ -[1]^\times \rho_x & R(\theta) \end{bmatrix} \quad (3.38)$$

For a SLAM problem with n number of landmarks, the equations above can be formulated as

$$\zeta = \begin{bmatrix} \theta \\ \rho_x \\ \rho_1 \\ \vdots \\ \rho_n \end{bmatrix} \quad (3.39)$$

$$\zeta^\wedge = \begin{bmatrix} \theta[1]^\times & \rho_x & \rho_1 & \dots & \rho_n \\ 0^T & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0^T & 0 & 0 & \dots & 0 \end{bmatrix} \quad (3.40)$$

$$\exp(\zeta^\wedge) = \begin{bmatrix} \exp(\theta[1]^\times) & E(\theta)\rho_x & E(\theta)\rho_1 & \dots & E(\theta)\rho_n \\ 0^T & 1 & 0 & \dots & 0 \\ 0^T & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0^T & 0 & 0 & \dots & 1 \end{bmatrix} \quad (3.41)$$

$$Ad(\zeta) = \begin{bmatrix} 1 & 0^T & 0 & \dots & 0 \\ -[1]^\times \rho_x & R(\theta) & 0 & \dots & 0 \\ -[1]^\times \rho_1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -[1]^\times \rho_n & 0 & 0 & \dots & 0 \end{bmatrix} \quad (3.42)$$

3.4. IEKF SLAM

The IEKF SLAM solution presented in this chapter is inspired by [1] and is based on the 2D unicycle model. The EKF and the IEKF are very similar to each other as the IEKF is a modification of the EKF. The main difference between the mentioned filter is that for the IEKF, a nonlinear error variable is chosen, which causes some of the equations to differ. The SLAM lie group in SE(2) will also be implemented for this filter.

The state vector of the IEKF SLAM given n landmarks is defined as

$$X = (R(\theta), x, p^1, \dots, p^n) \quad (3.43)$$

and the estimate of the state vector as

$$\hat{X} = (\hat{R}(\theta), \hat{x}, \hat{p}^1, \dots, \hat{p}^n) \quad (3.44)$$

Then state vector and its estimate is given in matrix form as

$$X = \begin{bmatrix} R(\theta) & x & p^1 & \dots & p^n \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \quad (3.45)$$

$$\hat{X} = \begin{bmatrix} R(\hat{\theta}) & \hat{x} & \hat{p}^1 & \dots & \hat{p}^n \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \quad (3.46)$$

Just like the EKF, the measurement of a landmark and its estimate for the IEKF is given as

$$z^i = R^T(\theta) (p^i - x) \quad (3.47)$$

$$\hat{z}^i = R^T(\hat{\theta}) (\hat{p}^i - \hat{x}) \quad (3.48)$$

and the model of the robot moving in a plane is defined as

$$\theta_k = \theta_{k-1} + h(\omega_k + w_k^\omega) \quad (3.49)$$

$$x_k = x_{k-1} + hR(\theta_{k-1})v_k + w_k^v \quad (3.50)$$

$$p_k^i = p_{k-1}^i \quad (3.51)$$

where the propagation of the estimated model of the robot is

$$\hat{\theta}_{k|k-1} = \hat{\theta}_{k-1|k-1} + h\omega_k \quad (3.52)$$

$$\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} + hR(\hat{\theta}_{k-1|k-1})v_k \quad (3.53)$$

$$\hat{p}_{k|k-1}^i = \hat{p}_{k-1|k-1}^i \quad (3.54)$$

3.4.1. Error equations and linearization of error equations

In subsection 3.1.4, when discussing the consistency issues of the EKF, it was mentioned how the error being linear can cause inconsistency and by choosing a more preferable error variable it is possible to improve the consistency. For the RIEKF, given \hat{X} and X , the error variable is defined as $\tilde{X} = X\hat{X}^{-1}$ in matrix form

$$\tilde{X} = \begin{bmatrix} R(\tilde{\theta}) & \tilde{x} & \tilde{p}^1 & \dots & \tilde{p}^n \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \quad (3.55)$$

with the error variables being

$$\tilde{\theta} = \theta - \hat{\theta} \quad (3.56)$$

$$\tilde{x} = x - R(\tilde{\theta})\hat{x} \quad (3.57)$$

$$\tilde{p} = p - R(\tilde{\theta})\hat{p} \quad (3.58)$$

The estimated state error for the IEKF is as mentioned earlier described as $\tilde{X} = X\hat{X}^{-1}$ and the kinematic differential equation is then defined as

$$\dot{\tilde{X}} = \frac{d}{dt} (X\hat{X}^{-1}) = \dot{X}\hat{X}^{-1} - X\hat{X}^{-1}\dot{\hat{X}}\hat{X}^{-1} = XU\hat{X}^{-1} - X\hat{X}^{-1}\hat{X}U\hat{X}^{-1} = 0 \quad (3.59)$$

This gives the time propagation of the estimated state error

$$\dot{\tilde{\theta}} = 0, \quad \dot{\tilde{x}} = 0, \quad \dot{\tilde{p}}^i = 0 \quad (3.60)$$

which means that the prior estimated state error is defined as

$$\tilde{\theta}_{k|k-1} = \tilde{\theta}_{k-1|k-1} \quad (3.61)$$

$$\tilde{x}_{k|k-1} = \tilde{x}_{k-1|k-1} \quad (3.62)$$

$$\tilde{p}_{k|k-1}^i = \tilde{p}_{k-1|k-1}^i \quad (3.63)$$

Here it is clear to see that the propagation of the estimated state error is only affected by the state estimation error and does not depend on the state and its estimate, nor the inputs. Looking at the state estimation error for the EKF (3.17), (3.18) and (3.19), it is observed how the choice of the error variables will affect the linearization and thus the filter itself. Later in this section, the linearized matrices will be presented and the effect of the error variable on the filter will be more clear.

Further, the measurement estimation error is defined as

$$\begin{aligned}
 \tilde{z}^i &= z^i - \hat{z}^i \\
 &= R^T(\theta) (p^i - x) - R^T(\hat{\theta}) (\hat{p}^i - \hat{x}) \\
 &= R^T(\theta) \left((p^i - x) - R(\theta)R^T(\tilde{\theta}) (\hat{p}^i - \hat{x}) \right) \\
 &= R^T(\theta) (\tilde{p}^i - \tilde{x}) \\
 &= R^T(\hat{\theta}) (\tilde{p}^i - \tilde{x}) + r
 \end{aligned} \tag{3.64}$$

For the linearization of the error variables, the error state is defined as

$$\xi = [\tilde{\theta}, \tilde{x}, \tilde{p}^1, \dots, \tilde{p}^n]^T \tag{3.65}$$

and the measurement error as

$$\tilde{z} = \begin{bmatrix} \tilde{z}^1 \\ \vdots \\ \tilde{z}^n \end{bmatrix} \tag{3.66}$$

Then the linearized equations are given by

$$\xi_{k|k-1} = A_k \xi_{k-1|k-1} \tag{3.67}$$

$$\tilde{z}_k = C_k \xi_{k|k-1} \tag{3.68}$$

where it is found that the linearized matrices A_k and C_k are

$$A_k = I_{3+2n, 3+2n} \quad (3.69)$$

$$C_k = \begin{bmatrix} 0 & -R^T(\hat{\theta}_{k|k-1}) & R^T(\hat{\theta}_{k|k-1}) & 0 & \dots & 0 \\ 0 & -R^T(\hat{\theta}_{k|k-1}) & 0 & R^T(\hat{\theta}_{k|k-1}) & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -R^T(\hat{\theta}_{k|k-1}) & 0 & 0 & \dots & R^T(\hat{\theta}_{k|k-1}) \end{bmatrix} \quad (3.70)$$

Earlier it was mentioned how the linearized matrices will show how the filter gets affected by the error variable. This can be seen by observing that the matrix $A_k = I_{3+2n, 3+2n}$ is defined as an identity matrix. This ultimately means that the propagation of the filter, the covariance matrix to be exact, is unaffected by noise.

At last, the linearized matrix G_k is defined as

$$G_k = Ad(X) = \begin{bmatrix} 0 & 0^T & 0 & \dots & 0 \\ -[1]^\times x & R(\theta) & 0 & \dots & 0 \\ -[1]^\times p_1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -[1]^\times p_n & 0 & 0 & \dots & 0 \end{bmatrix} \quad (3.71)$$

3.4.2. Kalman gain and estimates

Exactly like the EKF, the prior and posterior estimate of the covariance and also the Kalman gain is specified as

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + G_k Q_k G_k^T \quad (3.72)$$

$$K_{k|k} = P_{k|k-1} C_k^T S_k^{-1} \quad (3.73)$$

$$P_{k|k} = [I - K_k C_k] P_{k|k-1} \quad (3.74)$$

It is noted that both Q_k and R_k are defined exactly the same as for the EKF SLAM.

Unlike the EKF, the posterior estimate of the state is described as

$$\hat{X}_{k|k} = \exp(\xi^\wedge) \hat{X}_{k|k-1} \quad (3.75)$$

where ξ^\wedge is the logarithm of the vector, ξ , which is further defined by

$$\xi = K\tilde{z} = K \left(z^i - \hat{z}^i \right) \quad (3.76)$$

and has the vector form

$$\xi = [\xi_\theta, \xi_x, \xi_{p_1}, \dots, \xi_{p_n}]^T \quad (3.77)$$

In section 3.3 the equations for the SE(2) as a Lie group was presented. From these equations, the exponential of the logarithm can be described as

$$\exp(\xi^\wedge) = \begin{bmatrix} R(\xi_\theta) & E(\xi_\theta)\xi_x & E(\xi_\theta)\xi_{p_1} & \dots & E(\xi_\theta)\xi_{p_n} \\ 0^T & 1 & 0 & \dots & 0 \\ 0^T & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0^T & 0 & 0 & \dots & 1 \end{bmatrix} \quad (3.78)$$

Given the exponential of the logarithm and the prior estimate, the posterior estimated state is defined as

$$\hat{X}_{k|k} = \begin{bmatrix} R(\hat{\theta}_{k|k-1} + \xi_\theta) & R(\xi_\theta)\hat{x}_{k|k-1} + E(\xi_\theta)\xi_x & R(\xi_\theta)\hat{p}_{k|k-1}^1 + E(\xi_\theta)\xi_{p_1} & \dots \\ 0^T & 1 & 0 & \dots \\ 0^T & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad (3.79)$$

Then the posterior estimated state can be expressed in vector form as

$$\hat{X}_{k|k} = \begin{bmatrix} \hat{\theta}_{k|k-1} + \xi_\theta \\ R(\xi_\theta)\hat{x}_{k|k-1} + E(\xi_\theta)\xi_x \\ R(\xi_\theta)\hat{p}_{k|k-1}^1 + E(\xi_\theta)\xi_{p_1} \\ \vdots \\ R(\xi_\theta)\hat{p}_{k|k-1}^n + E(\xi_\theta)\xi_{p_n} \end{bmatrix} \quad (3.80)$$

3.4.3. IEKF SLAM Algorithm

From the formulas presented in this chapter, the algorithm for the IEKF SLAM can be described by a propagation and an update.

Algorithm 4 IEKF SLAM

Initialize P_0 and \hat{X}_0
loop
 Define A_k, C_k, G_k as in (3.69), (3.70), (3.71)
 Define Q_k, R_k as in (3.7), (3.8)
 Propagation
 $\hat{\theta}_{k|k-1} = \hat{\theta}_{k-1|k-1} + h\omega_k$
 $\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} + hR \left(\hat{\theta}_{k-1|k-1} \right) v_k$
 $\hat{p}_{k|k-1}^i = \hat{p}_{k-1|k-1}^i$
 $P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + G_k Q_k G_k^T$
 Update
 $\hat{z}_k = h \left(\hat{X}_{k|k-1} \right)$
 $S_k = C_k P_{k|k-1} C_k^T + R_k$
 $K_{k|k} = P_{k|k-1} C_k^T S_k^{-1}$
 $P_{k|k} = [I - K_k C_k] P_{k|k-1}$
 $\hat{X}_{k|k} = \exp(K_k (z_k - \hat{z}_k)) \hat{X}_{k|k-1}$
end loop

3.5. Analytical SLAM

The analytical SLAM is a technique that was introduced in recent years, where the idea is to avoid linearization while at the same time maintaining or reducing the complexity of the algorithm. As we can recall from the previous subchapters, the EKF SLAM and IEKF SLAM both are based on linearization at the estimates, which causes issues like inconsistency and convergence complications. Unlike the EKF- and IEKF SLAM, the measurements of the analytical SLAM are based on actual measurements instead of estimates, which makes it possible to reformulate the non-linear SLAM system into a simpler linear time-varying (LTV) system.

Combining the LTV-filter and contraction analysis, better convergence properties of the SLAM problem can be achieved. Briefly explained, with respect to the convergence of the neighboring trajectories, contraction theory converts a non-linear stability problem into a first-order LTV stability problem. A detailed explanation of the contraction theory can be found in [8], [7] and [10].

The computational complexity of the EKF- and IEKF SLAM is described as $\mathcal{O}(n^2)$ where n is the number of landmarks. Whereas for the analytical SLAM-DUNK, by introducing decoupling/DUNK, the computational complexity is reduced to $\mathcal{O}(n)$. From this, it is evident that the computational complexity of the analytical SLAM compared to the EKF- and IEKF SLAM is far less as the

number of landmarks n increases.

3.5.1. Decoupled Unlinearized Networked Kalman filter - DUNK

The idea behind decoupling is to assign a virtual vehicle to each landmark to process the information from the measurements independently. Further, a consensus vehicle, defined by the weighted average of all the virtual vehicles, summarizes the information from these virtual vehicles and gives feedback to the observer. By doing this, computation complexity can be reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$. A detailed explanation of the DUNK can be found in [10].

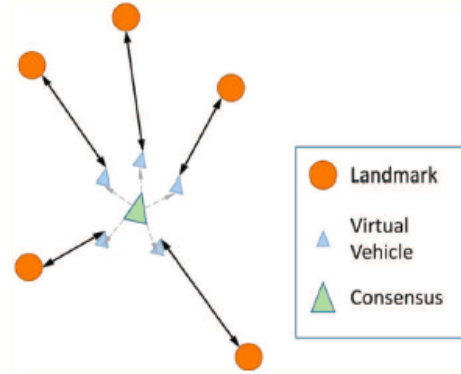


Figure 3.1.: Visualization of the SLAM-DUNK with virtual vehicles and consensus [10]

3.5.2. Implementation of the SLAM-DUNK

We know now that the idea behind the SLAM-DUNK is to assign a virtual vehicle for each landmark and then define a consensus vehicle which is the weighted average of the virtual vehicles. The consensus vehicle represents the trajectory of the actual robot. The implementation presented in this subsection is inspired from [10] and is based on the case where both the range measurement and bearing are available.

To implement the SLAM-DUNK, we have to compute a set of equations for each landmark. The state vector of each virtual vehicle is defined as

$$x^n = \begin{bmatrix} x_i \\ x_{vi} \end{bmatrix} \quad (3.81)$$

where $x_i = [x_{i1}, x_{i2}]^T$ represents the x- and y-coordinates of the assigned landmark

and $x_{vi} = [x_{vi1}, x_{vi2}]^T$ represents the x- and y-coordinates of the corresponding virtual vehicle.

The position of a landmark with respect to the global position of the vehicle is given by

$$x_i^b = R(\beta)^T (x_{lG} - x_{vG}) \quad (3.82)$$

where x_{lG} and x_{vG} represents the coordinates of the global landmark position and global vehicle position, respectively, and β represents the heading angle of the vehicle.

Then the bearing angle and measurement of the robot with respect to landmark i is

$$\theta_i = \text{Atan2}(x_{i2}^b, x_{i1}^b) \quad (3.83)$$

and

$$r_i = \sqrt{(x_{i1}^b)^2 + (x_{i2}^b)^2} \quad (3.84)$$

where x_{i1}^b is the x-coordinate and x_{i2}^b is the y-coordinate of the landmark position relative to the global position of the vehicle.

This relation can be rewritten into

$$h_i x_i^b = 0, \quad h_i^* x_i^b = r_i \quad (3.85)$$

where

$$h_i = [\sin \theta_i, -\cos \theta_i], \quad h_i^* = [\cos \theta_i, \sin \theta_i] \quad (3.86)$$

It is noted that in [10] the coordinate transformation matrix $T(\beta)$ is used whereas for the implementation in this paper the transformation matrix is defined as $R(-\beta) = R(\beta)^T$.

The vehicle velocity is defined as

$$u^b = \begin{bmatrix} u_x \cos \beta \\ u_y \sin \beta \end{bmatrix} \quad (3.87)$$

Now we define the following equations specific to the SLAM-DUNK with both measurement and bearing available. The measurement model is described as

$$y_{i1} = H_{i1} \begin{bmatrix} x_i \\ x_{vi} \end{bmatrix} \quad (3.88)$$

where $y_{i1} = [0, r_i]^T$ and

$$H_{i1} = \begin{bmatrix} h_i R(\beta)^T & -h_i R(\beta)^T \\ h_i^* R(\beta)^T & -h_i^* R(\beta)^T \end{bmatrix} \quad (3.89)$$

Earlier it was mentioned that a consensus vehicle is to be defined as a weighted average of the virtual vehicles. The coordinates of the consensus vehicle is then

$$X_{vc} = \left(\sum \Sigma_{vi}^{-1} \right)^{-1} \sum \left(\Sigma_{vi}^{-1} x_{vi} \right) \quad (3.90)$$

where Σ_{vi} comes from the covariance matrix

$$P_i = \begin{bmatrix} \Sigma_i & \Sigma_{ivi} \\ \Sigma_{vii} & \Sigma_{vi} \end{bmatrix} \quad (3.91)$$

Further, the position of the consensus vehicle, X_{vc} , is used as the measurement $y_{i2} = x_{vc}$.

We now define the H_i, y_i and u as

$$H_i = \begin{bmatrix} H_{i1} \\ H_{i2} \end{bmatrix} y_i = \begin{bmatrix} y_{i1} \\ y_{i2} \end{bmatrix} u = \begin{bmatrix} 0 \\ u_b \end{bmatrix} \quad (3.92)$$

with $H_{i2} = \begin{bmatrix} 0 & I \end{bmatrix}$.

Then at last, the estimated state is expressed as

$$\hat{x}^n = u + K(y_i - H_i \hat{x}^n) \quad (3.93)$$

where the Kalman Gain is

$$K = P_i H_i^T R^{-1} \quad (3.94)$$

3.5.3. SLAM-DUNK Algorithm

From the equations presented in this chapter, the algorithm for the analytical SLAM-DUNK is summarized. It is noted that the notation n used to define the loop in the algorithm refers to the total number of landmarks.

Algorithm 5 Analytical SLAM-DUNK

Initialize P_0 and \hat{x}_0

Define R

$X_{vc} = [0, 0]^T$

loop

loop in range n

 Define x_{lG}, x_{vG}

$\theta_i = \text{Atan2}(x_{i2}^b, x_{i1}^b)$

$r_i = \sqrt{(x_{i1}^b)^2 + (x_{i2}^b)^2}$

 Define y_i, H_i and u as in (3.92)

$K = P_i H_i^T R^{-1}$

$\hat{x} = \hat{x} + h[u + K(y_i - H_i \hat{x})]$

end loop

$X_{vc} = \left(\sum \Sigma_{vi}^{-1} \right)^{-1} \sum \left(\Sigma_{vi}^{-1} x_{vi} \right)$

end loop

Chapter 4.

Simulation

The simulations performed in this project thesis are simulated by python, and the codes are presented in appendix [A.1](#), [A.2](#), [A.3](#) and also as an attachment on this project thesis.

[A.3](#) contains four different cases for the analytical SLAM, and due to a limited amount of time available, only one of those cases, SLAM-DUNK with bearing and range measurement, is presented. It is noted that the remaining cases are less advanced than the SLAM-DUNK with bearing and range measurement. The four cases implemented in the code consist of analytical SLAM with bearing only, analytical SLAM with bearing and range measurement, analytical SLAM-DUNK with bearing only, and analytical SLAM-DUNK with bearing and measurement. The code contains instructions on how to choose the desired case to simulate.

4.1. Nonlinear attitude filtering

This section presents the simulation steps and parameters of the MEKF and the RIEKF for attitude filtering. The simulation is performed with 0.001 s time step with a total time of 30s, where the true trajectory is defined by the sinusoidal input $\Omega = \frac{1}{2}\sin(\frac{2\pi}{5}t) [0, 0, 1] \frac{\circ}{s}$.

The initial rotation defined by the unit quaternion and the initial bias is initialized with a standard deviation $std_{q0} = 60^\circ$ and $std_{b0} = 20 \frac{\circ}{s}$, respectively. The coefficient matrix Q_a is defined with a standard deviation $std_\Omega = 25 \frac{\circ}{s}$, and Q_b with a standard deviation $std_b = 0.1 \frac{\circ}{s}$ squared. Next, the coefficient matrix R_c is defined with a standard deviation $std_y = 30^\circ$. It is important to note that for the noise initialization, the standard deviations are converted from degrees to radians before they are implemented.

A complete overview of the system initialization parameters can be found in table

4.1 and the noise initialization parameters can be found in table 4.2. It is noted that the initialization parameters used for this simulation are identical for both the MEKF and RIEKF.

System initialization	
Parameter	Value
Ω	$\frac{1}{2}\sin(\frac{2\pi}{5}t) [0, 0, 1] \frac{\circ}{s}$
h	0.001 (s)
t	30 (s)
d_i	$d_1 = [1, 0, 0]^T, d_2 = [0, 0, 0]^T, d_3 = [0, 0, 1]^T$
q_0	$[0, 0, 0, 1]^T$
b_0	$[0, -0.5, 0.01]^T$

Table 4.1.: Nonlinear attitude filtering: Parameters and their respective values for the system initialization

Noise initialization	
Parameter	Value
std_{q0}	60°
std_{b0}	$20 \frac{\circ}{s}$
std_{Ω}	$25 \frac{\circ}{s}$
std_b	$0.1 \frac{\circ}{s}$
std_y	30°
Pa_0	$\frac{1}{std_{q0}^2} I_{3 \times 3}$
Pb_0	$\frac{1}{std_{b0}^2} I_{3 \times 3}$
Pc_0	$I_{3 \times 3}$
Qa_0	$diag([std_{\Omega}^2, std_{\Omega}^2, std_{\Omega}^2])$
Qb_0	$diag([std_b^2, std_b^2, std_b^2])$
Qc_0	$0_{3 \times 3}$
R_c	$diag([std_y^2, std_y^2, std_y^2])$

Table 4.2.: Nonlinear attitude filtering: Parameters and their respective values for the noise initialization and coefficient matrices

4.2. SLAM

This section presents the simulation steps and parameters of the EKF- and IEKF SLAM and analytical SLAM-DUNK. The true trajectory/robot moves in a circle with $h = 1s$ time step, for a total of $t = 50s$, and has a speed of $v = 1\frac{m}{s}$ and an angular velocity of $\omega = \frac{2\pi}{40}s^{-1}$. A number of 20 landmarks are placed in a circle around the outline of the true trajectory.

It is noted that, unlike the EKF and MEKF, the simulation of the analytical SLAM-DUNK is performed with a time step of $h = 0.1s$. While the system initialization parameters, except the time step, are close to identical for all the filters, this is not the case when it comes to noise initialization. The noise initialization for the EKF and IEKF is seen from table 4.5 and noise initialization for the analytical SLAM is seen from table 4.6. Table 4.3 and table 4.4 shows the system initialization parameters for the analytical SLAM-DUNK and the EKF and MEKF, respectively.

The covariance matrix P_0 used for the analytical SLAM-DUNK stays constant throughout the simulation, which is acceptable when simulating with the main purpose of filter convergence. This is commented by Slotine where he states: "In addition, the precision on \mathbf{Q} and \mathbf{R} does not affect the filter's stability and convergence rates, but only its optimality." [10].

System initialization - EKF and IEKF SLAM	
Parameter	Value
h	$1s$
t	$80s$
v	$1\frac{m}{s}$
ω	$\frac{2\pi}{40}s^{-1}$
$\hat{\theta}_0$	0
\hat{x}_0	$[0, 0]^T$
\hat{p}_0	$[z_0^1, \dots, z_0^n]^T$

Table 4.3.: SLAM: Parameters and their respective values for the system initialization of the EKF and IEKF

System initializaition - Analytical SLAM-DUNK	
Parameter	Value
h	$0.1s$
t	$80s$
v	$1 \frac{m}{s}$
ω	$\frac{2\pi}{40}s^{-1}$
β	0
\hat{x}_{i0}	$[0, 0]^T$
\hat{x}_{vi0}	$[z_0^1, \dots, z_0^n]^T$

Table 4.4.: SLAM: Parameters and their respective values for the system initialization for the analytical SLAM-DUNK

Noise initializaition - EKF- and IEKF SLAM	
Parameter	Value
P_0	$\begin{bmatrix} diag([0.1, 0.1, 0.1]) & 0 \\ 0 & diag([100, \dots, 100])_{2n \times 2n} \end{bmatrix}$
Q_k	$\begin{bmatrix} diag([0.1, 0.1, 0.1]) & 0 \\ 0 & diag([100, \dots, 100])_{2n \times 2n} \end{bmatrix}$
R_k	$diag([0.1, \dots, 0.1])_{2n \times 2n}$

Table 4.5.: SLAM: Noise initialization and covariance matrix parameters and their respective values for the EKF- and IEKF SLAM

Noise initializaition - Analytical SLAM-DUNK	
Parameter	Value
P_0	$\begin{bmatrix} diag([0.2, 0.2]) & 0 \\ 0 & diag([0.0001, 0.0001]) \end{bmatrix}$
R_k	$diag([0.1, 0.1, 0.1, 0.1])$

Table 4.6.: SLAM: Noise initialization and covariance matrix parameters and their respective values of the analytical SLAM-DUNK

Chapter 5.

Results & Discussions

5.1. Nonlinear attitude filtering

The nonlinear attitude filtering simulation is performed for a number of 15 Monte Carlo runs and the final results is the average of those runs.

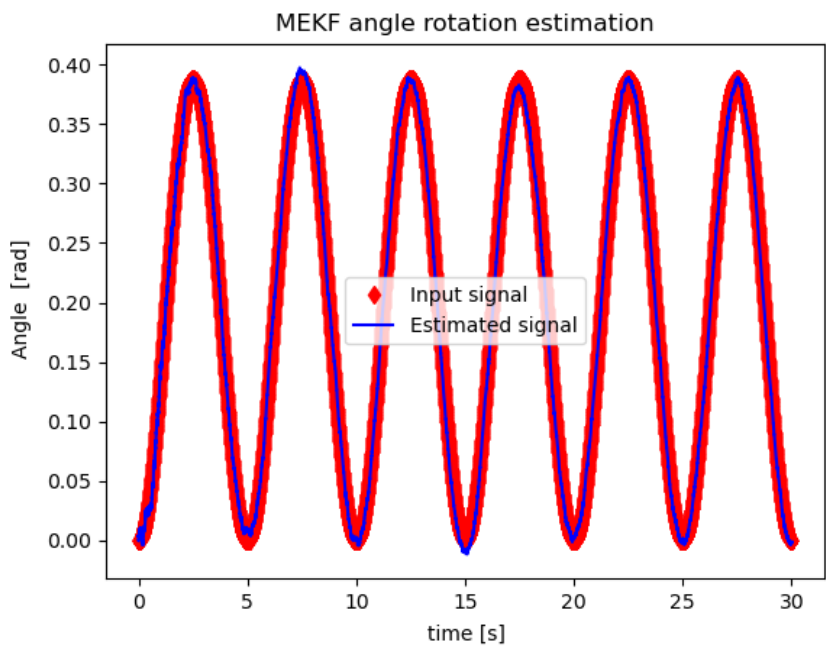


Figure 5.1.: Simulated angle rotation results for the MEKF

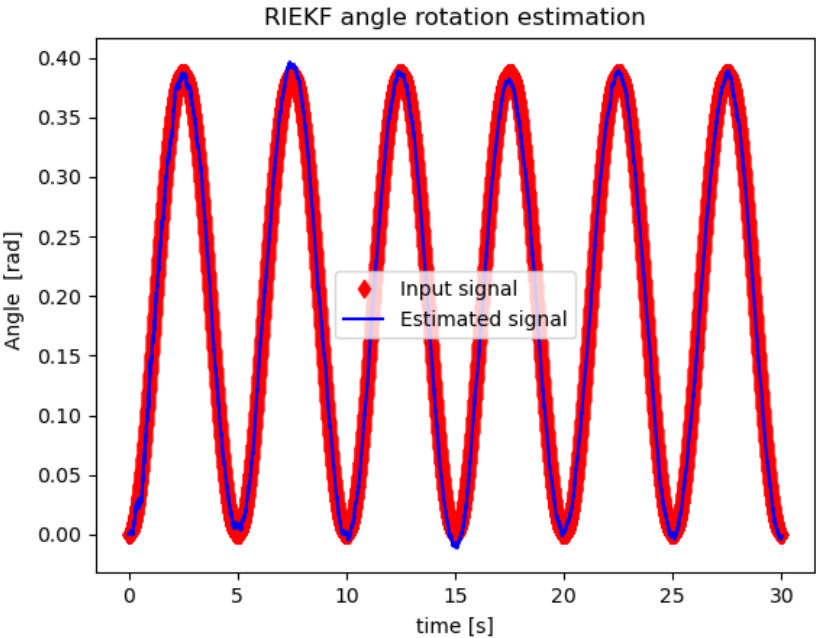


Figure 5.2.: Simulated angle rotation results for the RIEKF

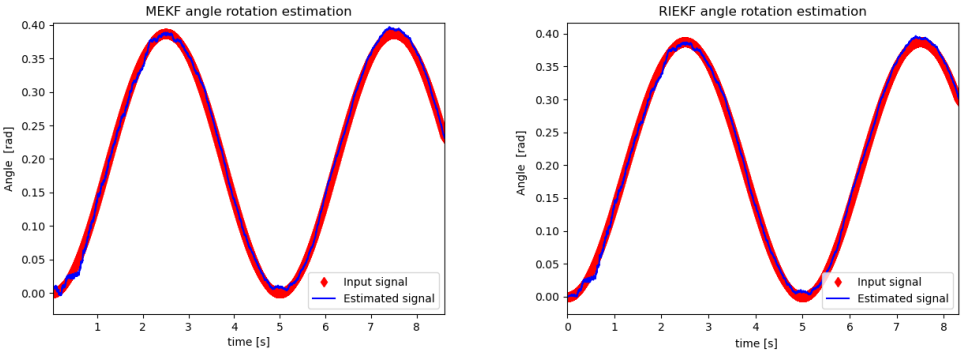


Figure 5.3.: Comparison of convergence time for the rotation angle estimation

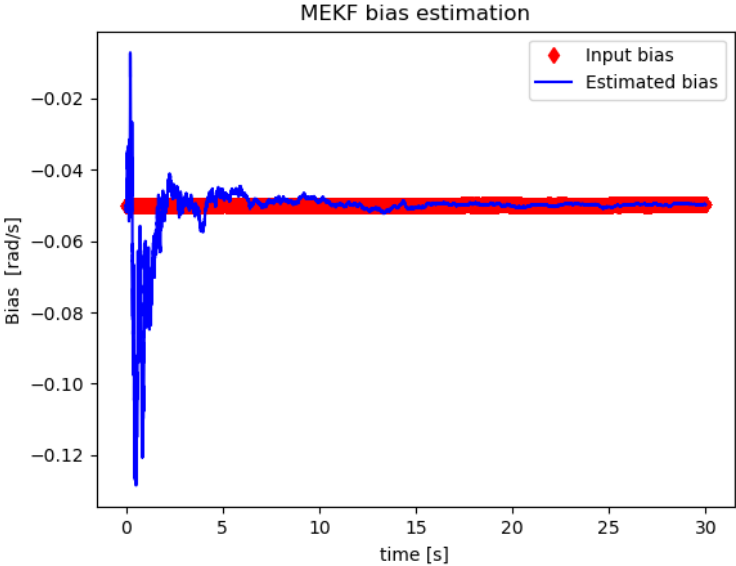


Figure 5.4.: Simulated bias results for the MEKF

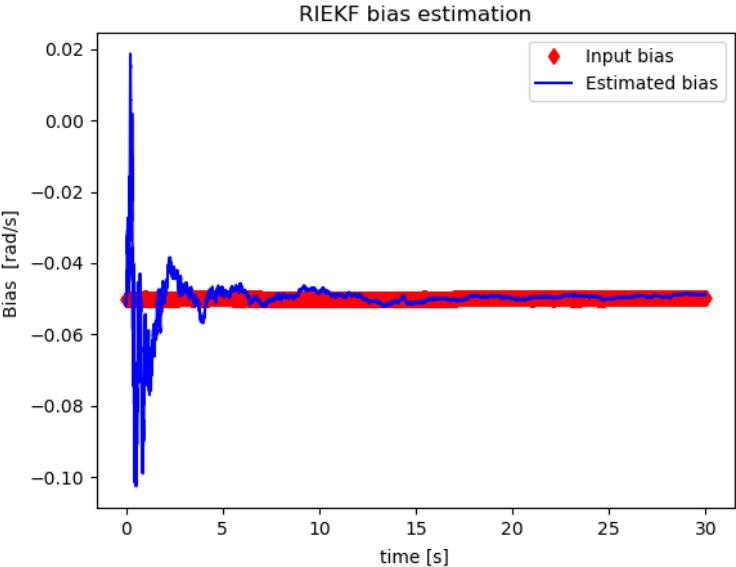


Figure 5.5.: Simulated bias results for the RIEKF

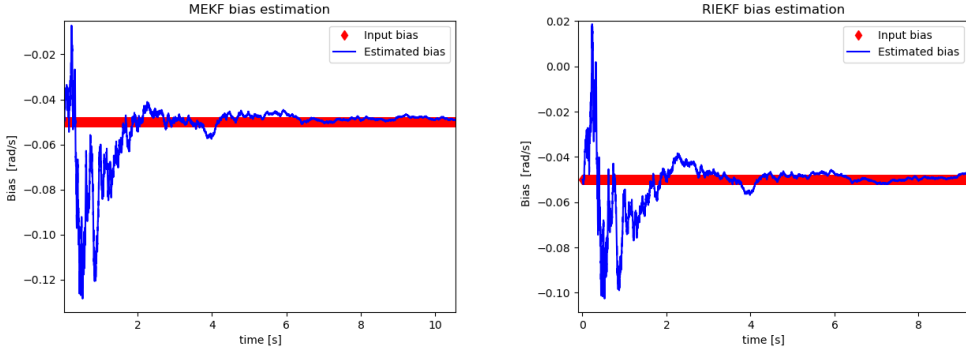


Figure 5.6.: Comparison of convergence time for the bias estimation

5.1.1. Error plots

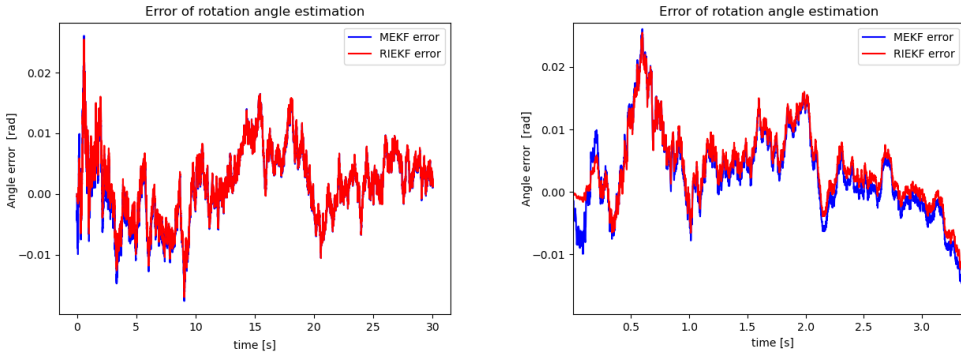


Figure 5.7.: Error of rotation angle estimation plotted against the time

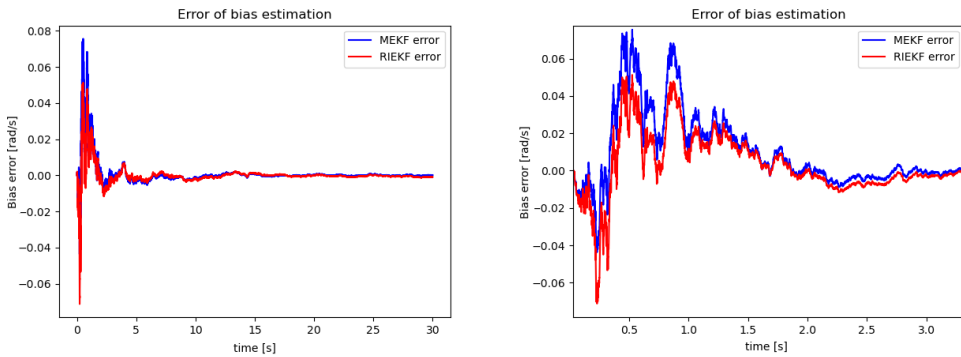


Figure 5.8.: Error of bias estimation plotted against the time

5.1.2. Discussion

Figure 5.1 and 5.2 shows the estimated and true trajectory of angle rotation for the MEKF and RIEKF, respectively. Also, from figure 5.3 it is obvious that both the MEKF and RIEKF are very precise concerning the chosen initialization parameters. Figure 5.4 and 5.5 shows the estimated and true bias trajectory, and it can be seen that the filter estimations for the bias fluctuate heavily for the first few seconds but quickly converge to the true bias. The difference in the filters is minimal, but the RIEKF proved to perform slightly better, which can be seen from figure 5.6 where the MEKF converges at $t \approx 6s$ and the RIEKF converges at $t \approx 4s$.

A comparison of the bias error estimation and the rotation angle estimation error is presented on figure 5.8 and 5.7. The minor difference in the filter estimations may not be very clear from this particular simulation study, which can be reasoned by the initialization parameters. The comparison study[11] by M. Zamani, J Trumpf, and R. Mahony presents the comparison by simulation between the MEKF and RIEKF and other filters, where the difference between the MEKF and RIEKF presents itself better. This comparison study confirms our statement and results about the RIEKF slightly outperforming the MEKF.

5.2. SLAM

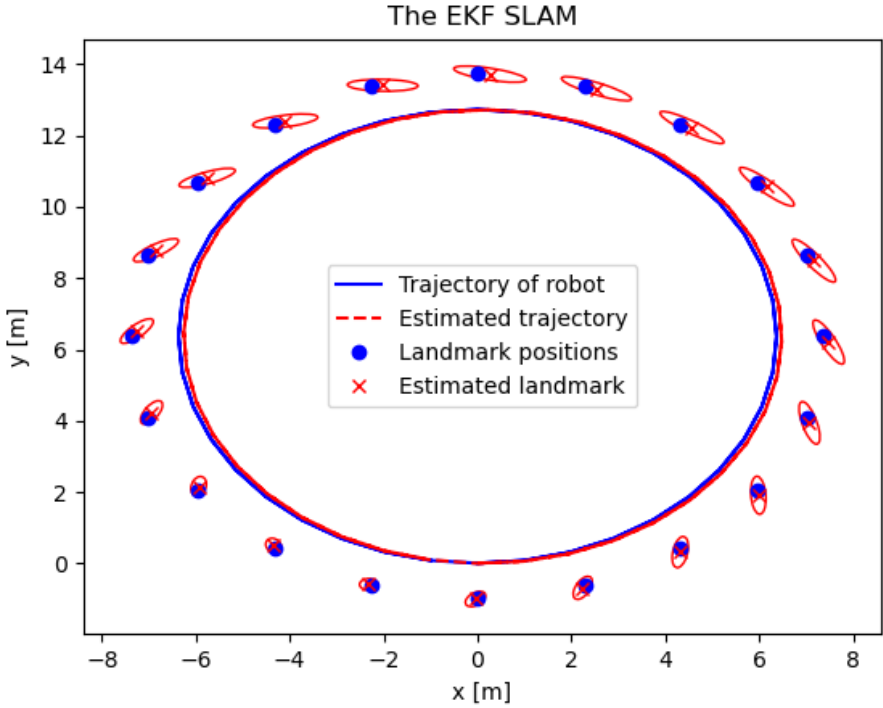


Figure 5.9.: Simulated results for the EKF SLAM with confidence ellipse

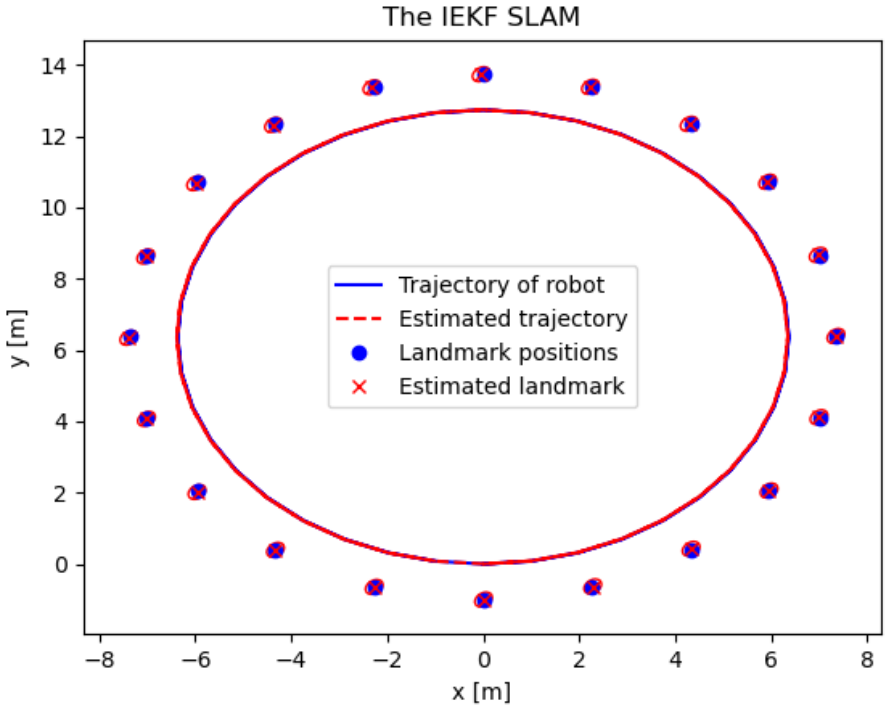


Figure 5.10.: Simulated results for the IEKF SLAM with confidence ellipse

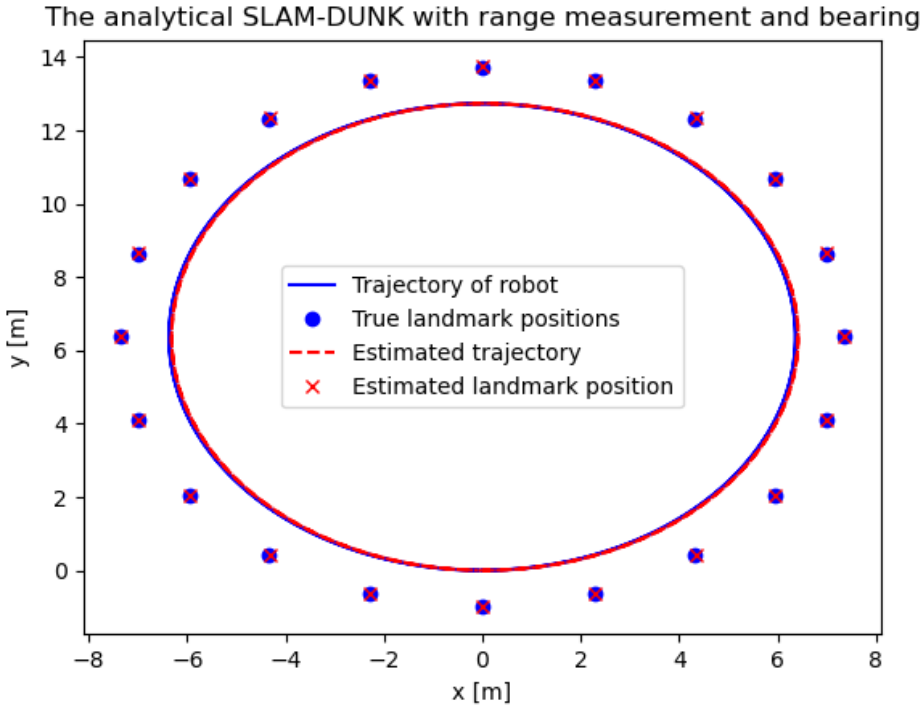


Figure 5.11.: Simulated results for the analytical SLAM-DUNK

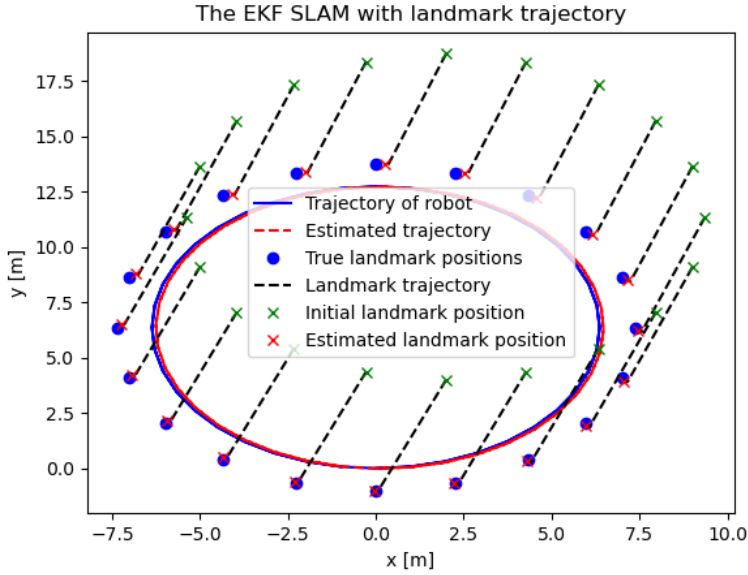


Figure 5.12.: Simulated results for the EKF SLAM with landmark trajectories and initial landmark positions

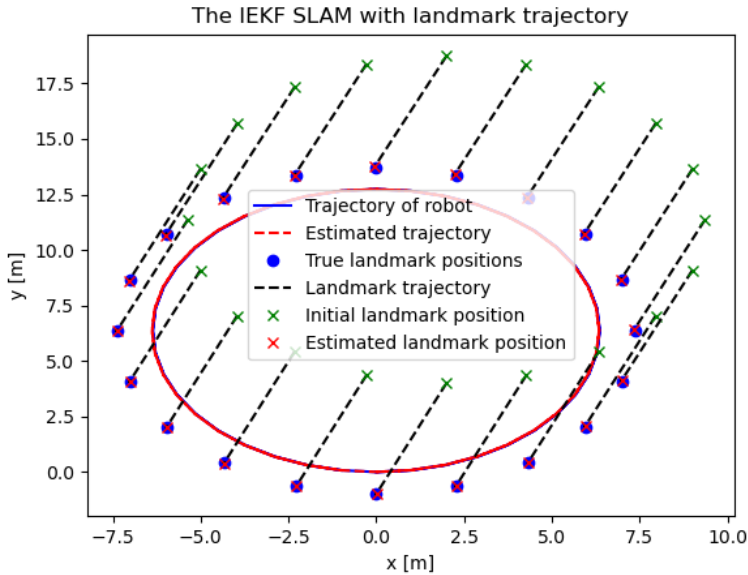


Figure 5.13.: Simulated results for the IEKF SLAM with landmark trajectories and initial landmark positions

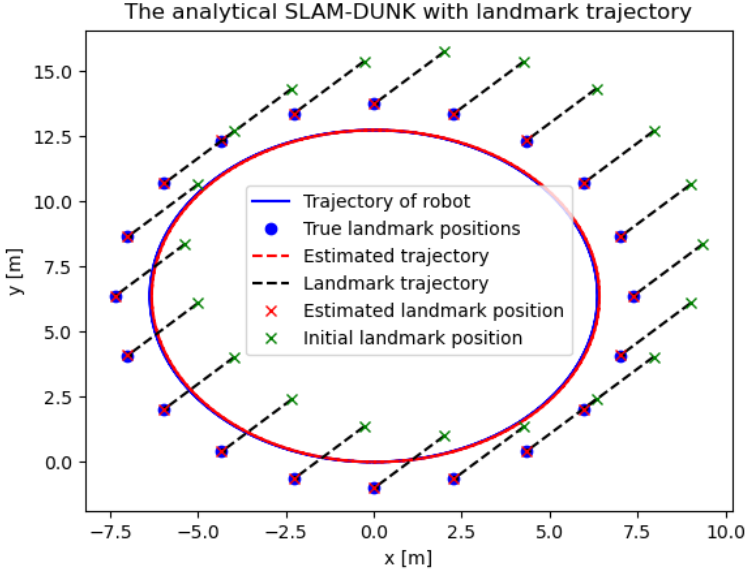


Figure 5.14.: Simulated results for the analytical SLAM-DUNK with landmark trajectories and initial landmark positions

5.2.1. Discussion

Figure 5.9, 5.10 and 5.11 shows the simulation performed for the EKF SLAM, IEKF SLAM and analytical SLAM-DUNK, respectively. Here it is obvious that the EKF is outperformed by both the analytical SLAM-DUNK and IEKF SLAM, which are shown to be very accurate filters when it comes to SLAM.

The confidence ellipse on figure 5.9 and 5.10 represents the accuracy of the landmark estimates for the respective filters. The consistency issues from the covariance estimate present themselves clearly by the fact that the radius of the confidence ellipse is increased for some landmarks and decreased for others. From the plot of the EKF SLAM, we observe an increase in the radius of the confidence ellipse as the distance between the landmark estimate and the true landmark position increases. As mentioned in subchapter 3.4.1, the covariance matrix for the IEKF is unaffected by noise, and this can be seen in figure 5.10 to result in a much more accurate landmark estimation for all landmarks. It is also worth to mention the landmark trajectories are identical for all filters, as can be seen from figure 5.12, 5.13 and 5.14.

Chapter 6.

Conclusion

The simulations performed in this project have shown the relations and differences between various filters. For the nonlinear attitude filtering simulations, we see that the RIEKF slightly outperforms the MEKF relative to the bias convergence. With the data presented, it is concluded that they both are accurate and effective filters. This can also be seen from the comparison study [11] where the MEKF and RIEKF proved to only be outperformed by the GAME filter amongst all the filters that were simulated.

For the SLAM simulations, the inconsistency of the EKF was exposed, and the IEKF demonstrated how the difference in error variable, due to linearization of the system, could eliminate/improve the inconsistency of the EKF. It is concluded that the reason behind the consistency issues is not the fact that the SLAM is based on a Kalman filter but rather the linearization of the system defined by the error variable. Further, the analytical SLAM-DUNK demonstrated the properties of SLAM without linearization. It was shown to converge with high accuracy and proved to perform at the same level as IEKF, relative to consistency and convergence, and at the same time avoid linearization.

References

- [1] Axel Barrau and Silvere Bonnabel. “An EKF-SLAM algorithm with consistency properties”. In: *arXiv preprint arXiv:1510.06263* (2015).
- [2] Silvère Bonnabel, Philippe Martin, and Erwan Salaün. “Invariant extended Kalman filter: theory and application to a velocity-aided attitude estimation problem”. In: *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 1297–1304.
- [3] RL Farrenkopf. “Analytic steady-state accuracy solutions for two common spacecraft attitude estimators”. In: *Journal of Guidance and Control* 1.4 (1978), pp. 282–284.
- [4] Guoquan P Huang, Anastasios I Mourikis, and Stergios I Roumeliotis. “Analysis and improvement of the consistency of extended Kalman filter based SLAM”. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 473–479.
- [5] Shoudong Huang and Gamini Dissanayake. “Convergence analysis for extended Kalman filter based SLAM”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE. 2006, pp. 412–417.
- [6] Ern J Lefferts, F Landis Markley, and Malcolm D Shuster. “Kalman filtering for spacecraft attitude estimation”. In: *Journal of Guidance, Control, and Dynamics* 5.5 (1982), pp. 417–429.
- [7] Winfried Lohmiller and Jean-Jacques Slotine. “Contraction analysis of non-linear Hamiltonian systems”. In: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, pp. 6586–6592.
- [8] Winfried Lohmiller and Jean-Jacques E Slotine. “On contraction analysis for non-linear systems”. In: *Automatica* 34.6 (1998), pp. 683–696.
- [9] F Landis Markley. “Attitude error representations for Kalman filtering”. In: *Journal of guidance, control, and dynamics* 26.2 (2003), pp. 311–317.

- [10] Feng Tan, Winfried Lohmiller, and Jean-Jacques Slotine. “Analytical SLAM without linearization”. In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1554–1578.
- [11] Mohammad Zamani, Jochen Trumpf, and R Mahony. “Nonlinear attitude filtering: A comparison study”. In: *arXiv preprint arXiv:1502.03990* (2015).

Appendix A.

Code listing

A.1. MEKF and RIEKF implementation for nonlinear attitude filtering

```
1  """
2  @author: thilogen
3  """
4
5  # Import the necessary modules
6
7  import numpy as np
8  from numpy import random
9  import matplotlib as mpl
10 import matplotlib.pyplot as plt
11
12 #-----
13
14 # Define some functions needed to calculate the necessary equations
15
16 def p_sym(M):
17     return 0.5*(M + M.T)
18 def skewm(r):
19     return np.array([[0,-r[2],r[1]], [r[2],0,-r[0]], [-r[1],r[0],0]])
20
21 def q2R(q):
22     s = q[0]; v = q[1:4]; V = skewm(v)
23     return np.eye(3) + 2*s*V + 2*V@V
24
25
26 def qprod(q1, q2):
27     if q1.shape[0] == 4:
28         s1 = q1[0]; v1 = q1[1:4]
29     elif q1.shape[0] == 3:
30         s1 = 0; v1 = q1[0:3]
31     if q2.shape[0] == 4:
```

```

32     s2 = q2[0]; v2 = q2[1:4]
33     elif q2.shape[0] == 3:
34         s2 = 0; v2 = q2[0:3]
35     return np.block([s1*s2 - np.dot(v1,v2), s1*v2 + s2*v1 + np.cross
36         (v1,v2)])
37
38 def expq(u):
39     un = np.linalg.norm(u); phi = np.arcsin(un)
40     s = np.cos(phi); v = np.sinc(phi/np.pi)*u
41     return np.block([s, v])
42
43 #-----
44
45 # Define the function that creates the true trajectory
46
47 def real_world_dynamics():
48     X = np.zeros((n_x, N_samples));
49     X[0:4, 0] = q0; X[4:7, 0] = b0
50     Om_m = np.zeros((3, N_samples))
51     Y = np.zeros((n_y, n_m, N_samples))
52     R0 = q2R(q0);
53     om = np.pi/4/N_time*ez
54     Om_m[:,0] = om
55     for i in range(0, n_m):
56         Y[:,i,0] = R0.T@d[:, i]
57     for k in range(1, N_samples):
58         om = 0.5*ez*np.sin(k*h * 2*np.pi/5)
59         X[0:4,k] = qprod(X[0:4,k-1], expq(0.5*h*om))
60         X[4:7,k] = X[4:7,k-1] + h*random.normal(scale=1*sigma_b,
61             size=(3))
62         Om_m[:,k] = om + X[4:7,k]
63         for i in range(0,n_m):
64             Y[:,i,k] = q2R(X[0:4,k]).T @d[:, i] \
65                 + random.normal(scale=1*sigma_y, size=(3))
66     return X, Om_m, Y
67
68 #-----
69 # The algorithms for the MEKF and RIEKF are presented here in this
70 # section
71
72 def mekf(xh, P, om_m, y):
73     qh = xh[0:4]; bh = xh[4:7]; y1 = y[:,0]; y2 = y[:,1]; y3 = y
74    [:,2]
75     yh1 = q2R(qh).T@d[:,0]; yh2 = q2R(qh).T@d[:,1]; yh3 = q2R(qh).
76     T@d[:,2]
77     Pa = P[0:3, 0:3]; Pc = P[0:3, 3:6]; Pb = P[3:6, 3:6]
78     Sh1 = skewm(yh1); Sh2 = skewm(yh2); Sh3 = skewm(yh3);
79     Rc_inv = np.linalg.pinv(Rc)
80     delta = Sh1 @ Rc_inv @ (yh1-y1) + Sh2 @ Rc_inv @ (yh2-y2)+ Sh3 @

```

```

    Rc_inv @ (yh3-y3)
78     S = - Sh1@Rc_inv@Sh1 - Sh2@Rc_inv@Sh2 - Sh3@Rc_inv@Sh3
79     omh = om_m - bh
80     iqh = expq(0.5*h*(omh + Pa@delta))
81     dbh = Pc.T @ delta
82     dPa = 2*p_sym(Pa@skewm(omh) - Pc) + Qa - Pa@S@Pa
83     dPb = Qb - Pc.T@S@Pc
84     dPc = -skewm(omh)@Pc - Pb + Qc - Pa@S@Pc
85     qh = qprod(qh, iqh); bh = bh + h*dbh
86     Pa = Pa + h*dPa; Pb = Pb + h*dPb; Pc = Pc + h*dPc
87     xh = np.block([qh, bh])
88     P = np.block([[Pa, Pc], [Pc.T, Pb]])
89     return xh, P
90
91 def riekf(xh, P, om_m, y):
92     qh = xh[0:4]; bh = xh[4:7]; y1 = y[:,0]; y2 = y[:,1]; y3 = y
   [:,2]
93     d1 = d[:,0]; d2 = d[:,1]; d3 = d[:,2]
94     Rh = q2R(qh)
95     yh1 = Rh @ y1; yh2 = Rh @ y2; yh3 = Rh @ y3
96     Pa = P[0:3, 0:3]; Pc = P[0:3, 3:6]; Pb = P[3:6, 3:6]
97     D1 = skewm(d1); D2 = skewm(d2); D3 = skewm(d3);
98     Rc_inv = np.linalg.inv(Rc)
99     delta = D1@Rc_inv@(d1-yh1) + D2@Rc_inv@(d2-yh2) + D3@Rc_inv@(d3-
    yh3)
100    S = - D1@Rc_inv@D1 - D2@Rc_inv@D2 - D3@Rc_inv@D3
101    omh = om_m - bh
102    iqh = expq(0.5*h*(omh + Pa@delta))
103    dbh = Rh.T @ Pc.T @ delta
104    dPa = -2*p_sym(Pc) + Qa - Pa@S@Pa
105    dPb = 2*p_sym(skewm(Rh@omh) @ Pb) + Qb - Pc.T@S@Pc
106    dPc = -Pc@skewm(Rh@omh) - Pb + Qc - Pa@S@Pc
107    qh = qprod(qh, iqh); bh = bh + h*dbh
108    Pa = Pa + h*dPa; Pb = Pb + h*dPb; Pc = Pc + h*dPc
109    xh = np.block([qh, bh])
110    P = np.block([[Pa, Pc], [Pc.T, Pb]])
111    return xh, P
112
113
114 #-----
115
116 # Define the system initialization parameters
117
118 ex = np.array([1,0,0]); ey = np.array([0,1,0]); ez = np.array
    ([0,0,1])
119 h = 0.001; N_time = 30; N_samples = int(N_time/h);
120 n_x = 7; n_y = 3; n_m = 2
121
122
123 d = np.block([ex.reshape(3,1), 0*ey.reshape(3,1), ez.reshape(3,1)])
124 n_m = d.shape[1]

```



```

125
126 #-----
127
128 # Define the noise initialization parameters
129
130 d2r = np.pi/180
131 sigma_q0 = 60*d2r; sigma_b0 = 20*d2r
132 q0 = np.array([1, 0, 0, 0])
133 b0 = np.array([0, -0.05, 0.01]) # random.normal(scale=0*sigma_b0, size
    =(3))
134 Pa0 = np.eye(3)/sigma_q0**2; Pb0 = np.eye(3)/sigma_b0**2; Pc0 = np.
    zeros((3,3))
135
136 sigma_b = 0.1*d2r; sigma_om = 25*d2r
137 Qa = np.eye(3)*sigma_om**2; Qb = np.eye(3)*sigma_b**2; Qc = np.zeros
    ((3,3))
138 sigma_y = 30*d2r
139 Rc = np.eye(3)*sigma_y**2
140
141 #-----
142
143 # Create the true trajectory
144 X, Om_m, Y = real_world_dynamics()
145
146 #-----
147
148 # Define and run the simulation for 15 Monte Carlo runs
149
150 runs = 15
151
152 mekf_final = 0
153 riekf_final = 0
154
155 for trial in range(runs):
156     Xh_mekf = np.zeros((n_x, N_samples));
157     Xh_riekf = np.zeros((n_x, N_samples));
158     Xh_mekf[0:4, 0] = q0; Xh_mekf[4:7, 0] = b0
159     Xh_riekf[0:4, 0] = q0; Xh_riekf[4:7, 0] = b0
160
161     P_mekf = np.block([[Pa0, Pc0],
162                        [Pc0.T, Pb0]])
163     P_riekf = np.block([[0.1*Pa0, Pc0],
164                        [Pc0.T, Pb0]])
165
166     for k in range(1, N_samples):
167         Xh_mekf[:,k], P_mekf = mekf(Xh_mekf[:,k-1].copy(), P_mekf,
168 Om_m[:,k-1], Y[:,k-1])
169         #for j in range(1, N_samples):
170             Xh_riekf[:,k], P_riekf = riekf(Xh_riekf[:,k-1].copy(),
171 P_riekf, Om_m[:,k-1], Y[:,k-1])

```

```

171     print(f"Trial {trial+1} finished. {trial+1}/{runs}...")
172
173     mekf_final += Xh_mekf
174     riekf_final += Xh_riekf
175
176 #-----
177
178 # Calculate the average
179
180 mekf_final /= runs
181 riekf_final /= runs
182
183 t = h*np.arange(N_samples)
184 mekf_error = X - mekf_final
185 iekf_error = X - riekf_final
186
187 #-----
188
189 # Plot the MEKF bias
190
191 plt.figure(1)
192 plt.figure(1).clear()
193 plt.plot(t, X[5,:], 'rd', t, mekf_final[5,:], 'b')
194 plt.xlabel("time [s]")
195 plt.ylabel("Bias [rad/s]")
196 plt.title("MEKF bias estimation")
197 plt.legend(("Input bias",
198            "Estimated bias"))
199 plt.show()
200 #-----
201
202 # Plot the RIEKF bias
203
204 plt.figure(2)
205 plt.figure(2).clear()
206 plt.plot(t, X[5,:], 'rd', t, riekf_final[5,:], 'b')
207 plt.xlabel("time [s]")
208 plt.ylabel("Bias [rad/s]")
209 plt.title("RIEKF bias estimation")
210 plt.legend(("Input bias",
211            "Estimated bias"))
212 plt.show()
213 #-----
214
215 # Plot the MEKF angle rotation
216
217 plt.figure(3)
218 plt.figure(3).clear()
219 plt.plot(t, X[3,:], 'rd', t, mekf_final[3,:], 'b')
220 plt.xlabel("time [s]")
221 plt.ylabel("Angle [rad]")

```

```

222 plt.title("MEKF angle rotation estimation")
223 plt.legend(("True trajectory",
224            "Estimated trajectory"))
225 plt.show()
226 #-----
227
228 # Plot the RIEKF angle rotation
229
230 plt.figure(4)
231 plt.figure(4).clear()
232 plt.plot(t, X[3,:], 'rd', t, riekf_final[3,:], 'b')
233 plt.xlabel("time [s]")
234 plt.ylabel("Angle [rad]")
235 plt.title("RIEKF angle rotation estimation")
236 plt.legend(("True trajectory",
237            "Estimated trajectory"))
238 plt.show()
239 #-----
240
241 # Define the rotation angle error for the MEKF and RIEKF
242
243 plt.figure(5)
244 plt.figure(5).clear()
245 plt.plot(t, np.log1p(mekf_error[3,:]), "b")
246 plt.plot(t, np.log1p(iekf_error[3,:]), "r")
247 plt.xlabel("time [s]")
248 plt.ylabel("Angle error [rad]")
249 plt.legend(("MEKF error",
250            "RIEKF error"))
251 plt.title("Error of rotation angle estimation")
252 plt.show()
253 #-----
254
255 # Define the bias error for the MEKF and RIEKF
256
257 plt.figure(6)
258 plt.figure(6).clear()
259 plt.plot(t, np.log1p(mekf_error[5,:]), "b")
260 plt.plot(t, np.log1p(iekf_error[5,:]), "r")
261 plt.xlabel("time [s]")
262 plt.ylabel("Bias error [rad/s]")
263 plt.legend(("MEKF error",
264            "RIEKF error"))
265 plt.title("Error of bias estimation")
266 plt.show()

```

A.2. EKF- and IEKF SLAM implementation

```

1 """
2 @author: thilogen
3 """

```

```

4
5 # Import the necessary modules
6
7 import numpy as np
8 import matplotlib as mpl
9 import matplotlib.pyplot as plt
10
11 from matplotlib.patches import Ellipse
12 import matplotlib.transforms as transforms
13
14 #-----
15
16 # A function to plot the confidence interval of a landmark is
    defined
17
18 def confidence_ellipse(x, y, cov, ax, n_std=3.0, facecolor='none',
    **kwargs):
19     """
20     Create a plot of the covariance confidence ellipse of *x* and *y*
    *.
21
22     Parameters
23     -----
24     x, y : array-like, shape (n, )
25         Input data.
26
27     ax : matplotlib.axes.Axes
28         The axes object to draw the ellipse into.
29
30     n_std : float
31         The number of standard deviations to determine the ellipse's
    radiuses.
32
33     **kwargs
34         Forwarded to '~matplotlib.patches.Ellipse'
35
36     Returns
37     -----
38     matplotlib.patches.Ellipse
39     """
40     pearson = cov[0, 1]/np.sqrt(cov[0, 0] * cov[1, 1])
41     # Using a special case to obtain the eigenvalues of this
42     # two-dimensionl dataset.
43     ell_radius_x = np.sqrt(1 + pearson)
44     ell_radius_y = np.sqrt(1 - pearson)
45     ellipse = Ellipse((0, 0), width=ell_radius_x * 2, height=
    ell_radius_y * 2,
46                       facecolor=facecolor, **kwargs)
47
48     # Calculating the stdandard deviation of x from
49     # the squareroot of the variance and multiplying

```

```

50     # with the given number of standard deviations.
51     scale_x = np.sqrt(cov[0, 0]) * n_std
52
53     # calculating the standard deviation of y ...
54     scale_y = np.sqrt(cov[1, 1]) * n_std
55
56     transf = transforms.Affine2D() \
57         .rotate_deg(45) \
58         .scale(scale_x, scale_y) \
59         .translate(x, y)
60
61     ellipse.set_transform(transf + ax.transData)
62     return ax.add_patch(ellipse)
63
64 #-----
65
66 # Define some functions needed to calculate the necessary equations
67
68 def skew2(a):
69     return a*np.array([[0, -1],[1, 0]])
70 def exp2(theta):
71     theta = theta.item()
72     return np.array([[np.cos(theta), -np.sin(theta)],
73                     [np.sin(theta), np.cos(theta)]])
74 def Ese2(theta):
75     a = np.sinc(theta/np.pi)
76     b = (theta/2)*(np.square(np.sinc(theta/(2*np.pi))))
77     return np.array([[a, -b], [b, a]])
78
79 def integrate_se2(h, x, v, om):
80     vv = np.array([v,0])
81     x_int = np.block([x[0] + h*om,
82                      x[1:3] + exp2(x[0]) @ Ese2(om*h) @ (vv*h)])
83     return x_int
84
85 #-----
86
87 # Define the function that creates the real trajectory and real
88 # landmark positions
89
90 def generate_map(v, omega, delta_r, n_L):
91     p_L = np.zeros((2,n_L)); r = v/omega
92     for i in range(0,n_L):
93         rho = r + delta_r
94         theta = 2*np.pi*i/n_L
95         p_L[:,i] = [rho*np.cos(theta), rho*np.sin(theta) + r]
96     return p_L
97
98 def real_world_dynamics(n_time, h, v, om, p_L):
99     n_L = np.size(p_L,1)
100     X = np.zeros((3,n_time))

```

```

101     Y = np.zeros((2, n_L, n_time))
102     for i in range(0,n_L):
103         Y[:,i,0] = p_L[:,i] - X[1:3,0]
104     for k in range(1, n_time):
105         X[0:3,k] = integrate_se2(h, X[0:3,k-1], v, om)
106         for i in range(0,n_L):
107             Y[:,i,k] = exp2(X[0,k]).T @ (p_L[:,i] - X[1:3,k])
108     return X, Y
109
110 #-----
111
112 # The algorithms for the EKF and IEKF are presented here in this
113 # section
114
115 def propagation_iekf(h, Xh, v, om, P, Q):
116     n_state = np.size(Xh)
117     Rh = exp2(Xh[0])
118     A = np.eye(n_state)
119     G = np.zeros((n_state,n_state)); G[0,0] = 1; G[1:3,1:3] = Rh
120     for i in range(0,n_L):
121         G[2*i+1: 2*i+2+1, 0] = -skew2(1) @ Xh[2*i+1: 2*i+2+1]
122     Xh[0:3] = integrate_se2(h, Xh[0:3], v, om)
123
124     Pp = A @ P @ A.T + G @ Q @ G.T
125     return Xh, Pp
126
127 def update_iekf(Xh, Pp, Yk, p_L, Rn):
128     Rp = exp2(Xh[0]); xh = Xh[1:3]
129     C = np.zeros((2*n_L, 3 + 2*n_L))
130     for i in range(0,n_L):
131         C[2*i:2*i+2, 1:3] = -Rp.T
132         C[2*i:2*i+2, 2*i+3:2*i+3+2] = Rp.T
133     S = C @ Pp @ C.T + Rn
134     K = Pp @ C.T @ np.linalg.inv(S)
135     y = np.zeros(2*n_L)
136     for i in range(0,n_L):
137         y[2*i:2*i+2] = Yk[:,i]
138     yh = np.zeros(2*n_L)
139     for i in range(0,n_L):
140         yh[2*i:2*i+2] = Rp.T @ (Xh[3+2*i: 3+2*i+2] - xh)
141     xi = K @ (y - yh)
142     R_xi = exp2(xi[0]); E_xi = Ese2(xi[0])
143     Xh[0] = Xh[0] + xi[0]
144     Xh[1:3] = R_xi @ Xh[1:3] + E_xi @ xi[1:3]
145     for i in range(0,n_L):
146         Xh[3+2*i: 3+2*i+2] = R_xi @ Xh[3+2*i:3+2*i+2] + E_xi @ xi
147         [3+2*i:3+2*i+2]
148     Pu = (np.eye(3+2*n_L) - K @ C) @ Pp
149     return Xh, Pu, K
150
151 def propagation_ekf(h, Xh, v, om, P, Q):

```

```

150     n_state = np.size(Xh); vv = np.array([v,0])
151     Rh = exp2(Xh[0])
152     A = np.eye(n_state); A[1:3,0] = Rh @ skew2(1) @ (h*vv)
153     G = np.zeros((n_state,n_state)); G[0,0] = 1; G[1:3,1:3] = Rh
154     Xh[0:3] = integrate_se2(h, Xh[0:3], v, om)
155     Pp = A @ P @ A.T + G @ Q @ G.T
156     return Xh, Pp
157
158 def update_ekf(Xh, Pp, Yk, p_L, Rn):
159     Rp = exp2(Xh[0]); xh = Xh[1:3]
160     M = - skew2(1) @ Rp.T
161     C = np.zeros((2*n_L, 3 + 2*n_L))
162     for i in range(0,n_L):
163         C[2*i:2*i+2, 0] = M @ (Xh[3+2*i: 3+2*i+2] - xh)
164         C[2*i:2*i+2, 1:3] = -Rp.T
165         C[2*i:2*i+2, 2*i+3:2*i+3+2] = Rp.T
166     S = C @ Pp @ C.T + Rn
167     K = Pp @ C.T @ np.linalg.inv(S)
168     y = np.zeros(2*n_L)
169     for i in range(0,n_L):
170         y[2*i:2*i+2] = Yk[:,i]
171     yh = np.zeros(2*n_L)
172     for i in range(0,n_L):
173         yh[2*i:2*i+2] = Rp.T @ (Xh[3+2*i: 3+2*i+2] - xh)
174     Xhu = Xh + K @ (y - yh)
175     Pu = (np.eye(3+2*n_L) - K @ C) @ Pp
176
177     return Xhu, Pu, K
178
179 #-----
180
181 # Define the system initialization parameters
182
183 v = 1; t_circ = 40; om = 2*np.pi/t_circ; h = 1; n_L = 20
184 n_time = 80
185 n_state = 3 + 2*n_L
186
187 t = h*np.arange(n_time)
188
189 # Create the true trajectory and true landmark positions
190
191 p_L = generate_map(v, om, 1, n_L)
192
193 X, Y = real_world_dynamics(n_time, h, v, om, p_L)
194
195 #-----
196
197 # Define the noise initialization parameters
198
199 Q = np.zeros((n_state,n_state))
200 sigmaq = np.array([0.1, 0.1, 0.1]); Q[0:3,0:3] = np.outer(sigmaq,

```

```

        sigmaq)
201 for i in range(3,3+2*n_L):
202     Q[i,i] = 100
203 Q = Q
204
205 P = Q
206
207 Rn = 0.1*np.diag(np.diag(np.ones(((2*n_L,2*n_L))))))
208
209 #-----
210
211
212 Xh0 = np.zeros((n_state))
213 for i in range(0,n_L):
214     # Apply an offset for the landmarks to confirm convergence
215     Xh0[3+2*i:3+2*i+2] = p_L[:,i] + [2,5]
216
217 ekf_Pu = np.zeros((n_state, n_state, n_time))
218 iekf_Pu = np.zeros((n_state, n_state, n_time))
219
220 ekf_Xhp = np.zeros((n_state, n_time))
221 ekf_Xhu = np.zeros((n_state, n_time))
222 ekf_Xhp[:,0] = Xh0.copy()
223 ekf_Xhu[:,0] = Xh0.copy()
224
225 iekf_Xhp = np.zeros((n_state, n_time))
226 iekf_Xhu = np.zeros((n_state, n_time))
227 iekf_Xhp[:,0] = Xh0.copy()
228 iekf_Xhu[:,0] = Xh0.copy()
229
230 #-----
231
232 for k in range(1, n_time):
233     ekf_Xhp[:,k], Pp = propagation_ekf(h, ekf_Xhu[:,k-1].copy(), v,
        om, P, Q)
234     ekf_Xhu[:,k], ekf_Pu[:, :, k], K = update_ekf(ekf_Xhp[:,k].copy(),
        Pp, Y[:, :, k], p_L, Rn)
235
236 for k in range(1, n_time):
237     iekf_Xhp[:,k], Pp = propagation_iekf(h, iekf_Xhu[:,k-1].copy(),
        v, om, P, Q)
238     iekf_Xhu[:,k], iekf_Pu[:, :, k], K = update_iekf(iekf_Xhp[:,k].
        copy(), Pp, Y[:, :, k], p_L, Rn)
239
240 #-----
241
242 # Plot the EKF SLAM
243
244 plt.figure()
245 ax = plt.gca()
246 plt.plot(X[1,:], X[2,:], "b", ekf_Xhu[1,:], ekf_Xhu[2,:], 'r--', p_L

```



```

    [0,:],p_L[1,:], 'bo')
247
248 for l in range(n_L):
249     plt.plot(ekf_Xhu[3+2*l,:][-1], ekf_Xhu[3+2*l+1,:][-1], "rx")
250     confidence_ellipse(ekf_Xhu[3+2*l,:][-1],
251                        ekf_Xhu[3+2*l+1,:][-1],
252                        ekf_Pu[3+2*l:5+2*l,3+2*l:5+2*l, -1],
253                        ax, edgecolor="r", n_std=0.5)
254
255
256 plt.title("The EKF SLAM")
257
258 plt.legend(("Trajectory of robot",
259            "Estimated trajectory",
260            "Landmark positions",
261            "Estimated landmark"))
262
263 plt.xlabel("x [m]")
264 plt.ylabel("y [m]")
265
266 plt.show()
267
268 #-----
269
270 # Plot the EKF SLAM with landmark trajectory visible
271
272 plt.figure()
273 ax = plt.gca()
274 plt.plot(X[1,:], X[2:], "b" ,ekf_Xhu[1,:], ekf_Xhu[2,:], 'r--', p_L
275         [0,:],p_L[1,:], 'bo')
276
277 for l in range(n_L):
278     plt.plot(ekf_Xhu[3+2*l,:], ekf_Xhu[3+2*l+1,:], "k--")
279     plt.plot(ekf_Xhu[3+2*l,:][0], ekf_Xhu[3+2*l+1,:][0], "gx")
280     plt.plot(ekf_Xhu[3+2*l,:][-1], ekf_Xhu[3+2*l+1,:][-1], "rx")
281
282
283 plt.title("The EKF SLAM with landmark trajectory")
284
285 plt.legend(("Trajectory of robot",
286            "Estimated trajectory",
287            "True landmark positions",
288            "Landmark trajectory",
289            "Initial landmark position",
290            "Estimated landmark position"))
291
292 plt.xlabel("x [m]")
293 plt.ylabel("y [m]")
294
295 plt.show()

```

```

296
297 #-----
298
299 # Plot the IEKF SLAM
300
301 plt.figure()
302 ax = plt.gca()
303 plt.plot(X[1,:], X[2:], "b" , iekf_Xhu[1,:], iekf_Xhu[2,:], 'r--',
304         p_L[0,:], p_L[1,:], 'bo')
305
306 for l in range(n_L):
307     plt.plot(iekf_Xhu[3+2*l,:][-1], iekf_Xhu[3+2*l+1,:][-1], "rx")
308     confidence_ellipse(iekf_Xhu[3+2*l,:][-1],
309                        iekf_Xhu[3+2*l+1,:][-1],
310                        iekf_Pu[3+2*l:5+2*l, 3+2*l:5+2*l, -1],
311                        ax, edgecolor="r", n_std=0.5)
312
313 plt.legend(("Trajectory of robot",
314            "Estimated trajectory",
315            "Landmark positions",
316            "Estimated landmark"))
317 plt.title("The IEKF SLAM")
318
319 plt.xlabel("x [m]")
320 plt.ylabel("y [m]")
321
322 plt.show()
323 #-----
324
325
326 # Plot the IEKF SLAM with landmark trajectory visible
327
328 plt.figure()
329 ax = plt.gca()
330 plt.plot(X[1,:], X[2:], "b" , iekf_Xhu[1,:], iekf_Xhu[2,:], 'r--',
331         p_L[0,:], p_L[1,:], 'bo')
332
333 for l in range(n_L):
334     plt.plot(iekf_Xhu[3+2*l,:], iekf_Xhu[3+2*l+1,:], "k--")
335     plt.plot(iekf_Xhu[3+2*l,:][0], iekf_Xhu[3+2*l+1,:][0], "gx")
336     plt.plot(iekf_Xhu[3+2*l,:][-1], iekf_Xhu[3+2*l+1,:][-1], "rx")
337
338
339 plt.title("The IEKF SLAM with landmark trajectory")
340
341 plt.legend(("Trajectory of robot",
342            "Estimated trajectory",
343            "True landmark positions",
344            "Landmark trajectory",

```

```

345         "Initial landmark position",
346         "Estimated landmark position"))
347
348 plt.xlabel("x [m]")
349 plt.ylabel("y [m]")
350
351 plt.show()

```

A.3. Analytical SLAM implementation

```

1  """
2  @author: thilogen
3  """
4
5  # Import the necessary modules
6
7  import numpy as np
8  import math
9  import matplotlib.pyplot as plt
10
11  #-----
12
13  # Define some functions needed to calculate the necessary equations
14
15  def skew2(a):
16      return a*np.array([[0, -1],[1, 0]])
17
18  def exp2(theta):
19      #theta = theta.item()
20      return np.array([[np.cos(theta), -np.sin(theta)],
21                      [np.sin(theta), np.cos(theta)]])
22
23  def Ese2(theta):
24      a = np.sinc(theta/np.pi)
25      b = (theta/2)*(np.square(np.sinc(theta/(2*np.pi))))
26      return np.array([[a, -b], [b, a]])
27
28  def integrate_se2(h, x, v, om):
29      vv = np.array([v,0])
30      x_int = np.block([x[0] + h*om,
31                      x[1:3] + exp2(x[0]) @ Ese2(om*h) @ (vv*h)])
32      return x_int
33
34  #-----
35
36  # Define the function that creates the real trajectory and real
37  # landmark positions
38
39  def generate_map(v, omega, delta_r, n_L):
40      p_L = np.zeros((2,n_L)); r = v/omega
41      for i in range(0,n_L):

```

```

42     rho = r + delta_r
43     theta = 2*np.pi*i/n_L
44     p_L[:,i] = [rho*np.cos(theta), rho*np.sin(theta) + r]
45     return p_L
46
47 def real_world_dynamics(n_time, h, v, om, p_L):
48     n_L = np.size(p_L,1)
49     X = np.zeros((3,n_time))
50     Y = np.zeros((2, n_L, n_time))
51     for i in range(0,n_L):
52         Y[:,i,0] = p_L[:,i] - X[1:3,0]
53     for k in range(1, n_time):
54         X[0:3,k] = integrate_se2(h, X[0:3,k-1], v, om)
55         for i in range(0,n_L):
56             Y[:,i,k] = exp2(X[0,k]).T @ (p_L[:,i] - X[1:3,k])
57     return X, Y
58
59 #-----
60
61 # Define the system initialization parameters
62
63 t_circ = 40
64 v = 1; om = 2*np.pi/t_circ;
65 h = 0.1; N_iter = 2*int(t_circ/h)
66 n_L = 20; n_state = 2*n_L + 2;
67
68 t = h*np.arange(N_iter)
69
70
71 #-----
72
73 # Create the true trajectory and true landmark positions
74
75 p_L = generate_map(v, om, 1, n_L)
76
77 X,Y = real_world_dynamics(N_iter, h, v, om, p_L)
78
79 beta_list = X[0]
80
81
82 #-----
83
84 # Here we have a function which will give the simulated results for
85 # the case
86 # that has been chosen to be simulated.
87
88 def case(config):
89
90     if config == "b":
91
92         # Define the noise initialization parameters for the chosen

```

```

102     case
103
104     pu = np.eye(n_state)*0.2
105     pu[-2:, -2:] = np.eye(2)*1e-4
106
107     R = np.eye(n_L)
108
109     # Define the algorithm for the chosen case
110
111     xu = np.zeros((n_state, N_iter));
112     for i in range(n_L):
113         xu[2*i:2*i+2,0] = p_L[:,i] + [2,2]
114
115     theta_list = np.zeros((n_L, N_iter))
116     for itr in range(N_iter):
117         for l in range(n_L):
118             x_lG, y_lG = exp2(-beta_list[itr]) @ (p_L[:,l] - X
119 [1:3,itr])
120
121             theta_i = math.atan2(y_lG, x_lG)
122             theta_list[l,itr] = theta_i
123
124     for itr in range(N_iter-1):
125
126         theta = theta_list[:,itr]
127         beta = beta_list[itr]
128
129         T_beta = exp2(beta).T
130         u = np.zeros(n_state)
131         u[-2:] = np.array([v*np.cos(beta), v*np.sin(beta)])
132
133         H = np.zeros((n_L, n_state))
134         for i in range(n_L):
135             Hi = np.array([np.sin(theta[i]), - np.cos(theta[i])
136 ])
137
138             H[i, 2*i:2*i+2] = Hi @ T_beta
139             H[i, -2:] = - Hi @ T_beta
140
141         K = pu @ H.T @ np.linalg.inv(R)
142
143         d_xu = u + K @ (- H @ xu[:,itr])
144         xu[:,itr+1] = xu[:,itr] + h * d_xu
145     return xu, config
146 #-----
147
148 elif config == "rb":
149
150     #Define the noise intialization parameters for the chosen
151     case

```

```

139     pu = np.eye(n_state)*0.2
140     pu[-2:, -2:] = np.eye(2)*1e-4
141
142
143     R = np.eye(2*n_L)
144
145     # Define the algorithm for the chosen case
146
147     xu = np.zeros((n_state, N_iter));
148     for i in range(n_L):
149         xu[2*i:2*i+2,0] = p_L[:,i] + [2,2]
150
151     theta_list = np.zeros((n_L, N_iter))
152     r_list = np.zeros((n_L, N_iter))
153     y_m = np.zeros((2*n_L, N_iter))
154
155     for itr in range(N_iter):
156         for l in range(n_L):
157             x_lG, y_lG = exp2(-beta_list[itr]) @ (p_L[:,l] - X
158 [1:3,itr])
159
160             theta_i = math.atan2(y_lG, x_lG)
161             r_i = np.sqrt(x_lG**2 + y_lG**2)
162
163             theta_list[l,itr] = theta_i
164             r_list[l,itr] = r_i
165
166             y_m[2*l+1,itr] = r_list[l,itr]
167
168     for itr in range(N_iter-1):
169
170         theta = theta_list[:,itr]
171         y = y_m[:,itr]
172         beta = beta_list[itr]
173
174         T_beta = exp2(beta).T
175         u = np.zeros(n_state)
176         u[-2:] = np.array([v*np.cos(beta), v*np.sin(beta)])
177
178
179         H = np.zeros((2*n_L, n_state))
180         for i in range(n_L):
181             Hi = np.array([[np.sin(theta[i]), - np.cos(theta[i])
182 ],
183                             [np.cos(theta[i]), np.sin(theta[i])
184 ]])
185
186             H[2*i:2*i+2,2*i:2*i+2] = Hi @ T_beta
187             H[2*i:2*i+2,-2:] = - Hi @ T_beta
188
189         K = pu @ H.T @ np.linalg.inv(R)

```

```

187         d_xu = u + K @ (y - H @ xu[:,itr])
188         xu[:,itr+1] = xu[:,itr] + h * d_xu
189
190     return xu, config
191 #-----
192
193     elif config == "db":
194
195         # Define the noise initialization parameters for the chosen
196         case
197
198         pu = np.eye(4)*0.2
199         pu[-2:, -2:] = np.eye(2)*1e-4
200
201         R = np.eye(3)
202
203         # Define the algorithm for the chosen case
204
205         xu = np.zeros((4, n_L, N_iter));
206         for i in range(n_L):
207             xu[0:2,i,0] = p_L[:,i] + [2,2]
208
209         X_vc = np.zeros((2,N_iter))
210
211         theta_list = np.zeros((n_L, N_iter))
212         r_list = np.zeros((n_L, N_iter))
213         y_m = np.zeros((3, n_L, N_iter))
214         for itr in range(N_iter):
215             for l in range(n_L):
216                 x_lG, y_lG = exp2(-beta_list[itr]) @ (p_L[:,l] - X
217                 [1:3,itr])
218
219                 theta_i = math.atan2(y_lG, x_lG)
220                 theta_list[l,itr] = theta_i
221
222         for itr in range(N_iter-1):
223
224             theta = theta_list[:,itr]
225             beta = beta_list[itr]
226
227             T_beta = exp2(beta).T
228             u = np.array([0, 0, v*np.cos(beta), v*np.sin(beta)])
229
230             H = np.zeros((3,4))
231             X_vc_tot = 0
232
233             for i in range(n_L):
234                 y_m[1:3, i, itr] = X_vc[:,itr]
235                 y = y_m[:, i, itr]

```

```

236         Hi = np.array([np.sin(theta[i]), - np.cos(theta[i])
237     ])
238
239     H[0, 0:2] = Hi @ T_beta
240     H[0, 2:4:] = - Hi @ T_beta
241     H[1:3, -2:] = np.eye(2)
242
243
244     K = pu @ H.T @ np.linalg.inv(R)
245     d_xu = u + K @ (y - H @ xu[:, i, itr])
246     xu[:, i, itr+1] = xu[:, i, itr] + h * d_xu
247     X_vc_tot += xu[2:4, i, itr+1]
248     X_vc[:,itr+1] = X_vc_tot / n_L
249
250     return xu, config, X_vc
251
252 # -----
253
254
255     elif config == "drb":
256
257         # Define the noise initialization parameters for the chosen
258         case
259
260         pu = np.eye(4)*0.2
261         pu[-2:, -2:] = np.eye(2)*1e-4
262
263         R = 0.1*np.eye(4)
264
265         # Define the algorithm for the chosen case
266
267         xu = np.zeros((4, n_L, N_iter));
268         for i in range(n_L):
269             xu[0:2,i,0] = p_L[:,i] + [2,2]
270
271         X_vc = np.zeros((2,N_iter))
272
273         theta_list = np.zeros((n_L, N_iter))
274         r_list = np.zeros((n_L, N_iter))
275         y_m = np.zeros((4, n_L, N_iter))
276         for itr in range(N_iter):
277             for l in range(n_L):
278                 x_lG, y_lG = exp2(-beta_list[itr]) @ (p_L[:,l] - X
279 [1:3,itr])
280
281                 theta_i = math.atan2(y_lG, x_lG)
282                 r_i = np.sqrt(x_lG**2 + y_lG**2)
283
284                 theta_list[l,itr] = theta_i
285                 r_list[l,itr] = r_i

```



```

284         y_m[1,1,itrr] = r_list[1,itrr]
285
286
287
288     for itr in range(N_iter-1):
289
290         theta = theta_list[:,itr]
291         beta = beta_list[itr]
292
293         T_beta = exp2(beta).T
294         u = np.array([0, 0, v*np.cos(beta), v*np.sin(beta)])
295
296
297         H = np.eye(4)
298         X_vc_tot = 0
299
300         for i in range(n_L):
301             y_m[2:4, i, itr] = X_vc[:,itr]
302             y = y_m[:, i, itr]
303
304             Hi = np.array([[np.sin(theta[i]), - np.cos(theta[i])
305 ],
306                             [np.cos(theta[i]), np.sin(theta[i])
307 ]])
308
309             H[0:2, 0:2] = Hi @ T_beta
310             H[0:2, 2:4:] = - Hi @ T_beta
311
312             K = pu @ H.T @ np.linalg.inv(R)
313             d_xu = u + K @ (y - H @ xu[:, i, itr])
314             xu[:, i, itr+1] = xu[:, i, itr] + h * d_xu
315             X_vc_tot += xu[2:4, i, itr+1]
316             X_vc[:,itr+1] = X_vc_tot / n_L
317
318         return xu, config, X_vc
319
320
321
322 """
323
324 This code contains the code necessary to simulate four different
325 cases for the
326 analytical SLAM that was presented in the paper written by Slotine
327 in 2017.
328 The following cases are described as:
329
330 Case 1: Bearing only                - "b"
331 Case 2: Bearing and range measurement - "rb"
332 Case 3: Bearing only with DUNK      - "db"

```

```

331 Case 4: Bearing and range measurement with DUNK - "drb"
332
333 Here the string that is assigned for each case is the input for the
334 simulation
335 that is to be simulated.
336
337
338 # Choose the case we want to simulate. The case presented in the
339 project
340 # thesis is Case 4: Bearing and range measurement with DUNK - "drb"
341 case_val = case("drb")
342 plot_config = case_val[1]
343
344 #-----
345
346 if plot_config == "b" or plot_config == "rb":
347
348     Xhu = case_val[0]
349
350     # Plot the results of the analytical SLAM for the chosen case
351
352     plt.figure()
353     plt.plot(X[1,:], X[2,:], "b", p_L[0,:], p_L[1,:], 'bo')
354     plt.plot(Xhu[-2,:], Xhu[-1,:], "r--")
355     for k in range(n_L):
356         plt.plot(Xhu[2*k,:][-1], Xhu[2*k+1,:][-1], "rx")
357     plt.xlabel("x [m]")
358     plt.ylabel("y [m]")
359
360     plt.legend(("Trajectory of robot",
361               "True landmark positions",
362               "Estimated trajectory",
363               "Estimated landmark position"))
364
365     if plot_config == "rb":
366         plt.title("The analytical SLAM with range measurement and
367 bearing")
368     else:
369         plt.title("The analytical SLAM with bearing only")
370     plt.show()
371
372     # Plot the results of the analytical SLAM with visible landmark
373     trajectory
374     # for the chosen case
375
376     plt.figure()
377     plt.plot(X[1,:], X[2,:], "b", p_L[0,:], p_L[1,:], 'bo')
378     plt.plot(Xhu[-2,:], Xhu[-1,:], "r--")
379     for k in range(n_L):

```

```

378         plt.plot(Xhu[2*k,:], Xhu[2*k+1,:], "k--",
379                 Xhu[2*k,:][-1], Xhu[2*k+1,:][-1], "rx",
380                 Xhu[2*k,:][0], Xhu[2*k+1,:][0], "gx")
381     plt.xlabel("x [m]")
382     plt.ylabel("y [m]")
383
384     plt.legend(("Trajectory of robot",
385               "True landmark positions",
386               "Estimated trajectory",
387               "Landmark trajectory",
388               "Estimated landmark position",
389               "Initial landmark position"))
390
391     plt.title("The analytical SLAM with landmark trajectory")
392
393     plt.show()
394
395     #-----
396
397 elif plot_config == "drb" or plot_config == "db":
398
399     Xhu = case_val[0]; X_vc = case_val[2]
400
401     # Plot the results of the analytical SLAM-DUNK for the chosen
402     case
403
404     plt.figure()
405
406     plt.plot(X[1,:], X[2,:], "b", p_L[0,:], p_L[1,:], 'bo')
407     plt.plot(X_vc[0,:], X_vc[1,:], "r--")
408
409     for k in range(n_L):
410         plt.plot(Xhu[0,k,:][-1], Xhu[1,k,:][-1], "rx")
411
412     if plot_config == "drb":
413         plt.title("The analytical SLAM-DUNK with range measurement
414 and bearing")
415     else:
416         plt.title("The analytical SLAM-DUNK with bearing only")
417
418
419     plt.legend(("Trajectory of robot",
420               "True landmark positions",
421               "Estimated trajectory",
422               "Estimated landmark position"))
423     plt.xlabel("x [m]")
424     plt.ylabel("y [m]")
425
426

```

```

427
428     plt.show()
429
430     # Plot the results of the analytical SLAM-DUNK with visible
431     # landmark
432     # trajectory for the chosen case
433
434     plt.figure()
435
436     plt.plot(X[1,:], X[2,:], "b", p_L[0,:], p_L[1,:], 'bo')
437     plt.plot(X_vc[0,:], X_vc[1,:], "r--")
438
439     for k in range(n_L):
440         plt.plot(Xhu[0,k,:], Xhu[1,k,:], "k--",
441                 Xhu[0,k,:][-1], Xhu[1,k,:][-1], "rx",
442                 Xhu[0,k,:][0], Xhu[1,k,:][0], "gx")
443     plt.title("The analytical SLAM-DUNK with landmark trajectory")
444
445     plt.legend(("Trajectory of robot",
446               "True landmark positions",
447               "Estimated trajectory",
448               "Landmark trajectory",
449               "Estimated landmark position",
450               "Initial landmark position"))
451
452     plt.xlabel("x [m]")
453     plt.ylabel("y [m]")

```