UNIVERSITY OF CALIFORNIA
RIVERSIDE

Implementation of a Distributed Kalman Filter for 3-D Moving Object Tracking by Using IMU/UWB

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Yizhi Zhou

June 2022

Dissertation Committee:

Dr. Wei Ren, Chairperson
Dr. Nanpeng Yu
Dr. Bir Bhanu

The Dissertation of Yizhi Zhou is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to express my greatest gratitude to Professor Wei Ren, my supervisor for his help with my research. Without his guidance, I am unable to complete this thesis.

I would like to give my appreciation to Pengxiang Zhu and Jie Xu for helping me on understanding the algorithms and the data collection during the experiments.

Finally, I want to express my love to my family for the love through my life.

ABSTRACT OF THE DISSERTATION

Implementation of a Distributed Kalman Filter for 3-D Moving Object Tracking by Using IMU/UWB

by

Yizhi Zhou

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, June 2022
Dr. Wei Ren, Chairperson

The focus of this thesis is on the implementation of distributed kalman filters over sensor networks for 3-D moving target tracking. Two distributed kalman filters, quaternion-based distributed extended kalman filter and distributed invariant extended kalman filter, are implemented based on crazyflie quadrotor platform in the local positioning system. In the algorithms, each agent is equipped with the communication and sensing capabilities to cooperatively track a 3-D moving object. The proposed algorithms are fully distributed and robust to time-varying communication topologies. Experimental results based on the multi-agent platform are shown to validate the proposed algorithms. Finally, we compare and analyze the performances of two algorithms based on the results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Distributed estimation is one of the most fundamental problems in sensor networks with a wide range of applications such as target tracking, area monitoring and rescue [11]. Generally, for the problem of target tracking, it is assumed that a state of interest can evolve based on a noisy dynamic model. Measurements related to the state can be obtained by each agent when the target are located in the observation area. The purpose is to estimate the target state (i.e.,orientation and position) on each agent accurately by using the dynamic noisy model and those measurements. For this problem, a centralized algorithm can provide an optimal solution by getting all the measurements in a fusion center and running a centralized Kalman Filter (CKF). But, CKF requires expensive communication from all agents to the fusion center and has heavy computational complexity. Therefore, the centralized algorithm is inapplicable for real-time applications in large sensor networks. In contrast to the centralized way, distributed algorithms only use the local information and

neighbors' information of each agent, which greatly improve the computational efficiency and maintain robustness [6]. Moreover, distributed estimation has advantages in handling with the issue of observation loss [14]. Thus, researchers in the area of control and robotics focus more on the distributed approaches in recent years.

In distributed algorithms, the consensus [9] and the Covariance Intersection (CI) [3] algorithms have been widely employed to obtain a consistent distributed Kalman Filter (DEKF). The objective of CI is to curry out the fusion of weighted information that represents the state and cross-covariances of the agent. A typical CI-based estimator is proposed in, where the CI is used to fuse the prior information pair first and then update it with all the measurements [12]. In addition, some approaches fuse the posterior information pairs among the neighborhoods with CI after updating from its own measurement [5]. However, all those algorithms can only work on vector space and has additive errors. It cannot be applied in most 3D cases which use the quaternion or rotation matrix to represent rotations.

To solve this issue, quaternion-based distributed EKF (QDEKF) [15] is proposed in the recent work to fuse the information of quaternions from other sensors in a sensor network. Specifically, it uses a unit quaternion to represent the 3D orientation and extend CI to the 3D case using "quaternion average". Although the proposed QDEKF shows good performance in target tracking, this algorithm is built upon error-state Kalman Filter which has potential inconsistency and hurts the accuracy [10]. To further address this problem, a distributed invariant EKF is designed to solve the problem of distributed state estimationover sensor networks in 3D environments. In this algorithm, CI algorithm is extend to matrix Lie groups for the first time. The proposed distributed invariant EKF (DIEKF) algorithm improves the

estimation accuracy and consistency compared with QDEKF. The objective of this thesis is to test the effectiveness of both QDEFK and DIEKF and compare their performances in a realistic ultra-wideband (UWB) environment by using quadrotors specified by crazyflies [4, 7].

1.2 Contribution

The contribution of this thesis is the setup of a UWB platform for validating the distributed estimation algorithms with crazyfiles, including the following:

1) the setup of a localization positioning system for crazyflies quadrotors based on UWB, achieving accurate UWB localization for two-way ranging (TWR) and time-difference-of-arrival (TDOA) measurements;

2) the design of an offline data acquisition and processing system for recording raw sensor data from crazyflies to allow for measurement and analysis of data such as IMU, UWB, and Vicon for state estimation;

3) the experimental implementation of the proposed DKF algorithms in a 3D indoor environment by using crazyflies to explore its effectiveness and performance.

1.3 Outlines for Chapters

Chapter 1 serves as the introduction of this thesis which includes the motivation and background of this research. My contributions to this work are mentioned as well.

Chapter 2 demonstrates the details of the proposed algorithms.

Chapter 3 illustrates the experimental platforms used in this thesis.

Chapter 4 demonstrates my work and contributions in this work.

Chapter 5 shows the simulation results and experimental results. It also discuss the performance of the proposed algorithms.

Chapter 6 gives the conclusion and future work of the thesis.

Chapter 2

Algorithm Introduction

2.1 Notations and Preliminaries

2.1.1 Quaternion

In the following, the definition is described in [10]. A quaternion is generally defined as

$$\bar{q} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4 \tag{2.1}$$

where $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are hyperimaginary numbers satisfying

$$\mathbf{i}^2 = -1, \mathbf{j}^2 = -1, \mathbf{k}^2 = -1, \ -\mathbf{ij} = \mathbf{ji} = \mathbf{k}, -\mathbf{jk} = \mathbf{kj} = \mathbf{i}, -\mathbf{ki} = \mathbf{ik} = \mathbf{j}, \tag{2.2}$$

Note that this definition is corresponding to the JPL definition instead of the Hamilton definition. For simplicity, $\bar{q}$ can be further written as a four-dimensional column matrix given by

$$\bar{q} = \begin{bmatrix} \mathbf{q} \\ \\ q_4 \end{bmatrix} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T \tag{2.3}$$

Orientation is represented as a unit quaternion satisfying $|\bar{q}| = \sqrt{|\mathbf{q}|^2 + q_4^2} = 1$. A rotation can be represent as a unit quaternion as

$$\mathbf{q} = \begin{bmatrix} ksin(\theta/2) \\ \\ cos(\theta/2) \end{bmatrix} \tag{2.4}$$

Where $\mathbf{k}$ is a unit vector defining the rotation aixs and θ is the angle of rotation.

In the vector space, the error vector and its covariance can be expressed in terms of the arithmetic difference between the true value and the estimation value. However, we cannot use this error representation for a quaternion. Instead, we will employ the error quaternion $\delta\bar{q}$ to define the error, a small rotation between the estimation value and true value respected to the local frame. This error is written as a multiplication as

$$\bar{q} = \delta\bar{q} \otimes \hat{\bar{q}} \tag{2.5}$$

Since the rotation corresponds to $\delta\bar{q}$ is assumed to be very small value. Its small angle approximation can be further obtained as

$$\delta\bar{q} = \begin{bmatrix} \delta q \\ \\ \delta q_4 \end{bmatrix} = \begin{bmatrix} msin(\delta\theta/2) \\ \\ cos(\delta\theta/2) \end{bmatrix} \approx \begin{bmatrix} \frac{1}{2}\delta\theta \\ \\ 1 \end{bmatrix} \tag{2.6}$$

A rotation can also be described as a rotation matrix R. The relation between a rotation matrix and a unit quaternion is

$$\mathbf{R} = (2q_4^2 - 1)I_3 - 2q_4[q\times] + 2qq^T \tag{2.7}$$

where $[\times]$ represents the skew symmetric matrix. Given a 3×1 vector $q = [q_1, q_2, q_3]^T$, its

skew symmetric matrix is defined as

$$\begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{2.8}$$

2.1.2 Lie Group and Lie Algebra

Here, matrix Lie group theory is employed to derive our algorithm. The material is from **??**. A matrix Lie group is a subset of square invertible $N \times N$ matrices satisfying

$$\begin{aligned} I_N &\in \mathcal{G}, \\ \forall a \in \mathcal{G}, a^{-1} &\in \mathcal{G}, \\ \forall a, b \in \mathcal{G}, ab &\in \mathcal{G}, \end{aligned} \tag{2.9}$$

Its Lie algebra is defined as $\mathfrak{g}$, which is a vector space with the same dimension as $\mathcal{G}$. $(\cdot)^\wedge : \mathbb{R}^{dim\mathfrak{g}} \to \mathfrak{g}$ is denoted as the linear map that transforms the elements in the Lie algebra to the corresponding matrix representation. Moreover, the exponent map is defined as $exp(\xi) = exp_m(\xi^\wedge) \in \mathcal{G}$,where $\xi \in \mathbb{R}^{dim\mathfrak{g}}$ is an element in $\mathfrak{g}$. Correspondingly, the inverse function of the exponent map, logarithm map, is defined as $log(\cdot)$, which satisfies the $log(\cdot) = (log_m(\cdot))^\vee : \mathcal{G} \to \mathbb{R}^{dim\mathfrak{g}}$. $(\cdot)^\vee$ is defined as the inverse operator of $(\cdot)^\wedge$

2.1.3 Notations and Definitions

In this thesis, we denote $\mathbf{0}_{m\times n}$ as a $m\times n$ zeros matrix, and $\mathbf{I}_n$ as a $n\times n$ identity matrix. We denote G, T and C_i respectively as the global frame, target's local frame and the ith sensor frame. Let ${}^T_G\bar{q}$ denotes the target's orientation from G to T, and ${}^T_G\mathbf{R}$ is the

corresponding rotation matrix. ${}^{G}\mathbf{p}_T$, the target's global position, denotes the position of T in G. For vector quantities, the error δx is defined as the standard additive error $\delta x = x - \hat{x}$.

In a sensor network of multiple agents, we define a directed communication graph $\mathcal{G}^k = (\mathcal{V}, \mathcal{E}^k)$, where $\mathcal{V}$ indicates the agent set and $\mathcal{E}^k$ represents the edge set defined as $\mathcal{E}^k \subseteq \mathcal{V} \times \mathcal{V}$. $\mathcal{E}^k$ stands for the communication links between agents at timestep k. We assume that self edge $(i,i) \in \mathcal{E}^k, \forall i \in \mathcal{V}$ always exists in the communication graph. If $(j,i) \in \mathcal{E}^k$, it means that agent j is a neighbor of agent i, and agent i can receive information from agent j. The set of all the communication neighbors of agent i at timestep k can be defined as $\mathcal{N}_i^k = \{i | (l,i) \in \mathcal{E}^k, l \in \mathcal{V}\}$.

2.2 Quaternion-Based Distributed EKF Algorithm

In this section, the algorithm is demonstrated in [15].

2.2.1 Problem Formulation

Consider a network of multiple agents in a 3D environment. Each agent can communicate with its one-hop neighbors and measure the target in a limited sensing region. The state of the target is defined as

$$x = \begin{bmatrix} {}^{T}_{G}\bar{q} \\ \\ x_v \end{bmatrix} = \begin{bmatrix} {}^{T}_{G}\bar{q}^T & {}^{G}p_T^T & {}^{G}v_T^T \end{bmatrix}^T \tag{2.10}$$

which includes the target's 6-DOF pose ${}^{T}_{G}\bar{q}$ and ${}^{G}p_T$, and the linear velocity ${}^{G}v_T$ in global frame. Consider that the target moves according to the following dynamic model

$${}^{T}_{G}\dot{\bar{q}} = \frac{1}{2}\omega^k \otimes {}^{T}_{G}\bar{q}, {}^{G}\dot{p_T} = {}^{G}v_T, {}^{G}\dot{v_T} = {}^{T}_{G}R^T a^k \tag{2.11}$$

where ω and a are the actual local angular velocity and linear acceleration. The corresponding noisy model are given as

$$\omega_m = \omega + n_\omega, a_m = a_G^L Rg + n_a \tag{2.12}$$

where n_ω and n_a are zero mean white Gaussian noise which are assumed to be known to each agent. The local measurement z_i^k obtained by each agent i at timestep k is given by the following nonlinear model with local measurement noise w_i

$$z_i^k = h_i(x^k, w_i^k) \tag{2.13}$$

The objective of our work is to compute an accurate target's state on every agent by using the information from itself and its one-hop communication neighbors.

2.2.2 Distributed Kalman Filter Algorithm

The distributed kalman filter includes two steps: propagation and upate. Suppose that at timestep k, agent i can obtain a prior estimator $(\bar{x}_i^k, \bar{p}_i^k)$ with state and covariance after propagation. In particular, the prior estimation $\bar{x}_i^k$ can be computed according to the following propagation model

$$\begin{aligned}
&{}_G^{T_k}\bar{q} = exp(\frac{1}{2}\Omega\omega_m^{k-1}\Delta t)_G^{T_{k-1}}\hat{\hat{q}}, \\
&{}^G\bar{v}^k = {}^G\hat{v}^{k-1} - {}^G g\Delta t + {}_G^T\hat{R}^T a_m^{k-1}\Delta t, \\
&{}^G\bar{p}^k = {}^G\hat{p}^{k-1} + {}^G\hat{v}^{k-1}\Delta t - \frac{1}{2}g\Delta t^2 + \frac{1}{2}{}_G^T\hat{R}^T a_m^{k-1}\Delta t^2
\end{aligned} \tag{2.14}$$

Then, agent i aims to update its local estimator $(\bar{x}_i^k, \bar{p}_i^k)$ by using the information of itself and one-hop communication neighbors. First, we will fuse all the prior informations among the neighborhood, i.e., $(\bar{x}_i^k, \bar{p}_i^k), \forall j \in \mathcal{N}_i^k$. Note that we cannot compute the

information vector of a quaternion and fuse the information. Therefore, we employ the following method to obtain a closed form solution of the averaged orientations described by quaternion ${}^{T_i}_{G}\check{\bar{q}}$ as

$$\begin{aligned} {}^{T_i}_{G}\check{\bar{q}} &= \arg\max_{\bar{q}\in\mathcal{S}^3} \ \bar{q}^T M \bar{q}, \\ M &= \sum_{j\in\mathcal{N}_i^k} \pi_j^k ({}^{T_j}_{G}\widetilde{\bar{q}})^T \ {}^{T_j}_{G}\widetilde{\bar{q}} \end{aligned} \tag{2.15}$$

where ${}^{T_j}_{G}\widetilde{\bar{q}}$ is agent j's prior estimation of ${}^{T}_{G}\bar{q}$, and $\mathcal{S}^3$ represents the unit 3-sphere. π_j is a parameter that weighted synchronize the prior estimation to avoid the uncertainty. In order to make sure that we don't overuse any information among the neighborhood, the parameter π_j satisfies $\pi_j \in [0,1]$, and $\sum_{j\in\mathcal{N}_i} \pi_j = 1$. We can obtain a quaternion that minimizes the weighted sum of the orientation errors by solving the above equation. Moreover, the fusion of vector quantities x_v in x is computed by

$$\check{x}_{v_i}^k = \sum_{j\in\mathcal{N}_i} \pi_j^k \ \bar{x}_{v_j}^k \tag{2.16}$$

Then we can obtain the weighted average state vector as

$$\check{x}_i^k = \begin{bmatrix} {}^{T}_{G}\check{\bar{q}} \\ \\ \check{x}_{v_i}^k \end{bmatrix} = \sum_{j\in\mathcal{N}_i^k} \pi_j^k \boxtimes \bar{x}_j^k \tag{2.17}$$

where the symbol $\boxtimes$ is defined for computing the weighted average. Due to the quaternion error is calculated by the error of the rotational angle, the covariance can be directly computed in vector quantity as

$$\check{p}_i^k = \sum_{j\in\mathcal{N}_i^k} \pi_j^k \bar{p}_i^k \tag{2.18}$$

We calculate the weight π_j^k to minimize the trace of $\check{p}_i^k$ according to the following simplified

algorithm

$$\pi_j^k = \frac{1/Trace(\bar{p}_j)}{\sum_{j\in\mathcal{N}_i^k} 1/Trace(\bar{p}_j)} \tag{2.19}$$

Second, all the intermediate estimation pair $(\check{x}_i^k, \check{p}_i^k)$ will be fused with its local measurements z_j^k, $\forall j \in \mathcal{N}_i^k$. After linearize the measurement z_i^k on the current estimated state as

$$\tilde{z}_i^k = z_i^k - h_i(\bar{x}_i^k), H_i^k = \frac{\partial h_i(\bar{x}_i^k)}{\partial x_i^k} \tag{2.20}$$

We can compute the updated covariance p_i^k and the state correction $\delta x_i{}^k$ according to

$$\begin{aligned} p_i^k &= [\check{p}_i^k + \sum_{j\in\mathcal{N}_i^k} s_j^k]^{-1}, \\ \delta x_i^k &= \begin{bmatrix} \delta\theta_i^k \\ \\ \delta x_{v_i}^k \end{bmatrix} = p_i^k \sum_{j\in\mathcal{N}_i^k} y_j^k \end{aligned} \tag{2.21}$$

where $\delta\theta_i$ is the orientation correction and δx_{v_i} is the corrections of the vector quantities, and p_i^k and δx_i^k is calculated as

$$\begin{aligned} s_i^k &= (H_i^k)^T (R_i^k)^{-1} H_i^k, \\ y_i^k &= (H_i^k)^T (R_i^k)^{-1} \tilde{z}_i^k \end{aligned} \tag{2.22}$$

Then $\check{x}_i$ can be updated by using δx_i according to

$$\hat{x}_i^k = \begin{bmatrix} {}_G^{T_{i,k}}\hat{\bar{q}} \\ \\ \hat{x}_{v_i}^k \end{bmatrix} = \check{x}_i^k \boxplus \delta x_i^k \tag{2.23}$$

where we define the symbol $\boxplus$ for updating $\check{x_i^k}$. Specifically, the vector quantities $\check{x}_{v_i}$ can be directly updated using $\delta x_{v_i}^k$, and the quaternion ${}_G^{T_{i,k}}\check{\bar{q}}$ is updated by a small rotation error $\delta\bar{q}_i$ according to

$$\begin{aligned} \hat{x}_{v_i}^k &= \check{x}_{v_i}^k + \delta x_{v_i}^k, \\ {}_G^{T_{i,k}}\hat{\bar{q}} &= {}_G^{T_{i,k}}\check{\bar{q}} \otimes \delta\bar{q}_i \end{aligned} \tag{2.24}$$

2.3 Distributed Invariant EKF Algorithm

In the following, the algorithm in this section is described in [13].

2.3.1 Problem Formulation

The state of a target that is tracked by a networks of agents in a 3D environment, is defined as

$$x = \begin{bmatrix} {}^{G}_{T}R & {}^{G}v_{T} & {}^{G}p_{T} \end{bmatrix}^{T} \tag{2.25}$$

which includes the target's orientation ${}^{T}_{G}R$ and position${}^{G}p_{T}$, and the linear velocity ${}^{G}v_{T}$ in global frame. Consider that the target moves according to the following dynamic model

$$\begin{aligned} {}^{T}_{G}\dot{R} &= {}^{T}_{G}R(\omega - n_{\omega})_{\times}, \\ {}^{G}\dot{v} &= {}^{T}_{G}R(a - n_{a}) + g, \\ {}^{G}\dot{p} &= {}^{G}v \end{aligned} \tag{2.26}$$

where ω and a are the corresponding angular velocity and the linear acceleration of the target in the target frame, and n_{ω} and n_{a} are white Gaussian noises. The local measurement z_{i}^{k} obtained by each agent i at timestep k is given by the following nonlinear model with local measurement noise w_{i}

$$z_{i}^{k} = h_{i}(x^{k}, w_{i}^{k}) \tag{2.27}$$

2.3.2 Matrix Lie Group Representation

The state of the target at timestep k is represented by a matrix Lie group $SE(3)$ as

$$X = \begin{bmatrix} {}^{G}_{T}R & {}^{G}v & {}^{G}p \\ 0_{1\times 3} & 1 & 0 \\ 0_{1\times 3} & 0 & 1 \end{bmatrix} \in \mathbb{R}^{6\times 6} \tag{2.28}$$

Let $\hat{X}_k$ be the state estimation, and its right invariant estimation error η is defined as

$$\eta_t = X_t(\hat{X}_t)^{-1} \tag{2.29}$$

The corresponding error vector ξ_k defined in the Lie algebra of $se(3)$ is given by

$$\xi_k = \begin{bmatrix} (\xi_{R_k})^T & (\xi_{v_k})^T & (\xi_{p_k})^T \end{bmatrix}^T \in \mathbb{R}^9 \tag{2.30}$$

where ξ_{R_k}, ξ_{v_k}, and $\xi_{p_k} \in \mathbb{R}^3$ are the corresponding error vectors of rotation, velocity and position. The estimation error $\eta_k$can be computed by

$$\eta_k = exp(\xi_k) = exp_m(\xi_k^{\wedge}) \tag{2.31}$$

where $\xi_k^{\wedge}$ is given as

$$\xi_k^{\wedge} = \begin{bmatrix} (\xi_{R_k})_{\times} & \xi_{v_k} & \xi_{p_k} \\ 0_{1\times 3} & 1 & 0 \\ 0_{1\times 3} & 0 & 1 \end{bmatrix} \in \mathbb{R}^{6\times 6} \tag{2.32}$$

It has been shown that the dynamic model of ξ_k is given as

$$\frac{d}{dt}\xi_k = A_k\xi_k - Ad_{\bar{X}_t}U_t \tag{2.33}$$

where $Ad_{\bar{X}_t}$ is the adjoint of SE(3) at $\bar{X}_t$, and $U_t = [n_{\omega t}^T, n_{at}^T, 0_{1\times 3}]^T$. A_t is represented as

$$A_t = \begin{bmatrix} 0_3 & 0_3 & 0_3 \\ (g)_\times & 0_3 & 0_3 \\ 0_3 & I_3 & 0_3 \end{bmatrix} \tag{2.34}$$

2.3.3 Distributed Kalman Filter Algorithm

Suppose that we know the motion of the target. Let X^k denote the target state at timestep t_k, and $\bar{X}_i^k$ and $\hat{X}_i^k$ denote the agent's prior and posterior estimation of the target state at timestep t_k respectively. Also let $\eta_i^{k|k-1} = X^k(\bar{X}_i^k)^{-1} = exp(\eta_i^{k|k-1})$ which represents the agent's prior error. The agent's posterior error is represented as $\eta_i^{k|k} = X^k(\hat{X}_i^k)^{-1} = exp(\eta_i^{k|k})$. Note that here $\eta_i^{k|k-1}$ and $\eta_i^{k|k}$ is defined in $se(3)$. The covariances of $\eta_i^{k|k-1}$ and $\eta_i^{k|k}$ are defined as $\bar{P}_i^k$ and $\hat{P}_i^k$.

The proposed algorithm includes three steps. The first step is to obtain the prior estimation $(\bar{X}_i^{k-1}, \bar{P}_i^{k-1})$ by propagating the posterior estimation $(\hat{X}_i^{k-1}, \hat{P}_i^{k-1})$ based on the dynamic model. Specifically, the individual states in $\bar{X}_i^k$ can be propagated as

$$\begin{aligned} {}^G_T\bar{R}_i^k &= {}^G_T\hat{R}_i^{k-1}\Gamma_0(\omega^{k-1}\Delta t), \\ {}^G\bar{v}_i^k &= {}^G\hat{v}_i^{k-1} + {}^G_T\hat{R}_i^{k-1}\Gamma_1(\omega^{k-1}\Delta t)a^{k-1}\Delta t + g\Delta t, \\ {}^G\bar{p}_i^k &= {}^G\hat{p}_i^{k-1} + {}^G\hat{v}_i^{k-1}\Delta t + {}^G_T\hat{R}_i^{k-1}\Gamma_2(\omega^{k-1}\Delta t)a^{k-1}\Delta t^2 + \frac{1}{2}g\Delta t^2 \end{aligned} \tag{2.35}$$

The second step is to update the agent's local estimation by fusing the information from its neighbors. The goal is to obtain an intermediate measurement $(\check{X}_i^k, \check{P}_i^k)$. For agent $j \in \mathcal{N}_i$,$\eta_j^{k|k-1}$ denote the prior estimation error in SE(3), and $\xi_j^{k|k-1} = log(\eta_j^{k|k-1})$ is the prior estimation error in se(3).Here we extend CI algorithm in the context of error-state EFK in Lie groups to estimate the error states in se(3). The intermediate estimation $(\check{\xi}_j^{k|k-1}, \check{P}_i^k)$

can be calculated by

$$\begin{aligned}\check{P}_i^k &= [\sum_{j\in\mathcal{N}_i^k} \pi_j^k (\bar{P}_j^k)^{-1}]^{-1}, \\ \check{\xi}_i^{k|k-1} &= \check{P}_i^k [\sum_{j\in\mathcal{N}_i^k} \pi_j^k (\bar{P}_j^k)^{-1} log((\bar{X}_j^k)(\bar{X}_i^k)^{-1})]\end{aligned} \tag{2.36}$$

where the weight π_j^k can be computed according to equation(2.19). Then the intermediate state estimation X^k in SE(3), denoted by $\check{X}_i^k$, can be mapped from $\check{\xi}_i^{k|k-1}$ by

$$\check{X}_i^k = exp(\check{\xi}_i^{k|k-1})\bar{X}_i^k \tag{2.37}$$

The third step is to fuse the agent's intermediate estimation $(\check{X}_i^k, \check{P}_i^k)$ with all the measurements, which includes the measurement from itself and its neighbhors. First, by linearizing the measurement z_i^k on the current estimated state and compute the Jacobian as

$$\tilde{z}_i^k = z_i^k - h_i(\bar{x}_i^k), H_i^k = \frac{\partial h_i(\bar{x}_i^k)}{\partial x_i^k} \tag{2.38}$$

we can compute

$$\begin{aligned}s_i^k &= (H_i^k)^T (R_i^k)^{-1} H_i^k, \\ y_i^k &= (H_i^k)^T (R_i^k)^{-1} \tilde{z}_i^k\end{aligned} \tag{2.39}$$

Then the updated covariance $\hat{P}_i^k$ and the posterior estimate of $\hat{\xi}_i^{k|k-1}$ can be obtained by

$$\begin{aligned}\hat{P}_i^k &= [(\check{P}_i^k)^{-1} + \sum_{j\in\mathcal{N}_i^k} s_j^k]^{-1}, \\ \hat{\xi}_i^{k|k-1} &= \hat{P}_i^k \sum_{j\in\mathcal{N}_i^k} y_j^k\end{aligned} \tag{2.40}$$

In the last step, the estimated state $\hat{X}_i^k$ is recovered from $\hat{\xi}_i^k$ and $\bar{X}_i^k$ as the following

$$\hat{X}_i^k = exp(\hat{xi}_i^k)\bar{X}_i^k \tag{2.41}$$

Now, we obtain the posterior estimated information pair $(\hat{X}_i^k, \hat{P}_i^k)$.

Chapter 3

Experiment Environment

3.1 Hardware

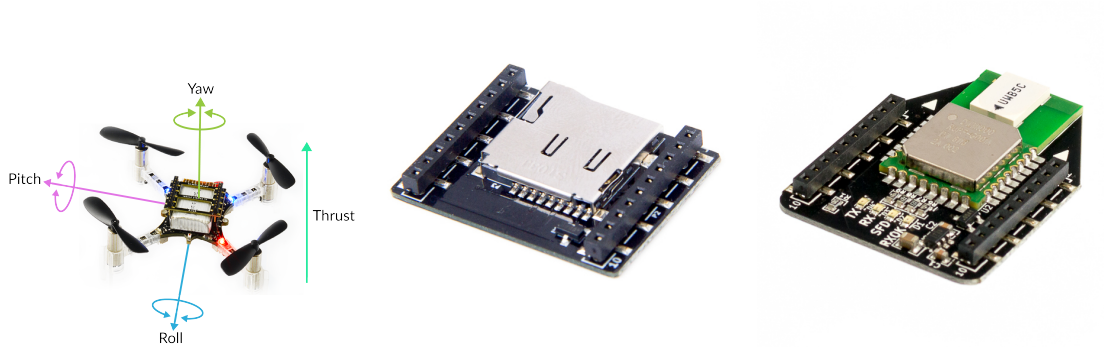
3.1.1 Crazyflie Quadcopter

In this work, we use crazyflie 2.1 quadrotors. As shown in Figure 3.1, the crazyfile 2.1 is a versatile open source flying development platform designed by Bitcraze. It is equipped with four 7x16 mm coreless DC motors and 45 mm plastic propellers. Crazyflie 2.1 is supplied by 1 Cell (3.7 V), 20x30x7 mm, 240 mAh LiPo battery. It provides energy up to 7 minutes of continuous flight.

The crazyfile 2.1 has two micro-controllers called main mirco-controller and additional mirco-controller. The main micro-controller is an ARM 32-bit STM32F405 Cortex-M4 embedded processor running at 168 MHz. It communicates with ground PC computer with a USB dongle and being controlled over the 2.4 GHz Crazyradio PA. The additional micro-controller is a 32 MHz nRF51822 ARM Cortex-M processor for power energy manage-

ment and radio communication managment. The robot on-board sensors system is equipped with an Inertial Measurement Unit (IMU), i.e., MPU-9250 with three axis: gyro, accelerometer and magnetometer, with an additional high precision pressure sensor (LPS25H).

Multiple expansion decks such as AI deck and LED-ring deck can be easily installed on crazyflie 2.1 for different purposes. In our experiment, we use local positioning deck for the setup of local positioning system, and micro-sd card deck to record data synchronously in relatively-high frequency. Specifically, the Loco Positioning deck is a tag in a Loco Positioning system which can measure the distances to anchors in three different mode (TWR, TDOA2, and TDOA3). The distance measurements can be used to conduct position estimation for the quadrotor. The position is calculated on-board of the quadrotor without any need for an external computer. The micro-sd card deck has a micro-sd card reader to the crazyflie, and makes it possible to record data from crazyflie up to 1kHZ and write it in a micro-sd card. Although we can also obtain experimental data via radio in real-time, the communication rate is very low which hurts the synchronicity of raw data. Note that the micro-sd card is using the SPI bus on the deck port to communicate. This has turned out to have some implications for the other decks such as local positioning deck that use the SPI bus which limits the speed of logging. Thus, we have to use the hidden SPI port in the main micro-controller to avoid the communication conflict issue. In our experiment, IMU and UWB distance measurement are recorded for state estimation. We also record Vicon data, i.e., position and orientation as our ground truth.



(a) Crazyflie2.1 (b) Micro-sd card deck (c) Local-positioning deck

Figure 3.1: Crazyflie2.1 with expansion decks

3.1.2 Vicon System

The vicon system is an optical passive motion capture system. This system can track and anaylze the motion of a moving object in a 3D workspace. It uses passive spherical markers that placed on objects. The markers can be easily detected by vicon cameras by reflecting infrared. Vicon system is widely applied due to its high performance: generally position errors are less than one milimeter. In comparison, some state-of-the-art localization systems such as UWB has very large position errors of over 10 centimeters, which is imaplicable for dense formations. In this thesis, we use vicon system to obtain the position and orientation informations of our quadrotors as ground truth.

3.2 Software

3.2.1 Crazyswarm ROS Package

The crazyswarm platform is a ROS package which allows us to fly a swarm of crazyflie 2.x. quadrotors in tight, synchronized formations. This platform supports multiple localization systems including LPS, motion capture system and LightHouse. In contrast to some related platforms, the majority of in-flight computation is implemented onboard. The frequency of main onboard loop is 500 Hz. In each iteration, the onboard microcontroller fuses IMU and motion capture system measurements in an Extended Kalman Filter (EKF) to improve the performance of state estimation and reduce the communication workloads. The EKF implementation follows the indirect error-state approach. Crazyswarm package designs a set of onboard trajectory planning methods that reduce the amount of information change between the base station and crazyflie. This package also sets up reliable communications for large swarm of crazyflies with low latency. Furthermore, crazyswarm develops a high-level and effective controller for crazyflies. Figure 3.2 shows the diagram of major system components of crazyswarm. In our experiments, we make use of this package to conduct high level control and set up effective communication frameworks between the host computer and crazyflies.

3.2.2 Crazyflie Client

The crazyflie PC client is an important tool in our experiments. This client enables users to flash and control the crazyflie. It implements the user interface and high-level control commands such as gampad handling. Figure 3.3 shows the user interface of the

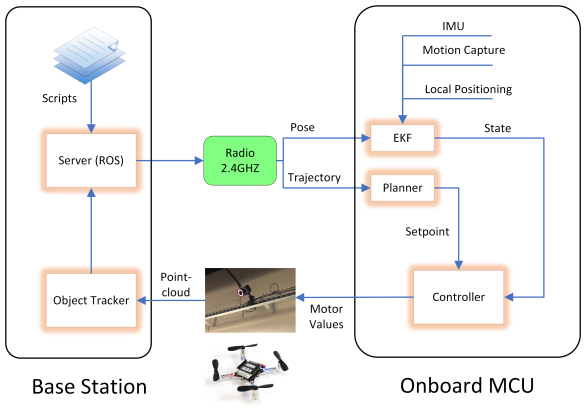


Figure 3.2: Crazyswarm diagram. A point cloud of markers are detected by the motion-capture system to track the crazyflie. All estimated poses are broadcasted by the radios on the base PC. State estimation, path planning, and control run onboard at 500 Hz.

crazyflie client. In this thesis, we use crazyflie client to assign the address of each crazyflies and select the communication channel. The radio communication frequency is designed to be 100Hz between the crazyflie and the host PC. The communication frequency needs to be high enough to ensure smooth trajectory. The radio address for the crazyflie is configured as "0xE7E7E7E70X" where "X" represents the number of the crazyflie in the hexadecimal system. Note that this number should be corresponding to the crazyflies' name in the motion capture system.

In addition, the configuration of LPS is required to be manually done from crazyflie client. This configuration is an essential part in the setup of our LPS system and has to be operated according to the following steps: 1) Open the crazyflie client and connect to the crazyflie. Configure the crazyflie in 2Mbit radio mode if not already done. This can reduce interferences with the UWB radio. If the configuration is changed a restart of the crazyflie is required. 2) Check anchors' status and verify the communication between the anchors and crazyflie. If there is any issue with the communication, verify the anchor is configured

correctly. 3) Each the anchors' position manually and verify the anchors' position in the crazyflie client. 4) Select the system mode and the caliboration will be automatically done in the crazyflie client. The LPS supports three different modes: Two-way Ranging (TWR), Time-of-difference arrival 2 (TDOA2) and Time-of-difference arrival 3 (TDOA3).

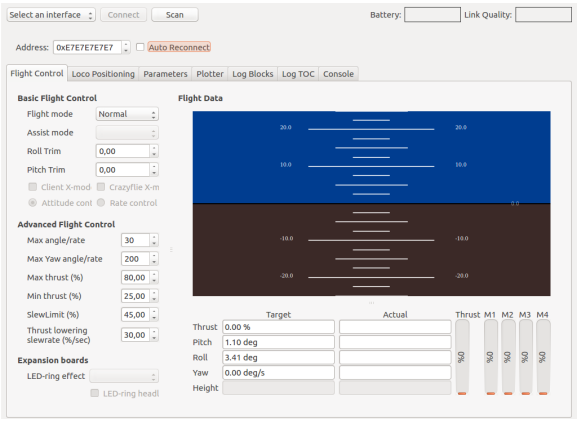


Figure 3.3: Crazyflie clients

3.2.3 Eclipse

Diving into the C-based firmwares is one of the key components about developping crazyflies. Eclipse is a proper tools to make some major changes to the intrinsics of the firmwares and debug the STM32. Eclipse also allows us to place the break point and read the real-time values of the code which make us easily debugging the firmwares. Note that a debug adapter and a ST Link debugger are required for on-chip debugging through eclipse. Generally, the crazyflie firmware can be coded and debugged in Eclipse according to the following steps: 1) Install required Eclipse Plugins including the C++ development tools and GNU MCU plugin; 2) Import crazyflie firmware into Eclipse and set up the debugging

environment; 3) Connect to crazyflie with a debug adapter and a ST link debugger, and configure the firmware; 4) Set up the breakpoints in the code and debug. Eclipse can run into an dedicated debugging environment automatically. 5) Eclipse can automatically halt the crazyflie's firmware before it enter the main function of the firmware.

Chapter 4

My Work and Contribution

In this thesis, the LPS platform is setup based on crazyflie2.1 and LPS decks, and the UWB localization in TWR and TDOA modes is established accordingly. In addition, we build an offline data acquisition system to record and process the measurement data on crazyflie2.1. The proposed algorithms are validated by the experimental platform.

4.1 The Setup of the Experimental Platform

Figure 4.1 shows the experimental diagram of this thesis. In our experiments, the Ubuntu 18.04 system and ROS have been installed on the ground PC. The crazyflie client is installed on the ground PC to control the crazyflie and setup the LPS. In order to establish and improve the UWB localization system, we use the Loco-Positioning configuration tool (LPCT) created by Bitcraze to configure the UWB anchors according to the following steps:

1) Update the LPS firmware of each anchor by LPCT: First label each anchor with an appropriate ID from 0-7, and upload the firware. Then select the anchor mode including

TWR and TDOA. Note that the UWB measurement by TWR always has an offset which directly related to the parameter named "antenna delay". To reduce the offset and improve the accuracy of the UWB measurement, we build a few tests to find a proper value of the "antenna delay".

2) Place the anchors in the experimental space: Four anchors are placed in an approximately rectangle space on the ground, and the other four anchors are placed on tripods with 1.8m above the ground. All the anchors are fixed and powered by 5V batteries. To avoid interferences, each anchor is placed at least 15cm away from the obstacles. The anchor positions are measured by using vicon motion capture systems.

3) Configure and verify the system: When all the anchors are mounted and powered, we use crazyflie client to configure the system. The commonication between the client and the anchors are relayed through the LPS decks and the crazyflie. First, we place the crazyflie in the center of the flying area and connect it through crazyflie client. Then we can click the loco-positioning tab in the client to check the status of each anchor. If the communication between the client and the anchors works well, we can manually enter anchor positions to each anchor. The crazyflie client can automatically calibrate the system. Finally, we verify the built system and switch the system mode through cfclient.

4.2 The Data Acquisition and Processing System

In our work, we aims to obtain the data including IMU, UWB distance measurements, and ground truth. Other data such as barometer measurement and control parameters can also be recorded. All the data are logged offline by using a micro-sd card

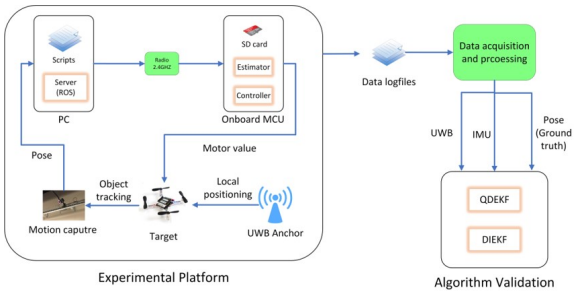


Figure 4.1: Experimental diagram.

attached to the crazyflie. The micro-sd card is using the SPI bus on the deck port to communicate with the crazyflie. Due to the communication issue we mentioned in the chapter 3, the connection between the decks (UWB tag and micro-sd card deck) and the crazyflie has to be re-designed for data logging. Here, we use a "hidden" SPI port on the deck named "SPI3" for the micro-sd card, separate from the SPI port named "SPI1" which is using for the LPS deck[1]. Figure 4.2 shows the communication framework between the crazyflie and the expansion decks. The following steps can be used to work around this issue:

1) Patch and solder the micro-sd card deck and the LPS deck, Figure 4.3 illustrates the patching and soldering procedures: cut some small lines and solder small patch wires between the crazyflie and the decks based on the wire connections shown in Figure 4.4. Then, attach all the decks to the crazyflie.

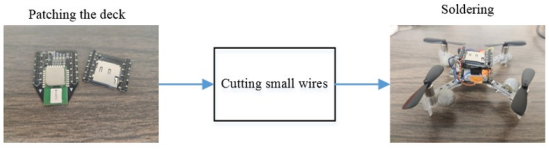


Figure 4.2: Soldering procedures.

2) Update the firmware to the hidden SPI mode and compile the firmware respectively.

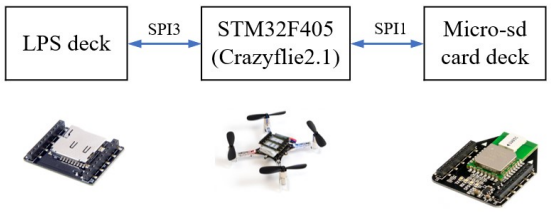


Figure 4.3: Communication framework between the crazyflie and the expansion decks. The micro-sd card deck communication with crazyflie using SPI3 while the LPS deck uses SPI1 port for communication.

The micro-sd card data logging system is formatted as a FAT32 file system. This data logging system is contained in the firmware of the crazyflie and can be activated using a configuration file in the micro-sd card. Logging variables such as sensor measurements and state estimations can be logged synchronized in a fixed frequecy. The maximize logging frequency is 1000 Hz. In order to reduce the data size and avoid communication conflicts, the number of logging variables is limited to 18 in our experiments. Once the logging process is started, a logfiles containing all the loggin variables will be created in the micro-sd card. The logfiles can be enumerated in ascending order from 0-99, which allows multiple logs. A new logfiles can be automatically created when resetting the crazyflie. When the logging process is stopped, we will obtain a binary logfile which can be decoded simply by python[1].

Chapter 5

Results and Discussion

5.1 Experimental Validation

In this section, we use the real data collected by crazyflie2.1 and LPS to test the performance of the proposed algorithms offline. The target, crazyflie2.1, is tracked by LPS consisting of 8 uwb anchors in a 3D environment, denoted as agents 1-8. The crazyflie is also equipped with a UWB tag. Of the 8 anchors, 4 are placed in an approximately rectangle on the ground, and the other 4 are placed at a height of approximately 1.8m above the ground. Figure 5.1 shows the anchor positions. The vicon motion capture system is applied to measure the pose of the interest target as ground truth. The real data is logged by the micro-sd card in another deck named micro-sd card deck attached on the crazyflie.

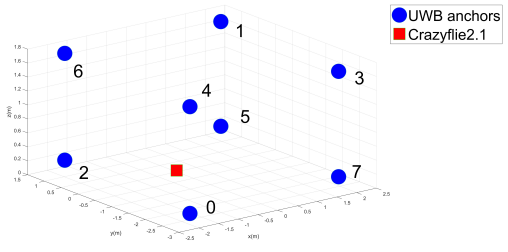


Figure 5.1: Spatial distribution of the networked agents (UWB anchor).

5.1.1 Estimator Parameters

The IMU data is obtained by the IMU sensor BMI088 on the crazyflie2.1, including a 3-xais accelerometer and gyroscope meter. The IMU model is followed by (2.12). Due to the fact that the motors on crazyflie are almost directly connected to the PCB which induces a lot of extra vibrations, the IMU noise parameters on a flying crazyflie is totally different with it on a stationary platform. Based on the observations, the estimation performance changes significantly when the process noise are selected within a wide range. Thus, the process noise in each axis is chosen adaptively for better estimator performances in the experiments as follows

$$\begin{aligned} \eta_{acc} &= (0.02, 0.02, 0.05) m/sec^2/\sqrt{Hz}, \\ \eta_{gyro} &= (0.15, 0.15, 0.20) deg/sec/\sqrt{Hz} \end{aligned} \tag{5.1}$$

where η_{gyro} is the noise for the angular rate gyroscope and η_{acc} is the noise for the accelerometer. We consider two different modes of UWB measurements in our experiments. One is two-way ranging (TWR)that returns the distance between the tag at its current position ${}^{G}p_{T_k}$ and the anchor position $p_{uwb,i}$. The covariance of the UWB TWR measurement is set to $Var(\eta_{twr}) = 0.0225m^2$ based on the analysis of the UWB distance measurements. We

place the crazyflie in the different locations of the tracking space, record the distance measurements from UWB anchors and compute the covariance based on the distance obtained from the ground truth. The local measurement model for agent i at timestep k hence has the following form:

$$z_i^k = h_i({}^G p_{T_k}) + \eta_{twr} = \|p_{uwb,i} - {}^G p_{T_k}\| + \eta_{twr} \tag{5.2}$$

Another measurement mode named time-difference of arrival (TdoA) measurements is also used in our experiments. Tdoa mode cannot directly measure the distance bwtween the tag and the anchor. In this mode, the measurement can be regarded as the difference between the distances to two different anchors. The measurement model is given as

$$z_i^k = h_i({}^G p_{T_k}) + \eta_{tdoa} = \|p_{uwb,i} - {}^G p_{T_k}\| - \|p_{uwb,j} - {}^G p_{T_k}\| + \eta_{tdoa} \tag{5.3}$$

5.1.2 Experiments

The experiment is conducted in $4m \times 3m \times 2m$ space shown in Figure 4.2, and the coordinates of 8 fixed anchors are represented in Table 1. Each agent is able to obtain measurements when the target of interest is within its field of sensing. Moreover, we assume that each agent can communicate with the other cameras with certain percentages to show the benefits of distributed tracking. For example, 60% communication means that each agent has the probability of 60% to communicate with other agents. The communication neighbors of each agent are randomly choosen at each timestep, and hence the communication graph here is time varying. We assume that the agents have the knowledge of the target's motion model. We compare the results of the proposed two algorithms (DIEFK and

QDEKF) of different communication rate in estimating both position and orientation. The results are quantified by rooted mean square error (RMSE) that evaluates the estimation accuracy.

Table 5.1: UWB anchor positions

ID	x [m]	y [m]	z [m]
0	-1.83	-1.34	0.20
1	-1.88	1.10	1.73
2	2.02	1.10	0.20
3	1.98	-1.32	1.73
4	-1.83	1.34	1.73
5	-1.87	1.11	0.20
6	2.02	1.10	1.73
7	1.98	-1.32	0.20

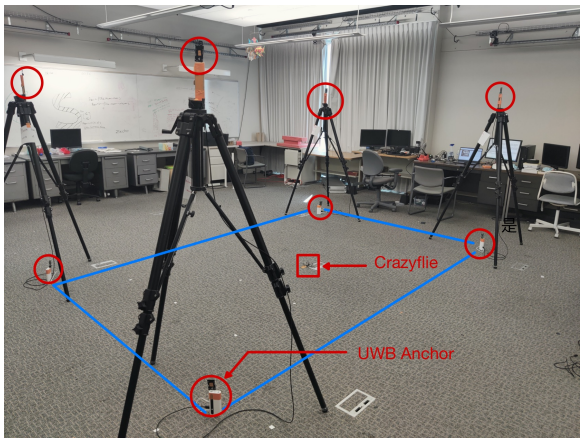


Figure 5.2: Experimental platforms: a crazyflie2.1 quadrotor robot and the LPS.

In our experiment, real data is collected in UWB-TWR mode. The target is flying following a pre-designed 3-D trajectory. The initial position of the crazyflie is chosen as the

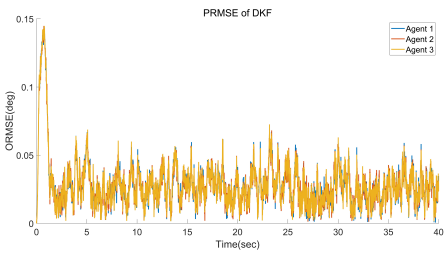
original point in the space. The initial covariances $P_i^{0|0}$ in QDEKF is given as

$$P_i^{0|0} \sim diag(0.15^2, 0.15^2, 0.20^2, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}), \forall i \in \mathcal{V} \tag{5.4}$$

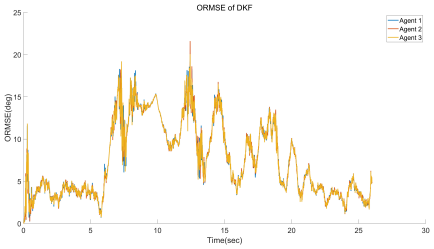
where $P_i^{0|0}$ will be converted respectively in DIEFK. Due to that the micro-sd card uses shared SPI bus with the LPS deck on the crazyflie, it would slow down the data transmission or stop working if it is too congested. This finally causes the micro-sd card logging to fail when LPS is put in TWR mode. In addition, because of the limitation to the number of logging variables that fit in the data log block of the crazyflie, we cannot record too many data synchronized in a relatively high frequency in TWR mode. Thus, we implement two independent experiments which estimate the target's position and orientation separately. The data is collected on a crazyflie2.1 at 100Hz including real IMU, UWB, and groud-truth data. Table 4.2 shows the average RMSE for position and orientation for different communication rate in TWR mode. It is clear that the estimation performance of position improves as the communication rate increase. Further, to illustrate the estimation performance over time, we plot the RMSE(PRMSE) and RMSE(ORMSE) results over agent 1,2 and 3 with 60% communication for both algorithms in Figure 4.3. The estimated trajectory of the agent 1 is shown in Figure 4.

Table 5.2: Average RMSE for the pose estimation in TWR mode

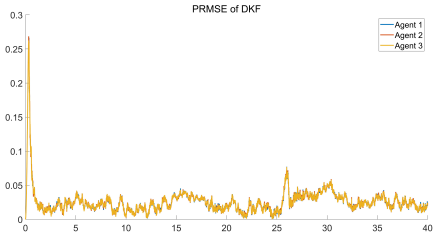
Communication rate		10%	20%	30%	40%
QDEKF	PRMSE (m)	0.131	0.092	0.085	0.076
	ORMSE (deg)	9.425	9.857	9.570	10.212
DIEKF	PRMSE (m)	0.098	0.073	0.062	0.058
	ORMSE (deg)	8.16	7.75	8.25	8.10



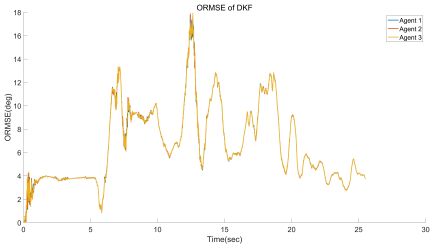
(a) PRMSE for QDEKF



(b) ORMSE for QDEKF

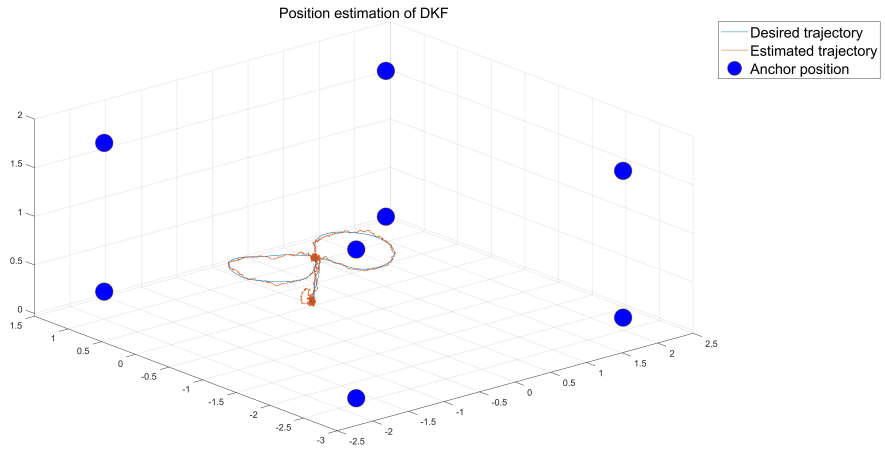


(c) PRMSE for DIEKF

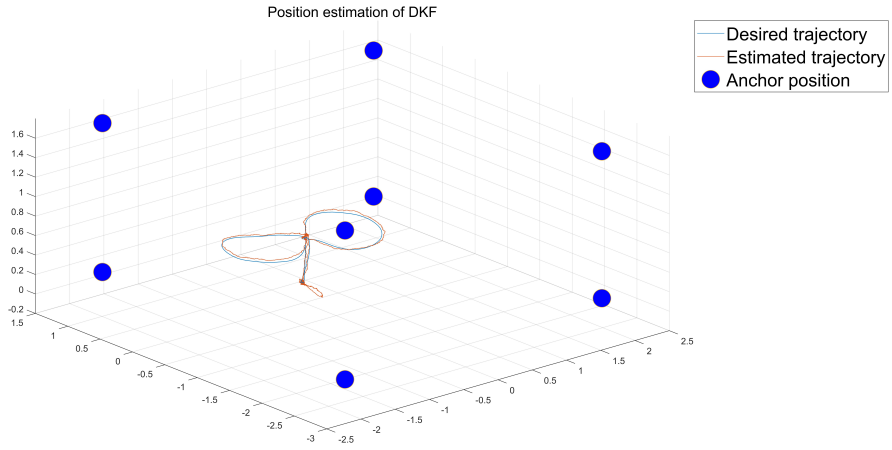


(d) ORMSE for DIEKF

Figure 5.3: RMSE for the estimated target pose of agent 1,2,and 3.



(a) 3-D trajectory for QDEKF



(b) 3-D trajectory for DIEKF

Figure 5.4: Estimated 3-D trajectory of the agent 1.

Chapter 6

Conclusions

This thesis studies two DKF algorithms (QDEFK and DIEFK), and shows the experimental validation of the algorithms. In order to validate the effectivenss of both algorithms and compare their performances, both algorithms were implemented on the experimental platforms. In particular, the DKF algorithms for quadrotor localization with UWB was developed and tested. Using the considered DKF algorithms, the target tracking process can be achieved by only using the local information and its local communications. Further, the experimental results shows that the DIEFK algorithm is more accurate in the position and orientation estimation, and coverage faster than QDEKF algorithm. The future work will be extend to online situations to test the real-time performances of both algorithms on different robot platforms.

Bibliography

[1] `https://www.bitcraze.io/products/micro-sd-card-deck/`.

[2] Axel Barrau and Silvère Bonnabel. The invariant extended kalman filter as a stable observer. *IEEE Transactions on Automatic Control*, 62(4):1797–1812, 2017.

[3] G. Battistelli, L. Chisci, G. Mugnai, A. Farina, and A. Graziano. Consensus-based linear and nonlinear filtering. *IEEE Transactions on Automatic Control*, 60(5):1410–1415, 2015.

[4] Wojciech Giernacki, Mateusz Skwierczyński, Wojciech Witwicki, Paweł Wroński, and Piotr Kozierski. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 37–42, 2017.

[5] Xingkang He, Wenchao Xue, and Haitao Fang. Consistent distributed state estimation with global observability over sensor network. *Automatica*, 92:162–172, 2018.

[6] Bin Jia, Khanh D. Pham, Erik Blasch, Dan Shen, Zhonghai Wang, and Genshe Chen. Cooperative space object tracking using space-based optical sensors via consensus-based filters. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1908–1936, 2016.

[7] Mark W. Mueller, Michael Hamer, and Raffaello D'Andrea. Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1736, 2015.

[8] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In *2007 46th IEEE Conference on Decision and Control*, pages 5492–5498, 2007.

[9] Wei Ren, Randal W. Beard, and Ella M. Atkins. Information consensus in multivehicle cooperative control. *IEEE Control Systems Magazine*, 27(2):71–82, 2007.

[10] Nikolas Trawny and Stergios I. Roumeliotis. Indirect kalman filter for 3 d attitude estimation. 2005.

[11] Shaocheng Wang, Yang Lyu, and Wei Ren. Unscented-transformation-based distributed nonlinear state estimation: Algorithm, analysis, and experiments. *IEEE Transactions on Control Systems Technology*, 27(5):2016–2029, 2019.

[12] Shaocheng Wang and Wei Ren. On the convergence conditions of distributed dynamic state estimation using sensor networks: A unified framework. *IEEE Transactions on Control Systems Technology*, 26(4):1300–1316, 2018.

[13] Jie Xu, Pengxiang Zhu, and Wei Ren. Distributed invariant extended kalman filter for 3-d dynamic state estimation using lie groups. *2022 American Control Conference (ACC)*, 2022.

[14] Yingrong Yu, Jianglong Yu, Yishi Liu, and Zhang Ren. Distributed state estimation for heterogeneous mobile sensor networks with stochastic observation loss. *Chinese Journal of Aeronautics*, 35(2):265–275, 2022.

[15] Pengxiang Zhu and Wei Ren. Distributed kalman filter for 3-d moving object tracking over sensor networks. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 2418–2423, 2020.