# A Low-cost SLAM Fusion Algorithm for Robot Localization

by

Ruifan Wu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

University of Alberta

# Abstract

Autonomous navigation has been a popular research topic over the last two decades. The ability for a robot to solve the simultaneous localization and mapping (SLAM) problem is required to navigate unknown environments. One such example is a self-driving car: it needs to build maps of new environments and simultaneously localize itself. Many approaches for solving the SLAM problem have been proposed, but most of these require expensive LiDAR sensors. In this work, we investigate a strategy to solve the SLAM problem using an RGB-depth sensor Kinect v2, an open-source SLAM package called RTAB-Map, and an Extended Kalman Filter (EKF). Although SLAM algorithm we chose is computationally tractable and provides accurate results, the resulting pose data is available only at 1 Hz. To overcome the disadvantage of the low data frequency from RTAP-Map, we utilize an EKF filter to fuse in odometry estimates and obtain higher-rate (30 Hz) pose estimates. The system is implemented onboard a Jackal Unmanned Ground Vehicle (UGV), equipped with a Kinect v2 camera as the only external sensor. A motion capture system from Vicon is used to obtain a ground truth of the robot's motions.

Our results show good agreement between the pose estimates from our system and the ground truth. Our work demonstrates the ability of an inexpensive RGB-depth sensor such as the Kinect v2, combined with an open-source SLAM package and fused with high-rate odometry estimates through an EKF, to achieve good performance and accuracy results.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Navigation is a fundamental technology for autonomous vehicles or robots. The earliest navigation systems were used on guided missiles of the US government in the early 1950s, developed by the MIT Instrumentation Laboratory. This system is based on inertial navigation which obtains the state estimates of a missile or vehicle by dead reckoning calculated from inertial sensors. Such inertial sensors are capable of measuring the acceleration and angular velocity of the vehicle at a high rate. The drawback of inertial navigation is that the error in the state estimates grows without bound as time passes and cannot be corrected by itself [1]. Later with the appearances of technologies such as the NavStar Global Positioning System (GPS) or star trackers, the state estimates computed from the integration of inertial sensors could be corrected to rectify the accumulated errors. This kind of system in which low rate sensors like GPS are used to correct the state estimate produced by high rate inertial sensors is referred to as an aided navigation system. Among aided navigation systems, GPS-aided navigation is the most common and widely used solution in commercial vehicles and the aerospace area and considered as the state-of-the-art until the late 1990s.

As stated above, a GPS-aided navigation system can determine the state of the vehicle accurately. However, in some situations such as indoor, underground or outer space where GPS is noneffective, GPS-aided navigation cannot be used. In 1986 during the IEEE Robotics and Automation Conference held in San Francisco, the concept of Simultaneous Localization and

Mapping (SLAM) problem was first proposed. The problem asks if the robot can build a map of its unknown environment and locate itself at the same time without an external reference system like GPS, which can be stated as a navigation problem in environments like indoor, underground and outer space. A SLAM problem is more challenging than a mapping problem or a localization problem, since neither map nor pose is known [2]. Ever since the proposal of SLAM, it has been considered as a fundamental problem in robotics. SLAM solutions are employed in wheeled robots, but also unmanned aerial vehicles, autonomous underwater vehicles and even inside the human body [3]. The SLAM problem is usually solved using inertial sensors and sensors which can provide visual features. Because of the accumulated error from inertial sensors and measurement errors, the estimated state of the robot and the positions of observed landmarks (map) all have uncertainty. Thus, the SLAM problem is described in a statistical framework based on the work by Smith and Cheeseman following the IEEE conference [4]. In the late 1980s, Kalman filter-type algorithms began to be used in visual navigation [5] and sonar-based navigation [6] [7]. Inspired by the above works, Smith *et al.* [8] proposed to estimate the state of robot using the Extended Kalman Filter (EKF) in 1990, which was later considered as the first solution of the SLAM problem, called EKF-SLAM. In 1995, Whyte *et al.* first proved that the SLAM problem estimates converge when combining mapping and localization into a single estimation problem [9]. After this, large varieties of SLAM solutions became available. The SLAM problem can be divided into two main categories: *online* SLAM and *full* SLAM [2]. The first one involves only estimating the instantaneous state of the robot and the map while the second one seeks the complete trajectory of the robot state as well as the map. The solutions of the online SLAM problems are classified as a *filtering* approach [10], which includes the EKF-SLAM solution stated above. Some other popular filtering approaches also falling into this category include particle [11] [12] [13] and information filters [14] [15]. The solutions of the full SLAM problem are classified as a *smoothing* approach which typically relies on least-square error minimization techniques [10]. One of the solutions belonging to the *smoothing* category is

Graph-based SLAM approach which was first addressed by Lu and Milios in 1997 [16]. This solution has become a popular research trend in recent years because of its advantages over filtering approaches such as support for very large-scale mapping. In our study, we use an approach which belongs to the Graph-based SLAM category.

The procedure for solving graph-based SLAM can be divided into front-ends and back-ends. Front-ends can be understood as the process of constructing a set of map nodes from onboard sensors, and detecting loop closures which occur when the robot returns to a previous visited location. This involves multiple problems such as feature extraction, frame matching, and rejection of false closures. There are a variety number of methods to improve the performance of front-ends. Neira and Tardos [17] proposed a joint compatibility branch and bound method to find the largest loop closure hypothesis, which guards against wrong detections. Other methods such as spectral clustering [18] and tree-structured Bayesian [19] can also improve the data association. Compared with front-ends, back-ends solve a more time-consuming problem which is adjusting and optimizing the estimated map based on measured and detected constraints between nodes. The solutions of back-ends are often known as graph optimization methods. Lu and Milios [16] gave the first solution to globally optimize the map. Later in 1999, Gutmann and Konolige [20] proposed a technique called Local Registration and Global Correlation (LRGC) which can efficiently add new information to the system and perform loop closure detection. Relaxation approaches to solve back-ends were introduced by Howard *et al.* [21]. Frese *et al.* [22] applied a variant of Gauss-Seidel relaxation on multiple levels of resolution to optimize the map. Sparse Cholesky and QR factorization were first used by Dellaert and Kaess [23] to solve the SAM (smoothing and mapping) problem. They later presented iSAM [24] which implements real-time SLAM application. Konolige [25] exploited conjugate gradient descent method to optimize the map. Graph optimization methods such as TORO (a tree-based network optimizer) [26] and g2o (general graph optimizer) [27] use theclassic optimization methods of gradient descent and Levenberg-Marquardt algorithm respectively to minimize the non-linear least

squares problem in back-ends.

A large variety of open-source SLAM solutions which run in real-time are available for the Robot Operating System (ROS). Most of these can be classified as either lidar-based SLAM approach or visual SLAM approach based on their sensor type. Among the lidar-based SLAM approaches, GMapping [13] proposed in 2007 is considered as a state-of-the-art SLAM solution, which utilizes Rao-Blackwellized particle filters. Google Cartographer [28] is a more recent open-source lidar-based SLAM solution published in 2016, which creates sub-maps instead of individual map nodes and optimizes the estimated poses of sub-maps when a loop closure is detected. It has shown good results in a backpack-based mapping platform, but its hard-to-tune parameters and the need for an expensive Velodyne lidar sensor make it difficult to implement on a real robot. Visual SLAM approaches usually utilize stereo cameras or RGB-D cameras. ORB-SLAM2 [29] is considered as a state-of-the-art visual SLAM approach, which uses bundle adjustment for graph optimization, but as the map grows the time to detect loop closures and update the map also increases, making ORB-SLAM2 unable to meet real-time constraints. The open-source SLAM solution we chose for this thesis is Real-time Appearance-Based Mapping (RTAB-Map) [30] [31] whose memory management architecture makes it possible to achieve multi-session and large-scale mapping under real-time constraints. RTAB-Map supports both depth cameras and lidar sensors and can generate either 2D or 3D maps. Although SLAM solution solves mapping and localization problem, its computational complexity makes the output rate slow. The pose output of RTAB-Map is around 1 Hz.

Our objective in this thesis is to obtain accurate and high rate global pose data using low-cost sensor by achieving SLAM-aided navigation though sensor fusion. In other words, we want to find a solution that has better performance than single SLAM solution. We use a commercial mobile robot equipped with wheel encoder, IMU and computer as platform. We fuse the output of SLAM with inertial sensors of robot to obtain high frequency and accurate robot pose data. In 2004, Kim and Sukkarieh proposed a method to improve SLAM performance by adding a feedback control to inertial sensors which as the in-

put for SLAM [32]. However, the publish rate of SLAM is still low. Many studies have been done on sensor fusion for robot localization problem, especially the combination of visual and inertial sensors [33]. In 2002, Strelow and Singh [34] introduced a hybrid visual-inertial sensor fusion system using Levenberg-Marquardt algorithm to optimize combined error from visual sensor and inertial sensors, which showed good results in short-range. In 2003, Rehbinder and Ghosh [35] proposed a pose estimation method by combining line-based vision and inertial sensors. At the early 2000s, researchers in control area were exploring the navigation system that can provide better pose estimates and they mainly focused on the visual-inertial system. However, most of visual-inertial sensor fusion navigation algorithms do not solve the full SLAM problem, their estimated poses will diverge over time and space. Meanwhile, researchers in computer science has been working on the SLAM problem as stated above. They realized that without loop closure detection estimated poses will diverge with no boundary. Thus we come up an idea to use the output of SLAM to be fused with inertial sensors, which ensures the filtered poses are global pose data. And this way has not been used by others before. There are multiple categories to achieve sensor fusion: Kalman filer [36], Bayesian network [37] [38], neural network [39] and so on. A variety different type Kalman filter have been used for localization application such as Extended Kalman filter [40], Unscented Kalman filter [41]. Among all these solutions, nonlinear Kalman filter has been used for a long time to deal with localization problem and it is the most mature and easy to implement technique. Thus in our work, we used these two Kalman filters as our sensor fusion solution and compared their performances.

## 1.1 Thesis Overview

This chapter has given a review of the history of the SLAM problem, provided a literature review and outlined our proposed work. The remainder of the thesis is structured as follows:

**Chapter 2**: Two filters, the Extended Kalman Filter (EKF) and Un-

scented Kalman Filter (UKF) are discussed first. We then introduce fundamentals of filtering SLAM and graph-based SLAM.

**Chapter 3**: The Robot Operating System (ROS) software environment and the hardware used for our experiments are introduced. We cover the data collection method used for multiple computers with different operating systems and explain how to reconcile pose data from different sources.

**Chapter 4**: We describe our chosen SLAM solution, RTAB-Map, and discuss the methods used by RTAB-Map. Three major graph optimization approaches supported by RTAB-Map, TORO, g2o and GTSAM, are also discussed.

**Chapter 5**: The method for selecting the odometry input of RTAB-Map is discussed. Next, we present the results of comparing the performance of two state estimators (EKF and UKF) and the results of implementing SLAM-aided navigation. The experimental results of different configurations of the overall system are then given.

**Chapter 6**: We give conclusions and suggest future work.

# Chapter 2

# Fundamentals

In this Chapter, we will first explain the fundamentals of nonlinear Kalman filters since they are not only the core of the first SLAM solution, EKF-SLAM, but also treated as the motion state estimators for sensor fusion. Next, filtering SLAM approach and smoothing SLAM approach, specifically EKF-SLAM and graph-based SLAM are covered.

## 2.1 Kalman Filter

The idea of Kalman Filter (KF) is using a series of measurements with noise to estimate the state of the system. The KF algorithm can be divided into two steps: prediction and update. In the prediction step, KF estimates the current state variables based on the previous estimate. As soon as new measurements are observed, the KF updates the current estimate according to the certainty of the measurements. The estimate obtained in the prediction step actually happened prior to the current time. This prior estimate is usually denoted as $\hat{\mathbf{x}}_{k|k-1}$. The estimate is updated after the measurements are observed, denoted as $\hat{\mathbf{x}}_{k|k}$. The state of system can be described as

$$\mathbf{x}_{k+1} = \mathbf{\Phi}_k \mathbf{x}_k + \mathbf{w}_k \tag{2.1}$$

The measurement of the process is a linear function of system state, which is denoted as

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{2.2}$$

The notations above are explained below:

$\mathbf{x}_k$ - $(n \times 1)$ state vector.

$\mathbf{\Phi}_k$ - $(n \times n)$ transition matrix describes the relationship between $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$.

$\mathbf{w}_k$ - $(n \times 1)$ vector, process noise whose covariance is known.

$\mathbf{z}_k$ - $(m \times 1)$ measurement vector.

$\mathbf{H}_k$ - $(m \times n)$ matrix describes the nominal relationship between measurement $\mathbf{z}_k$ and state vector $\mathbf{x}_k$.

$\mathbf{v}_k$ - $(m \times 1)$ measurement error vector. Like $\mathbf{w}_k$, its covariance is known, and we assume $\mathbf{v}_k$ and $\mathbf{w}_k$ are independent meaning uncorrelated.

The covariance matrices of error vectors $\mathbf{w}_k$ and $\mathbf{v}_k$ are defined by

$$E[\mathbf{w}_k \mathbf{w}_i^T] = \begin{cases} \mathbf{Q}_k, & i = k \\ \mathbf{0}, & i \neq k \end{cases} \tag{2.3}$$

$$E[\mathbf{v}_k \mathbf{v}_i^T] = \begin{cases} \mathbf{R}_k, & i = k \\ \mathbf{0}, & i \neq k \end{cases} \tag{2.4}$$

Assume we have a prior estimate $\hat{\mathbf{x}}_{k|k-1}$ and we want to use the measurement to improve this estimate. We can express the updated estimate by taking into account both prior estimate and noisy measurement in the form of the following equation:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}) \tag{2.5}$$

where $\mathbf{K}_k$ is known as *Kalman gain*. Except for $\mathbf{K}_k$, other variables in Equation (2.5) are known. Thus, the problem now lies on how to find the Kalman gain that can produce the optimal updated estimate. We can solve the problem of finding the optimal estimate $\hat{\mathbf{x}}_{k|k}$ by finding the minimum mean-square error. Thus, we express the covariance of the error between the updated estimate $\hat{\mathbf{x}}_{k|k}$ and the desired state $\mathbf{x}_k$ as

$$\mathbf{P}_{k|k} = E[\mathbf{e}_{k|k} \mathbf{e}_{k|k}^T] = E[(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})^T] \tag{2.6}$$

where we assume the mean of estimation error $\mathbf{e}_{k|k}$ is zero. After substituting Equation (2.2) and Equation (2.5) into Equation (2.6), we get

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T \tag{2.7}$$

Now, we can find the gain $\mathbf{K}_k$ by minimizing the error covariance $\mathbf{P}_{k|k}$. Because the diagonal entries of $\mathbf{P}_{k|k}$ are the estimation error variances of elements in the system state, the optimization problem can be seen as minimizing diagonal terms in $\mathbf{P}_{k|k}$ (minimize trace $\mathbf{P}_k$). Using matrix differentiation formulas, we can obtain the differentiation of the trace of $\mathbf{P}_k$ with respect to $\mathbf{K}_k$. The detailed process of derivation can be found in [42]. We can solve the Kalman gain by setting the derivative equal to zero:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{2.8}$$

Substituting the optimal Kalman gain into estimation error covariance (2.7) leads to three possible expressions for computing the $\mathbf{P}_{k|k}$. These expressions vary only in the numerical performance on real data. Here, for easy notation, we list the simplest equation which is

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{2.9}$$

Now, we can use the optimal Kalman gain in Equation (2.5) to update the estimate. The recursive procedure of Kalman filter can be understood as a recursive predict-update process:

1. Predict the prior estimate and its prior error covariance.

2. Compute the Kalman gain. Update the estimate and its error covariance.

3. Use the updated estimate and error covariance to predict the prior estimate and error covariance at the next time step (go back to Step 1).

The prior estimate in Step 1 is obtained from the previous estimate via the transition matrix $\mathbf{\Phi}$ as:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{\Phi}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} \tag{2.10}$$

The error and corresponding error covariance are expressed as

$$\begin{aligned}\mathbf{e}_{k|k-1} &= \mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1} \\ &= (\mathbf{\Phi}_{k-1}\mathbf{x}_{k-1} + \mathbf{w}_{k-1}) - \mathbf{\Phi}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} \\ &= \mathbf{\Phi}_{k-1}\mathbf{e}_{k-1} + \mathbf{w}_{k-1}\end{aligned} \tag{2.11}$$

$$\begin{aligned}\mathbf{P}_{k|k-1} &= E[\mathbf{e}_{k|k-1}\mathbf{e}_{k|k-1}^T] \\ &= E[(\mathbf{\Phi}_{k-1}\mathbf{e}_{k-1} + \mathbf{w}_{k-1})(\mathbf{\Phi}_{k-1}\mathbf{e}_{k-1} + \mathbf{w}_{k-1})^T] \\ &= \mathbf{\Phi}_{k-1}\mathbf{P}_{k-1}\mathbf{\Phi}_{k-1}^T + \mathbf{Q}_{k-1}\end{aligned} \tag{2.12}$$

After determining the prior estimate $\hat{\mathbf{x}}_{k|k-1}$ and its covariance $\mathbf{P}_{k|k-1}$, we compute the Kalman gain $\mathbf{K}_k$ via Equation (2.8) and update the estimate $\hat{\mathbf{x}}_{k|k}$ and its corresponding covariance $\mathbf{P}_{k|k}$. Then use the updated state to predict prior estimate at time $t_{k+1}$. A more detailed recursive process is expressed as follows, assuming we specified an initial estimate $\hat{\mathbf{x}}_0$ and its error covariance $\mathbf{P}_0$:

1. Compute Kalman gain

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + R_k)^{-1}$$

2. After obtaining a measurement $z_k$, we update the state estimate as well as its covariance as

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1})$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$$

3. Predict state and error covariance at next time step

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{\Phi}_k\hat{\mathbf{x}}_{k|k}$$

$$\mathbf{P}_{k+1|k} = \mathbf{\Phi}_k\mathbf{P}_{k|k}\mathbf{\Phi}_k^T + \mathbf{Q}_k$$

4. Go back to step 1, calculate Kalman gain and update state values.

### 2.1.1 Extended Kalman Filter

In the real world, especially in navigation, the system models are usually nonlinear. EKF is one of the early extensions of KF which copes with nonlinear models by re-linearizing them about the latest state estimate.

In EKF, the transition and observation models can be specified as differentiable nonlinear functions

$$\dot{\mathbf{x}}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \tag{2.13}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{2.14}$$

where $f$ and $h$ are known, and $\mathbf{u}_k$ is the control vector. The EKF algorithm has the same recursive predict-update procedure as KF. Due to the nonlinear functions used in EKF, some formulas may be different from that of KF. EKF linearizes the functions by applying Taylor's series expansions and keeping only the first-order term. The results are

$$\begin{aligned} f(\mathbf{x}_{k-1}, \mathbf{u}_k) &= f(\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{e}_{k-1|k-1}, \mathbf{u}_k) \\ &\approx f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) + \left[\frac{\partial f}{\partial \mathbf{x}}\right]_{\hat{\mathbf{x}}_{k-1|k-1}} \mathbf{e}_{k-1|k-1} + \frac{\partial f}{\partial \mathbf{u}_k} \\ &= f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) + \mathbf{F}_k \mathbf{e}_{k-1|k-1} + \mathbf{G}_k \end{aligned} \tag{2.15}$$

$$\begin{aligned} h(\mathbf{x}_k) &= h(\hat{\mathbf{x}}_{k|k-1} + \mathbf{e}_{k|k-1}) \\ &\approx h(\hat{\mathbf{x}}_{k|k-1}) + \left[\frac{\partial f}{\partial \mathbf{x}}\right]_{\hat{\mathbf{x}}_{k-1|k-1}} \mathbf{e}_{k|k-1} \\ &= h(\hat{\mathbf{x}}_{k|k-1}) + \mathbf{H}_k \mathbf{e}_{k|k-1} \end{aligned} \tag{2.16}$$

As for the process in KF, the goal is to find a Kalman gain such that the state can be updated. To get this Kalman gain, We need to determine the updated error covariance first. After following the same steps in KF to obtain $\mathbf{P}_{k|k}$, we get the same equation of covariance as Equation (2.7). The only difference is that $\mathbf{H}_k$ in EKF is a Jacobian instead of a measurement matrix. Thus, the Kalman gain and updated error covariance in EKF have the same expressions as Equation (2.8) and Equation (2.9), respectively.

The prior error and corresponding error covariance can be obtained by substituting Equation (2.16) into Equation (2.11):

$$\begin{aligned} \mathbf{e}_{k|k-1} &= \mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1} \\ &= \mathbf{F}_k \mathbf{e}_{k-1|k-1} + \mathbf{w}_k \end{aligned} \tag{2.17}$$

$$\begin{aligned} \mathbf{P}_{k|k-1} &= E[(\mathbf{e}_{k|k-1})(\mathbf{e}_{k|k-1})^T] \\ &= E[(\mathbf{F}_k \mathbf{e}_{k-1|k-1})(\mathbf{F}_k \mathbf{e}_{k-1|k-1})^T] + \mathbf{Q}_k \\ &= \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \end{aligned} \tag{2.18}$$

The recursive procedure of EKF is similar to that of KF. We summarize it as follows denoting the initial prior estimate as $\mathbf{x}_0$ and prior error covariance as $\mathbf{P}_0$:

1. Compute Kalman gain

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + R_k)^{-1}$$

$$\mathbf{H}_k = \left[\frac{\partial h}{\partial \mathbf{x}}\right]_{\mathbf{x}_{k|k-1}}$$

2. After obtaining the measurement $z_k$, update the estimate as well as its covariance

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}))$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$$

3. Predict state and error covariance at next time step

$$\hat{\mathbf{x}}_{k+1|k} = f(\hat{\mathbf{x}}_{k|k}, \mathbf{u}_{k+1})$$

$$\mathbf{P}_{k+1|k} = \mathbf{F}_{k+1}\mathbf{P}_{k|k}\mathbf{F}_{k+1}^T + \mathbf{Q}_{k+1}$$

$$\mathbf{F}_{k+1} = \left[\frac{\partial f}{\partial \mathbf{x}}\right]_{\mathbf{x}_{k+1|k}}$$

4. Go back to the step 1, calculate Kalman gain and update state values.

In conclusion, the EKF utilizes first-order Taylor expansion to linearize the nonlinear measurements and functions to implement KF in navigation. There do exist second- or higher-order linearizations that give more accurate approximation, but they also give more complexity in solutions and require more computational time.

### 2.1.2 Unscented Kalman Filter

The idea of UKF is introducing more points to linearize the nonlinear function instead of using only one point as in EKF [43]. These points are called *sigma points* which means they are not randomly chose. Passing all the points from a Gaussian distribution through a nonlinear function is time consuming, and it is not guaranteed that after a nonlinear transform one can still get a Gaussian-distributed output. Thus, a limited number of points from this distribution with the same mean as the original are chosen. A method called *Unscented*

*Transform* can help determine the estimates of the mean and covariance of a random variable after a nonlinear transform.

Assume the estimated state $\hat{\mathbf{x}}_{k-1|k-1}$ and mean $\bar{\mathbf{x}}$ and covariance $\mathbf{P}$ of the process noise are known. A set of $2N+1$ sigma points is chosen where $N$ is the dimension of the state [43]. The weighted sigma points can be calculated as follows:

$$\begin{array}{lll} \mathcal{X}^0 & = \bar{\mathbf{x}} & W^0 = \kappa/(n+\kappa) \\ \mathcal{X}^i & = \bar{\mathbf{x}} + (\sqrt{(n+\kappa)\mathbf{P}})_i & W^i = 1/2(n+\kappa) \\ \mathcal{X}^{i+n} & = \bar{\mathbf{x}} - (\sqrt{(n+\kappa)\mathbf{P}})_i & W^{i+n} = 1/2(n+\kappa) \end{array} \tag{2.19}$$

where $\kappa$ is a constant. When the state $\mathbf{x}$ is assumed Gaussian distribution, usually $n+\kappa = 3$ is selected because it can give the best performance to reduce the overall prediction error [43]. $(\sqrt{(n+\kappa)\mathbf{P}})_i$ is the $i$th row or column of the matrix square root of $(n+\kappa)\mathbf{P}$ and $W^i$ is the corresponding weight.

The set of these sigma points is denoted as $\mathcal{X}_{k-1|k-1}$ at time $k-1$. We denote the sigma points transformed though the transition function $f(\cdot)$ as

$$\mathcal{X}^i_{k|k-1} = f(\mathcal{X}^i_{k-1|k-1}) \quad i = 0, ..., 2N \tag{2.20}$$

To produce the predicted state $\hat{\mathbf{x}}_{k|k-1}$ and covariance $\mathbf{P}_{k|k-1}$, transformed sigma points are assigned weights and combined as follows

$$\hat{\mathbf{x}}_{k|k-1} = \sum_{i=0}^{2N} W^i \mathcal{X}^i_{k|k-1} \tag{2.21}$$

$$\mathbf{P}_{k|k-1} = \sum_{i=0}^{2N} W^i \left[\mathcal{X}^i_{k|k-1} - \hat{\mathbf{x}}_{k|k-1}\right] \left[\mathcal{X}^i_{k|k-1} - \hat{\mathbf{x}}_{k|k-1}\right]^T \tag{2.22}$$

After we obtain the predicted state and covariance, using the same approach as stated above we create sigma points and project them though the observation function $h(\cdot)$

$$\gamma^i_k = h(\mathcal{X}^i_{k|k-1}) \quad i = 0, ..., 2N \tag{2.23}$$

We can obtain the predicted measurement and predicted measurement covariance by combining the weighted sigma points

$$\hat{\mathbf{z}}_k = \sum_{i=0}^{2N} W^i \gamma^i_k \tag{2.24}$$

$$\mathbf{P}_{z_k z_k} = \sum_{i=0}^{2N} W^i \left[\gamma_k^i - \hat{\mathbf{z}}_k\right] \left[\gamma_k^i - \hat{\mathbf{z}}_k\right]^T \tag{2.25}$$

The Kalman gain of UKF is computed as

$$\mathbf{K}_k = \mathbf{P}_{x_k x_k} \mathbf{P}_{z_k z_k}^{-1} \tag{2.26}$$

where the cross-covariance matrix of state and measurement $\mathbf{P}_{x_k z_k}$ is

$$\mathbf{P}_{x_k z_k} = \sum_{i=0}^{2N} W^i \left[\mathcal{X}_{k|k-1}^i - \hat{\mathbf{x}}_{k|k-1}\right] \left[\gamma_k^i - \hat{\mathbf{z}}_k\right]^T \tag{2.27}$$

As with the EKF above, the updated state is derived by the predicted state plus the measurement error weighted by the Kalman gain

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{z}}_k) \tag{2.28}$$

And the updated covariance is derived by the predicted covariance minus the predicted measurement covariance weighted by the Kalman gain

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{P}_{z_k z_k} \mathbf{K}_k^T \tag{2.29}$$

When the transition function $f(\cdot)$ and observation function $h(\cdot)$ are highly non-linear, the EKF typically gives poor performance [43]. UKF gives more accurate mean and covariance estimates for certain systems [44]. UKF also does not require calculating Jacobians, which reduce the complexity of computation. In our experiment, we compared the performances of both state estimators in Chapter 6 to see whether our nonlinear system (evolving on $SE(3)$) will influence their performances.

## 2.2 Filtering SLAM

Solving the SLAM problem cannot rely on the external measurement system such as GPS, so only measurements $u_{1:t}$ and observations $z_{1:t}$ from onboard sensors are given, where $t$ is the time. Based on these inputs, the SLAM problem is to find the trajectory of robot $x_{1:t}$ and the map described by features $m$.

The online SLAM problem seeks to estimate the most recent pose and map. We can describe the online SLAM problem as estimating the posterior probability of momentary pose and the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{2.30}$$

Filtering SLAM solutions usually involve the following procedures, which repeat at each time step: At an initial position, landmarks are observed and their locations are noted. When the robot moves to a new location, the uncertainty of the robot pose grows due to noise sources and errors. New observed landmarks have a high uncertainty because of errors in the measurements and uncertainty of the robot pose; and locations of previously observed landmarks are predicted based on the robot's motion. At the new location of the robot, measurements of previously observed landmarks are obtained. Based on the residual of predicted location and measured location, the robot corrects its current estimated location and the locations of observed landmarks.

EKF-SLAM is one of the classic filtering solutions. In the rest of this section, we will present the procedures of EKF-SLAM [8] as an example of the filtering SLAM solution. The state vector of EKF-SLAM contains the estimated poses of both robot and landmarks:

$$\mathbf{x} = [R, L_1, ..., L_n]^T \tag{2.31}$$

where $R$ is the robot state and $L$ represents landmark state, with $n$ the number of landmarks. The covariance matrix of the state vector is

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{RR} & \mathbf{P}_{RM} \\ \mathbf{P}_{MR} & \mathbf{P}_{MM} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{RR} & \mathbf{P}_{RL_1} & \dots & \mathbf{P}_{RL_n} \\ \mathbf{P}_{L_1R} & \mathbf{P}_{L_1L_1} & \dots & \mathbf{P}_{L_1L_n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{L_nR} & \mathbf{P}_{L_nL_1} & \dots & \mathbf{P}_{L_nL_n} \end{bmatrix} \tag{2.32}$$

The goal of EKF-SLAM is to keep the state of the robot and landmarks (map) up to date. Following the procedures stated above and EKF algorithm stated in Section 2.1.1, the state vector is updated based on robot motion $\hat{\mathbf{x}}_{k|k-1} = g(\mathbf{x}_{k-1|k-1}, \mathbf{u}_k)$ which only affects robot state $R$. The covariance is updated as Equation (2.18), where $\mathbf{F}_k$ is replaced by $\mathbf{G}_k$, the Jacobian of

$g$. The Jacobian only affects covariance of robot state $\mathbf{P}_{RR}$. Among all the observed measurements of the locations of landmarks, if the landmarks have not been observed, their locations are updated by adding their states to the estimated robot location. If the landmarks have been observed, based on the previously observed locations and current robot location we can obtain predicted measurements $\hat{\mathbf{z}}_k = h(\hat{\mathbf{x}}_{k|k-1})$, and compare it with measurements we observed to get the measurement residual $\mathbf{z}_k - \hat{\mathbf{z}}_k$. Following Equation (2.8), we can obtain the Kalman gain $\mathbf{K}_k$ after we calculate the Jacobian of $h$, $\mathbf{H}_k$. Now the predicted state vector and covariance can be corrected and updated as follows:

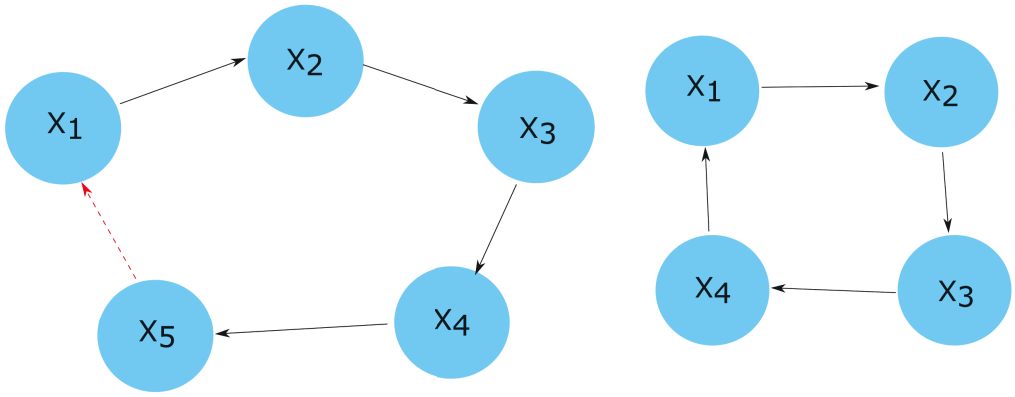
$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{z}}_k)$$

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$$

The above procedures of EKF-SLAM mostly follows the machinery of the original EKF, the only difference is EKF-SLAM adds an extra step, *landmark initialization*, which is adding the states of unobserved landmarks into the state vector.

## 2.3 Graph-based SLAM

The graph in Graph-based SLAM solution is formed by *nodes* and *edges* as can be seen in Figure 2.1. Nodes contain information of corresponding robot poses and observations, and they are connected by edges obtained from measurements. Measurements are either acquired from odometry or loop closure detection. Before a loop closure is detected, where *nodes* present poses of robot or landmarks and *edges* are added between nodes by odometry measurements. The graph building process is known as front-end as can be seen in Figure 2.1 (a). After a loop closure is detected, as location $X_1$ and location $X_5$ shown in Figure 2.1, a configuration needs to be found (the former poses need to be corrected) since there is a transformation residual between two locations due to the drift of odometry measurements. This task is called back-end which can be treated as a large error minimization problem. The technique to solve




(a) (b)

Figure 2.1: A representation of a graph-based SLAM process. Circles represent nodes which are connected by arrows that represent edges. Arrows with solid lines are constraints from odometry measurements, while the arrow with dashed red line represents constraint resulted from loop closure detection

back-end is usually called graph optimization method. Figure 2.1 (b) is an example of corrected poses after a loop closure is detected.

### 2.3.1 Graph Optimization

Back-ends solve a more crucial problem than front-ends since they need to correct all the previous pose estimates based on the constraints. The back-end problem can be solved by minimizing the sum of squared errors between estimated and measured transitions between graph nodes. In this section, we will explain the overall problem of graph optimization.

The measurements are obtained from robot odometry first. A function $f_{ji}(\mathbf{x})$ is defined to compute a transition from node $i$ to node $j$ based on the current set of pose estimates $\mathbf{x} = (\mathbf{x}_1^T, ..., \mathbf{x}_n^T)^T$. The measured transition between the node $i$ and $j$ is represented by $\delta_{ji}$. The error $e_{ji}$ between the desired transition and measured transition is defined as:

$$e_{ji}(\mathbf{x}) = f_{ji}(\mathbf{x}) - \delta_{ji} \tag{2.33}$$

We define the cost $F_{ji}$ associated to error $e_{ji}$ as

$$F_{ji}\mathbf{x} = \frac{1}{2}(f_{ji}(\mathbf{x}) - \delta_{ji})^T \Omega_{ji}(f_{ji}(\mathbf{x}) - \delta_{ji}) \tag{2.34}$$

where $\Omega_{ji}$ is the Fisher information matrix $\Omega$ corresponding to measurement $\delta_{ji}$ which represents . Fisher information matrix is the inverse of covariance matrix. The cost function of the overall configuration $\mathbf{x}$ is

$$F(\mathbf{x}) = \frac{1}{2} \sum_{<j,i> \in \mathfrak{C}} e_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}) \tag{2.35}$$

where $\mathfrak{C} = \{< j_1, i_1 >, ..., < j_M, i_M >\}$ contains all the pairs of nodes with constraints.

Ultimately, the purpose of graph optimization is to find the configuration $\mathbf{x}^*$ which minimizes the cost function (2.35):

$$\mathbf{x}^* = \underset{\mathbf{x}}{argmin}\, F(\mathbf{x}) \tag{2.36}$$

In this Chapter, we have stated the mechanism of filters we used in sensor fusion, the differences between two major SLAM categories and procedures of graph-based SLAM. Next, we will present the hardware and related software we used in our experiments.

# Chapter 3

# Experimental Mobile Robot System and Tools

The goal of this thesis is to demonstrate that pose data obtained from sensor fusion is more accurate than single output pose estimate from SLAM. A mobile robot would be the desired carrier to help us validate this hypothesis. Therefore, we utilized a commercial mobile robot as the experimental platform. The reason we did not build our own robot is that it would save time and efforts of dealing with hardware problems and so that we can focus on the core issues. In this chapter, we describe the hardware and the related software which allow us to modulate and access the hardware.

## 3.1 Robot Operating System

All our experiments are based on the ROS (Robot Operation System). Although ROS is called an "operating system", it is not an operating system in the traditional sense. Instead, ROS is a platform which contains a collection of robot software frameworks and provides communications between heterogeneous computer cluster. The original developers of ROS are Willow Garage Lab and Stanford Artificial Intelligence laboratory. They released the first version of ROS in 2007, and from then almost every year a new version of ROS has been released. We utilized the version "Indigo Igloo" ROS which was released on July 22 2014 since it is the most reliable version and most of the ROS packages have released fully developed version on ROS Indigo. ROS Indigo

was designed to run at the Ubuntu 14.04 LTS which is the operating system of our mobile robot's computer and monitoring computer. The computer in the mobile robot is also integrated with Indigo ROS.

There are some important concepts of ROS: nodes, messages, topics and services. *Nodes* can be understood as "software module" that contain specific functions. They are executable within ROS package. The term "node" is from the visualization of ROS system at runtime. Nodes are communicating with each other by passing *messages*. A message is a simple data structure. The name of a message is formed by the name of the package plus the name of the .msg file. For example, RTAB-Map has a node which can produce visual odometry information, the message it published is called `rtabmap/odom`. But if a node wants to subscribe all the odometry information in the system, then it is a pain to write down all the odometry messages from different packages. ROS introduced a concept called *topic* to improve. A node can specify what *topics* to subscribe, and all the messages belong to that topic will be subscribed. A node may be not aware of the existence of other nodes, since all the communications are done by specifying subscribed and published topics and passing messages. Except for this topic-based publish-subscribe model, ROS has another communication paradigm called *service* which is designed for synchronous transactions. Service can provide a one-time server-client transport. A service is defined by a pair of messages: one is the request sent by client and one is the response sent by server. The server is the node that provide services.

There are other functions and commends in ROS we utilized frequently during our experiments. `roslaunch` is the command that can open the launch file which enables the executable of multiple nodes. `roslaunch` is required when you are working with a big project. Another command is called `rosbag` which can record ROS topics during a certain period of time. It can also republish the messages from a bag file. `rosbag` enables us to collect raw sensor data into a bagfile, and play it back through algorithm with different settings and configurations to debug.

### 3.1.1 ROS Enhancement Proposals

There are two important ROS Enhancement Proposals (REP) when executing ROS in mobile platforms: REP 103 (Standard Units of Measure and Coordinate Conventions) and REP 105 (Coordinate Frames for Mobile Platforms). REP 103 offers standards about preferred conventions of units and coordinate used within ROS. The default units in ROS is SI units and the convention of axis orientation of a body in ROS follows the right hand rule. In our experiments all the topics from different packages and nodes follow REP 103 standards. REP 105 specifies the naming conventions of frames of mobile robot and relationship between frames. The names of the frames that specified are:

`base_link, odom, map, earth`

The coordinate frame named `base_link` is the frame that attached to the mobile robot base. The position of this frame is usually preset at the gravity center of the robot. `odom` is defined as a world-fixed coordinate frame, it usually stays at the start position of robot. The pose of the robot in the `odom` frame is computed based on odometry sources. The `odom` frame is an accurate, short-term local reference, but it will drift as time passes. The coordinate frame called `map` is also a world-fixed frame that stays at the start position of robot. The pose in this frame is calculated from sensor observations, such as SLAM. Thus `map` frame is defined as long-term global reference. The coordinate frame `earth` is the origin of Earth Center, Earth Fixed (ECEF), it is designed to implement the interaction of multiple robots. During our experience we only utilized one robot. Thus the `earth` frame is not defined. ROS chooses a tree representation to attach frames, which means each frame can only have one parent frame but multiple child frames. Except for the `earth` frame, the rest three frames are attached by transforms as follows:

$$\texttt{map} \rightarrow \texttt{odom} \rightarrow \texttt{base\_link}$$

Each transform between two frames can only be published by one node. Except for these four conventional frames, one can define other frames, such as

the frames of onboard sensors. The position of a onboard sensor can be defined by creating static transform between its frame to the `base_link` frame.

## 3.2 Mobile Robot Platform

During the research experiments, the mobile robot platform *Clearpath Jackal Unmanned Ground Vehicle* (UGV) (see Figure 3.1) was utilized. This is a four-wheeled mobile robot equipped with GPS, Inertial Measurement Units (IMU) and an onboard computer fully integrated with Indigo ROS. It provides an excellent communication structure for hardware and software. This kind of commercial mobile robot with the pre-setup equipment can allow us start the research immediately. The robotic platform is customizable, which allows adding our own sensors and other accessories that can be powered through internal onboard PC. During the experiments, we directly use the existing ROS packages in Jackal to extract the data from the internal sensors, such as odometry and IMU. The robot's motion is controlled by a wireless PS4 joystick.

Figure 3.1: Jackal UGV

Jackal is equipped with GPS patch module PA6H manufactured by GlobalTop Technology Inc. and IMU multi-chip module MPU-9250 manufactured by TKD InvenSense. These modules are installed in the cylinder attached on Jackal chassis (see Figure 3.2).

### 3.2.1 The Robot Jackal UGV System Background

Large applications on a robot like Jackal typically involve several interconnected launch files. When Jackal's computer is powered up, a ROS package

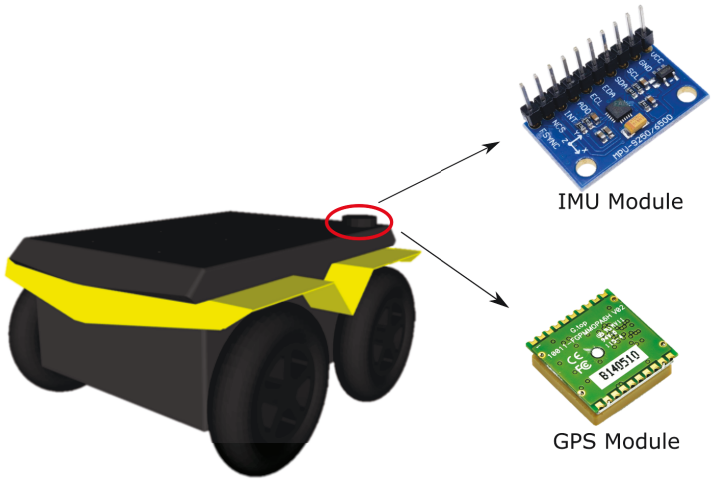



Figure 3.2: GPS module and IMU module that installed in Jackal. The size of GPS module is $15mm \times 15mm \times 2.5mm$. The size of IMU module is $25.5mm \times 15.4mm \times 3mm$

named `robot_upstart` automatically opens the top-level launch file. This launch file includes lines which control what launch files will run next. Each launch file brings up the nodes required for specific function. By using this hierarchy structure, users can modify the function easily. The default hierarchy of open launch file can be seen in Figure 3.3.

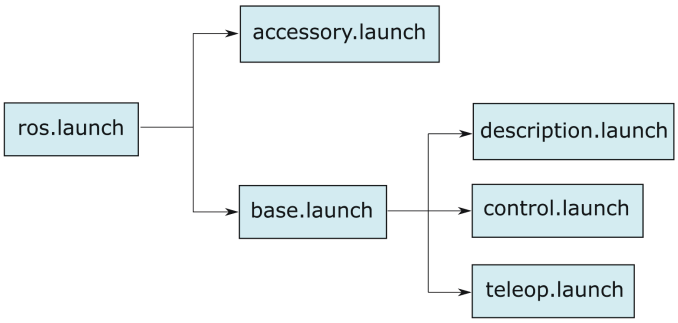



Figure 3.3: The flow diagram of opening launch file when boot the Jackal

The top-level launch file `ros.launch` consists of the commands to open the second-level launch files. `accessories.launch` file contains the nodes and parameters of Clearpath-supported sensors. This file will automatically open any supported sensors if detected. This file can be modified to adapt

customized sensor configuration. The `base.launch` file launches IMU, GPS and WiFi communication of Jackal. It also opens the launch files on the third-level, where the description launch file contains the Unified Robot Description Format (URDF) data of Jackal and positions of frames, which is used to transform sensor readings for data fusion, as well as for visualization of the Jackal model in Rviz (3D Visualization tool for ROS). The last two launch files `control.launch` and `teleop.launch` are both in the `jackal_control` package. Intuitively, this package is in charge of the control issues of Jackal. `control.launch` file handles the differential drive control and opens the EKF node for base localization. `teleop.launch` file manages the teleoperation of Jackal via joystick or via path planning in RViz. Since the PS4 controller provides direct control of the Jackal's motions, we did not employ the path planning functionality.

For the teleoperation via joystick, the configuration file (`teleop_ps4.yaml`) of the corresponding controller will be loaded first. This file contains the information of the index number of each button, related functions and maximum linear and angular velocities. One can always remap the button and modify the maximum velocities. Once the user press the button, the information of that button will be transfered to a ROS message named `/joy` by ROS node `joy_node`. Then this message will be translated to velocity commands by a ROS node `teleop_twist_joy`. The wheels of Jackal will receive the command and rotate at assigned rate. The rotary encoders attached to the wheels measure the rotations of the wheels, which is used for the navigation of the robot.

### 3.2.2 Odometry

Because Jackal has four fixed wheels, it needs a controller to split the velocity to each of the wheel so that it can execute turns. A ROS node called `diff_drive_controller` can implement differential driving.

Jackal has two motors. Optical encoder mounted integrally to the back of the motor. One set is mounted on the front left wheel, and the other is mounted on the rear right wheel. The motors are manufactured by Midwest Motion Products. Series number of the motor is "MMP S22-346F-24V

GP52-025 EU 1024". Optical encoder belongs to the "EU" series with 1024 Cycles Per Revolution (CPR). The two wheels of each side are connected using a caterpillar belt, meaning they are forced to move at the same speed. Thus Jackal UGV can be considered as a two wheel differential drive system. `diff_drive_controller` splits and assigns the velocities to the left wheels and right wheels. While the wheels get the command to move at a specific speed, encoders convert the rotations of wheels into digital signals. The output of encoder will be used as the feedback of `diff_drive_controller` to compute the odometry. The simplified odometry setup is illustrated in Figure 3.4, where $S = 0$ represents the mobile robot starting position; $S_L$ and $S_R$ represent the traveled distances of left wheel and right wheel, respectively.

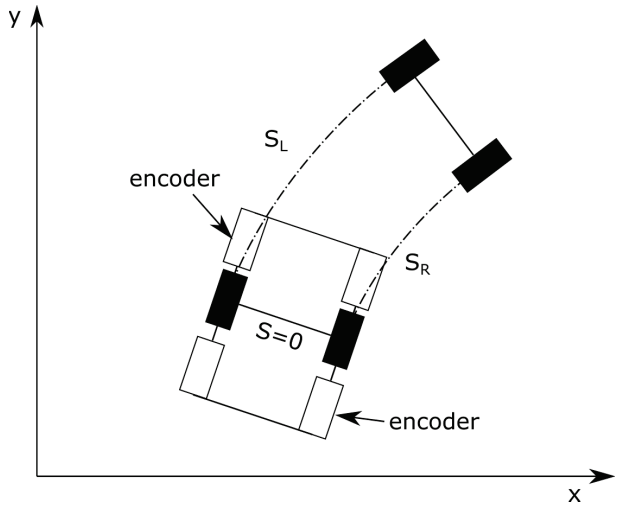



Figure 3.4: Odometry based navigation. Jackal's wheel system can be simplified as a two-wheel system. Encoder on each side of wheel enables separate actuation so that each wheel (each side of wheels) can move independently on the axle

While the two wheeled system has the advantage of reducing the complexity of driving control, it also has disadvantages such as the inevitable friction while the robot is steering and wheel skid while turning the robot. Once wheel skid happens, the accuracy of odometry navigation can get affected significantly [1]. Besides, optical encoders cannot track the exact movement of robot when the wheel is skidding, which will affect the calculation by `diff_drive_controller` of the distance the robot traveled. Thus the robot will get a poor position

estimate of itself. Besides of the skid steering drive, there are other sources which can cause odometry error. Because the measurement of the odometry is cumulative, the error will increase with time. Thus for pure odometry navigation, the position error will increase without bound. Extra sensors are needed for improving the navigation.

Jackal is equipped with another internal sensor IMU employed for sensor fusion and to determine the robot's attitude. The IMU in Jackal consists of gyroscope, accelerometer and magnetometer, which can report robot's angular velocity, accelerations and the magnetic field surrounding the body. Important parameters of each sensor can be found in Table 3.1. For more details please visit the manufacturer website and download related product datasheet.

Table 3.1: Important parameters of three sensors in IMU

| SENSOR | PARAMETER | VALUE | UNITS |
|---|---|---|---|
| Gyroscope | Total RMS Noise | 0.1 | °/s-rms |
| Accelerometer | Total RMS Noise | 8 | mg-rms |
| Magnetometer | Sensitivity Scale Factor | 0.6 | $\mu$T/LSB |

Sensor data from the IMU is fed into a node called `imu_filter_madgwick` which calculates the orientation of the robot. This ROS package is based on the code of Sebastian Madgwick who is the developer of the filter [45]. Later, Clearpath fuses this attitude data with odometry via an Extended Kalman Filter (EKF) using the `robot_localization` ROS package [46]. The sensor fusion of odometry and IMU can be seen in Figure 3.5, where $\dot{X}$, $\dot{Y}$, $\dot{Z}$ represent the linear velocities along axes $x$, $y$ and $z$, respectively; $\phi$, $\theta$ and $\psi$ represent rotations about axes $x$, $y$ and $z$, respectively (see Figure 3.11 for the definitions of $x$, $y$ and $z$). The resulting pose estimate is published as the `odometry_filtered` topic and is used as the "odometry" input of RTAB-Map. In addition to the original IMU equipped in Jackal, we installed an external IMU (*KVH 1750 IMU* Figure 3.6) to compare the performances of different IMUs as the odometry input for the SLAM solution package since graph optimization method in SLAM solution is sensitive to the twist covariance generated by IMU which we will talk about in Chapter 4. The KVH 1750 IMU

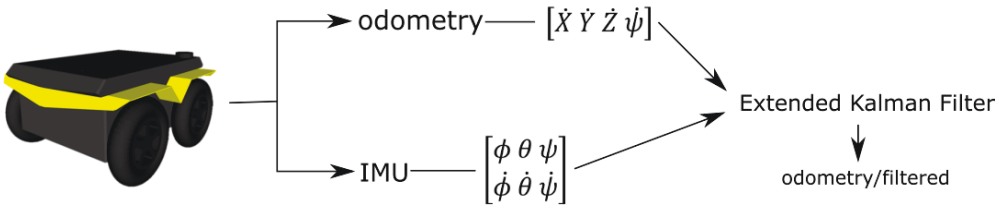



Figure 3.5: Default robot localization configuration

has tactical-grade quality and accuracy and provides an onboard state estimator for roll and pitch angles. A ROS package called `ros_kvh1750` provides an interface between this IMU and ROS and publishes topic `kvh_1750_imu/imu` which belongs to message type `nav_msgs/Imu`.


Figure 3.6: KVH 1750 IMU

## 3.3 Kinect v2

Ever since the release of Kinect for Windows v1 (Kinect v1) in 2010, it has been used in the field of robot navigation and computer vision as a reliable and low-cost sensor to acquire depth images of environment. Kinect v1 belongs to the "structured light" sensor, which means it calculates depth information using a IR (Infrared Radiation) light pattern projection. Later in 2012, Microsoft Kinect for Windows v2 (Kinect v2) an upgraded version of Kinect v1 was released, with a better resolution up to 1920×1080, a wider field of view (horizontal field of view of 70 degrees, vertical field of view of 60 degrees). Kinect v2 is time-of-flight (ToF) camera. The light from the light source infrared illuminate in Kinect v2 travels to the object, and reflects back to the

receiver infrared camera. Timing generator calculates the time $\Delta t$ of travel during this procedure. The system calculates depth from the speed of light in air and the time it traveled [47]. This method is more stable, precise and less prone to errors as compared to structured light utilized by Kinect v1.

Figure 3.7: Jackal with Kinect v2 camera

During our research, we used Kinect v2 as the only external sensor in Jackal UGV. This RGB-Depth sensor is mounted on the aluminum platform at the front of Jackal which is reserved for customization (Figure 3.7) for getting a clear and complete view of the environment. The pillars are 20 centimeters high. For the adaption of the system, we utilized an open-source Kinect v2 driver `libfreenect2` [48] and its corresponding ROS package `iai_kinect2` [49]. For obtaining a better performance, we calibrated the Kinect v2 by following the steps in the tutorial of `iai_kinect2` using OpenCV and a chessboard (Figure 3.8). Kinect v2 will publish the depth image and color image which are required by SLAM solution and its visual odometry module.

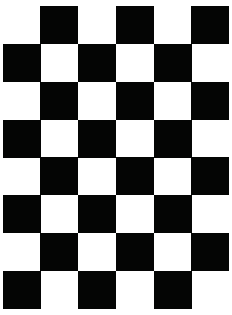


Figure 3.8: Chessboard used in calibration. The size of the cubes in the pattern is $0.03m \times 0.03m$. Chessboard can be printed on A4 paper

### 3.3.1 Visual odometry

The SLAM solution we used in our experiment includes a visual odometry module which can be used to replace wheel odometry and IMU fusion, for instance for hand-held Kinect mapping. RTAB-Map supports various visual odometry approaches. One of them is called `rgbd_odometry` which employs RGBD images produced by a camera like the Kinect v2. The visual odometry is computed using image features extracted from successive RGBD images. In order to transfer the odometry information from the Kinect frame into the robot base frame (`/base_link`), a `tf` is required between the two frames. We utilized the node `static_transform_publisher` to specify a static coordinate transform to `tf`, which was determined by manual measurement. During the experiments, the Kinect v2 was tilted by a small pitch angle to avoid seeing too much of the ground.

Whenever Kinect v2 captures an RGBD frame, image features are extracted and compared with features in a previous image. The feature matching process is based on the nearest neighbor distance ratio (NNDR). Using the matching feature correspondences between consecutive images, the transformation between them is calculated using a Perspective-n-Point (PnP) RANSAC approach [50].

## 3.4 VICON Motion Capture System

To validate the accuracy of the pose output of the research method, we set up the VICON motion caption system to provide the pose of Jackal UGV as the ground truth. VICON employs passive optical motion capture technology which uses retroreflective markers that are tracked by infrared cameras. There are ten VICON Vero cameras mounted around the experimental environment for obtaining a clear view of Jackal in 5m $\times$ 5m planar environment (Figure 3.9).

A gigabit Power over Ethernet (PoE) switch is used for the connection between cameras and the host PC. The VICON Tracker software is installed in the host PC. Six motion markers were placed on Jackal UGV (Figure 3.10),

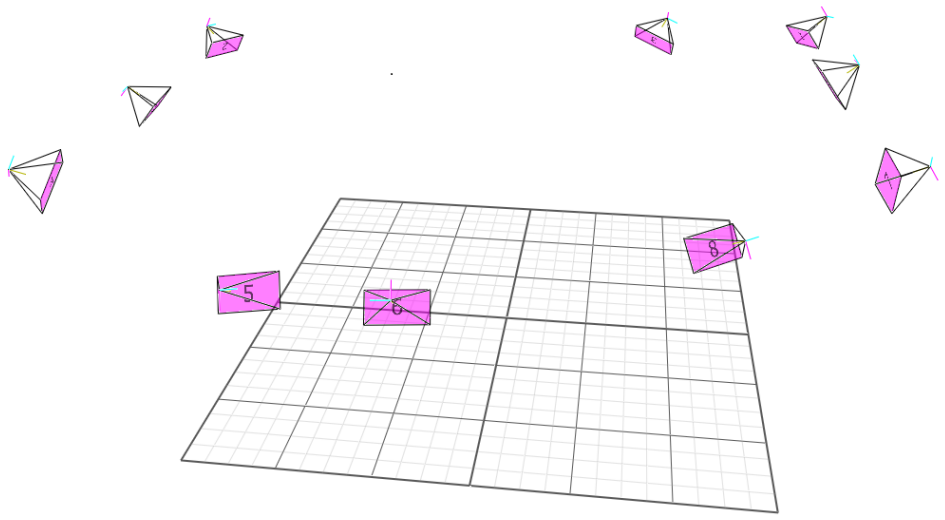



Figure 3.9: Screen capture of VICON software. Ten cameras are mounted around the experimental environment which guarantee we will not lose any frame of Jackal

which can define the rigid body object of Jackal in the VICON system. The locations of markers are based on the Jackal `base_link` frame which can be found in Figure 3.11 (a). After the object is created, we can track the motion of Jackal during the experiment with frequency of 100 Hz.

We aligned the body frame created in VICON and `base_link` frame defined in Jackal system for comparison. Also, due to the different world frames, we need to transfer the pose data from two systems into a common frame. This is done as explained below.

The Vicon system defines a robot body frame which is different from the `base_link` frame defined in Jackal (Figure 3.11). Thus pose data from the Vicon system and the on-board estimates need to be transformed into a common frame for comparison. This is done as explained below.

There is a total of PCs participating in the test: the PC in Jackal, the host PC for VICON system and the PC for data collection. When collecting data from different computers, the time synchronization is an issue because each computer uses its own timer and operating systems (the PC used for VICON system is Windows 10, while Jackal onboard PC is Ubuntu 14.04).

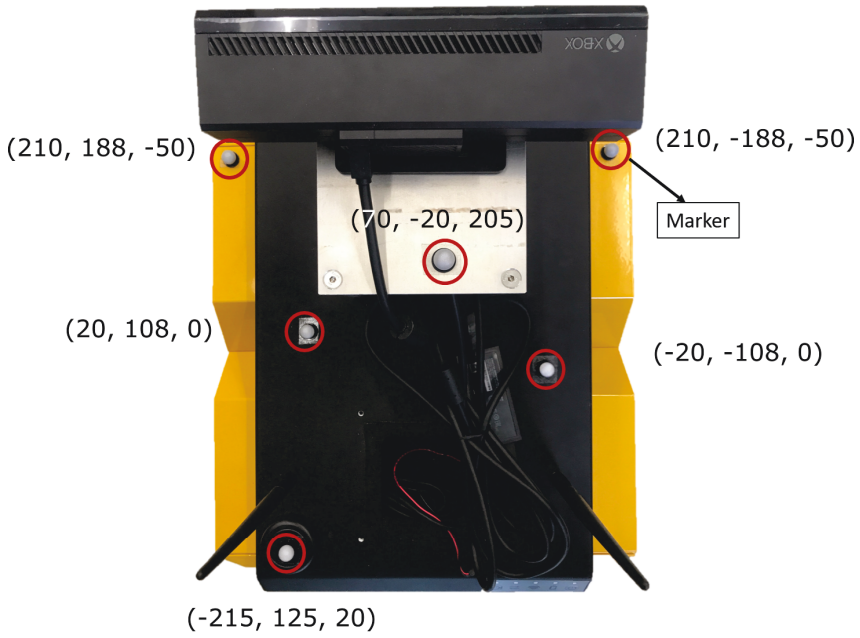
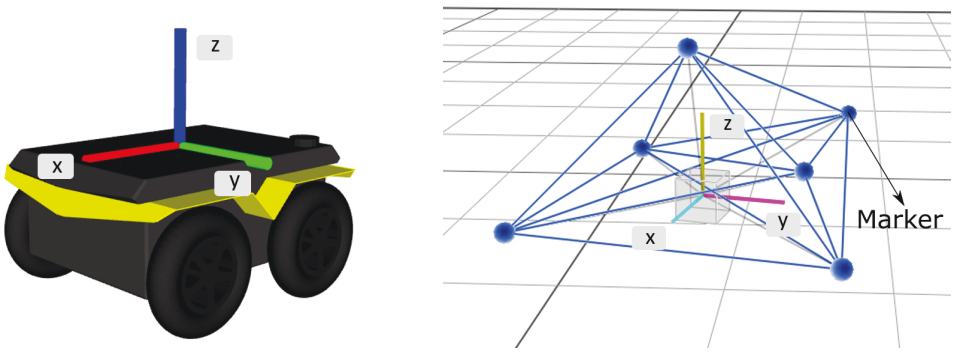



Figure 3.10: Six VICON motion capture markers are placed on the Jackal and their locations.




(a) (b)

Figure 3.11: The frames of Jackal that the pose data are based on. (a) is the Jackal base_link frame in RVIZ. (b) is the Jackal frame in VICON software

The connection between Jackal's PC and control PC can be implemented by setting Jackal as `ROS_MASTER`. Modifying Jackal's network setting can create its own WIFI access point. Once the control PC connects to Jackal via the wireless network, it can "see" the activity of Jackal including the ROS topics published by the sensors mounted on Jackal. This means the control PC can collect pose data from Jackal easily by using `rosbag record {desired_topics}`.

In order to get the datastream from VICON system, we utilized a ROS

package called `vicon_bridge`, which can publish "transform" ROS topic of assigned rigid body to specified PC (data collection PC). The data collection PC can use `rosbag` to record the data from Jackal and host PC. See the following architecture Figure 3.12 for detail.

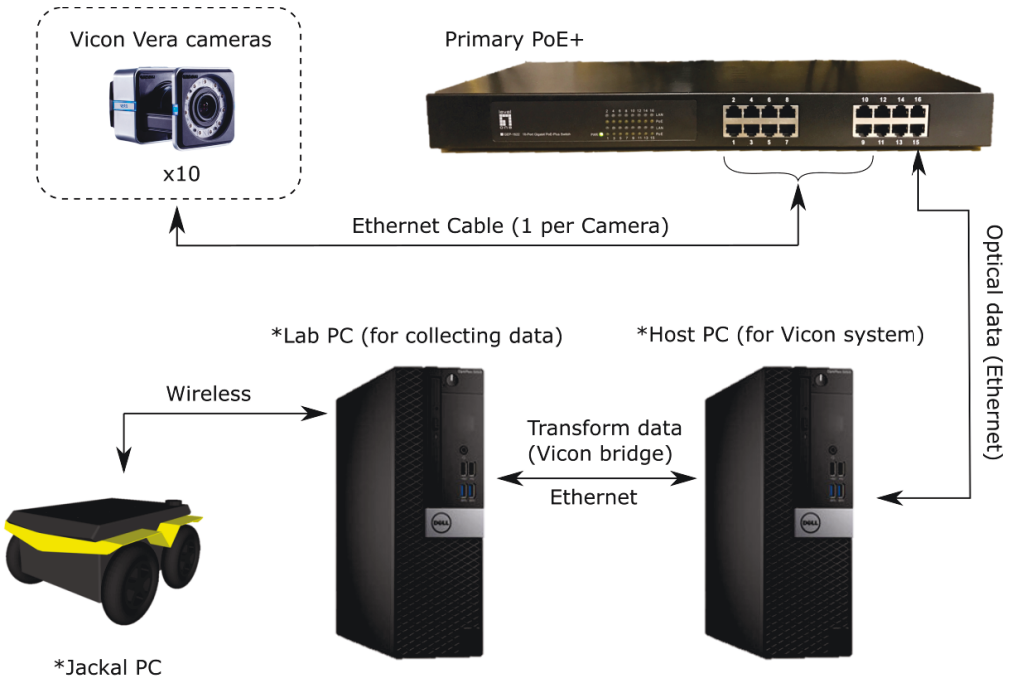
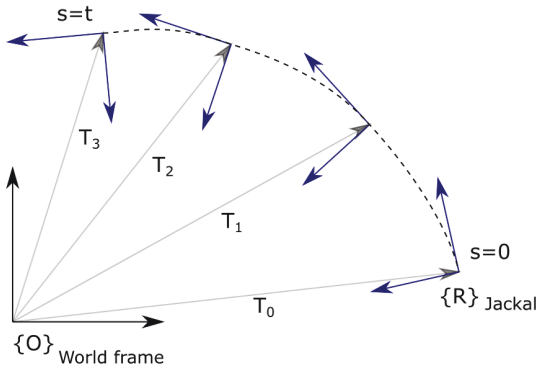


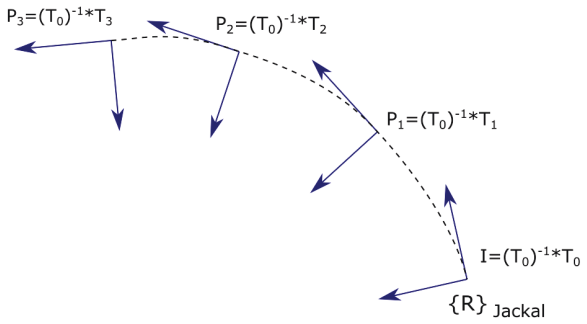

Figure 3.12: Data flow chart

As stated above, a ROS package `vicon_bridge` was utilized to build a connection that enables ROS to collect the transform data from VICON system. `vicon_bridge` publishes data generated from VICON which is called `geometry_msgs/TransformStamped`. It gives the pose with timestamp of the Jackal with respect to the VICON *world frame*. The world frame in VICON is defined during the calibration. Calibration is a two-step process: 1. Calibrating cameras; 2. Setting volume origin. The first step determines each camera's physical position and orientation. The second step uses the calibration object (wand) to define the world frame of the VICON system. In other words, the world frame is defined to be the wand, and can be changed every time the system is recalibrated. Since RTAB-Map uses the first position of the `base_link` frame as the ground-fixed frame, we must reconcile the two poses.

To implement comparison between VICON and RTAB-Map under the same world frame, we take the initial VICON derived pose $T_0 \in SE(3)$, then apply its inverse to all subsequent poses as $T_0^{-1}T$. In this way we convert the world frame in VICON from the frame of wand into the initial frame of Jackal. This method allows us to directly compare the poses from both systems. Figure 3.13 illustrates an example of this process. $\{O\}$ and $\{R\}$ are world coor-



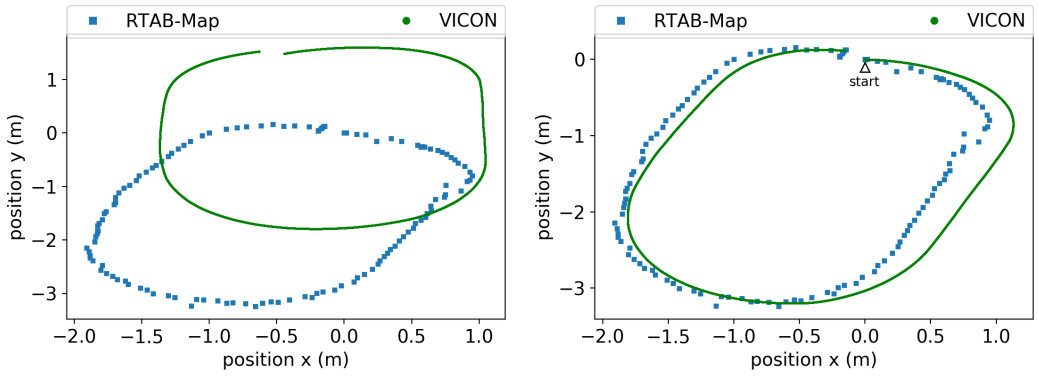

(a)




(b)

Figure 3.13: (a) is the trajectory of robot regarding world frame. (b) is the trajectory after convert the transform to pose

dinate frame and robot frame, respectively. $T_*$ represent the transform data with respect to the VICON world frame. After applying ${T_0}^{-1}$ to all transform data, we obtain the pose data $P_*$ of mobile robot. This approach allows us to implement robot pose comparison between VICON and RTAB-Map as can be seen in Figure 3.14 (b). We also put a plot of the trajectories Figure 3.14 (a) before we applied this method.




(a) (b)

Figure 3.14: (a) is the trajectory of Jackal from VICON before we applied the convert method, (b) is the trajectory after we applied the method

## 3.5 Robot Localization Package

As stated in Section 3.2.2, the ROS package `robot_localization` is first used in Jackal odometry for fusing IMU and wheel encoder data. This package contains two state estimation nodes which are based on EKF (Extended Kalman Filter) [46] and UKF (Unscented Kalman Filter) [43] and another node for integrating GPS data into the estimator. The default state estimation node in `robot_localization` is based on EKF called `ekf_localization_node` which is also the estimator utilized for fusing wheel encoder and IMU in local navigation.

Before `robot_localization` package was developed by Tom Moore [51], there already exists a ROS package aimed to implement robot pose estimation in 3D space called `robot_pose_ekf`. As the name suggests, this package uses only EKF estimator. Unlike `robot_localization`, `robot_pose_ekf` can not specify the measurements that one wants to fuse. Instead, a large covariance value needs to be manually assigned to make it ignore the specific measurement. `robot_localization` supports more input topics than `robot_pose_ekf`. Besides odometry topic (`nav_msgs/Odometry`) and IMU topic (`sensor_msgs/Imu`), `robot_localization` can subscribe pure pose and twist data with covariance:

`geometry_msgs/PoseWithCovarianceStamped`

`geometry_msgs/TwistWithCovarianceStamped`

As stated in Section 3.2.2, both data from wheel encoder and IMU are fused through EKF estimator node in `robot_localization` to provide odometry data of Jackal which is based on `odom` frame. Later, we use other estimator node to implement global sensor fusion of which the output is based on `map`. This two modes for `robot_localization` is defined by specifying four parameters: *map_frame*, *odom_frame*, *base_link_frame*, *world_frame*. The values for the first three parameters are set to the conventional frame names in REP 105 which are `map`, `odom` and `base_link`, respectively. For the last parameter *world_frame*, if it is set to `odom` the estimator node provides odometry output and publishes transform between `odom` frame to `base_link` frame. If *world_frame* is set to `map`, then the estimator provides global pose estimates and publishes transform between `map` frame to `odom` frame.

Initially the transform from `odom` to `base_link` is computed and broadcast by the odometry source of Jackal, and there is no `map` frame exist. When we run SLAM algorithm in Jackal, the transform between `map` to `odom` is broadcast by the SLAM algorithm. The transform from `odom` to `base_link` is still published by local EKF node.

After we execute the global state estimator the transform from `map` to `odom` broadcast by SLAM algorithm is disabled since one transform cannot have multiple broadcaster. The output data from SLAM algorithm is absolute pose data whose topic is `geometry_msgs/PoseWithCovarianceStamped`, which is based on the world frame defined in RTAB-Map. The world frame is the initial `base_link` frame when RTAB-Map is running on Jackal. The output from EKF node is also based on its *world_frame* which is the initial `base_link` frame when it is running. Thus we modified the output from RTAB-Map to zeros so that its world frame is converted to that of EKF when fused into EKF. In `robot_localization`, there are two parameters called "_relative" and "_differential" can help us to adjust input data. If you set "_relative" to "true", then the corresponding measurements will be fused relative to the first data it received. Parameter "_differential" is only effective to the pose data, if

it is set to "true" the corresponding pose data will be converted to *velocity*. In next Chapter, we will discuss how RTAB-Map work to achieve mapping and localization simultaneously.

# Chapter 4

# A Graph-based SLAM with Kinect v2

RTAB-Map is a graph-based solution for full SLAM problem which is an intuitive way to address the SLAM problem [10]. A graph-based SLAM problem builds the graph of the environment using the raw sensor measurements. The nodes in the graph contain the corresponding poses estimated by odometry and the edges contain the measured transitions between them. This graph construction process is usually known as the front-end in graph-based SLAM. After loop closures are added in the graph, the pose estimates at each node are adjusted to minimize an overall error cost function. This procedure is known as the back-end and can be viewed as solving a nonlinear least-squares problem.

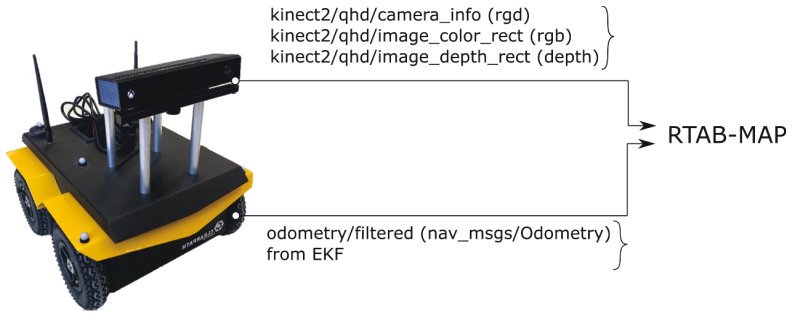
In this chapter, we will discuss RTAB-MAP, a pose graph-based solution of the SLAM problem, and procedures RTAB-Map utilized during our experiments. We will first discuss the front-end in RTAB-Map, and the results of the comparison of using different odometries as the input of RTAB-Map.

## 4.1 Experimental Setup

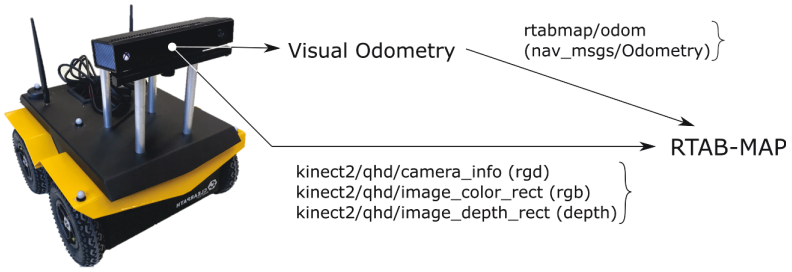
The recommended equipment for running RTAB-Map on a mobile robot is a robot equipped with a 2D LiDAR sensor, odometry, and a calibrated RGBD sensor. Jackal's odometry is provided by onboard sensors, wheel encoder and IMU, as stated in Section 3.2.2. Because the goal of our project is to imple-

ment SLAM by utilizing "low-cost" sensors and applying sensor fusion method. Thus, besides of the onboard sensors, Jackal is equipped only with Kinect v2.

RTAB-Map requires the user to provide RGBD images and robot odometry information. RGBD images can be obtained from the Kinect v2. For odometry information, there are two possible sources: wheel odometry from the Jackal, or visual odometry from the Kinect v2. The odometry from Jackal is obtained from an EKF node which fuses wheel encoder data and IMU data. Visual odometry is obtained by using RGBD information from Kinect v2, as discussed in Section 3.3.1. See Figure 4.1 for details. The ROS driver `iai_kinect2` of Kinect v2 publishes three important topics that RTAB-Map subscribes with `kinect2` prefix, `qhd` specifies the resolution of Kinect v2 optical data.




(a)




(b)

Figure 4.1: Configuration (a) using Jackal's odometry. Configuration (b) using visual odometry. Visual odometry is generated from the three RGB-D topics from Kinect v2

Before loop closure detection, the map is generated from nodes and edges created in short-term memory (STM) from odometry poses and RGBD images. The RGBD images are processed to extract image descriptors, used to form visual words for loop closure detection and for visual odometry. In the

next section, we will discuss the image descriptors utilized by RTAB-Map to implement fast image matching.

## 4.2 Image Descriptors

When Kinect v2 acquires a new image, the features of the image instead of the image itself are saved. The descriptors that can describe the visual features are called image descriptors and are formed from features such as colours, edges and textures. RTAB-Map supports almost all image features implemented in OpenCV. Speeded-Up Robust Features (SURF) used to be the default image feature of RTAB-Map, but due to patenting issues the default image features are now BRIEF+GFTT. GFTT is the abbreviation for Good Features to Track, and BRIEF stands for Binary Robust Independent Elementary Features.

### 4.2.1 SURF

SURF can be considered as a speeded-up version of SIFT. SIFT (Scale Invariant Feature transform) was invented by David Lowe in 2004 [52]. As the name suggests, SIFT features are features which stay invariant to image scaling and rotation. Keypoints are first extracted from the image and then used to calculate the corresponding descriptors. The image is convolved with Gaussian filters at different scales. DoG (Difference of Gaussians) image is the difference of two nearby scale convolved with the image. Once the DoG image is found, it will be searched for local maxima/minima over scale and space. Local extrema will be considered as the candidates of keypoints. Later, the edges in these potential keypoints will be removed using a method similar to Harris corner detector. An orientation is assigned to each keypoint by calculating the directions of gradient magnitude of neighborhood. Once we have the position, scale and orientation of the keypoint, a descriptor needs to be assigned to it. A 16×16 pixel neighborhood around the keypoint is divided into 16 4×4 blocks. Each block calculates 8 bin orientation histogram. A total 4×4×8 (128) bin descriptor is created.

Although the SIFT descriptor is widely used, it is also slow due to its high dimension. As a speeded-up version of SIFT, SURF [53] follows the overall steps of SIFT. Instead of using DoG to find keypoints which are scale invariant, SURF utilizes a Box Filter to obtain different scale images and a Hessian Matrix as a blob detector to find keypoints. Thanks to an integral image, the speed of this process is fast. For orientation assignment, SURF uses Haar wavelet responses in horizontal and vertical directions. The maximum of sum of responses within a sliding orientation window of angle 60 degrees is chosen as the dominant orientation assigned to the keypoint. Ultimately, a total 4×4×4 (64) bin descriptors is created. The smaller storage size of SURF reduces the computation cost of descriptor matching which makes SURF faster than SIFT.

### 4.2.2 Binary Image Descriptor

Because SURF is patented (non-free), the default image descriptor now in RTAB-Map is *GFTT+BRIEF*. Because BRIEF is a feature descriptor, it does not contain the ability to find the features. Thus, RTAB-Map combines GFTT, a feature detector, with BRIEF. GFTT is a corner detector proposed in late 1994 [54] which is a modified version of the Harris detector [55]. A corner is defined as the intersection of two edges. The image gradient is computed first. Each location in the image needs to be searched for the existence of a corner (feature). Thus, a small window is used to grab an image patch for analyzing and tracking the features. Within this window, displacement and deformation are applied at the center of the window $\mathbf{x}$. Then GFTT calculates the sum of squared differences between the origin image patch and the shifted one. The final approximation of the differences can be written in the form

$$S = \mathbf{d}^T Z \mathbf{d}$$

where $Z$ is the structure tensor:

$$Z = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$Z$ is a 2×2 symmetric matrix whose eigenvalues $\lambda_1$ and $\lambda_2$ can represent the intensity variations in a window. In other words, eigenvalues can determine

whether the area in the window is a flat region, an edge or a corner. If both eigenvalues are greater than a predefined threshold, then it is considered as a corner, and saved as a feature. The mechanism of GFTT corner detection can be seen in Figure 4.2.

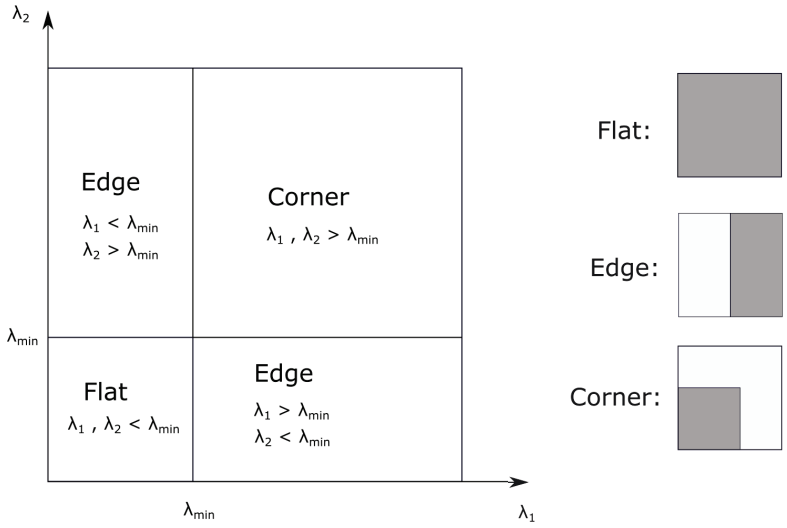



Figure 4.2: GFTT corner detection mechanism

After GFTT captures interest points, BRIEF creates a square patch and applies Gaussian smoothing to it to reduce the sensitivity. Location pairs are selected and the pixel intensity between them is compared. If the intensity of the interest point is smaller than that of paired point, then the result is 1, else it is 0. BRIEF provides three options for the numbers of location pairs: 128, 256 or 512. The corresponding storage size would be 16 bytes, 32 bytes and 64 bytes. Recall the SIFT and SURF descriptors are 128-dimensional and 64-dimensional floating point values. Each SIFT descriptor requires 512 bytes storage and each SURF descriptor requires 256 bytes. The default size of BRIEF descriptor in RTAB-Map is 16 bytes which is obviously much smaller than the 256 bytes of SURF descriptor. Due to the smaller size, BRIEF greatly reduces the feature matching time with respect to SIFT or SURF.

### 4.2.3 Nearest Neighbor Search

Once the image features of the frame are extracted and described as image descriptors, these descriptors are quantized to an incremental visual vocabulary. This process is done in STM (Short Term Memory). To decrease the time used for finding the loop closure detection, RTAB-Map divides its memory into WM (Working Memory) and LTM (Long Term Memory). The nodes in WM are used for loop closure detection, while STM is used to store the current image. Each new node created in STM is assigned a 0 weight and its visual words are compared with the previous node. If the similarity is higher than a threshold, the weight of the new node is increased to one plus the weight of the last node, and the previous node is discarded. This way, if the robot is stationary, new nodes are not added to WM. When WM reaches its memory threshold or search time threshold, nodes with lowest weights and oldest timestamp are moved into LTM (Long Term Memory) to preserve real-time operation.

The method utilized for comparing visual words between two nodes depends on the type of image descriptor. The goal is to find nearest neighbor and second-nearest neighbor of each descriptor according to the Euclidean distance. If the distance ratio of closest match and second-closest match is less than 0.8, then the closest match can be considered as the match for the descriptor [52]. Researchers found by choosing the ratio to 0.8 can eliminate 90% of the false matches [52]. For high dimensional descriptor like SURF and SIFT descriptor, using exhaustive search would be extremely time-consuming. RTAB-Map uses randomized kd-tree to find the match of image descriptors of one node with image descriptors of another node. Randomized kd-tree constructs a tree-structure searching model which can reduce the searching time by $1/2^n$ where $n$ is the number of levels of the tree. For binary descriptors like BRIEF, due to its small dimension, RTAB-Map chooses brute force method for searching for matches. If the visual odometry node in RTAB-Map is used, the features extracted are employed for both visual odometry and loop closure detection.

## 4.3 Loop Closure Detection

If the size of WM is too large, the searching time for loop closure detection will exceed the time available to maintain real-time performance. Thus, a threshold $T_{times}$ is set to control the size of WM. As stated in Section 4.2.3 when the searching time is greater than the threshold, the old locations with the lowest weights are transferred from WM to LTM. $T_{time}$ is determined by the computational capacity. If a location is transferred to LTM, its corresponding visual words will also be transferred. A location where a loop closure is identified, its neighbors are not allowed to be transferred to LTM. Also, the locations that are newly added into WM are ignored using a threshold $T_{recent} = 0.2$. The transfer process helps keep the size of WM and vocabulary to save the computational cost for finding loop closure and building the nearest-neighbor index.

The discrete Bayesian filter is utilized to follow the loop closure hypotheses of all nodes in WM, which determines whether the current location is from a previously visited location or a new location [30]. If a loop closure occurred, the full posterior is updated. The expression of the discrete Bayesian filter is:

$$p(S_t|L^t) = \eta p(L_t|S_t) \sum_{i=-1}^{t_n} p(S_t|S_{t-1} = i)p(S_{t-1} = i|L^{t-1}) \tag{4.1}$$

where $L^t$ consists of all the locations in WM and STM ($L^t = L_{-1}, ..., L_t$) and $S_t$ represents the states of loop closure hypotheses. $S_t = i$ means location $L_t$ finds its loop closure at past location $L_i$. $S_t = -1$ means there is no loop closure which occurred at location $L_t$. $\eta$ is the normalization term.

If the hypothesis $p(S_t = -1|L^t)$ is lower than the loop closure threshold $T_{loop}$, and there exits a highest loop closure hypothesis $p(S_t|L^t)$ occurred when $S_t = i$, which means location $L_t$ closes a loop at location $L_i$. Later, the rigid transformation ($SE(3)$)is calculated based on the matched features from two locations by RANSAC approach. When features are extracted, the depth information of each feature is assigned to it. If the number of inliers exceeds the threshold $I$, loop closure is accepted and a loop-closure link is created between $L_i$ and $L_t$ [56]. The weight of $L_i$ is added t the weight of $L_t$ and reset

to 0.

If the neighbors of location $L_i$ are in LTM instead of WM, then they are transferred back to the WM. This procedure is called *retrieval*. A maximum of two locations can be retrieved at each time, because loading locations from the LTM is very time consuming. And neighbors in time are preferred for retrieval more than neighbors that are linked by loop closure. For instance, if the current location's highest hypothesis is location $L_i$, the highest loop closure hypothesis of $L_i$ is $L_r$. The neighbors $L_{i-1}$ and $L_{i+1}$ will be retrieved first before the $L_r$ according to this priority guideline. Also, the retrieved locations might contain some visual words that are not in the visual vocabulary. Thus, their visual descriptors need to be determined whether they exist in the vocabulary or not. The words that cannot find matches will be added back to visual vocabulary.

## 4.4 Graph Optimization Methods

Different graph optimization approaches can be considered as different solutions for solving the problem (2.36). There are three major graph optimization methods supported by RTAB-Map: TORO (a Tree-based network optimizer) [26], g2o (general graph optimization) [27], and GTSAM (Georgia Tech Smoothing and Mapping) [57].

Both TORO and g2o optimize the problem by finding the minimizer of nonlinear squares problem iteratively. They use different optimization methods, TORO uses gradient descent, g2o uses Levenberg-Marquardt algorithm. But the update equations from both methods are executable under the assumption that the space of $\mathbf{x}$ is Euclidean. For SLAM problem, the variable belongs to $SE(3)$ group which is non-Euclidean. Therefore, they chose to express the increment $\mathbf{\Delta x}$ in the form that use a minimum representation. One can represent an increment by a translation vector and the axis of normalized quaternion with corresponding rotation angle. $\Delta x = (x, y, z), (\hat{e}, \theta)$. And $\mathbf{x}$ can be represented by translation vector and a full quaternion. The increment will be first converted to the same form as pose and then added to it.

### 4.4.1 A Tree-based Network Optimizer

Before TORO was proposed, most graph optimization approaches only considered two-dimensional pose space $SE(2)$, but could not be used for the three-dimensional pose space $SE(3)$. One of the reasons for this is the non-commutativity of rotations in the 3D case. When an error occurred in 2D space, each node can be corrected easily by applying proportional of the error. However, due to the non-commutativity of 3-dimensional rotations, this technique cannot be applied to the $SE(3)$ case:

$$R(\varphi, \theta, \psi) \neq \prod_{1}^{n} R(\frac{\varphi}{n}, \frac{\theta}{n}, \frac{\psi}{n}) \tag{4.2}$$

The technique TORO used for finding the minimizer in Equation (2.36) is gradient descent (GD), a classic first-order iterative optimization algorithm. GD updates poses along the negative gradient of $F$ evaluated at the current pose by the following equation:

$$\mathbf{x}^{t+1} = \mathbf{x}^{t} \ominus \underbrace{\lambda \cdot J_{ji}^{T} \Omega_{ji} e_{ji}}_{\Delta x} \tag{4.3}$$

where $\ominus$ indicates this operation is performed on $\mathbf{x} \in SE(3)$ poses, meaning $\Delta x$ is transformed appropriately, and $\lambda$ is the learning rate, which is adjusted as the iterations go on, and $J_{ji}$ is the Jacobian of $f_{ji}$. The term $J_{ji}^{T}$ maps the error into the parameter space, which is a linear mapping. This mapping might increase the error when applying GD. To overcome this problem, TORO separates the process into two steps: first update the rotational components and then, update the translational components.

TORO decomposes the rotational matrix $B$ that rotates the node into the desired orientation based the error into a set of increments by using the spherical linear interpolation (slerp) [58] and the eigenvalues of the covariance of the constraints: $B = B_{1:n} = B_1 B_2 ... B_n$. The number of the increments is the number of the nodes. Given the set of increments, we can calculate the set of rotations $A_n$ that update each rotation into the desired one.

$$\forall_{k=1}^{n} : A_k = R_k^{T} (B_{1:k-1})^{T} R_k B_{1:k}. \tag{4.4}$$

Once the matrices $A_n$ are determined, the rotation matrices of each nodes in the chain are updated by

$$R_k \leftarrow R_k A_k \tag{4.5}$$

The method used to update of the translational component is more straightforward than that of rotation component. The error $e_{ji}$ is distributed on each node linearly. The fraction of the error assigned on each node depends on the corresponding covariance which shows the uncertainty of front-end constraints. According to the experiment result, this type of update can give a smooth deformation along the path, and has shown good performance on reducing the error.

### 4.4.2 General Graph Optimization

In 2011, Kummerle *et al.* proposed a general framework for the graph optimization problem called g2o [27]. In this approach, the error function (2.33) is iteratively approximated by a first order Taylor expansion at the initial guess $\breve{\mathbf{x}} = (\breve{\mathbf{x}}_i, \breve{\mathbf{x}}_j)$:

$$\begin{aligned} \mathbf{e}_{ji}(\breve{\mathbf{x}}_i + \Delta\mathbf{x}_i, \breve{\mathbf{x}}_j + \Delta\mathbf{x}_j) &= \mathbf{e}_{ji}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \\ &\simeq \mathbf{e}_{ji} + \mathbf{J}_{ji}\boldsymbol{\Delta}\mathbf{x} \end{aligned} \tag{4.6}$$

The Equation (2.35) is updated by substituting its error term by above equation

$$\mathbf{F}_{ji}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) = c + 2\mathbf{b}\boldsymbol{\Delta}\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}\boldsymbol{\Delta}\mathbf{x} \tag{4.7}$$

where $c = \sum_{<j,i>\in\mathfrak{C}} \mathbf{e}_{ji}^T\boldsymbol{\Omega}_{ji}\mathbf{e}_{ji}$, $\mathbf{b} = \sum_{<j,i>\in\mathfrak{C}} \mathbf{e}_{ji}^T\boldsymbol{\Omega}_{ji}\mathbf{J}_{ji}$, $\mathbf{H} = \sum_{<j,i>\in\mathfrak{C}} \mathbf{J}_{ji}^T\boldsymbol{\Omega}_{ji}\mathbf{J}_{ji}$.

The above quadratic form of the cost function can be minimized by solving the linear function which is introduced by Levenberg-Marquardt algorithm

$$(\mathbf{H} + \lambda\mathbf{I})\boldsymbol{\Delta}\mathbf{x}^* = -\mathbf{b} \tag{4.8}$$

where $\lambda$ is a factor which is used for controlling the step size. $\lambda$ is modified at each iteration, and is decreased if the new error is smaller then the previous one. (details of update $\lambda$). The LM algorithm prevents the divergence problem happened when Hessian matrix is not positive definite which guarantee that

the direction of movement is a descent direction. The new guessing point is updated by adding the increment $\mathbf{\Delta x}$ to the previous guess

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{\Delta x} \tag{4.9}$$

In every iteration, the previous guess is used as the initial point and the input of calculating the increment.

### 4.4.3 Georgia Tech Smoothing and Mapping

Both TORO and g2o perform the optimization by solving a non-linear least squares problem (2.36). Unlike them, GTSAM expresses the optimization problem as a factor graph. A factor graph is a type of probabilistic graphical model, which is motivated by the hidden Markov model (HMM). Figure 4.3 shows a Bayesian network of a three time-step HMM. Measurements $z_1$ $z_2$ $z_3$ are known. We can obtain the hidden states $X_1$ $X_2$ $X_3$ by finding the maximum posterior probability $P(X_1, X_2, X_3|Z_1 = z_1, Z_2 = z_2, Z_3 = z_3)$.

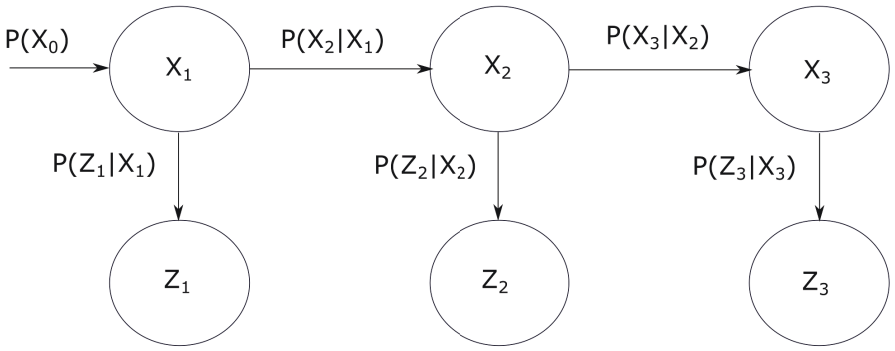



Figure 4.3: A hidden Markov model over three time steps along the direction where the arrow points. $X$ represents the state of the robot. Measurements $Z$ depend only on the state. $P(X_0)$ and $P(X_{t+1}|X_t)$ are prior probability and transition probabilities, respectively

A factor graph contains the variables and related factors with probabilistic information. The variables in GTSAM are the estimated poses. The factors are the measured transitions with associated probabilistic information obtained from odometry or from detected loop closures. The value of the factor graph

is the product of all factors

$$f(X_1, X_2, ..., X_n) = \prod_{j=1}^{m} f_j(\mathcal{X}_j) \tag{4.10}$$

where $X$ are variables, $f(\mathcal{X})$ is a factor of the variable $\mathcal{X}$. Figure 4.4 illustrates an example factor graph. The red line represents a loop closure detected between location $x_5$ and $x_2$. Except for factor $f_0x_1$ which is unary, the rest of the factors are all binary. The term $o$ in factors which relate successive poses such as $f_1(x_1, x_2; o_1)$ represents odometry measurements.

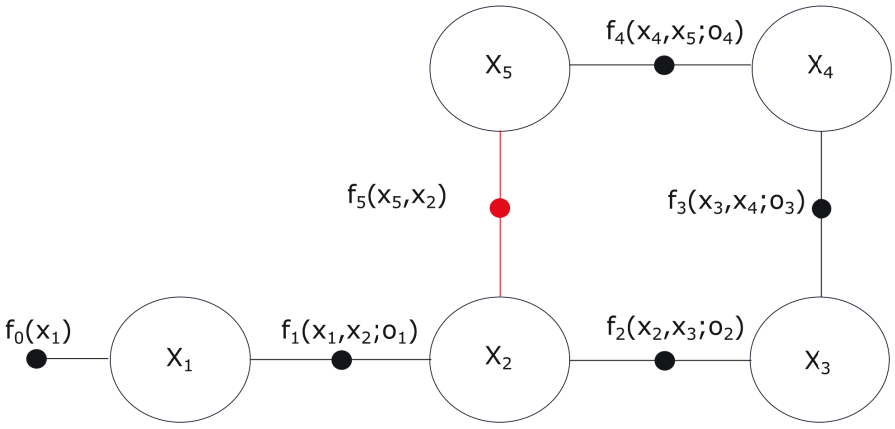



Figure 4.4: An example factor graph

The goal of GTSAM is to find the maximum a-posteriori (MAP) of the function (4.10). The factors can be considered as the prior probabilities. The method to obtain the MAP is to first assign an initial guess and then search iteratively. Due to the nonlinear $SE(3)$ dynamics of the problem as described before, a nonlinear optimizer is needed. GTSAM chose the Levenberg-Marquardt optimizer to find the MAP.

Among all three graph optimization approaches, the covariance of odometry measurements is an important factor. Poorly estimated or unreasonable covariance values may cause divergence or erroneous outputs. TORO is less sensitive to unreasonable covariances. Based on experimental evaluations [31], g2o is more likely to diverge or give poor estimates due to poor covariance estimates compared to GTSAM. The RTAB-Map developer has compared the

performances of the three graph optimization approaches, and found that GTSAM is slightly more robust than g2o in multi-session mapping [31]. Thus, we chose to use GTSAM as the graph optimization method for our experiments.

# Chapter 5

# Experimental Results

The first objective of this chapter is to compare the robot pose estimates generated by different odometries (Visual odometry and wheel odometry) using the same motion estimator (EKF). Next, robot pose estimates generated by different filters (EKF and UKF) are compared. Finally, the robot pose estimates generated by the chosen filter under different fusing modes in RTAB-Map with the chosen odometry method are compared against the ground truth from the VICON system.

## 5.1 Odometry Comparisons and Results

The raw sensor data is collected from the two different IMUs, the wheel encoders, and the Kinect v2 in the same experiment. The total of six topics are saved to a .bag file as shown in the left side of Figure 5.1. Next, RTAB-Map is run in offline mode using the different odometry inputs. As can be seen in right side of Figure 5.1, two wheel odometries are obtained from EKF nodes corresponding to the two IMU units, while visual odometry is obtained from the `rgbd_odometry` node in RTAB-Map.

### 5.1.1 Comparisons of different odometries

Each odometry mentioned above is fed separately into RTAB-Map. This is done by modifying the launch file of RTAB-Map to listen to the specified topic while replaying the .bag file. After the .bag file playback is finished, RTAB-Map has generated a grid map and a 3D map for the current odometry, which

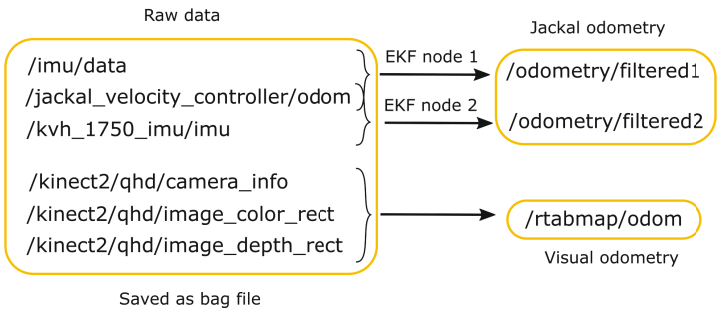



Figure 5.1: Six topics representing raw input data. First two topics are from Jackal's IMU and wheel encoders. The third topic is published by the KVH IMU. The last three topics are raw image data from Kinect v2. Two EKF nodes fuse data from different IMU with the same wheel encoder data

are saved as files. Since the image information used for each session is the same, the quality of the odometry decides the quality of the estimates. We first compared trajectories of Jackal estimated from the three odometries with that from VICON, which is shown in Figure 5.2. The jackal is controlled by PS4 controller with average speed of 0.05 m/s maximum speed of 0.1 m/s. The unit of length in ROS is meter. The time published in ROS can reach nanoseconds accuracy.

We compare each element of pose (x, y and yaw) from odometries and VICON in Figure 5.3.

Finally, the RMS (Root Mean Squared) errors of each pose element are given in Table 5.1 where r stands for radian, m stands for meter. All the raw measurement data are calculated through filters and we saved filtered estimated trajectory data from ROS. ROS publishes maximum 14 decimal length data. For RMS errors, we saved 5 decimal data.

As can be seen from the Figure 5.2, all three odometries diverge over time, and visual odometry deviates the most from the ground truth. The two wheel-derived odometries show similar performance, although wheel odometry with the KVH IMU shows a better consistency in terms of numerical results (Table 5.1): the RMSE of position $x$ from wheel odometry with the KVH IMU is 0.05 m smaller than that from wheel odometry with the Jackal IMU, although the RMSEs of position $y$ and angle yaw are close. Based on these odometry

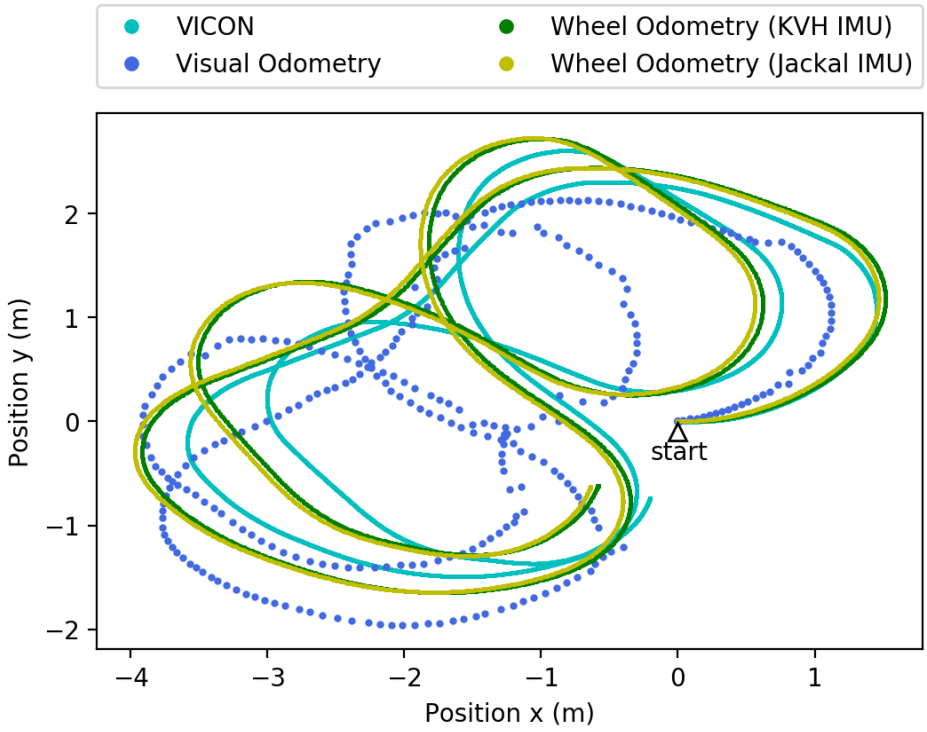



Figure 5.2: Trajectories of Jackal generated from different odometries and VICON motion capture system

Table 5.1: Root mean squared error results of the pose estimation errors from the three odometries.

| Odometry Input | Pose | Root Mean Squared Error |
|---|---|---|
| Wheel Odometry (KVH IMU) | x | 0.27562 [m] |
| | y | 0.15823 [m] |
| | yaw | 0.09554 [r] |
| Wheel Odometry (Jackal IMU) | x | 0.32004[m] |
| | y | 0.15466 [m] |
| | yaw | 0.09355 [r] |
| Visual Odometry | x | 0.71193 [m] |
| | y | 0.46460 [m] |
| | yaw | 0.09938 [r] |

comparison results, we can predict the quality of the map generated from each odometry will be:

$$\text{WO (KVH IMU)} > \text{WO (Jackal IMU)} > \text{VO}$$

where WO means wheel odometry, VO means visual odometry. Since there

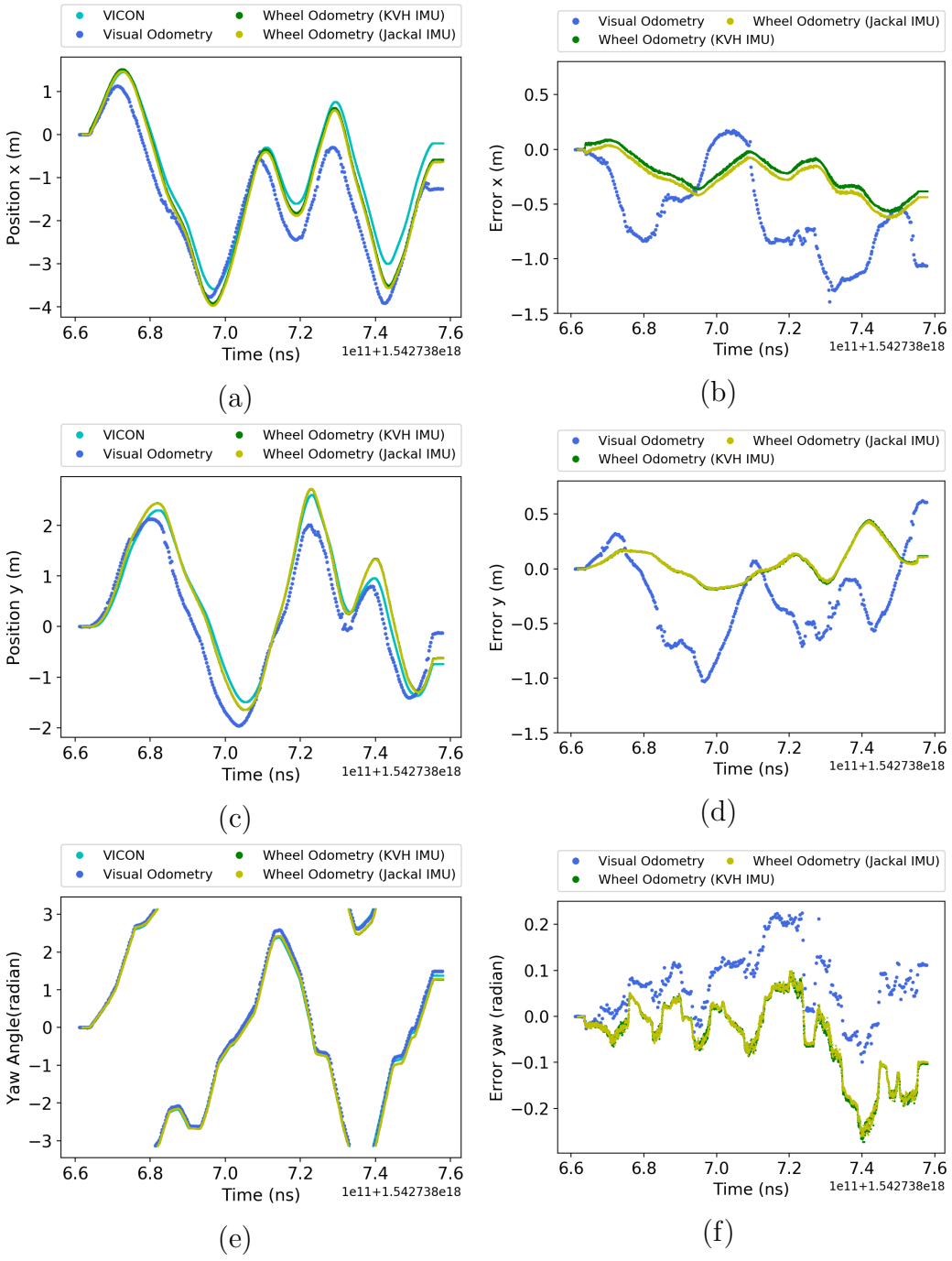



Figure 5.3: Pose from different odometries compared against VICON ground truth and corresponding error plots

is no single criteria to evaluate the quality of a map, among all the intuitive methods proposed in [59], evaluating the accuracy of the poses estimated by the SLAM system (RTAB-Map) is the most useful approach. Thus we will run RTAB-Map in localization mode to estimate the poses of the robot under each odometry while employing the corresponding map generated in this section.

### 5.1.2 Odometry Comparisons Through RTAB-Map

Instead of simply extracting pose data from the earlier mapping experiments, we ran new experiments while employing the map we obtained from the previous experiment, saving the results to another .bag file. The same topics as before (as shown in Figure 5.1) were logged. For each session, RTAB-Map was run in localization mode with each odometry input in turn, using the map created through the corresponding odometry in the previous experiment. The resulting pose estimates were saved to a .bag file, then extracted to a .csv file. The resulting trajectory estimates of the Jackal are illustrated in Figure 5.4. We also show plots of the state estimates [x,y,yaw] from each odometry compared with those from VICON in Figure 5.5. Table 5.2 shows the corresponding RMS errors.

Table 5.2: Root mean squared error results for different odometries against ground truth

| Odometry Input | Pose | Root Squared Mean Error |
|---|---|---|
| Wheel Odometry (KVH IMU) | x | 0.075815 [m] |
| | y | 0.103185 [m] |
| | yaw | 0.044472 [r] |
| Wheel Odometry (Jackal IMU) | x | 0.086022 [m] |
| | y | 0.126301 [m] |
| | yaw | 0.035164 [r] |
| Visual Odometry | x | 0.436703 [m] |
| | y | 0.247331 [m] |
| | yaw | 0.066772 [r] |

Due to the high publishing frequency of the VICON system, the corresponding trajectory looks like a smooth continuous line. The remaining three

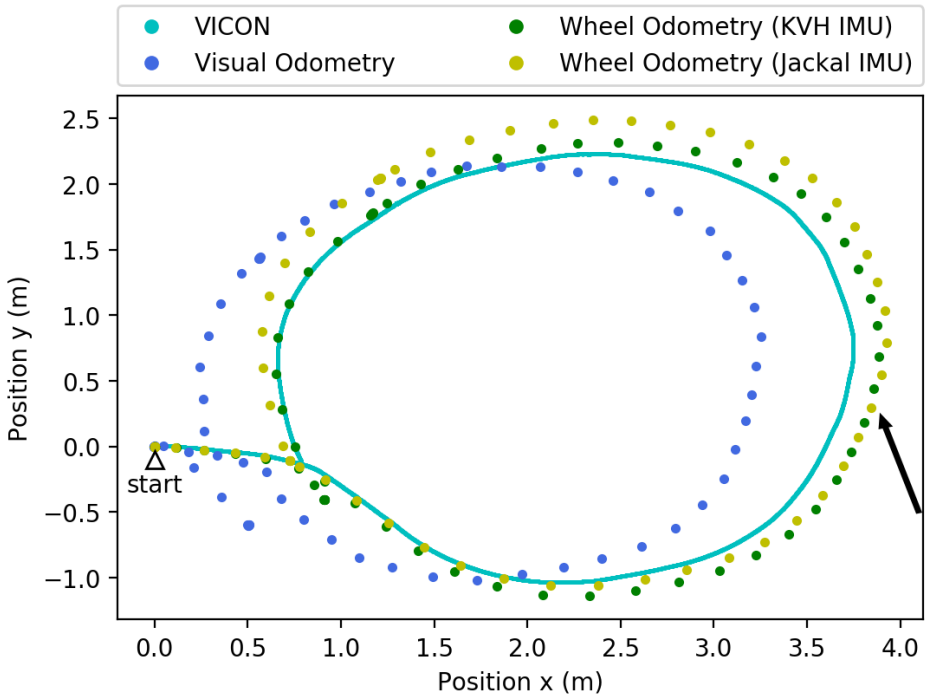



Figure 5.4: Trajectories of the Jackal estimated by RTAB-Map using three different odometries and VICON motion capture system

trajectories are sparse and discrete because the output rate of the SLAM solution is around 1 Hz. If the trajectory estimated by RTAB-Map is close to the one from VICON, then we can conclude the corresponding map correctly models our lab. As can be seen in Figure 5.4, the trajectory obtained from visual odometry has the least consistency relative to the VICON ground truth. The estimates from both wheel odometries marked in green and yellow are close to the ground truth. This result confirms the assumption we made above that the quality of the odometry input determines the quality of the map.

A low publishing rate is one of the reasons visual odometry shows poor results. The computer onboard the Jackal does not contain a Graphics Processing Unit (GPU). Thus `iai_kinect` cannot use GPU acceleration methods to perform visual odometry calculations. The frequency of visual odometry estimates in our experiment is around 4 Hz. Thus when the mobile robot moves fast, this leads to odometry loss and inaccurate outputs. On the other hand, wheel odometry can achieve a 50 Hz data publish rate.

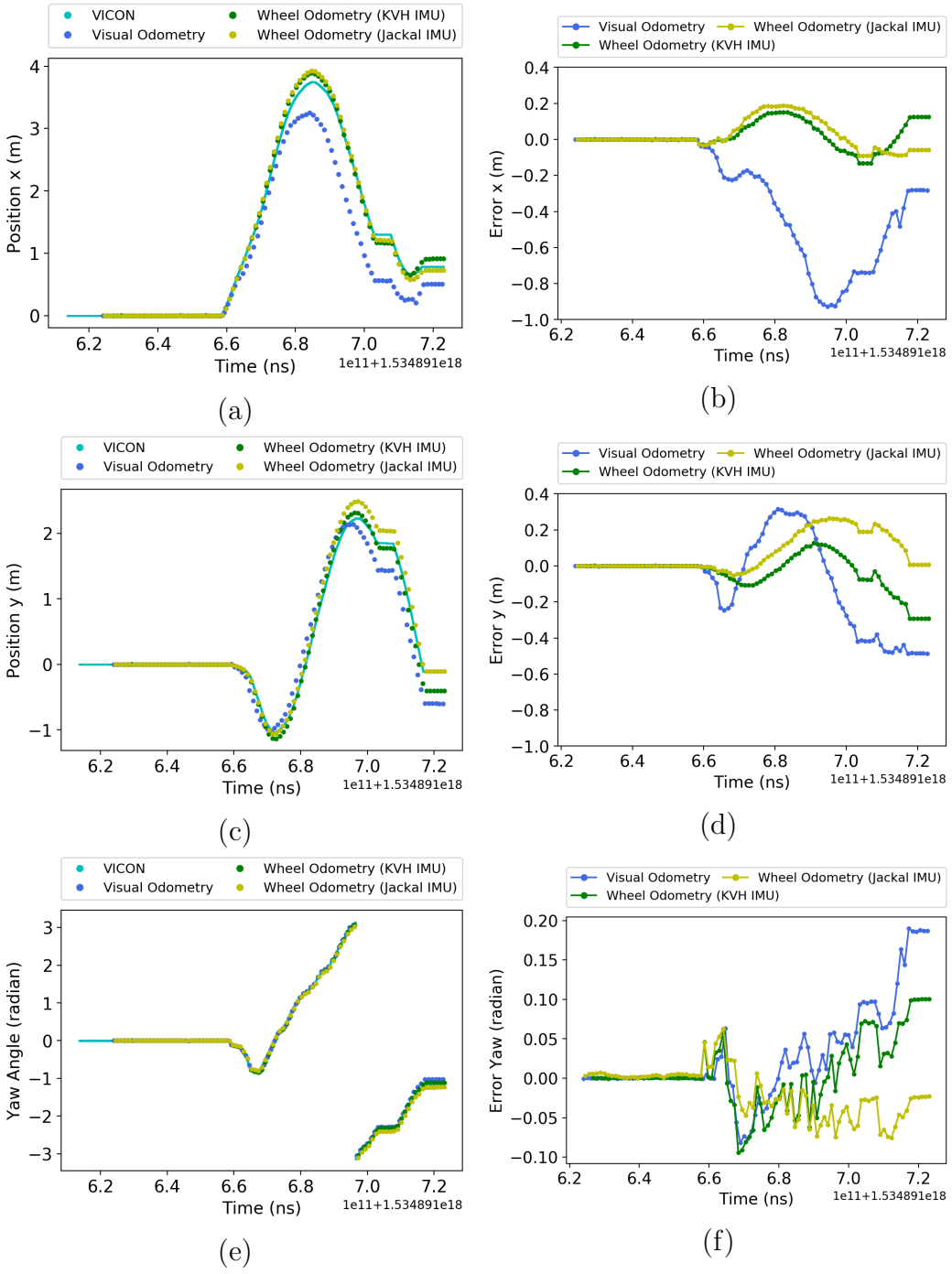



Figure 5.5: Pose estimates from RTAB-Map with different odometry inputs compared with VICON ground truth and corresponding error plots

As can be seen in Figure 5.4, the Jackal started at the point (0,0) and ended around point (0.78,-0.11). Up until the point denoted by the black arrow, the wheel odometry fused with the Jackal's (old) IMU yields a slightly

more accurate trajectory than the wheel odometry fused with the KVH (new) IMU. Later, the wheel odometry fused with the new IMU is closer to the ground truth. From the numerical results shown in Table 5.2, wheel odometry with the new IMU gives smaller RMSE values in both x and y positions than the wheel odometry with the old IMU. Figure 5.5 represents the pose estimates $(x, y, yaw)$ from the three odometries compared against the VICON ground truth. Position $x$ and position $y$ shown in Figure 5.5 (a) and (b) illustrate that the position estimates obtained from visual odometry have the worst accuracy as compared with the other two wheel odometries.

In summary, we have shown the performance of RTAB-Map pose estimation under different odometry inputs, and how the quality of the IMU affects the map generated by RTAB-Map. In the end, we choose visual odometry as the input for RTAB-Map, since we intend to prove that one can still obtain improved state estimates under sensor fusion even if the input data is lousy. Also, since wheel odometry will later be used as one of the inputs of the EKF filter, also using it as the input of RTAB-Map would lead to violating the EKF's assumption of statistical independence of the process and measurement noise, leading to erroneous results.

## 5.2 Filter Comparison Results

In Section 2.1, we introduced two filters in the `robot_localization` package: EKF and UKF. Both of them can be applied in the nonlinear dynamics case. But when the state model functions $f$ and $h$ are highly nonlinear, the EKF can give poor performance due to its reliance on linearization [43]. Also, the UKF can obtain more accurate estimates of the mean and covariance compared with the EKF under some conditions. In our experiments, we compared the performances of EKF and UKF for estimating the global pose of the Jackal. Figure 5.6 illustrates the estimated Jackal trajectories obtained form EKF and UKF compared with VICON ground truth. The trajectories from EKF and UKF are overlapping with each other. Both state estimators that available in `robot_localization` package show close performance, thus we chose the

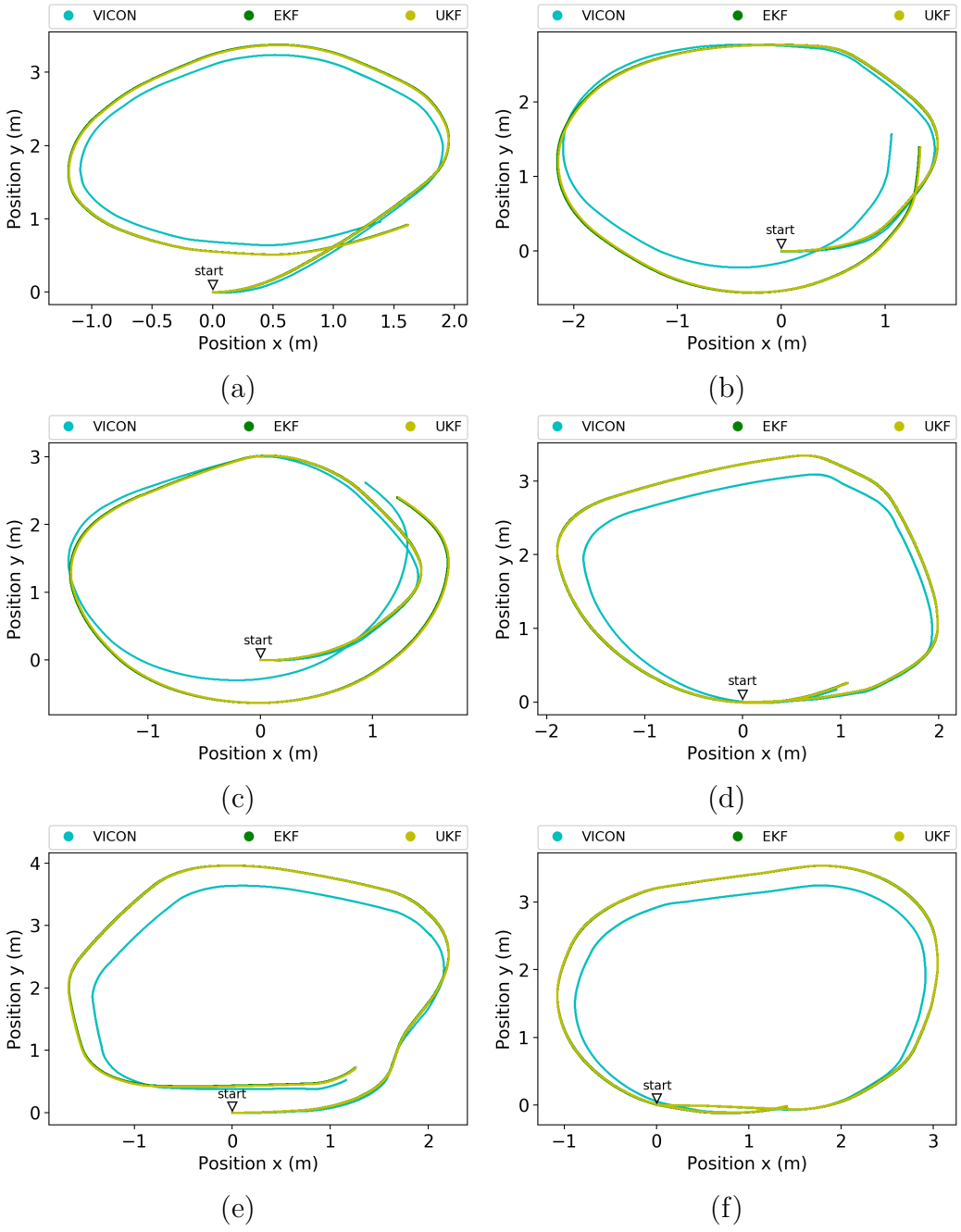



Figure 5.6: Trajectories of Jackal obtained from two different state estimation filters (EKF, UKF) and VICON motion capture system in six trials

default state estimator EKF as our estimator for later experiment.

### 5.2.1 Configuration for State Estimator

In our experiments, we implemented sensor fusion by fusing global pose estimates from RTAB-Map with onboard sensor data through an EKF estimator. We seek to find a configuration which yields good state estimates. The data from the onboard sensors (wheel odometry and IMU) that we will fuse in the EKF node are the same as those in the local EKF node (Figure 3.5). Wheel odometry provides both pose and twist data, while the IMU provides attitude and angular velocity information about the Jackal. The position data $(X, Y, Z)$ from wheel odometry diverges over time and is thus considered unreliable. Thus they are modified into velocities when fused into EKF. Our experiment results show that fusing full pose data and fusing only position data from RTAB-Map into EKF lead to similar performances. Figure 5.7 shows the trajectory estimation results of the Jackal in three trials.

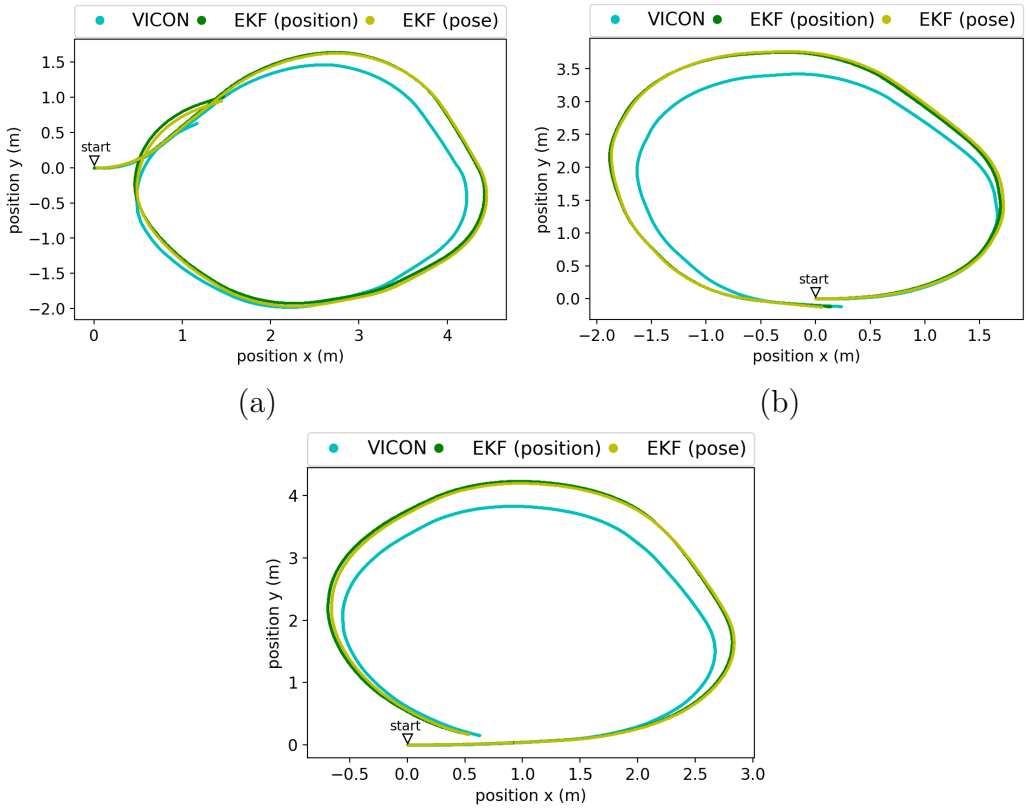



Figure 5.7: Trajectories of Jackal from EKF-based state estimator with two different configurations in three trials (a), (b), (c)

The publishing frequency of RTAB-Map estimated poses is around 1 Hz, and the orientation data usually has a high corresponding covariance. The EKF estimator thus puts more trust into the orientation data from the IMU with high frequency. Thus fusing only position data versus full pose data from RTAB-Map shows little difference. The flow chart of EKF estimator process is shown in Figure 5.8.

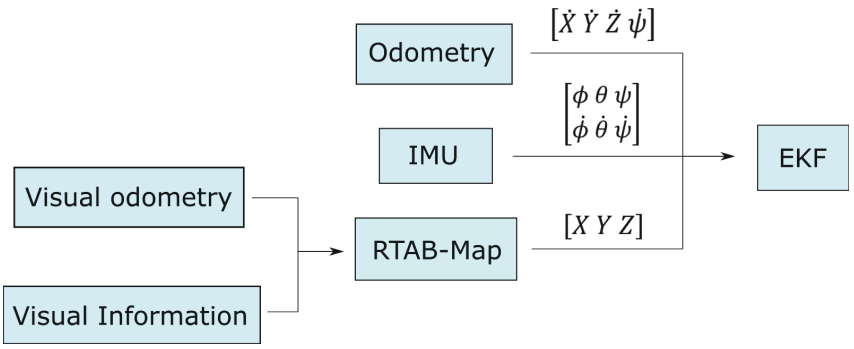



Figure 5.8: Flow chart of measurement of estimation process

## 5.3 Global Sensor Fusion Results

In Section 3.5, we mentioned that by setting the parameters "_relative" and "_differential", the pose data input into the EKF estimator node can be modified to start at the current frame. The pose data from RTAB-Map is generated relative to the world_frame defined within RTAB-Map. Thus these poses need to be modified before being passed to the EKF.

### 5.3.1 Results of _differential and _relative

By setting the parameter "_relative" to "true", the EKF estimator converts the input pose data to a pose relative to the first input pose data point. When the parameter "_differential" is enabled, the absolute pose data from RTAB-Map is integrated differentially. Both parameters cannot be set to "true" at the same time. The trajectories of VICON, RTAB-Map and EKF fusion-derived results of 6 trials are shown in Figure 5.9. Pose estimates from EKF results compared with VICON and corresponding error plots can be found in Appendix A. Table

5.3 shows the RMSE of each trial and the average.

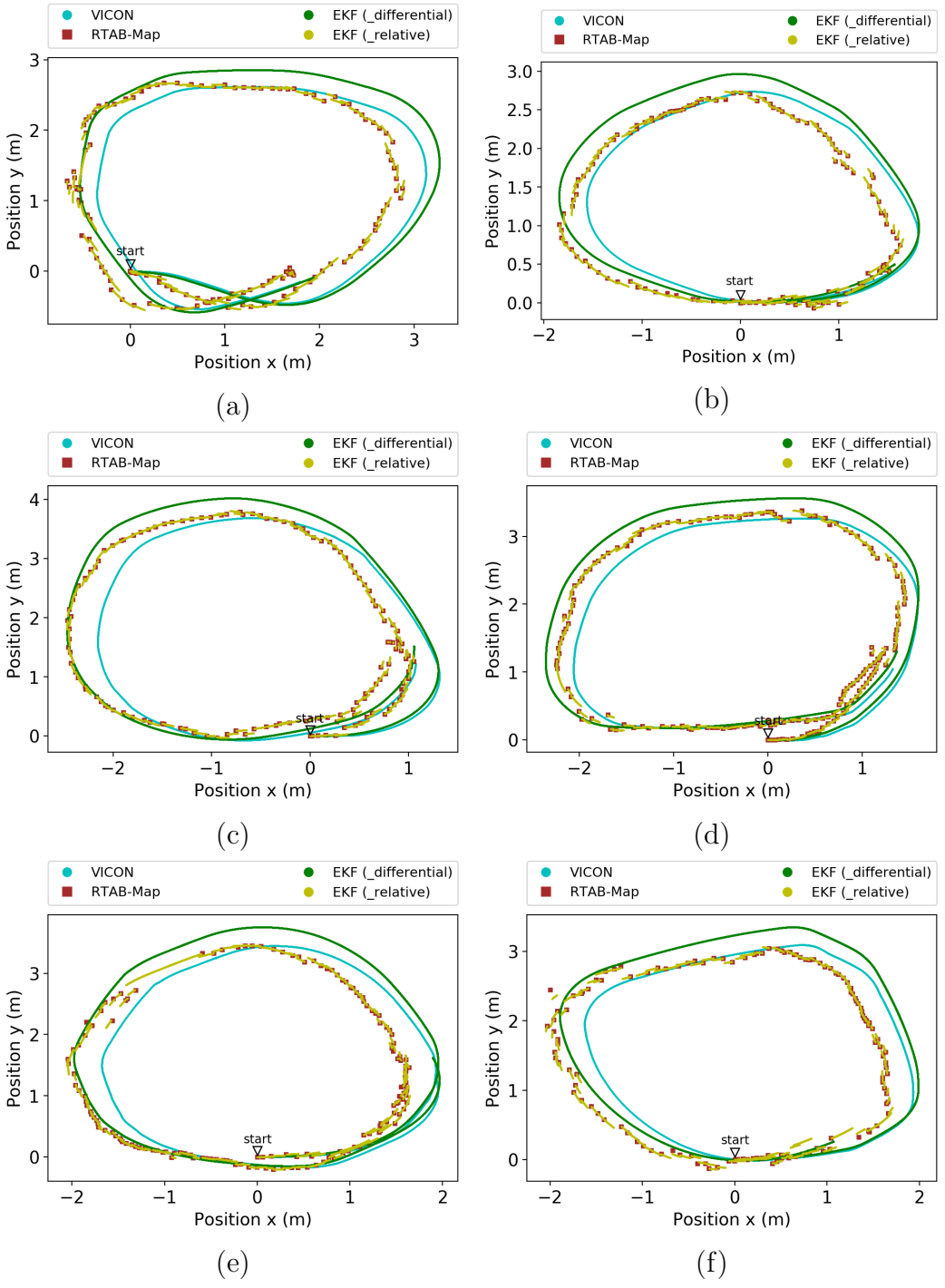



Figure 5.9: Pose outputs from EKF fusion and RTAB-Map compared with VICON ground truth

Comparing the estimated trajectories using the two settings, those obtained from "_relative" have discrete jumps in position, while those from

Table 5.3: Root mean squared error results for EKF fusion against VICON ground truth

| | Parameter | Position x [m] | Position y [m] | Yaw angle [r] |
|---|---|---|---|---|
| Trial1 | EKF (_differential) | 0.1396 | 0.1335 | 0.01460 |
| | EKF (_relative) | 0.2740 | 0.1276 | 0.01461 |
| Trial2 | EKF (_differential) | 0.1781 | 0.1541 | 0.01439 |
| | EKF (_relative) | 0.3172 | 0.1671 | 0.01448 |
| Trial3 | EKF (_differential) | 0.1854 | 0.2071 | 0.01612 |
| | EKF (_relative) | 0.2796 | 0.1974 | 0.01614 |
| Trial4 | EKF (_differential) | 0.1765 | 0.1819 | 0.01886 |
| | EKF (_relative) | 0.2102 | 0.1497 | 0.01889 |
| Trial5 | EKF (_differential) | 0.1798 | 0.1940 | 0.01760 |
| | EKF (_relative) | 0.3083 | 0.1504 | 0.01762 |
| Trial6 | EKF (_differential) | 0.1518 | 0.1808 | 0.01475 |
| | EKF (_relative) | 0.3442 | 0.1642 | 0.01478 |
| Average | EKF (_differential) | 0.1685 | 0.1752 | 0.01605 |
| | EKF (_relative) | 0.2890 | 0.1594 | 0.01609 |

"_differential" are continuous and smoother. The difference between "_relative" and "_differential" can be understood as the difference between fusing position and fusing velocities in an EKF estimator. Because the absolute position is input into the EKF, the trajectory obtained from "_relative" has discrete jumps. When "_differential" is set to true, all the input sensor data input into the EKF do not contain absolute data: linear velocities are provided by wheel odometry and RTAB-Map, while rotational velocities are provided by IMU. In other words, the pose estimates are calculated from velocities only. From RMSE results of two different settings, results from "_differential" show better results on position x. "_relative" gives slightly better results on position y. Finally, both settings show close performances on yaw angle. When set "_different" to true, its trajectories are smoother and closer to VICON ground truth than the conditions when "_relative" is enabled. However, using absolute poses can guarantee that the estimated pose does not diverge over time. But the quality of the EKF pose estimates relies on the quality of the absolute pose fused into it. If the RTAB-Map generates poor global poses, then there

is a high chance for the EKF estimator to give poor estimates.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this thesis, we started by reviewing inertial navigation systems, which are limited by the accumulation of estimation error. Aided navigation, in particular GPS-aided navigation, can be used to update state estimates from inertial sensors and bound this error. We then introduced a navigation system which does not rely on an external reference system such as GPS. This design is called SLAM, considered as a fundamental problem in robotics. The objective of our thesis was to combine the SLAM solutions with inertial sensors to implement SLAM-aided navigation which could give better results than just SLAM or inertial navigation.

We compared SLAM solutions from two different categories, filtering and smoothing. We chose graph-based SLAM which belongs to the smoothing category as our SLAM solution for experiment. Among the open-source SLAM solutions, RTAB-Map was chosen for our project. The robot we used in our work is a commercial wheeled platform called Jackal designed and manufactured by Clearpath Robotics. Next, we successfully implemented RTAB-Map on the Jackal using only a Kinect v2 as the depth sensor, and we studied the influence of odometry input choice on the performance of RTAB-Map. The choice of input odometry was found to have a great effect on estimation accuracy. The pose estimates obtained from RTAB-Map were then fused with inertial measurements from the robot's wheel encoder and IMU sensors using a state estimator. There are two types of state estimators available, EKF

and UKF, and we compared their performance and chose EKF based on our results. We investigated the effects of fusing either full pose or just position from RTAB-Map, as well as tuning various parameters. Orientation data from RTAB-Map was found to have little influence on the final results, and both variations showed good fidelity against ground truth information obtained from a Vicon indoor motion capture system. We thus successfully implement SLAM-aided navigation on a mobile robot under the following conditions:

- Mobile robot Jackal as the base platform initially mounted with IMU, wheel encoders, GPS and computer
- Kinect v2 as the only external sensor
- Robot is running at average speed of 0.05 m/s, maximum speed of 0.1 m/s

## 6.2 Future Work

Future work can include adding the ROS package `move_base` to our system. This package provides the robot with the ability to move itself to a given goal in the world map while avoiding obstacles. It can be considered as a criteria to evaluate the quality of the map created by RTAB-Map by assessing how well the robot is able to maneuver in complex environments.

From experimental results, we found that the quality of odometry greatly affects the performance of RTAB-Map. The low output rate of visual odometry might be the reason of causing the poor performance of RTAB-Map. In the future, we could investigate these issues and improve the situation by adding an extra CPU, GPU or choosing a different algorithm for visual odometry.

Finally, long-term and large-scale indoor testing, outdoor testing and 3D (SE(3)) condition should be considered. Kinect v2 should be changed to other sensors that can provide depth information during daytime since depth camera in Kinect v2 will be blinded by direct sunlight [60]. In our experiments, the VICON motion capture system provided the ground truth of robot pose, but

this confined testing to a small indoor lab, which is why we did not run large-scale indoor or outdoor test. However these tests would allow to assess the memory-management feature offered by RTAB-Map.

# References


[1] J. Farrell, *Aided Navigation: GPS with High Rate Sensors*, ser. McGraw-Hill professional engineering: Electronic engineering. McGraw-Hill Education, 2008. 1, 25

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. 2

[3] P. Mountney, D. Stoyanov, A. Davison, and G.-Z. Yang, "Simultaneous stereoscope localization and soft-tissue mapping for minimal invasive surgery," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2006*, R. Larsen, M. Nielsen, and J. Sporring, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 347–354. 2

[4] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986. 2

[5] N. Ayache and O. D. Faugeras, "Building, registrating, and fusing noisy visual maps," *The International Journal of Robotics Research*, vol. 7, no. 6, pp. 45–65, 1988. 2

[6] R. Chatila and J. Laumond, "Position referencing and consistent world modeling for mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, Mar. 1985, pp. 138–145. 2

[7] J. L. Crowley, "World modeling and position estimation for a mobile robot using ultrasonic ranging," in *Proceedings, 1989 International Conference on Robotics and Automation*, May 1989, 674–680 vol.2. 2

[8] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous Robot Vehicles*, I. J. Cox and G. T. Wilfong, Eds. New York, NY: Springer New York, 1990, pp. 167–193. 2, 15

[9] H. Durrant-Whyte, D. Rye, and E. Nebot, "Localization of autonomous guided vehicles," in *Robotics Research*, G. Giralt and G. Hirzinger, Eds., London: Springer London, 1996, pp. 613–625. 2

[10] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, pp. 31–43, 2010. 2, 37

[11] M. Montemerlo, "Fastslam: A factored solution to the simultaneous localization and mapping problem with unknown data association," PhD thesis, Carnegie Mellon University, Pittsburgh, PA, Jul. 2003. 2

[12] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 1, Oct. 2003, 206–211 vol.1. 2

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb. 2007. 2, 4

[14] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters for view-based slam," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1100–1114, Dec. 2006. 2

[15] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004. 2

[16] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, Oct. 1997. 3

[17] J. Neira and J. D. Tardos, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 890–897, Dec. 2001. 3

[18] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 2262–2269. 3

[19] D. Hähnel, S. Thrun, B. Wegbreit, and W. Burgard, "Towards lazy data association in slam," in *Robotics Research. The Eleventh International Symposium*, P. Dario and R. Chatila, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 421–431. 3

[20] J. .-.-. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No.99EX375)*, Nov. 1999, pp. 318–325. 3

[21] A. Howard, M. Mataric, and G. Sukhatme, "Relaxation on a mesh: A formalism for generalized localization," vol. 2, Feb. 2001, 1055–1060 vol.2. 3

[22] U. Frese, P. Larsson, and T. Duckett, "A multilevel relaxation algorithm for simultaneous localization and mapping," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 196–207, Apr. 2005. 3

[23] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006. 3

[24] M. Kaess, A. Ranganathan, and F. Dellaert, "Isam: Fast incremental smoothing and mapping with efficient data association," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Apr. 2007, pp. 1670–1677. 3

[25] K. Konolige, "Large-scale map-making," in *Proceedings of the 19th National Conference on Artifical Intelligence*, ser. AAAI'04, San Jose, California: AAAI Press, 2004, pp. 457–463. 3

[26] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard, "Efficient estimation of accurate maximum likelihood maps in 3d," *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3472–3478, 2007. 3, 44

[27] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011, pp. 3607–3613. 3, 44, 46

[28] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1271–1278. 4

[29] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *CoRR*, 2016. 4

[30] M. Labbé and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, Jun. 2013. 4, 43

[31] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 0, no. 0, 4, 48, 49

[32] J. Kim and S. Sukkarieh, "Slam aided gps/ins navigation in gps denied and unknown environments," *The 2004 International Symposium on GNSS/GPS*, Jan. 2004. 5

[33] J. Kelly and G. S. Sukhatme, "Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration," *The International Journal of Robotics Research*, vol. 30, no. 1, pp. 56–79, 2011. 5

[34] D. Strelow and S. Singh, "Optimal motion estimation from visual and inertial measurements," in *Sixth IEEE Workshop on Applications of Computer Vision, 2002. (WACV 2002). Proceedings.*, Dec. 2002, pp. 314–319. 5

[35] H. Rehbinder and B. Ghosh, "Pose estimation using line-based dynamic vision and inertial sensors," *Automatic Control, IEEE Transactions on*, vol. 48, no. 2, pp. 186–199, 2003. 5

[36] G. Antonelli, S. Chiaverini, and S. Costantini, "An experimental implementation or sensor fusion for mobile robots based on kalman filtering," in *Proceedings World Automation Congress, 2004.*, vol. 15, Jun. 2004, pp. 479–484. 5

[37] A. Singhal and C. Brown, "A multilevel bayesian network approach to image sensor fusion," in *Proceedings of the Third International Conference on Information Fusion*, vol. 2, Jul. 2000, WEB3/9–WEB316 vol.2. 5

[38] Y. Zhang and Q. Ji, "Active and dynamic information fusion for multisensor systems with dynamic bayesian networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 2, pp. 467–472, Apr. 2006. 5

[39] A.-M. Zou, Z.-G. Hou, S.-Y. Fu, and M. Tan, "Neural networks for mobile robot navigation: A survey," in *Advances in Neural Networks - ISNN 2006*, J. Wang, Z. Yi, J. M. Zurada, B.-L. Lu, and H. Yin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1218–1226. 5

[40] S. Rönnbäck, *Developement of a ins/gps navigation loop for an uav*, Validerat; 20101217 (root), 2000. 5

[41] M. L. Anjum, J. Park, W. Hwang, H. Kwon, J. Kim, C. Lee, K. Kim, and D. "Dan" Cho, "Sensor data fusion using unscented kalman filter for accurate localization of mobile robots," in *ICCAS 2010*, Oct. 2010, pp. 947–952. 5

[42] R. G. Brown and P. Y. C. Hwang, *Introduction to random signals and applied kalman filtering: with MATLAB exercises and solutions; 3rd ed.* New York, NY: Wiley, 1997. 9

[43] J. K. U. Simon J. Julier, *New extension of the kalman filter to nonlinear systems*, 1997. 12–14, 34, 57

[44] F. Gustafsson and G. Hendeby, "Some relations between extended and unscented kalman filters," *IEEE Transactions on Signal Processing*, vol. 60, no. 2, pp. 545–555, Feb. 2012. 14

[45] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm," in *2011 IEEE International Conference on Rehabilitation Robotics*, Jun. 2011, pp. 1–7. 26

[46] R. E. Kalman, "A new approach to linear filtering and prediction problems," *ASME Journal of Basic Engineering*, 1960. 26, 34

[47] J. Sell and P. O'Connor, "The xbox one system on a chip and kinect sensor," *IEEE Micro*, vol. 34, no. 2, pp. 44–53, Mar. 2014. 28

[48] L. Xiang, F. Echtler, C. Kerl, T. Wiedemeyer, Lars, hanyazou, R. Gordon, F. Facioni, laborer2008, R. Wareham, M. Goldhoorn, alberth, gaborpapp, S. Fuchs, jmtatsch, J. Blake, Federico, H. Jungkurth, Y. Mingze, vinouz, D. Coleman, B. Burns, R. Rawat, S. Mokhov, P. Reynolds, P. Viau, M. Fraissinet-Tachet, Ludique, J. Billingham, and Alistair, *Libfreenect2: Release 0.2*, Apr. 2016. 28

[49] T. Wiedemeyer, *IAI Kinect2*, `https://github.com/code-iai/iai_kinect2`, Accessed June 12, 2015, University Bremen: Institute for Artificial Intelligence, 2014 – 2015. 28

[50] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*, 2nd. O'Reilly Media, Inc., 2013. 29

[51] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, 2014. 34

[52] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. 39, 42

[53] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. 40

[54] J. Shi and C. Tomasi, "Good features to track," 1994, pp. 593–600. 40

[55] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, 1988, pp. 147–151. 40

[56] M. Labbé and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 2661–2666. 43

[57] F. Dellaert, *Overview in this document i provide a hands-on introduction to both factor graphs and gtsam.* 2012. 44

[58] T. Barrera, A. BarreraKristiansen, A. Hast, and E. Bengtsson, "Incremental spherical linear interpolation," 2004. 45

[59] B. M. F. da Silva, R. S. Xavier, T. P. do Nascimento, and L. M. G. Gonsalves, "Experimental evaluation of ros compatible slam algorithms for rgb-d sensors," in *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, Nov. 2017, pp. 1–6. 54

[60] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, *Conference presentation slides on kinect v2 for mobile robot navigation: Evaluation and modeling*, Jul. 2015. 65

# Appendix A

# Pose Elements Comparisons Results of EKF Fusion

Following figures are the pose estimates from EKF fusion compared with VICON and corresponding error plots of 6 trials.

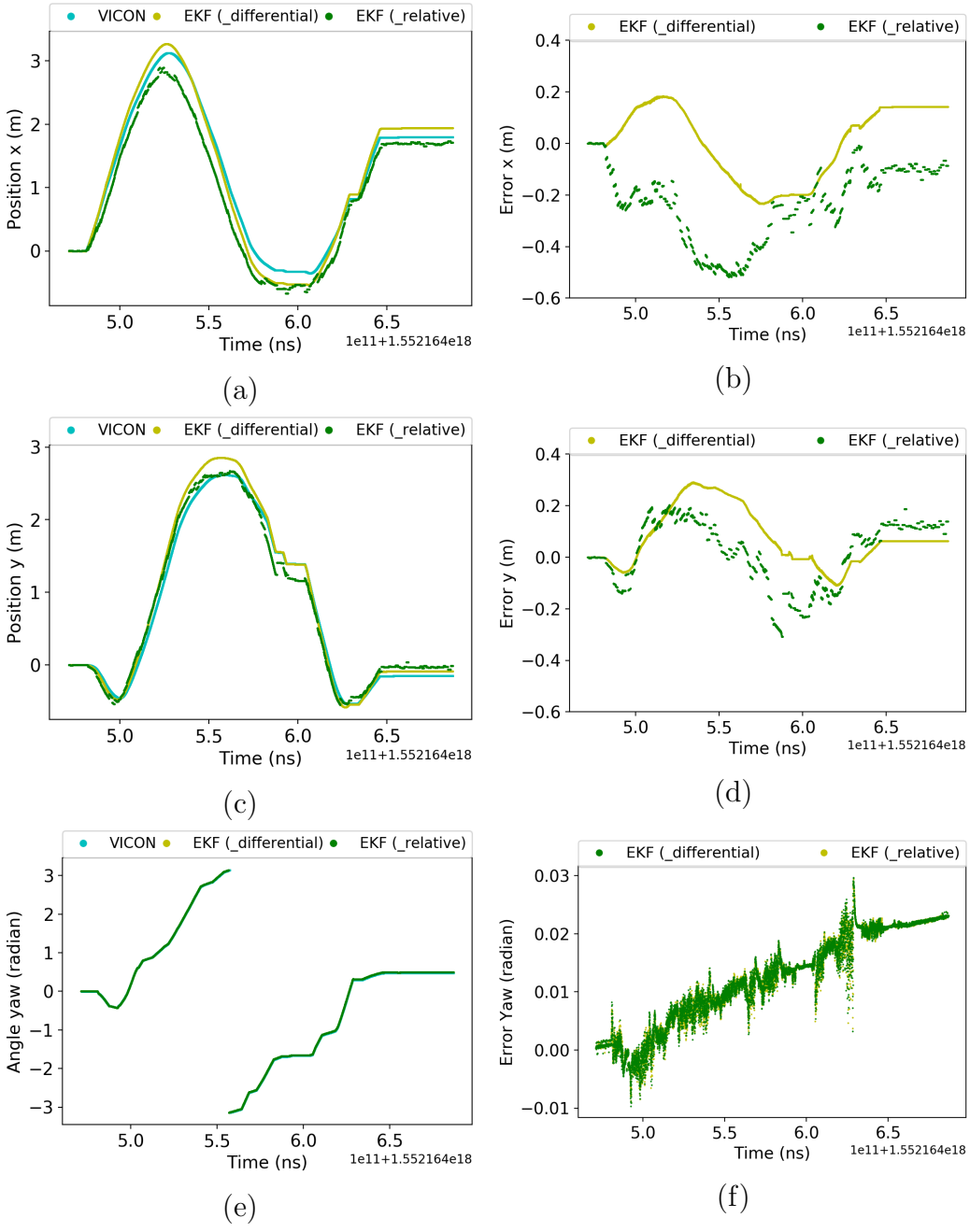



Figure A.1: Pose estimates from EKF fusion results compared with VICON and corresponding error plots of trial 1

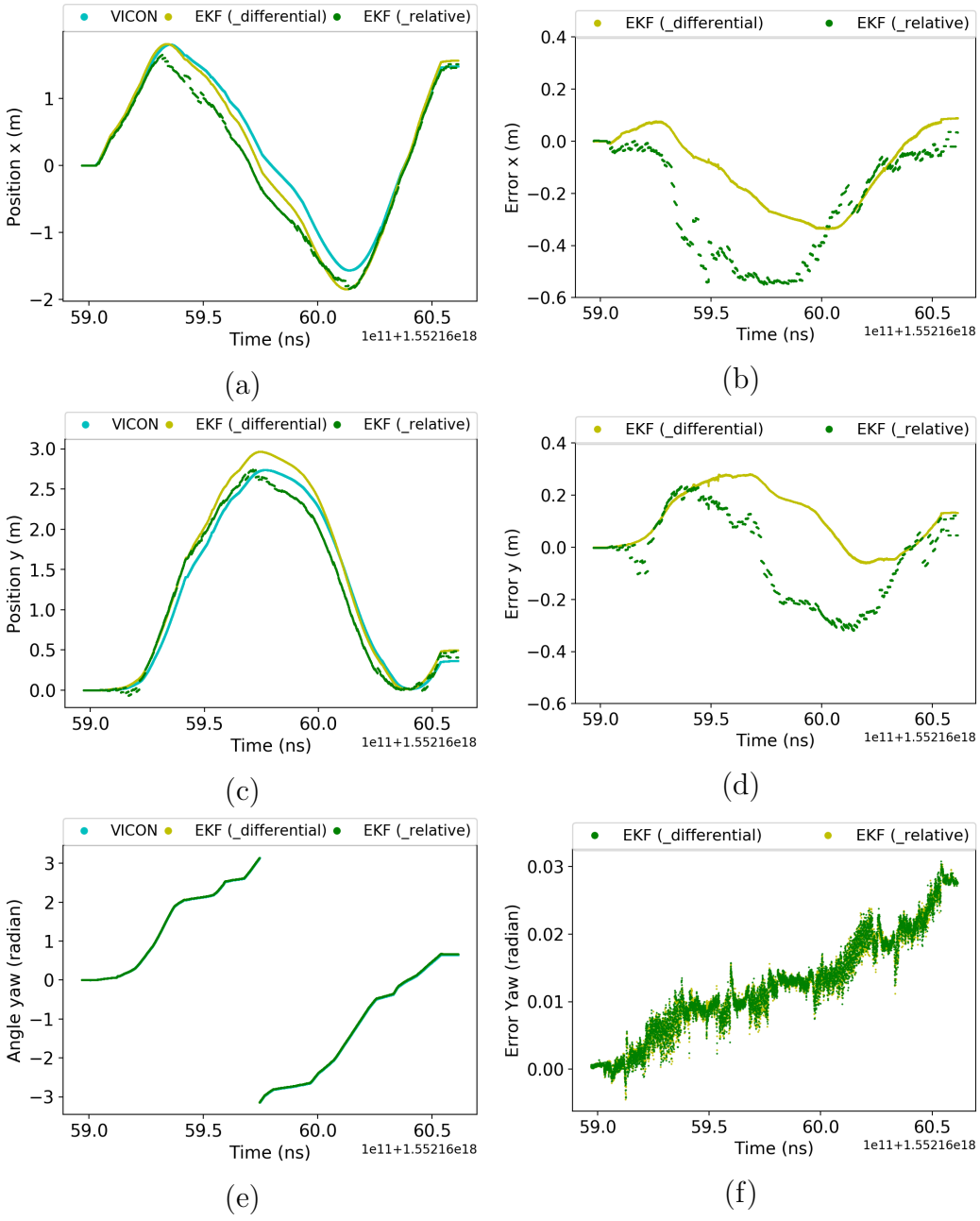



Figure A.2: Pose estimates from EKF fusion results compared with VICON and corresponding error plots of trial 2

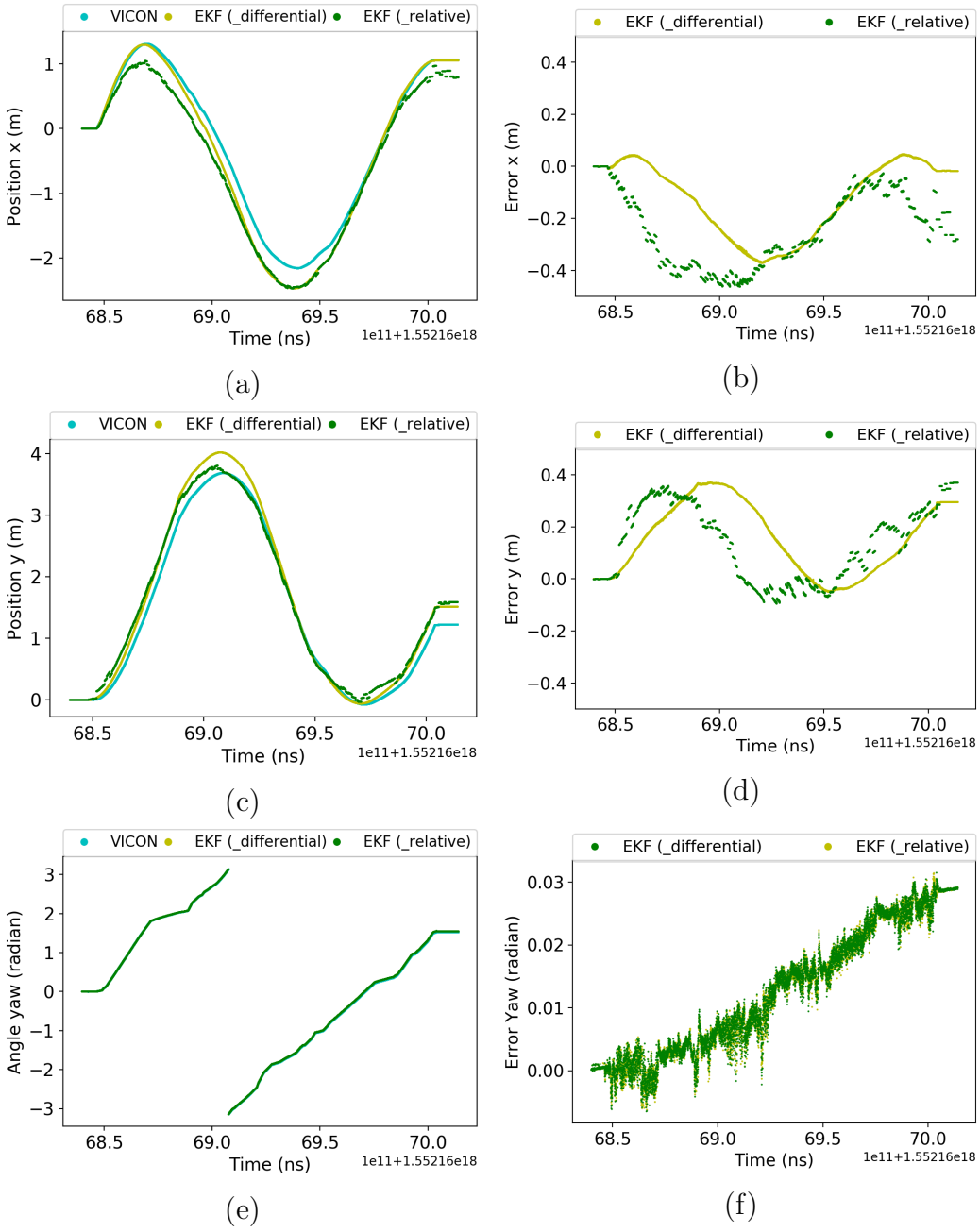



Figure A.3: Pose estimates from EKF fusion results compared with VICON and corresponding error plots of trial 3

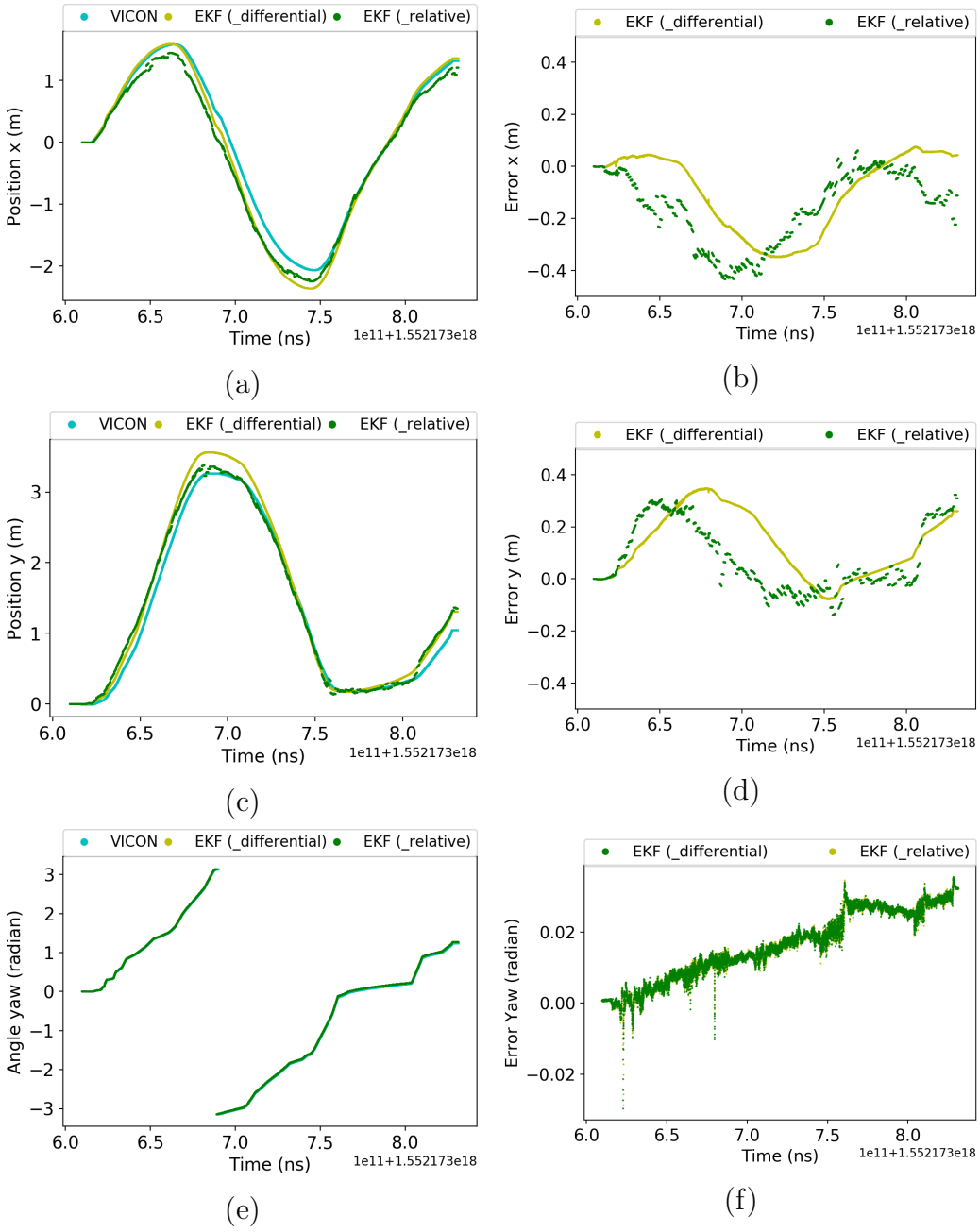



Figure A.4: Pose estimates from EKF fusion results compared with VICON and corresponding error plots of trial 4

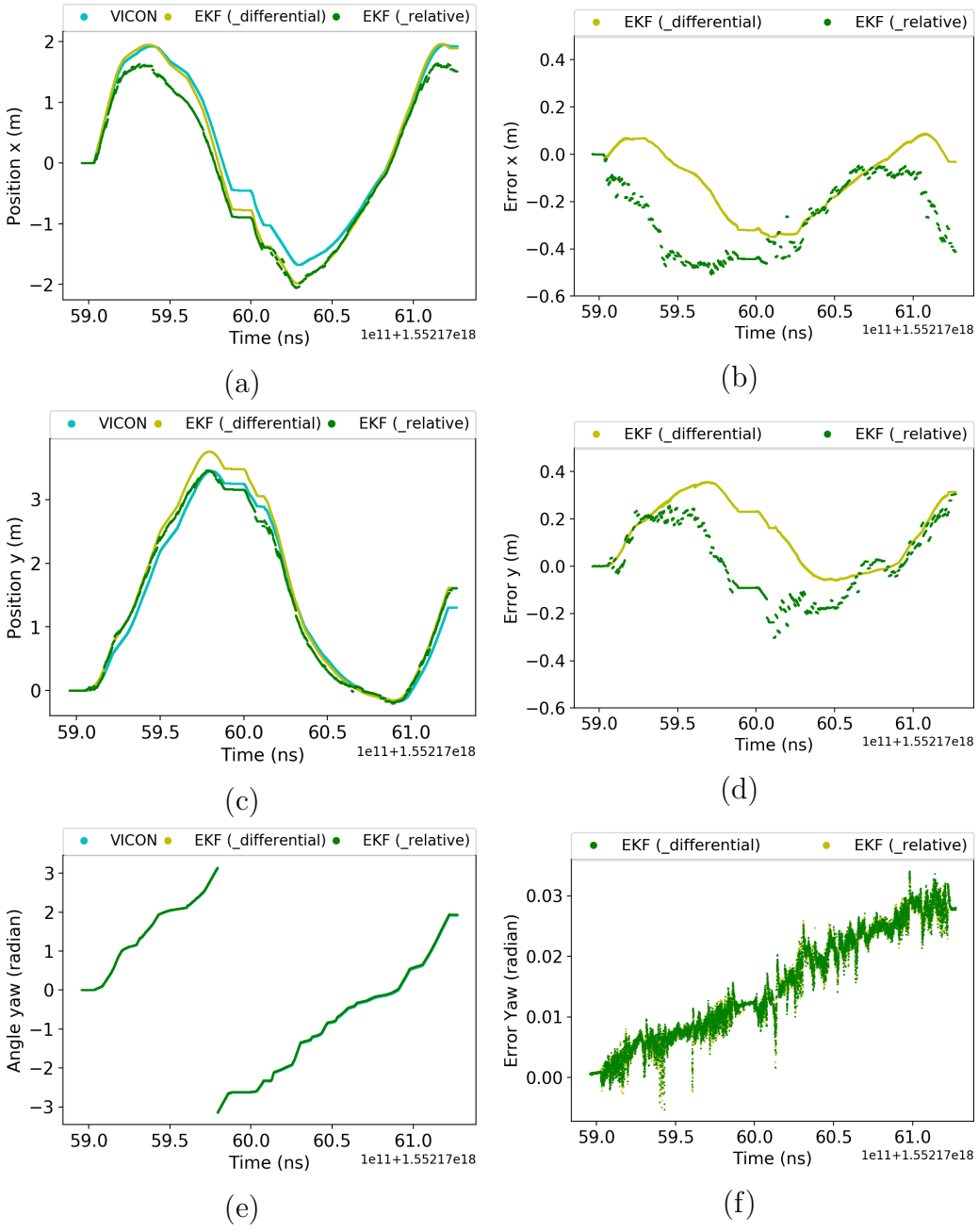



Figure A.5: Pose estimates from EKF fusion results compared with VICON and corresponding error plots of trial 5

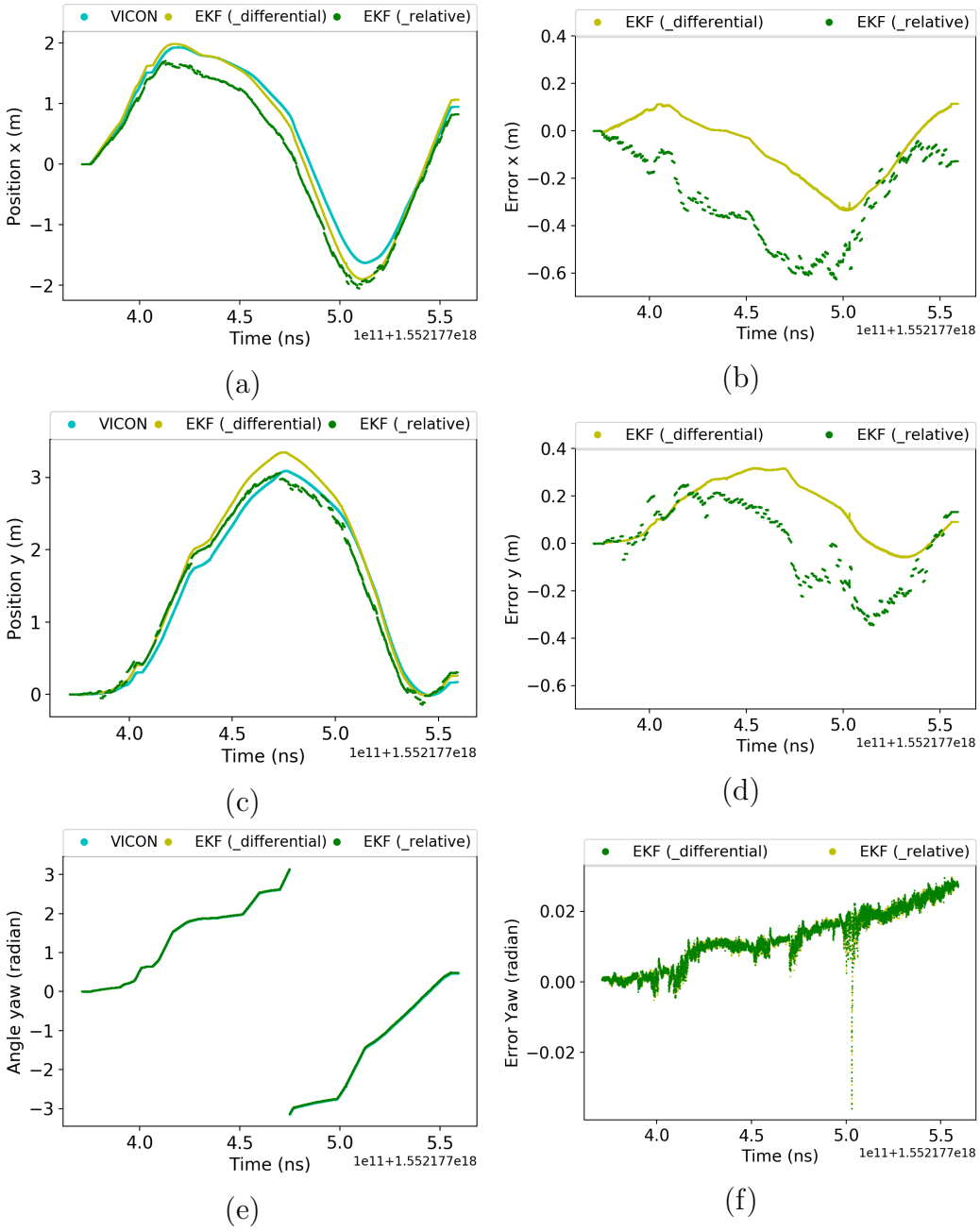



Figure A.6: Pose estimates from EKF fusion results compared with VICON and corresponding error plots of trial 6