Nikola Georgiev

Investigation of Pose Estimation From Sparse Inertial Data

Computer Science Tripos – Part II

Clare College

September 22, 2022

Declaration

I, Nikola Georgiev of Clare College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed **Nikola Georgiev**

Date September 22, 2022

Acknowledgements

I would like to thank my supervisor Andrea Ferlini for his continued support throughout the duration of the project, Andreas Grammenos and Manon Kok for providing highly valuable advice on problems regarding sensor calibration and fusion, as well as the members of the Cyber-Human Lab for access to the OptiTrack motion capture system.

Proforma

Candidate Number:	**2425D**
Project Title:	**Investigation of Pose Estimation From Sparse Inertial Data**
Examination:	**Computer Science Tripos – Part II, June 2022**
Code Line Count:	**2511**[1]
Word Count:	**8526**[2]
Project Originator:	Dissertation Author
Supervisor:	Andrea Ferlini

Original Aims of the Project

The aim of the project is to use sensor fusion algorithms for pose estimation using inertial measurements.

Work Completed

A fixed-lag smoothing and filtering algorithm based on a probabilistic model is designed and used for pose estimation.

Special Difficulties

None

[1]Code line count calculated using VSCodeCounter editor extension.
[2]Word count was calculated through the TeXcount web service.

Contents

List of Figures

Chapter 1

Introduction

The goal of this dissertation is to develop sensor fusion methods for automatic calibration and pose estimation. To that end I present algorithms for non-linear smoothing and filtering are applied on IMU data.

1.1 Motivation

Pose estimation using inertial sensors was crucial to many engineering achievements in the 20th century, such as the Apollo space missions. Over the past two decades, the increased miniaturization and affordability of micro-electromechanical systems (MEMS) has led to their use in a wider variety of applications, e.g. motion capture systems and wearable trackers. Motion capture has uses in entertainment to assist animation, sports to study player performance, and in health to detect ailments through gait analysis. Working with inertial measurements is challenging however due to noise and environmental disturbances.

Recent hype around the metaverse has birthed numerous virtual and augmented reality projects with various forms of sensing and tracking, e.g. using electromagnetic sensors [27] or rapidly spinning IR beacons [37]. Popular VR headsets like the Meta Quest 2 [33] use a combination of optical, infrared and inertial sensors for head- and hand-tracking, which can be extended to upper-body pose estimation through inverse kinematics [10]. Alternatively, human pose estimation from webcam video has made full-body tracking accessible to content creators and indie game developers, but is limited to the field of view of the optical sensor and struggles with occlusion [46]. Inertial sensors naturally do not have such issues, as they do not require line of sight. In the results section, the application of sparse inertial sensors on motion capture is explored. More generally, there has been a rise in spatial AI research – designing machine learning models to model spatio-temporal relationships in multi-modal data [12].

1.2 Related Work

Prior research on the topic has used an assortment of optimization, Bayesian inference, and deep learning methods.

Numerous works have used probabilistic models designed for inertial data [41, 28, 21] and applied a variety of filtering and smoothing techniques, some of which are discussed in section 3.2. Furthermore, the simultaneous localization and mapping (SLAM) problem from robotics is closely related. In particular, work by Dellaert and Kaess on robot perception using factor graphs [13] was a large influence on the way pose estimation is approached here. I show how their multi-frontal factorization method is a specific case of a more general algorithm, which enables me to create a fixed-lag smoother in section 3.3.

Deep learning has seen a boom in applications over the past decade and work in [23, 45, 27] has used recurrent neural networks and the SMPL 3D human body model as a prior to kinematically constrain human pose for estimation.

Chapter 2

Preparation

In this section I define the pose estimation problem and build up the mathematical background needed to implement the algorithms presented in the next section.

2.1 Problem Description

A pose is defined as the combination of position $p \in \mathbb{R}^3$ (m) and orientation R in a chosen coordinate frame. In the case of human pose, the skeleton is often modeled as a kinematic chain – a group of rigid bodies connected by joints. With this model the pose can be completely represented by joint orientations, with positions calculated using forward kinematics.

The inertial sensors used in this dissertation are the accelerometer, gyroscope and magnetometer. Inertial measurement units (IMUs) combine these sensors into a single electronic device. The accelerometer records specific force $s \in \mathbb{R}^3$ (m/s^2), which is zero during free fall and opposite gravitational acceleration g when stationary. The gyroscope measures angular velocity $\omega \in \mathbb{R}^3$ (rad/s) and the magnetometer returns a direction and strength vector $m \in \mathbb{R}^3$ (μT) of the surrounding magnetic field. It is assumed to point in the direction n of magnetic North.

While working in three dimensions, it is useful to define the relevant reference frames. Let e be the reference frame of the Earth. Then define the navigation frame n to approximate a small area of the Earth's surface as a plane. It aligns its normal vector with the direction opposite gravitational acceleration. Moreover, it is assumed that the n frame is not located near the magnetic poles, so the magnetic North vector has a consistent angular displacement, known as the dip angle δ, from the ground plane. This angle may change depending on the presence of ferromagnetic materials in the environment. Finally, define the body frame b as the IMU reference frame from which we sample all the data, i.e. the b frame and sensor frames

are aligned. Using rotations we can convert between them, e.g. a vector in the body frame v^b can be transformed to the navigation frame with $R^{bn}v^n = v^b$. The superscript notation helps us keep track of the current frame of a term.

Hence, the problem of inertial sensor fusion for pose estimation can be defined as follows: given a time series of IMU data $\gamma_t = (s^b, \omega^b, m^b)_t$, return a series of corresponding poses $(p^n, R^{bn})_t$. It is possible to determine the absolute orientation in the navigation frame, but the accelerometer readings from the body frame can only provide position relative to a starting point rather than the absolute value. Hence, we have to assume a known initial position or start at the origin of the coordinate system and apply an offset when necessary.

2.2 Orientation Parametrization

While the position and IMU data are vectors in $\mathbb{R}^3$, there are numerous parametrizations of orientation in three-dimensions. Due to Euler's Rotation Theorem [5], the most compact parametrization around a fixed point is the angle-axis form. Rotation vector $\varphi = \alpha n$ is a rotation by an angle $\alpha = \|\varphi\|$ around a unit axis $n = \varphi/\alpha$. The angle-axis form is not unique; when the magnitudes of rotation vectors are a multiple of 2π apart, the overall rotations are indistinguishable.

Another three-vector parametrization is using Euler angles (φ, θ, ψ), where each element is the angle amount to rotate around a pre-defined axis. For example, a XYZ Euler angle representation reduces to a chain of three consecutive angle-axis rotations around the x, y, z-axes. Unfortunately, since these are separate rotations, Euler angles are prone to gimbal lock – a loss in one degree of freedom when two of the axes become parallel. They are therefore impractical for our application.

The unit quaternion form does not have singularities and adds redundancy in a fourth value. A quaternion is defined as $q = [v^\top \quad w]^\top$ where v is a three-vector and w a scalar. A rotation vector n converts to q using $v = n\sin(\alpha)$ and $w = \cos(\alpha)$. Unit quaternions are not unique since $q = -q$. Quaternions are composed using a Hamilton product but matrix isomorphisms exist that enable composition through matrix multiplication [22].

A third parametrization is by rotation matrices – 3×3 matrices that satisfy $R^{-1} = R^\top$. These are the elements of the special orthogonal group SO(3) (also called the 3D rotation group), where each rotation transforms an orthonormal basis to another [5]. When applied to vectors, $Ru = [x \quad y \quad z]^\top u$ rotates vector u by projecting it onto its axes during multiplication. Rotation matrices are arguably a more natural result from rotation vectors in 3D than unit quaternions [3].

SO(3) is a matrix Lie group so it has a corresponding Lie algebra $\mathfrak{so}(3)$ that linearly approximates its local structure at the identity. The elements of $\mathfrak{so}(3)$ are skew-symmetric matrices. The hat operator $[\cdot]^\wedge : \mathbb{R}^3 \to \mathfrak{so}(3)$ defined in Equation 2.1 converts a vector to skew-symmetric form. We use Einstein notation, i.e. subscripts represent the indices of each tensor and repeated indices are summed over.

$$\begin{aligned}[x]^\wedge_{ij} &= \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}_{ij} \\ &= -\varepsilon_{ijk} x_k \end{aligned} \tag{2.1}$$

To convert back from skew-symmetric form we use $x_k = -\frac{1}{2}\varepsilon_{ijk}[x]^\wedge_{ij}$. In three-dimensions, the Levi-Civita symbol ε is 1 when its indices are a cyclic permutation of $[1, 2, 3]$, -1 for an acyclic permutation, and 0 otherwise [20].

The SO(3) exponential map $\exp(\cdot) : \mathfrak{so}(3) \to \mathrm{SO}(3)$ coincides with the matrix exponential, so we can define function $\mathrm{Rot}(\cdot) : \mathbb{R}^3 \to \mathrm{SO}(3)$ which maps rotation vector φ to a rotation matrix $\mathrm{Rot}(\varphi)$. This conversion is known as the Rodrigues rotation formula:

$$\begin{aligned}\mathrm{Rot}(\varphi) &= \exp([\varphi]^\wedge) = \exp(\alpha[n]^\wedge) \\ &= I + \left(\alpha - \frac{\alpha^3}{3!}\dots\right)[n]^\wedge + \left(\frac{\alpha^2}{2!} - \frac{\alpha^4}{4!}\dots\right)[n]^\wedge[n]^\wedge \\ &= I + (\sin\alpha)[n]^\wedge + (1-\cos\alpha)[n]^\wedge[n]^\wedge \\ &= I + \frac{\sin\alpha}{\alpha}[\varphi]^\wedge + \frac{(1-\cos\alpha)}{\alpha^2}[\varphi]^\wedge[\varphi]^\wedge \end{aligned} \tag{2.2}$$

Note that $\mathrm{Rot}(\varphi)^\top = \mathrm{Rot}(-\varphi)$. The exponential map has a corresponding logarithm which we can use in $\mathrm{Rot}^{-1}(\cdot)$ to convert a rotation matrix to a rotation vector. We observe that the first and last terms of the Rodrigues formula are symmetric, while the second term is not and can therefore be isolated. Given non-identity rotation $R = \mathrm{Rot}(\varphi)$:

$$\mathrm{Rot}^{-1}(R) = -\frac{\varepsilon}{2}\log(R) = -\frac{\varepsilon}{2}\left[\frac{\alpha}{2\sin\alpha}\left(R - R^\top\right)\right] \tag{2.3}$$

To calculate the rotation angle α, we use that the trace of the rotation is the sum of the traces of the first and last terms. Moreover, the trace of the third term is a multiple of the

angle squared. The Kronecker delta δ below effectively renames an index.

$$\begin{aligned}
[\varphi]^\wedge_{ij}[\varphi]^\wedge_{jk} &= \varepsilon_{ijp}\varphi_p\varepsilon_{jkq}\varphi_q = (\delta_{pk}\delta_{iq} - \delta_{pq}\delta_{ik})\varphi_p\varphi_q \\
&= \varphi_p\varphi_q - \varphi_q\varphi_q = \varphi^2 - \alpha^2 \\
\mathrm{Tr}(R) &= \mathrm{Tr}(I) + \frac{(1-\cos\alpha)}{\alpha^2}\mathrm{Tr}(\varphi^2 - \alpha^2) \\
&= 3 + (1-\cos\alpha)(-2) \\
\alpha &= \arccos\left(\frac{\mathrm{Tr}(R)-1}{2}\right)
\end{aligned} \tag{2.4}$$

Due to the non-uniqueness of rotation vectors, the Rodrigues rotation formula is surjective, i.e. each rotation matrix $\mathrm{Rot}(\varphi)$ could be generated by rotation vectors with magnitudes $\alpha+2\pi k$ for $k \in \mathbb{Z}_{\geq 0}$. Consequently, we limit the rotation angle to $(-\pi, \pi)$.

The Lie algebra $\mathfrak{so}(3)$ is a vector space tangent to the identity rotation, which is evident in the first-order approximation of the SO(3) exponential map: $\exp([\varphi]^\wedge) \approx I + [\varphi]^\wedge$. With this linearization we can calculate the derivative of a rotation matrix near the identity.

$$\frac{\partial \mathrm{Rot}(\varphi)_{jk}}{\partial \varphi_m} \approx \frac{\partial \left(I + [\varphi_i]^\wedge\right)_{jk}}{\partial \varphi_m} = -\frac{\partial \varepsilon_{ijk}\varphi_i}{\partial \varphi_m} = -\varepsilon_{ijk}\delta_{mi} = -\varepsilon_{mjk} \tag{2.5}$$

The linearization accurately approximates the change in rotation assuming $\varphi \to 0$. Under the small-angle assumption, we can do the same for the inverse using $\lim_{\alpha\to\infty} \frac{\sin\alpha}{\alpha} = 1$.

$$\begin{aligned}
\frac{\partial [\mathrm{Rot}^{-1}(R)]_k}{\partial R_{mn}} &\approx -\frac{\varepsilon_{ijk}}{4}\frac{\partial R_{ij} - R_{ji}}{\partial R_{mn}} = -\frac{\varepsilon_{ijk}}{4}(\delta_{mi}\delta_{nj} - \delta_{mj}\delta_{ni}) \\
&= -\frac{\varepsilon_{ijk}}{4}\varepsilon_{ijl}\varepsilon_{mnl} = -\frac{2\delta_{kl}}{4}\varepsilon_{mnl} = -\frac{\varepsilon_{mnk}}{2}
\end{aligned} \tag{2.6}$$

Applying the chain rule on Equation 2.5 and Equation 2.6 shows that the derivative of a rotation vector obtained by applying a rotation deviation η to rotation matrix M, is approximated by index renaming.

$$\frac{\partial [\mathrm{Rot}^{-1}(\mathrm{Rot}(\eta)M)]_i}{\partial \eta_m} = \frac{\partial [\mathrm{Rot}^{-1}(R)]_i}{\partial R_{pr}}\frac{\partial \mathrm{Rot}(\eta)_{pq}M_{qr}}{\partial \eta_m} \approx \tfrac{1}{2}\left[\varepsilon_{pri}\varepsilon_{mpq}M_{qr}\right]_{mi} \tag{2.7}$$

2.3 Dead Reckoning

Pose estimation using kinematic modeling, also known as dead reckoning in inertial navigation systems, assumes an ideal accelerometer and gyroscope pair. In Equation 2.8 the pose and IMU measurements are treated as functions of time. The change of orientation can be calculated by integrating the rotations $\Omega^b(t)$ over time. The change in position is calculated through double integration; integrating linear acceleration gives velocity and integrating linear velocity gives change in position. As the accelerometer readings are of the specific force rather than linear acceleration, the gravity vector is rotated to the body frame and deducted from the measurements.

$$\begin{aligned} R^{bn}(t) &= \left(\int_0^t \Omega^b(x)\, dx \right) R^{bn}(0) \\ v^b(t) &= v^b(0) + \int_0^t s^b(x) - R^{bn}(x) g^n \, dx \\ p^b(t) &= p^b(0) + \int_0^t v^b(x)\, dx \end{aligned} \tag{2.8}$$

In practical applications the sensors are sampled, so a numerical integration method is required [43]. In Equation 2.9 we use the Euler method for approximate integration [21]. Since the intervals between samples are not precisely constant, a parameter T stores interval lengths.

$$\begin{aligned} R^{bn}_{t+1} &= \mathrm{Rot}(T_t w^b_t)^\top R^{bn}_t \\ a^b_t &= s^b_t - R^{bn}_t g^n \\ v^b_{t+1} &= v^b_t + T_t a^b_t \\ p^b_{t+1} &= p^b_t + T_t v^b_t + \frac{1}{2} T_t^2 a^b_t \end{aligned} \tag{2.9}$$

Furthermore, the IMU sensor measurements contain error from a variety of sources, which accumulates and results in significant inaccuracy over time. This is known as integration drift [28]. Figure 2.1 shows the orientation error obtained from the stationary measurements of a gyroscope used throughout the project. An ideal gyroscope would record zero rotation outside of some environmental vibrations, but data from the real IMU used in this project results in about 25° error over 20 seconds. Due to double integration and orientation error, the estimated position exhibits a rapidly increasing quadratic error. Hence, beyond the simple integration we need algorithms to calibrate and fuse sensor data.

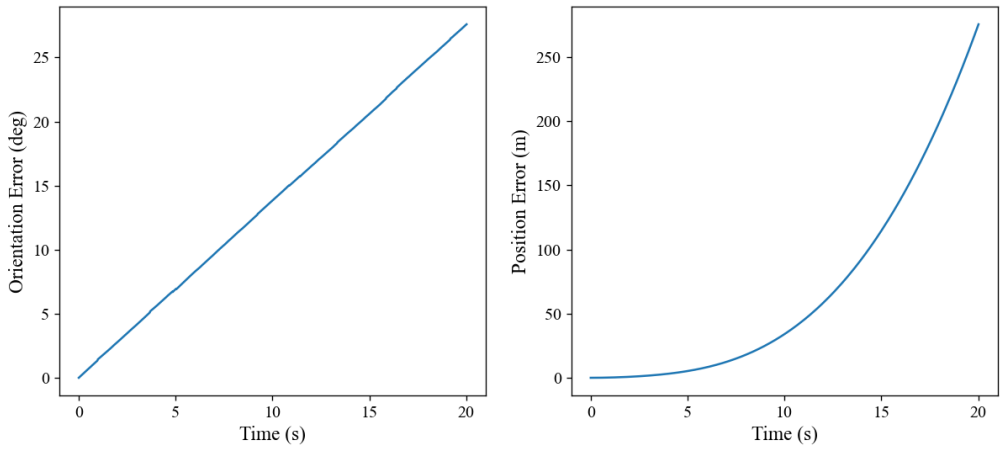

Figure 2.1: Dead Reckoning Error of Stationary, Uncalibrated IMU

2.4 Least Squares Fitting

Among the IMU calibration and pose estimation algorithms presented, several are models whose parameters we fit to data. Suppose we have a data set (x, y) and model $f(x, \beta)$ with parameters β. The residual r is the difference between the model prediction and observed data y. By minimizing an objective function that is the sum of squared residuals for each (x_i, y_i) pair, we obtain parameters $\hat{\beta}$ fitting our model f to the data. This is called the least squares problem.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i r_i^2 = \underset{\beta}{\operatorname{argmin}} \sum_i (y_i - f(x_i, \beta))^2 \tag{2.10}$$

In *ordinary least squares* we use a model that is linear in its parameters, i.e. $f(X, \beta) = X\beta$. We can minimize the total squared residuals with the first derivative.

$$\nabla \|y - X\beta\|^2 \big|_{\hat{\beta}} = -2X^\top (y - X\beta)\big|_{\hat{\beta}} = 0 \quad \Longrightarrow \quad \hat{\beta} = (X^\top X)^{-1} X^\top y \tag{2.11}$$

To fit a model that is non-linear in its parameters, a different technique is required. The condition $\nabla \|r\|^2 \big|_{\hat{\beta}} = 0$ should still hold for minima of the objective function [9]. Hence, we can initialize with hypothesized parameters and iteratively adjust them. On the kth iteration, let $\hat{r}^{(k)}$ be the linearization of residual r around an estimate of the parameters $\beta^{(k)}$ and denote the Jacobian as $J^{(k)} = \nabla r\big|_{\beta^{(k)}}$.

$$\hat{r}^{(k)}(X, \beta) = r(X, \beta^{(k)}) + (J^{(k)})^\top (\beta - \beta^{(k)}) \tag{2.12}$$

Based on the assumption that the sum of squared residuals is locally quadratic, the above linearization enables us to solve each iteration as an ordinary least squares problem. Thus, we derive the update step $\beta^{(k+1)}$ of the *Gauss-Newton algorithm*.

$$\begin{aligned} \nabla \left\| \hat{r}^{(k)} \right\|^2 \Big|_{\beta^{(k+1)}} &= 2J^{(k)}\hat{r}^{(k)}\Big|_{\beta^{(k+1)}} = 0 \\ \implies \beta^{(k+1)} &= \beta^{(k)} - (J^{(k)}J^{(k)\top})^{-1}J^{(k)}r(X, \beta^{(k)}) \end{aligned} \tag{2.13}$$

The *Levenberg-Marquardt algorithm* adds a distrust hyperparameter λ that regularizes the Gauss-Newton update step. With each iteration that successfully decreases the sum of squared residuals, the distrust is decreased, and we increase it on failed updates [16].

$$\beta^{(k+1)} = \beta^{(k)} - (J^{(k)}J^{(k)\top} + I\lambda)^{-1}J^{(k)}\hat{r}^{(k)} \tag{2.14}$$

When the distrust hyperparameter is large, the regularizing term forces a step in the direction of steepest descent. On the other hand, when λ is small, the step is the same as in the Gauss-Newton algorithm. In theory, as the model nears a local minimum, λ will continuously decrease and the Gauss-Newton step will accelerate the model towards the minimum.

2.5 Data Sets and Collection

2.5.1 Arduino Trackers

To apply the algorithms presented to a real-world context, we use data from the LSM9DS1 modules on six Arduino Nano 33 BLE Sense boards. The LSM9DS1 is a 9-axis IMU combining an accelerometer, gyroscope and magnetometer.

IMU Configuration

The `Arduino_LSM9DS1.h` library [4] and module datasheet [25] are used to set control registers to values corresponding to desired sensor properties. The measurement range is a bound on the physical quantities a sensor can measure while satisfying the technical specifications. The output data rate (ODR), or sample rate, describes the frequency with which the sensor converts the analog signal to digital format. The datasheet also lists a typical resolution value at certain ranges, but this depends on how electrical noise is handled and without this information it is not useful [39]. Specifically, the LSM9DS1 has integrated lowpass filtering that can be used to remove noise above a chosen cutoff frequency, which is upper-bounded by the Nyquist frequency $\frac{\text{ODR}}{2}$. Highpass filtering is also supported, but can be bypassed.

Register	Value	Range (±)	ODR (Hz)	Cutoff (Hz)
`CTRL_REG6_XL` (accelerometer)	`0xD7`	$4g$	952	105
`CTRL_REG1_G` (gyroscope)	`0xD8`	2000 dps	952	100
`CTRL_REG1_M` (magnetometer)	`0xFE`	4 gauss	155	—

Table 2.1: Selected LSM9DS1 IMU configuration

The IMU configuration limits the kinds of motion the IMU can estimate the poses for. Focusing on human pose estimation, we can create assumptions based on human motion to justify the chosen values shown in Table 2.1. For example, an acceleration of $4g$ would displace an object by ≈ 20 meters in one second, which would be extremely rapid for a human. The elbows of baseball pitchers have been observed to reach angular velocities of 2434 ± 552 dps [34], which maxes out the largest range of our gyroscope. Nevertheless, the extreme nature of these velocities suggests 2000 dps could suffice for more ordinary examples of human motion. Finally, we also assume that the data is recorded in an environment where the main magnetic force is Earth's magnetic field, whose strength is $\approx 60\mu T = 0.6$ gauss. If the signals are biomechanical, studies like [11] suggest a lowpass filter cutoff of 10 Hz. We set our cutoff frequencies to an order of magnitude greater (see Table 2.1), which enables further external filtering if necessary. The magnetometer automatically performs averaging in its ultra-high performance mode [31], which works as a lowpass filter and results in an effective ODR of 155 Hz.

Note that the accelerometer and gyroscope readings use a left-handed coordinate system, while the magnetometer data is in a right-handed system with an inverted x-axis. Here we convert them all to a right-handed system.

BLE Communication

In applications of pose estimation, like motion capture, wired solutions can be intrusive and inhibit full range of motion. Fortunately, the Arduino Nano 33 BLE Sense is equipped with Bluetooth Low Energy (BLE) capability. Using a PC with a Bluetooth adapter, we can establish a connection with the Arduinos and record IMU samples. The BLE protocol has several layers for operation and it is particularly valuable to understand the GAP and GATT layers [17].

The Generic Access Profile (GAP) specifies how BLE devices should discover and connect with each other. Devices with the Central role can initiate and maintain connections with multiple devices. Devices given the role of Peripheral advertise their presence to be discovered by Centrals and generally have fewer computational resources.

The Generic Attribute Profile (GATT) is a stateless client/server protocol that uses attributes to define the rules for formatting and transmitting data. The attributes are located on a GATT Server, which responds to requests from GATT Clients. The attributes are organized in a hierarchy analogous to an object-oriented class structure. Services (classes) group related attributes in characteristics (member variables), which in turn contain an attribute for their type, another for their value, and descriptors such as the procedures that can use them.

There is no limitation on how devices are assigned GAP and GATT roles, as the layers are independent. In this use case, data stored on multiple Arduinos must be sent to a central computer. Hence, we run a GAP Central and GATT Client on the PC, treating the Arduinos as GAP Peripherals and GATT Servers. We define a single service and two characteristics within it – one that notifies the PC of new data, and another, which the PC can write to, containing the target notification rate. A loop running on each Arduino board continuously polls the number of microseconds since the program was initialized to sample the IMU and notify the GATT Client in intervals defined by the PC. We have to take into account that the `micros()` function on Arduino overflows in ≈ 72 minutes because its data type is `unsigned long`.

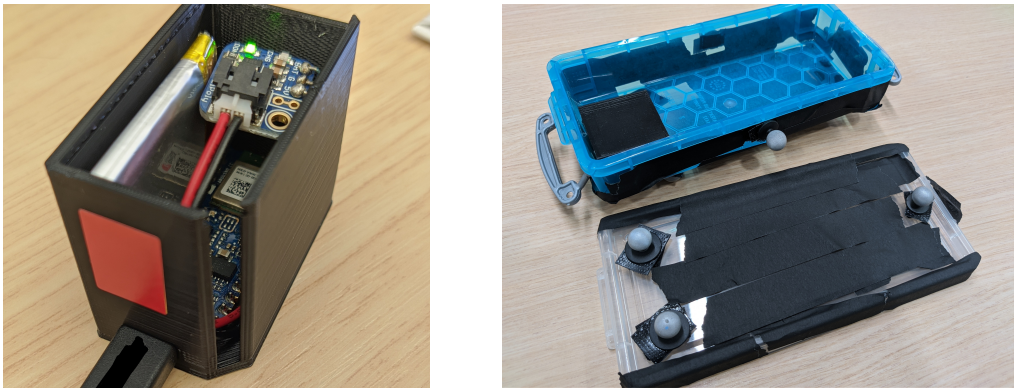

Figure 2.2: Portable case and box used in motion capture

Portable Enclosure

In addition to wireless recording of data, I designed and 3D printed an enclosure for the Arduinos and LiPo batteries using Fusion360. The LiPo batteries were connected by soldering an AdaFruit LiPoly backpack [1] to each Arduino. The connected pins are listed in Figure 2.3. While connected with micro USB to a power source, the Arduino charges the connected LiPo battery. Disconnecting the power source switches

Arduino	**Backpack**
5V	5V
GND	G
VIN	BAT

Figure 2.3: Pins

to battery power.

2.5.2 Generating Ground Truth

An OptiTrack optical motion capture system is used to collect accurate pose data of the portable enclosure created for the Arduinos. A rigid body object is created in the Motive software and the pose is exported as `csv` and processed using `pandas`.

I supplement the Arduino board readings with synthetic data. The MoVi dataset [18] is a large collection of recordings of human motion using a variety of sensors. In particular I use pose output from their 17 IMU body suit to generate synthetic IMU readings based on the sensor model described in section 3.1.

2.6 Software Methods and Tools

2.6.1 Software Engineering Methodology

I followed an Agile development style with bi-weekly meetings with my supervisor to discuss progress and next steps. Unit tests are run on the core modules of the package. This ensures that the components work as intended before they are integrated with other code. Synthetic data is generated using the formulas for dead reckoning as a way of checking that the algorithms work as intended on toy examples.

2.6.2 Requirements Analysis

Component	**Priority**	**Difficulty**	**Risk**
Rotations Module	high	medium	high
Kinematic Models	high	low	high
Non-Linear Optimization Method	high	medium	high
Calibration Procedure	medium	high	medium
Multi-Frontal QR	high	high	high
Pose Estimation	medium	high	medium
Arduinos and BLE Communication	medium	medium	low
Evaluation	medium	medium	low

Table 2.2: Requirements with specified priority, difficulty and risk (its impact on project implementation if poorly designed).

I chose to write my own module for rotations as I wanted full control over the methods used to work with rotation matrices and quaternions. This proved useful when implementing orientation estimation but came with a high risk, since many of the other components depended on working 3D rotation formulae. The kinematic models were used for dead reckoning and generation of synthetic data in testing. The non-linear optimization and multi-frontal QR algorithms modules were used for calibration and pose estimation, respectively.

2.6.3 Starting Point

From the IB Data Science course I had some experience manipulating vectors and matrices using Python `Numpy`, as well as plotting data using `Matplotlib`. From the Algorithms and Probability courses I had also had an introduction to algorithms on graph data structures and Bayesian statistics, respectively. I had also found papers describing a variety of approaches to IMU-based pose estimation and gained a general idea as to how their methods worked.

2.6.4 Choice of Tools

The calibration procedures and probabilistic models are written in Python using `PyTorch` for linear algebra and tensor operations. This allowed fast prototyping in a `Jupyter` notebook and abstracted the complexity of certain mathematical operations. For testing I used `unittest` and `matplotlib` for plotting. To discover, connect and receive notifications with BLE I used the `bleak` platform agnostic Python GATT Client [8].

Code uploaded to the Arduino boards is written in C++ within the Arduino IDE and uses libraries provided by Arduino for accessing readings from each IMU sensor as well as `ArduinoBLE` to enable Bluetooth Low-Energy communication.

The dissertation is written in LaTeX using the Overleaf online editor. Both source code and dissertation were backed up regularly on my personal GitHub account and bi-weekly on an external solid-state drive.

Chapter 3

Implementation

3.1 IMU Calibration

In this section I develop a calibration procedure for a 9-axis IMU and present sensor fusion algorithms based on probabilistic modelling. The resulting algorithm is a fixed-lag smoother that generalized the Kalman filter algorithm, which is applied to the pose estimation problem.

3.1.1 Sensor Error Model

As illustrated in section 2.3, applying the dead reckoning equations on raw IMU readings is unfruitful. Nonetheless, by analyzing the types of error that affect the sensors, we can design models to reduce it. Based on literature [19] [43], we categorize the errors into three types:

Constant bias	Modelled with a measurement offset b. It results in linear and quadratic error after single and double integration, respectively.
Misalignment and scaling	Represented by a square matrix S. If its off-diagonal elements are zero, then it is purely a scalar matrix. It can additionally include misalignment with the body frame of the IMU.
Thermo-mechanical white noise	Modelled with multivariate Gaussian variable $\epsilon \sim \mathcal{N}(0, \Sigma)$.

For example, magnetometer measurements in a controlled environment are affected by hard and soft iron distortions [21]. The hard iron effect is caused by magnetic disturbance within the sensor's frame, resulting in a fixed offset. The soft iron effect comes from the surrounding environment, resulting in scaling.

Over time, deterministic error terms S and b gradually change due to flicker noise in the sensor's electronic components and susceptibility to fluctuations in temperature, either environmental or through self-heating [43]. Hence, we linearly model a sensor's measurement $\gamma(x)$ with added Gaussian noise.

$$\gamma(x) = Sx + b + \epsilon \tag{3.1}$$

3.1.2 Ellipsoid Fitting

Ideally, a stationary IMU would read unit vectors from the accelerometer and magnetometer corresponding to the opposite of gravitational acceleration and the direction pointing to the magnetic North pole, respectively. Using this fact, we can devise a simple calibration procedure known as the 3-point tumble [40], where we align the IMU with each axis and use the measurements to calculate a bias and scalar matrix.

$$\begin{aligned} S &= \tfrac{1}{2}\left(\mathbf{E}[\gamma(x^+)] - \mathbf{E}[\gamma(x^-)]\right) \\ b &= \tfrac{1}{2}\left(\mathbf{E}[\gamma(x^+)] + \mathbf{E}[\gamma(x^-)]\right) \end{aligned} \tag{3.2}$$

However, aligning and recording data this way is quite tedious and assumes no cross-axis sensitivity (non-zero measurement in perpendicular axes) or misalignments. A more general solution observes that one can fit an ellipsoid to the data [7], as ellipsoids are obtained by deforming a unit sphere (the ideal measurement) with affine transformations. Hence, we define a residual:

$$\begin{aligned} r = m^\top m - 1 &= (S^{-1}(\gamma(m) - b))^\top S^{-1}(\gamma(m) - b) - 1 \\ &= (S^{-1}\gamma(m))^2 - 2((S^{-1})^2 b)\gamma(m) + (S^{-1}b)^2 - 1 \end{aligned} \tag{3.3}$$

$$\gamma(m)^\top X \gamma(m) + 2p\gamma(m) - 1 = 0 \quad \text{for} \quad X = \frac{(S^{-1})^2}{1 - (S^{-1}b)^2} \quad p = -\frac{(S^{-1})^2 b}{1 - (S^{-1}b)^2} \tag{3.4}$$

To compute X and p we create a vector from the concatenation of a vectorized X and p, which creates a linear system that can be solved by ordinary least squares estimation. With the result we compute the constant bias.

$$b = -X^{-1}p \tag{3.5}$$

Now we deduct b from the sensor samples and perform least squares again, on the second-degree terms only.

$$(\gamma(m) - b)^\top Y(\gamma(m) - b) - 1 = 0 \quad \text{where} \quad Y = (S^{-1})^\top S^{-1} \tag{3.6}$$

We can then find S^{-1}, e.g. through Cholesky decomposition. For a reliable estimate of S and b, data from each axis direction is required, since the residual function in Equation 3.4 is not exclusive to ellipsoids and fits other three-dimensional quadric surfaces, e.g. hyperboloids.

3.1.3 Sensor Alignment

Accelerometer

While the ellipsoid fit can make use of all magnetometer measurements during calibration in a controlled setting, the accelerometer reading is only expected to be a unit vector when the IMU body frame is stationary relative to the navigation frame and has no linear acceleration. To identify such static moments, we run a window over the data samples and record the covariance in each window, which can be interpreted as the error in the assumption that the samples are all the same and therefore stationary. The covariance Σ is equivalent to the affine matrix of an ellipsoid with volume $V = (\det \Sigma)^{\frac{1}{2}}$. We then hypothesize that any window with $V \leq z^3 V_{\min}$ is static, as it is within z standard deviations from the mean measurement in each axis. This is performed for both raw accelerometer samples and calibrated magnetometer data with $z = 3$, where a window with rotation or linear acceleration in any axis increases its covariance volume. Different window sizes can return different results, so we define a reasonable range of sizes, e.g. 0.3–0.5 seconds worth of samples, and keep the one that returns the most static moments.

After identifying static moments, we can use the ellipsoid fitting method to estimate $\hat{S}^{-1}$ and $\hat{b}$ from accelerometer samples. However, it is not guaranteed that the accelerometer and magnetometer coordinate systems are aligned. Therefore, we select the magnetometer reference frame to be the IMU body frame and use the dip angle δ, which is expected to stay constant within our trials, to remove any misalignment. We optimize a multi-objective problem with the residuals in Equation 3.7, to enforce a constant dip angle and estimate its value. For our assumptions to hold, it only uses data from moments when the IMU is presumed stationary.

$$\begin{aligned} e_g(s) &= \left\| S^{-1}(\gamma(s) - b) \right\|^2 - \|g\|^2 \\ e_\delta(s) &= \left[S^{-1}(\gamma(s) - b) \right] m - \cos\delta \end{aligned} \tag{3.7}$$

We apply the Levenberg-Marquardt algorithm with the Jacobians below.

$$\frac{\partial e_g(a)}{\partial S^{-1}_{ij}} = 2a_i(\gamma(a) - b)_j \qquad \frac{\partial e_g(a)}{\partial b_i} = -2S^{-1}_{ji}a_j \qquad \frac{\partial e_g(a)}{\partial \cos\delta} = 0 \tag{3.8}$$

$$\frac{\partial e_\delta(a)}{\partial S^{-1}_{ij}} = 2m_i(\gamma(a) - b)_j \qquad \frac{\partial e_\delta(a)}{\partial b_i} = -S^{-1}_{ji}m_j \qquad \frac{\partial e_\delta(a)}{\partial \cos\delta} = -1 \tag{3.9}$$

The state is initialized to $(\hat{S}^{-1}, \hat{b}, \cos\hat{\delta})$ based on the results of the ellipsoid fitting and the dip angle given by a World Magnetic Model ($\hat{\delta} \approx 67°$ in Cambridge, UK) [44]. The objectives can be assigned different weights depending on the uncertainty of our sample data, but should avoid emphasizing e_g too greatly, as the accelerometer fit might degenerate to one

of a very small scale, effectively collecting all data into a point, and unit bias. If the dip angle alignment works as intended, it should converge on a slightly rotated $\hat{S}$ matrix, assuming the accelerometer and magnetometer axes are not grossly misaligned.

Gyroscope

To calibrate the gyroscope, we postulate that a stationary IMU would read zero angular velocity, giving us a formula for estimating its constant bias using the identified static moments.

$$b \approx \mathbf{E}[\gamma(\omega_{\text{static}})] \quad \text{where} \quad \omega_{\text{static}} \approx 0 \tag{3.10}$$

To estimate the scalar and alignment matrix though, we have to observe rotations. Fortunately, the change in orientation of the calibrated magnetometer samples should match the angular velocity measurement at each timestep. This dynamic relationship is modelled as:

$$R_t^{nb} m_t^b = R_{t+1}^{nb} m_{t+1}^b = R_t^{nb} \text{Rot}(T_t w_t^b) m_{t+1}^b \tag{3.11}$$

Hence, we minimize objective e_ω^2, where the residual is the difference between each magnetometer sample m_{t+1} (except the first) and its estimated value from rotating the previous sample m_t by the gyroscope sample w_t over a T_t second interval.

$$e_\omega(\gamma(\omega_t)) = \text{Rot}(T_t \omega_t)^\top m_t - m_{t+1} \tag{3.12}$$

We use the Levenberg-Marquardt algorithm for optimization, and apply the approximation in Equation 2.6 to derive the Jacobian below for the t-th samples.

$$\begin{aligned} \frac{\partial e_\omega(a)_p}{\partial S_{ij}^{-1}} &= \frac{\partial A_{qp} m_q}{\partial A_{rs}} \frac{\partial \text{Rot}(\eta)_{rs}}{\partial \eta_x} \frac{\partial T S_{xy}^{-1} (\gamma(\omega) - b)_y}{\partial S_{ij}^{-1}} \\ &= \delta_{rq} \delta_{sp} m_q (-\varepsilon)_{xrs} T \delta_{ix} \delta_{jy} (\gamma(\omega) - b)_y = \\ &= m_q (-\varepsilon)_{iqp} T (\gamma(w) - b)_j = \\ &= T [m]_{pi}^\wedge (\gamma(\omega) - b)_j \end{aligned} \tag{3.13}$$

3.1.4 Calibration Procedure

Finally, we combine the methods described for a complete IMU calibration procedure. It is important to record a variety of orientations and rotations, particularly for the ellipsoid fit

and rotation alignments. To calibrate the IMUs used in this project, I performed four 90° clockwise rotations around the vertical axis for each face of the portable enclosure.

Algorithm 1 Sensor Model Fit

Input: IMU samples $[\gamma(s), \gamma(\omega), \gamma(m)]$, time deltas T, dip angle prior δ

$S_m^{-1}, b_m = \texttt{ellipsoid\_fit}(\gamma(m))$

$m = S_m^{-1}(\gamma(m) - b)$

static $= \texttt{static\_moments}(\gamma(s), m)$

$S_t^{-1}, b_t = \texttt{ellipsoid\_fit}(\gamma(s)[\text{static}])$

$S_s^{-1}, b_s = \texttt{dip\_aligner}(S_t^{-1}, b_t, \delta)$

$b_\omega = \texttt{mean}(\gamma(\omega)[\text{static}])$

$S_\omega^{-1} = \texttt{rotation\_aligner}(\gamma(\omega) - b_\omega, m, T)$

Output: $(S_s^{-1}, b_s), (S_\omega^{-1}, b_\omega), (S_m^{-1}, b_m)$

3.1.5 Basic Orientation Estimation

With the calibrated IMU, dead reckoning estimates are greatly improved. While in Figure 2.1 we witnessed double-digit orientation error growing linearly due to a large constant bias, after calibration it is $\approx 1°$ in 20s.

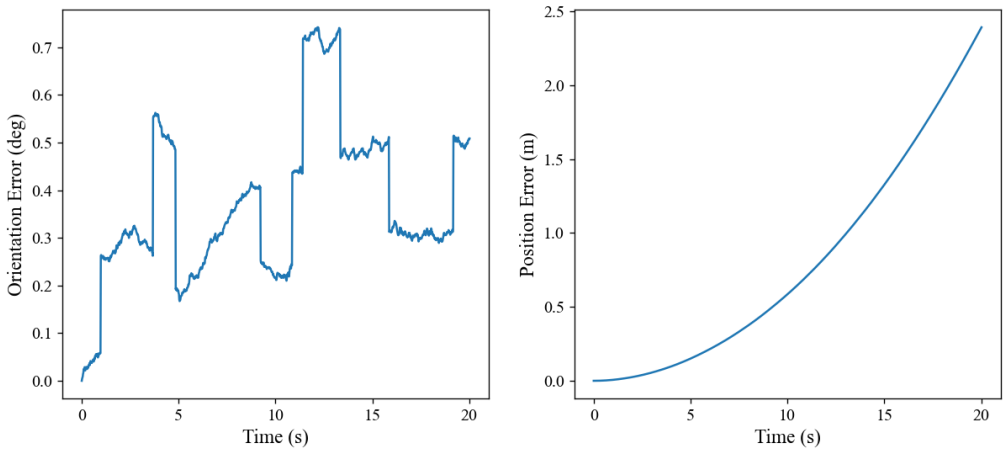

Figure 3.1: Dead Reckoning Error of Stationary, Calibrated IMU

A calibrated measurement $\hat{x}$ based on Equation 3.1 is the true value x with added Gaussian noise, which turns into a random walk after integration.

$$\hat{x} = x + (-S^{-1})\epsilon \tag{3.14}$$

To further improve the orientation estimation, we will need to use sensor fusion – a method of combining measurements to obtain information with lower uncertainty than if obtained from a single sensor. A preliminary option, is to employ a *complementary filter* [15], which combines noisy measurements of the same signal, each perturbed either by high frequency noise or low frequency disturbance [32]. Algorithm 2 is based on the implementation described in [36]. The dead reckoning estimation of orientation has a low frequency random walk error, while the magnetometer measurements exhibit high frequency noise but correct orientations long-term. Hence, we apply slight adjustments using the rotation between the estimated and measured North vectors. The amount of correction is tuned with scalar parameter α and, empirically, values ≈ 0.01 work well.

Algorithm 2 Complementary Filter for Orientation

Input: IMU data $[\omega^b, m^b]$, time deltas T, initial rotation R_0^{bn}, north vector n^n, parameter α

```
rots = List
rots.insert(R_0^bn)
for t ← 0 to count(T) do
    R_ω = matmul(transpose(Rot(T_t ω_t^b)), rots.last)
    n̂ = matmul(R_ω, n_t^n)
    v_n = rotation vector from n̂ to n_{t+1}^n
    R_{t+1}^bn = matmul(Rot(α v_n), R_ω)
    rots.insert(R_{t+1}^bn)
end for
```

Output: `rots`

3.2 Probabilistic Modelling

To develop more complex sensor fusion methods we use probabilistic modelling to take into account uncertainty. Our objective is to estimate the unknown state x using knowledge of sensor measurements γ, dynamic models, and measurement models. We assume the *Markov property* holds, i.e. given information on the present, the past and future are conditionally independent. This is evident in dead reckoning, as the current state is calculating using only the previous one. If we take measurements to be samples from random variables, we can combine the models into a joint probability density function (PDF) of the form $p(x|\gamma)$ and derive a filtering and a smoothing algorithm.

3.2.1 MAP Estimation

The Maximum a Posteriori (MAP) estimator is used to infer point estimates $\widetilde{x}$ maximizing a posterior probability. In our problem, we seek to compute x that maximizes $p(x|\gamma)$, effectively treating the state as parameters being fit to measurement data. Compared to maximum likelihood estimation, MAP includes state priors $p(x)$ through Bayes' rule. As the measurements are constant during estimation, $p(\gamma)$ is factored out into a normalization constant.

$$\widetilde{x} = \underset{x}{\operatorname{argmax}}\, p(x|\gamma) = \underset{x}{\operatorname{argmax}}\, \frac{p(\gamma|x)p(x)}{p(\gamma)} = \underset{x}{\operatorname{argmax}}\, p(\gamma|x)p(x) \tag{3.15}$$

We use the notation $x_{a:b}$ to denote a set of values $\{x_a, x_{a+1}, \ldots, x_b\}$ over time. In offline applications, we can make use of measurements $\gamma_{1:t}$ from all timesteps to find $x_{1:t}$. This is equivalent to a MAP estimation over $p(x_{1:t}|\gamma_{1:t})$ and is known as *smoothing*. In an online setting however, it is more efficient to calculate only the most recent state based on new measurements. This is called *filtering* and maximizes $p(x_t|\gamma_{1:t})$. By virtue of only working with the current state, filtering does not use new measurements to improve previous state estimates. A *fixed-lag smoother* generalizes the two notions by operating on the n most-recent timesteps, i.e. maximizing $p(x_{t-n:t}|\gamma_{1:t})$, such that filtering has $n = 0$ and $n = t - 1$ gives smoothing [13].

Gaussian case

Suppose the joint PDF is a product of factors and each one is a Gaussian probability. For clarity we define $(a-b)^\top A^{-1}(a-b) = \|a-b\|_A^2$. Hence, can take the negative log of the joint PDF to transform it into a least squares form with added normalizing constant.

$$\begin{aligned} p(x|\gamma) &\propto \prod_i \phi_i(x) \propto \prod_i \exp -\tfrac{1}{2} \|y_i - f_i(x)\|_{\Sigma_i}^2 \\ \implies -\log p(x|\gamma) &= C + \sum_i \|y_i - f_i(x)\|_{\Sigma_i}^2 \end{aligned} \tag{3.16}$$

If all f_i are linear, we can apply the ordinary least squares method to solve for $\widetilde{x}$. If they are non-linear, we can use the Gauss-Newton or Levenberg-Marquardt algorithms described in section 2.4 instead.

$$\widetilde{x} = \underset{x}{\operatorname{argmax}}\, p(x|\gamma) = \underset{x}{\operatorname{argmin}} \sum_i \|y_i - f_i(x)\|_{\Sigma_i}^2 \tag{3.17}$$

3.2.2 Non-Linear Smoothing

Using the Markov property, we expand the joint PDF for a smoother over N timesteps.

$$\begin{aligned}\widetilde{x} &= \underset{x}{\operatorname{argmax}}\, p(x_{1:N}|\gamma_{1:N}) = \underset{x}{\operatorname{argmax}}\, p(\gamma_{1:N}|x_{1:N})p(x_{1:N}) \\ &= \underset{x}{\operatorname{argmax}}\, p(x_1)\prod_{i=2}^{N} p(x_i|x_{i-1})\prod_{i=1}^{N} p(\gamma_i|x_i)\end{aligned} \tag{3.18}$$

To estimate orientation, our dynamic model e_ω is identical to dead reckoning in section 2.3, but reformulated as a least squares residual (inverted). Moreover, we add measurement residuals e_m and e_s that use the magnetometer and (when static) accelerometer direction vectors. We also have a prior residual e_p around the initial orientation.

$$\begin{aligned}e_p &= \mathrm{Rot}^{-1}(R_{\mathrm{p}}^{bn} R_0^{nb}) \\ e_\omega &= \frac{1}{T_t}\mathrm{Rot}^{-1}(R_t^{bn} R_{t+1}^{nb}) - \omega_t^b \\ e_m &= R_t^{nb} n^n - m_t^b \\ e_s &= R_t^{nb} g^n + s_t^b\end{aligned} \tag{3.19}$$

With the Levenberg-Marquardt algorithm, we minimize the following objective to obtain a MAP estimation for orientation [28]. The covariance matrices act as weights for the multi-objective optimization.

$$\widetilde{x} = \underset{x}{\operatorname{argmin}} \underbrace{\|e_p\|^2_{\Sigma_p}}_{\text{prior}} + \underbrace{\|e_\omega\|^2_{\Sigma_\omega}}_{\text{dynamic}} + \underbrace{\|e_m\|^2_{\Sigma_m} + \|e_s\|^2_{\Sigma_s}}_{\text{measurement}} \tag{3.20}$$

Since we linearize the rotation matrices around the identity with $IR^{bn} = \mathrm{Rot}(\eta)R^{bn}$, we differentiate over the deviation parameter initialized as $\eta = 0$ rather than the orientation being estimated. Derivation of the Jacobians with respect to η is simplified using the partial derivatives in section 2.2.

$$\frac{\partial e_p}{\partial \eta_0} = -R_p^{bn} R_0^{nb} \qquad \frac{\partial e_\omega}{\partial \eta_t} = \frac{R_t^{bn} R_{t+1}^{nb}}{T_t} \qquad \frac{\partial e_\omega}{\partial \eta_{t+1}} = -\frac{R_t^{bn} R_{t+1}^{nb}}{T_t} \tag{3.21}$$

$$\frac{\partial e_m}{\partial \eta_t} = [R_t^{bn} n^n]^\wedge \qquad \frac{\partial e_s}{\partial \eta_t} = [R_t^{bn} g^n]^\wedge \tag{3.22}$$

After calculating the rotation deviation, we update the orientations, reset η and repeat the optimization.

3.2.3 Non-Linear Filtering

The Kalman filter is a widely used estimation algorithm for discrete-time linear systems [24]. It repeatedly performs two steps; first it predicts the current state from the previous one using dynamic models, then it applies a correction based on measurements. To support non-linear functions, the extended Kalman filter (EKF) makes a first order Taylor approximation [38], while the iterated extended Kalman filter (IEKF) also does multiple corrections per timestep.

Our derivation is based on MAP estimation but is closely related to the Kalman filter variants. To expand the joint PDF for a filter, we condition it on measurements from previous timesteps.

$$\begin{aligned} \widetilde{x} &= \operatorname*{argmax}_x p(x_t|\gamma_{1:t}) = \operatorname*{argmax}_x p(\gamma_t|x_t, \gamma_{1:t-1}) p(x_t|\gamma_{1:t-1}) \\ &= \operatorname*{argmax}_x \underbrace{p(\gamma_t|x_t)}_{\text{measurements}} \underbrace{p(x_t|\gamma_{1:t-1})}_{\text{prediction}} \end{aligned} \tag{3.23}$$

While $p(\gamma_t|x_t)$ is simply a probability of the measurement model, finding $p(x_t|\gamma_{1:t-1})$ requires marginalization of the joint PDF of the current and previous state. Intuitively, this is a way of retaining knowledge from prior measurements.

$$\begin{aligned} p(x_t|\gamma_{1:t-1}) &= \int p(x_t, x_{t-1}|\gamma_{1:t-1})\, dx_{t-1} \\ &= \int \underbrace{p(x_t|x_{t-1})}_{\text{dynamics}} \underbrace{p(x_{t-1}|\gamma_{1:t-1})}_{\text{recursive step}}\, dx_{t-1} \end{aligned} \tag{3.24}$$

In the recursive step we use the PDF of the previous state conditioned on the measurements up to that point. Suppose the MAP estimate of the previous state is μ_{t-1} with added Gaussian noise $\sim \mathcal{N}(0, P)$, and that the dynamic model is a non-linear function $f(x_{t-1}, u_t, \epsilon)$ with $\epsilon \sim \mathcal{N}(0, Q)$ and input u_t. We take a first-order approximation of f to model the dynamic noise as Gaussian and simplify the integration.

$$\begin{aligned} f(x_{t-1}, u_t, \epsilon) &\approx f(\mu_{t-1}, u_t, 0) + F_t(x_{t-1} - \mu_{t-1}) + G_t\epsilon \\ &= f(\mu_{t-1}, u_t, 0) + F_t(x_{t-1} - \mu_{t-1}) + \mathcal{N}(\epsilon; 0, G_t Q G_t^\top) \\ F_t &= \frac{\partial f(x_{t-1}, u_t, 0)}{\partial x_{t-1}} \qquad G_t = \frac{\partial f(\mu_{t-1}, u_t, \epsilon)}{\partial \epsilon} \\ p(x_t|x_{t-1}) &\approx \mathcal{N}(x_t; f(\mu_{t-1}, u_t, 0) + F_t(x_{t-1} - \mu_{t-1}), G_t Q G_t^\top) \end{aligned} \tag{3.25}$$

Substituting $R = G_t Q G_t^\top$, the resulting integral is:

$$\begin{aligned} p(x_t|\gamma_{1:t-1}) &= \int p(x_t|x_{t-1}) p(x_t, x_{t-1}|\gamma_{1:t-1})\, dx_{t-1} \\ &= \int \mathcal{N}(x_t; f(\mu_{t-1}, u_t, 0) + F_t(x_{t-1} - \mu_{t-1}), R) \times \mathcal{N}(x_{t-1}; \mu_{t-1}, P)\, dx_{t-1} \\ &= Z \int \exp\left[-\tfrac{1}{2}\left(\|x_t - f(\mu_{t-1}, u_t, 0) - F_t(x_{t-1} - \mu_{t-1})\|_R^2 + \|x_{t-1} - \mu_{t-1}\|_P^2\right)\right] dx_{t-1} \\ &= Z \int \exp\left[-L_t(x_t, x_{t-1})\right] dx_{t-1} \end{aligned} \tag{3.26}$$

In the integration over x_{t-1} our goal is to split $L_t(x_t, x_{t-1})$ into a sum $M_t(x_t, x_{t-1}) + N_t(x_t)$. Taking the first and second derivatives of L_t give the mean and inverse covariance of a Gaussian probability distribution only in terms of x_{t-1}. Therefore, we can design M to be a probability density function such that integrating it results in a constant and can be absorbed into the normalizing factor Z'.

$$\begin{aligned} p(x_t|\gamma_{1:t-1}) &= Z \exp\left[-N_t(x_t)\right] \int \exp\left[-M_t(x_t, x_{t-1})\right] dx_{t-1} \\ &= Z' \exp\left[-N_t(x_t)\right] \end{aligned} \tag{3.27}$$

Now by solving the first and second derivatives of $N_t(x_t) = L_t(x_t, x_{t-1}) - M_t(x_t, x_{t-1})$ we find the mean and inverse covariance of the state from the dynamic model. For further details, see [38].

$$p(x_t|\gamma_{1:t-1}) \approx \mathcal{N}(x_t; f(\mu_{t-1}, u_t, 0), F_t P F_t^\top + R) \tag{3.28}$$

As shown in Equation 3.17, a Gaussian noise assumption reduces the MAP problem to one of minimizing the sum of square residual norms. Hence, the IEKF is equivalent to Gauss-Newton optimization over measurement models and the prior created in the prediction step [6].

In the application of orientation estimation, we can use the deviations as states and adapt the dead reckoning model. The objective is identical to Equation 3.20, although the prior probability distribution changes every timestep according to Equation 3.28.

$$\begin{aligned} \eta_{t+1} &= f(\eta_t, \omega_t, \epsilon) \\ &= \mathrm{Rot}^{-1}(\mathrm{Rot}(T_t(\omega_t + \epsilon))^\top \mathrm{Rot}(\eta_t) R_t^{bn} R_{t+1}^{nb}) \\ F_{t+1} &= I \qquad G_{t+1} = -T_t \mathrm{Rot}(T_t \omega_t) \end{aligned} \tag{3.29}$$

3.3 Estimation Through Message-Passing

The non-linear smoother and filter presented can be adapted quite naturally to the problem of orientation estimation. However, the smoother does not exploit our models' inherent sparsity resulting from the Markov property, and it would be beneficial to generalize these methods as special cases of a fixed-lag smoother. To this end, we re-interpret the probabilistic models as graphs, then apply a message passing algorithm to obtain MAP estimates.

3.3.1 Graphical Models

There are a variety of graphical models, which more explicitly represent the structure of a joint PDF. We briefly describe three variants.

Bayesian Networks

A Bayesian network $\mathcal{B} = (V, E)$ is a directed acyclic graph (DAG). Its nodes $V = \{X_1 \dots X_n\}$ represent random variables and the edges $E \subset V \times V$ show conditional dependencies. It encodes a probability distribution where factors are either a prior of a variable, or a posterior conditioned on its parents [42]. If we denote the parent nodes of X_i as π_i, the joint PDF over n variables is defined as

$$p_{X_1 \dots X_n}(x_1 \dots x_n) = \prod_i p(x_i|\pi_i)p(x_i) \tag{3.30}$$

Markov Random Fields

A Markov random field (MRF) is an undirected graph $\mathcal{M} = (V, E)$, which encodes conditional independence between variables in a joint PDF. For example, a three node graph $X - Z - Y$, the joint probability is $p_{XYZ}(x, y, z) = p_{X|Z}(x; z)p_{Y|Z}(y; z)p_Z(z)$. In particular, since all paths between nodes X and Y on the graph pass through node Z, we have $X \perp\!\!\!\perp Y \mid Z$. This is similar to the Markov property and motivates a factorization over cliques $\{(X, Z), (Y, Z)\}$. A clique is a complete induced subgraph and is maximal when it is not contained within a larger clique. If nodes in V are grouped into a set of maximal cliques C, the joint probability represented by the undirected graph can be written as a product of potential functions with inputs in C and normalizing partition function Z.

$$p_{X_1 \dots X_n}(x_1 \dots x_n) = \frac{1}{Z} \prod_{c \in C} \psi_c(x_c) \tag{3.31}$$

A DAG can be converted to an undirected graph through moralization – fully connecting each node's parents. This can result in loss of some conditional independencies though [29].

Factor Graphs

Factor graph $\mathcal{F} = (V, E, F)$ is an undirected bipartite graph with edges $E \subseteq V \times F$ between nodes V representing random variables and nodes F for factors, making the factorization explicit [30]. A variable connected to a factor means it is given as input to the factor function. Hence, the joint PDF with normalizing partition function Z can be expressed as:

$$p_{X_1 \dots X_n}(x_1 \dots x_n) = \frac{1}{Z} \prod_{\phi \in F} \phi(x_\phi) \tag{3.32}$$

3.3.2 Exploiting Sparsity

To motivate the use of graphical models as a way of leveraging the sparse structure, consider how we could more efficiently marginalize an arbitrary joint density function $p(x, y, z) = f(x, y)f(y, z)f(z)$.

$$\begin{aligned} p_X(x) = \int_y \int_z p(x, y, z)\, dzdy &= \int_y \int_z f(x, y)f(y, z)f(z)\, dzdy \\ &= \int_y f(x, y) \left[\int_z f(y, z)f(z)\, dz \right] dy \end{aligned} \tag{3.33}$$

By factorizing the joint probability and grouping the factors based on variables common to their inputs, we reduce the total number of functions we have to integrate. This is known as *variable elimination* and it works here because sums (integrals) distribute over multiplication, meaning the z-integral can be shifted inwards. Alternatively, we could have shifted the y-integral, i.e. used a different variable *elimination order*.

The Generalized Distributive Law (GDL) formalizes this notion with an algorithm to solve the computational problem of marginalizing a product of functions (MPF) [2]. It is closely related to belief propagation from machine learning literature and the sum-product algorithm over factor graphs [30]. The algorithm is defined for any commutative semiring – a set of elements equipped with a pair of binary operations that each have an identity element, are associative, commutative, and satisfy the distributive law. Two notable pairs of such operations are the sum-product $(+, \times)$ and, its negative log, the min-sum $(\min, +)$. It is easy to see that the min operation distributes over addition as $\min(x + y, x + z) = x + \min(y, z)$.

GDL Algorithm

Suppose we are given a list of variables $x = \{x_1, \dots, x_N\}$ each assigned values from their corresponding sets in list $A = \{A_1, \dots, A_N\}$ such that $x_i \in A_i$, and a set of index lists $S = \{S_1, \dots, S_M\}$ where $S_i \subseteq \{1, \dots, N\}$. Now let the *global kernel* $\beta : A \to R$ be the product

of M *local kernels* $\alpha_i : A_{S_i} \to R$ with local domains x_{S_i}. The MPF problem is then defined as computing the marginalization of the global kernel over a chosen local domain, i.e. we solve:

$$\beta_i(x_{S_i}) = \sum_{x \setminus x_{S_i}} \beta(x) = \sum_{x \setminus x_{S_i}} \prod_{j=1}^{M} \alpha(x_{S_j}) \tag{3.34}$$

The GDL algorithm requires that each index list S_i can be assigned to a node in a *junction tree* (also known as a Bayes' Tree when directed [13]). For an undirected tree $\mathcal{T} = (V, E)$ with nodes V and edges $E \subseteq V \times V$ to be considered a junction tree, it must satisfy the *running intersection property* – for all vertices $v_i, v_j, v_k \in V$ where v_k is on the path from v_i to v_j, it holds that $S_i \cap S_j \subseteq S_k$. This means that each index $d \in \{1, \ldots, N\}$ is assigned to nodes that form a sub-tree.

The single-vertex MPF problem is to find some marginal function β_r. The GDL algorithm efficiently solves it by setting the node assigned S_r as the root of a junction tree and filtering information upward, starting from the leaves. Each node creates a function from the local kernel α_{S_i} and the messages of its child nodes C_i. The message $\mu_{i \to j}$ passed over edge $(v_i, v_j) \in E$ is obtained by marginalizing over the variables whose indices are not in the *separator* $S_{ij} = S_i \cap S_j$. The resulting function computed at the root is the marginalization of the global kernel over its corresponding local domain.

$$\mu_{i \to j}(x_{S_{ij}}) = \sum_{x_{S_i} \setminus S_{ij}} \alpha_i(x_{S_i}) \prod_{v_k \in C_i} \mu_{k \to i}(x_{S_{ki}}) \tag{3.35}$$

GDL is essentially a dynamic programming algorithm for performing exact inference, since each marginalization gives an exact numerical result.

Junction Tree Construction

The joint PDF in MAP estimation can be explicitly represented by a factor graph, but to use the GDL algorithm the model needs to be in junction tree form. This motivates the design of a conversion procedure.

We can create junction trees from *chordal* Markov random fields, where every cycle of length greater than three has an edge (chord) connecting non-consecutive nodes [42]. In graph theory this is known as tree decomposition. Moreover, the representation of a factor graph as an MRF is apparent when comparing their joint PDFs; each factor is simply a clique over the variable nodes in its domain and if two factors have domains D and E such that $D \subseteq E$, the maximal clique over E represents their product. However, the resulting MRF is not necessarily chordal.

Consider a factor graph with three factors $\{f(i,x,y), g(y,z), h(x,z)\}$. Its equivalent MRF has a non-chordal cycle $[i,x,z,y]$ and if we tried to build a tree it would violate the running intersection property. Our goal is to split the cycles of length four or more into length three cycles (triangles). Hence, we *triangulate* the graph by adding chords between the non-adjacent neighbors of each variable following a selected elimination ordering. This process merges factors, e.g. if we start with variable i and add the edge (x,y), we combine $g(x,y)$ and $h(i,j)$ into their product $s(x,y,z)$.

Furthermore, a junction tree can be built concurrently with message passing. First, perform triangulation for a variable, then factorize the resulting factor and keep only the message component. Since the eliminated variable is not in the message factor's domain, we ensure that the running intersection property holds. Repeat this following the elimination order.

3.3.3 Multi-frontal QR Fixed-Lag Smoother

Now, we apply the GDL algorithm to MAP estimation by interpreting it as an MPF problem. We convert the factor graph into a junction tree, where the global kernel is the joint PDF and each local kernel is a product of factors.

The Kalman filter solving Equation 3.23 can be construed as a single-vertex GDL algorithm on the min-sum semiring. The corresponding junction tree contains the cliques $x_1 = (x_{t-1}, x_t)$ and root $x_2 = (x_t)$. The recursive step is simply a message from x_1 to x_2, while the root local kernel consists of the current measurement models and dynamics. When factorizing using $L_t = M_t(x_t, x_{t-1}) + N_t(x_t)$ in Equation 3.27, we designed M_t to be a Gaussian with mean and inverse covariance equal to the first and second derivatives of L_t with respect to x_{t-1}. Doing so was equivalent to sending a message $\mu_{1\to 2}(x_t) = N_t(x_t)$, which assumes x_{t-1} maximizes L_t.

While this technique works, *partial QR decomposition* offers a more general method of factorizing Gaussian PDFs without presupposing the factor graph structure. Taking the negative log of Equation 3.35, results in a sum of squares.

$$-\log(\alpha_i(x_{S_i})) + \sum_{v_k \in C_i} -\log(\mu_{k\to i}(x_{S_{ki}})) = \tfrac{1}{2}\left\|Ax_{S_i} - b\right\|^2 + \sum_{v_k \in C_i} \tfrac{1}{2}\left\|Ax_{S_{ki}} - b\right\|^2 \tag{3.36}$$

By collecting the coefficients into matrices $\bar{A}$ and $\bar{b}$ and taking the partial QR decomposition over the $x_{S_i \setminus S_{ij}}$ elements, we factorize the expression.

$$\begin{aligned} \tfrac{1}{2}\left\|\bar{A}x_{S_i} - \bar{b}\right\|^2 &= \tfrac{1}{2}\left\|Q^\top \bar{A}x_{S_i} - Q^\top \bar{b}\right\|^2 \\ &= \tfrac{1}{2}\left\|Rx_{S_i \setminus S_{ij}} + Tx_{S_{ij}} - d\right\|^2 + \tfrac{1}{2}\left\|\bar{A}_\mu x_{S_{ij}} - \bar{b}_\mu\right\|^2 \\ &= -\log(p(x_{S_i \setminus S_{ij}} | x_{S_{ij}})) + \mu_{i\to j}(x_{S_{ij}}) \end{aligned} \tag{3.37}$$

Hence, we can create a fixed-lag smoother by extending the Kalman filter using partial QR factorization for marginalization. It performs message passing up the junction tree until the root node has received messages from all its child nodes. Then we calculate the MAP estimate at the root and backtrack to compute estimates for the previous $N-1$ timesteps using Equation 3.38.

$$\widetilde{x}_{S_i \setminus S_{ij}} = R^{-1}(d - T\widetilde{x}_{S_{ij}}) \tag{3.38}$$

Curiously, the overall procedure is tantamount to *multi-frontal QR factorization* – an algorithm used to decompose sparse matrices in a distributed manner [14].

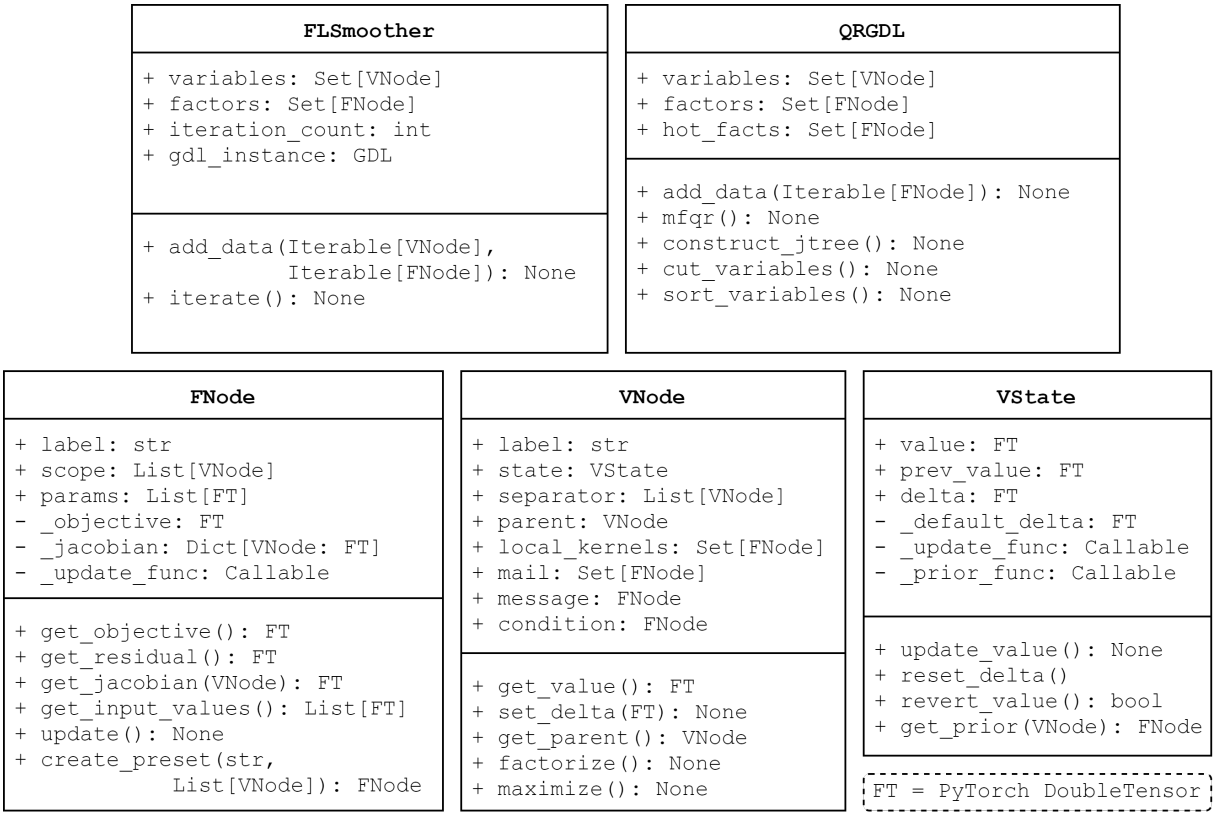

Figure 3.2: Fixed-Lag Smoother UML Diagram

Adding New Data

In the setup described thus far, we take a factor graph representing a joint PDF, and convert it into a junction tree over which we compute the MAP estimate efficiently. However, reconstructing the tree at each timestep to include new factors and variables is redundant. Instead, we can leverage the sparse structure to only update the affected portion of the factor

graph, i.e. design an algorithm that can add data to the smoother with minimal change to the junction tree. This method for incremental smoothing (assuming Gaussian noise) is presented as the iSAM2 algorithm in [26].

To add new factors to a junction tree, we locate the nodes that contain variables of their domains, then remove them and their parents up to the root, while retaining messages from their child nodes [13]. The junction tree is then rebuilt from the updated factor graph of affected nodes and subtrees rooted at child nodes are re-attached.

With these considerations, the algorithm is implemented in Python with classes designed according to Figure 3.2. The variable and factor nodes have corresponding classes while a state class enables the incremental updating of rotations. The `QRGDL` class is a general version of the algorithm with no specific application, while the `FLSmoother` additionally manages the rates at which variables are added and cut from model. The smoother can then be adapted for a variety of non-linear estimation problems.

3.4 Repository Overview

The repository is broken up into several modules, grouping related classes and functions. The `math3d` module consists of functions used to perform operations on rotations and dead reckoning, in addition to miscellaneous useful linear algebra methods. The `optim` module contains my implementation of the Levenberg-Marquardt algorithm, a smoother for orientation estimation and the more general fixed-lag smoother. Moreover, `calibration` holds the algorithms used to fit parameters to the sensors based on the models described in section 3.1. In `visuals` I have a useful function for visualizing 3D plots using `Tk` and `matplotlib`. Finally, in `data_collection` I wrote classes to control BLE connectivity, the sensor model, and the saving/loading of capture sessions that comprise of trials with asynchronous recording of data from multiple connected IMUs. For the high-risk components involve processing of rotations and kinematics I wrote unit `tests` to ensure they handle certain edge cases.

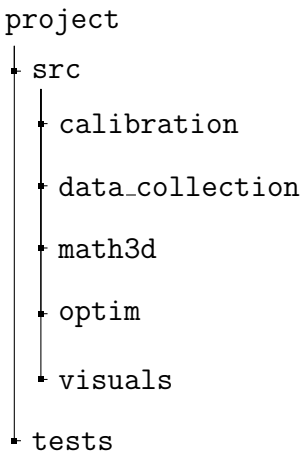

Figure 3.3: Modules

Chapter 4

Evaluation

In this section I evaluate the performance of the calibration procedure and fixed-lag smoothing algorithm. Dead reckoning and the complementary filter are used as a baseline, as they are the simplest solutions. The two metrics used to test the accuracy of pose estimation are orientation error in degrees and position error in meters. Trials from the MoVi dataset are used to evaluate the smoother, while the calibration method is tested on the Arduino boards and its estimation is compared to ground truth tracking by an OptiTrack motion capture system.

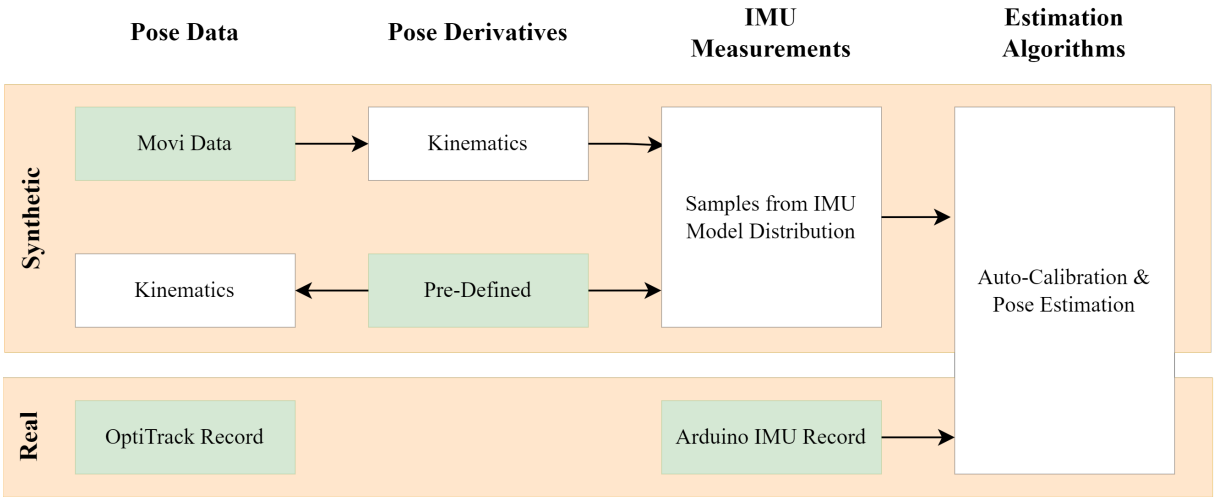

Figure 4.1: Test Pipeline

The `poser.py` file within `data_collection` is crucial in enabling easy conversion between IMU readings, pose derivatives, and pose calculated with dead-reckoning. It was especially useful in creating confidence checks for the calibration of a sensor with Gaussian noise (middle row above), as well as in the evaluation of the smoother (top row).

4.1 Calibration Tests

The Arduino boards are run through the calibration procedure in ten separate 90 second trials followed by 20 seconds of no motion (stationary pose). For stationary trials, the bias removal is immediately evident in the reduced drift in both estimated orientation and position. In Figure 4.2 the red lines are an offset of two standard deviations or approximately the 95th percentile.

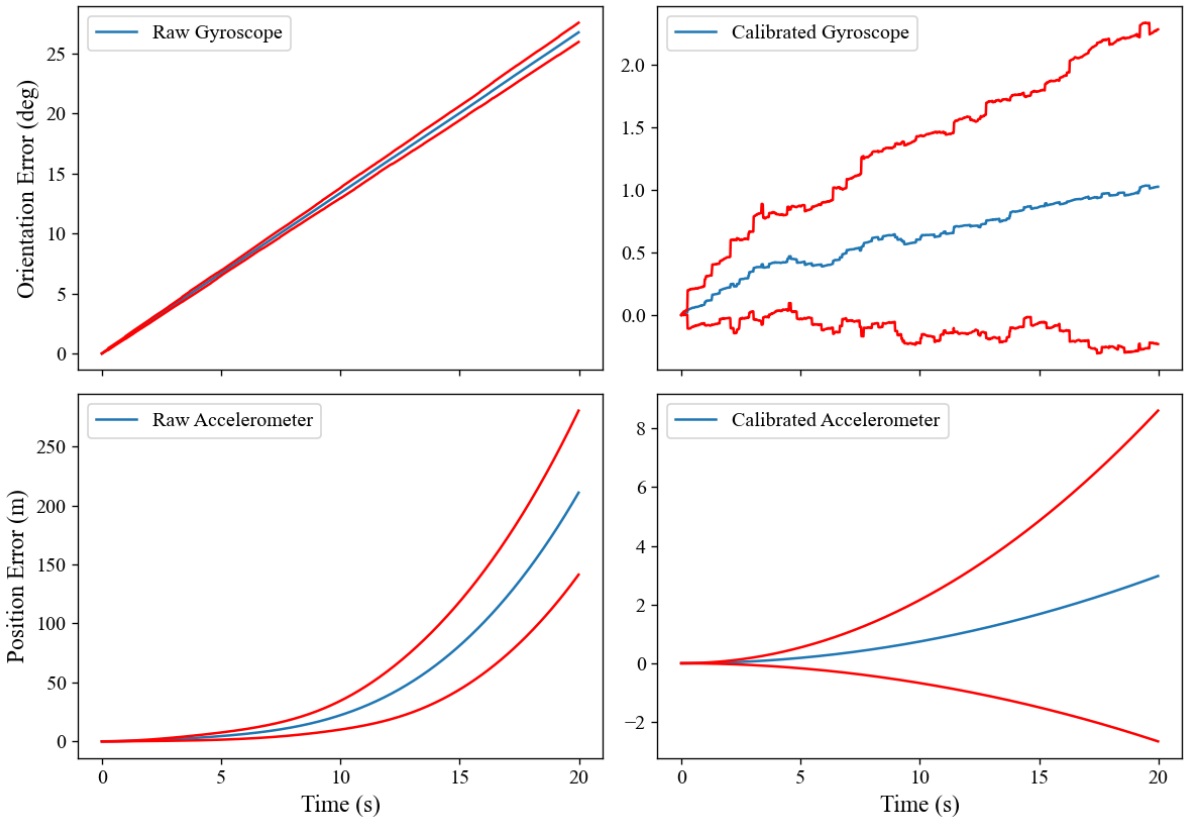

Figure 4.2: Pose Drift Before and After Calibration

For trials that involve movement, we can also verify that the gyroscope has been calibrated for scale by comparing the amount of rotation recorded by a motion capture system. The OptiTrack recording and data loaded using the `IMU_Record` class are aligned based on Unix timestamps and linear interpolation. In the case of rotation, the interpolation is spherical, so we calculate the rotation vector between consecutive orientations and divide by the time interval to obtain an angular velocity.

4.2 Pose Estimation Tests

The MAP estimate returned by the smoother is used to incrementally find parameters (a pose) that fits the non-linear model. Hence, it is essential to tune the hyper-parameters such that the optimization is fast but also accurate. To determine a suitable learning rate, I perform a grid search in the range $[0.001, 1.0]$ and inspect the loss curve for the first few iterations of the smoother. The optimization is initialized with poses from a faster method during smoothing. If doing filtering of the estimated pose is too inaccurate, we can also use the value of the previous state, assuming consecutive poses are not too different.

To test the algorithms on IMU data from human motion I used the MoVi dataset [18], specifically the joint poses it provides for the SMPL body model's skeleton. I then generated synthetic IMU readings and applied zero-mean Gaussian noise with the covariances measured from the calibrated IMUs on the Arduino boards. To estimate the sensor model covariance matrices required for the probabilistic model, I used ten trials of stationary data from the Arduinos, calculated the mean covariance ellipsoid volume using the determinant, and kept the volume cube root as a per axis standard deviation.

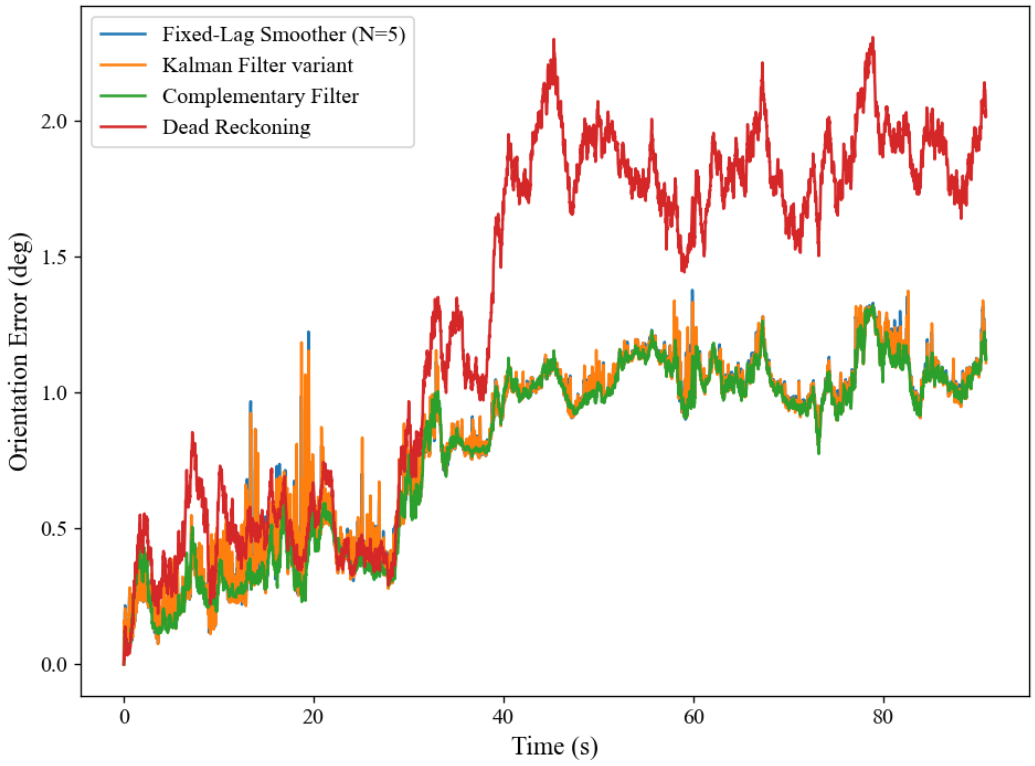

Figure 4.3: Single Trial of Orientation Estimation

As shown in Figure 4.3, the difference in accuracy between the complementary filter and fixed-lag smoother is inconclusive. This could be since the MoVi trials have lengths of about two minutes, so the error build up from sensor noise is not sufficient to reveal the benefits of the smoother. Nonetheless, the fusion of the IMU sensors results in decreased drift after 90 seconds.

Chapter 5

Conclusion

5.1 Summary

In this dissertation I investigated the problem of estimating pose from inertial sensor measurements. The implemented algorithms were successful at decreasing error in estimated pose compared to dead-reckoning on the tested IMU sensors as well as the synthetic dataset of human motion. The fixed-lag smoother using QR factorization is derived using the GDL algorithm and theoretically connected to the Kalman filter. An algorithm for alignment to a human subject was not tested, although a procedure (as in [35]) using a pre-defined user pose and determining global orientation through the IMU is feasible after calibration.

5.2 Lessons Learned

When starting the project it was difficult to determine its scope and what the expected results should be. Because of this my plan was not very clearly defined. I originally planned to use the six Arduino boards to create my own motion capture suit. This required calibrating the sensors and communication with a central processor. However, in retrospect, this led to many difficulties unrelated to the theoretical problem of pose estimation, and it was a poor decision because it essentially entailed creating a dataset from physical data, which is an entire separate challenge. If I were to attempt this project again, I would have used an entirely synthetic dataset and focused more on the Bayesian statistical methods from the start, giving me more time for the extensions as well. Nevertheless, I learned many new concepts related to state estimation and probabilistic modelling, which I will continue to explore in the future.

5.3 Further Work

While working on this project there were several opportunities for further exploration.

- It would be interesting to investigate sensor fusion of IMUs with other sensors (e.g. optical or electromagnetic).
- The models used in this project assume Gaussian noise, which is crucial for framing the MAP estimation as a least squares problem. One could study how other types of noise can also be added to the system.
- As seen in other papers, rather than using an exact inference method, a deep learning approach using RNNs could be more successful.
- The implementation of the fixed-lag smoother could be further improved in the future by porting it to C++.

Bibliography

[1] *Adafruit Pro Trinket LiPoly/LiIon Backpack*. en-US. URL: https://learn.adafruit.com/adafruit-pro-trinket-lipoly-slash-liion-backpack/pinouts (visited on 04/27/2022).

[2] S.M. Aji and R.J. McEliece. "The generalized distributive law". In: *IEEE Transactions on Information Theory* 46.2 (Mar. 2000). Conference Name: IEEE Transactions on Information Theory, pp. 325–343. ISSN: 1557-9654. DOI: 10.1109/18.825794.

[3] Simon L. Altmann. "Hamilton, Rodrigues, and the Quaternion Scandal". In: *Mathematics Magazine* 62.5 (1989). Publisher: Mathematical Association of America, pp. 291–308. ISSN: 0025-570X. DOI: 10.2307/2689481. URL: https://www.jstor.org/stable/2689481 (visited on 12/24/2021).

[4] *Arduino_LSM9DS1 - Arduino Reference*. URL: https://www.arduino.cc/reference/en/libraries/arduino_lsm9ds1/ (visited on 02/15/2022).

[5] Timothy D. Barfoot. *State Estimation for Robotics*. en. Cambridge: Cambridge University Press, 2017. ISBN: 978-1-316-67152-8. DOI: 10.1017/9781316671528. URL: http://ebooks.cambridge.org/ref/id/CBO9781316671528 (visited on 01/07/2022).

[6] B.M. Bell and F.W. Cathey. "The iterated Kalman filter update as a Gauss-Newton method". In: *IEEE Transactions on Automatic Control* 38.2 (Feb. 1993). Conference Name: IEEE Transactions on Automatic Control, pp. 294–297. ISSN: 1558-2523. DOI: 10.1109/9.250476.

[7] Bridget Bertoni. "Multi-dimensional Ellipsoidal Fitting". en. In: (), p. 7.

[8] Henrik Blidh. *bleak*. original-date: 2018-04-26T21:15:45Z. Apr. 2022. URL: https://github.com/hbldh/bleak (visited on 04/26/2022).

[9] Stephen Boyd and Lieven Vandenberghe. *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. en. 1st ed. Cambridge University Press, June 2018. ISBN: 978-1-316-51896-0 978-1-108-58366-4. DOI: 10.1017/9781108583664. URL: https://www.cambridge.org/core/product/identifier/9781108583664/type/book (visited on 01/05/2022).

[10] Polona Caserman, Philipp Achenbach, and Stefan Göbel. "Analysis of Inverse Kinematics Solutions for Full-Body Reconstruction in Virtual Reality". In: *2019 IEEE 7th International Conference on Serious Games and Applications for Health (SeGAH)*. ISSN: 2573-3060. Aug. 2019, pp. 1–8. DOI: 10.1109/SeGAH.2019.8882429.

[11] Francesco Crenna, Giovanni Battista Rossi, and Marta Berardengo. "Filtering Biomechanical Signals in Movement Analysis". en. In: *Sensors* 21.13 (Jan. 2021). Number: 13 Publisher: Multidisciplinary Digital Publishing Institute, p. 4580. ISSN: 1424-8220. DOI: 10.3390/s21134580. URL: https://www.mdpi.com/1424-8220/21/13/4580 (visited on 04/26/2022).

[12] Professor Andrew Davison. *Spatial AI: Augmenting SLAM technology with deep learning*. en. URL: https://blog.slamcore.com/spatial-ai-slam-deep-learning (visited on 05/07/2022).

[13] Frank Dellaert and Michael Kaess. "Factor Graphs for Robot Perception". en. In: *Foundations and Trends in Robotics* 6.1-2 (2017), pp. 1–139. ISSN: 1935-8253, 1935-8261. DOI: 10.1561/2300000043. URL: http://www.nowpublishers.com/article/Details/ROB-043 (visited on 03/12/2022).

[14] Frank Dellaert, Alexander Kipp, and Peter Krauthausen. "A Multifrontal QR Factorization Approach to Distributed Inference applied to Multi-robot Localization and Mapping". en. In: (), p. 6.

[15] Jay Farrell. "State Estimation". en. In: *The Electrical Engineering Handbook*. Elsevier, 2005, pp. 1049–1059. ISBN: 978-0-12-170960-0. DOI: 10.1016/B978-012170960-0/50081-5. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780121709600500815 (visited on 04/29/2022).

[16] Henri P Gavin. "The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems". en. In: (2020), p. 19.

[17] *Getting Started with Bluetooth Low Energy*. en. ISBN: 978-1-4919-0055-0. URL: https://learning.oreilly.com/library/view/getting-started-with/9781491900550/ (visited on 04/27/2022).

[18] Saeed Ghorbani et al. *MoVi: A Large Multipurpose Motion and Video Dataset*. 2020. arXiv: 2003.01888 [cs.CV].

[19] Andreas Grammenos, Cecilia Mascolo, and Jon Crowcroft. "You Are Sensing, but Are You Biased?: A User Unaided Sensor Calibration Approach for Mobile Sensing". en. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2.1 (Mar. 2018), pp. 1–26. ISSN: 2474-9567. DOI: 10.1145/3191743. URL: https://dl.acm.org/doi/10.1145/3191743 (visited on 02/03/2022).

[20] Patrick Guio. "Levi-Civita symbol and cross product vector/tensor". en. In: (), p. 4.

[21] Jeroen D Hol et al. *Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS*. en. OCLC: 1132377649. 2011. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-66184 (visited on 12/24/2021).

[22] Berthold K P Horn. "Some Notes on Unit Quaternions and Rotation". en. In: (), p. 4.

[23] Yinghao Huang et al. "Deep Inertial Poser: Learning to Reconstruct Human Pose from Sparse Inertial Measurements in Real Time". In: *arXiv:1810.04703 [cs]* (Oct. 2018). arXiv: 1810.04703. URL: http://arxiv.org/abs/1810.04703 (visited on 05/07/2022).

[24] Jeffrey Humpherys, Preston Redd, and Jeremy West. "A Fresh Look at the Kalman Filter". en. In: *SIAM Review* 54.4 (Jan. 2012), pp. 801–823. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/100799666. URL: http://epubs.siam.org/doi/10.1137/100799666 (visited on 05/04/2022).

[25] "iNEMO inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer". In: (), p. 72.

[26] Michael Kaess et al. "iSAM2: Incremental smoothing and mapping using the Bayes tree". en. In: *The International Journal of Robotics Research* 31.2 (Feb. 2012), pp. 216–235. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364911430419. URL: http://journals.sagepub.com/doi/10.1177/0278364911430419 (visited on 03/17/2022).

[27] Manuel Kaufmann et al. "EM-POSE: 3D Human Pose Estimation from Sparse Electromagnetic Trackers". en. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Montreal, QC, Canada: IEEE, Oct. 2021, pp. 11490–11500. ISBN: 978-1-66542-812-5. DOI: 10.1109/ICCV48922.2021.01131. URL: https://ieeexplore.ieee.org/document/9710700/ (visited on 04/28/2022).

[28] Manon Kok. *Probabilistic modeling for sensor fusion with inertial measurements.* en. OCLC: 967540571. Linköping: Linköping University Electronic Press, 2016. ISBN: 978-91-7685-621-5. URL: https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=4772047 (visited on 12/22/2021).

[29] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. en. Adaptive computation and machine learning. Cambridge, MA: MIT Press, 2009. ISBN: 978-0-262-01319-2.

[30] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm". In: *IEEE Transactions on Information Theory* 47.2 (Feb. 2001). Conference Name: IEEE Transactions on Information Theory, pp. 498–519. ISSN: 1557-9654. DOI: 10.1109/18.910572.

[31] *LSM9DS1 data sheet says that the FAST_ODR mode enables magnetometer data rates higher than 80Hz. How do we specify this data rate?* en-US. URL: https://community.st.com/s/question/0D50X00009XkYNMSA3/community.st.com/s (visited on 04/27/2022).

[32] R. Mahony, Tarek Hamel, and Jean-Michel Pflimlin. "Nonlinear Complementary Filters on the Special Orthogonal Group". In: *IEEE Transactions on Automatic Control* 53.5 (June 2008). Publisher: Institute of Electrical and Electronics Engineers, pp. 1203–1217. DOI: 10.1109/TAC.2008.923738. URL: https://hal.archives-ouvertes.fr/hal-00488376 (visited on 04/30/2022).

[33] *Powered by AI: Oculus Insight*. en. URL: https://ai.facebook.com/blog/powered-by-ai-oculus-insight/ (visited on 04/28/2022).

[34] Neil T. Roach et al. "Elastic energy storage in the shoulder and the evolution of high-speed throwing in Homo". en. In: *Nature* 498.7455 (June 2013), pp. 483–486. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature12267. URL: http://www.nature.com/articles/nature12267 (visited on 04/26/2022).

[35] Daniel Roetenberg, Henk Luinge, and Per Slycke. "Xsens MVN: Full 6DOF Human Motion Tracking Using Miniature Inertial Sensors". In: 8 (2009). URL: www.xsens.com.

[36] Sheng Shen, Mahanth Gowda, and Romit Roy Choudhury. "Closing the Gaps in Inertial Motion Tracking". In: New York, NY, USA: ACM, Oct. 2018, pp. 429–444. ISBN: 978-1-4503-5903-0. DOI: 10.1145/3241539.3241582. URL: https://dl.acm.org/doi/10.1145/3241539.3241582.

[37] Arnaud Taffanel et al. "Lighthouse Positioning System: Dataset, Accuracy, and Precision for UAV Research". In: *arXiv:2104.11523 [cs]* (Apr. 2021). arXiv: 2104.11523. URL: http://arxiv.org/abs/2104.11523 (visited on 04/28/2022).

[38] Sebastian Thrun. "Probabilistic robotics". en. In: *Communications of the ACM* 45.3 (Mar. 2002), pp. 52–57. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/504729.504754. URL: https://dl.acm.org/doi/10.1145/504729.504754 (visited on 05/01/2022).

[39] *Understanding Sensor Resolution Specifications and Performance*. Tech. rep. Lion Precision, Sept. 2014. URL: https://www.lionprecision.com/wp-content/uploads/2019/04/article-0010-sensor-resolution.pdf.

[40] Andrea Vitali. "1-point or 3-point tumble sensor calibration". en. In: (2022), p. 5.

[41] Daniel Vlasic Rolf Adelsberger et al. "Practical Motion Capture in Everyday Surroundings". In: *ACM Trans. Graph* 26 (2007). DOI: 10.1145/1239451.1239486. URL: http://doi.acm.org/10.1145/1239451.1239486.

[42] Martin J Wainwright and Michael I Jordan. "Graphical models, exponential families, and variational inference". en. In: (), p. 301.

[43] Oliver J Woodman. "An introduction to inertial navigation". en. In: (), p. 37.

[44] *World Magnetic Model — NCEI*. URL: https://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml (visited on 04/28/2022).

[45] Xinyu Yi, Yuxiao Zhou, and Feng Xu. "TransPose". In: *ACM Transactions on Graphics* 40.4 (Aug. 2021), pp. 1–13. DOI: 10.1145/3450626.3459786. URL: https://dl.acm.org/doi/10.1145/3450626.3459786.

[46] Xiaowei Zhou et al. "MonoCap: Monocular Human Motion Capture using a CNN Coupled with a Geometric Prior". In: *arXiv:1701.02354 [cs]* (Mar. 2018). arXiv: 1701.02354. URL: http://arxiv.org/abs/1701.02354 (visited on 05/07/2022).