Enhanced Design, Optimal Tuning, and Parallel Real-Time Implementation of Monocular Visual-Inertial-GNSS Navigation System

by

Soroush Sheikhpour Kourabbaslou

A dissertation submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

This Ph.D. thesis presents novel design, optimal tuning, and parallel real-time implementation of Tightly-Coupled (TC) Visual-Inertial Navigation (VIN) systems integrated with the Global Navigation Satellite System (GNSS) for autonomous vehicle applications. The Visual-Inertial-GNSS Navigation (VIGN) problem concerns estimating the position, velocity, and attitude of a mobile platform carrying only an on-board camera, an Inertial Measurement Unit (IMU), and a GNSS receiver. Although the cost and size efficiency of VIGN systems offer great commercialization potentials, achieving high accuracy, robustness, and real-time performance on embedded computing platforms are still challenging. Accuracy and robustness in VIGN systems heavily rely on two factors: first, a tightly-coupled fusion scheme that harvests the deep inter-modality correlations in the sensory data, and second, a well-tuned fusion model that sufficiently characterizes the actual behaviour of the VIGN system in practice. However, fusing multi-modal sensory data in the TC fashion scheme, as in the VIGN system, inevitably imposes a high computational burden due to the large state space and diverse volumes of visual processing, making it quite challenging to satisfying strict real-time constraints on embedded computing platforms. To address these challenges, this Ph.D. thesis proposes novel design and optimization approaches that have resulted in three main contributions.

This Ph.D. research initially develops an enhanced VIN system based on Multi-State Constraint Kalman Filter (MSCKF). Further, it proposes a novel systematic design and automatic tuning framework to adjust its design parameters for optimal state estimation. Evolutionary techniques based on the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) algorithm have been employed to efficiently search in a high-

dimensional search space for a set of parameters that optimize the performance of the developed VIGN system. The second contribution takes a step further and reformulates the combined problem of designing and tuning a VIGN system as a single end-to-end learning problem. A novel Deep Convolutional Recurrent Neural Network (DCRNN) architecture has been proposed to automatically train a model-free and calibration-free VIGN system. Finally, this thesis takes the VIN system developed in the first contribution and proposes a parallel and real-time implementation on embedded computing hardware. To satisfy strict real-time constraints, the computationally burdensome modules of the developed VIN's pipeline have been parallelized and distributed on the multi-core and many-core architectures of an embedded CPU-GPU-enabled computing platform.

The accuracy and real-time performance of the proposed solutions have been evaluated on real datasets supported by a hardware-in-the-loop simulation setup. The results demonstrate that the proposed enhanced MSCKF VIN system achieves 70% more accurate position estimation compared to the reduced state vector in the original work of MSCKF VIN. The proposed parallel CPU-GPU hardware implementation speeds up the processing time by 38%.

Acknowledgments

It has been a privilege working under the supervision of Dr. Mohamed Atia, whose patience, professional guidance, ongoing encouragement, and wholehearted support had a profound impact on my professional and personal life. I would also like to thank the Department of Electronics and my colleagues at the Embedded Multi-Sensor Systems Lab at Carleton University for their valuable assistance and inspiration. I would like to express my sincerest gratitude to my family for their unconditional love and support.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronym	Definition
3D	Three Dimensional
ANN	Artificial Neural Network
ASIC	Application-Specific Integrated Circuit
AV	Autonomous Vehicle
BA	Bundle Adjustment
CKF	Cubature Kalman Filter
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DCRNN	Deep Convolutional Recurrent Neural Network
DNN	Deep Neural Network
EKF	Extended Kalman Filter
FOG	Fibre Optics Gyro
FPGA	Field Programmable Gate Array
fps	frame per second
GA	Genetic Algorithm
GCC	GNU Compiler Collection
GM	Gauss Markov
GN	Gauss Newton
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HIL	Hardware In the Loop
Hz	Hertz
IDP	Inverse Depth Parameterization
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
KF	Kalman Filter

LC	Loosely Coupled
LiDAR	Light Detection And Ranging
MEMS	Micro Electro Mechanical System
MSCKF	Multi-State Constraint Kalman Filter
NVCC	Nvidia CUDA Compiler
PF	Particle Filter
PMOPSO	Parallel Multi-Objective Particle Swarm Optimization
PSO	Particle Swarm Optimization
RADAR	RAdio Detection And Ranging
RGB	Red Green Blue
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RTK	Real-Time Kinematic
SD	Secure Digital
SLAM	Simultaneous Localization And Mapping
SM	Streaming Multiprocessor
SoC	System on Chip
SP	Streaming Processor
STD	Standard Deviation
TC	Tightly-Coupled
VIGN	Visual-Inertial-GNSS Navigation
VIN	Visual-Inertial Navigation
VIO	Visual-Inertial Odometry
VO	Visual Odometry

List of Symbols

Symbol	Definition
$\{\boldsymbol{B}\}$	Body frame
$\{\boldsymbol{L}\}$	Local reference frame
$\{\boldsymbol{I}\}$	IMU frame
$\{\boldsymbol{C}\}$	Camera frame
$\{\boldsymbol{u}\}$	Image frame
$\{\boldsymbol{E}\}$	Earth-Centred Earth-Fixed
$\{\boldsymbol{i}\}$	Earth-Centred Inertial Frame
$\boldsymbol{x(t)}$	State vector in system state space
$\boldsymbol{u(t)}$	Control input vector in system state space
$\boldsymbol{y(t)}$	Measurement vector in system state space
$\boldsymbol{w(t)}$	Process/System noise
$\boldsymbol{Q(t)}$	Process/System noise covariance matrix
$\boldsymbol{v(t)}$	Measurement noise
$\boldsymbol{R(t)}$	Measurement noise covariance matrix
$\boldsymbol{F(t)}$	Jacobian of the system model
$\boldsymbol{G(t)}$	Jacobian of the input model
$\boldsymbol{H(t)}$	Jacobian of the measurement model
$\boldsymbol{P(t)}$	State error covariance matrix
$\boldsymbol{T_s}$	Sampling time of continuous system discretization
$\boldsymbol{K}$	Kalman Gain
$\boldsymbol{q_I^B}$	Quaternion rotation from $\{I\}$ frame to $\{B\}$ frame
$[\boldsymbol{a,b,c,d}]$	Quaternion components a:scalar, $[b,c,d]$: vector
$\boldsymbol{R_I^B}$	Rotation matrix from $\{I\}$ frame to $\{B\}$ frame
$\boldsymbol{t_C^B}$	Translation vector from $\{C\}$ frame to $\{B\}$ frame
$\boldsymbol{r}$	Roll angle
$\boldsymbol{p}$	Pitch angle
$\boldsymbol{A}$	Azimuth angle
$\boldsymbol{\omega_{iB}^B}$	The corrected rotation rate of the $\{B\}$ frame with respect to the $\{\boldsymbol{i}\}$ frame as represented in the $\{B\}$ frame

$\bar{\boldsymbol{\omega}}_{\boldsymbol{iB}}^{\boldsymbol{I}}$	The raw rotation rate measurement of the $\{B\}$ frame with respect to the $\{\boldsymbol{i}\}$ frame as represented in the $\{B\}$ frame
$\boldsymbol{v}_{\boldsymbol{LB}}^{\boldsymbol{L}}$	The velocity of the $\{B\}$ frame with respect to $\{L\}$ frame as represented in the $\{L\}$ frame
$\boldsymbol{a}_{\boldsymbol{iB}}^{\boldsymbol{L}}$	The corrected acceleration measurement of the $\{B\}$ frame with respect to the $\{\boldsymbol{i}\}$ frame as represented in the $\{B\}$ frame
$\bar{\boldsymbol{a}}_{\boldsymbol{iB}}^{\boldsymbol{I}}$	The raw acceleration measurement of the $\{B\}$ frame with respect to the $\{\boldsymbol{i}\}$ frame as represented in the $\{B\}$ frame
$\boldsymbol{g}^{\boldsymbol{L}}$	The gravity vector in the $\{L\}$ frame
$\boldsymbol{\omega}_{\boldsymbol{iE}}^{\boldsymbol{L}}$	The Earth rotation rate in the $\{L\}$ frame
$\boldsymbol{\omega}_{\boldsymbol{EL}}^{\boldsymbol{L}}$	The transportation rate in the $\{L\}$ frame
$\boldsymbol{\omega}_{\boldsymbol{ie}}$	The Earth rotation rate in the $\{\boldsymbol{i}\}$ frame
$\boldsymbol{\varphi}$	Lattitude
$\boldsymbol{\lambda}$	Longitude
$\boldsymbol{h}$	Altitude
$\boldsymbol{R}_{\boldsymbol{m}}$	The meridian radius of curvature of the Earth's ellipsoid model
$\boldsymbol{R}_{\boldsymbol{n}}$	The normal radius of curvature of the Earth's ellipsoid model
$\boldsymbol{e}$	The eccentricity of the Earth's ellipsoid model
$\boldsymbol{P}_{\boldsymbol{LB}}^{\boldsymbol{L}}$	The position of the $\{B\}$ frame with respect to $\{L\}$ frame as represented in the $\{L\}$ frame
$\boldsymbol{b}_{\boldsymbol{a}}$	The accelerometer measurement bias
$\boldsymbol{s}_{\boldsymbol{a}}$	The accelerometer measurement scale factor
$\boldsymbol{b}_{\boldsymbol{g}}$	The gyroscope measurement bias
$\boldsymbol{s}_{\boldsymbol{g}}$	The gyroscope measurement scale factor
$\boldsymbol{\tau}_{\boldsymbol{x}}$	The time constant of the Gauss-Markov process
$\boldsymbol{\rho}^{\boldsymbol{m}}$	The pseudo-range from the m^{th} satellite
$\boldsymbol{r}^{\boldsymbol{m}}$	The true range from the m^{th} satellite
$\boldsymbol{C}$	The speed of light
$\boldsymbol{t}_{\boldsymbol{R}}$	Time of GNSS message transmission
$\boldsymbol{t}_{\boldsymbol{T}}$	Time of GNSS message arrival

$\boldsymbol{D^m}$	The Doppler frequency shift
$\boldsymbol{1^m}$	The line of sight vector
$\boldsymbol{b_{\mathbb{c}}}$	The GNSS recevier clock bias
$\boldsymbol{d_{\mathbb{c}}}$	The GNSS receiver clock drift
$\boldsymbol{x_{C_{ints}}}$	The cameras intrinsic parameters state
$\boldsymbol{x_{C_{exts}}}$	The cameras extrinsic parameters state
$\boldsymbol{f_u, f_v}$	The camera's focal length along the $\{\boldsymbol{u}\}$ frame axes
$\boldsymbol{o_u, o_v}$	The camera's image center in the $\{\boldsymbol{u}\}$ frame
$\boldsymbol{z^i_{f_j}}$	The i^{th} Image frame coordinate of the j^{th} feature
$\boldsymbol{P^{C_i}_{C_i f_j}}$	The location of the j^{th} feature in the i^{th} camera frame
$\boldsymbol{\theta^i_j}$	The location of the j^{th} feature in the i^{th} camera frame in the IDP form
$\boldsymbol{M_j}$	The number of observation of the j^{th} feature
$\boldsymbol{r^k_{f_j}}$	The reprojection error of the j^{th} feature on the k^{th} image
$\boldsymbol{r_{f_j}}$	The reprojection error vector of all observations of the j^{th} feature
$\boldsymbol{J_{r_{f_j}}}$	The Jacobian matrix of the reprojection error vector
$\mathbb{r}$	The stacked reprojection error vector of all features
$\boldsymbol{F}$	Number of tracked features in MSCKF
$\boldsymbol{N}$	The size of the window part of the MSCFK state vector
$\boldsymbol{x^{<t>}}$	LSTM input at time t
$\boldsymbol{h^{<t>}}$	LSTM hidden state at time t
$\boldsymbol{c^{<t>}}$	LSTM cell state at time t
$\boldsymbol{o^{<t>}}$	LSTM output at time t
$\boldsymbol{W_c, W_f, W_u, W_o}$	LSTM cell, forget, update, and output gates weights
$\boldsymbol{b_c, b_f, b_u, b_o}$	LSTM cell, forget, update, and output gates biases
$\mathbb{v}_{\boldsymbol{i}}$	The velocity of the i^{th} particle in PSO population
$\mathbb{x}_{\boldsymbol{i}}$	The location of the i^{th} particle in PSO population
$\mathbb{x}_{\boldsymbol{Gbest}}$	The location of the global best particle
$\mathbb{x}_{\boldsymbol{Pbest_i}}$	The past best location of the i^{th} particle
$\boldsymbol{\mathcal{F}}$	PSO/GA objective function

$\boldsymbol{se(3)}$	Lie Algebra
$\boldsymbol{SE(3)}$	Lie Group
$\boldsymbol{\xi_t^{t+1}}$	The relative pose in Lie Algebra
$\mathbb{V}$	The velocity vector of all particles in PMOPSO
$\mathbb{X}$	The location vector of all particles in PMOPSO
$\mathbb{X}_{\boldsymbol{Gbest}}$	The location vector of best particles in PMOPSO
$\mathbb{X}_{\boldsymbol{pbest}}$	The past best location vector of particles in PMOPSO

Chapter 1: Introduction

This chapter begins by providing the motivation behind this Ph.D. research. The chapter first introduces sensor fusion's fundamentals for autonomous navigation purposes and then discusses challenges facing researchers and engineers in realizing accurate navigation systems on embedded computing systems. The innovative solutions proposed in this Ph.D. research to tackle these challenges and a list of publications are then demonstrated. Finally, the chapter is concluded by the thesis organization.

1.1 Motivation

In Autonomous Vehicles (AVs), navigation is the most fundamental capability that concerns estimating the vehicle's position, velocity, and attitude in a 3-Dimensional (3D) space [1]. As shown in Figure 1. 1, navigation systems responsible for state estimation are often driving engines of other systems (e.g. control, mapping, and planning). Navigation systems play an essential role in guaranteeing short-term stability as well as ultimate mission objectives.

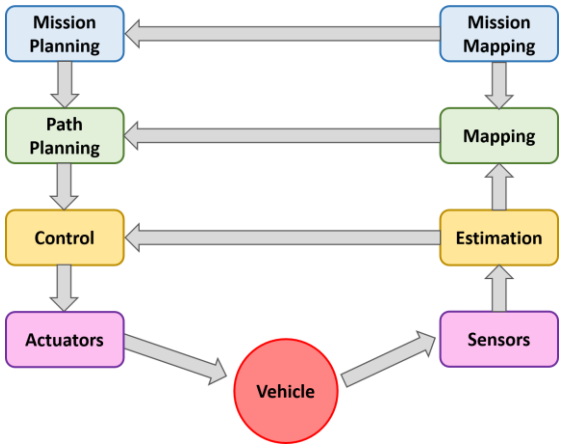


Figure 1. 1. Abstract scheme of sub-systems in an autonomous mobile system.

As sensor and embedded processors become more affordable, there is a great demand in various applications to employ autonomous navigation technologies on-board to obtain accurate estimates of the pose (i.e. position and attitude) of the mobile platform's and its surrounding objects in real-time [2][3][4]. As shown in Figure 1. 2, from robotics vacuum cleaners foraging for dust in a house, autonomous shuttles transporting passengers between airport terminals, to unmanned aerial vehicles assisting engineers in geo-referencing construction sites, all rely on navigation systems on-board the vehicle.

(a) (b) (c)

Figure 1. 2. Applications of autonomous navigation. (a) iRobot® Roomba® vacuum cleaner [2]**. (b) Autonum® Shuttle for passenger transportation in airports** [3]**. (c) Applanix™ photogrammetry technology on an unmanned aerial vehicle** [4]**.**

In most AV applications, the map of the environment is either too expensive or not static to obtain in advance. Hence, the AVs should solely rely on their on-board sensors to navigate in new and previously unmapped environments and even build a map progressively as they move in that environment. The most common sensors in AVs navigation include the Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU), Camera, RAdio Detection and Ranging (RADAR), and Light Detection And Ranging (LiDAR) [5]. However, no single sensor suits all applications and can provide an accurate navigation solution under all environmental conditions. For instance, Global Positioning System (GPS), as one of the GNSS constellations, provides an absolute

positioning solution with respect to a fixed reference frame at a low rate (e.g. 1Hz) within an accuracy range of few meters [6]; however, GPS does not provide reliable navigation information in dense urban and indoor areas, neither does it provide any information about the relative position of other surrounding objects. Commercial IMUs built with Micro Electro Mechanical Systems (MEMS) technology provide high-rate measurements (e.g. few hundred Hertz) of acceleration, angular speed, and heading; however, these measurements suffer from noise and different sources of error (e.g. bias and scale factor) [7]. Therefore, the inertial navigation solution gradually diverges and eventually becomes unreliable in the long term [8]. Table 1. 1 summarizes the advantages and disadvantages of IMU sensors and GNSS receivers in navigation applications.

Table 1. 1. Advantages and disadvantages of IMU sensors and GNSS receivers.

	GNSS	IMU
Advantage	• Absolute navigation solution • Available on any place on the Earth and above • No service fees • Cost-efficient	• Highly cost-efficient • Highly size-efficient • Highly power-efficient • High sampling rate • High measurement resolution
Disadvantage	• Not available indoors • Interrupted in urban canyons • Few meters accuracy	• Noisy measurements • Online calibration or error compensation is necessary

In the family of exteroceptive sensors, cameras can provide reasonably accurate navigation solutions and estimate the relative position of the surrounding objects by tracking sufficient salient visual features over multiple images [9]. Although RADARs are affordable and reliable in harsh conditions (e.g. dark or snowy), they provide low-resolution data that is more suitable for collision avoidance and long-range sensing rather than accurate navigation purposes [10]. LiDARs are very accurate in ranging and offer higher resolution, but two factors of cost and size negatively affect their popularity in small-sized navigation

systems [11]. Table 1. 2 compares Camera's, RADAR's, and LiDAR's attributes in terms of price, size, quality, accuracy, and working conditions in navigation applications [12].

Table 1. 2. Attributes comparison among the three exteroceptive sensors commonly used in AVs [12]**. The sensor's suitability is scaled from 1 (low) to 5 (high).**

Attribute	LiDAR	RADAR	Camera
Proximity Detection	2	4	2
Range	4	4	5
Resolution	4	3	5
Sensor Size	1	5	5
Sensor Cost	1	5	5
Work in Dark	5	5	1
Work in Snow/Fog/Rain	3	5	2
Detect Speed	4	5	2

Among these sensors, the combination of camera, IMU, and GNSS have shown such complementary characteristics that they have become very popular as a standalone, affordable, and compact multi-sensor navigation system for a wide range of commercial products. Perhaps this popularity has been strengthened further by biological findings that discovered Visual-Inertial fusion for perception and navigation also exists in some forms in the visual and hearing system in humans and some animals [13]. Hence, this Ph.D. research employs an on-board Camera, an IMU sensor, and a GNSS receiver to design an accurate multi-sensor fusion system implemented on an embedded computing platform for real-time navigation purposes.

To benefit the most from Visual-Inertial-GNSS fusion, sensory modalities and the potential correlations between them should be effectively utilized to achieve higher accuracy, reliability, and robustness than any of these sensors can achieve individually [14]. Hence, in the multi-sensor fusion literature, Tightly-Coupled (TC) fusion is a term commonly used to refer to a fusion scheme where sensory information is involved in the fusion process

with minimum pre-processing to harvest the deeper correlations between the modalities. For instance, the IMU measurements suffer from bias and scale factor errors that, if not compensated, lead to divergence of the inertial navigation solution within a short period; however, measurements of the GNSS receiver, such as position, velocity, or even range measurements from individual satellites, provide enough information that can potentially contribute to the estimation of the IMU's errors and consequently improve the overall navigation accuracy [15]. In another example, the monocular camera provides translation estimates up to an unknown scale factor; however, the IMU's corrected measurements can resolve the scale ambiguity [16]. Figure 1. 3 illustrates the VIGN problem and shows conditions in which IMU, Camera, or GNSS sensors can potentially complement other sensors and provide quality navigation solutions even in temporary failure or outage of other sensors.

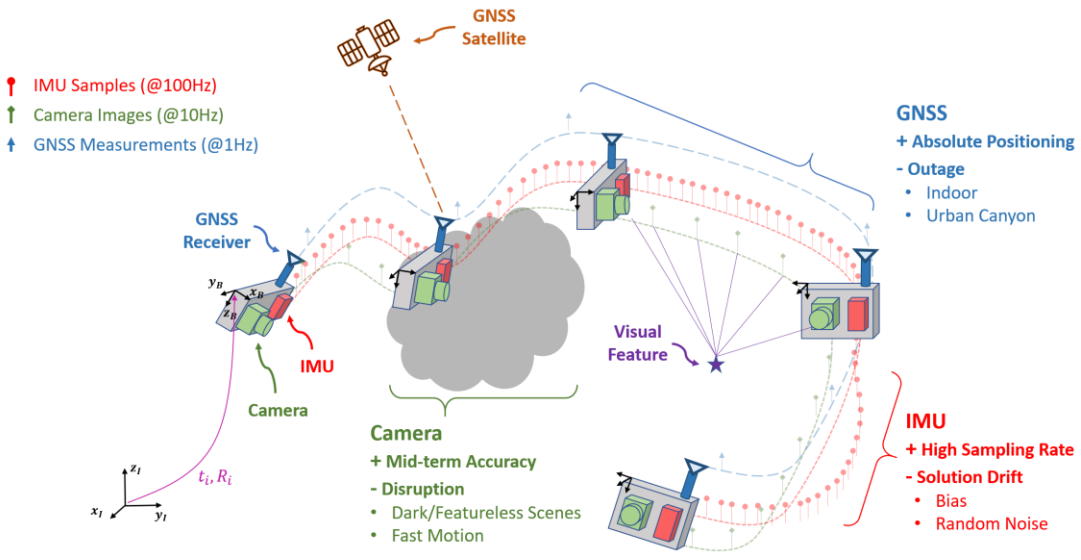


Figure 1. 3. The Visual-Inertial-GNSS navigation system and sensors' complementary characteristics.

It is worth discussing that any multi-sensor navigation system may have different performances depending on environmental conditions. For instance, in urban canyons

where GNSS signals are affected by multi-path and blockage, GNSS solution is poor, or through underground tunnels, GNSS service is unavailable. On the other hand, visual features in fog or dark environments are very sparse and short-lasting. However, it should be highlighted that as an interoceptive sensor, IMU sensors are highly reliable in different environmental conditions. When sufficiently long periods of GNSS or visual coverage provide quality updates to accurately estimate the IMU errors, if any GNSS or visual outage scenario occurs, the well-compensated inertial-only navigation system is legitimately capable of bridging this temporary outage scenario with acceptable accuracy. If the navigation system is supposed to work in particular environmental conditions such as fog or snow, Table 1. 2 can be used as a guideline to select the appropriate set of sensors. Current research trends, including this work, tend to introduce techniques to switch between multi-sensor fusion modes depending on the environmental conditions rather than building an all-purpose navigation system [17][18].

In the rest of this thesis, unless otherwise stated, Visual-Inertial Navigation (VIN) and Visual-Inertial Odometry (VIO) are used interchangeably to refer to the motion estimation/pose system using visual and inertial modalities. If, in addition to motion/pose estimation, mapping is also considered, the term Simultaneous Localization and Mapping (SLAM) is used to distinguish the system from the navigation- or odometry-only systems.

1.2 Multi-Sensor Fusion Techniques

Visual-Inertial-GNSS fusion in the literature has been addressed from broad perspectives that can be classified into three main categories: filtering-based, optimization-based, and

learning-based. In this section, these categories are introduced briefly, and a more in-depth survey is postponed until the following chapters after preliminary knowledge is reviewed.

In the filtering-based techniques that are often based on Extended Kalman Filter (EKF) or Particle Filter (PF), IMU's measurements are used to predict/propagate the system's states through time. States prediction/propagation is performed using a system model that describes how the system's states evolve dynamically through time. Along with the prediction step, the filtering-based techniques also keep track of the uncertainty around the estimations. Once a new image or a new GNSS message becomes available, it provides observations of the system's states in the measurement model. The fusion engine then applies a Bayesian inference with the prior distribution already estimated by the prediction step and the posterior distribution provided by the measurements to correct the predictions [19][20].

Optimization-based techniques (also known as batch techniques) reformulate the multi-sensor navigation problem into a minimization problem. The objective function is a nonlinear optimization problem that comprises a weighted sum of visual and inertial error terms. The visual error term in the objective function comprises the reprojection error of salient visual features or landmarks tracked over multiple images. On the other hand, the inertial measurements are treated as prior and impose temporal constraints on the camera's pose states from which the features were observed [21][22]. The objective function may also include error terms for features' 3D position when mapping is also required, as in SLAM algorithms. Nonlinear optimization algorithms such as the Levenberg-Marquardt algorithm [23] are often employed to solve this problem. The optimization is often

performed over a moving window of last poses to manage the processing load of optimization-based techniques.

Learning-based techniques revisit the multi-sensor navigation problem from a different perspective. The VIGN system, as a data processing system, performs two types of processing: visual and temporal. The visual processing analyses the discrepancies between any two successive images to geometrically estimate the relative poses of the camera when it captured those images. The temporal processing employs the dynamics of the system and constrains how systems states evolve through time. In learning-based techniques, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are jointly employed to perform, respectively, visual and temporal processing [24][25]. The main advantage of learning-based systems is that the fusion and estimation tasks are trained in an end-to-end and data-driven learning fashion without explicitly developing the system and measurement models.

Filtering-based algorithms enjoy lower computational complexity compared to optimization-based and learning-based algorithms, and therefore, they are more suitable for real-time implementation. Optimization-based algorithms closely follow filtering-based algorithms' accuracy, although with considerably higher computational requirements. Despite being a considerably newer paradigm, the learning-based approaches' advancement in terms of accuracy and robustness over recent years is very promising; however, their computational requirements are their main drawbacks. Hence, with the accuracy and performance targeted in this Ph.D. research, the EKF-based approach forms the backbone of the fusion engine of the developed VIGN system. However, innovative approaches based on optimization and learning techniques have also been

employed to enhance the developed EKF-based VIGN system. Moreover, the adoption of learning-based techniques has resulted in developing a standalone VIGN system as well, which will be discussed in the following chapters.

1.3 Thesis Problem Statement

The objectives of this Ph.D. research are threefold: design, automatic tuning, and real-time implementation of a VIGN system. To accomplish these objectives, different challenges had to be overcome first. The following sub-sections discuss the three main challenges of the VIGN systems that are considered in this Ph.D. research.

1.3.1 Problem I: Accurate Design and Automatic Tuning of the VIGN System

The promised superior performance of the Tightly-Coupled fusion scheme entails that sensors be modelled thoroughly to characterize different sources of error that potentially contaminate the measurements (e.g. GNSS's receiver clock errors, camera's calibration parameters, and IMU's biases, scale factors, and misalignment). Hence, the sensors' errors are often modelled as some extra states in the fusion system to be estimated along with navigation states (e.g. position, velocity, and attitude). Sensors' models are often characterized by stochastic parameters to account for the uncertainty that naturally exists in their measurements. These uncertainties might arise because of unmodelled or poorly modelled phenomena that affect the sensors' measurements. These uncertainties might also be due to discretization of the naturally continuous-time models or linearization approximations in the system [26]. The stability, accuracy, and robustness of the VIGN system are highly dependent on optimal tuning of these stochastic parameters to

characterize, as accurately as possible, the actual uncertainty and noise that contaminates the models under real working conditions. Traditionally, these parameters were tuned manually, which requires experts' knowledge and experience in a time-consuming and ad-hoc trial-and-error process. Given the large number of tunable parameters in VIGN systems (i.e. approximately 50 parameters), a tuning technique that both optimally and automatically determines the system's parameters is highly demanded.

1.3.2 Problem II: Automatic Modelling of the VIGN System

In the first problem discussed before, it was assumed that it is possible to obtain analytical models that describe the VIGN problem. The challenge was then focused on the optimal and automatic tuning of the system's stochastic parameters that best describe the models' actual stochastic behaviour in practice. However, when an analytical model is derived, it is often a simplified representation of the actual system that suffers from several approximations arising from linearization and discretization. In dead-reckoning navigation systems, as in VIN systems, obtaining accurate and well-tuned models that best characterize the actual behaviour of the system is highly appreciated because the models will remain valid for more extended periods when there are no absolute corrections from GNSS, for example. Given sufficient samples of the system's states, while operating under real conditions, the system's dynamic model and even its stochastic behaviour can be obtained using powerful learning and regression algorithms that directly extract relevant information and patterns from the real data. Therefore, the challenge is to design automatic modelling and training techniques to effectively learn a VIGN fusion model in a data-driven fashion using a large dataset containing raw visual, inertial, and navigation data.

1.3.3 Problem III: Real-Time Implementation of VIGN Systems

The TC VIGN system is sophisticated. First, it should be recognized that, as a multi-sensor fusion system, the VIGN system has a tangled pipeline with highly interactive modules performing various types of processes. Moreover, the TC VIGN problem is represented in a large state space, which indicates that the VIGN problem is a high-dimensional state estimation problem with high computational complexity at the algebraic level. More importantly, the TC integration of sensory modalities in a VIGN problem requires processing a mass of raw sensory data at a high rate (e.g. High-Definition images at 10-30fps, 6-9 elements of IMU measurements at 100-200Hz, and 6-12 elements of GNSS data at 1Hz). Hence, to achieve real-time performance by the VIGN system, all the required processing must be performed fast enough to keep pace with the sensors' sampling rate. These challenges breakdown into two main challenges. First, the VIGN pipeline comprises highly interactive modules that entail sequential execution orders and data dependencies. Such a sequential pipeline does not operate optimally when fast processing modules are inefficiently stalled for slow modules. Second, the VIGN pipeline is comprised of processing modules that must be executed repeatedly for a relatively large set of data in each iteration/epoch of the fusion process. Although these processes are not computationally complex, the iterative nature of these processes at a large scale accumulatively imposes a high computational burden on the computing platform. Hence, to achieve real-time performance for the VIGN on embedded computing platforms, the fusion pipeline and its processing modules must be optimized or even redesigned in the

most compatible form with the modern processors' architecture of the target embedded system.

1.4 Objectives and Contributions

This Ph.D. research seeks three main objectives in the VIGN system design, tuning, and implementation. In this section, objectives and contributions are briefly reviewed.

Objective 1: Accurate Modeling and Automatic tuning of the IMU-GNSS and the Visual-Inertial navigation systems

- **Contribution-1:** Development of an enhanced Multi-State Constraint Kalman Filter (MSCKF) for VIN system with a state vector augmented with more detailed IMU's errors and camera's calibration parameters. Estimating these hidden states online along with navigation states has led to 70% more accurate position estimation compared to the conventional system with the reduced state vector.
- **Contribution-2:** This thesis proposes a tuning technique based on the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) algorithm to automatically determine the parameters of an EKF-based IMU-GNSS system and MSCKF VIN system. The proposed tuning technique provides a set of optimal parameters while maintaining the generalizability of the parameters.

Objective 2: Automatic modelling of the Visual-Inertial navigation system

- **Contribution-3:** A novel Deep Convolutional Recurrent Neural Network architecture has been developed that automatically learns how to fuse multi-modal

visual and inertial sensory data to perform motion estimation. The system has been trained in an end-to-end and data-driven learning fashion that has also made it possible to develop a model-free and calibration-free VIN system and achieve competitive motion estimation results compared to classic VIN algorithms.

Objective 3: Real-time embedded hardware implementation of the Enhanced MSCKF VIN system

- **Contribution-4:** A novel parallel pipeline for the enhanced MSCKF VIN system has been developed that efficiently distributes the processing load on two parallel threads. 41% speedup in the per-epoch processing time has been achieved compared to the conventional single-thread implementation.
- **Contribution-5:** The visual backend module of the MSCKF VIN has been reformulated as a multi-objective optimization problem, with an efficient implementation on the GPU has been proposed. Integration of the GPU-accelerated visual backend module in the MSCKF VIN parallel pipeline developed in Contribution-4 has led to a 33% faster per-epoch processing time and satisfaction of strict real-time constraints on the embedded computing platform.

1.5 Thesis Organization

This thesis has been presented by gathering common background and mathematical formulations in Chapter 2. The three main objectives of this thesis and associated contributions are presented in Chapters 3-5. In each contribution chapter, a brief related

work section is presented, followed by thorough discussions on the contributions and experimental results. Finally, Chapter 6 concludes the thesis.

Chapter 1: Introduction: This chapter introduces the research motivations, fundamental concepts of multi-sensor fusion, associated challenges, objectives, and contributions.

Chapter 2: Background and Preliminaries: This chapter reviews background knowledge to prepare for more profound discussions of the following chapters. Topics covered in this section include coordinate frames, Extended Kalman Filter, system and measurement model of the IMU-GNSS and Visual-Inertial systems, evolutionary optimization algorithms, and deep neural networks.

Chapter 3: Automatic Tuning of the VIGN System: This chapter relates to the thesis's first objective, "Accurate Modeling and Automatic tuning of the IMU-GNSS and the Visual-Inertial navigation systems". This chapter first reformulates the automatic parameters tuning problem of EKF-based navigation systems into an optimization problem and then discusses the proposed automatic tuning based on evolutionary optimization algorithms.

Chapter 4: End-To-End Learning Approach to VIN Problem: This chapter relates to the thesis's second objective, "Automatic modelling of the Visual-Inertial navigation system". This chapter discusses the proposed Deep Convolutional Recurrent Neural Network architecture for the VIGN system.

Chapter 5: Embedded Real-Time Implementation of VIN System: This chapter relates to the thesis's third objective, "Real-time embedded hardware implementation of the Enhanced MSCKF VIN system". This chapter discusses the optimization of the pipeline of the enhanced MSCKF VIN system for real-time implementation on a CPU-GPU-enabled embedded system.

Chapter 6: Conclusion: This chapter concludes the thesis and discusses potential directions for future works.

1.6 List of Publications

This Ph.D. research consists of works from the following journal paper, conference proceedings, and papers awaiting publications:

[1] S. S. Kourabbaslou, M. M. Atia, and S. Waslander, "A flexible simulation and design environment for IMU/GNSS sensors integration," in *Midwest Symposium on Circuits and Systems*, 2019, Windsor, ON, Canada, 2018, pp. 472–475.

[2] S. S. Kourabbaslou, A. Zhang, and M. M. Atia, "A Novel Design Framework for Tightly Coupled IMU/GNSS Sensor Fusion Using Inverse-Kinematics, Symbolic Engines, and Genetic Algorithms," *IEEE Sensors Journal*, vol. 19, no. 23, pp. 11424–11436, Aug. 2019, 10.1109/JSEN.2019.2935324

[3] S. S. Kourabbaslou and M. M. Atia, "An Enhanced Visual-Inertial Navigation System Based on Multi-State Constraint Kalman Filter," in *Midwest Symposium on Circuits and Systems*, 2020, vol. Springfield, MA, USA, 2020, pp. 361–364.

[4] S. S. Kourabbaslou and M. M. Atia. “An Enhanced Multi-State Constraint Kalman Filter for Visual-Inertial Navigation with Particle Swarm Optimization Tuning.” *Journal of Robotics and Autonomous Systems* (2020). (Under Review, Submission Date: July 23, 2020, Manuscript ID: ROBOT-D-20-00213).

[5] S. S. Kourabbaslou and M. M. Atia, “Calibration-free visual-inertial fusion with deep convolutional recurrent neural networks,” in *Proceedings of the 32nd International Technical Meeting of the Satellite Division of the Institute of Navigation, ION GNSS+ 2019*, Miami, FL, USA, 2019, pp. 2198–2209.

[6] S. S. Kourabbaslou and M. M. Atia. “Toward Real-Time CPU Embedded Implementation of the Tightly-Coupled Visual-Inertial Navigation System.” *Journal of Parallel and Distributed Computing* (2020). (Under Review, Submission Date: November 8, 2020, Manuscript ID: JPDC-D-20-00355).

[7] S. S. Kourabbaslou and M. M. Atia. “GPU-Accelerated Parallel Multi-Objective Particle Swarm Optimization for Real-Time Tightly-Coupled Visual-Inertial Navigation System.” *IEEE Transactions on Evolutionary Computation* (2021). (Under Review, Submission Date: March 24, 2021, Manuscript ID: TEVC-00164-2021).

Chapter 2: Background and Preliminaries

This chapter presents background knowledge related to the VIGN navigation systems. It covers preliminary topics on the system of coordinate frames, Extended Kalman Filter (EKF), Inertial Navigation System (INS), Global Navigation Satellite Systems (GNSS) measurement model, visual measurement model, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and CPU-GPU-enabled embedded computing platforms.

2.1 System of Coordinate Frames

Any navigation system involves transforming quantities of interests (e.g. position. velocity, attitude, acceleration, and rotation) between different coordinate frames. In the VIGN navigation system, a system of seven coordinate frames is sufficient to describe quantities of interests. Figure 2. 1 demonstrate these coordinate frames.

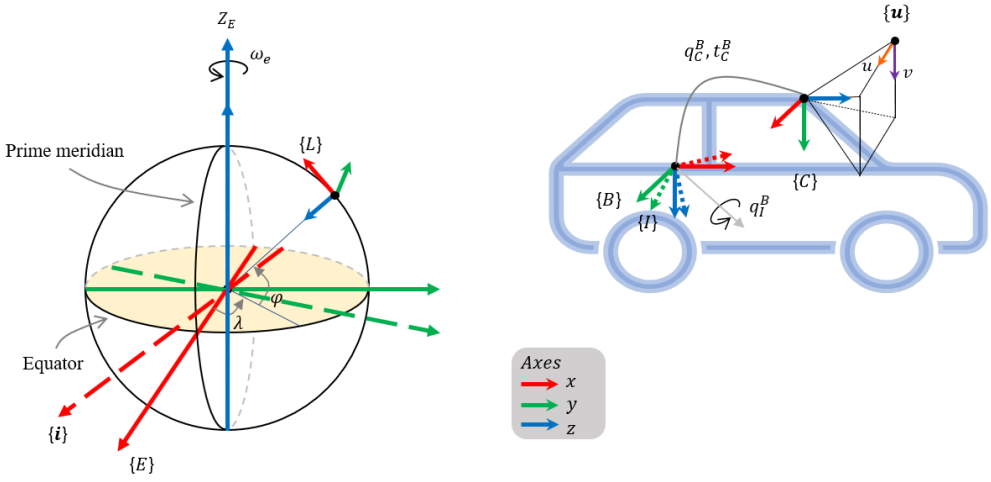


Figure 2. 1. The system of coordinate frames.

The body frame, $\{B\}$, is attached to the center of mass of the vehicle. The objective of the navigation system is to estimate the position, velocity, and attitude of the body frame with respect to a local reference frame $\{L\}$. The $\{L\}$ frame is always levelled with respect to the Earth's ellipsoid surface, and its axes point to the true North, East, and Down (NED). To account for sensor mounting errors, the IMU frame $\{I\}$ can be assumed to be slightly oriented with respect to the $\{B\}$ frame with quaternion angle q_I^B. Attached to the center of the projection of the camera is the camera frame $\{C\}$. This frame is oriented and displaced with respect to the $\{B\}$ frame by q_C^B and t_C^B, respectively. Sometimes, q_C^B and t_C^B are called *extrinsic parameters* of the camera. The 2D image frame, $\{\boldsymbol{u}\}$, (in the unit of pixel) is located at the top left corner of the image plane. The rotating Earth-Centered Earth-Fixed (ECEF) frame is located at the center of the Earth and is shown by $\{E\}$. The Earth-Centered Inertial Frame, $\{\boldsymbol{i}\}$, coincides with the $\{E\}$ frame at an initial time. It is worth mentioning that Newton's laws of motion apply in the $\{\boldsymbol{i}\}$ frame, but IMU's measurements are represented in the $\{I\}$ frame. Table 2. 1 summarizes the coordinate frames.

The attitude of the $\{B\}$ frame with respect to the $\{L\}$ frame can be represented in different forms such as Rotation Matrix $R_B^L \in \mathbb{R}^{3x3}$, quaternion vector $q_B^L \in \mathbb{R}^4$, rotation vector $\emptyset_B^L \in \mathbb{R}^3$, or Euler angles (Roll: r, Pitch: p, and Azimuth: A). In navigation systems, the rotation calculation in quaternion is often preferred over Euler angles because of their associated linear differential equations, lack of singularities, and a smaller number of parameters than the Rotation matrix [27][28]. However, Euler angles are often preferred for visualization. Quaternion q_B^L has four components $[a, b, c, d]$ where $a = cos(\emptyset_B^L/2)$ is the scalar part and $[b, c, d]^T$ is the vector part given by $sin(\emptyset_B^L/2)[l \quad m \quad n]^T$, where $[l, m, n]^T$ is the

rotation vector $\emptyset_B^L$. Quaternion q_B^L respects the unity constraint $a^2 + b^2 + c^2 + d^2 = 1$. In this work, the Hamilton convention for quaternions has been used [29].

Table 2. 1. The axes and origin of coordinate frames in the VIGN problem.

Frame	Definition
{B}: Body Frame	Origin: Vehicle's center of mass X: Longitudinal (forward) direction Y: Transversal (lateral) direction Z: Down (vertical) direction
{I}: IMU Frame	Attached to the IMU Oriented with respect to the $\{B\}$ frame by quaternion angle q_I^B X, Y, X toward the X, Y, Z of the IMU sensor
{C}: Camera Frame	Attached to the camera's center of projection Oriented and displaced with respect to the $\{B\}$ frame by q_C^B and t_C^B X: Toward the right side of the image plane Y: Toward the upside of the image Z: Perpendicular to image plane toward the outside of the image plane
{u}: Image Frame	Origin: The top left corner of the image plane (In the unit of pixel) u: Points to the right side of the image plane v: Points to the downside of the image
{L}: Local Level navigation frame	Levelled with respect to Earth ellipsoid surface Origin: Vehicle location X: True North direction Y: True East direction Z: Down vertical direction
{E}: Earth-Centred Earth-Fixed (ECEF)	Rotates with Earth Origin: Center of the Earth X: Passes through the intersection of the equatorial plane and the prime meridian. Y: Completes the right-hand coordinate system in the equatorial plane. Z: Extends through the North Pole.
{i}: Earth-Centred Inertial Frame	Coincides with the ECEF frame at a specific initial time, and it does not rotate.

2.2 Extended Kalman Filter

The Extended Kalman Filter (EKF) is an extension of the Kalman Filter (KF) that was proposed for state estimation in nonlinear dynamic systems [30]. The EKF linearizes a nonlinear system around its current estimate of the mean of the states. As a result, the system is represented linearly in terms of the states error (instead of the absolute states as in KF), allowing the Kalman to be still applicable on the linearized system. Although the mean square error optimality is not guaranteed in the EKF case; however, in many applications, EKF provides an accurate approximation of it [30]. A nonlinear system can be defined as follows:

$$\dot{x}(t) = f\big(x(t), u(t), w(t)\big), \tag{2.1}$$

$$y(t) = h\big(x(t)\big) + v(t), \tag{2.2}$$

where $x(t) \in \mathbb{R}^n$ is the n-element state vector, $u(t) \in \mathbb{R}^d$ is d-element control input vector, and $w(t)$ models the process noise. $f(.)$ is a nonlinear function defining the dynamics of the system and how the system's states evolve through time as a function of the states, control input, and process noise. $y(t) \in \mathbb{R}^m$ represents m-element measurement vector. Measurements are nonlinearly related to the system's states via the measurement model, $h(.)$, and the additive random noise $v(t) \in \mathbb{R}^m$ that models the measurement noise. Expanding (2.1) and (2.2) with Taylor series approximation, a linear system in terms of states error can be obtained as follows:

$$\delta\dot{x}(t) = F(t)\delta x(t) + G(t)w(t), \tag{2.3}$$

$$\delta y(t) = H(t)\delta x(t) + v(t), \tag{2.4}$$

where

$$F_{n\times n}(t) = \frac{\partial f\big(x(t), u(t), w(t)\big)}{\partial x}, \tag{2.5}$$

$$G_{n\times n}(t) = \frac{\partial f\big(x(t), u(t), w(t)\big)}{\partial w}, \tag{2.6}$$

$$H_{m\times n}(t) = \frac{\partial h\big(x(t)\big)}{\partial x}, \tag{2.7}$$

where F, G, and H matrices are the Jacobians of the system, input, and measurement models, respectively.

System states error, $\delta x = \tilde{x} - x$, is the difference between the actual value, $\tilde{x}$, and the estimated value of the state vector. Similar to the KF, process and measurement noises are considered as zero-mean Gaussian noise with covariance $Q(t) \in \mathbb{R}^{n\times n}$ and $R(t) \in \mathbb{R}^{m\times m}$, respectively, as follows:

$$Q(t) = \langle w(t)w^T(t)\rangle, \tag{2.8}$$

$$R(t) = \langle v(t)v^T(t)\rangle, \tag{2.9}$$

where $<.>$ is the mathematical expectation operator. The state error estimation covariance can then be defined as:

$$P(t) = \langle \delta x(t)\delta x^T(t)\rangle. \tag{2.10}$$

In discrete space, (2.3) and (2.4) can be rewritten as follows:

$$\delta x_k = (I + F_k T_s)\delta x_{k-1} + G_k w_{k-1}, \tag{2.11}$$

$$\delta y_k = H_k \delta x_k + v_k, \tag{2.12}$$

where T_s is the sampling period of the discretization, and subscript k denotes the discrete time step. Given measurement y_k, the estimation and correction with the EKF are performed with the following recursive Prediction and Update steps:

Prediction Step:

$$x_k^- = f(x_{k-1}, u_k, 0) \tag{2.13}$$

$$P_k^- = (I + F_{k-1}T_s)P_{k-1}^-(I + F_{k-1}T_s)^T + G_k Q G_k^T T_s \tag{2.14}$$

Measurement Update Step:

$$K_k = P_k^- {H_k}^T \left(H_k P_k^- {H_k}^T + R\right)^{-1} \tag{2.15}$$

$$x_k^+ = x_k^- + K_k[y_k - h(x_k^-)] \tag{2.16}$$

$$P_k^+ = (I - K_k H_k)P_k^- \tag{2.17}$$

Equations (2.13) and (2.14) propagate, respectively, the state and covariance to the next time step. Once a new measurement is obtained, the Kalman gain, K from equation (2.15), is computed, and then equations (2.16) and (2.17) correct the state and covariance estimates, respectively.

2.3 Inertial Navigation System

In this section, the system model (also called the dynamic model) of an Inertial Navigation System (INS) is discussed. This model defines how a rigid body's position, velocity, and attitude evolve dynamically through time.

The fundamental rotation equation of a rigid body is defined as follows [6]:

$$\begin{aligned} \dot{q_B^L} &= \frac{1}{2}\Omega(\omega_{iB}^B)q_B^L = \frac{1}{2}\begin{bmatrix} 0 & -{\omega_{iB}^B}^T \\ \omega_{iB}^B & -[\omega_{iB}^B]_\times \end{bmatrix}\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \\ &= \tfrac{1}{2}\begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix}\begin{bmatrix} 0 \\ \omega_{iB_x}^B \\ \omega_{iB_y}^B \\ \omega_{iB_z}^B \end{bmatrix}, \end{aligned} \tag{2.18}$$

where ω_{iB}^{B} is the corrected rotation rate of the $\{B\}$ frame with respect to the $\{\boldsymbol{i}\}$ frame as represented in the $\{B\}$ frame, q_B^L is the attitude of the body frame with respect to the $\{L\}$ frame, and $[.]_\times$ is the skew operator as follows:

$$[\omega]_\times \triangleq \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{2.19}$$

The velocity of a rigid body in the $\{L\}$ frame evolves according to the following equation [6]:

$$\dot{v}_{LB}^{L} = a_{\boldsymbol{i}B}^{L} + g^{L} - (2\omega_{\boldsymbol{i}E}^{L} + \omega_{EL}^{L}) \times v_{LB}^{L}, \tag{2.20}$$

where $v_{LB}^{L} = [v_n, v_e, v_d]$ is the velocity vector in the $\{L\}$ frame along the NED axes, $a_{\boldsymbol{i}B}^{L}$ is the corrected acceleration measurement in the $\{L\}$ frame, g^L is the gravity vector in the $\{L\}$ frame, and operator $\times$ is the cross product. In the above equation, $\omega_{\boldsymbol{i}E}^{L}$ and ω_{EL}^{L} are, respectively, the Earth rotation rate and transportation rate (rotation arising from motion parallel to the surface of the Earth ellipsoid model) as follows [6]:

$$\omega_{\boldsymbol{i}E}^{L} = [\omega_{ie}\cos(\varphi), \quad 0, \quad -\omega_{ie}\sin(\varphi)], \tag{2.21}$$

$$\omega_{EL}^{L} = \left[\frac{v_e}{(R_n + h)}, \quad \frac{-v_n}{(R_m + h)}, \quad \frac{-v_e}{R_n + h}\tan(\varphi)\right], \tag{2.22}$$

where $\omega_{ie} = (2\pi/(24 \times 60 \times 60))\ rad/s$ is the Earth's rotation rate represented in the $\{\boldsymbol{i}\}$ frame, and φ, λ, h are, respectively, the latitude, longitude, and altitude. R_m and R_n are, respectively, the meridian radius of curvature and normal radius of curvature of the Earth's ellipsoid model [6].

The position of the $\{B\}$ frame in the $\{L\}$ frame evolve according to the following equation [6]:

$$\dot{P}_{LB}^{L} = v_{LB}^{L} - \omega_{EL}^{L} \times P_{LB}^{L}, \tag{2.23}$$

where P_{LB}^{L} is the position of the $\{B\}$ in the $\{L\}$ frame.

Primary sources of error in IMU's measurements include bias, scale factor, and random walk noise. Figure 2. 2 shows how these errors affect the actual measurements of the IMU.

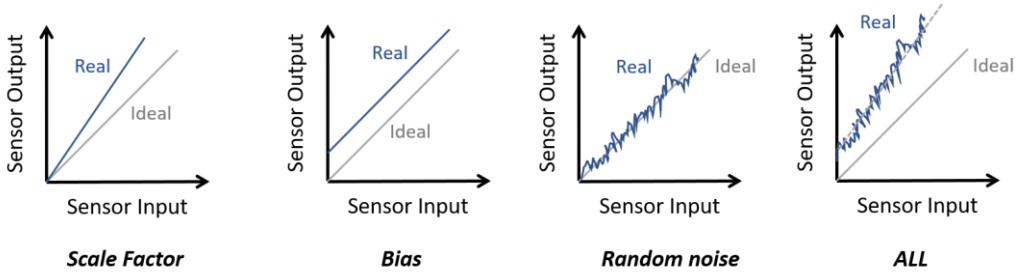


Figure 2. 2. From left to right, Scale factor, bias, random walk noise, and all combined errors on the IMU's measurements.

A commonly used error model for IMU's measurements is defined as follows:

$$a_{iB}^{B} = R_{I}^{B}(s_a \circ \bar{a}_{iB}^{I} - b_a) + n_a\,, \tag{2.24}$$

$$\omega_{iB}^{B} = R_{I}^{B}\left(s_g \circ \bar{\omega}_{iB}^{I} - b_g\right) + n_g\,, \tag{2.25}$$

where a_{iB}^{B} and $\bar{a}_{iB}^{I}$ are, respectively, the corrected and raw accelerometer's measurements, ω_{iB}^{B} and $\bar{\omega}_{iB}^{I}$ are, respectively, the corrected and raw gyroscope's measurements, $R_{I}^{B} = R\{q_{B}^{L}\}$ is the rotation matrix that corrects the $\{B\}$ and $\{I\}$ frames misalignment. s_a and s_g are, respectively, accelerometer's and gyroscope's scale factor, b_a and b_a are, respectively, accelerometer's and gyroscope's biases, and n_a and n_g are zero-mean Gaussian noise.

Commercial IMU sensors often suffer from stochastic behaviour on their measurement errors. To best model the stochastic behaviour of scale factor and bias errors of the IMU, the first-order Gauss-Markov (GM) random processes are commonly used as follows [31]:

$$\dot{x} = -\frac{1}{\tau_x}x + \sqrt{\frac{2}{\tau_x}\sigma_x{}^2}n_x\,, \tag{2.26}$$

where x is the IMU error (e.g. gyroscope's or accelerometer's bias or scale factor), τ_x is a time constant, n_x is zero-mean Gaussian noise, and σ_x is the standard deviation of the GM process.

It is worth mentioning that an INS is a dead reckoning navigation system where its current estimates of navigation states are determined based on its previous states. Dead-reckoning navigation systems are prone to the accumulation of error, leading to the inevitable divergence of the navigation solution from the actual solution. The integration of the GNSS's absolute navigation solution with the INS results in a position-fix navigation system where the periodic fixed reference position and velocity updates of GNSS can correct these errors and reset the accumulation of errors. Integration of the visual navigation system, which is a dead-reckoning navigation system itself, with the INS does not solve the error accumulation; however, it slows down the divergence by bounding the error growth.

2.3.1 Inverse Kinematics

Given a navigation solution (i.e. position, velocity and attitude estimates), one can follow the dynamics equation in reverse and obtain pseudo-error-free acceleration and gyroscope measurements. However, to mimic real IMU measurements, all motion effects, such as

Coriolis effects, should be considered. Accelerometer measurements can be derived by differentiating the velocity and then including the gravity and Coriolis effects as follows [6]:

$$a_{iB}^{L} = \frac{dv_{LB}^{L}}{dt} \tag{2.27}$$

$$a_{iB}^{B} = R_{L}^{B}(a_{iB}^{L} - g^{L} + (2\omega_{iE}^{L} + \omega_{EL}^{L}) \times v_{LB}^{L}) \tag{2.28}$$

Similarly, Gyroscope measurements can be obtained by finding the difference between consecutive rotation matrices and dividing the result by the sampling period, and then including the Coriolis effects as follows [6]:

$$\dot{R_{B}^{L}} \cong \frac{R_{B}^{L}(k) - R_{B}^{L}(k-1)}{T} \cong R_{B}^{L}(k-1)([\omega_{iB}^{L}]_{\times}) \tag{2.29}$$

$$\omega_{iB}^{B} = R_{L}^{B}(\omega_{iB}^{L} + \omega_{iE}^{L} + \omega_{EL}^{L})) \tag{2.30}$$

2.4 IMU-GNSS Integration

Integration of GNSS information with INS can be performed at various levels: Loosely-Coupled, Tightly-Coupled, Deep- or Ultra-Tightly-Coupled. These levels are mainly different in the type of GNSS data that is involved in the fusion process [32][33]. In the Loosely-Coupled (LC) integration, the final 6-Element estimates of position and velocity obtained directly from a GNSS receiver build the measurement model of the fusion system. These types of measurements provide direct observability on the system's states. These updates are available when the receiver observes at least four satellites. However, in the Tightly-Coupled (TC) integration, raw measurements of the GNSS receiver from the individual satellites are utilized. These raw measurements include pseudo-range and pseudo-range-rate, which are nonlinearly related to the system's states. The advantage of

this technique over LC integration is that even a single visible satellite can contribute to state correction. In urban canyons with poor coverage (less than four visible satellites), the TC integration bound the diverges rate of the solution and potentially provide a more accurate solution compared to LC integration [34]. In the Ultra-Tightly-Coupled integration, all visible satellites are tracked individually, and advanced signal processing techniques at a very low level of GNSS electromagnetic signals are utilized in the fusion engine. This type of integration method is not available in commercial GNSS receivers. The GNSS measurement models for LC and TC integration systems are reviewed in more detail in the following sub-sections.

2.4.1 IMU-GNSS LC Integration

Figure 2. 3 shows the block diagram of the IMU-GNSS LC integration system. Here, the IMU measurements are 3-axis accelerometer and 3-axis gyroscope measurements. The GNSS measurements are 3-Element position and 3-Element velocity.

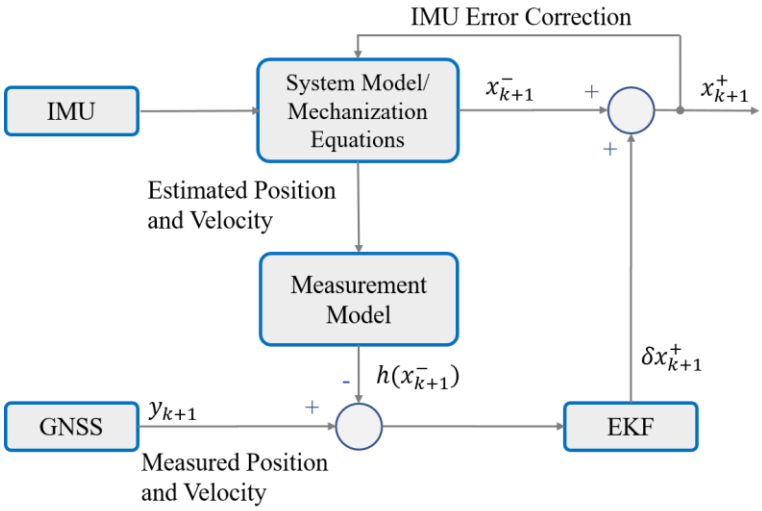


Figure 2. 3. IMU-GNSS LC integration block diagram.

In the formulation of the IMU-GNSS LC integration in this thesis, the 22-element state vector is defined as follows:

$$x = [(P_{LB}^{L})_{1\times3}, (v_{LB}^{L})_{1\times3}, (q_{B}^{L})_{1\times4}, (b_g)_{1\times3}, (b_a)_{1\times3}, (s_g)_{1\times3}, (s_a)_{1\times3}\ ]_{22\times1}, \tag{2.31}$$

where P_{LB}^{L}, v_{LB}^{L} , and q_{B}^{L} are respectively, position, velocity, and attitude of the $\{B\}$ frame with respect to the $\{L\}$ frame. b_a and b_g are accelerometer's and gyroscope's biases, and s_a and s_g are accelerometer's and gyroscope's scale factors, respectively.

The measurement model in the IMU-GNSS LC integration system is linear since the position and velocity measurements are directly received from the GNSS receiver.

$$y(t) = h\big(x(t)\big) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}_{6\times22} \times x_{22\times1} \tag{2.32}$$

2.4.2 IMU-GNSS TC Integration

Figure 2. 4 shows the block diagram of the IMU-GNSS TC integration system. The measurements from the GNSS receiver are raw pseudo-range, ρ, and pseudo-range-rate, $\dot{\rho}$. Therefore, the measurement model is no longer the direct observation of navigation states; instead, it is a nonlinear function of navigation states.

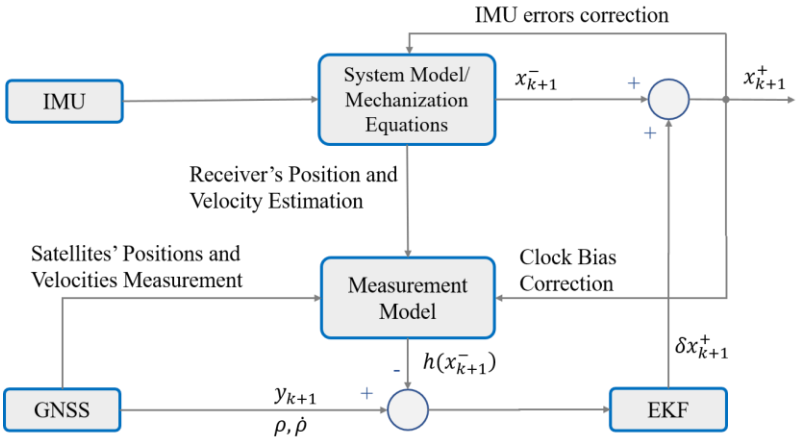


Figure 2. 4. IMU-GNSS TC integration block diagram.

The pseudo-range from the m^{th} satellite to the GNSS receiver is given by the time it takes a navigation message to travel from the satellite to the receiver:

$$\rho^m = \mathcal{C}(t_R - t_T), \tag{2.33}$$

where ρ^m is the pseudo-range from the m^{th} satellite, $\mathcal{C}$ is the speed of light, and t_R and t_T are, respectively, the arrival and transmission times. The major source of error in measuring the pseudo-range is the receiver clock bias. To account for this error, the previous equation would change as follows:

$$\rho^m = r^m + \mathcal{C}\delta t_r + v_\rho^m \tag{2.34}$$

$$r^m = \|P_E - P_E^m\| = \sqrt{(x_E - x_E^m)^2 + (y_E - y_E^m)^2 + (z_E - z_E^m)^2} \tag{2.35}$$

where r^m is the true range, δt_u is the receiver clock bias, and v_ρ is random zero-mean Gaussian noise. P_E and P_E^m are, respectively, the positions of the receiver and the m^{th} satellite in the $\{E\}$ frame.

The Doppler Effect is the other quantity that is often used in IMU-GNSS TC integration. It reveals the relative velocity of the satellite and the receiver along their line of sight. The pseudo-range-rate measurement can be defined as follow:

$$\dot{\rho}^m = -\frac{D^m \mathcal{C}}{f_1} + \mathcal{C}\delta\dot{t_r} + w_{\dot{\rho}}^m \tag{2.36}$$

$$D^m = \frac{\{(v_E^m - v_E).\mathbf{1}^m\}f}{\mathcal{C}} \tag{2.37}$$

$$\mathbf{1}^m = \frac{[(x_E - x_E^m), \quad (y_E - y_E^m), \quad (z_E - z_E^m)]^T}{\sqrt{(x_E - x_E^m)^2 + (y_E - y_E^m)^2 + (z_E - z_E^m)^2}} \tag{2.38}$$

where $\dot{\rho}^m$ is the pseudo-range-rate between the m^{th} satellite and the receiver, f is the carrier frequency, $\delta\dot{t_u}$ is the receiver clock drift, and $w_{\dot{\rho}}$ is random zero-mean Gaussian noise. D^m is the Doppler frequency shift, v_E and v_E^m are, respectively, the velocities of the receiver and the m^{th} satellite in the $\{E\}$ frame, and $\mathbf{1}^m$ is the line of sight unit vector.

Equations (2.34) and (2.36) are nonlinear with respect to the navigation states (i.e. P_E and v_E). Hence the linearization of those equations leads to the measurement model as follows:

$$\begin{bmatrix} \delta\rho^1 \\ \vdots \\ \delta\rho^M \\ \delta\dot{\rho}^1 \\ \vdots \\ \delta\dot{\rho}^M \end{bmatrix} = \begin{bmatrix} (\mathbf{1}^1)^T & 0_{3\times1} & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ (\mathbf{1}^M)^T & 0_{3\times1} & 1 & 0 \\ 0_{3\times1} & (\mathbf{1}^1)^T & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0_{3\times1} & (\mathbf{1}^M)^T & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta P_E \\ \delta v_E \\ \delta \boldsymbol{b}_{\mathbb{c}} \\ \delta \boldsymbol{d}_{\mathbb{c}} \end{bmatrix} + \begin{bmatrix} v_\rho^1 \\ \vdots \\ v_\rho^M \\ w_{\dot{\rho}}^1 \\ \vdots \\ w_{\dot{\rho}}^M \end{bmatrix} \tag{2.39}$$

where $\delta\rho^m$and $\delta\dot{\rho}^m$ are, respectively, the pseudo-range and pseudo-range-rate errors of the m^{th} satellite. $\delta\boldsymbol{b}_{\mathbb{c}} = \mathcal{C}\delta t_r$ and $\delta\boldsymbol{d}_{\mathbb{c}} = \mathcal{C}\delta\dot{t_r}$ are the receiver clock bias and clock drift errors in the same unit as range and range rate, respectively.

Therefore, the state vector in the IMU-GNSS TC integration can be defined as follows:

$$x = [(P_{LB}^L)_{1\times3}, (v_{LB}^L)_{1\times3}, (q_B^L)_{1\times4}, (b_g)_{1\times3}, (b_a)_{1\times3}, (s_g)_{1\times3}, (s_a)_{1\times3}, \boldsymbol{b}_{\mathbb{c}}, \boldsymbol{d}_{\mathbb{c}}\,]_{24\times1}, \tag{2.40}$$

where P_{LB}^L, v_{LB}^L , and q_B^L are respectively, position, velocity, and attitude of the $\{B\}$ frame in the $\{L\}$ frame. b_a and b_g are accelerometer's and gyroscope's biases, and s_a and s_g are

accelerometer's and gyroscope's scale factors. $\boldsymbol{b}_{\mathbb{c}}$ and $\boldsymbol{d}_{\mathbb{c}}$ are respectively the clock bias and drift of the GNSS receiver.

Since the system model is already defined in the $\{L\}$ frame, the error in position and velocity states in (2.39) must be transferred from the $\{E\}$ frame to the $\{L\}$ frame. This transformation can be obtained by linearization on the following equations that relate the $\{E\}$ frame to the $\{L\}$ frame [6]:

$$x = (R_N + h)\cos(\varphi)\cos(\lambda) \tag{2.41}$$

$$y = (R_N + h)\cos(\varphi)\sin(\lambda) \tag{2.42}$$

$$z = [R_N(1 - e^2) + h]\sin(\varphi) \tag{2.43}$$

$$N = (\varphi - \varphi_0)(R_M + h) \tag{2.44}$$

$$E = (\lambda - \lambda_0)(R_N + h)\cos(\varphi) \tag{2.45}$$

$$D = -h \tag{2.46}$$

where $e = 0.0818119$ is the eccentricity of the ellipsoid model of the Earth [6].

2.5 Camera-IMU TC Integration in MSCKF Framework

Traditionally, the fusion of visual and inertial modalities for navigation purposes was approached in an LC integration scheme. In this scheme, visual and inertial systems are almost two separate navigation systems, and integration happens only on position and attitude estimates. In the LC integration scheme, visual features are only tracked over pairs of images (the current and previous images). Once enough pair-wise matches are collected (5 or 8 pairs [35][36]), a geometric constraint known as Epipolar constraint is employed to estimate the relative position and attitude of the camera at which it took those two images

[35]. Therefore, the measurement model of the LC integration fuses the 6-element relative pose (i.e. 3-element translation and 3-element rotation) obtained from the visual system with the pose estimates predicted by the INS. This fusion scheme enjoys low computational complexity; however, it ignores the fact that strong visual features (e.g. sharp edges or corners) can be tracked over more than two images. Therefore, in the LC integration scheme, the corrections are built upon weak constraints that have only limited impacts on position and attitude states and do not contribute to correct other system's states (e.g. IMU errors and camera calibration parameters).

On the other hand, in the visual-inertial TC integration, the measurement model directly engages the observations of the visual features (i.e. image coordinate of the tracked features) along with the estimate of the camera's poses when it observed the features. Integrating visual and inertial modalities at such low and raw levels harvest deeper correlations that potentially exist between these modalities. For instance, in estimating the camera pose, the pose estimates of the body frame and the camera-to-body relative pose states are also involved. Body frame pose estimate itself is affected by IMU errors (e.g. bias, scale factor, and mounting misalignment). Hence, the TC integration involves more system's states than just the camera's position and attitude, and therefore, more system's states receive corrections with visual updates [37]. Among the filtering-based visual-inertial TC integration algorithms, MSCKF VIN is one of the state-of-the-art VIN systems [19]. In the MSCKF, the geometrical constraints arising from observing visual features from multiple camera poses are directly involved in the measurement model, making it possible to have two significant advantages. First, MSCKF updates, in addition to the hidden states of the system, correct pose estimates in the past. Second, since multiple

observations of the visual features are involved in the measurement model, the resulting geometrical constraints are more constrained, and therefore, the corrections are more effective and robust. In fact, the leveraging point of the MSCKF lies in the Kalman updates that have a broader and more powerful influence on the states.

Compared to rival visual-inertial fusion systems that jointly estimates the navigation states as well as the 3D location of visual features as in EKF-based SLAM algorithms [38][39], the MSCKF VIN enjoys less computational complexity and better scales with respects to the number of tracked features because in SMCKF VIN, instead of augmenting the state vector with the 3D locations of tracked features', the poses of the camera from which visual features were observed are tracked in the state vector. For instance, in a scenario with 100 visual features tracked over 10 camera poses, the MSCKF augments the state vector with only 10 states; whereas, in EKF-based SLAM algorithms [38][39], 100 states are augmented in the state vector. Despite not tracking visual features' 3D location in the state vector, MSCKF VIN acknowledges the correlations between features' 3D location and multiple poses of the camera and consequently decorrelates the measurement model from featuress' 3D location before applying Kalman update [19].

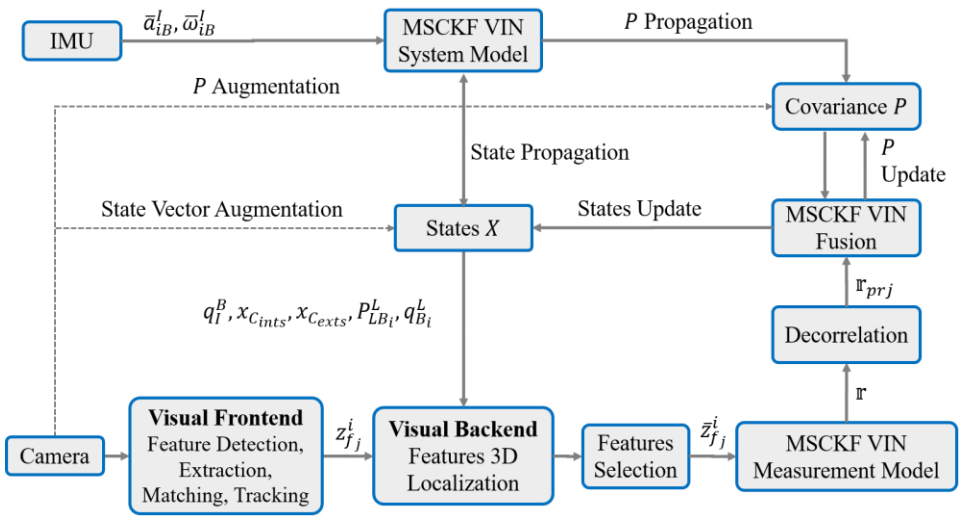


Figure 2. 5. Visual-Inertial TC integration in the MSCKF framework.

Figure 2. 5 shows the block diagram of the MSCKF visual-inertial fusion system and how its different modules work together. The IMU measurements are used to propagate the system's states through time. When a new image is captured, the visual frontend module detects salient visual features (e.g. edges or corners) and matches them with already tracked visual features from previous images. The inertially aided visual backend then localizes the tracked features in the 3D space using the most updated estimates of the system's states. The 3D localized features are then projected again on the observing images to obtain observation-prediction error (also called reprojection error). The MSCKF fusion engine then stacks the reprojection error of all 3D localized features and forms the Kalman residual, and applies the correction on all engaged system's states. In the MSCKF VIN system, one epoch of the filter involves state propagation followed by the visual update at the end. The following sub-sections discuss individual components of the MSCKF framework.

2.5.1 MSCKF State Vector

The state vector in the MSCKF formulation of the VIN problem is comprised of two parts: a fixed-length part and a variable-length part (also called the window part):

$$\boldsymbol{X} = [\boldsymbol{X}_{fixed} | \boldsymbol{X}_{window}]_{(37+7N)\times 1}, \tag{2.47}$$

$$\boldsymbol{X}_{fixed} = [P_{LB}^{L}, v_{LB}^{L}, q_{B}^{L}, b_g, b_a, s_g, s_a, q_{I}^{B}, x_{C_{ints}}, x_{C_{exts}}]_{37\times 1}, \tag{2.48}$$

$$\boldsymbol{X}_{window} = [P_{LB_1}^{L}, q_{B_1}^{L}, \dots, P_{LB_N}^{L}, q_{B_N}^{L}]_{7N\times 1}, \tag{2.49}$$

where P_{LB}^{L}, v_{LB}^{L}, and q_{B}^{L} are, respectively, the position, velocity, and attitude of the $\{B\}$ frame with respect to the $\{L\}$ frame. b_g and b_a are, respectively, biases of the gyroscope's and accelerometer's measurements, and s_g and s_a are, respectively, the scale factors of the gyroscope's and accelerometer's measurements. Compared to the original formulation of the VIN problem in the MSCKF framework reported in [19], in this research, the state vector has been augmented with 1) the IMU misalignment correction quaternion, q_{I}^{B}, 2) the camera's intrinsic parameters (i.e. the focal length and center of the image), $x_{C_{ints}} = [f_u, f_v, o_u, o_v]$, and 3) the camera's extrinsic parameters (i.e. the relative pose between the $\{C\}$ and $\{B\}$ frames), $x_{C_{exts}} = [q_{C}^{B}, P_{BC}^{B}]$. Hence, in the rest of the thesis the term "enhanced MSCKF VIN" is used to distinguish the MSCKF VIN system developed in this research from the conventional MSCKF VIN system with the reduced state vector originally introduced in [19].

The window part of the state vector holds a history of N recent poses of the $\{B\}$ frame from which the camera observed the currently being tracked visual features. In the system model of the MSCKF, no dynamics is assumed for q_{I}^{B}, $x_{C_{ints}}$, and $x_{C_{exts}}$ in the fixed-length part of the state vector. The past pose states in the window part are also static.

It is worth discussing that compared to the conventional version of the MSCKF VIN introduced in [19], augmenting the state vector with IMU and Camera calibration states in the enhanced MSCKF VIN proposed in this work, necessitates consequent modifications in the fusion framework of Figure 2. 5. The system model and the corresponding noise covariance matrix must be modified to incorporate the dynamic and stochastic impact of the augmented states in the state and covariance matrix propagation. The visual frontend module remains intact as it independently processes the raw images; however, the visual backend in the enhanced MSCKF VIN now must involve IMU and Camera calibration states in the features localization task and must always employ the most recent estimates of these states in each epoch. The measurement model and update step undergo the most modifications to tightly integrate all the states in the augmented state vector with visual features observations. In the following sub-sections, these modifications will be discussed in further detail.

2.5.2 MSCKF State and Covariance Matrix Augmentation and Contraction

When a new image is captured, the variable-length part of the state vector is augmented with a copy of the current estimate of the body frame pose. The covariance matrix, P, accordingly undergoes the same augmentation process at the associated rows and columns as follows:

$$P_{(37+7(N+1))} \leftarrow \begin{bmatrix} & I_{37+7N} & \\ I_{3\times3} & 0_{3\times3} \quad 0_{3\times4} & 0_{27+7N} \\ 0_{4\times3} & 0_{4\times3} \quad I_{4\times4} & \end{bmatrix} P_{(37+7(N))} \begin{bmatrix} & I_{37+7N} & \\ I_{3\times3} & 0_{3\times3} \quad 0_{3\times4} & 0_{27+7N} \\ 0_{4\times3} & 0_{4\times3} \quad I_{4\times4} & \end{bmatrix}^T \tag{2.50}$$

This augmentation must be done because these pose states and their corresponding covariance estimates are tied with the viewpoints of tracked visual features and are involved in the MSCKF measurement model and update step.

The visual features lost in the current image, but were observed in more than four previous images, are localized, and then used in the MSCKF update. When all features that were initially detected in a particular image are lost, there is no need to keep the corresponding pose states in the variable-length part of the state vector; hence, the state vector and the covariance matrix contract to exclude the redundant entries. Contraction is performed by removing the associated entries of the state vector and covariance matrix. This contraction must be done to prevent the state vector and covariance matrix from growing unnecessarily.

2.5.3 MSCKF Visual Frontend

When the camera captures a new image, first, several salient visual features such as edges and corners are detected in the image. There are various feature detector algorithms in the literature, each having its own merits and drawbacks. The most common feature detectors are Scale-Invariant Feature Transform (SIFT) [40], Speed Up Robust Feature (SURF) [41], Features from Accelerated Segment Test (FAST) [42], Oriented FAST and Rotated BRIEF (ORB) [43]. Readers are referred to [44] for a survey of different features. In this work, SURF and ORB feature detectors have been used. The feature detector provides a list of 2D image coordinates of the detected edges and corners. These features are then assigned a descriptor extracted from their surrounding local pattern of pixels. These descriptors are utilized to be matched with those of features detected in previous images. Features that are successfully matched are recorded to be tracked in future images. Features that are not

matched are called lost features. Lost features tracked for less than four images are discarded, but those tracked in more than four images are localized and then used in the MSCKF update that will be discussed in the coming sub-section.

2.5.4 MSCKF Visual Backend

The MSCKF VIN system directly integrates the observation of visual features in the measurement model. The observation of a single feature, f_j, on the i^{th} images can be defined with the camera's pinhole projection model as follows [45]:

$$\begin{aligned} z_{f_j}^i = \boldsymbol{u}_{f_j}^i \left(u_{f_j}^i, v_{f_j}^i\right) &= \boldsymbol{h}\left(X_{C_i f_j}^{Ci}, Y_{C_i f_j}^{Ci}, Z_{C_i f_j}^{Ci}\right) + n_{z_j^i} \\ &= \begin{bmatrix} o_u \\ o_v \end{bmatrix} + \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \begin{bmatrix} X_{C_i f_j}^{Ci} / Z_{C_i f_j}^{Ci} \\ Y_{f_j}^{Ci} / Z_{C_i f_j}^{Ci} \end{bmatrix} + n_{z_j^i}, \end{aligned} \tag{2.51}$$

Where $z_{f_j}^i = \boldsymbol{u}_{f_j}^i \left(u_{f_j}^i, v_{f_j}^i\right)$ is the image 2D coordinate of the j^{th} feature as observed on the i^{th} image, $\boldsymbol{h}(.)$ is the camera's pinhole projection model, $P_{C_i f_j}^{C_i}\left(X_{C_i f_j}^{Ci}, Y_{C_i f_j}^{Ci}, Z_{C_i f_j}^{Ci}\right)$ is the 3D camera coordinate of the j^{th} 3D feature point as observed in the i^{th} camera frame. o_u, o_v and f_u, f_v are, respectively, the image center and focal length of the camera normalized by the image sensor size along the image frame axes. $n_{z_j^i}$ is the random zero-mean Gaussian noise contaminating the observation of features on the image.

In the MSCKF measurement model, it is often required to transform features' coordinates (2D image coordinates and 3D camera coordinate) among viewpoints of the camera. Given the relative pose between two camera poses, $\left(P_{C_k C_i}^{C_k},\ R_{C_i}^{C_k}\right)$, the camera coordinate of a 3D feature point in one camera pose can be transformed to the other camera pose using the following transformation:

$$P_{C_k f_j}^{C_k} = R_{C_i}^{C_k} P_{C_i f_j}^{C_i} + P_{C_k C_i}^{C_k} \tag{2.52}$$

The above equation can be rewritten in a new form as follows:

$$P_{C_k f_j}^{C_k} = Z_{f_j}^{i} \boldsymbol{g_k} \begin{pmatrix} \alpha_j^i \\ \beta_j^i \\ \rho_j^i \end{pmatrix} = Z_{f_j}^{i} \left(R_{C_i}^{C_k} \begin{bmatrix} \alpha_j^i \\ \beta_j^i \\ 1 \end{bmatrix} + \rho_j^i P_{C_k C_i}^{C_k} \right) \tag{2.53}$$

$$\alpha_j^i = \frac{X_{C_i f_j}^{C_i}}{Z_{C_i f_j}^{C_i}} \ , \ \beta_j^i = \frac{Y_{C_i f_j}^{C_i}}{Z_{C_i f_j}^{C_i}} \ , \rho_j^i = \frac{1}{Z_{C_i f_j}^{C_i}} \tag{2.54}$$

This new form is known as Inverse Depth Parameterization (IDP) [46]. It has been shown that the IDP form enjoys more stability when used in the optimization step for features localization that will be discussed next [46].

As shown in Figure 2. 6, tracking a visual feature on multiple images provides sufficient geometrical constraints that, if combined into an optimization problem, can lead to 3D localization of that feature in the camera frame.

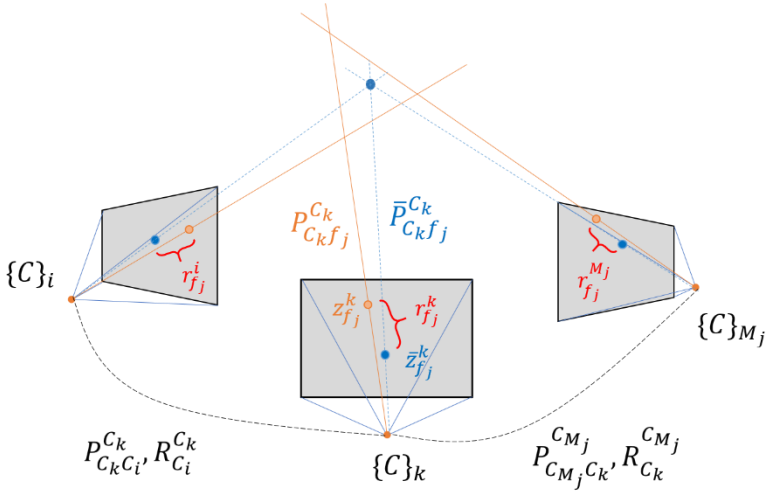


Figure 2. 6. Observing a single feature from multiple viewpoints of the camera.

The objective function used for this localization task is considered the sum of reprojection error squared as follows:

$$\boldsymbol{\mathcal{F}}\left(\theta_j^i\right) = \sum_{k=1}^{M_j} {r_{f_j}^k}^2 = \sum_{k=1}^{M_j} (z_{f_j}^k - \bar{z}_{f_j}^k)^2 = \sum_{k=1}^{M_j} (z_{f_j}^k - \boldsymbol{h}\left(\boldsymbol{g_k}\left(\theta_j^i\right)\right))^2, \tag{2.55}$$

where $r_{f_j}^k$ is called reprojection error, $z_{f_j}^k$ is the actual observation of the j^{th} feature on the k^{th} image frame, $\theta_j^i = (\alpha_j^i, \beta_j^i, \rho_j^i)$ is the candidate solution of this optimization problem in the IDP form. $\bar{z}_{f_j}^k = \boldsymbol{h}\left(\boldsymbol{g_k}\left(\theta_j^i\right)\right)$ transforms the j^{th} 3D feature point from the i^{th} camera frame and project it on the k^{th} image frame.

Conventionally this minimization problem has been solved by iterative Gauss-Newton (GN) algorithm as follows [47]:

$$\bar{\theta}_j^{i(s+1)} = \bar{\theta}_j^{i(s)} - (J_{\boldsymbol{r}_{f_j}}{}^T J_{\boldsymbol{r}_{f_j}})^{-1} J_{\boldsymbol{r}_{f_j}}{}^T \boldsymbol{r}_{f_j}{}^{(s)}, \tag{2.56}$$

$$J_{\boldsymbol{r}_{f_j}} = \frac{\partial \boldsymbol{r}_{f_j}}{\partial \theta_j^i} \tag{2.57}$$

$$\boldsymbol{r}_{f_j} = \begin{bmatrix} r_{f_j}^1 & \dots & r_{f_j}^{M_j} \end{bmatrix}^T, \tag{2.58}$$

where the superscript s represent the iteration index of the GN method, $\boldsymbol{r}_{f_j}$ is a $2M_j \times 1$ vector containing all partial residuals on all involved images, $J_{\boldsymbol{r}_{f_j}}$ is a $2M_j \times 3$ Jacobian matrix of the reprojection error vector, $\boldsymbol{r}_{f_j}$, with respect to its parameters θ_j^i.

2.5.5 Feature Selection

Not all tracked features carry useful information for state correction. Poor features can be those that were incorrectly matched or those features that are in the distance and do not carry valuable information about the ego-motion of the camera. Therefore, to avoid processing these features, a feature rejection policy should be adopted. In the MSCKF VIN system in this research, two feature rejection techniques have been adopted. The first

feature rejection happens after feature 3D localization with the GN optimization algorithm, and the second technique is a statistic technique.

In addition to the optimal solution, the GN optimization algorithm can also provide the uncertainty of the proposed solution. If $\hat{\theta}_j^i$ in the optimal solution to the optimization problem in (2.55), the covariance of this solution can be estimated as follows [48]:

$$cov\left(\hat{\theta}_j^i\right) = \left(\sigma_{r_j}^i\right)^2 \left(H_{\boldsymbol{F}}\left(\hat{\theta}_j^i\right)\right)^{-1} \approx \frac{\boldsymbol{r}_{f_j}^T \boldsymbol{r}_{f_j}}{2M_j} \left(J_r(\hat{\theta}_j^i)^T J_r(\hat{\theta}_j^i)\right)^{-1} \tag{2.59}$$

where $\sigma_{r_j}^i$ is the standard deviation of the jth feature's residuals, and can be approximated by the sum of squared partial residuals divided by $2M_j$. The term, $H_{\boldsymbol{F}}(\theta_j^{i^*}) = \nabla^2 \boldsymbol{F}(\theta_j^{i^*})$ is the Hessian matrix which in the GN optimization algorithm, is approximated by $J_r(\theta_j^{i^*})^T J_r(\theta_j^{i^*})$, where $J_r\left(\theta_j^i\right) = {\partial \boldsymbol{r}_{f_j}}/{\partial \theta_j^i}$ is the Jacobian of the partial residuals with respect to the optimization parameters. The resulting covariance is in the IDP form.

From the covariance law, the covariance of the feature's camera coordinate can be derived as follows [49]:

$$cov\left(\hat{P}_{C_i f_j}^{C_i}\right) = J_\theta \, cov\left(\hat{\theta}_j^i\right) J_\theta^T, \tag{2.60}$$

where J_θ is the Jacobian of θ_j^i with respect to $\hat{P}_{C_i f_j}^{C_i}$. The Covariance reveals the uncertainty in the 3D localization of the feature and can be used as a feature rejection criterion. For instance, if the camera poses observing a distant feature are not sufficiently far apart from each other, then the depth of the feature point cannot be estimated with high confidence, and therefore, the $cov(\hat{Z}_{C_i f_j}^{C_i})$ would be large. These features carry small information about the camera's ego-motion and are better to be rejected to avoid redundant processing in the Update step of the MSCKF.

In addition to feature rejection using GN covariance, there is a need for another feature rejection step for those features that are incorrectly matched over images. The residual in (2.58) is already the difference between actual and predicted observation. When the residual of a feature point is large, there is a large discrepancy between the prediction/hypothesis and the actual observations, and therefore, that feature is a potential outlier. The Chi-square test is a common statistical technique to detect these anomalies between prediction and observation to identify the outliers [50]. Chi-square test provides a single value, γ , that measures the difference between the observations and the hypothesis:

$$\gamma_j = \boldsymbol{r}_{f_j} \sigma_{\boldsymbol{r}_{f_j}} \boldsymbol{r}_{f_j}^{T}, \tag{2.61}$$

where $\boldsymbol{r}_{f_j}$ is the residual vector as in (2.58) and $\sigma_{\boldsymbol{r}_{f_j}}$ is the residual covariance. In the Chi-square test of the j^{th} feature point that has been observed over M_j images, the γ_j in (2.61) is statistically normalized by the number of observations, M_j, (also called the Chi-square test's degrees of freedom) to account for the number of observations in measuring the hypothesis-observations distance.

2.5.6 MSCKF Measurement Model and Update

When features are localized in the 3D space, they will be projected again on all images in which they were initially detected to obtain the reprojection errors. In the measurement model of the MSCKF, the reprojection errors of all localized visual features on all images are then stacked in a residual vector, $\mathbb{r}$, as follows:

$$\mathbb{r} = [\boldsymbol{r}_{f_1} \quad \cdots \quad \boldsymbol{r}_{f_F}]^T \tag{2.62}$$

$$\boldsymbol{r}_{f_j} = \left[r_{f_j}^1, \ldots, \quad r_{f_j}^k, \ldots \quad r_{f_j}^{M_j}\right]^T \tag{2.63}$$

$$r_{f_j}^k = z_{f_j}^k - \hat{z}_{f_j}^k \tag{2.64}$$

where $\mathbb{r}$ is the residual vector stacking all partial residuals of all F features selected to be involved in the MSCKF update step. $r_{f_j}^k$ is the distance between the actual observation of the j^{th}feature and its estimated observation obtained from projecting the 3D localized feature on the k^{th} image.

Given $\hat{\theta}_j^i$ as the solution to the optimization problem, $\hat{z}_{f_j}^k = \boldsymbol{h}\left(\boldsymbol{g_k}\left(\theta_j^i\right)\right)$ can be expanded in the following form that is more readable in terms of states that were already defined in the state vector in (2.47).

$$\hat{z}_{f_j}^k = \boldsymbol{h}\left(\hat{P}_{C_k f_j}^{C_k}\right) = \boldsymbol{h}(R_C^{B\,T}(R_{B_k}^{L\;T}(\hat{P}_{Lf_j}^L - P_{LB_k}^L) - P_{BC}^B)) \tag{2.65}$$

However, adopting this formulation in the partial residual calculation is problematic because the residual will then have dependencies on the features' location in the $\{L\}$ frame, $P_{Lf_j}^L$, that does not exist in the state vector already defined in (2.47):

$$\mathbb{r} = H_{\boldsymbol{X}}\delta\boldsymbol{X} + H_{\boldsymbol{f_j}}\delta\boldsymbol{P}_{Lf_j}^L + n_{\mathbb{r}} \tag{2.66}$$

Here, $H_{\boldsymbol{X}}$ is the Jacobian of the stacked residual vector with respect to the states defined in (2.47) and $H_{\boldsymbol{f_j}}$ is the Jacobian of the stacked residual vector with respect to the features' location in the $\{L\}$ frame. However, to be able to apply the Kalman update, the stacked residual vector, $\mathbb{r}$, should be decorrelated from $P_{Lf_j}^L$. This decorrelation can be done by projecting $\mathbb{r}$ and $H_{\boldsymbol{X}}$ in the above equation to the null space of the $H_{\boldsymbol{f_j}}$ using the method explained in [19]. The resulting measurement equation would then be in the following form that is independent of $\boldsymbol{P}_{Lf_j}^L$ and can be utilized in the MSCKF update:

$$\mathbb{r}_{prj} = H_{\boldsymbol{X}_{prj}}\delta\boldsymbol{X} + n_{\mathbb{r}} \tag{2.67}$$

where

$$\mathbb{r}_{prj} = null\left(H_{f_j}\right)^T \mathbb{r},$$
$$H_{X_{prj}} = null\left(H_{f_j}\right)^T H_X \tag{2.68}$$
$$null\left(H_{f_j}\right)^T H_{f_j} = 0$$

2.6 Deep Neural Networks

Recent researches in the field of Artificial Intelligence (AI) and Deep Neural Networks (DNNs) have shown that it is possible to harvest a large amount of data to learn complex patterns [51][52]. In the deep learning community, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are two types of DNNs that are powerful in extracting the spatial and temporal patterns in the data, respectively. In this section, CNNs and RNNs are reviewed.

2.6.1 Convolutional Neural Networks

CNNs are types of Artificial Neural Networks (ANNs) capable of extracting spatial patterns in visual images. CNNs are inspired by traditional computer vision techniques; however, the main difference is that filters/kernels in traditional computer vision techniques are hand-engineered, but in CNNs, these filters/kernels are the neural network's parameters and are learned through training [53]. The architecture of CNNs is inspired by the neurons' local and hierarchical connectivity in the human brain's visual cortex. Neurons have a restricted visual field, and for any particular visual stimuli (e.g. colour, edge, corner, texture, pattern), only a subset of neurons respond. These restricted fields of response overlap and cover the whole field of view. A typical deep CNN comprises several

convolutional and pooling layers followed by fully-connected layers at the output. Figure 2. 7 graphically demonstrates the pooling and convolutional layers in CNNs.

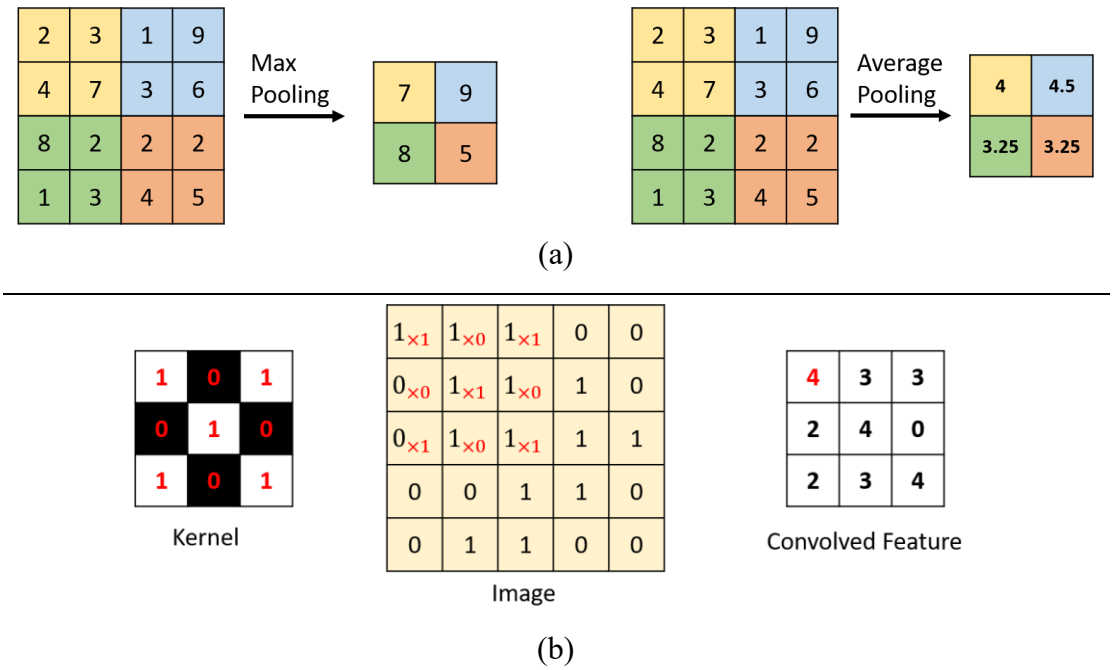


Figure 2. 7. (a) Pooling layers (e.g. Max Pooling and Average Pooling) in CNN. (b) Binary convolutional layers in CNN.

Figure 2. 8 shows a typical deep CNN for a hand-written 0-9 digit classification problem [54]. A typical convolutional layer is comprised of various filters/kernels of a particular size. Each filter or kernel extracts a particular pattern or feature (e.g. edges of different angles, blobs, corners) from the previous layer. For instance, the kernel in Figure 2. 7 extracts X-shape pixel patches in the image. Convolutional layers are often followed by a pooling layer to subsample the features of the previous layer. Features are often flattened at the final layer before being fed to fully-connected layers to predict or estimate the output for classification or regression problems.

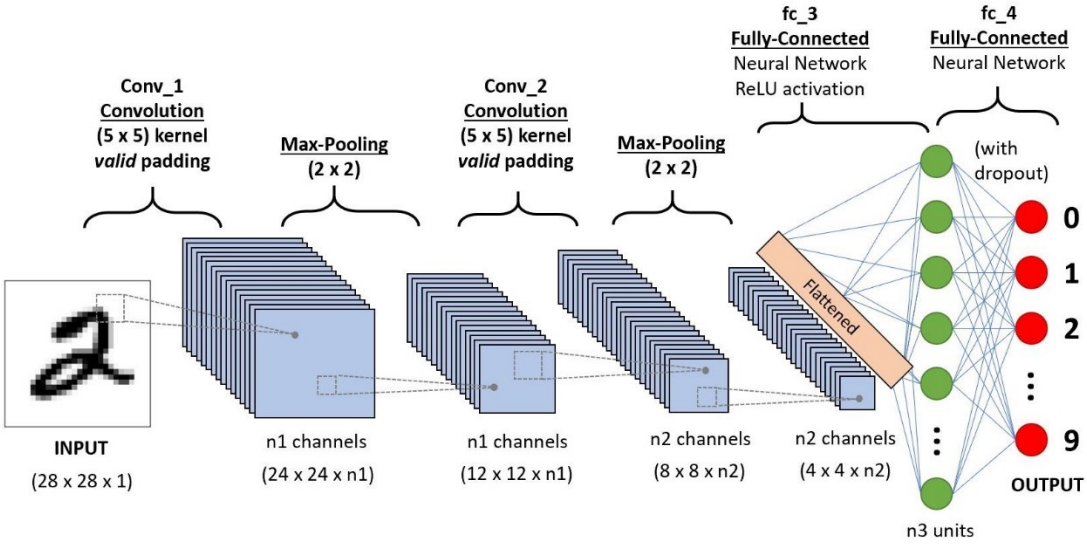


Figure 2. 8. An overview of a typical deep CNN for hand-written 0-9 digit classification problem [54]**.**

2.6.2 Recurrent Neural Networks

RNNs are types of ANNs capable of extracting temporal patterns in a sequence of data. RNNs have this capability because of the memory-like property resulting from passing the network's states from one time step to the next. Figure 2. 9 shows the RNN cells in two forms of rolled and unrolled.

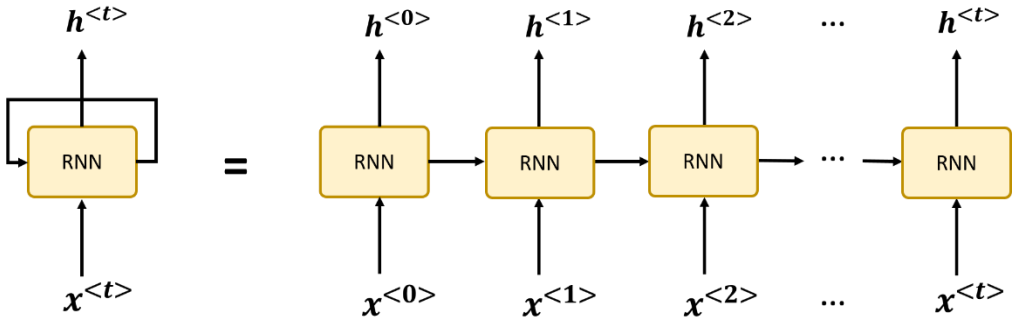


Figure 2. 9. Rolled and unrolled form of RNN. Hidden states are feedbacked in the rolled form or feedforwarded in the unrolled form.

One of the most popular variants of RNNs is Long Short Term Memory (LSTM) which introduces two gates called Forgetting and Updating gates that make an RNN better extract short- and long-term dependencies in sequences [55][56]. The feedback or hidden state feedforwarding in LSTM plays a vital role in its memory-like property. Each LSTM cell, in addition to the current information, processes the incoming information from the past to inference about the current output and the outgoing information to be used in the future cells.

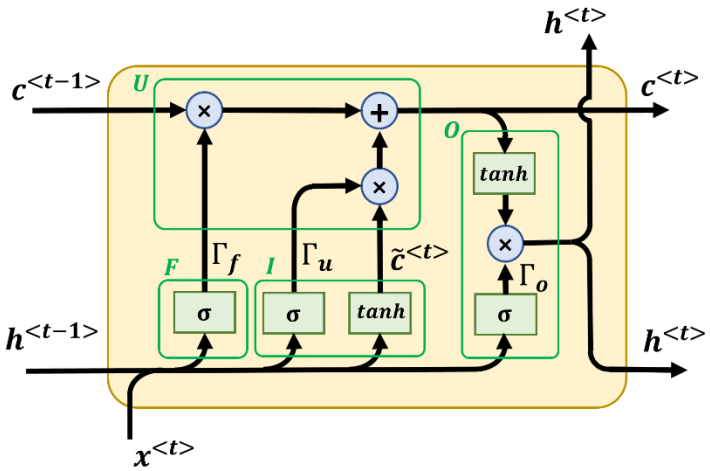


Figure 2. 10. The internal structure of an LSTM cell. σ, $tanh$ blocks represent respectively, $sigmoid$ and $tanh$ activation functions. Addition and multiplication modules are elementwise.

As shown in Figure 2. 10, each LSTM cell is comprised of four interacting gates named Input (I), Forget (F), Update (U), and Output (O) gates. The Forget gate takes the input ($x^{<t>}$) and the incoming hidden state ($h^{<t-1>}$), and then determines which elements of the incoming cell state ($c^{<t-1>}$), should be discarded. The Input gate receives the input ($x^{<t>}$) and incoming hidden state ($h^{<t-1>}$), and then determines which elements of the incoming cell state ($c^{<t-1>}$) should be updated with the new information. The Forget and Update processes happen in the Update gate by, first, element-wise multiplication with the

forgetting factor, Γ_f, followed by element-wise addition with the update factor Γ_u. Eventually, the Output gate, receiving the input ($x^{<t>}$), the incoming hidden state ($h^{<t-1>}$), and the outgoing cell states ($c^{<t>}$), determines the LSTM cell current output ($o^{<t>}$) and the outgoing hidden state ($h^{<t>}$). These processes are mathematically formulated as follows [55]:

$$\tilde{c}^{<t>} = tanh\ (W_c[h^{<t-1>}, x^{<t>}] + b_c) \tag{2.69}$$

$$\Gamma_f = \sigma(W_f[h^{<t-1>}, x^{<t>}] + b_f) \tag{2.70}$$

$$\Gamma_u = \sigma(W_u[h^{<t-1>}, x^{<t>}] + b_u) \tag{2.71}$$

$$\Gamma_o = \sigma(W_o[h^{<t-1>}, x^{<t>}] + b_o) \tag{2.72}$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \tag{2.73}$$

$$h^{<t>} = \Gamma_o * tanh\ (c^{<t>}) \tag{2.74}$$

2.7 Evolutionary Optimization Algorithms

Evolutionary optimization algorithms are a class of optimization techniques designed to avoid complications of classic optimization techniques. These difficulties often arise from the large size of the optimization problem or from the lack of an analytical formulation for the optimization problem. In these problems, classic optimization algorithms may not solve these problems in a reasonable amount of time or may fail to propose a solution at all. Different evolutionary optimization algorithms are employed in these problems when completeness, accuracy, or precision of the solution can be traded with the shortcoming of the solution proposed, if any, by the classic optimization algorithm. In the following two sub-sections, two powerful population-based evolutionary optimization algorithms, Genetic Algorithm and Particle Swarm Optimization, are reviewed. Interested readers are

referred to [57] for a detailed survey on biologically inspired optimization algorithms, including but not limited to Differential Evolution, Paddy Field, Artificial Bee Colony, Intelligent Water Drops, and Invasive Weed Colony optimization algorithms.

2.7.1 Genetic Algorithm

Genetic Algorithm (GA) is an evolutionary population-based search algorithm inspired by the principle of survival of the fittest and is one of the powerful optimization techniques [58]. The optimization process starts with an initial set of potential solutions. The next step is to measure the fitness of each solution and select the best solutions to involve them in the reproduction process of the next-generation solutions. The reproduction process is done using crossover or motion operators, which add variations to the solution to guarantee that the same population is not regenerated again. This process is repeated several times until the desired result is achieved or the maximum number of iterations is reached.

More formally, a potential solution in the GA is called a Chromosome which is characterized by its Genes which are the parameters/variables of the solution. In each generation, the set of all solutions is called Population. GA operators such as selection, mutation, and crossover are applied to one generation to produce the next generation population. The fitness function evaluates each solution and assigns it a fitness value. The fitness assigned to individual solutions determines which operators will affect them.

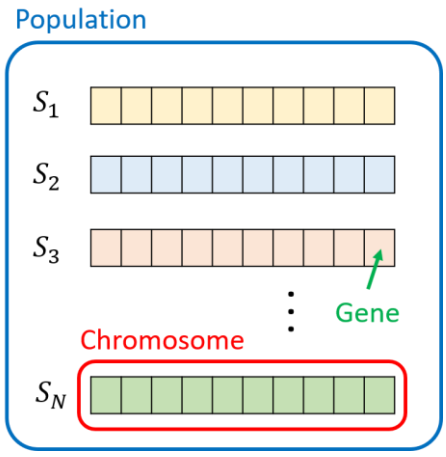


Figure 2. 11. Population, Chromosome, and Gene illustration in GA. S_i is the ith solution in the population.

The GA applies different operators to the population:

Selection operator__ selects the fittest solution and carries it over to the next generation intact to guarantee that the best solution in each generation is not worse than the previous generation's best solution.

Crossover operation__ is one of the most important operators in GA. Given two candidate chromosomes, the crossover operation exchanges a subset of genes in the two involved chromosomes. For instance, in a single-point crossover operation shown in Figure 2. 12, the parent chromosomes exchange their genes for producing a pair of new chromosomes. The crossover point is selected randomly, and in general, an n-point crossover is also applicable.

Mutation operation__ causes small changes in random genes of a chromosome. Depending on how the main optimization problem is coded to chromosomes' genes, the mutation can be considered as a change in the magnitude, sign, or angle depending on the applications.

Figure 2. 13 shows the flowchart of the GA.

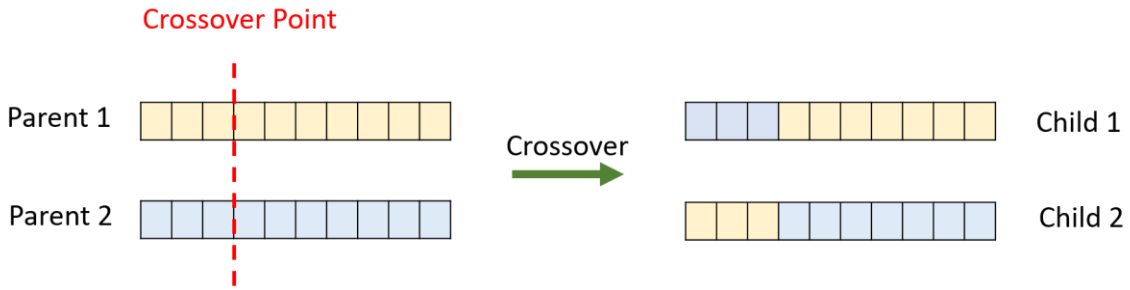


Figure 2. 12. Crossover operation in the GA.

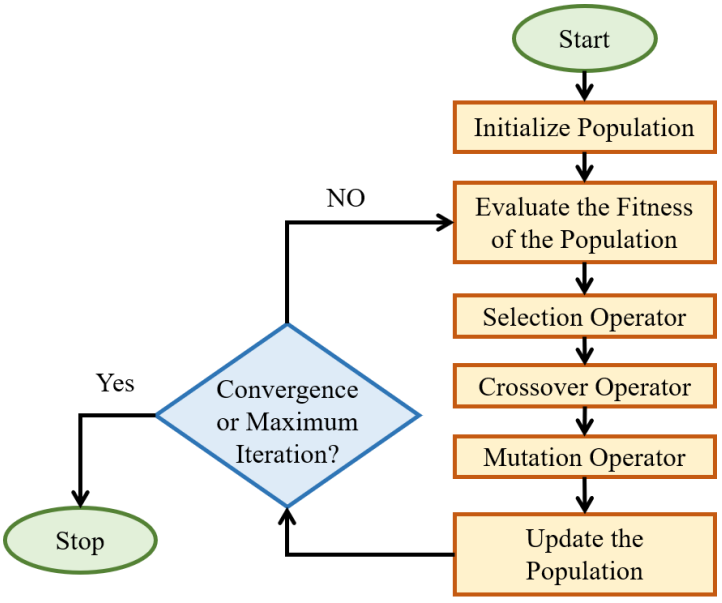


Figure 2. 13. GA flowchart.

2.7.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic population-based search algorithm inspired by the flock of birds and the school of fish foraging for food resources [59]. Similar to other population-based algorithms, PSO starts with distributing random solutions in the search space. Individual solutions are called particles, and together they form the swarm.

At each iteration/generation, particles are evaluated by a fitness function and are assigned a fitness value. At the end of each iteration, all particles are informed about the fittest particle in the swarm. To prepare for the next iteration, each particle then moves toward the fittest particle with momentum maintaining it on its initial search direction and on its personal best fitness so far. Figure 2. 14 demonstrates how the next location of the particle is influenced by the swarm intelligence and personal past experience. This process is repeated several times until the desired result is achieved or the maximum number of iterations is reached.

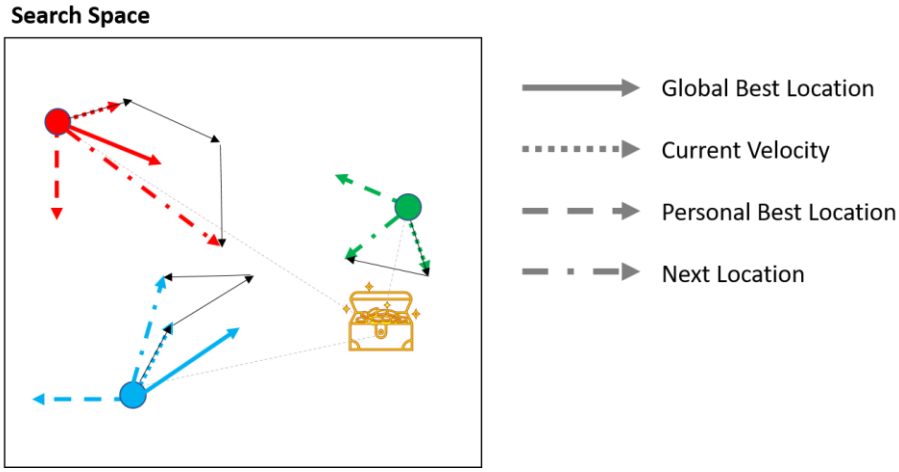


Figure 2. 14. PSO algorithm with three particles. The fitness in this example is the distance to the treasure. In this iteration, the green particle is the global/swarm's best particle.

Mathematically, the i^{th} particle is initially assigned a random position, $x_i(0)$, and velocity, $\mathbb{v}_i(0)$, in the search space. at the end of each iteration, particles update their position and velocity according to the following rules [59]:

$$\mathbb{v}_i(t+1) = m * \mathbb{v}_i(t) + c_1 * \mathfrak{r} * \left(\mathbb{x}_{Gbest} - \mathbb{x}_i(t)\right) + c_2 * \mathfrak{r} * \left(\mathbb{x}_{Pbest_i} - \mathbb{x}_i(t)\right), \tag{2.75}$$

$$\mathbb{x}_i(t+1) = \mathbb{x}_i(t) + \mathbb{v}_i(t+1), \tag{2.76}$$

where subscript i represent the i^{th} particle, m is the momentum or inertia, $\mathbb{x}_{Gbest}$ is the location of the global/population best particle, $\mathbb{x}_{Pbest_i}$ is the associated location of the so-far best fitness of the i^{th} particle. $\mathbf{r}$ is a random number in the range [0, 1]. c_1 and c_2 are the two coefficients adjusting the exploration-exploitation policy of the particles. Figure 2. 15 shows the flowchart of the PSO algorithm.

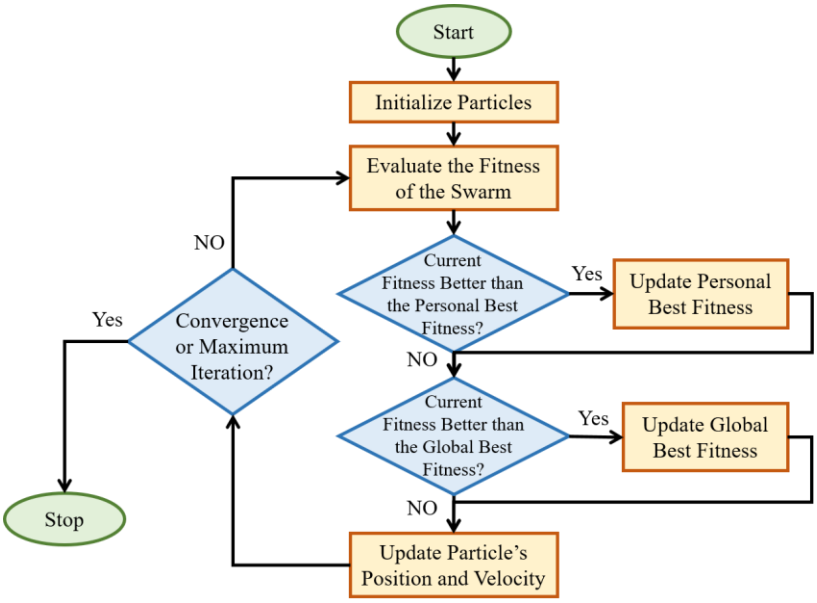


Figure 2. 15. PSO flowchart.

2.8 Embedded Computing Platform

Modern embedded computing platforms are often empowered by one or more processors, including Digital Signal Processors (DSPs), Central Processing Units (CPUs), Graphics Processing Units (GPUs), Application-Specific Integrated Circuits (ASICs), and Field Programmable Gate Arrays (FPGAs). Depending on the application, one or more of these processors might be preferred over the others. However, in robotics and vision-based navigation systems, the combination of CPUs and GPUs has become very popular because

of the great flexibility and computational power that comes at a reasonable cost and size [60][37].

2.8.1 GPU Versus CPU Architecture

GPUs were initially introduced for image rendering; however, their applications in parallel computing have become very popular among engineers and researchers. Nowadays, General Purpose GPUs are employed in various applications where the processing algorithm must apply repetitive operations on large-scale data. GPUs' computational power originates from their highly parallel architecture.

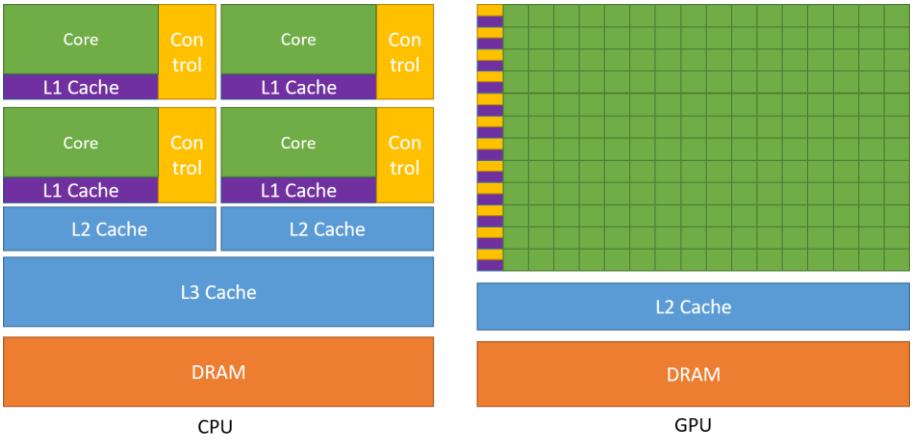


Figure 2. 16. CPU and GPU resources distribution comparison [61]**. GPU devotes more transistors to the arithmetic unit.**

Figure 2. 16 compares the architectural differences between CPUs and GPUs. As shown in the figure, GPUs devote more transistors to arithmetic than CPUs do; hence, GPUs provide higher instruction throughput than CPUs. From the application point of view, CPUs are optimized for serial and sequential processing; whereas, GPUs are optimized for parallel processing. However, leveraging the GPU's parallel architecture for faster processing is

application-dependent and relies on how parallelizable the computations are. Most applications involve both sequential and parallel computations; hence, CPU-GPU-enabled computing platforms are very popular.

In 2006, NVIDIA® introduced Compute Unified Device Architecture (CUDA®) as a programming model to efficiently leverage the computing power of its GPU's parallel architecture. CUDA® allows C++ developers to develop CUDA functions, called *kernels*, that, when called, are executed multiple times simultaneously by parallel CUDA *threads* (i.e. a sequence of operations). In CUDA®, threads are identified by their unique thread index, *threadIdx,* that can have 1, 2 or 3 dimensions for further scalability.

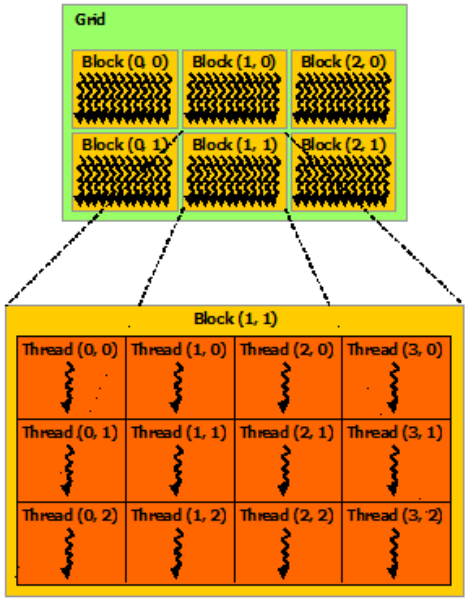


Figure 2. 17. Hierarchical organization of threads, blocks, and grids [61]**.**

As shown in Figure 2. 17, threads in a group of 32 are called *warps*. Threads form *blocks* that can be 1,2, or 3-dimensional and are identified by their block index, *blockIdx*. Since all threads in a block are executed on the same core and share memory resources, up to 1024 threads can be contained in a block. Thread blocks are organized into 1,2, or 3-dimensional *grids.* Parallel processors in NVIDIA GPUs are grouped into Streaming

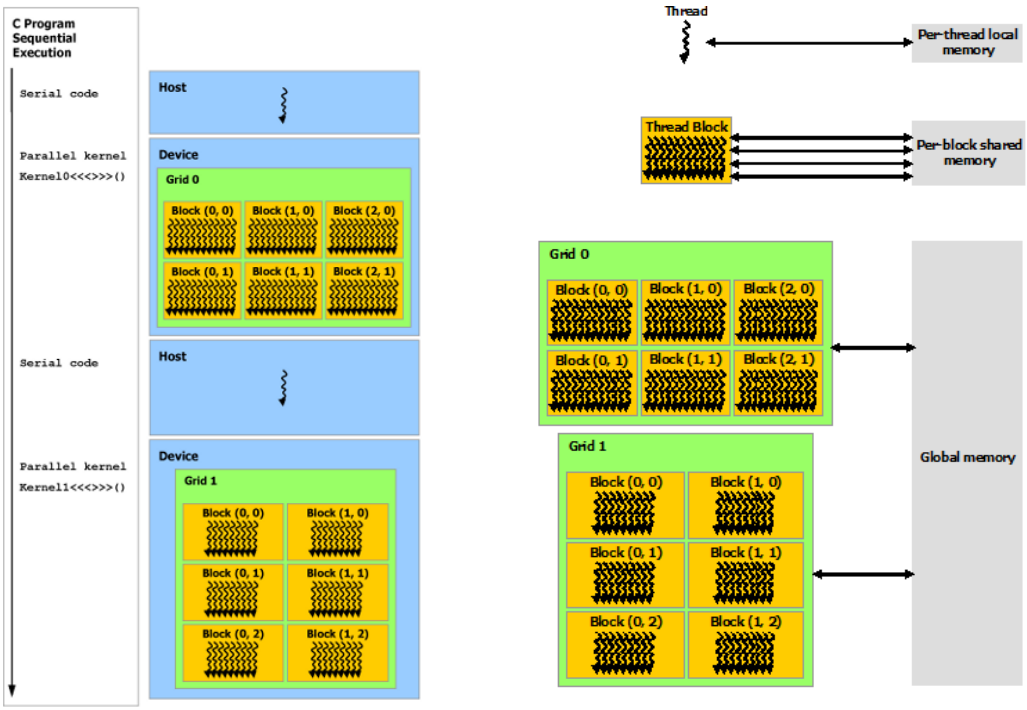
Processors (SPs) or CUDA® cores. CUDA cores are organized in Streaming Multiprocessors (SMs) that create, manage, schedule, and execute instructions for many threads in parallel.

The following snippet of code illustrates how element-wise addition of two arrays can be done in parallel in CUDA. The kernel, *VecAdd*, has been defined with `__global__` declaration specifier to specify that this kernel can be called from the CPU and executed on the GPU. This kernel has been invocated in the main function with one block and N threads per block using execution configuration syntax `<<<numBlocks, blockSize>>>`. Each thread then recovers its build-in index and performs a single add operation on one element of the array concurrently with other threads.

```
// Kernel definition
__global__ void __VecAdd(float* A, float* B, float* C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main() {
    // Kernel invocation with 1 block and N threads
    __VecAdd<<<1, N>>>(A, B, C);
}
```

In the CUDA programming model, threads are executed on the GPU (also called *device*). GPU is assumed to be a co-processor to the CPU (also called a *host*). Device and host have separate memory spaces called *device memory* and *host memory*, respectively. Each thread has a private local memory space. Threads within a block have access to a shared memory that is created with the thread block and is destroyed with it. All thread blocks have access to a shared global memory. Figure 2. 18 shows the memory hierarchy and programming model in a CPU-GPU environment.



(a) (b)

Figure 2. 18. (a) Host-Device programming model, (b) The memory hierarchy [61]**.**

2.8.2 NVIDIA® Jetson TX2 Embedded System

The embedded computing platform employed in this work to implement the multi-sensor logger system and VIN system has been the NVIDIA® Jetson TX2 development board [62] shown in Figure 2. 19. This embedded system is empowered by a dual-core Denver-2 CPU, Quad-core ARM® Cortex®-A57 Complex CPU, and NVIDIA® Pascal™ GPU. Detailed technical specifications are reported in Table 2. 2.

Figure 2. 19. The top view of the NVIDIA® Jetson TX2 development board [62]**.**

Table 2. 2. Technical Specifications of NVIDIA® Jetson TX2 [62]**.**

Specification	Details	Specification	Details
GPU	NVIDIA® Pascal™ Architecture with 256 CUDA® cores	**CPUs**	A Dual-core Denver-2 64-bit CPU and a Quad-core ARM® Cortex®-A57 complex
Memory	8GB 128-bit LPDDR4 1866MHz – 59.7 GB/s	**Storage**	32GB eMMC 5.1 Flash
Video Encoder/ Decoder	3x 4K @ 30 (HEVC) / 4x 4K @ 30 (HEVC)	**Camera**	5 MP Fixed Focus MIPI CSI
Connectivity	On-board 802.11ax Wi-Fi, Bluetooth, Ethernet	**IO**	USB 3.0, USB 2.0, HDMI, PCI-E x4, SD card, SATA, GPIOs, I2C, I2S, SPI, CAN, UART
Dimension	Module: 87mm×50mm, board: 17cm×17cm	**Power**	7.5W

Chapter 3: Automatic Tuning of the VIGN System

In Chapter 2, two multi-sensor navigation systems, IMU-GNSS and Visual-Inertial, were introduced. The state estimation in these fusion systems is performed in the EKF framework in relatively large state spaces (24 states in the IMU-GNSS TC integration, and (37+7N) states in the enhanced MSCKF Visual-Inertial integration). The involved states in each fusion system interact in a highly nonlinear form with other states that makes the derivation of the system and measurement models a tedious and challenging task. These sophisticated models often undergo linearization steps in the EKF framework that affect their accuracy around highly nonlinear linearization points. Moreover, to account for un-modelled behaviour and the inherent uncertainties of the fusion systems, the models are often parameterized by stochastic parameters (e.g. the initial covariance matrix P, the process noise covariance matrix, Q and measurement noise covariance matrix R). Therefore, designing an accurate and robust multi-sensor navigation system in practice heavily relies on how accurately these models describe the system's actual behaviour in real conditions. Adopting the TC integration scheme addresses the modelling accuracy to a great extent; however, it also adds extra parameters to the system that must be carefully adjusted. Tuning these stochastic parameters of the EKF using the information provided in the sensors' datasheets does not necessarily guarantee optimal performance in practice because testing conditions often differ considerably from controlled experiments performed by the manufacturers. The conventional iterative hand-tuning is also time-consuming and entails an expert's experience in a trial-and-error process. In this chapter, first, the EKF parameters tuning is reformulated in an optimization problem, and then an

automatic tuning technique based on evolutionary algorithms (i.e. GA and PSO) is proposed that offloads the navigation system designers from the tedious, time-consuming, and ad-hoc process of tuning the EKF stochastic parameters. The main motivation to employ evolutionary optimization algorithms are twofold: first, the optimal stochastic parameters of EKF lie in a considerably high-dimensional search space, and second, the function relating the stochastic parameters to the performance of the EKF is highly complicated. Therefore, solving this optimization problem with classic analytical optimization algorithms that require obtaining the derivative of the objective function with respect to high-dimensional tunable parameters is almost impossible; whereas, evolutionary algorithms solve these high-dimensional and complicated optimization problems with simpler computations. PSO and GA provide a simple yet powerful tool for users even with limited experience in optimization to solve complicated problems. Designing a PSO- or GA-based optimizer involves relatively straightforward and intuitive design choices such as population/generation size and new solution generation rules. Moreover, by adopting the Training-Validation-Test policy, the set of tuned parameters are selected based on their generalizability to avoid overfitting a particular training scenario.

This chapter is divided into two main parts. In the first part, a flexible design and simulation environment for the IMU-GNSS TC integration system is developed. This design environment employs the symbolic engine and inverse kinematics as two powerful tools to build system models and to verify the design procedure of the IMU-GNSS fusion system. This design environment then proposes a systematic tuning technique based on the GA to optimally determine the parameters of the EKF in the IMU-GNSS TC integration system.

In the second part of this chapter, the problem of tuning the enhanced MSKCF VIN system is considered. The proposed tuning technique adopts a similar evolutionary search algorithm based on the PSO algorithm. Compared to the challenges of tuning the parameters of the IMU-GNSS navigation system, tuning the enhanced MSCKF VIN system is more critical and sensitive because, first, VIN is a dead-reckoning navigation system that does not receive absolute position updates and, therefore, optimal operation of the Kalman filter is critical in long-term, and second, because parameters of the enhanced MSCKF VIN system lie in a larger search space.

Experimental results on real datasets have been provided to validate the efficacy and generalizability of the proposed tuning technique on both IMU-GNSS and the VIN systems.

3.1 Related works

In the literature, tuning the Kalman filter generally refers to the determination of values in three covariance matrices, process/system noise covariance matrix, Q, measurement noise covariance matrix, R, and the initial states error covariance matrix, P_0 [63]. Improper selection of these parameters in the worst case leads to filter instability or divergence, but in less critical cases, it leads to sub-optimal performance of the filter. For instance, the filter may perform poorly to correct a specific state because either an unnecessarily large value in the R matrix has given less credibility to an accurate measurement received recently or because an unrealistically small variance estimate at the corresponding element in the P matrix has made the filter biased toward giving more credibility to its current prediction rather than to a quality correction suggested by the measurement.

The conventional technique to tune these parameters is by hand. The filter designers often begin with some guided guesses from the sensors' datasheets, followed by an ad-hoc and time-consuming trial-and-error process of adjusting the parameters for a specific application. However, this technique performs poorly in real-world applications due to a large set of tunable parameters. Hence, in recent years, techniques based on artificial intelligence have become an alternative solution for optimal tuning. Considering the system's nonlinearity and a large number of tunable parameters, it is difficult, if not impossible, to apply gradient-based techniques. Depending on the availability of a ground truth data, tuning techniques are categorized as error-based or innovation/residual-based techniques. If an error measure such as Root Mean Squared Error (RMSE) of the estimations is available, then this measure can serve as the objective function or heuristics for the optimization/search algorithm; otherwise, the innovation sequence or the residual of the Kalman filter itself would be the alternative option [64].

In this section, a wide range of tuning techniques for EKF parameters are reviewed. In [65], the EKF performance optimization was reformulated as a regression problem. Two neural networks, Generalized Regression Neural Network (GRNN) and Regular Radial Basis Network (RBNN) [66], were proposed and compared in estimating the performance of the EKF as a nonlinear function of its noise parameters. In [67], a tuning technique based on Fuzzy logic [68] was proposed to modify the measurement noise covariance matrix, R, in an IMU-based attitude estimation system. In that work, the normalized measurements of the accelerometer and the gyroscope were used as fuzzy variables to adjust the entries of the measurement noise covariance matrix, R, online. However, the determination of the parameters of the fuzzy inference system (e.g. characteristics of the fuzzy sets and fuzzy

rules) remained an open problem. The authors of [69] considered an autonomous underwater vehicle navigation and approached the EKF tuning from innovation-based techniques. The authors of that work compared an approximation of the actual covariance of the innovation sequence with the theoretical definition of the covariance of the innovation sequence and then proposed a fuzzy inference system that uses the result of this comparison and suggests modifications in the noise covariance matrices. A neuro-fuzzy innovation-based technique for tuning the measurement noise covariance matrix, R, of an EKF-based SLAM system was proposed in [70]. In that work, it was recognized that the neuro-fuzzy system itself imposes some free parameters that need to be tuned. The authors of that work then proposed the PSO algorithm to adjust the parameters of the fuzzy inference system offline during the training phase of the neuro-fuzzy system. The PSO algorithm was employed for EKF tuning in target tracking application in [71], where potential choices for the elements of R and Q matrices were considered as particles' position, and the RMSE of estimated and actual states was used as the fitness function. From a different perspective than previous approaches, a bank of multiple Kalman filters estimating in parallel but with a different set of parameters was considered in failure detection in flight control application [72]. The adaptive behaviour of this technique lies where the state estimates obtained from all filters of the bank are combined with variable and adaptable weights to determine the final estimates. However, the computational burden of this approach scales with the number of Kalman filter in the bank and therefore, it is not practical in large state space problems.

To address the limitations mentioned above and to further enhance the EKF tuning, this work also proposes a novel flexible design environment for IMU-GNSS TC integration

systems. This design framework employs inverse kinematic, symbolic engine, and GA of MATLAB software to provide navigation system researchers and engineers with powerful tools to design IMU-GNSS navigation systems. This simulation environment is introduced in the next section.

3.2 IMU-GNSS Navigation System Tuning Framework

An overview of the developed flexible design and simulation environment for the IMU-GNSS TC navigation system is shown in Figure 3. 1. The developed simulation environment in MATLAB software and provides the filter designer with complete control over the noise and error characteristics to evaluate the designed navigation system under diverse conditions.

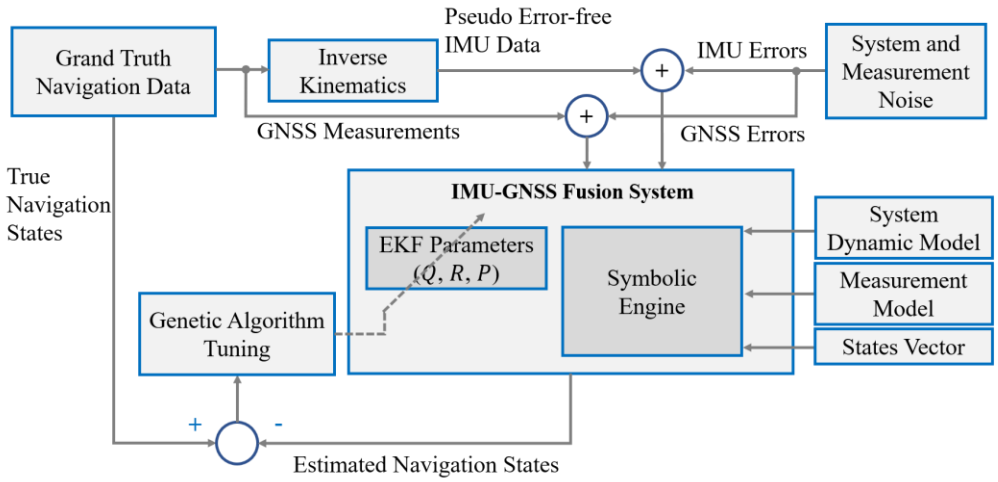


Figure 3. 1. An overview of the developed design and simulation environment for the IMU-GNSS TC navigation system.

The design and tuning procedure are as follows:

A. Given a ground truth navigation solution, Inverse Kinematics (IK) is first applied to generate pseudo-error-free IMU and GNSS measurements.

B. These measurements are then contaminated by different sources of noise and error with known characteristics to generate simulated noisy data. The IMU's errors include accelerometer's and gyroscope's biases and scale factors as well as random noise on the raw inertial measurements. In the TC integration, the GNSS errors include random noise on the receiver clock bias, receiver clock drift, and the range and range rate measurements.
C. Employing the Symbolic engine of MATLAB software [73], the Jacobians of the system and measurement models are accurately and automatically calculated for the TC integration engine.
D. Taking the weighted sum of navigation states' RMSE as the performance evaluation metric of the integration system, the GA-based tuning module optimizes the performance with respect to the tunable parameters (e.g. entries of process noise covariance matrix, Q, measurement noise covariance matrix, R, and initial state error covariance matrix, P).
E. Once the correctness and performance of the filter are verified under different controlled and simulated conditions, the filter is then tuned and evaluated on the real data.

Employing a symbolic engine improves the readability and flexibility of the environment. To show an example of how MATLAB Symbolic Engine is used to represent the system model, consider the gyroscope x bias modelled as a first-order Gauss-Markov process. This can be represented in MATLAB symbolic engine as follows:

```
%Gyro bias dynamic model symbols
syms gyro_bias_x_dot gyro_bias_x gyro_bias_x_time_cnst
gyro_bias_gauss_markov_stdv_x wg_noise;

%Gyro bias dynamic system equations
gyro_bias_x_dot = (-1/gyro_bias_x_time_cnst)*gyro_bias_x
+sqrt(2*(gyro_bias_gauss_markov_stdv_x^2)/gyro_bias_x_ti
me_cnst)*wg_noise;
```

where gyroscope's bias rate of change along the x-axis is represented by the symbol variable "gyro_bias_x_dot" and the zero-mean unity-variance Gaussian noise is represented by the symbol variable "wg_noise".

It is worth discussing that the noise characteristics of sensors reported in their datasheet are often obtained from highly controlled laboratory experiments. However, when the sensor is used in practice, it shows different noisy behaviour. Re-applying GA on the real dataset has the advantage that the GA optimizes the performance on the actual field-collected data, which naturally carries the error and noisy characteristics that the system may experience in practice. Hence, the parameters tuned on the real dataset are expected to better suit scenarios where the navigation system is supposed to operate.

3.2.1 The Developed Embedded Multi-Sensor Logger System

The dataset used to evaluate the IMU-GNSS TC navigation systems in this chapter was collected in collaboration with a team at BlackBerry QNX Company. The dataset was collected by two multi-sensor logger systems working side by side. The first logger system was developed by QNX that has been used as the reference. The second logger system was our developed logger system. In this section, first, the specifications of our developed logger system developed on the NVIDIA® Jetson TX-2 embedded system are introduced. Then the specifications of QNX logger systems and the collected dataset are discussed.

Our developed logger system interfaces with multiple sensors, as shown in Figure 3. 2 and Table 3. 1. Our multi-sensor logger system has been developed using C++ programming language in a Linux Ubuntu 16.04 operating system environment. To perform the logging from multiple sensors efficiently and simultaneously, the program has been developed using the multi-thread feature of the C++ programming language, *PThread* [74]. The logged data from all sensors are synchronized using a high-resolution timer, *Chrono* [74]. The recorded sensory data are stored in binary formats in a single file on an on-board SD card. To resolve the potential conflicts among the threads in accessing the shared output file, the writing operations are guarded by the *Mutex* locking mechanism [74]. When a thread attempts to write into the output file, it first checks whether other threads have already locked the file. Considering the sampling rate and data size of the interfaced sensors, the locking mechanism is critical to guarantee consistent and continuous logging of all sensors. To parse the logged file and extract the desired data, the sensory data is first put into a data structure with a unique header followed by time tag, structure size, and sensory data. A software architecture view of the developed logger system is shown in Figure 3. 3.

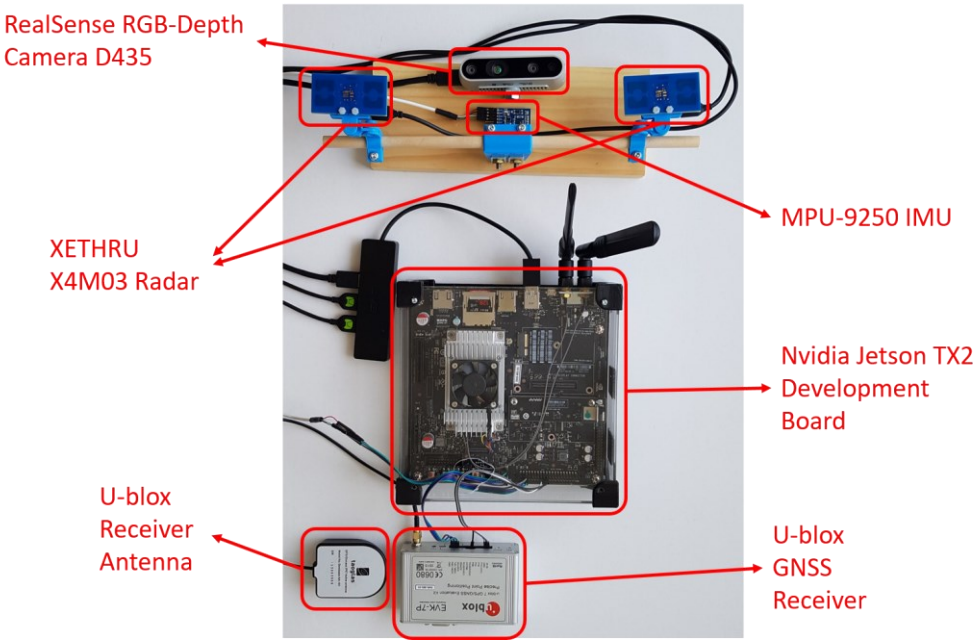


Figure 3. 2. The developed multi-sensor logger system.

Table 3. 1. The specifications of the interfaced sensors in the developed multi-sensor logger system.

Sensor	Specification
MPU-9250 IMU [75]	3-axis accelerometer 3-axis gyroscope 3-axis magnetometer 100 samples per second I2C interface
u-blox EVK-7P GPS/GNSS Receiver [76]	Position, Velocity, time, Range, Range rate 1Hz UART interface
Realsense Depth Camera D435 [77]	RGB camera 1280×720 resolution at 30 fps Depth map 1280×720 resolution at 30 fps USB 3.0 interface
XETHRU X4M03 Ultra-Wide Band RADAR [78]	6-10.2 GHz transmission range 10m nominal range 10-400 samples per second USB 2.0 interface

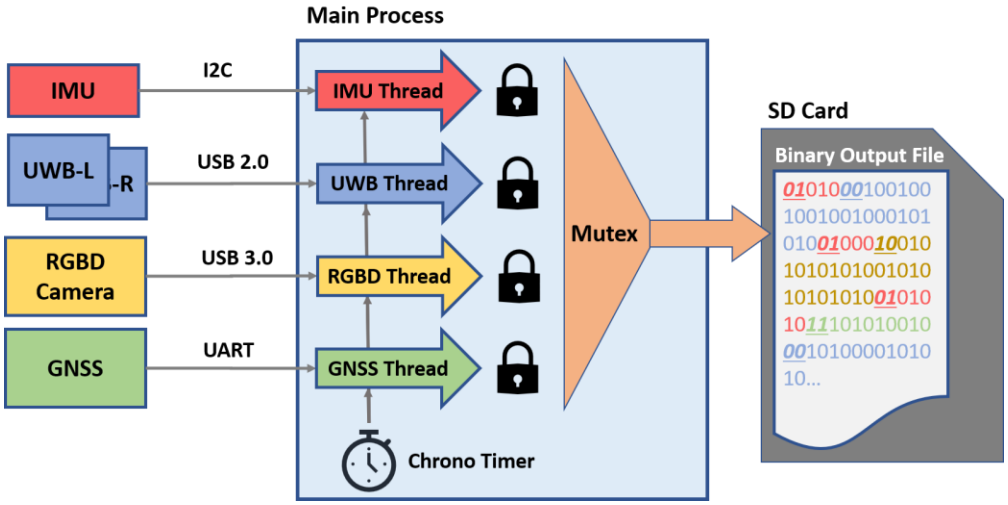


Figure 3. 3. The software system view of the developed logger. A unique header distinguishes the logged data in the output binary file. All logged data are time tagged.

The QNX logger system, which is used as the reference, was a NovAtel ProPak6 equipped with OEM6 GNSS receiver and a high-end KVH1750 IMU with Fiber Optic Gyros (FOG) and MEMS accelerometer measuring at 200Hz [79][80]. The ground truth positioning solution on this system is provided through Novatel Real-Time Kinematic (RTK) technology with centimetre-level accuracy utilizing satellite systems from GPS, Galileo, and GLONASS [79]. It should also be mentioned that in our developed logger system, the range and range rate measurements were obtained only from the GPS satellite system. The specifications of both logger systems are summarized in Table 3. 2. Figure 3. 4 shows the installation of these two logger systems in the trunk of the test vehicle.

(a) (b)

Figure 3. 4. (a) The test vehicle. (b) the NovAtel and our developed logger systems installed in the trunk of the test vehicle.

Table 3. 2. The specification of the two logger systems.

Specification	Value
NovAtel ProPak6 [79]	
DGPS horizontal accuracy (RMS)	0.4 (m)
KVH1750 [80]	
Accelerometer technology	MEMS
Accelerometer bias stability	7.5 (mg)
Velocity random walk	0.07 $(\mathrm{m/s}/\sqrt{hr})$
Gyroscope technology	Fibre Optics
Gyroscope bias stability	0.05 $(°/hr)$
Angular random walk	0.012 $(°/hr/\sqrt{Hz})$
Our Developed Logger System	
MPU9250 [81]	
Accelerometer technology	MEMS
Accelerometer random noise	0.0294 $(m/s^2/\sqrt{Hz})$
Gyroscope technology	MEMS
Gyroscope noise	0.01 $(°/s/\sqrt{Hz})$

Using both logger systems working side by side, a dataset was collected in an open sky condition in Kanata, Ottawa, Ontario, Canada. In this data collection (i.e. inertial and GNSS data) in collaboration with BlackBerry QNX company, all the required permissions required to respect the regulations for data collection in public and urban areas were

managed by the BlackBerry QNX company. Figure 3. 5 shows three different portions of this dataset used in the experiments of this section.

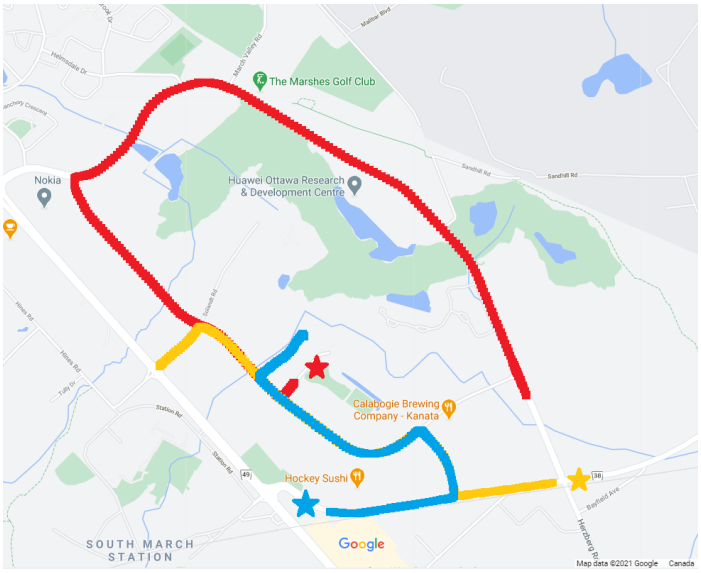


Figure 3. 5. The collected dataset in Kanata, Ottawa, ON, Canada. The Training, Validation, and Test sequences have been highlighted by red, blue, and yellow colours, respectively. The starting point of each sequence has been marked by a star sign.

3.2.2 IMU-GNSS TC Integration Experimental Results

In this section, the experimental results of the GA-tuned IMU-GNSS TC integration system are discussed. The IMU-GNSS integration system was developed in MATLAB 2019a software, and the GA toolbox was employed for the tuning. The developed IMU-GNSS TC integration system was tested in two cases. The first case is a controlled case with a simulated dataset obtained from the IK step followed by contamination with pre-defined errors (e.g. IMU's biases, scale factor, and noise, and GNSS noise). The second case is on the real dataset directly obtained from the real IMU sensor and GNSS receiver of our developed logger system. The motivation for initial testing on the simulated dataset is to verify that the fusion system provides a quality solution under controlled conditions. This verification is done by analyzing that the filter's estimates of hidden states converge to the

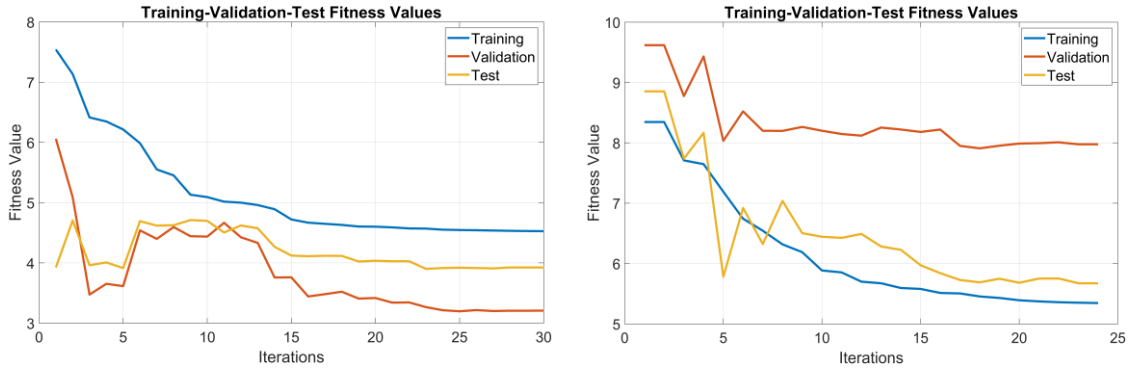
pre-defined values. To prepare the simulated dataset, the contaminating parameters of the IMU's measurements were chosen from the MPU-9250 datasheet [81] as if the accelerometer's and gyroscope's measurements were obtained directly from a real sensor. However, the IMU's measurement biases were chosen manually. The contaminating parameters are reported in Table 3. 3.

Table 3. 3. The values of errors and noise contaminating the simulated dataset used the IMU-GNSS TC navigation system.

Quantity	Error Value
IMU	
Gyroscope bias XYZ ($\boldsymbol{deg/s}$)	[0.2, 0.4, 0.6]
Gyroscope random noise XYZ ($\boldsymbol{deg/s}$)	[0.1, 0.1, 0.1]
Accelerometer bias XYZ ($\boldsymbol{m/s^2}$)	[0.1, 0.2, 0.3]
Accelerometer noise XYZ ($\boldsymbol{m/s^2}$)	[0.029, 0.029, 0.29]
GNSS Receiver	
Position noise NED (m)	[1, 1, 5]
Velocity noise NED ($\boldsymbol{m/s}$)	[0.01, 0.01, 0.05]
Pseudo-range noise ($\boldsymbol{m}$)	0.5
Receiver clock bias (m/s)	10

The Training-Validation-Test policy for training requires the GA to be tuned on a training sequence (the red trajectory in Figure 3. 5), the optimal parameters to be selected based on the performance on a validation sequence (the blue trajectory in Figure 3. 5), and a final evaluation to be done on a test sequence (the yellow trajectory in Figure 3. 5). This dataset splitting is to prevent the tuning technique from overfitting to a particular dataset as well as to guarantee that the tuned parameters are still effective for more general scenarios than just the dataset on which the GA was tuned. The fitness function in this experiment was the weighted sum of the RMSE of the 3-Element position and 3-Element attitude in the $\{L\}$ frame. The GA fitness values of the Training, Validation, and Test sequences evaluated by the IMU-GNSS TC integration systems for both simulated and real datasets are shown in

Figure 3. 6. The minimum fitness value on the validation sequence is observed at the 30th iteration on the simulated dataset and at the 18th iteration on the real dataset. Therefore, the parameters at those iterations were used as the optimal parameters.



(a) (b)

Figure 3. 6. Training, Validation, and Test fitness values evaluated by the IMU-GNSS TC integration system on (a) simulated and (b) real datasets. The minimum fitness value on the validation sequence happens at the 30th and 18th iteration on the simulated and real datasets, respectively.

One interesting observation on the simulated dataset is that since the errors and noise contaminating the Validation and Test sequences have been drawn from exactly the same distribution, the Validation and Test datasets' fitness values also show similar ascending and descending behaviours through iterations. For instance, whenever there is a peak on the Validation sequence fitness value, there is a corresponding peak on the Test sequence as well. On the real dataset, this correlation is weaker because Validation and Test sequences might have been affected by slightly different noise characteristics. However, the proposed Training-Validation-Test policy has been adopted to address this natural event and achieve optimal parameters while maintaining their generalizability.

The RMSE of the position, velocity, and attitude estimates of the IMU-GNSS TC integration system using the datasheet-based tuning and GA-based tuning are reported in

Table 3. 4. Comparison of the error results confirms that the GA-based tuning with the Training-Validation-Test policy provides a generalizable set of parameters that lead to consistent accuracy on all sequences on the simulated and real dataset.

Table 3. 4. The RMSE and standard deviation of the full navigation states estimated by the IMU-GNSS TC integration system on the simulated and real datasets.

	Simulated Data		Real Data	
	RMSE	**STD**	**RMSE**	**STD**
Datasheet-based Tuning				
Training Sequence				
Position NED ($\boldsymbol{m}$)	[1.88, 2.56, 5.28]	[1.86, 1.98, 2.77]	[6.85, 3.63, 2.26]	[5.66, 3.54, 2.24]
Velocity NED ($\boldsymbol{m/s}$)	[0.56, 0.80, 0.77]	[0.56, 0.70, 0.57]	[1.27, 0.98, 0.64]	[1.12, 0.97, 0.63]
Attitude $\boldsymbol{r, p, A}$ ($\boldsymbol{deg}$)	[0.65, 0.66, 4.61]	[0.58, 0.66, 4.31]	[2.74, 3.32, 12.63]	[0.54, 0.71, 10.64]
GA-based Tuning				
Training Sequence				
Position NED ($\boldsymbol{m}$)	[0.36, 0.41, 0.27]	[0.34, 0.41, 0.26]	[1.50, 1.29, 1.45]	[1.46, 1.27, 1.45]
Velocity NED ($\boldsymbol{m/s}$)	[0.15, 0.18, 0.04]	[0.15, 0.18, 0.03]	[0.32, 0.32, 0.19]	[0.31, 0.31, 0.18]
Attitude $\boldsymbol{r}$, $\boldsymbol{p}$, $\boldsymbol{A}$ ($\boldsymbol{deg}$)	[0.18, 0.34, 2.00]	[0.18, 0.30, 1.93]	[3.02, 3.15, 1.52]	[0.34, 0.33, 1.50]
Validation Sequence				
Position NED ($\boldsymbol{m}$)	[0.28, 0.24, 0.52]	[0.25, 0.22, 0.50]	[0.67, 0.63, 0.62]	[0.67, 0.62, 0.55]
Velocity NED ($\boldsymbol{m/s}$)	[0.17, 0.11, 0.16]	[0.16, 0.10, 0.15]	[0.31, 0.35, 0.11]	[0.31, 0.35, 0.11]
Attitude $\boldsymbol{r}$, $\boldsymbol{p}$, $\boldsymbol{A}$ ($\boldsymbol{deg}$)	[0.36, 0.33, 1.15]	[0.33, 0.31, 1.14]	[2.86, 2.84, 5.24]	[0.72, 0.61, 5.12]
***Test* Sequence**				
Position NED ($\boldsymbol{m}$)	[0.26, 0.36, 0.35]	[0.26, 0.36, 0.33]	[1.64, 0.61, 1.19]	[1.33, 0.60, 1.11]
Velocity NED ($\boldsymbol{m/s}$)	[0.12, 0.20, 0.09]	[0.12, 0.20, 0.09]	[0.42, 0.25, 0.15]	[0.36, 0.25, 0.14]
Attitude $\boldsymbol{r}$, $\boldsymbol{p}$, $\boldsymbol{A}$ ($\boldsymbol{deg}$)	[0.31, 0.38, 1.27]	[0.29, 0.37, 1.21]	[2.67, 2.87, 1.55]	[0.33, 0.40, 1.53]

Table 3. 5. Sample of tuned parameters proposed by the GA for the IMU-GNSS TC integration system on the simulated and real datasets.

Parameter	Simulated Dataset	Read Dataset
Accelerometer noise Std ($\boldsymbol{m/s^2/\sqrt{Hz}}$)	[1.82, 1.23, 0.32]	[0.31, 0.11, 0.24]
Accelerometer bias GM time constant (s)	[0.2e4, 1.7e4, 0.6e4]	[5.6e4, 1.7e4, 6.2e4]
Gyroscope noise Std ($\boldsymbol{°/s/\sqrt{Hz}}$)	[0.14, 0.1.89, 0.26]	[1.91, 0.35, 1.16]
Gyroscope bias GM time constant (s)	[2.3e4, 0.07e4, 4.2e4]	[5.9e4, 1.9e4, 0.2e4]
Range noise Std ($\boldsymbol{m}$)	10.16	49.3

A sample of tuned parameters of the IMU-GNSS TC integration system tuned by the GA is reported in Table 3. 5. From the Kalman filter's point of view, the absolute values of the tunable parameters do not play a critical role in defining the behaviour of the filter; instead,

the relative value of Q, R and P matrices defines how the filter behaves [63]. Hence, analyzing the absolute magnitude of individual parameters in Table 3. 5 may not necessarily convey a definitive meaning.

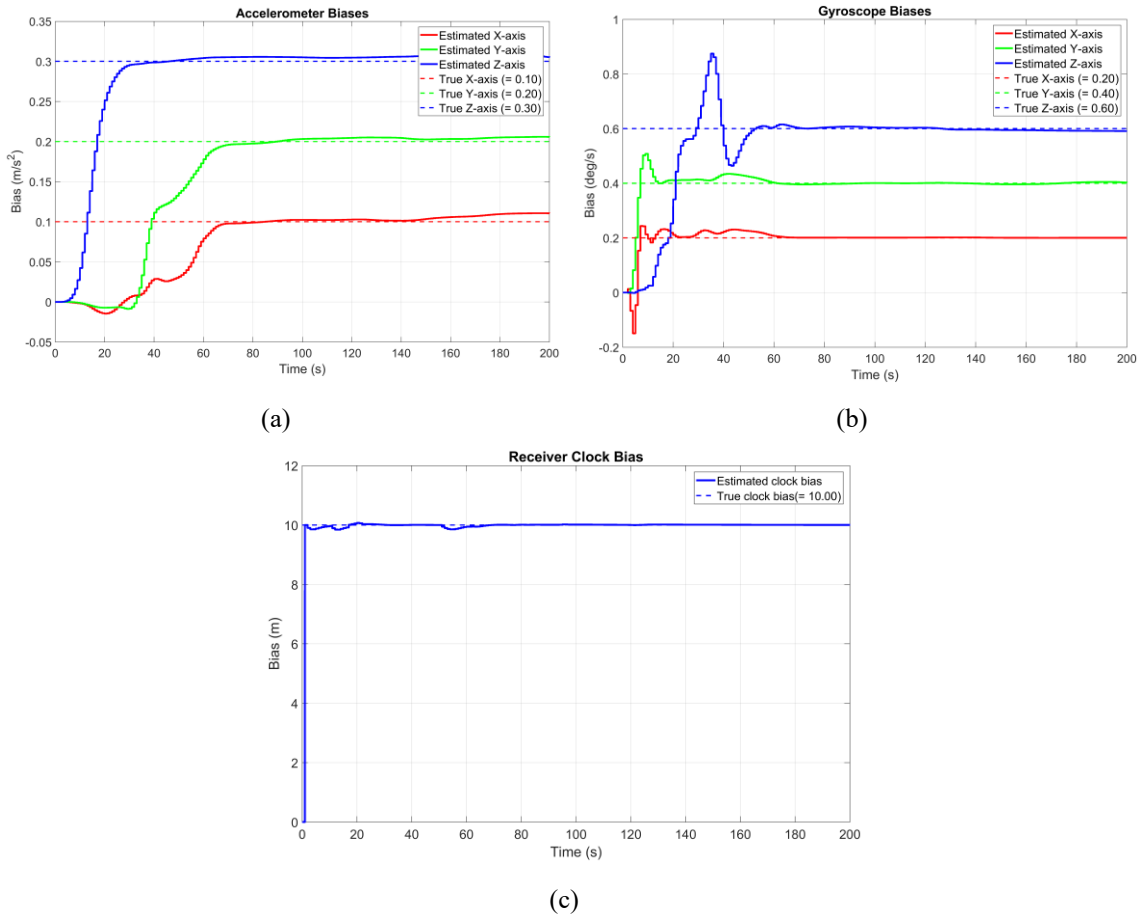


Figure 3. 7. The IMU-GNSS TC integration system performance on estimating the (a) accelerometer's bias, (b) gyroscope bias, and (c) receiver clock bias on the simulated dataset.

In the simulated dataset, to further evaluate the filter's performance, it is important to analyze the filter's performance in estimating the hidden states of the system, such as IMU's biases. Figure 3. 7 shows the convergence of the accelerometer's and gyroscope's biases and the receiver clock bias to their expected values on the simulated data.

The 2D position estimate and the errors on the navigation states on the real dataset are shown in Figure 3. 8. The estimates of the accelerometer's and gyroscope's biases and the receiver clock bias in this experiment are also shown in Figure 3. 9.

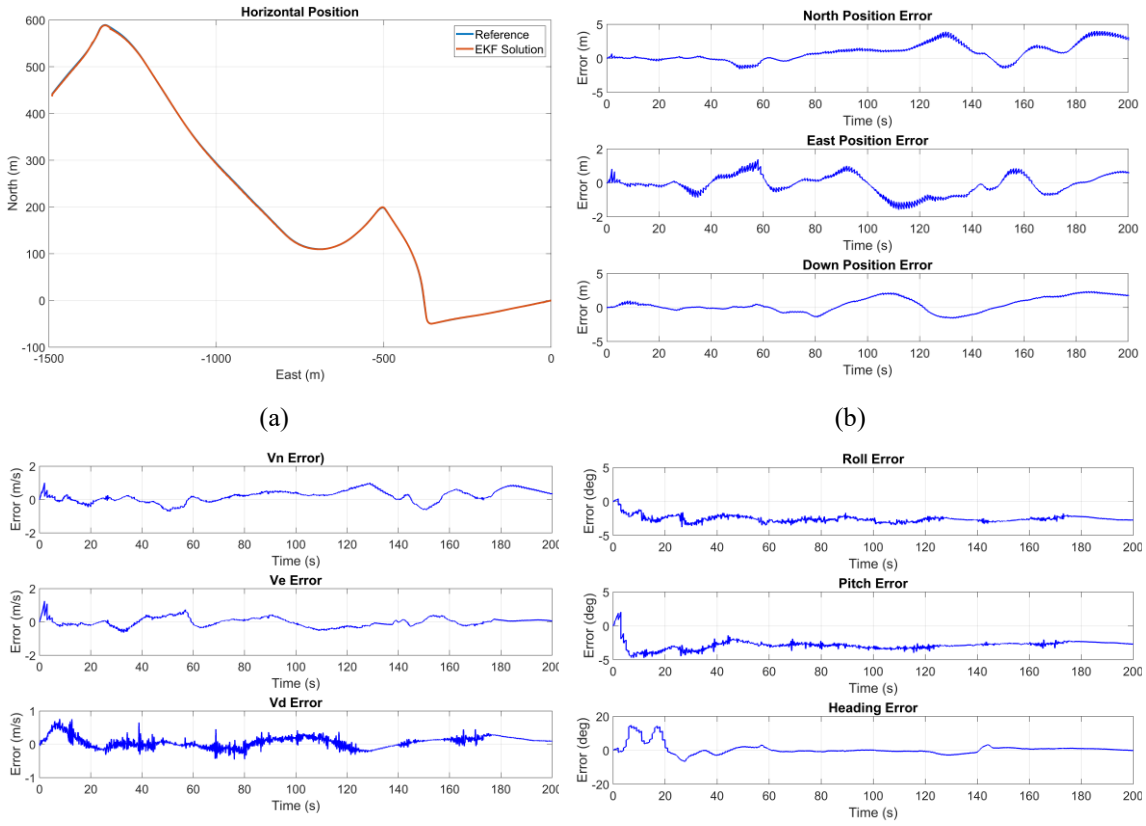


Figure 3. 8. (a) 2D position estimate, (b) position estimate error, (c) velocity estimate error, and (d) attitude estimate error on the real dataset.

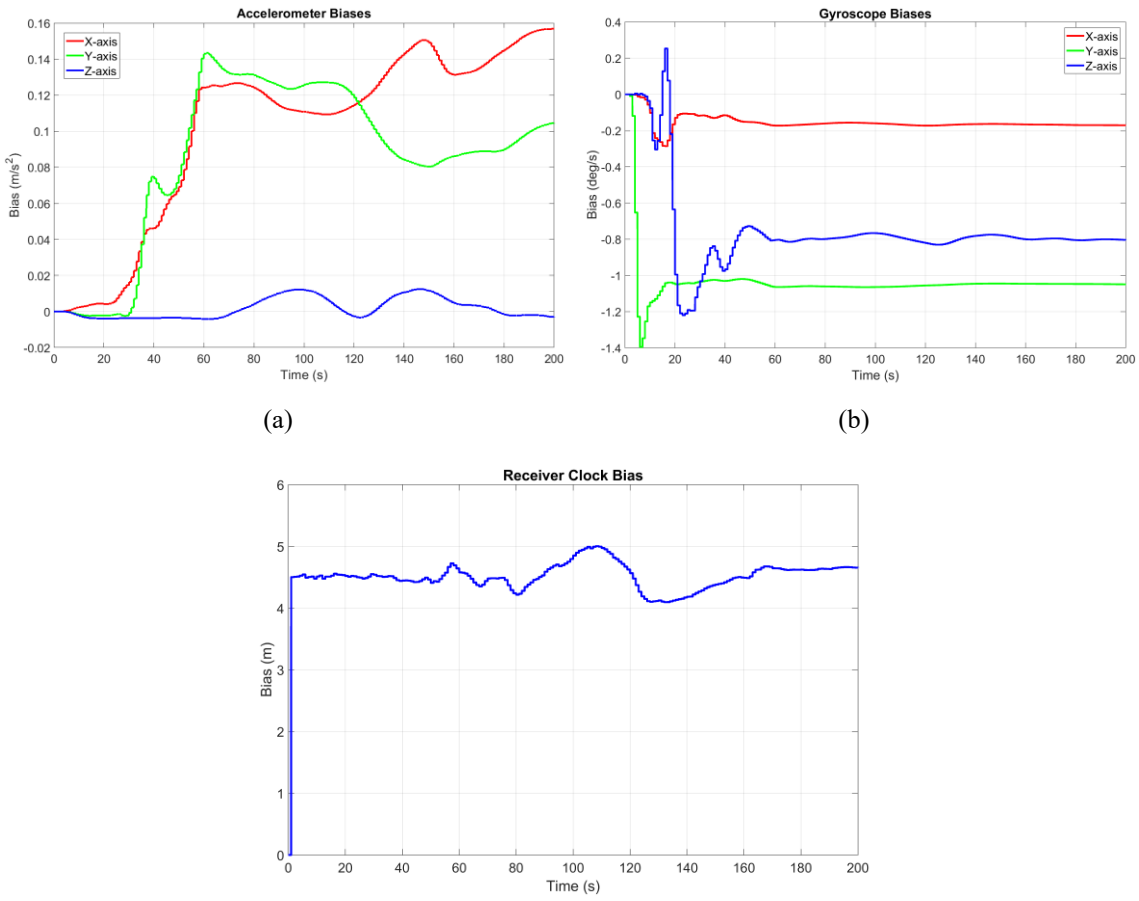


Figure 3. 9. (a) Accelerometer's bias, (b) gyroscope's bias, and (c) receiver clock bias estimated by the GA-tuned IMU-GNSS TC integration system on the real dataset.

To further evaluate the GA-tuned IMU-GNSS TC integration system, the horizontal position RMSE of the filter was evaluated under different outage durations and distances. Figure 3. 10 shows the RMSE of the IMU-GNSS TC integration system under complete outage, where no satellite is observed, and partial outage, where one to four satellites are observed. As shown in Figure 3. 10, receiving updates from even a single satellite contributes to considerably bounding the errors compared to the complete outage. The closeness of the results under 2, 3, or 4 visible satellites on the simulated and real datasets might be because of particular formations of the satellites with respect to the receiver

location such that 2D position estimate using one or more extra visible satellites did not contribute considerably to the final estimation accuracy.

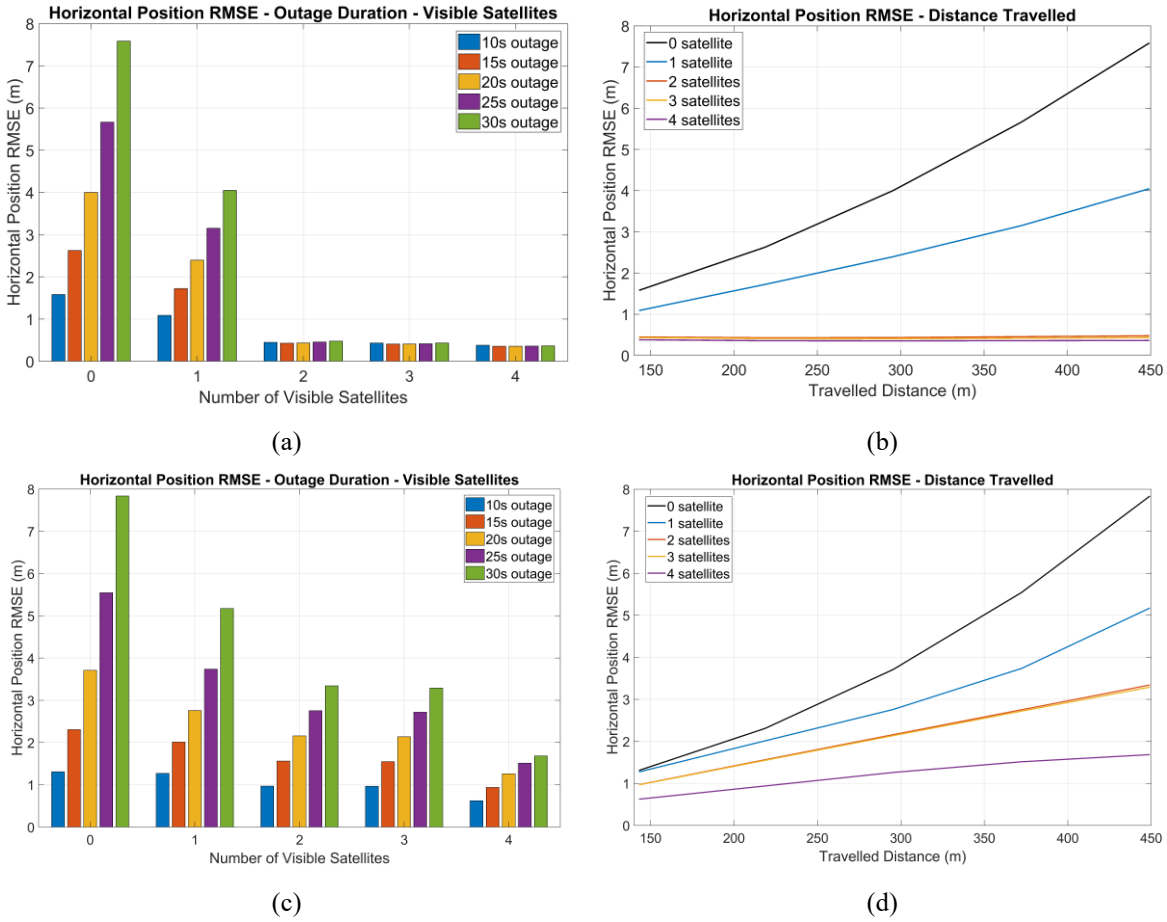


Figure 3. 10. Horizontal position RMSE with respect to the outage duration and distance travelled evaluated by the IMU-GNSS TC integration system on (a, b) the simulated and (c, d) real datasets.

3.3 MSCFK Visual-Inertial Navigation System Tuning Framework

An overview of the developed design environment for the enhanced MSCKF VIN system is shown in Figure 3. 11. The design and tuning procedure for the enhanced MSCKF VIN system were performed in MATLAB 2019a software, similar to the IMU-GNSS TC integration system discussed in the previous section.

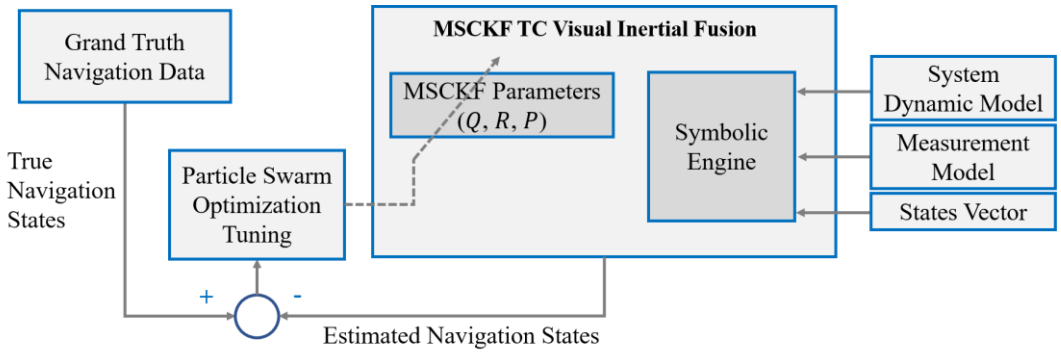


Figure 3. 11. An overview of the developed design environment for the MSCKF VIN system.

Compared to the tuning procedure of the IMU-GNSS TC navigation system, the tuning procedure of MSCFK VIN has a few modifications. The first modification is that the tuning technique used for the enhanced MSCKF VIN system is now based on the PSO algorithm using MATLAB 2019a PSO toolbox. PSO and GA are both evolutionary population-based optimization techniques that often have comparable performance in achieving a quality optimization solution [82]. One marginal advantage of the PSO over GA is its less computational complexity. This advantage, to a great extent, is application-dependent; however, in the Kalman filter tuning application, the PSO's operators to generate new solutions (continuous movement of particles in the search space) is more compatible with the continuous nature of tunable parameters compared to Crossover operator in GA that can cause large displacement of the solutions in the search space. The second modification

in the design environment of the enhanced MSCKF VIN system is that it directly works on the real dataset. This is mainly because the correctness of the INS system was already verified in the IMU-GNSS integration system, and more importantly, generating simulated visual data with desired controllability and flexibility requires expensive processing and a 3D animated environment which is beyond the scope of this Ph.D. research.

The proposed PSO-tuned enhanced MSCKF VIN system has been tested on the publicly available KITTI odometry dataset [83]. This dataset contains different sequences collected outdoor in residential, city, and highway areas by a vehicle equipped with two stereo colour and grayscale cameras, a laser scanner, and an IMU-GNSS positioning system with RTK technology with positioning error less than 5cm under open-sky condition. In this experiment, raw IMU measurements (3-axis accelerometer and 3-axis gyroscope at 100Hz sampling rate) and the images of a monocular grayscale camera (with 1392×512 pixels resolution at 10fps) were used.

3.3.1 MSCKF VIN Experimental Results

In this experiment, one of the challenging sequences of the KITTI odometry dataset has been chosen. “Sequence-07” is characterized by different motion patterns: straight driving, smooth and sharp turns, and stationary periods. What makes this sequence challenging is visually dynamic situations that truly challenge any VIN system. Figure 3. 12 shows the KITTI “Sequence-07” and the three sections chosen for Training, Validation, and Test. “Sequence-07” is 692meters long, and the vehicle travels it in 110 seconds. The Training, Validation, and Test portions of this dataset are 240, 182, and 270 meters long, respectively. It is worth mentioning that the dynamic scene in this sequence happens during

the Test sections of the sequence. The fitness function used in the PSO algorithm is the weighted sum of the RMSE of the 3-Element position and 3-Element attitude in the $\{L\}$ frame.

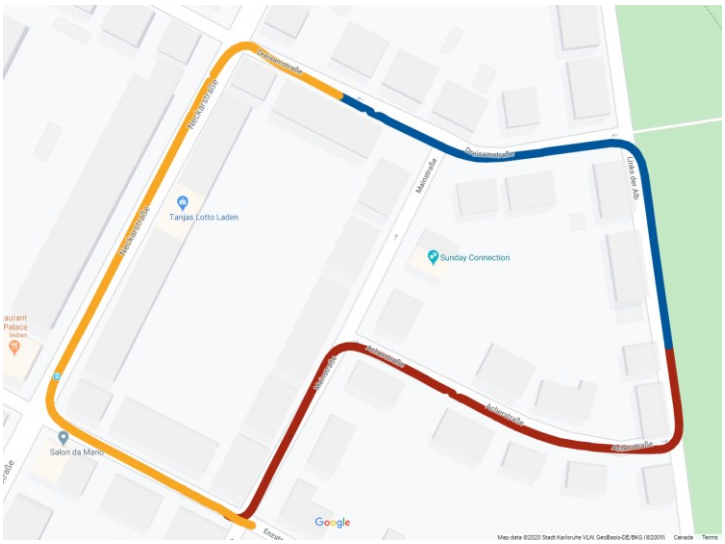


Figure 3. 12. The KITTI "Sequence-07". The Training, Validation, and Test sections of this sequence have been highlighted by red, blue, and yellow colours, respectively. The vehicle starts its travel from the red portion, and the direction of the sequence is anti-clockwise.

To avoid overfitting and guarantee generalizability, selecting the optimal parameters was based on the fitness value on the Validation section. However, since the VIN system is a dead-reckoning system, the accumulated fitness values on the Training, Training+Validation, and Training+Validation+Test were found to be more informative in selecting the optimal parameters. Figure 3. 13 shows the accumulated fitness values on the three sections of the sequence. As highlighted in the figure, the optimal parameters were selected before the ascending behaviour of the fitness value on the Validation+Test at the 75^{th} iteration. A sample of tuned parameters is reported in Table 3. 6. In addition to entries of Q, R and P matrices, the critical probability of Chi-Square test and the depth estimation standard deviation threshold were also considered as tunable parameters. As discussed in

the previous sections, in the Kalman filter, rather than the absolute values of the stochastic parameters, the relative magnitude of these parameters is more critical in determining the Kalman gain and, consequently, the filter's behaviour [63].

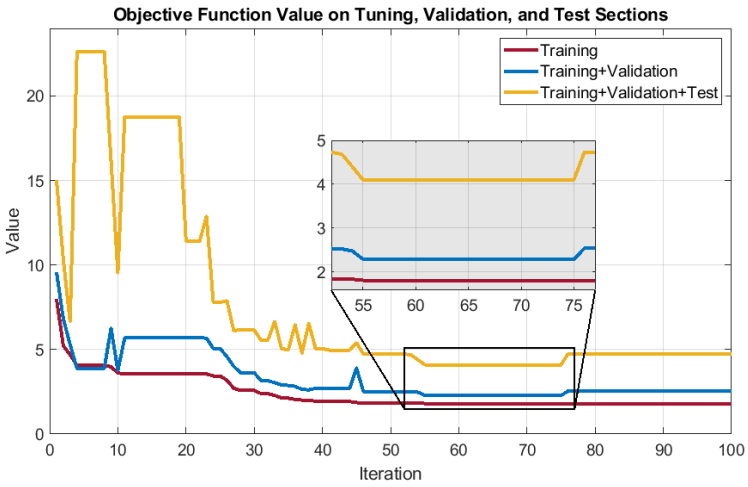


Figure 3. 13. Training, Training+Validation, and Training+Validation+Test fitness values of the MSCKF VIN system evaluated on the KITTI "Sequence-07". The minimum fitness value on the Training+Validation section happens at the 75th iteration.

Table 3. 6. Sample of tuned parameters proposed by the PSO for the MSCKF VIN system on the KITTI "Sequence-07".

Parameter	Values
Accelerometer Noise Std ($\boldsymbol{m/s^2}$ $/\sqrt{\boldsymbol{Hz}}$)	[2.62e-6, 8.42e-6, 8.20e-6]
Gyroscope Noise Std ($°/\boldsymbol{s}/\sqrt{\boldsymbol{Hz}}$)	[6.12e-9, 4.25e-7, 6.69e-7]
NED Position noise Std ($\boldsymbol{m}$)	[5.67e-4, 2.71e-10, 5.92e-5]
NED Velocity noise Std ($\boldsymbol{m/s}$)	[6.80e-4, 1.96e-4, 1.30e-4]
Quaternion Attitude noise Std	[8.61e-5, 1.26e-7, 1.24e-7, 7.65e-5]
Visual Feature Noise Std ($\boldsymbol{pixel}$)	39.5
Chi-Square Test Critical Probability	0.91
Features Depth Std threshold ($\boldsymbol{m}$)	2.5

Table 3. 7. The RMSE of navigation states estimates of the PSO-tuned MSCKF VIN system evaluated on the KITTI "Sequence-07".

Dataset Section	RMSE
Training	
Position NED $(\boldsymbol{m})$	[0.84, 1.15, 0.50]
Velocity $(\boldsymbol{m/s})$	[0.12, 0.09, 0.06]
Attitude $\boldsymbol{r, p, A}$ $(\boldsymbol{deg})$	[0.10, 0.12, 1.02]
Training+Validation	
Position NED $(\boldsymbol{m})$	[1.43, 1.32, 0.55]
Velocity $(\boldsymbol{m/s})$	[0.12, 0.11, 0.05]
Attitude $\boldsymbol{r, p, A}$ $(\boldsymbol{deg})$	[0.10, 0.12, 1.06]
Training+Validation+Test	
Position NED $(\boldsymbol{m})$	[1.84, 3.42, 0.69]
Velocity $(\boldsymbol{m/s})$	[0.15, 0.24, 0.04]
Attitude $\boldsymbol{r, p, A}$ $(\boldsymbol{deg})$	[0.10, 0.11, 1.08]

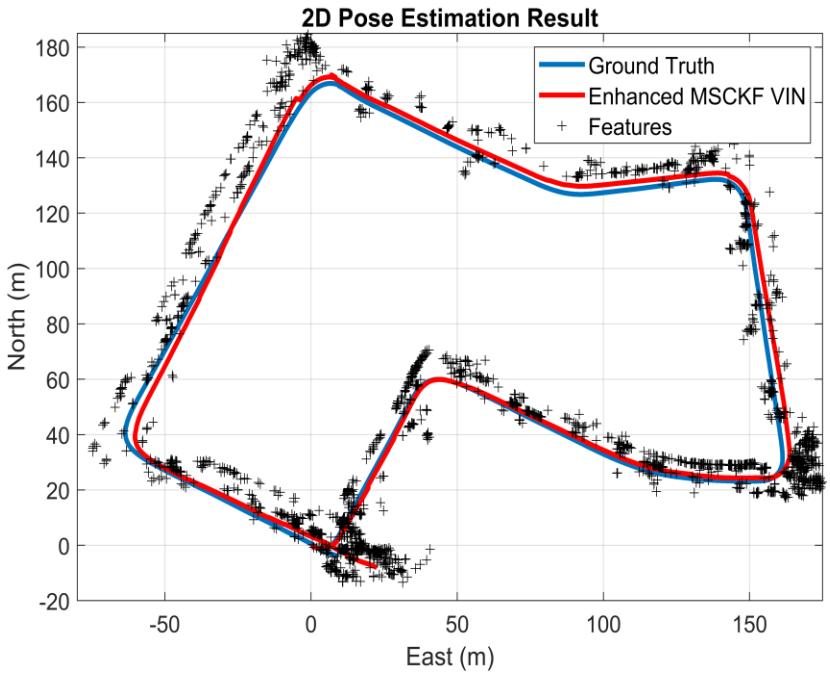
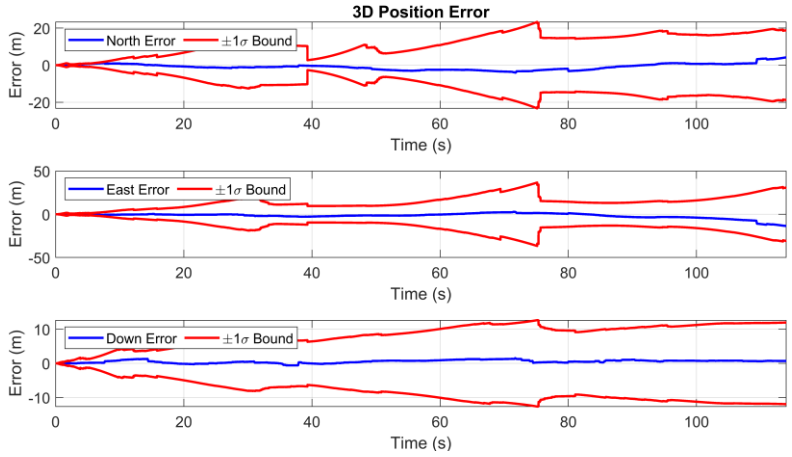


Figure 3. 14. Horizontal position estimate by the PSO-tuned MSCKF VIN system evaluated on the KITTI "Sequence-07".

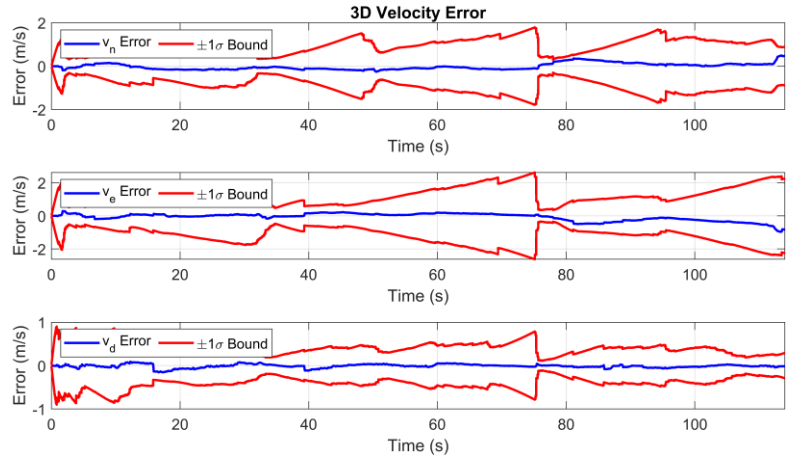
Table 3. 7 shows the RMSE of the navigation states estimated by the PSO-tuned MSCKF VIN system on the three sections of the KITTI "Sequence-07". Figure 3. 14 shows the

horizontal position estimated by the PSO-tuned MSCKF VIN system compared to the ground truth solution. In this experiment, in each image, 100 SURF features [41] were detected. Figure 3. 14 also shows the localized features in the $\{L\}$ frame. It is worth mentioning that no post-processing such as re-localization or loop-closure using these features were applied in the MSCKF VIN system.

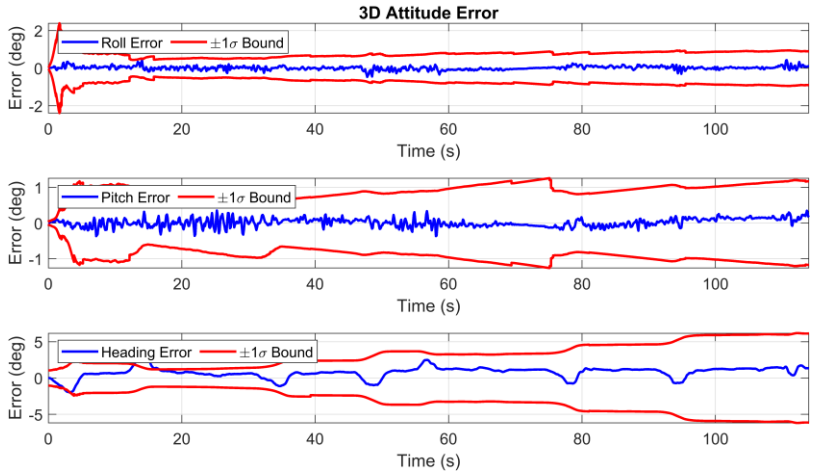
In a dead-reckoning EKF-based navigation system such as MSCKF VIN, in addition to state estimation error, it is also important to analyze the state error covariance estimates. Figure 3. 15 shows how the PSO-tuned MSCKF VIN system successfully bound the navigation states' errors within the states error covariance. The $\pm 1\sigma$ bound is reasonably tight and conveys this conclusion that the PSO-tuned MSCKF operates optimally. Excessively tight $\pm 1\sigma$ bound means that the filter has underestimated the uncertainty of its predictions that may undervalue quality updates. On the other hand, excessively loose $\pm 1\sigma$ bound means that the filter relies more than necessary on the updates. In both cases, the filter is working sub-optimal.



(a)



(b)



(c)

Figure 3. 15. (a) Position, (b) velocity, and (c) attitude estimation error and their corresponding ±1σ bounds.

Figure 3. 16 and Figure 3. 17 show the estimates of IMU's error and the camera's intrinsic and extrinsic parameters. IMU errors were initialized to zero, but camera intrinsic and extrinsic parameters were initialized with the configuration values reported in the KITTI odometry dataset [84]. Table 3. 8 compares the camera's intrinsic and extrinsic parameters reported in the KITTI odometry dataset with estimates proposed by the MSCKF VIN.

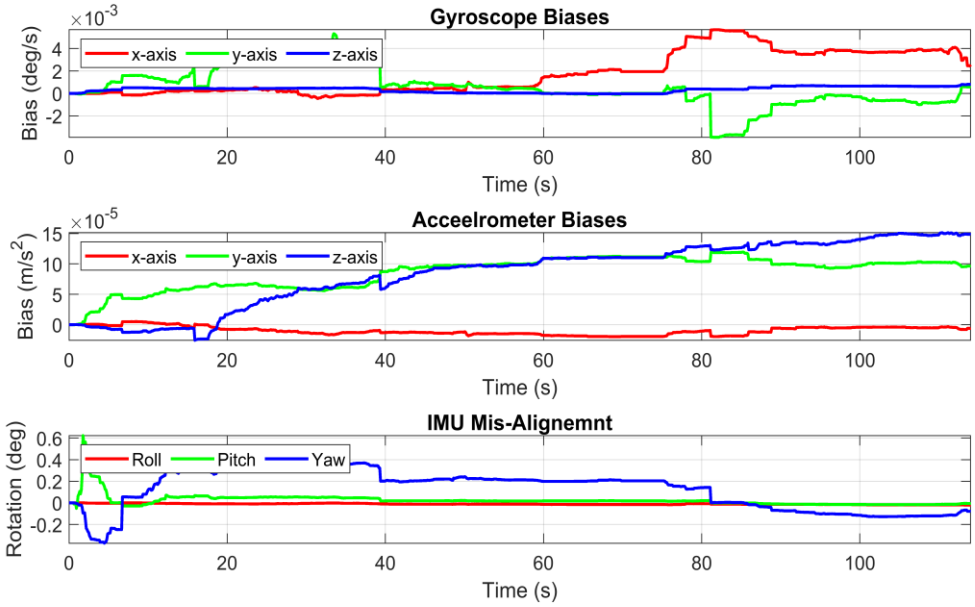
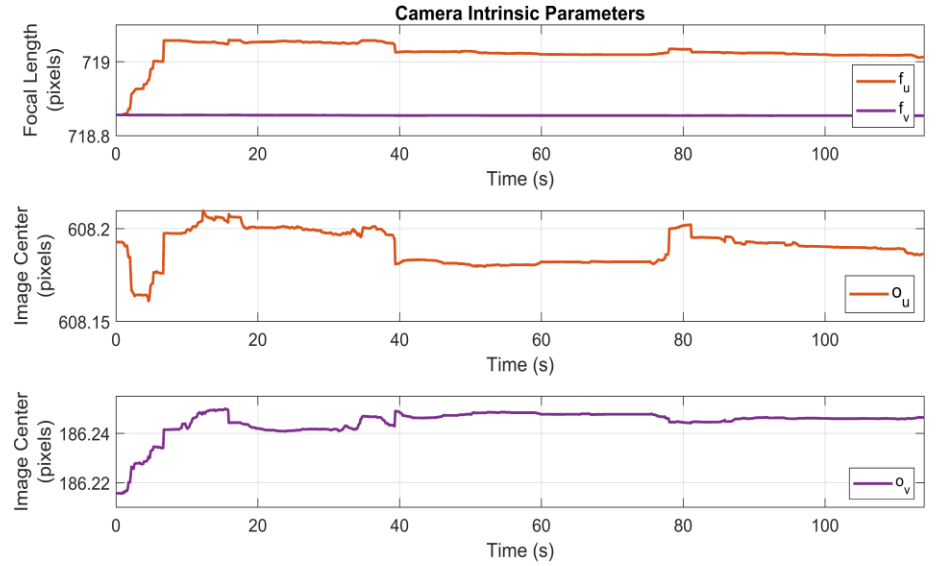
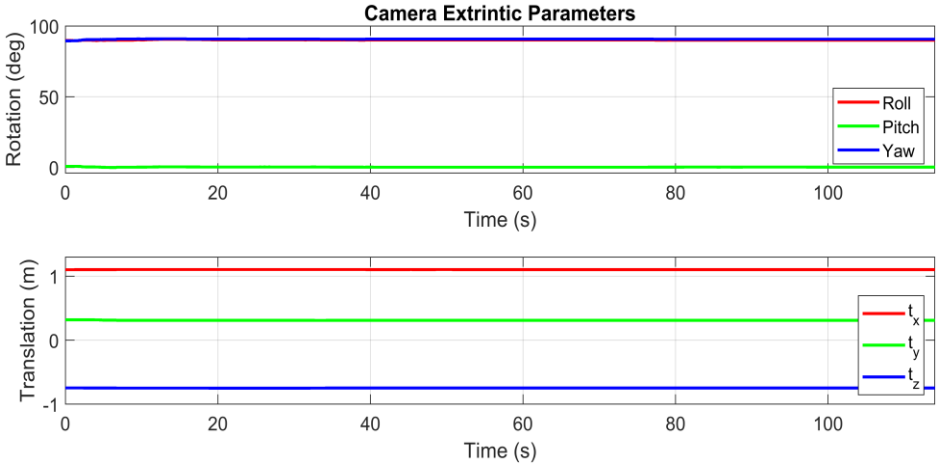


Figure 3. 16. Estimation of the accelerometer's bias, gyroscope's bias, and IMU mounting misalignment angles.



(a)



(b)

Figure 3. 17. The estimation of camera's (a) intrinsic and (b) extrinsic parameters. These parameters were initialized with values that were provided initially with the KITTI dataset.

Table 3. 8. The estimation of the IMU's misalignment and camera's intrinsic and extrinsic parameters on KITTI "Sequence-07" compared to the original values reported in the KITTI dataset [84]**.**

	KITTI Dataset [84]	**MSCKF VIN Estimation**
IMU's Misalignment ($\boldsymbol{deg}$)	[0.0,0.0,0.0]	[-0.01, -0.02, -0.08]
Camera's Intrinsics (pixel) $[\boldsymbol{f_u}, \boldsymbol{f_v}, \boldsymbol{o_u}, \boldsymbol{o_v}]$	[718.86,718.86,608.19,186.22]	[719.02,718.85,608.19,186.25]
Camera's Extrinsic		
$\boldsymbol{P_{BC}^{B}}$ $(\boldsymbol{m})$	[1.10, 0.32, -0.75]	[1.10, 0.31, -0.75]
Euler$\{\boldsymbol{q_C^B}\}$ $(\boldsymbol{deg})$	[89.72, 0.80, 89.50]	[89.93, 0.26, 90.65]

Since the proposed PSO-tuned MSCKF VIN system is a classic filtering-based estimation system with an AI nature for tuning, a comparison with various navigation techniques in a similar class has been drawn. In this comparison, stereo visual and visual-inertial fusion techniques with loop-closure and global pose optimization have been included as a reference solution. Hybrid techniques employing classic and DNN-based techniques have also been considered for a fair comparison. Following Visual and Visual-Inertial navigation systems have taken part in this experiment.

- **ORB-SLAM2-S V-SLAM:** Stereo Visual-SLAM method with Loop-Closure correction and global Bundle Adjustment (BA) optimization on the whole sequence [85].
- **ORB-SLAM2-S+MSF-EKF VI-SLAM:** Loosely-coupled fusion of Stereo ORB-SLAM [85] with INS using the Multi-Sensor Fusion EKF (MSF-EKF) framework [86]. The results of this method have been reported from [87].
- **Deep-EKF VIO**: Fusion of Robo-centric EKF-based INS estimates with updates obtained from a DNN-based monocular visual odometry module [87].
- **VIOLearner (RGB-IMU) VIO**: DNN-based monocular VIO using multi-level pair-wise visual convolutional processing [88]**.**

The comparison in Table 3. 9 shows that the proposed method in this thesis, although tuned only on small sections of the sequence, outperforms DNN-based approaches that were trained on the full sequence. Moreover, the comparison shows a comparable accuracy compared to the ORB-SLAM2-S method, which performs computationally expensive stereo processing and a global Bundle Adjustment optimization on the whole sequence.

Table 3. 9. Accuracy comparison on KITTI "Sequence-07". Translation and rotation errors are normalized by the travelled distance.

Methods	Translation Error (%)	Rotation Error ($\mathbf{deg/100}m$)
ORB-SLAM2-S [85]	0.50	0.28
ORB-SLAM2-S+MSF-EKF [86]	1.73	0.48
Deep-EKF [87]	0.93	0.31
VIOLearner [88]	0.84	0.66
Proposed Enhanced MSCKF	0.57	0.16

3.3.2 Enhanced MSCKF VIN Versus Conventional MSCKF VIN

Compared to the original formulation of the VIN problem in the MSCKF framework introduced in [19], the state vector in this thesis has been augmented with extra states to model the IMU's measurements and the Camera's projection more accurately. In the original MSCKF VIN system, IMU's measurements were assumed to be affected only by static biases; however, in this work, IMU's biases and scale factors were modelled with GM stochastic process to model the time-varying and dynamic nature of the IMU's errors. Moreover, the IMU's mounting misalignment has also been considered as the system's states. Furthermore, in this work, the camera's intrinsic and extrinsic parameters have been included in the state vector to be estimated along with the navigation states. To analyze the impact of these modifications, an experiment was conducted to compare the accuracy of the enhanced MSCKF VIN system proposed in this research with the original version of the MSCKF VIN system that has only IMU bias as hidden states [19]. This experiment was conducted on KITTI "Sequence-06". This sequence is a 1217-meter loop that the vehicle travels in 113 seconds. All design choices were the same for both MSCKF VIN systems except the state vector. The fixed part of the state vector in the two systems are as follows:

MSCKF[19]: $\boldsymbol{X}_{fixed} = [P_{LB}^{L}, v_{LB}^{L}, q_{B}^{L}, b_g, b_a]_{16\times1}$ (3.1)

This Research: $\boldsymbol{X}_{fixed} = [P_{LB}^{L}, v_{LB}^{L}, q_{B}^{L}, b_g, b_a, s_g, s_a, q_{I}^{B}, x_{C_{ints}}, x_{C_{exts}}]_{37\times1}$, (3.2)

For a fair comparison, both systems were tuned separately on the whole sequence with the tuning technique introduced in this chapter. Figure 3. 18 shows the horizontal position estimate comparison of the original and the enhanced MSCKF VIN systems. The RMSE of navigation states estimates in Table 3. 10 shows that the enhanced MSCKF VIN system achieved higher accuracy on position, velocity, and attitude by 70%, 53% and 3%, respectively.

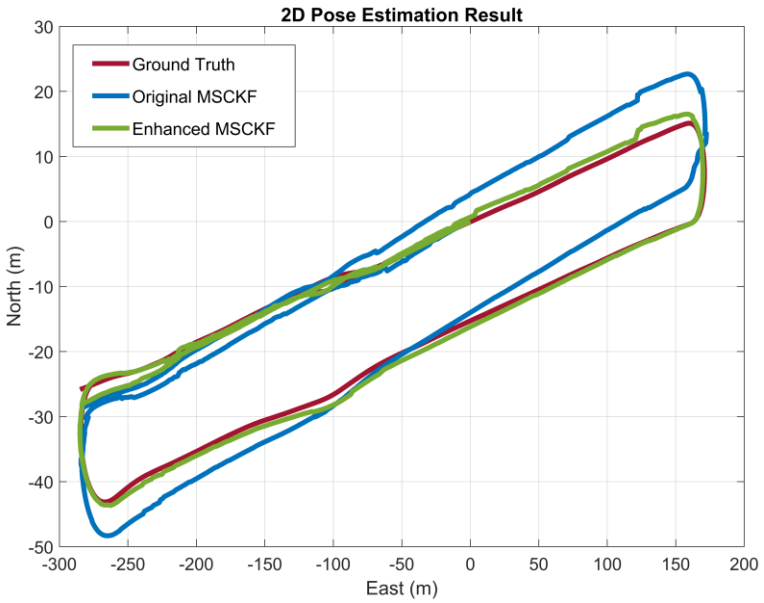


Figure 3. 18. Horizontal position estimate comparison between the original and enhanced MSCKF VIN systems on the KITTI "Sequence-06". Both MSCKF VIN systems were PSO-tuned on the whole sequence. The vehicle starts at coordinate (0,0), and the direction is anti-clockwise.

Table 3. 10. The RMSE of navigation states estimates by the original and Enhanced MSCKF VIN system on KITTI "Sequence-06".

	Enhanced MSCKF VIN	**Original MSCKF VIN** [19]
Position NED ($\boldsymbol{m}$)	[0.88, 0.64, 0.69]	[3.76, 1.53, 1.55]
Velocity ($\boldsymbol{m/s}$)	[0.10, 0.17, 0.08]	[0.28, 0.36, 0.05]
Attitude $\boldsymbol{r, p, A}$ ($\boldsymbol{deg}$)	[0.08, 0.14, 0.76]	[0.18, 0.27, 0.73]

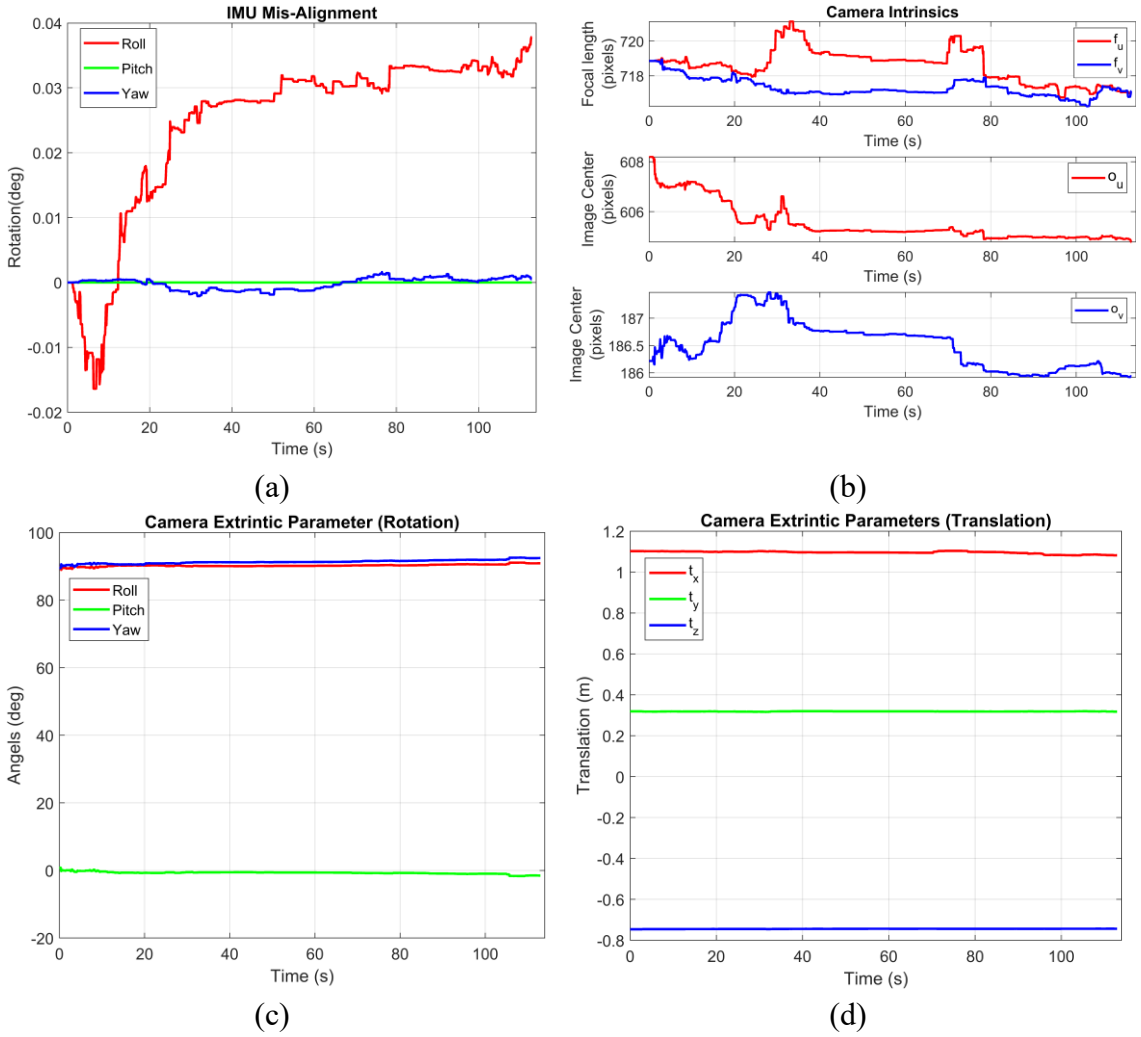


Figure 3. 19. Estimation of the (a) IMU's misalignment, (b) camera's intrinsic, and (c, d) extrinsic parameters by the enhanced MSCKF VIN system on the KITTI "Sequence-06".

Figure 3. 19 and Table 3. 11 show the estimates of the IMU's misalignment and the camera's intrinsic and extrinsic parameters in this experiment. Although the estimated values and reference values reported in the KITTI dataset show small differences; however, the accumulated impact of these hidden states led to improved accuracy. As shown in the figure and table, the camera's intrinsic parameters, which directly relate to noisy pixel-level observations, show considerably larger variations compared to the reference values reported in the KITTI dataset [84]. Accuracy comparison with other visual and visual-

inertial navigation systems on KITTI "Sequence-06" is also provided in Table 3. 12. Results show that the PSO-tuned MSCKF VIN outperforms other visual and visual-inertial navigation systems.

Table 3. 11. The estimation of the IMU's misalignment and camera's intrinsic and extrinsic parameters on the KITTI "Sequence-06" compared to the original values reported in the KITTI dataset [84]**.**

	KITTI Dataset [84]	**MSCKF VIN Estimation**
IMU's Misalignment ($\boldsymbol{deg}$)	[0.0,0.0,0.0]	[0.038, -6.6e-7, 6.3e-4]
Camera's Intrinsics (pixel) $[\boldsymbol{f_u, f_v, o_u, o_v}]$	[718.86,718.86,608.19,186.22]	[716.90,717.10,604.90,185.90]
Camera's Extrinsic		
$\boldsymbol{P_{BC}^{B}}$ $(\boldsymbol{m})$	[1.10, 0.32, -0.75]	[1.08, 0.32, -0.74]
Euler$\{\boldsymbol{q_C^B}\}$ $(\boldsymbol{deg})$	[89.72, 0.80, 89.50]	[90.87, -1.51, 92.43]

Table 3. 12. Accuracy comparison on KITTI "Sequence-06". Translation and rotation errors are normalized by the travelled distance.

Methods	**Translation Error (%)**	**Rotation Error** $(\boldsymbol{deg/100m})$
ORB-SLAM2-S [85]	0.51	0.15
ORB-SLAM2-S+MSF-EKF [86]	1.51	0.27
Deep-EKF [87]	2.14	0.16
Original MSCKF	0.62	0.11
Proposed Enhanced MSCKF	0.18	0.11

3.4 Summary

In this chapter, initially, a flexible simulation environment for the IMU-GNSS TC integration system was developed. In this simulation environment, the performance of the IMU-GNSS TC integration system can be thoroughly evaluated under different controlled error and noise conditions. Accuracy analysis in multi-sensor navigation systems is not confined to navigation states (i.e. position, velocity and attitude), and therefore, the simulation environment also provides the user with detailed information about the hidden states of the system (e.g. IMU biases and GNSS receiver clock bias). The main contribution

of the chapter was an automatic tuning technique based on the GA that optimally tunes the parameters of the IMU-GNSS integration systems (23-dimensional state space with 56 tunable parameters). To guarantee the generalizability of the tuned parameters, the selection of the optimal parameters was according to the Training-Validation-Test policy that avoided overfitting to the training data. Experimental results on a real dataset collected by our developed logger system were provided to show the efficacy and generalizability of the proposed tuning technique.

In the second part of this chapter, a similar tuning approach based on the PSO algorithm was adopted for tuning the proposed enhanced MSCKF VIN system (37+7N dimensional state space with 59 tunable parameters). Compared to the IMU-GNSS integration system, the MSCKF VIN raised new challenges, including higher-dimensional state and search spaces, and more sensitivity of the visual-inertial fusion system as a dead-reckoning navigation system. Experimental results on a real publicly available dataset showed that first, the enhanced MSCKF VIN system outperformed the original MSCKF VIN system on the navigation accuracy, and second, the proposed tuning technique can provide a set of parameters for optimal operation of the developed MSCKF VIN system.

It should be mentioned that the Kalman filter parameters tuning technique proposed in this section is an offline process. Therefore, GA or PSO do not change the Kalman filter parameters during the test scenario to avoid any potential risk to the filter stability. In sufficiently similar test conditions to the training condition, the selected parameters would still be suitable for the optimal operation of the Kalman filter; however, in noisy situations where rare outlier measurements might impose the risk of poor performance or even instability to the Kalman filter, outlier rejection techniques such as the Chi-Square test and

GN covariance test employed in MSCKF VIN, would protect the fusion system from divergence. In situations where the stochastic behaviour of the system is considerably different from that of the training dataset, if sub-optimal performance of the EKF might jeopardize its stability, fine-tuning is inevitable.

Chapter 4: End-To-End Learning Approach to VIN Problem

In the previous chapter, the automatic parameters tuning of EKF-based IMU-GNSS and VIN systems were considered. It was assumed that the analytical model of the fusion system is available; however, these models are characterized by stochastic parameters that need to be carefully tuned. In this chapter, the feasibility of automatically learning the whole Visual-Inertial fusion in a pure data-driven and end-to-end learning fashion is considered. The main motivation behind approaching the visual-inertial navigation from the learning aspect originates from the results of the previous chapter, where despite the simplicity of GA and PSO algorithms, the proposed data-driven tuning technique was shown to be very effective in adjusting a relatively small set of parameters of the EKF-based fusion system. The question that this chapter tends to answer is that given sufficient raw data and powerful learning capabilities of neural networks, how to formulate a fully data-driven approach toward motion estimation problem in an end-to-end fashion without the need for analytical models of the system and sensor calibration parameters. The next question that this chapter seeks answers to is that to what extent the expectations of achieving more generalizable models, less sensitivity to system parameters, and less design effort will be realized when the visual-inertial fusion problem is approached from a completely data-driven, model-free, and end-to-end learning technique.

The Visual-Inertial Odometry (VIO) problem fundamentally involves visual and temporal processing that deep convolutional and recurrent neural networks have been known to be capable of learning [53][55]. Moreover, because of the learning capabilities of these deep neural networks, given appropriate and sufficient training data, the potential correlations

within and between modalities can be learned automatically. Hence, in this chapter, a Deep Convolutional Recurrent Neural Network (DCRNN) approach has been adopted to perform Visual-Inertial fusion for navigation purposes. In the proposed deep neural network-based approach, as shown in Figure 4. 1, a CNN is employed to process the stream of image pairs. Meanwhile, an RNN is employed to process the sequence of IMU measurements. The outputs of these two single-modality networks are then fused and fed to the second RNN to estimate the relative pose between the two images.

The training data for the proposed VIO system can be provided by the IMU-GNSS integration technique discussed in the previous chapter. Once the results of the VIO neural network achieve satisfactory performance, then it can take over and estimate the pose when GNSS becomes unavailable. However, as expected, the VIO solution gradually drifts; therefore, whenever GNSS measurements become available again, it can correct the drift.

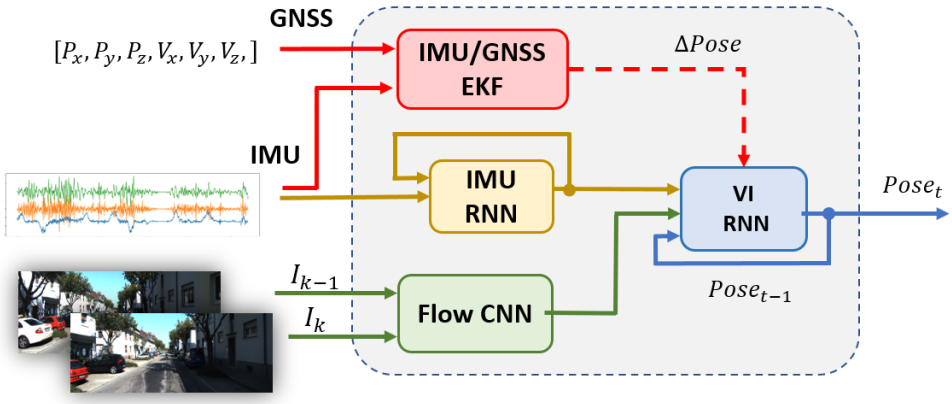


Figure 4. 1. High-level architecture of the proposed Visual-Inertial odometry neural network.

4.1 Related works

Despite geometry-based VIO techniques, learning-based techniques are new paradigms and have recently attracted the attention of researchers and engineers. Research on this new paradigm was pursued after preliminary learning-based works on fundamental computer vision and odometry tasks that will be reviewed in the order in this section. Optical flow is a computer vision technique that estimates the field of motion velocities of pixels/patches in an image caused by the relative motion of the observer and the scene [89]. The optical flow estimation was approached with deep CNN in [90], where the proposed network, called FlowNet, was trained on a pair of images and their corresponding ground truth flow field as the target. Despite training the network on a synthetic dataset, the network was able to generalize to real datasets and achieved competitive accuracy. Depth estimation from a single image using deep CNN was proposed in [91], where two CNNs designs were used to perform a coarse-scale and a fine-scale depth estimation. The former network was applied globally on the whole image to estimate the overall depth; whereas, the latter network acted locally to fine-tune the depth estimation. PoseNet [92] employed a deep CNN to perform the re-localization task of a single image from a previously seen environment or objects. Pose estimation was formulated as a regression problem with a training dataset containing individual images taken at different known poses along a trajectory. However, the main drawback of the proposed network was that the network's re-localization capability was limited to the scene or objects used in the training set. To solve the problem of pose estimation in previously unseen scenes, authors of [93] employed a similar architecture, but in contrast to PoseNet, the CNN was trained with pairs of images taken from two different camera poses as the input and the corresponding relative pose as

the output. However, the applications of that work were limited to the relative pose estimation, and no motion estimation was targeted in that work. Eventually, in the DeepVO network [94], the relative pose estimation was used in the context of Visual Odometry (VO). In that work, in addition to deep CNNs, RNNs were also used to handle the temporal processing that is required in the odometry task. The network was trained in an end-to-end learning fashion with consecutive raw images as the input and the relative pose as the output. However, that work was a vision-only motion estimation system, and no fusion of multi-sensory data was performed. In [95], a learning-based inertial odometry network, IONet, was proposed. IONet formulated the inertial odometry problem as a pose regression problem in polar coordinates using RNNs. It was shown that the same architecture could be used for a wide range of motion patterns, such as on a wheeled platform or in a hand-held bag carried by a person; however, all inertial motion estimation systems are prone to divergence. Visual-Inertial odometry, as a multi-modal learning problem, was proposed in [24], where the proposed network, VINet, employed the FlowNet to estimate the optical flow embeddings. The raw IMU measurements were passed through LSTM to obtain the inertial embeddings. The concatenation of inertial and visual encodings was then passed through LSTM layers to estimate the final pose. In [96], a masking layer was also trained on top of inertial and visual embeddings to control their contribution in the pose estimation step. The contribution control of the embeddings was performed by either soft fusion (i.e. weighted average) or hard fusion (i.e. binary ON or OFF). The authors of [96] concluded that in fast motions, the network relied more on inertial modality by masking visual modality, whereas, in large translation, visual features were masked less.

4.2 Visual-Inertial Odometry Neural Network

In this section, an end-to-end learning-based approach toward the VIO problem is proposed. Figure 4. 2 shows two time steps or two cells of the proposed VIO neural network. The proposed neural network is mainly comprised of a 2-layer LSTM (SLTM-1 and LSTM-2) processing the raw IMU data at IMU's sampling rate (i.e. 100 Hz), a pre-trained FlowNet [90] to estimate the optical flow embeddings at camera frame rate (i.e. 10 fps), and another 2-layer LSTM (LSTM-3 and LSTM-4) fusing the two modalities and estimating the relative pose at the same rate as the camera frame rate. Table 4. 1 shows the detailed specifications of different layers in the proposed neural network.

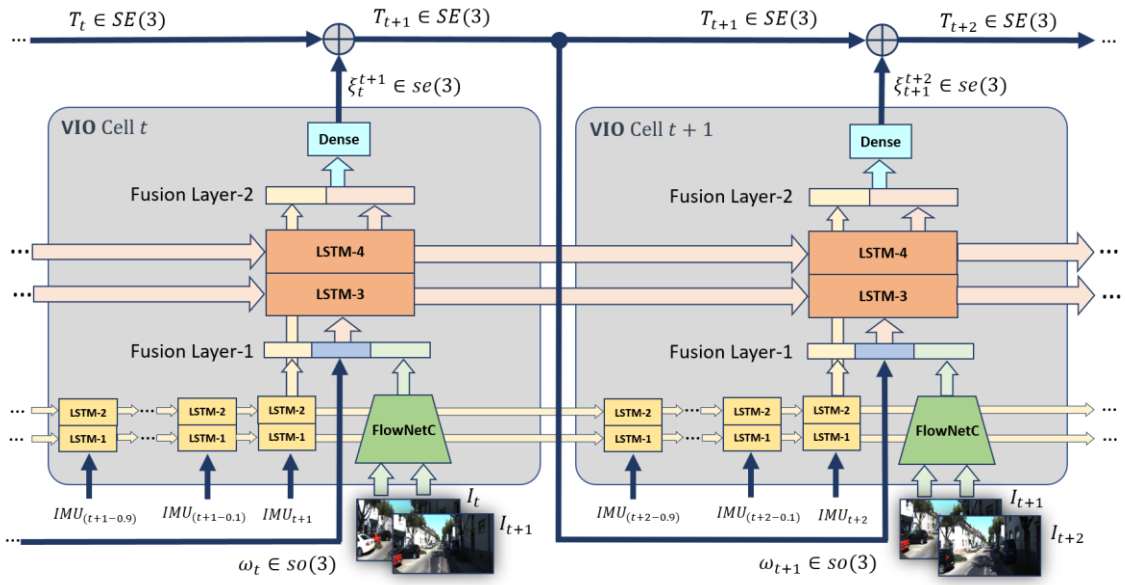


Figure 4. 2. Two VIO cells of the proposed VIO neural network architecture.

Table 4. 1. Detailed specifications of the VIO Net's layers.

Layers	Output Size	Number of Trainable Parameters	Activation Function
IMU Input	6	0	-
IMU Dense	64	448	$tanh$
LSTM-1	512	1181696	$tanh$ and $sigmoid$
LSTM-2	512	2099200	$tanh$ and $sigmoid$
LSTM-2 Dense	512	262656	$relu$
Attitude Input	3	0	-
Attitude Dense	64	256	$tanh$
Image Input	$1392 \times 512 \times 3$	0	-
FlowNet	122880	0 (pretrained)	-
FlowNet Dense	1024	125830144	$tanh$
LSTM-3	1024	10752000	$tanh$ and $sigmoid$
LSTM-4	1024	8392704	$tanh$ and $sigmoid$
Pose Dense (multilayers)	512, 256, 6	786944, 131328, 1542	$relu$, $relu$, and $linear$
Pose Output	6	0	-

The pre-trained CNN network used in the proposed neural network was the first six convolutional layers of a pre-trained FlowNetC [90]. There are two fusion layers in the proposed VIO cell. The first fusion layer combines the optical flow embeddings with inertial embeddings (the output of every 10 LSTM-2 Dense cells). The second fusion layer takes the inertial embeddings and fuses them with the output of the LSTM-4 before the final pose estimation by the fully connected layer (i.e. Pose Dense). Each VIO cell receives the following inputs:

1. Two consecutive image frames,
2. Ten IMU samples between images,
3. The current estimate of the attitude directly obtained from the previous cell, and
4. All the incoming hidden/cell states of the LSTMs from the previous VIO cell.

The output from each VIO cell is the relative pose estimate and all the outgoing hidden/cell states of the LSTMs to the next VIO cell. It should be mentioned that only during the training phase, the attitude input to each VIO cell comes from the ground truth pose, but in the test phase, each VIO cell receives the current Attitude estimated by the previous VIO cell feed-forwarded to the current VIO cell. The IMU measurements fed to the VIO cells are in their very raw format without any preprocessing or error corrections. Moreover, neither the FlowNetC network nor the VIO cell receives any camera calibration information (e.g. intrinsic or extrinsic parameters) in any form. The pose estimates in the VIO cell are performed in the Lie Algebra $se(3)$, rather than Lie Group, $SE(3)$ [97]. This preference is mainly because pose estimation in Lie Group should respect orthogonality and unity constraint on the rotation matrix that is difficult to enforce in neural networks; whereas, Lie Algebra performs the transformations in a lower-dimension and linear space with no constraints. More information about Lie Algebra and Lie Group can be found in [97].

Compared to the work in [24], the proposed VIO neural network in this thesis is different from the following aspects:

1. As it was justified in section 2.3, the current estimate of relative translation and rotation only depends on the previous attitude and not the previous position. Hence, the proposed VIO cell in this work receives only a 3-element attitude from the previous VIO cell and not the full 6-Element pose. To the best of the authors' knowledge, the 3-Element absolute position is redundant and can lead to overfitting the training set.

2. Since the KITTI odometry dataset used in this work is a considerably small dataset for a deep learning algorithm, both colour cameras of the stereo set were used as two separate monocular systems, and this added more diversity to the dataset, which consequently led to better test versus training performance.
3. The inertial data in this network is fused twice, once with the visual data before the LSTM-3 and once after LSTM-4 before the final pose estimation. This extra fusion layer at the output refines the final pose estimation and improves the performance of the VIO system in dynamic scenes.

4.3 Experimental Work

The proposed VIO neural network was implemented using Keras library with a Tensorflow backend [98] in Python programing language. The developed VIO neural network was trained on the KITTI odometry dataset [83]. This dataset contains 22 sequences (Named "Sequence-00" to "Sequence-21") collected in urban, highway, and rural areas by a vehicle equipped with two stereo colour and grayscale cameras, and a laser scanner, as well as an IMU-GNSS positioning system with RTK technology with positioning error less than 5cm. Out of these 22 sequences, raw IMU data is only available for ten sequences ("Sequences-00" to "Sequence-10" except "Sequence-03"). The colour stereo cameras stream at 10fps with 1392 × 512 pixels resolution. The baseline between the stereo pairs (named "Cam-2" and "Cam-3" in the KITTI dataset) is 0.54m. Camera frames are synchronized with the IMU, which samples at 100Hz and has a 3-axis accelerometer and a 3-axis gyroscope.

Preparing the dataset for training the LSTMs requires some considerations in terms of GPU memory management and training efficiency. Sequences in the KITTI dataset vary in

length from a few tens of seconds to a few minutes. Long sequences such as “Sequences-00” and “Sequence-02”, with all image frames and IMU samples, do not entirely fit in the GPU memory as a single training sample. Therefore, in this work, the sequences were sliced into 2-second mini-sequences, which contained 20 samples of poses and optical flow embeddings and 200 samples of IMU measurements. However, enough consideration has been made about the hidden states of the last LSTM cells in each mini-sequence to be correctly passed to the initial hidden states of the first LSTM cells in the next mini-sequence within an original sequence. Figure 4. 3 shows how hidden states have been fed-forward in the network.

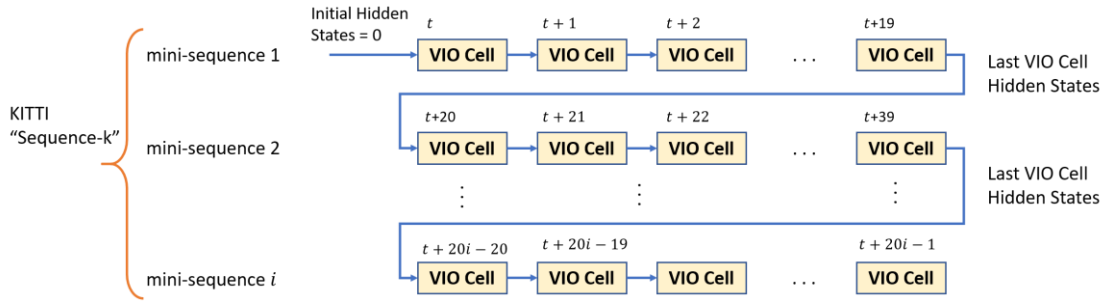


Figure 4. 3. Splitting KITTI Sequences into mini-sequences and properly feeding forward hidden states.

The KITTI odometry dataset for a deep learning approach is considerably small. Hence the network is prone to overfit the training set. One approach to avoid overfitting is to use regularization techniques such as $L1$ and $L2$ norms or dropout techniques [99], but in this work, both cameras of the stereo set (Cam-2 and Cam-3) were used as two separate monocular systems, which increased the diversity in the training set. Hence, each sequence in the KITTI dataset creates two sequences for the training called IMU/Cam-2 and IMU/Cam-3 in this chapter. The test set in this experiment contained “Sequence-07” (both cameras) and “Sequence00” (only Cam-3). The remaining sequences were used as the

training set. The objective function is the mean square error of relative translation and rotation as follows:

$$L = \sum \beta_v \|v - \hat{v}\| + \beta_\omega \|\omega - \hat{\omega}\|, \tag{4.1}$$

where β_v and β_ω are two weights to first, scale the translational and rotational error terms to the same order of magnitude and, second, to balance their contribution in the total objective function.

The proposed VIO neural network has more than 149,000,000 trainable parameters in total. The training and test were performed on NVIDIA® Tesla P100 that features 12GB of RAM and 3584 CUDA® cores, running at 1190MHz. The proposed VIO neural network was trained for 120 epochs in total, and each epoch took approximately 20 minutes on NVIDIA® Tesla P100. A forward pass of the proposed network to estimate and compose the full pose for one time step takes 12.2ms on the Tesla P100 GPU, which means the network can run and provide the pose estimates at 82fps. Figure 4. 4 shows the training and test loss of the proposed VIO neural network. To prevent overfitting the proposed deep neural network architecture in training on large datasets, dataset splitting into training, validation, and test can be employed, which suggests training should be stopped when objective function value on the validation set starts to worsen while the same quantity on the training set is still improving [52].

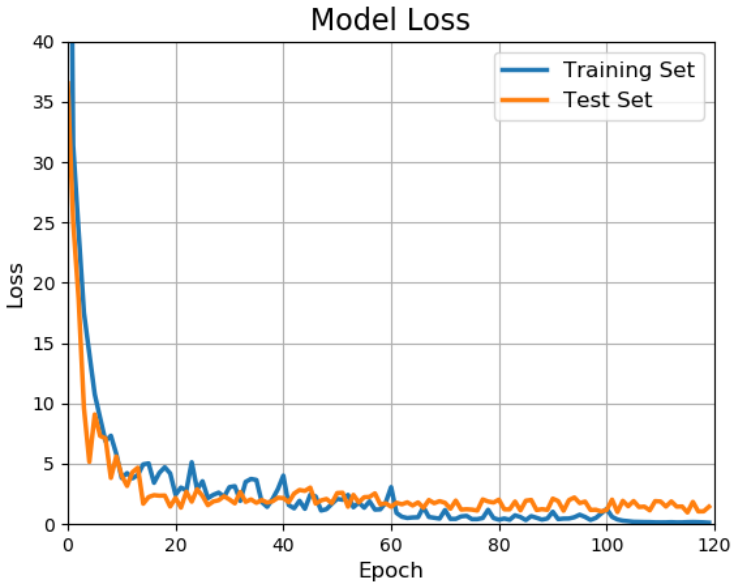
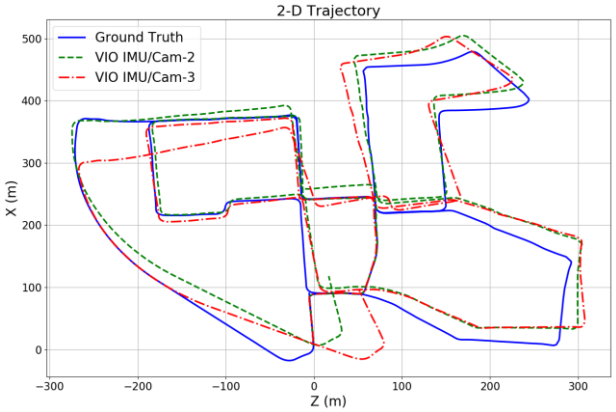


Figure 4. 4. Training and test loss over 120 epochs.

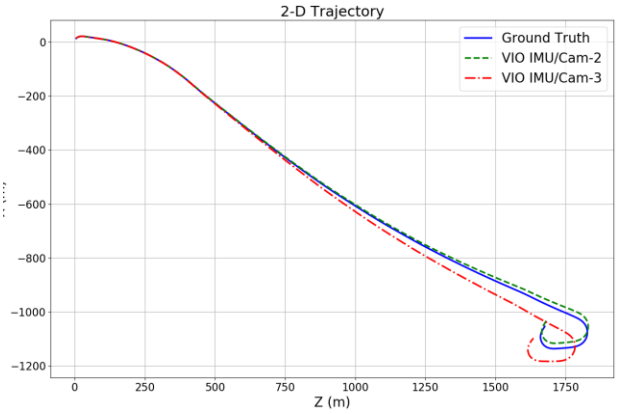
Figure 4. 5 shows the pose estimation results of the proposed VIO neural network on selected sequences of the KITTI dataset. Translation and rotation RMSE of all sequences in the KITTI dataset have been reported in Table 4. 2. “Sequence-07” (IMU/Cam-2 and IMU/Cam-3) and “Sequence-00” (only IMU/Cam-3) were used as test sequences, and other sequences were used as training sequences. In this experiment, during the first 10 seconds of the sequences, the ground truth attitude was provided for the VIO neural network to let the hidden states of the LSTMs be initialized to appropriate values. After the first 10 seconds, the VIO system operated under GNSS outage, and the relative pose estimated by the network at each time step was fed to the next time step as the input. The KITTI dataset contains sequences with a wide variety of motion patterns, including low and high speeds (e.g. “Sequence-01”), long stationary periods (e.g. “Sequence-07”), dynamic scene (e.g. “Sequence 07”), sharp and smooth turns (e.g. “Sequences-00, -02 and -08”).



“Sequence-00”
Duration: 454 seconds
Length: 3724 meters

RMSE of Position (m):
IMU/CAM-2: [14.87, 12.80, 13.22]
IMU/CAM-3: [22.23, 10.25, 27.39]

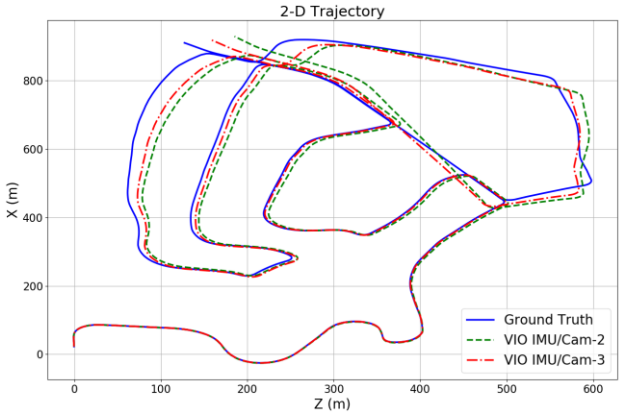
RMSE of Rotation (deg):
IMU/CAM-2: [2.13, 3.77, 2.35]
IMU/CAM-3: [2.48, 8.81, 2.76]



“Sequence-01”
Duration: 110 seconds
Length: 2453 meters

RMSE of Position (m):
IMU/CAM-2: [11.52, 60.55, 7.60]
IMU/CAM-3: [38.38, 64.60, 26.54]

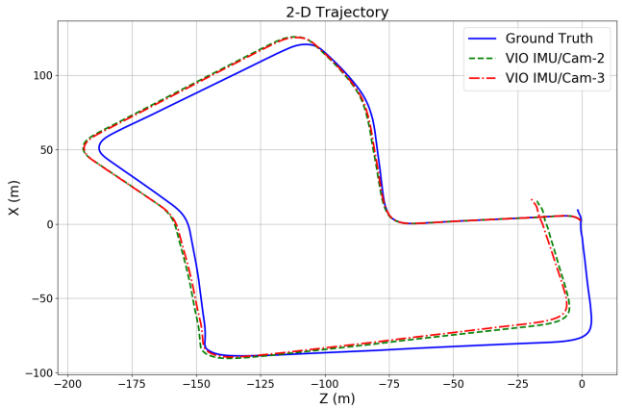
RMSE of Rotation (deg):
IMU/CAM-2: [4.34, 1.06, 3.42]
IMU/CAM-3: [3.89, 2.87, 2.37]



“Sequence-02”
Duration: 466 seconds
Length: 5067 meters

RMSE of Position (m):
IMU/CAM-2: [15.99, 21.06, 20.78]
IMU/CAM-3: [9.96, 27.14, 15.31]

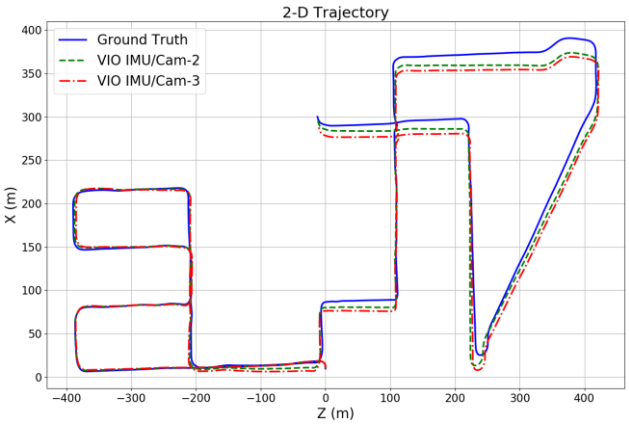
RMSE of Rotation (deg):
IMU/CAM-2: [4.01, 4.39, 4.40]
IMU/CAM-3: [4.71, 3.06, 4.55]



<u>"Sequence-07"</u>
Duration: 110 seconds
Length: 649 meters

RMSE of Position (m):
<u>IMU/CAM-2: [6.27, 3.97, 5.39]</u>
<u>IMU/CAM-3: [6.67, 5.34, 6.47]</u>

RMSE of Rotation (deg):
<u>IMU/CAM-2: [1.82, 4.03, 2.31]</u>
<u>IMU/CAM-3: [2.14, 4.60, 2.68]</u>



"Sequence-08"
Duration: 407 seconds
Length: 3222 meters

RMSE of Position (m):
IMU/CAM-2: [6.09, 12.83, 7.56]
IMU/CAM-3: [8.39, 10.94, 10.01]

RMSE of Rotation (deg):
IMU/CAM-2: [2.27, 1.65, 2.47]
IMU/CAM-3: [2.14, 1.67, 1.86]

Figure 4. 5. The 2D map and pose estimate errors on the selected sequence of the KITTI dataset. The test sequences have been underlined.

Table 4. 2. Translation and rotation RMSE estimated by the proposed VIO neural network on the KITTI dataset. Test sequences have been underlined.

Sequence		Position RMSE (m)	Rotation RMSE (deg)
Sequence-00 454 seconds 3724 meters	**IMU/CAM2**	[14.87, 12.80, 13.22]	[2.13, 3.77, 2.35]
	IMU/CAM3	[22.23, 10.25, 27.39]	[2.48, 8.81, 2.76]
Sequence-01 110 seconds 2453 meters	**IMU/CAM2**	[11.52, 60.55, 7.60]	[4.34, 1.06, 3.42]
	IMU/CAM3	[38.38, 64.60, 26.54]	[3.89, 2.87, 2.37]
Sequence-02 466 seconds 5067 meters	**IMU/CAM2**	[15.99, 21.06, 20.78]	[4.01, 4.39, 4.40]
	IMU/CAM3	[9.96, 27.14, 15.31]	[4.71, 3.06, 4.55]
Sequence-04 27 seconds 393 meters	**IMU/CAM2**	[0.25, 0.46, 1.46]	[0.33, 0.16, 0.13]
	IMU/CAM3	[0.13, 0.32, 1.70]	[0.23, 0.13, 0.12]
Sequence-05 276 seconds 2205 meters	**IMU/CAM2**	[5.16, 4.14, 4.44]	[1.25, 1.20, 1.20]
	IMU/CAM3	[4.04, 2.32, 4.53]	[1.00, 1.23, 0.99]
Sequence-06 110 seconds 1232 meters	**IMU/CAM2**	[4.77, 1.74, 2.86]	[0.60, 2.10, 0.95]
	IMU/CAM3	[3.83, 2.27, 2.30]	[0.65, 1.80, 1.04]
Sequence-07 110 seconds 649 meters	**IMU/CAM2**	[6.27, 3.97, 5.39]	[1.82, 4.03, 2.31]
	IMU/CAM3	[6.67, 5.34, 6.47]	[2.14, 4.60, 2.68]
Sequence-08 407 seconds 3222 meters	**IMU/CAM2**	[6.09, 12.83, 7.56]	[2.27, 1.65, 2.47]
	IMU/CAM3	[8.39, 10.94, 10.01]	[2.14, 1.67, 1.86]
Sequence-09 159 seconds 1705 meters	**IMU/CAM2**	[4.99, 5.88, 5.36]	[1.50, 1.58, 1.33]
	IMU/CAM3	[2.59, 5.26, 3.18]	[1.73, 1.13, 1.448]
Sequence-10 120 seconds 919 meters	**IMU/CAM2**	[3.95, 18.42, 4.22]	[2.92, 1.40, 1.46]
	IMU/CAM3	[5.95, 12.66, 5.70]	[2.56, 1.98, 1.19]

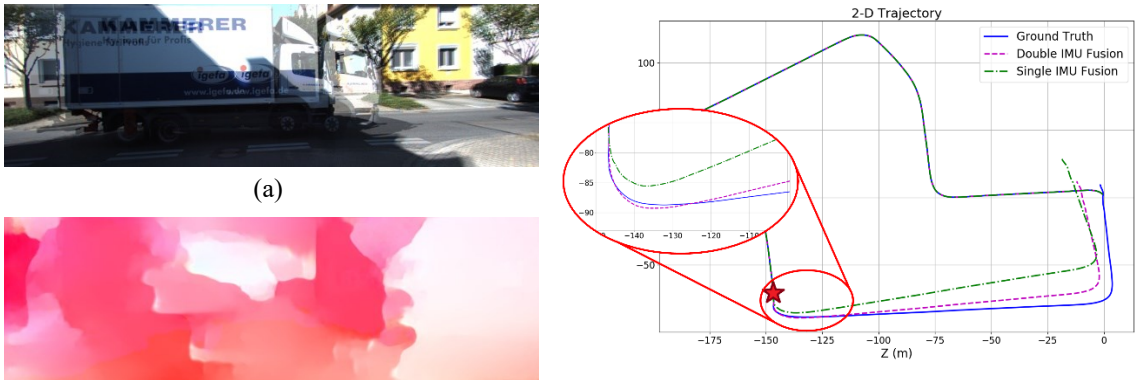
As shown in Figure 4. 5, the proposed VIO neural network successfully learned how to fuse visual and inertial modalities to perform the odometry task on a wide range of motion patterns. The dominant motion in the KITTI dataset is straight driving; however, the accuracy of the VIO neural network on rare motion patterns such as high-speed driving, dynamic scene, smooth and sharp turns confirms that the VIO neural network was able to generalize its learned knowledge to those motion patterns as well. However, it should be

mentioned that, on average, the translation error along the y axis is slightly large compared to the other two axes. This is mainly because, in the KITTI odometry dataset, the vehicle does not experience enough excitation along the gravity vector, and as a result, the VIO neural network was not trained on visual and inertial patterns that correspond to that type of motion. A similar observation holds for rotation along x and z axis that the VIO neural network has a slightly large error. Table 4. 3 shows the RMSE of the position and attitude estimates averaged on all training and test sequences normalized by the travelled distance.

Table 4. 3. The average of the position and attitude RMSE on the training and test sequences normalized by travelled distance.

	Position RSME (%)	**Attitude RMSE** ($\boldsymbol{deg/100m}$)
Training Sequences Average	[0.36, 0.71, 0.37]	[0.11, 0.09, 0.08]
Test Sequences Average	[0.86, 0.57, 0.85]	[0.23, 0.52, 0.28]

To better show the effectiveness of employing the second fusion layer in the proposed VIO neural network, a similar network without the second fusion layer was designed and trained with the same training conditions. The performance of both VIO neural networks was compared in a test case scenario in "Sequence-07". As sown in Figure 4. 6, the challenging situation in this sequence happens at an intersection (marked by a red star on the 2D map) where the vehicle moves forward at a very low speed, but a large truck is also moving to the right, which covers a considerably large area of the frame. A vision-only odometry algorithm will be easily tricked in dealing with this scenario because the optical flow shows a legitimate left ego-turn; however, a VIO algorithm should be able to resolve this ambiguity with IMU measurements.



(a)

(b)

(c)

Figure 4. 6. The effect of employing an extra fusion layer in the proposed VIO neural network. (a) The colour camera frames showing a truck moving to the right, (b) The optical flow of the associated frames, (c) The horizontal position estimate of single and double fusion VIO neural networks.

Figure 4. 6 shows how employing the second fusion layer has made the proposed VIO neural network more robust to this dynamic scene scenario. For a fair comparison, both networks received the ground truth input pose until the intersection. Without the second fusion layer, the VIO solution suffers from $10.66(deg)$ deviation from the ground truth trajectory after 100 meters forward motion, while the proposed VIO neural network with both fusion layers suffers only $4.1(deg)$ deviation.

4.4 VIO Neural Network Versus PSO-tuned Enhanced MSCKF VIN

In this section, a comparison in terms of accuracy, design effort, and computational complexity between the proposed VIO Neural Network and the PSO-tuned enhanced MSCKF VIN is provided. Despite a similar learning/tuning nature of both approaches, there are fundamental differences in the state estimation and fusion technique employed in these two visual-inertial navigation paradigms. Hence, the comparison in this section tends to draw a qualitative comparison by highlighting the weaknesses and strengths of each approach.

The PSO-tuned enhanced MSCKF VIN system discussed in the previous chapter is a model-based visual-inertial fusion system that performs the state estimation in the EKF framework. As a model-based system, the enhanced MSCKF VIN system highly depends on the accurate system and measurement models derived analytically by navigation system experts. System and measurement models were characterized by stochastic parameters that were tuned using the PSO algorithm. Even though deriving the models is a one-time process and can be done in a short time, the automatic PSO-tuning is time-consuming (e.g. 5-10 hours). The state vector in the enhanced MSCKF VIN system has the size of $(37 + 7N)$, where N is the number of tracked body poses (parameter N varied between 4 and 12 in our experiment). Hence, if F represent the number of features involved in the update step (parameter F varied between 10 and 40 in this experiment), the computational complexities of the state/covariance propagation and update steps of the enhanced MSCKF VIN system are of the order of $(37 + 7N)^3$ and $F \times (37 + 7N)^2$, respectively [63]. Therefore, on average, if $F = 25$ and $N = 8$, one epoch of the enhanced MSCKF VIN system (ten steps state/covariance propagation followed by one step update) involves approximately 8 million multiplications. It is also worth mentioning that the enhanced MSCKF VIN system is a highly sequential algorithm and is often executed serially on processors.

On the other hand, the VIO Neural Network discussed in this chapter takes a data-driven approach and does not require experts to derive the analytical model of the system. Using only raw sensory data, the VIO Neural Network learns how to perform multi-modal fusion in a navigation problem in a model-free and end-to-end learning fashion. The training phase of the VIO Neural Network has some similarities to the PSO tuning of the enhanced

MSCKF VIN's stochastic parameters; however, in the VIO Neural Network, the whole fusion model, as well as its parameters, are learned automatically in the training phases. Therefore, the training process of the VIO Neural Network takes longer to complete (e.g. few days). In terms of computational complexity, there are more than 150 million weights in the VIO Neural Network that must be multiplied to the inputs in every epoch; however, these operations are highly parallelizable and can be efficiently performed on many-core architectures such as on GPUs.

Figure 4. 7 and Table 4. 4 quantitatively compare the pose estimation accuracy of these two fusion algorithms on KITTI "Sequence-07". It is worth discussing that the KITTI "Sequence-07" was a test sequence in the VIO Neural Network, and therefore, it was not shown to the network during the training phase; whereas, the stochastic parameters of the enhanced MSCKF VIN were tuned on the first third section of this sequence. The enhanced MSCKF VIN was already equipped with an accurate analytical model that describes how state estimation using visual-inertial data should work, and then the enhanced MSCKF VIN was further guided by the PSO algorithm, in a data-driven fashion, to tune its scholastic parameters; whereas, the VIO neural network learned this knowledge from scratch, and generalized it through in an extensive training phase on other sequences of KITTI dataset.

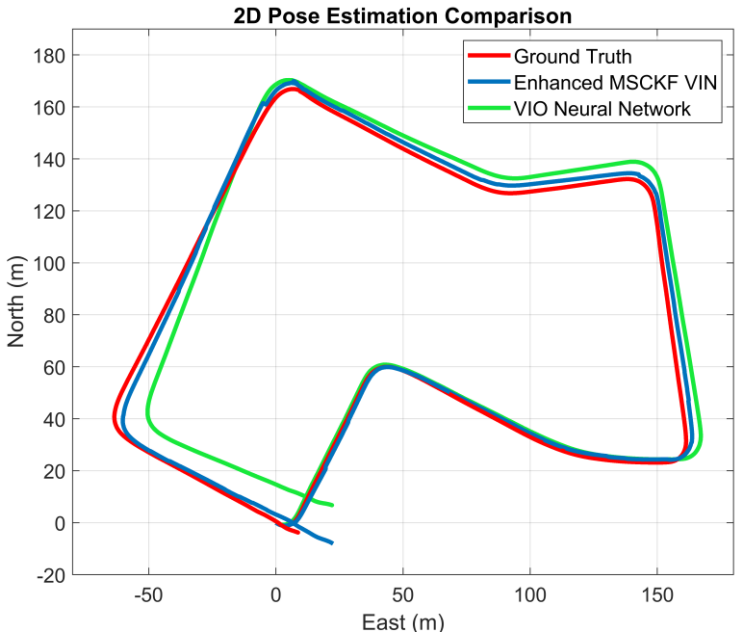


Figure 4. 7. Horizontal position estimate comparison between the VIO Neural Network and PSO-tuned MSCKF VIN on KITTI "Sequence-07".

Table 4. 4. The RMSE of navigation states estimates of the VIO Neural Network and PSO-tuned MSCKF VIN evaluated on the KITTI "Sequence-07".

	VIO Neural Network	**Enhanced MSCKF VIN**
Estimation RMSE		
Position NED ($\boldsymbol{m}$)	[6.27, 5.39, 3.97]	[1.84, 3.42, 0.69]
Attitude $\boldsymbol{r, p, A}$ ($\mathbf{deg}$)	[1.82, 2.31, 4.03]	[0.10, 0.11, 1.08]

One might be interested in comparing the accuracy of these two navigation algorithms on a sequence on which both fusion systems were trained/tuned. KITTI "Sequence-06" was already used as one of the training sequences in the VIO neural network, and in the previous chapter, the results of the enhanced MSCKF VIN system tuned on the whole length of this sequence was also reported. As shown in Figure 4. 8 and Table 4. 5, the PSO-tuned enhanced MSCKF VIN system outperforms the VIO neural network; however, it should be mentioned that the VIO neural network was trained on a dataset containing eight sequences other than the "Sequence-06"; therefore, the VIO neural network was not

overfitted to this sequence; whereas, the PSO-tuned enhance MSCKF VIN was tuned on only this sequence and on the whole length of this sequence.

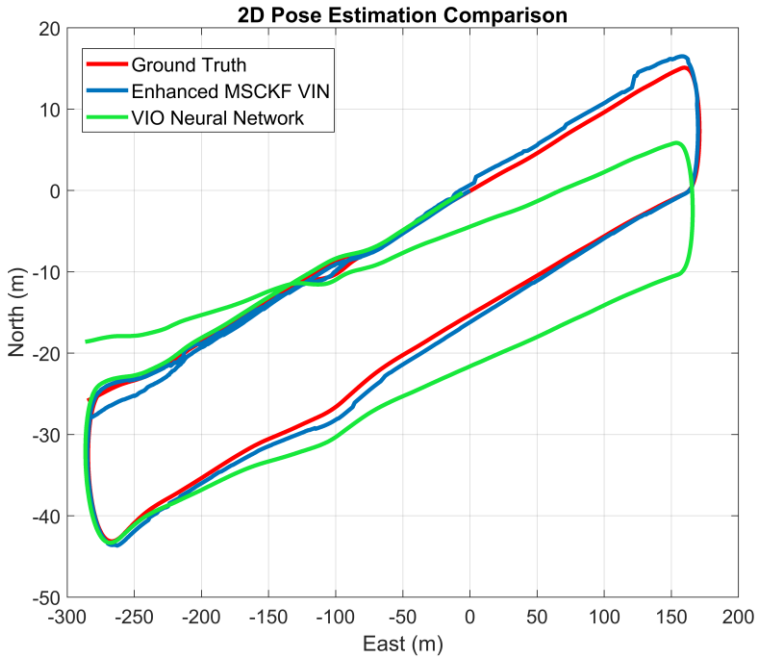


Figure 4. 8. Horizontal position estimate comparison between the VIO Neural Network and PSO-tuned MSCKF VIN on KITTI "Sequence-06".

Table 4. 5. The RMSE of navigation states estimates of the VIO Neural Network and PSO-tuned MSCKF VIN evaluated on the KITTI "Sequence-06".

	VIO Neural Network	Enhanced MSCKF VIN
Estimation RMSE		
Position NED ($\boldsymbol{m}$)	[3.83, 2.30, 2.27]	[0.88, 0.64, 0.69]
Attitude $\boldsymbol{r, p, A}$ **(deg)**	[0.65, 1.04, 1.80]	[0.08, 0.14, 0.76]

4.5 Summary

In this chapter, the Visual-Inertial fusion for odometry and navigation purposes was reformulated into an end-to-end learning problem. The objective of this reformulation was to make the design process fully automatic without any need for the model of the system or the sensors' calibration parameters. CNNs were employed to perform the visual processing, and RNNs were employed to perform the temporal processing. These two types

of networks were structured in a deep multi-modal neural network, and it was trained on nine sequences of the KITTI odometry dataset and was tested on one of the challenging sequences, “Sequence-07”. The experimental results showed that the proposed DCRNN learned, in an end-to-end learning fashion, how to fuse visual and inertial modalities to estimate the translation and rotation image by image. The proposed network achieved 1.2% horizontal translation error and 0.52 ($deg/100m$) heading errors with respect to the travelled distance on the test sequence that was not shown to the network in the training phase.

It is worth discussing that, although the proposed VIO neural network does not outperform the conventional model-based VIN systems (e.g. MSCKF VIN) in terms of accuracy on a particular sequence, the VIO neural network is a very powerful learning system that learned the sophisticated process of fusing multi-modal data for motion estimation from scratch by training on only 35 minutes of raw visual-inertial odometry training dataset. Moreover, it learns this general knowledge in a model-free and almost effortless design process. However, it should also be mentioned that adaptation of a model-based system to a different sensor setup is more straightforward than adapting a neural network-based system. A very few specific parameters must be changed in a model-based system to adjust it to a different sensor setup; whereas, a neural Network-based system is a black box, and the sensor setup is not modelled in a small and confined set of parameters/weights.

Chapter 5: Embedded Real-Time Implementation of VIN System

As discussed in the first chapter, the autonomous vehicles' stability and mission accomplishment tie with the continuous estimation of the vehicles' navigation states at a high rate. From the discussions in previous chapters, however, it was concluded that high accuracy and robustness in VIN systems are better achieved in TC integration schemes that, inevitably, imposes high computational requirements. Hence, implementing a real-time TC VIN system on embedded computing platforms is quite challenging and requires efficient implementation of the VIN's pipeline at hardware and software levels.

Between the two VIN systems developed in this Ph.D. research (i.e. the VIO neural network and the MSCKF VIN), the MSCKF VIN system better satisfied the accuracy and efficiency targeted in this work; therefore, the developed MSCKF VIN system was selected for real-time embedded implementation. The pipeline of the MSCKF VIN system is comprised of different processing modules that not only carry out compute-intensive tasks but also respect tangled data dependencies in the pipeline. Hence, the barriers to achieving real-time performance for the developed MSCKF VIN system on embedded computing platforms are twofold.

A. The conventional implementation of the MSCKF VIN system is a sequential pipeline in which the serial data flow inefficiently stalls dependent processing modules.
B. Regarding the large number of visual features that must be processed in every epoch of the filter, inefficient implementation of the visual backend module accumulatively imposes a high computational burden.

These two bottlenecks directly affect the per-epoch processing time and consequently jeopardize the overall real-time performance.

The first section of this chapter reviews related works on embedded real-time implementation of the VIN systems. It surveys the topic from a broad perspective addressing the problem from different hardware and software levels.

The next section of this chapter addresses the first barrier in front of the real-time implementation of the developed MSCKF VIN system. First, a more in-depth insight into the pipeline of the MSCKF VIN system is provided, and then the performance of the pipeline is accurately benchmarked. Analyzing this performance profile is the key to identify the pipeline's bottlenecks. This analysis reveals the potential spots in the sequential pipeline where it can be safely and efficiently broken into balanced parallel threads without compromising the original pipeline's integrity, synchrony, and data dependency and without compromising the modularity of the software architecture.

The following section of the chapter addresses the second barrier in the real-time implementation of the developed MSCKF VIN system. This section targets one of the time-consuming and compute-intensive modules of the pipeline, Feature Localization, for further performance optimization. This module is one of the most frequently invoked modules in the pipeline that directly affects the per-epoch processing time. It is first discussed how the features 3D localization can be parallelized, and then two parallel implementations on the GPU are proposed.

The two design optimization solutions introduced in this chapter, aided by the compiler optimization tool, show that the proposed MSCKF VIN parallel pipeline implemented on a CPU-GPU-enabled embedded system achieves 38% faster per-epoch processing time

compared to the conventional sequential pipeline and satisfies strict real-time constraints with ~50ms margin. In the last section of the chapter, a hardware-in-the-loop (HIL) simulation is provided to further evaluate and demonstrate the proposed parallel implementation in more challenging conditions. This HIL simulation considers additional limitations of the MSCKF VIN system, such as latency and frame rate of the camera and delays in transmitting the navigation solution to other sub-systems for visualization or subsequent processing.

5.1 Related works

The hardware implementation of VIN systems has been addressed from different perspectives in the literature. A comprehensive performance comparison of a wide range of VIN system implementations has been provided in [37]. The benchmarking includes the accuracy, processing time, and memory usage of the state-of-the-art VIN systems, including MSCKF [19], OKVIS [21], ROVIO [100] VINS-Mono [22] running on four different computing platforms, including Lenovo ThinkPad W540 laptop, Intel NUC desktop PC, UP Board embedded system, and ODROID embedded systems. It should be mentioned that the proposed implementation of the MSCKF VIN in this thesis shares a similar program structure with the open-source implementation of the MSCKF VIN that was originally introduced in [101] and used in [37]. However, the proposed implementation in this thesis is distinguished from the following aspects:

A. The open-source MSCKF VIN system proposed in [101] has been implemented in Robotics Operating System (ROS) environment; whereas, the developed MSCKF VIN system in this thesis does not depend on ROS for processing and data sharing between

the modules. The ROS independence makes the proposed implementation portable to embedded systems not supporting the ROS.

B. The open-source MSCKF VIN system proposed in [101] is a single-thread process; whereas, in the work presented in this thesis, the pipeline of the MSCKF VIN has been parallelized on two threads that enjoys 41% faster per-epoch processing time compared to the single-thread implementation of this work.

C. The developed MSCKF VIN pipeline presented in this work has been accelerated by a visual backend module running on GPU that speeds up the per-epoch processing time by 33%.

To the best of the author's knowledge, the processing time benchmarking presented in this work and the proposed multi-thread implementation of the MSCKF VIN system with the GPU-accelerated visual backend is the first of its kind in the literature.

Authors of [102] reformulated the TC VIN problem in the framework of a Cubature Kalman Filter (CKF) [103]. To better handle the nonlinearity in the system, CKF directly works with a set of sample points and propagates them through nonlinear functions to probabilistically approximate the mean and covariance estimates. Hence, CKF has a great potential for parallelization on multi-core architectures since each sample can be processed independently on a separate core. The CKF does not require computing the high dimensional filter gain and innovation matrix inverse contributing to the overall computational cost. The CKF-based fusion framework developed in [102] has also made it possible to redesign the measurement model to eliminate the need for the direct 3D localization of the features using a technique called Trifocal Tensor Geometry. This technique directly transfers feature points to consecutive images, and it was reported that

the Trifocal Tensor Geometry technique saves approximately 25ms per feature to compute the residuals of the CKF. However, the proposed CKF-based VIN in [102] was implemented in MATLAB software. The parallelization was done automatically using the Parallel Processing Toolbox of MATLAB software, and no embedded hardware implementation was targeted.

An on-board and high-speed LC VIN system was proposed in [104] for micro aerial vehicle attitude control. In that work, four parallel threads were dedicated on an Odroid U3 embedded computing platform to perform the Feature Extraction, Feature Tracking, EKF prediction, and EKF Update. However, no explanation about the parallelization of the pipeline was provided. To achieve high-speed control and navigation at 50Hz, the authors of [104] also made two simplifying design decisions: integration in the loosely-coupled scheme and tracking a small set of features (i.e. 40 features per image) at VGA resolution over only pairs of images. However, these critical design decisions were made because the operation condition of the aerial robot was restricted to a small-sized and static environment for a short period.

Authors in [105] implemented an optimization-based Visual-Inertial SLAM algorithm on a System-on-Chip (SoC) device, XILINX® Zynq-7020. This SoC device features a dual ARM® Cortex-A9 CPU and an Artix-7 FPGA. The intention to use an FPGA was to directly interface with CMOS image sensors of the camera and offload the CPU from low-level image processing. An important optimization in the visual fronted on the FPGA was done by performing the feature processing arithmetic in a fixed-point format. Comparison of the CPU-only and CPU-FPGA implementations of the feature processing modules (detection, extraction, and matching) showed that the CPU-FPGA implementation enjoyed

faster processing time than the CPU-only implementation with ~11ms versus ~33ms. However, the work in [105] focused only on the low-level image processing visual frontend without targeting the core fusion algorithm, including the features 3D localization that feeds the filter updates. In addition, the optimization on the FPGA level is often designed for a specific system environment. This hardware-specific optimization is difficult to modularize and generalize to other platforms.

Over the past few years, integrated circuits for VIN systems attracted considerable attention. Authors in [106] proposed a low-power and high-throughput integrated system optimized jointly at hardware and algorithm levels. One innovative optimization in that work is the efficient computation of structured and unstructured sparse data in both the visual frontend and optimization backend. However, the most effective optimization was applied to the customized stereo visual frontend that contains two independent image processing units with dedicated memory and arithmetic units. Despite some programmability features of the proposed hardware (e.g. frame resolution and number of features per frame), the customized integrated design offers less flexibility for future modifications. It also comes at a higher cost compared to implementation on general-purpose processors such as CPUs and GPUs.

In [107], a novel local maximum search technique (known as non-maxima suppression in the computer vision community) has been introduced upon which a GPU implementation of highly parallelizable FAST features [42] detection algorithm has been proposed. The GPU-accelerated FAST feature detection was then integrated into the visual frontend of an optimization-based VIN system. Experimental results showed an average speed-up of

factor 2.25 compared to a CPU-only implementation in feature detection-extraction on embedded and desktop CPU-GPU-enabled computing platforms.

In addition to the optimization at the fusion engine and visual features processing levels, optimization of the PSO algorithm has also been reported in the literature. In [108], the execution speed of two GPU implementations of the PSO algorithm has been compared with that of a sequential implementation on a CPU. The first variant of PSO is a synchronous PSO that updates the global optima synchronously at the end of each iteration/generation for all particles. The second variant is asynchronous PSO that adopts a ring topology and only locally informs the particles about the local optima. The speed-up of the GPU implementation over CPU implementation for synchronous PSO was shown to reach up to 30 times; whereas, the GPU implementation of the asynchronous variant of the PSO algorithm was shown to achieve up to 45 times faster execution time than the CPU implementation. In [109], the authors proposed a GPU implementation of Multi-Objective PSO for large swarm and high-dimensional optimization problems. Evaluations on four two-objective high-dimensional synthesized benchmarks showed that the expected speed-up of the proposed GPU implementation (compared to CPU implementation) could be between 30 and 135 on objective functions of 32 to 1024 dimensions, respectively. However, the datasets considered in [108] and [109] were all synthesized datasets, and no practical engineering applications with real data were targeted in those works.

5.2 Multi-Thread MSCKF VIN System

In this section, first, a more in-depth insight into the MSCKF pipeline is provided. Then through accurate profiling of its processing time, the bottlenecks of the pipeline are identified, and a more efficient parallel pipeline is introduced.

5.2.1 MSCKF VIN Sequential Pipeline

Figure 5. 1 shows the conventional sequential pipeline of the MSCKF VIN system. In the rest of the chapter, the term "S-MSCKF" is used to refer to this sequential pipeline. At each epoch, State and Covariance Propagation module (PR) takes the most recent estimates and predicts the systems' states and associated error covariance matrix through time using IMU samples. Upon capturing a new image, State and Covariance Augmentation module (AG) augments the state vector and state error covariance matrix. Feature Detection and Extraction modules (FD and FE) process the new image to detect salient visual features and extract their descriptors. These descriptors are then passed to the Feature Matching module (FM) to match with those of visual features tracked from previous images. Those features tracked over multiple images and lost in the current image are passed to the Feature Localization module (FL) to be localized in the 3D space. The Kalman Update module (KU) takes the AG module's output (i.e. augmented state vector and state error covariance matrix), along with the features reprojection errors from the FL module, to apply the corrections on the states and covariances estimates. The PR module in the next epoch takes the most recent estimates and repeats this process.

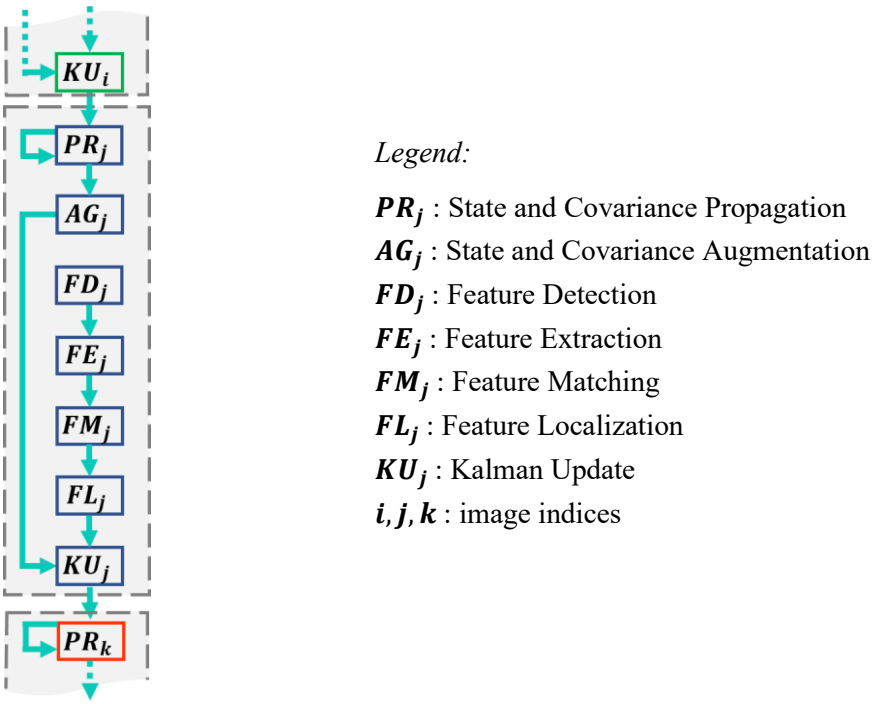


Figure 5. 1. S-MSCKF VIN pipeline and the dataflow. Epochs of the MSCKF are shown by a gray dashed rectangle, and subscripts and colours refer to the image indices

The S-MSCKF pipeline shown in Figure 5. 1 was implemented using C++ programming language in Linux Ubuntu 16.04 operating system. OpenCV (CPU libraries) was used for visual feature management [110]. Matrix operations were done using the *Eigen* library [111]. The processing time has been profiled using *gperftool* (Originally Google Performance Tools), which is a low-overhead sampling-based thread-friendly performance profiler [112]. The visualization of the processing profile has been made possible by the *KCachegrind* tool [113]. GNU Compiler Collection (GCC) has been used to compile the developed C++ programs [114]. Nvidia CUDA Compiler (NVCC) has been used to compile the developed CUDA programs [115].

To better analyze the processing time of the S-MSCKF VIN pipeline, it is essential to profile its processing time while it is running on a real embedded hardware. Hence, Figure

5. 2 visualizes the processing time profile of the S-MSCKF VIN system running on NVIDIA® Jetson TX2 tested on KITTI “Sequence-07”.

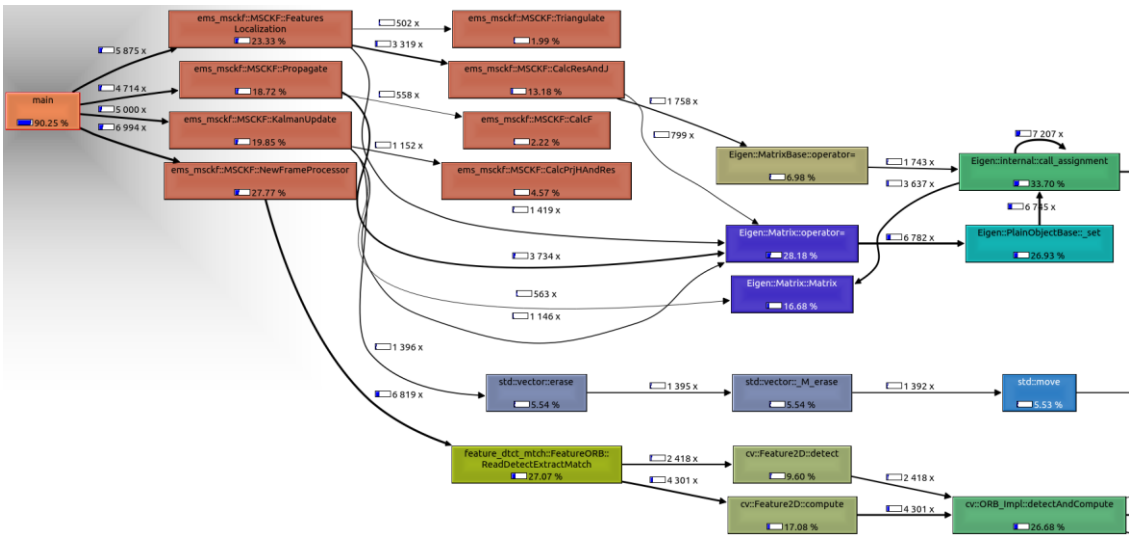


Figure 5. 2. The processing time profile visualization of the S-MSCKF VIN system running on the NVIDIA® Jetson TX2 tested on KITTI “Sequence-07”. The total execution time was 105.9 seconds, and 26381 samples were taken.

For the benefit of conciseness in Figure 5. 2, the main function callees are shown only four levels deep. In this figure, the number shown inside each box represents the percentage of the time the profiler sampled while that particular class function was being executed. Arrows connecting the boxes represent the callee-caller relationship among the class functions. The number on the arrows represents the number of samples taken in the class function pointed by the arrow. The combination of modules FD, FE, and FM (the *ems_msckf::MSCKF::NewFrameProcessor* class function in the figure) are to blame 27.7% of the total processing time. On the other hand, the FL module (the *ems_msckf::MSCKF::FeatureLocalization* class function in the figure) is solely responsible for 23.33% of the total processing time. These two modules in the S-MSCKF pipeline take approximately 51% of the total processing time. Considering that these

modules are executed serially in the sequential pipeline, one can easily acknowledge their critical role in the total processing time.

5.2.2 MSCKF VIN Parallel Pipeline

A more in-depth analysis of the S-MSCKF pipeline reveals that the FM module is the bottleneck in the pipeline. All information that the FM module must provide for the FL module is the list of lost features. Because lost features were being tracked until the current image, therefore, all information that the FL module needs (i.e. body pose states and 2D pixel observations of the features) are already available. Moreover, immediately after the FM module completes its task, FD and FE modules can start processing the next image without being stalled. Having identified the bottleneck and time-consuming modules in the S-MSCKF pipeline, a parallel pipeline with a balanced processing load is proposed. In the rest of the chapter, the term "P-MSCKF" is used to refer to this parallel pipeline.

Figure 5. 3 shows the proposed parallel pipeline. As shown in Figure 5. 3, in the P-MSCKF pipeline, the total processing of one epoch of the MSCKF is done simultaneously on two parallel threads. The figure shows that when a new image is captured, the visual frontend modules (i.e. FD, FE, and FM modules) immediately start their tasks on Thread-2. Simultaneously, on Thread-1, PR and AG can complete their tasks when the KU module, related to the previous epoch, provides them with corrected states. Once the FM module in Thread-2 completes its task, it sends a notification to the FL module in Thread-1 to start its localization task. Thread-2 is then freed to start processing the next image as soon as it is captured. When the KU module receives its required data from FM and AG modules, it starts correcting the current epoch's states. Once the correction is completed, the PR

module related to the new image starts its task, and these two threads synchronously process coming epochs.

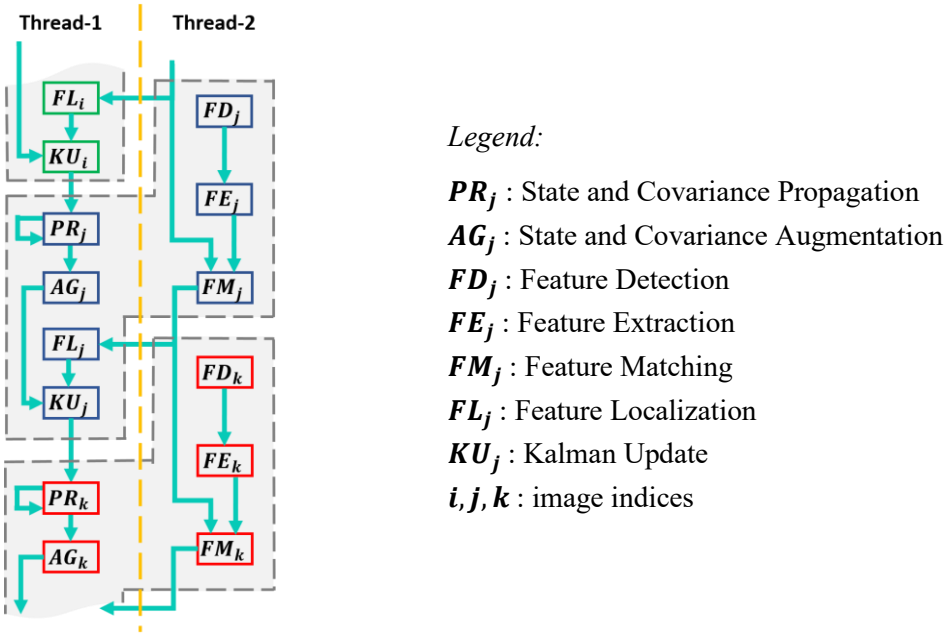


Figure 5. 3. The proposed P-MSCKF VIN pipeline running on two threads.

It should be noted that at each instance of time, Thread-1 and Thread-2 perform processing related to two MSCKF epochs. For instance, in Thread-2, when the FD module is busy detecting features in the new image, Thread-1 might still be completing the KU module related to the previous epoch. Therefore, extra considerations must be made to safely synchronize the two threads to avoid multi-thread hazards such as *lock* and *race* conditions. In particular, in each epoch of the MSCKF, the FM module should always keep the list of tracked features updated until all dependant modules safely use that information. Hence, the FM module should add and remove the features to/from the list of tracked features in a timely and synchronized manner considering other modules' dependencies. Moreover, the KU module must respect its data dependencies on the augmented states and associated covariance matrix provided by the AG module.

The proposed parallel P-MSCKF pipeline was implemented using *Thread* library in C++ programming language, and synchronization between threads was done using *Semaphore* library. Figure 5. 4 shows the processing time profile of the P-MSCKF VIN pipeline running on NVIDIA® Jetson TX2 tested on KITTI "Sequence-07". The profiler sampled 45.60% of the time in Thread-1 (*ems_MSCKF::EKFProcessor* class function in the figure) and 38.94% of the time in Thread-2 (*ems_msckf::NewFrameProcessor* class function in the figure), which indicates that the total processing was reasonably balanced on the two threads.

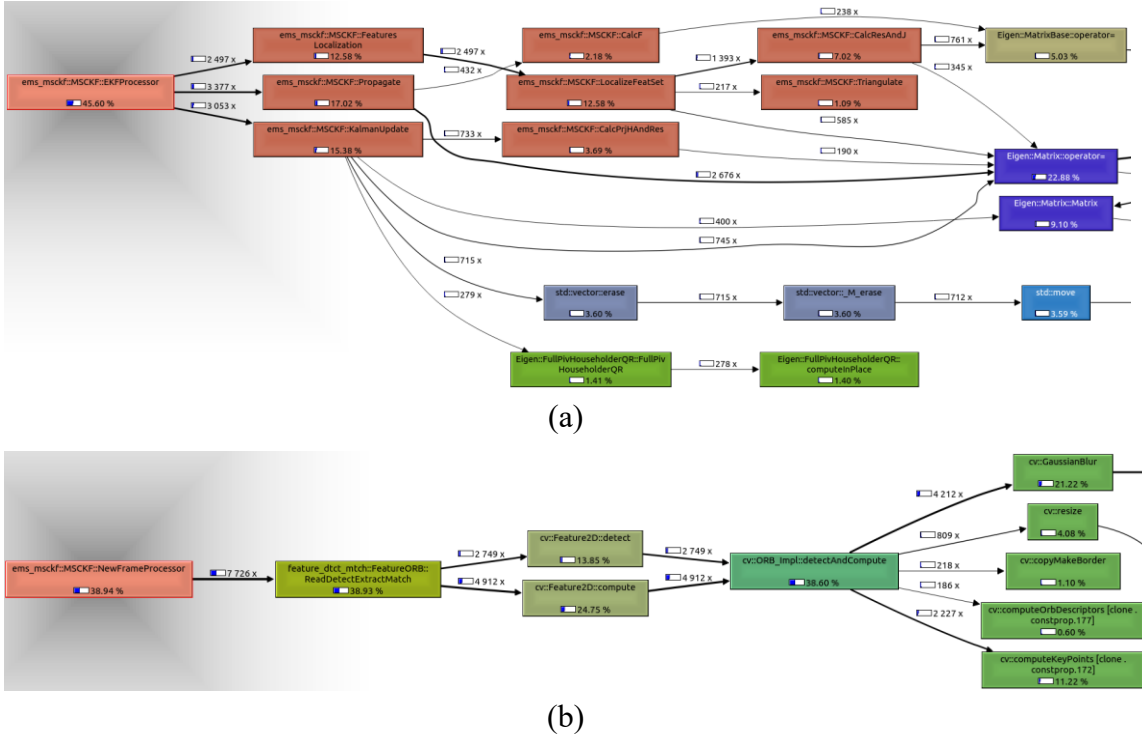


Figure 5. 4. The processing time profile visualization of the P-MSCKF VIN system running on NVIDIA® JetsonTX2 tested on KITTI "Sequence-07". (a) Thread-1. (b) Thread-2. The total execution time is 67.1 seconds, and 20490 samples were taken for this experiment by *gperftool*.

In addition to balancing the processing load on two threads, this distribution of processing modules has the advantage that the visual frontend and fusion engine are on two different

threads. This parallel pipeline design offers modularity and makes it easier to add new features to the navigation system (e.g. fusing new sensory modalities) or to integrate the current MSCKF VIN pipeline along with control and mission planning pipelines into the higher-level pipeline of autonomous systems.

5.2.3 S-MSCKF and P-MSCKF VIN Pipelines' Performance Comparison

The processing time analyses discussed in previous sections were conducted on KITTI "Sequence-07". This sequence is a 110-second sequence, and IMU samples were measured at 100Hz, and images were used at 10fps with 1104×333 resolution. Figure 5. 5 shows the horizontal position estimate by S-MSCKF and P-MSCKF VIN systems on the KITTI "Sequence-07". It is worth mentioning that the set of tuned parameters obtained in Chapter 3 for the developed MSCKF VIN system were utilized again for the experiments of this chapter with manual fine-tuning on one of the parameters, Visual Feature Noise Standard Deviation. The fine-tuning on this parameter was expected mainly due to employing different visual features compared to that in Chapter 3. In the MSCKF VIN system discussed in Chapter 3, SURF features [41] were used; whereas, for embedded implementation in this chapter, a more computationally efficient alternative features, ORB features, were used, which is one order of magnitude faster than SURF [43]. In the experiments of this chapter, 120 ORB features were detected in each image.

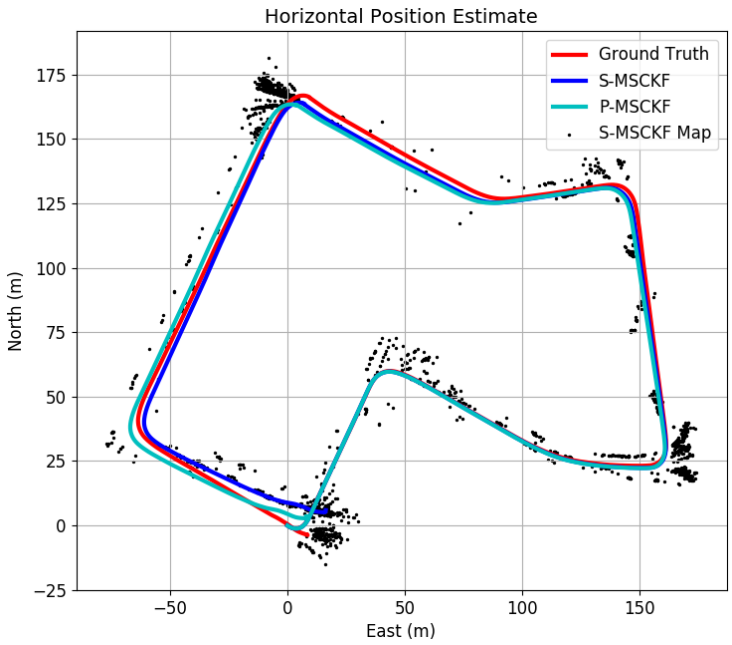


Figure 5. 5. Horizontal position estimate by S-MSCKF and P-MSCKF VIN systems on the KITTI "Sequence-07".

The S-MSCKF and P-MSCKF took 105.9 and 67.1 seconds, respectively, to process the whole sequence. Figure 5. 6 shows the per-epoch processing time statistics of S-MSCKF and P-MSCKF. Statistically, the P-MSCKF enjoys a considerably faster per-epoch processing time compared to S-MSCKF. The figure also shows that more than 75% of P-MSCKF epochs were completed faster than 60ms; whereas, more than 50% of S-MSCKF epochs took more than 90ms to complete. Figure 5. 7 shows the epoch processing time of the S-MSCKF and P-MSCKF VIN systems for the whole duration of KITTI "Sequence-07". Regarding the camera's frame rate (i.e. 10fps), there is a 100ms interval between any two images for the MSCKF to complete the epoch processing. The epochs processed slower than the camera frame rate violate the strict real-time constraints. In the P-MSCKF VIN system, less than 3.5% of epochs processed slower than the camera frame rate; whereas, in the S-MSCKF, 30% of epochs could not keep up with the camera frame rate.

Table 5. 1 compares the navigation states RMSE and processing time statistics of these two implementations.

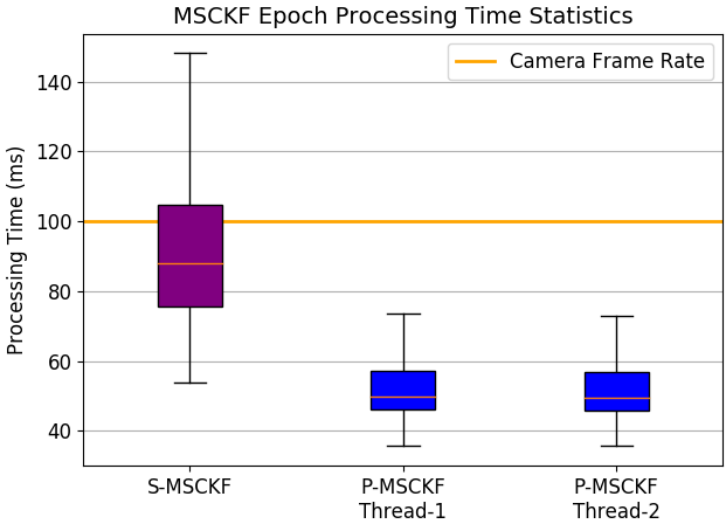
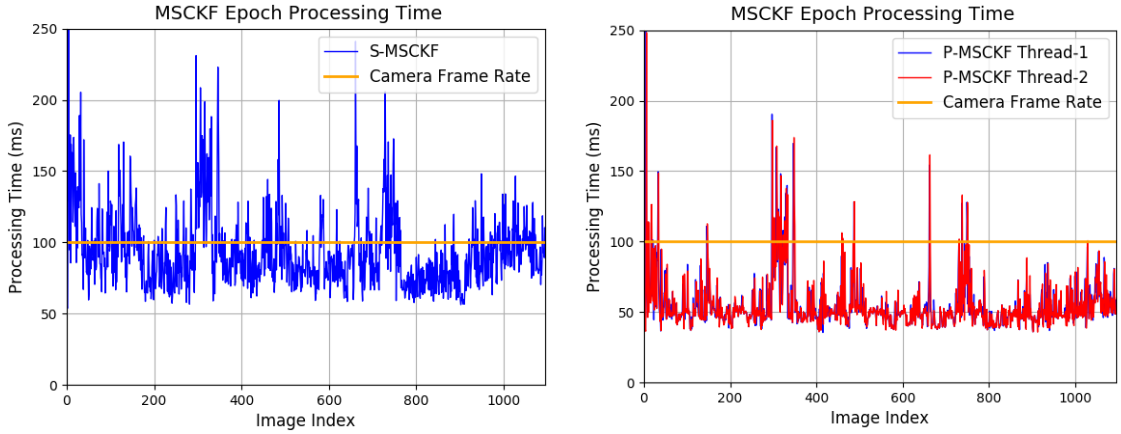


Figure 5. 6. Epoch processing time statistics of S-MSCKF and P-MSCKF pipelines.



(a) (b)

Figure 5. 7. Epoch processing time of (a) S-MSCKF and (b) P-MSCKF VIN systems on the KITTI "Sequence-07".

Table 5. 1. Navigation States RMSE and processing time comparison of S-MSCKF and P-MSCKF VIN systems on KITTI "Sequence-07".

	S-MSCKF	P-MSCKF
Position RMSE NED ($\boldsymbol{m}$)	[2.44, 2.91, 0.63]	[2.23, 3.10, 0.86]
Attitude RMSE $\boldsymbol{r, p, A}$ $(\boldsymbol{deg})$	[0.07, 0.11, 0.52]	[0.10, 0.12, 0.72]
Total processing time $(\boldsymbol{s})$	105.9	67.1
Epoch processing time average $(\boldsymbol{ms})$	93.75	55.07
Epoch processing standard deviation $(\boldsymbol{ms})$	27.00	16.31
Real-time Constraint Violations (%)	30.27	3.45

As shown in Figure 5. 7, the epoch processing time is not consistent for all epochs. One of the contributing factors in epoch processing time is the number of features that must be processed in that epoch. For example, in KITTI "Sequence-07", there is a stationary period before the vehicle turns at intersections. What makes this scenario computationally burdensome is that during this time, consecutive images captured by the camera highly overlap, and therefore the features detected in one image have a high chance to be successfully matched in the following images. When these large sets of matched features are required to be processed, a high computational burden is imposed on the pipeline, specifically on the FL and KU modules. Two bold examples of these scenarios happen at image indices 300 and 700 in Figure 5. 7 that corresponds to dense regions of the built map in Figure 5. 5 at coordinates $(160\ E, 25\ N)$ and $(0\ E, 170\ N)$, respectively.

In the vision-based navigation community, different solutions have been proposed for these computationally challenging situations. One common solution is Keyframe Selection, which suggests keeping only images that are sufficiently different from the last selected image [116]. However, excessive discarding of images might make the navigation system susceptible to losing track of valuable visual features that may eventually lead to divergence. The other common practice is to employ image buffers to temporarily store the

new image until it can be processed. This technique is most effective when buffering is rarely required, as in the P-MSCKF, where only 3.5% of images require buffering. An alternative solution is to detect stationary periods with GNSS or wheel odometer and then limit the number of detected features to avoid overloading the pipeline. While these solutions are applicable in many applications, this work has sought alternative design and implementation optimization solutions that are reliable in more frequent challenging scenarios.

One of the general solutions to optimize the codes is at the compilation time when high-level codes are translated to low-level machine codes. GCC offers optimization tools that scan the code for opportunities to improve the code size, memory usage, and/or execution time [114]. Common speed optimization techniques in the GCC optimization tool are as follows:

Strength Reduction_ This technique replaces expensive operators with simpler ones. For instance, the cost of division and multiplication operators even in modern processors is higher than subtraction and additions, but in some applications, the same results can be achieved with simpler instructions.

Inlining_ This optimization technique replaces a call to a frequently called function with the body of that function to avoid the overhead of repetitive jumps and to save stack operations. Inlining also helps the compiler to see more code at once and apply further optimizations.

Memory Access Optimization_ This technique optimizes the speed in consecutive or large memory access by data alignment and auto-increment/decrement of the memory address.

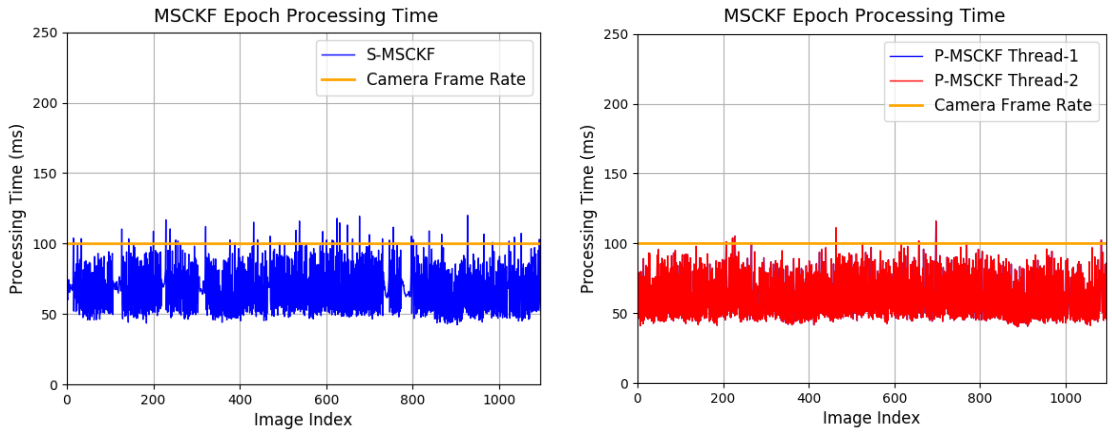
Loop Unrolling_ This optimization technique replicates the code inside the loop body to reduce the number of iterations and, consequently, the number of condition checks and jumps.

Common Subexpression Elimination_ This optimization technique simplifies a series of expressions by merging or factoring instructions. This optimization technique is most effective in the *Eigen* library used for the algebraic operations of this work.

It should be mentioned that the GCC optimization and the *gperftool* are not compatible. GCC optimization tool employs techniques that sometimes involve reordering the instructions (e.g. Common Subexpression Elimination) within and among class functions to find opportunities for further simplifications. This reordering interferes with the sampling nature of the *gperftool* that requires a predefined mapping between samples and their memory addresses associated with class functions.

Figure 5. 8 shows the per-epoch processing time of S-MSCKF and P-MSCKF pipelines optimized by the GCC optimization tool. One key observation in this figure is that the average epoch processing time of the GCC optimized S-MSCKF pipeline shows a considerable decrease compared to the original S-MSCKF and close to the GCC optimized P-MSCKF. The justification for this observation is that when the GCC optimization tool sees the whole code in one thread, it discovers more opportunities for optimization compared to a multiple-thread pipeline where more constraints and data dependencies must be respected. This observation also suggests that the contribution of the GCC in optimization was more effective on the fusion engine modules that are heavily based on the *Eigen* library. This is further supported by the reference documents of the *Eigen* library that have mentioned that the GCC optimization tool can speed up matrix operations by a

factor of ten [111]. Table 5. 2 compares the navigation accuracy and processing time statistics of GCC optimized S-MSCKF and P-MSCKF VIN systems. While both implementations enjoy a relatively small percentage of strict real-time constraint violations, the average and standard deviation of epoch processing time are severely close to the camera frame rate that jeopardizes strict real-time constraint satisfaction.



(a) (b)

Figure 5. 8. Epoch processing time of (a) GCC optimized S-MSCKF and (b) GCC optimized P-MSCKF VIN systems.

Table 5. 2. Navigation States RMSE and processing time comparison of GCC optimized S-MSCKF and P-MSCKF VIN systems on KITTI "Sequence-07".

	S-MSCKF GCC Optimized	**P-MSCKF** GCC optimized
Position RMSE NED ($\boldsymbol{m}$)	[2.68, 2.88, 0.66]	[2.23, 3.07, 0.84]
Attitude RMSE $\boldsymbol{r, p, A}$ ($\boldsymbol{deg}$)	[0.08, 0.12, 0.53]	[0.10, 0.12, 0.72]
Total processing time ($\boldsymbol{s}$)	71.9	69.5
Epoch processing time average ($\boldsymbol{ms}$)	68.19	63.15
Epoch processing standard deviation ($\boldsymbol{ms}$)	17.69	17.83
Real-time Constraint Violations (%)	3.9	1.0

To satisfy hard real-time constraints, all epochs must be processed strictly faster than the camera frame rate, ideally with a safe margin to guarantee real-time performance in all scenarios. One of the processing modules that is a barrier in achieving strict real-time

constraints is FL modules. In the next section, the FL module and its potentials for further optimization are discussed.

5.3 Parallel Features Localization

This section revisits the feature localization problem from algorithmic and implementation aspects to identify potentials for parallelization and more efficient implementations.

As discussed in section 2.5.4, the feature 3D localization problem is formulated in an optimization problem. The objective function and GN method's equations are provided again for the benefit of completeness of this discussion:

$$\boldsymbol{\mathcal{F}}(\theta_l) = \sum_{k=1}^{M_l} {r_{f_l}^k}^2 = \sum_{k=1}^{M_l} (z_{f_l}^k - \bar{z}_{f_l}^k)^2 = \sum_{k=1}^{M_l} (z_{f_l}^k - \boldsymbol{h}(\boldsymbol{g_k}(\theta_l)))^2, \tag{5.1}$$

$$\bar{\theta}_l^{(s+1)} = \bar{\theta}_l^{(s)} - (J_{\boldsymbol{r}_{f_l}}{}^T J_{\boldsymbol{r}_{f_l}})^{-1} J_{\boldsymbol{r}_{f_l}}{}^T \boldsymbol{r}_{f_l}{}^{(s)}, \tag{5.2}$$

$$J_{\boldsymbol{r}_{f_l}} = \frac{\partial \boldsymbol{r}_{f_l}}{\partial \theta_l}, \tag{5.3}$$

$$\boldsymbol{r}_{f_l} = \begin{bmatrix} r_{f_l}^1 & \dots & r_{f_l}^{M_l} \end{bmatrix}^T. \tag{5.4}$$

In the above equations, $\boldsymbol{\mathcal{F}}$ is the objective function as the sum of reprojection error squared, ${r_{f_l}^k}^2 = (z_{f_l}^k - \bar{z}_{f_l}^k)^2$. M_l is the number of observations of the l^{th} feature. In the GN equations, the superscript s represents the iteration index. $\boldsymbol{r}_{f_l}$ is a $2M_l \times 1$ vector containing all reprojection errors of the l^{th} feature on all images detecting it. $J_{\boldsymbol{r}_{f_l}}$ is a $2M_l \times 3$ Jacobian matrix of the reprojection error vector, $\boldsymbol{r}_{f_l}$, with respect to the optimization parameters, θ_l. It should also be noted that if in an MSCKF epoch, F number of features needs to be localized, F number of optimization problems must be solved. Therefore, the processing burden of the optimization solver accumulatively imposes a high

processing burden on the MSCKF VIN pipeline. In the next subsections, two solutions to address this problem are proposed.

5.3.1 Parallel Feature Localization Using Parallel Multi-Objective PSO

In addition to a large number of features that need to be localized in every epoch of MSCKF, the GN method requires Jacobian matrix calculation and inversion that must be done at each iteration of the optimization. One of the Jacobian-free and inversion-free optimization alternatives is the PSO algorithm that also offers more local minima immunity. PSO algorithm is highly parallelizable and involves less complicated computations per particle and per iteration. In reformulating the feature 3D localization in the PSO algorithm, the visual feature's location in the 3D space is coded as the location of the PSO's particles in the search space, and the objective function defined in (5.1) is used as the fitness function. For the benefit of completeness of the discussion, the equations of the PSO algorithm are provided again below:

$$\mathbb{v}_i(t+1) = m * \mathbb{v}_i(t) + c_1 * \mathbf{r} * \left(\mathbb{x}_{Gbest} - \mathbb{x}_i(t)\right) + c_2 * \mathbf{r} * \left(\mathbb{x}_{Pbest_i} - \mathbb{x}_i(t)\right), \quad (5.5)$$

$$\mathbb{x}_i(t+1) = \mathbb{x}_i(t) + \mathbb{v}_i(t+1), \quad (5.6)$$

where subscript i represent the i^{th} particle, $\mathbb{x}_{Gbest}$ is the location of the global/population best particle, $\mathbb{x}_{Pbest_i}$ is the location of the so-far best fitness of the i^{th} particle. $\mathbb{x}_i(t)$ and $\mathbb{v}_i(t)$ are, respectively, the location and velocity of the i^{th} particle.

The PSO algorithm for features 3D localization has great potentials for parallelization at the particles and features levels. For instance, if F number of visual features need to be localized in an MSCKF epoch, the PSO algorithm must be applied F times. However, since features are independent, the 3D localization of F number of features can be redesigned as

a single optimization problem with F-objective function. Moreover, particles in a PSO algorithm are highly independent, and their associated computation can be done concurrently. Hence, in a general case, if F represents the number of features that must be localized and π represents the number of particles that form the population of each PSO algorithm, then $F \times \pi$ particles form the population of the resulted Parallel Multi-Objective PSO (PMOPSO) algorithm. Accordingly, update equations are modified as follows:

$$\boldsymbol{fit}(\mathbb{X}) = \left[\boldsymbol{fit}\left(\mathbb{x}_{id}^{1}\right), \ldots, \quad \boldsymbol{fit}\left(\mathbb{x}_{id}^{l}\right), \ \ldots, \quad \boldsymbol{fit}\left(\mathbb{x}_{id}^{F}\right)\right], \quad id \in \{1, \pi\}, l \in \{1, F\}), \tag{5.7}$$

$$\mathbb{V}(t+1) = \ \mathrm{m} * \mathbb{V}(t) + c_1 * \mathfrak{r} * (\mathbb{X}_{Gbest} - \mathbb{X}) + c_2 * \mathfrak{r} * \left(\mathbb{X}_{pbest} - \mathbb{X}\right) \tag{5.8}$$

$$\mathbb{X}(t+1) = \ \mathbb{X}(t) + \mathbb{V}(t+1), \tag{5.9}$$

$$\mathbb{X} = [\mathbb{x}_1^1, \ldots, \mathbb{x}_\pi^1, \ldots, \mathbb{x}_{id}^l, \ldots \mathbb{x}_1^F, \ldots, \mathbb{x}_\pi^F]_{3\times(F*\pi)} \tag{5.10}$$

$$\mathbb{V} = [\mathbb{v}_1^1, \ldots, \mathbb{v}_\pi^1, \ldots, \mathbb{v}_{id}^l, \ldots \mathbb{v}_1^F, \ldots, \mathbb{v}_\pi^F]_{3\times(F*\pi)} \tag{5.11}$$

$$\mathbb{X}_{Gbest} = [\mathbb{x}_{Gbest}^1, \ldots, \mathbb{x}_{Gbest}^l, \ldots, \mathbb{x}_{Gbest}^F]_{\boldsymbol{3\times F}} \tag{5.12}$$

$$\mathbb{X}_{pbest} = [\mathbb{x}_{pbest_1}^1, \ldots, \mathbb{x}_{pbest_{id}}^l, \ldots, \mathbb{x}_{pbest_\pi}^F]_{\boldsymbol{3\times(F*\pi)}} \tag{5.13}$$

where $\boldsymbol{fit}\left(\mathbb{x}_{id}^{l}\right)$ is the fitness value of the id^{th} particle in the l^{th} PSO population. $\mathbb{X}$ and $\mathbb{V}$ are, respectively, the position and velocity vectors of all particles in all optimization problems. Global best and personal best positions are arranged in similar vectors. The subtraction operation in the term $(\mathbb{X}_{Gbest} - \mathbb{X})$ in (17), is a broadcasted subtraction to match the size of the $\mathbb{X}_{Gbest}$ vector with that of $\mathbb{X}$.

The pseudo-code of PMOPSO is provided in Algorithm 1. In each iteration of the proposed PMOPSO algorithm, F optimization problems are solved concurrently. Each particle in the PMOPSO is implemented as an individual GPU thread. Particles are distinguished by a unique thread index accessible by that thread inside the CUDA kernels. This index directs

the particle to get access to its relevant data in the data arrays storing the position ($\mathbb{x} \in R^{3\times(F\times\pi)}$), velocity ($\mathbb{v} \in R^{3\times(F\times\pi)}$), and fitness ($PBest \in R^{1\times(F\times\pi)}$, $GBest \in R^{1\times(F\times 1)}$). Once particles are evaluated at the end of each iteration, threads are synchronized to update the global best particle.

Algorithm 1 PMOPSO Algorithm

$\boldsymbol{F}$:	Number of features to localize
$\boldsymbol{\pi}$:	Number of particles in each PSO problem
$\boldsymbol{\_Init()}$:	Initialization kernel
$\boldsymbol{\_Eval}(\mathbb{x}_{id}^{l})$:	Fitness function kernel
$\boldsymbol{\_Update}(\mathbb{x}_{id}^{l}, \mathbb{v}_{id}^{l})$:	Position and velocity update kernel
$\boldsymbol{\_SyncThreads()}$:	Synchronizes all threads

1 Allocate GPU memory for PSO variables (i.e. $\mathbb{X}$, $\mathbb{V}$, $\mathbb{X}_{\boldsymbol{Gbest}}$, $\mathbb{X}_{\boldsymbol{pbest}}$, $\boldsymbol{Gbest}$, and $\boldsymbol{pbest}$)
2 Transfer features' observations and the required system's states (i.e. $\boldsymbol{z}_{f_l'}^{k}, \boldsymbol{R}_{C_i}^{C_k}, \boldsymbol{P}_{C_k C_i}^{C_k}$, and $\boldsymbol{x}_{C_{ints}}$) to the GPU/Device memory
3 $\boldsymbol{\_Init()}$
4 **for** $\boldsymbol{i}$=1 to *iterations* **do**
5 $\quad \boldsymbol{id} \leftarrow \boldsymbol{blockIdx.x} * \boldsymbol{blockDim.x} + \boldsymbol{threadIdx.x}$
6 $\quad \boldsymbol{l} \leftarrow \boldsymbol{id}\%\boldsymbol{F}$
7 $\quad \boldsymbol{fit}_{id}^{l} \leftarrow \boldsymbol{\_Eval}(\mathbb{x}_{id}^{l})$
8 $\quad$ **if** ($\boldsymbol{fit}_{id}^{l} < \boldsymbol{PBest}_{id}^{l}$) **then**
9 $\quad\quad \boldsymbol{PBest}_{id}^{l} \leftarrow \boldsymbol{fit}_{id}^{l}$
10 $\quad\quad \mathbb{x}_{id,\boldsymbol{Pbest}}^{l} \leftarrow \mathbb{x}_{id}^{l}$
11 $\quad \boldsymbol{\_SyncThreads()}$
12 $\quad$ **if** ($\boldsymbol{fit}_{id}^{l} < \boldsymbol{GBest}^{l}$) **then**
13 $\quad\quad \boldsymbol{GBest}^{l} \leftarrow \boldsymbol{fit}_{id}^{l}$
14 $\quad\quad \mathbb{x}_{\boldsymbol{GBest}}^{l} \leftarrow \mathbb{x}_{id}^{l}$
15 $\quad \boldsymbol{\_Update}(\mathbb{x}_{id}^{l}, \mathbb{v}_{id}^{l})$
16 Transfer $\mathbb{x}_{\boldsymbol{GBest}}^{l}$ and $\boldsymbol{r}_{f_l}$ to the CPU memory
17 Deallocate GPU memory

To fully utilize the memory bandwidth to access the GPU memory, it is necessary to store and access the data on the GPU memory in a Coalesced pattern. Coalescing suggests that when a warp of threads needs to have access to a set of data, cache access latency is minimized when that set of data is stored in adjacent locations in the device memory. Therefore, when a block

of memory is brought to the GPU cache, relevant data that adjacent threads potentially need are also brought to the cache. In the proposed implementations of the PMOPSO, data arrays were stored in a coalesced pattern, as shown in Figure 5. 9.

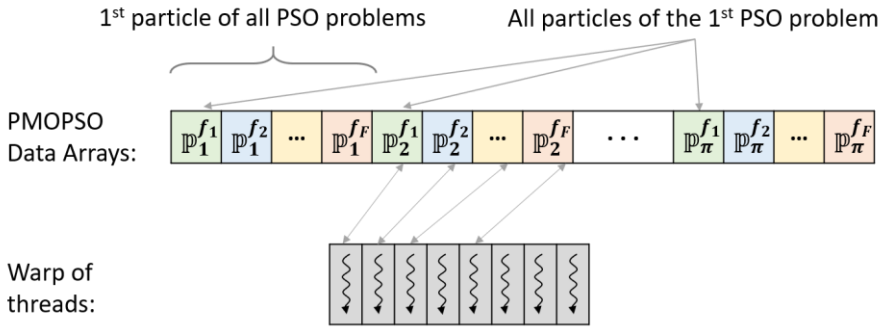


Figure 5. 9. Storing and indexing patterns of PMOPSO data arrays in a coalesced pattern to utilize maximum memory bandwidth.

To verify the accuracy of the PSO algorithm in feature localization problem compared to the GN method, a set of 128 features were directly selected from the KITTI "Sequence-07". Figure 5. 10 shows how the fitness of these 128 features in the PSO algorithm converges over 15 iterations to the fitness value achieved at the 15th iteration of the GN method. As the figure shows, by the 13th iteration, the ratio of the GN method's fitness to the PSO algorithm's fitness passes 0.976 for all 128 features, and the average of these ratios by the end of the 15th iteration is 0.998. In this experiment, the PSO population size per feature was 512 particles. For a fair comparison between the two algorithms, the initial solution that was provided for the GN method was also used as one of the particles in the PSO algorithm' population. This initial solution was obtained by triangulation of the first and last observations [45].

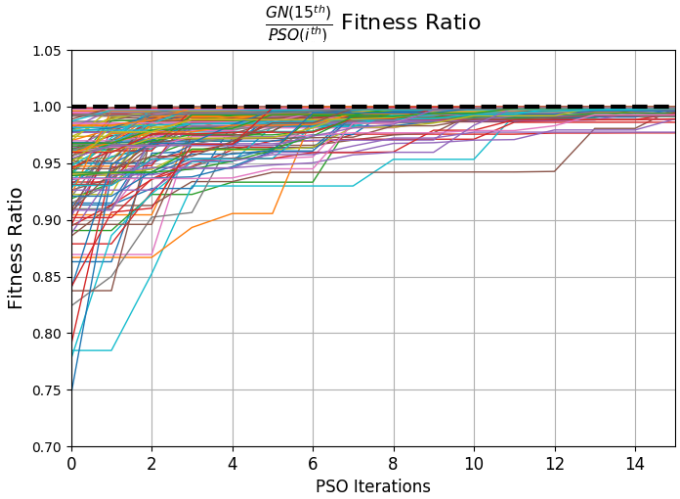


Figure 5. 10. The ratio of the GN method's fitness at the 15th iteration to the PSO algorithm's fitness over 15 iterations. In this experiment $F = 128$, and $\pi = 512$.

5.3.2 Parallel Feature Localization Using Parallel Multi-Objective GN

Since visual features are independent, the GN method can still benefit from parallelization at the features level. Hence, solving the 3D localization problem of F number of features in the Parallel Multi-Objective GN (PMOGN) framework entails the following modifications in the GN's equations:

$$\boldsymbol{\mathcal{F}}(\boldsymbol{\theta}) = [\boldsymbol{\mathcal{F}}_1(\theta_1), \dots, \boldsymbol{\mathcal{F}}_l(\theta_l), \dots, \boldsymbol{\mathcal{F}}_{\boldsymbol{F}}(\theta_{\boldsymbol{F}})]_{2\boldsymbol{F}\times 1} \tag{5.14}$$

$$\boldsymbol{\theta} = [\theta_1, \dots, \theta_l, \dots, \theta_{\boldsymbol{F}}]_{3\boldsymbol{F}\times 1} \tag{5.15}$$

$$\overline{\boldsymbol{\theta}}^{(s+1)} = \overline{\boldsymbol{\theta}}^{(s)} - (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \boldsymbol{r}^{(s)}, \tag{5.16}$$

$$\boldsymbol{J} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{\theta}} \tag{5.17}$$

$$\boldsymbol{J} = [J_{f_1}, \dots, J_{f_l}, \dots, J_{f_F}]_{2\sum M_l \times 3\boldsymbol{F}} \tag{5.18}$$

$$J_{f_l} = \frac{\partial \boldsymbol{r}_{f_l}}{\partial \theta_l} \tag{5.19}$$

$$\boldsymbol{r} = [\boldsymbol{r}_{f_1}, \dots, \boldsymbol{r}_{f_l}, \dots, \boldsymbol{r}_{f_F}]_{2\sum M_l \times 1} \tag{5.20}$$

$$\boldsymbol{r}_{f_l} = \left[r_{f_l}^1 \quad , \dots, \quad r_{f_l}^{M_l}\right]_{2M_l \times 1} \tag{5.21}$$

where the multi-objective function, $\boldsymbol{\mathcal{F}(\theta)}$, is now comprised of F objective functions as defined in (5.1). Since visual features are independent, the Jacobian matrix $\boldsymbol{J^T J}$ would have a block-diagonal shape. Therefore, its inverse would reduce to the inverse of its individual 3×3 blocks, which can be calculated efficiently in parallel.

The pseudo-code of PMOGN is provided in Algorithm 2. in each iteration of the proposed PMOGN algorithm, F optimization problems are solved concurrently. Each GN problem is implemented as an individual GPU thread, and they are identified by a unique index accessible by that thread inside the CUDA kernels. This index directs the GN solvers to get access to correct elements in the data arrays (i.e. parameters $\theta \in R^{3\times F}$, Residual $\boldsymbol{r} \in R^{2\sum M_l \times 1}$, and Jacobians $J \in R^{2\sum M_l 3\times}$).

Algorithm 2: PMOGN Algorithm

$\boldsymbol{F}$:	Number of features to localize (or number of GN problems)
$\boldsymbol{\_Init()}$:	Initialization kernel on the GPU
$\boldsymbol{\_Comp\_r\_J(\theta_l)}$:	Residual and Jacobian computation kernel on the GPU
$\boldsymbol{\_Inv\_JTJ(J)}$:	Jacobian inverse computation kernel on GPU

1 Allocate GPU memory for GN variables (i.e. $\boldsymbol{\theta_l}, \boldsymbol{J_{\mathcal{F}_l}}$, and $\boldsymbol{r_{f_l}}$ where $\boldsymbol{l} \in \{\boldsymbol{1}, \boldsymbol{F}\}$)
2 Transfer features' observations and the required system's states (i.e. $\boldsymbol{z^k_{f_l}}, \boldsymbol{R^{C_k}_{C_i}}, \boldsymbol{P^{C_k}_{C_k C_i}}$, and $\boldsymbol{x_{C_{ints}}}$) to the GPU/Device memory
3 $\boldsymbol{\_Init()}$
4 **for** $\boldsymbol{i}$=1 to *iterations* **do**
5 $\quad \boldsymbol{l \leftarrow blockIdx.x * blockDim.x + threadIdx.x}$
6 $\quad \boldsymbol{r_l, J_l \leftarrow \_Comp\_r\_J(\theta_l)}$
7 $\quad \boldsymbol{JTJ\_inv_l \leftarrow \_Inv\_JTJ(J_l)}$
8 $\quad \boldsymbol{\theta_l \leftarrow \theta_l - JTJ\_inv_l \times J_l^T \times r_l}$
9 Transfer $\boldsymbol{\theta}$ and $\boldsymbol{r}$ to the CPU/Host memory
10 Deallocate GPU memory

It is worth highlighting that given one optimization algorithm (either PSO or GN) and then reformulating the features localization task from a set of single-objective optimization problems into one multi-objective optimization problem does not impose heavy

computational complexity overhead. For instance, choosing PSO as the optimization algorithm, the independence of PSO sub-problems in the PMOPSO also offers separations among particles' populations; therefore, the computational complexity within each PSO sub-problem remains the same as that of the single-objective PSO problem because the search space per PSO sub-problem is still 3-dimensional. Similarly, if GN is chosen as the optimization algorithm, the independence of features offers a well-shaped block-diagonal Jacobian matrix in PMOGN, such that the inverse of a $3F \times 3F$ block-diagonal large Jacobian matrix is reduced to the inverse of its 3×3 blocks, which is the same size as the Jacobian matrix in a single-objective GN problem. From a different perspective, regardless of the objective function dimension (i.e. single- or multi-objective), the computational complexity of the PSO algorithm grows with respect to its population size until it exceeds the computational complexity of the GN algorithm. However, the results in Figure 5. 10 showed that an excessively large PSO population size is not necessary to achieve an accuracy level comparable to GN in this experiment.

Therefore, the overhead of performing the feature localization task on the GPU, instead of CPU, regardless of the optimization algorithm, is mainly determined by how fast the data transfer between the host and device memory can happen. Hence, the processing time speed-up of GPU-based implementations relative to the CPU-based implementation, and the comparison between PMOPSO and PMOGN implementations are best assessed not as an independent process rather within a complete operation of the proposed P-MSCKF pipeline with GPU-accelerated visual backend on the target embedded system. The next section discusses the processing time improvement of the proposed CPU-GPU-based MSCKF pipeline. Since the *gperftool* is not GPU compatible, the computation complexity

analysis is done by directly measuring the per-epoch processing time using the *Chrono* timer [74].

5.4 P-MSCKF VIN System Accelerated with Parallel Feature Localization

In this section, the performance of the GCC optimized P-MSCKF pipeline accelerated by the GPU implementation of the PMOPSO- and PMOGN-based feature localization is discussed. In the rest of the chapter, the terms P-MSCKF-PMOPSO and P-MSCKF-PMOGN are used to distinguish these implementations of the MSCKF VIN system from previous implementations. Figure 5. 11 shows the proposed CPU-GPU pipeline for the P-MSCKF-PMOPSO and P-MSCKF-PMOGN VIN systems.

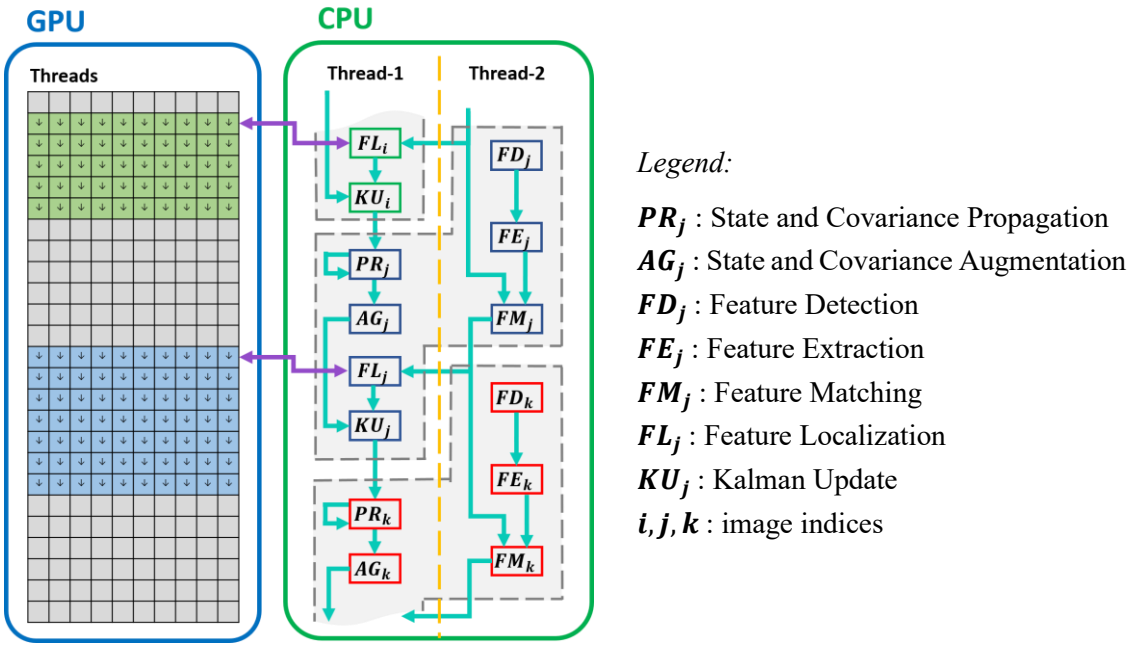
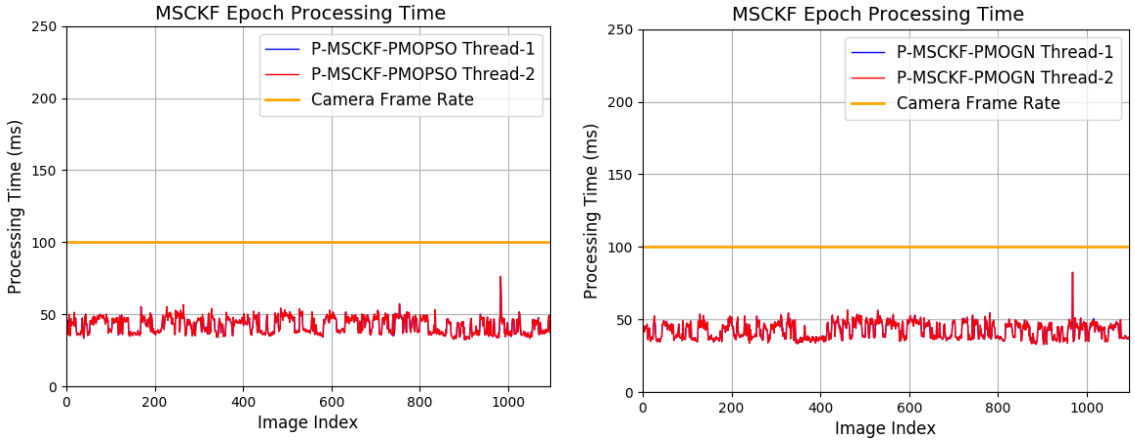


Figure 5. 11. P-MSCKF-PMOPSO and P-MSCKF-PMOGN pipeline and dataflow.

Figure 5. 12 shows the epoch processing time of the GCC optimized P-MSCKF-PMOPSO and GCC optimized P-MSCKF-PMOGN VIN systems tested on KITTI "Sequence-07". As shown in the figure, not only are all epochs have been processed strictly faster than the

camera frame rate, but also both implementations save approximately 50ms margin below the camera frame rate. Table 5. 3 summarizes the accuracy and processing time statistics of all MSCKF implementations in this chapter.

As Table 5. 3 shows, the GCC optimized P-MSCKF pipeline with GPU-accelerated feature localization achieves 33% faster epoch processing time with no real-time constraint violations compared to the GCC optimized P-MSCKF. It should be highlighted that the GCC optimization tool does not apply to the GPU code, and its contribution was confined to the CPU code only. The GPU-accelerated P-MSCKF VIN pipelines also offer more reliability by having a smaller variance in the per-epoch processing time.



(a) (b)

Figure 5. 12. Epoch processing time comparison of the (a) P-MSCKF-PMOPSO, and (b) P-MSCKF-PMOGN VIN systems on KITTI "Sequence-07". In PMOPSO, $\pi = 512$.

Table 5. 3. Accuracy and processing time comparison among all proposed implementations of MSCKF VIN systems tested on KITTI "Sequence-07".

	S-MSCKF	P-MSCKF	P-MSCKF GCC Optimized	P-MSCKF-PMOPSO GCC Optimized	P-MSCKF-PMOGN GCC Optimized
Estimation RMSE Position NED ($\boldsymbol{m}$)	$\begin{bmatrix}2.44\\2.91\\0.63\end{bmatrix}$	$\begin{bmatrix}2.23\\3.10\\0.86\end{bmatrix}$	$\begin{bmatrix}2.23\\3.08\\0.84\end{bmatrix}$	$\begin{bmatrix}1.84\\2.74\\0.72\end{bmatrix}$	$\begin{bmatrix}2.49\\2.97\\0.46\end{bmatrix}$
Attitude $\boldsymbol{r}, \boldsymbol{p}, \boldsymbol{A}$ ($\mathbf{deg}$)	$\begin{bmatrix}0.07\\0.11\\0.52\end{bmatrix}$	$\begin{bmatrix}0.10\\0.12\\0.72\end{bmatrix}$	$\begin{bmatrix}0.10\\0.13\\0.72\end{bmatrix}$	$\begin{bmatrix}0.10\\0.10\\0.72\end{bmatrix}$	$\begin{bmatrix}0.10\\0.10\\0.71\end{bmatrix}$
Processing Time					
Total ($\mathbf{s}$)	105.9	67.1	69.5	47.39	46.80
Epoch Average ($\mathbf{ms}$)	93.75	55.07	63.15	42.43	42.35
Epoch Standard Deviation ($\mathbf{ms}$)	27.00	16.04	17.83	5.52	5.4
Real-time Constraint Violations (%)	30.27	3.45	1.00	0.00	0.00

It is worth discussing that the optimization solutions for hardware implementation proposed in this chapter are applicable to any thread-enabled CPU and GPU computing platforms. However, re-implementation of the same pipelines on different computing hardware may lead to different processing time speed-up factors compared to the values reported in this work. Contributing factors include the CPU-GPU processors working frequency, memory size and bandwidth, number of core in the GPU, and operating system scheduling.

5.5 Hardware-in-the-loop Simulation

Hardware-in-the-loop (HIL) is a testing technique to verify and validate the performance of an embedded system working in simulated conditions that trick the embedded system into thinking that it is working with a real assembled system. HIL is very popular in the automotive industry because it provides the opportunity to perform cost-effective, comprehensive and repeatable tests [117]. For a HIL simulation to be of benefit, the simulated environment should adequately model a real system, including data transmission delays, sensors' and actuators' errors, and noise.

In previous sections, to utilize the full capacity of the proposed parallel pipeline, images and IMU samples were fed to the pipeline continuously as soon as the threads become free to process the next image or IMU sample. However, in a real scenario, the pipeline can only be fed with data as fast as the framerate of the camera and the sampling rate of the IMU. The proposed P-MSCKF-PMOGN system was tested in a HIL simulation scenario where the images and IMU samples measured between two images were fed to the pipeline in intervals of 100ms. In this experiment, to account for the delay in reading images from an actual camera, images were read from the SD Card storage onboard the NVIDIA® Jetson TX2 board. The position and attitude estimates were also transmitted in real-time via a serial interface to a laptop for visualization to further account for a potential delay that the state estimation causes in the higher-level pipeline of autonomous systems. Figure 5. 13 and Figure 5. 14 show the HIL simulation setup in this experiment.

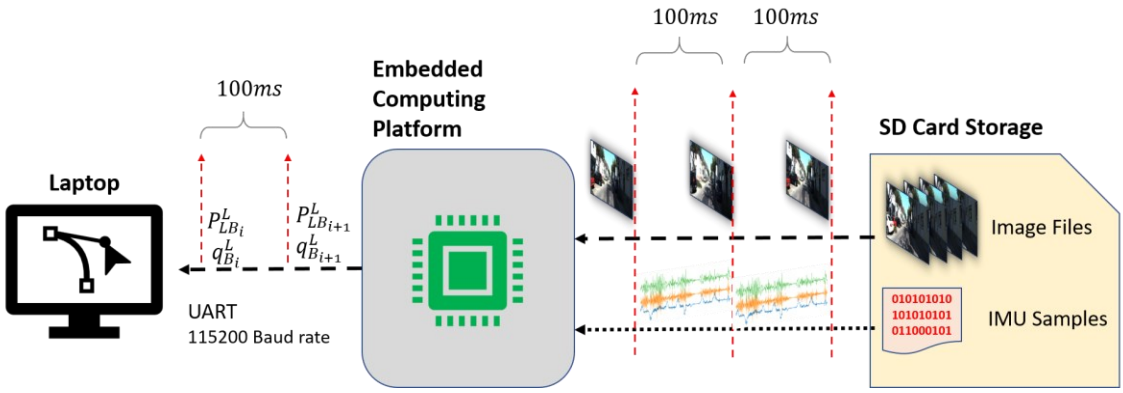


Figure 5. 13. Hardware-in-the-loop simulation setup block diagram.

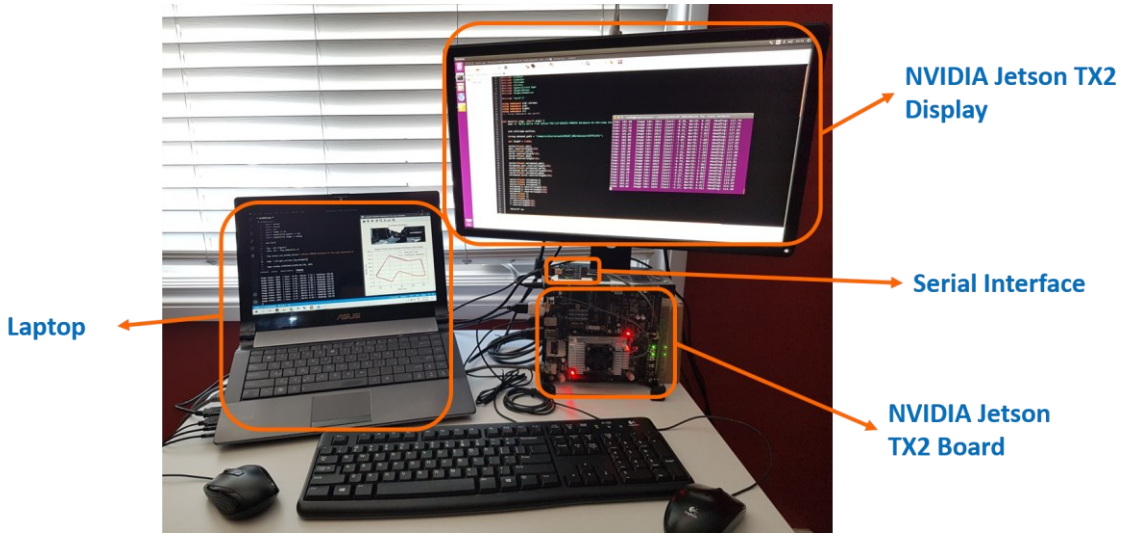


Figure 5. 14. Hardware-in-the-Loop simulation setup in practice.

In the HIL simulation with P-MSCKF-PMOGN on KITTI "Sequence-07", all design parameters were adjusted the same as before, and therefore, the estimation accuracy remained the same. The epoch processing time of the P-MSCKF-PMOGN in the HIL simulation is shown in Figure 5. 15. This timing includes the time that threads wait for their data to become available. Compared to Figure 5. 12, the average processing time shows an increase which is related to the HIL simulation conditions and to the operating system's scheduling.

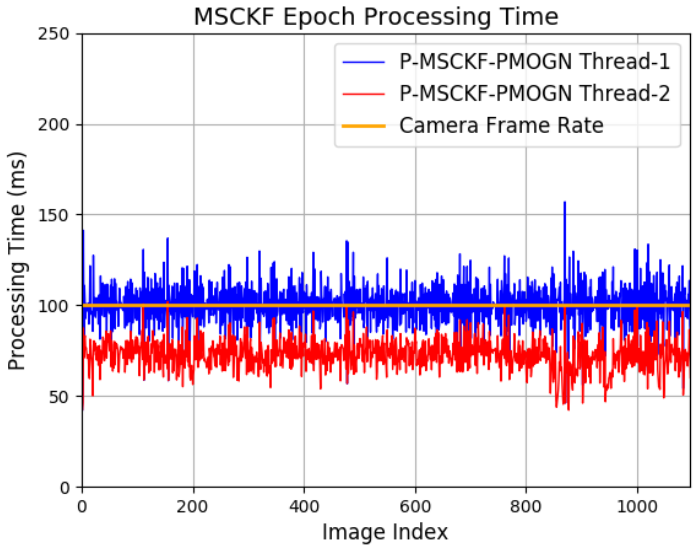


Figure 5. 15. Epoch processing time of P-MSCKF-PMOGN in HIL simulation setup.

5.6 Summary

In this chapter, first, the conventional sequential pipeline of the MSCKF VIN system was analyzed with an accurate performance profiling tool to identify its bottlenecks. The sequential pipeline, S-MSCKF, was then redesigned with two parallel threads that led to 41% faster per-epoch processing time. The parallel threads were carefully synchronized to guarantee the data dependencies of the original pipeline. The proposed parallel pipeline, P-MSCKF, was then further optimized with the GCC optimization tool. The computationally challenging scenarios would happen when a large set of features imposed a heavy processing burden on the fusion engine modules. This challenge was overcome to a great extent by adopting the GCC optimization techniques; however, strict real-time constraint violation was still 1%, and the per-epoch processing time average was severely close to the camera frame rate. Next, the features localization problem was reformulated as a single multi-objective optimization problem. Utilizing the GPU's parallel architecture, efficient parallel implementations based on PSO and GN algorithms were proposed that could

perform the localization task concurrently for all features. Adopting the GPU-accelerated feature localization module in the GCC optimized P-MSCKF pipeline led to the satisfaction of strict real-time constraints with an approximately 50ms safe margin faster than the camera frame rate. Finally, a HIL simulation was performed to evaluate the developed P-MSCKF-PMOGN under camera latencies scenario.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This thesis presented the novel design, automatic tuning, and real-time implementation of a TC VIN system integrated with GNSS. In each development stage, a thorough background review, detailed problem statement, in-depth explanation of the proposed solutions, and experimental results on real datasets were discussed.

In Chapter 3, first, the IMU-GNSS TC navigation system was considered, and a flexible simulation environment for its design and evaluation was proposed. Experimental results on real datasets showed that in challenging conditions where satellite coverage was poor, partial satellite observability could considerably slow down the divergence rate of the estimation and provide a quality navigation solution for a more extended period of GNSS outage. The chapter then emphasized the critical role of employing an accurate and well-tuned model of the system that best represents the actual behaviour of the system under real conditions. Then, the chapter proposed a systematic design and automatic tuning technique based on the GA that heuristically searches for an optimal set of system parameters.

Chapter 3 then considered the TC VIN problem. The visual-inertial fusion was performed in the MSCKF framework with the state vector augmented with more detailed error models for the IMU and the camera. The developed enhanced MSCKF VIN was shown to achieve higher navigation accuracy than the original MSCKF VIN formulation with the reduced state vector. The problem of optimal tuning of the parameters of the developed MSCKF VIN was revisited. This problem was more challenging than the IMU-GNSS TC navigation

system because of the dead reckoning nature of the VIN system and the larger set of tunable parameters. Experimental results on a real dataset showed that the proposed tuning technique is highly effective in finding a set of parameters that lead the developed MSCKF VIN system to operate optimally in terms of both accuracy and covariance estimations.

In continuation of the automatic tuning work, in Chapter 4, the VIN problem was approached from an end-to-end learning problem. The proposed VIN neural network architecture employed CNNs and RNNs to perform the required visual and temporal processing. In the training process, no assisting information such as system/measurement models or calibration parameters were provided for the network. Experimental results on the real dataset showed that the proposed VIN neural network was a powerful learning system that learned the sophisticated visual-inertial fusion for navigation purposes in a model-free and calibration-free fashion. Compared to filtering-based techniques (e.g. MSCKF VIN system), the deep learning-based approach was able to better generalize its learned knowledge in fusion and estimation within the same dataset; however, this generalizability was achieved at the cost of less accurate pose estimation on individual sequences.

Chapter 5 considered the real-time embedded hardware implementation of the developed enhanced MSCKF VIN system. First, the bottlenecks of the conventional sequential pipeline of the MSCKF VIN system were identified by analyzing its accurate processing time profile. To improve the processing time and satisfy strict real-time constraints, two solutions based on parallelization of the pipeline on the CPU and GPU architectures were proposed. In the first solution, the conventional sequential pipeline of the MSCKF VIN was efficiently parallelized on two threads of the CPU that led to a 41% faster per-epoch

processing time. In the second solution, one of the most time-consuming modules of the pipeline, the Feature localization module, was reformulated into a multi-objective optimization problem that was efficiently parallelized on the GPU architecture. The proposed GPU-accelerated Feature Localization module integrated into the multi-thread MSCKF pipeline aided by compiler optimization made it possible to satisfy strict real-time constraints with approximately 50ms margin below the camera frame rate. Experimental results on a real dataset in a HIL simulation setup were also provided to show the efficacy of the proposed parallel implementation.

6.2 Potential Directions for future research

The works presented in this thesis have opened new opportunities for further search in the following potential directions.

6.2.1 Embedded deep learning-based Visual-Inertial-GNSS navigation

The proposed neural network architecture for visual-inertial-GNSS fusion has more than 150 million trainable parameters. Training such a large neural network on the KITTI odometry dataset took approximately 40 hours on the NVIDIA® Tesla P100 GPU. To enhance the generalizability of the proposed neural network, more datasets with varied characteristics (images captured by different cameras and inertial data measured by different IMUs) must be included in the training set. A larger training set inevitably prolongs the training phase; hence, one promising direction for further research would be to optimize the neural network's size. However, it should be recognized that shrinking the neural network has a trade-off with its accuracy and generalizability. Additional motivation

to shrink the developed VIGN neural network's size would be to target real-time embedded implementation. In the test phase, the proposed neural network VIGN system was running at 60 fps on NVIDIA® Tesla P100. Because of the compatibility difficulties in testing the developed VIN neural network on NVIDIA® Jetson TX-2, it was not possible at the time of pursuing this research to test it on an embedded computing platform. However, one particularly effective measure to target an embedded implementation is to perform all floating-point computations in half-precision or fixed-precision. Although half- or fixed-precision computations might affect the accuracy, less memory requirement and faster computation are advantageous for real-time embedded performance [118]. An additional incentive to shrink the network's size is to make online fine-training possible on the whole or part of the proposed neural network [119]. This capability is critical in deep learning-based approaches to target pose estimation accuracy achievable by rival filtering-based techniques. To improve the current VIO neural network to outperform the model-based VIN systems such as MSCKF VIN, initially, the VIO neural network must be trained on a larger and more diverse dataset (e.g. different camera and IMU sensors setup) to reinforce the network to further learn the characteristics of different sensors as variables in the input data. More importantly, the VIO neural network should be equipped with an effective fine-training or initialization technique to quickly adjust its accuracy online. In this work, the initialization of neural networks was performed by providing the true pose estimates for the first ten images of the experiment. One possible alternative solution could be training a separate small-sized neural network separately on different visual-inertial data with sensors calibration/error information as the training target. This sub-network can be used as a pre-trained network within the VIO neural network, and its output would carry

initialization information. Consequently, the contributing factor of an accurate motion estimation task that depends on proper initialization on the slight variations in each test sequence is confined to a small sub-network rather than the whole VIO neural network. This sub-network would be significantly faster and easier to fine-train or initialize online. One might also study the effect of neural network size (e.g. number of hidden layers, number of neurons of each layer), objective function, activation functions, and alternative RNN architectures such as Gated Recurrent Unit (GRU) [120] on the accuracy as well as the speed of the whole VIO neural network.

6.2.2 Online fine-tuning of the Kalman filter parameters

Tuning the Kalman filter parameters in IMU-GNSS and Visual-Inertial navigation systems were performed offline, given a reference navigation solution. To provide generalizability, the tuning process was performed on three separate portions: training, validation, and test. However, adding online fine-tuning capabilities broadens the applications of the proposed automatic tuning technique and makes it applicable in harsher and less predictable environmental conditions. Further research on this direction could benefit from the fact that, in reasonably different environmental conditions, the new optimal set of parameters possibly lie in a close adjacency of the current optimal parameters. The online fine-tuning problem can then be reformulated to an efficient and fast search in the local space. One considerable contribution to speed up the fine-tuning process would be prioritizing the tunable parameters according to their impact on the performance and their sensitivity to environmental conditions. Initial research in this direction has shown considerable improvement, but further research is required to adopt it for online fine-tuning efficiently

[102]. Furthermore, the fine-tuning technique would be more applicable if it relies less on a reference navigation solution as the performance metric. A potential direction toward addressing this limitation would be employing the Kalman filter innovation sequence as an internal source of performance metrics [121]. One valuable analysis in the topic of Kalman filter covariance matrix determination is to evaluate and compare the performance of the traditional techniques on high-dimensional state estimation problems such as MSCKF VIN. The traditional tuning techniques, including Bayesian [122] and maximum likelihood [123], are not widely used because of their excessive computational time [124]. The covariance matching technique [125] has been shown to be prone to biased estimation of the optimal covariances [124]. The correlation technique [126] using the autocovariance of the innovation sequence was shown to be unbiased and have asymptotical convergence behaviour given a large sample size on a low-dimensional problem. Moreover, one area worth further investigation is the potentials of improving the complexity and performance of these traditional techniques aided by AI-based algorithms. Recent works combining GA and stochastic gradient-based technique [127] and PSO-aided maximum likelihood estimation [128] promise further advancement of hybrid techniques.

6.2.3 Resource usage optimization of the embedded MSCKF VIN

The MSCKF VIN system has different processing modules. Performance optimization in this work targeted only the most computationally burdensome modules; however, there are ample opportunities to further optimize the pipeline. One important area for parallelization and further improvement is the feature detection and extraction modules. In this work, only CPU classes of the OpenCV library were used for feature management that processes the

image on a row or column basis to detect and extract features; hence, the framerate is limited by how fast the feature detection module can cover the whole area of an image; however, recent researches have shown that a higher frame rate can be achieved with a customized parallel implementation of the feature detectors on GPU [107][129]. Efficient parallel implementations of feature detectors/extractors often divide an image into several tiles and distribute their processing tasks on the many-core architecture of the GPU. The importance of this optimization is significant in a stereo setup where pairs of images must be processed at each epoch.

List of References

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[2] "iRobot Roomba Robotics Vacuum Cleaner." https://www.irobot.ca/ (accessed Jan. 11, 2021).

[3] "Autonom Shuttle Evo." https://navya.tech/en/solutions/moving-people/self-driving-shuttle-for-passenger-transportation/ (accessed Jan. 11, 2021).

[4] "Applanix Photogrammetry Technology." https://www.applanix.com/news/as-vision-mapping-applanix-dg/ (accessed Jan. 11, 2021).

[5] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe, "Mobile robot positioning: Sensors and techniques," *Journal of Robotic Systems*, vol. 14, no. 4, pp. 231–249, Apr. 1997, doi: 10.1002/(SICI)1097-4563.

[6] J. Farrell, *Aided navigation: GPS with high rate sensors*. McGraw-Hill, Inc., 2008.

[7] P. Petkov and T. Slavov, "Stochastic modeling of MEMS inertial sensors," *Cybernetics and information technologies*, vol. 10, no. 2, pp. 31–40, 2010.

[8] W. H. Baird, "An introduction to inertial navigation," *American Journal of Physics*, vol. 77, no. 9, pp. 844–847, 2009, doi: 10.1119/1.3081061.

[9] D. Scaramuzza and F. Fraundorfer, "Visual odometry," *IEEE Robotics and Automation Magazine*, vol. 18, no. 4, pp. 80–92, Dec. 2011.

[10] J. W. Marck, A. Mohamoud, E. Vd Houwen, and R. Van Heijster, "Indoor radar SLAM A radar application for vision and GPS denied environments," in *Proceedings of European Radar Conference, Nuremberg, Germany*, 2013, pp. 471–474.

[11] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden*, 2016, pp. 1271–1278, doi: 10.1109/ICRA.2016.7487258.

[12] M. Barnard, "Tesla & Google Disagree About LIDAR — Which Is Right?"

https://cleantechnica.com/2016/07/29/tesla-google-disagree-lidar-right/ (accessed Apr. 09, 2021).

[13] P. Corke, J. Lobo, and J. Dias, "An introduction to inertial and visual sensing," *International Journal of Robotics Research*, vol. 26, no. 6, pp. 519–535, Jun. 2007, doi: 10.1177/0278364907079279.

[14] W. Elmenreich, "A Review on System Architectures for Sensor Fusion Applications," in *proceedings Software Technologies for Embedded and Ubiquitous Systems, Berlin, Heidelberg*, 2007, pp. 547–559.

[15] M. George and S. Sukkarieh, "Tightly coupled INS/GPS with bias estimation for UAV applications," in *Proceedings of the 2005 Australasian Conference on Robotics and Automation (ACRA), Sydney, Australia*, 2005, pp. 14–20.

[16] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of IMU and vision for absolute scale estimation in monocular SLAM," *Journal of Intelligent and Robotic Systems*, vol. 61, no. 1, pp. 287–299, 2011, doi: 10.1007/s10846-010-9490-z.

[17] A. L. Bustamante, J. M. Molina, and M. A. Patricio, "Multi-camera and Multi-modal Sensor Fusion, an Architecture Overview," in *Distributed Computing and Artificial Intelligence*, A. P. de Leon F. de Carvalho, S. Rodríguez-González, J. F. De Paz Santana, and J. M. C. Rodríguez, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 301–308.

[18] S. Saeedi, N. El-Sheimy, X. Zhao, and Z. Sayed, "Context Aware Mobile Personal Navigation Services Using Multi-Level Sensor Fusion," in *Proceedings of the 24th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2011), Portland, Or, USA*, Sep. 2011, pp. 1394–1403, Accessed: Jun. 11, 2021. [Online]. Available: http://www.ion.org/publications/abstract.cfm?jp=p&articleID=9697.

[19] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Rome, Italy*, 2007, pp. 3565–3572, doi: 10.1109/ROBOT.2007.364024.

[20] G. Bleser and D. Stricker, "Using the marginalised particle filter for real-time visual-

inertial sensor fusion," in *Proceedings of 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, Cambridge, UK*, 2008, pp. 3–12, doi: 10.1109/ISMAR.2008.4637316.

[21] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015, doi: 10.1177/0278364914554813.

[22] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018, doi: 10.1109/TRO.2018.2853729.

[23] J. J. Moré, "The Levenberg-Marquardt algorithm: Implementation and theory," in *Numerical Analysis*, Springer, 1978, pp. 105–116.

[24] R. Clark, S. Wang, H. Wen, A. Markham, N. T.- AAAI, and U. 2017, "VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem.," in *Proceedings of AAAI Conference on Artificial Intelligence, San Francisco,CA, USA*, 2017, vol. 31, no. 1, pp. 3995–4001.

[25] R. Li, S. Wang, Z. Long, and D. Gu, "UnDeepVO: Monocular Visual Odometry Through Unsupervised Deep Learning," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia*, 2018, pp. 7286–7291, doi: 10.1109/ICRA.2018.8461251.

[26] P. Abbeel, A. Coates, M. Montemerlo, A. Y. Ng, and S. Thrun, "Discriminative training of kalman filters," in *Robotics: Science and Systems*, 2005, vol. 1, pp. 289–296, doi: 10.15607/rss.2005.i.038.

[27] C. Grubin, "Derivation of the quaternion scheme via the Euler axis and angle," *Journal of Spacecraft and Rockets*, vol. 7, no. 10, pp. 1261–1263, Oct. 1970, doi: 10.2514/3.30149.

[28] A. L. Schwab, "Quaternions, finite rotation and euler parameters," *Cornell University Notes, Ithaca NY*, p. 28, 2002.

[29] J. Sola, "Quaternion kinematics for the error-state Kalman Filter," *Laboratoire*

dAnalyse et dArchitecture des Systemes-Centre national de la recherche scientifique (LAAS-CNRS), 2012.

[30] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," 1995, pp. 127–132.

[31] M. El-Diasty and S. Pagiatakis, "Calibration and Stochastic Modelling of Inertial Navigation Sensor Errors," *Journal of Global Positioning Systems*, vol. 7, no. 2, pp. 170–182, Dec. 2008, doi: 10.5081/jgps.7.2.170.

[32] A. Angrisano, "GNSS/INS Integration Methods," Ph.D. dissertation, Dept. App Sc, Parthenope Univ, Naples, Italy, 2010.

[33] M. Lashley, D. M. Bevly, and J. Y. Hung, "Analysis of deeply integrated and tightly coupled architectures," in *Proceedings of IEEE/ION Position, Location and Navigation Symposium, Brisbane, QLD, Australia*, May 2010, pp. 382–396, doi: 10.1109/PLANS.2010.5507127.

[34] G. Falco, M. Pini, and G. Marucco, "Loose and Tight GNSS/INS Integrations: Comparison of Performance Assessed in Real Urban Scenarios.," *Sensors*, vol. 17, no. 2, p. 255, 2017, doi: 10.3390/s17020255.

[35] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004, doi: 10.1109/TPAMI.2004.17.

[36] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, Sep. 1981, doi: 10.1038/293133a0.

[37] J. Delmerico and D. Scaramuzza, "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia*, 2018, pp. 2502–2509, doi: 10.1109/ICRA.2018.8460664.

[38] J. W. Langelaan, "State estimation for autonomous flight in cluttered Environments," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1414–1426, Sep. 2007, doi: 10.2514/1.27770.

[39] D. Strelow and S. Singh, “Motion estimation from image and inertial measurements,” 2004.

[40] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[41] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-Up Robust Features (SURF),” *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, Jun. 2008, doi: 10.1016/j.cviu.2007.09.014.

[42] E. Rosten and T. Drummond, “Machine Learning for High-Speed Corner Detection,” in *Proceedings of European conference on computer vision, Berlin, Heidelberg*, 2006, pp. 430–443, doi: 10.1007/11744023_34.

[43] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” in *Proceedings of the IEEE International Conference on Computer Vision, Barcelona, Spain*, 2011, pp. 2564–2571, doi: 10.1109/ICCV.2011.6126544.

[44] T. Tuytelaars and K. Mikolajczyk, *Local invariant feature detectors: A survey*, vol. 3, no. 3. Now Publishers Inc, 2007.

[45] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

[46] J. Civera, A. J. Davison, J. M. Martínez Montiel, J. Civera, J. M. Martínez, and) A J Davison, “Inverse Depth Parametrization for Monocular SLAM,” *IEEE TRANSACTIONS ON ROBOTICS*, vol. 24, no. 5, pp. 932–945, Oct. 2008, doi: 10.1109/TRO.2008.2003276.

[47] Å. Björck, *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics (SIAM), 1996.

[48] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[49] P. Vanicek and E. J. Krakiwsky, *Geodesy: the concepts*, 2nd ed. Elsevier, 2015.

[50] B. D. Brumback and M. D. Srinath, “A Chi-square test for fault-detection in Kalman

filters," *IEEE Transactions on Automatic Control*, vol. 32, no. 6, pp. 552–554, Jun. 1987, doi: 10.1109/tac.1987.1104658.

[51] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, Jun. 2014, doi: 10.1561/2000000039.

[52] I. Goodfellow, Y. Bengio, and C. Aaron, *Deep learning*. Cambridge, MA, USA: The MIT Press, 2016.

[53] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object Recognition with Gradient-Based Learning," in *Shape, contour and grouping in computer vision*, Springer, Berlin, Heidelberg, 1999, pp. 319–345.

[54] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks," 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed Apr. 11, 2021).

[55] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[56] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998, doi: 10.1142/S0218488598000094.

[57] S. Binitha and S. S. Sathya, "A Survey of Bio inspired Optimization Algorithms," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. 2, pp. 137–151, 2012.

[58] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 13th ed. Addison Wesley Publishing Co. Inc., 1989.

[59] J. Kennedy, R. Eberhart, and B. Gov, "Particle Swarm Optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia*, 1995, pp. 1942-1948 Vol.4.

[60] Z. Yang, F. Gao, and S. Shen, "Real-time monocular dense mapping on aerial robots

using visual-inertial fusion," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Marina Bay Sands, Singapore*, Jul. 2017, pp. 4552–4559, doi: 10.1109/ICRA.2017.7989529.

[61] "CUDA C++ Programming Guide Design Guide v11.2.0." 2020, Accessed: Jan. 03, 2021. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[62] "NVIDIA Jetson-TX2." https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.

[63] P. D. Groves, *Principles of GNSS, inertial, and multisensor integrated navigation systems*. Artech House, 2013.

[64] T. D. Powell, "Automated tuning of an extended Kalman filter using the downhill simplex algorithm," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 5, pp. 901–908, Sep. 2002, doi: 10.2514/2.4983.

[65] O. V Korniyenko, M. S. Sharawi, and D. N. Aloi, "Neural network based approach for tuning kalman filter," in *Proceedings of IEEE International Conference on Electro Information Technology, Lincoln, NE, USA*, 2005, pp. 1–5.

[66] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice-Hall, Inc., 2007.

[67] C. W. Kang and C. G. Park, "Attitude estimation with accelerometers and gyros using fuzzy tuned Kalman filter," in *Proceedings of 2009 European Control Conference, ECC. Budapest, Hungary*, 2009, pp. 3713–3718, doi: 10.23919/ecc.2009.7074977.

[68] L. A. Zadeh, "Fuzzy Logic," *Computer,* vol. 21, no. 4, pp. 83–93, 1988, doi: 10.1109/2.53.

[69] D. Loebis, R. Sutton, J. Chudley, and W. Naeem, "Adaptive tuning of a Kalman filter via fuzzy logic for an intelligent AUV navigation system," *Control Engineering Practice*, vol. 12, no. 12, pp. 1531–1539, Dec. 2004, doi: 10.1016/j.conengprac.2003.11.008.

[70] A. Chatterjee and F. Matsuno, "A neuro-fuzzy assisted extended kalman filter-based approach for simultaneous localization and mapping (SLAM) problems," *IEEE*

Transactions on Fuzzy Systems, vol. 15, no. 5, pp. 984–997, Oct. 2007, doi: 10.1109/TFUZZ.2007.894972.

[71] R. K. Jatoth and T. K. Kumar, "Particle Swarm optimization Based Tuning of Extended Kalman Filter for Manoeuvring Target Tracking," *International Journal of Circuits, Systems and Signal Processing*, vol. 3, no. 3, pp. 127–136, 2009.

[72] P. D. Hanlon and P. S. Maybeck, "Multiple-model adaptive estimation using a residual correlation Kalman filter bank," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 2, pp. 393–406, Apr. 2000, doi: 10.1109/7.845216.

[73] "Symbolic Math Toolbox - MATLAB." https://www.mathworks.com/products/symbolic.html (accessed Mar. 17, 2019).

[74] B. Stroustrup, *The C++ programming language*. Pearson Education India, 2000.

[75] "MPU-9250 Product Specification Revision 1.1 MPU-9250 Product Specification." http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf (accessed Apr. 07, 2019).

[76] "u-blox 7 GNSS evaluation kit." https://www.u-blox.com/en/product/evk-7 (accessed Apr. 11, 2021).

[77] "Realsense Depth Camera D435." https://www.intelrealsense.com/depth-camera-d435/ (accessed Apr. 11, 2021).

[78] "XETHRU X4M03 Radar." https://www.xethru.com/xethru-development-platform.html (accessed May 20, 2019).

[79] "NovAtel ProPak6 Triple-Frequency GNSS receiver." https://novatel.com/support/previous-generation-products-drop-down/previous-generation-products/propak6-receiver (accessed Apr. 11, 2021).

[80] "KVH-1759 IMU." https://www.kvh.com/Military-and-Government/Gyros-and-Inertial-Systems-and-Compasses/Gyros-and-IMUs-and-INS/IMUs/1750-IMU.aspx (accessed Apr. 11, 2021).

[81] "InvenSense MPU-9250 IMU." https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/ (accessed Apr. 11, 2021).

[82] R. Hassan, B. Cohanim, O. De Weck, and G. Venter, "A comparison of particle

swarm optimization and the genetic algorithm," in *Proceedings of 46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference, Austin, Texas, USA*, 2005, vol. 2, p. 1897, doi: 10.2514/6.2005-1897.

[83] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Providence, RI, USA*, 2012, pp. 3354–3361, doi: 10.1109/CVPR.2012.6248074.

[84] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013, doi: 10.1177/0278364913491297.

[85] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.

[86] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to MAV navigation," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems, Tokyo, Japan*, 2013, pp. 3923–3929, doi: 10.1109/IROS.2013.6696917.

[87] C. Li and S. L. Waslander, "Towards End-to-end Learning of Visual Inertial Odometry with an EKF," in *Proceedings of 2020 17th Conference on Computer and Robot Vision (CRV), Ottawa, ON, Canada*, 2020, pp. 190–197, doi: 10.1109/CRV50864.2020.00033.

[88] E. J. Shamwell, K. Lindgren, S. Leung, and W. D. Nothwang, "Unsupervised Deep Visual-Inertial Odometry with Online Error Correction for RGB-D Imagery," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2478–2493, Oct. 2020, doi: 10.1109/tpami.2019.2909895.

[89] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1–3, pp. 185–203, Aug. 1981.

[90] A. Dosovitskiy *et al.*, "FlowNet: Learning Optical Flow with Convolutional Networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile*, 2015, pp. 2758–2766.

[91] D. Eigen, C. Puhrsch, and R. Fergus, “Depth Map Prediction from a Single Image using a Multi-Scale Deep Network,” in *Proceedings of Advances in Neural Information Processing Systems, Montréal, QC, CANADA*, 2014, pp. 2366–2374.

[92] A. Kendall, M. Grimes, and R. Cipolla, “PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization,” in *Proceedings of IEEE International Conference on Computer Vision (ICCV), Santiago, Chile*, Dec. 2015, pp. 2938–2946, doi: 10.1109/ICCV.2015.336.

[93] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu, “Relative Camera Pose Estimation Using Convolutional Neural Networks,” in *Proceedings of International Conference on Advanced Concepts for Intelligent Vision Systems, Antwerp, Belgium*, 2017, pp. 675–687, doi: 10.1007/978-3-319-70353-4_57.

[94] S. Wang, R. Clark, H. Wen, and N. Trigoni, “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks,” in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Marina Bay Sands, Singapore*, 2017, pp. 2043–2050, doi: 10.1109/ICRA.2017.7989236.

[95] C. Chen, X. Lu, A. Markham, and N. Trigoni, “IoNet: Learning to cure the curse of drift in inertial odometry,” in *Proceedings of AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA*, 2018, pp. 6468–6476.

[96] C. Chen *et al.*, “Selective Sensor Fusion for Neural Visual-Inertial Odometry,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Long Beach, CA, USA*, 2019, pp. 10542–10551.

[97] B. Hall, *Lie groups, Lie algebras, and representations: an elementary introduction.* Springer, 2015.

[98] F. Chollet, “Keras API,” 2015. https://keras.io (accessed Apr. 11, 2021).

[99] R. Moradi, R. Berangi, and B. Minaei, “A survey of regularization strategies for deep models,” *Artificial Intelligence Review*, vol. 53, no. 6, pp. 3947–3986, Aug. 2020, doi: 10.1007/s10462-019-09784-7.

[100] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, “Robust visual inertial odometry using a direct EKF-based approach,” in *Proceedings of IEEE International*

Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 298–304, doi: 10.1109/IROS.2015.7353389.

[101] K. Sun *et al.*, "Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, Apr. 2018, doi: 10.1109/LRA.2018.2793349.

[102] T. Nguyen, "Computationally-Efficient Visual Inertial Odometry for Autonomous Vehicle," Ph.D. dissertation, Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NL, Canada, 2019.

[103] I. Arasaratnam and S. Haykin, "Cubature Kalman Filters," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, vol. 54, no. 6, pp. 1254–1269, Jun. 2009, doi: 10.1109/TAC.2009.2019800.

[104] D. Abeywardena, S. Huang, B. Barnes, G. Dissanayake, and S. Kodagoda, "Fast, on-board, model-aided visual-inertial odometry system for quadrotor micro aerial vehicles," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden*, 2016, pp. 1530–1537, doi: 10.1109/ICRA.2016.7487290.

[105] J. Nikolic *et al.*, "A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China*, 2014, pp. 431–437, doi: 10.1109/ICRA.2014.6906892.

[106] Z. Zhang, "Efficient Computing for Autonomous Navigation using Algorithm-and-Hardware Co-design," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA, 2019.

[107] B. Nagy, P. Foehn, and D. Scaramuzza, "Faster than FAST: GPU-accelerated frontend for high-speed VIO," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA*, 2020, pp. 4361–4368, doi: 10.1109/IROS45743.2020.9340851.

[108] L. Mussi, F. Daolio, and S. Cagnoni, "Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture," *Information Sciences,*

vol. 181, no. 20, pp. 4642–4657, Oct. 2011, doi: 10.1016/j.ins.2010.08.045.

[109] M. M. Hussain and N. Fujimoto, “GPU-based parallel multi-objective particle swarm optimization for large swarms and high dimensional problems,” *Parallel Computing*, vol. 92, p. 102589, Apr. 2020, doi: 10.1016/j.parco.2019.102589.

[110] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[111] G. G. and B. J. and Others, “Eigen v3,” 2010. http://eigen.tuxfamily.org (accessed Apr. 11, 2021).

[112] G. Groups, “Google Performance Tool.” GitHub, [Online]. Available: https://github.com/gperftools/gperftools.

[113] J. Weidendorfer, “KCachegrind.” [Online]. Available: http://kcachegrind.sourceforge.net/html/Home.html.

[114] R. M. Stallman and Community GCC Developer, “Using the GNU Compiler Collection (GCC).” GNU Press, 2003, Accessed: Feb. 25, 2021. [Online]. Available: http://www.gnupress.org.

[115] “CUDA Compiler Driver NVCC Reference Guide.” 2021, Accessed: Apr. 26, 2021. [Online]. Available: https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html.

[116] H. Strasdat, J. M. M. Montiel, and A. J. Davison, “Visual SLAM: Why filter?,” *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, Feb. 2012, doi: 10.1016/j.imavis.2012.02.009.

[117] H. K. Fathy, Z. S. Filipi, J. Hagena, and J. L. Stein, “Review of hardware-in-the-loop simulation and its prospects in the automotive area,” in *Modeling and Simulation for Military Applications*, May 2006, vol. 6228, no. 22, p. 62280E, doi: 10.1117/12.667794.

[118] B. Moons, D. Bankman, and M. Verhelst, *Embedded Deep Learning, Algorithms, Architectures and Circuits for Always-on Neural Network Processing*. Springer International Publishing, 2019.

[119] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, “Continual learning for robotics: Definition, framework, learning strategies,

opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, Jun. 2020, doi: 10.1016/j.inffus.2019.12.004.

[120] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar*, 2014, pp. 1724–1734, doi: 10.3115/v1/d14-1179.

[121] A. H. Mohamed and K. P. Schwarz, "Adaptive Kalman filtering for INS/GPS," *Journal of Geodesy*, vol. 73, no. 4, pp. 193–203, May 1999, doi: 10.1007/s001900050236.

[122] D. L. Alspach, "A Parallel Filtering Algorithm for Linear Systems with Unknown Time Varying Noise Statistics," *IEEE Transactions on Automatic Control*, vol. 19, no. 5, pp. 552–556, Oct. 1974, doi: 10.1109/TAC.1974.1100645.

[123] T. Bohlin, "Four cases of identification of changing systems," *Mathematics in Science and Engineering*, vol. 126, no. C, pp. 441–518, Jan. 1976, doi: 10.1016/S0076-5392(08)60878-4.

[124] B. J. Odelson, M. R. Rajamani, and J. B. Rawlings, "A new autocovariance least-squares method for estimating noise covariances," *Automatica*, vol. 42, no. 2, pp. 303–308, Feb. 2006, doi: 10.1016/j.automatica.2005.09.006.

[125] K. A. Myers and B. D. Tapley, "Adaptive Sequential Estimation with Unknown Noise Statistics," *IEEE Transactions on Automatic Control*, vol. 21, no. 4, pp. 520–523, Aug. 1976, doi: 10.1109/TAC.1976.1101260.

[126] R. Mehra, "On the identification of variances and adaptive Kalman filtering," *IEEE Transactions on Automatic Control*, vol. 15, no. 2, pp. 175–184, Apr. 1970, doi: 10.1109/TAC.1970.1099422.

[127] K. Theofilatos, G. Beligiannis, and S. Likothanassis, "Combining evolutionary and stochastic gradient techniques for system identification," *Journal of Computational and Applied Mathematics*, vol. 227, no. 1, pp. 147–160, May 2009, doi: 10.1016/j.cam.2008.07.014.

[128] J. Sun and X. Liu, "A novel APSO-aided maximum likelihood identification method

for Hammerstein systems," *Nonlinear Dynamics*, vol. 73, no. 1–2, pp. 449–462, Jul. 2013, doi: 10.1007/s11071-013-0800-4.

[129] K. A. Acharya, R. Venkatesh Babu, and S. S. Vadhiyar, "A real-time implementation of SIFT using GPU," *Journal of Real-Time Image Processing*, vol. 14, no. 2, pp. 267–277, Feb. 2018, doi: 10.1007/s11554-014-0446-6.