

**NAVIGATION OF UNMANNED AERIAL VEHICLES  
IN GPS-DENIED ENVIRONMENTS**

**Jinjiang Cui**

*(M.Eng., Northwestern Polytechnical University, 2008)*

**A THESIS SUBMITTED  
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY**

**NUS GRADUATE SCHOOL FOR INTEGRATIVE  
SCIENCES AND ENGINEERING  
NATIONAL UNIVERSITY OF SINGAPORE**

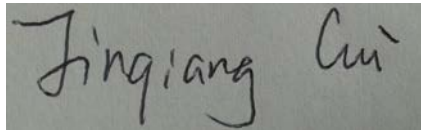
**2014**



## Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

A rectangular box containing a handwritten signature in black ink. The signature reads "Jinqiang Cui".

---

**JINQIANG CUI**

**February 24, 2015**

# Acknowledgments

First, my sincere gratitude goes to my supervisor, Professor Ben M. Chen, for his constant support and guidance during my Ph.D. study. Having been working in the MEMS industry for five years, I have found it extremely hard to pick up new knowledge in the UAV discipline. Prof. Chen has allowed me enough time to grasp the knowledge points and achieve a better understanding of the UAV technology. The encouragement and patience I have received from Prof. Chen are the key buoyancies which keep my Ph.D. boat from sinking in the past four years. Invaluable opportunities to take part in international competitions are not possible without Prof. Chen's support, through which I have gained much insights into the UAV area.

I am grateful to my co-supervisors, Professor Tong H. Lee and Dr. Chang Chen, for their kind encouragement and generous help. Prof. Lee has provided me with great teaching assistant opportunities, which have helped me think out of the box – ‘teaching is indeed the best way for learning’.

I would also like to thank my thesis advisory committee chair, Professor Shuzhi Ge, for his insightful comments to my research work.

Special thanks go to the NUS Unmanned Aircraft Systems Group. Working with the kind and talented fellow researchers has been a rewarding experience. In particular, I would like to thank my seniors: Dr. Feng Lin has helped propose the project for UAV navigation in forests; Prof. Biao Wang and Dr. Guowei Cai have provided generous help modeling the coaxial helicopter; Dr. Xiangxu Dong and Peidong Liu have helped on many onboard software issues; the discussions with Dr. Fei Wang have brought new ideas towards my first autonomous flight. In addition, Mr. Shupeng Lai has developed the path planning algorithm. The cooperation with Dr. Kevin Ang and Dr. Swee King Phang in other UAV competition events have led to lots of insights for this PhD research. I am also thankful for the generous help from all other group members and friends including

Dr. Shiyu Zhao, Kun Li, Jing Lin, Kangli Wang, Xiang Li, Limiao Bai, Zhaolin Yang, Di Deng, Tao Pang, Yijie Ke, Yingcai Bi and Jiaxin Li.

Moreover, I am grateful to my wife Wei Zhang and my parents-in-law. I sincerely thank my wife for the years of support and companion, from China to Germany and to Singapore. My parents-in-law have supported me in the financial and mental aspects ever since I met my wife.

Finally, I would like to thank my parents and my sister, for their everlasting love and care. My parents have been supportive for my decisions in my journey of education and research. My sister has shared the responsibility of taking care of the family.

# Contents

<b>Summary</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Literature Review . . . . .	2
1.2.1 GPS-denied Navigation . . . . .	2
1.2.2 Laser Data Scan Matching . . . . .	4
1.2.3 Simultaneous Localization and Mapping . . . . .	5
1.3 Challenges of This Study . . . . .	7
1.4 Thesis Outline . . . . .	8
<b>2 Design of UAV Platforms</b>	<b>10</b>
2.1 Introduction . . . . .	10
2.2 UAV Bare Platform Design . . . . .	11
2.2.1 Review of UAV Platform Configuration . . . . .	11
2.2.2 Comparison of VTOL Platforms . . . . .	14
2.2.3 Platform Selection and Design . . . . .	16
2.3 Avionics System Design . . . . .	20
2.3.1 UAV Function Blocks . . . . .	20
2.3.2 Avionics System Components . . . . .	21
2.3.3 Avionics System Integration . . . . .	29
2.4 Conclusion . . . . .	32

<b>3</b>	<b>Modeling and Control of UAV Platforms</b>	<b>34</b>
3.1	Introduction . . . . .	34
3.2	Modeling of Coaxial Helicopter . . . . .	35
3.2.1	Comprehensive Dynamics Model Structure . . . . .	35
3.2.2	Linear Dynamics Model and Parameter Identification . . . . .	42
3.3	Modeling of Quadrotor . . . . .	48
3.3.1	Overview of Quadrotor Model . . . . .	48
3.3.2	Linearized Model Identification . . . . .	50
3.3.3	Control Law Design . . . . .	55
3.3.4	Flight Test Results . . . . .	58
3.4	Conclusion . . . . .	59
<b>4</b>	<b>UAV State Estimation Using Laser Range Finder</b>	<b>62</b>
4.1	Introduction . . . . .	62
4.2	Feature Extraction . . . . .	64
4.2.1	Laser Range Finder Model . . . . .	64
4.2.2	Feature Extraction Procedure . . . . .	65
4.2.3	Scan Segmentation Algorithm . . . . .	67
4.2.4	Geometric Descriptors . . . . .	69
4.2.5	Feature Extraction Result . . . . .	73
4.3	Scan Matching . . . . .	74
4.3.1	Iterative Closest Point Matching . . . . .	74
4.3.2	Data Association . . . . .	76
4.3.3	Rigid Transformation Estimation . . . . .	79
4.3.4	Experiment Evaluation . . . . .	81
4.4	IMU-driven State Estimation . . . . .	84
4.5	Autonomous Flight Test . . . . .	88
4.6	Conclusion . . . . .	89
<b>5</b>	<b>Offline Consistent Localization and Mapping using GraphSLAM</b>	<b>92</b>
5.1	Introduction . . . . .	92
5.2	GraphSLAM System Structure . . . . .	93
5.3	GraphSLAM Back-end . . . . .	95

5.3.1	GraphSLAM Formulation . . . . .	95
5.3.2	Loop Detection . . . . .	97
5.3.3	Graph Optimization . . . . .	100
5.3.4	Error Linearization for 2D Poses . . . . .	103
5.4	Offline GraphSLAM Evaluation . . . . .	104
5.4.1	GraphSLAM Software Development . . . . .	104
5.4.2	Consistent Mapping with Synthetic Data . . . . .	106
5.4.3	Loop Closure Detection . . . . .	107
5.4.4	GraphSLAM Parameter Tuning . . . . .	110
5.5	Conclusion . . . . .	113
<b>6</b>	<b>Autonomous Flights with Online GraphSLAM</b>	<b>115</b>
6.1	Introduction . . . . .	115
6.2	Online GraphSLAM using Sliding Window . . . . .	116
6.3	Online Path Planning . . . . .	119
6.4	Onboard Software Development . . . . .	123
6.5	Experiment Results . . . . .	125
6.5.1	Autonomous Flight with Online GraphSLAM . . . . .	125
6.5.2	Autonomous Flight in Small Scale Forest . . . . .	127
6.5.3	Autonomous Flight with Online GraphSLAM and Online Path Planning . . . . .	129
6.6	Conclusion . . . . .	132
<b>7</b>	<b>Conclusions and Future Works</b>	<b>133</b>
7.1	Contributions . . . . .	133
7.2	Future Works . . . . .	135
	<b>Bibliography</b>	<b>145</b>
	<b>List of Author's Publications</b>	<b>146</b>



# Summary

This thesis studies the navigation and control of unmanned aerial vehicles (UAVs) in GPS-denied cluttered environments, such as forests. Research on modeling and control, state estimation, and simultaneous localization and mapping (SLAM) has been carried out with actual implementation and tests in real forest environments. Quadrotor and coaxial helicopter platforms are constructed and utilized in the flight experiments. A UAV state estimation framework has been presented to fuse the outputs of an inertial measurement unit (IMU) with that of scan matching. Taking forests as an example, tree trunks are extracted from data collected by the laser range finder based on a group of geometric descriptors. They are used as feature points in the scan matching algorithm to produce incremental velocity measurements. These measurement are then fused with the acceleration of the IMU in a Kalman filter. To achieve consistent mapping, GraphSLAM techniques are developed to formulate all the poses and measurements in a nonlinear least squares problem. Both an offline and an online GraphSLAM algorithms are developed, with the former one for the algorithm evaluation and the latter one for real-time flight control. The online GraphSLAM is based on a sliding window technique with constant time complexity. The proposed navigation system has been extensively and successfully tested in indoor and foliage environments.

# List of Tables

2.1	Comparison of three VTOL configurations . . . . .	15
2.2	Overview of the specifications of popular IMUs. . . . .	23
2.3	Typical specifications of range sensors. . . . .	24
2.4	Summary of three LionHubs. . . . .	32
3.1	Physical meaning of control input variables . . . . .	36
3.2	Physical meaning of state variables . . . . .	38
3.3	Parameters for roll-pitch dynamics. . . . .	44
3.4	Identified parameters of coaxial helicopter. . . . .	48
4.1	Hokuyo UTM-30LX specification . . . . .	64
4.2	List of geometric threshold for tree trunk extraction. . . . .	70
5.1	GraphSLAM parameter tuning table . . . . .	112

# List of Figures

2.1	Two single rotor UAV platforms. . . . .	11
2.2	Autonomous landing of Boeing’s Unmanned Little Bird . . . . .	12
2.3	List of coaxial UAV platforms. . . . .	13
2.4	List of quadrotor UAV platforms. . . . .	15
2.5	Kaa-350 coaxial helicopter. . . . .	17
2.6	The coaxial platform fuselage head. . . . .	17
2.7	Close view of the coaxial helicopter. . . . .	18
2.8	Coaxial helicopter flying in the air. . . . .	18
2.9	NUS quadrotor virtual design. . . . .	19
2.10	NUS quadrotor platform. . . . .	19
2.11	UAV functions with the required avionics system . . . . .	21
2.12	UAV avionics system diagram. . . . .	22
2.13	State-of-the-art IMUs suitable for UAV applications. . . . .	23
2.14	List of range sensors. . . . .	24
2.15	Two 3D laser scanners from Velodyne and SICK. . . . .	25
2.16	List of vision sensors. . . . .	26
2.17	The SLAM sensor suite developed by ETH. . . . .	27
2.18	High performance onboard computer Mastermind. . . . .	28
2.19	Gumstix Overo Fire computer-on-module. . . . .	28
2.20	Two-board configuration of servo control board. . . . .	29
2.21	One-board configuration of servo control board: UAV100. . . . .	30
2.22	A typical avionics system configuration for coaxial helicopter. . . . .	31
2.23	LionHub V1: first design featuring low cost and compact volume. . . . .	32
2.24	LionHub V2 and its application in T-Rex 90. . . . .	32
2.25	LionHub V3 and its application in quadrotor. . . . .	33

3.1	Fuselage head of coaxial helicopter with labeled key components. . . . .	36
3.2	Model structure of coaxial helicopter. . . . .	37
3.3	Definition of NED frame $O_n$ and body frame $O_b$ . . . . .	37
3.4	Testing of moment of inertia using trifilar pendulum. . . . .	40
3.5	Frequency response from roll input to roll angular rate. . . . .	44
3.6	Frequency response from pitch input to pitch angular rate. . . . .	45
3.7	Time domain verification from roll input to roll angular rate. . . . .	45
3.8	Time domain verification from pitch input to pitch angular rate. . . . .	46
3.9	Frequency response for Heave dynamics model identification. . . . .	46
3.10	Yaw rate feedback controller structure. . . . .	47
3.11	Overview of quadrotor model structure. . . . .	49
3.12	Quadrotor body frame definition. . . . .	49
3.13	Response comparison using frequency-sweep input $\{\delta_{ail}, \delta_{ele}\} - \{\phi, \theta\}$ . . . . .	51
3.14	Roll angle time domain model verification. . . . .	52
3.15	Roll angle time domain error between model prediction and experiment. . . . .	52
3.16	Time domain error from roll angle to y velocity. . . . .	53
3.17	Time domain comparison of yaw angle. . . . .	54
3.18	Time domain comparison of yaw angular rate. . . . .	54
3.19	Time domain comparison of heave velocity. . . . .	55
3.20	Control structure of the quadrotor UAV. . . . .	55
3.21	x direction tracking performance. . . . .	59
3.22	y direction tracking performance. . . . .	60
3.23	z direction tracking performance. . . . .	60
3.24	Yaw direction tracking performance. . . . .	61
4.1	The architecture of the IMU-driven Kalman filter. . . . .	63
4.2	Image and laser scanner data for a testing scenario. . . . .	64
4.3	Laser range finder measurement model. . . . .	66
4.4	Test scenario with UAV flying in the air. . . . .	66
4.5	Typical laser measurement in a foliage environment. . . . .	67
4.6	Segmentation threshold determination. . . . .	69
4.7	Two circle fitting methods using the bounding angle of clusters. . . . .	72

4.8	Comparison of three circle fitting algorithms . . . . .	73
4.9	One raw scan with the labeled clusters. . . . .	74
4.10	Clean scan with extracted circles. . . . .	75
4.11	Close view of three extracted circles. . . . .	75
4.12	Procedures of the ICP algorithm. . . . .	76
4.13	Initial transformed synthetic data. . . . .	77
4.14	Change of error in each iteration. . . . .	78
4.15	The aligned datasets after ICP. . . . .	78
4.16	The indoor test scenario for verifying scan matching. . . . .	81
4.17	Motion and path estimate at the start of path. . . . .	83
4.18	Motion and path estimate at the end of path. . . . .	83
4.19	Velocity and incremental heading angle estimates from scan matching. . . . .	84
4.20	Comparison between dead reckoning, scan matching and Kalman filter. . . . .	88
4.21	The testing scenario with the flying quadrotor. . . . .	89
4.22	Position tracking in x-y plane with the tracking error. . . . .	90
4.23	Position reference tracking in x-y plane. . . . .	90
5.1	GraphSLAM system structure. . . . .	93
5.2	System schematics illustrating front-end and back-end. . . . .	94
5.3	Composition of a graph. . . . .	95
5.4	GraphSLAM illustration [74]. . . . .	96
5.5	Loop closure after traveling a certain time. . . . .	97
5.6	Global and local search in loop detection. . . . .	99
5.7	Comparison of information matrix between local and global search. . . . .	100
5.8	The pose-graph structure in GraphSLAM. . . . .	100
5.9	Optimized trajectory comparison between Matlab and C++ . . . . .	106
5.10	Optimized map and trajectory in simulation environment. . . . .	107
5.11	Optimized tree contour projected on the optimal pose. . . . .	108
5.12	x position difference with respect to ground truth. . . . .	108
5.13	y position difference with respect to ground truth. . . . .	108
5.14	Heading angle difference with respect to ground truth. . . . .	109
5.15	Drifted map before loop closure. . . . .	110

5.16	Consistent map after loop closure. . . . .	111
5.17	Map details before and after loop closure. . . . .	111
5.18	Close view of optimized map compared to initial map. . . . .	113
5.19	Tree contour details for indoor forest using GraphSLAM. . . . .	114
6.1	System diagram of UAV navigation system. . . . .	115
6.2	Sliding window diagram with poses being pushed in and popped out. . .	117
6.3	A timing graph between front-end and back-end. . . . .	118
6.4	A Gaussian cost map in polar coordinate. . . . .	121
6.5	UAV response together with reference in map. . . . .	122
6.6	Software structure of UAV navigation system. . . . .	124
6.7	Indoor test scenario for GraphSLAM verification. . . . .	126
6.8	Comparison of initial map and optimized map using GraphSLAM. . . .	126
6.9	Close view of the optimized map compared to the initial map . . . . .	127
6.10	UAV flying in the small scale forest in front central library of NUS. . . .	128
6.11	Optimized map and trajectory in small forest. . . . .	129
6.12	Miscellaneous tree trunk conditions. . . . .	129
6.13	Consistent map with obstacle avoidance trajectory. . . . .	130
6.14	Onboard trajectory tracking performance with obstacle avoidance. . . .	131

# List of Symbols

## Latin variables

$\mathbf{F}_b$	aerodynamic forces vector
$F_{bx}$	body frame x axis aerodynamic force component
$F_{by}$	body frame y axis aerodynamic force component
$F_{bz}$	body frame z axis aerodynamic force component
$g$	the acceleration of gravity
$h$	NED frame altitude
$J_{xx}$	rolling moment of inertia
$J_{yy}$	pitching moment of inertia
$J_{zz}$	yawing moment of inertia
$K_I$	integral gains of the embedded controller
$K_P$	proportional gains of the embedded controller
$\mathbf{M}_b$	moment vector
$M_{bx}$	body frame rolling moment component
$M_{by}$	body frame pitching moment component
$M_{bz}$	body frame yawing moment component
$m$	mass of helicopter
$p$	body frame rolling angular velocity
$\mathbf{P}_n$	position vector in NED frame
$q$	body frame pitching angular velocity
$r$	body frame yawing angular velocity
$\mathbf{V}_b$	velocity vector in body frame
$\mathbf{V}_n$	velocity vector in NED frame
$w$	body frame z axis velocity

## Greek variables

$\delta_{\text{lat}}$	aileron servo input
$\delta_{\text{lon}}$	elevator servo input
$\delta_{\text{col}}$	collective pitch servo input
$\delta_{\text{ped}}$	rudder servo input
$\bar{\delta}_{\text{ped}}$	intermediate state in lower rotor dynamics
$\theta$	pitching angle in NED frame
$\phi$	rolling angle in NED frame
$\psi$	yawing angle in NED frame

## Acronyms

CG	center of gravity
CIFER	comprehensive identification from frequency responses
COTS	commercial off-the-shelf
CPU	central processing unit
DOF	degree-of-freedom
DR	dead reckoning
ESC	electronic speed controller
EKF	extended Kalman filter
FPGA	field-programmable gate array
GCS	ground control station
GPS	global positioning system
GNC	guidance, navigation and control
ICP	iterative closest point
IMU	inertial measurement unit
INS	inertial navigation system
KF	Kalman filter
Li-Po	lithium-polymer
LiDAR	light detection and ranging
LRF	laser range finder
MAV	micro aerial vehicle
NDT	normal distributions transform



NED	north-east-down
NUS	National University of Singapore
PWM	pulse-width modulation
RAM	random-access memory
RC	radio-controlled
RPM	rotations per minute
RS232	recommended standard 232
SISO	single-input/single output
SLAM	simultaneous localization and mapping
SVD	singular value decomposition
TPP	tip-path-plane
UAV	unmanned aerial vehicle
UGV	unmanned ground vehicle
UKF	unscented Kalman filter
USB	universal serial bus
WiFi	wireless fidelity
VTOL	vertical takeoff and landing
2D	two-dimensional
3D	three-dimensional



# Chapter 1

## Introduction

### 1.1 Introduction

Unmanned aerial vehicles (UAVs) are being applied to more applications, such as disaster monitoring, environment and traffic surveillance, search and rescue, aerial mapping, and cinematography [63]. With the increasing awareness of the UAV potentials, the requirements for UAVs are becoming more demanding and versatile. For example, UAVs are required to operate in obstacle-strewn environment, such as urban canyons and forests, without the aid of the global positioning system (GPS). Such requirements have led to the research goal of this study: To develop an advanced navigation system for UAVs to enable them to autonomously navigate in uncertain and cluttered outdoor environments, such as hostile buildings, radiation contaminated areas and forests. Most current research efforts for outdoor navigation of UAVs focus on the obstacle-free environments. The development of UAVs in obstacle-strewn environments is still in its infancy. Obstacle-strewn environments usually affect the reliability of inertial navigation systems due to the loss of GPS signals. Because of the large operation range in outdoor environments, long-range obstacle sensing technologies and map generation techniques are required. The limited payload capability of UAVs also greatly complicates the design of onboard systems, making it difficult for the system to achieve specified navigation tasks and obstacle avoidance.

In this thesis, we propose to develop an advanced outdoor navigation system for UAVs to achieve autonomous navigation in outdoor GPS-denied environments, such as forests. To develop the navigation system, several main topics need to be investigated,

including advanced sensing technologies, sophisticated navigation approaches and simultaneous localization and mapping (SLAM) techniques. A variety of sensing technologies are considered in the research, including electro-optical (EO) sensors, light detection and ranging (LiDAR) sensors and IMUs. The fusion techniques are investigated to combine the measurements of these sensors to realize robust navigation and obstacle detection without GPS. Special attention is paid to the SLAM problem in large-scale environments. In addition, a path planning scheme is studied to determinate a safe path for UAVs to successfully carry out required missions. All the algorithms developed in this thesis are verified by actual flight tests in forests.

## 1.2 Literature Review

### 1.2.1 GPS-denied Navigation

The navigation of mobile robotics platforms in GPS-denied environments has been intensively studied in the research community, in environments such as indoor offices [82, 68, 67], underwater [59] and urban canyons [28]. Without GPS signals, the robot platform has to rely on its onboard sensors for state estimation. The two most popular techniques are laser odometry [72] and visual odometry [65, 26]. Both methods are based on the 2 dimensional (2D) or 3 dimensional (3D) point cloud matching approach, which seek to match two sets of points to extract incremental transformation.

The use of vision perception techniques to aid UAV localization and mapping has been heavily investigated in the literature, and is still a hot research topic. Vision sensing is attractive due to its induced rich information and the light weight of the camera systems. The bottleneck limiting its applications is the intensive computation required by the vision processing pipeline, including feature detection and tracking, etc. The techniques used in the vision community can be categorized according to the camera configuration: the stereo camera configuration or the single camera configuration. Researchers in MIT [1] are the pioneers who first evaluated the possibility of integrating stereo vision odometry on a quadrotor for indoor applications. In 2013, Schmid *et al.* [66] reported about their stereo-based autonomous navigation of a quadrotor in indoor and outdoor environments, in which field-programmable gate array (FPGA) is used to process the stereo images using a semi-global matching algorithm [34]. For UAV naviga-

tion based on mono cameras, researchers in ETH<sup>1</sup> have produced some very promising results [64]. Using a hexacopter equipped with a single onboard downward-facing camera and an IMU, efficient state estimation and mapping of the environment have been achieved with three UAVs. However, the camera orientation is confined to pointing downward for feature detection and tracking on the ground. This means that the UAV has to fly high above the ground to get a large image overlap. This solution is thus not yet applicable to cluttered environments such as urban canyons and forests.

Laser sensing provides accurate range and bearing measurements, making it an ideal choice for mobile platforms. Early successful uses of laser range finders are mainly for obstacle detection and environment mapping. For example, Thrun *et al.* of Stanford University used five SICK laser range finders on a Volkswagen Touareg R5 for the DARPA Grand Challenge 2005 [75]. The laser range finders are extensively used for terrain mapping and obstacle detection, whereas the position of the vehicle is estimated using the GPS assembled on top of the car. The use of a light-weight scanning laser range finder on a quadrotor to achieve autonomous navigation are reported in [5, 67]. More relevantly, Wang *et al.* [82, 79] in National University of Singapore have produced interesting results for UAV navigation in indoor environments. A laser range finder and a monocular camera are used for the autonomous navigation of a quadrotor with a heuristic wall-following strategy.

The navigation of UAVs in outdoor GPS-denied environments, especially in forests, is rarely covered in the research community. Outdoor GPS-denied environments exhibit their own challenges, including complex terrain conditions, cluttered environment, etc. The navigation of ground vehicles in foliage environments has been addressed in [33, 32] reporting a car equipped with a laser range finder driving through Victoria Park in Sydney, Australia. The steep terrain, thick understorey vegetation, and abundant debris characteristic of many forests prohibit the deployment of an autonomous ground vehicle in such scenarios. Achieving autonomous flight of UAVs in forests has been attempted using a low-cost IMU and a monocular camera [41], in which an unscented Kalman filter (UKF) was used to estimate the locations of obstacles and the state of the UAV. Their experiment verification was carried out with a radio-controlled (RC) car running in a synthetic outdoor environment. More recently, Ross *et al.* [60] realized autonomous

---

<sup>1</sup><http://www.asl.ethz.ch>

flight through forests by mimicking the behavior of a human pilot using a novel imitation learning technique. The application of the learning technique is innovative, but the system suffers from a relatively high failure rate which a practical UAV cannot afford. Ultimately, a UAV with autonomous navigation capability in foliage environments would be of paramount importance for forest surveys, exploration, and reconnaissance [17].

### 1.2.2 Laser Data Scan Matching

Laser range finders are popular and promising sensors because of the accurate 3D point cloud they can generate, either by rotating a 2D laser scanner or an inherent 3D laser scanner. An accurate 3D point cloud is the cornerstone of extracting the relative transformation between two 3D scans. To align two 3D scans, two dominant methods are the iterative closest point (ICP) [8, 16] and the normal distributions transform (NDT) [9].

Starting with an initial guess, ICP obtains the transformation by repeatedly generating pairs of corresponding points and minimizing an error metric. The seminal work [61] separates ICP into six stages, four of which are point selection, point matching, error metric assignment and error minimization. A large number of ICP variants exist based on different strategies in any of the six stages. The two main steps are point selection and error minimization. Selecting the points for scan matching is the first critical step in ICP, affecting its accuracy and speed. There are different strategies: using all the available points [8], uniform subsampling of all the points [77], random sampling of the points [52], or using points with high intensities plus illumination information [83]. Using all the points is infeasible in practice due to the large number of measurement points, especially for 3D range scans. Subsampling or feature extraction is thus always desirable. The most popular error metrics are point-to-point error metric [8] and point-to-plane error metric [16]. The point-to-point error metric leads to a closed-form solution for determining the rigid-body transformations while minimizing the error. The point-to-plane error metric can be solved using a generic nonlinear method (e.g. Levenberg-Marquardt) [61].

Specifically, extracting features from laser range scans before scan matching is always preferable for onboard implementations. Indoor environments have structured walls and pillars, from which corners and lines can be extracted as features for scan matching [80]. In foliage environments, using tree stems as features for navigation has been studied by

researchers. Tree stems are assumed to be circular in shape, and can thus be extracted from the laser measurement points. In [32], the circle parameters are estimated with the clustered point together with Kalman filter-based tracking. In [6], the tree model is derived from the cluster bounding angle and the minimum range. Natural landmark extraction based on adaptive curvature estimation has been proposed in [58]. This curvature estimation applies only to segments with more than 10 points. This condition constrains its application to forest environments, as trees with small diameter cannot produce enough measurement point for the curvature estimation. In [70], the authors used static 3D laser range images to extract tree diameters and axes, but this is not applicable to UAVs which they are constantly moving.

The normal distributions transform is another promising alternative to register two sets of points. Given a first set of points, the space is divided into grids of equal size, and the probability of a point at a certain position is modeled by a collection of normal distributions [9]. Points from the second set are transformed to the first scan frame using the initial rigid transformation and an error metric is chosen to be the sum of the local normal distribution. NDT for 3D datasets has also been developed [50] and compared with ICP [51]. The NDT method is faster than ICP since normal distribution is used as the matching criteria, instead of the point-to-point nearest neighbor search. However, the NDT is reported to only work well in environments with enough structure, like indoor offices and mine tunnels. Outdoor environments such as urban canyons or forests may return sparse laser range data, making the NDT less appropriate.

### 1.2.3 Simultaneous Localization and Mapping

The navigation of UAVs requires the availability of both poses and maps at the same time. The research issue of estimating the map and pose at the same time is often referred to as the simultaneous localization and mapping (SLAM) problem. Localization and mapping are two interleaving processes: to better localize itself, a robot needs a consistent map; to acquire the consistent map, the robot requires a good estimate of its location. Any uncertainty in either localization or mapping increases the uncertainty of both processes. There are various SLAM approaches to tackle this dilemma, and the mainstream methodologies can be categorized into three formulations: extended Kalman filters (EKF-SLAM), particle filters (FastSLAM) and graph-based nonlinear optimiza-

tion (GraphSLAM). A comprehensive overview of the SLAM algorithms is presented in [22, 7]. All the three methods have their own merits and drawbacks.

Using EKF in SLAM has been proposed in the seminal paper [69] and later applied to a ground vehicle navigation [44]. The state vector of the EKF includes both the robot pose and the landmarks' coordinates. A covariance matrix of the same size as the augmented state is kept to represent the estimate uncertainty. Successful applications of EKF have been achieved in a wide range of practical mapping problems, including various robotic vehicles in the air, on the ground and underwater [73]. The primary drawback of the EKF-SLAM is the quadratic growth of the covariance matrix in the motion and the measurement update processes with the increasing number of features in the map. This limits EKF-SLAM to relatively scarce maps with less than 1,000 features; otherwise it is difficult for the data association. Another shortcoming of EKF-SLAM is the Gaussian noise model assumption of the motion model and the measurement model. This assumption is in practice not realistic, thus additional techniques to deal with spurious landmarks have to be adopted.

The second paradigm of SLAM is based on the Rao-Blackwellized particle filters [55, 56, 30]. The aim is to represent the state and map using a group of particles. Each particle represents one guess of the robot's pose and map in the environment. The curse of dimensionality is even worse for particle filter-based SLAM because the particle filters scale exponentially with the number of dimensions. The curse is released by assuming that the cross-correlation between landmarks are independent if the robot's path is known. This is the prerequisite for applying the *Rao-Blackwellized particle filters* to SLAM, or the FastSLAM [55]. FastSLAM uses particle filters to estimate the robot's path, each particle stores a guess of the robot's pose and a list of mean/covariance pairs of the landmark locations. The key advantage of FastSLAM is the robustness of data association, because the posterior is based on the voting of multiple data association in each particle. Another advantage of FastSLAM lies in the fact that particle filters can cope with nonlinear robot motion models. But the disadvantage of FastSLAM lies the resampling step, in which the low-probability particles are discarded and the high-probability ones are duplicated. This resampling strategy means that the correlation information between landmarks is gradually lost over time, causing problems when a large loop closure is required.



Graph-based nonlinear optimization techniques serve as the third major SLAM paradigm, i.e., GraphSLAM [74]. The basic idea is to optimize all the poses on the trajectory such that the maximum likelihood measurement is achieved. To form the graph, all the robot's poses and landmarks at a particular time represent nodes of a graph. The spatial constraints between the poses represent the edges. Once such a graph is constructed, the goal is to find a spatial configuration of the nodes that is most consistent with the constraints provided by the edges [46]. This involves solving a large error minimization problem. The state-of-the-art algorithms take advantage of the development of direct linear solvers and the sparseness of the graph constraints. Framework such as iSAM [36] and g<sup>2</sup>o [40] are available to serve as the non-linear optimization tools. From the perspective of users, only the construction of the graph is required.

### 1.3 Challenges of This Study

Navigation of UAVs without the help of GPS is itself a difficult task. It becomes even harder if there are obstacles in the vicinity of UAVs, requiring a range of autonomous capabilities including robust and perfect control, real-time path planning, and accurate motion estimation, etc. The main challenges for this study are identified as follows:

- **GPS-denied environment:** urban canyons and foliage environments render the GPS signals unreliable and inconsistent, making it impossible to navigate using GPS signals. Artificial beacons can be placed in advance but this is not feasible for most practical applications. To tackle this problem, localization of UAVs using onboard sensors, such as IMUs, laser range finders and vision sensing techniques, is to be evaluated and assessed.
- **Unknown map:** no prior map of the environment is provided for the UAV navigation, either in urban canyons or forests. This poses great challenges for onboard path planning and obstacle avoidance. The path planning algorithm must be fast enough to deal with unexpected objects, such as dynamic objects in the environment itself. The obstacle avoidance module is required to be reactive enough to avoid any obstacles measured by the onboard sensors.
- **No human intervention:** the UAVs are required to be fully autonomous once

started. The whole mission cannot be helped by any human intervention, meaning that all the developed algorithms have to be intelligent and comprehensive enough. Developing real-time onboard computing using the limited computing units is considered a big challenge.

- **Cluttered environment:** environments like urban canyons and forest are quite different from structured environments like indoor offices. The 2.5D assumption is not met since the environment consists of objects not consistent in the vertical direction. Using only a 2D laser range finder is thus not applicable in this case. The state-of-the-art 3D laser scanner is still too heavy for small-scale UAVs. The possible solution is either to spin a 2D laser scanner or use a stereo camera system.
- **Complex terrain condition:** the terrains of urban canyons and forests are uneven and covered with small and light objects like fallen leaves. The uneven terrain makes it even more challenging as the path planning has to guide the UAVs in the vertical direction besides the horizontal plane. The small objects may be blown away while the UAV flies by, causing dynamic objects to be captured in the onboard sensors, and making state estimation and obstacle avoidance even harder.
- **Limited payload:** the UAV platform has to be compact enough to fly through confined environments. Thus the avionics system, including the sensing modalities and computing units, cannot be too bulky or heavy. Only sensors with limited range capability and embedded computers with small footprints can be considered. The system integration of hardware and software is expected to be demanding.

## 1.4 Thesis Outline

This Ph.D. study has been dedicated to solve the problem of UAV navigation in GPS-denied environments with limited onboard payload capability. Each chapter in this thesis covers different topics, such as design and construction of platform, modeling and control of UAVs, and state estimation, etc. The outline of the thesis is as follows:

1. Chapter 2 addresses the topic of platform design, including the bare platform selection and the avionics system design. A wide range of state-of-the-art platforms are reviewed with the conclusion that coaxial and quadrotor are the two most

promising platforms. Then the requirements of the avionics components are given according to the UAV navigation tasks requirements. Available products suitable for this study are reviewed and selected. To achieve efficient system integration, a customized board is designed and developed to host the essential avionics.

2. Chapter 3 identifies the models for the coaxial and quadrotor UAVs. The model structure is formulated as the inner-loop rotation dynamics and the outer-loop translation dynamics. The rotation dynamics is stabilized by commercial autopilot. A robust and perfect tracking autonomous control law is designed for the outer-loop dynamics of the quadrotor, whose performance is verified by autonomous flights based on GPS.
3. Chapter 4 presents the state estimation of the UAV using a laser range finder. The estimation is based on a Kalman filter to fuse the acceleration measurements of IMU and the laser range finder. Data collected from the laser range finder are segmented to produce features for a scan matching process. The feature-based scan matching method estimates the incremental transformation between consecutive scans. The proposed state estimation solution is verified in actual flight tests.
4. Chapter 5 aims to develop a consistent mapping framework using the GraphSLAM technique. Procedures to build and optimize the graph are studied. The consistent mapping framework is verified using off line data collected during flight tests.
5. Chapter 6 presents autonomous flight test results with the online consistent mapping and online obstacle avoidance. A sliding window technique is applied for constant time GraphSLAM optimization. Software integration issues and onboard obstacle avoidance problems are addressed. All the techniques developed in previous chapters are integrated and verified in actual autonomous flight tests.

## Chapter 2

# Design of UAV Platforms

### 2.1 Introduction

The ultimate purpose of this research is to enable UAVs to fly autonomously in obstacle-strewn GPS-less environments. The testbed platform has to be investigated and constructed first so that flight tests can be performed to verify the navigation algorithms. In cluttered environments with obstacles, an ideal UAV platform should be able to take-off vertically and hover in the air at anytime to avoid the possible collision. Platforms with such capabilities are often referred to as the vertical take-off and landing (VTOL) UAVs. They are often categorized by the number of rotors, i.e., single rotor helicopters, coaxial helicopters, and quadrotors. In order to find a suitable platform for the future algorithms verification, we review the popular VTOL UAV platforms in each category and compare them with respect to several performance indexes. The comparison concludes that coaxial helicopters and quadrotors are the potential solutions. Hence, we design two platforms of each type and construct them.

In addition, to equip the bare platforms with intelligence capability, various avionics components need to be assembled onto UAVs support different navigation tasks. The guidance, navigation and control tasks are identified for UAVs. Corresponding to each task, a wide range of avionics modules, including processors, sensors, hardware-related controllers, etc., are reviewed and evaluated. A dual-computer structure with an IMU and a laser range finder is designed and tested. To facilitate easy integration of these components, three versions of motherboards connecting all the essential avionics modules are designed and assembled to various UAV platforms.

## 2.2 UAV Bare Platform Design

### 2.2.1 Review of UAV Platform Configuration

#### Single Rotor UAV Platforms

Single rotor helicopters have been adopted as UAV platforms due to their conventional design. The accumulative technologies developed in large manned helicopters have made the modeling and control of such UAV platforms very popular. Earlier research about UAVs has been focused on this type of platform. Fig. 2.1 lists two representative UAVs from industries and universities. Yamaha RMAX (Fig. 2.1(a)) is one of the early successful UAVs which is widely used in agriculture and industry applications. Later researchers begin to build their own customized UAVs based on radio-controlled (RC) model helicopters 2.1(b). A comprehensive study is reported by [12], in which the hardware configuration, software integration, aerodynamic modeling and automatic control system are extensively covered.



(a) YAMAHA Rmax



(b) NUS Helion

Figure 2.1: Two single rotor UAV platforms.

Single rotor UAVs have a typical fuselage size of 2.5 - 4 meters, making them ideal platforms for long-endurance flight with larger payload capabilities. The larger fuselage size also makes them more stable. For example, researchers [18] in Carnegie Mellon University have realized autonomous landing using Boeing's Unmanned Little Bird (Fig. 2.2). Operating such a large UAV requires a team of human operators to aid the missions, and its size limits its application in confined environments.



Figure 2.2: Autonomous landing of Boeing’s Unmanned Little Bird

### Coaxial UAV Platforms

Coaxial helicopter is another popular type of VTOL platform. Compared with single rotor helicopters, coaxial helicopters tend to be more compact by removing the tail rotor. They also produce less noise since there is no interaction between the airflow from the main rotor and tail rotors. They also produce better lift efficiency since all the rotors are used to lift the fuselage. Besides, they avoid the dissymmetric lift during forward flight, making them ideal UAV platforms with large payload and sufficient compactness.

The Russian Kamov helicopter design bureau has initiated and led the design of coaxial helicopters in the industry. Fig.2.3(a) is a coaxial UAV named Kamov Ka-37, which is designed for aerial photography, television and radio broadcasting, and several military roles. It uses an engine with 45 kW power, lifting 250 kg total weight with a operation range of 530 km and 45 minutes endurance.

Infotron from France has developed another coaxial UAV - IT180 (Fig. 2.3(b)), which has been designed for military and civil security purposes. IT180 has a rotor diameter of 1.8 m and can fly up to 120 minutes. It is propelled either by a 46 cc, 2-stroke engine or a brushless electric motor. The gasoline version IT180 allows for a maximum payload of 5 kg (3 kg for the electrical version) which can be fastened on the top and/or at the bottom of the structure.

A commercial coaxial UAV from Skybotix named as ‘CoaX’ is shown in Fig. 2.3(c). The CoaX helicopter is the product of research project ‘muFly’ from ETH [25]. It is now a micro UAV targeted at the research and educational markets. Weighing at 320 g, the

helicopter includes an IMU, a downward-looking and three optional sideward-looking sonars, pressure sensor, color camera, and Bluetooth or WiFi communication<sup>1</sup>.

Fig. 2.3(d) shows a coaxial helicopter from National University of Singapore [81]. It is fully customized from a toy helicopter named as ‘ESky Big Lama’. Onboard avionic modules are customized and assembled to realize autonomous flight capabilities. Preliminary indoor navigation is achieved using an onboard laser range finder. The modeling of the helicopter is very complex since the blades are not rigid, and it is further complicated by the aerodynamic interaction between the top rotor and the lower rotor.



(a) Ka-37



(b) France IT180



(c) Skybotics Coax



(d) NUS FeiLion

Figure 2.3: List of coaxial UAV platforms.

### Quadrotor UAV Platforms

Quadrotor platforms have become popular choices for UAV hobbyists and researchers. Compared with single rotors and coaxial helicopters, they have relatively simpler mechanical structure by removing the linkages from motors to rotor blades. This makes

<sup>1</sup><http://www.ros.org/news/robots/uavs>

the design and maintenance of the vehicle less time-consuming. Small-scale quadrotors can also be assembled with protection frames enclosing the rotors, allowing for flights through more challenging environments with a lower risk of damaging the vehicle or its surroundings. Amateur pilots usually use this type of platform to mount high resolution cameras for aerial photography, whereas researchers use this kind of platform to explore high level algorithms, such as SLAM, path planning, etc.

Based on the quadrotor ‘Pelican’ from Ascending Technologies (Fig. 2.4(a)), researchers in Technische Universität München (TUM) and MIT have mounted a stereo vision camera and a laser range finder into ‘Pelican’. The quadrotor is capable of carrying 500 g payload and continuously flying for 10 minutes. More creatively, the front rotor is placed below the arm to avoid camera obstruction while keeping the center of gravity low (see Fig. 2.4(c)). A laser range finder is mounted at the middle of the platform which is in charge of sensing surrounding obstacles. It is also the main sensor to collect information for mapping the environment. This UAV can perform fully autonomous navigation and exploration in indoor environments, including take-off, flying through windows, exploration and mapping, and searching for objects of interest. The whole system has been proven to be robustly stable and practically realizable [1]. Using a similar platform and sensor configuration, researchers in University of Pennsylvania [67] have realized indoor multi-floor exploration (Fig. 2.4(d)). Another noteworthy quadrotor platform is the AR.Drone from Parrot shown in Fig. 2.4(b). It is equipped with two cameras pointing forward and downward respectively, making it an ideal platform for researchers in the computer vision community [24].

### 2.2.2 Comparison of VTOL Platforms

Table. 2.1 summarizes various performance indexes of the three VTOL configurations. The table is adapted from [11] where a more comprehensive comparison of VTOL platforms is given. The comparisons show that the coaxial configuration is the most stable and least maneuverable near hover condition, while the quadrotor configuration is the least stable and most maneuverable. Choices of the platform configuration depend on the mission requirements. If maneuverability is of concern, the coaxial configuration should be discarded. If payload and duration of flight are critical, the coaxial configuration is the choice. In this thesis, the potential UAVs should have large payload





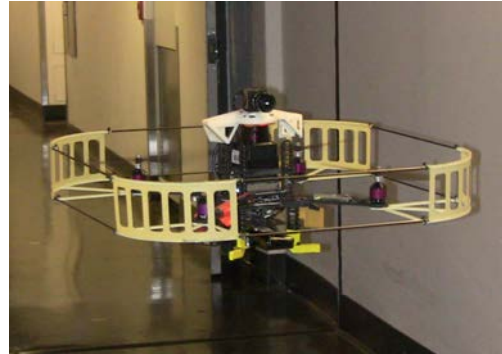
(a) Ascending Tech Pelican



(b) AR.Drone from Parrot



(c) Quadrotor from TUM and MIT



(d) Quadrotor used in Upenn

Figure 2.4: List of quadrotor UAV platforms.

capability with relatively low flight speed, as the UAV has to carry payload comparable to its own weight and perform autonomous flights in confined environments, especially forests. Therefore, we choose the coaxial helicopter and the quadrotor configuration.

Table 2.1: Comparison of three VTOL configurations ( 1 = bad, 4 = very good).

	Single rotor	Coaxial	Quadrotor
Power cost	2	2	1
Control cost	1	4	3
Payload volume	2	4	3
Ease of payload packing	2	2	4
Maneuverability	4	2	3
Mechanics simplicity	1	2	4
Aerodynamics complexity	1	1	4
Low speed flight	4	4	4
High speed flight	2	1	3
Miniaturization	2	4	3
Survivability	1	2	2
Stationary flight	4	4	4
<b>Total</b>	<b>26</b>	<b>32</b>	<b>38</b>

### 2.2.3 Platform Selection and Design

To build a functional UAV platform, the bare platform's frame and the avionic system are the first two things to prepare. Focusing on the navigation capabilities of UAV, we try to minimize the effort spent on the platform construction. For coaxial helicopters, we take advantage of the development of RC model helicopters and select the commercially available helicopter 'Kaa-350' as the basis. For quadrotors, it is straightforward to build such a platform using basic parts, such as carbon tubes, electric motors, autopilots, etc. We design and build the quadrotor platform from scratch. What's more, the avionics system design is of paramount importance and deserves special treatments, which shall be illustrated in Section 2.3.

#### Coaxial UAV Platform

The 'Kaa-350' is a coaxial helicopter made in Germany according to the design of full scale coaxial helicopters from the Kamov Design Bureau. This helicopter has a rotor diameter of 0.7 m and weighs 990 g without battery. Its rotor head is equipped with integrated hinges and shock resistant dampers. With the recommended configuration of motors, ESCs and blades, it can fly safely with a total weight of 2.3 kg. Fig. 2.5 shows the bare helicopter flying in the air by manual remote control. To increase its payload capability, the ESCs and motors are changed to allow for a larger take-off weight.

This helicopter mechanics possesses the typical characteristics of a full-scale coaxial helicopter. As shown in Fig. 2.6, the rotor blades are not assembled in order to better illustrate the structure. The helicopter consists of two counter-rotating rotors: the upper rotor and the lower rotor. The pitch angles of the two rotors are controlled by the top and lower swashplates respectively. The two swashplates are always parallel to each other since they are mechanically connected by three linkages which rotate with the top swashplate. The upper rotor is equipped with a stabilizer bar through a Bell-Hiller mixer which also influences the cyclic pitch of the upper rotor. The upper rotor and lower rotor are driven by the same brushless direct current (DC) electric motor through a chain of gears. The rotation speeds of the upper rotor and the lower rotor are thus always the same. The main power source is a 3-cell lithium-polymer battery. The collective and cyclic inputs from servos are transferred to the lower swashplate and upper swashplate simultaneously, resulting in the dynamic movement of the helicopter

in the heave or pitch-roll direction. The yaw direction control is realized by changing the collective pitch angle of the lower rotor. Fig. 2.7 shows the integrated platform after upgrading the bare platforms and assembling the avionics system. Fig. 2.8 describes the UAV flying in the air.



Figure 2.5: Kaa-350 coaxial helicopter.

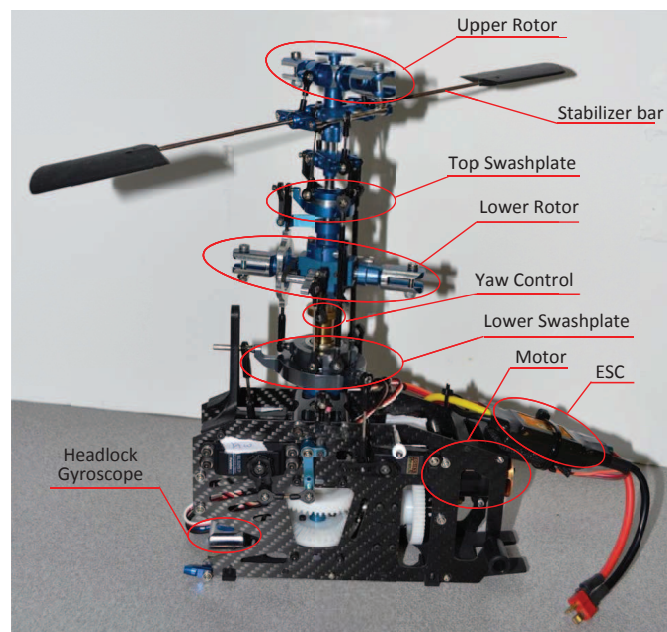


Figure 2.6: The coaxial platform fuselage head.

### Quadrotor Platform

The quadrotor platform is another UAV fully customized (Fig. 2.9-2.10) by NUS UAV Team. The platform is designed to be applicable in both indoor and outdoor environments, such as suburban towns and forested areas. The platform is composed of carbon fiber plates and rods with a durable acrylonitrile butadiene styrene (ABS) landing gear



Figure 2.7: Close view of the coaxial helicopter.

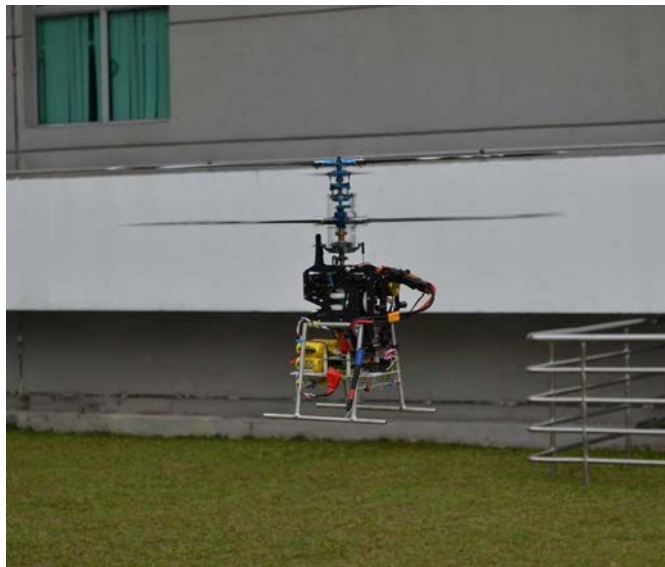


Figure 2.8: Coaxial helicopter flying in the air.

to reduce the bare platform weight. The overall dimensions are 35 cm in height and 86 cm from tip to tip. Different configurations of the rotor blade and the motor are compared before an optimal design is achieved. The motors used for the platform are 740 KV T-Motors with Turnigy Plush - 25 A Bulletproof ESCs. The propellers are APC 12×3.8 clockwise and anti-clockwise fixed pitch propellers. Each motor and propeller setup could generate 15 kN static thrust. The final bare platform's main body weighs 1 kg. Its maximum total take-off weight reaches 3.3 kg with a 4 cell 4300 mAh lithium polymer battery. We have tested that the platform was able to fly at 8 m/s for a period of 10 to 15 minutes depending on the onboard payload weight and the battery volume.

The platform is also fully customizable in terms of sensor arrangement and is scalable such that additional computational boards could be mounted with a stack-based

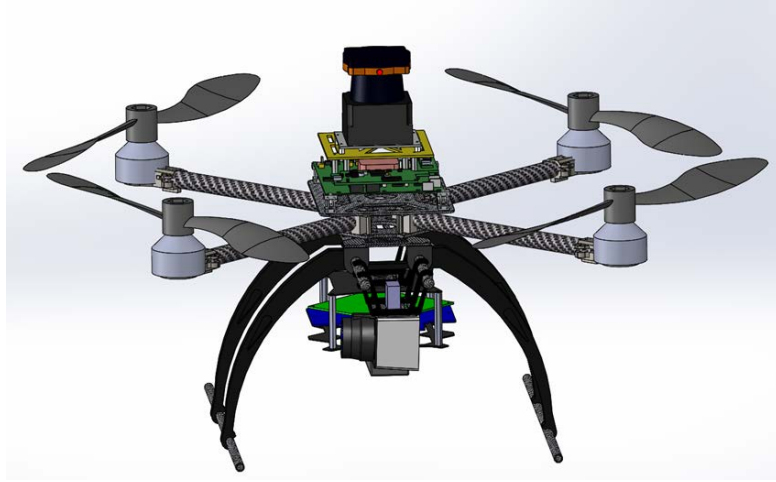


Figure 2.9: NUS quadrotor virtual design.



Figure 2.10: NUS quadrotor platform with two onboard laser range finders.

design. As shown in Fig. 2.10, the platform is equipped with two scanning laser range finders and other avionic systems. The above one is for detecting the environment in the horizontal plane, and the bottom one is for scanning the vertical plane to measure the height of the UAV. A front-facing camera is mounted for surveillance purpose. One noteworthy thing is that the whole avionics system is mounted on the platform through four mechanical isolators (CR1-100 from ENIDINE). Experiment results show that the noise of acceleration measurements in  $x$ ,  $y$ ,  $z$  axis of the IMU decreases by 5 times compared with that without any vibration isolation. The reduced noise of the acceleration improves the accuracy of future state estimation. The vibration isolation also benefits the laser range finder which can only withstand 20 g shock impact for 10 times.

## 2.3 Avionics System Design

### 2.3.1 UAV Function Blocks

A fully autonomous UAV should be able to accomplish the assigned missions without any intervention of a human operator or external system help. This defines that all the tasks in guidance, navigation and control (GNC) have to be carried out autonomously. According to the level of autonomy defined in [37], the task elements in guidance have the highest level of autonomy, while the task elements in navigation and control have middle and lowest level of autonomy respectively.

Based on the level of autonomy, there are two approaches to design and develop a functional UAV system: the top-down method and bottom-up method. The top-down method starts with the highest level of autonomy, researching on tasks such as reasoning, mission assignment, etc. This method treats the lower level tasks in navigation and control as a black-box and assumes a simple point-mass model with some dynamic constraints. On the other hand, the bottom-up method starts with the lowest level of autonomy, dealing with the practical UAV platforms first. The usual working principles follow a sequence including construction of UAV platforms, design of the avionics system, modeling and control of the developed UAV, and so on. These two approaches are adopted by different research groups and neither of these approaches has produced fully autonomous UAVs yet.

In this research, we adopt the bottom-up method, dealing with the platform and the avionics system first. Fig. 2.11 lists the major task elements in GNC on the left and identifies the required avionics modules on the right. In different level of autonomy, there are different required avionics modules. First, the tasks in guidance, such as decision making and path planning, usually involve complicated state machines and algorithms. Thus a high-performance computer is required, preferably with high CPU frequency, large RAM space with minimum weight in a small size factor. Second, tasks in navigation include perception and navigation. Perception tasks, such as mapping and obstacle detection, also require high computation power. Furthermore, perception tasks need various sensors, including laser range finders, sonars, stereo vision, etc. Tasks in navigation require GPS, IMU and a mid-performance computer. Third, flight control tasks require embedded autopilot, servo control board together with sensors used in the

navigation tasks like GPS and IMU. The details of the related avionics modules are presented in section 2.3.2.

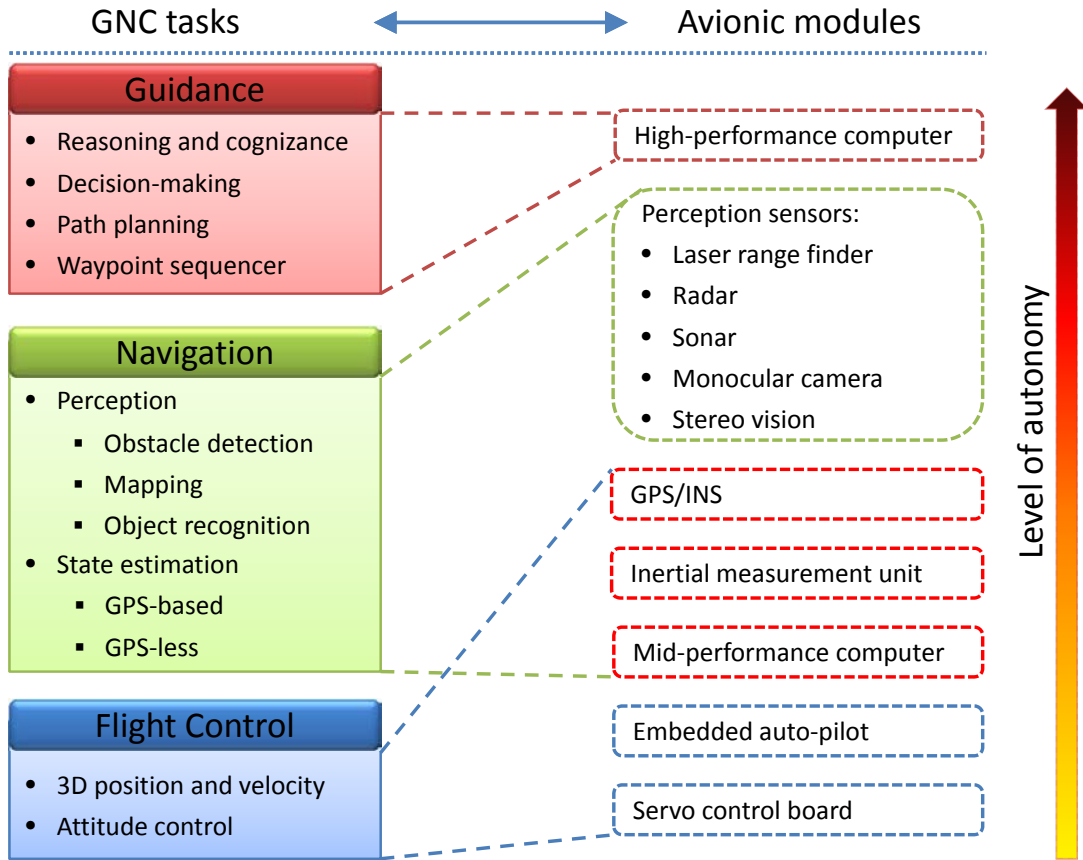


Figure 2.11: UAV functional blocks with the corresponding required avionic modules.

### 2.3.2 Avionics System Components

Fig. 2.11 has identified the required avionics components for a functional UAV system. They are mainly categorized into three groups: the perception group, the processing group and the implementation group. The perception group includes interoceptive sensors, such as IMUs and magnetometers, and exteroceptive sensors, such as GPS, sonars and laser range finders. The processing group includes onboard computing units of various CPU processor (Intel i7 or ARM A15, etc.) and failsafe-related modules. The implementation group includes other modules related to practical considerations such as motors, servos, power regulators and level shifters. This section presents an overview of the state-of-the-art avionics modules suitable for UAV applications.

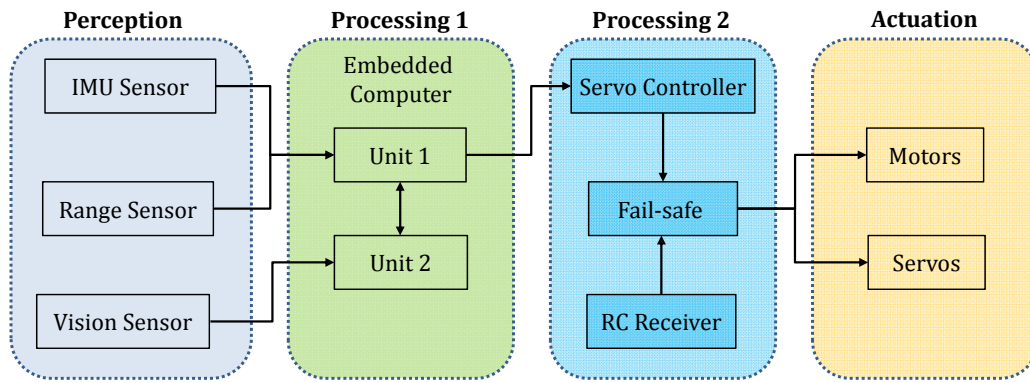


Figure 2.12: UAV avionics system diagram.

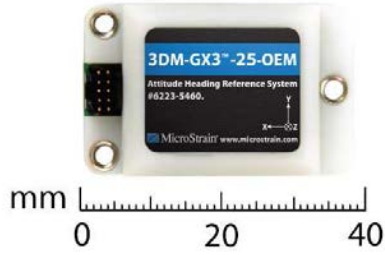
### Inertial Measurement Units

An IMU is the key sensor to detect the linear acceleration and angular rate of the platform, providing the essential measurements for future modeling and control of UAVs. Due to the development of MEMS technology, the state-of-the-art IMUs usually incorporate accelerators and gyroscopes to measure the 3-axis accelerations and the 3-axis angular rates. Besides the raw sensor outputs, modern IMUs often include attitude estimation algorithms to output the 3-axis attitude (roll, pitch, yaw) of the platform. Fig.2.13 shows four state-of-the-art IMUs from different companies. Table 2.2 compares the specifications of the four IMUs in terms of measurement range, update rate, weight, and so on. All of them are of small size and light weight, making them attractive for real-time applications for small-scale UAVs.

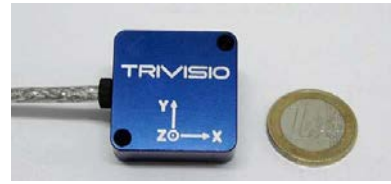
### Range Sensors

Range sensors are devices that capture the relative position of the surrounding environment with respect to the UAV body. The range information could be measured at a single point, across a scanning range or a full depth image at every point. Different types of range sensors utilize various types of waves, including infrared wave, ultrasonic wave and laser (light) wave, etc. An object is said to be detectable with respect to a particular kind of wave means the object surface can reflect that kind of wave effectively. The distance from the sensor to the interested object can be calculated by multiplying the wave speed and the return time (from emitting to reflecting to receiving) of the wave and divided by two. Table 2.3 lists the typical specification of range sensors used for mobile robots and Fig. 2.14 shows four representative sensors of each type.





(a) 3DM-GX3 from MicroStrain



(b) Colibri from Trivisio



(c) IG-500N from SBG



(d) ArduIMU from DIY Drones

Figure 2.13: State-of-the-art IMUs suitable for UAV applications.

Table 2.2: Overview of the specifications of popular IMUs.

Specification / Model	3DM-GX3	Colibri	IG-500N	ArduIMU
Accelerometer range (g)	5	16	5	3
Gyroscope range (deg/s)	300	1500	300	300
Static accuracy (deg)	0.5	0.5	0.5	N.A
Dynamic accuracy (deg)	2.0	2.0	1.0	N.A
Update rate (Hz)	1000	100	100	50
Interface options	USB 2.0 TTL	USB	RS232 TTL	TTL
Supply voltage (V)	3.1 5.5	5	3.3 30	5
Power consumption (mW)	400	200	800	200
Weight (g)	11.5	22	48	6
Size (mm)	40 × 20 × 9	30 × 30 × 13	49 × 36 × 25	39 × 29 × 3

The most accurate range sensor is the scanning laser range finder (Fig.2.14(d)). Laser beams are well focused and reliable. When a non-maximum range value is detected, it is certain there is an object at the specified point. The working principle of a laser range finder often operates on the time of flight principle. A laser pulse in a narrow beam is first sent towards the object. The beam is reflected by some targets and returned to the

Table 2.3: Typical specifications of range sensors.

Type	Ultrasonic Sensor	Infrared Sensor	LiDAR	Radar
Wave type	20 - 50 KHz	700 - 1400 nm	600 - 1000 nm	2.7 - 4.0 mm
Range (m)	15	0.1 - 0.8	< 250	< 250
Power (W)	< 1	< 0.2	4 - 36	< 10
Weight (kg)	< 0.8	< 0.01	0.16 - 4.5	< 1



(a) GP2D12 IR Sensor from Sharp



(b) Roke miniature radar altimeter



(c) Ultrasonic sensor from MaxBotix



(d) Hokuyo UTM-30LX Laser Scanner

Figure 2.14: List of range sensors.

sender. It's by measuring the time difference that the distance to the target is derived. If a mirror reflects the laser beam and rotates in a certain frequency, it becomes a scanning laser range finder. The Hokuyo UTM-30LX shown in Fig.2.14(d) is the state-of-the-art scanning range finder, which is widely used in UAV platforms.

There are also 3D laser scanners available in the market. Fig. 2.15(a) is a new product just announced in Sept. 2014 from Velodyne<sup>2</sup>. It has low power consumption (< 10 W), light weight (about 600 grams), compact footprint (100 mm × 65 mm), and dual return

<sup>2</sup><http://velodynelidar.com/lidar/lidar.aspx>

option. Due to these promising specifications, it is believed to be a revolutionary laser scanner which will be used in UAV applications extensively in future. Fig. 2.15(b) shows another 3D laser scanner which scans four planes simultaneously with a weight of 1 kg, which is normally equipped in ground vehicles.



(a) Velodyne Puck



(b) SICK LD-MRS

Figure 2.15: Two 3D laser scanners from Velodyne and SICK.

## Vision Sensors

Compared with the aforementioned active sensors in Fig. 2.14, the non-active range sensing technologies have gained popularity, especially the vision sensing technologies. Vision sensing technologies possess a series of advantages: they can provide rich information of objects of interest and the surrounding environments, such as color, structure of scene and shape of objects; they require natural light only and do not depend on any other signal source, such as beacon stations or satellite signals; they are generally low-cost and light-weight compared to other sensing systems such as radars.

Small and light cameras are becoming essential components for miniature and micro UAVs. Images captured by the onboard camera are either processed online or transmitted to the ground station where they are processed with the powerful ground stations. Some cameras are equipped with wireless communication function (see Fig. 2.16(a)) which could send the image sequence to the ground station for further vision processing. The computed results are then sent back to the onboard avionics system for control purposes. This approach is broadly used because vision processing algorithms usually require intensive computation that normal embedded computers cannot handle. The other approach is to process images with onboard embedded computers. This approach

requires powerful embedded computers as well as efficient vision processing algorithms. Fig. 2.16(b) shows a camera which communicates with the embedded computers directly. On the other hand, an omni-directional camera can also be used to capture all vision information around the air vehicle with  $360^\circ$  field of view (Fig. 2.16(c)).

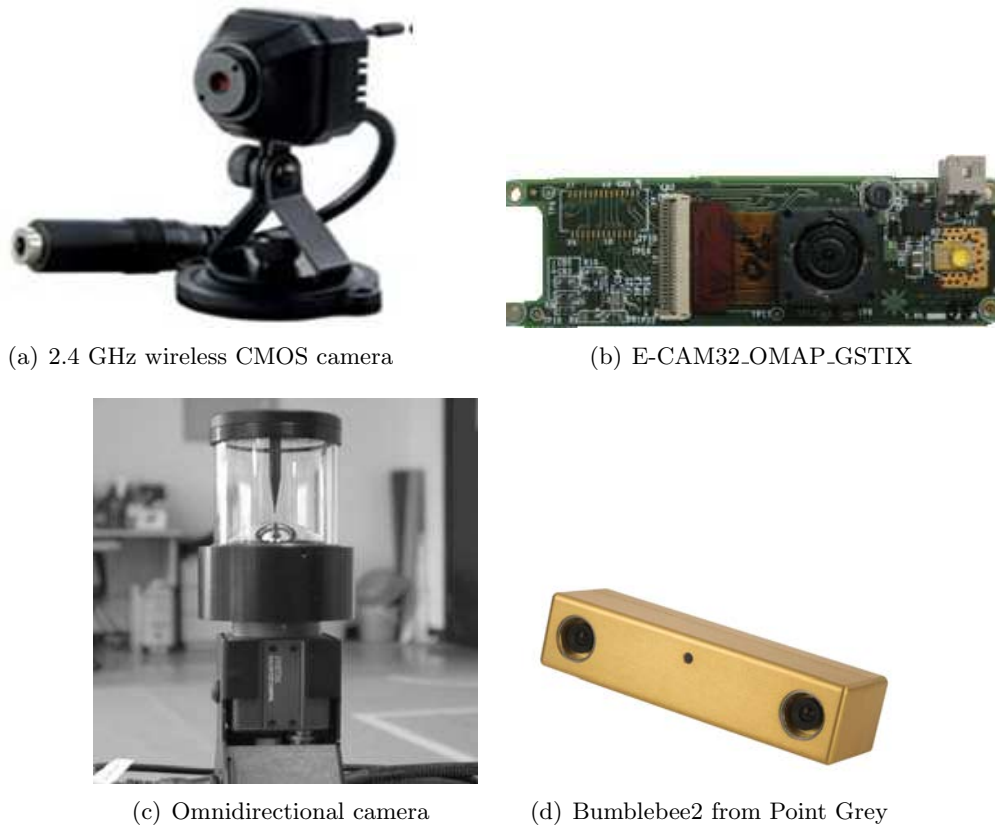


Figure 2.16: List of vision sensors.

The most straightforward approach to generate 3D depth map is the stereo vision technology. By projecting the same point in a scene to two inter-calibrated cameras and finding the disparity of the matching projected points in the two images, the depth information could be extracted using simple calculation. Even though building a stereo head with two calibrated camera is a trivial task itself, processing the two images and getting accurate 3D depth information require intensive computation and tremendous effort in the parameter tuning in different environment. Fortunately, there are available stereo vision systems in the market, such as Bumblebee2 produced by Point Grey<sup>3</sup> shown in Fig. 2.16(d). There is another customized stereo camera system equipped with an IMU for SLAM applications (Fig.2.17). It is developed by researchers in ETH [57] to provide FPGA-preprocessed data, such as visual keypoints, high-quality rate gyroscope

<sup>3</sup><http://ww2.ptgrey.com/stereo-vision/bumblebee-2>

and accelerometer measurements, and hardware-synchronized calibrated images. This device is still under development<sup>4</sup> and it is believed to be an ideal sensor suite for UAVs.

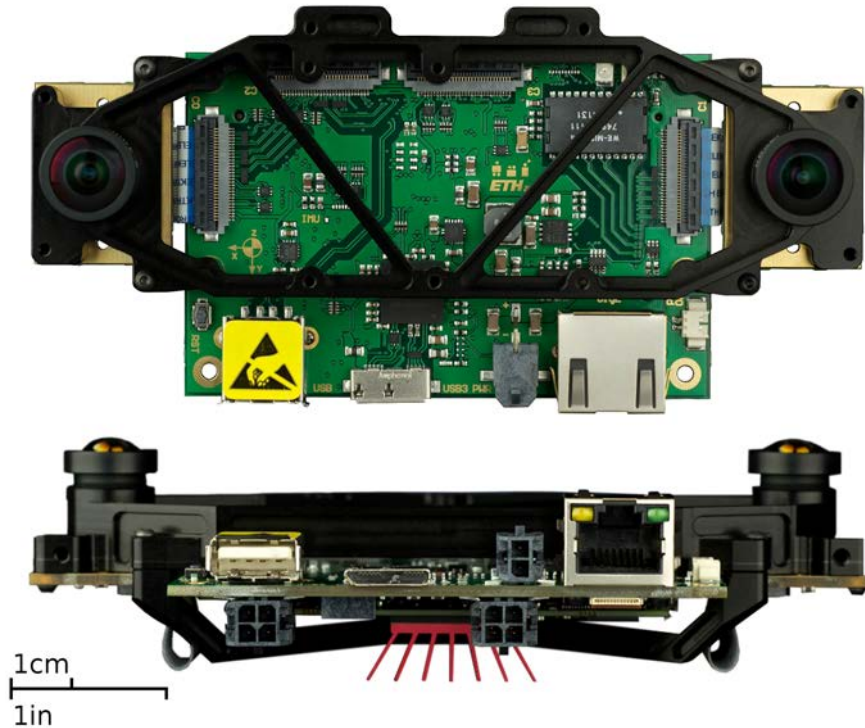


Figure 2.17: The SLAM sensor suite developed by ETH [57]. The suite provides visual keypoints provided by onboard FPGA together with rate gyroscope and acceleration measurements.

## Embedded Computers

Due to the size and weight constraints of small-scale UAVs, the onboard embedded computers have to be light and small. The embedded computers are responsible for all the computation tasks: taking in information from the sensors (IMU sensor, range sensor and vision sensor), applying data fusion, executing control laws, outputting control signals to the servo controller, online data logging, and communication with the GCS.

In this research, we adopt a dual-computer configuration: one for the intensive high-level tasks and the other one for the low-level tasks. For the high-level computer, Mastermind (Fig. 2.18) from Ascending Technologies is used. It features Intel i7 processor with 4 GB RAM with a weight of 325 g. It offers a wide range of interfaces, such as FireWire, USB 2.0 & 3.0, which could be used to connect different peripheral devices including cameras, laser range finders, and so on. The high-level computer is mainly for tasks requiring intensive computation, such as path planning, vision processing.

<sup>4</sup><http://www.skybotix.com/>

The low-level tasks use a Gumstix Overo Fire computer-on-module as shown in Fig. 2.19. It incorporates Texas Instruments OMAP3530 processor with 720 MHz speed in a very compact size (58 mm × 17 mm × 4.2 mm). The Gumstix module includes 802.11b/g WiFi which can be used for online debugging and communication to the ground control station. The low-level tasks include control law implementation, trajectory generation, sensor fusion, etc.

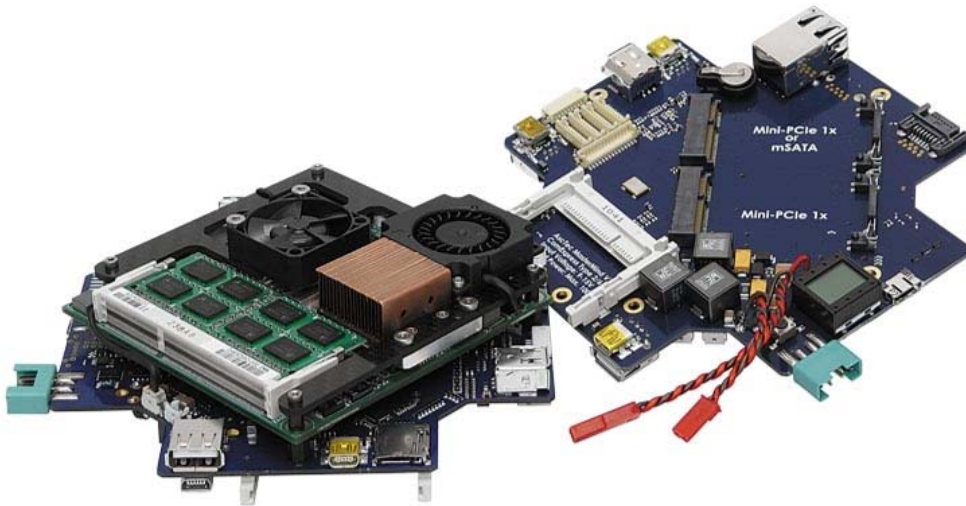


Figure 2.18: High performance onboard computer Mastermind.

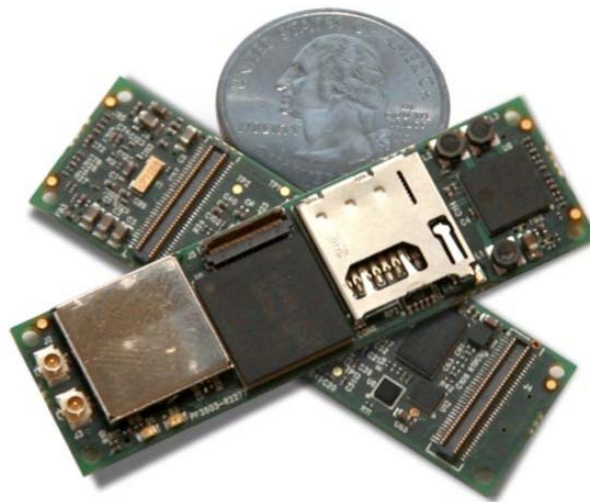


Figure 2.19: Gumstix Overo Fire computer-on-module.

## Servo Controllers

The autonomous control signals from the embedded computer do not directly control the motors and servos. The outputs of the embedded computer need to be transferred

to a servo controller via TTL serial signal and converted to multiple channels pulse-width modulation signals (PWM). Besides the signal conversion, the servo controller also provides a fail-safe function. The servo controller takes in the outputs of the remote controller and the autonomous controller at the same time. A switch signal from the remote controller controls whether the servo controller outputs autonomous control signal or manual control signal. This fail-safe function makes sure human intervention is instantly triggered in case of emergency or program malfunction. The manual control functionality is also necessary for system model identification, in which a sinusoidal manual input is required to stimulate the UAVs in roll, pitch, yaw and heave directions.

There are two versions of servo controller used in the UAV platforms. Fig. 2.20(a) and 2.20(b) show a two-board configuration controller where the signal conversion and multiplexing are implemented on two individual boards. These two boards are commercial off-the-shelf products costing less than \$20 each. Fig. 2.21 shows another type of servo control board developed by Pontech<sup>5</sup>, which is a customized board, integrating the signal conversion and multitasking into a single PCB. Both versions of the servo controller have been used in our UAV platforms.

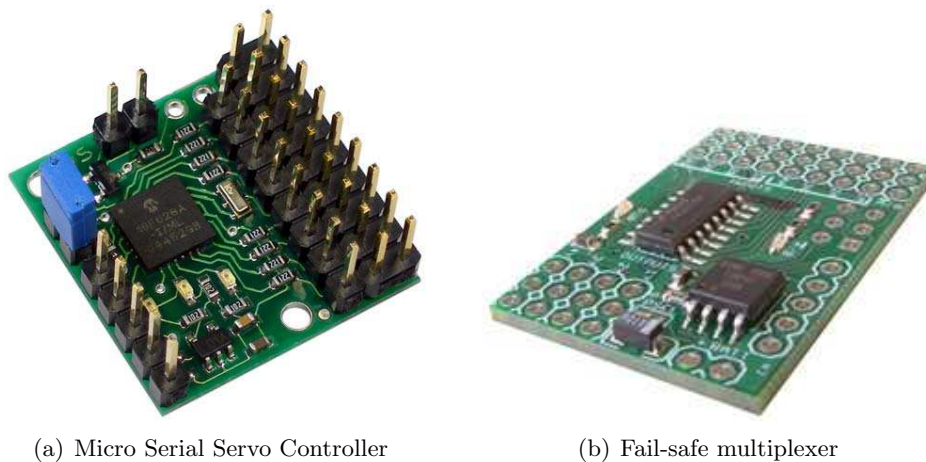


Figure 2.20: Two-board configuration of servo control board.

### 2.3.3 Avionics System Integration

To integrate the modules presented above is not a trial process. A typical avionics system is shown in Fig. 2.22, including sensors such as an IMU, a magnetometer, a GPS, a scanning laser range finder, a camera and two central processing units. Every arrowed

<sup>5</sup><http://www.pontech.com/details/138>



Figure 2.21: One-board configuration of servo control board: UAV100.

line in Fig. 2.22 represents one or more physical wires connecting the two components. Serial communication is established between the two gumstix units. One gumstix is for the autonomous control of the helicopter while the other one is for processing image sequences captured by the camera. Serial communication requires three wires: Rx, Tx and GND. The control gumstix also reads the outputs of onboard sensors, in which the IMU need four wires for serial communication and the laser range finder requires four wires for USB communication. The control gumstix also generates autonomous control signals which are passed to the servo controller using serial port, requiring three more wires. Manual control signals from a human pilot are transmitted to the receiver via a 2.4 GHz radio and fed into the multiplexer using four servo cables. Connecting all these components using jump wires is very messy and the connected system is prone to failure due to the vibrations in flight.

To simplify the assembly process of these components, a type of motherboard called LionHub is designed to connect them and provide essential powers, reducing the messy wires among these components. LionHub connects the main processor, IMU, servo controller and provides ports for power supply, serial communication and servo output. Various assembly holes are designed to mount the above modules. The LionHub also improves the robustness of the avionics system against mechanical vibration during flight. With the introduction of LionHub, messy jump wires are minimized to improve the reliability of the system.

Design of the avionics system has to cater to different requirements. Three versions of LionHubs have been developed to facilitate the research work both in this report and



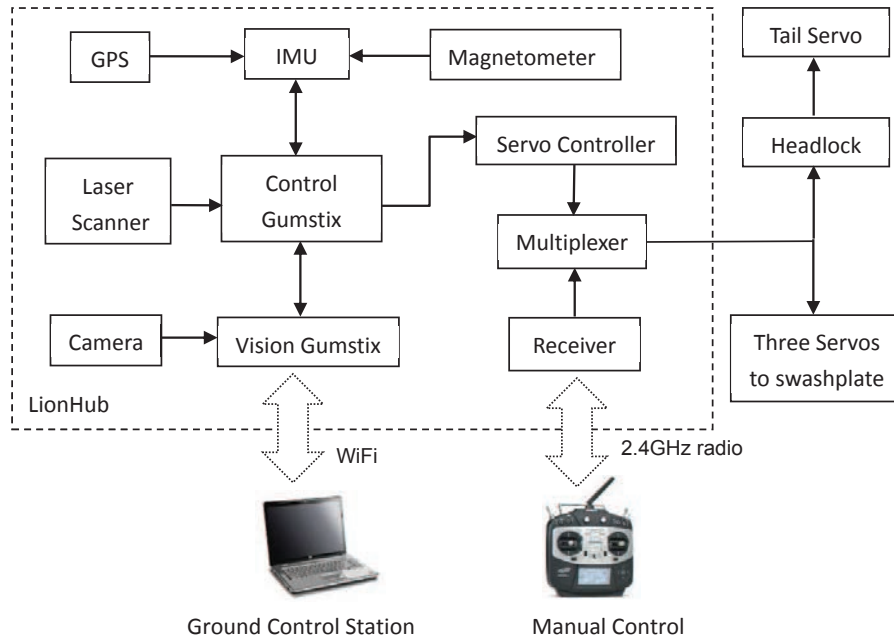


Figure 2.22: A typical avionics system configuration for coaxial helicopter.

other academic activities in NUS UAV group. Table. 2.4 summarizes the key components of all the three LionHubs. LionHub V1 (Fig. 2.23) is a low-cost avionics system featuring ArduIMU as the main IMU. It is targeted for applications requiring low accuracy measurement and compact size. It has been used in early projects such as the indoor coaxial helicopters developed in [79]. LionHub V2 (Fig. 2.24(a)) is a high performance avionics system featuring IG-500N as the IMU and other abundant interface options, such as USB and serial ports. It has been used in both quadrotors and single rotor helicopters. Fig. 2.24(b) presents one of its application in an Align T-Rex 90 scale helicopter with other avionics components such as a scanning laser range finder and vision computers. This helicopter took part in the 2nd AVIC Cup International UAV Grand Prix<sup>6</sup>, Beijing, in September of 2013 and won the 1st place in the final round. Fig. 2.25(a) shows LionHub V3 with similar functions but has a much smaller footprint. This is specifically designed for IMAV2014<sup>7</sup> held in Delft, the Netherlands, 2014. In this competition, five quadrotors are equipped with LionHub V3 with different peripheral device configurations. All the five quadrotors have flied autonomously in the competition and helped our team to win the 1st place in IMAV2014. In conclusion, the developed LionHubs in this study have been proven to be robust and user-friendly, laying the foundation for high-level algorithm design and implementation.

<sup>6</sup><http://uav.ece.nus.edu.sg/uavgp.html>

<sup>7</sup><http://www.imavs.org/2014/>

Table 2.4: Summary of three LionHubs.

	LionHub V1	LionHub V2	LionHub V3
Processor	Gumstix Overo Fire	Gumstix Overo Fire	Gumstix Overo Fire
Extension	Gumstix Summit	Gumstix Pino-TH	Gumstix Pinto-TH
Power input	4.5-6 V	13-18 V	13-18 V
Onboard power	5V	12 V, 5 V	12 V, 5 V
IMU	ArduIMU	IG-500N	Pixhawk <sup>8</sup>
Interface option	N.A	Serial TTL, USB RS-232	Serial TTL, USB RS-232
Size	60 mm × 100 mm	110 mm × 140 mm	90 mm × 90 mm

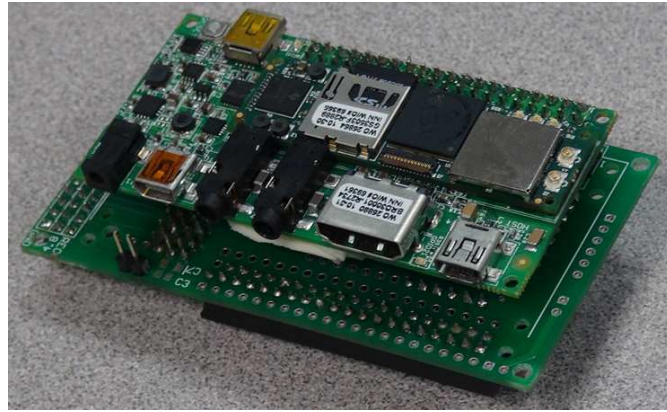
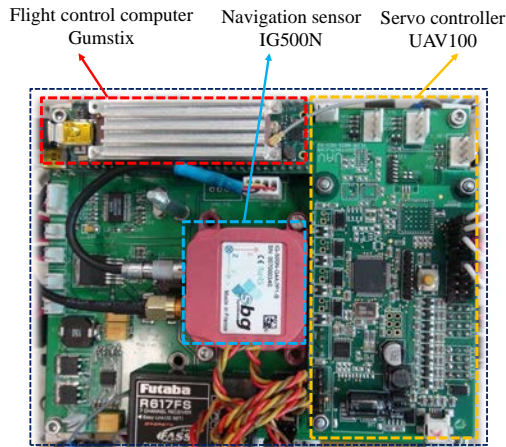
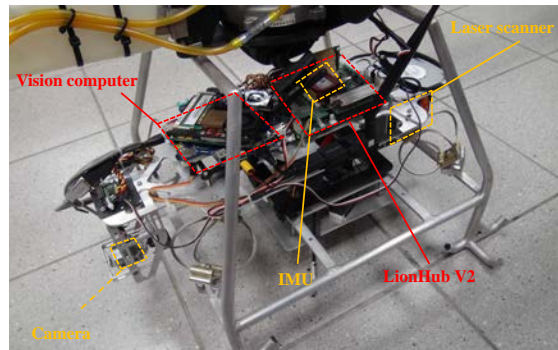


Figure 2.23: LionHub V1: first design featuring low cost and compact volume.



(a) LionHub V2



(b) LionHub V2 on helicopter for UAVGP 2013.

Figure 2.24: LionHub V2 and its application in T-Rex 90.

## 2.4 Conclusion

In this chapter, we have presented the design of UAV platforms, including the bare platform evaluation and the design of onboard avionics system. For the bare platform development, three common VTOL concepts are evaluated: single rotors, coaxial helicopters and quadrotors. A comparison of the three configurations leads to the conclusion



(a) LionHub V3



(b) LionHub V3 on quadrotor for IMAV2014

Figure 2.25: LionHub V3 and its application in quadrotor.

that coaxial helicopters and quadrotors are two promising platforms deserving more investigation. Two UAV platforms are then designed and constructed, including a coaxial helicopter and a quadrotor. The coaxial UAV is based on a COTS RC helicopter – ‘Kaa-350’. Modification of ESC, blade and motor are performed to increase the payload capability. The quadrotor is built from scratch due to its simple mechanical structure.

Then we present the design of UAV onboard avionics system. The requirements for the avionics system according to the GNC tasks of UAV are identified. Various tasks need different avionics modules, ranging from IMUs, range sensors, embedded computers to servo controllers. A review of the state-of-the-art avionics components is performed to analyze their applicability to small scale UAVs. With the selected avionics components, an integrated board ‘LionHub’ is designed to reduce the wiring and to increase the system robustness against vibration during flight. Three versions of boards have been designed to meet different mission requirements, including routine flight tests and international competitions.

## Chapter 3

# Modeling and Control of UAV Platforms

### 3.1 Introduction

For UAV navigation in obstacle-strewn and GPS-denied environments, the UAV platform itself needs to maintain high attitude stability and achieve waypoint tracking capability at the same time. As most modern control techniques are model-based, a precise dynamic model of the UAV needs to be derived first. Although it is always desirable to derive a nonlinear dynamic model to cover all flight conditions, it is practically feasible to obtain linear models at different operation points. In cluttered environments, the controlled UAV usually operates in a near-hover condition and avoids a high flight speed due to an extremely short time-to-collision. A single linearized model at this particular flight condition is thus sufficient. To derive a linear model of a UAV platform, the first-principle modeling method and the system identification method are adopted in a complementary manner. The first-principle method focuses on the mathematical formulation of the system based on the law of physics, giving a clear structure of the dynamics model, while the system identification method seeks to identify the model parameters in the region of operating point by processing the recorded flight data.

We apply the above two techniques for the modeling of the coaxial helicopter and the quadrotor constructed in the last chapter. Coaxial helicopters have more complex models than quadrotors, since they rely on the flapping of the blades to achieve horizontal movements. The same movements for quadrotors are obtained by adjusting the rotation

speed of the four rotors. Besides, the complex interaction of the upper rotor and the lower rotor in coaxial helicopters has made modeling of them quite challenging. We avoid the study of the aerodynamics of coaxial helicopters and derive a linear model using the two modeling techniques. Starting from the first principle approach, the model structure is analyzed in detail in Section 3.2 and the model parameters of the subsystem dynamics are identified in the frequency domain.

Quadrotor platforms have gained more popularity in research labs [68, 5, 31], which is mainly due to the simple dynamics model and the wide availability of quadrotor autopilot systems. Driven by four motors, the quadrotor can be easily maneuvered by changing the rotation speed of the four rotors. The mechanical symmetry of quadrotor ensures a decoupled dynamics model. The vast availability of attitude controllers for quadrotor also accelerates its development, some of them are even open source in both hardware and software, such as Pixhawk from ETH<sup>1</sup> and Arducopter<sup>2</sup>. Our developed quadrotor uses ‘NAZA-M2’ to stabilize the angular dynamics, thus only the outer-loop model and control need to be investigated. The outer-loop model exhibits typical second order system dynamics which are to be extracted using the frequency domain identification. Once the model is available, the autonomous control law based on robust perfect tracking is designed. Experiment results of the quadrotor following some predefined way points are given in Section 3.3.4. Part of the work about the quadrotor modeling and control has been presented in [80, 19].

## 3.2 Modeling of Coaxial Helicopter

### 3.2.1 Comprehensive Dynamics Model Structure

The coaxial helicopter has two rotors driven by the same electric motor and the rotating speed of the two rotors can not be adjusted during flight. The motion of the helicopter is achieved by changing the pitch angles of the upper rotor and the lower rotor in various combinations. Fig. 3.1 shows the helicopter main body. With the control inputs denoted in Table. 3.1, we can connect the control input to the corresponding helicopter motions. The upper rotor and lower rotor are connected by two swashplates. The two swashplates are always parallel to each other since they are connected by three mechanical linkages.

---

<sup>1</sup><https://pixhawk.ethz.ch/>

<sup>2</sup><http://copter.ardupilot.com/>

The collective input lifts the swashplates high or pulls them down to change the collective pitch angles of the upper and lower rotors, causing the helicopter to move in the heave direction. The yaw motion is achieved by adjusting the collective pitch angle of the lower rotor using another servo. One point to note is that the yaw channel control  $\delta_{ped}$  is first mixed with the output of the headlock gyro before it is applied to the lower rotor. This block serves to stabilize the yaw angular rate for easy manual control. The cyclic inputs  $\delta_{lat}$  and  $\delta_{lon}$  tilt the upper and lower swash-plates and generate flapping motion for both rotors, creating the longitudinal and lateral movements of the helicopter.

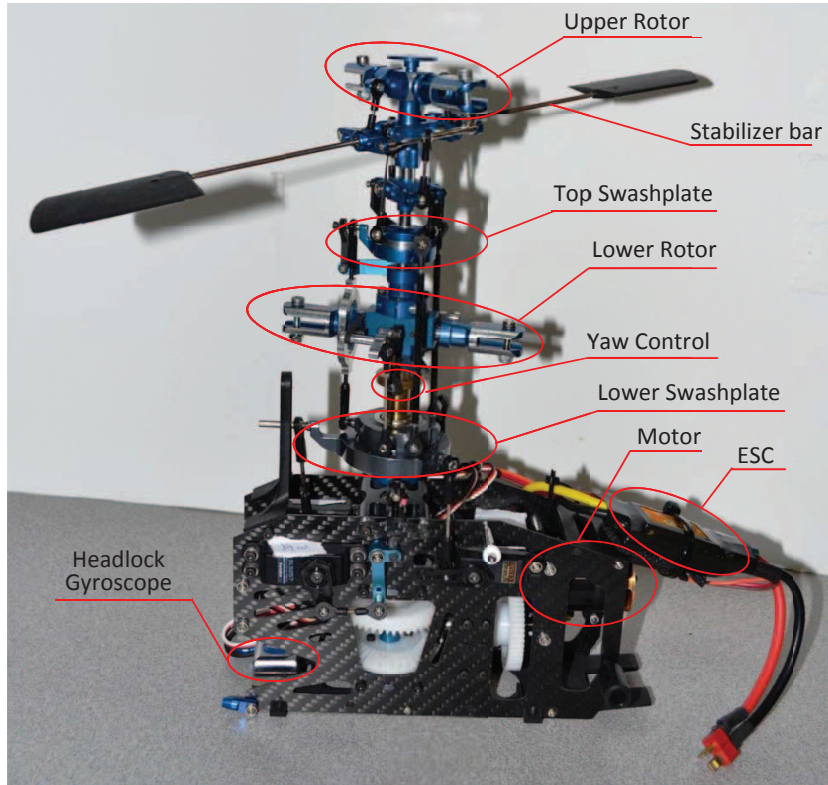


Figure 3.1: Fuselage head of coaxial helicopter with labeled key components.

Table 3.1: Physical meaning of control input variables

Variables	Physical meaning	Range
$\delta_{lat}$	control deflection for lateral cyclic pitch of main blades	$[-1, 1]$
$\delta_{lon}$	control deflection for longitudinal cyclic pitch of main blades	$[-1, 1]$
$\delta_{col}$	control deflection for collective pitch of upper and lower blades	$[-1, 1]$
$\delta_{ped}$	control deflection for collective pitch of the lower blades	$[-1, 1]$
$\bar{\delta}_{ped}$	control deflection for yaw-stability-augmentation controller	$[-1, 1]$

The nonlinear model of the coaxial helicopter is expressed in the following compact form,

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}), \quad (3.1)$$

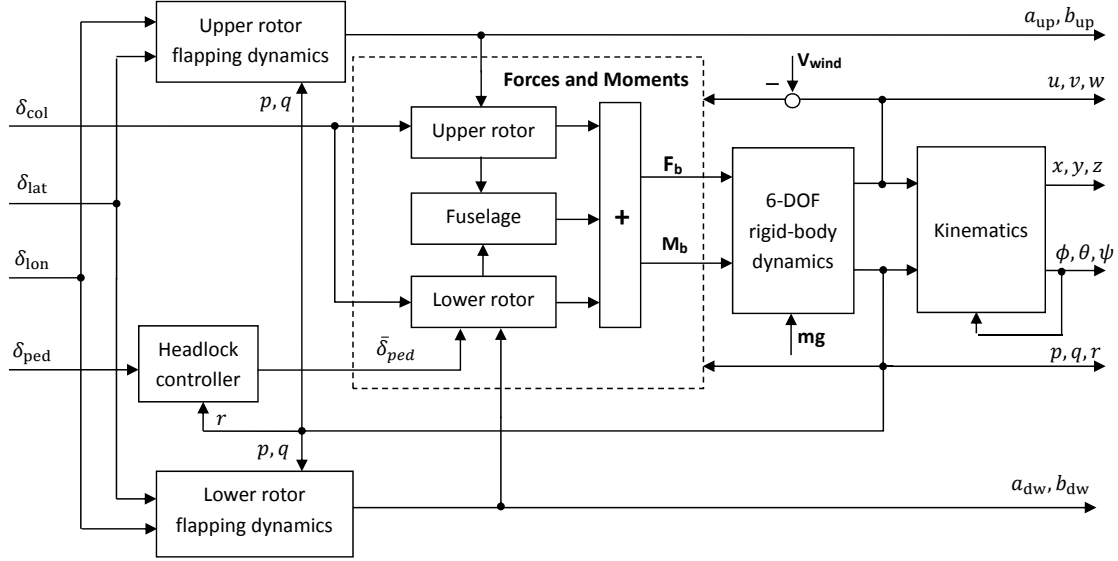


Figure 3.2: Model structure of coaxial helicopter.

where  $\mathbf{x}$  is the state vector,  $\mathbf{u}$  is the input vector, and  $\mathbf{w}$  is the wind velocity,

$$\begin{aligned}\mathbf{x} &= (x \ y \ z \ u \ v \ w \ \phi \ \theta \ \psi \ p \ q \ r \ a_{up} \ b_{up} \ a_{dw} \ b_{dw} \ r_f)^T, \\ \mathbf{u} &= (\delta_{lat} \ \delta_{lon} \ \delta_{col} \ \delta_{ped})^T, \\ \mathbf{w} &= (w_u \ w_v \ w_w)^T.\end{aligned}$$

The physical meanings of the state are listed in Table 3.2. There are two coordinate frames in Table 3.2 including the north-east-down (NED) frame and the body frame. They are defined as Fig. 3.3 for the sake of navigation expression. The NED frame is stationary with respect to a static observer on the ground and the body frame is fixed at the center of gravity of the helicopter.

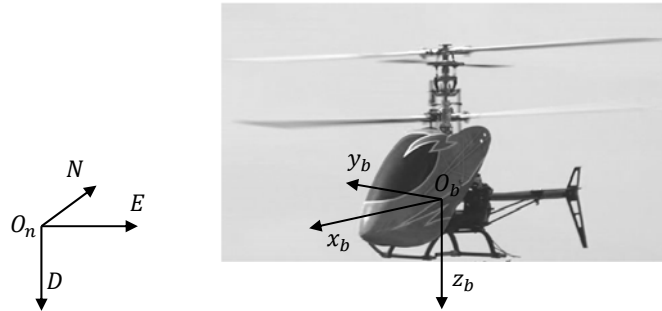


Figure 3.3: Definition of NED frame  $O_n$  and body frame  $O_b$ .

To build a mathematic model of the system in Fig. 3.2, all the subsystems in the chain from the left side input to the right side output have to be analyzed. The 6 degree of freedom (DOF) rigid body dynamics and the kinematics are general principles

Table 3.2: Physical meaning of state variables

Symbol	Physical Meaning	Unit
$x$		
$y$	$\mathbf{p}_n$ , position in the local NED frame	m
$z$		
$u$		
$v$	$\mathbf{v}_b$ , linear velocity in the body frame	m/s
$w$		
$\phi$		
$\theta$	roll pitch yaw } attitude angle	rad
$\psi$		
$p$		
$q$	$\omega_b$ , angular velocity in body frame	rad/s
$r$		
$a_{\text{up}}$		
$b_{\text{up}}$	longitudinal lateral } flapping angle of upper blades	rad
$a_{\text{dw}}$		
$b_{\text{dw}}$	longitudinal lateral } flapping angle of lower blades	rad
$r_f$		

which govern all rigid body motions. It is the mechanisms producing the force  $\mathbf{F}_b$  and the moment  $\mathbf{M}_b$  that make modeling coaxial helicopter special. In the following sections, the rigid body dynamics and kinematics are first covered. We then analyze each subsystem in detail and identify the parameters.

### Rigid Body Dynamics and Kinematics

The kinematics model transforms the translational and rotational motions from the body coordinate to the local NED coordinate. The translational motion transformation between body frame and local NED frame is,

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \underbrace{\begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}}_{\mathbf{R}_{n/b}} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad (3.2)$$

where  $\mathbf{R}_{n/b}$  represents the transformation matrix from the body frame to the local NED frame and  $s_* = \sin(*)$ ,  $c_* = \cos(*)$ . The transformation of rotational motion from body frame to the local NED frame is given by:



$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \underbrace{\begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi/c_\theta & c_\phi/c_\theta \end{bmatrix}}_{\mathbf{S}^{-1}} \begin{pmatrix} p \\ q \\ r \end{pmatrix}, \quad (3.3)$$

where  $t_* = \tan(*)$ , which does not hold for  $\theta = \pm 90^\circ$ . Eq. 3.3 suffices when the helicopter mainly operates in near-hover conditions, otherwise a quaternion representation is recommended.

The body frame translation and rotation can be formulated using the Newton-Euler equations, which describe the relations between the forces and moments on the rigid body and its induced translation and angular velocity.

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \frac{1}{m} \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} - \begin{pmatrix} p \\ q \\ r \end{pmatrix} \times \begin{pmatrix} u \\ v \\ w \end{pmatrix}, \quad (3.4)$$

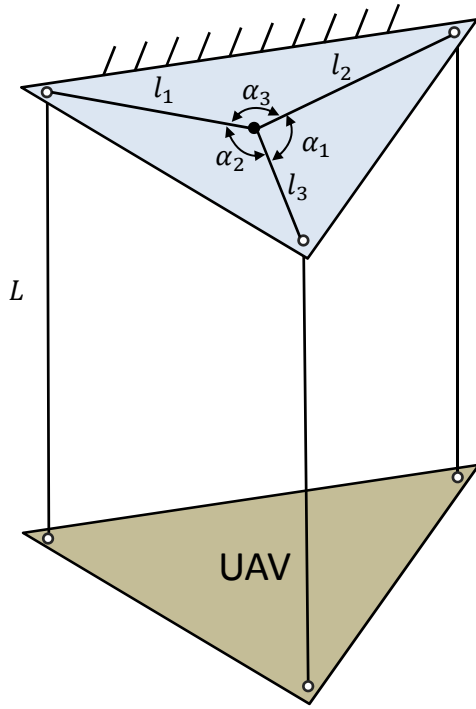
$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \mathbf{J}^{-1} \left\{ \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} - \begin{pmatrix} p \\ q \\ r \end{pmatrix} \times \mathbf{J} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \right\}, \quad (3.5)$$

where  $F_x, F_y, F_z$  are projections of the net force,  $\mathbf{F}_b$ , onto the body-frame  $x$ -,  $y$ -,  $z$ -axis, and  $M_x, M_y, M_z$  are projections of the net torque,  $\mathbf{M}_b$ , onto the body-frame  $x$ -,  $y$ -,  $z$ -axis and ‘ $\times$ ’ denotes cross product of vector.  $m$  is the helicopter mass and  $\mathbf{J}$  is the tensor of inertia matrix defined as,

$$\mathbf{J} = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix}, \quad (3.6)$$

where the off-diagonal elements  $J_{xy}, J_{xz}, J_{yz}$  are negligible for small-scale helicopters.

From Eq. 3.2 to Eq. 3.5 the only two parameters need to identify are the helicopter mass  $m$  and the platform moment of inertia  $\mathbf{J}$ . The mass of the platform can be measured by a weighing scale. The moment of inertia can be measured by the trifilar pendulum method [13]. As explained in Fig. 3.4, the UAV is suspended by three flexible lines



(a) Trifilar pendulum method.



(b) Measurement setup of trifilar pendulum.

Figure 3.4: Testing of moment of inertia using trifilar pendulum.

with equal length  $l$ . The distances between the attached points and CG are  $l_1$ ,  $l_2$ , and  $l_3$ , respectively. The UAV is perturbed along the line direction and oscillates around the body-frame axis ( $z$  axis in Fig.3.4(b)). The oscillation period  $t_1$  is recorded. The moment of inertia along this axis is given by,

$$J_{zz} = \frac{m g l_1 l_2 l_3 t_1^2}{4 \pi^2 L} \cdot \frac{l_1 \sin \alpha_1 + l_2 \sin \alpha_2 + l_3 \sin \alpha_3}{l_2 l_3 \sin \alpha_1 + l_1 l_3 \sin \alpha_2 + l_1 l_2 \sin \alpha_3}. \quad (3.7)$$

The same procedure can be applied to  $x$  and  $y$  axes.

### Compositions of Forces and Moments

The rigid-body dynamics listed in Eq.3.4 - 3.5 has built the connection between the forces and motions with the body frame translational and rotational velocity. The next task is to identify the composition of the forces and moments so that the complete model is derived. The two coaxial rotors provide the main lift for the helicopter and the moments are also induced by tilting the rotating disk. Other effects such as the gravity and the body resistance relative to the air should be also considered. The forces applied on the

body can be expressed as Eq. 3.8, in which the first term on the right side corresponds to the summed thrust generated by the upper and lower rotor  $\mathbf{T}_i$ , ( $i \in \{\text{up}, \text{dw}\}$ ). The second term is the projection of gravity force  $mg$  on the body frame. The third term is the fuselage force which is mainly caused by the air resistance in the horizontal direction and the downwash effect from the lower rotor in vertical direction.

$$\begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = \sum \mathbf{T}_i + mg \begin{pmatrix} -s\theta \\ s\phi c\theta \\ c\phi c\theta \end{pmatrix} + \begin{pmatrix} X_{\text{fs}} \\ Y_{\text{fs}} \\ Z_{\text{fs}} \end{pmatrix}, \quad (3.8)$$

$$\begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \sum \mathbf{l}_i \times \mathbf{T}_i + \sum K_\beta \begin{pmatrix} b_i \\ a_i \\ 0 \end{pmatrix} + \sum \mathbf{Q}_{\text{d},i}, \quad (3.9)$$

When the upper and lower rotor are tilted by the swashplates, the thrust of vectors  $\mathbf{T}_i$  does not pass through center of gravity, creating torques related in the roll and pitch direction. They are expressed as  $l_{\text{up}} \times \mathbf{T}_{\text{up}}$  and  $l_{\text{dw}} \times \mathbf{T}_{\text{dw}}$ , where  $l_{\text{up}}$  and  $l_{\text{dw}}$  are the distance from the upper rotor and lower rotor to the center of gravity respectively. Flapping of the upper rotor and lower rotor causes torques on the rotor hub, which can be described in the second term of Eq. 3.9, where  $K_\beta$  is the spring constant for both the upper and lower rotor. In addition, the rotation of the upper rotor and the lower rotor both have the drag torque besides the lift forces, which are denoted as  $\mathbf{Q}_{\text{d,up}}$  and  $\mathbf{Q}_{\text{d,dw}}$  respectively.

To build the comprehensive nonlinear model of the coaxial helicopter, the key lies in accurate aerodynamic analysis of the coaxial rotors. The relations between the pitch angle of the blade with the lift and drag force generated on the blade need to be presented using mathematics equations. Details of the aerodynamics analysis of the coaxial helicopter can be found in [43], in which the authors use blade element momentum theory (BEMT) to develop in analytical formulation for propeller analysis. Interested readers can refer to the paper for the details. For our case, the UAV does not need to perform any aggressive maneuvering. At hover condition, the total thrust of the coaxial rotors is approximately the same as the gravity force of the platform  $mg$ .

### 3.2.2 Linear Dynamics Model and Parameter Identification

There are four subsystem dynamics of the coaxial helicopter, including roll, pitch, heave and yaw dynamics. Due to the flapping of the tip-path-plane (TPP), there are strong coupling effects between the roll and pitch dynamics. The two dynamics are usually lumped to the same subsystem to capture the angular responses of helicopter to the cyclic inputs. It constitutes the core of helicopter dynamics [54] as lateral and longitudinal movement are more important for UAV navigation. The heave and the yaw dynamics are independent, which can be treated separately. They are less important in the sense that the UAV flying in the air will maintain its heading and height for most of the time. In the following sections, the three subsystems' dynamics will be presented with the identification of the corresponding parameters.

#### Roll Pitch Dynamics

The coaxial platform consists of two contra-rotating rotors. The top rotor includes a stabilizer bar coupled to the top rotor blade through Bell-Hiller mixer. The lower rotor contains only two blades without stabilizer bar. The upper rotor and the lower rotor receive the same cyclic input  $(\delta_{lon}, \delta_{lat})$  since the top and bottom swash-plates are always parallel via the mechanical linkages. To minimize the overall complexity of the model, the two counter rotating rotor discs are treated as one equivalent rotor disc and their flapping angle are unified as  $a_s$  and  $b_s$ . This assumption is valid with the condition that the helicopter does not perform rapid maneuvering. It also makes it simpler for the modeling of the roll-pitch dynamics, while still maintaining moderate accuracy. According to [13], the flapping dynamics subsystem could be represented in the following state space model:

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{a}_s \\ \dot{b}_s \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 & L_b \\ 0 & 0 & M_a & 0 \\ 0 & -1 & -\frac{1}{\tau} & \frac{A_b}{\tau} \\ -1 & 0 & \frac{B_a}{\tau} & -\frac{1}{\tau} \end{bmatrix} \begin{pmatrix} p \\ q \\ a_s \\ b_s \end{pmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & A'_{lon} \\ B'_{lat} & 0 \end{bmatrix} \begin{pmatrix} \delta_{lat} \\ \delta_{lon} \end{pmatrix}, \quad (3.10)$$

where  $L_b$  and  $M_a$  are obtained by combining Eq.3.5 and Eq.3.9. The total thrust from the coaxial rotor is assumed to be equal to the gravity force  $mg$ .

$$L_b = \frac{mg H_{mr} + K_\beta}{J_{xx}}, \quad M_a = \frac{mg H_{mr} + K_\beta}{J_{yy}}, \quad (3.11)$$

where  $H_{mr}$  is the average of the upper rotor hub distance  $l_{up}$  and the lower rotor hub distance  $l_{dw}$  to the center of gravity. The rotor spring constant  $K_\beta$ , the lateral and longitudinal control derivatives  $B'_{lat}$ ,  $A'_{lon}$ , the lateral and longitudinal control delay  $\tau_{lat}$ ,  $\tau_{lon}$ , and the equivalent flapping time constant  $\tau$  are to be identified via frequency domain identification. The coupling terms  $A_b$  and  $B_a$  are neglected.

The flapping dynamics identification makes full use of a toolkit called CIPHER developed by the U.S. Army and NASA specifically for rotorcraft applications [54]. It incorporates a range of utilities to support the various steps of the identification process. Flight tests featuring frequency-sweep input in the longitudinal and lateral directions are performed multiple times. During the flights, the control inputs and the helicopter angular rates are recorded online with a sampling rate of 50 Hz. CIPHER identifies the model parameters by searching for the best-fit parameters to match frequency responses between the flight test data and the hypothetical model. Fig. 3.5-3.6 shows two on-axis angular rate responses to the cyclic input. The coherence for both on-axis directions remain above 0.6 up to 30 rad/s. This good coherence indicates the good linearity of the helicopter in hover flight [76]. Table. 3.3 lists the value of the identified parameter together with their Cramer-Rao percent and insensitivity. The Cramer-Rao percent and insensitivity are less than 15% and 5% respectively, indicating the high accuracy of the identified parameter. Time-domain verification is also performed with another set of flight test data which is not used in the identification process. Figs. 3.7 - 3.8 show excellent agreement between the model simulation and the flight data in both longitudinal and lateral directions.

### Heave Dynamics

Similar to the Newton-Euler motion equations used for the longitudinal and lateral dynamics, the heave dynamics can be represented as:

$$\dot{w} = (-vp + uq) + \frac{F_z}{m}, \quad (3.12)$$

Table 3.3: Parameters for roll-pitch dynamics.

Parameter	Cramer-Rao Percent(%)	Insensitivity(%)	Physical meaning
$L_b = 675.8 \text{ s}^{-2}$	7.171	2.433	Lateral rotor spring derivative
$M_a = 794.7 \text{ s}^{-2}$	7.525	2.589	Longitudinal rotor spring derivative
$\tau = 0.068 \text{ s}$	9.301	3.537	Equivalent flapping time constant
$A'_{lon} = 0.898 \text{ rad/s}$	4.152	1.962	Longitudinal control derivative
$B'_{lat} = 1.069 \text{ rad/s}$	4.157	1.935	Lateral control derivative
$\tau_{lat} = 0.03355 \text{ s}$	12.08	4.477	Lateral control delay
$\tau_{lon} = 0.03390 \text{ s}$	12.17	4.440	Longitudinal control delay
$K_\beta = 11.5029 \text{ Nm}$	NA	NA	Rotor spring constant

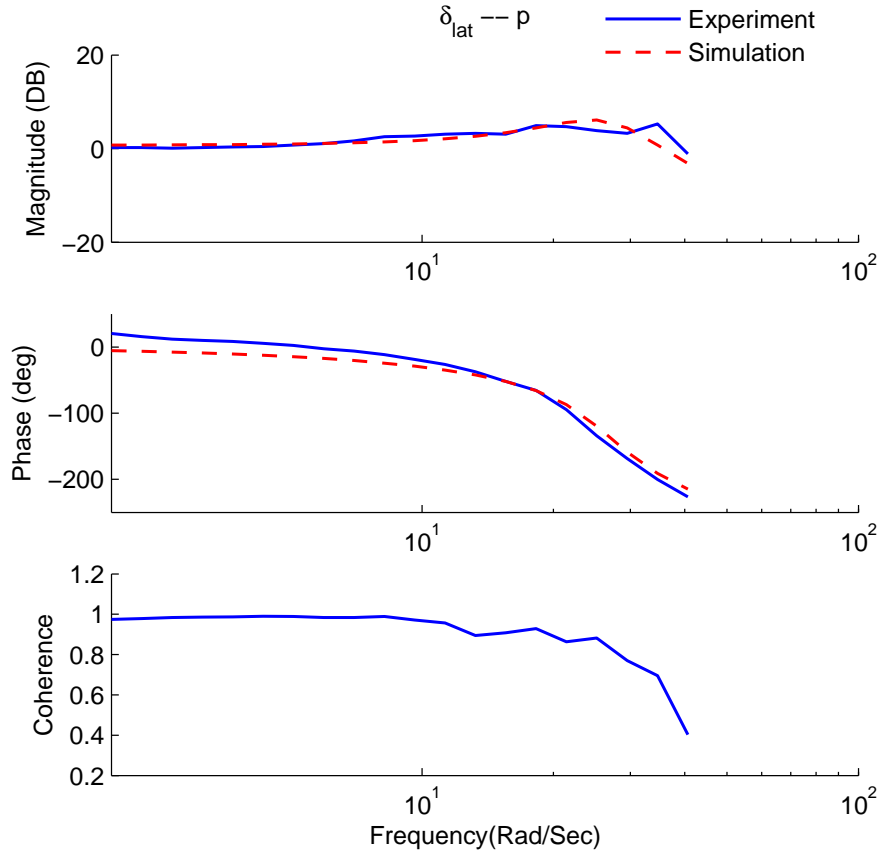


Figure 3.5: Frequency response from roll input to roll angular rate.

where  $u, v, p, q \approx 0$  at hovering condition and  $F_z$  is a combination of thrust force, UAV weight, and air frictional force. The liner model is assumed to be related the collective input  $\delta_{col}$  and the heave velocity  $w$ ,

$$\dot{w} = -\frac{w}{\tau_w} + K_w \delta_{col}, \quad (3.13)$$

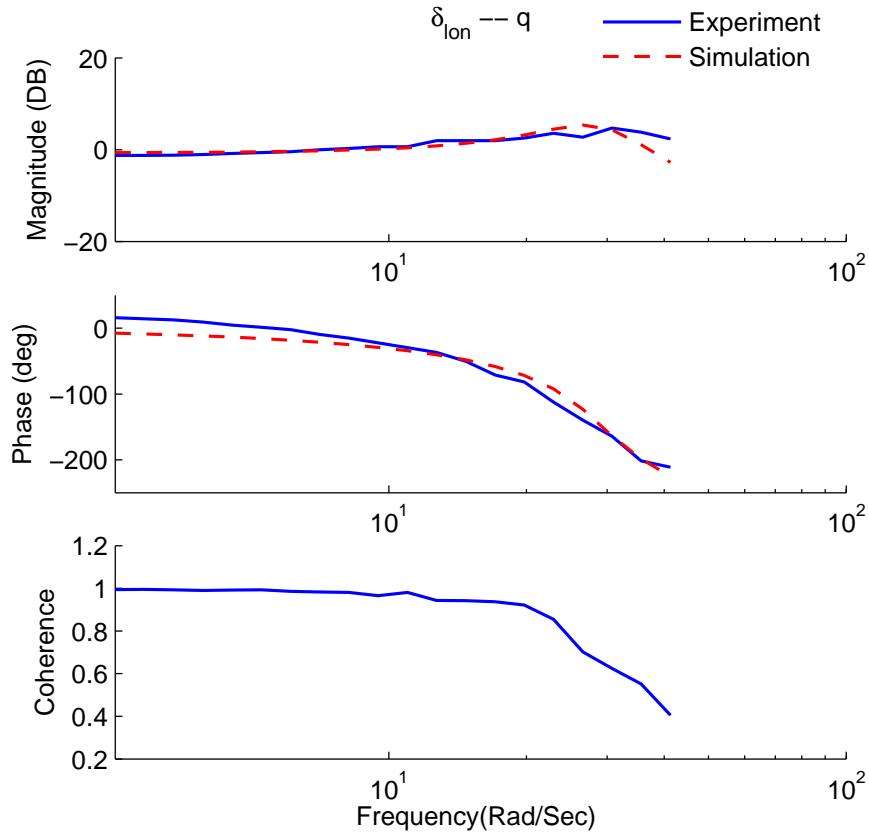


Figure 3.6: Frequency response from pitch input to pitch angular rate.

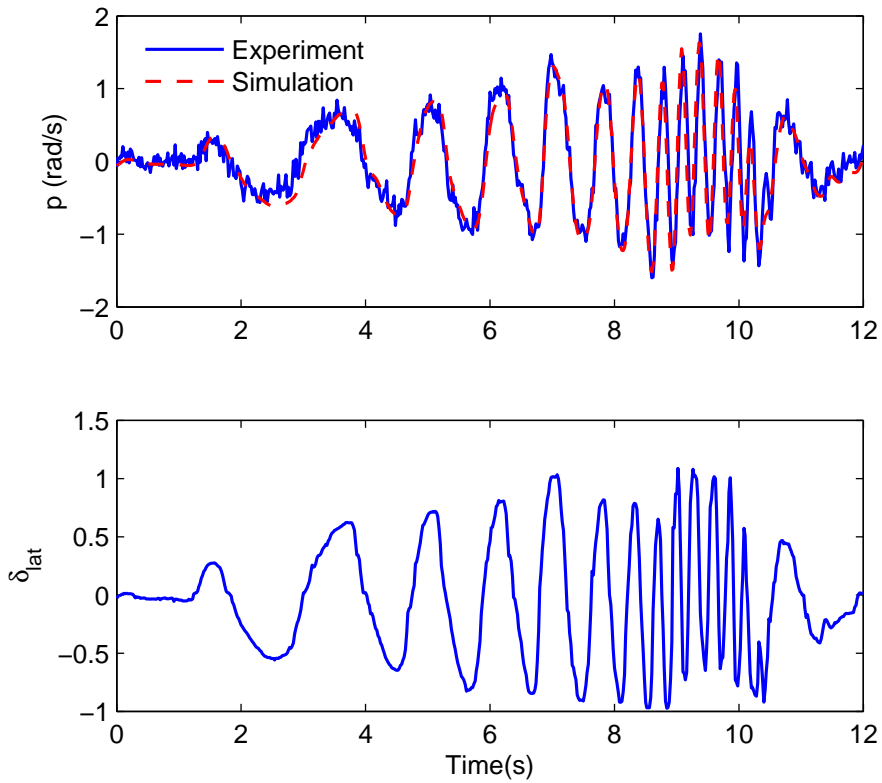


Figure 3.7: Time domain verification from roll input to roll angular rate.

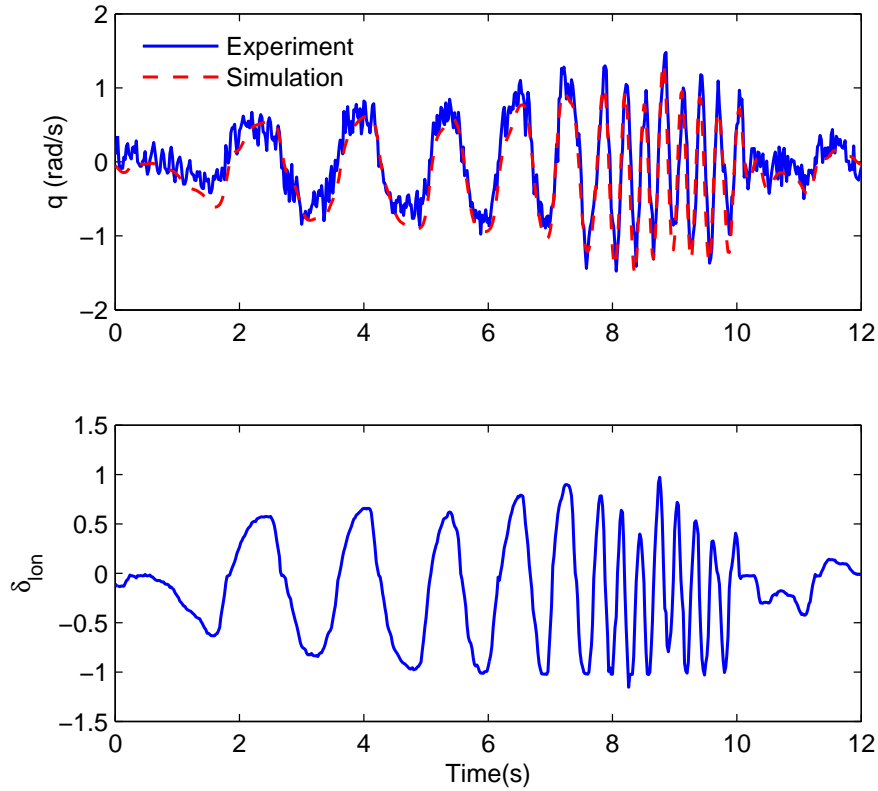


Figure 3.8: Time domain verification from pitch input to pitch angular rate.

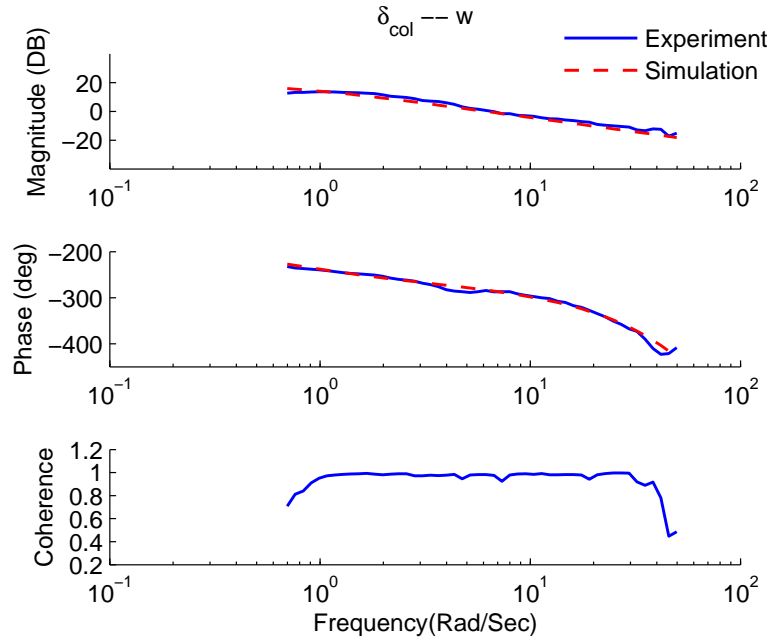


Figure 3.9: Frequency response for Heave dynamics model identification.

Fig.3.9 shows the system identification results for the heave dynamics. We can see that the coherence level remains above 0.8 from 1 to 40 rad/s, indicating the high fidelity of the heave dynamics model.



## Yaw Dynamics

The yaw direction motion is caused by changing the collective pitch angle of the lower rotor, creating torque difference between the upper rotor and lower rotor. It is conventional to have an internal yaw-rate controller to facilitate manual control of the platform. The yaw-stability-augmentation controller is assumed to be of proportional-integral (PI) controller with the system diagram shown in Fig. 3.10. The transfer function from the yaw control  $\delta_{ped}$  to the augmented yaw control  $\bar{\delta}_{ped}$  is,

$$\bar{\delta}_{ped} = \left( K_p + \frac{K_i}{s} \right) (K_a \delta_{ped} - r). \quad (3.14)$$

Breaking Eq.3.14 into more details, we get

$$e_r = K_a \delta_{ped} - r, \quad (3.15)$$

$$\dot{r}_f = K_i e_r, \quad (3.16)$$

$$\bar{\delta}_{ped} = r_f + K_p e_r, \quad (3.17)$$

where  $e_r$  is the tracking error of yaw rate  $r$ ,  $r_f$  is the augmented internal state for yaw controller,  $K_a$  is the feed forward gain,  $K_p$  and  $K_i$  are the gain for the PI controller. The three internal gains  $K_a$ ,  $K_i$  and  $K_p$  need to be identified. From Eq.3.15, we see that  $K_a$  is the static gain from  $\delta_{ped}$  to yaw rate  $r$ . The helicopter is manually piloted to perform hovering turn when the control input and the measured yaw rate  $r$  are recorded. At constant rotating motion in yaw direction, the ratio of  $r$  to  $\delta_{ped}$  is  $K_a$ .  $K_p$  and  $K_i$  can be identified as follows: we place the helicopter on the table without moving, a step input  $\delta_{ped}$  with known value is given to the PI controller. We record the output ( $\bar{\delta}_{ped}$ ) of the PI controller using an oscilloscope and observe the change of the pulse width. The initial ratio between the output and the input is  $K_p/K_a$  and the slope of the step response is  $K_i/K_a$ . The identified parameters are listed in Table.3.4

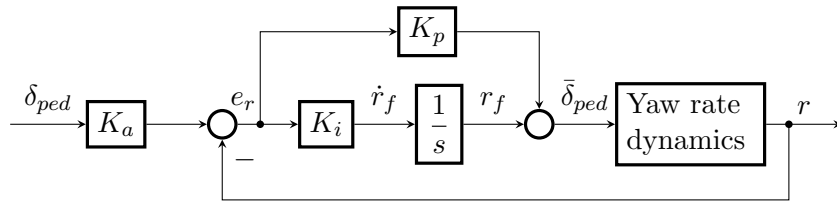


Figure 3.10: Yaw rate feedback controller structure.

Table 3.4: Identified parameters of coaxial helicopter.

Parameter	Physical meaning
$m = 2.080 \text{ kg}$	Total mass of platform
$g = 9.781 \text{ m} \cdot \text{s}^{-2}$	Earth gravitational constant
$J_{xx} = 0.0250 \text{ kg} \cdot \text{m}^2$	Rolling moment of inertia
$J_{yy} = 0.0294 \text{ kg} \cdot \text{m}^2$	Pitching moment of inertia
$J_{zz} = 0.0158 \text{ kg} \cdot \text{m}^2$	Yawing moment of inertia
$l_{\text{up}} = 0.235 \text{ m}$	Distance from upper rotor to center of gravity
$l_{\text{dw}} = 0.135 \text{ m}$	Distance from lower rotor to center of gravity
$H_{\text{mr}} = 0.185 \text{ m}$	Distance from equivalent rotor to center of gravity
$L_b = 675.8 \text{ s}^{-2}$	Lateral rotor spring derivative
$M_a = 794.7 \text{ s}^{-2}$	Longitudinal rotor spring derivative
$\tau = 0.068 \text{ s}$	Equivalent flapping time constant
$A'_{\text{lon}} = 0.898 \text{ rad/s}$	Longitudinal control derivative
$B'_{\text{lat}} = 1.069 \text{ rad/s}$	Lateral control derivative
$\tau_{\text{lat}} = 0.03355 \text{ s}$	Lateral control delay
$\tau_{\text{lon}} = 0.03390 \text{ s}$	Longitudinal control delay
$K_\beta = 11.5029 \text{ Nm}$	Rotor spring constant
$K_a = 2.415$	Scaling factor of the headlock gyro
$K_p = 0.263$	Proportional gain of the headlock gyro
$K_i = 0.149$	Integral gain of the headlock gyro

### 3.3 Modeling of Quadrotor

#### 3.3.1 Overview of Quadrotor Model

The model structure of the quadrotor platform follows the hardware configurations, which is illustrated in Fig. 3.11. The normalized control inputs ( $\delta_{\text{ail}}, \delta_{\text{ele}}, \delta_{\text{thr}}, \delta_{\text{rud}}$ ) are fed into the Naza-M controller, which is an all-in-one stability controller specially designed for multi-rotor flying platforms. With a standard quadrotor frame construction, the default control gains built in Naza-M can already stabilize the inner-loop dynamics very well. Naza-M controller outputs pulse-width modulation (PWM) signals ( $m_1, m_2, m_3, m_4$ ) to drive the four rotors to generate the thrust forces, which not only lift the platform but also maintain its attitude stability. From the perspective of Naza-M, the four inputs correspond to the control references for the roll angle  $\phi$ , pitch angle  $\theta$ , yaw angular rate  $r$ , and the UAV body-frame vertical axis velocity  $w$ .

In the outer-layer dynamics, the quadrotor heading  $\psi$  is the integration of yaw rate  $r$ , and its vertical axis position  $z$  is the integration of vertical velocity  $w_g$  in local NED

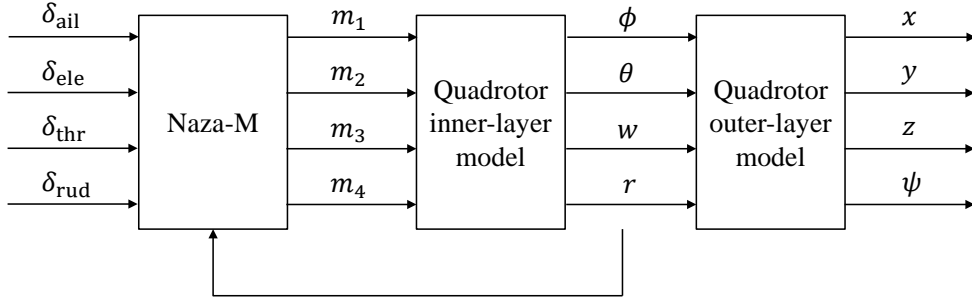


Figure 3.11: Overview of quadrotor model structure.

frame, which is almost the same as the body frame  $z$  velocity  $w$  at hover and steady flight conditions. For the lateral and longitudinal motion, non-zero  $(\phi, \theta)$  angles will induce accelerations in the UAV body-frame  $x$ - and  $y$ -axis. If transformed to the NED frame, they integrate to the NED velocity  $(u_g, v_g)$  and integrate again to extract the NED position  $(x, y)$ .

The quadrotor body coordinate frame is defined as the so-called ‘X’ mode, shown in Fig. 3.12, where the  $x$ -axis is 45 degrees to the physical arms of the frame. Following the right-hand rule, the  $y$ -axis is set to point rightward and the  $z$ -axis to point downwards. Since the structure configuration of the platform and the design of the onboard system are highly symmetric, it is reasonable to assume that the longitudinal and lateral dynamics of this platform are exactly the same, and the model is completely decoupled among all four channels. Hence, we can identify the dynamic models of the four channels independently. The overall system dynamics can be obtained by concatenating the four subsystem dynamics diagonally.

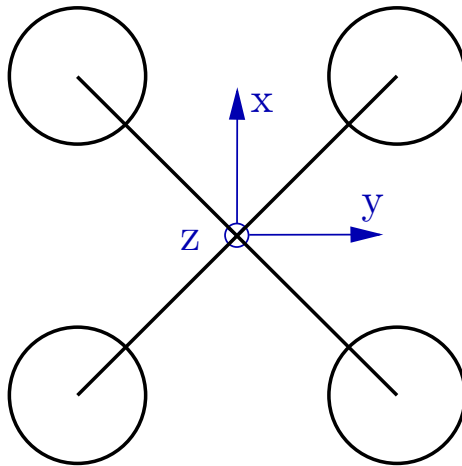


Figure 3.12: Quadrotor body frame definition.

The model identification process is performed in frequency domain, using the standard software—CIFER. It first converts the collected input-output data to frequency-domain responses. Then the frequency domain data are fed into NAVFIT, which is a low-order transfer function fitting module in CIFER. This is justified since the quadrotor model is decoupled and the subsystem dynamics are assumed to be low order linear time invariant systems.

### 3.3.2 Linearized Model Identification

#### Roll Pitch Channel Model Identification

Due to the symmetric structure of of the quadrotor platform, the roll and pitch dynamics share the same model structure as well as parameters. When the platform is perturbed in the aileron or elevator channels, the onboard avionics system can record down the responses of roll angle  $\phi$  (or pitch angle  $\theta$ ), the corresponding body-frame linear velocities  $v$  (or  $u$ ), and the synchronized control inputs  $\delta_{ail}$  (or  $\delta_{ele}$ ). The ultimate goal is to identify the dynamic model from control inputs to the body-frame velocities. However, we can divide this task into two sub-tasks, i.e., identify the model from control inputs to attitude angles and identify the model from angles to velocities. The former part contains information of inner-loop bandwidth and steady-state gain, while the latter part can be used to connect the outer-loop control outputs to the inner-loop control references. The details will be explained in Section 3.3.3.

#### Model from Control Input to Attitude Angle

Using NAVFIT in CIFER, the transfer function from the aileron (or elevator) control input  $\delta_{ail}$  (or  $\delta_{ele}$ ), to the roll  $\phi$  (or pitch  $\theta$ ) angle can be well fitted by the following 4th order linear process model:

$$\frac{\phi(s)}{\delta_{ail}(s)} = \frac{\theta(s)}{\delta_{ele}(s)} = \frac{9688}{s^4 + 27.68 s^3 + 485.9 s^2 + 5691 s + 15750}. \quad (3.18)$$

This transfer function has a bandwidth of 3.89 rad/s and a steady-state gain of 0.6151. The frequency response comparison between the identified model and the flight data is shown in Fig. 3.13. The third sub-plot Fig. 3.13 shows the coherence value of the model. At frequencies below 20 rad/s, the coherence value remains above 0.8, indicating that the system is well characterized by a linear process in this frequency range.

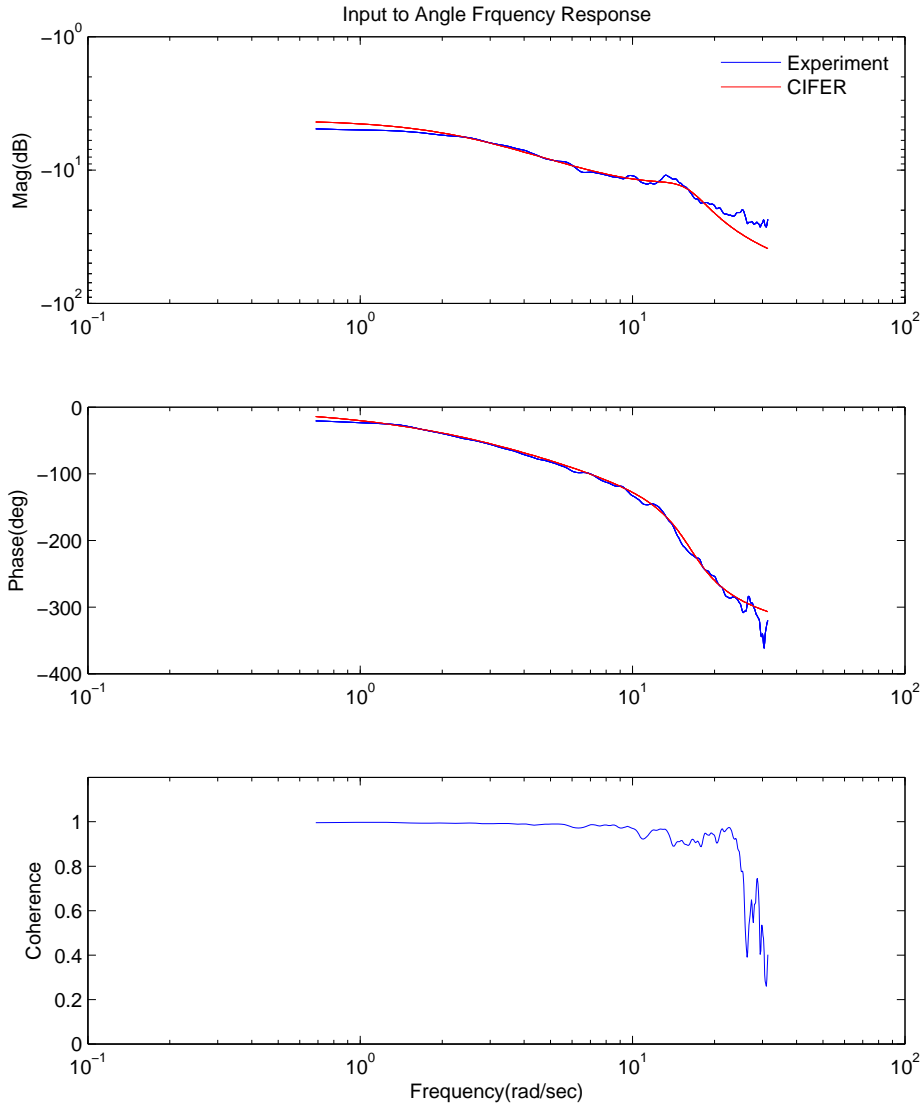


Figure 3.13: Response comparison using frequency-sweep input  $\{\delta_{ail}, \delta_{ele}\} - \{\phi, \theta\}$ .

Time domain verification of the model using a different set of experimental data is performed also. The input signal from the verification data set is fed into the model and its predicted output is compared with the experimental output. Fig. 3.14 shows the model performance for a series of chirp signals, and Fig. 3.15 shows the error difference between the model output and the experimental output. It can be seen that the error is very small, indicating that the obtained model is very reliable.

### Model from Attitude Angle to Linear Velocity

Using the same approach, the transfer function from roll  $\phi$  (or pitch  $\theta$ ) angle to the lateral (or longitudinal) velocity can be obtained by fitting a first order transfer function

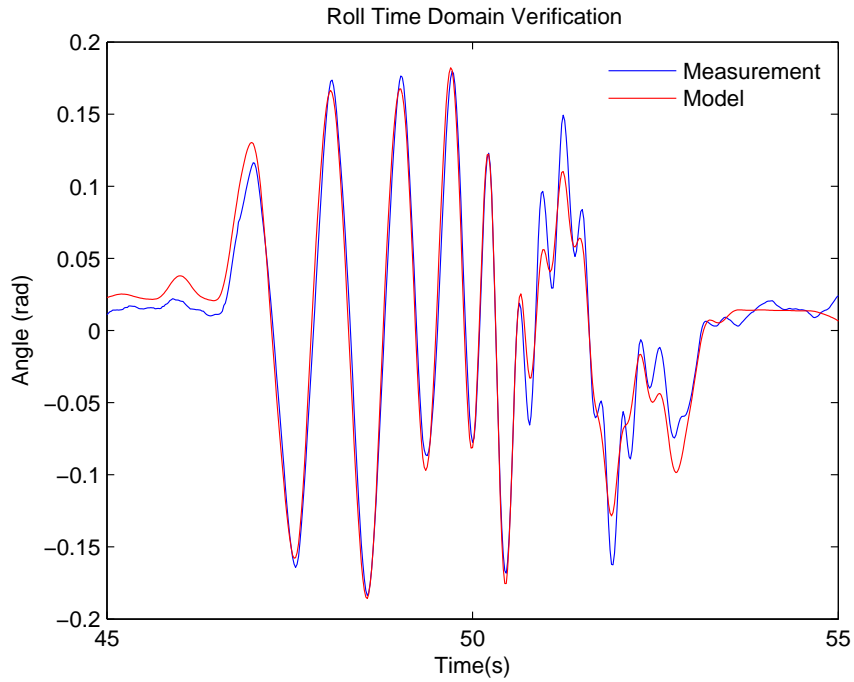


Figure 3.14: Roll angle time domain model verification.

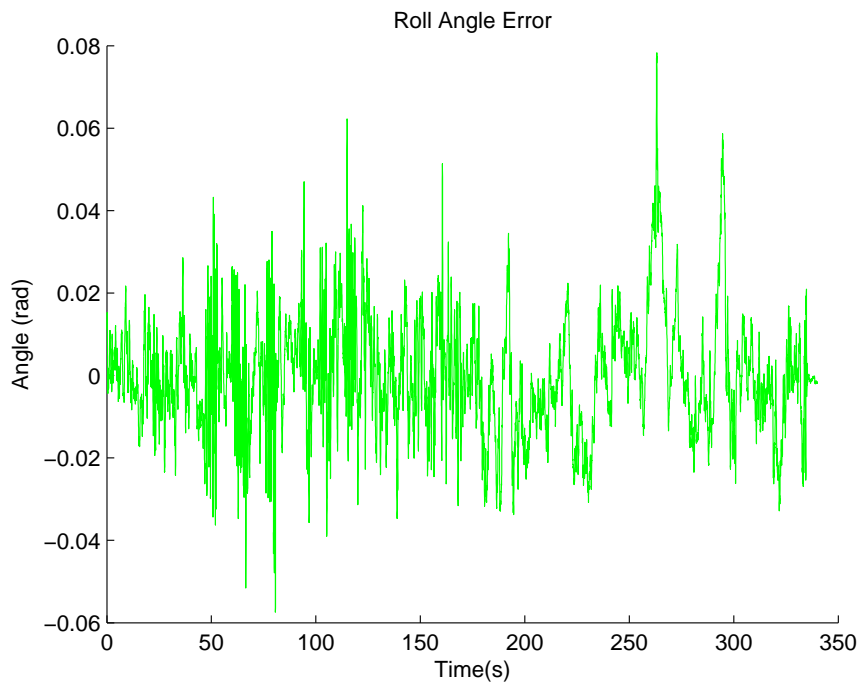


Figure 3.15: Roll angle time domain error between model prediction and experiment.

as below:

$$\frac{v(s)}{\phi(s)} = \frac{u(s)}{\theta(s)} = \frac{8.661}{s + 0.09508}. \quad (3.19)$$

This relationship will be used later in Section 3.3.3 to connect the inner-loop and outer-loop control layers. The time domain verification results are shown in Fig. 3.16.

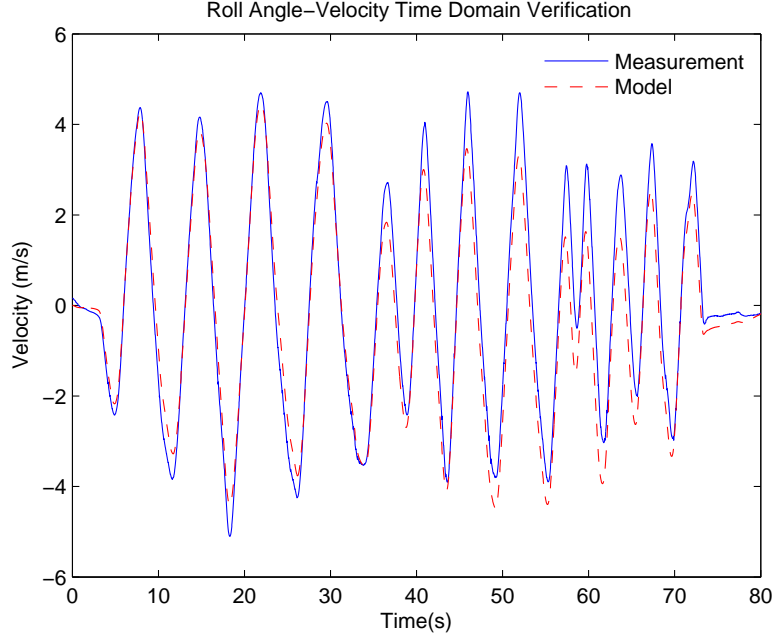


Figure 3.16: Time domain error from roll angle to y velocity.

### Yaw Channel Model Identification

Since the inner-loop dynamics in the yaw channel is extremely fast, thanks to the superb performance from Naza-M, the relationship between the rudder input  $\delta_{\text{rud}}$  and the yaw rate  $r$  can be treated as a static gain. If we consider the outer-layer dynamics in this channel also, then the transfer function from rudder input  $\delta_{\text{rud}}$  to the yaw angle  $\psi$  is just an integration of a constant:

$$\frac{\psi(s)}{\delta_{\text{rud}}(s)} = \frac{3.372}{s}. \quad (3.20)$$

Fig. 3.17 and Fig. 3.18 show the time domain verification results for both the yaw angle and angular rate. In both figures, the experimental data agrees well with that predicted by the identified model.

### Heave Channel Model Identification

The transfer function from the throttle input  $\delta_{\text{thr}}$  to the body-frame  $z$ -axis velocity  $w$  is identified as:

$$\frac{w(s)}{\delta_{\text{thr}}(s)} = -\frac{13.35}{s + 2.32}. \quad (3.21)$$

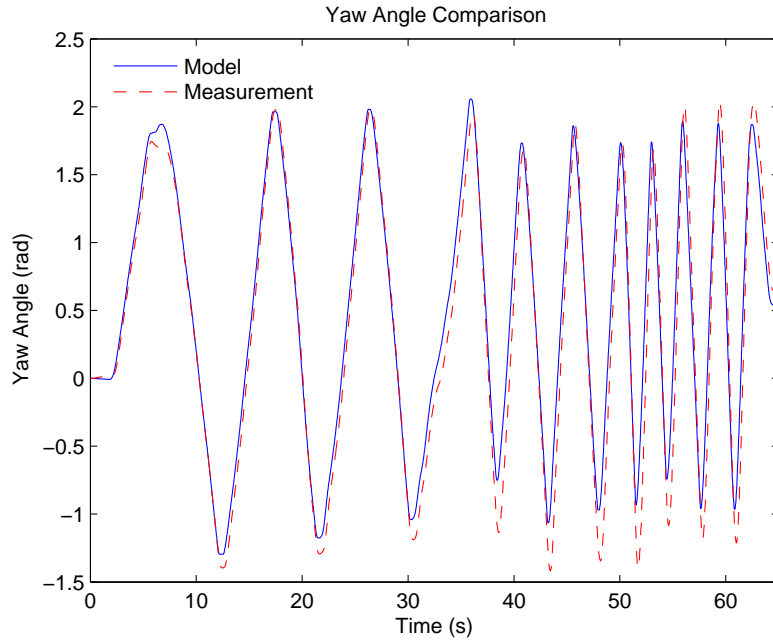


Figure 3.17: Time domain comparison of yaw angle.

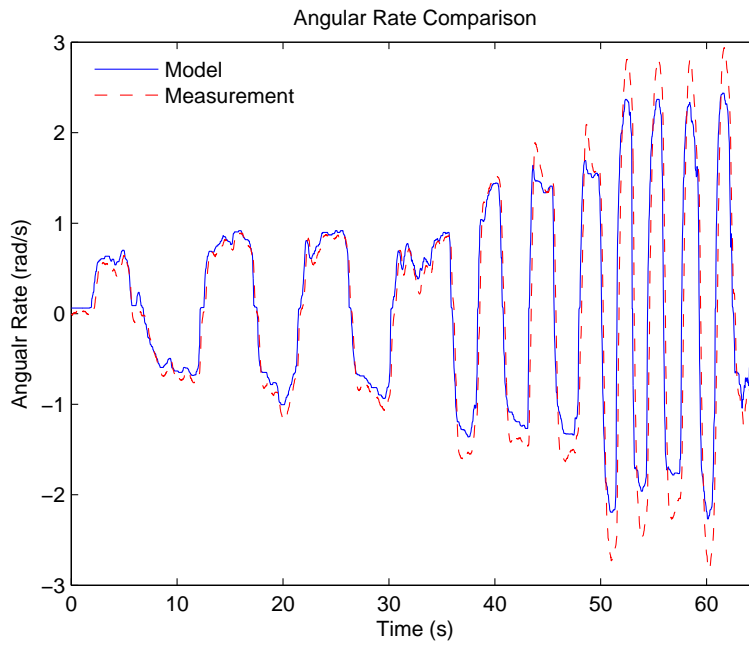


Figure 3.18: Time domain comparison of yaw angular rate.

The negative sign is due to the opposite definition of positive direction for the input and output. When the throttle stick is pushed up, all four motors speed up. The generated force will lift the UAV platform upwards. However, this upward motion is actually seen as a negative velocity as defined in the  $z$ -axis of the UAV body frame. Fig. 3.19 shows the time domain verification results for the heave velocity.



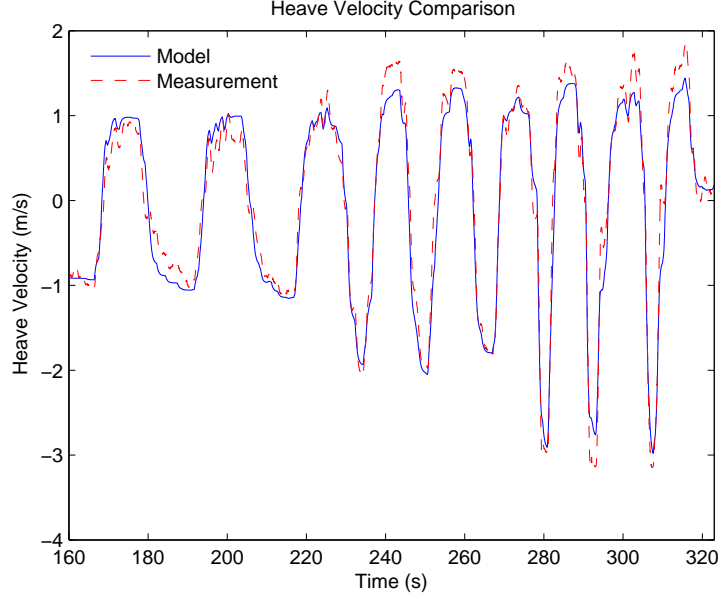


Figure 3.19: Time domain comparison of heave velocity.

### 3.3.3 Control Law Design

As the platform is already stabilized in the attitude dynamics by the Naza-M controller (see *Inner-loop controller* in Fig. 3.20), only the outer-loop controller (see *Outer-loop controller* in Fig. 3.20) can be customized to achieve the reference tracking function. The outer-loop controller enables the UAV to follow external references, including the linear position and the heading angle. To achieve this, the robust and perfect tracking (RPT) controller is adopted from [15, 45], from which the design procedures of the RPT controller for the state feedback case is followed. The applications of RPT to a single rotor UAV have been presented in [14, 78].

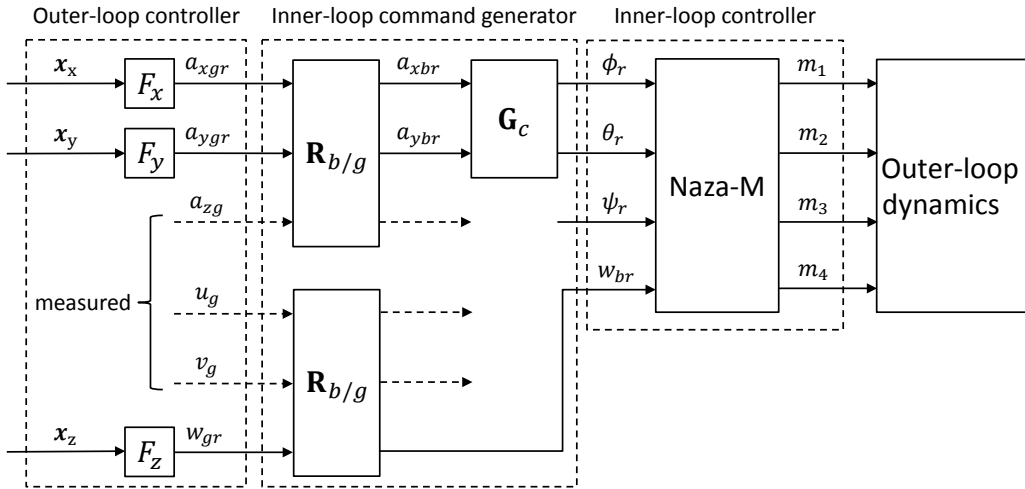


Figure 3.20: Control structure of the quadrotor UAV.

According to the study of [53], the outer dynamics of the quadrotor is differentially flat. That means all its state variables and inputs can be expressed in terms of algebraic functions of flat outputs and their derivatives. A proper choice of flat output is

$$\sigma = [x, y, z, \psi]^T. \quad (3.22)$$

The four outputs,  $x$ ,  $y$ ,  $z$  and  $\psi$  are independent. The UAV can be considered as a mass point with constrained velocity, acceleration, jerk, and so forth. A stand-alone RPT controller based on multiple-layer integrator in each individual axis can be designed.

### Control law design for x-y direction

For precision control, it's desirable to include an integrator to ensure zero steady state error in case of step input. We propose an RPT controller which considers the integration of position tracking error as an augmented state. The system with state-augmentation is formulated as,

$$\Sigma_{\text{AUG}}^{\text{xy}} : \left\{ \begin{array}{l} \dot{\tilde{\mathbf{x}}}_{\text{xy}} = \begin{bmatrix} 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tilde{\mathbf{x}}_{\text{xy}} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \mathbf{u}_{\text{xy}} \\ \tilde{\mathbf{y}}_{\text{xy}} = \tilde{\mathbf{x}}_{\text{xy}} \\ \tilde{\mathbf{h}}_{\text{xy}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tilde{\mathbf{x}}_{\text{xy}} \end{array} \right. \quad (3.23)$$

where  $\tilde{\mathbf{x}}_{\text{xy}} = [\int e \ r_p \ r_v \ r_a \ p \ v]^T$ ;  $r_p$ ,  $r_v$ ,  $r_a$  are the position, velocity and acceleration references;  $p$ ,  $v$  are the actual position and velocity;  $e = p - r_p$  is the tracking error of position. Since there is error integration  $\int e$  in the augmented states, the feedback control law would contain a term of  $K_i \int e$ . Following the steps in [45], a linear state feedback control law of the form (3.24) is acquired,

$$\mathbf{u}_{\text{xy}} = \mathbf{F}_{\text{xy}}(\varepsilon) \tilde{\mathbf{x}}_{\text{xy}} , \quad (3.24)$$

where

$$\mathbf{F}_{xy}(\varepsilon) = \begin{bmatrix} \frac{-k_i \omega_n^2}{\varepsilon^3} & \frac{\omega_n^2 + 2\zeta \omega_n k_i}{\varepsilon^2} & \frac{2\zeta \omega_n + k_i}{\varepsilon} \\ 1 & -\frac{\omega_n^2 + 2\zeta \omega_n k_i}{\varepsilon^2} & -\frac{2\zeta \omega_n + k_i}{\varepsilon} \end{bmatrix}, \quad (3.25)$$

where  $\varepsilon$  is a design parameter to adjust the settling time,  $\omega_n$ ,  $\zeta$ ,  $k_i$  are the parameters that determine the desired pole locations of the infinite zero structure of  $\Sigma_{\text{AUG}}^{\text{xy}}$  through:

$$p(s) = (s + k_i)(s^2 + 2\zeta \omega_n s + \omega_n^2). \quad (3.26)$$

In principle, when the design parameter  $\varepsilon$  is small enough, the RPT controller gives arbitrarily fast response. However, in practice, due to the constraints of physical system and inner loop dynamics, we would like to limit the bandwidth of the outer loop to be at least one third of the inner loop system bandwidth. The roll/pitch dynamics has a bandwidth of 3.82 rad/s. For roll/pitch outer loop controller, we select the parameters in Eq. 3.27 to have a bandwidth of 0.83 rad/s:

$$\omega_n = 0.4, \quad \zeta = 1.2, \quad \varepsilon = 1, \quad k_i = 0.8. \quad (3.27)$$

### Control law design for heave and yaw direction

For the outer loop controller in the heave dynamics and the yaw dynamics, the inner loop controller already controls the heave velocity  $w$  and the yaw angular velocity  $r$ . We only need to design a controller to control the height  $z$  and the yaw angle  $\psi$ . Similarly, the integral of tracking error is augmented to the original system and forms another augmented system  $\Sigma_{\text{AUG}}^{\text{hy}}$ :

$$\Sigma_{\text{AUG}}^{\text{hy}} : \begin{cases} \dot{\tilde{\mathbf{x}}}_{\text{hy}} = \begin{bmatrix} 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tilde{\mathbf{x}}_{\text{hy}} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \mathbf{u}_{\text{hy}} \\ \tilde{\mathbf{y}}_{\text{hy}} = \tilde{\mathbf{x}}_{\text{hy}} \\ \tilde{\mathbf{h}}_{\text{hy}} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tilde{\mathbf{x}}_{\text{hy}} \end{cases} \quad (3.28)$$

where  $\tilde{\mathbf{x}}_{\text{hy}} = [e \ r_p \ r_v \ p]^T$ ;  $r_p, r_v$  are the position, velocity references;  $p$  is the actual height or yaw angle;  $e = p - r_p$  is the tracking error of height or yaw angle.

A linear state feedback control law of the form (3.29) is acquired,

$$\mathbf{u}_{\text{hy}} = \mathbf{F}_{\text{hy}}(\varepsilon) \tilde{\mathbf{x}}_{\text{hy}}, \quad (3.29)$$

where

$$\mathbf{F}_{\text{hy}}(\varepsilon) = \begin{bmatrix} -\frac{\omega_n^2}{\varepsilon} & \frac{2\zeta\omega_n}{\varepsilon^2} & 1 & -\frac{2\zeta\omega_n}{\varepsilon^2} \end{bmatrix}. \quad (3.30)$$

For heave controller:

$$\omega_n = 0.5, \quad \zeta = 1.1, \quad \varepsilon = 1. \quad (3.31)$$

For yaw angle controller:

$$\omega_n = 1, \quad \zeta = 1, \quad \varepsilon = 1. \quad (3.32)$$

### Command generator

From Fig. 3.20, it can be seen that the output from the outer-loop controller in physical meaning is the desired accelerations in  $xy$ -axis and the desired velocity in  $z$ -axis, both in global frame. However, the inner-loop controller is looking for attitude references ( $\phi_r, \theta_r, \psi_r$ ) and the body-frame  $z$ -axis velocity reference. A conversion is needed to link the two control layers together. This leads to another functional block called the *inner-loop command generator*, in which a rotational conversion from the global frame to the body frame  $\mathbf{R}_{b/g}$  is needed and another matrix  $\mathbf{G}_c$  is used to convert the desired acceleration references to the desired attitude angles. For all quadrotor UAVs,

$$\mathbf{G}_c \approx \begin{bmatrix} 0 & 1/g \\ -1/g & 0 \end{bmatrix}, \quad (3.33)$$

where  $g$  is the gravity constant.

### 3.3.4 Flight Test Results

A full envelope trajectory is designed to validate the performance of the control law. The flight test is performed in open space where GPS signals are consistently available. Using GPS signal to provide absolute position estimate could isolate the potential problems

caused by other state estimates such as laser odometry or visual odometry. Flight in clear space also removes the need for obstacle avoidance. The full envelope trajectory includes taking off and ascending to 100 m, navigating to 4 waypoints, returning home and landing on the original take-off position.

Figures from Fig. 3.21 to Fig. 3.24 show the flight position, velocity and heading estimates in x, y, heave and yaw directions. The corresponding references are also plotted to show the tracking performance of the outer-loop control law. Before 75 seconds, the UAV ascends to 100 m during which the x, y and yaw remain constant. After reaching the 100 m height, the UAV begins to fly to the first waypoint. After finishing all four waypoints, the UAV goes back to the origin and lands. As shown in Fig. 3.21-3.22, during the waypoint navigation, the tracking error in both x and y directions are below 1 m. The height tracking tracking error is below 5 m as shown in Fig. 3.23. The height measurement undergoes fast drop or jump due to the wind disturbance as shown in 90 seconds and 130 seconds.

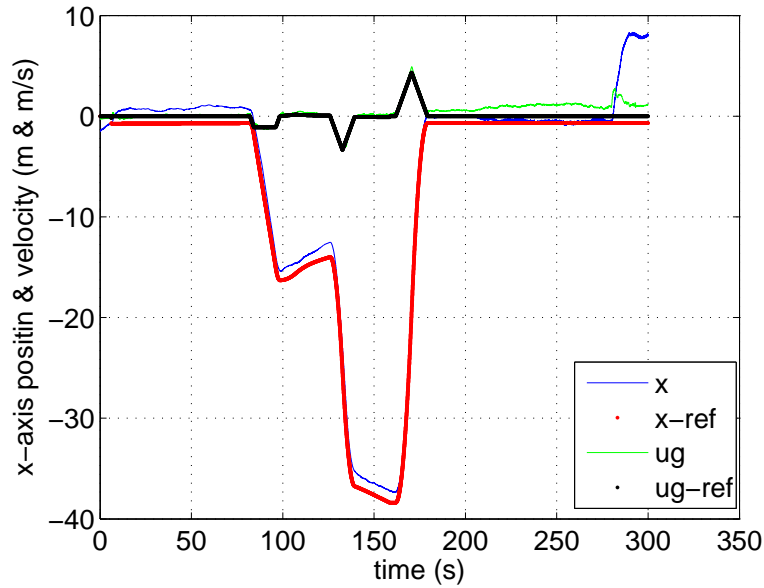


Figure 3.21: x direction tracking performance.

### 3.4 Conclusion

This chapter has discussed in detail the model and control of UAV platforms, including a coaxial helicopter and a quadrotor. The coaxial helicopter has more complex model than the quadrotor due to the complicated aerodynamics interaction between the upper rotor and the lower rotor. Thus we derive the coaxial helicopter model in detail by analyzing each its subsystem. With the model structure identified, we avoid deriving

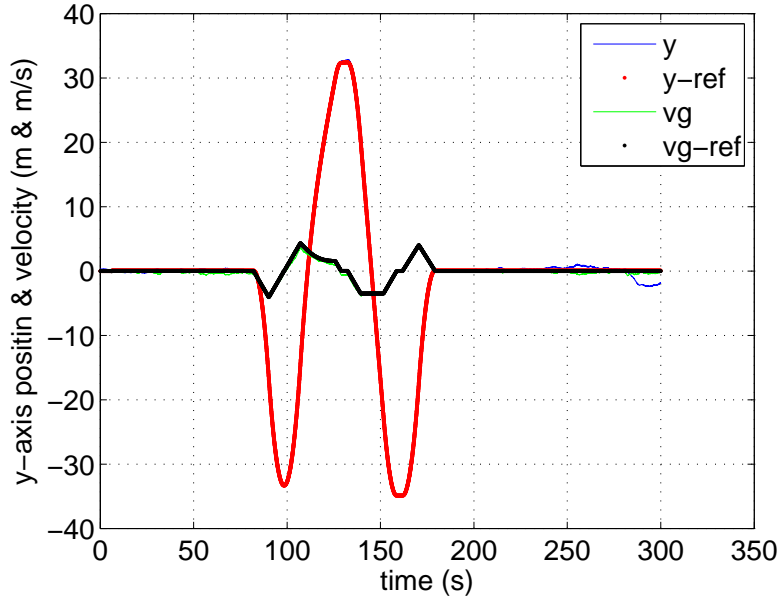


Figure 3.22: y direction tracking performance.

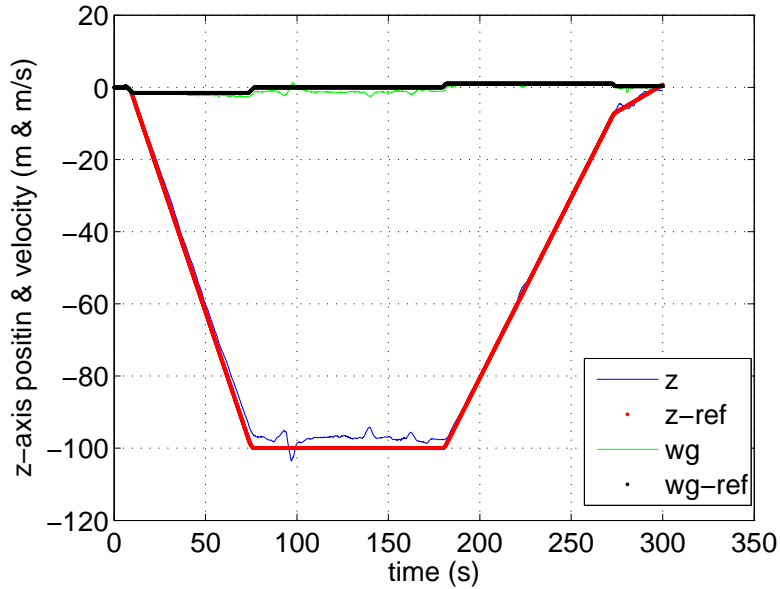


Figure 3.23: z direction tracking performance.

a complete nonlinear model for our coaxial helicopter since our designed UAV work only in near-hover conditions. By assuming that the total thrust of the coaxial rotor is equal to the gravity force of the platform, we identified a linear model consisting of roll-pitch dynamics, heave dynamics, and yaw dynamics. The parameters in the model are identified either by direct measurements or by system identification method.

During the modeling of the coaxial helicopter, we realize that it tends to not meet the navigation requirements. As discussed in the last chapter, our avionics system includes at least one laser range finder. But the fuselage of the coaxial helicopter is

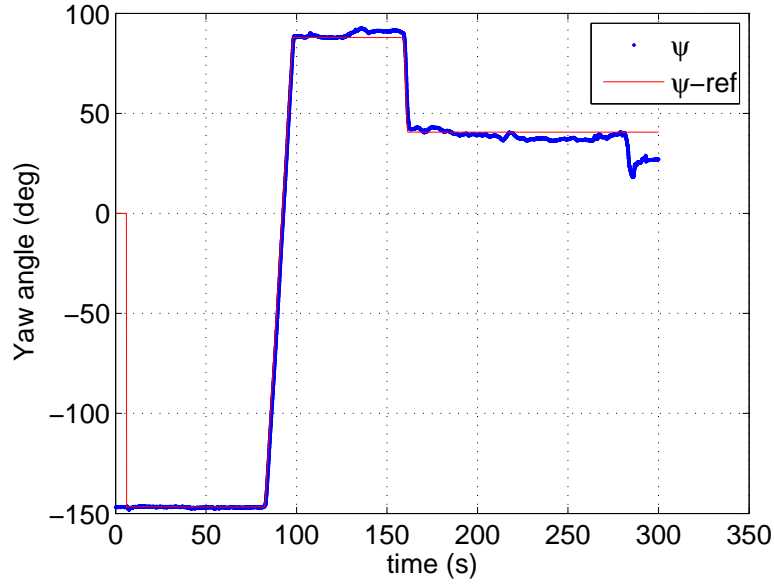


Figure 3.24: Yaw direction tracking performance.

right beneath the two rotating rotors, making it impossible to assemble the laser range finder without occlusion from the fuselage. Besides, there are lots of work to be done in order to make the coaxial helicopter maintain its stability. Since our focus is on the navigation algorithm development, which is independent of the platform itself, we turn to the quadrotor for our platform. However, coaxial helicopters still remain their unique advantages, especially the high energy efficiency and the compact size. Once the navigation algorithms are fully verified, implementing them on the coaxial helicopter will lead to a UAV with the same intelligence but a smaller footprint and longer endurance.

The quadrotor, because of its symmetric structure, has a very simple decoupled model structure, which is separated into the inner-loop and the outer-loop. Commercial inner-loop controllers have greatly reduced the work in quadrotors modeling. We need to only model and control the decoupled out-loop dynamics. The linear model of quadrotor is again derived from the system identification method. A robust perfect tracking control law is designed to control the quadrotor to track changing trajectory references. Fully autonomous flights based on GPS have been performed to verify the model and the control law developed in this chapter.

## Chapter 4

# UAV State Estimation Using Laser Range Finder

### 4.1 Introduction

Navigation of UAVs requires the states to be estimated at every time step to facilitate the autonomous control and the path planning. With respect to an inertial frame, the states of UAVs include the 3-axis position, velocity and orientation of the UAV body. The orientation can be estimated accurately with the onboard IMU and magnetometer. But for the velocity and position, pure integration of the acceleration would soon render the signal out of bound because of the bias and noise of accelerometers. It is conventional to use external absolute measurement, such as GPS or beacons, to fuse with the acceleration measurement to obtain real-time velocity and position estimates. The fusion of IMU and GPS is widely adopted in long range navigation of UAVs.

However, when GPS signal is not available, other sensing modalities have to be considered. Relative sensing techniques, based on wheel encoders, cameras, and laser range finders, are commonly adopted. Wheel encoders are universally applied in ground vehicles and produce acceptable results, but the uneven terrains or the wheel slippages affect their accuracy. Vision sensing is another choice, in which optic flow is a standard technique to obtain 2D velocity estimation. Vision sensing, however, requires specific illumination conditions of the environment. Laser range finders are the practically ideal and feasible method because they measure both the bearing and the distance of the environment relative to the UAV.



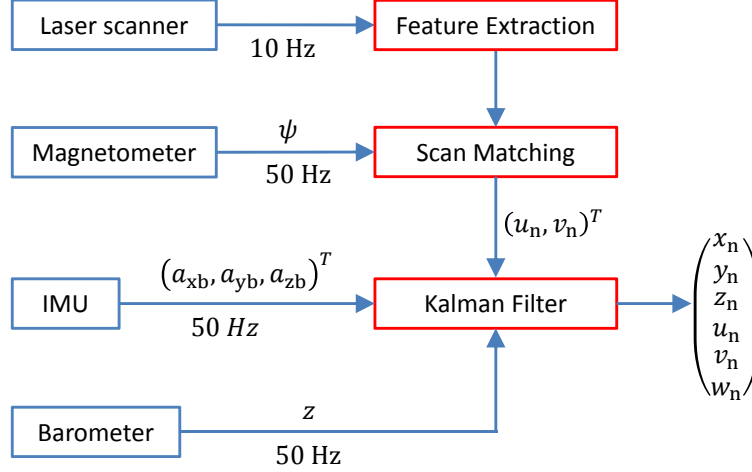


Figure 4.1: The architecture of the IMU-driven Kalman filter.

This chapter presents the UAV state estimation framework based on the laser odometry. As shown in Fig. 4.1, the framework consists of laser feature extraction, laser scan matching and sensor fusion using a Kalman filter. To extract features from forests environment, the range scan is processed in three steps: preprocessing, segmentation and extraction. The preprocessing step removes any invalid measurement points below 20 mm and above 10 m. The segmentation step separates the range scan to clusters with various numbers of points. The candidate clusters are evaluated based on a series of geometric descriptors, which help select the clusters corresponding to tree trunks. The estimated tree centers are treated as the feature points used for scan matching. The details about the laser feature extraction will be explained in Section 4.2.

For matching two consecutive scans, iterative closest matching (ICP) is widely used. There are several modular steps in the ICP process, in which the point selection, the data association and the rigid motion estimation are the three critical steps to make the scan matching fast and accurate. We will address these details in Section 4.3.

Feature-based scan matching produces incremental translation and rotation between two consecutive scans at a low update rate, hence a Kalman filter is required to fuse the velocity of scan matching with the high update rate acceleration measurement from the IMU. We discuss the details of the design and implementation of the Kalman filter in Section 4.4. Finally, the developed real-time scan matching and Kalman filter are integrated in a quadrotor and performed autonomous flights in a small forest. We present the results of the flights in Section 4.5.

## 4.2 Feature Extraction

### 4.2.1 Laser Range Finder Model

We use a Hokuyo UTM-30LX laser range finder (LRF) in this research for acquiring a 2D scan of the environment. Laser scanners are the most attractive sensors for localization and mapping on mobile robots due to the accurate measurement. The specification of the Hokuyo UTM-30LX is listed in Table. 4.1.

Table 4.1: Hokuyo UTM-30LX specification

Supply Voltage	12 VDC $\pm$ 10%
Supply Current	Max: 1 A, Normal: 0.7 A
Power Consumption	<8 W
Detection Range	0.1~30 m
Measurement Resolution	1 mm, 0.1 - 10 m, $\sigma < 10$ mm, 10 - 30 m, $\sigma < 30$ mm
Scan Angle	270 $^\circ$
Angular Resolution	0.25 $^\circ$
Scan Speed	25 ms (Motor speed: 2400rpm)
Interface	USB 2.0 full speed (12Mbps)
Weight	230 g (with customized cable)
Mechanical Dimension	60 mm $\times$ 60 mm $\times$ 85 mm

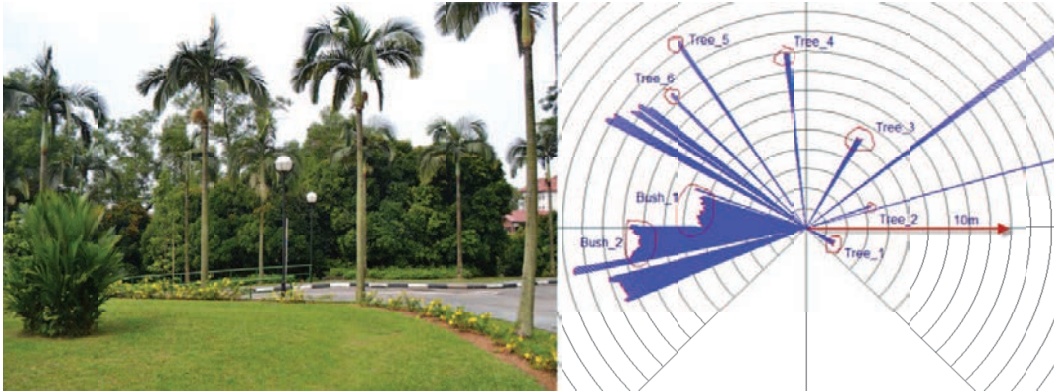


Figure 4.2: Image and laser scanner data for a testing scenario.

With respect to forest navigation, preliminary tests have been carried out to evaluate the performance of Hokuyo UTM-30LX. The visualization of one scan from the outdoor test scenario is shown in Fig. 4.2, where the UAV is surrounded by several trees and clusters of bushes. The scanning map is zoomed into 10 meters and the sensor is exposed to strong and direct sunlight. The range measurement returned by the laser scanning

range finder is spatially sparse, where clusters of points with different number of points are present. Intuitively it makes sense to assume each cluster correspond to one object in the environment, either tree trunks, bushes or ground strikes.

In order to extract salient features from the range scan, the sensor measurement model needs to be identified. The laser range finder has a measurement range of up to 30 m with a  $270^\circ$  scanning angle. The laser range image is expressed in the polar coordinate in the form of  $\{(p_i, \theta_i), i = 1 \dots N\}$ , where  $N$  is the total number of measurement points in each scan,  $p_i$  is the object distance to the origin of the LRF at angle  $\theta_i$ . Fig. 4.3 illustrates the measurement model, which is defined in a body-fixed coordinate system, in which x axis points forward and y axis points rightward, whereas the z axis points into the paper to comply with the right-hand rule. The angular position of each measurement point  $(p_i, \theta_i)$  is defined as the angle between x axis and the laser beam.

In the body frame  $b$ , the Cartesian representation of the range scan is,

$$\begin{pmatrix} x_i^b \\ y_i^b \end{pmatrix} = \begin{pmatrix} p_i \cos(\theta_i) \\ p_i \sin(\theta_i) \end{pmatrix}, i = 1 \dots N. \quad (4.1)$$

In a local NED navigation coordinate system  $n$ , the coordinate of the body frame origin is defined as  $O(x_o^n, y_o^n)$  and the yaw angle as  $\psi$ . The range image is represented in the NED frame as,

$$\begin{pmatrix} x_i^n \\ y_i^n \end{pmatrix} = \begin{pmatrix} x_o^n + p_i \cos(\theta_i + \psi) \\ y_o^n + p_i \sin(\theta_i + \psi) \end{pmatrix}. \quad (4.2)$$

#### 4.2.2 Feature Extraction Procedure

The foliage environment is more complex and unstructured compared to indoor environments. Regular features like lines and corners are absent. Fortunately, the forest is always full of trees, which can serve as the salient features. Fig. 4.4 shows a typical forest environment with close-to-vertical tree trunks in the surrounding environment. One scan of the area at the flight height of approximately 1.5 m is plotted in Fig. 4.5. The small clusters of points of circular contour correspond to the tree trunks while the long and wide clusters of line shape correspond to ground strikes or bushes. This section discusses the procedures of extracting tree features in complex range scans.

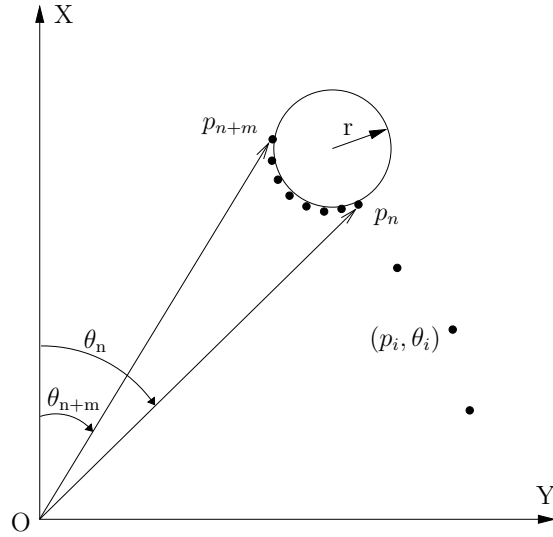


Figure 4.3: Laser range finder measurement model.



Figure 4.4: Test scenario with UAV flying in the air.

To extract the validated trees, the laser range scans are processed in three steps: preprocessing, segmentation and extraction. In the preprocessing step, the range points beyond 10 m range are removed due to two reasons. First, the laser range finder has different noise specification below and above 10 m as listed in Table. 4.1. Therefore, using points beyond 10 m requires two different noise models of the laser range finder. Second, the UAV needs to fly under the tree canopies, limiting its flight height to be less than 2 m. Large range points are more likely to be the points striking the ground. For example, assuming a planar ground plane, if the UAV flies at a height of 2 m and the UAV pitches forward by 11.5 degrees, the range points at 10 m will hit the ground.

The second step is segmentation. We separate the whole laser range scan into small clusters with different number of points. Clustering the range scan is preferable in

the case when the range scan is not continuous. The key step is the selection of the segmentation threshold. Unrealistic threshold will lead to redundant or insufficient clusters. The details are discussed in Section 4.2.3.

The third step is to examine the clusters in a series of geometric descriptors to produce the validate tree trunk centers. The geometric descriptors consist of the number of points, the cluster width, convexity, etc. The estimated centers of validated clusters are treated as the salient features for future state estimation. The details are discussed in Section 4.2.4.

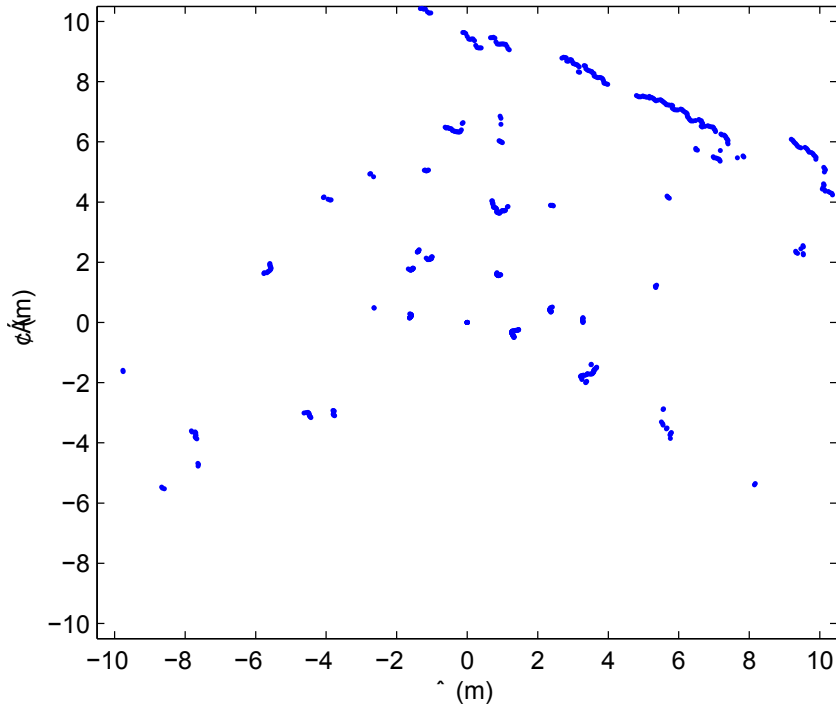


Figure 4.5: Typical laser measurement in a foliage environment.

### 4.2.3 Scan Segmentation Algorithm

In indoor environment, the laser scan data can be very dense since every point in the scan may represent a distance to a certain object. It is a different case in outdoor environment where some beams of laser scan may not hit any objects. In forested areas, tree stems and bush clusters demonstrate significant spatial discontinuity as shown in Fig. 4.2. Making use of the spatial discontinuity, the range scans can be clustered. It is popular to cluster a set of data using  $k$ -means [48], which partitions the observed data to the cluster with the nearest centroid. However, this method need to fix the number of clusters *a priori*, which is not practically feasible because the laser range finder is

scanning at different places while the UAV platform is moving. On the other hand, although the measurement points from one scan are spatially disconnected, but their index are consecutive spanning from 1 to 1081. We take this factor in to a account and use a single-linkage clustering algorithm [27]. This algorithm states that any two consecutive points that are closer to each other than a threshold are considered to belong to the same cluster.

For any two consecutive points in a laser scan in polar coordinates,  $(p_1, \theta_1)$  and  $(p_2, \theta_2)$ , the distance between these two points is defined as:

$$D = \sqrt{p_1^2 + p_2^2 - 2p_1p_2 \cos(\theta_1 - \theta_2)}. \quad (4.3)$$

Since we only evaluate the distance of the neighboring points,  $(\theta_1 - \theta_2)$  in Eq. 4.3 is equal to the laser scanner's angular resolution ( $0.25^\circ$ ). For fast onboard computation,  $\cos(\theta_1 - \theta_2)$  can be precomputed and save as a constant. The detailed realization of the algorithm is shown in Algo.1

---

**Algorithm 1:** Laser Scanner Data Clustering

---

**Input:** Laser range data  $\mathbf{y}$ , angle  $\theta$   
**Output:** Cluster with various number of points

- 1 Initialize the number of cluster to 1;
- 2 Initialize the number of points each cluster to 0;
- 3 **foreach** *point in y* **do**
- 4     Calculate the distance  $\mathbf{D}$  between the current point and the next point ;
- 5     **if**  $\mathbf{D} < d_{th} \ \& \ \mathbf{D} > 0$  **then**
- 6         Add the current point to the current cluster;
- 7         Increase current cluster point counter by 1;
- 8     **else**
- 9         Increase number of cluster by 1;
- 10    **end**
- 11 **end**
- 12 **return** *Clusters of points*

---

The distance threshold in step 5 is important in the segmentation process. We will analyze the determination of the threshold in next section. An unrealistic segmentation threshold will group points of different objects together or separate points which are actually on the same continuous surface. Referring to Fig. 4.3, we assume the shape of tree stems is circular. A realistic segmentation threshold is able to group points  $\{p_n \dots p_{n+m}\}$  to the same cluster, which corresponds to a single tree stem with radius  $r$ .

With the circular surface assumption of tree stems, the segmentation threshold should be larger than the maximum consecutive distance difference of all the points  $\{p_n \dots p_{n+m}\}$ . It is clear that the maximum distance difference occurs at the tangent line of the circle, between point  $p_{n+m}$  and  $p_{n+m-1}$ . On the other hand, the maximum distance varies at different tree radius and distance to the measurement origin. In order to find a suitable threshold, the distance and radius of tree stems are enumerated respectively. The tree radius ranges from 0.1 m to 0.4 m, and the tree distance to the measurement origin ranges from 0.6 m to 10 m. Every combination of tree radius and distance generates one maximum consecutive distance. Fig. 4.6 visualizes the matrix of the maximum consecutive distance, which clearly indicates that the maximum point occurs at the largest tree radius 0.4 m and distance of 10 m. The maximum distance difference is 0.1851 m labeled as the red eclipse in Fig. 4.6. In practice, the segmentation threshold is chosen to be 0.2 m, which is larger than the upper limit of the maximum distance difference matrix.

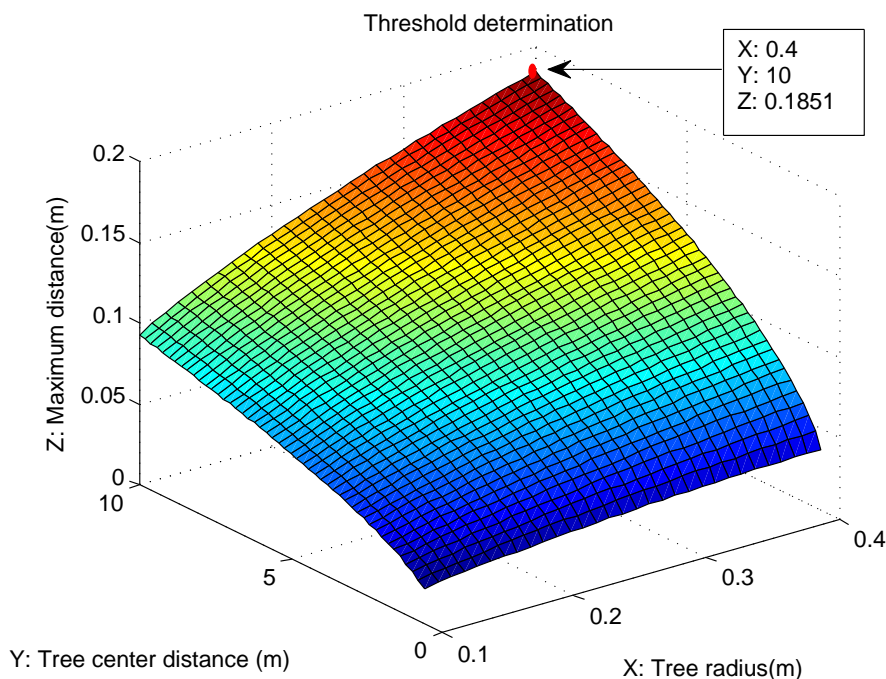


Figure 4.6: Segmentation threshold determination.

#### 4.2.4 Geometric Descriptors

The segments generated from the segmentation process may be tree stems, cluster of bushes or ground strikes. It is indispensable to distinguish the tree stems from other

segments. Referring to the feature list for people leg detection in [3], we assign a group of parameters which represent the characteristics of each segment. The geometric descriptors are defined as follows:

1. Number of points.
2. Jump distance to the next adjacent segments: the Euclidean distance between the current segment centroid to the next segment centroid.
3. Radius and center of the circle: fitting a circle using the points in the cluster.
4. Width: distance from the first point to the last point of the segment.
5. Average angle position: the mean angular position of a segment.
6. Average distance: the mean distance to the center of laser range finder.
7. Convexity: all the points of one segment should lie between line segment of the extracted circle center and the body frame origin. If any measurement points go beyond the extracted circle center, the current cluster is believed to be non-convex from the perspective of the body origin.
8. Flag: this is a flag to show whether the corresponding segment is a valid tree stem.

The candidate tree segments are validated through the series of threshold listed in Table 4.2. The thresholds are chosen by analyzing the data from flight test. Tuning of thresholds is critical since a too relax threshold cannot reject false tree candidate while a too strict threshold may remove potential features.

Table 4.2: List of geometric threshold for tree trunk extraction.

Feature	Lower limit	Upper limit
Number of points	3	50
Distance to adjacent segment	0.3 m	30 m
Tree radius	0.1 m	0.5 m
Width	0.1 m	0.7 m

The most important geometric descriptors are the circle radius and center. Fitting a circle model from a cluster of points can be realized using various methods. In this study, we have evaluated three methods: one uses least square fitting and the other two



use bounding angle of the cluster. These methods produce different results with respect to the circle position and radius.

### Fitting circle using least square

Referring to Fig. 4.7(a), for each cluster, let a circular stem represented in Cartesian coordinates to be  $(x - x_c)^2 + (y - y_c)^2 = r_c^2$ , where  $(x_c, y_c)$  is the origin and  $r_c$  is the radius. The unknown vector is formulated as  $\mathbf{x} = (x_c \ y_c \ x_c^2 + y_c^2 - r_c^2)^T$ . For the  $n$  points in the segment, the over determined system  $\mathbf{Ax} = \mathbf{b}$  is

$$\mathbf{A} = \begin{pmatrix} -2x_1 & -2y_1 & 1 \\ -2x_2 & -2y_2 & 1 \\ \vdots & \vdots & \vdots \\ -2x_n & -2y_n & 1 \end{pmatrix} \mathbf{b} = \begin{pmatrix} -x_1^2 - y_1^2 \\ -x_2^2 - y_2^2 \\ \vdots \\ -x_n^2 - y_n^2 \end{pmatrix}. \quad (4.4)$$

When  $n$  is larger than 3, the system solution is given by

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}, \quad (4.5)$$

the matrix  $\mathbf{A}^T \mathbf{A}$  may be singular or ill-conditioned when the number of measurement points are too small or the points in the segment are close to line. In practice, the condition number needs to be checked before the inverse operation. This method produces accurate results if the measurement points lie closely on the contour of circular tree trunks in the least square sense.

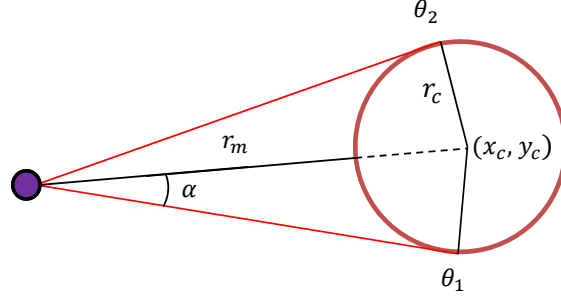
### Fitting circle using bounding angle

Using bounding angle to estimate the tree position and radius was first introduced by [6], which is illustrated as Fig. 4.7(a). Suppose a cluster consists of points spanning from angle  $\theta_1$  to  $\theta_2$  with the minimum range  $r_m$ . The circle lies at  $(x_c, y_c)$  with a diameter of  $r_c$ . According to simple trigonometry,

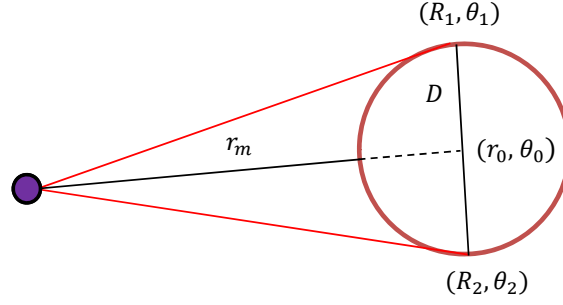
$$\sin \alpha = \frac{r_c}{r_c + r_m}, \quad (4.6)$$

where  $\alpha = \frac{\theta_2 - \theta_1}{2}$ . With  $\alpha$  and  $r_m$  calculated for each cluster, the circle radius is,

$$r_c = \frac{r_m \sin \alpha}{1 - \sin \alpha}. \quad (4.7)$$



(a) Bounding angle fitting method 1



(b) Bounding angle fitting method 2

Figure 4.7: Two circle fitting methods using the bounding angle of clusters.

Then in polar coordinate, the center of origin can be expressed as,

$$\text{Circle B}_1 : \begin{cases} r_p = r_m + r_c = \frac{r_m}{1 - \sin \alpha} \\ \theta_p = \frac{\theta_2 + \theta_1}{2} \end{cases} \quad (4.8)$$

Another fitting method using bounding angle is shown in Fig. 4.7(b), which estimates the diameter of the circle first instead of the radius,

$$\text{Circle B}_2 : \begin{cases} D = \frac{(R_1 + R_2)(\theta_1 - \theta_2 + \beta)}{2} \\ r_0 = r_m + 0.5D \\ \theta_0 = \frac{\theta_1 + \theta_2}{2} \end{cases} \quad (4.9)$$

where  $D$  is the circle diameter,  $(r_0, \theta_0)$  is the polar coordinate of the circle center, and  $\beta$  is the minimum angle resolution of the laser range finder (0.25 deg).

The above three methods produce similar estimated circle center and radius in the cases when the number of measurement points are sufficient and they are of an arc shape. However, in practice, the two conditions cannot be always met due to the influence of noise and the small size of the circles relative to the laser range finder. Noisy

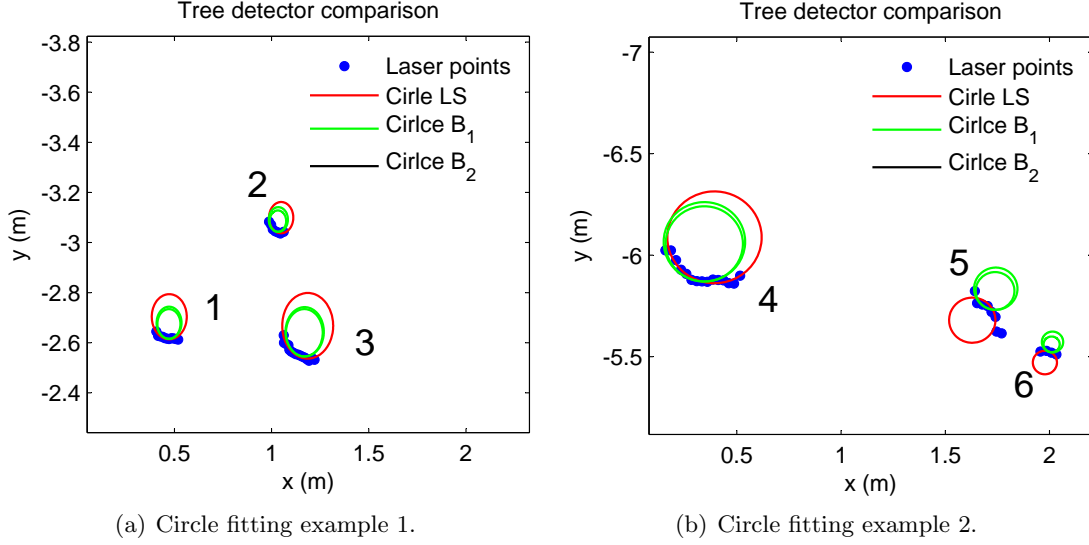


Figure 4.8: Comparison of three circle fitting algorithms.

measurement points may cause the circular arc shape to turn into a line shape or even an opposite direction arc. It is difficult to estimate a circle from the insufficient measurement points if the tree is far from the UAV or the trees are too small. Fig. 4.8 compares three methods discussed above, which are denoted as ‘Cirlice LS’, ‘Circle B<sub>1</sub>’ and ‘Circle B<sub>2</sub>’ respectively. Fig. 4.8(a) shows that for clusters with enough number of points, ‘Cirlice LS’ produces the largest circles while the other two methods using bounding angle are more conservative. Referring to Fig. 4.8(b), the estimated circle centers of 5 and 6 flip to the opposite side for ‘Cirlice LS’. The sensitivity to noise limits its application in practice even though it is the most accurate in mathematical sense. The bounding angle methods produce similar results in all cases with the ‘Circle B<sub>1</sub>’ a bit more conservative than ‘Circle B<sub>2</sub>’. Therefore, we use ‘Circle B<sub>1</sub>’ method to estimate the cirlice center and radius in this study.

#### 4.2.5 Feature Extraction Result

The raw laser scanner data is recorded in a small forest where the tree trunks distribute very densely with minimum tree-to-tree distance of 0.3 m. The diameters of three trunks at the flight height range from 0.15 m to 0.5 m. The quadrotor equipped with the LRF is manually piloted through the area while the onboard avionic system records both the laser range data and the IMU measurement. One raw laser scan is shown in Fig. 4.9, in which the initial segmentation process has created 62 segments. Not all the segments

correspond to tree stems, such as the segments (23-44) in the upright corner of Fig. 4.9. They might come from the measurement of ground plane when the platform is pitched for more than a certain angle. All scan segments are fed to the feature extraction procedure using the thresholds listed in Table 4.2. Fig. 4.10 shows the 19 extracted features with their radius and origin plotted. Fig. 4.11 zooms into three adjacent trees to show the performance of the circle fitting.

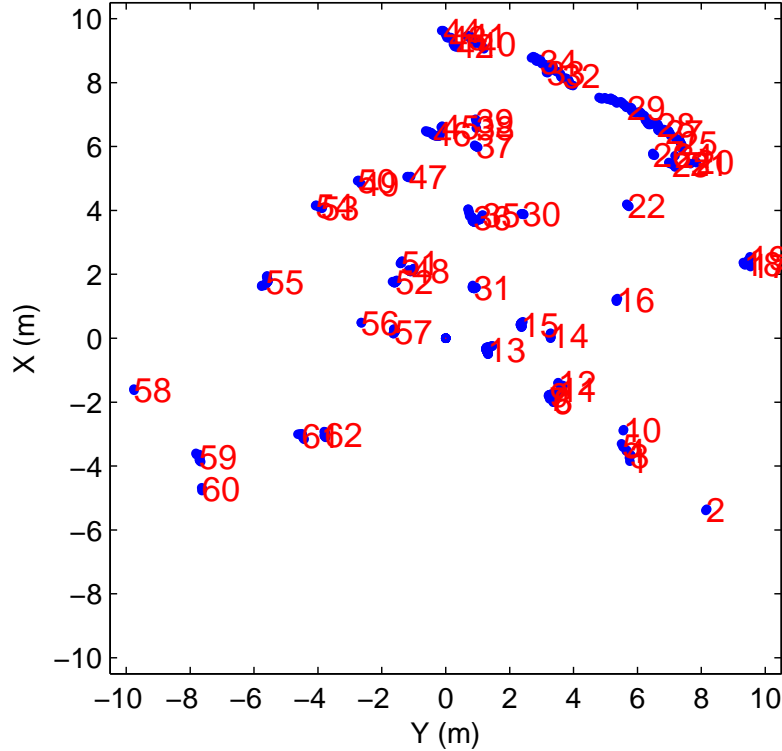


Figure 4.9: One raw scan with the labeled clusters.

### 4.3 Scan Matching

Various scan matching methods have been presented in literature. Algorithms based on ICP, with a wide range of variants[61], are most widely employed. The variants differ in such aspects like selection of points, matching strategy, error metric selection, etc. For each specific application, special modifications are needed to produce acceptable results. This section presents a specially designed ICP algorithm for UAV navigation in forests.

#### 4.3.1 Iterative Closest Point Matching

The ICP algorithm seeks to extract iteratively the rigid transformation (rotation and translation) from a group of measurement points to another group of reference points [4].

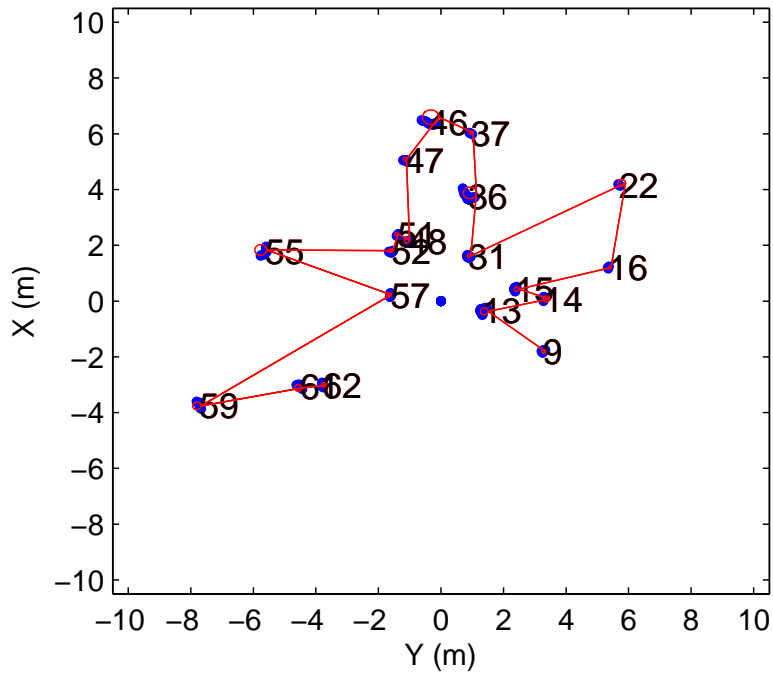


Figure 4.10: Clean scan with extracted circles.

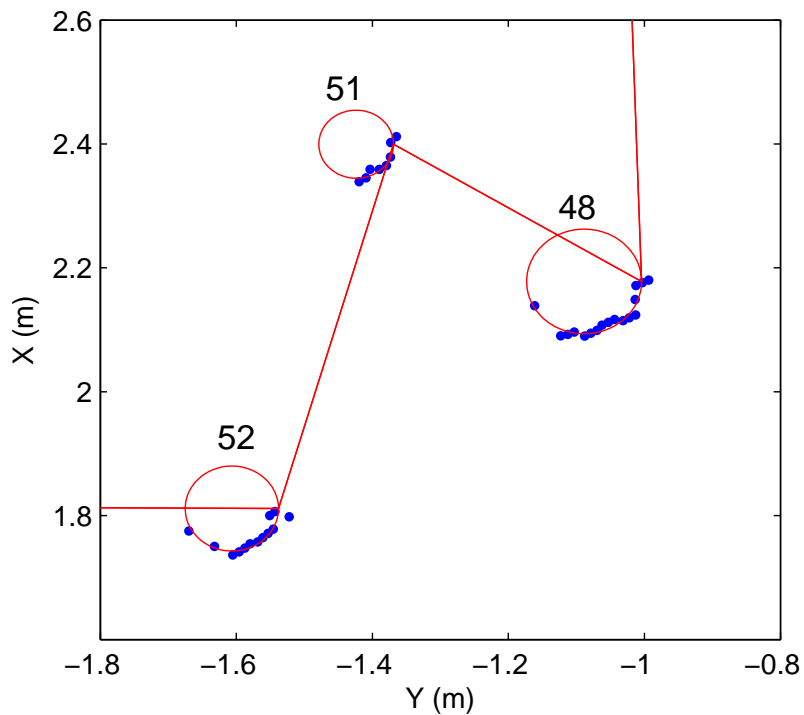


Figure 4.11: Close view of three extracted circles.

Fig. 4.12 presents the basic procedures of ICP. With two sets of input points, an error function is selected, which is usually the sum of the squared errors. The goal of the algorithm is to minimize the error function so that the optimal rotation and translation are obtained. First, the measurement points sets are transformed using an initial guess of the rotation and translation. Then a data association process (nearest neighbor search)

is employed to find pairs of corresponding points in the two data sets. The error function is then calculated and checked against a predefined threshold. If the error is smaller than the threshold, the iteration stops with the optimal transformation obtained. If not, the algorithm repeats again until the error is smaller than the threshold or the number of iterations is reached.

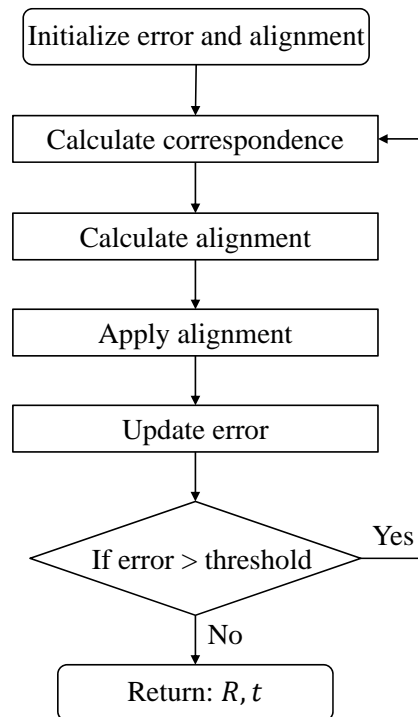


Figure 4.12: Procedures of the ICP algorithm.

### 4.3.2 Data Association

The correspondence searching aims to find the matching pairs according to a predefined distance metric (Euclidean or Mahalanobis distance). The searching algorithm is different with respect to different strategies in the selection of points: raw measurement or feature-based. For raw measurement points, searching a nearest neighbor in another dataset is often too intensive to use brutal force search. A  $k$ -dimensional ( $k$ -d) tree can accelerate the search process, with the extra cost of building up the  $k$ -d tree. Using raw measurement has another disadvantage: the nearest neighboring points in a pair may not correspond to the same physical object. This is due to the fact that the sensor measurement is always discrete. On the other hand, using features for data association reduces computation in searching dramatically at the expense of extra process for feature

extraction. Features can be points, lines or surfaces, etc. The inter-distance among the features are normally large enough so that the data association always produces correct correspondence pairs. Besides, the feature descriptors for different features also help reduce the possibility of wrong data association. Therefore, feature-based data association is preferable when the feature extraction process is not computationally intensive.

Determining whether two points are matched to each other is another crucial problem. Two strategies have been evaluated in this thesis: the closest point search and the limited range closest point search. The closest point search is to search all the points in another dataset given a point in the current dataset. This may be time consuming if the size of the dataset is large. Also the closest point may not be the correct correspondence, because sometimes the two datasets coincide with each other. The two points in the interconnection of the two datasets will certainly be the closest, but they are not the same points at all. The limited range closest point search can solve this problem, by limiting the angular searching range only in the vicinity of the current points. The introduction of searching range increases the computation speed and the possibility of the correct data association.



Figure 4.13: Initial transformed synthetic data.

To compare the two data association strategies, a dataset with 50 random points are rotated -5 degrees first and then translated by 5 m in x and y directions. The two initial datasets are shown in Fig. 4.13. The optimal transformation is estimated based on the

procedures in Fig. 4.12. The residual sum of squares is calculated at each step. The trend of the error function is plotted in Fig. 4.14. Both matching strategies can provide correct transformation but the limited range closest point searching strategy takes 2 iterations less than that of the normal closest point. The extent of speed improvement may be different for various datasets, but the improvement behavior is always observed. Fig. 4.15 shows the aligned datasets by projecting the transformed data back to the initial dataset. It can be seen that the extracted transformation is perfect when the data association is correct.

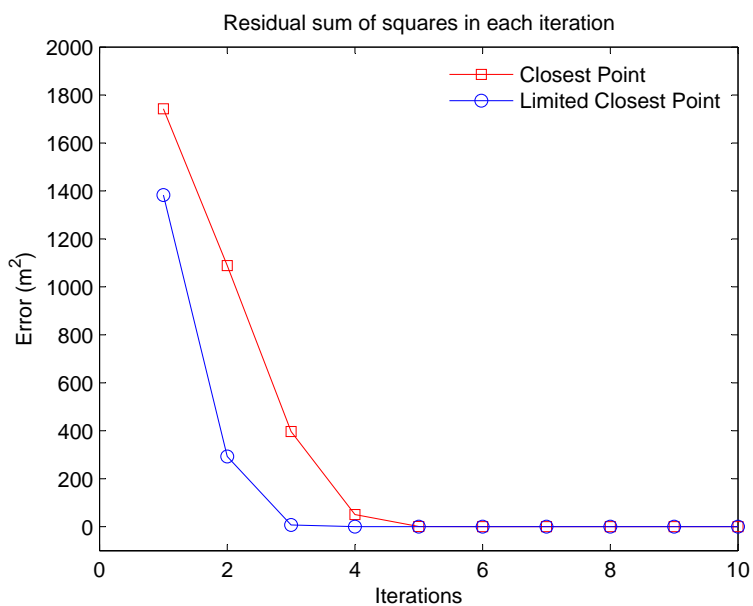


Figure 4.14: Change of error in each iteration.

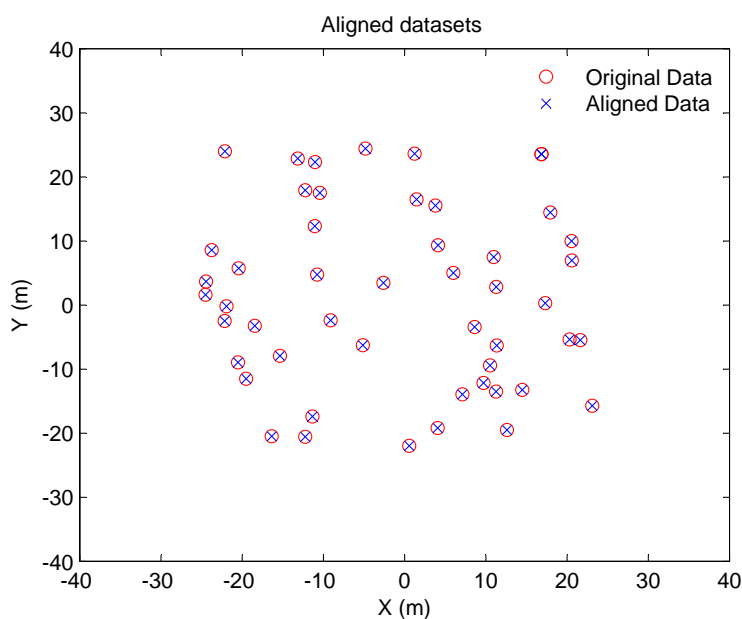


Figure 4.15: The aligned datasets after ICP.



In forest, the laser scanner have detected the tree trunks as described in Section 4.2. Each tree trunk is identified by its 2D central location and diameter, thus the features could be regarded as point features. The sparsity of tree trunks suggests that the relative distance between trees are above a certain threshold. The data association is implemented as a limited range nearest neighbor search discussed in this section.

### 4.3.3 Rigid Transformation Estimation

#### General 3D Rigid Transformation Estimation

With the correspondence pairs obtained, the alignment calculation procedure tries to calculate the optimal transformation. A wide range of methods have been developed and reviewed in [62]. These techniques are either iterative or closed form. The iterative methods have a series of problems: no convergence guarantee, local minima of the error function or the strict requirement of the initial estimate. Closed form solutions are preferable to iterative methods in terms of efficiency and robustness. Eggert et al.[23] compared four closed form algorithms for 3D rigid motion estimation and concluded that singular value decomposition (SVD) method [4] is most stable. Details of the mathematic derivation of the 3D transformation estimation using SVD is covered in [71].

For the purpose of comprehensiveness, the main steps of SVD-based 3-D transformation estimation are summarized in this section. Let  $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$  and  $\mathbf{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n\}$  be two sets of data points whose correspondence has been figured out. A rigid body transformation is found so that the weighted residual sum of squares as in Eq. 4.10 is minimized,

$$F(\mathbf{R}, \mathbf{t}) = \operatorname{argmin} \sum_{i=1}^n w_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i\|^2, \quad (4.10)$$

where  $w_i > 0$  is a weighting factor for each matched pair of points, which is chosen to be 1 in practice for data from the laser range finder. The following calculation steps are based on the fact that the optimal transformation of the weighted centroid of  $\mathbf{P}$  is equal to the weighted centroid of  $\mathbf{Q}$ .

1. Calculate the weighted centroid of the two datasets:

$$\bar{\mathbf{p}} = \frac{\sum_{i=1}^n w_i \mathbf{p}_i}{\sum_{i=1}^n w_i}, \quad \bar{\mathbf{q}} = \frac{\sum_{i=1}^n w_i \mathbf{q}_i}{\sum_{i=1}^n w_i}, \quad (4.11)$$

2. Remove the weighted centroid from the original datasets and generate two new datasets:  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  and  $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$ , where,

$$\mathbf{x}_i := \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{y}_i := \mathbf{q}_i - \bar{\mathbf{q}}. \quad (4.12)$$

3. Compute the  $3 \times 3$  covariance matrix,

$$\mathbf{S} = \mathbf{X}\mathbf{W}\mathbf{Y}^T, \quad (4.13)$$

where  $\mathbf{X}$  and  $\mathbf{Y}$  are the  $3 \times n$  matrices which have  $\mathbf{x}_i$  and  $\mathbf{y}_i$  as their columns, respectively, and  $\mathbf{W} = \text{diag}(w_1, w_2, \dots, w_n)$ .

4. Compute the singular value decomposition  $\mathbf{S} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ . The rotation matrix can be obtained as,

$$\mathbf{R} = \mathbf{V} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{V}\mathbf{U}^T) \end{bmatrix} \mathbf{U}^T. \quad (4.14)$$

5. The optimal translation is expressed as

$$\mathbf{t} = \bar{\mathbf{q}} - \mathbf{R}\bar{\mathbf{p}}. \quad (4.15)$$

### Closed-form Solution for 2D Point-based Matching

For 2D point-based features, if their correspondence has been specified, the relative translation and rotation could be derived in closed form [47]. Closed form solution is preferable since SVD operation is not needed so that onboard matrix inverse is avoided. Set the number of matched points to be  $n$ , and  $P(x_i, y_i)$ ,  $Q(x'_i, y'_i)$  represents the feature point in the two consecutive scans.

By minimizing the distance function in Eq. 4.10, the closed form solution for rotation angle  $\omega$  and translation  $(T_x, T_y)$  are given as follows:

$$\omega = \arctan \frac{S_{xy'} - S_{yx'}}{S_{xx'} + S_{yy'}} \quad (4.16)$$

$$T_x = \bar{x}' - (\bar{x} \cos \omega - \bar{y} \sin \omega) \quad (4.17)$$

$$T_y = \bar{y}' - (\bar{x} \sin \omega + \bar{y} \cos \omega) \quad (4.18)$$

where

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i \quad (4.19)$$

$$\bar{x}' = \frac{1}{n} \sum_{i=1}^n x'_i, \quad \bar{y}' = \frac{1}{n} \sum_{i=1}^n y'_i \quad (4.20)$$

$$S_{xx'} = \sum_{i=1}^n (x_i - \bar{x})(x'_i - \bar{x}'), \quad S_{yy'} = \sum_{i=1}^n (y_i - \bar{y})(y'_i - \bar{y}') \quad (4.21)$$

$$S_{xy'} = \sum_{i=1}^n (x_i - \bar{x})(y'_i - \bar{y}'), \quad S_{yx'} = \sum_{i=1}^n (y_i - \bar{y})(x'_i - \bar{x}') \quad (4.22)$$

#### 4.3.4 Experiment Evaluation

In order to validate the scan matching algorithms and test the customized assumptions, real flight tests have been carried out to collect data. The testing scenario is shown in Fig. 4.16 where a group of man-made trees are placed together to form a small synthetic forest. The trees possess ideal circular outer shapes, producing accurate circle parameters in the feature extraction process. Pillars with perpendicular corners and walls are also present in the scenario. The onboard laser scanner scans the environment and obtains measurement points, corresponding to trees, walls, and pillars. The feature extraction presented in Section 4.2 has solved this problem.



Figure 4.16: The indoor test scenario for verifying scan matching.

In the test, the UAV was piloted through the forest, following a close-to-rectangle shape of trajectory. The laser measurement data were recorded online and processed

offline using the scan matching method developed in this section. At the first scan, the UAV position is set to origin where the translation and rotation are set to zero. Then the incremental translation and rotation are obtained by matching each scan with the last scan. Accumulating these transformation increments produces a position and a heading estimates of the UAV in the horizontal plane. These estimates are also referred to as pseudo-absolute estimates since they are integrated from the increments. With pseudo-absolute pose estimated, each measured scan is projected to the same frame using the pose. Fig. 4.17 and Fig. 4.18 show the projected map and trajectory. The blue dots are the trajectories of the UAV while the red plots are the accumulated plots of the laser scans. Fig. 4.17 shows the accumulated path and map for the first 100 scans, in which the clear circular contour of trees can be seen. Fig. 4.18 shows the path and map at the end of the path after 280 scans. Comparing Fig. 4.17 and Fig. 4.18, the map remains consistent after 280 scans, corresponding to 28 seconds in time. This demonstrates that the scan matching algorithm can provide usable estimate for a short time. Meanwhile, we notice that the pose estimates have larger drifts at the end than that of the beginning, since the red contours of trees in Fig. 4.18 are more mixed up than the ones in Fig. 4.17. This drifting issues determine that scan matching can only be used for short time navigations. More advanced algorithms need to be developed to solve the drift issue. We will present our solution in the next chapter using the GraphSLAM.

Fig. 4.19 shows the estimated velocity in x and y directions and the incremental heading estimate. The UAV is originally piloted in the forest at a reasonably fast speed. In the first two subplots, the maximum velocity in both directions do not exceed 1 m/s, which indicates the speed is a safe flight speed for UAV operations in obstacle-strewn environment. Since the onboard laser scanner updates at 10 Hz, the incremental translation is below 0.1 m, setting an upper bound for incremental translation. From the third subplot, the incremental yaw angle is below 2 degrees, setting the upper limit of rotation angle. These translation and rotation bounds set the range to search for nearest neighbor in the data association process. In practice, the two bounding values are inflated two times to account for the effects of noise.

Fig. 4.19 also shows that the velocities estimated directly from scan matching are not smooth. For example, the x velocity at time 14.5 seconds exhibits a sudden jump. The jumps render the velocity estimate not directly applicable to onboard control. A

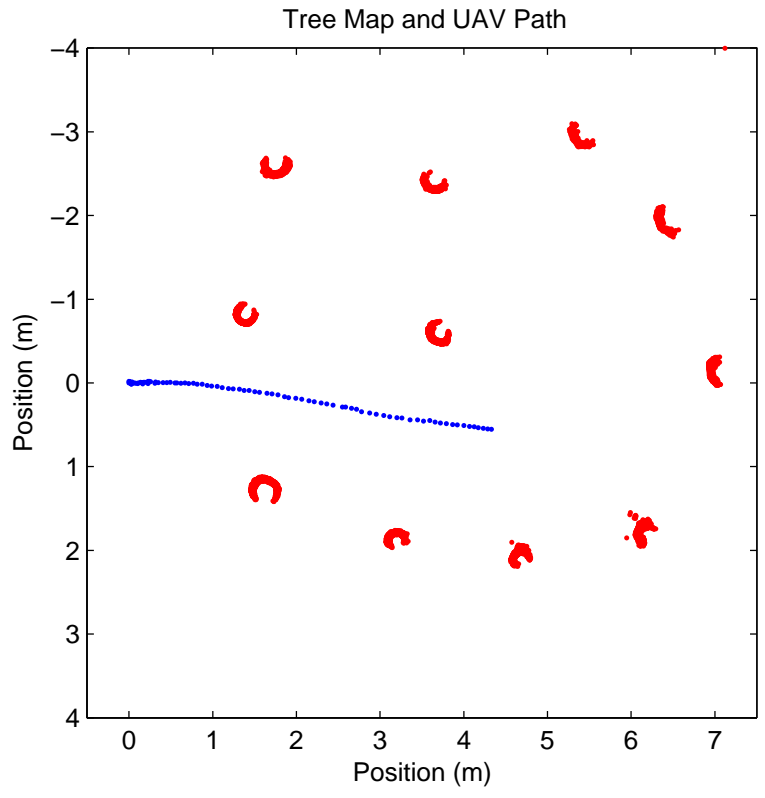


Figure 4.17: Motion and path estimate at the start of path.

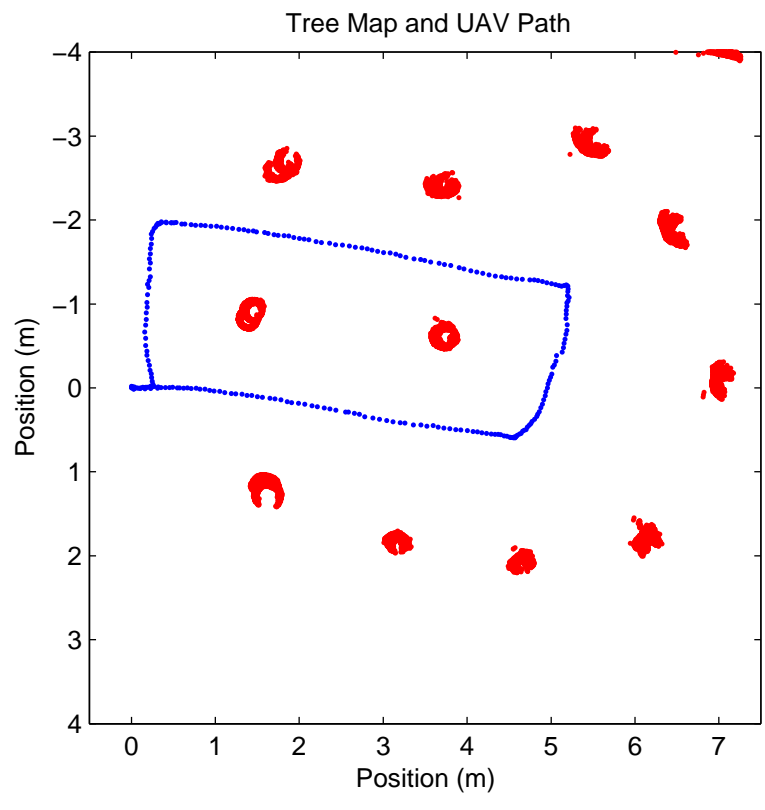


Figure 4.18: Motion and path estimate at the end of path.

smoothing mechanism needs to be implemented before feeding them to the control loop. Two popular solutions are considered: a low-pass filter or a Kalman filter. We choose to use the Kalman filter because it can not only smooth the velocity measurement, but also increase the estimate accuracy by fusing multiple sensor information.

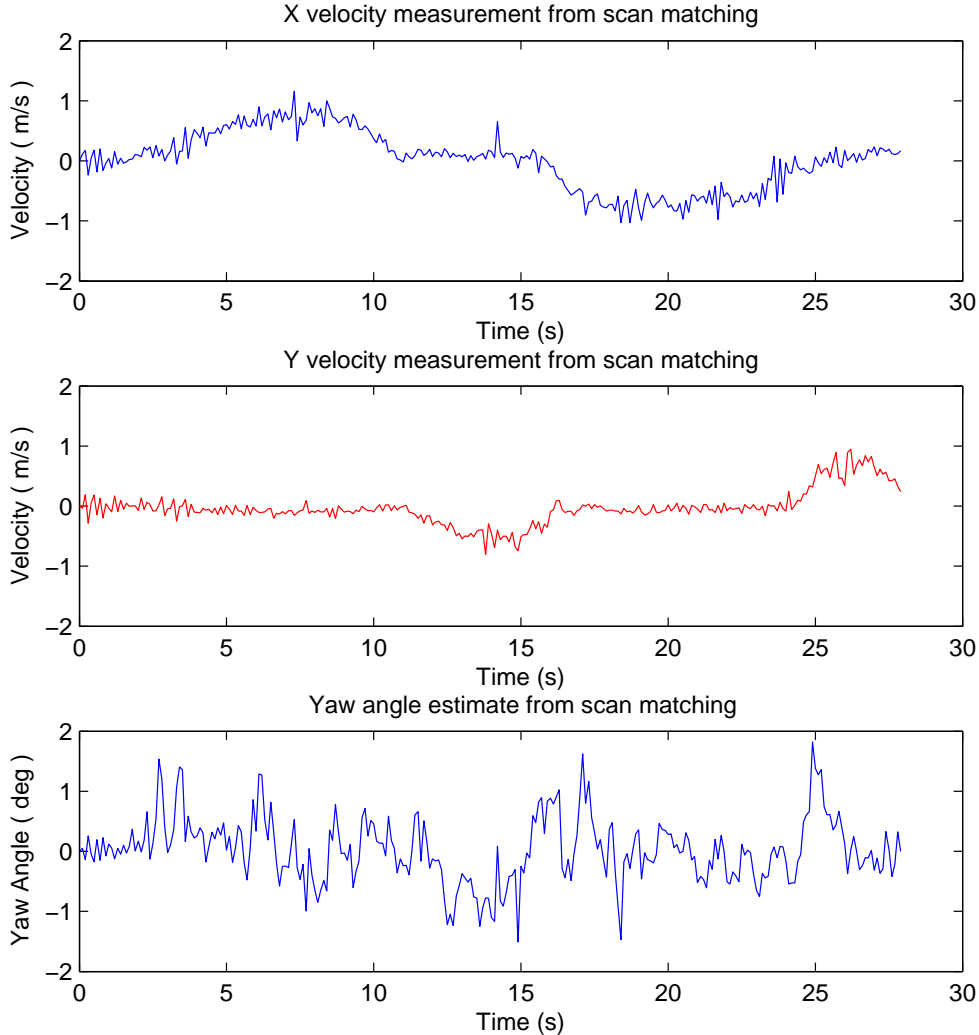


Figure 4.19: Velocity and incremental heading angle estimates from scan matching.

## 4.4 IMU-driven State Estimation

Kalman filter is one of the most popular filters to fuse measurement from multiple sensors. In this study, we propose a Kalman filter to fuse the velocity measurement from scan matching and the acceleration measurement from the IMU. In the current filter design, the orientation measurements from the IMU are assumed to be accurate enough and need no further filtering. Initially, the acceleration and incremental translation are all represented in the body frame. Using the orientation from the IMU, they could

be transformed to the local NED frame. The global acceleration measurements are considered as the driving force for the process model, thus we call it an IMU-driven Kalman filter. The incremental velocities transformed to the local NED frame are used as the measurement input.

To design a Kalman filter, the process model and measurement model are first identified. Let  $\mathbf{P}_n$  be the 3-axis position,  $\mathbf{v}_n$  be the 3-axis velocity in local NED frame. The process model is described by,

$$\begin{bmatrix} \dot{\mathbf{P}}_n \\ \dot{\mathbf{v}}_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_n \\ \mathbf{R}_{n/b}(\mathbf{a}_b + \mathbf{w}_a) \end{bmatrix}, \quad (4.23)$$

where  $\mathbf{a}_b$  is the acceleration measurement,  $\mathbf{w}_a$  is the acceleration measurement noise vector with normal distribution,  $\mathbf{a}_b + \mathbf{w}_a$  is the IMU acceleration measurement in body-fixed frame, and  $\mathbf{R}_{n/b}$  is the rotation matrix from the body frame to the local NED frame which is expressed as:

$$\mathbf{R}_{n/b} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}, \quad (4.24)$$

where  $\phi$ ,  $\theta$  and  $\psi$  are the roll, pitch and yaw angle respectively with  $s_* = \sin(*)$ ,  $c_* = \cos(*)$ .

For the measurement model, the planar velocities and the vertical height form the measurement vector,

$$\mathbf{y} = (z_n, u_n, v_n)^T, \quad (4.25)$$

where  $z_n$  is the height measurement,  $u_n$  and  $v_n$  are the incremental velocity from scan matching which have been transformed to the local NED frame using Eq.4.24. The height measurement can be obtained from any range sensing modality, such as a barometer, a sonar and a laser range finder.

For implementation of the Kalman filter in computer, the discrete-time process model Eq.4.23 and the measurement model Eq.4.25 are discretized using zero-order-hold method as follows,

$$\mathbf{x}(k+1) = \mathbf{A} \mathbf{x}(k) + \mathbf{B}(\mathbf{a}_n(k) + \mathbf{w}_n(k)), \quad (4.26)$$

$$\mathbf{y}(k) = \mathbf{C} \mathbf{x}(k) + \mathbf{v}(k), \quad (4.27)$$

where  $\mathbf{x}$ ,  $\mathbf{y}$  are the state and measurement vector respectively,

$$\mathbf{x} = (x_n, y_n, z_n, u_n, v_n, w_n)^T, \quad (4.28)$$

$$\mathbf{y} = (z_n, u_n, v_n)^T. \quad (4.29)$$

The input vector  $(\mathbf{a}_n(k) + \mathbf{w}_n(k))$  is the acceleration sequence in the local NED frame.  $\mathbf{v}$  and  $\mathbf{w}_g$  are the measurement noise vector and process noise vector with normal distribution. The discrete system matrices  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}$  are determined by discretizing Eq. 4.23 and Eq. 4.25 with 50 Hz sampling frequency,

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0.02 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0.02 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.02 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0.0002 & 0 & 0 \\ 0 & 0.0002 & 0 \\ 0 & 0 & 0.0002 \\ 0.02 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0.02 \end{bmatrix}, \quad (4.30)$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \quad (4.31)$$

The following procedure is a standard Kalman filter process with alternate time update and measurement update [10]:

**Time Update:**

$$\hat{\mathbf{x}}_{k,k-1} = \mathbf{A} \hat{\mathbf{x}}_{k-1} + \mathbf{B} \mathbf{u}_{k-1}, \quad (4.32)$$

$$\mathbf{P}_{k,k-1} = \mathbf{A} \mathbf{P}_{k-1} \mathbf{A}^T + \mathbf{B} \mathbf{Q} \mathbf{B}^T. \quad (4.33)$$



### Measurement Update:

$$\mathbf{H}_k = \mathbf{P}_{k,k-1} \mathbf{C}^T (\mathbf{C} \mathbf{P}_{k,k-1} \mathbf{C}^T + \mathbf{R})^{-1}, \quad (4.34)$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k,k-1} + \mathbf{H}_k (\mathbf{y}_k - \mathbf{C} \hat{\mathbf{x}}_{k,k-1}), \quad (4.35)$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{H}_k \mathbf{C}) \mathbf{P}_{k,k-1}, \quad (4.36)$$

where  $\hat{\mathbf{x}}_{k,k-1}$  and  $\mathbf{P}_{k,k-1}$  are the *a priori* state estimate and error covariance at step  $k$ ,  $\hat{\mathbf{x}}_k$  and  $\mathbf{P}_k$  are the *a posteriori* state estimate and error covariance,  $\mathbf{H}_k$  is the gain to decide how much the measurement is to be trusted. The optimal gain  $\mathbf{H}_k$  in Eq.4.34 minimizes the *a posteriori* error covariance  $\mathbf{P}_k$ .  $\mathbf{I}$  is the identify matrix with proper size.  $\mathbf{Q}$  and  $\mathbf{R}$  are the process noise covariance and measurement noise covariance.

Tuning of the Kalman filter can be achieved by adjusting the relative diagonal components of  $\mathbf{Q}$  and  $\mathbf{R}$ . The initial  $\mathbf{Q}$  and  $\mathbf{R}$  are determined by processing the offline acceleration and incremental velocity measurements and take their covariance respectively. Then changing the relative weight of the process covariance  $\mathbf{Q}$  and the measurement covariance  $\mathbf{R}$  is able to tune the performance of the Kalman filter. If measurement covariance  $\mathbf{R}$  is fixed, increasing  $\mathbf{Q}$  causes the state estimate to believe more on the measurement update. Any abrupt jump in the measurement will be reflected in the state estimate. On the other hand, decreasing  $\mathbf{Q}$  causes the Kalman filter to believe more on the process prediction. Since the acceleration measurement is continuous, smaller  $\mathbf{Q}$  exhibits stronger smoothing effect.

One noteworthy point is that the measurement update rate is 10 Hz while the acceleration measurement updates at 50 Hz. When the measurement is not available, the state is updated using only the process model as Eq. 4.32 - 4.33.

To evaluate the performance of Kalman filter, we compare it with the other two methods: IMU dead reckoning (DR) and scan matching. Fig. 4.20 shows the state estimation results of three methods on the same set of measurement data. The UAV was manually piloted in a forest as shown in Fig. 4.21 during which the onboard program recorded the acceleration and the laser scans. For the IMU dead reckoning method, the accelerations are transformed to the local NED frame and integrated twice to get the position estimate. The black solid line in Fig. 4.20 represents the path from IMU DR, whose path traverses the figure border quickly and drifts away. For the scan matching

method, feature extraction and scan matching were applied to the laser scans, providing the incremental translation and rotation estimates. The incremental translation are divided by the time difference between two consecutive scans to get the velocity. Using the IMU measurement of orientation, the incremental translation are transformed to local NED frame and integrated once to get the position estimate. The blue dash-dot line in Fig. 4.20 presents the path from scan matching. The third method, Kalman filter, fuses the IMU acceleration and the velocity derived from the scan matching, whose path is represented by red dashed line in 4.20. Comparing the paths from the three methods, it is clear to see that the scan matching and Kalman filter can significantly reduce the drift of position estimate of dead reckoning. The path from the Kalman filter resembles that of the scan matching because we choose to believe more on the measurement from scan matching. In the testing scenario, GPS signal is blocked by the dense tree canopies. We can not compare our estimated path with an external position reference.

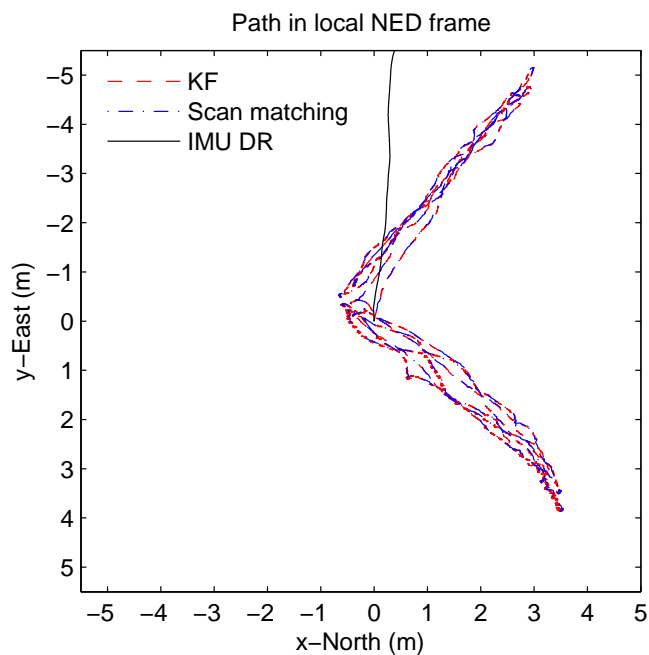


Figure 4.20: Comparison between dead reckoning, scan matching and Kalman filter.

## 4.5 Autonomous Flight Test

We have developed the techniques for UAV state estimation using a laser range finder in this chapter's previous sections. They are evaluated either with synthetic data or with practical data collected during manual flight. The fidelity of the state estimation

scheme cannot be justified without real flight test. We performed a series of autonomous flight test in a small forest (Fig. 4.21), which is in front of central library in National University of Singapore. To isolate the obstacle avoidance issues from the system, we designed a trajectory which does not collide with any objects in the environment. But the state estimation framework does not rely on the absolute position of the trees in the environment. The predefined trajectory was loaded into the system during power-up. Upon a single command on the ground control station, the UAV autonomously took off and started to travel in the forest following the predefined trajectory. At the end of the trajectory, the UAV landed autonomously in the original taken-off position. In the whole autonomous flight, the motion estimation scheme provided the position and velocity estimates for the onboard robust and perfect tracking controller.



Figure 4.21: The testing scenario with the flying quadrotor.

Figure 4.22 shows the position tracking performance. In both x and y direction, the RPT controller can track the reference very well. The tracking error in x direction is below 0.2 m and 0.5 m in y direction. This might be caused by the different motion estimation accuracy in x and y direction. Fig. 4.23 shows the reference position trajectory compared with the onboard estimates.

## 4.6 Conclusion

In this chapter, we have presented the state estimation of UAV for navigation in GPS-denied environment using laser odometry. The state estimation consists of a feature-

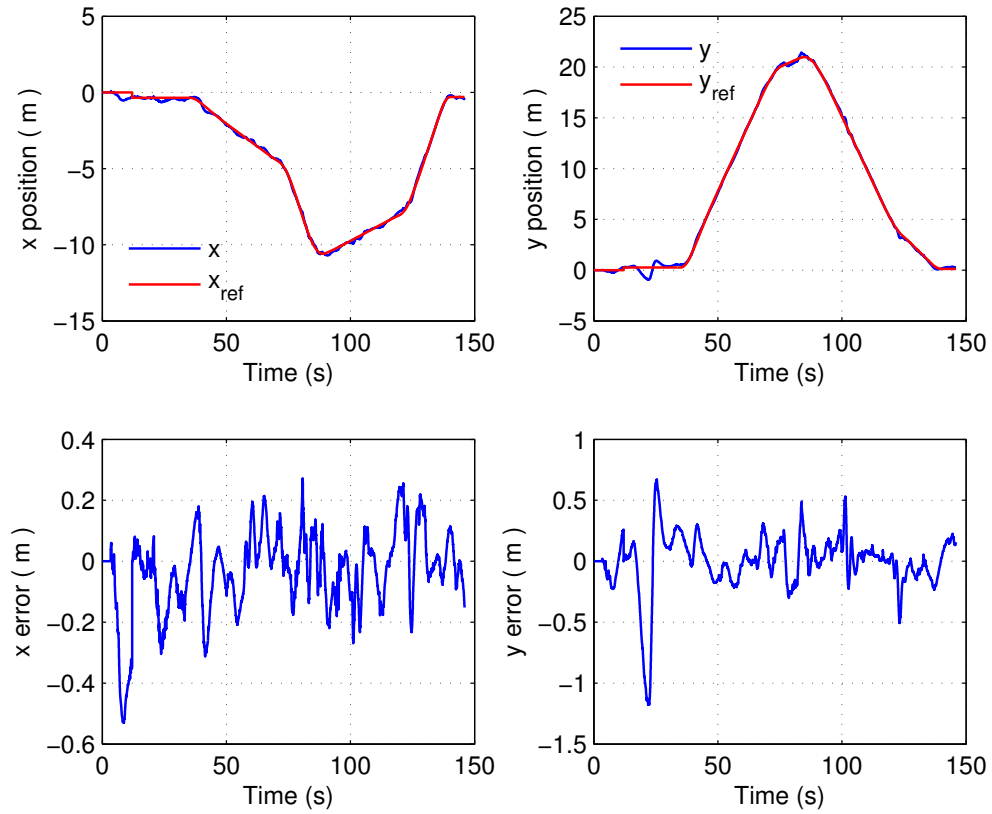


Figure 4.22: Position tracking in x-y plane with the tracking error.

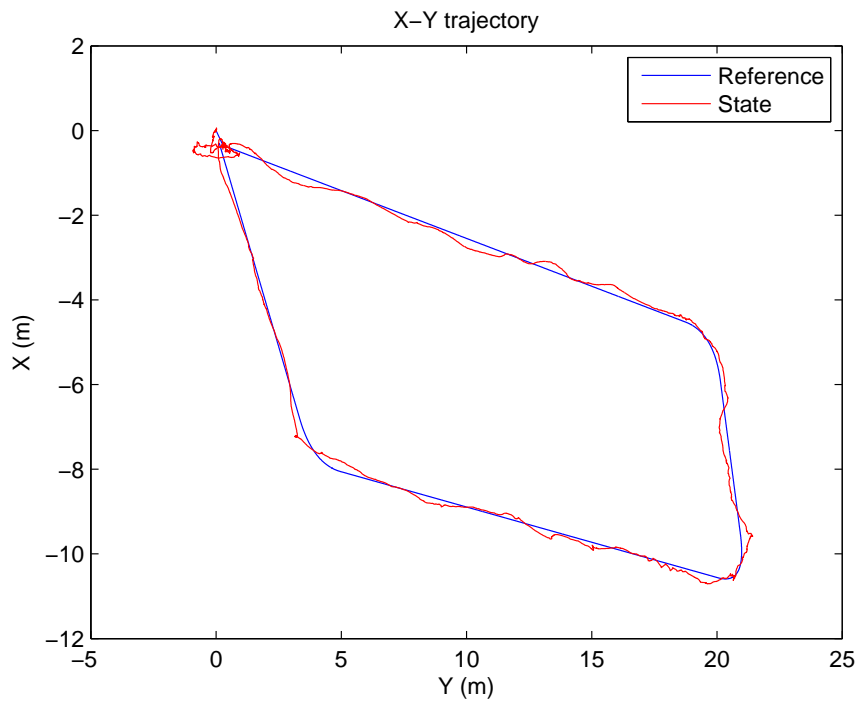


Figure 4.23: Position reference tracking in x-y plane.

based scan matching and a Kalman filter. We performed the flight tests in forest to evaluate the state estimation performance. On the one hand, tree trunks in forests are

extracted as features for scan matching. The range scan from the measurement of the laser range finder is first segmented using a carefully tuned threshold, separating each scan into a group of candidate clusters. The clusters are characterized with a series of geometric descriptors, which effectively distinguish the correct tree trunks from other objects like ground and bushes. The extracted tree centers serve as the point features for the closed-form scan matching, producing the measurements for incremental translation and rotation. On the other hand, a Kalman filter is designed to fuse the acceleration measurements of IMU with the velocity estimates from scan matching, providing 50 Hz state estimate for the real-time onboard control. The state estimation using laser odometry is verified by successful autonomous flights in foliage environments. The estimation framework is, however, not confined to laser odometry. Any other sensing modalities, like visual odometry, can be easily adopted in this framework.

## Chapter 5

# Offline Consistent Localization and Mapping using GraphSLAM

### 5.1 Introduction

We have presented the autonomous flights of UAV in GPS-denied environment in the last chapter. The scan matching method has been used with the Kalman filter to provide real-time state estimates for the flight tests. The measurements from laser odometry are incremental velocities, thus the position estimates from the Kalman filter are not observable, which are consequently prone to drift. The drift is acceptable for short range navigation in a small scale environment, but it is not applicable for long range navigation. SLAM technologies are often adopted to reduce the drift of position and achieve consistent mapping of the environment.

In this chapter, we present a consistent estimation framework based on the GraphSLAM technique. We decompose the framework into a front-end and a back-end. The front-end is responsible for interpreting the sensor data to build a graph, and the back-end is designed to optimize the graph for consistent mapping. We will list the procedures to build the graph based on the measurement of a laser range finder. The mathematics formulation of GraphSLAM will also be covered, which is essentially a nonlinear least squares problem. A standard Gauss-Newton method is illustrated to solve the nonlinear least squares problem. Implementation issues of the whole framework in Matlab and C++ will be discussed, with evaluation results given in the end of this chapter.

## 5.2 GraphSLAM System Structure

The standard GraphSLAM algorithms frequently discuss about different solvers to solve the nonlinear least square problem. The techniques of building up the constraints are rarely covered. A complete procedure including the process of building up the graph and optimizing the graph is indispensable to practical UAV navigation using GraphSLAM techniques. Typical GraphSLAM structure is presented in Fig. 5.1 [39], consisting of the front-end and the back-end. The front-end aims to obtain the constraints information based on the collected sensor data. The constraint is a link between two poses which describes how the two poses are similar to each other. It is related to the extent of overlap between the two measurements taken on the two poses. Therefore, the robust and accurate data association is critical to build the constraints. The back-end solves a nonlinear quadratic optimization problem, giving the optimal configuration of nodes that maximizing the likelihood of the measurements encoded in the constraints.

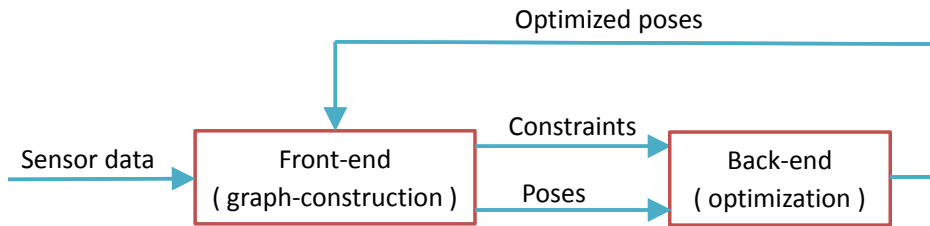


Figure 5.1: GraphSLAM system structure.

In Chapter 4, we have discussed about the state estimation using laser scanner in a Kalman filter framework. We present now a comprehensive and practical system structure from the front-end to the back-end in Fig. 5.2. The state estimated in the Kalman filter is prone to drift since only incremental velocities are measured from the scan matching instead of the absolute position. In the context of GraphSLAM, the drifting position serves as an initial guess, i.e, a node in the pose graph, which could be optimized in the back-end.

**Front-end:** this part processes the sensor information to produce the graph, including the initial pose and the pose constraints. First a motion estimation based on laser odometry is implemented from the consecutive measurement of the laser range finder. A Kalman filter is designed to fuse the sensor information from the IMU and the laser range finder. The acceleration measurement from the IMU serves as the input to the process model. The inputs to the measurement update are the velocity estimates

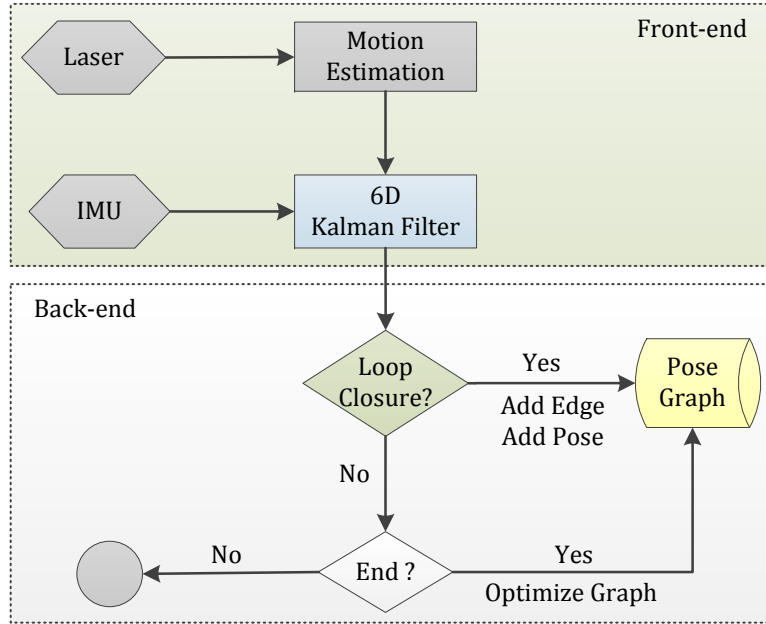


Figure 5.2: System schematics illustrating front-end and back-end.

from the motion estimation. The output of the Kalman filter is the estimated position and velocity which are used for autonomous control. The position estimate from the Kalman filter is only suitable for short time navigation. This is due to the fact that the planar position is not observable in the measurement equation and thus suffers from long term drift. Without external absolute position reference, the position estimate has to be bounded in another way. This is where the GraphSLAM back-end plays its part. Together with the features seen on the pose, the initial pose is fed into a loop closure detection block. The loop detection block builds constraints between the current pose and all the previous poses if sufficient measurement overlap is detected. The constraints are added to the graph for the future optimization. Detecting the correct overlap is the key step for loop detection. We use feature-based scan matching in this study because the tree features we extracted are quite robust during the data association.

**Back-end:** this part solves the nonlinear-least square problem to derive the optimal configuration of poses and landmarks. The nonlinear-least square problem is normally solved in an iterative manner: forming a linear system around the current state estimate, solving the linear system and iterating. The typical nonlinear optimization problem could be addressed with standard methods like Gauss-Newton, Levenberg-Marquardt (LM) [21] or variants of gradient descent algorithms are used to solve this problem.

In Fig. 5.2, we have put the loop closure detection module to the back-end. This



makes sense because we utilize the position estimates from the Kalman filter as the estimate for the real-time autonomous control of the UAV. By putting the loop detection into back-end, the GraphSLAM back-end operates in a self-contained manner which will not influence the real time performance of the UAV control. This modular design is preferable for system development and integration.

## 5.3 GraphSLAM Back-end

### 5.3.1 GraphSLAM Formulation

GraphSLAM is a full SLAM problem, which seeks to calculate a posterior solution for the offline SLAM defined over all poses and all features in the map. It aims to find the most consistent trajectory configuration which maximizes the measurement likelihood. The GraphSLAM uses a graph with nodes and edges to represent the information obtained by the UAV motion model and the measurement model. The graph consists of two types of nodes and two types edges respectively (Fig. 5.3). Nodes in the graph correspond to either the UAV poses or the feature positions. Edges in the graph represent the mutual constraints of the nodes. The edges in the graph are categorized into two types: one type is the edge describing the constraints between the UAV poses and the other type is the edges connecting the pose node with the sensed feature nodes at that pose. Each edge in the GraphSLAM is a non-linear quadratic constraint, where the motion constraints correspond to the motion model and the measurement constraints correspond to the measurement model.

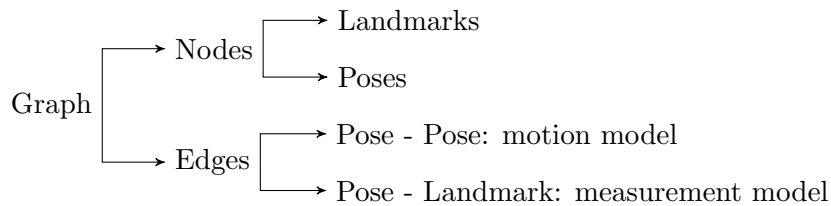


Figure 5.3: Composition of a graph.

For a linear system, the soft constraints are equivalent to entries in an information matrix  $\mathbf{\Omega}$  and an information vector  $\xi$ . Each measurement in  $\mathbf{z}_{1:t}$  and each control in  $\mathbf{u}_{1:t}$  lead to a local update of  $\mathbf{\Omega}$  and  $\xi$ . Given an environment  $m$  consisting of features  $m_j$ , a robot is driven by a set of controls  $\mathbf{u}_{1:t}$ , generating a set of poses  $\mathbf{x}_{1:t}$  and a set

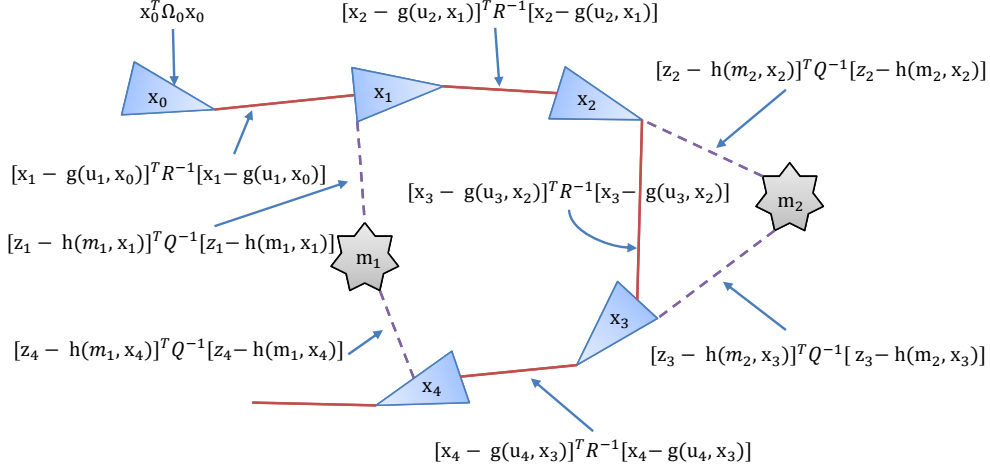


Figure 5.4: GraphSLAM illustration [74].

of measurement  $\mathbf{z}_{1:t}$  with associated correspondence variables  $c_{1:t}$ . Referring to Fig.5.4, the measurement  $\mathbf{z}_t^i$  provides information between the location of the feature  $j = c_t^i$  and the robot pose  $\mathbf{x}_t$  at time  $t$ . The measurement constraint is defined as the error between the real measurement and the predicted measurement weighted by the inverse of the covariance matrix, which is defined as follows,

$$(\mathbf{z}_t^i - h(\mathbf{x}_t, m_j))^T \mathbf{Q}_t^{-1} (\mathbf{z}_t^i - h(\mathbf{x}_t, m_j)), \quad (5.1)$$

where  $h$  is the measurement function and  $\mathbf{Q}_t$  is the covariance of the measurement noise.

For motion constraints, the control  $\mathbf{u}_t$  provides update the robot pose  $\mathbf{x}_{t-1}$  to  $\mathbf{x}_t$  using the robot kinematic model  $\hat{\mathbf{x}}_t = g(\mathbf{u}_t, \mathbf{x}_{t-1})$ . The relative error compared with the robot pose  $x_t$  forms the motion constraints,

$$(\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1}))^T \mathbf{R}_t^{-1} (\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1})), \quad (5.2)$$

where  $\mathbf{R}_t$  is the covariance of the motion noise.

After incorporating all measurements  $z_{1:t}$  and controls  $u_{1:t}$ , a sparse graph is obtained similar to the one in Fig. 5.4. The sum of constraints is defined as:

$$\begin{aligned} J_{GraphSLAM} &= \mathbf{x}_0^T \mathbf{\Omega}_0 \mathbf{x}_0 + \sum_t [\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1})]^T \mathbf{R}^{-1} [\mathbf{x}_t - g(\mathbf{u}_t, \mathbf{x}_{t-1})] \\ &+ \sum_t [\mathbf{z}_t - \mathbf{h}(m_{c_t}, \mathbf{x}_t)]^T \mathbf{Q}^{-1} [\mathbf{z}_t - \mathbf{h}(m_{c_t}, \mathbf{x}_t)]. \end{aligned} \quad (5.3)$$

Minimizing Eq. 5.3 means to maximize the measurement likelihood so that the robot poses on the trajectory are most consistent.

### 5.3.2 Loop Detection

The loop detection module searches over all the poses in the initial trajectory and connects those two poses with enough overlap. The initial poses are the position estimates of the Kalman filter. As shown in Fig. 5.5, the initial pose at time  $t$  is only related to the pose  $x_{t-1}$  one time step before. In the context of GraphSLAM, this corresponds to an initial graph consisting of edges between each pair of consecutive poses. In order to add more constraints, a pose  $\mathbf{x}_t$  at time  $t$  is compared with all the poses before  $t$  to check if an edge could be added. The edge built upon two poses which are far from each other in time is based on the fact that there are enough overlap of measurements in the two poses. For example, as the UAV travels a certain distance, it may revisit the same place which the UAV has visited before. At time  $t + 1$  the UAV observes the landmark  $m_1$  again. Based on the two measurements of  $m_1$  at the two poses  $\mathbf{x}_{t+1}$  and  $\mathbf{x}_1$ , a relative spatial configuration of  $x_1$  and pose  $x_{t+1}$  is obtained. If the number of revisited features is large, a strong spatial constraint between pose  $x_1$  and pose  $x_{t+1}$  can be built and added to the graph. This relates the pose  $x_{t+1}$  not only to the most recent previous pose  $x_t$ , but also to the poses along the trajectory. The multiple spatial constraints will in the end bound the current pose to the position which maximizes the likelihood of the measurements encoded in the constraints.

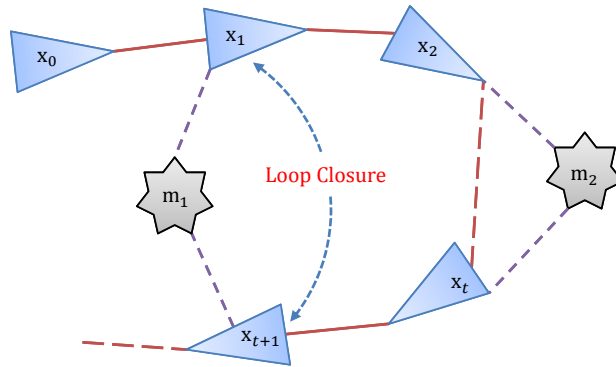


Figure 5.5: Loop closure after traveling a certain time.

The details of the loop detection are listed in Algo. 2. The main steps are to compare the current set of poses and measured features with all the previous poses and the features on the trajectory. If there are sufficient overlap of measurements, an edge is

built up between the two poses. Two issues need to be clarified further: the criteria for adding new edges and the search window. The first issue is a data association problem. The matching criteria is selected to be the sum of squares errors, which is defined as the sum of squares of the Euclidean distance between the matched features. Nearest neighbor search is first performed to generate a list of matched features. The matched features of one frame is then transformed to the other frame where a sum of Euclidean distance is taken. For any two nonconsecutive poses, if the error is smaller than the initial error provided by the consecutive scan matching, a new edge is added to the graph. An edge is described by the position error and the information matrix. The information matrix is essentially the inverse of the covariance matrix, describing the confidence level of the edges. If there are a large number of features matched between the two positions, the covariance matrix is small. Therefore, the information matrix is chosen to be a diagonal matrix with elements proportional to the number of matched features on the two poses.

---

**Algorithm 2:** GraphSLAM loop detection.

---

**Input:** Initial poses  $\mathbf{x}_i$ , features at each pose  $\mathbf{M}_i, i \in [1, n]$ ,  $n$  is number of poses.  
**Output:** Constraint set  $C = \langle e_{ij}(\cdot), \Omega_{ij} \rangle$

```

1 for  $i$  in  $\{1 : n\}$  do
2   for  $j$  in  $\{i + 1 : n\}$  do
3     Get the initial pose difference  $\mathbf{D}_{ij}$  between  $\mathbf{x}_i$  and  $\mathbf{x}_j$ ;
4     Calculate sum of squares error  $ssd1$  by projecting  $\mathbf{M}_j$  to the frame of  $\mathbf{x}_i$ 
      using  $\mathbf{D}_{ij}$ ;
5     Calculate the transformation  $\mathbf{T}_{ij}$  between  $\mathbf{x}_i$  and  $\mathbf{x}_j$  using scan matching;
6     Calculate sum of squares error  $ssd2$  by projecting  $\mathbf{M}_j$  to the frame of  $\mathbf{x}_i$ 
      using  $\mathbf{T}_{ij}$ ;
7     if  $ssd2 < ssd1$  then
8       Add the edge  $e_{ij}$  as the difference between  $\mathbf{D}_{ij}$  and  $\mathbf{T}_{ij}$ ;
9       Add the information matrix  $\Omega_{ij}$  to be diagonal matrix with elements
        proportional to the number of inliers during scan matching;
10    end
11  end
12 end
13 return Constraint set  $C = \langle e_{ij}(\cdot), \Omega_{ij} \rangle$ ;

```

---

The other issue is the search window size, which influences the speed and the performance of the back-end optimization. As shown in Fig. 5.6, there are two searching strategies: searching over all previous poses or searching only in a small time window. Both methods have their merits and drawbacks. For the global search, the loop detection block searches over all the previous poses along the trajectory and checks for

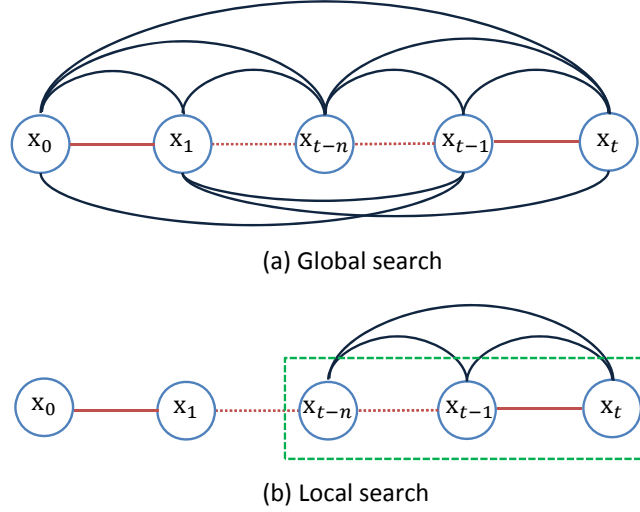
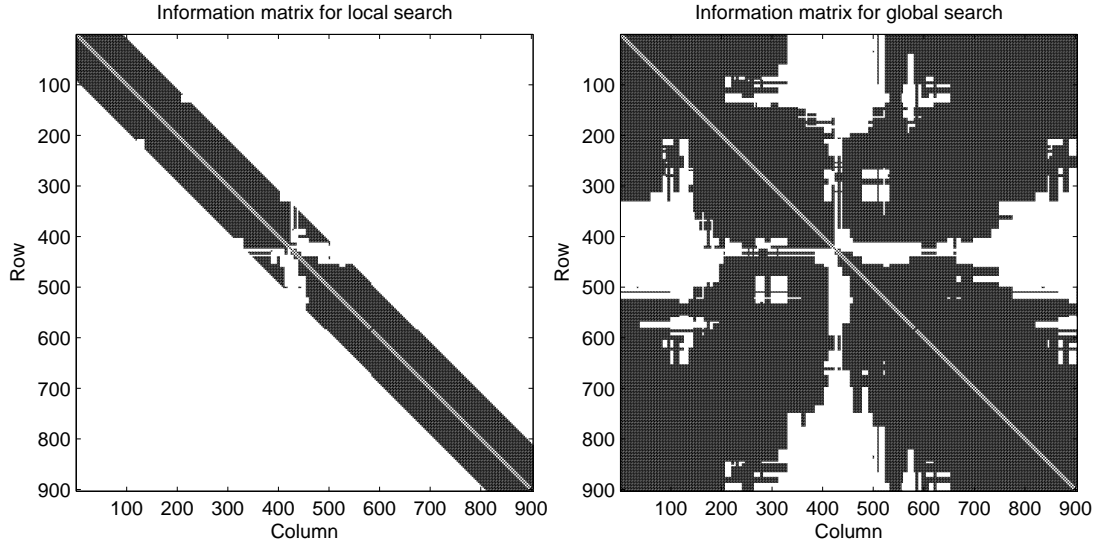


Figure 5.6: Global and local search in loop detection.

possible loop closure. This method incorporate the most comprehensive information collected on the trajectory and tends to generate the most accurate results. However, this exhaustive search will become prohibitively intensive after the UAV travels a long time, which becomes even more intensive if the sensor suites produce 3D feature points with high dimensional feature descriptors such as 3D LiDAR or stereo vision system. On the other hand, the local search strategy avoids the increasing size of graph at the expense of losing some information. A comparison of information matrix of the two searching methods for the same dataset is given in Fig. 5.7. Black squares in the figures correspond to edges between the two poses. The local searching method limits the search window to the most recent 30 poses, creating an information matrix with non-zero elements only close the main diagonal band (Fig. 5.7(a)). Fig. 5.7(b) shows that the global searching strategy produces a more dense matrix, indicating more edges are added. The non-zero off-diagonal elements of the information matrix indicate the existence of large loop closure between the beginning and end of the trajectory. Therefore, the size of search window is an important parameter to generate the graph. Large search window closes larger loops at the expense of longer running time but small search window size produces near real-time performance while losing some information. Experiment evaluation of the search window is covered in Section 5.4.3. Online GraphSLAM considering both the global and local search is covered in the next chapter.



(a) Information matrix for local search (30 poses)      (b) Information matrix for global search

Figure 5.7: Comparison of information matrix between local search and global search.

### 5.3.3 Graph Optimization

As discussed in [74], the information matrix for a graph with pose nodes and landmark nodes can be factorized to small information matrix, consisting of only poses nodes. The information between features and poses are shifted to the constraints between the two related poses. The resultant graph is a pose graph as illustrated in Fig. 5.8. The nodes are only the robot poses  $x_i$  and edges represent the spatial constraints between the two poses. Edges between two consecutive poses correspond to the odometry measurement, while the other nonconsecutive edges correspond to the spatial constraints arising from multiple observations of the same sets of features.

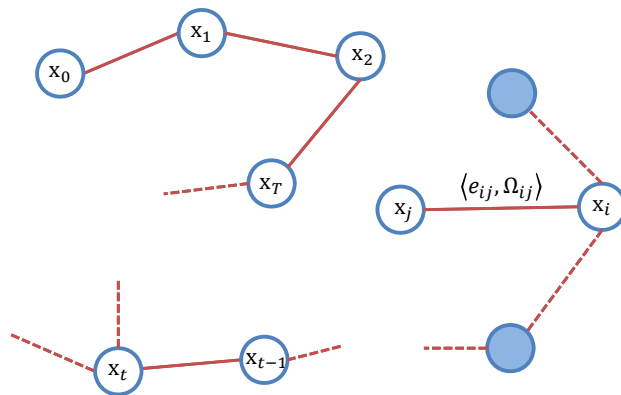


Figure 5.8: The pose-graph structure in GraphSLAM.

Let  $\mathbf{x} = (x_1, x_2, \dots, x_t)^T$  be the vector of poses,  $z_{ij}$  be the virtual measurement

which has been obtained in the factorization of  $\Omega$ ,  $\hat{\mathbf{z}}_{ij}$  be the prediction of the virtual measurement given the nodes  $x_i$  and  $x_j$ . We define the error,

$$e_{ij}(x_i, x_j) = z_{ij} - \hat{\mathbf{z}}_{ij}(x_i, x_j) \quad (5.4)$$

Let  $C$  be the set of pairs of indices for which a constraint  $z$  exists, the nonlinear optimization problem could be formulated as:

$$F(\mathbf{x}) = \sum_{(i,j) \in C} \underbrace{e_{ij}^T \Omega_{ij} e_{ij}}_{F_{ij}} \quad (5.5)$$

$$\mathbf{x}^* = \underset{x}{\operatorname{argmin}} F(\mathbf{x}). \quad (5.6)$$

Eq. 5.6 could be solved in many ways, either via the standard nonlinear least-square optimization, Gauss-Newton or the Levenberg-Marquardt algorithms. A good summary of solutions can be found in [49]. Gauss-Newton method is the most basic method upon which other methods are developed, thus we present the main procedures of solving Eq. 5.6 together with the sparse structure of the GraphSLAM formulation.

Given a reasonable initial guess of the robot pose  $\check{\mathbf{x}}$ , we illustrate the steps using Gauss-Newton algorithm to solve Eq. 5.6. First the error function is linearized at the current initial guess  $\check{\mathbf{x}}$ ,

$$e_{ij}(\check{\mathbf{x}}_i + \Delta x_i, \check{x}_j + \Delta x_j) = e_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x}) \quad (5.7)$$

$$\simeq e_{ij} + \mathbf{J}_{ij} \Delta \mathbf{x}. \quad (5.8)$$

here  $\mathbf{J}_{ij}$  is the Jacobian of  $e_{ij}$  computed at  $\check{\mathbf{x}}$ . Since the error  $e_{ij}$  depends only on the node  $\mathbf{x}_i$  and  $\mathbf{x}_j$ ,  $\mathbf{J}_{ij}$  has the following sparse form:

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \underbrace{\mathbf{A}_{ij}}_{\text{node } i} & \mathbf{0} & \cdots & \mathbf{0} & \underbrace{\mathbf{B}_{ij}}_{\text{node } j} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \quad (5.9)$$

$$F_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x}) = e_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x})^T \Omega_{ij} e_{ij}(\check{\mathbf{x}} + \Delta \mathbf{x}) \quad (5.10)$$

$$= (e_{ij} + \mathbf{J}_{ij} \Delta \mathbf{x})^T \Omega_{ij} (e_{ij} + \mathbf{J}_{ij} \Delta \mathbf{x}) \quad (5.11)$$

$$= \underbrace{e_{ij}^T \Omega_{ij} e_{ij}}_{c_{ij}} + 2 \underbrace{e_{ij}^T \Omega_{ij} \mathbf{J}_{ij}}_{b_{ij}} \Delta \mathbf{x} + \Delta \mathbf{x}^T \underbrace{\mathbf{J}_{ij}^T \Omega_{ij} \mathbf{J}_{ij}}_{H_{ij}} \Delta \mathbf{x} \quad (5.12)$$





---

**Algorithm 3: POSE GRAPH OPTIMIZATION USING GAUSS-NEWTON METHOD**

---

**Input:** Initial pose  $\check{\mathbf{x}} = \check{\mathbf{x}}_{1:T}$ , constraints  $C = \langle e_{ij}(\cdot), \Omega_{ij} \rangle$   
**Output:** Optimal pose  $\mathbf{x}^*$ , information matrix  $\mathbf{H}^*$

```
1 while not converged do
2    $\mathbf{b} \leftarrow 0, \mathbf{H} \leftarrow 0$ ;
3   for all constraints  $\langle e_{ij}(\cdot), \Omega_{ij} \rangle$  in  $C$  do
4     Calculate Jacobian at the current pose  $\check{\mathbf{x}}$  using Eq.5.9;
5     Update the local information matrix  $\mathbf{H}_{ij}$  at node  $x_i$  and  $x_j$  using Eq.5.20;
6     Update the local information vector  $\mathbf{b}_{ij}$  at node  $x_i$  and  $x_j$  using Eq.5.20;
7   end
8   Keep the first fixed by  $\mathbf{H}_{11+} = I$ ;
9   Solve the linear system with sparse Cholesky factorization,
    $\Delta \mathbf{x} \leftarrow \text{solve}(\mathbf{H} \Delta \mathbf{x} = -\mathbf{b})$ ;
10  Update the pose estimation  $\check{\mathbf{x}}_+ = \Delta \mathbf{x}$ ;
11 end
12  $\mathbf{x}^* \leftarrow \check{\mathbf{x}}, \mathbf{H}^* \leftarrow \mathbf{H}$ ;
13 return  $(\mathbf{H}^*, \mathbf{x}^*)$ ;
```

---

### 5.3.4 Error Linearization for 2D Poses

There are two types of errors in the GraphSLAM formulation: one is pose difference error and the other is the landmark measurement error. The position of landmark is expressed in 2D euclidean space. This type of error is implemented as the vector minus operation. For the pose error, the pose of UAV can be expressed as  $\mathbf{x}_i = (x_i, y_i, \theta_i)^T$  in 2D plane, where  $x_i$  and  $y_i$  are the positions in the global frame and  $\theta_i$  is the heading of the UAV. The pose parametrization belongs to the special Euclidean group SE(2). To facilitate motion composition, the pose is expressed in homogeneous coordinate. For a pose  $\mathbf{x}_i$ , its homogeneous coordinate is,

$$\mathbf{X}_i = \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \\ \mathbf{0} & 1 \end{bmatrix}, \quad (5.21)$$

where  $\mathbf{R}_i$  is a  $2 \times 2$  rotation matrix, and  $\mathbf{t}_i$  is a  $2 \times 1$  translation vector, which are expressed as

$$\mathbf{R}_i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \\ \sin \theta_i & \cos \theta_i \end{bmatrix}, \quad \mathbf{t}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}. \quad (5.22)$$

The inverse of homogeneous transform Eq.5.21 is,

$$\mathbf{X}_i^{-1} = \begin{bmatrix} \mathbf{R}_i^T & -\mathbf{R}_i^T \mathbf{t}_i \\ \mathbf{0} & 1 \end{bmatrix}. \quad (5.23)$$

Suppose there are two initial poses from the front-end,  $\mathbf{X}_i$  and  $\mathbf{X}_j$ , the initial pose difference is  $\mathbf{X}_i^{-1}\mathbf{X}_j$ . With the measurement between the two poses given as  $\mathbf{Z}_{ij} = (\mathbf{t}_{ij}, \theta_{ij})^T$ , the error function is

$$\mathbf{e}_{ij} = \mathbf{Z}_{ij}^{-1}(\mathbf{X}_i^{-1}\mathbf{X}_j) = \begin{bmatrix} \mathbf{R}_{ij}^T(\mathbf{R}_i^T(\mathbf{t}_j - \mathbf{t}_i) - \mathbf{t}_{ij}) \\ \theta_j - \theta_i - \theta_{ij} \end{bmatrix}. \quad (5.24)$$

We can derive the Jacobian matrix element  $\mathbf{A}_{ij}$  and  $\mathbf{B}_{ij}$  of the error function  $\mathbf{e}_{ij}$  at  $\mathbf{x}_i$  and  $\mathbf{x}_j$  as,

$$\mathbf{A}_{ij} = \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{x}_i} = \begin{bmatrix} -\mathbf{R}_{ij}^T \mathbf{R}_i^T & \mathbf{R}_{ij}^T \frac{\partial \mathbf{R}_i^T}{\partial \theta_i} (\mathbf{t}_j - \mathbf{t}_i) \\ \mathbf{0} & -1 \end{bmatrix}, \quad (5.25)$$

$$\mathbf{B}_{ij} = \frac{\partial \mathbf{e}_{ij}}{\partial \mathbf{x}_j} = \begin{bmatrix} \mathbf{R}_{ij}^T \mathbf{R}_i^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}. \quad (5.26)$$

## 5.4 Offline GraphSLAM Evaluation

We have presented the GraphSLAM system structure including the front-end and the back-end. In order to use it for online consistent state estimate for UAV onboard implementation, we first develop the algorithm in embedded system and verify the offline algorithm with pre-recorded data. This section aims to presents our work regarding the software development, the algorithm verification using synthetic data, the loop closure for real flight data and the paramter tuning of GraphSLAM.

### 5.4.1 GraphSLAM Software Development

In order to integrate the algorithm in embedded system, we follow a two-phase verification process: first we develop the code in Matlab for fast verification and better result visualization. Then the algorithm is implemented in C++ for the sake of running speed and the final code integration.

## Implementation using Matlab

In Matlab, we have implemented a complete SLAM system including the front-end and back-end. The front-end part consists of the following function block:

- Feature extraction: extract validate tree centers as landmarks;
- Scan matching: data association using nearest neighbor search.
- Sensor fusion: fusing velocity estimate from scan matching and acceleration measurements from IMU. This helps boost the state estimate from 10 Hz to 50 Hz.

In the back-end, the following functions has been implemented:

- Building the graph: the initial poses from front-end are checked with each other. For any two poses, we use the nearest neighbor search to determine whether the two scans have sufficient overlap. If yes, the loop is closed and an edge is added between the two poses. The pose is added to a vector structure storing the poses.
- Solving linearized equation: the error function is linearized at the current state and formulated as linearized equation. We use the default solver in Matlab to solve for an optimal state increment.
- Iterate to find the optimal solution.

## Implementation in Embedded System

In C++, most of the efforts concentrate on the front-end part. For the back-end part, thanks to the contribution of researchers in the SLAM community, there are quite a few open-source software packages available<sup>1</sup> suitable for 2D and 3D pose graph optimization. Some representative packages are listed as follows:

- **Hog-Man** [29]: it is a hierarchical optimization framework for online mapping on manifolds instead of in an Euclidian space. It utilizes a hierarchy of pose graphs to model the problem at different levels of abstraction. In online operation, only the coarse structure of the hierarchy is updated. Lower levels of graph are optimized only when the higher level optimization brings significant changes.

---

<sup>1</sup><http://openslam.org>

- **iSAM** [36, 35]: it is a library for batch and incremental optimization, recovering the exact least-squares solution. The library can easily be extended to new problems, and functions for common 2D and 3D SLAM problems are already provided.
- **g<sup>2</sup>o** [40]: it is a general framework for graph optimization. It is customizable in terms of error function definition and solve selection. Typical problems have been tackled such as 2D/3D SLAM and bundle adjustment. The framework achieves performance comparable to iSAM or HOG-Man and sometimes outperforms them.

Extensive comparison of the above-mentioned packages has been performed by researchers [20] and concluded that g<sup>2</sup>o is slightly better in terms of absolute trajectory error and the relative pose error. Therefore, we use g<sup>2</sup>o as the back-end optimization framework in this study. We have implemented the GraphSLAM algorithm in both Matlab and C++ languages. We compare the two sets of code for the same dataset we collected on our UAV. Fig. 5.9 shows the optimized trajectory, in which the agreement between the two results validates the developed software in C++.

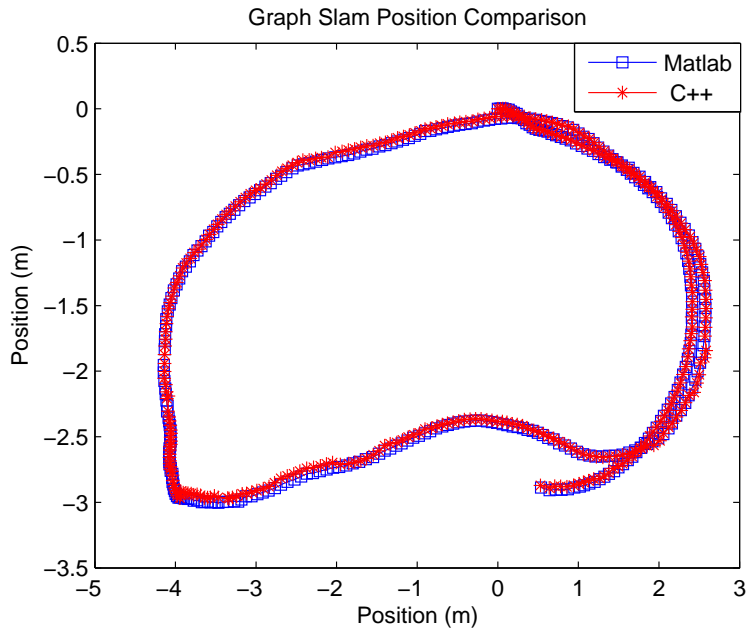


Figure 5.9: Optimized trajectory comparison between Matlab and C++

#### 5.4.2 Consistent Mapping with Synthetic Data

Building the simulation environment is essential in the sense that it can provide ground-truth measurement and trajectory, validating the algorithm developed. Random number

of trees with various radius are generated in a planar plane. The UAV is modeled as a mass point and moves according to a set of predefined waypoints. Along the trajectory, the measurement of laser range finder is logged. The laser range finder is modeled according to the practical specification, including the measurement range, resolution and field of view. The ground-truth position of UAV is also logged together with the laser range finder measurement. In order to show the effect of SLAM algorithm, we add Gaussian noise to the ground truth trajectory and treat it as the initial trajectory estimation. Fig 5.10 shows the overlapped map for the three types of trajectories: the ground-truth, the initial pose and the update pose. Fig 5.11 shows a enlarged view of a contour of one tree. From the zoom-in we could see that the measurement points projected on the initial trajectory (marked by green triangle) scatter around the ground truth contour of the tree (black dot). While the measurement points projected on the updated pose match the ground-truth perfectly. This proves that in Gaussian noise assumption, the GraphSLAM algorithm provides the optimal trajectory estimation. Fig. 5.12-5.14 compare the difference of position and heading angle with respect to the ground truth, showing again that GraphSLAM produces the optimal trajectory estimation.

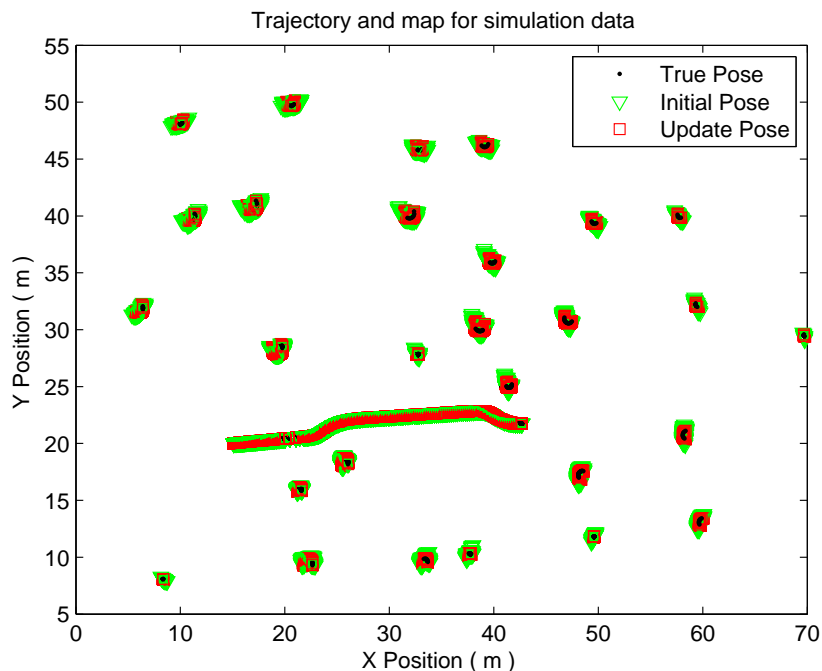


Figure 5.10: Optimized map and trajectory in simulation environment.

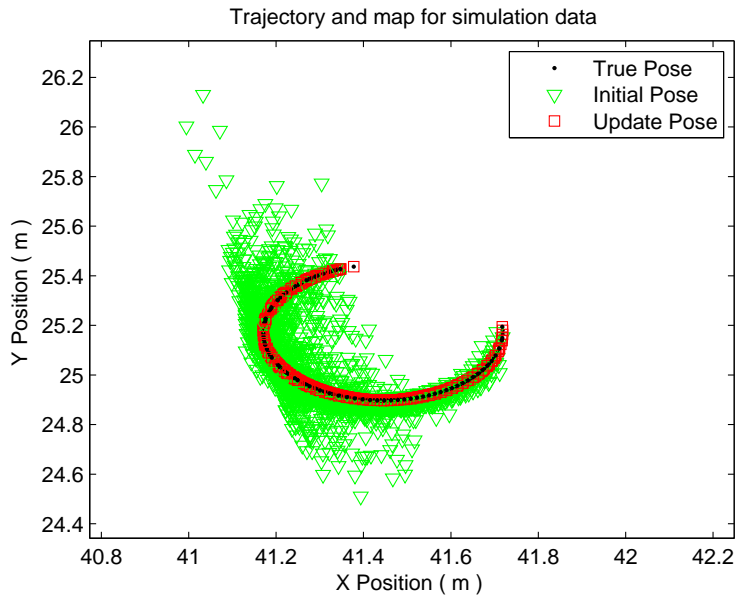


Figure 5.11: Optimized tree contour projected on the optimal pose.

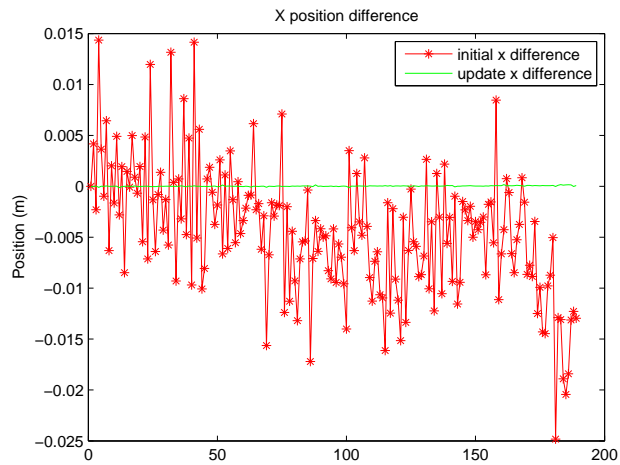


Figure 5.12: x position difference with respect to ground truth.

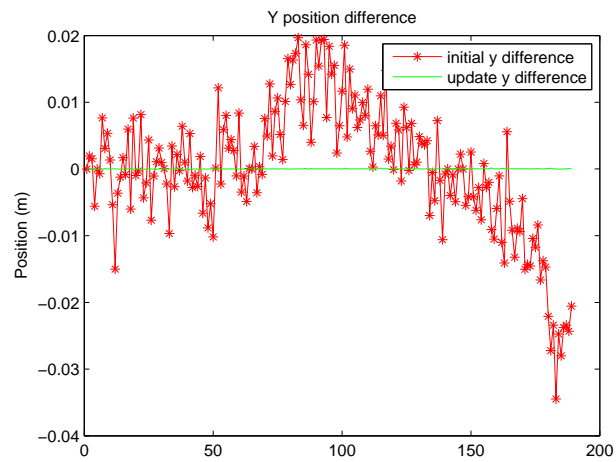


Figure 5.13: y position difference with respect to ground truth.

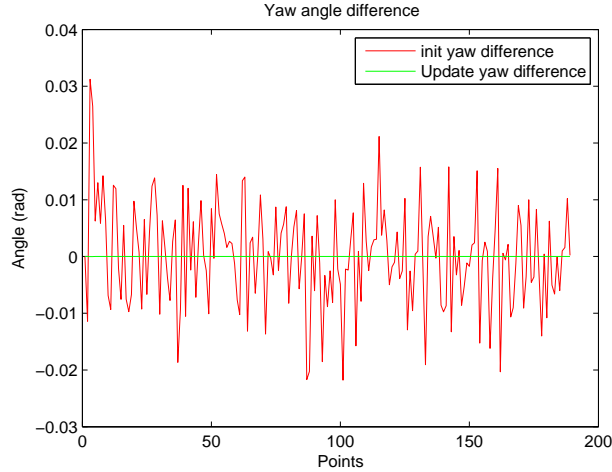


Figure 5.14: Heading angle difference with respect to ground truth.

### 5.4.3 Loop Closure Detection

Currently we use point-based scan matching to detect loop closure. Suppose there are two poses  $(\mathbf{x}_a, \mathbf{x}_b)$  with two sets of point features  $F_a, F_b$ . First a relative pose difference  $\mathbf{D}_{ij}$  from  $\mathbf{x}_b$  with respect to  $\mathbf{x}_a$  is calculated. Then, features  $F_b$  is projected to the frame of  $\mathbf{x}_a$  using the initial pose difference  $\mathbf{D}_{ij}$ , creating feature set  $F_{ba}$ . A nearest neighbor search is conducted between  $F_{ba}$  and  $F_a$ . If sufficient number of inliers are found according to a distance threshold, the two poses  $\mathbf{x}_a$  and  $\mathbf{x}_b$  indicate a loop closure. A new pose difference is determined using scan matching. An update pose  $\mathbf{x}'_b$  is generated with respect to  $\mathbf{x}_a$ .

Indoor forest dataset is used in loop detection evaluation. Fig. 5.15 shows a map for poses at time step 1 and 350, denoted as  $\mathbf{x}_a$  and  $\mathbf{x}_b$  respectively. The red plot is the measurement projected on the first pose  $\mathbf{x}_a$  and the green plot is the measurement projected on the second pose  $\mathbf{x}_b$ . By examining the plot, we find that the green square wall and pillars do not coincide with the red ones. We perform a loop closure update between the two poses and the updated map is shown in Fig. 5.16. The updated map is more consistent than in Fig. 5.15. Fig. 5.17 depicts the close view of the initial map and the updated map, in which the square pillars align with each other and the circular tree contours coincide. Performing the loop closure updates with more previous poses may further correct the pose to the right one.

In this test, we use point features and scan matching to test loop closure. This is indeed applicable in cases where the initial trajectory is not far from the optimal one and the loop is not large. If any of the above assumption fails, loop closure may produce

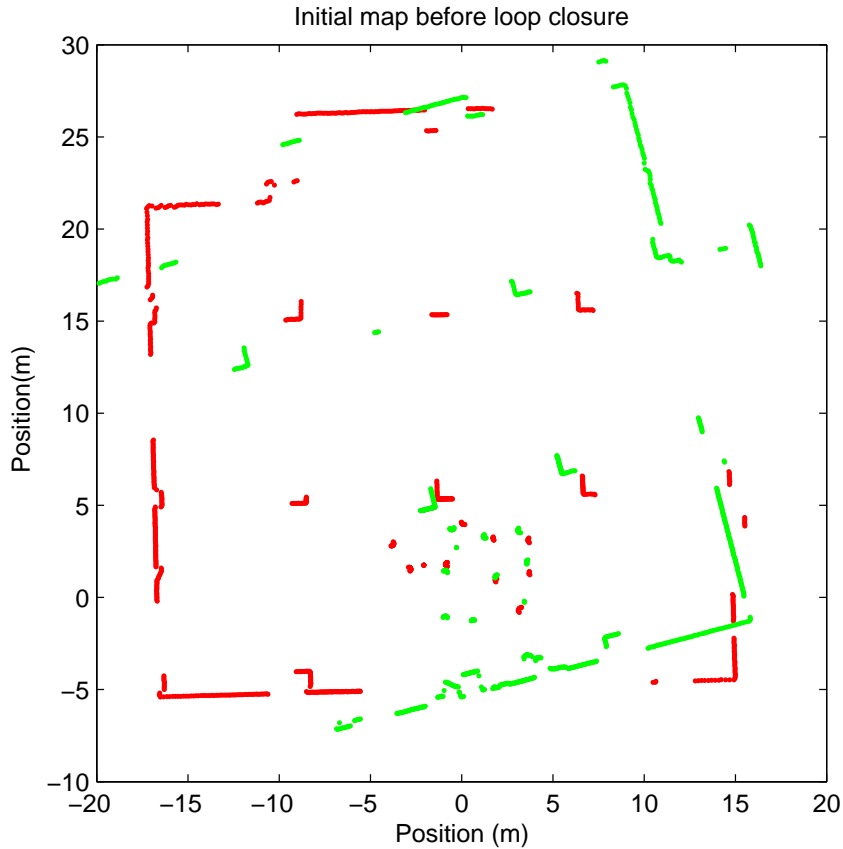


Figure 5.15: Drifted map before loop closure.

the wrong update. This problem could be alleviated by using more sensor information and more robust multi-hypothesis data association. Using visual salient features to close a loop is needed in future development.

#### 5.4.4 GraphSLAM Parameter Tuning

In the application of GraphSLAM, we have used point-based features and scan matching to build up the graph. The two main parameters to tune are the type of features used for scan matching and the size of searching window for loop detection.

For the feature point selection, we could select the mean of each cluster or the estimated center of each cluster. Each cluster is a portion of tree trunk facing the laser range finder. Assuming a cylinder shape of tree trunks, we could derive the position of trees as the estimated tree centers. When the cluster has many points and they show patterns of a circle, it is reasonable to use the estimated center since they remain constant when viewed from different angle. However, when there is limited number of points in each cluster, using mean of each cluster is preferable because the estimated



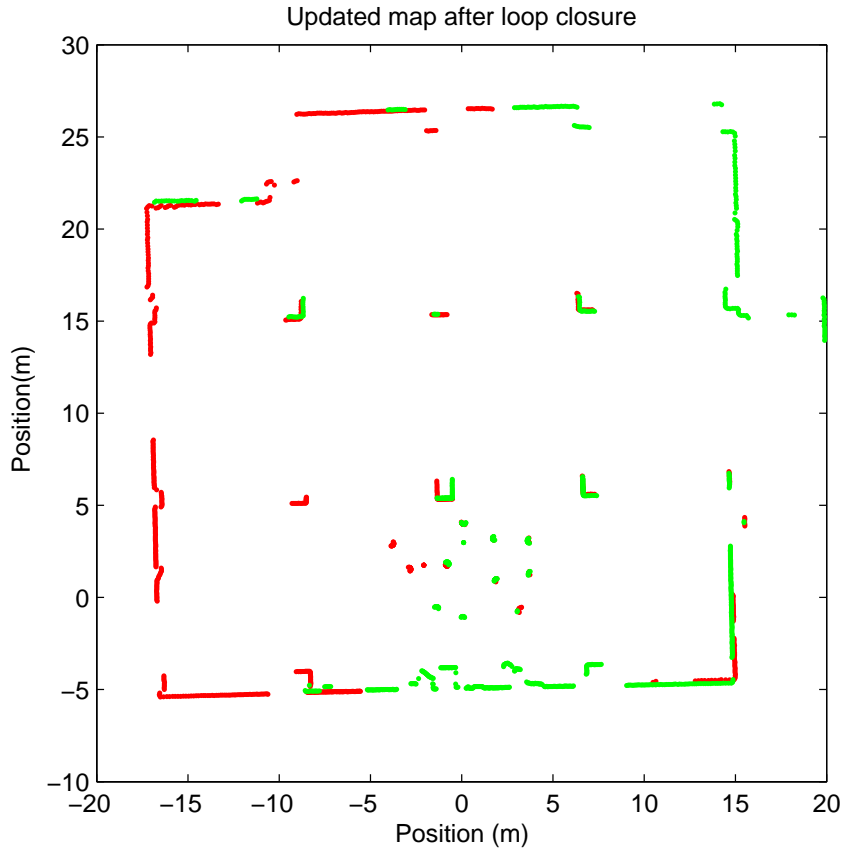


Figure 5.16: Consistent map after loop closure.

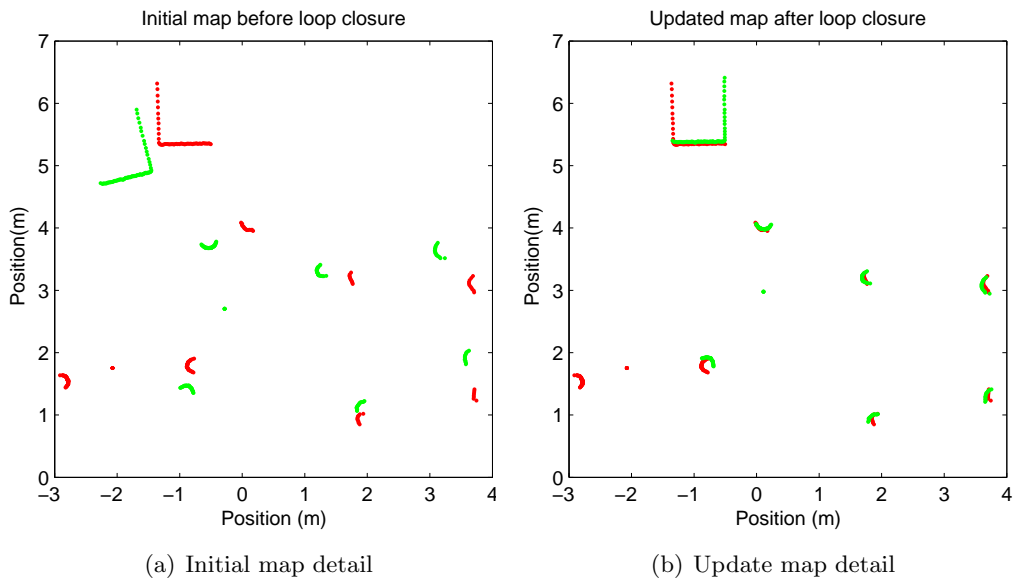


Figure 5.17: Map details before and after loop closure.

center is prone to large error. Using the mean of cluster leads to one problem: the features may shift a little when seeing from different angle of the cylinder. Therefore, estimated centers should be used as features as much as possible. But if there are too

little measurement points in each cluster, the mean of the cluster measurement are used.

For the loop closure, the window size to search possible loop closure is another important parameter. When a new pose node is added to the graph, it could search over all the previous poses in the graph. Or alternatively, the search window can also be limited to a certain range, such as only the poses in 5 seconds before. Using local window or global window would generate different consequences for the SLAM algorithm. Using global window will add all the possible edge constraint to the graph, producing the optimal estimate of the whole trajectory. The downside of global search window is that as the trajectory grows long, it becomes more and more expensive to search for loop closure up to the beginning of the trajectory. On the other hand, using local search window limits the search range to a fixed size, creating a smoothing SLAM algorithm which is constant in time complexity. But it is at the expense of discarding all the constraints beyond the search window. Using local window can be regarded as a sub-optimal solution.

In this test, we seek to evaluate the effect of the two parameters: feature selection and search window. There are four combinations as listed in Table 5.1. The GraphSLAM algorithm is performed for the four cases on the same dataset.

Table 5.1: GraphSLAM parameter tuning table

Case	Used Features	Search Window
A	Average points	Local
B	Average points	All
C	Tree centers	Local
D	Tree centers	All

The four sub-pictures in Fig. 5.18 show the overlapped maps for the indoor forest dataset in the same scale. By comparing the four pictures we can conclude as follows:

- Search window effect: comparing case A and B, which use the same average points as features, we found that case B produces more consistent maps. The rectangle pillar in case B is thinner than the one in case A.
- Feature selection effect: comparing case B and D, which both use global search window, we found that using tree centers as features is better. The circular tree contours in case D has an apparent hollow space in the tree contour estimation while case B does not. This also applies to case A and case C.

- Accuracy ordering: comparing case A, case B, case C and case D, we found that their consistency increases. Using tree centers and global search window produce the most consistent map.

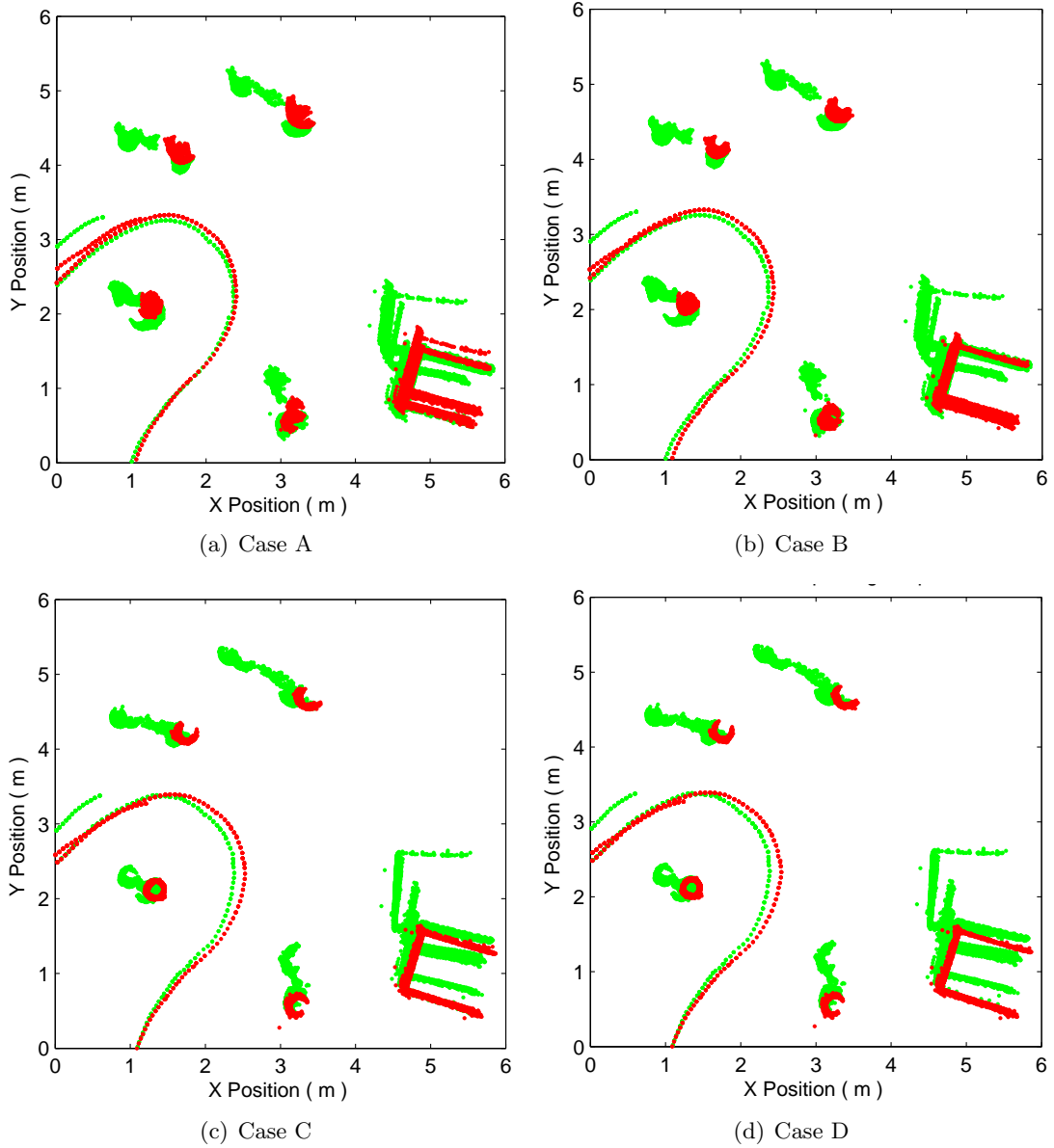


Figure 5.18: Close view of optimized map compared to initial map.

Fig 5.19 shows a close view of the map for case D. It can be seen that the red circle contour is more consistent than the green divergent one. This reemphasizes the fact that using tree centers as features and global search window produces the optimal trajectory. In practice, it might not be easy to extract the correct tree centers and search over global window will increase the computation burden.

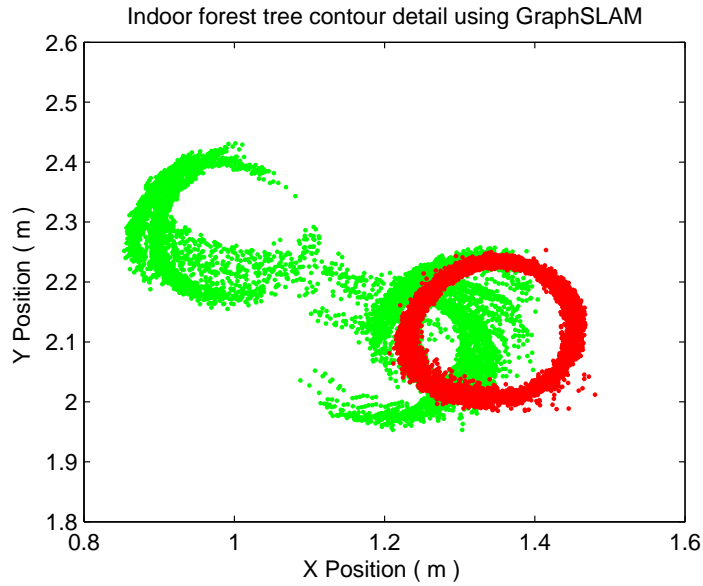


Figure 5.19: Tree contour details for indoor forest using GraphSLAM.

## 5.5 Conclusion

In this chapter, we have presented an offline consistent localization and mapping framework with the GraphSLAM as the back-end optimization technique. The framework is composed of the front-end Kalman filter and the back-end optimization. The procedures of interpreting the sensor information in the front-end are listed to help build up a pose graph, and the mathematic formulation of the graph is also given. The developed framework is implemented first with Matlab to evaluate its performance and then with C++ for faster computation and onboard implementation. The two versions of the framework perform alike on the same dataset. The developed framework has been extensively evaluated using various datasets from synthetic simulation, indoor and outdoor forests, to tune the parameters for a better performance. The evaluation results highlight the importance of reliable feature extraction and loop closure, which should be taken into account for real-time onboard applications. The evaluation also demonstrates that our framework significantly improves the consistency of the map compared with the map obtained by laser odometry.

## Chapter 6

# Autonomous Flights with Online GraphSLAM

### 6.1 Introduction

In previous chapters we have developed various techniques for UAV navigation system, including the avionics system design, the modeling of UAV dynamics, the design of control law, the motion estimation and the GraphSLAM. To autonomously navigate a UAV without GPS signals, all those proposed techniques need to be integrated in a systematic way. We present our system integration framework in Fig. 6.1, showing the signal flow of different modules, where  $\mathbf{x}_{\text{est}}$  is the estimated states,  $\mathbf{x}_{\text{ref}}$  is the trajectory reference,  $\{P, M\}$  are the estimated trajectory and map respectively.

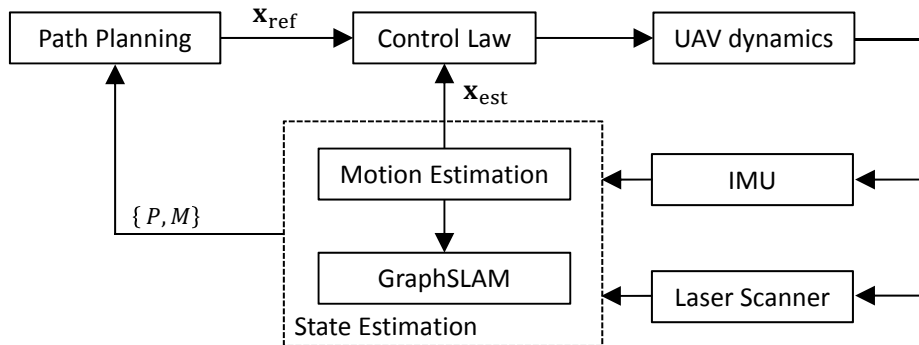


Figure 6.1: System diagram of UAV navigation system.

GraphSLAM is originally an offline algorithm which optimizes all the poses on the whole trajectory based on the measurements collected at each pose. However, the global optimization can only be initiated after all the measurement data are collected, meaning

that the UAV can only obtain a globally consistent state estimate at the end of the flight. This is not desirable as the UAV needs a real-time consistent state estimate for autonomous control, otherwise the fast drifting pose estimates will soon jeopardize the navigation. To tackle this problem, we develop an online GraphSLAM using a sliding window method in Section 6.2.

Fully autonomous navigation of the UAV demands a collision-free trajectory reference. Path planning is therefore indispensable for autonomous navigation of UAVs, especially in obstacle-strewn environment. We will present our solution to path planning in Section 6.3. We implement the developed algorithms on embedded system for practical flight tests. Multi-threading techniques are applied to organize the algorithms into different threads. The details of the onboard software system are presented in Section 6.4. Real flight tests are performed to verify the proposed UAV navigation solution in two GPS-denied environments: an indoor environment with synthetic trees and a real small scale forest. The experiment results of autonomous flights in such environments are presented in Section 6.5.

## 6.2 Online GraphSLAM using Sliding Window

GraphSLAM seeks to optimize the poses on the trajectory by maximizing the likelihood of the measurement. The most consistent map and trajectory can be obtained by checking whether each pair of two poses are overlapped based on the corresponding measurement. If the overlap of the two measurements exceeds a certain threshold, an edge is established to describe the relative pose difference between the two poses. In order to achieve the best optimization result, a new pose and its measurement are compared with all the preceding poses and measurements. The number of poses in the trajectory grows linearly with the time. The longer the UAV travels, the longer it takes to perform the global optimization. The increasing computation time of GraphSLAM makes it only suitable for offline optimization when all the measurements are collected.

To tackle the problem of linear time increase, we designed a constant time GraphSLAM to facilitate onboard optimization. The main idea is to set a sliding window along the trajectory, limiting the search range only to those poses lying in the time window from the current pose. As illustrated in Fig. 6.2, the first pose and its measurement is

denoted as  $\mathbf{x}_0$ . It also serves as a reference origin to which all the future poses will be referred. As the UAV collects more data, a series of new poses and measurements are added, including  $\{\mathbf{x}_2 \cdots \mathbf{x}_t\}$ . The sliding window is initialized with a capacity of  $n$  as the first pose  $\mathbf{x}_0$  is pushed into the window. More poses are pushed into the sliding window before it is full. After that, new pose  $\mathbf{x}_t$  can only be pushed in after the first pose in the window  $\mathbf{x}_{n-t}$  is popped out. At each time step  $t$ , three operations are performed: the first pose is popped out, the newest pose is pushed into the window and a local GraphSLAM optimization in the local sliding window is performed.

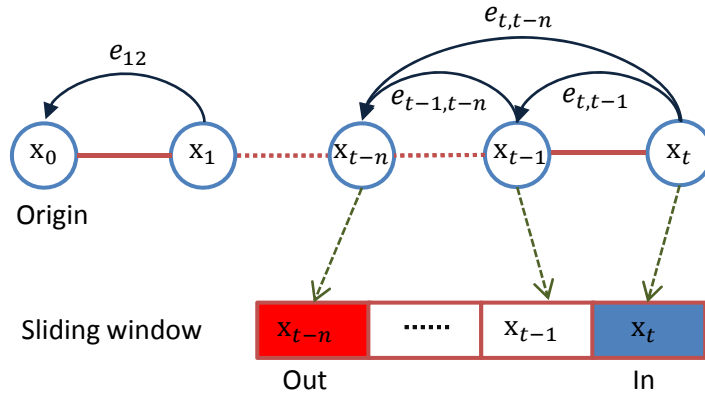


Figure 6.2: Sliding window diagram with poses being pushed in and popped out.

The size of the sliding window plays a critical role in the performance of the online GraphSLAM. As shown in Fig. 6.2, if the sliding window size is large enough to hold all the new poses and measurements, the online GraphSLAM becomes a full GraphSLAM with increasingly long computation time. If the window size is too small, only the most recent several poses are optimized, losing the capability to detect large loop closure when the UAV travels to a previously-visited place. To balance the computation time and the optimization performance, extensive comparisons have been performed from which a time window of 5 seconds is determined to be a practical choice.

The online GraphSLAM based on the sliding window significantly decreases the drift of the position estimate compared with that of the Kalman filter. However, the introduction of sliding window is indeed a sacrifice of the optimization performance by limiting the searching range only in the 5 seconds sliding window. For long time navigation, the UAV position is still prone to drift without global optimization. Therefore, a two-layer back-end framework is presented as shown in Fig. 6.3. Poses and features from the front-end are pushed into the sliding window at each time step. After the local opti-

mization, the optimized pose estimate is transferred back to the front-end for real-time control. At the same time, the locally optimized pose is pushed into a larger container to store all the poses and measurements, forming a global graph to be optimized after the mission. This two-layer graph setup makes sure the UAV achieves slow drift in the pose estimation during flight and eventually obtains a globally consistent trajectory and map afterwards. This configuration is justified by the fact that the UAV does not need perfect pose estimate during flight and the slow drift caused by the local sliding window optimization is acceptable for UAV operations lasting up to 10 minutes.

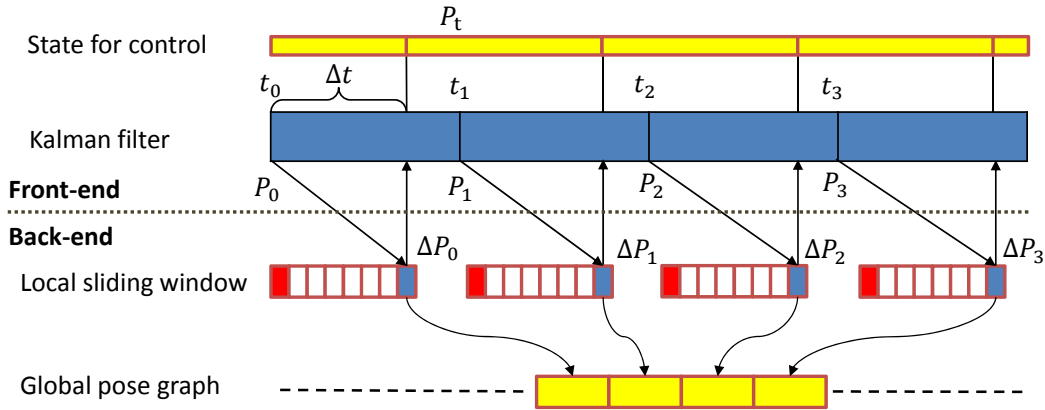


Figure 6.3: A timing graph showing the interaction between the front-end, the sliding window local optimizer and the global optimizer.

Due to hardware constraints, the front-end and back-end algorithms are run in two computers. The front-end algorithms, including the scan matching and the Kalman filter, run on the Gumstix Overo Fire. While the back-end algorithms, including the sliding window online GrpahSLAM and the global pose graph construction, run on Mastermind. The two computers are connected through a serial port. Fig. 6.3 shows the message interaction between the two computers. Initially, the state  $\mathbf{P}_0$  is directly fed to the autonomous control. To optimize the initial state  $\mathbf{P}_0$ , it is sent to Mastermind with its measurement. The time delay caused by the local optimization, the global pose construction and the serial communication make it impossible to use  $\mathbf{P}_0$  directly for flight control. In particular, experiments show that the delay  $\Delta t$  between the initial  $\mathbf{P}_0$  and the optimized pose  $\mathbf{P}_0^n$  is 300 ms. Recalling that the main loop in the front-end is 50 Hz, the delay of 15 loops is not negligible for real-time operation. To deal with the delay, we propose to design a state update scheme to take into account the delay which works as follows: At time  $t_0$ , we have an initial pose  $\mathbf{P}_0$  from the Kalman filter. After



time  $\Delta t$  we receive the pose correction  $\Delta\mathbf{P}_0$  of  $\mathbf{P}_0$ , we have a new pose for time  $t_0$ ,

$$\mathbf{P}_0^n = \mathbf{P}_0 \Delta\mathbf{P}_0. \quad (6.1)$$

At any time  $t > t_0 + \Delta t$ , the initial pose is,

$$\mathbf{P}_t = \mathbf{P}_0 \Delta\mathbf{P}_0^t, \quad (6.2)$$

where  $\Delta\mathbf{P}_0^t$  is the initial pose difference between  $\mathbf{P}_t$  and  $\mathbf{P}_0$ . The update pose  $\mathbf{P}_t^n$  of time  $t$  is,

$$\mathbf{P}_t^n = \mathbf{P}_0^n \Delta\mathbf{P}_0^t = \mathbf{P}_0 \Delta\mathbf{P}_0 \Delta\mathbf{P}_0^t. \quad (6.3)$$

At time  $t_1$  a new initial pose is sent to the back-end for optimization and after time  $t$  the update pose  $\Delta\mathbf{P}_2$  is returned. The update state is again updated using Eq. 6.3, except that the time index is changed from  $t_0$  to  $t_1$ .

### 6.3 Online Path Planning

Different methods and techniques for path planning have been proposed and any successful approach must satisfy the following requirements: the methods must provide a collision free and dynamically feasible trajectory that leads the vehicle to the target with the capability of fast online re-planning to deal with dynamic environments. Historically, a two-level structure with a global planner and a local planner is widely adopted. Usually, a coarse and lower-dimension state lattices are used for the global planner to decrease the searching complexity and increase the computation speed. For the local planner, various methods have been proposed for ground vehicles, including pure tracking controller, dynamic window approach or vector field histogram and their variations. For air vehicles, due to the complexity and high dimension of the model, it is difficult to use similar methods to generate a dynamically feasible trajectory. Most successful application uses motion primitives during the planning process. However, the use of motion primitives involves building giant look-up tables and thus limits the trajectory to be combination of these motion primitives. A better solution is to approach the trajectory generation as a two-point boundary value problem. The trajectory generator takes in a series of states that the vehicle needs to reach and returns a dynamically fea-

sible trajectory. Though some trajectory generators could handle an arbitrary number of states, they either sacrifice the ability to explicitly specify the dynamic constraints or simply are too computationally intensive. On the other hand, a trajectory generator that only consider the initial and final states is a two-point boundary value problem which is more efficient.

Based on the work mentioned above, we propose a path planning system with global path planner using A\* searching [42] and a local planner using efficient two-point boundary value problem solver [38]. It provides dynamically feasible trajectories to lead the vehicle from any initial position to any reachable final position. The detailed steps of the path planning structure are given in Algo. 4.

---

**Algorithm 4: ONLINE PATH PLANNING FRAMEWORK**

---

**Input:** Current pose  $\mathbf{x}$ , obstacle position  $\{m_i\}$  in local body frame,  $i = 1, \dots, n$ .

**Output:** Trajectory reference  $\mathbf{x}_{\text{ref}}$

- 1 Search in the configuration space using A\*;
  - 2 Connect the grids using split and merge, generating a series of line segments;
  - 3  $R_g \leftarrow$  take the first turning point of the line segment as the line segments;
  - 4  $\{\mathbf{r}_j\} \leftarrow$  sample multiple local targets around the current vehicle state  $\mathbf{x}$  and order them in descending order based on their distance to the global target  $R_g$ ;
  - 5 **for** all local targets in  $\{\mathbf{r}_j\}$  **do**
  - 6  $\mathbf{x}_{\text{ref}} \leftarrow$  Solve the boundary value problem between  $\mathbf{x}$  and  $\mathbf{r}_j$  ;
  - 7 collision  $\leftarrow$  check if collision happens between  $\{m_i\}$  and  $\mathbf{x}_{\text{ref}}$  ;
  - 8 **if** collision **then**
  - 9 Delete the current local target and choose the second best local target ;
  - 10 **else**
  - 11 Break ;
  - 12 **end**
  - 13 **end**
  - 14 **return**  $\mathbf{x}_{\text{ref}}$ ;
- 

The above algorithm consists of several main blocks: the global configuration space search using A\* (step 1) and the boundary value problem (step 6). The global configuration space search is to give a rough plan that ignores the complex dynamics of the vehicle but considers as much topological information as possible. Since there is no prior map of the environment, a local map based on the current laser range finder measurement is built up in polar coordinate. The A\* path planning algorithm is actually a graph search algorithm. To run the A\* searching algorithm, a polar coordinate map is first built from the input of a scanning laser range finder. For each obstacle point returned from the scanning laser range finder, a Gaussian-based cost field is added around it. A

preprocessed polar coordinate map is shown in Fig. 6.4. The boundary value problem seeks to generate a reference trajectory given two sets of conditions on the boundaries. Reflexxes Motion Libraries [38] provides a general solution to this problem.

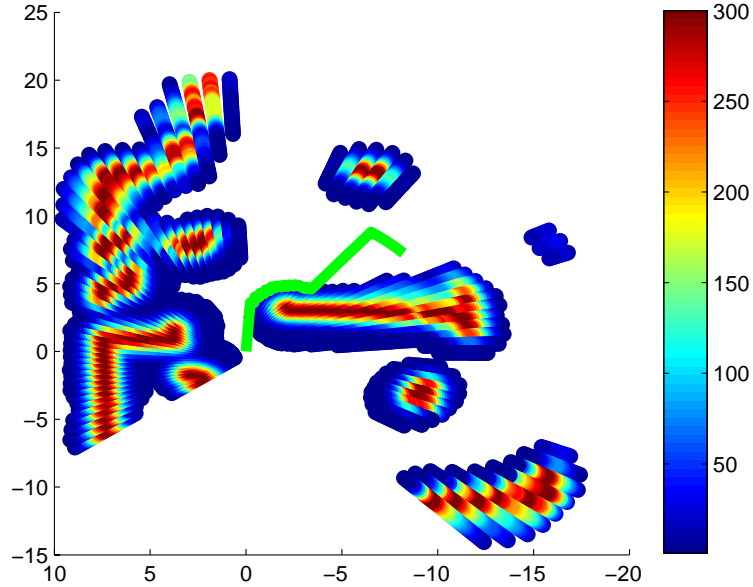


Figure 6.4: A Gaussian cost map in polar coordinate. The color ranging from red to blue indicates the closeness of the grid to the detected obstacle.

During the A\* searching, the algorithm would generate a path with the lowest cost from the current position of the vehicle to the target point. A typical path is shown in Fig. 6.4 as the green line. The resulted path normally consists of a series of waypoints located in each grid in the polar coordinate. In order to find the best direction the vehicle should aim for, a split and merge algorithm is used to transfer these waypoints into a series of line segments. The split and merge process finds the first turning point  $R_g$ , which is used to determine the best target point to go. The direction and the distance of the first turning point are then passed to the local path planner to search for a collision free path from the current vehicle position to the turning point.

For the local planner, a similar idea close to the vector field histogram (VFH) method is adopted. In VFH, the vehicle always turns to the direction that is both obstacle free and also towards the target  $R_g$  from the global planner. The behavior is realized by forming an optimal function that consists of different objectives, such as the distance to the global target and the angle difference compared to the last direction. We sample multiple local target points around the vehicle and calculate the resulted trajectory

based on the current vehicle states and the local target points. The trajectory that is both collision free and closest to  $R_g$  is selected. Each trajectory starts at the current vehicle states and ends at the local target points with zero velocity and zero acceleration. Therefore, the vehicle is always at a safe state so that it could stop and avoid obstacles when following the initial trajectory.

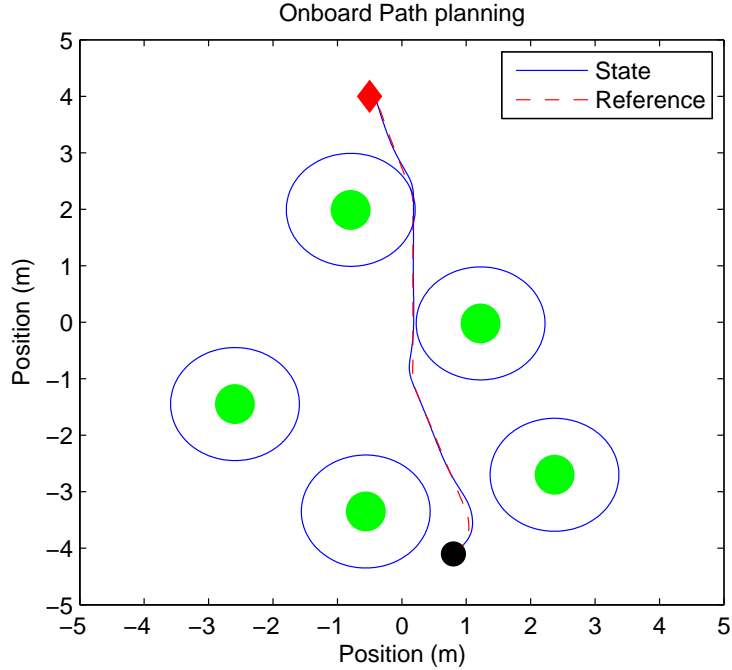


Figure 6.5: UAV response together with reference in map.

Fig. 6.5 depicts the simulation results integrating the UAV model and the path planning module. The initial starting point and the final target position are marked by the solid black circle and the red solid diamond respectively. The map consists of five trees marked by green solid circles and the large blue line circles represent the safe zone of each tree. The UAV has to fly through these trees and reach the target. At every time instant, a new reference position is obtained based on the current state of the UAV and the surrounding obstacles. The reference position has to be obstacle free and meets the UAV dynamics constraints. We could see a safe trajectory reference marked by red dashed line is generated. The blue solid line is the simulation state estimate using the UAV dynamics. The agreement in the simulation validates the performance of the path planning module. In Section 6.5.3, we will present the real flight test results about the onboard path planning.

## 6.4 Onboard Software Development

The software system of the UAV navigation can be decomposed into two main subsystems, the onboard system and ground control station respectively. The onboard system deploys the application to realize real time flight tasks while the ground control station is for monitoring the UAV state and sending commands to UAVs. We share the same ground control station developed in NUS UAV research team. Our main efforts regarding the software development have been focused on the onboard system.

Considering the comprehensive functions and logics implemented on UAV onboard system, it is further structured into two main modules according to the avionics system configuration. As shown in Fig. 6.6, two onboard processors are adopted to realize all the software modules developed in this study: *Mission plan processor* and *Flight control processor*. As mission plan tasks normally involve computationally intensive algorithms such as path planning, obstacle avoidance and SLAM, a high-end powerful Intel Core i7 based processor called Mastermind (from Ascending Technologies Germany) with Ubuntu 12.04 is deployed as the *mission plan processor*. The Ubuntu operating system has mature development environment with rich libraries for robotics applications, which can facilitate the overall development. For the critical flight control, a lightweight yet powerful OMAP3530 based Computer-On-Module (COM) called Gumstix Overo Fire is adopted. The flight control system is implemented based on QNX Neutrino real-time operating system (RTOS). QNX RTOS is developed with a true microkernel architecture which integrates only the fundamental services including CPU scheduling, interprocess communication, interrupt and timers. Drivers and user applications are all executed as user processes. This architecture can provide a quite small yet fully customizable and manageable user application suits with necessary drivers and libraries.

Based on the specifications from the system structure, tasks to realize the flight missions are examined. The tasks are assigned, from the high level navigation to the low level flight control, into the *Mission plan processor* and the *Flight control processor* respectively. Since the Mastermind processor possesses powerful processing capabilities, high level tasks such as *SLAM* and *Path planning* are scheduled. For flight control subsystem, its subtasks are scheduled into the following order to achieve the closed-loop control system. Navigation sensors are retrieved first with *Laser* and *IMU*. With

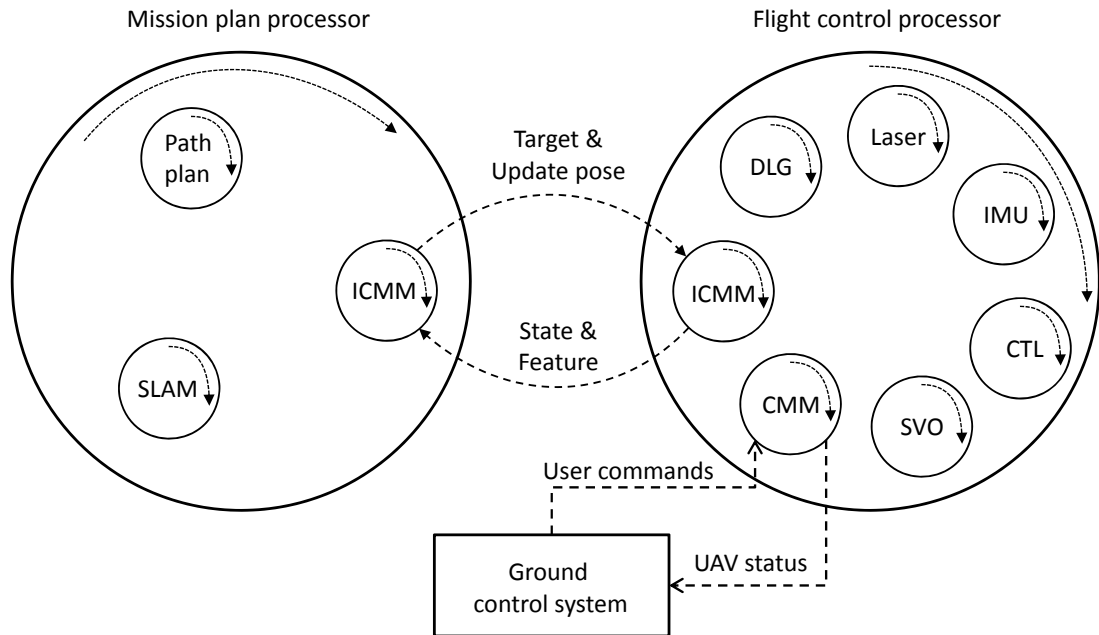


Figure 6.6: Software structure of the UAV navigation system. Robust perfect tracking control is implemented in *CTL*, and scan matching in *Laser*, Kalman filter in *IMU*, GraphSLAM in *SLAM* and obstacle avoidance in *Path plan*.

the laser data, the scan matching is performed from the two consecutive scan data. After fused with the IMU, a motion estimation with navigation data is further used for the control task *CTL*. With the generated automatic control signal, motor driving signals are sent to the UAV motors from the *SVO* task to achieve the 6-DOF movement. Other auxiliary tasks are also implemented: the communication task *CMM* is to send status data back to Ground Control System (GCS) for user monitoring and receive user commands, the data logging task *DLG* is used to record flight status data for post flight analysis. Finally, to pass high level navigation data to *Flight control processor* and share UAV status with *Mission plan processor*, the inter-processor communication task *ICMM* is implemented on both processors.

All the tasks are scheduled in a periodic fashion, whose executions follow the order in Fig. 6.6. On *Flight control processor*, most of the tasks are scheduled in 50 Hz, except the *CMM* and *DLG* which are executed every one second to for communication to the ground control station and onboard data logging. Each task is assigned a certain time for its implementation inside the 20 ms period. The multiple thread management of QNX schedules the different tasks. The high level algorithms, such as path planning and SLAM, are designed for navigation purpose and often computationally intensive, a relative low scheduling frequency of 10 Hz is implemented on the *Mission plan processor*.

## 6.5 Experiment Results

With the UAV navigation system integrated in the embedded system, we designed several flight tests to verify the navigation system in GPS-denied environments. Three experiments are designed:

1. The first one is autonomous flight in indoor environment with synthetic poles as features and with online GraphSLAM. A preplanned collision-free path is used as trajectory reference. The details are in Section 6.5.1.
2. The second test is autonomous flight in a small-scale forest with dense tree canopies and sparse tree trunks. The whole mission is autonomous with the online GraphSLAM being applied to optimize the trajectory. Practical forest exhibits a range of challenges including the uneven terrain and the slanted tree trunks. The details are presented in Section 6.5.2.
3. The last flight test aims to verify the online GraphSLAM with the online obstacle avoidance algorithms. The UAV is required to fly to five waypoints while maneuvering around obstacles with the state estimates from online GraphSLAM. The details are presented in Section 6.5.3.

### 6.5.1 Autonomous Flight with Online GraphSLAM

The performance of the online GraphSLAM can be assessed in two aspects: whether the optimized states can be used for the real-time autonomous control and whether the optimized trajectory is more consistent than the initial trajectory. In order to extract these two performance indexes, the influence of other ingredients like noisy measurement and oscillating trajectory must be minimized. The testing scenario as shown in Fig. 6.7 is built up to minimize the influence of measurement noise. The tree trunks are synthetic paper tubes with perfect cylindrical shapes. The perfect circular shape remains constant in vertical direction, fulfilling the assumption of vertical uniformity. The trees are placed at least 3 meters away from each other, making it impossible for wrong data association. To isolate the path planning algorithm, a collision free trajectory is predefined and loaded to the computer on the system startup.

Even though this testing scenario consists of synthetic paper tubes, it possesses its own significant merits when compared with the software simulation environment. First,



Figure 6.7: Indoor test scenario for GraphSLAM verification.

all the measurement data come from onboard sensors while the UAV is flying through the poles. The laser range scans and acceleration measurements from the IMU are still prone to noise and drift. Second, the environment is complicated, consisting of not only circular poles, but also square pillars and the direct interior of the walls. This poses challenges for the onboard feature extraction and motion estimation. Third, the whole mission is designed to be autonomous, with complete state estimation and onboard control. Practical constraints like the effects of measurement delays and the noise of IMUs make it challenging to fly in such environments.

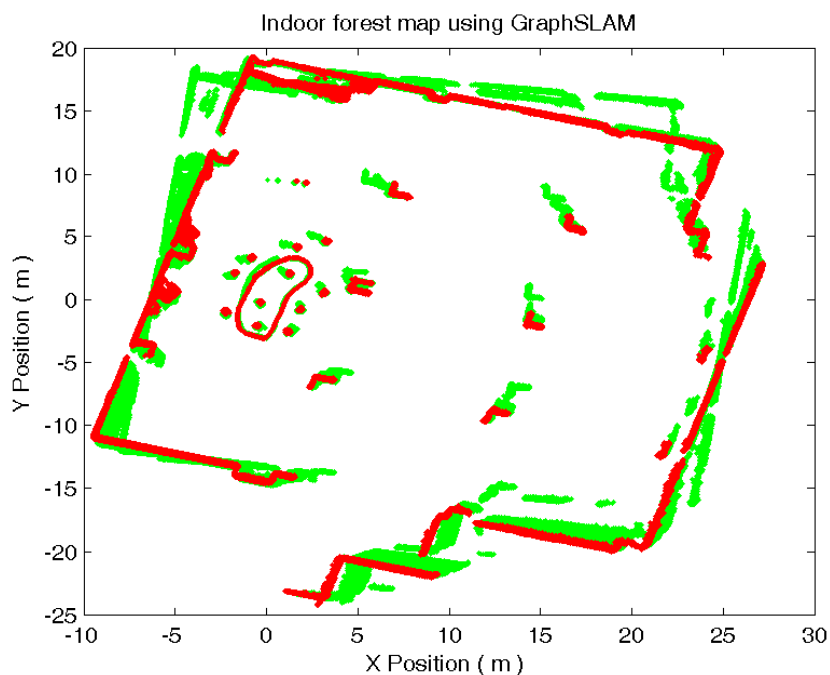


Figure 6.8: Comparison of initial map and optimized map using GraphSLAM.



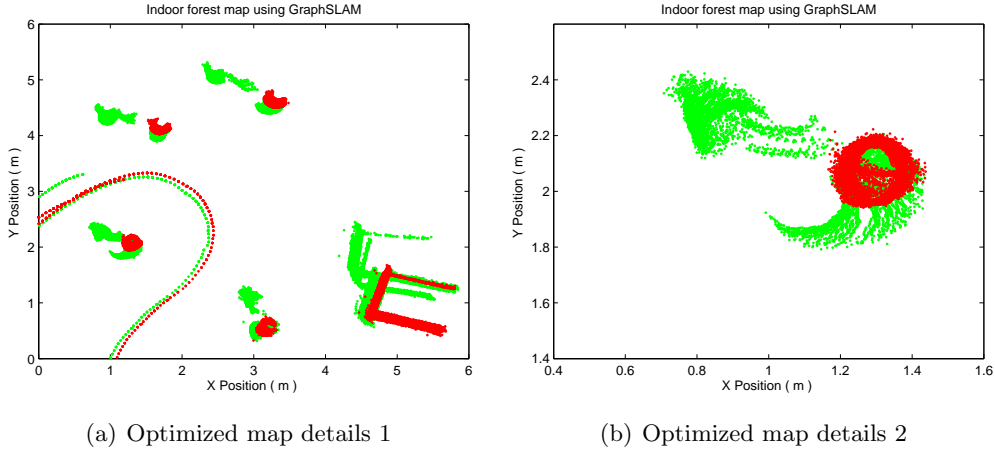


Figure 6.9: Close view of the optimized map compared to the initial map

The UAV flies autonomously around the poles, following the predefined trajectory. The laser range measurement and the corresponding initial poses and the optimal poses are all recorded onboard for data analysis. All the measurements are transformed to the frame of the first scan using the corresponding pose estimate. Fig. 6.8 shows a comparison of the maps projected on the initial trajectory and the optimized trajectory respectively. Since there is no ground truth available in the indoor, we can not quantitatively analyze the GraphSLAM performance. We consider the consistency of the projected map as the criteria. The initial map and trajectory are the results of motion estimation based on scan matching, marked by the green dot plot. The initial overall map is not consistent: the corners of the walls and the position of the square pillars all drift away. The red-dot plot is the optimized map and trajectory. Visual checking of the red map shows the optimized map is more consistent than the initial green map. Fig. 6.9 shows the details of the map. Fig. 6.9(a) depicts that the red pillar retains its rectangular shape while the green pillars drift 1 meter away. Fig. 6.9(a) zooms to the outer contour of one landmark, indicating that the optimized tree contour is more consistent than the green initial contour. The red contours remain to fall into a circle while the green contours spread in a larger area.

### 6.5.2 Autonomous Flight in Small Scale Forest

After the evaluation of the GraphSLAM algorithm in indoor environment, we performed autonomous flight in a real small forest with dense tree canopies and sparse tree trunks. Compared to the indoor forest environment, this environment exhibits several challenges:

first, the uneven terrain produces undesired ground strikes of the laser range finder, making the feature extraction more difficult. Second, the trees in the forest have small trunk size, irregular shape and slanted orientations. Small size tree trunks cause a small number of measurement points for each tree segment in the clustered range scan, reducing the accuracy of the circle fitting. The distribution of the points does not follow a circle shape due to noise effect and the shape of trees. The slanted tree orientations violates the vertical uniformity assumption. The data association is more prone to error than that of the first experiment. Therefore, the scan matching and the online GraphSLAM using these tree features are challenging problems.



Figure 6.10: UAV flying in the small scale forest in front central library of NUS.

Fig. 6.11 shows the comparison between the initial map and the optimized map for the forest. The green plot is the initial map from Kalman filter while the red one is the optimized trajectory and map. It can be seen that there are two neighboring clusters of green plots while only one cluster of red points. This is the result of GraphSLAM which corrects the trajectory of the UAV. Due to the complexity of the environment, there is no hollow tree contours plotted. There are still large clusters of objects which do not correspond to trees in the environment. This is mainly due to the complex tree conditions of the forest as shown in Fig. 6.12. The trunk of tree 'B' is slanted about 15 degrees and tree 'C' and 'D' have more than one thick branches at the flight height. These complexities cause serious problems in the feature extraction and data association processes. In conclusion, the GraphSLAM demonstrates its positive effect in correcting the trajectory. Improvement in the feature extraction and data association in such complex environment will lead to more consistent map.

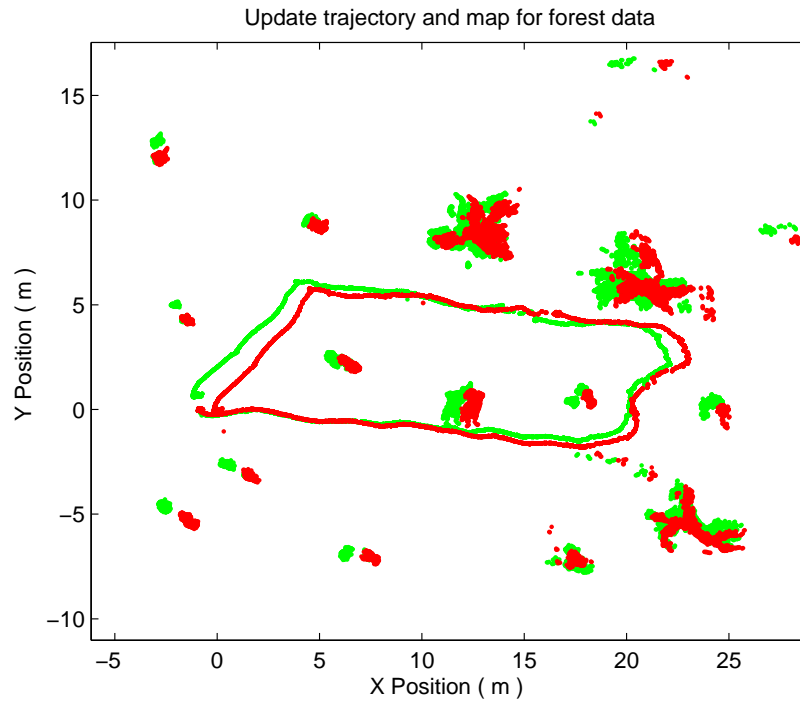


Figure 6.11: Optimized map and trajectory in small forest.



Figure 6.12: Miscellaneous tree trunk conditions.

### 6.5.3 Autonomous Flight with Online GraphSLAM and Online Path Planning

We have validated the performance of the UAV navigation system using online consistent state estimation and robust perfect tracking in the previous two tests. Predefined trajectory references have been used during the flights. However, in practice, the flight area is unknown prior to the take-off and the environment is always occupied with ob-

stacles. Online path planning is demanded to provide real-time trajectory references which avoid the obstacles and return to original trajectory plan as soon as possible. This section reports the experiment results integrating the online path planning.

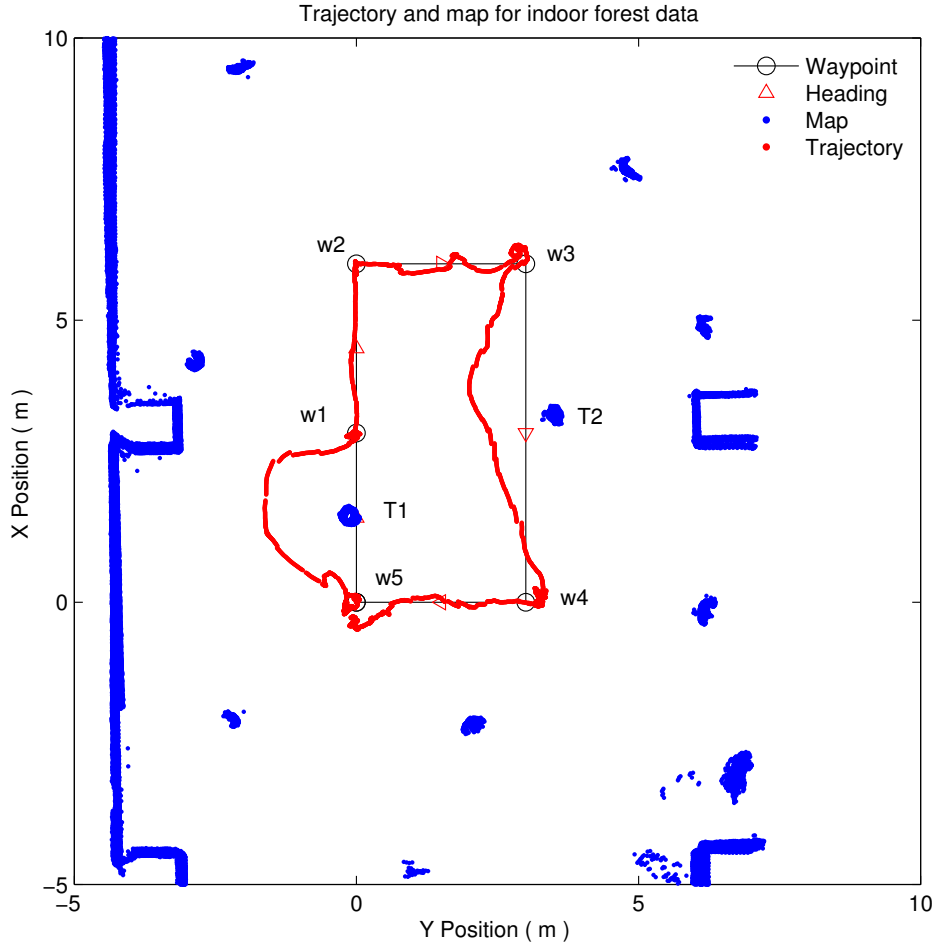


Figure 6.13: Consistent map with obstacle avoidance trajectory. The UAV is required to follow a rectangle shape trajectory with five waypoints (black circle w1-w5) along the way. At each corner of the rectangle, the UAV’s heading is shifted 90 degrees clock wise. The red dot trajectory is the real flight path which avoided obstacles (T1, T2) on the rectangle.

With path planning algorithm laid out in Section 6.3, we designed a flight test to verify the UAV navigation system including the online GraphSLAM and online path planning. The UAV is required to travel to five waypoints (w1-w5) which fall on a rectangle shape, which are labeled as black circles in Fig. 6.13. Once reaching a waypoint, a hover of 10 seconds is performed. At each corner of the rectangle, the UAV’s heading is shifted 90 degrees clock wise after the hovering. Without obstacles, the designed trajectory is a rectangle shape. However, as shown in Fig. 6.13, there are two obstacles (T1, T2) lying on the connected line of the waypoints. To reach the waypoints, the UAV must find other feasible path instead of the direct connection between the waypoints.

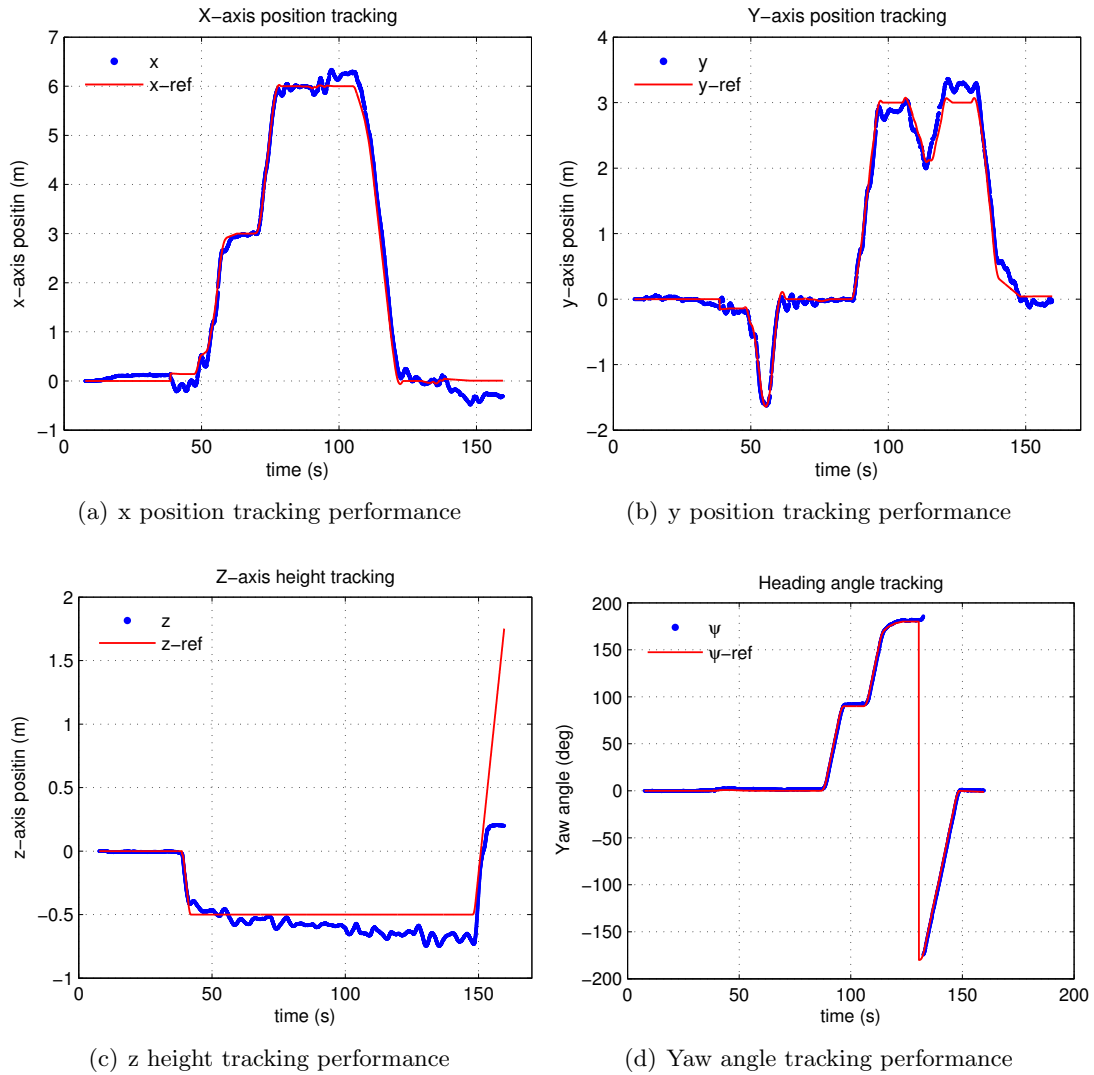


Figure 6.14: Onboard trajectory tracking performance with obstacle avoidance. The whole mission is fully autonomous, including take-off, waypoint navigation with online GraphSLAM and obstacle avoidance, and landing. Fig. 6.14(a) - 6.14(d) show the good tracking performance in  $x$ ,  $y$ ,  $z$ , and yaw directions, validating all the algorithms developed in this study: the consistent state estimation algorithm, the obstacle avoidance and the robust and perfect control.

The whole mission was fully autonomous, including take-off, waypoint navigation, obstacle avoidance, online GraphSLAM and landing. The flight data were recorded onboard and plotted in Fig. 6.13. The red dot trajectory is the real flight path which avoids obstacles nearby the rectangle. The blue dots plot is the accumulative plot of the environment, including trees, rectangle pillars and walls. The good shapes of walls and pillars demonstrate the consistency of the estimated trajectory. Fig. 6.14(a)-6.14(d) show the good tracking performance in  $x$ ,  $y$ ,  $z$  and heading  $\psi$  directions, validating all the algorithm developed in this study: the consistent state estimation algorithm, the obstacle avoidance and the robust and perfect tracking control technique.

## 6.6 Conclusion

We have presented the autonomous flight test results with online GraphSLAM and online path planning in GPS-denied environments in this chapter. First, to achieve the online GraphSLAM, a sliding window technique is applied to store the most recent poses in the last 5 seconds and optimize them with the GraphSLAM method. We present the system structure to describe the interaction between the front-end Kalman filter and the back-end optimization. At each time step, the initial pose from the Kalman filter is pushed into the sliding window first and optimized with the rest poses in the sliding window. The updated optimal pose is given back to the Kalman filter using specific update scheme to cope with the time delay between the front-end and the back-end. The same optimal pose is also added to a global pose graph in real-time, which will be further optimized after the flight mission to generate a globally consistent estimate.

Secondly, we also developed the online path planning algorithm, which consists of the global planner and the local planner. The global planner is an A\* path planning in the polar coordinate, which is responsible for finding the optimal intermediate waypoints from the current UAV position to the target position. The local path planner seeks to find the local optimal target which is collision free and closest to the waypoint from the global planner. A dynamic feasible trajectory is then obtained by solving the boundary value problem with the current UAV state and the local target.

Finally, autonomous flights based on a quadrotor platform are performed in various GPS-denied environments. Two kinds of GPS-denied environments are used: one is the indoor environment with synthetic trees and the other is a small scale forest. In the indoor environment, autonomous flights with online GraphSLAM and online path planning are successfully performed. Experimental results show that the online GraphSLAM algorithm significantly improves the consistency of the trajectory and the map simultaneously. The online path planning algorithm is able to provide feasible path references to avoid obstacles and maintain the original path route as much as possible. Flight tests in the real forest, consisting of trees of various orientation and uneven terrain conditions, also show promising improvement of the consistency. The autonomous flight tests verified all the algorithms developed in this study.

## Chapter 7

# Conclusions and Future Works

This Ph.D. study aims to realize the autonomous navigation of UAVs in GPS-denied environments. During the four-year study, we have made great efforts to the platform development and modeling, state estimation without GPS, SLAM algorithm implementation, and many autonomous flight tests. These developed techniques are modular enough to cater to new requirements, such as new sensing modalities. Although the flight tests have been performed mainly in foliage environments, a minimal change in the developed algorithms can make them applicable to UAV navigation in other GPS-denied environments, such as indoor offices or urban canyons.

### 7.1 Contributions

This research work has contributed to the development of UAV navigation systems in GPS-denied environment in the following aspects:

First of all, we have proposed a comprehensive methodology for designing and modeling small-scale UAVs. Platform design involves the bare platform configuration and the avionics system design. We have explored two configurations of platforms in this study: the coaxial helicopter and the quadrotor. The coaxial helicopter is promising due to its high lift-to-weight ratio and compact size, while the quadrotor stands out because of its simple mechanical structure and stable flight performance. To illustrate the modeling methodology, we make use of the coaxial helicopter. The nonlinear modeling techniques are applied to the roll, pitch, heave, and yaw dynamics with procedures to identify those parameters. The quadrotor, on the other hand, possesses a simple model,

serving as a good basis for designing control laws to track external reference arbitrarily. Chapters 2 and 3 are dedicated to this topic. The development of other UAV platforms can easily adopt the methods presented in these chapters.

Secondly, the real-time state estimation framework using odometry measurement is developed for UAV navigation in GPS-denied environments. The framework only needs an onboard IMU and a sensor measuring the odometry of the UAV. The odometry measurement may come from a laser range finder or a vision sensor. As a case study, Chapter 4 uses this framework to estimate the motion of the UAV in forest environments. The procedures of feature extraction and scan matching are presented in detail. Interested readers doing similar research can adopt this framework by changing the odometry method according to the sensor suite configuration.

Next, a consistent mapping system using GraphSLAM is developed in this study. The formulation of GraphSLAM as a nonlinear least squares problem has been addressed by other researchers, but there is little work discussing how to interpret the sensor data and build up the graph. Chapter 5 aims to answer these questions by giving the detailed procedures of building up the graph and optimizing it. The improved consistency of maps based on synthetic data and real flight test data have verified the techniques developed in this chapter.

Lastly, we have presented the successful navigation of UAVs in GPS-denied environments using online GraphSLAM and online path planning. Since GraphSLAM is an offline algorithm, using it for real-time UAV navigation demands tremendous effort. We present one solution consisting of local optimization using sliding window and global optimization to detect large loops. The sliding window method leads to a constant time local GraphSLAM whose states are still prone to drift, and thus a global optimization is used to further bound the position drift. For path planning, we have adopted a planning scheme with two layers: global planning and local planning. The global planning uses A\* algorithms based on the current scan of a laser range finder to generate a series of waypoints towards the target position. The boundary value problem is effectively solved using the Reflexxes Motion Library to generate the optimal trajectory in the local path planner. We have also discussed the software development for real-time onboard implementation. Multi-threading techniques are used to allocate the different algorithms in various threads for practical applications. Flight tests in this chapter incorporate



all the algorithms developed in the previous chapters. Successful flight tests with on-line GraphSLAM and online path planning are performed. The methods of integrating various algorithms into one functional navigation system are useful to other researchers.

## 7.2 Future Works

Although we have developed all the essential techniques for UAV navigation in GPS-denied environments and performed successful flight tests in this Ph.D. study, a lot of work are required to improve the performance of the overall system and make it more robust. The following topics are the focuses of our future works.

1. **Development of new 3D sensing techniques.** We have used a 2D laser range finder on the small UAV throughout this thesis. At the time of writing, there is news of a new product release of a 3D laser range finder weighing less than 300 g. Stereo vision suite with FPGA preprocessing is also under development. New sensing techniques require more and new functions integrated onto the UAV onboard navigation system. A robust and fast point cloud matching will be required in future to account for such developments.
2. **Multi-UAV cooperation in GPS-denied environment.** We have focused on single UAV navigation in this study. Because of the limits of the battery technology, the operation time of a single UAV is normally less than 30 minutes. To survey a large area, it would be difficult and inefficient to employ a single UAV, as its batteries would soon require repetitive charging. Instead, cooperative multi-UAV operation will greatly increase the surveying efficiency. Such cooperation requires a high-level autonomy of each UAV platform and map sharing among different UAVs.
3. **Operation in dynamic environment.** We have assumed the environment to be static during the UAV navigation. This assumption is strict since in many situations there are moving objects, either moving persons and cars, or shaking branches of trees. The capability to identify such dynamic objects will definitely expand the application of UAVs in our daily lives.

Finally, it is worth highlighting that UAV-related research is an interdisciplinary area requiring the efforts of people with different backgrounds. The UAV navigation system presented in this thesis would not have been possible without the genuine help and unstinting efforts of our fellow researchers in the NUS UAV group. Our collaborative teamworks have been demonstrated in two international competitions. The first event is the second AVIC Cup – International UAV Innovation Grand Prix, held in Beijing, China, in September 2013. The author has involved in developing a tail-sitter with reconfigurable wings [2] which won an new innovation award. The second event is the International Micro Air Vehicle (IMAV) competition, held in Delft, the Netherlands, in August 2014. In this event, the author has led the NUS UAV team consisting of 19 fellow researchers and won the first prize. Five UAVs with avionic systems developed in this thesis were used in the competition to perform four designated tasks simultaneously. Two of the competition tasks, i.e., the urban search and the indoor search and detection, required using GPS-less navigation techniques similar to the state estimation framework presented in this thesis. The experiences gained in these international competitions have been very rewarding and beneficial to all team members.

# Bibliography

- [1] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Stereo vision and laser odometry for autonomous helicopters in GPS-denied indoor environments. *Proc. SPIE 7332, Unmanned Systems Technology XI*, 733219, 2009.
- [2] K. Z. Ang, J. Cui, T. Pang, K. Li, K. Wang, Y. Ke, and B. M. Chen. Development of an unmanned tail-sitter with reconfigurable wings: U-lion. In *11th IEEE International Conference on Control Automation*, pages 750–755, 2014.
- [3] K. Arras, O. Mozos, and W. Burgard. Using Boosted Features for the Detection of People in 2D Range Data. In *IEEE International Conference on Robotics and Automation*, pages 3402–3407, April 2007.
- [4] K. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of Two 3D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, Sept. 1987.
- [5] A. Bachrach, S. Prentice, R. He, and N. Roy. RANGE-Robust autonomous navigation in GPS-denied environments. *Journal of Field Robotics*, 28(5):644–666, 2011.
- [6] T. Bailey. *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*. PhD thesis, University of Sydney, 2002.
- [7] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part II. *IEEE ROBOTICS & AUTOMATION MAGAZINE*, 13(3):108–117, 2006.
- [8] P. Besl and H. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb. 1992.

- [9] P. Biber and W. Straßer. The normal distributions transform: A new approach to laser scan matching. In *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, volume 3, pages 2743–2748. IEEE, 2003.
- [10] G. Bishop and G. Welch. An introduction to the Kalman filter. *Proc. of SIG-GRAPH*, 8:41, 2001.
- [11] S. Bouabdallah and R. Siegwart. Design and control of a miniature quadrotor. In K. Valavanis, editor, *Advances in Unmanned Aerial Vehicles*, volume 33 of *Intelligent Systems, Control and Automation: Science and Engineering*, pages 171–210. Springer Netherlands, 2007.
- [12] G. Cai, B. Chen, and T. Lee. An overview on development of miniature unmanned rotorcraft systems. *Frontiers of Electrical and Electronic Engineering in China*, 5(1):1–14, 2010.
- [13] G. Cai, B. M. Chen, and T. H. Lee. *Unmanned Rotorcraft Systems*. Springer, London, 2011.
- [14] G. Cai, B. Wang, B. M. Chen, and T. H. Lee. Design and implementation of a flight control system for an unmanned rotorcraft using RPT control approach. *Asian Journal of Control*, 85:95–119, 2013.
- [15] B. M. Chen. *Robust and  $H_\infty$  Control*. Springer, 2000.
- [16] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, IEEE International Conference on*, volume 3, pages 2724–2729, April 1991.
- [17] R. A. Chisholm, J. Cui, S. K. Y. Lum, and B. M. Chen. UAV LiDAR for below-canopy forest surveys. *Journal of Unmanned Vehicle Systems*, 01(01):67–68, 2013.
- [18] S. Choudhury, S. Arora, and S. Scherer. The Planner Ensemble and Trajectory Executive: A High Performance Motion Planning System with Guaranteed Safety. In *AHS 70th Annual Forum, Montreal, Quebec, Canada*, May 2014.
- [19] J. Cui, S. Lai, X. Dong, P. Liu, B. M. Chen, and T. H. Lee. Autonomous navigation of UAV in forest. In *International Conference on Unmanned Aircraft Systems*, pages 726–733, 2014.

- [20] M.-L. Doaa, M. A. M. Salem, H. Ramadan, and M. I. Roushdy. Comparison of Optimization Techniques for 3D Graph-based SLAM. *Recent Advances in Information Science*, 2013.
- [21] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287–300, 2002.
- [22] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part I. *Robotics Automation Magazine, IEEE*, 13(2):99–110, Jun. 2006.
- [23] D. W. Eggert, A. Lorusso, and R. B. Fisher. Estimating 3D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5-6):272–290, 1997.
- [24] J. Engel, J. Sturm, and D. Cremers. Scale-aware navigation of a low-cost quadcopter with a monocular camera. *Robotics and Autonomous Systems (RAS)*, 2014.
- [25] P. Fankhauser, S. Bouabdallah, S. Leutenegger, and R. Siegwart. Modeling and decoupling control of the coax micro helicopter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2223–2228, 2011.
- [26] F. Fraundorfer and D. Scaramuzza. Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications. *Robotics Automation Magazine, IEEE*, 19(2):78–90, Jun. 2012.
- [27] G. Gan, C. Ma, and J. Wu. *Data Clustering: Theory, Algorithms, and Applications*. ASA-SIAM Series on Statistics and Applied Probability. SIAM, 2007.
- [28] A. Georgiev and P. Allen. Localization methods for a mobile robot in urban environments. *IEEE Transactions on Robotics*, 20(5):851–864, October 2004.
- [29] G. Grisetti, R. Kuemmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping. In *IEEE International Conference on Robotics & Automation (ICRA)*, 2010.
- [30] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. Fast and accurate SLAM with Rao-Blackwellized particle filters. *ROBOTICS AND AUTONOMOUS SYSTEMS*, 55(1):30–38, Jan. 2007.

- [31] S. Grzonka, G. Grisetti, and W. Burgard. A Fully Autonomous Indoor Quadrotor. *IEEE Transactions on Robotics (T-RO)*, 8(1):90–100, Feb. 2012.
- [32] J. Guivant, F. Masson, and E. Nebot. Simultaneous localization and map building using natural features and absolute information. *Robotics and Autonomous Systems*, 40(2-3):79–90, 2002.
- [33] J. Guivant, E. Nebot, and S. Baiker. Localization and map building using laser range sensors in outdoor applications. *Journal of Robotic Systems*, 17(10):565–583, 2000.
- [34] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, Feb. 2008.
- [35] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31:216–235, Feb. 2012.
- [36] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO)*, 24(6):1365–1378, Dec. 2008.
- [37] F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012.
- [38] T. Kröger. *On-Line Trajectory Generation in Robotic Systems*, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, 2010.
- [39] R. Kümmerle. *State Estimation and Optimization for Mobile Robot Navigation*. PhD thesis, Albert-Ludwigs-University of Freiburg, Department of Computer Science, April 2013.
- [40] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g<sup>2</sup>o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.
- [41] J. W. Langelaan. *State estimation for autonomous flight in cluttered environments*. PhD thesis, Stanford University, March 2006.

- [42] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.
- [43] J. Leishman and S. Ananthan. Aerodynamic Optimization of a Coaxial Proprotor. In *62nd American Helicopter Society Annual Forum Proceedings*, number 1, 2006.
- [44] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, Jun 1991.
- [45] K. Liu, B. M. Chen, and Z. Lin. On the problem of robust and perfect tracking for linear systems with external disturbances. *International Journal of Control*, 74(2):158–174, 2001.
- [46] F. Lu and E. Miliot. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, 1997.
- [47] F. Lu and E. Miliot. Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. *Journal of Intelligent and Robotic Systems*, 18:249–275, 1997.
- [48] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. California, USA, 1967.
- [49] K. Madsen, H. Nielsen, and O. Tingleff. Methods for nonlinear least squares problems. Technical report, 2004.
- [50] M. Magnusson, A. Lilienthal, and T. Duckett. Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics*, pages 803–827, 2007.
- [51] M. Magnusson, A. Nuchter, C. Lorken, A. Lilienthal, and J. Hertzberg. Evaluation of 3D registration reliability and speed - A comparison of ICP and NDT. In *IEEE International Conference on Robotics and Automation*, pages 3907–3912, May 2009.
- [52] T. Masuda, K. Sakaue, and N. Yokoya. Registration and integration of multiple range images for 3D model construction. In *Proceedings of the 13th International Conference on Pattern Recognition*, volume 1, pages 879–883, 1996.
- [53] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2520–2525, 2011.

- [54] B. Mettler, M. B. Tischler, and T. Kanade. System identification of small-size unmanned helicopter dynamics. In *American Helicopter Society 55th Annual Forum Proceedings*, volume 2, pages 1706–1717, 1999.
- [55] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598, 2002.
- [56] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1151–1156, 2003.
- [57] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Y. Siegwart. A Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM. In *IEEE International Conference on Robotics and Automation*, 2014.
- [58] P. Nunez, R. Vazquez-Martin, J. del Toro, A. Bandera, and F. Sandoval. Natural landmark extraction for mobile robot navigation based on an adaptive curvature estimation. *Robotics and Autonomous Systems*, 56(3):247–264, 2008.
- [59] L. Paull, S. Saeedi, M. Seto, and H. Li. AUV Navigation and Localization: A Review. *IEEE Journal of Oceanic Engineering*, 39(1):131–149, 2014.
- [60] S. Ross, N. Melik-Barkhudarov, K. Shankar, A. Wendel, D. Dey, J. Bagnell, and M. Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1765–1772, May 2013.
- [61] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling*, pages 145–152, 2001.
- [62] B. Sabata and J. Aggarwal. Estimation of motion from a pair of range images: A review. *CVGIP: Image Understanding*, 54(3):309–324, 1991.
- [63] Z. Sarris. Survey of UAV applications in civil markets. In *The 9th IEEE Mediterranean Conference on Control and Automation*, 2001.



- [64] D. Scaramuzza, M. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. Lee, S. Lynen, M. Pollefeys, A. Renzaglia, R. Siegwart, J. Stumpf, P. Tanskanen, C. Troiani, S. Weiss, and L. Meier. Vision-Controlled Micro Flying Robots: From System Design to Autonomous Navigation and Mapping in GPS-Denied Environments. *IEEE Robotics Automation Magazine*, 21(3):26–40, Sept. 2014.
- [65] D. Scaramuzza and F. Fraundorfer. Visual Odometry Part I: The First 30 Years and Fundamentals. *IEEE Robotics & Automation Magazine*, 18(4):80–92, Dec. 2011.
- [66] K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller, and M. Suppa. Stereo vision based indoor/outdoor navigation for flying robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3955–3962, Nov 2013.
- [67] S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained MAV. In *IEEE International Conference on Robotics and Automation*, pages 20–25, May 2011.
- [68] S. Shen, N. Michael, and V. Kumar. Obtaining liftoff indoors: Autonomous navigation in confined indoor environments. *Robotics Automation Magazine, IEEE*, 20(4):40–48, 2013.
- [69] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, volume 8, pages 167–193. 1990.
- [70] M. Song, F. Sun, and K. Iagnemma. Natural landmark extraction in cluttered forested environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4836–4843, May 2012.
- [71] O. Sorkine. Least-Squares Rigid Motion Using SVD. Technical report, Courant Institute of Mathematical Sciences, New York University, Feb. 2009.
- [72] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 321–328, 2000.
- [73] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

- [74] S. Thrun and M. Montemerlo. The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.
- [75] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23:661–692, 2006.
- [76] M. Tischler and R. Remple. *Aircraft and rotorcraft system identification: Engineering Methods with Flight Test Examples*. AIAA, 2006.
- [77] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318. ACM, 1994.
- [78] B. Wang, B. M. Chen, and T. H. Lee. An RPT approach to time-critical path following of an unmanned helicopter. In *8th Asian Control Conference (ASCC)*, pages 211–216, May 2011.
- [79] F. Wang. *Indoor Navigation Systems for Unmanned Aerial Vehicles*. PhD thesis, National University of Singapore, 2014.
- [80] F. Wang, J. Cui, B. M. Chen, and T. H. Lee. A Comprehensive UAV Indoor Navigation System Based on Vision Optical Flow and Laser FastSLAM. *Acta Automatica Sinica*, 39(11):1889–1900, 2013.
- [81] F. Wang, J. Cui, B. M. Chen, and T. H. Lee. Flight Dynamics Modeling of Coaxial Rotorcraft UAVs. In *Handbook of Unmanned Aerial Vehicles*, pages 1217–1256. Springer Netherlands, 2014.
- [82] F. Wang, J. Cui, S. K. Phang, B. M. Chen, and T. H. Lee. A mono-camera and scanning laser range finder based UAV indoor navigation system. In *International Conference on Unmanned Aircraft Systems*, pages 694–701, May 2013.

- [83] S. Weik. Registration of 3D partial surface models using luminance and depth information. In *International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pages 93–100. IEEE, 1997.

# List of Author's Publications

## Refereed Journals

- [J1] R. A. Chisholm, **J. Cui**, S. K. Lum, and B. M. Chen, "UAV LiDAR for below-canopy forest surveys," *Journal of Unmanned Vehicle Systems*, vol. 01, no. 01, pp. 61-68, 2013
- [J2] F. Wang, **J. Cui**, B. M. Chen, and T. H. Lee, "A Comprehensive UAV Indoor Navigation System Based on Vision Optical Flow and Laser FastSLAM," *Acta Automatica Sinica*, vol. 39, no. 11, pp. 1889-1900, 2013
- [J3] F. Lin, K. Z. Y. Ang, F. Wang, B. M. Chen, T. H. Lee, B. Yang, M. Dong, X. Dong, **J. Cui**, S. K. Phang, B. Wang, D. Luo, K. Peng, G. Cai, S. Zhao, M. Yin, and K. Li, "Development of an unmanned coaxial rotorcraft for the DARPA UAVForge challenge," *Unmanned Systems*, vol. 1, no. 2, pp. 211-245, 2013

## Book Chapters

1. F. Wang, **J. Cui**, B. M. Chen and T. H. Lee, Flight dynamics modeling of coaxial rotorcraft UAVs, *Handbook of Unmanned Aerial Vehicles* (Edited by K. P. Valavanis and G. J. Vachtsevanos), Springer, pp. 1217-1256, 2014

## International Conferences

- [C1] K.Z. Ang, **J. Cui**, T. Pang, K.Li, K. Wang, Y. Ke, and B. M. Chen, "Development of an unmanned tail-sitter with reconfigurable wings: U-Lion," in *11th IEEE International Conference on Control Automation*, (Taichung, Taiwan), pp. 750-755, 2014

- [C2] **J. Cui**, S. Lai, X. Dong, P. Liu, B. M. Chen, and T. H. Lee, “Autonomous navigation of UAV in forest, ” in *2014 International Conference on Unmanned Aircraft Systems*, (Orlando, USA), pp. 726-733, 2014
- [C3] **J. Cui**, F. Wang, X. Dong, Z. Y. Ang, B. M. Chen, and T. H. Lee, “Landmark extraction and state estimation for UAV operation in forest,” in *Proceedings of the 2013 Chinese Control Conference*, (Xi’an, China), pp. 5210-5215, July 2013
- [C4] S. Zhao, X. Dong, **J. Cui**, Z. Y. Ang, F. Lin, K. Peng, B. M. Chen, and T. H. Lee, “Design and implementation of homography-based vision-aided inertial navigation of UAVs,” in *Proceedings of the 2013 Chinese Control Conference*, (Xi’an, China), pp. 5101-5106, 2013
- [C5] F. Wang, **J. Cui**, S. K. Phang, B. M. Chen, and T. H. Lee, “A mono-camera and scanning laser range finder based UAV indoor navigation system,” in *International Conference on Unmanned Aircraft Systems*, (Atlanta, USA), pp. 694-701, 2013
- [C6] **J. Cui**, F. Wang, Z. Qian, B. M. Chen, and T. H. Lee, “Construction and Modeling of a Variable Collective Pitch Coaxial UAV,” in *9th International Conference on Informatics in Control, Automation and Robotics*, (Rome, Italy), pp. 286-291, 2012
- [C7] F. Wang, S. K. Phang, **J. Cui**, G. Cai, B. M. Chen, and T. H. Lee, “Nonlinear modeling of a miniature fixed-pitch coaxial UAV,” in *American Control Conference*, (Montreal, Canada), pp. 3863-3870, 2012
- [C8] F. Wang, S. K. Phang, **J. Cui**, B. M. Chen and T. H. Lee, “Search and rescue: a UAV aiding approach” , in *Proceedings of the 23rd Canadian Congress on Applied Mechanics*, (Vancouver, Canada), pp. 183-186, 2011