Design and development of 3D visual mapping system for elevator shaft inspection

Ong, Eng Hui

2019

Ong, E. H. (2019). Design and development of 3D visual mapping system for elevator shaft inspection. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/87599>

<https://doi.org/10.32657/10356/87599>

**DESIGN AND DEVELOPMENT OF 3D VISUAL
MAPPING SYSTEM FOR ELEVATOR SHAFT
INSPECTION**

**ONG ENG HUI
SCHOOL OF MECHANICAL AND AEROSPACE
ENGINEERING
2019**

DESIGN AND DEVELOPMENT OF 3D VISUAL MAPPING SYSTEM FOR ELEVATOR SHAFT INSPECTION

ONG ENG HUI

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University in partial
fulfillment of the requirement for the degree of
Master of Engineering

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

22 May 2019

.....
Date

.....
Ong Eng Hui

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

22 May 2019

.....

Date

.....

Holden Li King Ho

Authorship Attribution Statement

Please select one of the following; *delete as appropriate:

*(A) This thesis **does not** contain any materials from papers published in peer-reviewed journals or from papers accepted at conferences in which I am listed as an author.

~~*(B) This thesis contains material from [x number] paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.~~

22 May 2019

.....

Date

.....

Ong Eng Hui

ACKNOWLEDGEMENT

I would like to express my gratitude to Assistant Professor Li King Ho Holden for his guidance and encouragement throughout the project timespan.

I wish to acknowledge the funding support for the project from Temasek Laboratories @ Nanyang Technological University.

Besides, I would also like to thank Temasek Laboratories @ Nanyang Technological University for providing space for testing and experiment.

Lastly, I would like to extend my appreciation to my friends and family who have been encouraging and supporting me throughout the project timespan.

TABLE OF CONTENTS

TABLE OF CONTENTS	ii
LIST OF FIGURES	vii
LIST OF TABLES	xii
ABSTRACT.....	xiii
1 Introduction.....	1
1.1 Background and Motivation	1
1.2 Objective and Scope	3
1.3 Thesis Chapter Organization.....	3
2 Literature Review	5
2.1 Introduction to Elevator	5
2.1.1 Traction Elevator	5
2.1.2 Elevator Defects.....	8
2.1.3 Traditional Elevator Shaft Inspection and Maintenance.....	10
2.1.4 Verticality of Elevator Shaft and Elevator Guide Rail	12
2.2 Past Research on Elevator Monitoring and Inspection	13
2.2.1 Maintenance Inspection	13
2.2.2 Verticality Inspection.....	15
2.3 Proposed Method for Elevator Shaft Inspection	16

3	Theory and Methodology	18
3.1	Localization of Scanning System in Elevator Shaft	18
3.1.1	Introduction.....	18
3.1.2	Robotic Operating System (ROS).....	19
3.1.3	Kalman Filter (KF)	21
3.1.3.1	KF Correction Update	22
3.1.3.2	KF Prediction Update	23
3.1.3.3	KF State Estimation Cycle	24
3.1.3.4	Extension of KF for Nonlinear Model System.....	25
3.1.3.4.1	EKF Formulation.....	25
3.1.3.5	Summary.....	27
3.1.4	Hardware Devices and Setup	27
3.1.4.1	Hardware Devices.....	27
3.1.4.2	Hardware Setup	29
3.1.5	Software Setup	32
3.1.5.1	Transformation	32
3.1.5.2	Ironsides Stereo Camera Odometry.....	34
3.1.5.3	MTi-7-DK Inertial Measurement	35
3.1.5.4	RPLIDAR A2M8 Laser Scan Data and Planar Position	42
3.1.5.5	TeraRanger Evo 60m Range Data	43

3.1.5.6	Pose Estimation	44
3.1.6	Experiment.....	46
3.1.6.1	Experiment Testbed.....	47
3.1.6.2	Experiment Procedure	49
3.2	3D Map Generation.....	57
3.2.1	Introduction.....	57
3.2.2	Smoothing of Scanning System Path Trajectory	57
3.2.3	3D Map Point Clouds Generation.....	59
3.2.3.1	Software Setup.....	60
3.3	Image Capture and Color Point Cloud Generation	61
3.3.1	Introduction.....	61
3.3.2	Panoramic 2D Image Capture	62
3.3.2.1	Hardware Devices and Setup.....	62
3.3.2.2	Software Setup.....	65
3.3.3	Experiment.....	65
3.3.4	2D RGB Images and 3D Point Cloud Fusion	67
3.3.4.1	Camera Calibration.....	67
3.3.4.2	Camera LIDAR Extrinsic Calibration	71
3.3.4.3	RGB Projection to 3D Point Cloud	75
4	Results and Discussions	78

4.1	Localization of Scanning System in Elevator Shaft	78
4.1.1	Results and Post Analysis	78
4.1.2	Conclusion	97
4.2	3D Map Generation.....	97
4.2.1	Result and Post Analysis.....	97
4.2.2	Conclusion	105
4.3	Image Capture and Color Point Cloud Generation	106
4.3.1	Panoramic 2D Image Capture.....	106
4.3.1.1	Result and Visualization.....	106
4.3.2	2D RGB Images and 3D Point Cloud Fusion	109
4.3.2.1	Result and Discussion.....	109
4.3.3	Conclusion	113
5	Conclusion and Future Works.....	114
5.1	Conclusion	114
5.2	Future Works for Improvements.....	116
	REFERENCES.....	118
	Appendix A	122
	Appendix B	123
	Appendix C	125
	Appendix D.....	126

Appendix E	127
Appendix F	129
Appendix G.....	131
Appendix H.....	140

LIST OF FIGURES

Figure 1: Type of dwelling by resident household in Singapore[1].	2
Figure 2: Flowchart of tasks being done in this thesis.	4
Figure 3: Simple working diagram of a traction elevator.	7
Figure 4: Detailed diagram of a gearless machine-room traction elevator[13].	8
Figure 5: Operational defects in elevator system.	9
Figure 6: Corroded elevator rope.	9
Figure 7: Water leakage in elevator shaft.	10
Figure 8: Technicians carry out inspection of elevator shaft on top of elevator car.	10
Figure 9: Plumb line hanging by the elevator shaft wall for verticality measurement[23].	12
Figure 10: Small scale experiment on UAV autonomous elevator rope inspection.	14
Figure 11: TLS Leica ScanStation C10 performing 3D scanning from the bottom of the elevator shaft.	16
Figure 12: Illustration of the prototype scanning system for elevator shaft inspection.	17
Figure 13: ROS communication diagram for a simple ground robot with laser scanner capable of perform mapping and path planning.	20
Figure 14: Summarized KF loop cycle algorithm.	25
Figure 15: Summarized EKF loop cycle algorithm.	27
Figure 16: Mounting frame.	29
Figure 17: Hardware setup on the mounting frame.	31
Figure 18: Hardware connection schematics of the scanning system.	32
Figure 19: Visualization of the placement of the coordinate frame of the sensors.	34
Figure 20: Odometry reporting message. (a) Original odometry reporting message by the ROS node, odometry frame complies ROS camera axis convention and it is fixed to the earth, camera_optical_link complies ROS camera axis convention and it is fixed to the camera; (b) Odometry reporting message by the modified ROS node, odometry frame complies ROS robot axis convention and it is fixed to the earth, base frame complies	

robot axis convention, camera_optical_link complies ROS camera axis convention and it is fixed to the camera, and it is statically fixed to the base frame.	35
Figure 21: Accelerometer output before and after calibration (Red: Z-axis, green: Y-axis, blue: X-axis).....	40
Figure 22: Resultant acceleration before and after calibration.	40
Figure 23: X-axis gyroscope output before and after calibration.	41
Figure 24: Y-axis gyroscope output before and after calibration.	41
Figure 25: Z-axis gyroscope output before and after calibration.....	42
Figure 26: Office hallway corridor to simulate elevator shaft as testbed.	47
Figure 27: Scanning system mounted on P3DX mobile robot.	49
Figure 28: 2D map generated from Mapper3, as seen on MobileEyes. The red circle represents the location of the mobile robot on the known 2D map.	50
Figure 29: ROS computational graph as seen on rqt_graph. Oval represents nodes, rectangular box represents namespace, arrow represents topic messages flow direction.....	51
Figure 30: Transforms tree of all sensors and robot frames as shown in rqt_tf_tree.....	52
Figure 31: Mobile robot moving to the End point autonomously as shown in MobileEyes client GUI software. The blue line represents the calculated path on the map by MobileEyes for the mobile robot to move from Start to End.....	53
Figure 32: Mobile robots moving to the End point in real world.	53
Figure 33: ROS computation graph as shown in rqt_graph GUI for post processing nodes. Oval is the node, inner box is topic and outer box is namespace.	56
Figure 34: Raw and filtered absolute trajectory output by EKF localization computation.	58
Figure 35: Raw and filtered absolute translation output by EKF localization computation.....	59
Figure 36: Raw and filtered absolute orientation output by EKF localization computation.	59
Figure 37: ROS computation graph for 3D map generation as seen on rqt_graph.....	61
Figure 38: Arducam MT9N001 with LS-81600 lens and Arducam USB shield.....	62
Figure 39: Hardware setup on the mounting frame with additional 3 cameras.	64
Figure 40: Hardware connection schematics of the scanning systems with additional 3 cameras.	64

Figure 41: Flowchart for the image capturing process of the scanning system.....	66
Figure 42: Pinhole camera model coordinate system.	67
Figure 43: Checkerboard corners detected by OpenCV <code>cv2.findChessboardCorners()</code> function from one of the cameras.....	70
Figure 44: (Left) Distorted image before calibration from one of the cameras; (Right) Undistorted image after calibration from the same camera.	71
Figure 45: Corner of the box projected by the laser scanner.	73
Figure 46: Laser from laser scanner projected onto the box as seen by IR filter removed camera. A blue tac is used to tag the location of the projected laser at the corner of the box as indicator.	74
Figure 47: Image point selected as indicated by the red dot.....	74
Figure 48: Projected laser points (green dots) onto the image after calibration.	75
Figure 49: Visible point cloud and hidden point cloud from a camera perspective.	76
Figure 50: Flowchart for 3D point cloud colorization.....	77
Figure 51: ATE plot for baseline configuration.....	80
Figure 52: RPE plot for baseline configuration.	81
Figure 53: Absolute trajectory plot for baseline configuration.....	81
Figure 54: Absolute translation plot for baseline configuration.	82
Figure 55: Absolute orientation plot for baseline configuration.....	82
Figure 56: ATE plot for realigned baseline configuration.....	84
Figure 57: Absolute trajectory plot for realigned baseline configuration.....	84
Figure 58: Absolute translation plot for realigned baseline configuration.	85
Figure 59: Absolute orientation plot for realigned baseline configuration.....	85
Figure 60: ATE plot for trajectories from EKF with Q , $0.01Q$ and $100Q$	87
Figure 61: RPE plot for trajectories from EKF with Q , $0.01Q$ and $100Q$	87
Figure 62: Absolute trajectory plot for trajectories from EKF with Q , $0.01Q$ and $100Q$	88
Figure 63: Absolute translation plot for trajectories from EKF with Q , $0.01Q$ and $100Q$	88

Figure 64: Absolute orientation plot for trajectories from EKF with Q , $0.01Q$ and $100Q$	89
Figure 65: ATE plot for trajectories from EKF with R , $100R$ and $0.01R$	90
Figure 66: RPE plot for trajectories from EKF with R , $100R$ and $0.01R$	91
Figure 67: Absolute trajectory plot for trajectories from EKF with R , $100R$ and $0.01R$	91
Figure 68: Absolute translation plot for trajectories from EKF with R , $100R$ and $0.01R$	92
Figure 69: Absolute orientation plot for trajectories from EKF with R , $100R$ and $0.01R$	92
Figure 70: ATE plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$	94
Figure 71: RPE plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$	94
Figure 72: Absolute trajectory plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$.	95
Figure 73: Absolute translation plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$	95
Figure 74: Absolute orientation plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$	96
Figure 75: Raw 3D point clouds map of the hallway.	98
Figure 76: Cleaned 3D point clouds map of the hallway compare with the real world image.....	99
Figure 77: (a) Scanning system with one laser scanner unable to scan corner resulted in shadowing effect in final point clouds; (b) Crisscross slanted multiple laser scanners can scan corner to minimize or eliminate shadowing effect.....	100
Figure 78: Feature to be measured for comparison between 3D point clouds map and real world to evaluate point clouds accuracy.	101
Figure 79: Colormap of the flatness of the point clouds of the floor. The color bar on the right shows the color corresponding the flatness error.	103
Figure 80: Colormap of the flatness of the point clouds of the wall. The color bar on the right shows the color corresponding the flatness error.	104
Figure 81: Floor flatness error distribution graph, the black curve represents normal distribution curve.	104
Figure 82: Wall flatness error distribution graph, the black curve represents normal distribution curve.	105
Figure 83: Overlap regions between two sets of subsequent hallway images captured by the scanning system.	106

Figure 84: 3D point cloud of the hall way shown in rviz. The semitransparent, clickable round interactive markers show the position where images are captured.	107
Figure 85: Images captured shown in a new window when the interactive marker at that position is clicked. The title of the window shows the 3D coordinate of the position where the images are captured in the 3D point cloud map.	108
Figure 86: RGB 3D point cloud of the hallway as seen on rviz.	110
Figure 87: RGB 3D point cloud of the hallway as seen on rviz in different perspective 1.	111
Figure 88: RGB 3D point cloud of the hallway as seen on rviz in different perspective 2.	111
Figure 89: RGB 3D point cloud of the hallway as seen on rviz in different perspective 3.	112
Figure 90: RGB 3D point cloud of the hallway as seen on rviz in different perspective 4.	112
Figure 91: RGB 3D point cloud of the hallway as seen on rviz in different perspective 5.	113

LIST OF TABLES

Table 1: List of hardware devices and corresponding description that is used in the scanning system. The more detailed specification of these hardware can be found in Appendix A to E.	29
Table 2: MTi-7-DK IMU sensor inertial data output and output rate.....	36
Table 3: Variance calculated from stationary MTi-7-DK inertial data.....	37
Table 4: IMU calibration parameters.	39
Table 5: Baseline configuration of robot_localization and laser_scan_matcher parameters for EKF estimation.	79
Table 6: ATE and RPE output for baseline configuration.	80
Table 7: ATE output for realigned baseline configuration.	83
Table 8: ATE and RPE output for trajectories from EKF with Q, 100Q and 0.01Q.....	86
Table 9: ATE and RPE output for trajectories from EKF with R, 100R and 0.01R.....	90
Table 10: ATE and RPE output for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$	93
Table 11: Comparison of chosen features between point clouds measurement and real measurement.	102

ABSTRACT

Elevator is an important vertical transportation system in Singapore as a highly urbanized country with limited geographical land. There are about 79.9% of the residents are staying in high-rise building. With population gradually increasing each year, due to limited land, more and more high-rise building is being built to accommodate the increased in population. Thus, elevator services becoming more and more important. However, more and more elevator in residential building has been plaguing with malfunctioning and erratic behavior which causes injuries or even death. This has led to tightening of elevator inspection and maintenance for public safety. The current inspection method involves labor intensive manual inspection, which compromises the safety of the inspector and causes delay in down time. In this thesis, the design and development of an 3D visual elevator shaft inspection system concept is proposed, which has potential to improve the yield and speeds up the elevator shaft inspection process. The proposed system is a scanning system that can be hoisted down the elevator shaft via pulley system to capture the 3D and 2D information of the elevator shaft, which eventually be used for offsite inspection. A prototype is built, and it is tested in a hallway simulated as a vertical elevator shaft. It is found out that the proposed concept method is technically feasible for elevator shaft inspection with added benefits, however there are some improvement that can be made before full deployment for real operation.

1 Introduction

1.1 Background and Motivation

Singapore is a small and urbanized island country in Southeast Asia, located between peninsular Malaysia and Indonesia. Although Singapore has almost zero natural resources as compared to neighbor countries, Singapore relies on exports to survive. Since 1965, the economy of Singapore has been growing rapidly as compared to its neighbor and become one of the strongest in terms of economy growth in Southeast Asia. With the rapid growth in economy, the relative Singapore currency exchange rate to its neighbor has also been increasing since then. Due to its strong currency, a lot of foreigners from China, Malaysia, Indonesia, Thailand, etc. are attracted to Singapore to make a living back to their home country. Most of them are registered as Singapore Permanent Resident or even obtained Singapore Citizenship to settle down in Singapore. As such, Singapore total population has also been increasing gradually in the last decades. According to Singapore Department of Statistics, the total population in year 2017 has reached 5.6123 million, with 1.26 million of resident households in year 2016[1]. Due to limited geographical land, Housing and Development Board (HDB) who manages public housing in Singapore has built and provided a lot of low cost apartment for resident households. In 2017, there is around 79.9% of resident households live in HDB apartment[1]. As such, elevator as a vertical transportation has become important in high-rise landscape. Today, there are a total of more than 60000 passenger elevators in Singapore[2].

Figure 1: Type of dwelling by resident household in Singapore[1].

However, the malfunctioning and erratic behavior of elevators across HDB apartment has been plaguing the community of Singapore in the past recent years. Erratic behavior such as jerking motion, sudden raise and drops, malfunction elevators doors has been occurred throughout the HDB apartment elevators especially the old ones. Such events have caused minor injury, major injury such as severing limb and even death. These spates of elevators malfunctioning have led to tightening of maintenance and safety regulations of elevators by Building and Construction Authority (BCA)[3-9]. According to our industrial partner, E M Services Pte Ltd (EMS) who manages over 9 Town Councils in Singapore, the current inspection of elevator shaft is carried out directly inside the elevator shaft by inspection workers using traditional tools and gadgets. The traditional ways of elevator shaft inspection are time consuming which causes delays in elevator down time. Besides, there is also workplace safety hazard when one must climb into elevator shaft to perform inspection. To improve the yield and speed up the elevator shaft inspection process, and to reduce the workplace hazards encountered by the inspection workers, EMS has been looking for solutions to achieve such objective.

1.2 Objective and Scope

According to EMS, they are seeking for solutions in 2 aspects of inspection, pre-installation inspection and post-installation inspections. In pre-installation inspection, the measurement of verticality/uprightness of elevator shaft is required before installing elevator during building construction. The measurement of the verticality of the elevator shaft is to ensure the walls are aligned to each other in an acceptable range, such that the elevator can be installed safely and with ease. It has a direct correlation with the elevator operation in terms of reduction in friction and wear and tear issue. On the other hand, the post-installation inspection is to check for any abnormality in the shaft, such that maintenance can be perform as soon as possible to minimize the occurrence of the malfunctioning and erratic behavior of the elevator which compromises the safety of residents.

The objective of this project is to develop an elevator shaft inspection system that can measure verticality of elevator shaft in pre-installation inspection, and to improve and speed up the traditional method of post-installation inspection with the consideration of workplace safety of inspection workers.

1.3 Thesis Chapter Organization

This thesis shows a new innovative way to carry out elevator shaft inspection which can improve upon the current traditional inspection method. Multiple tasks are being done systematically in this thesis in order to complete the work. The results of each tasks are carried forward to the next task. The flowchart of the tasks in systematic order can be seen in Figure 2.

Figure 2: Flowchart of tasks being done in this thesis.

Chapter 2 introduces the principle of elevator, defects that can be found in elevator and pass researches on elevator monitoring and inspection. At the end of Chapter 2, a brief proposed method for elevator shaft inspection is discussed. Chapter 3 discusses the theory and methodology of each tasks in order as shown in Figure 2, while Chapter 4 presents the results and discussions of each task in order. Chapter 5 concludes the thesis with future works follow up for improvement.

2 Literature Review

2.1 Introduction to Elevator

Elevator can be categorized into few types with different hoisting mechanism. In Singapore, 4 types of elevator can be found, traction system, hydraulic system, pneumatic system and climbing system[10].

Traction system uses electric motor to roll the rope attached to the elevator to allow ascend and descend of the elevator car, like a pulley system. Hydraulic system uses hydraulic power to push the piston attached to the elevator car in a cylinder to raise the car, while opening the valve in the cylinder to remove hydraulic fluid allows the car to descend by its own weight. Pneumatic system uses vacuum power which is located on the top of the car to move the elevator up, while opening the valve the pressurized the shaft enables the elevator car to descend by its own weight. Climbing system has its own propulsion system to move the car up and down. The propulsion system can be in form of electrical engine or combustion engine.

According to EMS, most of the elevators they are servicing in Singapore HDB apartment and high-rise building are powered by traction system. In this section, traction type elevator is further discussed.

2.1.1 Traction Elevator

Traction type elevator can be categorized into 2 sub categories, machine-room traction elevator and machine-room-less traction elevator. Machine-room traction elevator has a dedicated room at the top of the elevator shaft for control panel and motors, while Machine-room-less traction elevator does not have dedicated room for control panel and motor. They are instead

usually installed in the hoistway. One of the advantages of machine-room-less elevator compared to machine-room elevator is space saving as they don't require space for machine room. However, the downside is that it is harder to maintain as the control panel and motor is harder to reach as compare to machine-room elevator. As reported by EMS, machine-room-less and machine-room traction elevator are used in low-rise and high-rise building they are serving respectively.

The traction system of the traction elevator can be categorized into geared and gearless traction system. In geared traction system, the traction sheave is attached to a speed reduction gear box. The gear box is then drove by a motor which turns drive the traction sheave to move the elevator. In gearless traction system, the traction sheave is directly connected to the motor. The motor is then drove the traction sheave to move the elevator without going through speed reduction gear box. One of the advantages of geared traction system is that it requires less powerful motor to turn the traction sheave. Geared traction elevator usually operates at 125 to 500 feet per minute and used to carry heavy load. On the other hand, the gearless traction elevator is more efficient than geared traction system due to the absent of reduction gear. Gearless traction elevator usually operates at higher speed of up to 2000 feet per minute and usually found in high rise building. Geared and gearless traction system can be found on both machine-room and machine-room-less traction elevator, but gearless traction system with smaller size motor is usually found in machine-room-less traction elevator due to compact size.

In all the cases, they operate with the same principle. One end of the hoist ropes is attached to the elevator car, looped around the traction sheave, and the other end of the hoist ropes is attached to a counterweight. The friction between the groove in the traction sheave and the hoist ropes allows the transmission of lift force to the elevator. The weight of the elevator and counterweight ensures the hoist ropes to sit in the groove of the traction sheave[11]. When the

motor turns, the elevator goes up and counterweight goes down on their own rails, and vice-versa. The counterweight usually weighs around the weight of the elevator car plus 40% of the car capacity load[12]. The counterweight balances the both end of the traction sheave to ease the motor to raise and lower the elevator car, hence reduces the amount of energy needed to drive the motor. Figure 3 and Figure 4 show a simple diagram and detailed diagram of a traction elevator respectively.

Figure 3: Simple working diagram of a traction elevator.

Figure 4: Detailed diagram of a gearless machine-room traction elevator[13].

2.1.2 Elevator Defects

Elevator is a complex system which consists of numerous number of components which works in tandem such that the elevator can operate normally. One fault in one component can lead to catastrophic failure to the whole elevator system. Figure 5 shows the common operational defect which can be found in elevator system[14].

To prevent catastrophic failure which leads to elevator accidents, early sign of any defect in any of the components must be readily detected and responds correspondingly. One of the biggest enemies to the elevator is rust or corrosion. Corrosion weakens the structural strength of the material and causes early failure under load such as breakage of elevator rope[15]. Figure 6 shows the corrosion in elevator rope[16]. Besides, water stain is also a bad indicator of the health of the elevator shaft as it shows sign of water leakage occurs in the system. Water leakage is bad

for elevator shaft, it can cause electrical short circuit, flooding in elevator pit, etc. It also creates moist environment which promotes the growth of mold and corrosion in elevator structure[17, 18].

Figure 7 shows water leakage in elevator shaft[19].

The above mentioned are only some of the examples of early defects that can occur in elevator shaft. There are still many different early defects which can be detected before failure occurs.

Operational defects in elevator system				
Travelling performance	Machine room	Hoistway and elevator pit	Elevator car	Elevator lobby
<ol style="list-style-type: none"> Ride quality <ul style="list-style-type: none"> Car vibration Jerky movement Car acceleration <ul style="list-style-type: none"> Long waiting time Overspeed car ride Elevator breakdown <ul style="list-style-type: none"> Stop operating 	<ol style="list-style-type: none"> Room condition <ul style="list-style-type: none"> Excessive heat Dimming room Dirty room Controller <ul style="list-style-type: none"> Exposed wires Burned relay contacts Traction sheave drive <ul style="list-style-type: none"> Drive sheave grooves Wearing undercut grooves Machine brake <ul style="list-style-type: none"> Fail to operate Uneven surface Burn marks on brake pad Overspeed governor <ul style="list-style-type: none"> Fail to operate Concrete slab <ul style="list-style-type: none"> Abrasions marks on side on slab Water exposure <ul style="list-style-type: none"> Leaking ceiling 	<ol style="list-style-type: none"> Accessibility <ul style="list-style-type: none"> Inadequate accessibility Suspension ropes <ul style="list-style-type: none"> Breakage of ropes Safety gears <ul style="list-style-type: none"> Fail to operate Car frame <ul style="list-style-type: none"> Plunged to buffer Counterweight <ul style="list-style-type: none"> Components detached from steel frame Pulley bearing damage Elevator pit <ul style="list-style-type: none"> Leaking pit's concrete floor Dirty pit area 	<ol style="list-style-type: none"> Misalignment <ul style="list-style-type: none"> Car unaligned with landing floor Door <ul style="list-style-type: none"> Jammed door Slammed door Dirty door sill Dented jamb Scratch marks Gap between doors Door sensor <ul style="list-style-type: none"> Unable to detect doors Ceiling panel <ul style="list-style-type: none"> Loose ceiling Floor <ul style="list-style-type: none"> Damaged floor covering Corroded substrate Wall <ul style="list-style-type: none"> Corroded wall Lighting <ul style="list-style-type: none"> Bulb faulty Cat accessories <ul style="list-style-type: none"> Loose panel Faulty button Fan <ul style="list-style-type: none"> Faulty fan 	<ol style="list-style-type: none"> Call button <ul style="list-style-type: none"> Inoperative call button

Figure 5: Operational defects in elevator system.

Figure 6: Corroded elevator rope.

Figure 7: Water leakage in elevator shaft.

2.1.3 Traditional Elevator Shaft Inspection and Maintenance

According to EMS, currently the inspection is done via manual labor. The elevator technicians must work inside the elevator shaft during the inspection and maintenance. The elevator technicians carry out the inspection of the components in the elevator shaft by riding on top of the elevator car. The elevator car is then move slowly along the hoistway such that the whole elevator shaft can be inspected. Figure 8 shows the technicians carry our inspections by riding on top of elevator car.

Figure 8: Technicians carry out inspection of elevator shaft on top of elevator car.

However, there are some problems and challenges encountered with the current traditional method of elevator inspection and maintenance. According to survey on Singapore industry players[14], one of the problems and challenges is the lack of elevator inspection and maintenance reports from the technicians. In their routine elevator inspection and maintenance, neither the detailed descriptions or photos of system's fault are recorded, rendering difficulty in achieving agreement between building managers and contractors in case of replacing faulty components. Besides, limited availability of skilled manpower and lack of knowledgeable manpower is also one of the problems. As the elevator is a very specialized system, there are limited qualified technicians in the market according to the survey. Oftentimes, the technicians are lacking knowledge in detecting the early sign of wearing of elevator components in the elevator shaft. Furthermore, the manual inspection of elevator shaft is time consuming and in result causes long downtime. As a country full of high-rise building where elevator is a necessary form of vertical transport, long outage of elevator is not preferable as it impedes the ability for the people to travel along the building. Last but not least, the safety of the technicians who will be working in elevator shaft are compromised. Other than accidents encountered by elevator passengers, there are also reports on elevator accident during maintenance and inspection. According to the Centre of Construction Research and Training (CPWR), with limited access to US Bureau of Labor Statistics CFOI Research File, 1992-2009, there was approximately 263 deaths while working in or near elevator[20]. The death and injuries of workers in elevator can be traced to one of the factors including electric shock, struck by components in elevator, falling, caught between objects, asphyxiation, etc.[21]

2.1.4 Verticality of Elevator Shaft and Elevator Guide Rail

Verticality of a structure is important in building especially for high rise building. This also applies to elevator shaft. Before installing elevator, the verticality of the elevator shaft is being investigated as verticality is the main parameter to ensure safe and stable elevator operation. The verticality of the elevator shaft directly affects verticality of the elevator guide rail after installation. Verticality error in guide rail causes vibration and wear and tear to the elevator car. It also affect the comfort and stability of the elevator operation[22].

The traditional way of inspecting the verticality of the elevator shaft and guide rail is also done manually. A plumb line or plummet with weight attached at the end is hanged by the wall or guide rail on higher level like a pendulum. The inspector then measures the deviation of the wall or guide rail from the plumb line directly using measuring tape or ruler[22, 23]. Such manual labor method is time consuming, less efficient, and prone to error. It also compromises the safety of the inspector as the inspector must go into the elevator shaft to do measurement.

Figure 9: Plumb line hanging by the elevator shaft wall for verticality measurement[23].

2.2 Past Research on Elevator Monitoring and Inspection

2.2.1 Maintenance Inspection

Due to rapid development of elevator industry, new methods concerning elevator maintenance inspection become more concerned. There are several researchers have proposed several different methods on improving the monitoring and inspection system of elevator.

P. Seung-Tae et al. has proposed a risk-based inspection for effective elevator maintenance[24]. The risk-based inspection utilizes the combination of the probability of the components failure based on statistics and the consequences of such component failure occur to systematically decide the next step for elevator failure risk reduction. It is evaluated by using Korean disaster and failure statistic data, and the outcome states that it offers an effective method for elevator maintenance. The author only describes the approach to decide the following actions for maintenance, however he does not describe the method to monitor the elevator systems.

Several researchers have proposed using wireless sensor network (WSN) or Internet of Things (IoT) method to monitor the condition of the elevator system[25-29]. *E. Dawei et al.*, *Z. Xiangqun*, *F. Qizhi et al.* and *T. Shuilin et al.* proposed remote elevator monitoring system based on embedded sensor system and wireless communication system such as GPRS, ZIGBEE, etc. *I. Skog et al.* proposed using a non-invasive, low-cost wireless sensor node to remotely monitor condition of the elevator. With the information such as temperature, vibration, speed, acceleration, noise, etc. collected by the WSN, the condition of the elevator can be determined remotely. Such method allows remotely monitor the condition of elevator system in real time, however it does not produce a direct result on the condition of the elevator. Moreover, the WSN cannot detect the early sign of defect such as water seepage, rust, etc. Such defect can only be detected during visual inspection.

There are researchers who implemented the use of robotics and computer vision to automate the elevator inspection. *Y. Orhan et al.* has developed a system based on image processing for detection of wear in elevator guide rail[30]. The author uses cameras to capture the image of the guide rail and perform image processing to detect the wear on the guide rail. Although the monitoring system can monitor in real time, the cameras are fixed onto the elevator car, looking only at the guide rail, hence it is unable to monitor other parts of the elevator system. Such method also requires installation of additional infrastructure onto elevator, which is usually undesirable. *H. K. Tsun et al.* has proposed the use of Unmanned Aerial Vehicle (UAV) to visual inspect elevator rope[31]. The author uses visual based navigation and control to detect, track and inspect the elevator rope with UAV with camera and light source onboard autonomously. Figure 10 shows the small-scale experiment on UAV elevator rope inspection using Parrot AR Drone done by the author. Using drone for inspection in elevator shaft is very flexible as it does not require extra infrastructure. However, elevator shaft with low light condition, unpredictable air turbulence and narrow area poses challenges for the drone to navigate. Furthermore, there is a risk where the drone might damage the property if it is out of control.

Figure 10: Small scale experiment on UAV autonomous elevator rope inspection.

2.2.2 Verticality Inspection

Due to rapid development in elevator industry, new method concerning verticality of elevator shaft and guide rail is becoming more concerned. Several researchers have proposed several methods to improve the measuring and detection process.

L. Wenang et al. has developed an autonomous climbing robot that climbs along the guide rail. The climbing robot is equipped with laser to measure the linearity of the guide rail[32]. *Y. Yifeng et al.* has developed a system that is also moving along the guide rail but powered by traction system[22]. The system is equipped with tilt sensors and acceleration sensor to detect the verticality, straightness and step deviation of the guide rail based on the vibration and tilt angle obtained. Both methods have yields more efficient and better accuracy than traditional method. However, the components inside the elevator, i.e. the guide rail must be used to move the systems, which is usually undesirable for external object to in contact with the components. *N. Naoaki et al.* has proposed a method to estimate the deformation in guide rail by measuring operational acceleration data[33] by attaching acceleration sensor onto the elevator car. The down side of this method is it can only be implemented after installing the elevator into the shaft, which is good for maintenance inspection.

K. Ludovit has evaluated the use of terrestrial laser scanner to measure the verticality of elevator shaft[23]. The author scans the elevator shaft by using TLS Leica ScanStation C10 to perform 3D scanning from the bottom of the elevator shaft. It is reported that the use of laser scanner is able to speed up the in-situ work and also it is accurate enough for verticality measurement.

Figure 11: TLS Leica ScanStation C10 performing 3D scanning from the bottom of the elevator shaft.

2.3 Proposed Method for Elevator Shaft Inspection

The proposed method for elevator shaft inspection will be discussed in this section. The proposed method will serve two functions, inspect the condition of elevator shaft and inspect the verticality of the elevator shaft.

In this project, a prototype scanning system will be developed for elevator shaft inspection. The scanning system consists of cameras and Light Detection and Ranging system (LIDAR). The cameras are used to capture the images around the elevator shaft for visual inspection, while the LIDAR is used to capture the 3D space or point clouds of the elevator shaft space. The scanning system will be delivered into the elevator shaft by using hoisting mechanism such as motorized pulley system or by attaching at the bottom of moving elevator car. Figure 12 shows the illustration of the proposed method for elevator shaft inspection. The scanning system captures the 2D and 3D information of the surrounding elevator shaft while it is being lowered or raised by the hoisting mechanism. After capturing 2D and 3D information, post processing is performed to construct the 3D structure of the elevator shaft and to stitch the 2D images together with the 3D point clouds. After post processing, the result of the product will be used for analysis of the condition of the elevator shaft. Additional sensors or equipment will also be installed onto the scanning system such that it is able to work as intended such as local light source to lit up the elevator shaft for the

camera to capture properly, inertial measurement unit (IMU) for localization in the elevator shaft, etc.

Figure 12: Illustration of the prototype scanning system for elevator shaft inspection.

The 2D images is used for visual inspection like the traditional inspection to seek for early sign of defects or anomalies in the elevator shaft manually. Image processing or artificial intelligence (AI) detection can be performed to automate and speed up the visual inspection process. The 3D point cloud is used for verticality inspection. By importing the 3D point cloud into commercial-off-the-shelf software (COTS) such as *CloudCompare*, etc., the geometrical dimension of the elevator shaft can be determined. The 3D point clouds can also be used to generate Building Information Modelling (BIM) 3D model of the elevator. Besides, the 3D point cloud can provide added layer of 3D detailed information on top of 2D images for better representation of the elevator shaft for further analysis. This study only evaluates the feasibility of using such scanning system for maintenance and verticality inspection, further analysis of defect in elevator shaft is not discussed.

3 Theory and Methodology

3.1 Localization of Scanning System in Elevator Shaft

3.1.1 Introduction

This section concerns about the localization of the scanning system in the elevator shaft. Localization is important in robotics such that it can establish its own position and orientation within its frame of reference. With a proper localization, the robot can perform obstacle avoidance, path planning and in this case, map building.

To achieve a good localization, sensors are used to determine the internal state and external state of the robot. Internal state sensors are used for control of the internal states of the robot such as acceleration, velocity, etc. Example of internal state sensors are accelerometer, gyroscope, encoders, etc. External state sensors are used to perceive the surrounding of the robots and react intelligently. Example of external state sensors are camera, laser range finder, global positioning system (GPS), etc. The selection of these sensors is dependent to the environment as not all the sensors are able to perform in every situation. For instances, GPS can provide global position of the robot, however it cannot work in an indoor environment like inside the elevator shaft. Hence, sensors other than GPS such as vision-based positioning sensor must be used to replace GPS. Most of the time, multiple sensors are fused together and used simultaneously for state estimation in robotics. Fusing multiple sensors allows the increase in reliability of sensors and reduction in uncertainty thus increase the accuracy of state estimation.

In this section, the methodology for designing and developing of the inspection scanning system will be discussed. A simulated environment experiment for localization testing of the scanning system will also be discussed.

3.1.2 Robotic Operating System (ROS)

ROS is an open source, middleware for robotic system. It is not an operating system, rather it provides a structured communications layer above the host operating system of a heterogeneous computer cluster[34]. It provides services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes and package management[35]. The main client of ROS is written in C++, Python and Lisp programming languages. It is adapted towards UNIX-like system due to heavily dependent to large collections of open source software dependencies. Hence, it is widely supported on Ubuntu Linux. ROS has been widely used in a lot of research and commercial platform due to its flexibility in development such as TurtleBot, Pioneer 3-DX, ABB Manipulators, etc. In year 2012, National Aeronautics and Space Administration (NASA) has used ROS to power Robonaut 2, a humanoid robot on International Space Station (ISS)[36].

ROS is built around peer-to-peer (P2P) architecture that are loosely coupled with ROS communication infrastructure. Each component, sensor and computer in an ROS P2P network can communicate between each other directly, synchronously or asynchronously. The basic communication concepts of ROS are *Nodes*, *Topics*, *Services*, *Messages* and *Master*. *Nodes* are processes that perform computation. One robot usually contains several *Nodes*, such as one node controls wheel motor, one node control localization, one node control path planning, one node control camera, etc. Each of these *Nodes* upon initialization, it will register itself to *Master*. *Master* provides name registration and lookup to *Nodes* such that they can find each other in the network

to communicate. *Topics* are streams of data with a unique name. *Nodes* can send out and receive data by publishing and subscribing to the *Topics* respectively. By publishing and subscribing to the *Topics*, the *Nodes* can communicate asynchronously. *Messages* is a data structure used in ROS that is sent through *Topics*. For instances, *sensor_msgs/LaserScan* is data structure for 2D laser scanner. *Services* is a request-reply type communication in ROS, where the *Nodes* can communicate synchronously between them. Figure 13 shows a simple ROS communication diagram for a ground robot with laser scanner capable of perform mapping and path planning.

Figure 13: ROS communication diagram for a simple ground robot with laser scanner capable of perform mapping and path planning.

ROS has built in several tools that can aid in robotic development. For examples, *rviz* allows visualization of large varieties of datatypes, such as images from camera, point clouds from laser scanner, position of robot in space, trajectories of robots, etc. *rqt* tools has built in different tools for visualization such as *rqt_graph* to show *Nodes* tree, *rqt_plot* for graph plotting of

datatypes such as velocity with respect to time, etc. *rosvbag* allows recording of all the *Topics* for later use such as post processing, etc.

There is abundance of software libraries in ROS repository that are maintained by the ROS community. They comprise of researchers around the world who programmed their research algorithm for ROS as an open source software for the public to use such as Simultaneous Localization and Mapping (SLAM) algorithm, manufacturer who programmed driver of their product for ROS usage such as camera driver, etc. The abundance of open source software in ROS allows flexibility for the developers to integrate hardware and software algorithm together with minimal effort.

In this project, ROS will be used for the development of the scanning system for elevator shaft inspection. The required software libraries that can be found in ROS repository that are well maintained will be used to the greatest extent to minimize the effort to rewrite a new driver or algorithm.

3.1.3 Kalman Filter (KF)

KF is an efficient filter for states estimations in a linear dynamic system from sequence of noisy measurement data. It was proposed by *R. E. Kalman* in 1960 for a recursive solution to the discrete data linear filtering problem[37]. Since then, due to rapid advancement in digital computing, KF has been extensively research and applied in many applications such as navigation, tracking, image processing, etc. In this case, KF is also used in multiple sensors fusion for pose estimation in robotics.

The formulation and algorithm of KF is widely known[38-40]. The equation of KF cycle fall into two groups, time update equations and measurement update equations. In time update

equations, KF predicts into future the current state and error covariance estimates to obtain estimates for the next measurement update time step. In measurement update equations, KF incorporates new measurement into the previous predicted estimates to obtain improved estimates for the next cycle. Time update is also called prediction update, while measurement update is also called correction update.

3.1.3.1 KF Correction Update

KF model assumes the state at time $k + 1$ is grew from time k . Given a system represented as a linear discrete stochastic process, the state space model can be represented as the following equation.

$$x_{k+1} = Ax_k + Bu_k + w_k \quad (1)$$

Where x is m state vector, A is $m * m$ state transition matrix from time k to time $k + 1$, B is $m * p$ input transition matrix from time k to time $k + 1$, u is p input vector, w is process noise vector.

The measurement model can be represented as the following equation.

$$z_k = Hx_k + v_k \quad (2)$$

Where z is n measurement vector, H is $n * m$ measurement matrix, v is measurement noise vector.

w and v are assumed to be uncorrelated, zero mean, white Gaussian noise with covariance Q and R respectively.

$$Q = E(w_k w_k^T) \quad (3)$$

$$R = E(v_k v_k^T) \quad (4)$$

Where $E(x)$ denotes expectation of x .

The error between the actual and estimated state can be defined as follows.

$$\tilde{x}_{k+1} = x_{k+1} - \hat{x}_{k+1} \quad (5)$$

Where $\tilde{x}$ is the estimated state error, x is the actual state, $\hat{x}$ is the estimated state.

The estimated state at time $k + 1$, $\hat{x}_{k+1}$ is written as the linear combination of estimated state at time k , $\hat{x}_k$ and the difference between an actual measurement, z_k and predicted measurement, $H\hat{x}_k$, with Kalman gain, K_k .

$$\hat{x}_{k+1} = \hat{x}_k + K_k(z_k - H\hat{x}_k) \quad (6)$$

The covariance of the estimated state error, P_{k+1} can be written as follows.

$$P_{k+1} = E(\tilde{x}_{k+1}\tilde{x}_{k+1}^T) \quad (7)$$

The Kalman gain, K_k is chosen such that the covariance of the estimated state error, P_{k+1} is to be minimized. To calculate for suitable K_k , equation (6) is first substituted into equation (5). After that, the result is substituted into equation (7). By taking derivative of the result of P_{k+1} with respect to K_k and equating it to zero, K_k for minimizing P_{k+1} can be obtained as follows.

$$K_k = P_k H^T (H P_k H^T + R)^{-1} \quad (8)$$

By substituting equation (8) back to derived equation (7) from the previous step, P_{k+1} can be obtained as follows.

$$P_{k+1} = (I - K_k H) P_k \quad (9)$$

3.1.3.2 KF Prediction Update

In prediction update, the state space model is as follows.

$$x_k = A x_{k-1} + B u_{k-1} + w_k \quad (10)$$

The predicted estimated state, $\hat{x}_k$ can be written as follows.

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_{k-1} \quad (11)$$

Hence, the estimated error between the actual and predicted estimated state, $\tilde{x}_k$ and covariance of the estimated predicted error, P_k can be written as such.

$$\tilde{x}_k = x_k - \hat{x}_k \quad (12)$$

$$P_k = E(\tilde{x}_k \tilde{x}_k^T) \quad (13)$$

By substituting equation (10) and equation (11) into equation (12), then substitutes the result into equation (13), P_k can be obtained as follows.

$$P_k = AP_{k-1}A^T + Q \quad (14)$$

3.1.3.3 KF State Estimation Cycle

As a summary, the KF is first fed with user input initial estimated state, $\hat{x}_{k-1}$ and initial error covariance, P_{k-1} into prediction update equations to obtain predicted estimated state, $\hat{x}_k$ and predicted error covariance, P_k . $\hat{x}_k$ and P_k are then fed into correction update equations to obtain Kalman gain, K_k , and subsequently update the estimated state, $\hat{x}_{k+1}$ and improved error covariance, P_{k+1} . $\hat{x}_{k+1}$ and P_{k+1} are then fed into next KF loop cycle to complete the recursive filter. Figure 14 shows the summarize diagram for KF loop cycle algorithm.

Figure 14: Summarized KF loop cycle algorithm.

3.1.3.4 Extension of KF for Nonlinear Model System

KF can be used as state estimator when the process system is linear. However, most of the system in real life is nonlinear, and it is not optimal to use KF as state estimator. Since then, the extension of the KF has been sought, such as Extended Kalman Filter (EKF)[41, 42]. EKF is the most commonly approach when dealing with nonlinear system. In EKF, all the nonlinear model is first linearized with first order Taylor series approximation, after that the linearized model is applied to traditional KF.

3.1.3.4.1 EKF Formulation

The implementation of EKF is same as KF, except that Jacobian matrices of the nonlinear model is needed to be calculated at each step to determine the linearized model. The nonlinear state equation and measurement equation can be written as follows.

$$x_{k+1} = f(x_k, u_k, w_k) \quad (15)$$

$$z_k = h(x_k, v_k) \quad (16)$$

Where f is the function of x_k , u_k and w_k , h is the function of x_k and v_k .

Hence the time update and measurement update equations can be written as follows.

Time update equations,

$$\hat{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0) \quad (17)$$

$$P_k = A_{k-1}P_{k-1}A_{k-1}^T + W_{k-1}QW_{k-1}^T \quad (18)$$

Measurement update equations,

$$K_k = P_k H_k^T (H_k P_k H_k^T + V_k R V_k^T)^{-1} \quad (19)$$

$$\hat{x}_{k+1} = \hat{x}_k + K_k(z_k - h(\hat{x}_k, 0)) \quad (20)$$

$$P_{k+1} = (I - K_k H_k) P_k \quad (21)$$

Where A is Jacobian matrix of partial derivatives of f with respect to x , W is Jacobian matrix of partial derivatives of f with respect to w , H is Jacobian matrix of partial derivatives of h with respect to x , V is Jacobian matrix of partial derivatives of h with respect to v .

$$A_{k-1} = \frac{\partial f}{\partial x}(\hat{x}_{k-1}, u_{k-1}, 0) \quad (22)$$

$$W_{k-1} = \frac{\partial f}{\partial w}(\hat{x}_{k-1}, u_{k-1}, 0) \quad (23)$$

$$H_k = \frac{\partial h}{\partial x}(\hat{x}_k, 0) \quad (24)$$

$$V_k = \frac{\partial h}{\partial v}(\hat{x}_k, 0) \quad (25)$$

Figure 15 shows the summarize diagram for EKF loop cycle algorithm.

Figure 15: Summarized EKF loop cycle algorithm.

3.1.3.5 Summary

Due to the drawback of KF on nonlinear system, EKF has been used in many applications which exhibit nonlinear behavior. In this project, EKF is used to fuse multiple sensors on the scanning system for elevator shaft inspection to obtain a better state approximation.

3.1.4 Hardware Devices and Setup

In this section, the hardware devices and setup that are going to be used for elevator shaft inspection will be discussed. The hardware devices that are chosen are ROS compatible to save development time and to eliminate the needs to write a new ROS interface driver program for the hardware.

3.1.4.1 Hardware Devices

Table 1 shows the list of the hardware devices and corresponding description used in the scanning system.

Hardware Device	Description
<p>Perceptin Ironsides (PIRVS) stereo camera</p>	<ul style="list-style-type: none"> • Used as stereo visual inertial odometry sensor for pose estimation. • Equipped with 2 global shutter 640 by 480 pixels RGB image sensor with 133° by 100° field of view (FOV) fisheye lens. It can capture images at 60 frame per seconds (fps). • Equipped with IMU 6 axis inertial measurement unit (IMU) sensor which able to output data at 200 Hz. • Comes with proprietary SLAM algorithm based on EKF image points feature based optical flow tightly coupled with IMU[43].
<p>MTi-7-DK IMU sensor</p>	<ul style="list-style-type: none"> • Used as IMU sensor for pose estimation. • Equipped with accelerometer, gyroscope, magnetometer, barometer and GNSS with maximum inertial data output rate of up to 800 Hz. • Built in proprietary sensor fusion algorithm XKF3 for orientation estimation.
<p>RPLIDAR A2M8 laser scanner</p>	<ul style="list-style-type: none"> • Used as laser scanner for 3D mapping. • Omnidirectional 2D laser scanner with maximum 8 m detecting range. • Measuring resolution of 0.5 mm when detecting range is less than 1.5 m, and 1 % for range beyond 1.5 m. • Nominal scanning rate of 10 Hz and maximum sampling rate of 8000 Hz. • Uses laser triangulation method to measure range. • Because the elevator shaft is rectilinear in height wise planar, the shape of the shaft is virtually similar in height wise planar regardless of the altitude, the laser scanner is also used for 2D planar pose estimation.
<p>TeraRanger Evo 60 m</p>	<ul style="list-style-type: none"> • Used as relative altitude sensor for pose estimation. • Measuring range of up to 60 m with resolution of 0.5 cm when the range is less than 14 m, and 1.5 % for range beyond 14 m. • Range data output rate of up to 240 Hz. • Uses 2° FOV LED based time-of-flight (TOF) technology to measure range.

Hardware Device	Description
<p>Intel NUC7i7BNH</p>	<ul style="list-style-type: none"> • Used as host computer. • Core i7 processor, 16 GB RAM, 256 GB M.2 SATA SSD storage. • Comes with 4 USB 3.0 ports and 1 Thunderbolt 3 port for expansion via dongle. • Linux Mint 18.3 Sylvia 64-bit is chosen as OS which is compatible with ROS Kinetic.
<p>Xiao Mi Wi-Fi Youth edition router</p>	<ul style="list-style-type: none"> • Used as communication node to control the host computer via remote desktop. • Comes with 2 LAN port and 802.11 a/b/g/n 2.4 GHz Wi-Fi.
<p>BX-DD90M-19V DC-DC power adapter</p>	<ul style="list-style-type: none"> • Used to power host computer. • Accept input voltage from 9 V to 17 V. • Output maximum power of 90 W at 19 V.
<p>5V DC-DC USB adapter</p>	<ul style="list-style-type: none"> • Used to power USB powered devices such as Ironsides stereo camera and Xiao Mi router. • Accept input voltage of 8 V to 24 V. • Output maximum power of 15 W at 5 V.

Table 1: List of hardware devices and corresponding description that is used in the scanning system. The more detailed specification of these hardware can be found in Appendix A to E.

3.1.4.2 Hardware Setup

Figure 16: Mounting frame.

To mount all the sensors and devices properly, a customized mounting frame is fabricated and assembled as shown in Figure 16. The supports of the mounting frames are made of 20 mm by 20 mm aluminum extruded frames. A few 3 mm thick acrylic sheets are laser cut and bolted onto the aluminum frames to mount the sensors and other hardware devices. 3D printed mounting supports and enclosure are also fabricated for all the sensors to mount onto the acrylic sheet. The drawing and dimension of these mounting supports can be found in Appendix G.

As shown in Figure 17, RPLIDAR A2M8 laser scanner is mounted vertically on the front of the mounting frame such that it can capture the 3D laser scan data around the system while it is moving along the rotation axis of the laser scanner. The TeraRanger Evo 60m range finder sensor is mounted on the front below the laser scanner such that it can measure the relative altitude from the ground. The MTi-7-DK IMU sensor is mounted upside down on the back of the mounting frame. The Ironsides stereo camera is mounted at 45° tilted behind the laser scanner. The stereo camera is tilted instead of facing forward such that it can obtain more robust features on the wall rather than rely on faraway features or objects for more accurate pose estimation[44]. The Intel NUC7i7BNH computer is mounted under the stereo camera with the provided VESA mounting plate, while the rest of the devices such as the DC-DC regulator, router and 3 cell Lithium Polymer (LiPo) battery are mounted on the back of the mounting frame.

Figure 17: Hardware setup on the mounting frame.

All the sensors are connected to the computer via USB interface. RPLIDAR A2M8 laser scanner, MTi-7-DK IMU sensor and TeraRanger Evo 60m range finder sensor do not consume high power, hence the power from the USB 3.0 ports on the computer are sufficient to power these sensors. Ironsides stereo camera and the router consumes higher power than the USB 3.0 power standard, hence they are powered via external 5V DC-DC regulator. The computer is powered by the BX-DD90M-19V DC-DC power adapter. Both DC-DC regulator and adapter are powered by a 3 cell LiPo battery. The computer of the scanning system is connected to the router via LAN cable, and a remote computer is used to control the computer wirelessly via the Wi-Fi through the router. Figure 18 shows the schematic of the hardware connection among the hardware sensors and devices.

Figure 18: Hardware connection schematics of the scanning system.

3.1.5 Software Setup

In this section, the ROS software setup for the scanning system is discussed. Most of the software driver for the sensors is available online on open source platform, and they are used in setting up the software. Slight modification of software driver is made to obtain the required output from the sensor to achieve the needs for the system.

3.1.5.1 Transformation

In order for the scanning systems to work properly, the placement of the sensors must be known to obtain accurate pose estimation. The relationship between the coordinate frames of robot's base and the sensors can be maintained by using the built-in transformation library called *tf2*[45]. *tf2* allows user to keep track of multiple frames over the time and allows user to transform between two coordinate frames at any time. In ROS, the coordinate frame complies with right hand

rule, with x as forward, y as left, and z as upward on a typical body. For camera devices, x points to the right, y points downward and z points forward of the camera optical lens.

The transformations between the base of the scanning system to the sensors frame are published using the *static_transform_publisher* node included in the *tf2* package. It publishes static coordinate transform between a parent frame to child frame using x/y/z offset from parent frame in meters and then quaternion for orientation about the child frame. The following matrices are the transformation matrices between the base of the scanning system and the sensors setup on the *static_transform_publisher*. The base of the scanning system is fixed on the RPLIDAR A2M8 laser scanner coordinate frame.

$$T_{base}^{laser} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (26)$$

$$T_{base}^{mti} = \begin{pmatrix} 1 & 0 & 0 & -0.30272 \\ 0 & -1 & 0 & 0.0051 \\ 0 & 0 & -1 & 0.07671 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (27)$$

$$T_{base}^{camera_link} = \begin{pmatrix} 0 & 0 & 1 & -0.15869 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0.04686 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (28)$$

$$T_{camera_link}^{camera_optical_link} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.7071068 & -0.7071068 & 0 \\ 0 & 0.7071068 & 0.7071068 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (29)$$

Where T_a^b denotes transformation from frame a to frame b , *base* is the frame attached to the base of the scanning system, *laser* is the frame attached on the RPLIDAR A2M8 laser scanner, *mti* is the frame attached on the MTi-7-DK IMU, *camera_link* is the camera axis frame attached on the Ironsides stereo camera but the orientation is aligned with the gravity (*i.e.* y axis of *camera_link*

pointing towards the earth gravity) and *camera_optical_link* is the frame attached on the Ironsides stereo camera.

Figure 19: Visualization of the placement of the coordinate frame of the sensors.

3.1.5.2 Ironsides Stereo Camera Odometry

The odometry data of Ironsides stereo camera can be obtained using the provided ROS library by the developer called *pirvs_ros*[46]. It contains *stereo_camera_node* for stereo image visualization, *depth_camera_node* for depth visualization and *slam_node* for SLAM data. By default, the *slam_node* reports the odometry of the stereo camera with corresponding covariance from odometry frame with ROS camera axis convention with y aligned with gravity (x right, y down, z forward), to the *camera_optical_link*. The default odometry data does not consider the offset from the *base* frame, which is required for KF pose estimation. The original *slam_node* is slightly modified to output the odometry data which takes the offset of *camera_optical_link* from *base* into account from odometry frame with ROS robot axis convention (x forward, y left, z up),

to the *base* frame. A detailed illustration regarding on the original and modified odometry reporting message can be seen in Figure 20. The odometry of the scanning system with stereo camera offset considered can be calculated using the following equation.

$$Odometry_{odom}^{base} = T_{base}^{camera_link} M (T_{base}^{camera_link} T_{camera_link}^{camera_optical_link})^{-1} \quad (30)$$

Where M is the original odometry measurement from *slam_node*.

Figure 20: Odometry reporting message. (a) Original odometry reporting message by the ROS node, odometry frame complies ROS camera axis convention and it is fixed to the earth, camera_optical_link complies ROS camera axis convention and it is fixed to the camera; (b) Odometry reporting message by the modified ROS node, odometry frame complies ROS robot axis convention and it is fixed to the earth, base frame complies robot axis convention, camera_optical_link complies ROS camera axis convention and it is fixed to the camera, and it is statically fixed to the base frame.

3.1.5.3 MTi-7-DK Inertial Measurement

The inertial measurement from MTi-7-DK IMU sensor can be obtained by using the provided ROS library called *xsens_driver*[47]. It contains *mtnode.py* node which can output inertial data from the IMU sensor. The output of the inertial data can be configured using *MT*

Manager software provided by the developer. For this application, the following inertial data is configured as output from the IMU sensor.

Output Inertial Data	Output Rate (Hz)
Orientation in quaternion	100
Acceleration (m/s ²)	100
Angular velocity (rad/s)	100

Table 2: MTi-7-DK IMU sensor inertial data output and output rate.

The orientation in quaternion, acceleration, and angular velocity are output in a grouped data with the same timestamp. The 800 Hz high rate acceleration and angular velocity output is not chosen because the high rate settings outputs both inertial raw data separately without time synchronization and the data is unprocessed. Thus, the high rate data output consists of noisier data. By default, the *mtnode.py* node outputs the orientation, acceleration and angular velocity together in a single ROS IMU message at 100 Hz, hence it will be used directly for KF pose estimation.

However, *mtnode.py* node does not output the covariance for the IMU inertial data, which is required in KF pose estimation. Hence, *mtnode.py* node is slightly modified to include the covariance matrix into the IMU message. To obtain the covariance of acceleration and angular velocity, the acceleration and angular velocity inertial data are recorded for 20 s while the IMU sensor is in stationary. The recorded inertial data sample is then used to compute the variance of each sensor axis using the following equation.

$$\sigma^2 = \frac{\sum (x - \bar{x})^2}{n} \quad (31)$$

Where σ^2 is variance, x is the sample data, $\bar{x}$ is the sample mean and n is number of sample data.

The variance calculated from the recorded sample data is as follows.

Inertial Data	Variance
X acceleration	$5.86 * 10^{-5}$
Y acceleration	$8.24 * 10^{-5}$
Z acceleration	$1.61 * 10^{-4}$
X angular velocity	$7.47 * 10^{-7}$
Y angular velocity	$9.47 * 10^{-7}$
Z angular velocity	$7.15 * 10^{-7}$

Table 3: Variance calculated from stationary MTi-7-DK inertial data.

For orientation, the variance is obtained by squaring the dynamic accuracy of the orientation based on the specification sheet. For acceleration and angular velocity, the variance of each axis of the sensors is chosen using the largest variance obtained from the corresponding sensors. Hence, the covariance matrices for acceleration, angular velocity and orientation are as follows.

$$\sigma_{acceleration}^2 = \begin{pmatrix} 2 * 10^{-4} & 0 & 0 \\ 0 & 2 * 10^{-4} & 0 \\ 0 & 0 & 2 * 10^{-4} \end{pmatrix} \quad (32)$$

$$\sigma_{angular\ velocity}^2 = \begin{pmatrix} 10 * 10^{-7} & 0 & 0 \\ 0 & 10 * 10^{-7} & 0 \\ 0 & 0 & 10 * 10^{-7} \end{pmatrix} \quad (33)$$

$$\sigma_{orientation}^2 = \begin{pmatrix} 6.85 * 10^{-4} & 0 & 0 \\ 0 & 6.85 * 10^{-4} & 0 \\ 0 & 0 & 6.85 * 10^{-4} \end{pmatrix} \quad (34)$$

Although the inertial output is factory calibrated, there are biases inherited in the acceleration and angular velocity output. Hence, manual calibration of IMU is performed to minimize the biases in the measurement value. The calibrated output of acceleration and angular velocity can be calculated as follows.

$$\begin{pmatrix} a_{x_c} \\ a_{y_c} \\ a_{z_c} \end{pmatrix} = \begin{pmatrix} s_{x_a} & 0 & 0 \\ 0 & s_{y_a} & 0 \\ 0 & 0 & s_{z_a} \end{pmatrix} \left(\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} - \begin{pmatrix} n_{x_a} \\ n_{y_a} \\ n_{z_a} \end{pmatrix} \right) \quad (35)$$

$$\begin{pmatrix} g_{x_c} \\ g_{y_c} \\ g_{z_c} \end{pmatrix} = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix} - \begin{pmatrix} n_{x_g} \\ n_{y_g} \\ n_{z_g} \end{pmatrix} \quad (36)$$

Where a_{i_c} and g_{i_c} are the calibrated output of i^{th} axis of accelerometer and gyroscope respectively, a_i and g_i are the raw sensor output of i^{th} axis of accelerometer and gyroscope respectively, s_{i_a} is scale factor of i^{th} axis of accelerometer, n_{i_a} and n_{i_g} are the neutral value of i^{th} axis of accelerometer and gyroscope respectively.

The scale factor and neutral value sets is required to obtain the calibrated output. To calculate for the scale factor and neutral value, 6 sets of IMU data is recorded and saved into a .csv file. Each set of the data is recorded at different sensor orientation for 10 s at stationary as such,

1. Sensor Z-axis pointing up (away from gravity)
2. Sensor Z-axis pointing down (towards gravity)
3. Sensor Y-axis pointing up
4. Sensor Y-axis pointing down
5. Sensor X-axis pointing up
6. Sensor X-axis pointing down

n_{x_g} , n_{y_g} and n_{z_g} can be obtained by just taking the average value of all gyroscope output of each axis from all the 6 data sets.

For accelerometer calibration parameters, when IMU is at stationary, equation (35) can be derived as follows.

$$T_{sensor}^{global} \begin{pmatrix} 0 \\ 0 \\ 9.80655 \end{pmatrix} = \begin{pmatrix} s_{x_a} & 0 & 0 \\ 0 & s_{y_a} & 0 \\ 0 & 0 & s_{z_a} \end{pmatrix} \left(\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} - \begin{pmatrix} n_{x_a} \\ n_{y_a} \\ n_{z_a} \end{pmatrix} \right) \quad (37)$$

Where T_{sensor}^{global} is the transformation from sensor to earth.

By taking the norm, the following equation is obtained.

$$9.80655^2 = (s_{x_a}(a_x - n_{x_a}))^2 + (s_{y_a}(a_y - n_{y_a}))^2 + (s_{z_a}(a_z - n_{z_a}))^2 \quad (38)$$

$$E(x) = 9.80655^2 - (s_{x_a}(a_x - n_{x_a}))^2 - (s_{y_a}(a_y - n_{y_a}))^2 - (s_{z_a}(a_z - n_{z_a}))^2 \quad (39)$$

Hence, to solve for scale factors and neutral values, non-linear least square method is used with the input from the recorded acceleration data and $E(x)$ as the error equation. The calculation is done using *optimize.leastsq* function from *scipy* library, which uses Levenberg-Marquardt method. The calibration parameters are saved into a *.yaml* file. A node is created to load the calibration *.yaml* file to perform calculation using equation (35) and (36), and output calibrated IMU inertial values. The calibration parameters and the result of calibration process are shown in Table 4 and the following figures respectively.

Calibration Parameters	Value
s_{x_a}	1.00431341
s_{y_a}	1.00201743
s_{z_a}	1.00248661
n_{x_a}	0.00378134
n_{y_a}	0.00496966
n_{z_a}	0.12045878
n_{x_g}	0.00183175
n_{y_g}	0.00140294
n_{z_g}	-0.00014891

Table 4: IMU calibration parameters.

Figure 21: Accelerometer output before and after calibration (Red: Z-axis, green: Y-axis, blue: X-axis).

Figure 22: Resultant acceleration before and after calibration.

Figure 23: X-axis gyroscope output before and after calibration.

Figure 24: Y-axis gyroscope output before and after calibration.

Figure 25: Z-axis gyroscope output before and after calibration.

3.1.5.4 RPLIDAR A2M8 Laser Scan Data and Planar Position

The laser scan data from RPLIDAR A2M8 can be obtained using the provided ROS library called *rplidar_ros*[48]. It contains *rplidarNode* node which outputs laser scan message from the sensor. By default, the scanning rate of the laser scanner is set at 10 Hz, and the sampling rate is set at 4000 Hz. The scanning rate and sampling rate can be adjusted as parameter value in *rplidarNode* node. For this application scanning rate is set as default of 10 Hz, while the sampling rate is set at 8000 Hz to obtain more scan point per rotation in order to get denser point cloud for better 3D mapping.

The 2D planar position of the scanning system is estimated using *laser_scan_matcher* library[49, 50]. *laser_scan_matcher* intakes scan data from 2D laser scanner to estimate the planar position of a robot incrementally. It implements high speed point-to-line metric Iterative Closes Point (ICP) algorithm between each consecutive laser scan frame to estimate the position. To speed up the scan prediction, it allows user to input IMU data into the computation. It will only take the

yaw element of the IMU sensor data for computation to speed up the rotation prediction. To achieve this, *imu_transformer* library is used to transform the IMU data from *mti* frame to *laser* frame such that the IMU data is reported in *laser* frame to feed into *laser_scan_matcher* for planar pose estimation[51]. *laser_scan_matcher* outputs position from odometry frame to *base* frame in form of *PoseWithCovarianceStamped* ROS message. However, a proper computed covariance is not included and have to set manually for KF pose estimation. The configuration of the covariance and the rest of the settings for *laser_scan_matcher* will be discussed in a later section.

3.1.5.5 TeraRanger Evo 60m Range Data

The range data from TeraRanger Evo 60m range finder can be obtained using the provided ROS library called *teraranger*[52]. It consists of several nodes for different sensor model, *evo*, *duo* and *one*. *evo* node is used to obtain the range data from Evo 60m sensor.

By default, the node output the range data in *Range* ROS message. The range message does not compensate for tilting of the scanning system. This may cause wrong output of relative altitude of the scanning system when it is tilted. To compensate for tilting, a new node is created to output tilt compensated altitude. The tilt compensated altitude is calculated using the following equation.

$$Altitude_{tilt\ compensated} = R \overrightarrow{R_{IMU}} \cdot \vec{G} \quad (40)$$

Where R is the measured range data from the sensor, $\overrightarrow{R_{IMU}}$ is the orientation vector of the axis where the range sensor is aligned with IMU. In this case, the range sensor is measuring the range in the x axis direction of the MTi-7-DK IMU sensor, hence $\overrightarrow{R_{IMU}}$ is equal to the first column of the orientation matrix measured by the IMU sensor. $\vec{G}$ is gravitational unit vector, which is equal to $0\hat{i} + 0\hat{j} - 1\hat{k}$.

Due to the KF library does not accept *Range* ROS message for pose estimation, a new node is created to convert the *Range* message to *PoseWithCovarianceStamped* message which is acceptable by the KF library. The altitude pose of the scanning system is reported from odometry frame to the *base* frame. The pose value is set to equal to the range value reported by the tilt compensated sensor value, and the variance of the pose which is required for KF pose estimation is based on the accuracy of the sensor obtained from the specification sheet and can be calculated as follows.

$$\sigma_{altitude}^2 = \begin{cases} 0.0016, & R < 14 \\ (0.015R)^2, & R \geq 14 \end{cases} \quad (41)$$

3.1.5.6 Pose Estimation

robot_localization library which is available in ROS repository is used for pose estimation of the scanning system in this project[53, 54]. *robot_localization* is a collection of non-linear state estimation nodes developed by Charles River Analytics. It consists of *ekf_localization_node* which implements EKF for non-linear state estimation of robot moving in 3D space. It also provides *navsat_transform_node* which aids in integration of GPS data for state estimation, however it is not used in this project.

This library supports arbitrary number of input sources for sensor fusion to feed into the state estimator. It supports multiple types of ROS message, such as *Odometry*, *Imu*, *PoseWithCovarianceStamped* and *TwistWithcovarianceStamped*. It also allows the user to configure the input of sensor to the state estimator. By using Boolean matrix, the users can configure whether to include or exclude multiple data reported by the sensor to be included into the state estimator. Process noise covariance is exposed to the user as parameter allows fine tuning of the performance state estimator. It has also a parameter setting for every sensor input called

differential which allows to convert absolute pose data into velocity for prediction. *robot_localization* keeps track of 15-dimensional state of the robot X . The 15 robot states and the transfer function for EKF are as follows.

$$X_k = AX_{k-1} + W_{k-1} \quad (42)$$

Where,

$$Cr = \cos(roll) \quad (43)$$

$$Cp = \cos(pitch) \quad (44)$$

$$Cy = \cos(yaw) \quad (45)$$

$$Sr = \sin(roll) \quad (46)$$

$$Sp = \sin(pitch) \quad (47)$$

$$Sy = \sin(yaw) \quad (48)$$

$$R = \begin{pmatrix} CpCy & SrSpCy - CrSy & CrSpCy + SrSy \\ CpSy & SrSpSy + CrCy & CrSpSy - SrCy \\ -Sp & SrCp & CrCp \end{pmatrix} \quad (49)$$

$$X_k = \begin{pmatrix} x \\ y \\ z \\ roll \\ pitch \\ yaw \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{roll} \\ \dot{pitch} \\ \dot{yaw} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix}_k \quad (50)$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & R_{11}t & R_{12}t & R_{13}t & 0 & 0 & 0 & \frac{1}{2}R_{11}t^2 & \frac{1}{2}R_{12}t^2 & \frac{1}{2}R_{13}t^2 \\ 0 & 1 & 0 & 0 & 0 & 0 & R_{21}t & R_{22}t & R_{23}t & 0 & 0 & 0 & \frac{1}{2}R_{21}t^2 & \frac{1}{2}R_{22}t^2 & \frac{1}{2}R_{23}t^2 \\ 0 & 0 & 1 & 0 & 0 & 0 & R_{31}t & R_{32}t & R_{33}t & 0 & 0 & 0 & \frac{1}{2}R_{31}t^2 & \frac{1}{2}R_{32}t^2 & \frac{1}{2}R_{33}t^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & R_{11}t & R_{12}t & R_{13}t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & R_{21}t & R_{22}t & R_{23}t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & R_{31}t & R_{32}t & R_{33}t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & t & t & t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & t & t & t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & t & t & t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (51)$$

Where R_{ij} is the element of matrix R at i^{th} row, j^{th} column, t is timestep.

robot_localization takes into account of the placement of the sensor by listening the transformation message published by *tf2*. Hence, the setup for transformation of the sensor from the *base* frame of the robot is important. *robot_localization* outputs odometry data and transformation message from *odom* frame to *base* frame. *odom* frame is a world fixed odometry frame where the pose of the robot or *base* frame is reported. The configuration of *robot_localization* for state estimation of the scanning system will be further discussed in the next section.

3.1.6 Experiment

In this section, the procedure of the experiment for scanning system localization is discussed. In general, the localization of the scanning system is done offline. All the sensors data are first recorded via *rosvbag* into *.bag* files, then the *.bag* file is used in post processing for offline localization. This method can eliminate the needs for in-situ run to save time, and the *.bag* file can be used repeatedly for different post processing.

3.1.6.1 Experiment Testbed

Due to inaccessible to an elevator shaft at the time of experiment, a long rectilinear office hallway corridor is used as a testbed for the localization experiment. The corridor is almost rectilinearly rectangular in lengthwise planar with minimum dynamic changes in features that cause significant change in planar shape. The corridor is about 1.8 m widthwise, 8 m lengthwise and 2.7 m height wise. These features make the corridor suitable to be simulated as an elevator shaft but in horizontal orientation. Figure 26 shows the hallway corridor that is used as the testbed for this experiment. A pair of tripods holding a long rod to hang a piece of reflective cloth is placed at the end of the corridor to simulate the ground base of the elevator shaft such that TeraRanger Evo 60m ranger finder can measure the “altitude” of the scanning system relative to the “ground”.

Figure 26: Office hallway corridor to simulate elevator shaft as testbed.

To deliver the scanning system down the corridor, P3DX mobile robot is used as the mobile platform. The P3DX mobile robot comes with SICK LMS100 270° indoor LIDAR laser scanner,

which allows P3DX mobile robot to perform accurate indoor navigation, localization and mapping. The scanning system is mounted on top of SICK LIDAR on P3DX using 3M Dual-Lock fastener as shown in Figure 27. Such mounting position gives us the transformation matrix for $tf2$ setup from the SICK LIDAR to the *base* frame of the scanning system as follows.

$$T_{laser_frame}^{base} = \begin{pmatrix} 1 & 0 & 0 & 0.0813 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.15558 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (52)$$

Where *laser_frame* is the frame attached to the SICK LMS100 LIDAR.

To use P3DX mobile robot, a host computer with USB connection is required to act as a server for the robot. As such, the computer on the scanning system is used as a host computer for P3DX. An ROS library is developed by P3DX developer called *rosarnl*, which allows the user to use P3DX in ROS environment[55]. *rosarnl* contains a node called *rosarnl_node* which acts as server node for P3DX. *rosarnl_node* uses the input from SICK LIDAR and wheel odometry to localized P3DX against a known map using Monte Carlo particle filter based localization method. The node publishes static transform from *base_link* frame which is the base of P3DX mobile robot to *laser_frame*. It also publishes transform from *map* frame which is attached to the known map to *base_link* frame in accordance to the location of the mobile robot in the known map. Hence, the location transform publishes by *rosarnl_node* is used as ground truth for the scanning system as reference. As for the client to control the P3DX mobile robot, a proprietary software called MobileEyes which is developed by the same developer is used. MobileEyes allows the user to configure and control P3DX mobile robot remotely via Wi-Fi.

Figure 27: Scanning system mounted on P3DX mobile robot.

3.1.6.2 Experiment Procedure

The corridor is first pre-mapped using P3DX mobile robot provided LIDAR mapping software to create a known 2D map for *rosarnl_node* prior to the experiment. This is done by running *arnlServer* server program on the host computer connected to P3DX and using *MobileEyes* client remotely to drive the mobile robot along the corridor. This will create a *.2d* file which can be edited and processed using the provided software called *Mapper3* to generate *.map* file which can be loaded into the *rosarnl_node* as a known map. Two goal points *Start* and *End* is added to the map at the starting and ending point of the corridor respectively such that the mobile robot can navigate and drive autonomously between these two points without manual driving. Figure 28 shows the 2D map of the corridor generated by *Mapper3* as seen on *MobileEyes* client.

Figure 28: 2D map generated from Mapper3, as seen on MobileEyes. The red circle represents the location of the mobile robot on the known 2D map.

To start the scanning of the corridor, *rosarnl_node* is first run using *roslaunch* command to turn on the server for P3DX mobile robot. *MobileEyes* on a remote computer is then connected to the *rosarnl_node* server. The mobile robot with the scanning system mounted on is first driven to the *Start* point using *MobileEyes*. The maximum translation velocity of the mobile robot is set to 200 mm/s via *MobileEyes* client.

A launch file is created to allow running multiple nodes using single command. The launch file consists of *rplidarNode* node to run RPLIDAR A2M8 laser scanner, *slam_node_2* node which is the modified version of the original *slam_node* node to run PIRVS Ironsides stereo camera, *mtnode.py* node to run MTi-7-DK IMU sensor, *evo* node to run TeraRanger Evo 60m range finder and *tf2_static_publisher.py* node to publish static *tf2* transforms for all the frames. In

$tf2_static_publisher.py$, $T_{laser_frame}^{base}$, $T_{base}^{camera_link}$, $T_{camera_link}^{camera_optical_link}$, T_{base}^{mti} , T_{base}^{laser} and T_{map}^{odom} are published. T_{map}^{odom} is the transform from *map* frame to *odom* frame, where *odom* is the odometry frame where the position of the scanning system frame is reported to. T_{map}^{odom} is equal to initial T_{map}^{base} when the mobile robot is on the *Start* point. The *odom* frame will be used as reference frame to report the position of the scanning system *base* frame for localization analysis. Figure 29 and Figure 30 show the ROS computation graph and frame transforms tree generated from *rqt_graph* and *rqt_tf_tree* respectively. After the launch file is executed using *roslaunch* command, recording of all the topics is started using *rosbag record* command. Subsequently, P3DX mobile robot is commanded to move to *End* point via *MobileEyes* client autonomously as shown in Figure 31 and Figure 32. The *rosbag* recording is stopped upon arrival to the *End* point, and a *.bag* files is generated that contains all the topics recorded during the scanning phase.

Figure 29: ROS computational graph as seen on *rqt_graph*. Oval represents nodes, rectangular box represents namespace, arrow represents topic messages flow direction.

Figure 30: Transforms tree of all sensors and robot frames as shown in `rqt_tf_tree`.

Figure 31: Mobile robot moving to the End point autonomously as shown in MobileEyes client GUI software. The blue line represents the calculated path on the map by MobileEyes for the mobile robot to move from Start to End.

Figure 32: Mobile robots moving to the End point in real world.

After *.bag* is recorded, a launch file is created to run all the necessary nodes for post processing to obtain the trajectories of the system. *use_sim_time* ROS parameter is set to true such that the process uses the clock in the recorded bag file instead of the current time. *imu_calibrated* node reads the IMU calibration parameter file and publishes the calibrated IMU message. *imu_transformer_node* node transforms the calibrated IMU message from IMU sensor frame *mti* to RPLIDAR laser scanner frame *laser* using *imu_transformer* library[51]. The purpose of this transformation is to facilitate *laser_scan_matcher* to read the IMU message because it only accepts the yaw or z-axis angular velocity data in IMU message. *laser_scan_matcher_node* node reads the laser scan message from laser scanner and IMU message from transformed calibrated IMU message and publishes position of the system in *PoseWithCovarianceStamped* message. Three parameters in this node are considered as variable to tune the performance of the pose estimation, i.e. *outliers_maxPerc*, *position_covariance* and *orientation_covariance*. *outliers_maxPerc* is the percentage of correspondences to be considered, i.e. 0.9 means always discard the top 10% of correspondence with more error. *position_covariance* and *orientation_covariance* are the values of the diagonal covariance matrices of position and orientation. *tilt_compensated_altitude* node reads calibrated IMU message and range message from TeraRanger altitude sensor and publishes tilt compensated altitude. In this case, since the experiment is performed on simulated horizontal hallway instead of vertical elevator shaft, the gravitational vector $\vec{G}$ is set to be parallel with the scanning direction, $1\hat{i} + 0\hat{j} + 0\hat{k}$. *altitude_pose* node converts the tilt compensated altitude range message to position message, *PoseWithCovarianceStamped* such that the altitude data can be accepted by *robot_localization* library. *ekf_localization_node* node is the EKF computation node from *robot_localization* library for pose estimation. It reads all the sensor data and publishes position of the system in *Odometry* message. There are a few key parameters that can be adjusted

in the localization node. *[sensor]_config* is the Boolean matrix which allows user to specify which variables from the sensor message should be included for estimation. *[sensor]_differential* allows sensor pose variables to be integrated differentially, means convert sensor pose into velocity. *[sensor]_relative* allows the sensor data to be fused relative to the first measurement. *process_noise_covariance* allows the user to set the process noise for fine tuning. *initial_estimate_covariance* allows the user to set the initial estimated covariance for initialization. *tf2_publisher* node reads the *Odometry* message from stereo camera and *robot_localization* and publishes the transform of the system from *odom* frame to the corresponding base frame. *path_generator* node publishes the trajectory of all corresponding base frame for visualization in *rviz*. Finally, *extract_tf* node extracts the trajectory of each corresponding base frame (ground truth position, visual inertial position and EKF position) as *.txt* file for later analysis. Figure 33 shows the ROS computational graph for post processing.

Figure 33: ROS computation graph as shown in `rqt_graph` GUI for post processing nodes. Oval is the node, inner box is topic and outer box is namespace.

3.2 3D Map Generation

3.2.1 Introduction

This section describes the generation of 3D map for the hallway which simulated the elevator shaft horizontally. With proper localization, the scanning system can build a 3D map out of the scanning planar area using the vertical mounted laser scanner. The scanning system builds up the vertical 2D omnidirectional laser scan layer by layer while the scanning system is moving forward and finally form a 3D map. The 3D map is generated in the form of point clouds. The point clouds can then be post-processed into 3D model, BIM model, etc. The point clouds can also be used for inspection such as point clouds metrology, surface quality, crack inspection, etc. Point clouds add new layer of 3D information into 2D image for visual inspection which improve the efficiency, accuracy, quality and confidence of the inspection.

In this section, the methodology of generating the 3D map of the hallway is discussed. The generated 3D map is compared with the real world dimensions to compute the accuracy of the 3D map generated.

3.2.2 Smoothing of Scanning System Path Trajectory

The trajectory output of the scanning system by EKF localization with baseline configuration from previous section's result is chosen to be used for 3D map generation. However, the trajectory output by EKF localization contains spikes which may cause degradation in quality of the 3D map generated. To minimize the spikes, the trajectory is filtered via low pass filter offline to smoothen the trajectory.

A first order Butterworth low pass digital filter with cutoff frequency of 1 Hz normalized at 30 Hz is used to filter out the spikes in the trajectory. The filter is then applied via forward-

backward filtering or zero phase filtering to smoothen the trajectory without introducing time lag. This can be done via *butter* and *filtfilt* functions from *scipy.signal* library. To perform the filtering, first the odometry or trajectory output by the EKF localization node of the whole scanning process is extracted. After that, the trajectory is filtered by the low pass filter. The filtered trajectory is then converted to ROS transformation message from *odom* frame to *base* frame, *tfMessage* with the associated timestamp. The transformation message is then saved into a new *.bag* file using *rosbag* API. The laser scan message by the RPLIDAR A2M8 laser scanner from the original *.bag* file is also extracted and saved together with the transformation message into the new *.bag* file. Figure 34 to Figure 36 show the raw and the filtered trajectory of the EKF localization output. One can see that the filtered trajectory minimized the sudden jumps or spikes while retaining the low frequency locomotion and rotation.

Figure 34: Raw and filtered absolute trajectory output by EKF localization computation.

Figure 35: Raw and filtered absolute translation output by EKF localization computation.

Figure 36: Raw and filtered absolute orientation output by EKF localization computation.

3.2.3 3D Map Point Clouds Generation

In this section, the methodology for generation of the 3D map point clouds of the hallway is discussed. The generated map is compared in the following section with the real world dimension to evaluate the accuracy of the map generated from EKF trajectory.

3.2.3.1 Software Setup

To generate 3D map of the hallway, the laser scan data from the laser scanner must first be converted from *LaserScan* ROS message into *PointCloud2* ROS message. After the conversion, the point clouds of each successive planar scans are accumulated and eventually formed 3D point clouds. This can be done via *laser_scan_assembler* node in *laser_assembler* library in ROS[56]. *laser_scan_assembler* subscribes to *LaserScan* message. These scans are projected into 3D space to form point clouds and then they are transformed into the fixed frame. This results in the point clouds to be saved into the rolling buffer. The point clouds are then assembled with a service call. *odom* frame is set as the fixed frame in *laser_scan_assembler* node. The parameter *ignore_laser_skew* is set to false such that the assembler will transform each individual point in each scan to the fixed frame to minimize the skewing effect on the laser scan due to translational and rotational movement. A node called *cloud_assemble.py* is created to periodically service call to the *laser_scan_assembler* node to assemble and publish the point clouds at the interval of 10 s. To save the assembled point clouds such that it can be used outside of ROS environment, *pointcloud_to_pcd* node in *pcl_ros* library is used[57]. *pointcloud_to_pcd* subscribes to a *PointCloud2* message and convert it into *.pcd* file, which can be used in external software such as *CloudCompare*. The *.pcd* files are saved separately every time the service is called to assemble the point clouds. Hence, *pcl_concatenate_points_pcd* application which is included in *pcl* source code is used to combine all the separated files into one single point clouds file. A launch file is created to run these 2 nodes simultaneously. Two static transform publishers are also included in the launch file to publish T_{map}^{odom} and T_{base}^{laser} because they are not included in the newly created *.bag* files which contains only the T_{odom}^{base} *tfMessage* and *LaserScan* messages. To build the 3D map, the

launch file is first executed following by the new *.bag* file. Figure 37 shows the ROS computation graph for 3D map generation.

Figure 37: ROS computation graph for 3D map generation as seen on *rqt_graph*.

3.3 Image Capture and Color Point Cloud Generation

3.3.1 Introduction

This section describes the panoramic 2D image capture of the hallway which simulated the elevator shaft horizontally and hallway 3D point cloud colorization. With 3D point cloud generated in the previous section's result, it can be used for inspection such as point cloud metrology, surface quality, crack inspection, etc. With the 360° 2D image capture of the elevator shaft that will be discussed in this section, the scanning system can capture the images of the elevator shaft and it can cover the whole shaft wall height wise. The images captured by the scanning system can be used for visual inspection to inspect the surface quality, surface crack, and every other anomaly that can be seen and inspect visually, which improves the efficiency, accuracy, quality and confidence of the inspection on top of the 3D point cloud. To make the 3D point cloud more realistic and more representative to the real world, color point cloud is required. This can be done by fusing the 2D captured image together with laser scanned 3D point cloud.

3.3.2 Panoramic 2D Image Capture

In this section, the methodology and the result for panoramic 2D image capture of the hallway are discussed.

3.3.2.1 Hardware Devices and Setup

To capture panoramic image around the scanning system at different height in elevator shaft, wide angle or fisheye lens camera is required. The proposed method to achieve this is by installing 4 wide angle or fisheye lens cameras on 4 sides of the scanning system, 90° apart from each other.

The fisheye lens camera chosen for this application is Arducam MT9N001 with LS-81600 lens and Arducam USB shield as shown on Figure 38. Arducam MT9N001 comes with 9 MP RGB rolling shutter CMOS Aptina 1/2.3" image sensor. Matched together with 2.8 mm focal length LS-81600 fisheye lens, it gives the Arducam MT9N001 camera horizontal FOV of 120°, vertical FOV of 98° and diagonal FOV of 155°. The Arducam USB shield allows the camera to interface with the computer on the scanning system via USB cable. A more detailed specification of this camera set can be found on Appendix F.

Figure 38: Arducam MT9N001 with LS-81600 lens and Arducam USB shield.

For testing the image capturing of the scanning system in the simulated hallway, only 3 cameras are used as one side of the scanning system is occupied for mounting to the mobile robot. To mount the cameras on to the scanning system, 3 laser cut 3 mm thick acrylic sheet are fabricated as adapter to mount the camera onto the scanning system as shown in Figure 39. The drawing and dimension of the acrylic sheet camera adapter can be found in Appendix G. To provide space for the camera, the MTi-7-DK IMU sensor is moved to other location with transformation as follows.

$$T_{base}^{mti} = \begin{pmatrix} 1 & 0 & 0 & -0.27172 \\ 0 & -1 & 0 & 0.0051 \\ 0 & 0 & -1 & 0.05371 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (53)$$

All the 3 cameras are connected to the computer using USB cable via the Thunderbolt 3 port with USB C to 4 x USB 3.0 hub. The cameras are powered directly by the computer via the USB cable as the Thunderbolt 3 has high power delivery capability. An addition 3 cell LiPo battery is added and connected in parallel with the previous battery to increase the total battery capacity for longer run time. Figure 40 shows the updated hardware connection schematic.

Figure 39: Hardware setup on the mounting frame with additional 3 cameras.

Figure 40: Hardware connection schematics of the scanning systems with additional 3 cameras.

3.3.2.2 Software Setup

The ROS software setup for the updated scanning system is the same as in previous section, but with additional ROS library for the Arducam MT9N001 camera and changes in transformation T_{base}^{mti} .

The image captured by Arducam MT9N001 via its USB shield can be obtained using the provided ROS library *arducam_usb2_ros*[58]. It consists of *arducam_ros_node.py* node which outputs the image captured by the camera continuously and *arducam_trigger_ros_node.py* node which outputs the image captured by camera via external or software trigger. Due to limited buffer memory on the USB shield (8 MB) which is insufficient to store 9 MP raw image data, the *arducam_trigger_ros_node.py* node does not work with MT9N001 camera. Instead, *arducam_ros_node.py* node provides capture service, which will capture the latest image available from the continuously captured image to another image topic when the service is called. Hence, *arducam_ros_node.py* node is used to capture the images of the hallway.

3.3.3 Experiment

The experiment testbed and procedure are the same as described in previous section. The same mobile robot P3DX is used to deliver the scanning system down the hallway. Three additional nodes called *arducam_ros_node.py* are added to the launch file to run three of the Arducam MT9N001 USB cameras. Instead of scanning the hallway continuously, a new node is also written and added to the launch file such that the scanning system will call the Arducam node capture service to capture images every 1 m the scanning system moved based on the reading from TeraRanger Evo 60m range finder. When the range finder reading indicates the scanning system has moved for 1 m, it triggers the mobile robot to stop moving and the scanning system to capture images. Once the images at that point are captured, the mobile robot continues to move forward.

The process is repeated until the mobile robot reaches the *End* point. Figure 41 shows the summarized flow chart of the image capturing process for the new node. This node also publishes topic with type *UInt64* message which indicates the time in nanosecond when the images are captured. The purpose of the image capture time message is to correlate the position of the cameras in the 3D point cloud map at that time stamp from the EKF filtered trajectory.

Figure 41: Flowchart for the image capturing process of the scanning system.

Same as the procedure described in previous section, all the data is first recorded into a *.bag* files before post processing. The trajectory of the scanning system is obtained via EKF sensor fusion and low pass filtered as described in the previous section with default EKF and sensor settings. A python script is created to read the *.bag* files to extract the images data and image captured time. The extracted images are then saved to corresponding folder based on the images captured time.

3.3.4 2D RGB Images and 3D Point Cloud Fusion

In this section, the methodology and result for fusing 2D RGB images and 3D point cloud to generate 3D RGB point cloud are discussed.

3.3.4.1 Camera Calibration

Before projecting RGB information from images to point cloud, the transformation or relative placement between the cameras and the laser scanner must be known. To get this transformation, the intrinsic parameter of the cameras must first be known. This can be done by camera calibration. There are 2 camera models in camera calibration, pinhole camera model and lens distortion model. Figure 42 shows the pinhole camera model coordinate system[59] and equation (54) to (56) show the pinhole camera model in camera calibration.

Figure 42: Pinhole camera model coordinate system.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = (R \mid t) \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} \quad (54)$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{Z_c} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \quad (55)$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (56)$$

In equation (54), $(R | t)$ is a 3 x 4 extrinsic matrix or extrinsic parameter, $\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}$, which maps the points in world coordinate system (X_w, Y_w, Z_w) to camera coordinate system (X_c, Y_c, Z_c) . The points in camera coordinate system is then normalized to get the coordinate in the image plane coordinate system (x, y) which is done by equation (55). Finally,

in equation (56), K is the intrinsic matrix or intrinsic parameter of the camera, $\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$,

which maps the coordinate in image plane coordinate system (x, y) to pixel coordinate system (u, v) . f_x and f_y are the focal length in x and y axis respectively, c_x and c_y are principal point or optical center of the camera in x and y axis respectively. According to *Z. Zhang* proposed camera calibration process, by observing feature points on a same planar such as checkerboard, the intrinsic and extrinsic parameters of the camera can be solved using closed-form solution[60].

Because the lens come with the camera is not totally perfect, especially the cheap ones or lenses such as fisheye lens, the lens distorts the image significantly. Hence, the distortion model is required to accurately map the world coordinate to pixel coordinate. Equation (57) to (60) show the equations of applying generic lens distortion model which can be used by different type of lenses to map the distorted points onto image plane coordinate system[61].

$$r^2 = x^2 + y^2 \quad (57)$$

$$\theta = \tan^{-1}(r) \quad (58)$$

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \quad (59)$$

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} \theta_d \left(\frac{x}{r} \right) \\ \theta_d \left(\frac{y}{r} \right) \end{pmatrix} \quad (60)$$

Where (x_d, y_d) is the coordinate of the distorted point in image plane coordinate system. Hence, by applying (x_d, y_d) to equation (56), the true pixel coordinate of the point can be obtained.

To solve for distortion parameter k_n where $n = 1, 2, 3, 4$, and optimize K, R and t , non-linear least square method or Levenberg-Marquardt Algorithm is used by minimizing equation (61) with initial guess value of K, R and t from the closed-form solution method and 0 for all k_n [60].

$$\sum_{i=0}^n \sum_{j=0}^m \|m_{ij} - \tilde{m}(K, k_1, k_2, k_3, k_4, R_i, t_i, M_j)\|^2 \quad (61)$$

Where m_{ij} is the observed point M_j in image i , $\tilde{m}(K, k_1, k_2, k_3, k_4, R_i, t_i, M_j)$ is the projection of point M_j in image i according to equation (56), followed by distortion according to equation (60).

Rather than calibrating the camera manually, there are open source software that includes camera calibrating tools such as OpenCV[62]. OpenCV is a cross platform programming function for real time computer vision. It supports wide range of computing language from C++, Python, Java, etc. In this work, OpenCV Python is used.

OpenCV comes with a set of camera calibration tools which follows *Z. Zhang* method[60]. The first step of camera calibration is to capture several position of internal checkerboard corners in different viewing angle. This can be done by using the OpenCV function *cv2.findChessboardCorners()* as shown in Figure 43. After capturing the positions of the corners,

`cv2.cornerSubPix()` function is used to optimize the position of the captured corners. With the set of optimized position of checkerboard corners, `cv2.fisheye.calibrate()` is called to calibrate the camera to obtain the intrinsic parameter K and distortion parameter k_n where $n = 1, 2, 3, 4$. Equation (62) to (64) show the results of the camera calibration for 3 of the cameras and Figure 44 shows the undistorted image after the camera calibration.

$$cam0 \rightarrow \begin{cases} K = \begin{pmatrix} 1586.644446 & 0.000000 & 1805.993570 \\ 0.000000 & 1600.437149 & 1231.266613 \\ 0.000000 & 0.000000 & 1.000000 \end{pmatrix} \\ D = (0.027211 \quad -0.010915 \quad 0.020443 \quad -0.008353) \end{cases} \quad (62)$$

$$cam1 \rightarrow \begin{cases} K = \begin{pmatrix} 1621.107826 & 0.000000 & 1797.662514 \\ 0.000000 & 1615.638652 & 1207.679913 \\ 0.000000 & 0.000000 & 1.000000 \end{pmatrix} \\ D = (0.017950 \quad -0.001574 \quad 0.014469 \quad -0.005424) \end{cases} \quad (63)$$

$$cam3 \rightarrow \begin{cases} K = \begin{pmatrix} 1603.307835 & 0.000000 & 1886.895365 \\ 0.000000 & 1597.988146 & 1235.335817 \\ 0.000000 & 0.000000 & 1.000000 \end{pmatrix} \\ D = (-0.018166 \quad 0.075004 \quad -0.054713 \quad 0.015506) \end{cases} \quad (64)$$

Figure 43: Checkerboard corners detected by OpenCV `cv2.findChessboardCorners()` function from one of the cameras.

Figure 44: (Left) Distorted image before calibration from one of the cameras; (Right) Undistorted image after calibration from the same camera.

3.3.4.2 Camera LIDAR Extrinsic Calibration

With the intrinsic parameter of the cameras obtained, the extrinsic parameter or transformation between the laser scanner and the camera can be calculated. The extrinsic calibration can be done by placing the world coordinate system on the laser scanner, such that all the laser points are in the same plane or $Z_w = 0$. From equation (54) to (56), equation (65) can be derived.

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K(R \mid t) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad (65)$$

Where s is an arbitrary constant, $\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$ is the image pixel point, K is the camera intrinsic parameter,

$(R \mid t)$ is the extrinsic parameter which in this case is the transform from camera to laser scanner,

$\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$ is the world coordinate point which in this case is the coordinate of the point where the

laser scanner projected.

With known $\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$, K and $\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$, but unknown scale factor s , $(R | t)$ can be solved using

Direct Linear Transform (DLT) method[63]. OpenCV provides a function called *cv2.solvePnP()* which able to solve such problem to obtain the pose of the camera, given image points, object points, camera intrinsic parameter and camera distortion parameter as inputs. *cv2.solvePnP()* uses DLT method to obtain $(R | t)$, then it uses Levenberg-Marquardt Algorithm to minimize the reprojection error as shown in equation (61) to get an optimized $(R | t)$.

To obtain the object points $\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$, *2D Nav Goal* tool from *rviz* is used. The tool publishes the

position and orientation of the clicked point in *rviz* main map window to topic */move_base_simple/goal*. Hence, the object points can be extracted from the topic. To easily distinguish the laser point in real world, a unique feature such as corner of a box is used as the object points as shown in Figure 45.

To obtain image points $\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$, *matplotlib* is used. The pixel coordinate of the image can be

obtained from the image shown in *matplotlib* plot window by using mouse click event callback feature provided by *matplotlib*. To look for the position of the laser point projected by the laser scanner onto the box, a camera with IR filter removed is used to tag the position. Figure 46 shows

the laser points projected by the laser scanner onto the box, and Figure 47 shows the pixel point selected where the laser point falls onto the corner of the box.

A camera LIDAR extrinsic calibration tool is created to perform such calibration process to obtain the pose of the cameras relative to the laser scanner[64]. Equation (66) to (68) show the extrinsic calibration results, and Figure 48 shows the reprojection of the laser scanner points onto the image after the calibration.

$$T_{cam0}^{laser} = \begin{pmatrix} -0.99909087 & 0.0303948 & 0.02989288 & -0.025845 \\ 0.02967971 & -0.00744044 & 0.99953177 & 0.320383 \\ 0.03060298 & 0.99951028 & 0.00653156 & -0.151331 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (66)$$

$$T_{cam1}^{laser} = \begin{pmatrix} 0.02970302 & -0.99949626 & 0.0111783 & -0.031134 \\ 0.00389266 & 0.01129882 & 0.99992859 & 0.335143 \\ -0.99955119 & -0.02965738 & 0.0042263 & -0.126293 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (67)$$

$$T_{cam3}^{laser} = \begin{pmatrix} 0.00753912 & 0.99949215 & 0.03096134 & -0.016822 \\ -0.00650834 & -0.03091252 & 0.9995009 & 0.338533 \\ 0.9999504 & -0.00773687 & 0.00627198 & -0.14029 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (68)$$

Figure 45: Corner of the box projected by the laser scanner.

Figure 46: Laser from laser scanner projected onto the box as seen by IR filter removed camera. A blue tac is used to tag the location of the projected laser at the corner of the box as indicator.

Laser coordinate: (-0.926044, -0.105670)
Alpha: 3.100000 Beta: 0

Figure 47: Image point selected as indicated by the red dot.

Figure 48: Projected laser points (green dots) onto the image after calibration.

3.3.4.3 RGB Projection to 3D Point Cloud

With the relative pose between cameras and laser scanner, the global coordinate of the cameras in the 3D map can be obtained using equation (69). Hence, with the image captured positions obtained in section 5.2, the global coordinate of each of the cameras can be obtained.

$$T_{map}^{camera} = T_{map}^{base} * T_{base}^{laser} * (T_{camera}^{laser})^{-1} \quad (69)$$

With the cameras and point cloud position in global coordinate, the position of the point cloud in the image pixel coordinate system can be calculated by using equation (54) to (60). OpenCV provides a function called `cv2.fisheye.projectPoints()` which able to perform such task. The RGB information of the pixel can then be assigned to the corresponding point cloud. The process is

iterated for all the images captured by all the 3 cameras. A python script is created to perform such task. Python wrapper version of *pcl* is used as input/output module for the *.pcd* point cloud file to/from the python script[65].

However, there are one problem that *cv2.fisheye.projectPoints()* does not take into the account, the hidden points which is not visible by the camera as illustrated in Figure 49. Hence, when assigning RGB information to the point cloud, the hidden point cloud which is not visible by the camera is also colorized. Hence, it may lead to inaccuracy in RGB 3D point cloud. To fix this, Hidden Point Removal (HPR) algorithm developed by *S. Katz et al.* is used[66, 67]. The HPR algorithm is applied before *cv2.fisheye.projectPoints()* function to remove the hidden point cloud which cannot be seen by the camera. Figure 50 shows the summarized flow chart for the point cloud colorization process.

Figure 49: Visible point cloud and hidden point cloud from a camera perspective.

Figure 50: Flowchart for 3D point cloud colorization.

4 Results and Discussions

4.1 Localization of Scanning System in Elevator Shaft

4.1.1 Results and Post Analysis

To evaluate the performance of localization, absolute trajectory error (ATE) and relative pose error (RPE) metrics are used[68]. ATE directly measures the difference between points in true and estimated trajectory of timestamp. It first associates the nearest timestamp between the points of true and estimated trajectory, then calculates the difference between each pair of associated points to compute the root mean squared error (rmse) and standard deviation (std) of the trajectory error. RPE on the other hand measures the local accuracy of the trajectory over a fixed time interval. It computes the error in relative motion between true and estimated trajectory of timestamp which can be seen in equation (70), and then computes the rmse and std of the error. ATE is well suited to evaluate the overall performance of the localization system while RPE is well suited for measuring the drift of the localization system. The python scripts to compute ATE and RPE is readily available online[69].

$$E_i = (Q_i^{-1}Q_{i+\Delta})^{-1}(P_i^{-1}P_{i+\Delta}) \quad (70)$$

Where Q_i is estimated trajectory point at timestamp i , P_i is true trajectory point at timestamp i , Δ is time interval.

Table 5 shows the baseline configuration for the parameters required for EKF computation. The y , z and $roll$ from laser scan planar odometry, x from altitude, $pitch$ and yaw from IMU are fused for correction update. Translational and rotational velocity of all axis from visual inertial odometry, translation acceleration and $roll$ rate from IMU are fused for prediction update. The process noise covariance matrix is set at the

highest of default value from each five different states, 0.06 for translation and orientation, 0.04 for translational velocity, 0.02 for angular velocity and 0.015 for acceleration. The initial estimated covariance is set at 0.1 in diagonal term of the matrix with the rest is 0. The translational and rotational covariance for the laser scanner planar odometry is set at 0.0025. The *outliers_maxPerc* is set at 0.5 to remove 50 % laser scan data to prevent sudden jump in planar position due to minor feature change in lengthwise planar of the hallway, such as corner of door frame, etc.

Configuration for <i>robot_localization</i>					
Sensor input	Sensor parameter to fuse		Relative	Differential	
Visual inertial odometry	$x, y, z, \alpha, \beta, \gamma$		False	True	
IMU inertial	$\ddot{x}, \ddot{y}, \ddot{z}, \dot{\alpha}, \beta, \gamma$		True	False	
Laser scanner planar odometry	y, z, α		True	False	
Altitude	x		True	False	
* α, β, γ represent roll, pitch, yaw respectively.					
Process noise covariance	$Q_{translation}$	$Q_{rotation}$	$Q_{velocity}$	$Q_{angular\ velocity}$	$Q_{acceleration}$
	0.06	0.06	0.04	0.02	0.015
Initial estimated covariance diagonal term	0.1				

Configuration for <i>laser_scan_matcher</i>	
Translation sensor noise covariance	0.0025
Rotational sensor noise covariance	0.0025
<i>outliers_maxPerc</i>	0.5

Table 5: Baseline configuration of robot_localization and laser_scan_matcher parameters for EKF estimation.

With the baseline configuration, the trajectory, ATE and RPE are as following table and figures.

Pose Estimation method	rmse	std
ATE (m)		
PIRVS	0.686417	0.347267
EKF	0.084247	0.046517
Translation RPE (m)		
PIRVS	0.036108	0.023349
EKF	0.024732	0.015899
Rotation RPE (deg)		
PIRVS	0.624369	0.377678
EKF	0.398503	0.222

Table 6: ATE and RPE output for baseline configuration.

Figure 51: ATE plot for baseline configuration.

Figure 52: RPE plot for baseline configuration.

Figure 53: Absolute trajectory plot for baseline configuration.

Figure 54: Absolute translation plot for baseline configuration.

Figure 55: Absolute orientation plot for baseline configuration.

From Table 6 and Figure 51 to Figure 55, it can be seen that with pure visual inertial odometry PIRVS, the scanning system drift significantly away from the ground truth with maximum drift error of about 1 m especially on y and z axis, while it falls short in x axis. PIRVS also suffers a major drift in orientation especially in roll axis by about 6° . This resulted in PIRVS

has about 8 times higher ATE metric than EKF method. PIRVS is also perform worse than EKF in both translation and rotation RPE metrics. Due to the ground truth localization uses global positioning against a known map to localize in x-y plane, the odometry for other localization method do not compensate for the drift due to global correction, hence there is some drift as can be seen in the other trajectories away from ground truth. To compensate for the drift, closed form singular value decomposition (SVD) Horn's method is used to realign the trajectory to the ground truth in x-y plane. The SVD Horn's method computation is provided in the script for ATE computation. After realignment, from Table 7 and Figure 56 to Figure 59, one can see that translation and orientation for EKF method improves significantly especially in x-y plane and yaw orientation. However, for PIRVS, although the ATE improves from before realignment, the trajectory is drifted because the starting point is 50 cm and 3.3° off the ground truth's starting point. Besides, PIRVS also drifts in z axis. Hence, one can concludes that PIRVS is drift inherently. This is most likely due to lack of features in the tested hallway, hence the lack of feature key points for optical flow causes significant drift and inaccuracy in visual inertial odometry. The lighting conditions of the hallway can also affect the performance of the visual inertial odometry of PIRVS. In general, EKF is still performing significantly better than PIRVS in ATE metric.

Pose Estimation method	rmse	std
ATE (m)		
PIRVS	0.319134	0.14733
EKF	0.070075	0.03132

Table 7: ATE output for realigned baseline configuration.

Figure 56: ATE plot for realigned baseline configuration.

Figure 57: Absolute trajectory plot for realigned baseline configuration.

Figure 58: Absolute translation plot for realigned baseline configuration.

Figure 59: Absolute orientation plot for realigned baseline configuration.

To fine tune the model, several parameters are adjusted for comparison with the baseline configuration. The parameters are process noise covariance Q , translation and rotation sensor noise

covariance for *laser_scan_matcher* R (y-z plane odometry) and *outliers_maxPerc* P. All the trajectories are first realigned with the ground truth before comparison.

Firstly, the baseline trajectory with process noise covariance Q is compared with trajectories with process noise covariance matrix multiply by 100, 100Q and process noise covariance matrix multiply by 0.01, 0.01Q. The following table and figures show ATE and RPE of the trajectory with different Q.

Pose Estimation method	rmse	std
ATE (m)		
EKF, Q	0.070075	0.03132
EKF, 0.01Q	0.054529	0.018519
EKF, 100Q	0.073761	0.033101
Translation RPE (m)		
EKF, Q	0.024732	0.015899
EKF, 0.01Q	0.034516	0.023243
EKF, 100Q	0.025964	0.017726
Rotation RPE (deg)		
EKF, Q	0.398503	0.222
EKF, 0.01Q	0.530285	0.307653
EKF, 100Q	0.391447	0.221303

Table 8: ATE and RPE output for trajectories from EKF with Q, 100Q and 0.01Q.

Figure 60: ATE plot for trajectories from EKF with Q , $0.01Q$ and $100Q$.

Figure 61: RPE plot for trajectories from EKF with Q , $0.01Q$ and $100Q$.

Figure 62: Absolute trajectory plot for trajectories from EKF with Q , $0.01Q$ and $100Q$.

Figure 63: Absolute translation plot for trajectories from EKF with Q , $0.01Q$ and $100Q$.

Figure 64: Absolute orientation plot for trajectories from EKF with Q , $0.01Q$ and $100Q$.

From Table 8 and Figure 60 to Figure 64, one can see that at $0.01Q$, the ATE metric shows that it performs better than baseline Q and $100Q$. However, there is significant offset in the starting point, hence the ATE metric of $0.01Q$ may not conclude its performance. Besides, the trajectory of $0.01Q$ is more sensitive to spikes which resulted in worse translation and rotation RPE metrics than Q and $100Q$. The trajectory of $100Q$ performs slightly better in rotation RPE metric than Q , but slightly worse in ATE and translation RPE metric. However, the difference in all the metrics between Q and $100Q$ are negligible. Hence one can say that both Q and $100Q$ perform about the same. Nonetheless, the trajectory of $100Q$ exhibits more noise than Q . Hence, one can conclude that setting process noise covariance at Q yields a better performance.

Furthermore, trajectories with different sensor noise R in translation and orientation for y - z plane positioning from *laser_scan_matcher* are compared. The comparison is made between trajectory with baseline R , trajectory with baseline R multiply by 100, $100R$ and trajectory with baseline R multiply by 0.01, $0.01R$, and the results are shown as follows.

Pose Estimation method	rmse	std
ATE (m)		
EKF, R	0.070075	0.03132
EKF, 0.01R	0.070526	0.031132
EKF, 100R	0.072247	0.03041
Translation RPE (m)		
EKF, R	0.024732	0.015899
EKF, 0.01R	0.024104	0.016566
EKF, 100R	0.030497	0.019573
Rotation RPE (deg)		
EKF, R	0.398503	0.222
EKF, 0.01R	0.396977	0.218083
EKF, 100R	0.429849	0.245547

Table 9: ATE and RPE output for trajectories from EKF with R, 100R and 0.01R.

Figure 65: ATE plot for trajectories from EKF with R, 100R and 0.01R.

Figure 66: RPE plot for trajectories from EKF with R , $100R$ and $0.01R$.

Figure 67: Absolute trajectory plot for trajectories from EKF with R , $100R$ and $0.01R$.

Figure 68: Absolute translation plot for trajectories from EKF with R, 100R and 0.01R.

Figure 69: Absolute orientation plot for trajectories from EKF with R, 100R and 0.01R.

From Table 9 and Figure 65 to Figure 69, the trajectory at 0.01R performs slightly better in translation RPE metric than baseline R, but slightly worse in ATE and rotation RPE metrics. However, the differences in all the metrics between R and 0.01R are negligible. Hence it can be

said that R and $0.01R$ perform about the same. At $100R$, all the metrics perform worse than R . There is also significant offset from ground truth starting point. Increasing R implies that the system model to trust more on other sensors such as PIRVS, hence the model is more sensitive to PIRVS velocity which drifts significantly, causing degrades in overall performance. Decreasing R implies that the system model to trust more on *laser_scan_matcher*, hence it is less sensitive to other sensors. If there are significant feature changes in y-z plane, it may cause the trajectory to be unstable. At a nutshell, it is suggested to increase R if other sensors can provide accurate and less drift data, decrease R if the feature changes in y-z plane is insignificant.

Lastly, the trajectories with different *outliers_maxPerc* parameter, P in *laser_scan_matcher* is also compared. The comparison is made between baseline P of 0.5, P of 0.9 and P of 0.1 and the results are as follows.

Pose Estimation method	rmse	std
ATE (m)		
EKF, $P = 0.5$	0.070075	0.03132
EKF, $P = 0.1$	0.07053	0.031134
EKF, $P = 0.9$	0.079488	0.024367
Translation RPE (m)		
EKF, $P = 0.5$	0.024732	0.015899
EKF, $P = 0.1$	0.024856	0.015678
EKF, $P = 0.9$	0.034423	0.024192
Rotation RPE (deg)		
EKF, $P = 0.5$	0.398503	0.222
EKF, $P = 0.1$	0.37201	0.207601
EKF, $P = 0.9$	0.556367	0.404416

Table 10: ATE and RPE output for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$.

Figure 70: ATE plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$.

Figure 71: RPE plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$.

Figure 72: Absolute trajectory plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$.

Figure 73: Absolute translation plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$.

Figure 74: Absolute orientation plot for trajectories from EKF with $P = 0.5$, $P = 0.1$ and $P = 0.9$.

From Table 10 and Figure 70 to Figure 74, at higher P value of 0.9, the *laser_scan_matcher* considers the best 90% of the laser scan data for pose estimation in laser scan plane or y-z plane. Hence, it is much more sensitive to dynamic changes in feature in y-z plane. This can be seen between the time frame 10 s to 20 s where the laser scanner pass by a door frame which is indented into the parallel side wall. This causes sudden jump in y and roll axes, hence it has significant higher ATE and RPE than baseline P value of 0.5. At lower P value of 0.1, *laser_scan_matcher* only considers the best 10% of the laser scan data for pose estimation in y-z plane. This may cause the pose estimation in y-z plane to under report when there is slight lateral motion. However, in this case, $P = 0.5$ and $P = 0.1$ performs almost the same as shown by ATE and both RPE metrics. Hence, $P = 0.5$ is a good compromise between sensitivity to feature change in y-z plane and lateral movement in this case.

4.1.2 Conclusion

In this task, the methodology for pose estimation and localization of the scanning system for elevator inspection has been discussed and experiment is done on a 8 m long horizontal hallway to simulate vertical elevator shaft. ROS compatible IMU, altitude LED range finder, 2D omnidirectional laser scanner, and visual inertial odometry sensor are used for pose estimation. A mounting frame made from aluminum extrusion frames and acrylic sheet is fabricated and assembled to mount all the sensors and necessary electrical and electronic devices. Multiple sensor fusion method, EKF is used to fuse all the sensors together for pose estimation. From the experiment, it is found that EKF performs better and obtains more accurate pose estimation than single odometry sensor such as the visual inertial odometry sensor. Several parameters such as process noise covariance Q , sensor noise covariance for y-z plane pose estimation R and *outliers_maxPerc* P in *laser_scan_matcher* are also tuned to investigate their effect on pose estimation. It is found out that the model with baseline parameter values performs the best in this case.

4.2 3D Map Generation

4.2.1 Result and Post Analysis

In this section, *CloudCompare* is used for 3D map visualization and post analysis. *CloudCompare* is an open source software for 3D point cloud post processing. It provides basic tools for editing and rendering point clouds. It also provides various advanced processing algorithm such as ICP registration, cloud to cloud distance, etc.

Figure 75 shows the generated 3D point clouds. One can see that there are floating point clouds outliers due to sunlight shining on the laser scanner. The outliers are manually removed via

Segment tool in *CloudCompare*. The final cleaned 3D point clouds compare with real world image are as shown in Figure 76. From the cleaned 3D point clouds map, we can see that there is shadowing effect in certain places especially in sharp corner like the corner of the brown door frame on the right. This is due to the laser scanning plane is parallel to the corner, hence the surface at such places is not registered into the point clouds. This issue can be minimized or eliminated as a future work improvement by adding an expensive 3D laser scanner or by adding 2 low cost 2D laser scanners crisscross slanted at an angle to scan the environment as illustrated in Figure 77. Adding additional laser scanners can also increase the point clouds density to create a more robust 3D model representation.

Figure 75: Raw 3D point clouds map of the hallway.

Figure 76: Cleaned 3D point clouds map of the hallway compare with the real world image.

Figure 77: (a) Scanning system with one laser scanner unable to scan corner resulted in shadowing effect in final point clouds; (b) Crisscross slanted multiple laser scanners can scan corner to minimize or eliminate shadowing effect.

To evaluate the accuracy of the 3D point clouds map, several chosen features in the 3D map are measured using *Point Picking* tool in *CloudCompare* and compared against the measurement in real world. Figure 78 and Table 11 shows the chosen features to be compared and the result of the measurement. The overall average absolute error of the chosen features is 0.00545 m with standard deviation of 0.0254 m. One can see that the largest error occurs in the width of the hallway with absolute error of 0.07 m and relative error of 3.98 %. This may be due to alignment measuring error when measuring the distance in both point clouds and real world due to lack of reference feature points for measurement such as corner to corner of window, etc.

Figure 78: Feature to be measured for comparison between 3D point clouds map and real world to evaluate point clouds accuracy.

Feature	Description	Point Clouds measurement (m)	Real measurement (m)	Absolute Error (m)	Relative Error (%)
a	Back hallway width	1.8	1.775	0.025	1.408451
b	Back window width	2.66	2.65	0.01	0.377358
c	Front window width	2.6	2.6	0	0
d	Front hallway width	1.83	1.76	0.07	3.977273
e	Indented wall width	2.61	2.62	-0.01	-0.38168
f	Indented wall height	2.18	2.185	-0.005	-0.22883
g	Door height	2.1	2.13	-0.03	-1.40845
h	Door width	1.59	1.57	0.02	1.273885
i	Door window height	0.3	0.3	0	0
j	Door window width	0.15	0.15	0	0
k	Wall width between windows	1.58	1.6	-0.02	-1.25
Average				0.005454545	
Standard Deviation				0.025446427	

Table 11: Comparison of chosen features between point clouds measurement and real measurement.

Another way to evaluate the accuracy of the 3D point clouds map is to measure the flatness of the floor and the wall. It is assumed that the floor and the wall is perfectly flat for this analysis. To measure the flatness of the floor and wall, first the point clouds of the two surfaces are segmented out from other non-interested surfaces using *Segment* tool. After that, *Fit* function is used to create the nearest fit flat plane as reference to the point clouds of the two surfaces respectively. *Cloud/Mesh Dist* tool is used to calculate the distance between the point clouds of the surface to the best fitted flat plane. Figure 79 to Figure 82 shows the colourmap of the flatness and the flatness error distribution graph of the two surfaces. From the calculation, the flatness error of the floor has a mean of 0 m, standard deviation of 0.004804 m and maximum absolute error of 0.024484 m, while the flatness error of the wall has a mean of 0 m, standard deviation of 0.006744

m and maximum absolute error of 0.028167 m. One can see that there is sudden significant up and down slopes near the middle of the point clouds of the floor. This is due to the sudden up and down in the z-axis pose estimation. Besides, the spread of the flatness error is wider in the point clouds of the wall. This is due to the pose estimation in y-axis is more unstable than z-axis. As conclusion, the quality and accuracy of the 3D point clouds map heavily depends on the pose estimation of the scanning system.

Figure 79: Colormap of the flatness of the point clouds of the floor. The color bar on the right shows the color corresponding to the flatness error.

Figure 80: Colormap of the flatness of the point clouds of the wall. The color bar on the right shows the color corresponding to the flatness error.

Figure 81: Floor flatness error distribution graph, the black curve represents normal distribution curve.

Figure 82: Wall flatness error distribution graph, the black curve represents normal distribution curve.

4.2.2 Conclusion

In this task, the methodology for 3D point clouds map generation has been discussed and mapping is done on the same long hallway as previous task to simulate vertical elevator shaft. The trajectory from EKF pose estimation with baseline configuration is used as the position of the scanning system for 3D mapping. The trajectory is first passed through Butterworth low pass zero phase filter for smoothening. After that, the 3D point clouds map is built up using the laser scan data from laser scanner corresponding to the position of the scanning system. The 3D point clouds map is compared with the several features dimension in real world and the flatness of the floor and wall are measured to evaluate the accuracy of the point clouds. It is found out that the 3D map has an average error of 0.00545 m with 0.02544 m standard deviation and maximum absolute error of 0.07 m in real world dimension comparison. In flatness evaluation, the 3D point clouds map of the floor achieves mean error of 0 m with standard deviation of 0.004804 m and maximum absolute error of 0.024484 m, while the wall achieves mean error of 0 m with standard deviation of 0.006744 m and maximum absolute error of 0.028167 m.

4.3 Image Capture and Color Point Cloud Generation

4.3.1 Panoramic 2D Image Capture

4.3.1.1 Result and Visualization

Figure 83 shows two sets of subsequent images of the hallway captured by the scanning system. The entire panoramic images of the hallway can be found in Appendix H. One can see that the 3 camera placements on the scanning system are able to fully capture the hallway horizontally as shown by the overlap region in the same sequence set of images. By capturing images at every 1 m distance travelled sequentially, the scanning system is able to fully capture the hallway vertically as shown by the overlap region in the subsequent set of images. As such, by using the scanning system with 4 cameras on every sides to scan the elevator shaft, the images captured by the scanning system can fully cover travelled height wise and 360° sidewise of the elevator shaft. Hence, the inspector can inspect every corner of the elevator shaft by using the images captured by the scanning system.

Figure 83: Overlap regions between two sets of subsequent hallway images captured by the scanning system.

To better visualize where the position of the set of images is captured, *rviz* is used. First, the generated *.pcd* point cloud file of the hallway is converted to ROS *PointCloud2* message using *pcd_to_pointcloud* node in *pcl_ros* library. This allows the generated hallway point cloud to be visualized from *rviz*. Next, a new node is created to insert interactive markers on the 3D point cloud map to show the position of the images is captured, and by clicking the marker, the set of images taken at that position will be shown. This can be done using *interactive_markers* library built into the ROS. The position of the image captured spots are obtained by correlating the timestamp where the images are captured by the scanning system with the EKF filtered trajectory. Figure 84 and Figure 85 show the 3D point cloud of the hallway in *rviz* together with the interactive markers where the images are captured, and the images captured on a new window when an interactive marker is clicked respectively. The images are plotted and shown using *matplotlib* python library, where it allows user to use the zoom and pan tool.

Figure 84: 3D point cloud of the hall way shown in *rviz*. The semitransparent, clickable round interactive markers show the position where images are captured.

Figure 85: Images captured shown in a new window when the interactive marker at that position is clicked. The title of the window shows the 3D coordinate of the position where the images are captured in the 3D point cloud map.

With this set of tools and system, the inspector can do the scanning of the elevator shaft by using a crane pulley system to deliver the scanning system to scan through the elevator shaft to obtain 2D color and 3D geometry information. This method can reduce the down time of the elevator significantly compare to traditional inspection method. Besides, this method does not require the inspector to work inside the elevator shaft, hence the workplace hazards is reduced or eliminated. After the scanning is performed, the raw data is post processed to generate the 3D point cloud and 2D images as described previously. The inspector then can inspect the verticality, geometry and condition of the elevator shaft offsite. This method improves the yield and efficiency of the inspection compare to traditional method. Besides, the generated data can be kept as inspection record. Furthermore, with images of the defect in the elevator shaft, these image datasets can be used to train AI engine such that it can be used to inspect the elevator shaft quickly and

automatically to further improve the efficiency. As such, the lack of manpower problem in this area is solved by replacing them with the AI engine.

4.3.2 2D RGB Images and 3D Point Cloud Fusion

4.3.2.1 Result and Discussion

Similar method described in section 5.2 is used to visualize the RGB 3D point cloud. Figure 86 to Figure 91 shows the RGB 3D point cloud generated after colorization. The RGB data from images in general accurately projects into the 3D point cloud. However, the result exhibits several artifacts.

Firstly, there are wavy patterns can be seen on the RGB 3D point cloud as indicated in Figure 90. This is due to the reduction of image brightness towards the corner of the image. This effect is called vignetting and it is common in all optical camera lenses. Vignetting effect can be reduced via image post processing vignetting correction. Without vignetting correction, the overlap between the darker corner of the image and the brighter central of the image creates the wavy pattern as shown.

Secondly, there is misalignment in RGB information in the RGB 3D point cloud as indicated in Figure 90 and Figure 91. There are several reasons that might causes this. One of the reasons is bad camera calibration parameter. Bad camera calibration parameter causes the world point to be inaccurately mapped to the image pixel point. This affects the extrinsic calibration parameter between laser scanner and camera which eventually causes misalignment between RGB information and 3D point cloud. Besides, a bad quality lens also affects the quality of the camera calibration parameter as the calibration in OpenCV assumes the distortion in lens to be radially homogeneous and it does not take tangential distortion into account. Hence, the RGB information

misalignment issue can be improved by using lens with better quality and by performing a better camera calibration procedure such as cover as much field of view of the camera as possible when capturing the checkerboard pattern. Furthermore, due to the accuracy of the 3D point cloud is purely based on the accuracy of the trajectory of the scanning system, the 3D point cloud might not accurately represent the geometry in the real world as compare to the images captured by the camera. Hence, the RGB information is inaccurately mapped to the 3D point cloud. To improve the accuracy of the 3D point cloud, a 3D laser scanner or multiple 2D laser scanner is needed to perform scanning. The point cloud generated by 3D laser scanner or multiple 2D laser scanner overlaps in each successive scan. Thus, Iterative Closest Point (ICP) algorithm can be used to update the 3D point cloud map and also the odometry of the scanning system. With ICP algorithm, the accuracy of the 3D point cloud as well as the trajectory of the scanning system can be significantly improved, which eventually reduced the RGB misalignment issue.

Figure 86: RGB 3D point cloud of the hallway as seen on rviz.

Figure 87: RGB 3D point cloud of the hallway as seen on rviz in different perspective 1.

Figure 88: RGB 3D point cloud of the hallway as seen on rviz in different perspective 2.

Figure 89: RGB 3D point cloud of the hallway as seen on rviz in different perspective 3.

Figure 90: RGB 3D point cloud of the hallway as seen on rviz in different perspective 4.

Figure 91: RGB 3D point cloud of the hallway as seen on rviz in different perspective 5.

4.3.3 Conclusion

In this task, the methodology for panoramic image capture and RGB 3D point cloud generation has been discussed and the testing is done on the same hallway which simulated the elevator shaft. Three fisheye lens cameras at 90° apart are used to capture the panorama image. The scanning system triggers the cameras to capture image at every 1 m distance moved interval. As a result, the panorama images cover every part of the hallway that is visible by the cameras, which makes it suitable for visual inspection if used in elevator shaft. Camera intrinsic and extrinsic calibration is performed as a necessary step for RGB projection from image to 3D point cloud. The RGB projection to point cloud is generally accurate, however there are several artifacts such as wavy pattern and RGB misalignment. There are several approaches that can reduce these artifacts such as vignetting correction, use high quality lens for the camera, perform better calibration process and ICP mapping and localization.

5 Conclusion and Future Works

5.1 Conclusion

The purpose of this thesis is to design and develop an inspection method that can improve the yield and efficiency of elevator shaft and verticality inspection over the traditional manual inspection which accompanies many sorts of problems such as safety hazard, lack of skilled manpower, time consuming, etc. The proposed method described in this thesis is to use a scanning system that consists of laser scanner and cameras which can be delivered along the elevator shaft via pulley system, to scan the elevator shaft to obtain the 3D point cloud and 2D visual images. The 3D point cloud can be used to measure the verticality of elevator shaft, while the 2D images can be used to perform visual inspection to look for anomaly offsite. A prototype is built and tested in a long hallway to simulate vertical elevator shaft.

There are 4 tasks that are carried out to accomplish the final output, localization of the scanning system, 3D map scanning and generation, 2D panoramic image capture and 3D point cloud – 2D RGB image stitching. ROS is used to perform all the tasks on the scanning system. During the scanning, all the sensors data are recorded into ROS *.bag* file which then is used for post processing to perform all the tasks mentioned.

1. Localization of the scanning system

EKF is used to fuse several sensors in the scanning system to obtain a more accurate pose estimation. In ROS, *robot_localization* library is used to perform EKF computation. The sensors used in the scanning system are inertial measurement unit for inertia measurement, stereo camera for visual inertial odometry, laser scanner for laser scanner

planar odometry and laser range finder for altitude measurement. Baseline values as shown in Table 5 are used for EKF configuration. Experiment is done with various combination of configuration Q, R and P, and it is found out that the baseline configuration yields the best result, where ATE = 0.070075 m with standard deviation of 0.03132 m, translation RPE = 0.024732 m with standard deviation of 0.015899 m and rotation RPE = 0.398503° with standard deviation of 0.222°.

2. 3D map scanning and generation

The timestamped EKF filtered trajectory from localization task and timestamped laser scan data are used to generate 3D map. The trajectory is first passed through a first order Butterworth low pass filter to smoothen the trajectory. The laser scan data is then assembled into 3D point cloud based on the smoothen EKF filtered trajectory. This is done using ROS *laser_scan_assembler* library which takes in laser scan data and odometry or position data to generate 3D point cloud. The 3D map generated can be seen in Figure 76. There are 2 methods used to evaluate the accuracy of the 3D map, feature dimension comparison such as size of door, etc., and flatness measurement of the floor and wall. In feature dimension comparison, it is found out that the average absolute error achieved is 0.005454545 m with standard deviation of 0.025446427 m. In flatness measurement, the maximum absolute error achieved is 0.006744 m with standard deviation of 0.028167 m.

3. 2D panoramic image capture

USB cameras are installed on each plane of the scanning system for 2D panoramic image capture. However, in this thesis, only 3 cameras are used as one of the planes is attached to mobile robot for testing. The flowchart of image capturing procedure can be

seen in Figure 41. As a result, the scanning system can fully capture every corner of the simulated environment as seen in Figure 83 where all the image captured by each camera overlaps on one to another. For full image dataset, one can refer to Appendix H.

4. 3D point cloud – 2D RGB image stitching

2D RGB information captured by the cameras are projected into the generated 3D point cloud to obtain 3D color point cloud for better representation. To do this, camera intrinsic and camera laser scanner extrinsic calibration are carried out first to obtain intrinsic and extrinsic parameters. The parameters are needed for projecting RGB information from camera image to point cloud. The flowchart for the point cloud colorization can be seen in Figure 50. All the process is done using Python *OpenCV* library. The result of the final colorization process can be seen from Figure 86 to Figure 91. From the results obtained, the RGB projection to point cloud is generally accurate, but there are some artifacts generated such as wavy pattern and RGB misalignment due to some limitations of current prototype.

With these tasks, RGB 3D point cloud and 2D panoramic images of the surrounding are produced. In a real operation, with the generated output by the scanning system, the yield and efficiency of the inspection process can be improved, because the inspector does not have to manually inspect onsite which compromises their safety as well as time consuming. AI based image processing algorithm can also be applied to the output to speed up the process and improve the accuracy. Furthermore, the output can be kept as inspection record for a routine inspection.

5.2 Future Works for Improvements

Although the proposed concept is technically feasible, there are improvements that can be done in future research to improve the current output. In the current setup, the accuracy of the 3D map

generation is purely based on accuracy of EKF pose estimation, which may degrade the quality of the 3D point cloud if it's unstable. In the experiment performed in this thesis, the maximum absolute error obtained in flatness of the 3D point cloud is 0.028167 m, which may cause false report in verticality inspection especially in high rise building. This can be improved by using 3D ICP algorithm to update and correct the 3D map as well as to improve the localization based on the global features. However, the current concept setup uses only one 2D laser scanner, which is impossible to apply 3D ICP algorithm. Thus, a 3D laser scanner or multiple 2D laser scanner must be used such that there is overlap in each successive scan for the 3D ICP algorithm to work. Furthermore, there are artifacts such as wavy pattern and RGB misalignment can be found on the current RGB point cloud output. The wavy pattern can be solved by applying vignetting correction post processing. As for the RGB misalignment, there are several possible sources that causes this issue. First is the inaccuracy in the system's trajectory and 3D point cloud map. This can be solved by using ICP algorithm as described previously. Second source is the bad camera intrinsic and extrinsic calibration parameters. This can be improved by using a higher quality camera lens and a better, well planned calibration procedure. Other than these improvements, currently the output data is visualized from *rviz*, which does not have the best graphical user interface (GUI). In the future, it is recommended to create a software that has improved GUI over *rviz* for the user to visualize and interact with the output data such as web app.

REFERENCES

- [1] "Population Trends, 2017," Singapore Department of Statistics 2017.
- [2] (2018). *Lift Safety*. Available: <https://www.bca.gov.sg/LiftSafety/lift.html>
- [3] (2017). *HDB lift suspended after falling four floors, injuring passenger*. Available: <http://www.todayonline.com/singapore/hdb-lift-suspended-after-falling-four-floors-injuring-passenger>
- [4] (2018). *Elderly couple trapped in lift for more than 2 hours in Yishun*. Available: <https://www.channelnewsasia.com/news/singapore/elderly-couple-trapped-in-lift-for-more-than-2-hours-in-yishun-9824360>
- [5] T. I. Raymond, Juanda. (2016). *Woman suffers spine fracture in 5th lift accident in 8 months*. Available: <https://www.todayonline.com/singapore/woman-suffers-spine-fracture-5th-lift-accident-8-months>
- [6] G. Timothy. (2017). *Jurong West lift that plummeted four storeys was serviced just two weeks before accident*. Available: <http://www.straitstimes.com/singapore/housing/jurong-west-lift-that-plummeted-four-storeys-was-serviced-just-two-weeks-before>
- [7] G. Timothy. (2017). *Jurong West HDB lift suspended for further investigation after accident injures one: BCA*. Available: <http://www.straitstimes.com/singapore/housing/woman-taken-to-hospital-after-jurong-lift-drops>
- [8] A. Yusof. (2017). *Woman injured after HDB lift fell from 4th floor in Jurong West*. Available: <https://www.channelnewsasia.com/news/singapore/woman-injured-after-hdb-lift-fell-from-4th-floor-in-jurong-west-8997250>
- [9] J. Hong. (2017). *BCA warning after spate of lift breakdowns*. Available: <http://www.straitstimes.com/singapore/bca-warning-after-spate-of-lift-breakdowns>
- [10] *Hoist Mechanism*. Available: <http://home-lift-singapore.com/hoist-mechanism/>
- [11] G. R. Strakosch and R. S. Caporale, *The Vertical Transportation Handbook*. John Wiley & Sons, 2010, p. 20.
- [12] G. R. Strakosch and R. S. Caporale, *The Vertical Transportation Handbook*. John Wiley & Sons, 2010, p. 56.
- [13] *Elevator Structure and Equipment*. Available: http://www.mitsubishielectric.com/elevator/overview/elevators/str_eqp.html
- [14] N. A. Siti, A. S. Asmone, and M. Y. L. Chew, "An assessment of maintainability of elevator system to improve facilities management knowledge-base," *IOP Conference Series: Earth and Environmental Science*, vol. 117, no. 1, p. 012025, 2018.
- [15] EMSD, "Report on Lift Incident on 9 November 2008 at Lift No. 4, Wan Lam House, Wan Tau Tong Estate, Tai Po," 2008.
- [16] *Special Elevator Applications - Mine Elevators*. Available: <http://www.elevatorbobs-elevator-pics.com/mine.html>
- [17] A. Waterproofing. (2015). *Why Elevator Waterproofing Is Crucial For Every Building*. Available: <https://aquaguardwaterproofing.com/why-elevator-waterproofing-is-crucial-for-every-building/>
- [18] KJA. (2018). *Rising Waters - Impact on Elevator and Escalators*. Available: <https://www.kja.com/rising-waters-impact-on-elevator-and-escalators-p154285>

- [19] Langeproon. *Jõe 4A elevator shaft*. Available: <http://www.langeproon.ee/en/solution/injection/joe-4a-elevator-shaft/>
- [20] M. McCann, "Deaths and Injuries Involving Elevators and Escalators " 2013.
- [21] "Comparison of worldwide lift safety standards," in "ISO Standard 11071," International Organization for Standardization 2004.
- [22] Y. Yang, W. Guo, Y. Wang, and W. Lu, *Automated Detecting System for Elevator Guide Rails Based on Tilt Sensors and Acceleration Sensors*. 2018.
- [23] L. Kovanič, "Determination of Elevator Shaft Uprightness Applying the Terrestrial Laser Scanning Method/Určovanie Zvislosti Výťahovej Šachty Metódou Terestrického Laserového Skenovania," *GeoScience Engineering*, vol. 59, no. 3, pp. 6-20, 2013.
- [24] S.-T. Park and B.-S. Yang, "An implementation of risk-based inspection for elevator maintenance," *Journal of Mechanical Science and Technology*, journal article vol. 24, no. 12, pp. 2367-2376, December 01 2010.
- [25] E. Dawei and Z. Xiaobin, "The design and implementation of remote elevator monitoring system based on GPRS and embedded technology," in *2011 International Conference on Computer Science and Service System (CSSS)*, 2011, pp. 2695-2698.
- [26] Z. Xiangqun, "Mechanics Performance of Elevator Monitoring System Based on GPRS and ZIGBEE Network," *Applied Mechanics and Materials*, vol. 214, pp. 543-7, / 2012.
- [27] F. Qizhi, G. Yongzhe, and X. Hongyu, "The design of elevator monitoring terminal Based on the GPRS," *Applied Mechanics and Materials*, vol. 313-314, pp. 1180-3, / 2013.
- [28] T. Shuilin, W. Zhengyang, and Q. Bin, "Research of the elevator monitoring system based on the internet of things," *Applied Mechanics and Materials*, vol. 423-426, pp. 2380-5, / 2013.
- [29] I. Skog, I. Karagiannis, A. B. Bergsten, J. Härdén, L. Gustafsson, and P. Händel, "A Smart Sensor Node for the Internet-of-Elevators—Non-Invasive Condition and Fault Monitoring," *IEEE Sensors Journal*, vol. 17, no. 16, pp. 5198-5208, 2017.
- [30] O. Yaman, M. Baygin, and M. Karakose, "A New Approach Based on Image Processing for Detection of Wear of Guide-Rail Surface in Elevator Systems," *International Journal of Applied Mathematics, Electronics and Computers*, vol. 4, no. Special Issue-1, pp. 296-300, 2016.
- [31] H. T. Kit and H. Chen, "Autonomous Elevator Inspection with Unmanned Aerial Vehicle," in *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, 2016, pp. 26-33.
- [32] L. Wenang, C. Li, and X. Zhiliang, "The design of automatic straightening machine system for high precision elevator guide rails," in *Proceedings of the 33rd Chinese Control Conference*, 2014, pp. 2964-2968.
- [33] N. Noguchi, A. Arakawa, T. Yoshimura, and Y. Sekiya, *Identification Method of Elevator Rail Deformation Using Operational Acceleration Data*. 2012, pp. 431-445.
- [34] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, 2009, vol. 3, no. 3.2, p. 5: Kobe, Japan.
- [35] D. Thomas. (2014). *ROS Introduction*. Available: <http://wiki.ros.org/ROS/Introduction>
- [36] J. Badger, D. Gooding, K. Ensley, K. Hambuchen, and A. Thackston, *ROS in space: A case study on robonaut 2*. 2016, pp. 343-373.
- [37] B. Tamer, "A New Approach to Linear Filtering and Prediction Problems," in *Control Theory: Twenty-Five Seminal Papers (1932-1981)*: Wiley-IEEE Press, 2001, p. 532.

- [38] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Department of Computer Science, University of North Carolina 2006.
- [39] H. Guoqiang, "Kalman Filter and Sensor Integration," ed. School of Electronic and Electrical Engineering, Nanyang Technological University.
- [40] T. Lacey. *The Kalman Filter*. Available: <http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf>
- [41] S. Yang and H. Li, "Application of EKF and UKF in Target Tracking Problem," in *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, 2016, vol. 01, pp. 116-120.
- [42] J. J. LaViola, "A comparison of unscented and extended Kalman filtering for estimating quaternion motion," in *Proceedings of the 2003 American Control Conference, 2003.*, 2003, vol. 3, pp. 2435-2440 vol.3.
- [43] Z. Zhang, S. Liu, G. Tsai, H. Hu, C.-C. Chu, and F. Zheng, *PIRVS: An Advanced Visual-Inertial SLAM System with Flexible Sensor Fusion and Hardware Co-Design*. 2017.
- [44] S. Hong, J. B. Song, J. H. Baek, and J. K. Ryu, "Visual odometry for outdoor environment using a downward-tilting camera and self-shadow removal algorithm," in *2012 12th International Conference on Control, Automation and Systems*, 2012, pp. 960-963.
- [45] T. Foote, "tf: The transform library," in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013, pp. 1-6.
- [46] *PerceptIn Ironsides SDK*. Available: <https://github.com/PerceptIn/IRONSIDES>
- [47] *ROS node driver for Xsens devices*. Available: https://github.com/xsens/xsens_mti_ros_node
- [48] *rplidar_ros*. Available: https://github.com/robopeak/rplidar_ros
- [49] A. Censi, "An ICP variant using a point-to-line metric," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 19-25.
- [50] *ROS laser scan tools*. Available: https://github.com/ccny-ros-pkg/scan_tools
- [51] *Transforms sensor_msgs/Imu messages into new coordinate frames using tf*. Available: https://github.com/ros-perception/imu_pipeline
- [52] *Ros nodes for single TeraRanger sensors*. Available: <https://github.com/Terabee/teraranger>
- [53] *robot_localization is a package of nonlinear state estimation nodes. The package was developed by Charles River Analytics, Inc. Please ask questions on answers.ros.org.* <http://www.cra.com>. Available: https://github.com/cra-ros-pkg/robot_localization
- [54] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," Cham, 2016, pp. 335-348: Springer International Publishing.
- [55] *Provides ROS interface to basic ARNL features. Requires ARNL and ArnlBase to be installed separately*. Available: <https://github.com/MobileRobots/ros-arnl>
- [56] *Provides nodes to assemble point clouds from either LaserScan or PointCloud messages*. Available: https://github.com/ros-perception/laser_assembler
- [57] *PCL (Point Cloud Library) ROS interface stack* http://wiki.ros.org/perception_pcl. Available: https://github.com/ros-perception/perception_pcl
- [58] *ROS node for Arducam camera with USB shield*. Available: https://github.com/ArduCAM/ArduCAM_USB_Camera_Shield
- [59] Y. M. Wang, Y. Li, and J. B. Zheng, "A camera calibration technique based on OpenCV," in *The 3rd International Conference on Information Sciences and Interaction Sciences*, 2010, pp. 403-406.

- [60] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330-1334, 2000.
- [61] J. Kannala and S. S. Brandt, "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 8, pp. 1335-1340, 2006.
- [62] *Open Source Computer Vision Library*. Available: <https://github.com/opencv/opencv>
- [63] I. E. Sutherland, "Three-dimensional data input by tablet," *Proceedings of the IEEE*, vol. 62, no. 4, pp. 453-461, 1974.
- [64] *ROS camera 2D Lidar extrinsic calibration tool*. Available: https://github.com/ehong-tl/camera_2d_lidar_calibration
- [65] *Python bindings to the pointcloud library (pcl)*. Available: <https://github.com/strawlab/python-pcl>
- [66] S. Katz, A. Tal, and R. Basri, *Direct visibility of point sets*. 2007.
- [67] *This is a Python project implementing the Hidden Points Removal operator on a point cloud seen from a chosen point of view*. Available: https://github.com/williamsea/Hidden_Points_Removal_HPR
- [68] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 573-580.
- [69] *Useful tools for the RGB-D benchmark*. Available: <https://vision.in.tum.de/data/datasets/rgbd-dataset/tools>

Appendix A

Specification of PIRVS stereo camera.

Specs	
CPU	Dual-core Cortex-A72 up to 2.0GHz Quad-core Cortex-A53 up to 1.5GHz
RAM	2GB
Internal Memory	8GB
External Memory	MicroSD up to 64GB
Stereo Baseline	65mm
Max Field of View	(D/H/V) 166°/133°/100°
Shutter	Global Shutter
Chroma	Color
Image Frame Rate	60fps
Image Resolution	640 x 480
IMU Frame Rate	200Hz
Size	103mm x 79mm x 33mm
Interface	USB 3.0 / UART

Appendix B

Specification of MTi-7-DK IMU.

Specifications MTi-7	
Orientation accuracy	
Roll/pitch (static)	0.5° 1σ RMS
Roll/pitch (dynamic)	0.5° 1σ RMS
Yaw (dynamic)	1.5° 1σ RMS
Position and velocity (with MTi-7-DK)	
Horizontal position 1σ STD (SBAS)	1.0 m
Vertical position 1σ STD (SBAS, baro)	2.0 m
Velocity 1σ RMS	0.05 m/s
Clock drift	1 ppm or external reference
All above specifications based on typical application scenarios	
Inertial sensor performance	
Gyroscope full-scale range	±2000°/s
Gyroscope bias stability	10 deg/hr
Gyroscope noise density	0.007°/s/√Hz
Gyroscope non-linearity	0.1% FS
Accelerometer full-scale range	±16 g
Accelerometer bias stability	0.03 mg
Accelerometer noise density	120 μg/√Hz
Accelerometer non-linearity	0.5% FS
Module specifications	
Power consumption	<100 mW
Input voltage	2.19 to 3.6 V
Package	SMD, footprint compatible with JEDEC PLCC-28
Size	12.1 x 12.1 x 2.55 mm
Weight	<1 g
Interfacing	
Hardware interface	I ² C, SPI, UART (selectable)
Software interface	Xsens Xbus binary protocol Driver source code supplied
Output data rate	0-800 Hz

Gyroscope specification ¹	Unit	MTi 1-series
Standard full range	[°/s]	±2000
In-run bias stability	[°/h]	10
Bandwidth (-3dB)	[Hz]	255
Noise density	[°/s/√Hz]	0.007
g-sensitivity (calibrated)	[°/s/g]	0.001
Sensitivity variation	[%]	0.05
Non-linearity	[%FS]	0.1

Accelerometer ²	Unit	MTi 1-series
Standard full range	[g]	±16
In-run bias stability	[mg]	0.03
Bandwidth (-3dB)	[Hz]	324 (Z: 262)
Noise density	[μg/√Hz]	120
Sensitivity variation	[%]	0.05
Non-linearity	[%FS]	0.5

Magnetometer ²	Unit	MTi 1-series
Standard full range	[G]	8
Non-linearity	[%]	0.2
Total RMS noise	[mG]	0.5
Resolution	[mG]	0.25

Parameter ²	Unit	MTi 1-series
Non-orthogonality (accelerometer)	[°]	0.05
Non-orthogonality (gyroscope)	[°]	0.05
Non-orthogonality (magnetometer)	[°]	0.05
Alignment (gyr to acc)	[°]	0.05
Alignment (mag to acc)	[°]	0.1
Alignment of acc to the module board	[°]	0.2

Appendix C

Specification of RPLIDAR A2M8 laser scanner.

Measurement Performance

- For Model A2M8 Only

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	0.15	A2M8-R3 and the belowing models -	8	Based om white objects with 70% reflectivity
			A2M8-R4 -	12	
Angular Range	Degree	-	0-360	-	-
Distance Resolution	mm	-	<0.5	-	<1.5 meters
			<1% of the distance		All distance range*
Angular Resolution	Degree	0.45	0.9	1.35	10Hz scan rate
Sample Duration	Millisecond(ms)	-	0.125	-	-
Sample Frequency	Hz	2000	8000	8010	
Scan Rate	Hz	5	10	15	The rate is for a round of scan. The typical value is measured when RPLIDAR takes 400 samples per scan

Laser Power Specification

- For Model A2M8 Only

Item	Unit	Min	Typical	Max	Comments
Laser wavelength	Nanometer(nm)	775	785	795	Infrared Light Band
Laser power	Milliwatt (mW)	-	3	5	Peak power
Pulse length	Microsecond(us)	60	87	90	-
Laser Safety Class	-	-	FDA Class I	-	-

Appendix D

Specification of TeraRanger EVO 60 m range sensor.

	TeraRanger Evo 60m
Principle:	Infrared Time-of-Flight (ToF)
Range:	*0.5m up to 60m
Update Rate:	Up to 240 readings per second
Output Resolution:	0.5cm below 14m, 2cm from 14m
Accuracy:	±4cm in the first 14m, 1.5% above 14m
Field of view:	Approx. 2°
Supply voltage:	5V DC +/-5%
Supply current (min-max):	90mA - 330mA
Interfaces:	USB 2.0 Micro-B
	UART, +3.3V level, 115200,8,N,1.
	I2C, +3.3V level, 400kHz
Connectors:	Single 9 pin Hirose DF13
	Micro USB
Weight:	9g (sensor) + 3g (backboard)
Dimensions:	Approx. 29x29x22mm (sensor + backboard)
Eye safety:	Yes (CE certified)

**Specifications are derived from tests in controlled conditions. Bright sunlight, target surface reflectivity and other variables will affect sensor performance. The combination of very bright sunshine and low reflectivity targets (such as grass) can reduce maximum range to 10m or less. For additional information, please find the following [test results report](#).*

Appendix E

Specification of Intel NUC7i7BNH barebone kit.

INTEL® NUC KIT NUC7i7BNH

Technical Specifications

Processor

- Intel® Core™ i7-7567U processor (3.5 GHz to 4.0 GHz Turbo,¹ Dual Core 4 MB cache, 28W TDP)

Graphics

- Intel® Iris™ Plus Graphics 650
- One HDMI* 2.0 port with 4K at 60 Hz
- Thunderbolt™ 3 port with support for USB* 3.1 Gen 2, DisplayPort* 1.2 and 40 Gb/s Thunderbolt

System Memory

- Two DDR4 SO-DIMM sockets (up to 32 GB, 2133 MHz), 1.2V

Storage Capabilities

- Micro SDXC slot with UHS-I support on the side
- One M.2 connector supporting 22x42 or 22x80 M.2 SSD
- One SATA3 port for connection to 2.5" HDD or SSD (up to 9.5 mm thickness)

Peripheral Connectivity

- Intel® Gigabit LAN
- Thunderbolt 3 port with support for USB* 3.1 Gen 2, DisplayPort* 1.2 and 40 Gb/s Thunderbolt
- Four Super Hi-Speed USB 3.0 ports (two back panel ports and two front ports including one charging port)
- Two additional Hi-Speed USB 2.0 ports via internal header

- Intel® Dual Band Wireless-AC 8265 (802.11ac), 2x2, up to 867 Mbps
- Dual Mode Bluetooth* 4.2

System Bios

- 64 Mb Flash EEPROM with Intel® Platform Innovation Framework for EFI Plug and Play
- Advanced configuration and power interface V3.0b, SMBIOS2.5
- Intel® Visual BIOS
- Intel® Express BIOS update support

Hardware Management Features

- Processor fan speed control
- Voltage and temperature sensing
- Fan sensor inputs used to monitor fan activity
- ACPI-compliant power management control

Expansion Capabilities

- Two Internal USB 2.0 ports via two 1x4 headers (for replaceable lid support)
- One AUX_PWR header
- One Consumer Electronics Control header

Audio

- Up to 7.1 surround audio via HDMI
- Headphone/microphone jack on the front panel
- Dual-array microphones on the front panel

Front Panel Header

- Reset, HDD LED, Power LEDs, power on/off

Mechanical Chassis Size

- 4.53" x 4.37" x 2.01"
- 115 mm x 111 mm x 51 mm

Baseboard Power Requirements

- 19V, 65W wall-mount AC-DC power adapter

Environment Operating Temperature

- 0° C to +40° C

Storage Temperature

- -20° C to +60° C

Product Safety Regulations and Standards

- IEC 60950-1
- UL 60950-1
- EN 60950-1
- CAN/CSA-C22.2 No. 60950-1

EMC/RF Regulations and Standards (Class B)

- CISPR 32
- FCC CFR Title 47, Chapter I, Part 15, Subparts B, C, E
- ICES-003
- EN 55032
- EN 55024
- ETSI EN 300 328
- ETSI EN 301 489-1
- ETSI EN 301 489-17
- ETSI EN 301 893
- EN 62311
- AS/NZS 2772.2
- AS/NZS 4268
- VCCI V-2, V-3, V-4
- KN-32
- KN-24
- CNS 13438

Environmental Regulations

- RoHS Directive 2011/65/EU
- WEEE Directive 2012/19/EU
- China RoHS - Management Methods for Restricted Use of Hazardous Substances in Electrical and Electronic Products

Appendix F

Specification of Arducam MT9N001 and LS-81600 fisheye lens.

Parameter		Value
Optical format		1/2.3-inch (4:3)
Active imager size		6.104mm(H) x 4.578mm(V) 7.63mm diagonal
Active pixels		3488H x 2616V
Pixel size		1.75 x 1.75 μ m
Chief ray angle		0°, 25°
Color filter array		RGB Bayer pattern
Shutter type		Electronic rolling shutter (ERS) with global reset release (GRR)
Input clock frequency		6–48 MHz
Maximum data rate	Parallel	96 Mp/s at 96 MHz PIXCLK
	CCP2	640 Mbps
	MIPI (two-lane)	1.536 Gbps
Frame rate	Full resolution	Programmable up to 9.7 fps parallel
	VGA	640H x 480V with 2X skip and 2X bin: 56.8 fps (full power), 50 fps (low power)
ADC resolution		12-bit, on-die
Responsivity		0.44 V/lux-sec (550nm)
Dynamic range		62.7dB
SNR _{MAX}		37.4dB
Power	Full resolution	485mW
Consumption	Preview	150mW low power VGA
	Standby	50 μ W (typical, EXTCLK disabled)
Operating temperature		–30°C to +70°C (at junction)

NO.	Items	公差
1	Sensor	1/2.3"
2	Size	41.65mmx14.65mmx107.75mm
3	Element	5C 2MG
4	TH	23.59±0.1mm
5	Resolution	16M
6	EFL	2.8±5%
7	BFL	4.95mm
8	F/NO	2.8±5%
9	F.O.V	垂直 95°
		水平 120°
		對角線 155°
10	Optical distortion	< 55.6%
11	TV distortion	< 15%
12	Thread size	M7.0P0.5mm

原稿	变更文件号	担当	变更日期	批准	审查	代号
原本	续款					LS-81600
						V0

Appendix G

G.1 Drawing and dimensions of aluminum mounting frame.

G.2 Drawing and dimension of acrylic plate to mount PIRVS stereo camera.

G.3 Drawing and dimension of acrylic plate to mount MTi-7-DK IMU sensor.

G.4 Drawing and dimension of acrylic plate to mount RPLIDAR A2M8 laser scanner and TeraRanger EVO 60 m range finder.

G.5 Drawing and dimension of acrylic plate to mount Intel NUC7i7BNH PC and the rest of the components.

G.6 Drawing of 3D printed mount for PIRVS stereo camera.

G.7 Drawing of 3D printed mount for RPLIDAR A2M8 laser scanner.

G.8 Drawing of 3D printed enclosure mount for MTi-7-DK IMU sensor.

G.9 Drawing and dimension of acrylic plate to mount Arducam MT9N001 USB cameras.

Appendix H

Panoramic images of the entire hallway.

$x = 4.152639, y = 0.011780, z = -0.006321$

cam3

cam0

cam1

$x = 5.207397, y = 0.021242, z = -0.005102$

cam3

cam0

cam1

$x = 6.253768, y = 0.044269, z = -0.006438$

cam3

cam0

cam1

$x = 7.281829, y = 0.032075, z = -0.005238$

cam3

cam0

cam1