Lidar-Camera Smart Simultaneous Localization and Mapping (LCS-SLAM) For Use in Racing Quadcopters

a project presented to

The Faculty of the Department of Aerospace Engineering

San José State University

in partial fulfillment of the requirements for the degree

Master of Science of Aerospace Engineering

by

Walter Harper

May 2022

approved by

Dr. Srba Jovic

NASA Ames Research Center/MFRA

ABSTRACT

Lidar-Camera Smart Simultaneous Localization and Mapping (LCS-SLAM) For Use in Racing Quadcopters

Walter Harper

Recent advances in sensor and computational technology have led to much research in autonomous robotics. In the airspace, autonomous aerial vehicles have been the focus of much research in various initiatives in private industry and government which have identified requirements for advanced sensing and navigation abilities of aircraft, even if they are piloted. As hardware computational capability has grown over recent years, it is now possible to create a SLAM system with both visual and LiDAR sensors that can generate detailed maps of an environment that a robot can learn to efficiently navigate which can help fulfill these requirements. LiDAR-Camera Smart Simultaneous Localization and Mapping (LCS-SLAM), leverages both LiDAR and visual sensors to accurately localize and map an indoor environment in which a racing drone must navigate. When combined with existing navigation modules, the LCS-SLAM system allows for a robot to navigate an environment efficiently and intelligently. The LCS-SLAM algorithm is developed and tested in a flight simulation environment using a racing quadcopter with proven results that showed an optimized global map of the environment and robots estimated trajectory generated from the LiDAR, monocular camera, and fused pipelines which leverage the sensors equipped onto a virtual quadcopter. A quantitative and qualitative analysis of the localization and mapping performance of LCS-SLAM has shown that LCS-SLAM has the potential to become an accurate solution for localization and mapping for autonomous racing quadcopters and potentially autonomous aerial vehicles in the future.

ACKNOWLEDGEMENTS

I would like to thank Dr. Srba Jovic for his guidance and insight into the field of autonomous aircraft. Additionally, I would like to thank Dr. Jovic for introducing me to autonomy in the airspace through his extensive contributions to the UAS in the NAS project at NASA Ames Research Center. I would also like to thank my colleagues Steven Rispoli and Brayan Mendez for their support in the development of a prototype autonomous quadcopter which provided the inspiration for this project. Lastly, I would like to thank my family for their constant support in my pursuit of graduate degree.

Table of Contents

List of Tables

List of Figures

Nomenclature

Symbol	Definition	Units (SI)
b	Stereo baseline distance	Meters (m)
d	Disparity (or parallax)	Pixels
e	Epipole, Error set of point cloud	Meters (m), Meters (m)
E	Absolute Trajectory Error Matrix	Special Euclidean 3
F	Relative Pose Error Matrix	Special Euclidean 3
f	Focal length	Meters (m)
I	Image matrix or Identity matrix	Pixels
J	Translation matrix for Root Mean Square minimization	-------
K	Kalman Gain, Camera Intrinsic Parameter Matrix	-------, -------
k	Radial distortion coefficient	-------
L	Rotation matrix for Root Mean Square minimization	-------

Symbol	Definition	Units (SI)
N	Dimensionality of Kalman Filter model	-------
n	Dimensionality of Kalman Filter model, pose samples	-------, -------
O	Optical center	Meters (m)
O’	Projection of optical center on image place	Pixels
P	3D Point in world coordinate frame, Pose estimate matrix	Meters (m), Special Euclidean 3
P’	Normalized 3D point in projected 2D image coordinate frame	Pixels
p	Point cloud set, tangential distortion coefficient	Meters (m), -------
Q	Process noise, Ground-truth estimate matrix	-------, Special Euclidean 3
R	Process noise in measurement space, Rotation Matrix	-------
S	Co-variance matrix in measurement space, Rigid-	-------, Special Euclidean 3

Symbol	Definition	Units (SI)
	transformation matrix between ground-truth and estimates.	
T	Cross co-relation matrix	-------
t	Time, Translation vector	Seconds (s), Meters (m)
u	Velocity in x-direction, Pixel coordinate in x-direction	Meters Per Second (m/s), Pixels
v	Velocity in y-direction, Pixel coordinate in y-direction	Meters Per Second (m/s), Pixels
w	Sigma point weights	-------
X	Sigma points matrix in state space, Coordinate in P-space	-------, Meters (m)
X’	Coordinate in P’-space	Pixels
Y	Coordinate in P-space	Meters (m)
Y’	Coordinate in P’-space	Pixels
Z	Sigma points in measurement space, Coordinate in P-space	-------, Meters (m)
Z’	Coordinate in P’-space	Pixels

Symbol	Definition	Units (SI)
Greek Symbols		
μ	Mean in state space	-------
Σ	Co-variance matrix, sum operator	-------, -------
ζ	Mean in measurement space	-------
$\tilde{\lambda}$	Free scaling parameter	-------
Δ	Time step between pose estimates	-------
Subscripts		
$()_0$	Initial condition	-------
$()_i$	Ith element of set	-------
$()_l$	Left camera coordinate frame	-------
$()_r$	Right camera coordinate frame	-------
$()_w$	World coordinate frame	-------
$()_c$	Camera coordinate frame	-------

Symbol	**Definition**	**Units (SI)**
$()_{c'}$	New camera coordinate frame	-------
$()_{uv}$	Pixel coordinate frame	-------
$()_{n}$	Nth element of set	-------
$()_{RMSE}$	Root-mean square of element	-------
$()_{\Delta}$	Time step between elements	-------
Acronyms		
2D	Two Dimensional	-------
3D	Three Dimensional	-------
AAM	Advanced Air Mobility	-------
ATE	Absolute Trajectory Error	-------
BOW	Bag-of-Words	-------
BRIEF	Binary Robust Independent Elementary Features	-------
CLAHE	Contrast Limited Adaptive Histogram Equalization	-------

Symbol	Definition	Units (SI)
DDS	Distributed Discovery Service	-------
DNN	Deep Neural Network	-------
EKF	Extended Kalman Filter	-------
FAST	Features from Accelerated Segment Test	-------
FDBOW	Fast Digital Bag-of-Words	-------
FLANN	Fast Library for Approximate Nearest Neighbors	-------
GICP	Generalized Iterative Closest Point	-------
GPU	Graphics Processing Unit	-------
GRV	Gaussian Random Variables	-------
ICP	Iterative Closest Point	-------
IMU	Inertial Measurement Unit	-------
INS	Inertial Navigation System	-------

Symbol	Definition	Units (SI)
LCS-SLAM	Lidar-Camera Smart Simultaneous Localization and Mapping	-------
LiDAR	Light Detection and Ranging	-------
LOAM	LiDAR Odometry and Mapping	-------
LSH	Locally Sensitive Hashing	-------
MLE	Maximum Likelihood Estimation	-------
PID	Proportional, Integral, and Derivative	-------
MAP	Maximum-a-Posteriori	-------
NAS	National Air Space	-------
NRMSE	Normalized Root-Mean Square Error	-------
ORB	Oriented BRIEF Rotated FAST	-------
RGB	Red, Green, Blue	-------
RGB-D	Red, Green, Blue – Depth	-------

Symbol	Definition	Units (SI)
RMSE	Root Mean Square Error	-------
ROS	Robert Operating System	-------
RPE	Relative Pose Error	-------
SE3	Special Euclidean Lie Group	-------
SIFT	Scale Invariant Feature Transform	-------
SLAM	Simultaneous Localization and Mapping	-------
SURF	Speeded Up Robust Features	-------
SVD	Singular Value Decomposition	-------
UAS	Unmanned Aerial Systems	-------
UAV	Unmanned Aerial Vehicle	-------
UKF	Unscented Kalman Filter	-------
VGICP	Voxelized Generalized Iterative Closest Point	-------

Symbol	Definition	Units (SI)
VO	Visual Odometry	-------
VR	Virtual Reality	-------

Chapter 1 – Introduction

1.1 Motivation

Mapping of a quadcopter racing track provides a proof of concept of the state-of-the-art technologies needed for larger scale AAM and UAS in the NAS [1] navigation in urban environments. Racing quadcopters provide particularly challenging conditions for robot sensing and navigation due to its high speeds, aggressive maneuvers, and limited computational resources. Autonomous mapping capabilities allows for a robot to track its location within an environment for collision avoidance or trajectory determination. Given that a robot has knowledge of its pose within an environment, optimized flight trajectories and adaptive control algorithms can be applied to allow for complex control of an aerial vehicle.

Racing quadcopters are typically controlled by skilled human pilots with the use of Virtual Reality (VR) goggles. This allows for a human to experience a first-person perspective of a quadcopter in real-time to aid in their decision making while piloting. In the last few years there has been a push to introduce autonomous racing quadcopters into the mainstream media with Lockheed Martin [2], Microsoft [3], and various universities throughout the world hosting competitions to drive quadcopter autonomy further. Autonomy in vehicles is not a novel concept, however recent advances in sensor and computational technology has led to much research in autonomous robotics. Recent research has allowed for engineers to leverage robotics technologies to advance the field of autonomous vehicles. In the airspace, autonomous aerial vehicles have been the focus of much research in Advanced Air Mobility (AAM) and Unmanned Aerial Systems (UAS) in the National Air Space (NAS) pushed forward by NASA. AAM aims to introduce aerial vehicles into urban areas that can transport packages and people. This requires advanced sensing and navigation abilities of aircraft, even if they are piloted. The UAS in the NAS initiative aims to reduce the technological and safety barriers required for unmanned aerial vehicles to navigate in the airspace [4]. Real-time multi sensor mapping and localization within an aircraft's environment is a critical need in the field of autonomy for aerospace applications.

State-of-the-art Simultaneous Localization and Mapping (SLAM) algorithms are currently used to solve robot mapping problems. Modern day SLAM algorithms are built around visual or Light Detection and Ranging (LiDAR) sensors and are capable of localization and mapping in real-time. LiDAR sensors have been a popular choice in the last 20 years for SLAM applications, however visual sensors such as Red, Green, Blue (RGB) color spectrum cameras have been the driving force behind modern SLAM research. Little research is publicly available that discuss SLAM algorithms that uses both LiDAR and visual sensors. As hardware computational capability has grown over recent years, it is now possible to create a SLAM system with both visual and LiDAR sensors that can generate detailed maps of an environment that a robot can learn to efficiently navigate.

LiDAR-Camera Smart Simultaneous Localization and Mapping (LCS-SLAM), leverages both LiDAR and visual sensors to accurately localize and map an indoor environment in which a racing drone must navigate. When combined with existing navigation modules, the LCS-SLAM system allows for a robot to navigate an environment efficiently and intelligently. LCS-SLAM is designed as an all-in-one modular system that can easily be adapted for different racing drones and indoor environments.

1.2 Literature Review

Review of existing literature on localization and mapping solutions for robotic applications using LiDAR, stereo cameras, monocular cameras, and Inertial Measurement Units (IMUs) are investigated further in detail below.

1.2.1 LiDAR SLAM

LiDAR odometry is defined as the pose estimate of a robot over a defined time history. Each individual robot pose is estimated using a localization algorithm. In the context of autonomous vehicles, LiDAR sensors come in three flavors: 1D range finders, 2D scanners, and 3D scanners. LiDAR sensors have enabled robots to determine their motion and localize within both 2D and 3D maps with a high degree of accuracy. 3D LiDAR sensors today are expensive but produce some of the most accurate mapping results which is often needed for large scale autonomous vehicles navigating large indoor or outdoor environments. 2D LiDAR's are more common in smaller robots and small unmanned aerial vehicles (UAV) for their low cost and acceptable mapping capabilities while indoors or in small outdoor spaces. Each of these sensors are critical for spatial awareness of unmanned vehicles and complex navigation in previously unmapped environments.

A state-of-the-art LiDAR odometry and mapping algorithm is presented by Zhang and Singh [5] called LiDAR Odometry and Mapping (LOAM). LOAM performs real-time odometry and mapping using 3D LiDAR scans of an environment. LOAM performs mapping and odometry using two separate algorithms which can also fuse IMU data to increase the accuracy of the odometry and mapping estimates. The odometry algorithm runs at a higher frequency in comparison to the mapping algorithm to allow for odometry data to be processed as quickly as possible. Mapping capabilities typically require additional optimization procedures that do not need to be run as fast as the odometry algorithm for successful robot localization.

In addition to 3D LiDAR sensors, 2D LiDAR sensors have been proven capable of mapping 3D environments with the aid of additional sensors such as an IMU, video camera, or by leveraging encoded servo motors which measures the rotation of a 2D LiDAR on a third axis. In Fang et al [6], a method for reconstructing 3D environment using a 2D LiDAR scanner is presented. The method leverages the use of high-level reconstruction algorithm and motion controller software in conjunction with an encoded servo to construct a 3D map in real time.

Improvements to the performance of 2D and 3D LiDAR SLAM algorithms can be achieved with the introduction of a pre-existing map of the environment. Zhen and Scherer [7] has developed a SLAM algorithm that combines the use of the Error State Kalman Filter with a Gaussian Particle Filter to estimate the state of the agent inside a pre-existing map. Testing of the authors algorithm was completed in both real-world and simulation environments which confirmed the hypothesis that given a pre-existing map of the environment, more accurate LiDAR based state estimation and mapping can be achieved.

In addition to providing a pre-existing map, additional sensors can be fused with a LiDAR sensor to obtain a more accurate state estimate. In Zhang and Singh [8] LiDAR odometry and mapping in real-time is investigated. The LiDAR odometry algorithm uses feature point extraction, then finds the feature point correspondence between LiDAR scans to estimate the odometry of a robot. The LiDAR odometry in combination with an IMU is used to assist in the development of a 3D map for an environment. The algorithm was tested successfully in both indoor and outdoor environments for a robotics application which proved that fusing IMU data with LiDAR gives more accurate state estimation. Xie et al [9] performed research on a similar sensor fusion between LiDAR odometry and IMU data to generate a 2.5D map of the environment using scan matching to perform localization. Test results showed that a 2.5D map of a real-life 3D environment was achieved at an acceptable accuracy at a reduced computational cost than traditional single sensor SLAM algorithms. In Li et al [10] a method for LiDAR and IMU based SLAM is proposed. The algorithm depends first on feature extraction of the environment to construct a feature map. The IMU data is then used to correct inaccuracy in the LiDAR sampled data with the use of a Kalman Filter. A prototype of the system was developed by the researchers to perform validation of the proposed system. The tested system proved that the LiDAR feature extraction algorithm was more accurate when fused IMU than if used alone.

With the high cost of LiDAR systems, testing in simulation environments has been critical to the development of LiDAR SLAM algorithms. To further test LiDAR in a virtual 3D environment, Allden et al [11] proposed a method to realistically simulate LiDAR sensor data. The Unity3D game engine is used to build and test a virtual LiDAR system and compare its output to real world output of a similar LiDAR sensor model. The authors take advantage of the Nvidia PhysX engine which leverages computer graphics rendering techniques such as ray-casting to generate laser data to model a LiDAR sensor. The authors concluded that it is feasible to build and simulate a LiDAR sensor in a virtual 3D environment that provides a level of realism that is acceptable for developing and testing SLAM algorithms.

Techniques to perform accurate place recognition and map optimizations for LiDAR SLAM applications has seen much research in the last few decades. In Wen et al [12] an alternative approach to map optimization is investigated. Fusion of IMU and LiDAR are used for environmental mapping with place recognition. The authors propose a method that provides accurate mapping of an environment leveraging a graph-based SLAM algorithm. The results showed acceptable accuracy in a 2D LiDAR slam system. In Hess et al [13] place recognition for a 2D LiDAR SLAM system is discussed. The system defines a global and local grid of which an environment is mapped to sub-maps. The sub-maps are compared to the LiDAR scanned global map to determine if a previously location has been visited before. Testing on GPU accelerated hardware showed acceptable real-time performance in real-world testing.

1.2.2 Camera SLAM

LiDAR data can be combined with data from visual sensors such as RGB cameras. The data from cameras can be used to perform object detection, or in the case of multiple cameras, depth perception. Depth sensing provides one of the foundational tools needed to perform stereo mapping and odometry. In addition, localization of objects within the stereo camera field of view allows for mapping of objects within an environment with the correct scale when compared to monocular camera systems. A stereoscopic camera can allow absolute 3D perception of the environment that can be used to generate a map, something that a monocular camera system is incapable of providing due to a problem known as scale ambiguity.

In Campos et al [14] a visual, visual-inertial, and multi-map SLAM framework known as ORB-SLAM 3 is discussed. ORB-SLAM3 is an accurate and advanced visual SLAM algorithm which is considered the current state-of-the-art visual SLAM system for its ability to handle a multitude of sensor configurations and handle scenes with poor features for tracking. ORB-SLAM3 uses a modified Oriented Fast Rotated Brief (ORB) feature detector and descriptor to find features in the environment. ORB-SLAM3 relies on Maximum-a-Posteriori (MAP) estimation to obtain accurate localization and mapping results. In addition, ORB-SLAM3 uses a Bag-of-Words (BoW) representation for its place recognition module. A state-of-the-art multi mapping framework, Atlas, is used to handle disconnected SLAM maps to allow real-time place recognition and robot re-localization in real-time. In Mito et al [15] methods for adapting object detection and stereo vision for robot state estimation are presented. These methods are based on feature extraction and matching corresponding feature points to generate an odometry estimate for a robot. Experiment results by the authors show that under stable robot motion the proposed algorithm can detect and extract moving objects with a stereo camera. Scaramuzza and Fraundorfer [15] discuss the techniques to perform visual odometry for both monocular and stereo camera systems. The underlying algorithms used for visual odometry, and their applications are discussed. Feature detection of 2D and 3D points are discussed with robot pose estimation methods including essential matrix estimation, perspective-n-point transformation, and iterative closest point algorithms. Scale recovery methods for monocular camera systems and triangulation methods for stereo camera systems are discussed. The authors concluded that the combination of 3D and 2D systems using stereo cameras provide some of the most accurate estimation of a robots odometry for well-formed environments.

Further research by Hu et al [16] investigates a place recognition algorithm which introduces an improved pyramid similarity score function for a camera-based SLAM system. A stereoscopic camera is used to provide depth measurements and pose estimation of a robot for the front end for a SLAM algorithm. Additional data is processed in the back end of the algorithm that improves the real-time performance of the place recognition algorithm. The author’s algorithm is tested against other prevalent algorithms and the results show that the author’s place recognition module results in more accurate real-time mapping of an environment than state-of-the-art algorithms. Liang et al [17] propose a novel approach to perform place recognition using Bag-of-

Words and ORB-SLAM features in conjunction with the local registration and global correlation framework with sparse pose adjustment optimization. The authors tested the algorithm in large real-world environments and verified its effectiveness versus traditional place recognition algorithms for computationally constrained robots.

1.2.3 LiDAR and Camera Fusion for SLAM

The combination of LiDAR and visual odometry data allow for the development of a robust SLAM system. In Li [18] close coupling of a Camera and LiDAR is investigated. The authors use a 2D LiDAR sensor on a moving platform to generate 3D LiDAR data. Close coupling of the LiDAR and Camera involve the use of rigid-body extrinsic calibration via feature targeting to accurately fuse the data between the two sensors. To calibrate both sensors, the relative translation and rotation between the sensors are determined from extracted features from a control checkerboard and matching point correspondences. To fuse the data from the LiDAR and camera, the 3D points generated by the LiDAR in the world coordinate frame are projected into coordinates on a 2D plane. Meanwhile, the camera is subject to rotation from a servo which allows for each captured image to be stitched together to form a panoramic image of the environment, mimicking the scanning capability of a LiDAR sensor. Results of the proposed sensor fusion method prove a viable approach to LiDAR and camera coupling that give reasonably accurate data for environmental mapping given that the sensors are properly calibrated with each other

In Zhang and Singh [19] LiDAR and a monocular camera are fused together to generate robust odometry and mapping of the environment. Camera images are used to estimate visual odometry at frequency of 60 Hz with LiDAR odometry running at a frequency of 1 Hz. The visual odometry estimate is refined by LiDAR odometry, with 3D LiDAR points used to help augment the visual odometry algorithm. The method proposed by the authors is considered one of the highest accuracy methods tested on some of the most popular benchmarking datasets.

In Deilamsalehy and Havens [20] sensor fusion accuracy is investigated between three different configurations of 2D LiDAR, IMU, and a camera. Sensor outputs are fused with an Extended Kalman Filter (EKF) and the accuracy of each configuration for state estimation is analyzed. The authors were able to show that the state estimation from a system with a 2D LiDAR, IMU, and camera were nearly an order of magnitude more accurate than a system with IMU and camera alone. 3D LiDAR further improved the state estimation accuracy by between two to five-fold. In Lopez et al [21] an approach of camera and LiDAR fusion with an IMU is presented to estimate the pose of an autonomous quadcopter in an 3D environment. The authors propose a fusion algorithm with the use of an Extended Kalman Filter. The HectorSLAM algorithm is used to generate a 2.5D map of the environment at low computational cost with the aid of an IMU. The researchers concluded that the fusion of LiDAR, IMU, and a camera increased the accuracy of a 2D mapped environment versus traditional non-fusion methods. Additional research by Jiang et al [22] using a multi-sensor system of LiDAR, Camera, and IMU resulted in the development of a new sensor fusion framework. The framework consists of graph optimization for a Red-Green-Blue Depth (RGB-D) camera for visual odometry, and feature extraction with scan matching for LiDAR odometry. The LiDAR and camera sensor data are fused to generate a 2.5D occupancy grid map of the environment. The mapping method results in a computationally efficient system when compared to other methods such as Adaptive Monte Carlo Localization and ORB-SLAM. In addition, the author's algorithm is more accurate than state-of-the-art methods at performing place recognition and robot re-localization which are critical components to building robust and accurate mapping capabilities.

In Smolyanskiy et al [23] an alternative approach to fused SLAM is presented by using Deep Neural Networks (DNNs). DNNs have been adopted for use in some sectors of the SLAM industry, most notably by the Nvidia Corporation for their Drive AGX platform. The authors use a DNN approach to navigate a low-flying autonomous aircraft in an unknown environment using fused sensor data. The robot performed autonomous navigation over a trail primarily using cameras with the aid of LiDAR. The authors were able to conclude that with a graphics processing units embedded onto a flight computer, such as the Jetson TX1, monocular vision-based navigation with the aid of LiDAR was sufficient for a drone to leverage DNNs to localize and map along a forest path.

1.3 Project Proposal

Based on the current state-of-the-art techniques and state estimation requirements needed for 3D indoor navigation of an autonomous racing drone the requirements of the LCS-SLAM algorithm are determined as follows:

- Sensor fusion between 3D LiDAR, cameras, and IMU sensors to enable robust and resilient SLAM in real-time.
- Optimized for real-time applications and easily extendable for future deployment on embedded hardware.

The final deliverable of the LCS-SLAM system is:

- LCS-SLAM algorithm tested in a virtual flight simulator on a racing quadcopter with proven results that show the map and path generated from the sensors equipped onto the virtual quadcopter.
- Quantitative and qualitative analysis of the localization and mapping performance of LCS-SLAM compared to state-of-the-art SLAM algorithms that use similar sensor configurations.

1.4 Methodology

To produce the deliverable for the LCS-SLAM system the following approach is taken:

I. Hardware Selection

   1. Selection of a LiDAR system that can be used on a racing quadcopter. This system is modeled in simulation for testing of the LCS-SLAM algorithm.

2. Selection of a stereo camera system for use in LCS-SLAM. This system is also modeled in simulation.

3. Selection of IMU system for use in LCS-SLAM. This system is modeled in simulation.

4. Selection of a racing quadcopter to test LCS-SLAM. This quadcopter is modeled in simulation.

II. Software Development

1. Selection of a 3D virtual simulation environment to test LCS-SLAM algorithm.

2. Selection of a framework to build LCS-SLAM to be used easily in both simulation and hardware environments.

3. From the selected LiDAR sensor model, virtual sensor data must be generated in the simulation environment.

4. From the selected camera model, virtual data must be generated in the simulation environment.

5. From the selected IMU model, virtual data must be generated in the simulation environment.

6. A flight controller capable of stabilizing and autonomously navigating the selected quadcopter model.

7. Development of an algorithm to fuse IMU, camera and LiDAR data with place recognition and map optimization.

III. Testing

1. Complete LCS-SLAM system equipped to a racing quadcopter in a virtual simulation that outputs robot odometry and a 3D map of the environment.

Chapter 2 – High Level System Architecture

The LCS-SLAM system consists of multiple pipelines known as: mono, LiDAR, and fused. These pipelines are selected for their ability to accurately map large scale environments with acceptable accuracy and real-time performance. These pipelines work together to produce accurate localization and mapping estimates. The system contains elements of both serial and parallel computation to ensure best real-time performance on a quadcopter. The outputs of the LCS-SLAM system are optimized poses forming flight paths (i.e., odometry) and point clouds (i.e., maps) from the LiDAR and fused pipelines. A diagram of the high-level overview of LCS-SLAM is shown in Fig. 2.1.

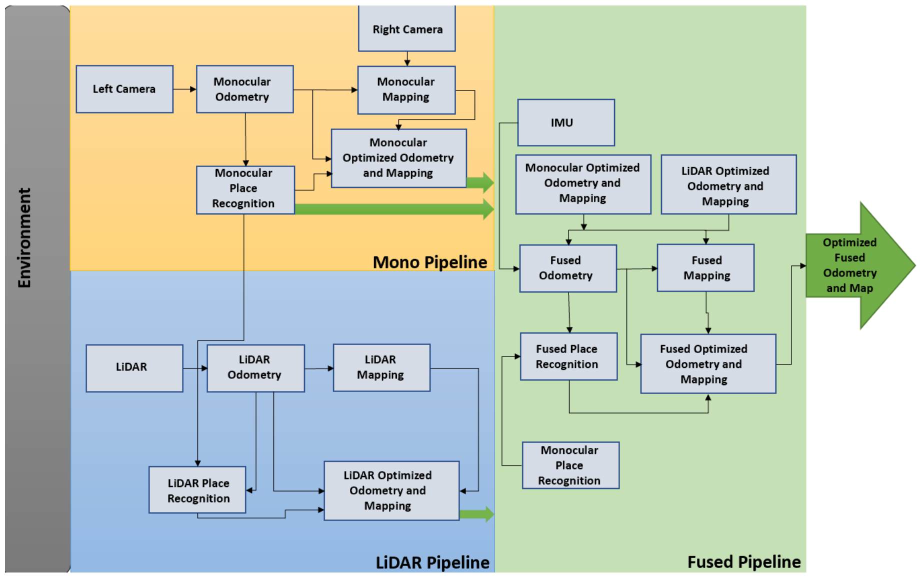
Environment
Right Camera
Left Camera
Monocular Odometry
Monocular Mapping
Monocular Optimized Odometry and Mapping
Monocular Place Recognition
Mono Pipeline
IMU
Monocular Optimized Odometry and Mapping
LiDAR Optimized Odometry and Mapping
Fused Odometry
Fused Mapping
Optimized Fused Odometry and Map
Fused Place Recognition
Fused Optimized Odometry and Mapping
Monocular Place Recognition
Fused Pipeline
LiDAR
LiDAR Odometry
LiDAR Mapping
LiDAR Place Recognition
LiDAR Optimized Odometry and Mapping
LiDAR Pipeline

Figure 2.1 – High level architecture for LCS-SLAM

2.1 LiDAR Pipeline

The LiDAR pipeline for LCS-SLAM consists of a 3D LiDAR sensor equipped onboard the selected quadcopter. This sensor can be real or simulated in a virtual environment for easier testing of the pipeline. Data output from the LiDAR sensor is registered with the LiDAR odometry module which performs localization of the quadcopter in the world frame. Combining the odometry estimate and LiDAR sensor returns, the LiDAR mapping module can map the environment around the quadcopter. The place recognition module uses monocular place recognition information to enable optimization of the localization and mapping outputs from the

LiDAR pipeline. The final output of the LiDAR pipeline is the optimized odometry and mapping estimates.

2.2 Monocular Pipeline

The monocular pipeline for LCS-SLAM uses the left camera in a stereo camera pair equipped onboard the selected quadcopter as an input. This sensor can be real or simulated in a virtual environment for easier testing of the pipeline. Data output from the camera is registered with the monocular odometry module which performs localization of the quadcopter pose in the world frame. The monocular odometry module is used to enable the place recognition module which is used by both the fused and LiDAR pipelines. The final output of the monocular pipeline is the optimized odometry and mapping estimates.

2.3 Fused Pipeline

The fused pipeline for LCS-SLAM uses the optimized outputs odometry and mapping from the mono and LiDAR pipelines combined with an IMU sensor equipped onboard the selected quadcopter. The IMU sensor can be real or simulated in a virtual environment for easier testing of the pipeline. The fusion of the robot odometry and mapping estimates from the various optimized pipelines leverages the Unscented Kalman Filter (UKF). The monocular place recognition module is leveraged to enable optimization of fused localization and mapping outputs from the pipeline. The final output of the fused pipeline is the optimized odometry and mapping estimates.

2.4 Other Pipeline Considerations

The current implementation of LCS-SLAM does not contain complete pipelines for stereo cameras or RGB-D cameras for various reasons given the selected application. The addition of a complete stereo camera pipeline could increase the accuracy of the localization and mapping estimate, however, would be significantly limited with its capabilities in large scale racing environments. Although the usage of a depth-sensing algorithm that can be implemented for real-time use is common, the feasibility of such system for a racing quadcopter is questionable given the very small baseline length of the stereo camera for a small application like a drone, which significantly reduces the effective range of a stereo camera system. In applications in which the stereo camera is not reliable, algorithms typically rely on monocular camera estimates for odometry which LCS-SLAM uses. An RGB-D camera pipelines has not been implemented given similar constraints to the stereo camera pipeline rational above. Different course configurations or use-cases in the future may present a more desirable test environment to further enhance LCS-SLAM with complete stereo and RGB-D pipelines.

Chapter 3 – Hardware Selection

Commercial-of-the-shelf hardware components are selected for emulating a quadcopter, LiDAR, camera, and IMU. Each of the sensor components are attached to the selected quadcopter, which have their physical properties modeled in a virtual simulation environment for testing of the complete LCS-SLAM system.

3.1 Quadcopter Selection

The selected quadcopter is presented in [24]. The design is based off the Lockheed Martin racing quadcopter, RacerAI, used in the 2019 AlphaPilot competition shown in Fig. 3.2. The quadcopter design includes various sensors including a stereoscopic camera and IMU. Its flight performance, flight characteristics, and computational capabilities are known for the purpose of testing the LCS-SLAM system. Although a LiDAR sensor is not presented in [24] on the selected quadcopter, a sensor is selected such that it could be integrated into the existing design. Relevant specifications of the chosen quadcopter for testing LCS-SLAM are presented in Table 3.1 and the selected Quadcopter can be seen in Fig. 3.1.

Table 3.1 - Selected quadcopter specifications.

Property	**Quantity**	**Units**
Take-off weight (w/o LiDAR)	1.846	Kilograms
Total static thrust	2,178	Gram-force
Thrust-to-Weight ratio	4.79	N/A

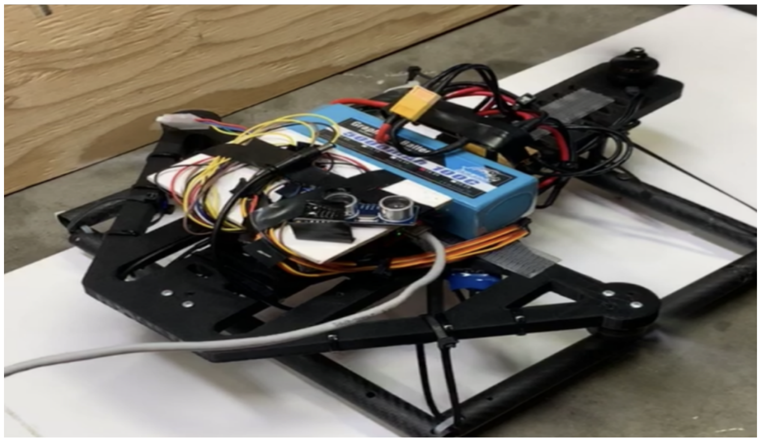
Figure 3.1 - Selected quadcopter design [24].

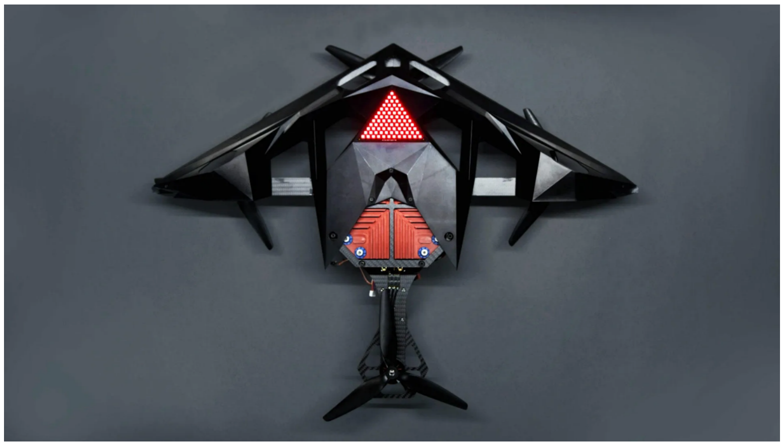
Figure 3.2 – AlphaPilot RacerAI drone [25].

3.2 LiDAR Selection

The LiDAR chosen to integrate with the selected quadcopter for testing LCS-SLAM is the Velodyne VLP-16 Puck Lite. The LiDAR was selected for its relatively small form factor, long range, and 3D scanning capability. In addition, the VLP-16 Puck Lite has desirable mass properties which allow for seamless integration with the selected drone design with minimal impact to its stability, assuming it is placed about the current center of gravity. The specifications of the VLP-16 Puck Lite are shown in Table 3.2, while the sensor can be seen in Fig. 3.3.

Table 3.2 – VLP-16 Puck Lite LiDAR specifications.

Property	**Quantity**	**Units**
Weight	0.590	Kilograms
Sample rate	Adjustable: 5 / 10 / 20	Hertz
Scanning range	100	Meters
Distance resolution	2	Millimeters
Azimuth resolution	Adjustable: 0.1 / 0.2 / 0.4	Degrees

Figure 3.3 - VLP-16 Puck Lite LiDAR sensor [26].

3.3 Camera Selection

The selected camera is the Arducam BO200. The camera was selected for testing with LCS-SLAM as it is equipped in a stereo configuration on the selected quadcopter as shown in [24]. The camera specifications when configured as a stereo pair are detailed further in Table 3.3, with an image of the camera in Fig. 3.4.

Table 3.3 - Selected camera specification.

Property	Quantity	Units
Resolution	640x480	Pixels x Pixels
Configuration	Horizontal Stereo	N/A
Diagonal field of view	100	Degrees
Baseline	14	Centimeters
Frame rate	30	Frames Per Second

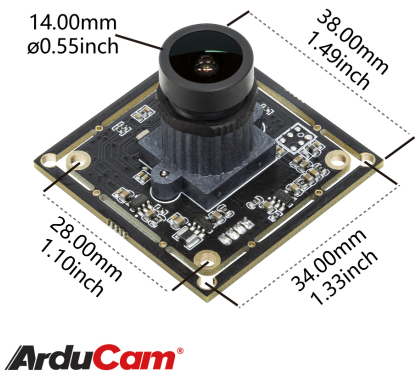

Figure 3.4 - Arducam BO200 camera [27].

3.4 IMU Selection

The selected IMU is the Bosch BNO055 based on the quadcopter design in [24]. The selected sensor has an accelerometer, gyroscope, magnetometer, and thermometer. This sensor is chosen for its low cost, accurate attitude measurements, and internal sensor fusion algorithms. The specifications of the sensor are shown in Table 3.4, and the IMU can be seen in Fig. 3.5.

Table 3.4 - Selected BNO055 sensor specification.

Property	**Quantity**	**Units**
Sample rate (accelerometer)	100	Hertz
Sample rate (gyroscope)	80	Hertz

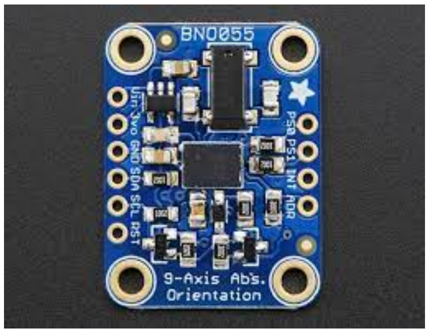

Figure 3.5 - BNO055 IMU sensor [28].

Chapter 4 – Virtualization of Sensors

The selected sensors for testing LCS-SLAM are modeled in a virtual environment to allow for synthetic data generation. The platform used to simulate the selected hardware is critical in enabling accurate simulation that can seamlessly transition to real-world applications. In the real-world, sensors would be directly integrated with the LCS-SLAM system, however, in a virtual environment there must be a compatibility layer to supply communication between the simulated sensors and the various modules and pipelines in the LCS-SLAM algorithm. The communication framework used in the development of LCS-SLAM is Robot Operating System 2 (ROS2). The Galactic Geochelone distribution of ROS2 that was initially released in May of 2021 is used. ROS2 is a framework which supplies tools, libraries, and conventions aimed at simplifying the task of creating complex robot systems. ROS2 uses individual applications, or nodes, which communicate between each other using defined messages assigned to topics in a publish-subscribe software messaging paradigm. This allows for researchers to prototype software using simulated sensor data and quickly deploy the software to hardware with minor changes to the underlying software design. The middleware layer in ROS2 that allows communication between applications is provided by a selected Distributed Discovery Service (DDS) such as eProsima Fast DDS, Eclipse Cyclone DDS, or RTI Connext. The selected DDS middleware used in the development of LCS-SLAM is eProsima Fast DDS. Another major ROS2 package, Rviz2, provides the visualization tools that are helpful for developing and evaluating the performance of LCS-SLAM.

4.1 Selection of Simulation Environment

The Unity3D game engine is selected as the virtual simulation environment to test the LCS-SLAM system and virtualize the required sensors. Unity3D is chosen for its support of the Nvidia PhysX simulation engine which allows for classical control of a robot with Newtonian physics. Features such as computer graphics ray-casting and 3D rendering of environments allows for simulation of both LiDAR and camera sensors. The simulation environment also provides support for connectivity with ROS2 via the Unity Robotics Hub. For visualization of LCS-SLAM performance, the ROS2 package Rviz2 is used. Two quadcopter racing courses were developed to test LCS-SLAM. A quadcopter was navigated around the two quadcopter course configurations and the relevant sensor data was recorded to provide easy testing of the LCS-SLAM system without the need to run the simulation environment. These two recorded datasets are named *Track-Easy* and *Track-Hard.* The quadcopter racing courses developed to test LCS-SLAM onboard a racing quadcopter are presented in Figs. 4.1-4.4 below.

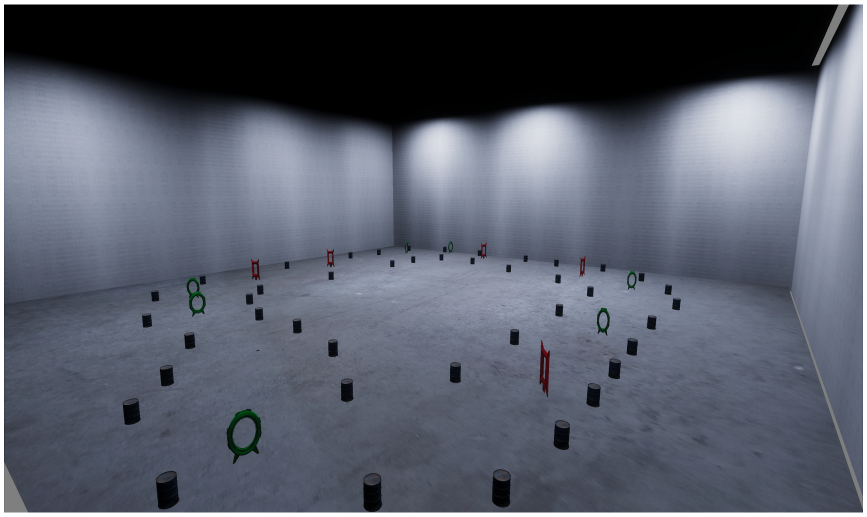

Figure 4.1 – Isometric view of *Track-Easy* course built in Unity3D.

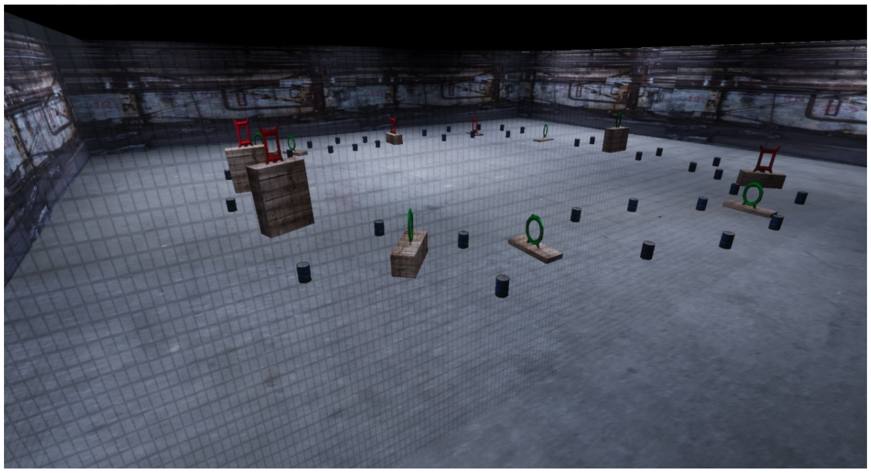

Figure 4.2 – Isometric view of *Track-Hard* course built in Unity3D.

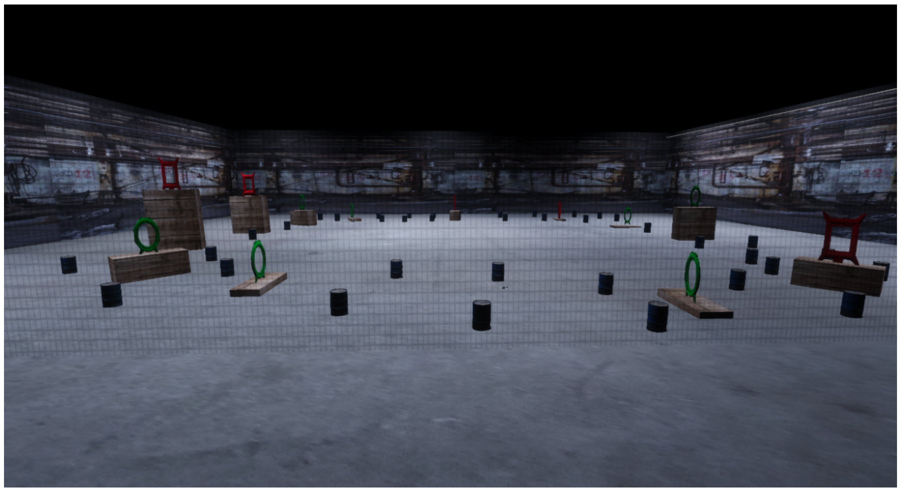

Figure 4.3 – Side view of *Track-Hard* course built in Unity3D.

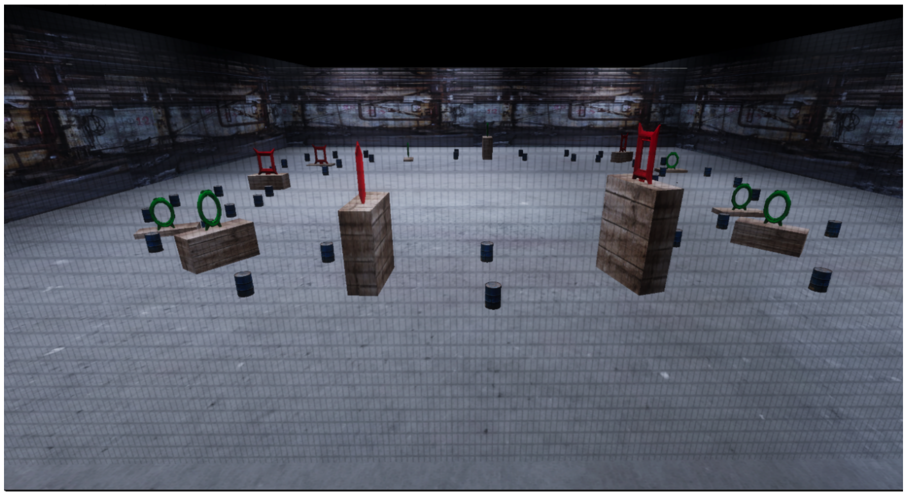

Figure 4.4 – Back view of *Track-Hard* course built in Unity3D.

4.2 Quadcopter Dynamics Model

The mass properties and dynamics of the selected quadcopter are already known and are presented in [24]. A flight controller that allows for simple flight through a virtual course via waypoints or manual control is provided for interfacing with the LCS-SLAM system. The provided controller is a Proportional, Integral, and Derivative (PID) controller tuned to the parameters in [24]. The physics model used in simulation includes aerodynamic drag, motor torques, and motor forces applied to rigid-body dynamics for simulation of the physical forces applied to and from the quadcopter. The Nvidia PhysX simulation engine allows for customization of the physics time step used for integration in solving the rigid body dynamics. The selected physics integration frequency was chosen to be 1000 Hz which allows for realistic modeling of motion in real-time with the selected sensor sampling rates and flight controller signal frequencies. The simulated quadcopter model is provided in Figure 4.5.

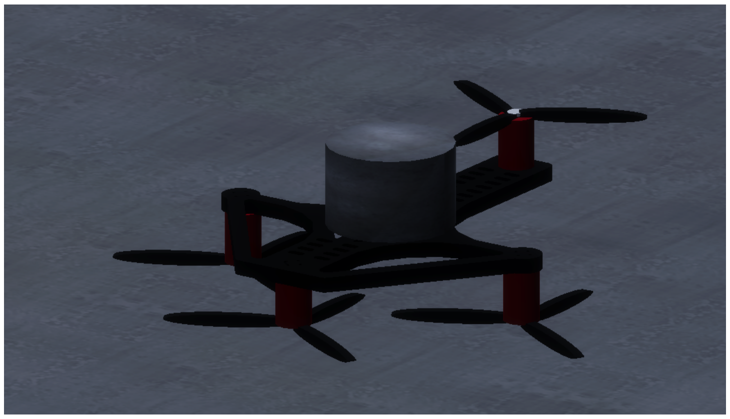

Figure 4.5 - Simulated quadcopter model.

4.3 LiDAR Data Generation

The selected LiDAR model is simulated using the Nvidia PhysX ray-casting functionality. Ray-casting is a computer rendering technique which allows for directional light rays to be sent from a source point which can reflect off objects back to the source. When objects in a scene are detected, the returned ray-casts hold information on the distance and time it took to return from a collision with an object. The LiDAR model is built in Unity3D using the technique presented in [11] to emulate the datasheet specifications of the Velodyne VLP-16 Puck Lite LiDAR presented in [26]. The LiDAR can produce up to nearly 300,000 points a second with this technique which is then consolidated into a stream of packets that is published by ROS2 that conforms to the specification defined in [26]. It should be noted that real-world materials that may be non-reflective or highly refractive are not modeled for simplicity of the simulation. Fig. 4.6-4.8 show the LiDAR scanner in use during the simulation used for testing LCS-SLAM.

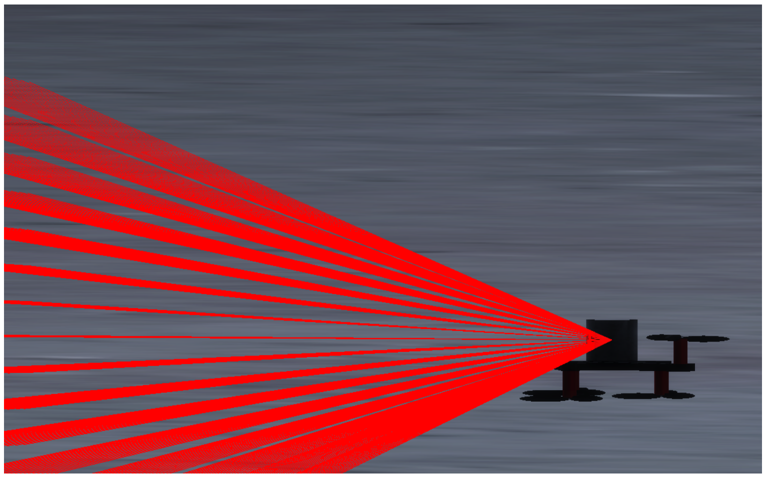

Figure 4.6 - Velodyne VLP-16 Puck Lite scanning pattern.

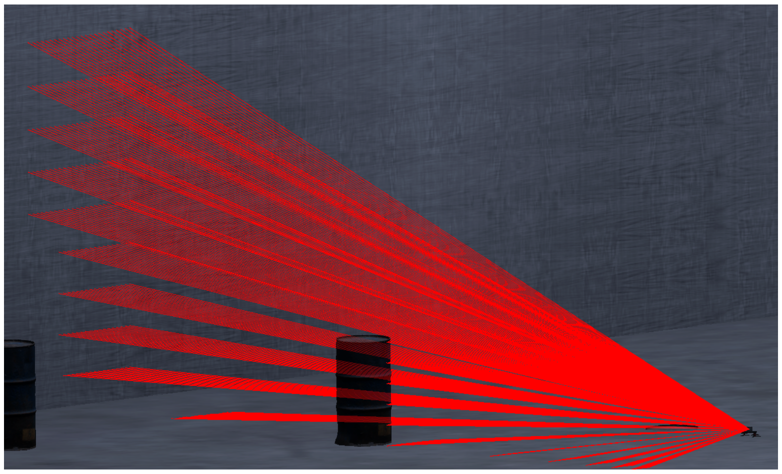

Figure 4.7 - PhysX ray-cast detection from Velodyne VLP-16 Puck Lite.

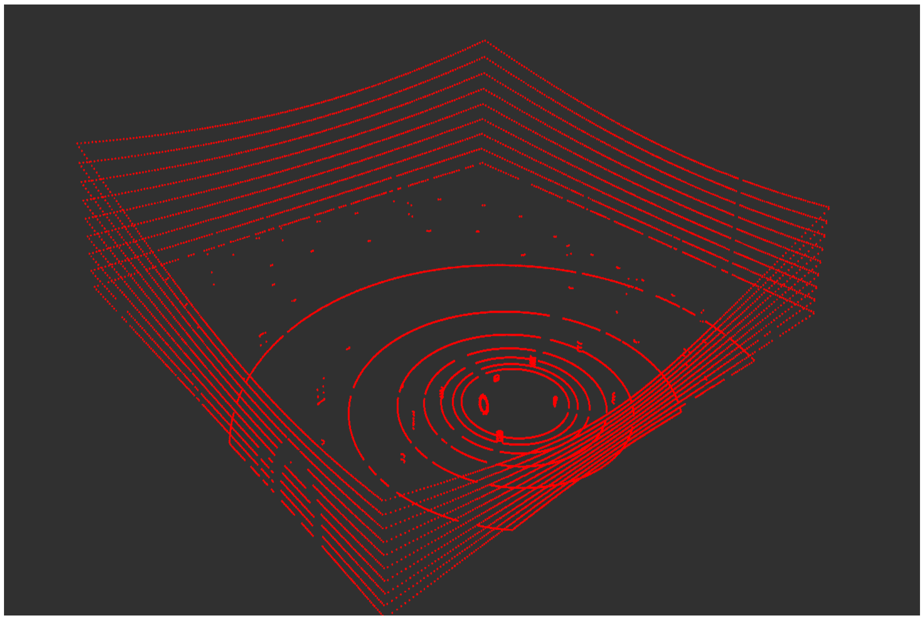

Figure 4.8 - LiDAR point cloud from a VLP-16 scan generated from Unity3D.

4.4 Camera Data Generation

The selected camera is simulated as a pinhole camera in a horizontal stereo configuration with the specifications that match what is shown in Table 3.3. Both cameras in the stereo configuration are modeled to be hardware synchronized and have motion blur when moving at high speeds. The stereo camera output data is compressed into JPG format and published by ROS2. The simulated left camera output in the stereo configuration is provided in Fig. 4.9.

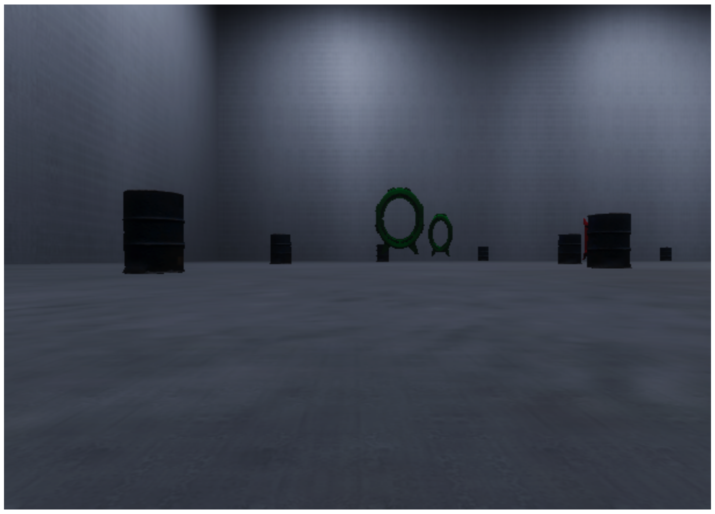

Figure 4.9 – Simulated left camera output in a stereo configuration.

4.5 IMU Data Generation

IMU data is generated from the quadcopter's rigid body motion while under the control of the Nvidia PhysX engine. Within the simulation environment, the position and attitude of the quadcopter in the world frame is captured at a fixed time step and can be considered the "ground truth" data. To introduce sensor noise, the IMU sensor readings are artificially varied from its true value. The artificial noise is assumed to be Gaussian in nature. The data from the IMU is generated at a rate of a 100 Hz and has accelerometer and gyroscope data that is published by ROS2.

Chapter 5 – Monocular Pipeline

The simulated sensors from Chapter 4 provide the ROS2 topics that the monocular pipeline requires as an input. The outputs of this pipeline provide the basis for the place recognition modules of the LiDAR and fused pipelines. A scale ambiguous odometry estimate is also provided as a pipeline output for reference. The two major modules within the monocular pipeline are the visual odometry and place recognition modules, which are discussed in further detail below.

5.1 Monocular Visual Odometry

In LCS-SLAM, the monocular visual odometry algorithm uses successive images taken from the left camera in the stereo camera pair to estimate the pose of the quadcopter. Since each individual camera output is an image of the 3D world projected onto a 2D image plane, it is important to establish a geometric model which can be used to describe the projection process. The projection process allows for the 3D point of an object relative to the camera to be determined from a 2D image captured of the environment. The most popular camera geometry model used today in computer vision is the pinhole camera geometry model. The pinhole camera geometry model describes an infinitely small hole which projects 3D object points into the 2D image space. The 2D image space is known as the image plane, while the plane where the pinhole exists is known as the camera plane. The camera plane has an optical center *O* which describes the hole in the pinhole model. This optical center is separated from the image plane projection of the optical center *O'* by a known focal length *f*. Both *O* and *O'* are the centers of their respective coordinate systems (*x',y',z'*) and (*x,y,z*). Light from a point *P* in 3D space passes through the pinhole and is projected onto the image plane as *P'*. Fig. 5.1 shows the geometric relationship between the image and camera planes with respect to a 3D point in space.

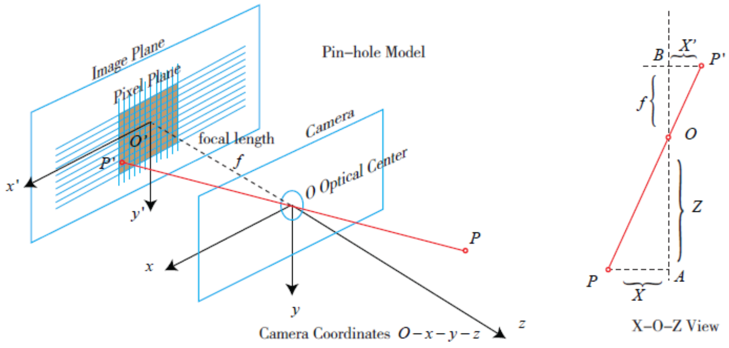

Figure 5.1 - Pinhole camera model.[29]

Using the similarity of triangles from the pinhole geometric model, the relationships between the 3D point *P* in the (*X, Y, Z*) coordinate system can be related to the projected point *P'* coordinate system (*X', Y', Z'*) as seen in Eq. (5.1). Note that images are typically inverted when projected onto the image plane, however most modern cameras do not output an inverted image which the pinhole model suggests. The image plane can be symmetrically placed in front of the camera frame to flip the inverted image and give us the final form of the geometric relationships in the pinhole camera model shown in Eq. (5.1).

$$\frac{Z}{f} = \frac{X}{X\prime} = \frac{Y}{Y\prime} \tag{5.1}$$

Using the pinhole camera model, a pixel coordinate system, and Eq. (5.1) one can determine the projection of 3D point *P* onto a camera normalized plane. The relationship between the pixel coordinate system and the image coordinate system can be described by the camera intrinsic calibration parameters. These parameters can be provided by the manufacturer or determined from a calibration process. The derivation of such is outside the scope of this paper and it assumed that the internal camera calibration parameters are described by a matrix *K.* In addition to the camera intrinsic parameters, lenses are used in modern cameras to get a larger field of view of a scene. Lenses change how light is projected onto the image plane from the 3D world by bending it. To continue using the pinhole camera model with cameras equipped with lenses, a correction for the radial and tangential distortion from the lens is required for complete camera calibration. Like the camera intrinsic parameters, the derivation of such correction factors is out of the scope of this paper and is described by a set of coefficients k_n and p_n up to an *n* order of accuracy which can be determined from the popular checkboard camera calibration processes.

Given a calibrated camera, the pixel coordinate system from images taken with modern cameras can be mapped into the image coordinate system and evidently the camera coordinate system through the pinhole model. In addition to the camera intrinsic calibration parameters, there are camera extrinsic parameters which describe the rotation of the camera *R* and the translation of the camera *t* with respect to the world coordinate system. The camera extrinsic parameters are used in solving for motion in LCS-SLAM when given an input from a single camera streaming video. With these concepts in mind, the relationship between a point in the pixel coordinate space *(u, v)* can be related to a point in the 3D world space *(X, Y, Z)* with Eq. (5.2).

$$P_{uv} = K(RP_W + t) \tag{5.2}$$

Note that the point in world space P_W is described in the camera normalized coordinate space where the camera coordinates are normalized by the depth of the scene *Z*. This is important since this means a single camera view is not sufficient in determining the depth of a scene as it is lost during the projection process. The 3D position of objects in space are only able to be determined up to a scale of depth *Z* for monocular cameras, leading to the well-known scale ambiguity problem.

In LCS-SLAM, unprocessed camera images are passed to the odometry algorithm and corrected for camera lens distortion. Once corrected for lens distortion, the input image is filtered using Contrast Limited Adaptive Histogram Equalization (CLAHE). CLAHE is a histogram equalization algorithm with a contrast limiting threshold that splits the image into a rectangular grid which enhances edges for feature detection. CLAHE filtered images can be seen in Fig. 5.2.

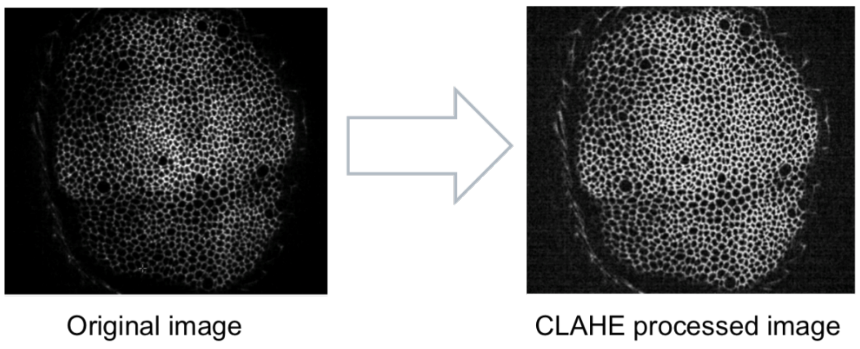

Figure 5.2 – Generic CLAHE processed image [29].

Once equalized the image is then converted from RGB to greyscale for further processing by the pipeline. Using the undistorted and equalized image view, 2D features are detected using an ORB feature detector and binary descriptor. Features from an image can consist of corners, edges, and blocks that are scale and rotation invariant. The exact algorithm that ORB [30] uses for finding features is outside the scope of this paper, however Fig. 5.3-5.4 show the ORB feature detection breakdown and image pyramid used to achieve scale and rotation invariant features.

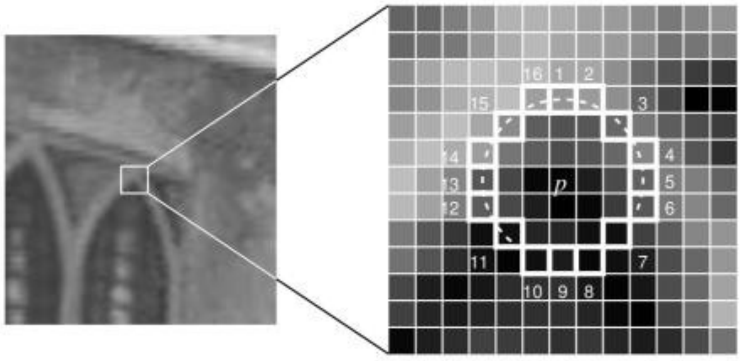

Figure 5.3 – ORB feature extraction process [30].

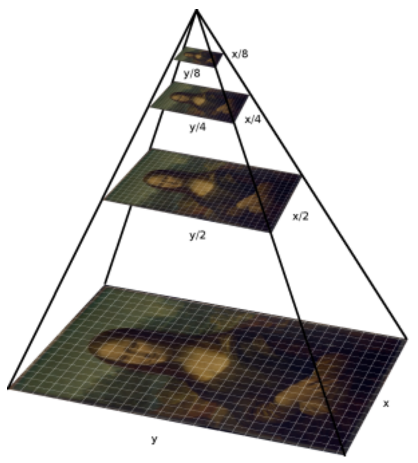

Figure 5.4 – Image pyramid showing how scale invariant features are detected [30].

In LCS-SLAM the ORB detector and descriptor is tuned to generate the best 1500 features in an image calculated by the Harris Score of the features. In addition, the Features from Accelerated Segment Test (FAST) detector threshold used in ORB is limited to increase the number of features initially detected. Once features are detected in an image view, they are matched with against a temporally successive image frame. LCS-SLAM uses a Fast Library for Approximate Nearest Neighbors (FLANN) based Matcher using Locally Sensitive Hashing (LSH) to handle matching of binary feature descriptors using Hamming distances which represent the distance between two binary strings. FLANN is used to quickly perform feature matching when compared to traditional binary feature matching methods such as the Brute-Force Matcher because FLANN only performs approximations of matches and is typically used for the more computationally taxing floating point feature descriptors such as Speeded Up Robust Features (SURF) and Scale Invariant Feature Transform (SIFT). In LCS-SLAM, the best two nearest neighbor matches are calculated for each feature in the image from the FLANN matcher, which are then filtered based on its estimated ambiguity. For each feature in the current image view, if there are two features that closely match from an earlier view then Lowe's Ratio Test is used to reject features that don't meet a specified threshold of uniqueness. In LCS-SLAM, the threshold for the Lowe's Ratio Test is a 25% difference between the first "best matched" feature and the second "best matched" feature to be considered sufficiently unique matches. The best matched features between two image views of a generic scene are shown in Fig. 5.5. The real-time matched features when running LCS-SLAM is shown in Fig. 5.6.

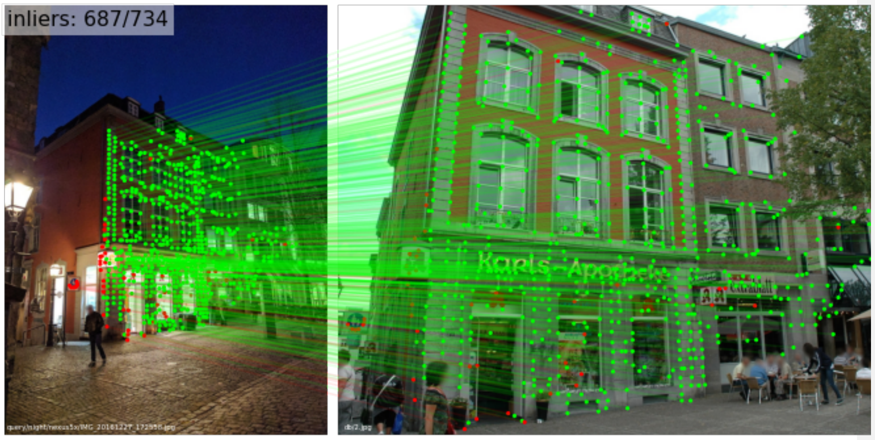

Figure 5.5 – Keypoint feature matching between two images [29].

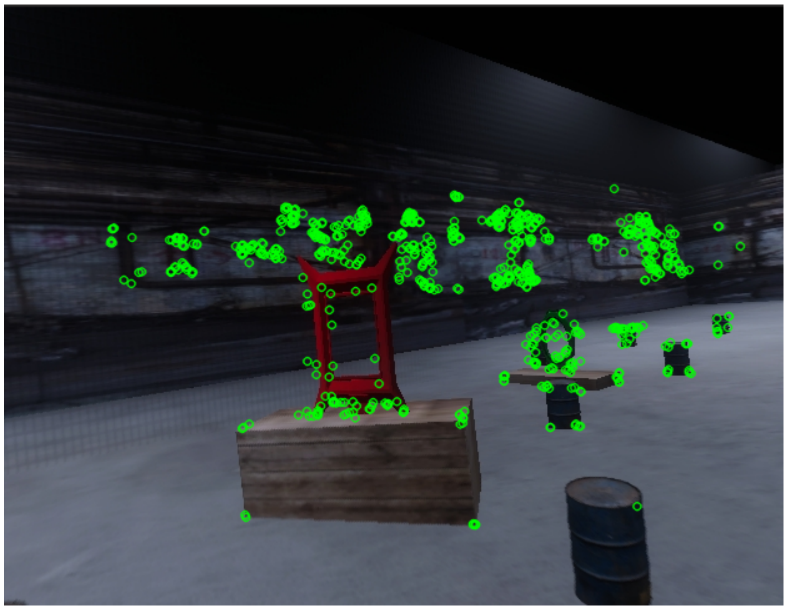
Figure 5.6 – Best matched ORB features in LCS-SLAM.

The “best matches” that pass Lowe’s Ratio Test are then used to estimate the relative pose of the robot by estimation a technique known as essential matrix estimation using the well-known Nister 5-point algorithm [31] with Levenberg-Marquardt least squares optimization to handle noisy matches. Decomposing the final estimated essential matrix yields the relative rotation between two camera frames and a unit vector of the relative translation between the two camera frames with ambiguous scale between the two image views. Depending on the number of feature inliers recovered from the essential matrix estimation process, the relative pose estimate is either kept as a keyframe and used for odometry or rejected in favor of the next image view. To recover an estimate of the proper translation scale, the most recent LiDAR position magnitude estimate is used, however the system is still prone to scale drift because the LiDAR and monocular pipeline keyframes are not synchronized in LCS-SLAM.

Based on the estimated pose between frames, the magnitude of the translation and rotation is checked against a set of thresholds to determine whether the current image view should be considered a keyframe. If so, the keyframe can be used to estimate the odometry of the robot. The criteria to be considered a keyframe in the monocular keyframe is experimentally obtained and shown in Table 5.1. The monocular odometry keyframes are shown in Fig. 5.7.

Table 5.1 – Monocular odometry keyframe selection properties.

Keyframe Property	**Quantity**	**Units**
Translation minimum	0.5	Meters
Rotation minimum	3.0	Degrees

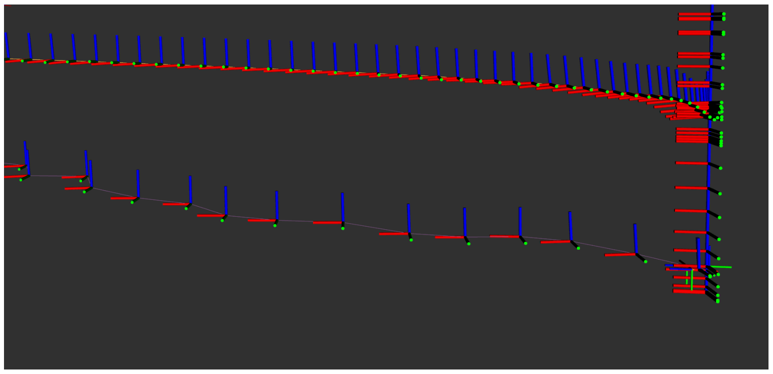
Figure 5.7 – Monocular keyframe selection (purple) compared to ground truth (red).

With the pose recovered and keyframes selected, the data can be used for further processing in the place recognition module and mapping modules.

5.2 Monocular Mapping

Using the odometry estimates and the aid of multiple camera views from the left camera in the stereo camera pair, 2D features detected in the image can be triangulated into 3D space given that the stereo camera is calibrated properly. The concept of stereo vision aims to solve for the depth by using two perspectives to view a point from the same scene. Fig. 5.8 shows two cameras that are placed in a parallel horizontal stereo configuration observing the same point in a scene. Once again P represents the point in 3D space, where u_L and u_R represent the pixel coordinate along the horizontal axis for each camera.

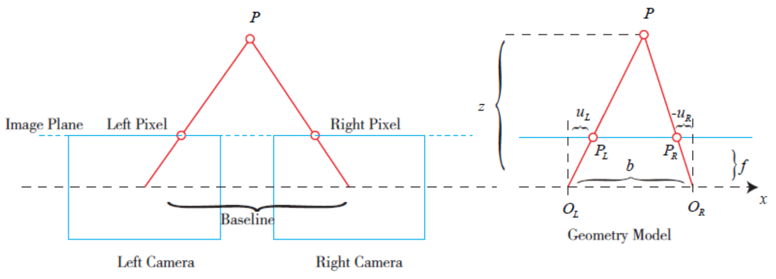

Figure 5.8 – Horizontal stereo camera geometry model [29].

O_L and O_R represent the optical centers of the left and right camera planes, f represents the focal length of the camera, b represents the baseline between the two cameras, and P_L and P_R represent the projection of the 3D object points on the left and right image planes.

Using the similarity of triangles from the geometric model in Fig 5.9, it is possible to determine the relationship required to obtain the depth of a point in the scene as shown in Eq. (5.3).

$$z = \frac{fb}{d}, d \triangleq u_L - u_R \tag{5.3}$$

Eq. (5.3) shows that the depth of the scene for parallel cameras can be recovered by a stereo camera if the disparity (sometimes referred to as parallax) d can be found for the point projected onto each image plane. Given the general case when the cameras are not parallel the relative rotation and translation between the cameras must be known to triangulate points in 3D space. This combination of rotation and translation along with the camera intrinsic matrix forms the projection matrix for each camera view. The solution to perform triangulation for the monocular pipeline in LCS-SLAM is to use successive camera views of matched features in the scene combined with odometry estimates to initialize projection matrices for each camera view. Given matched features between two camera views and the relative translation and rotation between them in 3D space, Direct Linear Transformation can be used to project points into 3D space. Points that are triangulated incorrectly are filtered out by LCS-SLAM. The output of the monocular mapping module of LCS-SLAM using this triangulation technique is shown in Fig. 5.9.

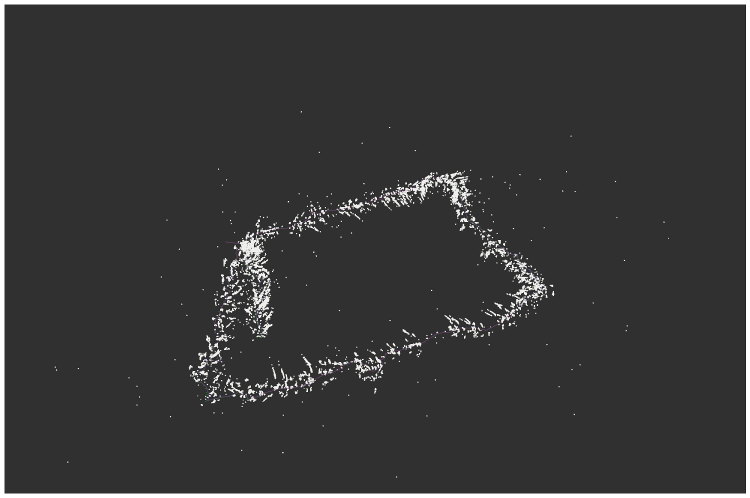

Figure 5.9 – Monocular mapping module.

5.3 Monocular Place Recognition

Given that there are inaccuracies in the state estimation process, errors in odometry and mapping grow exponentially over time if left uncorrected. To enable a global optimization process, the robot must know if it is at a previously visited location also known as loop-closure detection. In the context of an autonomous racing drone, this means the robot must know when it has arrived at the same part of the course that it has visited previously. Loop closure detection requires a robot to have place recognition capabilities. Loop closure in LCS-SLAM uses features from each monocular camera keyframe that is indexed and saved into a database. This database holds a Bag-of-Words representation that allows for real-time recall of previous monocular camera keyframes. By leveraging the *Fast Digital Bag-of-Words (FDBoW)* open-source software library, a Bag-of-Words representation of a keyframe and its respective features allow for efficient recall of similar images in a database, with relatively good precision. The exact details on how Bag-of-Words works is outside the scope of this paper, however it is important to identify Bag-of-Words as a histogram-based similarity comparison method using a predetermined vocabulary database that describes features [32]. A high-level visualization of the process is shown in Fig. 5.10.

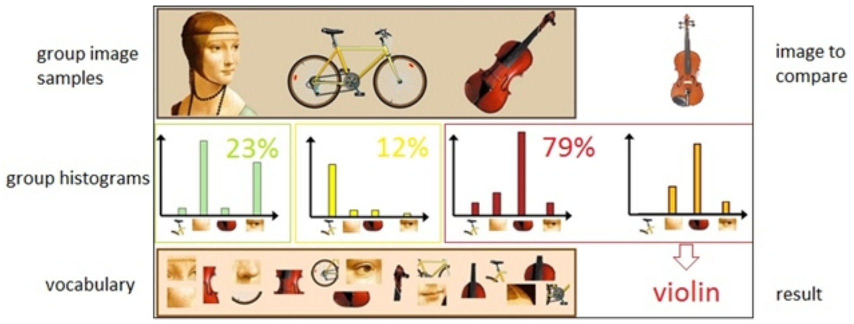

Figure 5.10 – Bag-of-Words high-level overview [32].

By using histograms to compare vocabulary between images, a k-d tree can be used to efficiently lookup a database of existing images and their respective features. Each new keyframe is compared to the keyframes stored in the Bag-of-Words database, and if the new keyframe meet a similarity threshold which uniquely identify an environment, then the robot has successfully detected a loop closure. Once loop closure is detected, a global optimization processes such as Pose Graph Optimization can be used to successfully optimize the graph or “close the loop” as used in the LiDAR pipeline. An example of “closing the loop” from current state-of-the-art SLAM algorithms can be seen in Fig. 5.11.

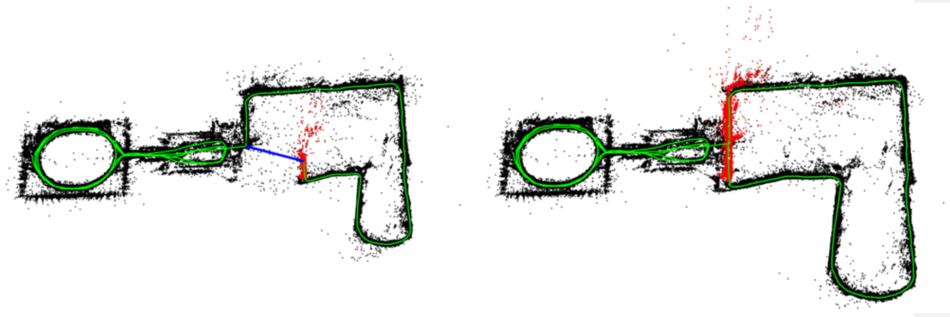

Figure 5.11 – Generic loop closure impact on map where red is the detected loop closure points [33].

Chapter 6 – LiDAR Pipeline

The simulated sensors from Chapter 4 provide the ROS2 topics that the LiDAR pipeline requires as input. The output of this module is optimized quadcopter odometry and map of the environment which can be used in the fusion pipeline. The LiDAR pipeline consists of odometry, mapping, place recognition, and optimization modules described in further detail below.

6.1 LiDAR Odometry

LiDAR odometry estimates the quadcopter motion based on successive LiDAR scans of the surrounding environment. Each LiDAR packet sent from the sensor is collected, and based on the sensor calibration and configuration, a point cloud can be obtained. The LiDAR processing algorithm in LCS-SLAM uses the *velodyne_pointcloud* ROS2 package which takes raw sensor measurements of azimuth, intensity, and distance for each laser scan in conjunction with the sensor calibration parameters and determines the local cartesian coordinate points returned by each laser. The selected LiDAR sensor, the VLP-16, contains 16 lasers with their own unique geometric positions which must be accounted for to accurately transform the sensor measurements to the cartesian coordinate system. Once the initial processing of the of LiDAR sensor data is complete, a point cloud of each scan is generated and can be used for further processing by the odometry module.

The first step in the LCS-SLAM LiDAR odometry pipeline is to filter the incoming point cloud scans to ensure only unique points and inliers are considered. This allows the entire pipeline to run faster and produce a more accurate result. Three different filters are applied to the data to prepare it for further processing: box filter, voxel down-sample Filter, and removing invalid points. A box filter eliminates all points outside of defined Cartesian coordinate bounds. For the selected VLP-16 LiDAR, bounds of +/- 120 meters is selected for each axis which is set based on the maximum range of the VLP-16 sensor given in Table 3.2. Voxel down-sampling of the data then reduces the number of points in the point cloud, drastically decreasing computation times in further processing steps. Voxel down-sampling applies a voxel grid with a specified leaf size in which points that fall within a leaf are binned together. This helps eliminate similar points from a point cloud by representing them as one single entity. The specified leaf size is dependent on the sensor resolution from Table 3.2 as well as the desired computational efficiency increase, which comes at the cost of accuracy. A leaf size of 1.0 meter is selected through experimentation for use in LCS-SLAM. Lastly, any points in the cloud that do not have a valid cartesian coordinate are removed, ensuring only valid points are considered by the odometry module. The parameters used for LiDAR point cloud filtering are provided in Table 6.1.

Table 6.1 – LiDAR point cloud filtering parameters.

Property	**Quantity**	**Units**
Box filter radius	120	Meters
Voxel leaf size	1.0	Meters

Next, the odometry module uses successive scans and compares them to each other in a process known as scan matching. Scan matching is used to determine the relative pose of the quadcopter between successive scans by computing a transform which minimizes the distance between corresponding points in two temporally successive point clouds with similar scene views. If the point clouds from two successive scans *P* and *P'* are defined as a set of points p_i and p'_i respectively, the transformation between each of the scans defined by the rotation *R* and translation *t* can be represented by Eq. (6.1).

$$p_i = Rp'_i + t \tag{6.1}$$

It is assumed that the closest points between the point clouds are the same points that undergo a Euclidean transformation between consecutive scans. Using the matched points, a point cloud registration technique known as Voxelized Generalized Iterative Closest Point (VGICP) is used to determine the transformation between the point clouds. The VGICP algorithm is an augmented version of the ICP algorithm with a focus on real-time processing and better performance in indoor 3D environments. An example of the ICP registration process for a pair of generic point clouds is shown in Fig. 6.1.

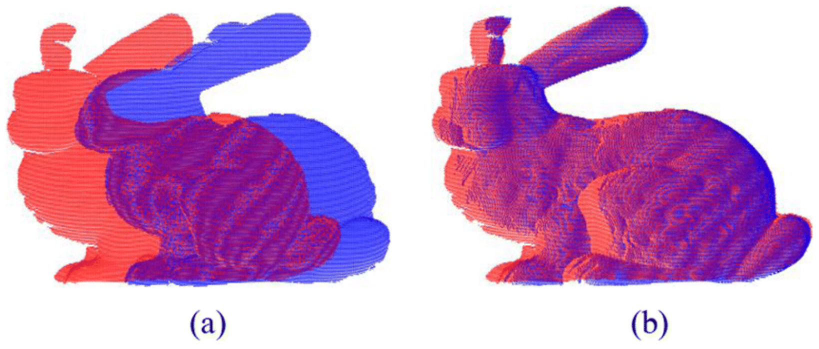

Figure 6.1 – ICP algorithm alignment for two similar point clouds.

The Iterative closest point algorithm defines the error *e* between two points to be defined as in Eq. (6.2).

$$e_i = p_i - (Rp_i' + t) \tag{6.2}$$

The error term is then used to construct a least-square problem between two points which when minimized is used to determine the rotation *R* and translation *t* as shown in Eq. (6.3).

$$\min_{R,t} \frac{1}{2} \sum ||(p_i - (Rp_i' + t))||^2 \tag{6.3}$$

Given the least-square problem in Eq. (6.3), the iterative closest point algorithm requires that the centroid of the point clouds to be computed. Once the centroids are computed, the rotation that aligned the current scan to the previous scan can be obtained by using Singular Value Decomposition (SVD). The translation can be solved by solving for the vector that connects the two centroids from the consecutive point clouds. By integrating each estimated transformation over each previous transformation, an absolute pose of the quadcopter can be determined as shown in Eqs. (6.4) and (6.5).

$${}_{w}R_{c'} = {}_{w}R_{c} * {}_{c}R_{c'} \tag{6.4}$$

$${}_{w}t_{c'} = {}_{w}t_{c} + {}_{c}t_{c'} \tag{6.5}$$

ICP alone can estimate the relative and absolute robot translation and rotation. To allow for more robust transformation estimates in indoor environments, GICP can be used. The GICP algorithm is based on the Iterative Closest Point (ICP) and Point-to-Plane algorithms which can be broken down into steps:

Iterative Closest Point Algorithm

1. Given two input clouds *A* and *B*, compute correspondences for each point within a specified matching threshold.

2. Compute a transformation which minimizes the distance between two corresponding points.

Point-to-Plane Algorithm

1. Given two input clouds *A* and *B*, compute correspondences for each point within a specified matching threshold.

2. Compute a transformation which minimizes the error along the surface normal of *one* of the cloud scans.

However, for more robust and real-time performance a GPU accelerated VGICP algorithm is used in LCS-SLAM. VGICP extends the GICP algorithm. This allows VGICP to perform plane-to-plane correspondences which minimizes the error along the surface normal for both point clouds which is common in LiDAR scans, and not well handled by ICP and Point-to-Plane algorithms individually. To ensure real-time performance and accurate results, there are three termination criteria selected for the algorithm as shown in Table 6.2.

Table 6.2 – GICP termination criteria in LCS-SLAM.

Termination Criteria	**Criteria Number**	**Quantity**	**Units**
Transformation epsilon	1	1e-10	Meters
Euclidean fitness	2	1e-3	Meters
Number of iterations	3	250	N/A

Termination criterion 1 and 2 stop the algorithm if a good match is found and reduce unnecessary iterations from being performed. Termination criteria enforces the real-time constraint at the cost of accuracy of the scan matching process. These values were determined based off experimentation with the given simulation environment.

Next in the LiDAR odometry module, keyframes are selected based on the transformation estimated from the scan matching step. Keyframes are selected based on enough rotation or translation occurring between successive scans which limits drift when the quadcopter is not moving. The experimentally obtained keyframe threshold between scans are shown in Table 6.3.

Table 6.3 – LiDAR keyframe selection properties.

Keyframe Property	**Quantity**	**Units**
Rotation minimum	3.0	Degrees
Position minimum	1.0	Meters

If the LiDAR scan passes the threshold for being a keyframe, it can be used to estimate the odometry and for further processing by the mapping module. If there have been no other keyframes processed by LCS-SLAM, the odometry system will mark the scan as the reference scan and be initialized, thus all successive scans will be relative to the reference scan. Each keyframe from this module is given a unique identification number and registered with the most recent monocular camera keyframe identification number. This registration process is important for the optimization module in the pipeline. A visualization of the LiDAR odometry keyframes are shown in Fig. 6.2.

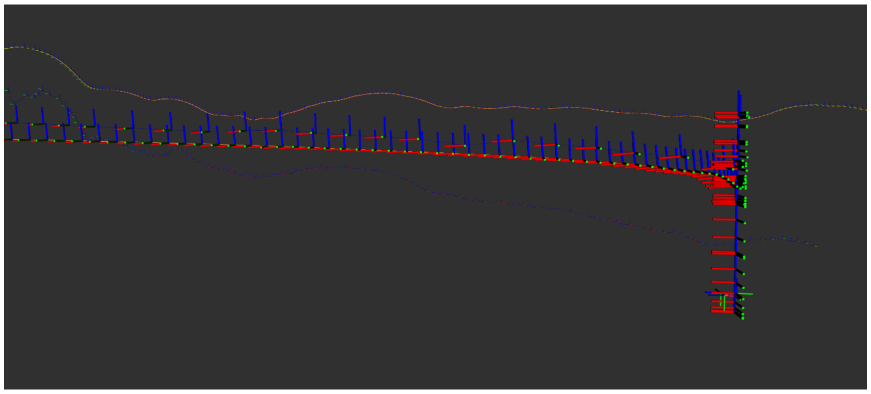
Figure 6.2 – LCS-SLAM LiDAR odometry keyframes (blue) compared to ground truth (orange).

6.2 LiDAR Mapping

LiDAR maps are created from the odometry of successive LiDAR scans of the environment and the 3D points directly generated from the LiDAR sensor returns. Laser returns from each of the scans generate sparse map points of the environment up to 100 meters in any direction given the selected sensor. Visualization of the LiDAR map generated by LCS-SLAM is shown in Fig. 6.3.

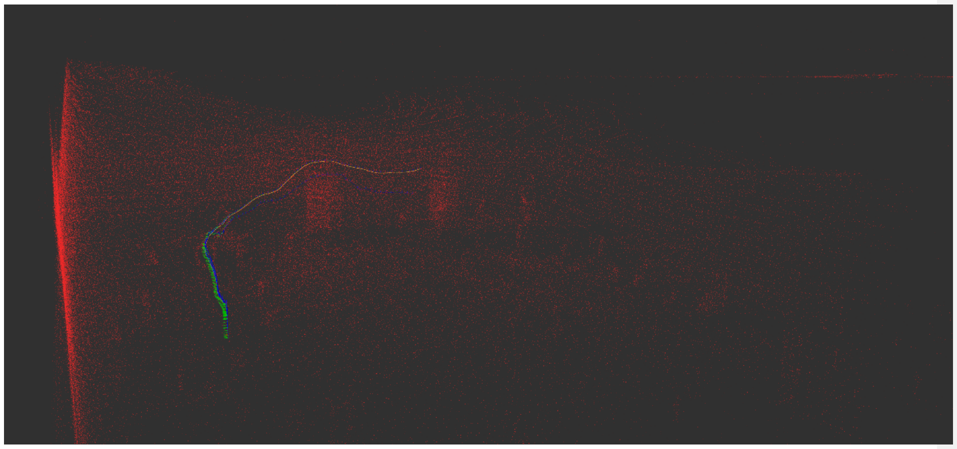
Figure 6.3 – LCS-SLAM LiDAR mapping module.

6.3 LiDAR Place Recognition

Given that the monocular pipeline has correctly identified a “loop closure” from its place recognition module, the monocular camera keyframe identification number and its registered LiDAR scan identification number associated can be used to recover the LiDAR scans in which the loop closure occurs. These two scans are then aligned using VGICP as described in the LIDAR odometry module. The relative odometry between these two scans is then used to create a loop closure relationship in the optimization module.

6.4 LiDAR Optimization

A technique to reduce the impact on odometry drift is to perform Pose Graph Optimization. The open source *g2o* hypergraph optimization software library is leveraged which allows for representation of the robot odometry as a pose graph. Pose Graph Optimization, or more generally factor graph optimization, uses Graph Theory and Lie-Algebra groups to represent the SLAM problem in terms of nodes and edges to perform global optimization of for SLAM systems. Nodes are represented by robot state measurements generated from odometry and converted to a Special Euclidean (SE3) Lie-Algebra Group representation, where edges are the relative odometry measurements between each robot state. Each edge also consists of an information matrix or Fisher Matrix which consists of the inverse covariances from the measurement uncertainty for each state variable used in the state estimation process. When a loop closure is detected, a new edge is added to the graph constraining the original state measurement and the matched state measurement from the place recognition module. The estimated rotation and translation between the matched scans are then used as the edge constraint in the graph and optimization of the graph can commence.

To properly perform an optimization, the initial node of the graph must be set fixed, otherwise the optimization problem will not be properly constrained. During the optimization process a gradient descent method is used to iteratively calculate the Maximum-Likelihood-Estimate (MLE) of the robot poses by minimizing the weighted sum of residual errors present in the graph. It should be also noted that bundle adjustment is a specific case of pose graph optimization where landmark points are identified as additional vertices connected by edges containing the projection information to generate the landmarks, however it is not used by the LCS-SLAM LiDAR pipeline.

Optimization of the pose graph is done by minimizing the least-squares problem using the well-known Levenberg-Marquardt gradient descent optimization method.

After reaching termination criterion or exhausting the maximum number of iterations allowed for solving the problem, the updated graph is used to optimize the previous odometry estimates and map points on a global scale.

Chapter 7 – Fused Pipeline

The simulated sensors from Chapter 4 and the outputs from the LiDAR and mono pipelines provide the ROS2 topics that the fused pipeline requires as inputs. The output of this module is the optimized quadcopter odometry and map of the environment which is the optimal output of LCS-SLAM. The pipeline contains odometry, mapping, place recognition, and optimization modules.

7.1 Fused Odometry

In the LCS-SLAM algorithm, LiDAR, Camera, and IMU sensor data can be fused together using an Unscented Kalman Filter (UKF). The UKF is an extension of the Extended Kalman Filter (EKF) which is capable of fusing non-linear sensor data assuming the data points are Gaussian Random Variables (GRV). The EKF leverages a Taylor expansion to achieve first-order linearization of the non-linear sensor model which can result in sub-optimal state estimation performance and in some cases divergence of the filter. The UKF remedies the flaws of an EKF by sampling a set of GRV sample points that are transformed via the Unscented Transform which can capture the system performance up to third-order accuracy without explicit linearization as required in the EKF. The basis of the UKF can be defined with state prediction and measurement update steps like the traditional Kalman Filter.

The prediction step of the UKF depends on a set of sigma points. Sigma points are individual points from the data distribution which are representative of the whole distribution itself. The algorithm behind the prediction step is like that of the EKF without the explicit linearization step. This step is replaced by the Unscented Transform which consists of:

1. Calculating sigma points
2. Determining weights of the sigma points
3. Transformation of sigma points to generate a predicted mean and co-variance.

The number of sigma points needed for accurate representation of the system is given by Eq. (7.1), where *N* is the dimensionality of the problem.

$$NumberofSigmaPoints = 2N + 1 \tag{7.1}$$

The sigma points matrix *X* of dimensionality *N* is calculated as shown in Eqs. (7.2) and (7.3) where μ is the mean of the Gaussian distribution, *n* is the dimensionality of the system, λ is the scaling factor which defines the distance from the mean for selecting sigma points which is typically selected as 3-*n*, and Σ is the co-variance matrix. The square root of a matrix can be calculated with the aid of the Cholesky decomposition.

$$X_0 = \mu \tag{7.2}$$

$$X_i = \mu + \left(\sqrt{(n + \lambda)\Sigma}\right)_i \tag{7.3}$$

Next in the algorithm is to compute the weights *w* of each sigma point as shown in Eqs. (7.4) and (7.5).

$$w_0 = \frac{\lambda}{(n+\lambda)} \tag{7.4}$$

$$w_i = \frac{1}{2(n+\lambda)} \tag{7.5}$$

Once the weights are computed, the transformation of the sigma points to calculate a new mean and co-variance is done as shown in Eqs. (7.6) and (7.7) where *f* is defined as the non-linear process function, and *R* is defined as the process noise.

$$\mu' = \sum_{i=0}^{2n} w_i \, f(X_i) \tag{7.6}$$

$$\Sigma' = \sum_{i=0}^{2n} w_i \, (f(X_i) - \mu')(g(X_i) - u')^T + R_t \tag{7.7}$$

The update step of the UKF uses the sensor measurement data to further enhance the estimate from the prediction step. To achieve this, the sigma points must be transformed from state space to measurement state space by a function *h*, then the mean and co-variance from the prediction step can be represented in measurement space by ζ and *S* respectively, with external process noise *Q* as shown in Eqs. (7.8), (7.9), and (7.10).

Deleted: ¶

$$Z = h(X_i) \tag{7.8}$$

$$\zeta = \sum_{i=0}^{2n} w_i \, Z \tag{7.9}$$

$$S = \sum_{i=0}^{2n} w_i \, (Z - \zeta)(Z - \zeta)^T + Q \tag{7.10}$$

To calculate the Kalman gain, *K*, Eqs. (7.11) and (7.12) are defined where *T* is the cross-co-relation matrix between state and predicted space.

$$K = TS^{-1} \tag{7.11}$$

$$T = \sum_{i=0}^{2n} w_i \, (X_i - \mu')(Z_i - \zeta)^T \tag{7.12}$$

Given the calculated quantities variables above, the final predicted mean and co-variance is given by Eqs. (7.13) and (7.14). This updated value is used in the prediction of the next iteration of the filter.

$$\mu_{i+1} = \mu_i{}' + K_i\left(\zeta_{measured,i} - \zeta_i\right) \tag{7.13}$$

$$\Sigma_{i+1} = (I - K_i T_i)\Sigma_i{}' \tag{7.14}$$

The UKF essentially allows all equipped sensors on the robot to be used to provide the best estimate of the robot odometry and map, given each sensors strengths and limitations which are reflected in the covariances that the filter is initialized with. The LCS-SLAM fused odometry keyframes are visualized in Fig 7.1.

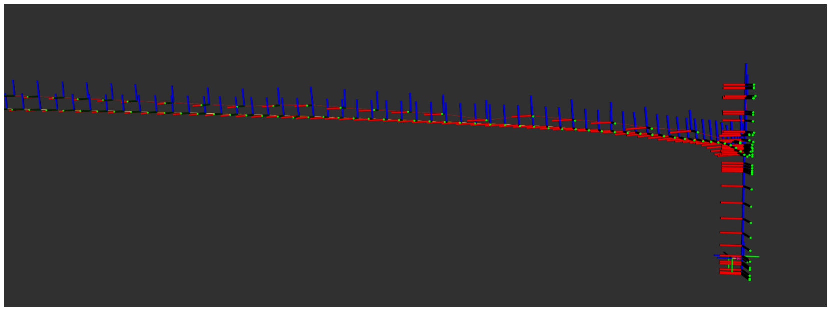

Figure 7.1 – LCS-SLAM fused odometry keyframes (red) vs ground truth (orange).

7.2 Fused Mapping

Using the fused odometry estimate, the map points from the LiDAR sensor can be registered with the most optimal robot localization estimate. There is no added computational cost in the mapping module other than re-alignment of map points with fused odometry estimates.

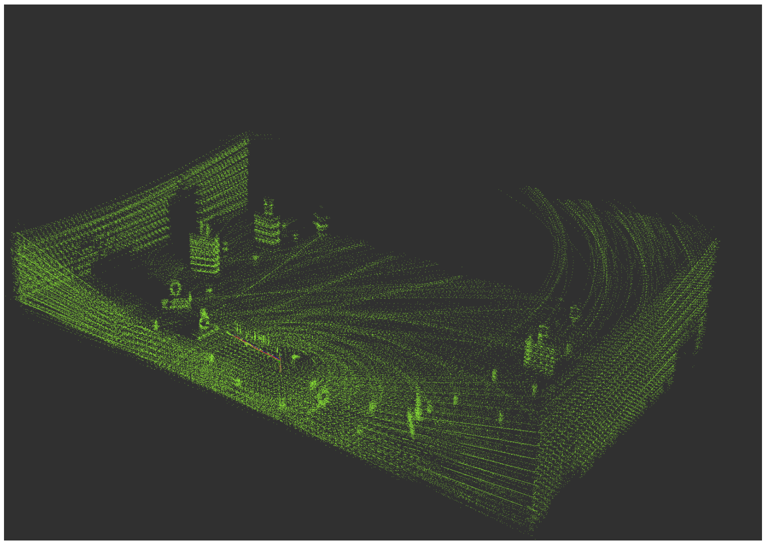

Figure 7.2 – LCS-SLAM fused mapping keyframes.

7.3 Fused Place Recognition

Given that the monocular pipeline has correctly found a “loop closure” from its place recognition module, the monocular camera keyframe identification number and its registered LiDAR and fused identification numbers can be used to recover the fused odometry keyframes in which the loop closure occurs. The relative odometry between these two scans is assumed to be the calculated LiDAR loop closure odometry since it is the most accurate method available in LCS-SLAM.

7.4 Fused Optimization

For fused optimization, pose graph optimization is used to perform global optimization. Pose graph optimization techniques used in the fused pipeline are identical to those used in the LiDAR pipeline which is discussed in detail in Chapter 6. Fig. 7.3 and Fig. 7.4 show the fused odometry estimate before and after optimization, respectively.

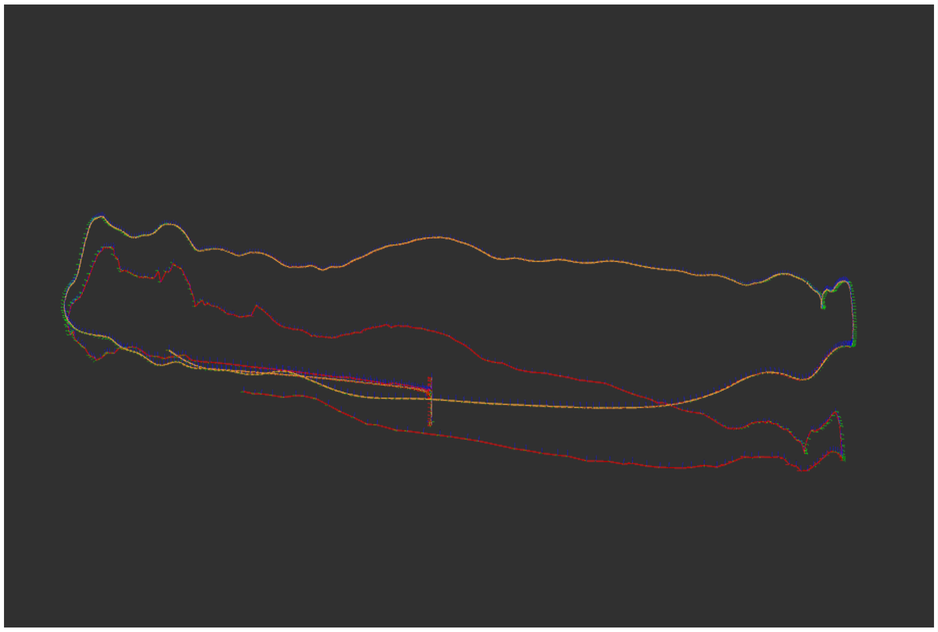

Figure 7.3 – LCS-SLAM fused (red) and truth (orange) before pose graph optimization.

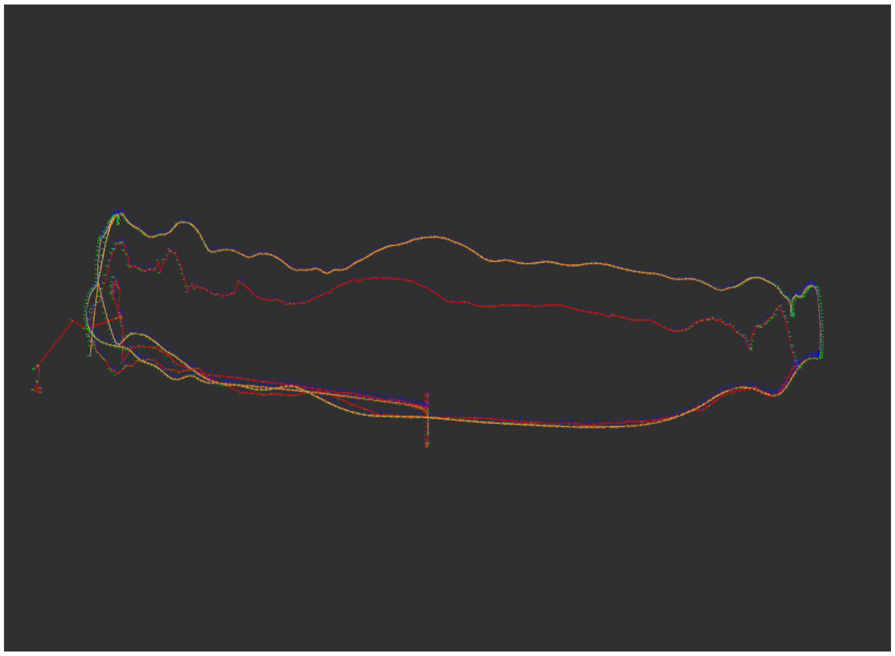

Figure 7.4 – LCS-SLAM fused (red) and truth (orange) after pose graph optimization.

Chapter 8 – LCS-SLAM Benchmark Test

Benchmark testing of the LCS-SLAM used the core software discussed in the chapters above in addition to computer system clocks calibrated for nanosecond accuracy. There were three tests conducted for each dataset: *Track-Easy* and *Track-Hard* which generated six total data points for benchmarking the performance of LCS-SLAM for use in a racing quadcopter traveling at approximately 10 miles per hour. The machine used to test the system contained 64GB of DDR4 RAM, an AMD Ryzen 3950x with 32 CPU cores, and an RTX 3090 GPU on a 64-bit x86 machine running Ubuntu 20.04. Each test had ten seconds at the beginning to allow LCS-SLAM to start-up before the dataset would begin supplying data into the system. When the dataset ended, the generated log files for trajectories, execution times, and maps from LCS-SLAM were then post-processed to evaluate for the metrics described below. The selected metrics aim to measure accuracy, repeatability, and real-time performance of LCS-SLAM.

8.1 Absolute Trajectory Error (ATE)

The absolute trajectory error is calculated by comparing the ground-truth trajectories from the simulator and performing Root Mean Square (RMS) analysis between the estimated robot trajectory for each of the various path outputs in LCS-SLAM: LiDAR, Monocular, and Fused which is good for evaluating global consistency of the trajectory. The Absolute Trajectory Error (ATE) for can be calculated for each synchronized time sequence *i* using Eq. (8.1), where *E* is the absolute trajectory error in Special Euclidean 3D space (SE3), *Q* is the ground-truth trajectory in SE3 space, *S* is the rigid-body transformation that aligns the ground truth and estimated trajectory frame, and *P* is the estimated trajectory in SE3 space. Since both the estimated trajectory and ground-truth trajectory are taken at difference time sequences, an association step is used in which the nearest timestamps are matched between the two datasets and are considered "time-synchronized". This results in a slight variation in the error estimate as there is inherent error associated with the data association step.

$$E_i = Q_i^{-1} S P_i \tag{8.1}$$

The root-mean square error of the error matrices in Eq. (8.1) can be solved using Eq. (8.2) where *n* is the number of time synchronized samples.

$$ATE_{RMSE} = \left(\frac{1}{n}\sum_{i=1}^{n} \| translation(E_i) \|^2\right)^{1/2} \tag{8.2}$$

Eq. (8.2) can be normalized against the total length traveled by the robot to allow for datasets of different lengths to be compared against each other. This normalization process yields the distance normalized root-mean square ATE and is given in Eq (8.3).

$$ATE_{NRMSE} = \frac{ATE_{RMSE}}{\sum_{i=1}^{n} \| translation(Q_i) \|} \tag{8.3}$$

The results of this analysis are summarized in Tables 8.1-8.2. Figs. 8.1-8.24 contain various visualizations in both isometric and overhead views of the optimized absolute trajectory obtained from the mono, LiDAR, and fused odometry pipelines compared to the ground-truth. The *Track-Easy* and *Track-Hard* datasets were used to test LCS-SLAM and contain three tests for each dataset. Figs. 8.1-8.12 show all the tests between the two datasets, while Figs. 8.13-8.24 shows only one of the three tests from each dataset.

Table 8.1 – LCS-SLAM RMSE ATE results.

Scenario	**Monocular Camera RMSE ATE (m)**	**LiDAR RMSE ATE (m)**	**Fused UKF RMSE ATE (m)**
Track-Easy Test 1	14.555	3.280	4.251
Track-Easy Test 2	13.714	3.329	3.537
Track-Easy Test 3	21.915	3.680	4.112
Track-Easy Average	***16.728***	***3.429***	***3.967***
Track-Hard Test 1	19.093	6.068	9.139
Track-Hard Test 2	14.124	11.369	12.553
Track-Hard Test 3	18.818	8.765	9.039
Track-Hard Average	***17.345***	***8.734***	***10.244***

Table 8.2 – LCS-SLAM NRMSE ATE results.

Scenario	**Monocular Camera NRMSE ATE (%)**	**LiDAR NRMSE ATE (%)**	**Fused UKF NRMSE ATE (%)**
Track-Easy Test 1	4.506	1.016	1.316
Track-Easy Test 2	4.246	1.031	1.095
Track-Easy Test 3	6.785	1.139	1.273
Track-Easy Average	***5.179***	***1.062***	***1.228***
Track-Hard Test 1	7.074	2.157	3.249
Track-Hard Test 2	5.020	4.041	4.462
Track-Hard Test 3	6.689	3.115	3.213
Track-Hard Average	***6.261***	***3.104***	***3.641***

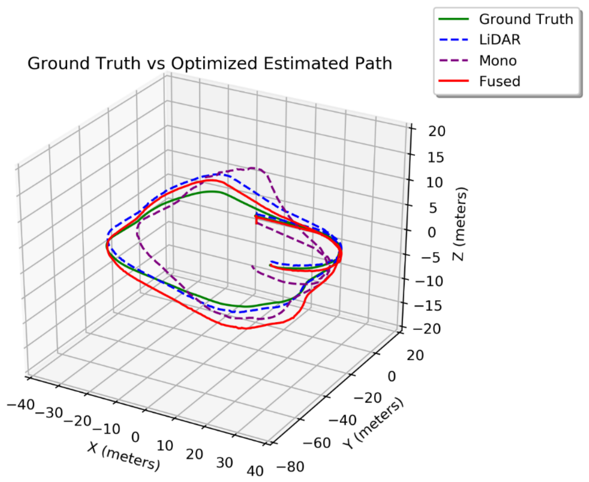

Figure 8.1 – Isometric view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Easy* Test 1.

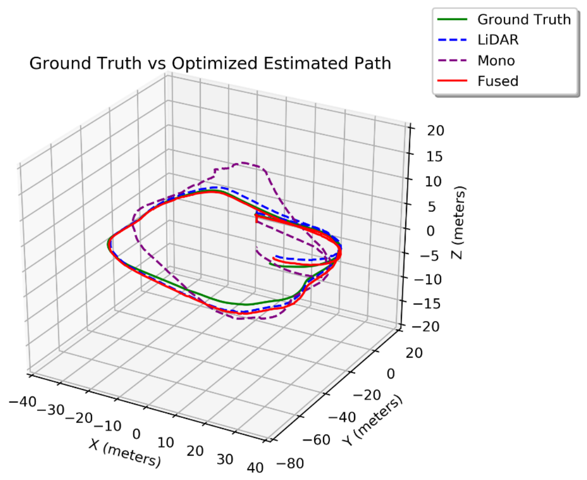

Figure 8.2 – Isometric view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Easy* Test 2.

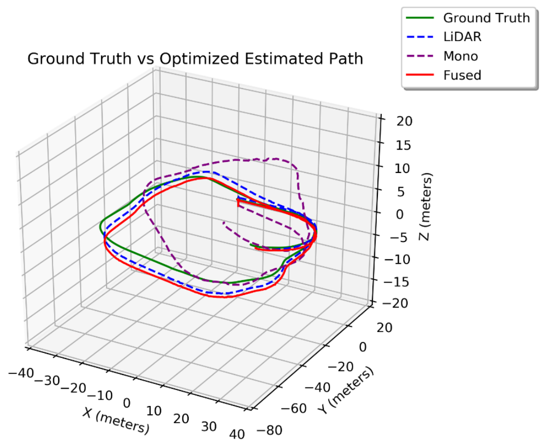

Figure 8.3 – Isometric view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Easy* Test 3.

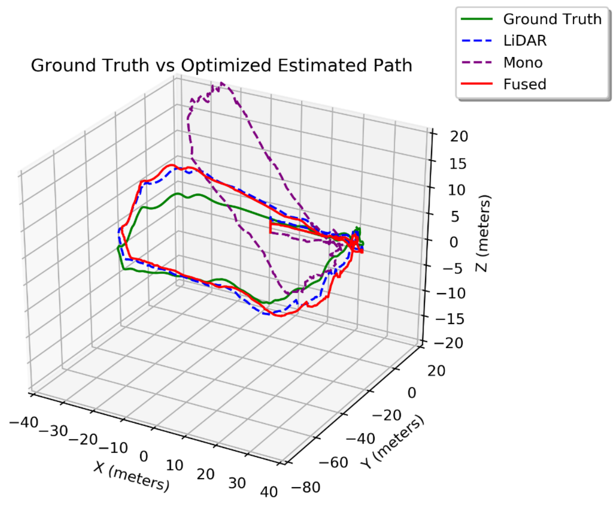

Figure 8.4 – Isometric view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Hard* Test 1.

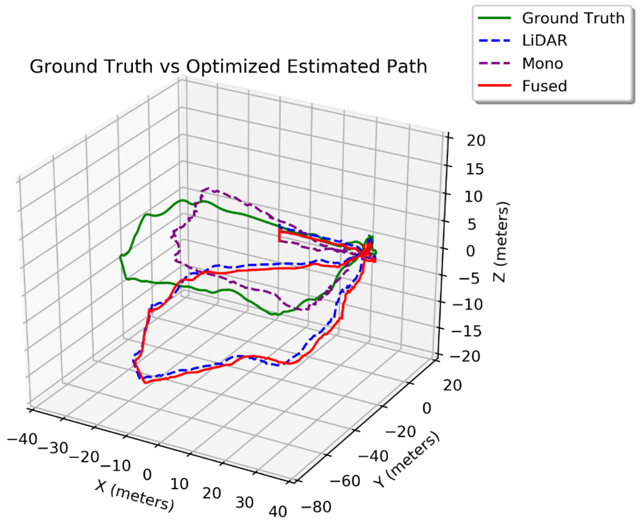

Figure 8.5 – Isometric view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Hard* Test 2.

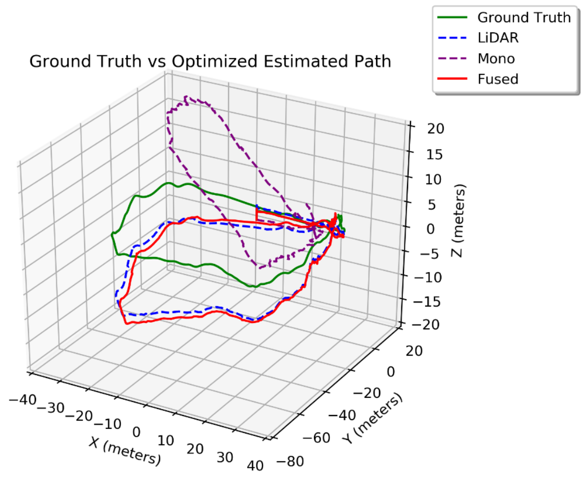

Figure 8.6 – Isometric view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Hard* Test 3.

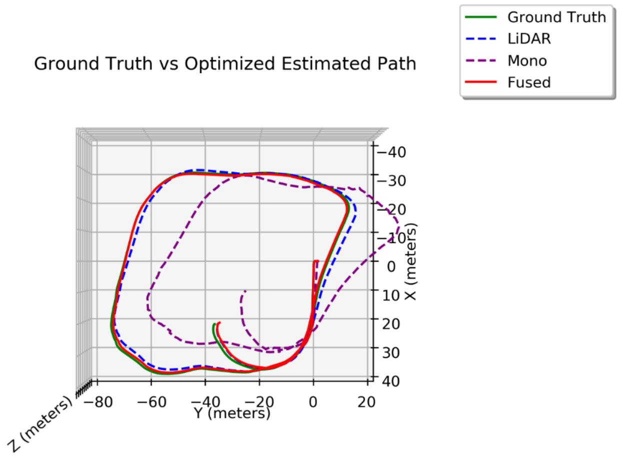

Figure 8.7 – Top view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Easy* Test 1.

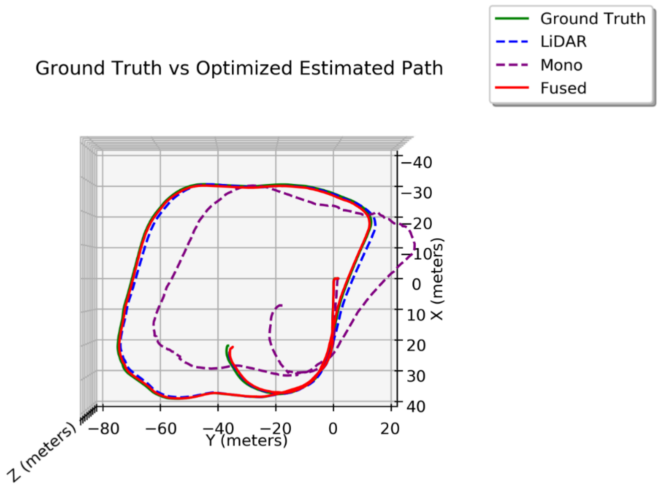

Figure 8.8 – Top view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Easy* Test 2.

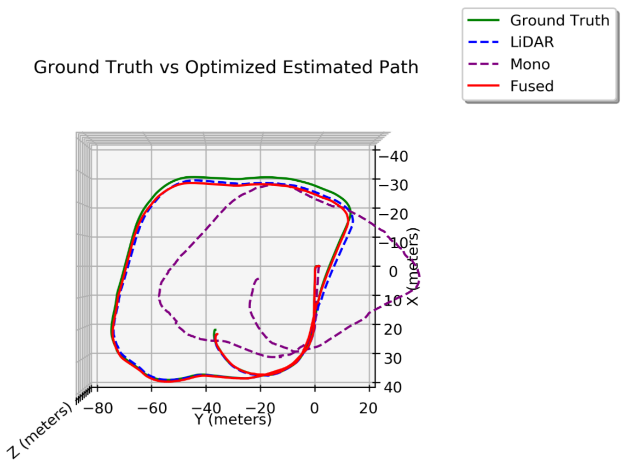

Figure 8.9 – Top view of LCS-SLAM final estimated trajectory and ground-truth on *Track-Easy* Test 3.

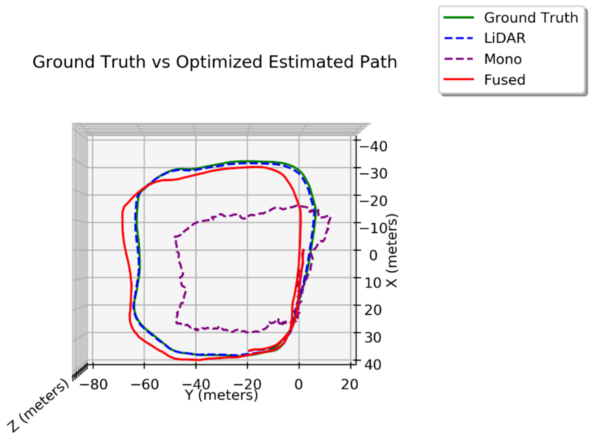

Figure 8.10 – Top view of LCS-SLAM final estimated trajectory and ground-truth on Track-Hard Test 1.

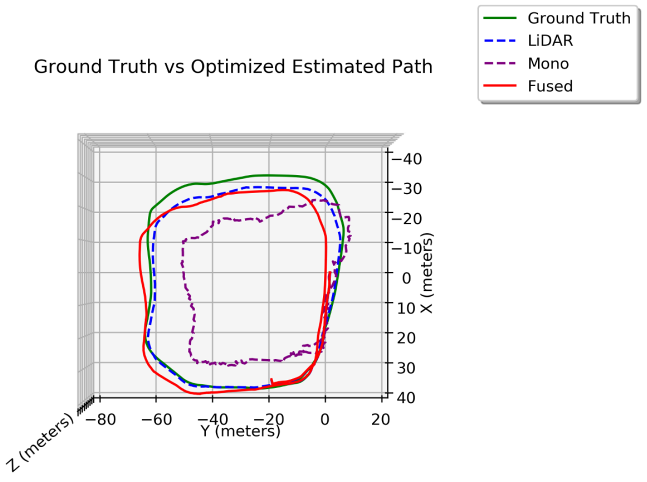

Figure 8.11 – Top view of LCS-SLAM final estimated trajectory and ground-truth on Track-Hard Test 2.

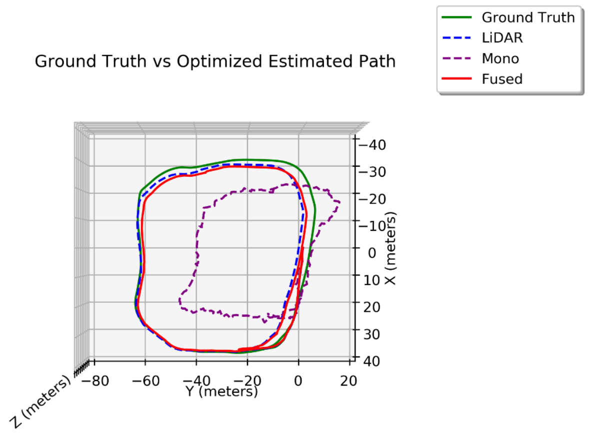

Figure 8.12 – Top view of LCS-SLAM final estimated trajectory and ground-truth on Track-Hard Test 3.

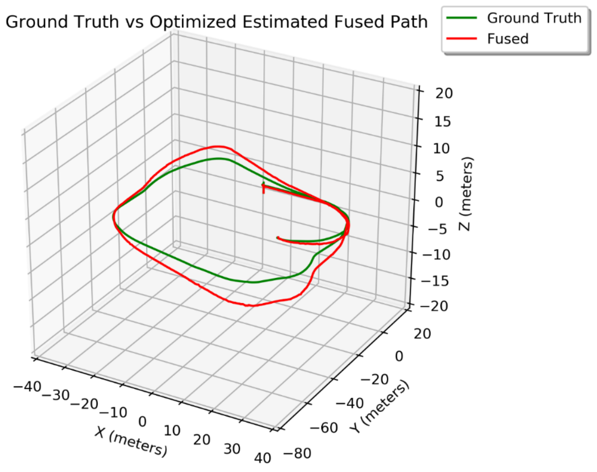

Figure 8.13 – Isometric view of LCS-SLAM fused estimated trajectory and ground-truth on Track-Easy Test 1.

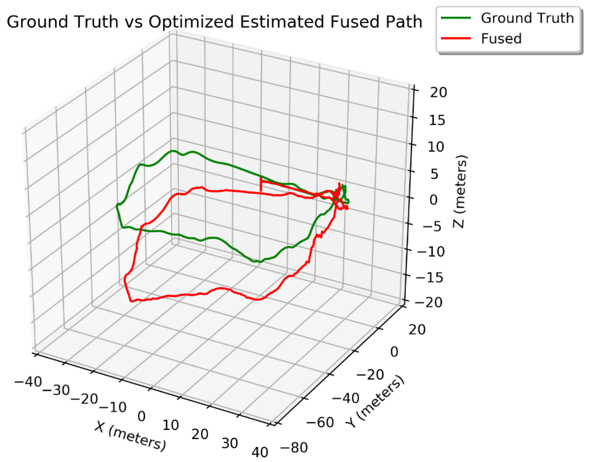

Figure 8.14 – Isometric view of LCS-SLAM fused estimated trajectory and ground-truth on Track-Hard Test 1.

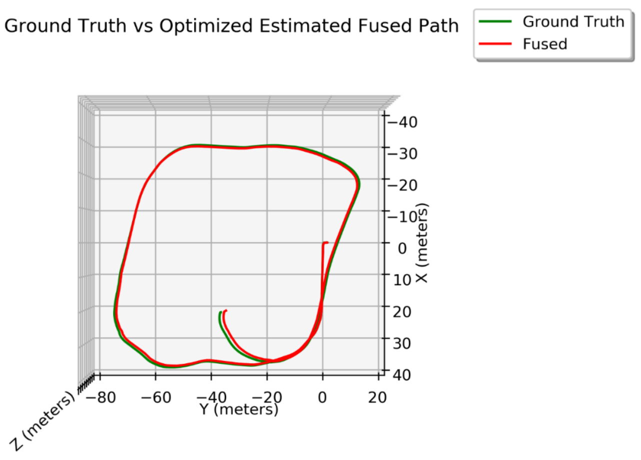

Figure 8.15 – Top view of LCS-SLAM fused estimated trajectory and ground-truth on Track-Easy Test 1.

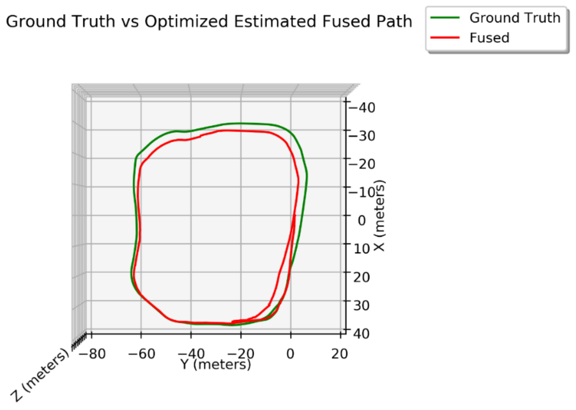

Figure 8.16 – Top view of LCS-SLAM fused estimated trajectory and ground-truth on Track-Hard Test 1.

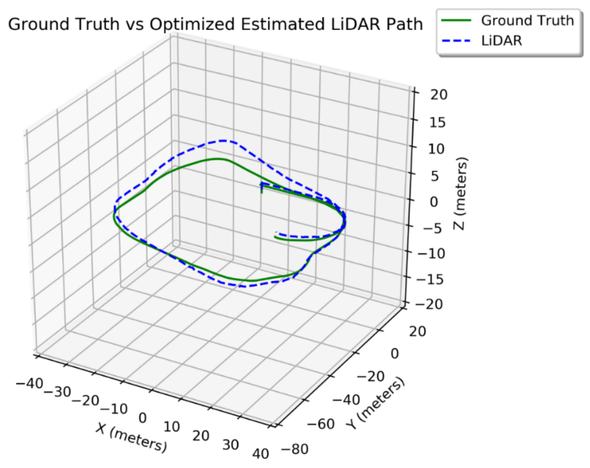

Figure 8.17 – Isometric view of LCS-SLAM LiDAR estimated trajectory and ground-truth on Track-Easy Test 1.

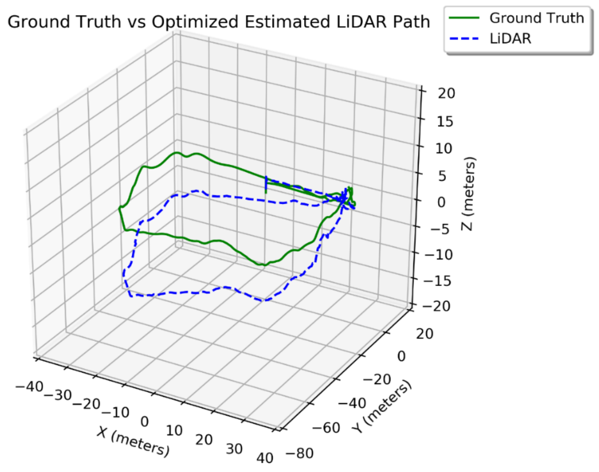

Figure 8.18 – Isometric view of LCS-SLAM LiDAR estimated trajectory and ground-truth on Track-Hard Test 1.

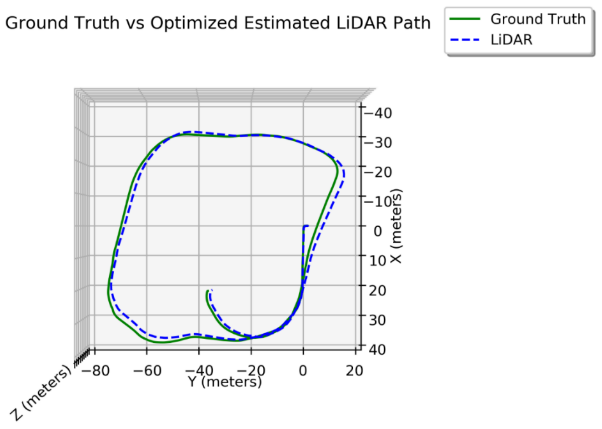

Figure 8.19 – Top view of LCS-SLAM LiDAR estimated trajectory and ground-truth on Track-Easy Test 1.

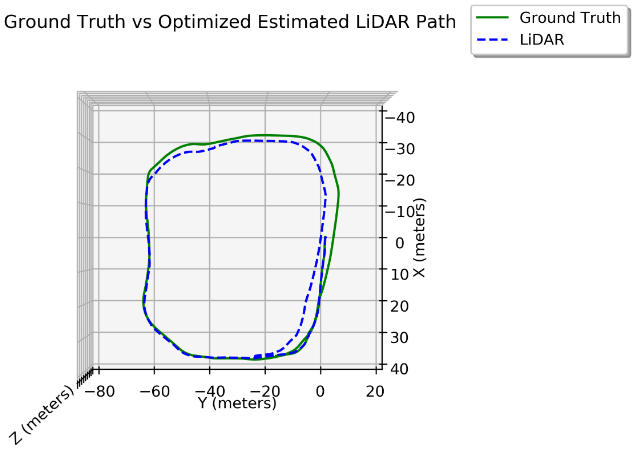

Figure 8.20 – Top view of LCS-SLAM LiDAR estimated trajectory and ground-truth on Track-Hard Test 1.

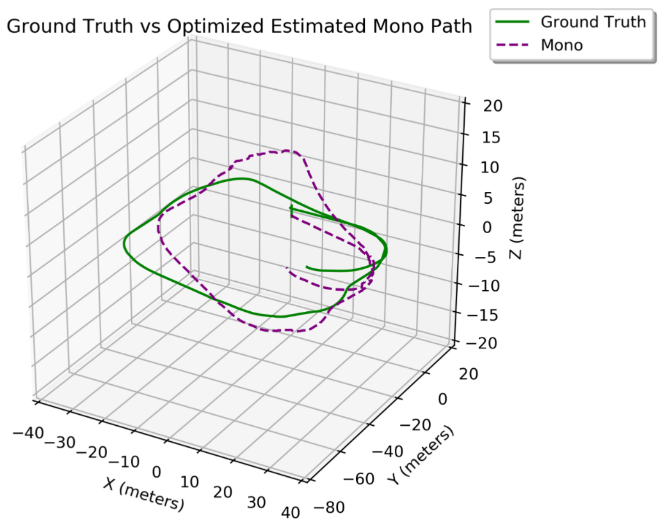

Figure 8.21 – Isometric view of LCS-SLAM mono estimated trajectory and ground-truth on Track-Easy Test 1.

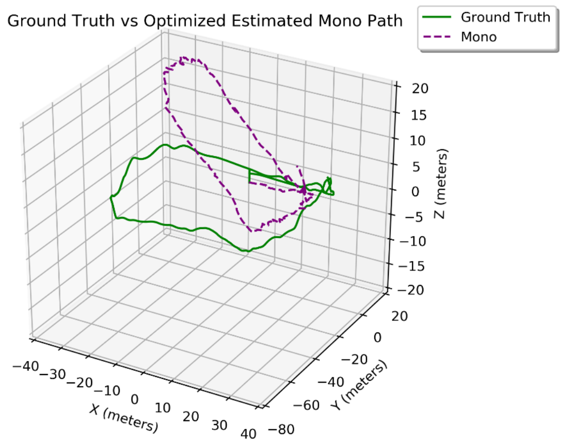

Figure 8.22 – Isometric view of LCS-SLAM mono estimated trajectory and ground-truth on Track-Hard Test 1.

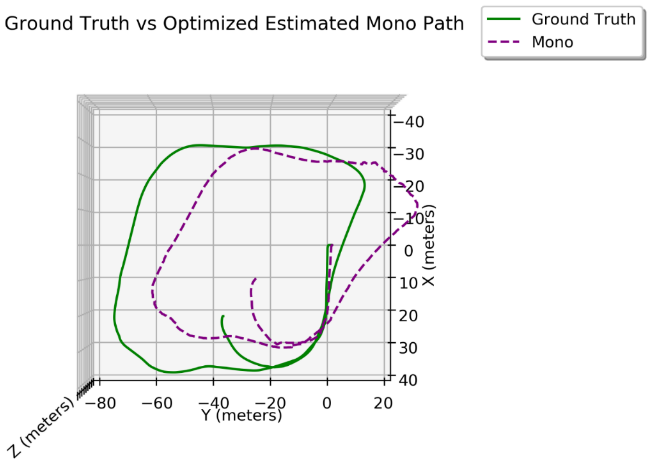

Figure 8.23 – Top view of LCS-SLAM mono estimated trajectory and ground-truth on Track-Easy Test 1.

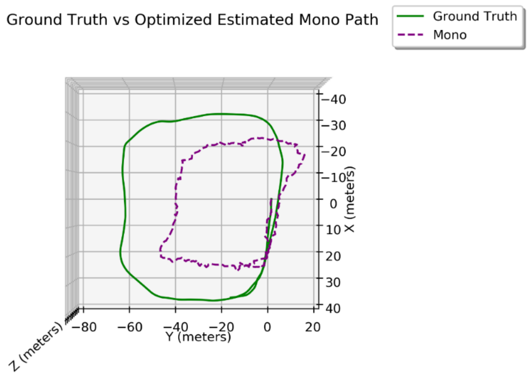

Figure 8.24 – Top view of LCS-SLAM mono estimated trajectory and ground-truth on Track-Hard Test 1.

8.2 Relative Pose Error (RPE)

The relative pose error is calculated by comparing the ground-truth poses from the simulator and performing root mean square (RMS) analysis between each relative pose estimate of the various path outputs in LCS-SLAM: LiDAR, Monocular, and Fused which is useful in evaluating the drift of a trajectory. The relative pose error (RPE) is typically broken into two components: translation and rotation. First the relative pose error for each time interval Δ must be calculated for each sequence of *i*. The relative pose error can be calculated using Eq. (8.4) where *F* is the relative pose error, *Q* is the ground-truth pose, and *P* is the estimated pose.

$$F_i = (Q_i^{-1} Q_{i+\Delta})^{-1}(P_i^{-1} P_{i+\Delta}) \tag{8.4}$$

Using Eq. (8.4) the root-mean square error for both the rotation and translation components of the relative pose error can be calculated using Eq. (8.5) and Eq. (8.6), respectively, where *n* is the number of time synchronized samples.

$$RPE_{trans}^{i,\Delta} = (\frac{1}{n}\sum_{i=1}^{n} \| translation(F_i) \|^2)^{1/2} \tag{8.5}$$

$$RPE_{rot}^{i,\Delta} = \frac{1}{n-1}\sum_{i=1}^{n-1} \arccos\left(\frac{trace(rotation(F_i))-1}{2}\right) \tag{8.6}$$

Eq. (8.6) can be normalized over the total distance traveled to obtain a distance invariant error metric as seen in Eq. (8.7).

$$RPE_{normalized\ rot}^{i,\Delta} = \frac{RPE_{rot}^{i,\Delta}}{\sum_{i=1}^{n} \| translation(Q_i) \|} \tag{8.7}$$

The RPE measured in testing of LCS-SLAM using Eq. (8.5) and Eq. (8.7) are summarized in Table 8.3.

Table 8.3 – LCS-SLAM RPE results.

Scenario	**Monocular Camera RMSE RPE Translation (m)**	**LiDAR RMSE RPE Translation (m)**	**Fused RMSE RPE Translation (m)**	**Monocular NRMSE RPE Rotation (deg/m)**	**LiDAR NRMSE RPE Rotation (deg/m)**	**Fused NRMSE RPE Rotation (deg/m)**
Track-Easy Test 1	1.053	1.009	0.615	0.011	0.011	0.006
Track-Easy Test 2	1.053	1.006	0.593	0.012	0.012	0.005
Track-Easy Test 3	1.089	1.035	0.609	0.012	0.012	0.005
Track-Easy Average	***1.065***	***1.017***	***0.606***	***0.012***	***0.012***	***0.005***
Track-Hard Test 1	1.244	1.131	0.891	0.018	0.018	0.012
Track-Hard Test 2	1.231	1.154	0.909	0.019	0.018	0.012
Track-Hard Test 3	1.229	1.165	0.891	0.019	0.018	0.013
Track-Hard Average	***1.234***	***1.150***	***0.897***	***0.019***	***0.018***	***0.012***

8.3 Tracking, Mapping, and Optimization Execution Times

The mean tracking execution times are calculated for each sensor in LCS-SLAM: LiDAR, Monocular, and Fused. The results of this analysis are summarized in Table 8.4.

Table 8.4 – LCS-SLAM tracking execution time results.

Scenario	**Monocular Odometry Mean Execution Time (ms)**	**Monocular Odometry Standard Deviation Execution Time (ms)**	**LiDAR Odometry Execution Time (ms)**	**LiDAR Odometry Standard Deviation Execution Time (ms)**	**Fused Odometry Mean Execution Time (ms)**	**Fused Odometry Standard Deviation Execution Time (ms)**
Track-Easy Test 1	48.566	11.764	4.544	9.462	0.037	0.021
Track-Easy Test 2	49.799	11.378	4.425	9.136	0.040	0.020
Track-Easy Test 3	49.679	12.043	4.389	8.844	0.039	0.020
Track-Easy Average	***49.448***	***11.728***	***4.453***	***9.147***	***0.039***	***0.020***
Track-Hard Test 1	63.371	12.818	6.044	13.481	0.042	0.019
Track-Hard Test 2	62.867	11.311	6.567	15.164	0.042	0.018
Track-Hard Test 3	62.738	11.848	6.683	16.237	0.043	0.019

Scenario	Monocular Odometry Mean Execution Time (ms)	Monocular Odometry Standard Deviation Execution Time (ms)	LiDAR Odometry Execution Time (ms)	LiDAR Odometry Standard Deviation Execution Time (ms)	Fused Odometry Mean Execution Time (ms)	Fused Odometry Standard Deviation Execution Time (ms)
Track-Hard Average	***62.992***	***11.992***	***6.431***	***14.960***	***0.042***	***0.019***

The mean mapping execution times are calculated for each sensor in LCS-SLAM: LiDAR, Monocular, and Fused. The results of this analysis are summarized in Table 8.5.

Table 8.5 – LCS-SLAM mapping execution time results.

Scenario	Monocular Mapping Mean Execution Time (ms)	Monocular Mapping Standard Deviation Execution Time (ms)	LiDAR Mapping Execution Time (ms)	LiDAR Mapping Standard Deviation Execution Time (ms)	Fused Mapping Mean Execution Time (ms)	Fused Mapping Standard Deviation Execution Time (ms)
Track-Easy Test 1	46.534	32.497	0.182	0.108	0.276	0.261
Track-Easy Test 2	47.277	34.266	0.188	0.124	0.324	0.321
Track-Easy Test 3	49.952	36.671	0.189	0.131	0.311	0.256
Track-Easy Average	***47.921***	***31.478***	***0.186***	***0.121***	***0.304***	***0.279***
Track-Hard Test 1	35.038	26.086	0.182	0.094	0.171	0.143

Scenario	**Monocular Mapping Mean Execution Time (ms)**	**Monocular Mapping Standard Deviation Execution Time (ms)**	**LiDAR Mapping Execution Time (ms)**	**LiDAR Mapping Standard Deviation Execution Time (ms)**	**Fused Mapping Mean Execution Time (ms)**	**Fused Mapping Standard Deviation Execution Time (ms)**
Track-Hard Test 2	38.532	28.364	0.148	0.092	0.168	0.141
Track-Hard Test 3	35.235	27.045	0.150	0.095	0.176	0.144
Track-Hard Average	***36.268***	***27.165***	***0.160***	***0.094***	***0.172***	***0.428***

The mean optimization execution times are calculated for each sensor in LCS-SLAM: LiDAR, Monocular, and Fused. The results of this analysis are summarized in Table 8.6.

Table 8.6 – LCS-SLAM optimization execution time results.

Scenario	**Monocular Opt. Mean Execution Time (ms)**	**Monocular Opt. Standard Deviation Execution Time (ms)**	**LiDAR Opt. Execution Time (ms)**	**LiDAR Opt. Standard Deviation Execution Time (ms)**	**Fused Opt. Mean Execution Time (ms)**	**Fused Opt. Standard Deviation Execution Time (ms)**
Track-Easy Test 1	46.700	N/A	53.905	11.218	119.099	48.143
Track-Easy Test 2	31.907	N/A	48.575	N/A	80.656	N/A
Track-Easy Test 3	41.973	N/A	50.829	N/A	83.017	N/A

Scenario	**Monocular Opt. Mean Execution Time (ms)**	**Monocular Opt. Standard Deviation Execution Time (ms)**	**LiDAR Opt. Execution Time (ms)**	**LiDAR Opt. Standard Deviation Execution Time (ms)**	**Fused Opt. Mean Execution Time (ms)**	**Fused Opt. Standard Deviation Execution Time (ms)**
Track-Easy Average	***40.193***	***N/A***	***51.103***	***N/A***	***94.257***	***N/A***
Track-Hard Test 1	34.624	N/A	30.377	N/A	46.663	N/A
Track-Hard Test 2	36.917	N/A	41.321	N/A	47.873	N/A
Track-Hard Test 3	41.515	N/A	36.349	N/A	51.334	N/A
Track-Hard Average	***37.685***	***N/A***	***36.016***	***N/A***	***48.623***	***N/A***

The mean total execution times are calculated for the entirety of the LCS-SLAM algorithm. The results of this analysis are summarized in Table 8.7.

Table 8.7 – LCS-SLAM total execution time results.

Scenario	**Total Mean Execution Time (ms)**	**Total Standard Deviation Execution Time (ms)**	**Total Max Execution Time (ms)**	**Total Min Execution Time (ms)**	**Total Median Execution Time (ms)**
Track-Easy Test 1	54.098	57.439	729.147	5.184	23.693
Track-Easy Test 2	54.364	56.701	707.873	5.155	24.394
Track-Easy Test 3	55.307	60.048	748.409	5.199	23.860
Track-Easy Average	***54.589***	***58.063***	***728.476***	***5.179***	***23.982***
Track-Hard Test 1	76.099	62.045	560.005	6.876	81.191
Track-Hard Test 2	78.392	63.614	543.987	6.639	83.486
Track-Hard Test 3	75.724	61.844	544.923	6.843	81.830
Track-Hard Average	***76.738***	***62.501***	***549.638***	***6.786***	***82.169***

8.4 Mapping Accuracy

The mapping accuracy in LCS-SLAM is calculated for only the best fused pipeline results for each dataset. Test 1 from both the *Track-Easy* and *Track-Hard* datasets were used for evaluation of mapping accuracy. The 3D models used in the datasets were exported from Unity3D and sampled into point clouds. These point clouds are used as ground-truth maps when compared to the estimated point cloud maps generated from LCS-SLAM. The point clouds from LCS-SLAM are filtered to remove outliers and then aligned to the ground-truth using the Iterative Closest Point (ICP) algorithm. Once aligned, the mean distance between closest matched points in both the ground-truth and estimated maps are calculated. To perform this analysis the point cloud alignment software known as *CloudCompare* is used. The results of this analysis are summarized in Table 8.8 and Figs. 8.25-8.29.

Table 8.8 – LCS-SLAM mapping accuracy results.

Dataset	**Mean Distance (m)**	**Standard Deviation (m)**
Track-Easy, Test 1	0.892	1.006
Track-Hard, Test 1	1.385	1.422

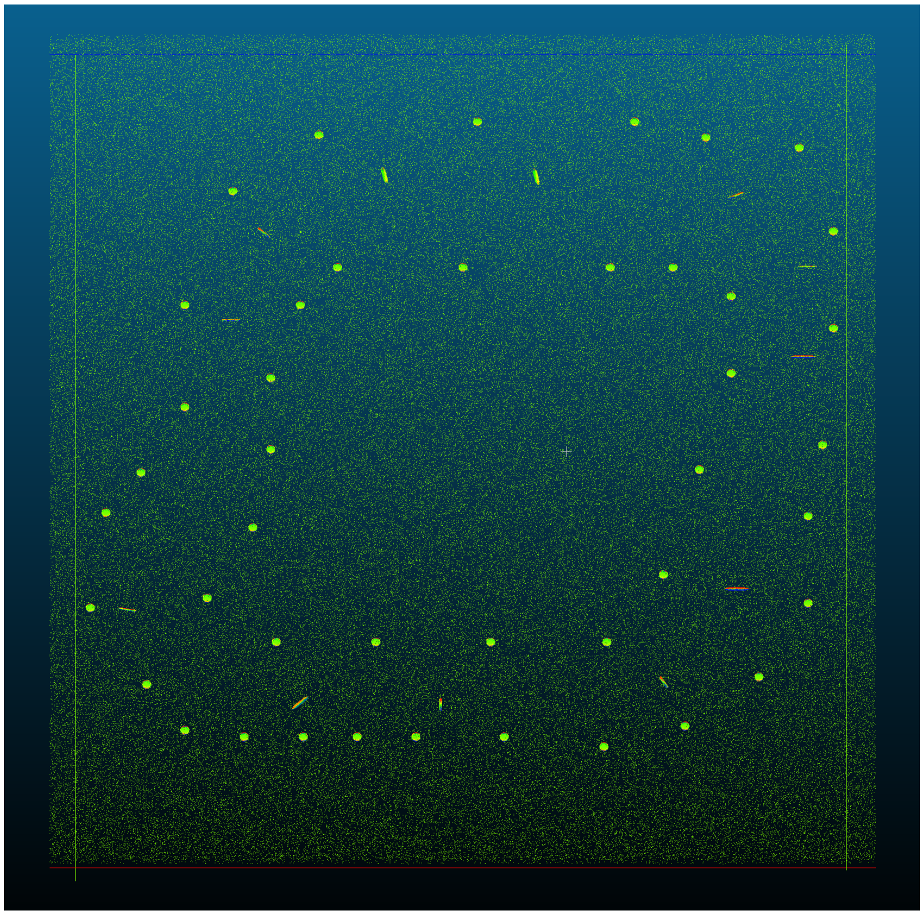

Figure 8.25 – Top view of ground-truth map of Track-Easy Unity3D course.

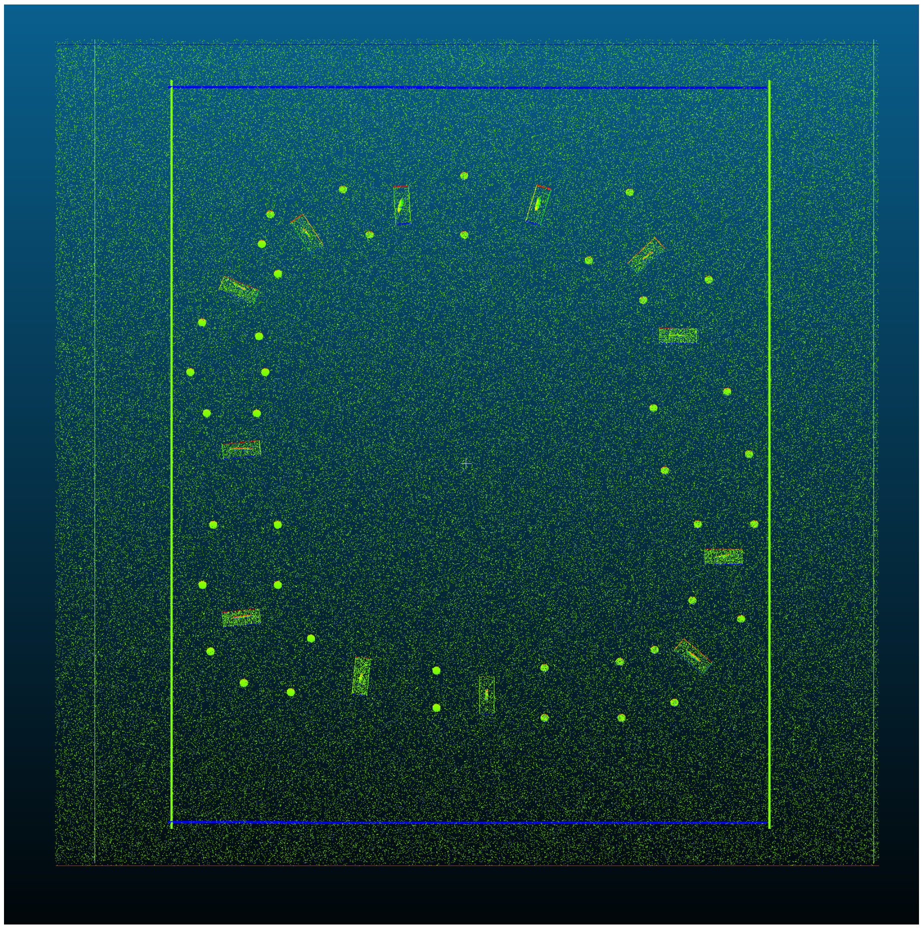
Figure 8.26 – Top view of ground-truth map of Track-Hard Unity3D course.

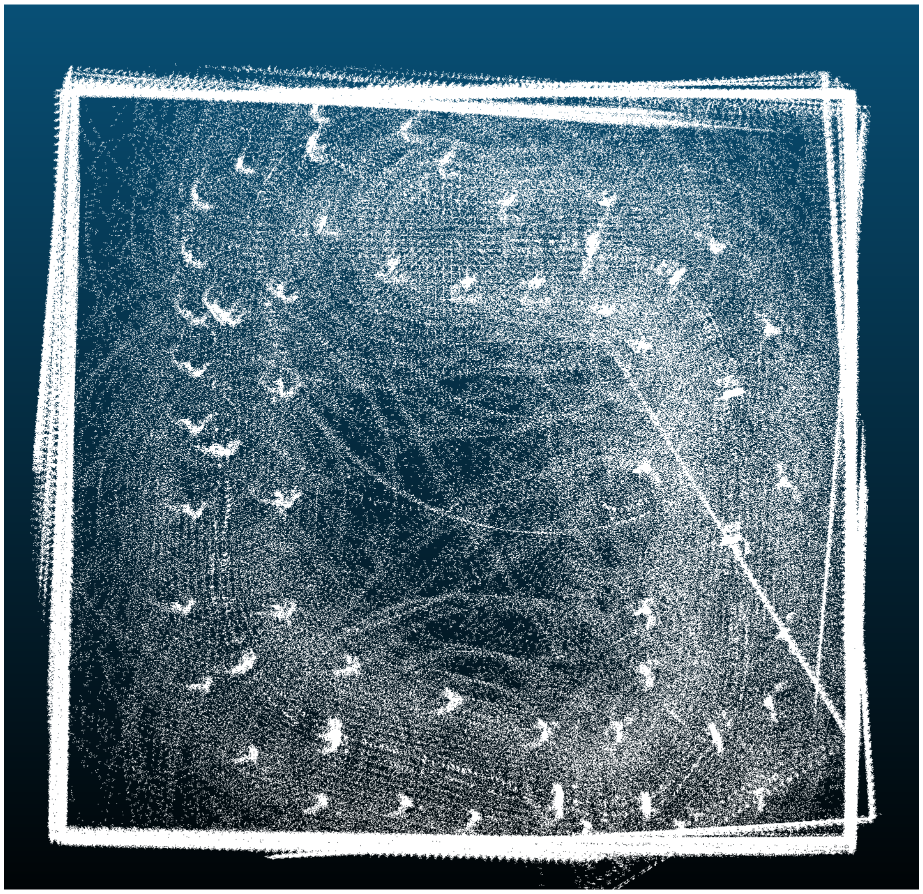

Figure 8.27 – Top view of estimated fused map of Track-Easy dataset, test 1.

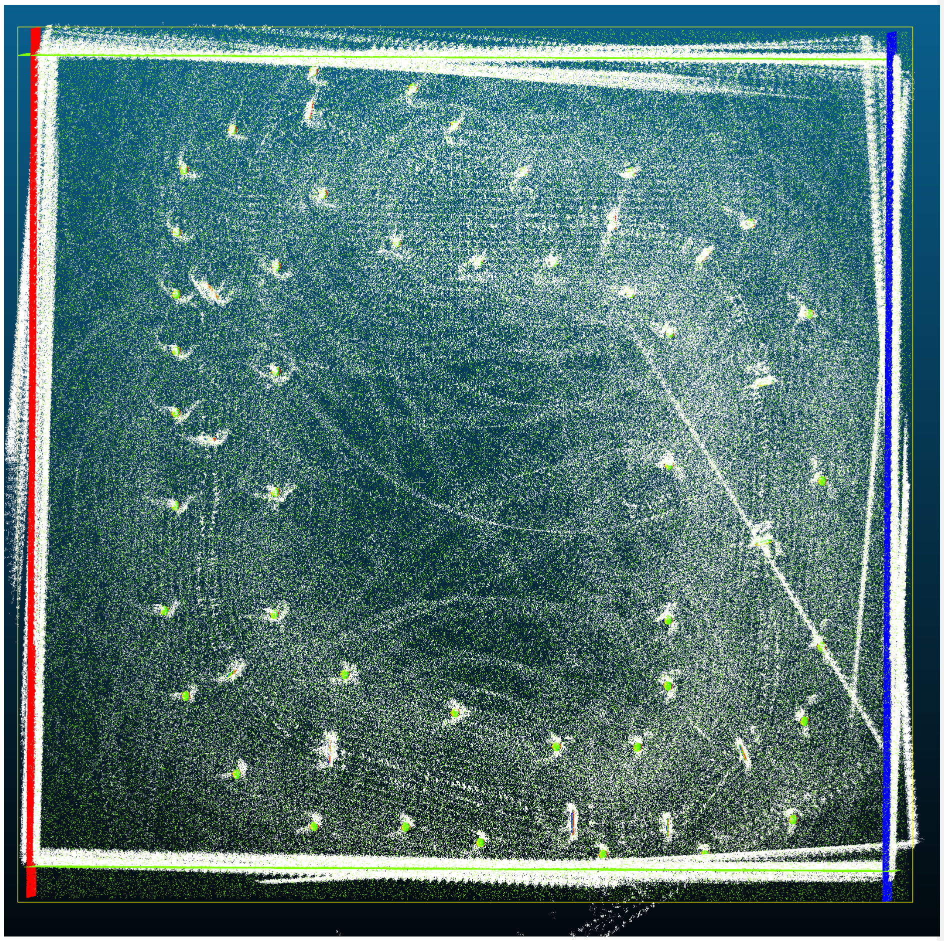

Figure 8.28 – Top view of estimated fused map (white) registered with ground-truth map (colored) of Track-Easy dataset, test 1.

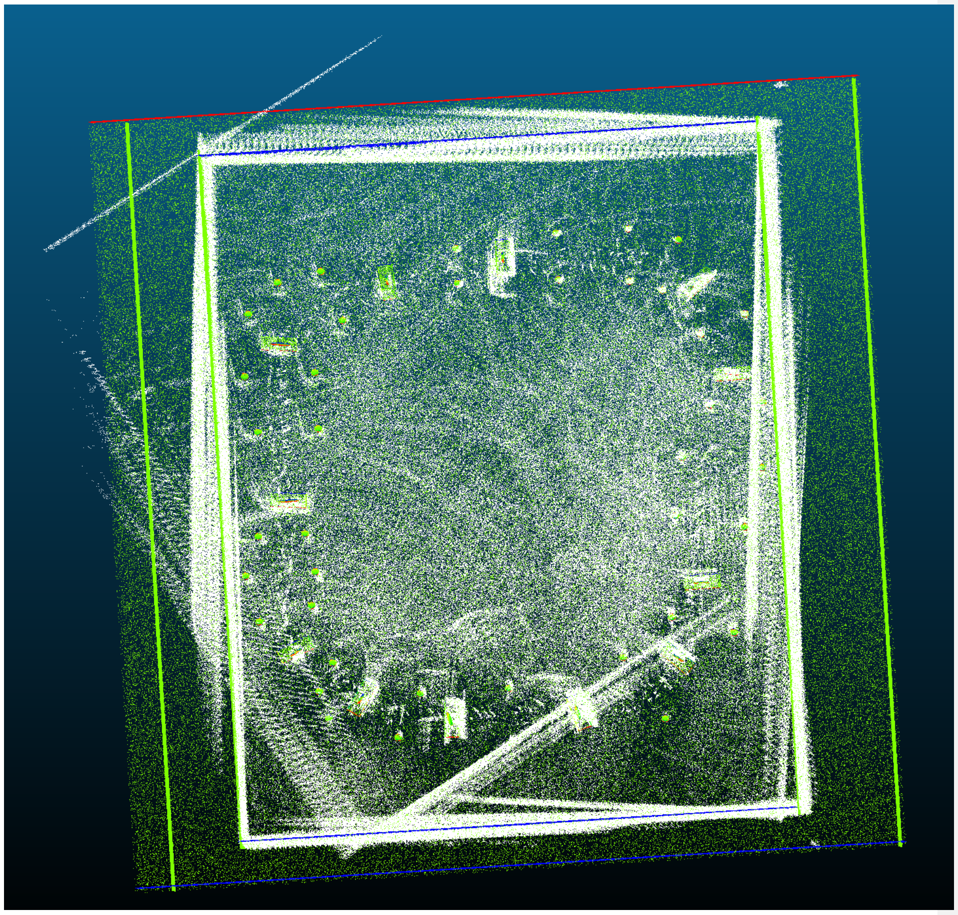

Figure 8.29 – Top view of estimated fused map (white) registered with ground-truth map (colored) of Track-Hard dataset, test 1.

Chapter 9 – Analysis of LCS-SLAM Performance

9.1 Analysis of LCS-SLAM Performance

The overall performance of LCS-SLAM on the two datasets collected for analysis is defined by its tracking accuracy, mapping accuracy, repeatability, and execution times.

LCS-SLAM is globally consistent in a simple scene up to 1.062% trajectory error, and in a difficult scene 3.104% trajectory error. Mean relative translation error is as low as 0.606 meters in a simple scene and as high as 0.897 meters for a difficult scene for the fused sensor pipeline, which is the most accurate for relative pose estimation. Distance normalized relative rotation error is as low as 0.005 degrees per meter or as high 0.012 degrees per meter in the worst case for the most accurate sensor pipeline.

LCS-SLAM excels in its mean tracking execution time performance in the LiDAR and fused pipelines with best-case tracking times as low as 4.453 milliseconds and 0.039 milliseconds, respectively. Monocular tracking is an order of magnitude slower than LiDAR tracking at 49.448 milliseconds. In the worst case LCS-SLAM performs LiDAR and fused tracking at 6.431 milliseconds and 0.042 milliseconds, respectively. Monocular tracking in the worst-case tracks at 62.992 milliseconds. The large difference between the tracking times can be attributed to the complexity of the monocular tracking pipeline and the use of a GPU-accelerated LiDAR odometry algorithm. The fusion tracking pipeline is the quickest as it uses IMU data and any new data from the monocular or LiDAR pipelines to perform sensor fusion with a highly optimized implementation of the Unscented Kalman Filter.

In the mapping execution time performance, LCS-SLAM once again sees very fast mean response times in the LiDAR and fused pipelines. These two pipelines ingest existing sensor data and must only concatenate current transforms and maps with existing map data, resulting in worst-case performance of less than 1 millisecond. Monocular mapping is more intensive as it must perform a triangulation step between multiple geometry views. Best case performance is 36.628 milliseconds, while worst case performance is 47.912 milliseconds.

Mean optimization execution time performance for all sensor pipelines in LCS-SLAM average under 100 milliseconds on average. Monocular optimizations take up to 40.193 milliseconds in the worst case and as little as 37.685 milliseconds in the best case. LiDAR optimizations take up to 51.103 milliseconds in the worst case, and as little as 36.016 milliseconds in the best case. Fused optimization performance can take up to 94.257 milliseconds in the worst case, and 48.623 milliseconds in the best case. The increasing optimization time can be attributed to the number of pose graph nodes, weights on pose graph edges, and map points associated with the optimization process for each pipeline.

Mean total execution time for LCS-SLAM in the best case is 58.063 milliseconds and in the worst case is 62.501 milliseconds. Maximum execution time of LCS-SLAM can be upwards of 728.476 milliseconds, while minimum execution times can be a little as 5.184 milliseconds. The median execution time in the best case is 23.982 milliseconds, and 82.169 milliseconds in the worst case.

Mapping performance of LCS-SLAM is accurate to the meter level, with mapping registration mean distance error of 0.892 meters and 1.385 meters over distances of a simple course of 322.99 meters and a difficult course of 281.347 meters respectively. Incorrect pose estimates can have a catastrophic impact on the map, thus reducing drift and performing global optimization are critical to globally consistent maps as done in LCS-SLAM.

Overall, the performance of LCS-SLAM can be described as real-time capable in 3D racing environments of both simple and difficult scenes with average mapping and tracking capabilities, but prone to drift without continuous loop closures. LCS-SLAM is capable of execution at approximately 15 Hz, with the potential for LiDAR maps to be generated up to 100 Hz, given that sensor input rates of that frequency can be achieved.

9.2 Comparison Versus ORB-SLAM

ORB-SLAM3 is a state-of-the-art real-time SLAM framework for monocular, stereo, and RGB-D cameras that is popular in the open-source computer vision community. Like LCS-SLAM, ORB-SLAM uses the ORB feature detector to generate keypoints in an image which are used for localization and mapping. To improve the robustness of the framework's localization and mapping accuracy, keyframes are selected from images with sufficient motion. In addition, ORB-SLAM is capable of both loop detection and loop correction by optimizing an essential graph using similarity transformations and Levenberg-Marquardt optimization to complete a least squares minimization problem. Like LCS-SLAM, loop detection in ORB-SLAM uses an optimized visual vocabulary database called Bag-of-Words.

Tracking in ORB-SLAM requires an initialization step in which essential matrix estimation or homography is used and only points which are visible in the local map are tracked in a co-visibility graph. Keyframes are inserted using a technique called *"Survival of the Fittest"* and if tracking is lost, new image frames are compared to the Bag-of-Words database for the most likely matches and corresponding odometry estimates of those points. Fig. 9.1 shows the ORB-SLAM framework architecture and feature matching and mapping capabilities.

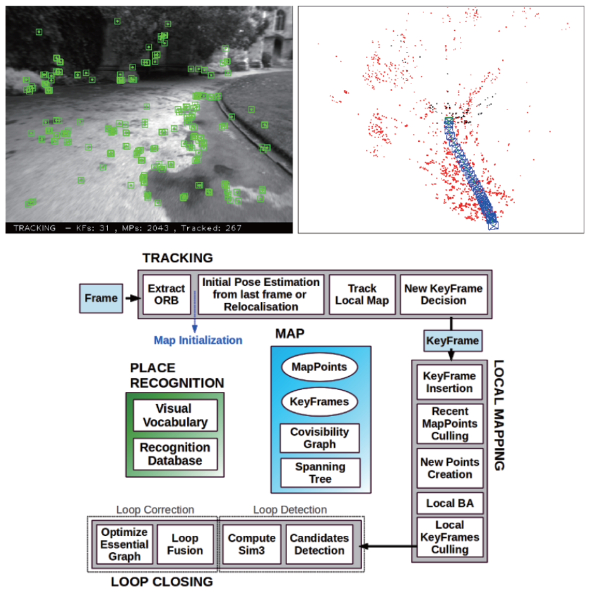

Figure 9.1 – ORB-SLAM framework architecture [33].

9.3 Comparison Versus LOAM

LiDAR Odometry and Mapping (LOAM) is a state-of-the-art odometry and mapping framework which uses LiDAR scans. Like LCS-SLAM, LOAM uses point cloud registration to estimate the odometry at a 10Hz update rate. The odometry data is used to generate 3D maps of the environment given the raw sensor scans from a LiDAR. The mapping process takes the scanned 3D points and transforms them into the odometry frame at a frequency of 1 Hz. The architecture for LOAM can be seen in Fig. 9.2, while the output odometry and mapping procedures from LOAM are shown in Fig. 9.3.

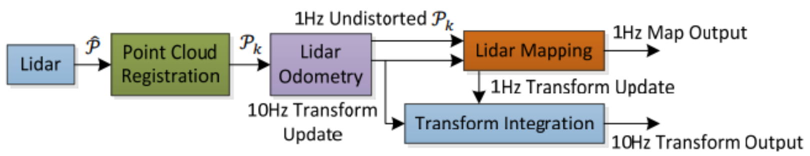

Figure 9.2 – LOAM high level system architecture [8].

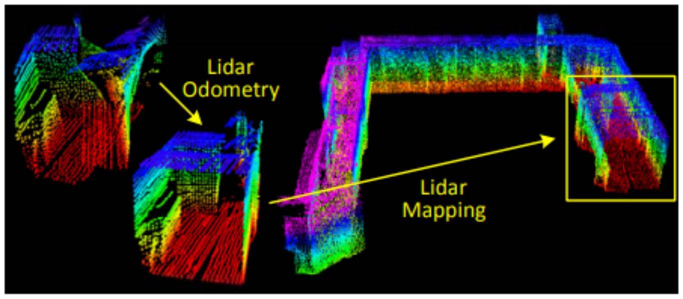

Figure 9.3 – LOAM odometry and mapping output [8].

9.4 Comparison Versus HDL-Graph-SLAM

HDL-Graph-SLAM integrates systems that use LiDAR and Inertial Navigation System (INS) sensors which together can create a system that sustains long-term and wide-area measurements. HDL-Graph-SLAM is based on graph SLAM, which is the same approach used in the optimization process of LCS-SLAM. Scan matching is used between consecutive frames using the Normal Distribution Transform algorithm to estimate odometry and construct the pose graph. INS data that provides angular velocity is used to compensate for rotational drift of the LiDAR scan matching algorithm. The angular velocity is integrated with a UKF, much like what is used in LCS-SLAM. The high-level system architecture of HDL-Graph-SLAM is shown in Fig. 6.4, while HDL-Graph-SLAM in action is shown in Fig. 9.5.

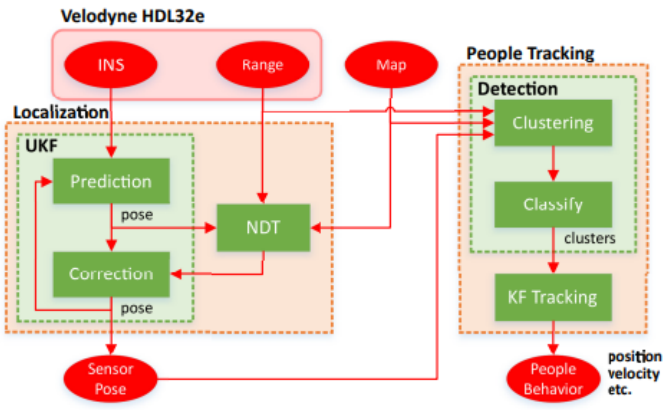

Figure 9.4 – HDL-Graph-SLAM system architecture [34].

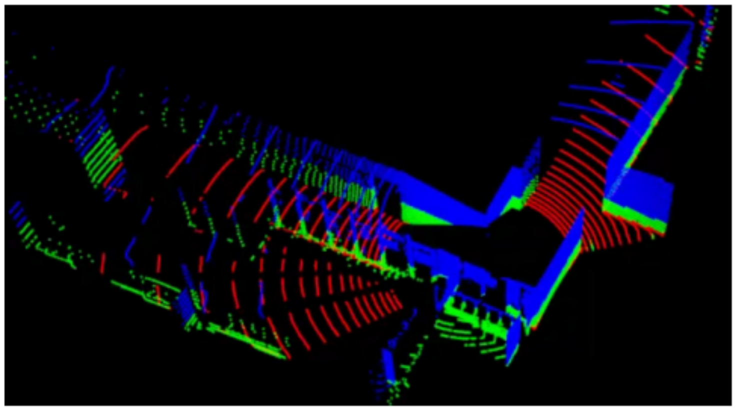

Figure 9.5 – HDL-Graph-SLAM scan matching capability [34].

Chapter 10 – Conclusion

LCS-SLAM has shown that sensor fusion between 3D LiDAR, cameras, and IMU sensors have the potential to enable robust and resilient SLAM in real-time using traditional SLAM and computer-vision techniques. In addition, the power of GPU-accelerated algorithms can also be used for real-time applications given the increasing computation capabilities of modern technology. The LCS-SLAM algorithm was developed and tested in a flight simulation environment using a racing quadcopter with proven results that showed an optimized global map of the environment and robots estimated trajectory generated from the LiDAR, monocular camera, and fused pipelines which leverage the sensors equipped on the virtual quadcopter. A quantitative and qualitative analysis of the localization and mapping performance of LCS-SLAM has shown that LCS-SLAM has the potential to become an accurate solution for localization and mapping for autonomous racing quadcopters. Future work could include performance optimizations to speed up the monocular camera pipeline, improvements to the localization techniques used for visual odometry, the addition of a stereo camera or RGB-D pipeline, and the addition of local optimization processes such as sliding window bundle adjustment.

References

[1] National Aeronautics and Space Administration. "Advanced Air Mobility National Campaign Overview," *National Aeronautics and Space Administration*, 2020. [https://www.nasa.gov/aeroresearch/aam/description/. Accessed 30 October 2020.]

[2] Herox.com. "Alphapilot – Lockheed Martin AI Drone Racing Innovation Challenge," *Herox*, 2020. [https://www.herox.com/alphapilot/overview/. Accessed 30 October 2020.]

[3] Microsoft Research. "Game Of Drones - Competition At Neurips 2019," *Microsoft Research*, 2020. [https://www.microsoft.com/en-us/research/academic-program/game-of-drones-competition-at-neurips-2019/. Accessed 30 October 2020.]

[4] National Aeronautics and Space Administration. "Autonomous Systems". *National Aeronautics and Space Administration*, 2020. [https://www.nasa.gov/feature/autonomous-systems/. Accessed 30 October 2020.]

[5] Zhang, J., and Singh, S., "Loam: Lidar Odometry and mapping in real-time," *Robotics: Science and Systems X*, 2014.

[6] Fang, Z., Zhao, S., Wen, S., and Zhang, Yu, "A Real-Time 3D Perception and Reconstruction System Based on a 2D Laser Scanner," *Journal of Sensors*, Vol. 2018, published online on 16 May 2018. https://doi.org/10.1155/2018/2937694

[7] Zhen, and W., Scherer, S., "A Unified 3D Mapping Framework using a 3D or 2D LiDAR," *Springer Proceedings in Advanced Robotics*, Vol. 11, published online on 23 January 2020. https://doi.org/10.1007/978-3-030-33950-0_60

[8] Zhang. J., and Singh, S., "LOAM: Lidar Odometry and Mapping in Real-time," *Robotics: Science and Systems 2014*, published online on 2014. https://doi.org/10.15607/RSS.2014.X.007

[9] Xie, Y., Hao, C., Zhang, W., Li, S., and Qian, H., "Lidar-IMU Fusion for 2.5D Mapping," *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO),* Kuala Lumpur, Malaysia, 2018, pp. 832-837, https://doi.org/10.1109/ROBIO.2018.8665103

[10] Li, J., "Fusion of Lidar 3D Points Cloud with 2D Digital Camera Image," Master Thesis, Department of Electric and Computer Engineering, Oakland University, Rochester, Michigan, 2015.

[11] Allden, T., Chemander, M., Davar, S., Jansson, J., and Laurenious, R., "Virtual Generation of Lidar Data for Autonomous Vehicles," Bachelor Thesis, Department of Computer Science and Engineering, University of Gothenburg, Sweden, 2017.

[12] Wen, K., Qiam. C., Tang, J., Liu, H., Ye, W., and Fan, X., "2D LiDAR SLAM Back-End Optimization with Control Network Constraint for Mobile Mapping," *Journal of Sensors*, Vol. 2018, published online on 29 October 2018. https://doi.org/10.3390/s18113668

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

[14] Campos, C., Elvira, R., Rodriguez, J. J., M. Montiel, J. M., and D. Tardos, J., "Orb-slam3: an accurate open-source library for visual, visual-inertial, and Multimap Slam," *IEEE Transactions on Robotics*, 2021, pp. 1-17.

[15] Mito, Y., Masakazu, M., and Fujii, K., "An Object Detection and Extraction Method Using Stereo Camera," *2006 World Automation Congress*, Budapest, 2006, pp. 1-6. https://doi.org/10.1109/WAC.2006.375746

[16] Hu, Z., Qi, B., Yuan, L., Zhang, Y., and Chen., Z, "Mobile robot V-SLAM based on improved closed-loop detection algorithm," *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC),* Chongqing, China, 2019, pp. 1150-1154, https:/doi.org/ 10.1109/ITAIC.2019.8785611

[17] X. Liang, H. Chen, Y. Li and Y. Liu, "Visual laser-SLAM in large-scale indoor environments," *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO),* Qingdao, 2016, pp. 19-24, https://doi.org/10.1109/ROBIO.2016.7866271

[18] Li, R., Liu, J., Zhang, L., and Hang, Y., "LIDAR/MEMS IMU Integrated Navigation (SLAM) Method for a Small UAV in Indoor Environments," *2014 DGON Inertial Sensors and Systems (ISS),* Karlsruhe, 2014, pp. 1-15, https://doi.org/10.1109/InertialSensors.2014.7049479

[19] Zhang, J., and Singh, S., "Visual-lidar odometry and mapping: Low-drift, robust, and fast," 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015.

[20] Deilamsalehy, H., and Havens, T. C., "Sensor Fused Three-dimensions Localization Using IMU, Camera, and LiDAR," *2016 IEEE SENSORS*, Orlando, FL, 2016, pp. 1-3, https://doi.org/10.1109/ICSENS.2016.7808523

[21] López, E., Barea, R., Gómez, A., Saltos, Á., Bergasa, L. M., Molinos, E. J., and Nemra, A., "Indoor Slam for Micro aerial vehicles using visual and laser sensor fusion," *Advances in Intelligent Systems and Computing*, 2015, pp. 531–542.

[22] Jiang, G., Lei, Y., Jin, S., Tian, C., Ma, X., and Ou, Y., "A Simultaneous Localization and Mapping (SLAM) Framework for 2.5D Map Building Based on Low-Cost LiDAR and Vision Fusion," *Applied Sciences*. 2019. https://doi.org/10.3390/app9102105

[23] Smolyanskiy, N., Kamenev, A., Smith, J., and Birchfield, S., "Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Vancouver, BC, 2017, pp. 4241-4247, https://doi.org/10.1109/IROS.2017.8206285

[24] Harper, W., Rispoli, S., and Mendez, B., "Conceptual Design and Modeling of an Autonomous Racing Quadcopter, "*AIAA* (not yet published).

[25] The Drone Racing League. "Airr," *The Drone Racing League*, 2020. [https://thedroneracingleague.com/airr/. Accessed 30 October 2020.]

[26] Velodyne Lidar. 2019. VLP-16 User Manual. [online] Available at: <https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf> [Accessed 18 September 2021].

[27] Arducam. "Arducam 1080p low light WDR USB camera module for computer, 2MP 1/2.8' CMOS IMX291 100-degree wide angle mini UVC webcam board with microphone," *Arducam,* 2020. [https://www.arducam.com/product/arducam-1080p-low-light-wdr-usb-camera-module-for-computer-2mp-1-2-8-cmos-imx291-100-degree-wide-angle-mini-uvc-spy-webcam-board-with-microphone-3-3ft-1m-cable-for-windows-linux-mac-os/. Accessed 30 October 2020.]

[28] Adafruit Industries. "Adafruit 9-DOF absolute orientation IMU Fusion Breakout - BNO055," *Adafruit Industries*, 2020. [https://www.adafruit.com/product/2472. Accessed 30 October 2020.]

[29] Gao, X., and Zhang, T., "Introduction to Visual SLAM From Theory to Practice". *Publishing House of Electronics Industry*, 2017.

[30] Mur-Artal, R., Montiel, J. M., and Tardos, J. D., "Orb-Slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, 2015, pp. 1147–1163.

[31] Nister, D. "An efficient solution to the five-point relative pose problem", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, 2004, pp. 756-770

[32] Zhang, Y., Jin, R., and Zhou, Z.-H., "Understanding bag-of-words model: A statistical framework," *International Journal of Machine Learning and Cybernetics,* vol. 1, 2010, pp. 43–52.

[33] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G., "Orb: An efficient alternative to SIFT or surf," *2011 International Conference on Computer Vision*, 2011.

[34] Koide, K., Miura, J., and Menegatti, E., "A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement," *International Journal of Advanced Robotic Systems*, vol. 16, 2019, p. 172988141984153.