UAV SLAM and Interior Modeling with 3D LiDAR in GNSS-Denied Environments

Auburn REU on SMART UAVs 2019
Technical Report # CSSE 19-01

Green, Conor
Loyola Marymount University
cgreen18@lion.lmu.edu

Stevens, Brenden
University of California, Los Angeles
brenden728@ucla.edu

Biaz, Saad
Auburn University
biazsaa@auburn.edu

Chapman, Richard
Auburn University
chapmro@auburn.edu

July 17, 2019

Abstract

Small, quadrotor helicopter, or quadcopter, unmanned aerial vehicles (**UAV**s) have unique abilities to map environments, particularly utilizing 3D flash light detection and radar (**LiDAR**) technologies. However, the majority of applications to date use global navigation satellite systems (**GNSS**) to determine location in order to stitch together LiDAR frames, which excludes mapping in environments without readily available or reliable GNSS (e.g. inside concrete buildings or underground.) Previous projects have been able to confirm the viability of mapping with LiDAR and achieve simultaneous localization and mapping without GNSS. The project presented equipped a quadcopter UAV with a LiDAR sensor and manually navigated through an interior environment to provide navigation and point cloud data. With estimates of pose, a virtual mock-up was generated in post from the point cloud data to provide a comprehensive, 3D representation of the interior. The proposed system was able to create extensive and accurate models of the interior of a building in constrained circumstances. Comprehensive models can be created if the UAV is not actually flying but instead placed on a chair and rotated/translated. Due to the extra weight of the LiDAR and single board computer, the UAV could not fly properly and the data collected in flight was so low quality it could not generate a reasonable model.

1 Introduction

In the past, unmanned aerial vehicles (**UAV**s) have primarily been motivated by military applications, namely mapping, due to the unique advantages of relatively inexpensive platforms and maneuverability compared to manned systems when performing simple tasks. Recently, UAVs have become a common platform for data acquisition in geomatics- the collection, analysis, and interpretation of data relating to the earth's surface-and terrain mapping for other uses. Many of the UAVs utilized for surveying land employ light detection and radar (**LiDAR**) technologies, often in conjunction with conventional cameras [1]. LiDAR, invented in the 1960s, has seen a dramatic rise in civilian applications with the miniaturization of semiconductor components in the past two decades. LiDAR uses time of flight (**ToF**) principles to determine the distance from the camera to objects in the surrounding area. For years, this was a one-dimensional beam that had to be scanned over an area to gather information; however, recently, three-dimensional, flash ToF cameras capable of sending out light to a square area in the field of view (**FOV**) and determining the

distance to thousands of points simultaneously have been invented and become feasible to implement in mapping applications. These 3D flash LiDAR cameras operate similar to conventional cameras but add the extra dimension of *depth*. UAVs equipped with a LiDAR have the distinct ability to generate point clouds (sets of data points in 3D space) and have the clear applications to mapping such as gauging properties of the ocean water and atmosphere, ice sheets, and the forest canopy for topographic mapping. Attaching LiDAR sensors to a mini-UAV has been shown to be feasible and advantageous for fine scale mapping in outdoor environments [2].

These mapping systems require accurate location and attitude information in order to properly register—consistently aligning various 3D point cloud data views into a complete model—together frames. Without accurate spatial-location data, each frame of the ToF camera are unconnected/unrelated and hard to mesh into a cohesive model. Attitude and heading reference systems (**AHRS**) have been well developed in order to estimate attitude. Often utilizing gyroscopes, accelerometers, and magnetometers, AHRS can achieve high levels of accuracy over long periods of time. Satellite localization has become the prevalent method of geo-localization for position estimation since the advent of the global navigation satellite system (GNSS) in 1960. Inertial measurement units (**IMU**s) provide accurate position information relative to the body for short periods of time. Together, AHRS, GPS, and IMU sensors provide the necessary pose (location and attitude) information to properly combine various frames into a usable model. The United States implements the Global Positioning System or GPS, a more commonly recognized implementation of GNSS, but various counties and regions have their own systems. GNSS works across the globe with various redundancies and accuracy within a few meters; however, they are limited in areas without a direct line of sight to satellites, underground, or in buildings. This poses an obstacle for using GNSS within buildings or structures. Applying 3D ToF technologies to map interior environments (i.e. without GNSS) would have important applications to search and rescue. In the event of a disaster, first responders could send a small UAV into the potentially unsafe area and receive a comprehensive map of the interior, which could be explored virtually for dangers prior to introducing any human into danger. This technology would also have clear extensions in usage to the military, police, or civilian domains.

Various methods have achieved simultaneous localization and mapping (**SLAM**) in GNSS-denied environments. One novel approach is to utilize features generated from LiDAR data and how they change over time to determine location and movement [3] [4] [5]. Using feature extraction for SLAM is feasible but not necessary. The navigation data given should offer a sufficiently reliable estimation of pose for the 3D model: using a close estimate of pose, manipulations of the point cloud should hone in the pose estimates to create the 3D model. In control theory, there are well established methods to estimate internal states such as location. A complementary filter is an intuitive and powerful tool to calculate estimates through sensor fusion. Some sensors, like gyroscopes or accelerometers, are accurate over short periods of time but tend to yield greater errors over long periods due to bias [6]. In outdoor situations, this bias accumulation can be rectified using GNSS.

The prevalent solution to sensor fusion and state estimation is a Kalman filter, or an Extended Kalman Filter for non-linear systems; however, there are certain drawbacks to the prevalent method. In a closed loop system, the Kalman filter can only assure local exponential stability. The Kalman parameters are non-intuitive and notoriously difficult to tune [7]. A viable alternative is a complementary filter. The complementary filter is a simple, intuitive sensor fusion technique that utilizes the information from short-term and long-term stable sensors. In order to create a powerful sensor fusion algorithm, the UAV's dynamics must be understood and modeled. The physics of a quadcopter are well understood and can be described accurately in mathematical models that can help determine the accuracy of the sensor data and aid in location state estimation [8].

A concise summary of the problem faced will be described subsequently. The end product of this paper is an accurate and comprehensive virtual model of an indoor environment. In order to accomplish this task, LiDAR data must be collected and put into a single model using pose estimations and point cloud manipulation algorithms. The related works done about GNSS-denied pose estimation and LiDAR mapping will be reviewed. Many projects were able to achieve pose estimation without GNSS, often using odometry. Other works were able to map areas, inside and outdoors; however, many used 2D LiDAR scanners (operate

similar to sweeping sonar or radar) and no works have been done to map using a 3D flash LiDAR. To map indoor environments with a 3D flash LiDAR is novel work. The methods of achieving this goal will be outlined, first listing the technology employed then describing the algorithms necessary to turn the raw data into a comprehensive model.

2 Problem Statement

LiDAR mapping is a powerful tool that can yield a comprehensive, 3D virtual model that can be explored in detail; however, the majority of work in LiDAR mapping uses GNSS and IMU data, along with other minor sensors like magnetometers and pressure sensors, to determine location and attitude [1]. However, GNSS is unavailable indoors and as such, we propose a method to create a comprehensive 3D model of indoor environments using 3D LiDAR equipped upon a small UAV. A weighted combination of navigation data and iterative closest point (**ICP**) algorithms will provide the transformations between each frame necessary to register the LiDAR frames. The crux of the problem of indoor environments is accurately calculating the location and attitude of the drone without GNSS information. Sensor data from the IMU, magnetometer, and pressure sensors are both inaccurate and imprecise and as such cannot be solely used for location/attitude estimation. Sensor data must be digitally processed and fused. The drone will be assumed to have smooth angular and linear velocities and accelerations over time.

After achieving accurate location and attitude estimation for the entirety of the flight path, the collected point cloud data needs to be registered, or put together, frame by frame. To mesh together numerous frames of 3D data points, with a large amount of redundancy, poses a significant data processing problem. There are libraries that implement iterative closest point algorithms; however, they still require that all the frames are referenced against an absolute frame of reference. Therefore, each successive frame must be transformed to be referenced from that absolute reference based on the location and attitude estimations. Furthermore, the localization and LiDAR data need to be precisely synchronized in time for accurate frame stitching, which poses issues in implementation because the navigation data and LiDAR are gathered on separate systems, that cannot communicate directly during testing.

3 Related Works

GNSS-denied localization is a prevalent research topic considering the difficulty and various applications. It is hard to determine absolute location without an absolute reference like the global navigation satellite system. An IMU is often employed to determine change in position and dead reckoning—estimating the direction and distance traveled—to calculate position; however, this is notoriously prone to error. Many projects utilize supplemental sensors to gather an absolute reference. For the purposes of this project, pose is estimated from an IMU with a few supplemental sensors from the specific UAV used. A myriad of works discuss localization efforts without GNSS and many of the conclusions and methods will prove helpful. In GNSS-denied localization, sensor fusion and filtering is necessary and various techniques have been applied and analyzed. This project will implement a complementary filter along with various low level signal processing filters. Related works on localization through a complementary filter are the most pertinent.

Research is currently ongoing and relevant to using LiDAR in feature mapping and extraction. Various projects concerned with LiDAR mapping focus on simultaneous feature mapping and extraction, treating the problem of LiDAR mapping as both a localization and mapping (i.e. SLAM) problem. Some research has been done involving the extraction of sparse LiDAR data onboard the drone itself to aid with localization [9]. The presented system is a solution to many of these methods which require either larger amounts of processing power and/or different types of LiDAR. The 3D ToF camera employed is, at the time of this paper, still in development. The problem of localization through LiDAR is relevant since the novel LiDAR used here may provide entirely new methods for localization through feature extraction.

3.1 GNSS-Denied Pose Estimation

To determine the pose of a UAV in a GNSS-denied environment is a known problem that has been approached and solved in various fashions. There are two main approaches: using solely electromechanical sensors (e.g. IMU) and/or utilizing visual cues, often with LiDAR or conventional video cameras, and determining the difference in frames using feature extraction. The former will apply more readily to location/attitude estimation in the context of mapping considering feature extraction necessitates additional complexity while only yielding a marginal increase in accuracy over the common on-board sensors of quadcopters. However, the literature on determining pose from electromechanical sensors often use GNSS while the work determining location and attitude from visual cues often work without GNSS. Therefore, the literature on both approaches prove informative to the context of pose estimation in GNSS-denied environments. Furthermore, in any situation that seeks to estimate an unobservable state from sensor data with various sources of error, the information must be processed through smoothing and/or filtering.

Vasilijevic *et. al.* [7] sought to estimate the location of an underwater vehicle susceptible to noise, bias, and outliers through a simple sensor fusion algorithm, namely a complementary filter. A simple, intuitive, and relatively accurate model was designed and implemented to fuse various sensors with differing types of errors. In order for the sensor to work, the sensors needed to be calibrated meticulously [7]. Although the filter was implemented for an underwater vehicle, the same principles apply to UAVs with the same six degrees of freedom. Gustavsson [10] designed and implemented a Kalman and complementary filter in order to estimate the pose of a quadcopter over various test tracks in an outdoor environment. The complementary filter was a useful and reliable framework for sensor fusion of information from IMUs and supplemental sensors. However, the use of visual tags would be necessary to achieve the high accuracy in indoor areas [10].

Electromechanical inertial sensors such as six axis IMUs and magnetometers provide data on the acceleration and orientation respective to a magnetic field and can be utilized to determine pose. These sensors suffer from bias over large intervals of time and as such, the data needs to be filtered and processed to provide accurate and precision pose information. Kok *et. al.* [11] employed algorithms such as optimization-based smoothing and filtering, and Kalman and complementary filters to achieve reasonably good estimates of pose. The smoothing algorithm and Kalman filter took more computational power to provide marginally more accurate pose estimations compared to the complementary filter. Furthermore, the complementary filter was found to be accurate with little parameter tuning [11].

Feature mapping is important in location without GNSS; however, it does not provide enough accuracy to solely rely on. Li *et. al.* [4] first estimated the location of a UAV through LiDAR feature mapping and extraction: the nearby environment was mapped and the extracted features give an indication of position. A Micro-Electro-Mechanical System (MEMS) based (IMU) was used to determine the errors and fed into a Kalman filter to hone in the pose estimation. LiDAR mapping and MEMS IMU pose estimation was shown to be a viable method for small UAV navigation in indoor environments [4].

3.2 LiDAR Mapping

The RPLIDAR system developed by Slamtec extends the range finder past one dimension by spinning a 1D LiDAR at a fixed rate producing a distance estimation for all objects in a 2D slice. The RPLIDAR system mentioned has been shown to create a 3D map by mounting the RPLIDAR vertically, and spinning the entire system through a horizontal plane [12]. Effectively merging numerous slices of 2D vertical planes containing distances into a complete 3D model. This paper is relevant to newer 3D LiDAR systems that take a snapshot of the scene much like a standard camera but also return distances in a matrix described by the resolution such that each pixel contains information on the distance to that particular object with which it reflected.

Generating LiDAR point cloud maps involves gathering proper position/localization data of frames so that each individual frame can be transformed in space to an absolute reference. Localization and autonomy using ToF cameras on micro-drones for GNSS-denied indoor environments was proven suitable [1] [4] and even achieved a level of autonomy via a method of erosion [3]. By taking the normal vectors of two consecutive

feature extracted planes of an object detected by the ToF camera, rotations and translations of the drone were accurately estimated with an error less than 10%. The limitation was the quality of the image and demonstrated a need for better and possibly heavier ToF equipment; however, LiDAR feature extraction and slam was proven feasible [3]. Geo-referencing the position data with the LiDAR data has not been attempted. Instead of using autonomy, the method proposed in this paper will manually pilot UAVs. The position estimates produced in [3] would likely be sufficient for ToF data adjustment and Geo-Referencing. However, the feature extraction method described would require sufficient processing power. Instead, this research will rely heavily on filtering timestamped navigation data to produce the position estimates for geo-referencing.

Rather than completely forgoing GNSS/GPS, other methods have created a pseudo GNSS/INS by using the on-board computer to extract a sparse 2D plane from the full 3D LiDAR scan and cross-referencing the data with other sensors including the IMU and GPS when available. The pseudo GNSS/INS unit created produced time-stamped position estimates allowing the drone to provide an accurate model of the interior environment to be generated using the iterative closest point(ICP) registration technique [9]. The proposed method of in this paper parallels the established pseudo GNSS/INS system by creating an accurate position estimate with complementary filtering. The difference in LiDAR camera usages between the proposed pseudo GNSS/INS system and the system proposed here is its field of view. In this research, a smaller forward facing camera that gathers a fixed field of view 3D point cloud image that inhibited the ability to extract a 2D plane surrounding the drone, used in [9]. Moreover, this research focuses particularly on aerial vehicles without GNSS, instead of ground based vehicles, which is exactly the proposed direction of future research in the conclusion of [9].

Zhang [12] explained an algorithm for simultaneous feature extraction and point cloud mapping using a line scanning LiDAR attached to a rotating motor. The key for the accuracy in mapping was updating the navigation data and pose of the drone at a much higher frequency than the point cloud map to improve accuracy. This way, the movement of the point cloud can be adjusted for with a high level of confidence. In this paper, the intention is to also take the navigation data and the point cloud data at a near 5:1 ratio such that every frame of point cloud data would correspond to 5 frames of navigation data. Unlike the methodolgy in LOAM, this research centers around getting the ToF camera on an airborne drone, required a much lighter ToF camera. Rotation is no longer an issue both physically and mathematically since the line of 3D LiDAR cameras take pictures much a like a camera.

4 Experiment

In order to achieve a 3D model, two technologies, the flash LiDAR, or alternatively 3D ToF, and UAV were manually navigated through the interior of a building to produce two data files. The 3D ToF records for a given period of time and saves the LiDAR frames to a proprietary file type, .rrf, which can be parsed in MATLAB. The drone transmits the navigation/sensor data (**navdata**) live and the data was handled through a Python script and saved in a colon delimited text file. Considering the computational effort required to properly register point clouds, the processing was done in post on desktop/laptop computers. The LiDAR data was transformed into a point cloud while the navdata was put through a complementary filter (to be described subsequently) to provide relatively accurate estimations of pose. Together, the point cloud and pose estimations were utilized to determine the transformations between each frame. These transformations provide an initial estimate as to how to move each frame relative to the first frame. Finally, ICP algorithms register the frames into a clean, 3D model of the environment traversed. A flowchart of the total process is given in 1.

4.1 Equipment

In order to properly map the interior of a building with LiDAR, a small, stable quadcopter is mounted with a powerful single board computer (**SBC**) and a 3D ToF flash camera. The quadcopter is controlled from a stationary laptop while the SBC controls the LiDAR camera. Many SBCs would work considering they

must simply be lightweight and compatible with the flash LiDAR. The flash LiDAR and SBC were taped on the drone as shown in 2.

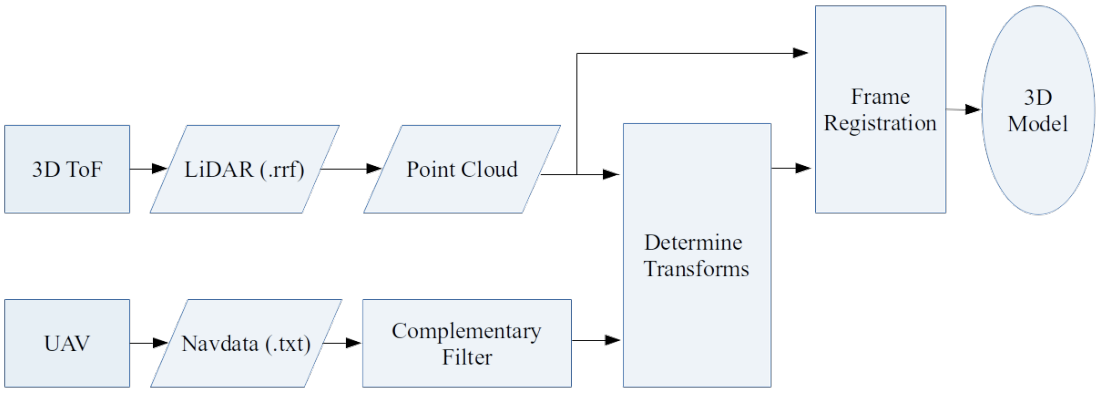


Figure 1: UAV Flash LiDAR Mapping Flowchart

Figure 2: Complete UAV Configuration

4.1.1 Quadcopter UAV

In this particular setting, a Parrot AR Drone 2.0 quadcopter, or simply AR Drone, was used for its stability, ability to carry up to 200 grams, and existing software interfaces. The AR Drone was also chosen for the various on board sensors: three-axis accelerometer, three-axis gyroscope, three-axis magnetometer, downward facing ultrasonic sensor, and barometer. Of the previously mentioned sensors, all but the latter were utilized. For the purposes of this project, the navdata was queried at 200Hz (the fastest of the options) because the majority of the sensors are only available in this "non-demo" mode. Beyond the conventional sensors, the AR Drone has firmware to estimate the velocity and attitude of the drone. The official documentation on the sensors and estimation techniques is sparse; however, Bristeau *et. al.* [13] analyzed the Parrot AR Drone 1.0 and state that the attitude and velocity estimation is accomplished through sensor fusion and visual odometry. The Parrot AR Drone 1.0 has a three-axis accelerometer, two-axis gyroscope, a one-axis vertical gyroscope, and two ultrasonic sensors. Using these sensors, Parrot achieves an accurate estimation

of attitude is derived from sensor fusion between the gyroscope and accelerometer. The accelerometer bias is not dealt with by the attitude estimator but instead through the a visual odometry velocity estimation algorithm. A downward facing camera uses optical flow to determine velocity and accelerometer error [13]. It can be assumed that the Parrot AR Drone 2.0 utilizes similar principles and algorithms for attitude and velocity estimation. Unfortunately, in practice, the velocity estimations proved extremely unstable and inaccurate while the attitude sensors were highly accurate. In order to obtain an accurate estimation of position without GNSS, various filters had to be applied to the acceleration sensor data.

The software for controlling and querying the drone is implemented in Python and abstracted from the PS Drone API [14]. The PS Drone API provides functionality for configuring the drone, flying with simple commands, and getting sensor data in at a single instance as well as example codes for how to expand upon those functions (e.g. fly with key controls using simple user input and flying commands.) A class, Chief_Drone.py is created to achieve flight and handle the navdata. In order to both flight and query the navdata, threading is implemented in Thread_Drone.py and shares the drone object instanciation. Both classes import and utilize heavily the PS Drone API. The script connects to the drone, allows for a calibration period, and allows the user to take off and fly as desired, all while tracking the current global time as well as the navdata. After a pre-determined period of time, the script moves to print the navdata, including time stamps, to a text file to be processed later.

4.1.2 Time of Flight Camera

A PMD Pico Flexx 3D ToF flash camera, or simply, Pico Flexx was utilized for the extremely low weight of eight grams. Unlike previous LiDAR technology, the Pico Flexx is considered a 3D/4D$(x, y, z, color)$ ToF camera and does not involve creating a 3D model by spinning a 2D slice but instead by taking a single image, much like a conventional camera and returning distance values in a 171×224 matrix for each distance in the X, Y, and Z. Along with the distances are confidence values, and grayscale values that allow for any needed filtering and heatmap (distance) generation.

The Pico Flexx camera has a 62°x 45°(H x V) field of view and a resolution of 224×171 or 38k pixels. Depending on the use case (changed programmatically), the camera can record up to 45 frames per second at a range varying from 0.1 to 4 meters. Being a low cost and very light weight (8 grams) 3D ToF camera, it was the ideal choice for testing the viability of low cost 3D ToF technology for indoor use.

4.2 Methodology

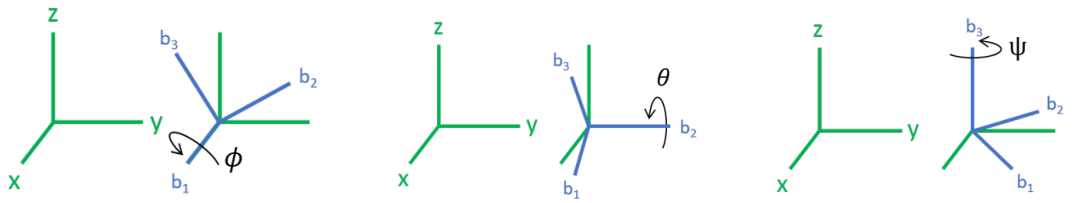
4.2.1 UAV Dynamics

Before retrieving navdata and processing that into meaningful pose estimations, the dynamics of quadcopter movements must be defined.

Figure 3: Axes Definition of Drone [15]

Considering the large errors that arise in pose estimation, many factors can be assumed marginal. Clearly, in indoor environments, wind effects can be ignored. For simplicity, the quadcopter can be assumed to have smooth travel and low acceleration, which will help create heuristics in the filtering algorithm. The quadcopter has six degrees of freedom, three in translation and three in rotation. Commonly, the axes of a quadcopter are designated based on the drone used, which has a forward facing camera. The X-axis is down the camera of the drone and the Y and Z axes are orthogonal as illustrated in 3, except the Z-axis will be taken as vertical.

The three rotations about the X, Y, and Z axes are roll, pitch, and yaw, respectively, with notations of ϕ, θ, and ψ and are illustrated in 4. Yaw will be in the domain $\psi \in (-180°, 180°]$, the common convention in UAVs. Roll will retain the intuitive domain of $\phi \in (-180°, 180°]$. The domain of pitch will be limited to $\theta \in (-90°, 90°]$ in order to avoid singularities in rotation matrices; however, in reality this is not limiting considering the quadcopter will never experience pitch near 90°. Due



(a) Roll (b) Pitch (c) Yaw

Figure 4: Three Types of Attitude Rotations [16]

to this limitation, quaternions, which never result in singularities, do not need to be used: Euler angles will be used for their clear interpretation.

The quadcopter follows simple kinematic motion equations; however, the sensors can only measure any forces/motions in relation to itself or the body. Therefore, the frame of reference of the drone body will be labeled b. Assuming the movement and any curvature of the earth to be negligible, the initial position of the drone can be taken as the reference inertial frame, labeled e for the earth frame. Technically, the choice of frame is arbitrary and as such, the initial position of the drone will be taken as the origin of the e frame of reference. The e frame is taken as the absolute reference because the LiDAR and drone need to share a common frame of reference in order to construct the 3D map. Any motion through space can be completely described as a combination of translation and rotation between the inertial reference frame e and the body frame b as seen in 5. The drone's pose, position and attitude, in reference to the inertial reference frame e over the entire flight window will be necessary for the creation of the LiDAR model. The position and attitude in reference to the inertial frame at any point in time k will be noted as $\mathbf{p}_k^e = [x_k, y_k, z_k]^T$ and $\boldsymbol{\theta}_k^e = [\psi_k, \theta_k, \phi_k]^T$. $\boldsymbol{\theta}$ will always be in reference to the inertial frame and will subsequently be labeled without the e superscript for concise formatting.

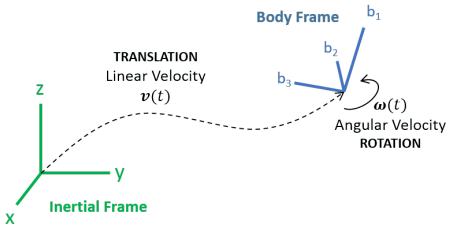


Figure 5: Quadcopter Body versus Intertial Frame Motion [17]

Given that the sensor readings will be measured against the body frame b, velocity estimation, accelerometer, and gyroscope values will need to be transformed through a rotational matrix to the reference frame e. At a time instance k, the rotational matrix to change the frame of reference of acceleration and velocity values will be labeled as R_b^e and is a function of the attitude at time k according to

$$R_b^e(\boldsymbol{\theta}_k) = \begin{bmatrix} c(\psi)c(\theta) & -s(\psi)c(\phi) + c(\psi)s(\theta)s(\phi) & s(\psi)s(\phi) + c(\psi)s(\theta)s(\phi) \\ s(\psi)c(\theta) & c(\psi)c(\phi) + s(\psi)s(\theta)s(\phi) & -c(\psi)s(\phi) + s(\psi)s(\theta)c(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{1}$$

where $s(x)$ and $c(x)$ are shorthand for $sin(x)$ and $cos(x)$, respectively. R_b^e is orthonormal; therefore, $R_e^b = (R_b^e)^{-1} = (R_b^e)^T$. The rotational matrix to change frame of reference of angular velocity will be labeled Γ_b^e and is a function of the attitude at time k according to

$$\Gamma_b^e(\boldsymbol{\theta}_k) = \begin{bmatrix} 1 & s(\psi)t(\theta) & c(\psi)t(\theta) \\ 0 & c(\psi) & -s(\psi) \\ 0 & s(\psi)/c(\theta) & c(\psi)/c(\theta) \end{bmatrix}$$

where $s(x)$, $c(x)$, and $t(x)$ are shorthand for $sin(x)$, $cos(x)$, and $tan(x)$, respectively. Given attitude $\boldsymbol{\theta}$ and the sensor values of velocity, acceleration, and angular velocity relative to the body frame $\mathbf{v}^b$, $\mathbf{a}^b$, and $\boldsymbol{\omega}^b$ at a time k, the sensor values can be rotated into the reference frame e according to

$$\mathbf{v}_k^e = R_b^e(\boldsymbol{\theta}_k)\mathbf{v}_k^b \tag{2}$$

$$\mathbf{a}_k^e = R_b^e(\boldsymbol{\theta}_k)\mathbf{a}_k^b \tag{3}$$

$$\boldsymbol{\omega}_k^e = \Gamma_b^e(\boldsymbol{\theta}_k)\boldsymbol{\omega}_k^b \tag{4}$$

The sensors that provide information to the drone's position are limited to the accelerometer and the velocity estimator. The position and velocity at time $k+1$ can be estimated using dead reckoning where the change in time between each measurement Δt is determined by the sampling frequency, f_s, as $\Delta t = \frac{1}{f_s}$. In the reference frame, e, the estimates of position and velocity are simple kinematics.

The attitude of the drone can be estimated using dead reckoning and through the magnetometer and accelerometer readings. Estimating the attitude given the attitude and angular velocities of the previous time step is again kinematics,

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta t \boldsymbol{\omega}_k^e \tag{5}$$

where $\boldsymbol{\omega}_k^e$ is calculated using 4.

The magnetometer can estimate of yaw while the accelerometer can estimate pitch and roll. Together, these two sensors can provide a complete estimate of attitude at any time. Over short distances, the magnetic field of the earth can be assumed constant and as such, the X and Y axis components of the magnetometer's readings can be utilized in order to determine heading. The magnetometer's values are always in reference to the inertial frame e, considering they measure the earth's magnetic field as as such, $\mathbf{m^e}$ will be denoted without the frame superscript for clarity. Given the magnetometer's sensor values as $\mathbf{m} = [m_x, m_y, m_z]^T$, yaw is a function of the magnetometer's x and y components m_x and m_y, respectively, the yaw are calculated as

$$\psi(\mathbf{m}) = tan^{-1}(\frac{m_x}{m_y}).$$

The sign of the magnetometer values is important and in many coding languages, a sign discriminant command for $tan^{-1}()$ is $atan2()$, which will be used. Given the accelerometer sensor values as $\mathbf{a^b} = [a_x^b, a_y^b, a_z^b]^T$, the pitch and roll of the drone are given by

$$\theta(\mathbf{a^b}) = tan^{-1}(\frac{a_x^b}{a_z^b})$$

and

$$\phi(\mathbf{a^b}) = tan^{-1}(\frac{a_y^b}{a_z^b}).$$

Together, the magnetometer and accelerometer can provide a long term stable estimate of attitude at any time k as

$$\boldsymbol{\theta_k} = [\psi(\mathbf{m_k}), \theta(\mathbf{a_k^b}), \phi(\mathbf{a_k^b})]^T \tag{6}$$

4.2.2 Digital Signal Processing

Each sensor and firmware estimate given by the UAV is susceptible to noise and some have bias; therefore, many of them will need to be processed to give reasonable/accurate results. Some filtering will be done in sensor fusion while others will be first filtered and then put into sensor fusion. The two signal processing filters implemented for this project is a moving mean and recursive filter. The moving mean and recursive filter are low pass filters (**LPF**) which allow lower frequencies and attenuate higher frequencies. The UAV is assumed to have smooth flight and as such, the true sensor signals will be composed of lower frequencies: high frequencies indicate noise.

The moving average filter is a simple implementation of a LPF optimal for reducing random noise while retaining a sharp step response and as such, provides a simple filter for the incoming IMU data. The moving

average filter is the convolution of a signal with a rectangular pulse of area of one and in practice, operates on a discrete signal over a window as described by

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

where y[ ] is the output signal, x[ ] is the input signal, and M is the window size. Window size relates to cutoff frequency: a larger window corresponds to a lower cutoff frequency and vice-versa. The amount of noise reduction is equal to the square root of the window size (e.g. a 100 point window moving average filter equates to a noise reduction factor of 10) [18]. The moving average filter is applied to the firmware attitude estimations to reduce random noise while retaining as much information as possible. It performs exceedingly well at reducing noise while retaining information. An example period of the firmware's estimation of roll was taken and filtered through the moving mean LPF as illustrated in 6.

Recursive filters are an implementation of a LPF based on a cutoff frequency, f_c. The recursive filter is also called an Infinite Impulse Response (**IIR**) and are capable of achieving a long impulse response. A simple implementation of an IIR filter is a single pole low-pass recursive filter. The single pole recursive filter acts nearly identical to an RC filter in an analog setting where any sharp discontinuities are slowly shaped. The single pole which operates according to

$$y[k] = a_0 x[k] + b_1 y[k-1].$$

The coefficients are chosen by $a_0 = 1 - \alpha$ and $b_1 = \alpha$ where α is a value between zero and one. Theoretically, α is determined by the cutoff frequency ω_c in radians according to

$$\alpha = e^{-\omega_c}.$$

In practice, α can be tuned to achieve desired results in situations where the desired output is known [18].

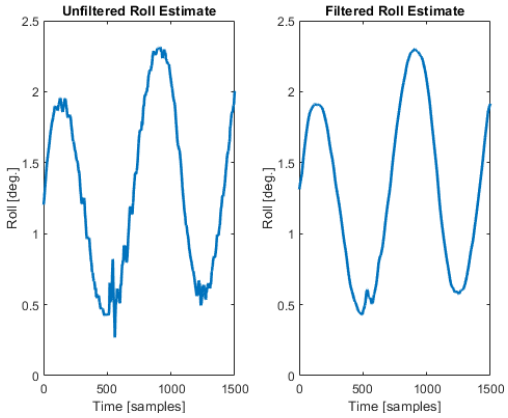


Figure 6: Example of Roll Data Before and After Applying a Moving Mean Filter

In order to determine the optimal cutoff frequency ω_c, the rotors of the AR Drone were removed and the hull was placed completely still. The drone was instructed to "fly" (certain sensors read zero when in "landed" mode) and record navdata for 30 minutes. The true sensor values should be constant and all except yaw should read zero; however, due to noise and bias, none of the sensors provided perfect values. Every sensor displayed bias, outliers, and white noise—random signal with constant power spectral density. The bias seems to change with every test/flight and as such cannot be simply removed. Before each flight, the drone was kept grounded for a small calibration period (30 seconds) and in the data processing, the average values over that period are subtracted from the signal.

The sensors contained a significant amount of white noise: enough to significantly decrease the quality of pose estimates. Because of the presence of white noise, there is no clear cutoff frequency where frequencies higher than his cutoff have a distinct increase in noise. The cutoff frequency must be chosen delicately to remove as much white noise as possible without distorting the actual information present in the low frequencies. The cutoff frequency for each sensor was chosen based on trial runs. Additionally, outliers were removed for all sensors except yaw (magnetometer and firmware estimate). The yaw value could theoretically be any value in its domain $\psi \in [0°, 360°)$ but the other sensors have clear limitations. In reality, roll and pitch are limited without the quadcopter crashing. Assuming smooth acceleration and rotation, the acceleration and angular velocity values are limited as well. Given reasonable limitations, any value outside the range can be averaged to the surrounding points. A reasonable factor to multiple the operating range is $1.25x$ as determined by testing/calibration. The ranges of certain sensors are given in 1. Any value outside the given ranges is equated to the average of 100 neighboring points. The ranges for outliers is relatively arbitrary and inconsequential but it was able to remove a few outliers that could significantly impact the results.

Table 1: Range of Outliers

Sensor/Estimate	Expected Range	Threshold
Pitch/Roll	[-30°,30°]	[-37.5°,37.5°]
Horizontal Acceleration (a_x,a_y) [G]	[-0.25 , 0.25]	[-0.3125,0.3125]
Vertical Acceleration (a_z) [G]	[0.75 , 1.25]	[0.6875,1.3125]
Angular Velocity	[-100°/s,100°/s]	[-125°/s,125°/s]

4.2.3 Complementary Filter

The complementary filter achieves sensor fusion intuitively by summing high noise, no bias sensors fed through a LPF with low noise, biased sensors fed through a high pass filter (**HPF**). Both filters are digitally implemented through first order LPF, $G_1(z)$, and HPF, $G_2(z)$, and simply summed to provide an estimation. To assure that no information is lost, the filters are defined as the complement of each other simply given as

$$G_2(z) = 1 - G_1(z).$$

Through this definition, $G_1(z)$ and $G_2(z)$ effectively span the entire frequency spectrum and converge at the cutoff frequency ω_c as seen in 7. Furthermore, both $G_1(z)$ and $G_2(z)$ have unit gain in their pass bands. In this way, the complementary filter retains the data from the entire frequency spectrum while obtaining simple and powerful sensor fusion.

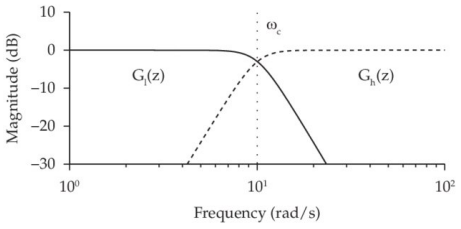


Figure 7: Frequency Response of Complementary Filter with Cutoff Frequency ω_c [19]

In order to estimate attitude, the gyroscope, accelerometer, and magnetometer sensor readings and the firmware estimated attitude are fused to a single estimate. An AHRS was designed using a complementary filter to fuse the accelerometer, magnetometer, and gyroscope readings $\mathbf{a}$, $\mathbf{m}$, and $\boldsymbol{\omega}$, respectively. The system to determine attitude from the sensors is given in 8 as the "Sensor Complementary Filter" dashed section. Attitude estimation from the gyroscope, $\boldsymbol{\theta_{gyro}}$, is calculated using 4 and 5. $\boldsymbol{\theta_{gyro}}$ is accurate over the short term but is prone to bias and walking errors over long periods; therefore, it will be fed through the HPF. An estimation of attitude from both the accelerometer and magnetometer, $\boldsymbol{\theta_{acc,mag}}$, is calculated using 6. $\boldsymbol{\theta_{acc,mag}}$ is inaccurate but the error does not accumulate and as such, attitude estimations from the magnetometer and accelerometer are long term stable. This estimate will be fed through the LPF.

$\boldsymbol{\theta_{gyro}}$ and $\boldsymbol{\theta_{acc,mag}}$ will be fused using a complementary filter to provide an IMU based attitude estimate, $\boldsymbol{\theta_{IMU}}$. Considering the estimate for the attitude from the gyroscope as given by 5 is calculated through integration, the complementary filter can be simplified. Instead of performing the LPF and HPF separately and summing, as described in 8, the gyroscope values are multiplied by a constant, summed with the estimate from the accelerometer, and passed through a LPF. In practice, this becomes a weighted average

$$\boldsymbol{\theta}_{\mathbf{IMU}} = x * \boldsymbol{\theta}_{\mathbf{acc,mag}} + (1 - x) * \boldsymbol{\theta}_{\mathbf{gyro}} \tag{7}$$

where $0 \leq x \leq 1$ to assure unit gain. The weight, x, was determined through trial runs and calibrations where the attitude was known.

The AR Drone offers the unique output of an estimated attitude from on-board firmware and probably employs a similar technique as previously described [13]. The estimate is reasonably accurate but offers drastically different values when the drone is in "flight" or "landed" modes and is thus hard to use. Furthermore, this estimate has small perturbations and as such is filtered with a LPF, $G_3(z)$. A moving average filter is used for $G_3(z)$ due to the aforementioned benefits. A small window, corresponding to a high cutoff frequency, is used to only filter the noise while retaining most of the information present in the signal. The attitude

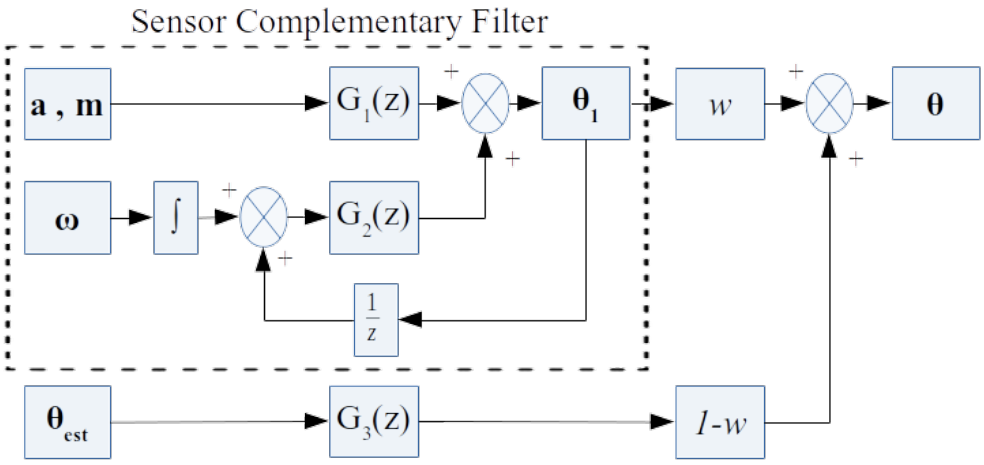


Figure 8: AHRS for Attitude Estimation

estimation given from the AR Drone, $\boldsymbol{\theta_{drone}}$ will be fused with the previously found estimate, $\boldsymbol{\theta_{IMU}}$. The estimate of attitude through the fusion of the IMU and firmware estimate is a weighted average

$$\boldsymbol{\theta} = w * \boldsymbol{\theta}_{\mathbf{IMU}} + (1 - w) * \boldsymbol{\theta}_{\mathbf{est}} \tag{8}$$

where $0 \leq w \leq 1$ to assure unit gain. The weight, w, is a different yet similar value to the aforementioned x and was determined experimentally with test runs of known attitudes similar to the previous system. The entire system of filtering the sensor data and fusing the result with the firware estimate is given in 8.

In order to estimate the position of the UAV, another complementary filter was implemented. The only sensor that provides position information is the accelerometer; however, the on-board firmware of the AR Drone provides an odometry based estimation of velocity [13]. Dead reckoning position estimation can be achieved by assuming the initial position $\mathbf{p}_0 = [0, 0, 0]^T$; utilizing the measured acceleration, $\mathbf{a}^b$, and estimated velocity, $\mathbf{v}^b_{est}$; and changing the frame of reference using 2 through 3. The velocity was maintained as a state and is determined as the output of the complementary filter between the acceleration and estimated velocity as described in 9. In estimating velocity, the anti-derivative of acceleration is an accurate estimation of the change in velocity over short periods; however, it is prone to bias. As such, the accelerometer readings were fed through the HPF $G_2(z)$. Visual odometry is relatively inaccurate but maintains the same errors over long periods of time and as such was fed through the LPF $G_1(z)$. The complementary filter of acceleration and velocity provided a reasonable estimation of velocity. With this velocity estimation, the position was calculated through dead reckoning as described in 9.

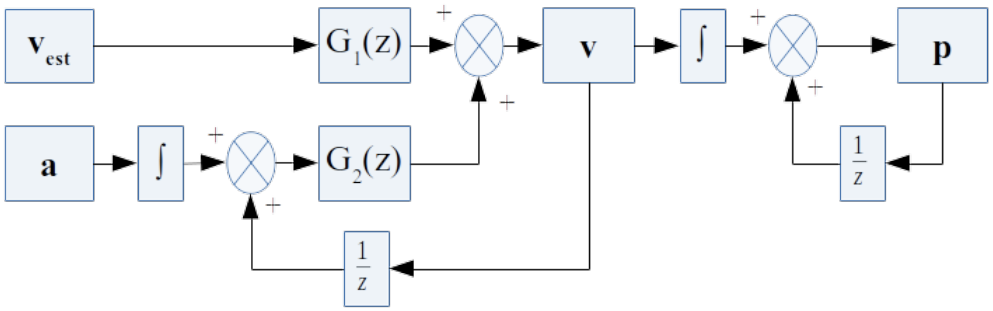


Figure 9: Position Estimation Implemented through a Complementary Filter and Dead Reckoning

4.2.4 Data Synchronization

Navigation data coming from the drone and camera data coming from the SBC controlling the camera need to be synchronized in time. Python libraries controlling both the drone and the computer connected to the camera can be set to use the NTP protocol to connect to the same internet servers when automatically synchronizing their clocks with the internet. This method ensures the times will rarely be more than a few milliseconds off. Navigation data from the drone in this use case is recorded at the highest possible rate, 200Hz. In contrast, the Pico Flexx ToF camera is considerably limited to specific use cases of 5-45Hz recording rate, which directly corresponds to its frame-rate. The frame-rate is inversely proportion to the range such that 45 FPS corresponds to less than a meter of range and even less exposure time. Interior mapping, ideally, uses a camera with at least a couple meters of range so that the drone does not have to carry the camera so far that it is considerably displaced from its planned path. Thus, data will be recorded at the lowest possible frame-rate use case of 5FPS (indoor room reconstruction), giving 4 meters of range.

Navigation data corresponds to Pico Flexx data at a $200:5$ or $40:1$ ratio and may be synchronized via simple linear mapping. Given the set of navigation data times $T = \{t \mid t \in \mathbb{R}(a,b)\}$ where a and b represent the start and stop times, respectively, and the set of Pico Flexx times $T^{'} = \{t \mid t \subseteq T\}$, we are looking for the subset of times such that $T_s^{'} = \{t \mid T^{'} \cap T\}$ represents the synchronized times. Since all data is timestamped, the relevant IMU data is taken along with the timing synchronization in a chronologically indexed fashion. With the understanding that we are really working with discrete time-steps, the matching was done using a seeking threshold that found the smallest distance between times in each time set.

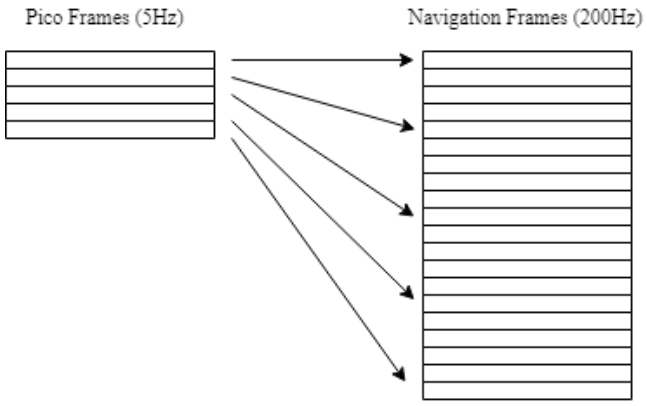


Figure 10: Data Synchronization

After synchronization of the data, the navigation data is held in a matrix

$$N = \{\mathbb{R}^{f \times a} \mid f = [1, 2, ..., Num\_Frames], a = [1, 2, ..., Num\_Attributes]\}$$

where f, a denote the frames and attributes such that each row corresponds to a single frame, and each column corresponds to a different attribute (angle, position, velocity, etc...).

4.2.5 Point Cloud Rendering and Denoising

Each frame of the Pico Flexx ToF data comes with a timestamp and set of coordinate matrices $\{X, Y, Z\}$ such that $X, Y, Z \in \mathbb{R}^{171 \times 224}$ whose dimensions represent the resolution of the camera. Every element in either of the matrices X, Y, Z contain a value that is a distance to the object from which it reflected. Therefore, taking any *ith* element from all three matrices produces the vector (X_i, Y_i, Z_i) as euclidean coordinate representing distance in $\mathbb{R}^3$ to the object for which it reflected back to the image sensor of the camera(origin) in meters.

In order to work with this data as point clouds in MATLAB, the X,Y,Z coordinate matricies must be reshaped such that $X^*, Y^*, Z^* \in \mathbb{R}^{38304 \times 1}$. After horizontal concatenation, this forms a matrix $P \in \mathbb{R}^{38304 \times 3} =$

$[X^*, Y^*, Z^*]$. This way P_1 and P_{38304} represent the first (X_1^*, Y_1^*, Z_1^*) and last $(X_{38304}^*, Y_{38304}^*, Z_{38304}^*)$ euclidean distances in the set.

K-nearest neighbor(KNN) is an algorithm often used in machine learning classification and regression. Here it will be used to to denoise the point clouds, if necessary. Each point in the point cloud can be assigned a number of neighbors $\boldsymbol{k}$ as well as a threshold $\boldsymbol{T}$ representing the distance for which, when over, the point will be deleted. In figure 11, the point with coordinate (X_k^*, Y_k^*, Z_k^*) will be deleted because $K < 3$ ad $T > 0.1$. In our use case T would measure distance in meters.

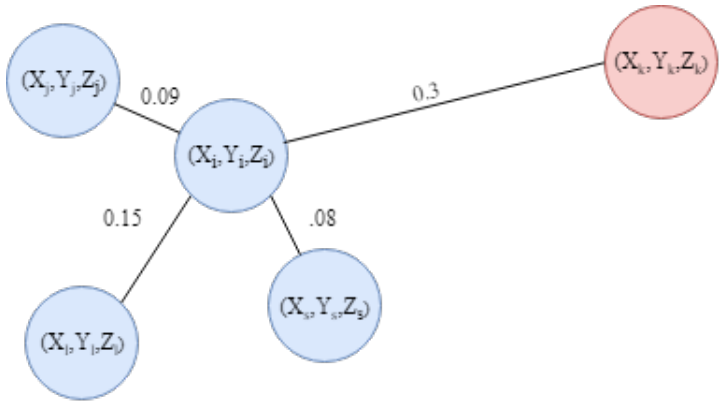


Figure 11: K-nearest neighbor denoising

4.2.6 Transforming Point Clouds

Four primary methods will be used to develop a complete model using the point clouds recorded by the Pico Flexx:

1. 100% navigation data transform
2. weighted combination of Navigation transform and ICP transform
3. 100% ICP transform
4. 100% navigation transform followed by 100% ICP transform

These methods will be refered to as methods 1-4 as listed.

In each method aside from method 3, the navigation data must be converted into useful transformations. Again looking at matrix $N = \{n \mid n = \mathbb{R}^{f \times a}\}$ containing the synchronized navigation data, form another matrix $D = \{d \mid d = \mathbb{R}^{f-1 \times a}\}$ such that each row is the difference between each sequential row in N:

$$D = \{d \mid i \in \{2, 3, ..., f\} \mid d = N(i, all) - N(i-1, all)\}$$

Therefore the data contained in $D(i)$ would be the data needed to produce a transform of the form

$$Q_i = \begin{bmatrix} 1 & a & b & dx \\ c & 1 & d & dy \\ e & f & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where {a,b,c,d,e,f} represents the rotational result from the use of (1) and $\{dx, dy, dz\}$ represents translational difference such that $P[i\text{-}1]Q_i = P[i]$, with $P[i]$ denoting the ith point cloud.

4.2.7 Iterative Closest Point Registration

Relying completely on pose estimates to perform the transforms often leads to propagating errors that skew entire parts of the model away from their true positions. In this paper, navigation data will be heavily supplemented by iteratively matching the closest points in two sequential cloud frames and returning a transform that best fits the $[n+1]$ frame to the $[n]$ frame.

Consider two point clouds $\{F, M\}$ where F denotes the fixed cloud, and M denotes the moving cloud to be transformed. $\{F, M\}$ are finite dimensional in any real vector space: $F, M \in \mathbb{R}^N$. There exists a transformation Q_i such that $Q_i(M)$ best reduces the euclidean distance S between M and F. A function $\boldsymbol{dist}$ can then be formed by simply taking the square of the distance between each set of points and guessing a transform exhaustively such that the function

$$\boldsymbol{dist}(\boldsymbol{Q_i(M), S}) = \sum_{\boldsymbol{m \in M}} \sum_{\boldsymbol{d \in S}} \boldsymbol{(m-s)^2}$$

is **minimized**. This function is almost always minimized locally as global minimization requires exhaustively iterating through every possible transformation Q_i, where some extrapolation methods can be used to reduce the amount of iterations needed. For use in this paper, a registration returns a final **rigid** transformation Q_i that best matches the two point clouds $\{F, M\}$ through only rotation and translation.

Algorithm 1 ICP Registration [20]

```
function ICP(M, S :)
q = q_0
while (dist > tolerance) do
    X = ∅
    for (m_i ∈ M) do
        s' = s ∈ S that is closest
        X = X + ⟨m_i, s'⟩
        q = least_squares(X)
    end for
end while

return q
end function ICP
```

The $\boldsymbol{least\_squares}$ function minimizes the distance function in each $\boldsymbol{\langle m_i, s' \rangle}$ pair. This algorithm works best when the points being transformed are initially "sufficiently" close [20].

4.2.8 Total Transformations and Merging

The last two sections laid the groundwork for producing the proper transformations that will geo-reference each frame, sequentially, to the previous. In order to develop a complete model, the transformation from each synchronized frame of navigation data must be weighted with the ICP-returned transformation, and then accumulated into the total transformation between each frame. The percentage trust in the navigation data W_n is a value that changes experimentally depending on the accuracy of the filtering. This leaves the trust in the ICP algorithm to be $W_{ICP} = 1 - W_n$. Therefore the **updated set of transformations**

$$\boldsymbol{Q = \{Q_i \mid Q_i = (Q_{i,nav} * W_n + Q_{i,ICP} * W_{ICP})\}} \tag{9}$$

Algorithm 2 Transform and Merge

 # Initialize
$accumulate\_transform = identity(4,4) * Q_2$
$pt\_cloud\_scene = pt\_Cloud[1]$
for $i = 2\_through\_Number\_of\_Frames$ **do**
  $accumulate\_transform = accumulate\_transform * Q_i$ **# Gather the total transform**
  $moving\_cloud = pt\_Cloud[i]$ **# Gather the *ith* cloud**
  $aligned\_cloud = moving\_cloud * (accumulate\_transform)$ **# Perform transformation**
  $pt\_cloud\_scene = merge(pt\_cloud\_scene, aligned\_cloud)$ **#Merge with previous**
end for
return pt_cloud_scene **# Return total stitch**

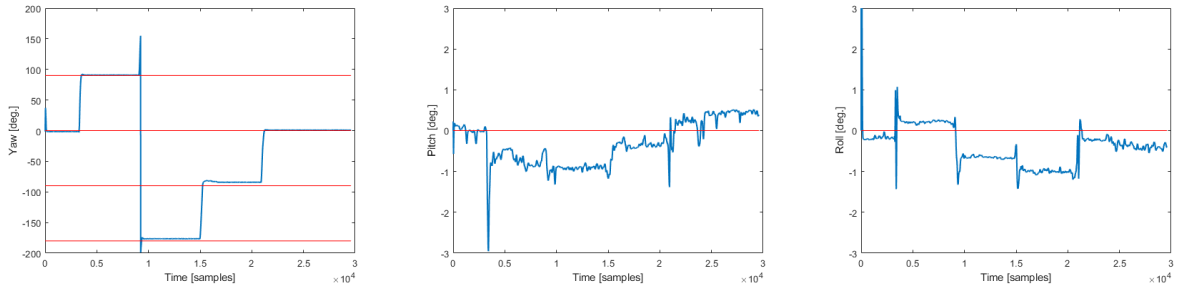
Since any change in trust weight for the navigation data is taken into account in (9), this algorithm will take into account the first three methods discussed for developing a full model. At the extremes, trusting Navigation 100% will result in the ICP transforms being zeroed out, and trusting the ICP algorithm 100% will result in the navigation transforms being zeroed out.

In order to accomplish the fourth method of stitching, applying the navigation data and then the ICP algorithm afterwards, the transformation data is stored as if the trust in the navigation data was 100%, and 0%, then the frames were separately run through the ICP algorithm and the transforms produced there were also applied. The nature of the ICP algorithm governs that it works best when the moving cloud is already sufficiently close to the fixed frame [20]. This means that the model would be produced with the highest level of accuracy when the navigation data is most accurate and ICP is integrated in afterwards, as in method four.

5 Results

5.1 Pose Estimation

The yaw, pitch, and roll components of the navigation unit on-board the Parrot AR Drone 2.0 proved to be the most reliable data pieces. The accuracy and precision of the AHRS was tested by setting the UAV on the ground and rotating the yaw by 90° for four 30 second periods. The filtered attitude values along with the expected values (horizontal lines) are given in 12.



(a) Yaw (b) Pitch (c) Roll

Figure 12: AHRS Attitude Estimations

The expected results, measured values, and standard deviation are given in 2. A good indication of the accuracy of the attitude estimates is the difference in means between the expected and measured values for yaw, pitch, and roll. As listed in 2, no value differed by more than 7°and five out of six test cases was within

4°. In the real world implementation tested, these values provided sufficiently accurate estimations for point cloud registration. Precision can be described through standard deviation and as listed in 2, each test case had a standard deviation less than 1°. This was precise enough to register point clouds.

Table 2: AHRS Test Expected and Measured Values

Attitude	**Expected**	**Measured**	
	Mean [deg]	Mean [deg]	Standard Deviation [deg]
Yaw	0	0.82	0.070
	90	90.78	0.227
	± 180	-176.51	0.070
	-90	-83.86	0.809
Pitch	0	-0.33	0.537
Roll	0	-0.39	0.536

GNSS-denied position estimation was significantly less accurate, particularly over large distances. It is difficult to provide an exhaustive test for position estimation considering the laborious process of flying and processing navdata; however, a few representative samples will be illustrated. The drone was flown left along the positive Y-axis and the expected and measured position estimates are given in 3 trial 1. The drone was flown in a straight line forward, the positive X-axis, and the expected and resultant position estimates are given in 3 trial 2. The drone was flown along the positive X-axis and the positive Y-axis and the expected and measured estimates are given in 3 trial 3.

Table 3: Final Position Test Expected and Measured Values

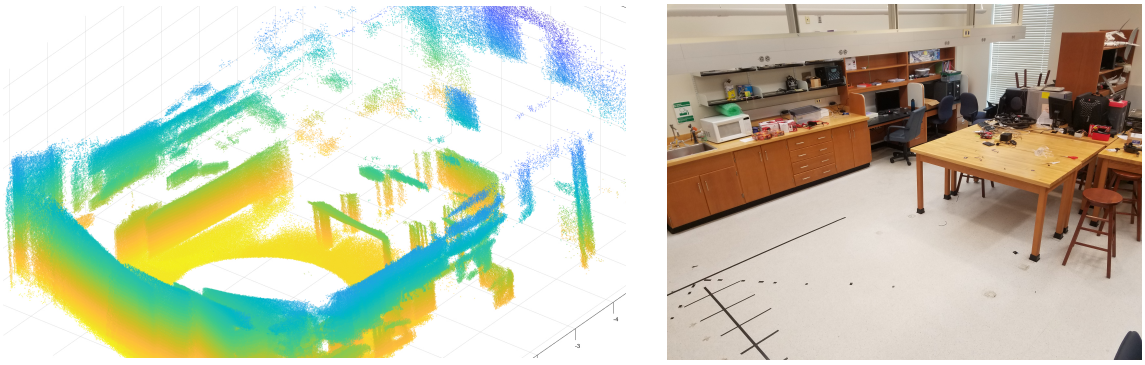
Axis	**Trial**	**Expected**	**Measured**	**Difference**	**% Error**
X	1	0	-1.60	1.6	-
Y		6.7	-5.71	.99	14.8
X	2	39.6	68.20	28.6	72.2
Y		0	-7.80	7.8	-
X	3	14.0	24.74	10.74	76.7
Y		-4.3	-2.46	1.84	42.8

Clearly, the position estimates are wildly inaccurate. Over short distances like trail 1, the error remains within 2m but over long distances, the error is extreme, reaching up to 70%. The accelerometer was observed to have an extreme bias, when the acceleration should be zero like in trial 1 and 2 in the X and Y axis, respectively, the acceleration was non-zero. Any non-zero acceleration results in displacement that is simply not correctable without GNSS. Tuning the complementary filter for long and short distances proved difficult because the parameters made it accurate for one or the other but not both. The presented tests in 3 are not a complete evaluation of the position estimation but they give clear indication to the inaccuracies. The only way a comprehensive model can be created over long translations is with the aid of ICP estimations of movement.

5.2 LiDAR Mapping

When the drone was stationary and rotated about the Z-axis, method one (applying navigation data transforms only) was the most effective in developing the transformations necessary to put together a stitch that physically modelled the desired room. In approximately 1 out of every 4 recordings, the navigation data

gathered from the drone was highly variable. In these cases, method four (applying navigation data transforms and then correcting the frames through the ICP algorithm) proved more effective than navigation transforms alone. A 360° scan of the workspace was taken by rotating the UAV and flash LiDAR on a chair and gathering the navigation information and frames. The two sets of data were processed in post, weighting more towards the navigation data transforms and the resultant 3D model and a camera image of the workspace are given in 13. Rendering the models required sufficient processing power and a discrete GPU is highly recommended for at least viewing the models.



(a) LiDAR Model of Workspace

(b) Camera Panorama Photo of Workspace

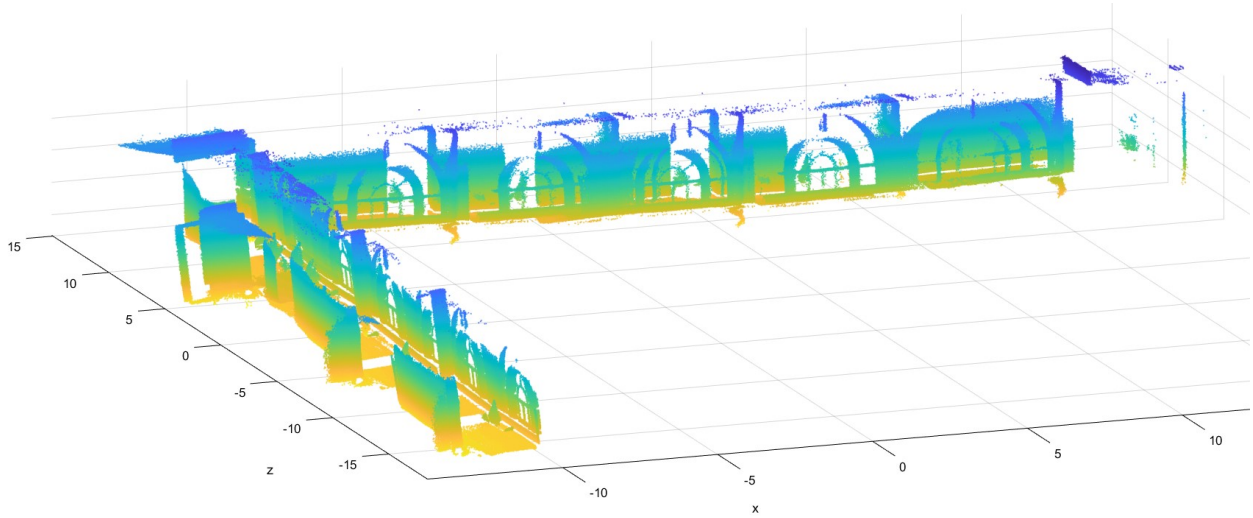
Figure 13: Workspace Model with Corresponding Photo

The model given in 13a is a visually sufficient representation of the workspace from a limited point of view. The white spaces and lack of detail towards the rear (top right of image) is due to the limited visibility of the flash LiDAR placed in the center of the open space at about 1m off above the ground. The LiDAR was placed at about the same height as the table and as such, cannot view the top of the table nor the counter-top. Despite the limitations of the placement of the LiDAR, the point cloud was registered well. There is little to no repetitions of features. The walls are single planes and the table legs are correct, meaning each frame was correctly transformed into the absolute reference. The accuracy of the attitude estimations based on navdata is illustrated considering the model was created using method one (100% navdata.) The accuracy of the method 1 registration is supported by the accuracy of attitude estimation given in 2.

Methods two and three (weighted combination of navigation, full ICP, respectively) proved more effective in scenes with little to no rotation and simple, one-dimensional translation. The ICP algorithm was significantly more accurate compared to navdata in scans with many large, flat surfaces and little rotation. The power of ICP in these situations is likely due to the point-to-plane, KD-Tree extrapolation method best suited for planar features. The pose estimation is still highly accurate for rotation but prone to bias over large translations (e.g. a 40m hallway.), likely due to IMU bias and lack of GNSS. A scan of the hallway was taken by putting the UAV on a rolling chair and carting it down the hallway. The resultant LiDAR frames and navigation data were processed, putting more weight (85% to 100%) on the ICP transforms and the resultant model is shown in figure 14. The hallway produced in figure 14 is the result of 85% to 100% ICP stitching and merging more than 900 Pico Flexx pictures taken at 5FPS. The stitch was produced by using the ICP algorthim to merge 100 frames at a time, then manually transforming each of the 9 segments, and once again using the ICP algorithm to correct the position. Manual touch up was simply needed to guide the stitch every 100 frames (approximately 7 meters), roughly every two windows length, to avoid accumulation of error. For a total of 4.9 billion $\langle x, y, z \rangle$ data points, the hallway in 14 is approximately 60 meters in total length. The hallway stitch contains holes in the floor that show as white spots on the stitch and correspond to the black/blue squares in the real photo. It is not clear how these were caused but it seems plausible it is an issue with the matte color of the equally spaced spots denying reflection of the Pico Flexx's flash. The Pico Flexx camera does produce its own light and is considerably better performing in dimly lit scenes. As such, this is possibly a result of the combination between the scenes contrast and the cameras exposure.

Additionally, there is not data on the inlets on the wall side of the hallway because the UAV was simply translated down the hall, which created blind spots.

The LiDAR flash camera was fixed to the quadcopter and flown down the same hallway as given in figure 14; however, the resultant 3D model was erroneous. Implementing LiDAR onto a small quadcopter is a physical challenge and is detrimental to the maneuvering ability of the UAV. As seen in 2, the LiDAR is at the very front of the UAV and enacts a moment upon the drone. Even at such a lightweight, the distance from the center makes the drone unstable, nearly uncontrollable, and constantly pitch forward, reducing the LiDAR video quality significantly. UAV flight tests with the LiDAR camera attached on top and the micro-computer controlling the LiDAR unit added nearly 200 grams to the total weight of the drone and rendered it unable to pick up altitude more than a foot or respond to any significant amounts of input.



(a) LiDAR Model of Hallway

(b) Camera Panorama Photo of Hallway

Figure 14: Hallway Model with Corresponding Photo

Figure 13 was best produced under purely navigation data, showing that the program did produce the transforms required from the data. Yet, figure 14 could only be produced via ICP stitching. Failed attempts at reconstruction of the hallway using navigation data only were supported by the lack in proper pose estimates of the UAV, yet validate attitude estimates.

6 Conclusions and Future Work

Registering a sequence of frames from an indoor 3D ToF LiDAR camera into a single comprehensive model was demonstrated to be viable. A quantitative analysis of the accuracy of the models is unavailable; however, each virtual representation is visually sufficient to describe the area. The Pico Flexx has proven a suitable and powerful tool in LiDAR mapping. 3D flash LiDAR is a viable alternative to 2D LiDAR technologies in applications requiring light weight, airborne micro-drone deployment with cost efficient mapping. This proposed method of data collection and point cloud registration could be more accurate and robust given a UAV with more carrying capacity, stability, and accurate navigation data. The estimation of pose without GNSS was accurate in estimating attitude but inaccurate in estimating position due to accelerometer bias: attitude has long term stable sensors while position requires the long term stable sensor of GNSS. ICP algorithms were able to supplement this inaccuracy in situations with large flat surfaces. Despite reasonable results when the UAV was carted, flying with a 3D ToF camera and SBC did not return a good model.

Despite difficulties in flight, the project clearly demonstrates the efficacy of LiDAR mapping and provides justification for further research and development in the area using new 3D ToF LiDAR cameras. Improved position estimation through a Kalman filter may yield more accurate estimations. Furthermore, kerneling techniques on the point cloud data could significantly improve and even replace ICP for the transform estimations from the point cloud.

References Cited

[1] Francesco Nex and Fabio Remondino. Uav for 3d mapping applications: a review. *Applied Geomatics*, 6(1), November 2013.

[2] Yi Lin, Juha Hyypp, and Anttoni Jaakkola. Mini-uav-borne lidar for fine-scale mapping. *IEEExplore Journal of Geoscience and Remote Sensing*, 8(3), April 2011.

[3] Aaron Montoya, Bertrand Vandeportaele, Simon Lacroix, and Gautier Hattenberger. Flight autonomy of micro-drone in indoor environmentsusing lidar flash camera. Technical report, IMAV 2010, International Micro Air Vehicle Conference and Flight Competition, July 2010.

[4] Rongbing Li, Jianye Liu, Ling Zhang, and Yijun Hang. Lidar/mems imu integrated navigation (slam) method for a small uav in indoor environments. *Inertial Sensors and Systems*, 2014.

[5] Kruno lenac, Andrej Kitanov, Robert Cupec, and Ivan Petrovic. Fast planar surface 3d slam using lidar. *Robotics and Autonomous Systems*, 2017.

[6] Naser El-Sheimy, Haiying Hou, and Xiaoji Niu. Anaylsis and modeling of inertial sensors using allan variance. *IEEE Transactions on Instruments and Measurement*, January 2008.

[7] Antonio Vasilijevic, Bruno Borovic, and Zoran Vukic. Underwater vehicle localization with complementary filter: Performance analysis in the shallow water environment. *Springer Science+Business Media*, August 2012.

[8] Randall W. Beard. Quadrotor dynamics and control. Technical report, Bringham Young University, May 2008.

[9] Tamer Shamseldin, Ankit Manerikar, Magdy Elbahnasawy, and Ayman Habib. SLAM-based pseudo-gnss/ins localization system for indoor lidar mobile mapping systems. Technical report, Purdue University, April 2018.

[10] Kenneth Gustavsson. *UAV Pose Estimation using Sensor Fusion of Inertial, Sonar and Satellite Signals.* PhD thesis, Uppsala Universitet, 6 2015.

[11] Manon Kok, Jeroen Hol, and Thomas Schon. using intertial sensors for position and orientation estimation. *Foundations and Trends in Signal Processing*, 11(1-2), 2017.

[12] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. Technical report, Carnegie Mellon, Jul 2014.

[13] Pierre-Jean Bristeau, Francois Callou, David Vissiere, and Nicolas Petit. The navigatiioin and control technology inside the ar.drone micro uav. *IFAC World Congress*, August/September 2011.

[14] J. Philipp de Graaff.

[15] Alessandro Benini, Adriano Mancini, and S Longhi. An imu/uwb/vision-based extended kalman filter for mini-uav localization in indoor environment using 802.15.4a wireless sensor network. *Journal of Intelligent & Robotic Systems*, 70, 04 2013.

[16] Charles Tytler. Euler angles, 2018. Online; accessed July 3, 2019.

[17] Charles Tytler. Inertial motion expressed in body frame components, 2018. Online; accessed July 3, 2019.

[18] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Pub, 1997.

[19] Armando Alves Neto, Douglas Guimaraes Macharet, Victor Costa da Silva Campos, and Mario Fernando Montenegro Campos. Adaptive complementary filtering algorithm for mobile robot localization. *Journal of the Brazilian Computer Society.*

[20] Wikipedia. Point set registration. https://en.wikipedia.org/wiki/Point$_set_registration$. *Accessed* : $2019 - 7 - 10$.