Project Report: Autonomous Flight & Mapping

Mike Atwood David Ord Alex Vasquez Jacob Zenger
https://dronebot.home.blog/

***Abstract*—Our project achieves autonomous drone flight by using PID feedback loops, interfacing with various peripherals and sensors, and performing data transformations to control a quadcopter. Visual simultaneous localization and mapping (VSLAM) from a stereo camera to achieve autonomous localization was developed and demonstrated by achieving image registration and point-cloud creation. Initially Angle of Arrival capabilities of Bluetooth 5.1, were to be included. Autonomous indoor mapping and localization has numerous applications to safety and services. Current systems are often inefficient, inaccurate, and their usability is limited by the need for beacons. The Bluetooth 5.1 standard introduces a solution by pairing angle of arrival information and distance measurements. When combined with 3D mapping, the capabilities of Bluetooth 5.1 enable the development of a drone which can determine a path to an object within a large space. The proposed project intended to prove this concept by developing a drone which uses VSLAM to create a 3D representation of a space, such as the floor of a building.**

***Index Terms*—Simultaneous Localization and Mapping, drone, autonomous, Angle of Arrival, PID feedback-loop**

I. Introduction

The potential for autonomous vehicles is extreme, ranging from improvements in public safety to economic growth. Google, Tesla, Amazon, and Uber are all developing autonomous solutions for many applications. Autonomous cars have the potential to increase road safety, lower transportation costs, increase economic growth, and decrease commute times. Uber could utilize autonomous vehicles to introduce lower fares. Autonomous drones have uses in delivery, tracking, localization, security. Amazon could use autonomous drones for instant local delivery or within a distribution warehouse. The proposed project combines existing technology with the capabilities of the latest Bluetooth standard to develop an autonomous drone which uses a novel method to locate objects. The project scope includes Bluetooth communication, wireless communications, drones, motors, motor control, embedded systems, collision detection sensors, autonomous flight control systems, and more.

Other autonomous drone systems have been built. The military uses unmanned aerial vehicles. Commercial drones allow users to initiate autonomous aerial maneuvers. The footage focused Skydio R1 performs autonomous aerial maneuvers which yield interesting video recording effects [1]. Amazon is also working on automated drone delivery systems which they claim will deliver packages in thirty minutes or less [1].

Similarly, the field of localization is receiving a lot of attention because of the proliferation of smart home technology. Automation systems can benefit greatly from the ability to know where you are. For instance, lighting systems could be designed to turn on lights only for the room you are in. Security systems could be set to automatically arm when you leave your house. Current localization techniques include geofencing and beacon-based positioning systems. However, these struggle with inefficiency and inaccuracy. The introduction of the Bluetooth 5.1 standard allows for sub-1m localization, promising to improve the existing systems [4].

This project encompasses the concepts of autonomous flight, localization, and 3D mapping to develop a drone which can locate an object using a Bluetooth-based positioning system. The goal is to provide a basis for accurate, efficient indoor positions systems that can be used for smart home automation as well as a variety of other applications.

The proposal will address the project in the following order: Section II will provide background information on the areas of study involved in the project; Section III will discuss the proposed body of work; Section IV will address important technical considerations and detail many of the chosen components; Section V will give a list of all components used in the project; Section VII will propose a timeline for the project; and, Section VIII will summarize the proposal.

II. Background

This project involves three major fields of study: autonomous flight, localization using Angle of Arrival (AoA) estimation, and visual simultaneous localization and mapping (VSLAM).

A. Autonomous Flight

Autonomous flight will be controlled through software flashed to a microcontroller flight board attached to the drone. The firmware will send signals to the electronic speed controllers (ESC), which will drive the motors. The motors turn the propellers that produce lift and move the drone in different directions. Autonomous flight allows the drone to hover and move stably without the need of a human pilot. The spin pattern of the propellers is shown in Fig 1. The firmware will eventually be controlled by pathfinding algorithms. Path finding algorithms will control where the drone moves to, such as to a beacon. Proximity sensors will be used to gather information on the drones environment for the path finding software.

Flight stabilization is achieved using an inertial measurement unit (IMU). An IMU generally consists of an accelerometer, gyroscope, and barometric pressure sensors. The sensors allow the MCU to track all the parameters necessary to stably fly the drone.

B. AoA-Based Direction Finding

Angle of Arrival (AoA) estimation is commonly used in signal direction finding. It is a method of determining the angle

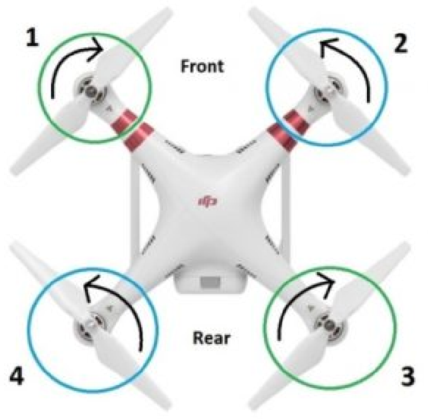

Fig. 1. Propeller spin patterns

at which the transmitted signal arrives at the receiver. It utilizes a single transmitter (TX) and a receiver (RX) consisting of multiple antennae, as shown in Fig 2.

The receiver requires multiple antennae to simultaneously read multiple phase samples. It computes the phase differences in the samples to estimate the signal direction using an algorithm such as multiple signal classification. Techniques must be applied to eliminate the effects of multipathing. A 3D array of antennae allows for true 3D measurements of the azimuth and elevation angles [4]. We will use AoA estimation, added to the Bluetooth 5.1 standard, as input to our path finding system.

AoA estimation has been added to the Bluetooth 5.1 standard, so that will be medium of choice. The receiver will utilize a Bluetooth module capable of AoA and an antenna array to accomplish the direction finding.

C. Collision Detection

Our priority is to implement autonomous flying and beacon tracking in an open space, such as outdoors or large open indoor space. We would also like to add collision detection

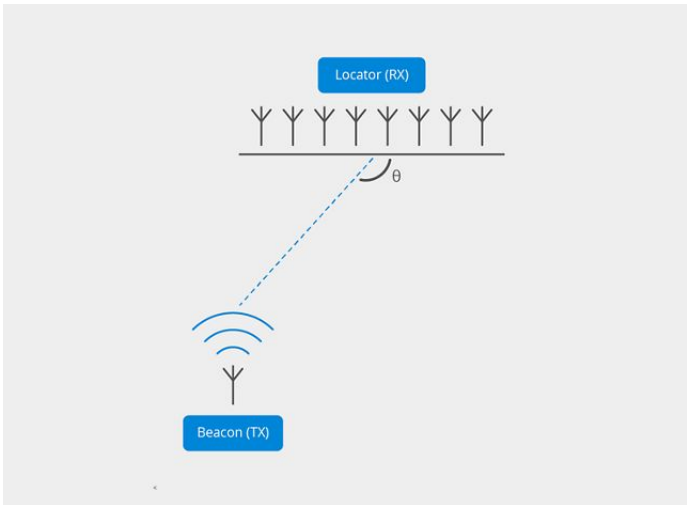

Fig. 2. Angle of Arrival TX/RX configuration. Image from SiLabs [4]

functionality to our drone. A drone with collision detection would be able to operate in smaller rooms where it might bump into walls, a ceiling, tables, light fixtures, and more. We will add collision detection sensors to the drone.

We could use ultrasonic, infrared, camera, or light detection and ranging (LIDAR) sensors for collision detection. The sensor or sensors we use will have to be effective in quickly detecting objects to allow time for the drone to prevent collisions. The listed sensors vary in effectiveness and cost, which will factor into the design of the system.

D. Pathfinding

Information from the Bluetooth and collision detection sensors will provide data to the pathfinding system. The pathfinding system will consist of search algorithms that compute a path based off sensor data. We will try to implement the pathfinding system on the drone's MCU, but we may also offload this computing task to an external machine. If we offload the computation, we will need a communication channel between the drone's MCU and the external machine.

E. Visual Simultaneous Localization and Mapping

Visual Simultaneous Localization and Mapping (VSLAM) allows for the simultaneous calculation of the cameras position and the creation of an environment map in the form of a point cloud, such as the one in Fig 3. There are five modules which make up vSLAM: initialization, tracking, mapping, relocalization, and global map optimization [6]. Initially, the robot constructs a portion of the environment to determine an initial map. Tracking and mapping can then be performed in real-time by comparing images seen by a depth camera. Relocalization refers to the ability of the robot to determine its localization when a swift movement would cause it to lose its position. Errors are reduced by the process of global map optimization [6].

III. Implementation

Our project consisted of building a quad-copter from specifically selected parts to fit our specifications. The drone motors were controller by a flight controller running the Betaflight software. The project involved the selection of a control unit to run our flight and visual programs that could connect the rest of the equipment. Our flight controller would not directly control the motors, but interface with them via the flight controller running Betaflight. The flight controller was connected to the motors using ESC's. A custom SBUS client had to be developed to interface with the flight controller running Betaflight.

Software had to be developed to connect to and read data properly from the time-of-flight distance sensor. Software had to be developed to connect to and read from the camera sensor. To house the drone equipment, a case was 3d printed which required research into the 3d printing process and the design of a model. An Android application was developed to add additional wireless control to the drone, and specifically to initialize the main flight routine. The team developed a multi-threaded software to run various programs in parallel, such

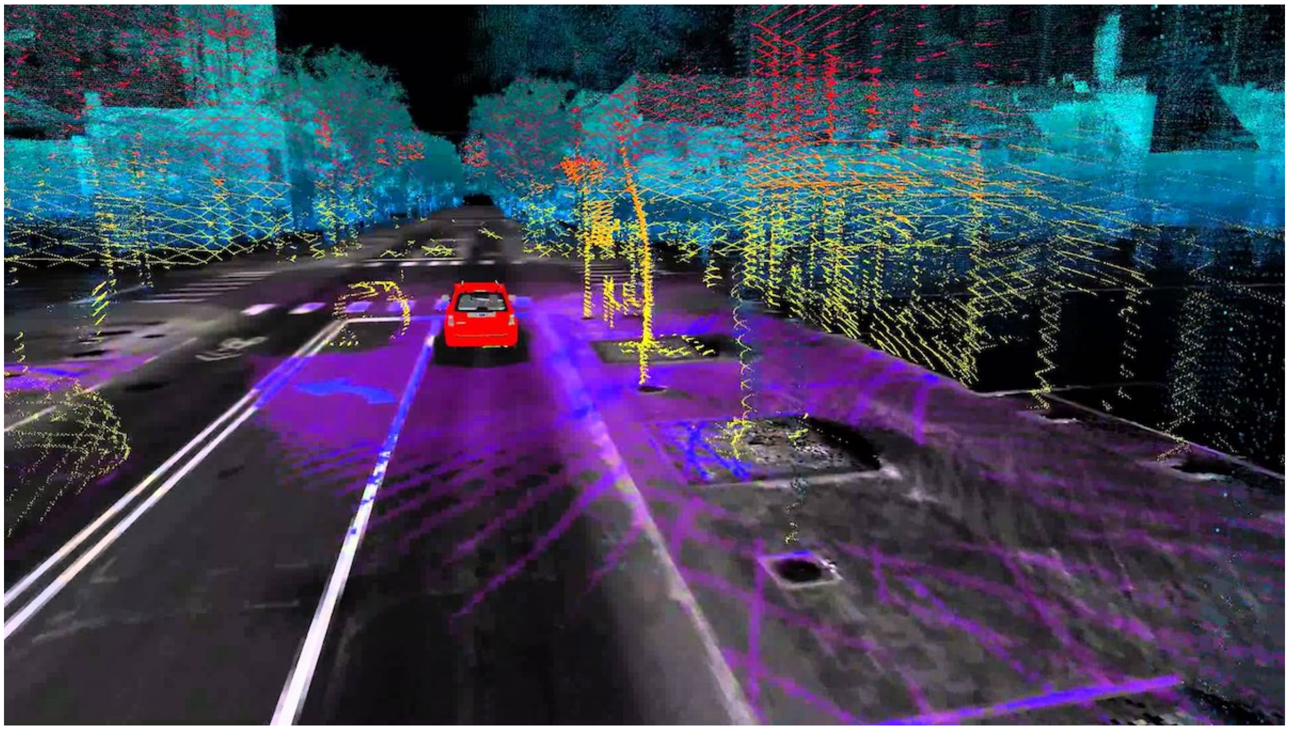

Fig. 3. Direct SLAM point cloud. Image from Imperial College London.

as gathering sensor data, performing calculations on that data, and using those calculations to perform some action. VSLAM was researched and image data manipulation was developed, such as image registration and point-cloud creation.

A. *Quadcopter & Flight Controller*

For our drone frame we used the iFlight IX5 V3 Hybrid FPV Racing Frame. We chose the CL Racing F45 Flight Controller because we could load Betaflight onto it. At the beginning of the project we experimented with the Betaflight code and attempted to modify it directly to load software onto the flight controller. This proved to be difficult and we decided it would be easier to develop on a separate board. We would have had to do this anyways to communicate with the Intel camera, but we could have continued to try to implement custom flight control on the flight controller itself.

We chose 5x4.5 inch propellers and the 2207-2300 KV-G Champion Series Gold Edition Motor which were chosen to provide enough lift for the drone and all the equipment. We used the Turnigy 2200mAh 3S 25C Lipo Pack to power the drone and flight controller.

B. *Jetson Nano Setup & Configuration*

One of our first hardware picks was the Intel Realsense D435i Camera. The camera uses a USB 3 connection. This constraint narrowed our options for our board. We chose the NVIDIA Jetson Nano Developer Kit because of its USB 3 ports and GPU.

The Jetson Nano Developer Kit is a small computer board made by NVIDIA. Is is run using an Ubuntu image that was flashed onto an sd card. All of the computing for the project is written in C++ and compiled and executed on the board.

To communicate with the board, a Wi-Fi card was installed into the board's M.2 Key E slot. The chosen Wi-Fi card is the Intel Dual Band Wireless-AC 7265 802.11ac, Dual Band, 2x2 Wi-Fi + Bluetooth 4.0 (7265NGW). This card was chosen for probable Bluetooth integration with the Jetson Nano, but the Bluetooth was not implemented in the final design due to lack of support in the market from being a new standard. An IPEX MHF4 Antenna was installed on the Wi-Fi card to enable wireless communication. The Jetson Nano then had to be configured to create a wireless hot-spot, which allowed the team to access the board through ssh.

When the board was connected to a wired network, remote access to the board was also made possible through ssh through port forwarding. This was useful in the early stages of the project to enable the team to work remotely, but became less useful near the end of the project when testing flight was necessary, and a wired connection could not be used.

The Jetson Nano interfaced wirelessly to an Android application and computer program by being set as a Wi-Fi hotspot. The computer and phone would need to be connected to the Wi-Fi, then would communicate to our program on the Jetson Nano via an IP port to send messages.

When initially attempting to read data from the Intel Realsense D435i Camera, we ran into a problem where we were unable to properly read IMU frames from the camera. After much researching on how to connect the Jetson Nano to the Intel Realsense, we saw many people having issues connecting some of the Intel cameras specifically to the Jetson Nano. We attempted one solution which was compiling and applying kernel patches to successfully communicate with the Jetson

Nano.

C. SBUS Protocol

The communication between the Jetson Nano and the flight controller was done via a UART connection. Normally, the flight controller is controlled using a radio remote. Radio communication works by connecting a radio receiver module to the flight controller. The radio receiver must be paired to the remote control. Communication from the radio receiver to the flight controller is done via a serial UART connection as the underlying protocol. It is a 100,000 baud rate, 8E2 configuration, and inverted logic connection. An actual SBUS packet consists of twenty-five bytes which sends the values for sixteen eleven-bit ”channels” [5].

For the flight controller, some channels were configured to set the throttle, roll, pitch, and yaw values. Additional channel values were sent to control the ”arm” mode of the flight controller which enabled the drone to drive the motors using the direction values received. Using a protocol guide, we developed routines to take an array of integers used to represent each channel value we wanted to send and wrap those values into the corresponding bytes and order needed to send to the flight controller. One of the encoding steps we had to implement involved packaging channel values of eleven bits into eight bit packets [5].

To configure the Jetson Nano for this serial connection we used the termios and termios2 structs. These allowed us to set the correct serial configuration and non-standard baud rate. We did not invert the serial values because we found a configuration setting on the flight controller that we were able to set via the Betaflight console and loading the updated setting onto the flight controller board. Initially we used a USB-to-serial cable, but later transitioned to using a built in serial pin on board the Jetson Nano. We were able to access this pin in the same way we did the USB cable, by reading from the corresponding ”/dev/ttySx” file. In addition, we also implemented various helper routines to easily change the values of flight directions, such as pitch and throttle.

D. Time-of-Flight Sensor

We used the TFMini Micro LiDAR Module to detect the distance from the drone to the ground. This distance sensor is actually a time-of-flight laser sensor. Initially we decided to use the I2C protocol to communicate from the TFMini to the Jetson Nano. The Jetson Nano has two pairs of SDA/SCL I2C pins. We began to implement the logic to connect to the sensor, but were unable to ever connect and get proper data. We are unsure whether this was a I2C implementation or other issue.

Next we attempted to the UART version of the TFMini. Initially we had various issues with the Jetson Nano itself when connecting the sensor. For example, sometimes the Jetson Nano would not boot or Wi-Fi would not turn on. We are unsure of the exact issue, but it was possibly a power problem. We then tried connecting to a different serial pin on the Jetson Nano and that was more successful. Looking back, while testing we had intermittent Jetson Nano problems where its Wi-Fi would not activate. Perhaps the sensor was a contributing factor.

The UART TFMini sends data in a series of bytes such as distance low byte and distance high byte. These values need to be kept track of and added together to obtain the full value. In addition, the TFMini sends a checksum value to validate the received data and signal strength data [2]. For this UART connection we used the LibSerial open source library to configure the serial port, but developed all the code to read, parse, and package the individual byte data from the TFMini ourselves [3].

At first we had a problem where we would receive the same data from the sensor regardless of the height. We also developed routines to flush the serial port and logic to execute re-reads when the sensor could not be properly read from to fix this problem.

E. Realsense D435i Camera

The Intel Realsense D435i is a stereo camera which provides color and depth frames through a USB-C interface. The camera also contains an internal inertial measurement unit having an accelerometer and a gyroscope. Intel provides a library, librealsense, which has an API for getting frames from the camera and manipulating them as necessary. Librealsense is published in an apt repository, so installation on the Nano was fairly straightforward. However, a kernel patch and compiling was needed to add compatibility for the D435i.

Initial testing of the camera was done using an application provided by Intel called realsense-viewer. Realsense-viewer allowed us to view each type of frame, adjust the resolution, and familiarize ourselves with the camera in general. From there, we implemented a series of small programs to perform basic tasks. Those programs formed the basis for what we later did for the 3D mapping.

F. 3D-Printed Payload

We used Autodesk Inventor to design the box which carried the payload for the drone. It was designed in two pieces: a lid and a base. It was designed in this way to allow for easy removal of the base when access to the components inside was needed.

The lid was designed such that four screw holes were used to attach the lid to the bottom of the drone, in the center. The base was then attached to the lid at the corners. The base was designed to provide sufficient support to the sensors, batteries, and the Jetson Nano so as to not allow damage.

The box and base were 3D printed out of PETG using the Lulzbot Taz 6 printer in the Senior Hardward Lab. The first version of the box took roughly 36 hours to print. Subsequent versions could be completed in just over 12 hours.

Using a 3D printed box in this manner allowed for easy prototyping. However, because the boxes were easily broken, considerable delays were suffered after each crash.

G. Mobile Phone Application

An Android app was developed to start and stop the main program run on the Jetson Nano board. The intended purpose of the app is to give the user a way to switch the program on and off without needing to ssh into the board to start and stop the program. We also had the idea of a program which would run as an independent service in Ubuntu Linux, and the mobile application would then communicate with this service to start the quadcopter.

As a precautionary measure, to mitigate the risk of losing the all components in a crash, the program was not run as an independent service so that the quadcopter would stop running if it lost connection with the ssh session running the main program. The mobile application, however, still served as the starting and ending signals for the end user.

The program was written and compiled using Android Studio. The main buttons that were used are the following:

- CONNECT TO JETSON - used to establish a socket connection with the main program running on the Jetson Nano board
- ARM - used to ”arm” the quadcopter, which is a necessary step in enabling flight
- START - used to start the quadcopter’s flight routine
- FORWARD - used to change the state of the quadcopter to flight
- REVERSE - used to tell the quadcopter to decrease throttle and land

Signals were sent to the Jetson Nano with each button press over an established socket connection and using the JSON message format. The phone would need to be connected to the Jetson Nano’s wifi.

H. Drone Flight Code

The control software was based on a multi-threaded model using the Boost thread libraries. The main program thread initializes necessary interfaces, creates several worker threads, and then runs the TCP server, which blocks execution.

Angle calculation, hover stabilization, frame polling, throttle control, and SBUS communication were all done on separate threads. Boost mutexes were used to synchronize the data.

1) WiFi to Android Application: The android app connects via Wi-Fi to the Jetson Nano, and establishes a socket connection with the main server code. It then sends messages to the main server code in the JSON format. The main server code uses a JSON library to decipher the messages and retrieve the message type in order to change it’s states.

2) Communication to Betaflight Flight Controller: The multi-threaded program contained one thread to send data to the flight controller via serial connection. This thread would read data under a mutex system and send that data to the flight controller. The thread would send channel values to control yaw, pitch, roll, throttle, and the ”arm” value.

3) Drone Orientation: The orientation of the camera was derived using the internal IMU. X and Z angles are initialized using the accelerometer. The Y angle is set to π by default. For each frame received from the camera, the orientations are updated using the gyroscope. Accelerometer input is used to filter noise from the gyroscope and smooth the angle estimations.

4) Drone Height: Another thread in our flight control program is the height control. The height thread would set up the serial connection to the TFMini sensor and begin a loop to continuously read data from the UART connection. This would read data from the distance sensor pointed at the ground to get the drone height from the ground. The thread would then take the raw TFMini data and parse it correctly to produce distance and signal strength values. The thread uses a state machine controlled with flags to keep track of byte order. The final values would then be saved in global variables. Locks were used to safely read and write to the height variables.

To control the height of the drone, we used the height data from the TFMini to control the throttle values we would send to the flight controller. We initially implemented a constant value feedback loop: while the height was less than the desired height increase throttle and vice versa. We later implemented a PID control for the height. The height sensor has a limitation of only producing valid data from a minimum of thirty centimeters. The minimum height changed the constraint for our set point height.

We also implemented a landing routine once the drone reached a hard-coded height, though located in another portion of the program.

5) Hover Stabilization: PID controllers were implemented to control the pitch and roll of the drone. We experimented with various methods for hover stabilization. The approach we ended with was setting a constant set point angle for the drone’s pitch and roll angles. The idea was finding an angle values, which were calculated from sensors on the Intel camera, which would keep the drone steady.

The program and drone then would correct until we stabilized at those values. Additionally we experimented with limiting the range that the values for pitch, roll, and throttle could reach. The value limits were extremely helpful in fine tuning the control of the drone and avoid huge swings of values that could completely knock the drone off course. We also ran into the issue of limiting those values so much the drone would reach a maximum and still be unable to physically correct. We had to experiment with these values.

We had many issues tuning our PID system. For example, we ran into an issue that when the drone was on the ground, an error would build up in certain angle direction. With the drone tilted in that direction, throttle would increase and the drone would take off in that direction. Due to the constraints of our space, sometimes the drone would not have the space and time to correct in another direction.

We also struggled with the drone correcting quick enough. While we disagreed whether this issue was due to the camera, some of the group believed the drone angles calculated from data received from the camera were delayed which means the drone would perform feedback based off of an incorrect angle. The drone orientation that lead to a hover was determined experimentally and used as a set point. Trial and error gave proper values for the gain of each term in the PID controller.

Researching PID feedback loops, implementing, and tuning was were we spent many nights working on.

6) Flight and Landing Routine: We developed a flight routine in our program. Once the drone had reached a hard-coded height, from forty to sixty centimeters, we changed the set-point angle. The program would change the angle for the drone to tilt slightly forward and produce forward motion.

The team experimented with various techniques to land the drone. Routines were triggered based off when the drone had reached and certain height and a time stamp. After the drone hit the target height a time stamp would be recorded. The flight control program would check that time stamp against the current time and would modify the drone throttle. For example, 4 seconds after the drone reached sixty centimeters, the load would start to land.

One routine we implemented continuously decreased the throttle of the drone until it reached a floor value. Another routine did the same, but had an additional timer, which after it would expire, the throttle would be decreased even more until reaching zero. In the end we changed the routine to a routine that after the drone reached a certain height and a constant amount of seconds passed, the throttle would be decreased to a constant value below the value needed to maintain height. The throttle at that value would make the drone descend.

I. Mapping

It is possible to create a 3D point cloud by stitching together multiple frames taken with a stereo camera. This is done using an algorithm called iterative closest point, or ICP.

First, depth frames are taken using the Realsense D435i camera. These frames are converted in a set of 3D points using a library called Point Cloud Library, or PCL. PCL provides a large set of routines for performing calculations on point clouds.

Once the point clouds have been obtained, they are filtered using a Voxel Grid. The Voxel Grid filter takes all of the points within a cube of a given size and averages them into a single point. This effectively down-samples the point cloud so that calculations can be done more quickly.

After the filter has been applied, ICP is run on the two frames. ICP first determines a set of critical points in the overlapping frames. These are points by which the frames should be aligned. Typically, they are a feature such as window or door, but the selection is done behind the scenes. Once critical points have been obtained, a rigid transformation is calculated over a number of iterations until a transformation is obtained which brings the frames close enough within a given tolerance.

Once the transformation is obtained, it is applied to one of the frames to bring it into the coordinate system of the other. The point clouds can simply be appended to create the larger image.

We were able to get this process to work to great effect. However, in the beginning we were seeing odd warping of the images. It was discovered that ICP is effective only with transformations which are relatively small. If the difference between frames is too large, the transformation returned by ICP is deficient. As long as we are sure to take consecutive frames which are very close to each other, the algorithm runs quite well. Any further distortion is caused by limitations in the operation of the D435i camera.

IV. Project Completion

There were three components proposed initially: autonomous drone flight, simultaneous localization and mapping, and Bluetooth-based tracking. We were able to make considerable progress on the first two objectives.

We implemented a multi-threaded C++ application to interface with various sensor, execute computations on that data, and modify drone flight parameters to autonomously control the drone. The research and implementation of autonomous flight took most of our time. Autonomous flight involved setting up various components, interfacing with various sensor, and creating a complex program to integrate it all. Many problems were encountered here. We were unable to implement some features such as collision detection and path finding due to problems with computing camera data quick enough.

The Android application was completed to control the drone.

We were able to implement the VSLAM but never integrated it with the drone. The time that it took to capture and process each frame would slow down the drone and make it impossible to fly. In addition, because of how much the drone would move between capturing each frame the Iterative Closest Point (ICP) method would be unable to align two frames with each other. Each frame could only be around a degree change from its previous to work. We split the code to complete this into three portions. The first portion is the capture portion which just continuously takes frames and then saves them all at the end. The second portion is the align portion which loops through all of the frames and aligns frames with the frame taken before it until they are stitched together. The third portion is just a display function which visualizes any PCL cloud for the user.

3d printing was researched and used to print a case to house our components from a model.

Our Bluetooth objective was dropped due to lack of Bluetooth 5.1 sensor supply.

V. Project Retrospective

The Bluetooth tracking would have been based on the Bluetooth 5.1 standard which was finalized in the spring of 2019. Unfortunately, the development timeline for commercially available products lagged too far behind. We are still unable to find the modules we would have needed to perform tracking.

Instead of directly controlling the drone's motors using pulse-width-modulation, we opted to use Betaflight, an open-source drone control software. Betaflight abstracted the fine motor control we would have had if we were connected to the ESC's. While Betaflight may have gotten the team to begin control the motors more quickly, we lost the fine motor control we would have had otherwise.

We use the Intel Realsense D435i camera to detect changes in angle. From the camera we were able to read accelerometer and gryoscope data. Unfortunately this data would fluctuate so much it made it more difficult to control the position of the drone.

The stability of the drone was controlled using PID feedback loops to change the values sent to the flight controller to control yaw, pitch, roll, and throttle. We experimented with various PID values to better stabilize the drone. If we had more time, the group could have additionally spent more time analyzing this stability problem to better calculate PID values and even use more sophisticated techniques to analyze values and their results.

If we were to create this drone again we would add an Intel Realsense T265 camera which has the ability to track location in the x,y, and z directions from its starting point. This would have made drone stabilization easier because you could use the PID feedback loops on the locations instead of angles which do not always act in the desired way. This would also have made VSLAM more possible because instead of relying on the ICP method, which requires that two frames are very close to each other already, we could have used the locations from the T265 and the angles from the D435i to complete the transforms from one frame to the next. This would eliminate the need for the camera to take frames and process all of the pixels in them which took an extremely long time even running on a laptop.

VI. Conclusion

Our team was able to construct a drone from curated parts to fit our requirements and specifications. We purchased drone motors, propellers, and frames specifically for our estimated weight. We were able to select equipment such as sensors to also fit our requirements, specifically a time-of-flight distance sensor, camera sensor, and flight controller peripheral. The team assembled the drone and soldered ESC's, motors, jumper wires, and connectors to the flight controller.

To physically house the equipment, we researched, designed, and printed various iterations of plastic cases. Using this equipment, we were able to construct a drone capable of autonomous flight and control by implementing a multithreaded application in C++. This program interacted with various peripherals such as distance, camera, IMU, and flight control sensors and boards. Data from peripherals had to be read and parsed correctly to be used. For example, we had to implement our own SBUS client to correctly send data to the flight controller. We implemented a state machine to read data from the time-of-flight sensor.

The program performed various PID feedback loops to physically control control drone. The team researched, designed, and 3d printed a custom made case to hold the equipment. We were also able to implement various programs using the Intel Realsense D435i camera such as composing, or registering, camera images and generating point clouds from them.

References

[1] Amazon.com, Inc. Amazon Prime Air. https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011. Online. accessed Mar 2019.

[2] Benewake (Beijing) Co. Ltd. TFmini Infrared Module Specification. https://cdn.sparkfun.com/assets/5/e/4/7/b/benewake-tfmini-datasheet.pdf.

[3] crayzeewulf. Libserial. https://github.com/crayzeewulf/libserial.

[4] S. Lehtimaki. *Bluetooth Angle Estimation for Real-Time Locationing*. Silicon Labs, 2018.

[5] Robotics and Perception Group. SBUS Protocol. https://github.com/uzh-rpg/rpg_quadrotor_control/wiki/SBUS-Protocol.

[6] H. Uchiyama T. Taketomi and S. Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 2017.