Autonomous Aerial Drone with Infrared Depth Tracking

Rishi Jain, Caleb Jones, Ryan Lucas, and Hamza Siddiqui

DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING, UNIVERSITY OF CENTRAL FLORIDA, ORLANDO, FLORIDA, 32816-2450

***ABSTRACT* — Autonomous drones have broad applications including but not limited to: defense, search and rescue, firefighting, and package delivery. Despite the different applications that these drones may perform, they have similar underlying components and software architectures. This paper provides an overview on the processes and methodologies used to implement an autonomous aerial drone, also known as a UAV, for our senior design project. This project identifies known obstacles and attempts to maneuver around them in specific patterns. Moreover, the project incorporates sound localization in order to identify the approximate location of an audio source.**

***Index Terms* — Aircraft Navigation, Autonomous Systems, Image Recognition, Machine Learning, Sensor Systems.**

I. INTRODUCTION

The purpose of this project is to create an autonomous aerial drone in a multidisciplinary team composed of electrical/computer engineering students as well as mechanical/aerospace engineering students. The original objectives of this project were provided by our project sponsor, Lockheed Martin. The main objective was to create an autonomous drone to navigate through an obstacle course consisting of a ring, a pylon, a double-pylon, and an acoustic waypoint.. This obstacle course would have been set up indoors at Lockheed's indoor lab. Our team's drone's would have competed against other drones designed by other teams to determine which drone could navigate the most obstacles autonomously through the course. In addition to navigating the obstacles, the drone was also to avoid being brought down by a drone "mine" designed by another team to stop the drone from completing the course. The demonstration was initially planned to be a competition against the other drone teams, with the goal of accumulating the most points. As a part of this challenge, we were allocated a strict budget of $1100 for the final build price of our drone and an additional $550 for prototyping.

After the pandemic broke out and it was determined the drone competition was not going to occur as planned, the project requirements were reduced to demonstrating autonomous around two of the obstacles: a ring and a pylon. This report details our progress to accomplishing these requirements.

II. HARDWARE COMPONENTS

The completed drone project consists of various individual components used to realize the finished product. This section provides an overview on the major components selected, including a brief analysis on the decision making process.

A. Flight Controller

The flight controller is responsible for controlling the movement of the drone by adjusting the power delivered to each motor. Flight controllers can measure the level and speed of the drone, and use that information to correct the orientation of the drone during flight. The selected flight controller for this project is the Readytosky Pixhawk. The reason why this flight controller was selected was because in addition to the basic gyroscope and accelerometer sensors that all flight controllers have, this one also includes a barometer, magnetometer, and inertial measurement unit (IMU) which provides more accurate and consistent flight performance. Additionally, the Pixhawk is 32-bit, has I2C and SPI interfaces, and has the ability to run either the PX4 or ArduPilot flight stack. We utilized the expandability of this platform to test the drone using both a PX4Flow Optical Flow Sensor and a HereFlow Optical Flow/Lidar sensor.

B. Camera

For our camera, we were looking at a few different options. One option was to use a single, basic RGB camera. This would provide us video data from which we could feed into an object detection algorithm in which to determine the location of the objects. However, we determined with this kind of set up it would be difficult to determine the distance to most objects. A solution that we came up with was to use two RGB cameras and use the relative difference in pixels between the two images to determine the distance to objects on the computer. The downside to this approach is the large amount of processing power it would take to determine distances on a computer that is already running an computationally expensive object detection algorithm.

Therefore, instead of using two RGB cameras, we decided to use an Intel Realsense D435 depth camera. This camera uses two infrared cameras to produce a depth image that is calculated on a processor located in the camera. The depth image along with an RGB image from

an RGB camera also located on the D435 is sent to the drone computer via USB. The drone computer then uses the RGB image for object recognition while the depth image was intended to be used for determining the distance to the object. Figure 1 shows a 3D mapped image processed by the camera.

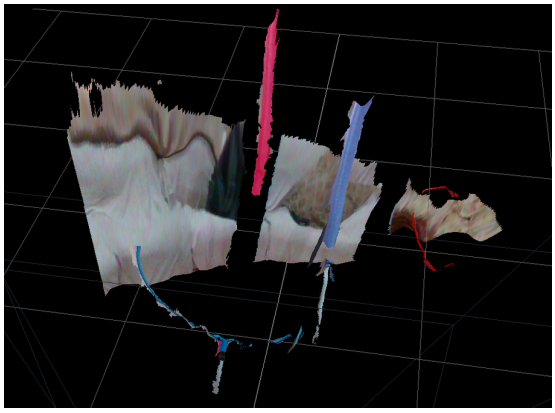

Figure 1: 3D mapped image of pylons and rings viewed from the positive z-axis

C. Microphone

To detect the acoustic waypoint we utilized the Mic Array v2.0, which uses four ST MP34DT01TR-M digital microphones to triangulate sound and filter out different sounds into up to six channels. Sound triangulation is done with time of arrival for a sound to each of the four microphones, resulting in an angle which can be used to determine the location of the acoustic waypoint. Volume intensity can be used to determine the distance from the drone to the acoustic waypoint.

Sound filtering is done with the onboard XMOS XVF-3000 chip, with the use of Acoustic Echo Cancelation (AEC) which is designed to remove echoes, reverberation, and unwanted added. AEC utilizes an adaptive filter which samples the direct audio with the echoed audio to result in one clean audio reading, which is then further filtered with residual echo suppression and noise reduction (Figure 2).

The onboard XMOS XVF-3000 chip also has a configuration file which allows for adjusted parameters for how many channels that can be filtered, as well as the frequency range that will be recorded by the microphones. These parameters can be adjusted because the Mic Array v2.0 utilizes digital microphones which are compatible with a more advanced stereo sound processor codec such as the XMOS.

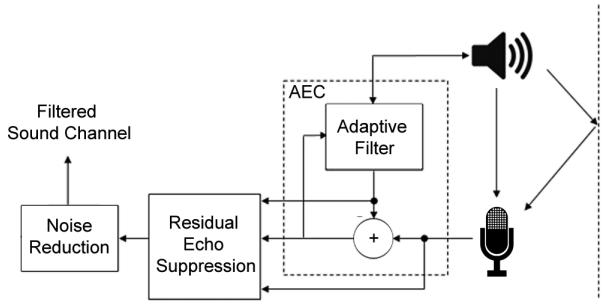

Figure 2: Microphone array acoustic echo cancelation

D. Companion Computer

The computer of the drone is used to control the main functions of the drone. These functions include image recognition on frames received from the camera, sending commands to the flight controller, and sending video to the ground station. When choosing a computer to integrate into our system, we were looking for a computer that was powerful enough to run all of these systems simultaneously in addition to being relatively light and consuming relatively little power.

At first we considered using a Raspberry Pi 4 Model B. This computer has a 4-core 1.5 GHz processor with 4GB of memory. It is very light at 46 grams and uses up to 5W at max power. However, this computer had the downsides of having a GPU that is not well documented, likely not very powerful, and unable to take advantage of specific libraries designed for object detection models.

We instead decided to use a Nvidia Jetson Nano Dev Kit. This computer has a 4-core 1.42GHz processor with 4GB of memory. It is slightly heavier at 116 grams and uses up to 10W at max power, but it uses an Nvidia CUDA core GPU that is more broadly compatible with software that can run object detection algorithms. We believed this combination of CPU and GPU would provide enough computational power to run the object detection algorithm along with any other software we were running. We would also be able to experiment with Nvidia specific libraries for deep learning and computer vision. For this reason, we selected the Jetson Nano over the Raspberry Pi 4B.

E. Microcontroller

A microcontroller was selected to manage four laterally mounted ultrasonic range sensors, the HC-SR04. Utilizing a separate microcontroller (as opposed to incorporating this subsystem with the companion computer) allowed us to increase the modularity of the system. This highly modular design allowed us to increase the reliability of the subsystem, and allowed us to easily examine various design iterations. The chosen architecture was based on the Atmel ATMEGA328-PU processor which utilizes an

8-bit 20 MHz design. This integrated circuit (IC) is sufficient in driving the ultrasonic range sensors and relaying distance information to the companion computer via UART. Finally, this architecture was chosen due to its available documentation/online resources, as well as ease of use via the Arduino development platform.

F. Wi-Fi Module

The drone computer requires Wi-Fi capability in order to communicate with a ground control computer to send video data. The Jetson Nano does not come with a Wi-Fi module built in, so we decided to buy a Geekworm USB Wi-Fi adapter. We decided to go with a USB Wi-Fi module rather than a Wi-Fi card because a Wi-Fi card would have required installation of a separate antenna in order to operate properly. The USB module came with a small antenna and allowed for quicker integration with the rest of the system.

The Geekworm USB Wi-Fi module was specifically chosen because it uses USB 3.0 which is on the Jetson Nano, allowing for data speed of up to 4,800 Mbps, which is ten times faster than USB 2.0 ports. A USB Wi-Fi module with a larger antenna allows for a larger reception range, and the Geekworm USB Wi-Fi module uses a 5dBi RP-SMA which has an effective range of up to 1000 feet.

G. Battery

A Lithium Polymer (LiPo) battery will be used as the power source for the drone. It will be connected to the power distribution board (PDB), which will then distribute power to the rest of the components. The power requirements of all the components of the drone's system is listed in Table 1..

Table 1: Expected Power Draw of System Components

Component	Voltage	Current	Power
Jetson Nano	5V	2-4A	10-20W
Intel RealSense Depth Camera	1.8V	83mA	150mW
Readytosky PixHawk	0.3-3V	0.83-8.3A	0.249-24.9W
Cobra CM-2206/17 2400kV Motors	6-8V	0.97-1.06A	5.82-8.48W
Spedix GS30 32bit DShot 1200 30A ESC	8-25V	30A	240W
Hereflow Optical Flow and Lidar Sensor	5V	800mA	4W
Speed's ReSpeaker Mic Array v2.0	5V	180mA	0.9W
PCB with Distance/Proximity Sensors	5V	1mA	5mW

The battery used was a "Venom Fly" 14.8 volt, 3200 mAh, 4 cell, 330g LiPo battery. The battery was chosen based on two primary specifications; its total output voltage and weight.

The "Dynamite Reaction" and the Tattu batteries were taken into consideration. The "Dynamite Reaction" was a 5000 mAh, 11.1 volt, 3 cell, 204g battery; this battery's higher capacity and lighter weight were good specifications to consider. However, the output voltage of the battery was too low to provide adequate power of the drone's systems.

The "Tattu" was a 10000 mAh, 22.2 volt, 6 cell, 1400g battery; this battery's higher capacity and output voltage were good specifications to consider. However, the total weight of the battery was too high to provide adequate thrust for the drone's takeoff of maneuverability.

H. Universal Battery Eliminator Circuit

To power the Jetson Nano using power from the drone's power distribution board, the ZTW 6Amp universal battery eliminator circuit (UBEC) was used. The Jetson Nano is rated to take a DC power supply of 5V and 4Amps, and despite providing 5V and 5.5Amps via the ZTW 6Amp UBEC, it did not provide an adequate amount of power when the Jetson was running the computer vision algorithm, resulting in a shutdown of the CPU.

III. System Overview

This section will outline the physical connectivity and operation of the complete system.

A. Architecture

Figure 3 below provides a high level overview of how the components are connected in this project. In the figure, it can be seen the battery is connected to PDB, which is connected to the ESC and motor set, the flight controller, the CPU, the camera, the microphone, and the microcontroller and sensor set.

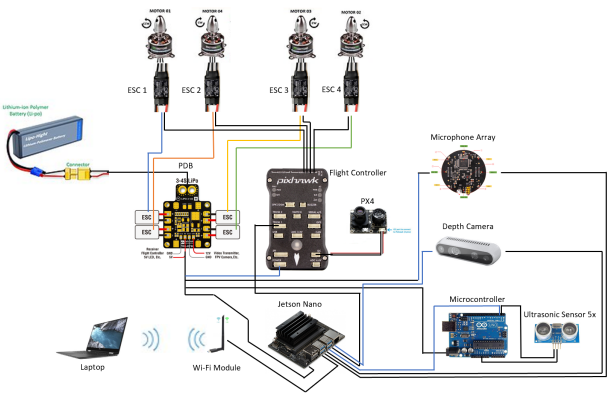

Figure 3: System component connection overview

As seen above, the drone has two primary systems: the flight controller and the Jetson Nano (companion computer). The flight controller manages all the flight controls of the drone, while the companion computer manages all the image and sound recognition, processing, and navigation instructions of the drone. The companion computer connects to the flight controller in place of a radio receiver, mimicking the signals used to fly a drone. It also provides the communication interface to control the

drone remotely. As the Jetson Nano is able to connect to a Wi-Fi network through the wireless adapter, a user is able to SSH into the drone and execute flight commands. Moreover, the user is able to launch the video feed and stream it over the network to a remote ground control station (in this case, a laptop).

B. Operation

Connecting the battery to the drone powers all of the system components immediately, however the computer takes about 45 seconds to boot up. Once the computer has booted, and the Jetson Nano connects to the known access point, a user can remotely SSH into the computer over the same network. Once logged in via SSH, the user can launch the software for autonomous functionality. To manually fly the drone, the remote control receiver must be inserted into the flight controller, replacing the Jetson Nano.

IV. SOFTWARE DESIGN AND INTEGRATION

The software for the project heavily revolves around the Robot Operating System (ROS). Figure 4 (below) outlines how the various software nodes interact with one another.

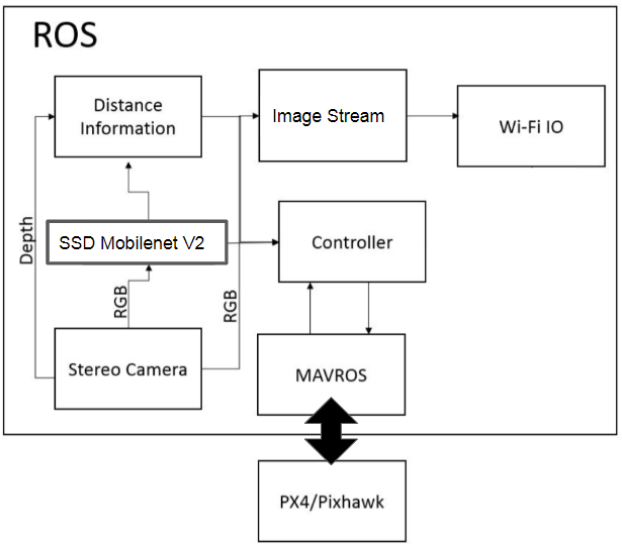

Figure 4: Software system diagram using ROS

A. Object Detection Model and Training

Object detection is used in our system to determine the location of obstacles compared to the drone. The decision of the type of model mainly focused on speed as extreme accuracy is not necessary to navigate through hoops and around pylons. Using data for models running on the Jetson Nano [4], we decided to go with a Single Shot Detector (SSD) using a Mobilenet V2 to identify objects for its speed. It would allow us to perform path corrections at a rate at at least 30 frames per second.

This model was trained using one of our personal computers with a GTX 980ti GPU using Tensorflow. Tensorflow was chosen due to its theoretical compatibility with the Jetson hardware. The images used for training were collected by us during our test sessions in the Lockheed Martin drone lab and outside, and these images were labeled using labelImg [1]. The configuration of our training was based on that Tensorflow researchers used for training an SSD for the COCO dataset [2].

Once the model was trained, the frozen model was transferred to the Jetson where the model was converted into a UFF file, a form usable by TensorRT. TensorRT was then used to optimize the model for use with the GPU located on the Jetson Nano. This process was difficult due to issues in compatibility between particular versions of Tensorflow and TensorRT. In this case, Tensorflow 14.0 was creating nodes in the graph that were unable to be properly interpreted by our version of TensorRT. We were able to resolve these issues after finding a source detailing how these issues could be resolved [3].

B. Robot Operating System

The Robot Operating System (ROS) is a set of message passing protocols typically used with autonomous systems. ROS allows for integration of different systems by standardizing messages including the passing of images from the camera node to the image processing node. Nodes can be established for various functions such as packaging sensor data and device control instructions.

We decided to go with ROS because of the large amount of resources that we could find a large amount of software that would allow for the quick integration of various components of our system.

C. Controller Node

The controller node is a ROS node designed to receive information from the object detection node and use that information to send commands to the flight controller via MAVROS. The controller node contains a control loop that switches between a set of modes. The main two modes are Autonomous Navigation (AutoNav) and Autonomous Maneuver (AutoMan).

While in AutoNav mode, the drone would attempt to navigate towards a spot in front of the obstacle. The drone does this using object data from the object detection node. Once the drone has navigated to a spot in front of the obstacle, the drone then switches into AutoMan mode. The drone then would attempt to navigate around the obstacle. In the case of rings, the drone would attempt to fly through them. In the case of pylons, the drone would make a loop around them. Once the drone has finished

AutoMan mode, the drone would be put into AutoNav mode again to look for another obstacle.

The drone would be able to be switched out of AutoNav and AutoMan modes by flicking a switch on the radio controller to throw the flight controller into a human operated mode. The drone computer will then notice the change in mode and stop sending commands to the flight controller.

D. Camera Node

The camera node we are using to interface with the Intel Realsense D435 is a node called realsense-ros created by Intel [5]. This node provides an RGB image and a depth image to the other nodes.

E. Object Detection Node

The object detection ROS node is created by Nvidia employees called ros_deep_learning [5] that was edited by us in order to accept an RGB image from the camera as opposed to the BGR image that the program expects. The object detection node subscribes to the RGB image output by camera node. The node then takes our custom model in UFF format, converts it to a form to be used by TensorRT (if it hasn't already from a previous run), and then uses that model to output bounding boxes of obstacles. These bounding boxes are labeled such that we can determine what kind of obstacle is detected. These bounding boxes would have been passed to the distance estimation node for use in determining distances to each object.

F. Distance Estimation Node

The distance estimation ROS node would take an input of a depth image from the camera node and the bounding boxes of objects from the object detection node to estimate the distance to each obstacle in the field of view of the camera. The closest obstacle to the drone is then determined, and the bounding box for that obstacle and distance to that obstacle would then have been passed to the drone controller node for navigation.

G. Microcontroller Software

The microcontroller software was written in embedded C, as available in the Arduino Integrated Development Environment. The microcontroller transmits sensor information over the Universal Asynchronous Receiver-Transmitter (UART) interface. On the other side of the communication link, a ROS node on the companion computer can read the serial communication and publish the data as necessary.

V. DRONE DESIGN

The drone was customly designed by the mechanical/aerospace engineering members of our team with the purpose of creating a highly modular design. The benefit to creating our own frame is that it can evolve to meet the changing needs of our project. Designing our own drone was also cost effective and allowed the team to utilize the project budget on better components. In particular, the drone was 3D printed using PET-G with custom mounts to securely house our components. We were able to quickly reprint parts of the drone if we needed to accomodate a new component or replace any broken parts. A model of drone is shown below in Figure 5:

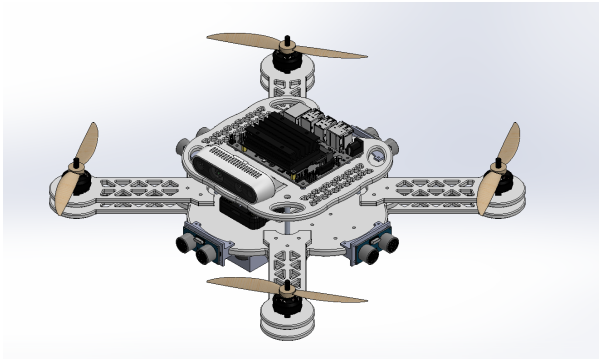

Figure 5: Custom design drone frame with components mounted

VI. RESULTS

Due to the pandemic canceling the trial runs at Lockheed Martin's drone facility, the school being shut down and denying access to the senior design lab, and generally being unable to congregate with the entire team due to stay-at-home orders, the ability to complete this project as originally planned was severely impacted. We did our best to pull together what we could in these circumstances.

A. Object Detection

We were able to train an object recognition model and successfully execute it on the drone's onboard computer. We created and utilized a database of approximately eight hundred images to create our model. We were able to correctly identify hoops and pylons. An example of this can be seen in Figure 6. As part of the project requirements, the object recognition system should be able to identify the type of obstacle detected, and the confidence level of the object detected, amongst other requirements. In figure 6, it can be seen the object recognition model was able to detect two types of obstacles; a ring and a single pylon. The ring was detected

with a confidence level of 83 percent, and the pylon was detected with a confidence level of 75.7 percent.

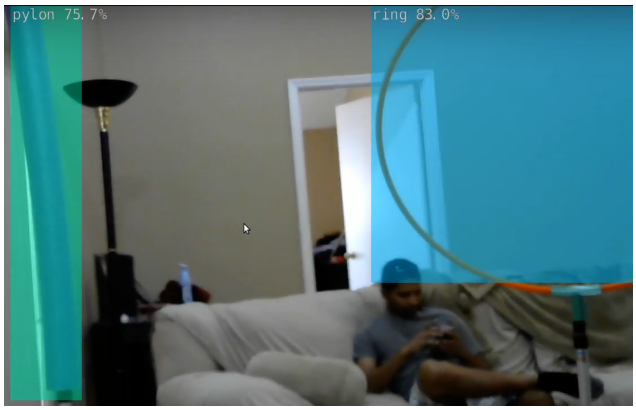

Figure 6: Object detection with confidence level

In addition to the visual identification, we were able to extract numerical coordinates for the identified object with respect to the rest of the captured frame.

B. Sound Triangulation

Using the previously mentioned microphone array, we were able to identify the direction that a sound was playing from. Primarily, we were able to extract the angles of a sound source in reference to the zero position of the microphone. With this information, we could determine which direction the drone needed to navigate to in order to complete the acoustic waypoint obstacle as part of the competition. An example of the microphone data is shown in Figure 7. In the figure, the angle of the sound source from the drone's current location can be seen. The angle is measured counterclockwise, taken from the horizontal front end of the drone.

The sound that the microphones were set to listen to any sound above 180Hz. The XMOS could limit its sound cutoff range to 80Hz, 125Hz, and 180Hz. This was the closest number to Lockheed's specification of close to 500Hz for the sound emitted from the acoustic waypoint. The 180Hz limit was enough to permit sounds from a high frequency hand bell, and exclude lower frequency sounds such as voices when testing for the direction of arrival for high frequency sounds. The frequency of the sound from the drone's propellers to where the mic array was placed underneath the drone, was below the 180Hz threshold.

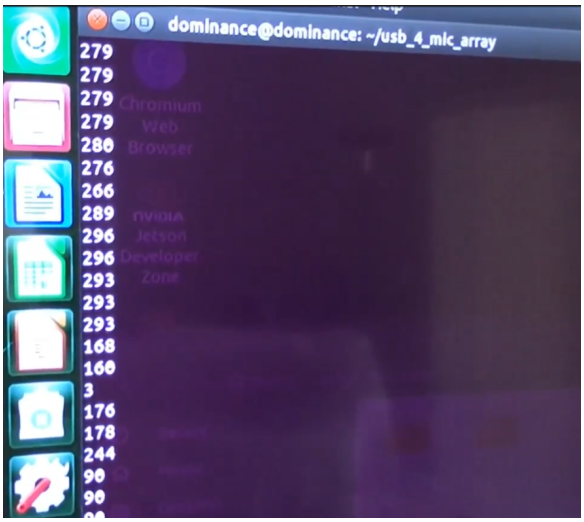

Figure 7: Angles of detected sound samples (with respect to a 0° starting position)

C. Printed Circuit Board & Microcontroller

We were able to design and prototype the PCB that would be able to manage the ultrasonic range sensors used for the measurements between the drone and potential obstacles. It is based on the Atmel ATMEGA328-PU integrated circuit and an Arduino Uno was used as the development board. While we originally wanted to incorporate a DC/DC power converter to serve as a voltage regulator for the various components in our system, the electronic speed controllers and motors draw upwards of 30A each. We were advised to instead use a power distribution board designed for drone applications, which we then utilized to power our components. The schematic for our circuit is shown in Figure 8.

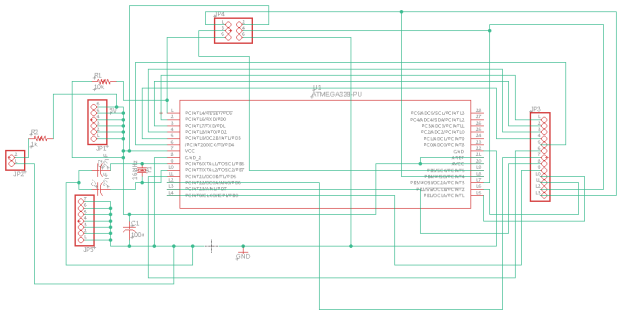

Figure 8: PCB schematic utilizing the Atmel ATMEGA328-PU

We tested the sensor measurements and verified the ability to communicate data between the microcontroller and Jetson Nano over UART to exchange data, as seen in Figure 9.

Though we were able to establish a proof of concept for this subsystem, we experienced unreliable performance

when using the fabricated PCB that we designed. We believe that due to tracing errors, some signals were interfering with each other and causing poor performance in relaying information back to the drone's companion computer. We would be able to confirm this hypothesis with the use of the university's lab equipment.

```
21:31:15.438 -> S1: 3; S2: 6; S3: 6; S4: 3; S5: 0
21:31:16.911 -> S1: 3; S2: 12; S3: 13; S4: 3; S5: 0
21:31:18.329 -> S1: 3; S2: 4; S3: 7; S4: 3; S5: 0
21:31:19.789 -> S1: 3; S2: 4; S3: 6; S4: 3; S5: 0
21:31:21.229 -> S1: 4; S2: 4; S3: 6; S4: 3; S5: 0
21:31:22.681 -> S1: 3; S2: 4; S3: 6; S4: 3; S5: 0
21:31:24.111 -> S1: 3; S2: 4; S3: 5; S4: 3; S5: 0
21:31:25.572 -> S1: 7; S2: 4; S3: 5; S4: 3; S5: 0
21:31:27.028 -> S1: 5; S2: 4; S3: 5; S4: 3; S5: 0
21:31:28.474 -> S1: 242; S2: 4; S3: 5; S4: 3; S5: 0
21:31:29.941 -> S1: 239; S2: 4; S3: 5; S4: 3; S5: 0
21:31:31.406 -> S1: 237; S2: 4; S3: 5; S4: 3; S5: 0
21:31:32.834 -> S1: 202; S2: 4; S3: 5; S4: 3; S5: 0
21:31:34.306 -> S1: 238; S2: 4; S3: 5; S4: 3; S5: 0
```

Figure 9: UART transmission of ultrasonic sensor data (measured in centimeters)

D. Power System

All of the drone's components were powered through the onboard battery. The drone was originally designed to fly for at least 10 minutes on a full charge of a battery. However, during testing we discovered that our battery was only able to provide about 6 minutes of flight with all components attached. After discussions with our team, we discovered that the power draw from our motors was greater than that was what was expected. We decided to upgrade our battery to a 6000mAh, which is almost double what we had before. We also decided to upgrade to larger motors because the additional battery weight would have put significant strain on our current motors. However, we were unable to order these batteries due to the lack of access to sponsor funds because of the pandemic.

During our testing with our drone computer connected to the battery, we noticed that we were having issues with the computer turning off automatically. When the drone was idle, we found that each component was stable and remained powered on. However, when we ran the computer vision algorithm, the CPU load would spike and the computer would shut off shortly thereafter. The power to the computer was being routed from the computer using a regulator rated for 5V at 6A. Since the computer is designed to run at 5V and pull at max 20W with all of the peripherals attached, we believe that the reason the computer kept shutting off was that the regulator that we bought wasn't able to provide the necessary amperage quickly enough when the computationally intensive object recognition started. The use of lab equipment such as an oscilloscope would have assisted us in helping to diagnose this as the certain cause. As it is, it seems to us that the regulators that we used either weren't properly rated or were not stable enough when a large amount of power is suddenly needed. If we had additional time, we would have ordered or designed another regulator that hopefully would provide enough current fast enough.

E. Distance Estimation

The initial plan for our distance estimation node was to use the bounding box from the vision algorithm and line that up with the depth image to determine the distance to each obstacle. However, due to time constraints that occurred partially due to the pandemic, we were unable to implement this node. Instead, we used object detection to determine if the drone was close enough to the obstacle to perform a maneuver. This was able to be done because instead of the large number of obstacles that could be in our field of view at the same time in the originally planned obstacle course, the drone only ever needed to navigate a single obstacle at a time.

F. Manual and Autonomous Flight

We were able to control the drone manually via an RC controller and achieve stable flight. With all the components attached to the drone, the drone had no problem sustaining lift.

Before implementing autonomous flight, we needed to ensure that the flight controller was able to hold the drone at a constant height and position. This position hold mode is crucial to autonomous flight as it holds the drone in a stationary position in a 3D plane. We first attempted to utilize a PX4Flow Optical Flow sensor to implement this feature. The optical flow camera would send data to the flight controller about how much the pixels in the frame of the camera have shifted. However, we found that the drone would drift in the air and occasionally drop out of the sky as it poorly attempted to correct itself. After doing considerable testing, we found that the PX4Flow was providing severely incorrect data to the flight controller, which we believed to be causing our issues. We believed initially that the issue could be due to the fact that this was not a new PX4Flow and was instead used in a previous drone project.

Next, we replaced the PX4Flow sensor with a HereFlow Optical Flow/Lidar sensor. Sensor was designed to provide the same basic functionality as the PX4Flow, but instead of a sonar sensor attached to determine height, a LiDAR sensor was used. Through testing, we found that the HereFlow appeared to give more accurate data than the PX4FLow, but the flight controller was still unable to properly hold position.

We believe there are two potential reasons for why this problem continues to occur. First, the drone was primarily being flown over the grass. It is possible that the grass that we were flying the drone above was simply too noisy to

accurately track movement using the camera. Our project was originally intended to be flown indoors, and it is possible that a more consistent floor pattern may have improved the performance of position hold. The second could be an issue with the accuracy of the height sensor. The height sensor is essential to determining how much a drone needs to adjust compared as pixels will appear to move faster when close to ground while the pixels appear to move slower when farther from the ground even when moving at the same speed. The LiDAR found on the HereFlow likely wasn't working as well outside in the open field as it was when we were testing it indoors. We initially believed that the degradation in performance wouldn't be significant enough at the heights we were flying at, but it may have been enough to cause issues. Although we had enough money remaining in our budget to order a replacement flight controller, the pandemic made it impossible to order and receive parts during the second half of the semester.

Without the ability for a drone to properly stabilize in the air, we felt uncomfortable running an autonomous flight algorithm. The autonomous algorithm that we intended to employ relied on fairly consistent stabilization by the drone to ensure that the drone would not drift significantly while carrying out movement commands based on the local coordinate system of the flight controller. Without the assurance of stable flight, flying via our autonomous algorithm posed both a risk to ourselves and to the hardware as a whole. Additionally, our drone power system was unable to support the power requirements of the Jetson Nano Dev Board, making testing navigation difficult while performing flight.

VII. CONSIDERATIONS FOR THE FUTURE

A major realization that the team had when working on this project was how much power the Jetson Nano seemed to consume. This not only reduced the operating time of the drone, but also gave us unreliable performance when CPU utilization spiked. The Raspberry Pi 4 Model B weights approximately 100 grams less by itself, does not require an external Wi-Fi dongle, and most importantly, has a power consumption of 5W (opposed to 10W on the Jetson Nano). We believe that the Raspberry Pi 4B would have sufficiently been able to run our object detection model, while reducing power draw and saving weight on the drone. Having the ability to use lab equipment to analyze the power draw would have provided insights as to why our power delivery was unsustainable. This may also allow us to sustain a minimum 10 minute operating time instead of the 6-7 minutes we estimated with the selected battery.

VIII. CONCLUSION

This project was able to demonstrate a competent design for an autonomous drone. It produced a successful custom object detection model to identify the provided obstacles. Although we were severely hindered in our ability to deliver a finished product due to the global pandemic, this project laid the framework to incorporate environmental information in an autonomous device.

ACKNOWLEDGEMENT

We would like to thank Lockheed Martin in Orlando, Florida for sponsoring this project. In particular, we would like to thank Aaron Phu from Lockheed Martin and George Loubimov from the Aerospace Engineering department at the university for their time and guidance. We would also like to thank our ECE senior design professor, Dr. Richie.

REFERENCES

[1] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg

[2] "Speed/accuracy trade-offs for modern convolutional object detectors." Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, Murphy K, CVPR 2017. https://github.com/tensorflow/models/blob/master/research/object_detection/samples/configs/ssd_mobilenet_v2_coco.config

[3] Bédorf, Jeroen. "Deploying SSD mobileNet V2 on the NVIDIA Jetson and Nano platforms." Dec 13, 2019. https://www.minds.ai/post/deploying-ssd-mobilenet-v2-on-the-nvidia-jetson-and-nano-platforms

[4] "Jetson Nano: Deep Learning Inference Benchmarks." Nvidia. https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmark

[5] "ROS Wrapper for Intel® RealSense™ Devices." IntelRealSense. Git code (2020). Licensed under Apache License V2.0 (http://www.apache.org/licenses/LICENSE-2.0). https://github.com/IntelRealSense/realsense-ros

[6] Franklin, Dustin. "ros_deep_learning." Git code (2020). https://github.com/dusty-nv/ros_deep_learning