Unmanned Aircraft System Operating Autonomously in GPS Denied Environments

MAE 435: Project Design and Management II

Old Dominion University

October 16th, 2019

Advisors: Dr. Thomas Alberts, Dr. Ahmed Mekky

Team: Nicholas Darden, Vivian Fakharizadeh, Zavia Harris, Brendan Regan, Richard Jones, Trent Jones, Joshua Lee, Ryan McLaughlin, Rushal Patel, Quinton Vehon

Table of Contents

[List of Figures 3](#_Toc22073430)

[List of Tables 4](#_Toc22073431)

[Abstract 5](#_Toc22073432)

[Introduction 6](#_Toc22073433)

[Completed Methods 7](#_Toc22073434)

[LiDAR 7](#_Toc22073435)

[IMU 7](#_Toc22073436)

[OpenCV, ArUco and Microsoft Visual Studio C++ Setup 9](#_Toc22073437)

[ArUco Setup and Camera Calibration 9](#_Toc22073438)

[GPS: 12](#_Toc22073439)

[Proposed Methods 14](#_Toc22073440)

[IMU: 14](#_Toc22073441)

[LiDAR Mounting and Flight Testing 15](#_Toc22073442)

[Raspberry Pi and Code implementation 15](#_Toc22073443)

[Configuring the Primary GPS: 17](#_Toc22073444)

[Using Pi to Create Spoof GPS Signals: 18](#_Toc22073446)

[Preliminary Results 19](#_Toc22073447)

[LiDAR: 19](#_Toc22073448)

[IMU: 19](#_Toc22073449)

[Visual Odometry: 22](#_Toc22073450)

[Discussion 23](#_Toc22073451)

[References 24](#_Toc22073452)

# List of Figures

Figure 1: The two-step process of the Kalman Filter

Figure 2: ArUco Markers

Figure 3: Raspberry Pi 3B+

Figure 4: Pintouts for GPS module to PixHawk

Figure 5: ArUco Marker showing three axes using calibration code

Figure 6: ArUco Marker Identification

Figure 7: Enabling Raspberry Pi Serial Port

Figure 8: Raw IMU acceleration data for 1-D test in y-direction

Figure 9: Example of testing the IMU at rest

Figure 10: Closer look at accelerometer bias in X and Y direction.

Figure 11: ArUco Marker showing three axes using calibration code.

Figure A-A.1: ArUco Marker and Camera Calibration Code

Figure A-A.2: YouTube tutorial link

Figure A-B.1: MATLAB Dead Reckoning Code

Figure A-B.2: IMU Test Results (x-direction motion)

Figure A-B.3: IMU Test Results (y-direction motion)

Figure A-B.4: IMU Test Results (z-direction motion)

Figure A-B.5: IMU stationary test (Checking for bias in IMU)

Figure A-B.6: Raw IMU acceleration data for 1-D test in x-direction (1) and z-direction (2)

# List of Tables

Table 1: LiDAR Sensor Experimental Data

Table 2: Rotation and Translation Vectors

# Abstract

An Unmanned Aircraft System (UAS) can be used for different purposes including enjoyment, filming, or mapping locations. They usually depend on a Global Positioning System (GPS) and Inertial Measurement Unit (IMU) for navigation and altitude controls, which in certain applications, such as search and rescue missions or transporting cargo in dangerous territory, are vulnerable to being hacked and traced. To remedy this, the United States Navy requested a UAS that is capable of carrying a payload to a predetermined location without GPS to improve stealth in hostile environments. This project will provide alternative navigation systems that do not utilize GPS. Sensors including Light Detection and Ranging (LiDAR) and a barometric altimeter will be used to fly to a predetermined altitude of 40 meters from the ground and around any unforeseen obstacles. The IMU will estimate the location of the UAS through the calculation of its instantaneous motion, and through visual odometry, the on-board camera will calculate the distance and orientation of landing markers allowing the UAS to land through visual recognition of an encoded landing zone. All sensors were sufficiently accurate within desired parameters and will be integrated to address the problems associated with flying without GPS.

# Introduction

The Navy's current primary means of transporting equipment offshore include Vertical Replenishment (VERTREP), where helicopters deliver munitions to ships on the deep ocean, smaller vessels travelling from land, or the return of the ship itself to shore in a time consuming, expensive, and dangerous process. Therefore, a procedure involving high system security [1], low cost [2], and low system complexity [3] is needed for the delivery of payloads to seagoing vessels across large distances.

The VERTREP methods have required range finding capability, onboard guidance without GPS, and landing zone recognition, while LiDAR and barometric altimeter sensors are capable of providing obstacle avoidance and desired altitude. Additionally, an IMU linked with a magnetometer have allowed UASs to stay on course without manual controls and visual optics, utilizing a Raspberry Pi that runs multiple integrated software, have allowed UASs to execute landing on a secure pre-generated marker. UASs are capable of autonomous navigation using these on-board sensors without GPS [4, 5]. In the past, accumulated navigational errors have resulted in a lack of success accurately navigating over the ocean on missions over 200 miles [1, 4]. To provide a solution to this problem, the aforementioned array of sensors will be integrated into a Vertical Take-Off and Landing (VTOL) platform that will allow the UAS to accurately fly across Kaufman Mall without remote or GPS guidance. Therefore, the primary objective of this project is to produce an inexpensive UAS designed that will ensure the safety of offshore resupply missions for the U.S. Navy without the aid of GPS.

# Completed Methods

LiDAR**:**

The SF11/C (LightWare Optoelectronics, Gauteng, South Africa) LiDAR sensor was used for testing over still, turbulent, murky, and ocean water surface profiles. Since its accuracy was within parameters, the SF11/C’s wires were soldered to the connector wires in order of 5V, GND, RXD, TXD in connector holes 1,2,3, and 6 respectively in GPS Port 2 of the main Pixhawk 2.1 (Hex, Sha Tin, Hong Kong) flight controller.

Mission Planner (ArduPilot Dev Team) was used to set up flight parameters and record the data feedback from the LiDAR that was interpreted with the PX4 Companion Software (Dronecode Project, San Francisco CA). The functionality of the LiDAR was tested by lifting the VTOL UAS and comparing the recorded data with a predetermined height.

IMU**:**

The IMU, which utilizes a combination of accelerometers and gyroscopes, will eventually be implemented intodead reckoning methods, which estimate current states based off past data readings. These methods will be used to predict the current position of the UAS once it is deprived of GPS.The data for this process was collected using an ADIS16475 IMU (Analog Devices, Norwood, MA), which provides acceleration and angular velocity data about the x, y, and z axes. A Raspberry Pi 3 B+ (Raspberry Pi, UK) board was linked to the IMU to read the output data through the 32-bit pinout.

The IMU was attached to a file cabinet door with a rolling track that provided a stable mount, reduced noise, and a single axis of motion when pulled out 0.5 meters from its initial position. The recorded data was then run through the custom IMU position code on MatLab (The Mathworks, Natick, MA) (Figure A-B.1), which uses Euler integration to find velocity and position values from the sets of acceleration and angular velocity data readings. In order to convert the body frame velocities to inertial frame velocities, a rotation matrix, R, was used (Appendix C; Eq.1), which computed the Euler angles, yaw (ψ), pitch (θ), and roll (φ) by equating each element in R with the corresponding element in the matrix product Rz(ψ)Ry(θ)Rx(φ). This value was then used to find the inertial frame velocities (Eq. 2).

(2)

A similar process was done for the angular rates, in which transformation matrices (Appendix C; Eq. 3)(Eq.4) were used to convert from body frame angular rates to inertial frame angular rates.

(4)

Testing of the IMU, showed bias and noise measurements that will need to be corrected so that the position of the quadrotor can be calculated accurately. Extensive research was done on the Kalman filter method, which is an optimal estimation algorithm, to eliminate this noise. . It is a two-step process that predicts and updates recursively and in real time. In the prediction step, the Kalman Filter produces estimates of the current state and the uncertainty in that state estimate. The next time step measurement is then observed with the state estimate and the estimates are updated, with more weight given to the estimate with less uncertainty. The Extended Kalman filter (EKF) is a nonlinear version of the Kalman filter. The EKF linearizes the non-linear function around the mean of the current state estimate and at each time step, the linearization is performed locally. Multiple EKF MATLAB codes were compiled so that they could be modified to fit our quadrotor model.

A screenshot of a cell phone

Description automatically generated

Figure 1: The two-step process of the Kalman Filter

Upon testing, the IMU was found to be reading acceleration in the x and y axis when the IMU is at rest, indicating bias in these directions. This was reduced using an average of the bias across a 10 second static test of the IMU (IMU at rest on flat surface). The average bias across the 10 second test was subtracted from the biased measurements to come up with measurements much closer to true measurements.

OpenCV, ArUco and Microsoft Visual Studio C++ Setup**:**

The first step to fly the GPS denied autonomous UAS at a predetermined location and land on a specific symbol is first to attempt the process on a laptop. In the first portion of senior design, the object detection script was running using Microsoft Visual Studio C++ 2017 (Microsoft, Redmond, WA) and OpenCV version 4.1.0 (Open Source Computer Vision Library) on a laptop [1]. OpenCV is an open-source database for object tracking and camera-based algorithms. For the algorithms and commands to work from OpenCV, the directories were connected by entering it into the properties of the Visual Studio project. This connection allowed Visual Studio to pull specific commands from the OpenCV database and use them within the script, described in more detail [1].

ArUco Setup and Camera Calibration**:**

For the UAS to detect the predetermined landing symbols, another open-source database called ArUco (GitHub Inc., San Francisco, CA) was used in parallel with OpenCV [2,3]. ArUco was implemented to create its unique black and white markers and contains functions to identify them using a camera and can estimate their distance and rotation (Figure 2).



Figure 2: ArUco Markers

To accurately read the markers using a camera, calibrations must be made using a checkerboard marker and performing relative distance tests. The calibration will involve holding up a checkerboard marker, tilting it, and recording the data points the executable saved.

Integration of Open Source Software:

In the second portion of senior design, instead of working on a laptop, a Raspberry Pi was introduced and used with the hope of enacting the same process and results. The objective was to download OpenCV and ArUco on a Raspberry Pi so the ArUco markers can be detected, and markers position can be estimated. After that, the Raspberry Pi will be connected to the Pixhawk allowing the raspberry pi can communicate with the Pixhawk and determine the position of the UAS and make the UAS land on the markers autonomously. It was found that the installation process would be different from the laptop, and the code will be written in Python (Python Software Foundation, Wilmington, DE) instead of C++.

First, a Raspberry Pi (Figure 3) was set up with Raspbian, which is an operating system for the Raspberry Pi. In order to allow convenient use of the Raspberry Pi, two pieces of open source software were used without the need for a separate computer station. The first was PuTTY (Cambridge, England) which is a server software that allows the user to access the terminal of the Raspberry Pi from their laptop screen [9]. The second was VNC Viewer (Cambridge, UK) a virtual desktop software that connects to the Raspberry Pi via the internet.[11] This software allows the user to use the laptop screen as the display for the raspberry pi itself. Then the Raspberry Pi was set up to download many tools, dependencies, libraries, and packages to accurately setup ArUco and OpenCV [4]. A virtual environment was created so that different versions of Python can run on the raspberry pi. Then the OpenCV and ArUco libraries were downloaded and compiled in this environment. To make sure OpenCV was installed on the raspberry pi, the terminal was asked to detect what version of OpenCV was downloaded. After the version was confirmed, the Raspberry Pi Camera Module V2-8 Megapixel 1080p (Raspberry Pi, UK) was enabled, and a test code using Python was compiled to make sure the camera and OpenCV were running correctly [5]. The camera on the raspberry pi had to be calibrated to accurately detect and estimate the three-dimensional coordinates of the markers relative to the camera [6]. The calibration was conducted by taking several pictures of a chessboard marker at different angles. After the calibration was finished, code was created to make the camera identify the markers and then estimate the position of those markers in x, y, and z coordinates relative to the UAS [7].



Figure 3: Raspberry Pi 3B+

# GPS:

A proxy GPS will be used to test the data accuracy and module connection for GPS module and raspberry Pi. The test was done to determine how to connect GPS to Pi and what kind of data should be expected. Once it is determined that the test-GPS connects and can transmit data accurately, Berry GPS V3 (Berry GPS-IMU, ozzmaker, Australia) will be used during the final GPS integration stage of the project. In order to test the 3DR Ublox GPS (3DR ublox GPS with compass kit, 3dr, Berkeley, CA) data accuracy with Pi, a connector was fashioned to have one end connect to the serial connection of the Pi. For the purpose of the test, soldering method was used to modify the connector. This step is necessary because most GPS modules have pin connectors that are designed to connect to Pixhawk, and since connecting to Pi requires serial connection, the wire must be modified to fit on the serial pins of the Pi. Before connecting the GPS to Pi, it is recommended to check the voltage requirement for the GPS. For this specific model, 5V were required to run the GPS module. Connecting a 3.3V GPS into a 5V pin of the Pi might cause the GPS and Pi to malfunction and make them unusable. Once the power requirement for the GPS is established, four pins connector is required to power and transmit data to Pi. GPS pin for the power will connect to the appropriate voltage pin on the Pi, the RX pin of the GPS will connect to the TX pin on the Pi, TX pin on the GPS will connect to RX pin on the Pi and ground pin on GPS will to the ground pin on Pi. Figure 4 shows pinouts for a different model of GPS, however, it was discovered that all GPS modules have universal pin connection; the first pin from the left will always be the power connection regardless of what model the GPS is and all the models have power, RX, TX and ground pins at most. If in any case, a GPS module have more than said pins, it is highly recommended to check the power requirements and pinouts for that module.

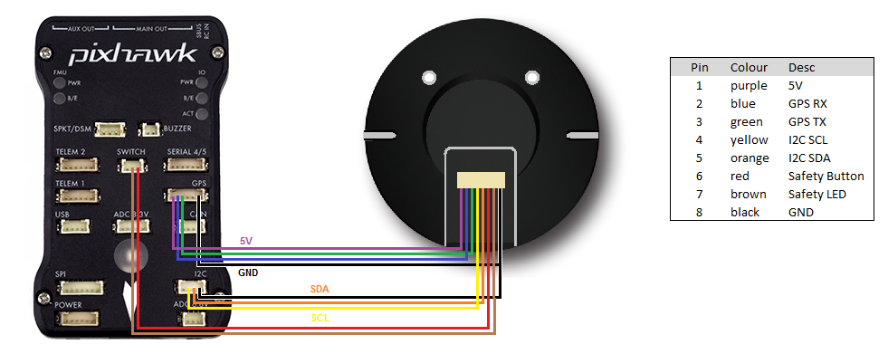


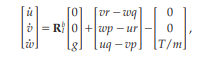
Figure 4: Pintouts for GPS module to PixHawk

Once the GPS module is connected to Pi, GPS libraries will be required to read data to Pi. Minicom, gpsd and gpsmon are such are python functions that will help in testing GPS data, communicate with GPS module and convert GPS data into a readable format. Minicom library can used to test raw data GPS data. Data from GPS will be transmitted in NMEA sentences, and using minicom will print that data, consisting of latitude, longitude etc. gpsd client is a service that will run as a background process which can receive data from GPS and transmits it to gpsmon into a readable format. gpsmon or cgps are two sudo commands that shows readable NMEA sentences into a table consisting of latitude, longitude, altitude, speed etc. To install these libraries, a command must be written into the terminal window of Pi to download and install files from internet. For installation process, an internet connection is required for Pi to install any kind of library onto the system. For example, by typing “sudo apt-get install minicom”, in the terminal window of Pi will prompt it to install minicom library to the system. To test the accuracy of the GPS, latitude and longitude coordinates can be checked in a web browser to test the location.

Proposed Methods:

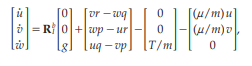
IMU:

Dr. Mekky will working on producing an EKF MATLAB code from the code the subgroup has compiled. The EKF will allow for a better estimation of the position of the quadrotor during flight. If time allows, an augmented EKF will also be researched, which uses a model slightly different to the traditional quadrotor model (Eq. 5), which assumes that the significant forces acting on the quadrotor are gravity and the rotor thrust.



(5)

The augmented EKF uses a drag-force-enhanced model (Eq. 6) by adding a drag force, which is proportional to the body-fixed-frame velocity of the quadrotor. The last term in (Eq. 6) accounts for rotor drag, which represents the induced drag forces on the rotor.



(6)

The bias control method used so far needs updating to account for moving tests. A dynamic model may be needed to tackle the bias in the IMU and clean out some of the measurements.

LiDAR Mounting and Flight Testing**:**

A lightweight 3D printed frame must be designed to fit the LiDAR to the UAS arm. After successfully integrating LiDAR SF11/C Sensor into the PixHawk 2.1 platform, additional testing will be performed over the four water conditions. Once the test platform UAS accomplishes this goal, the electronics and software will be migrated to the fixed-wing VTOL UAS. The current objective is to have it fly across 300m of Kaufman Mall without the assistance of GPS signal. Once that objective has been obtained, the final test for this iteration of the project will be to fly the Fixed-Wing VTOL UAS 5 miles over water and return autonomously to land on a pre-generated marker.

Raspberry Pi and Code implementation**:**

In the future, the goal is to connect the Raspberry Pi to the Pixhawk. This goal would allow the position of the UAS can be determined during flight. The Raspberry Pi can also estimate the distance of the markers relative to the UAS and make the UAS land when it sees the markers [8].

In regards to testing, the Pi Camera will undergo multiple phases of testing. There will be a stationary test, where the pi camera is mounted in place, and the ArUco marker is moved along the 3D axis. The position coordinates will be recorded. The Pi Camera will be mounted to custom 3D printed frames. One of the frames will have the camera positioned with the lens facing the ground. Another mount will have the camera facing the ground at an angle of 35 degrees from the horizontal. This dual-angle setup is to see if the Pi Camera can lock onto the ArUco markers at different positions.

Results

From the calibration performed on the laptop, the C++ code (Appendix A-A.1) using a laptop camera successfully identified a pre-generated marker within its library database (Figure 5).

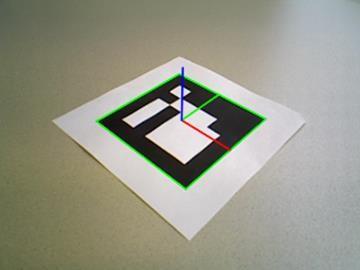


Figure 5: ArUco Marker showing three axes using calibration code

The green, red, and blue lines indicate the x, y, and z axes, respectively. The data shows the rotation and translation vectors relative to the camera [Appendix A: Table 1]. From the Raspberry Pi, the camera was also successful in identifying a pre-generated marker (Figure 6).

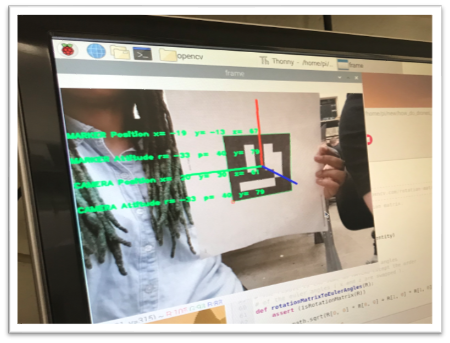


Figure 6: ArUco Marker Identification

Configuring the Primary GPS:

The first proposed task will be to set up our primary gps (BerryGPS-IMU 3) now that we have tested the capabilities of the raspberry pi on our test platform. When setting up a new configuration on a pi it is essential that you begin with updating the existing libraries and programs already installed. This can be done using the sudo commands in the devices terminal.

pi@raspberrypi ~ $ sudo apt-get update  
pi@raspberrypi ~ $ sudo apt-get upgrade  
pi@raspberrypi ~ $ sudo reboot

After updating our existing resources, we would then enable the serial port on our pie by assessing the config file and enabling the serial port while ensuring that the serial console remains disabled. This can be accomplished in the device’s config file by typing ‘pi@raspberrypi ~ $ sudo raspi-config’, then select Interfacing options, Serial, No, Yes, and then Yes to reboot (Figure 7). This should allow the GPS to function on the raspberry pi and the group to move on to scripting the GPS for the objective of autonomous flight.

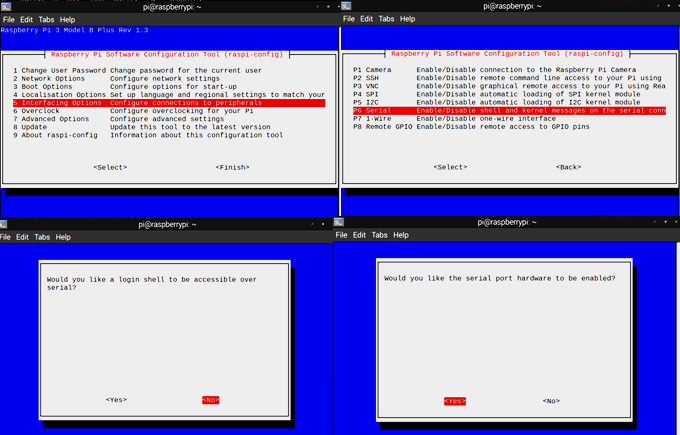


Figure 7: Enabling Raspberry Pi Serial Port

Using Pi to Create Spoof GPS Signals:

We will use sudo codes such as ‘pi@raspberrypi ~ $ screen /dev/serial0 9600’ and ‘pi@raspberrypi ~ $ gpspipe –r’ to extract the GPS data and inject it into a code that will ultimately allow the UAS to continue accurate flight once the GPS signal is terminated. We propose that a code that takes the GPS’s stationary initial readings and provides them to a sufficiently accurate IMU would be able to use a feed loop through to spoof the UAS’s current location long enough for it make and unguided journey to its target. Once the Optical sensors detect the landing zone the GPS will be reactivated, and landing would be initiated.

# Preliminary Results

LiDAR:

Altitude data readings from the LiDAR over varying water surface profiles showed a maximum of 3.1% error (table 1) when over turbulent water. With the readings being within the desired parameters of accuracy within 6 centimeters, the SF11/C sensor was made the primary altitude measurement device required for the final product.

|  |  |  |  |
| --- | --- | --- | --- |
| Measured height (m) | Actual height (m) | Water | Error (%) |
| 3.3 | 3.20 | turbulent | 3.1 |
| 3.0 | 3.02 | still | 0.7 |
| 3 | 3.05 | murky | 1.6 |
| 3.8 | 3.79 | ocean | 0.3 |

Table 1: LiDAR Sensor Experimental Data

IMU:

Once the Raspberry Pi and IMU were connected, tests were run for one-dimensional motion. When the IMU was restricted and only moved in the y-direction, the data still showed acceleration in the x-direction (Figure 8). This IMU sensor bias is corrected by subtracting the average bias from readings.

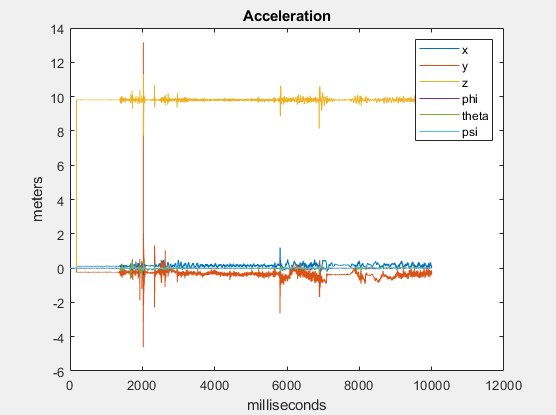


Figure 8: Raw IMU acceleration data for 1-D test in y-direction

Results from several of the three one-dimensional tests in the x, y, and z directions (A-B.2-A-B.6) did not meet the accepted design parameters of 2% error. Hypotheses on the inaccuracy were directional bias and noise interference.

Due to some errors with the sensor, the IMU had to be diagnosed and fixed before testing could resume. Once the IMU was ready for use again, testing resumed for acceleration and gyroscope readings so that any IMU bias could be identified.

The bias was determined by running the IMU while stationary, producing the results below:

A screenshot of a social media post

Description automatically generated

Figure 9: Example of testing the IMU at rest

A screenshot of a cell phone

Description automatically generated

Figure 10: Closer look at accelerometer bias in X and Y direction.

An average bias, represented by the deviation of the plotted lines from the x-axis (Figure 10), was taken across a 10 second period while the IMU was at rest. The average bias was then subtracted from each individual timestep measurement. This corrected the bias significantly and improved results in the position code, but still produced some error. From 1-3 seconds, the position reading was zero for all three axes, which was correct given the IMU was at rest, but after 3 seconds, the small error eventually grew, and the position estimate was thrown off. Despite the IMU remaining stationary during the testing period, MATLAB position code showed that the IMU moved approximately 0.05m in all three directions. This confirmed the accumulation of error due to the approximated bias correction.

Visual Odometry:

From the calibration performed in the methods, the C++ code (Figure A-A.1) using a laptop camera successfully identified a pre-generated marker within its library database (Figure 11).

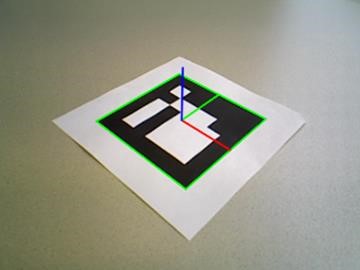


Figure 11: ArUco Marker showing three axes using calibration code.

The green, red, and blue lines indicate the x, y, and z axes respectively. The data shows the rotation and translation vectors relative to the camera [2, Appendix C].

# Discussion

The purpose of this project was to produce a VTOL UAS capable of unguided autonomous VERTREP operations. In this project’s development, a UAS flight system that met the Navy’s requirements was devised using a LiDAR sensor, an IMU, and the ArUco data base for library marker landing in order to fly and land with GPS disabled.

The results show that the software requires further post-processing due to current inaccuracies of the sensors. A higher powered SF11/C LIDAR unit showed major improvement in the accuracy of altitude estimation with under 4% error, which is accurate enough to be the primary altitude measurement device for the UAS. The IMU required the average bias that was shown on tests to be subtracted from the x, y, and z axes readings to correct drift. The optic odometry software successfully identified the pre-generated markers, measuring the rotation and translation vector after running the ArUco and camera calibration code.

The primary limitation of this project has been the team members’ lack of knowledge with components and software in regards to UAS technology. Future plans for this project include integrating the aforementioned sensors and software into the Fixed-Wing VTOL UAS. The Fixed-Wing VTOL is expected to travel further distances at higher speeds with more energy efficiency than the original quadcopter UAS.

# References

1. Braga, Jose R. G., et al. “Estimation of UAV Position Using LiDAR Images for Autonomous Navigation over the Ocean.” 2015 9th International Conference on Sensing Technology (ICST), 2015, doi:10.1109/icsenst.2015.7438508
2. I. Sa, M. Kamel, M. Burri, M. Bloesch, R. Khanna, M. Popovic, J. Nieto, and R. Siegwart, “Build Your Own Visual-Inertial Drone: A Cost-Effective and Open-Source Autonomous Drone,” IEEE Robotics & Automation Magazine, vol. 25, no. 1, pp. 89–103, Dec. 2017.
3. E. Fresk, G. Nikolakopoulos, and T. Gustafsson, “A Generalized Reduced-Complexity Inertial Navigation System for Unmanned Aerial Vehicles,” IEEE Transactions on Control Systems Technology, vol. 25, no. 1, pp. 192–207, Apr. 2016.
4. J. Pestana, I. Mellado-Bataller, J. L. Sanchez-Lopez, C. Fu, I. F. Mondragón, and P. Campoy, “A General Purpose Configurable Controller for Indoors and Outdoors GPS-Denied Navigation for Multirotor Unmanned Aerial Vehicles,” Journal of Intelligent & Robotic Systems, vol. 73, no. 1-4, pp. 387–400, Oct. 2013.
5. T. Wang, C. Wang, J. Liang, Y. Chen, and Y. Zhang, “Vision-Aided Inertial Navigation for Small Unmanned Aerial Vehicles in GPS-Denied Environments,” International Journal of Advanced Robotic Systems, vol. 10, no. 6, pp. 1–12, May 2013.
6. OpenCV team, “About,” OpenCV. [Online]. Available: https://opencv.org/about/. [Accessed: 09-Jul-2019].
7. ArUco, “Detection of ArUco Markers,” OpenCV. [Online]. Available: https://docs.opencv.org/3.1.0/d5/dae/tutorial\_aruco\_detection.html. [Accessed: 27-Jul-2019].
8. racer993, racer993, T. J. 10, C. J. 8, roma M. 30, racer993 P. authorM. 31, eric O. 29, K. A. 17, and racer993 P. authorA. 17, “Connect to the Raspberry Pi via SSH / Putty,” *raspberrypi4dummies*, 24-Mar-2013. [Online]. Available: <https://raspberrypi4dummies.wordpress.com/2013/03/17/connect-to-the-raspberry-pi-via-ssh-putty/>. [Accessed: 14-Oct-2019].
9. racer993 and racer993, “First time configuration of my Raspberry Pi,” *raspberrypi4dummies*, 24-Mar-2013. [Online]. Available: <https://raspberrypi4dummies.wordpress.com/2013/03/17/first-time-configuration-of-my-raspberry-pi/>. [Accessed: 14-Oct-2019].
10. “Raspberry Pi VNC Server - Setup Remote Desktop for your Pi,” *YouTube*, 14-Feb-2016. [Online]. Available: <https://youtu.be/JYKZjUgtOYw>. [Accessed: 14-Oct-2019].

Appendices

Appendix A:

A screenshot of a social media post

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

Figure A-A. 1: ArUco Marker and Camera Calibration Code

 Figure A-A.2:

G. Lecakes, “OpenCV Basics - 11 - Building OpenCV Contribute with CMake,”  YouTube, 08-Apr-2016. [Online]. Available: https://www.youtube.com/watch?v=fIpTks0G2m0&list=PLAp0ZhYvW6XbEveYeefGSuLhaPlFML9gP&index=11. [Accessed: 09-Jul-2019].

Appendix B:

1 
2 
3 
4 
5 
6 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
30 
31 
32 
33 
34 
clear ; clc; close 
all 
% Daniel Montgomery 
% Inertial Dead Reckoning 
% Last Modified 3/28/2019 
% Modified 4 / 18/2019 By: Ahmed Mekky 
% Initial Conditions 
COORDINATE SYSTEM 
Z: Up 
X: Forward (North) 
Y: Left 
(West) 
Inertial Frame 
% Rotation 
% Translatio 
Body Frame 
% Rotation 
wB-[O, O, 0] ; 
thetaB— [0, 0, 0] ; 
Translation 
Inertial Angular Rate 
Euler Angle 
Inertial 
Inertial 
Inertial 
Acceleration 
Velocity 
Position 
[phidot 
[phi 
[xddot 
[xdot 
[x 
[p 
[pint 
[udot 
[u 
thetadot 
theta 
yddot 
Ydot 
psidot] 
psi] 
zddot] 
zdot] 
z] 
Body 
Body 
Body 
Body 
Angular Rate 
Angle 
Acceleration 
Velocity 
qlnt 
vdot 
r] 
rlnt] 
wdot ] 
w] 
Infile Data to Matrix 
A=xlsread ( ' imuDat . xlsx ' 
load ('CTEST l.mat'); 
A=IMU DATA; 

35 
36 
37 
38 
39 
40 
41 
42 
43 
44 
45 
46 
49 
50 
51 
52 
53 
54 
55 
56 
57 
58 
59 
60 
61 
62 
63 
64 
65 
—9.81] 
66 
67 
68 
Shift matrix down to 
while j < 100 
count=j ; 
if A (j, 3) 
j=j+l,• 
else 
j=100,• 
end 
end 
(count : ) ; 
T i me step 
Hz=1000; 
dt—1/Hz; 
skip random initial data 
9.81 *cos (pi/ 6) 
tstart— (count—I) *dt; 
tstop (n—l) *dt; 
t=tstart : dt : tstop ; 
Gravity calibration 
G=A (count : 50+count, 
gx=mean (G (1:50, 1)) 
gy=mean (G (1 : 50, 2) ) 
gz=mean (G (1 : 50, 3)) 
[gx, gy, gz] ; 
% Hertz 
% Time step 
% Starting time of calc 
% Final time at n 
% Time vector 
collect averages of component accelerati ns 
assuming drone is somewhat level and sti 1 
for first .05 seconds 
k ky 
State 
Vector 

69 
70 
71 
72 
73 
74 
75 
76 
77 
78 
79 
80 
81 
82 
83 
84 
85 
86 
87 
88 
89 
90 
91 
92 
93 
95 
97 
98 
LOO 
LOI 
L02 
X—zeros (n count, 21+3+3+3) ; 
% Euler Integration Method 
rlB=r13; 
for i=count : n—count+l 
Read in data 
4:6) . * (pi/ 180); 
aB=A (i, 1: 3) ; 
9.8* [—sin (thetal (1)), 
a bb =aB—g b; 
Angular Rate Transformation 
T=Transform(thetaI) ; 
wl=wl ' ; 
[WE, aB,w1, 
vlb 
thetal, al, VI, rl] 
WB @t(i) 
aB @t(i) 
sin (thetal (1) ) *cos (thetal (2)), cos (thetal (1) ) *cos (thetal (2)) ] ; 
Linear Acceleration Transformation 
R=Rotate (thetal) ; 
al=al ' —g; 
all—all' ; 
State Defined @ t (i) 
X (i, :)=[wB,a bb,wl, thetal,all,vl, 
Euler Integration 
theta12 = Eulerlnt (thetal, WI, dt) ; 
Acce. in Body frame without gra ty 
thetal (i) 
WI @t(i) 
thetal (i) 
al @t(i) 
Bl',r13]; 
vlb 
r13 
VI 2 
r12 
Eulerlnt (vB,a bb, dt) ; 
Eulerlnt (rlB,V El' , dt) ; 
Eulerlnt (VI, al, dt) ; 
Eulerlnt (rl, VI, dt) ; 

103 
104 
vl=v12 ; 
105 
vB=v1b ; 
106 
r1B=r13; 
107 
r1=r12; 
108 
109 
end 
110 
111 
112 
x . *180/pi; 
113 
114 
115 
116 
117 
0,0] 
118 
119 
120 
121 
122 
123 
124 
125 
126 
127 
128 
129 
130 
131 
132 
133 
134 
135 
136 
% Update vectors 
Cheta 1=theta12 ; 
Trapezoidal 
Integration Method 
vB-[O, O, 0] ; 
wB2-[O, O, 0] ; 
aB2 
=[0, 
for i=count : n—count+l 
Read in data 
4:6) . * (pi/ 180) ; 
aB=A (i, 1:3) ; 
if i < n—count+l 
4:6) . * (pi/ 180) ; 
aB2=A ; 
Angular Rate Transformation 
w12=w12' ; 
Linear Acceleration Transformation 
R=Rotate (thetal) ; 
WB 
WB 
(1+1) 
(1+1) 
thetal (i) 
WI (1+1) 
thetal (i) 

137 
al=al ' —g; 
138 
R=Rotate (theta12) ; 
139 
140 
a12=a12 ' —g; 
141 
142 
State Defined @ t (i) 
143 
XT (i, :)=[wB,a bb, WI, thetal, al, vl, rl] 
144 
145 
Trapezoidal Integration 
146 
theta12=thetaI+ (wI+w12) *dt/2 ; 
147 
148 
v12=vI+ (al+a12) *dt/2; 
149 
r12=rI+ (vI+v12) *dt/2; 
150 
151 
% Update vectors 
152 
theta 1=theta12 ; 
153 
154 
155 
156 
157 
158 
159 
160 
161 
. *180/pi; 
162 
. *180/pi; 
163 
164 
165 
166 
167 
168 
169 
170 
al 
al 
(1+1) 
vI=v12 ; 
rl=r12; 
else 
end 
end 
XT(:, 
XT(:, 
[wB,a bb, WI, thetal, 
: . *180/pi; 
Tables 
tR=tstart : dt * 100 : tstop ; 
len] =size (tR) ; 
rl=zeros (len, 13) ; 
r2=zeros (len, 10) ; 

171 
172 
173 
174 
175 
176 
177 
178 
fprintf ( 'Time (s) 
179 
180 
181 
%7.2f 
%7.2f 
182 
fprintf ( 'Time (s) 
183 
184 
185 
%7.2f %7.2f %7.2f 
%7.2f 
186 
187 
188 
189 
190 
191 
192 
193 
194 
195 
196 
197 
198 
199 
200 
201 
202 
203 
204 
205 
for : Ien 
rl (m, 1 : 13) = [tR (m) 
r2 (m, 1 : 10) = [tR (m) 
end 
rl (Ien+l, 
r2 (Ien+l, 
fprintf (' 
fprintf (' 
fprintf ( 
'%4.2f 
fprintf (' 
fprintf (' 
fprintf ( 
'%4.2f 
% Graphs 
figure (1) 
hold on 
subplot (3, 1, 1) 
plot (t, X ( : , 13) g' 
X (100* (m 
X (100* (m 
x (i, 1:12)]; 
x (i, 13:21)]; 
Body (deg/ s) 
%7.2f %7.2f %7.2f 
NED (m/sA2) 
Body (m/ s A 2) 
ud 
Phi 
NED (deg/ s) 
thetad psid 
NED (deg) \ n') 
Phi 
the ta psi\n' ) 
\n') 
Xdd 
ydd 
z dd 
%7.2f %7.2f 
%7.2f %7.2f 
NED (m/ s) 
yd 
%7.2f %7.2f 
%7.2f %7.2f 
%7.2f %7.2f 
,rl') 
NED (m) \ n') 
z\n') 
\n') 
% 7.2f\n',r2') 
title ( 'Acceleration NED frame (m/s 
legend('x', 'y', 'z', 'xtrap', 'ytrap', 'ztrap 
subplot (3, 1, 2) 
% plot (t, X t, X 
plot (t, X ( : , 25) , 'g' , t, X ( : , 26) r' 
title( 'Velocity NED frame (m/s) ' 
legend('x', 'y', 'z') 
subplot (3, 1, 3) 
% plot (t, X 
plot (t, X ( : , 28), 'g' , t, X ( : , 29) r' 
title('Position NED frame (m) ' ) 

206 
207 
208 
209 
subplot (2, 1, 1) 
210 
plot (t, X t, X 'b') 
211 
title ( 'Angle Rate 
Body frame (deg/ s) ' 
212 
legend ('p', 'q', 'r' 
213 
214 
subplot (2, 1, 2) 
215 
plot (t, X t, X 'b') 
216 
title ( 'Acceleration Body frame (m/sA2) ' 
217 
218 
219 
220 
12), 'b' 
221 
222 
223 
224 
225 
226 
227 
228 
229 
230 
231 
232 
233 
234 
235 
236 
237 
238 
239 
240 
legend ( 'x' , 
figure (2) 
hold on 
legend ( 'udot' , 'vdot' , 'wdot ' ) 
figure (3) 
plot (t, X (:,10), g' 
title ( 'Angles Inertial frame 
legend ( 'phi', 'theta', 'psi ') 
% Functions 
% Euler State Integrator 
r',t,X(:, 
(deg) 
function x2 = Eulerlnt (x, xdot, dt) 
x2=x+xdot . *dt; 
end 
% Rotation Matrix 
function R = Rotate (a) 
Rx [1 0 0; 0 cos(a(l)) —sin (a (1)); 
(a (2)) 0 sin (a (2)); 
010; 
0 sin (a (1)) cos(a(l))]; 
—sin (a (2)) 0 cos(a 
(a (3)) —sin (a (3)) 0; sin (a (3)) cos (a (3)) 0; 0 0 1]; 

240 
241 
242 
243 
244 
245 
246 
247 
248 
249 
end 
Transformation Matrix 
function T 
Transform (a) 
T—[l sin (a *tan (a (2)) cos (a *tan (a (2)); 
0 cos(a (1)) —sin (a (2)) 
0 sin (a (2)) cos(a (2))]; 
end 

*Figure A-B.1: MATLAB Dead Reckoning Code*

A screenshot of a video game

Description automatically generated*Figure A-B.2: IMU Test Results (x-direction motion)*

A screenshot of a cell phone

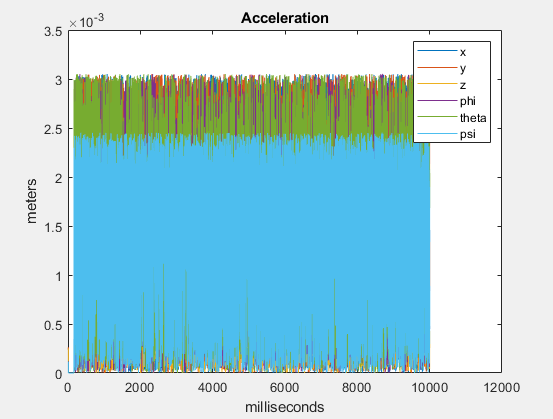
Description automatically generated

*Figure A-B.3: IMU Test Results (y-direction motion)*

A screenshot of a video game

Description automatically generated

*Figure A-B.4: IMU Test Results (z-direction motion)*



*Figure A-B.5: IMU stationary test (Checking for bias in IMU)*

A close up of a map

Description automatically generated

*Figure A-B.6: Raw IMU acceleration data for 1-D test in x-direction (1) and z-direction (2)*

Appendix C:

(1)

` (3)

A close up of text on a white background

Description automatically generated

Table 2: Rotation and Translation Vectors