GROUND VEHICLE NAVIGATION WITH DEPTH CAMERA AND TRACKING CAMERA

THESIS

Hongseok, Kim, Major, ROKAF

AFIT-ENV-MS-22-M-217

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENV-MS-22-M-217

GROUND VEHICLE NAVIGATION WITH DEPTH CAMERA AND TRACKING CAMERA

THESIS

Presented to the Faculty

Department of System Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in System Engineering

Hongseok, Kim, B.S.

Major, ROKAF

March 2022

AFIT-ENV-MS-22-M-217

GROUND VEHICLE NAVIGATION WITH DEPTH CAMERA AND TRACKING CAMERA

THESIS

Hongseok, Kim, B.S.
Major, ROKAF

Committee Membership:

approved:

David R. Jacques, PhD (Chairman)	date: 18 Feb 2022
LtCol Warren J. Connell, PhD (Member)	date: 18 Feb 2022
Mr. Jeremy Gray (Member)	date: 18 Feb 2022

Abstract

The aim of this research is to provide autonomous navigation of a 4 wheel vehicle using commercial, off-the-shelf depth and tracking cameras. Some sensitive operations need accuracy within a few inches of navigation ability for indoor or outdoor scenarios where GPS signals are not available. Combination of the Visual Odometry(VO), Distance-Depth(D-D), and Object Detection data from the cameras can be used for accurate navigation and object avoidance. The Intel RealSense D435i, a depth camera, generates depth measurements and the relative position vector of an object. The Intel RealSense T265, a tracking camera, generates its own coordinate system and grabs coordinate goals. Both of them can generate Simultaneous Localization and Mapping (SLAM) data. The cameras share their data to provide a more robust capability. Combining the Intel cameras with a Pixhawk autopilot, it was demonstrated that the vehicle can follow a desired path and avoid objects along that path.

AFIT-ENV-MS-22-M-217

This thesis dedicate the development and demonstration of ground vehicle of autonomous navigation using depth camera and tracking camera.

Acknowledgements

There is always trembling in taking on a challenge for the first time. Living abroad and studying for a master's degree with a language barrier was a challenge to me. The people surrounding me helped me overcome through this challenge. My instructor, Professor Jacques, always encouraged me to keep going on the process. Thanks to my wife; she always cheers me up when I am frustrated with the language barrier. And my two little sons; always think their daddy does not know how to give up.

Hongseok, Kim

Table of Contents

Page

List of Figures

List of Tables

GROUND VEHICLE NAVIGATION WITH
DEPTH CAMERA AND TRACKING CAMERA

I. Introduction

1.1 Intro and Motivation

Tragically, natural and man-made disasters are very harsh conditions for people to perform rescue operations and search for survivors. Often debris is scattered everywhere in the unknown area, it is very hard to find a way to a safe path, and sometimes it is too narrow for a direct human approach. A small vehicle can be used in these harsh conditions, and situations not conducive to human function. Vehicles must avoid crashing into obstacles in order to avoid adding to the debris field, so the desired path has to be modified in real time. For these reasons, position accuracy within a few inches and the ability to navigate and avoid obstacles is important for the small vehicles. Disasters like the collapse of a Florida building in 2021 should never happen again, but when it inevitably does, quick search and rescue is very important. It should be noted that small ground vehicles were used in the aftermath of the Florida building collapse [1].

GPS is widely used for navigation, and it is often taken for granted that GPS will always be available. But considering our operational environments, the GPS signal may not be accessible by the small vehicles. Furthermore, GPS accuracy depends on the surrounding area. While normal GPS accuracy is about 2.1 ft [2], it can be significantly degraded due to multipath environment. Further, considering the length of the vehicle may be less than 2 ft, the GPS error is almost the same as the length of

the vehicle. For these reason, GPS accuracy is not sufficient for vehicles that require only navigation accuracy within inches.

The navigation ability of the vehicle can be increased by adding Visual Odometry (VO) sensors; examples of these are depth and tracking cameras. The depth camera is a stereo camera, where two cameras are taking pictures at the exact same time from different vantage points. The stereo camera calculates the Distance-Depth (D-D) and relative position from the target to the camera. A tracking camera can set the visual reference point for navigation. Simultaneous Localization and Mapping (SLAM) is generated using these data. Beyond the localization and positioning, SLAM allows the construction of global maps base on the combined images obtained from the operations area.

1.2 Problem Statement

There are many situations where small, autonomous vehicle will have to navigate within tight spaces without the benefit of GPS. These vehicles will need to avoid obstacles while still ensuring arrival at specified locations.

1.3 Research Objective

Demonstrate autonomous navigation using VO with a combination of depth and tracking cameras and a low cost commercial autopilot. Determine how factors such as distance, path circuits, path shape, and obstacle avoidance impact the navigation accuracy.

1.3.1 Investigative Questions

1) How can depth and tracking cameras be integrated with a low cost, commercial autopilot to support autonomous navigation without the aid of GPS?

2) How do factors such as path geometry, distance, path circuits and obstacle avoidance impact the accuracy of navigation?

3) Is the demonstrated navigation system appropriate for vision-based guidance of SUAV using low-cost componentry?

1.4 Methodology

The experiments implemented VO using a depth camera and a tracking camera on 4 wheel vehicle for calculating the distance by the computer when the vehicle reached a final waypoint after following the path. The tests was conducted in several circumstances for assuming the vehicles' harsh operation area; two different lengths of the paths, single and repeated closed circuit path, triangle shape and square shape path, and with and without obstacle. All tests recorded vehicles parameters (vehicle's position and velocity, and detected object's position), pictures from the both cameras, and point cloud data at every certain point.

1.5 Assumptions and Limitations

A test expedient vehicle was tested indoors in a constrained environment. A single location was used for all measured tests, and time to complete all test points was 5 minutes or less. While a variety of obstacles were used for development, all measured tests used the same obstacles. Because of time constraints, testing was limited to 4 variables and 6 repetitions for each test point. A more comprehensive test plan would be required to fully evaluate a proposed system.

The vehicle does not have gimbals for moving cameras. The cameras were fixed and aligned with the vehicle forward axis. Therefore, the camera angular field of regard for detecting an object is limited by the field of view of the camera lens and the fixed camera position.

The vehicle uses 4 wheels for locomotion [3]. The front 2 wheels have a 45° steering angle limitation [4]. That means the vehicle cannot make a sharp turn at a corner waypoint. Therefore, the vehicle overshoots turns necessitating a recovery maneuver to the get back on the path. An alternative would have been to initiate the turn prior to the waypoint (an undershoot), but that was not implemented for this research.

The depth camera, D435i, has limitations. First, for capturing object features both indoors and outdoors, the ideal range is 0.3-3m; therefore, objects outside this range were ignored. It should be noted that the measurement error for the depth data is 2 percent at 2m. An additional limitation is due to the depth data update rate of 90fps being faster than the RGB frame rate of 30fps [5]. For this reason, the algorithms for following the route and avoiding objects are limited to the RGB frame update rate.

The board used for this research is the UP Squared board with Intel Atom X7-E3950 processor, onboard 8GB DDR4, and 64GB eMMC [6]. The Atom board generates movement commands by analyzing the data from the pictures, which are taken from both depth and tracking cameras. It sends the movement commands to the Pixhawk autopilot, which then commands wheel and steering motions. The latency between taking pictures and sending commands to the Pixhawk will affect the vehicle's navigation accuracy. The line of the Python code for movement command and saving pictures and data is about 1500 lines, which takes too long to calculate. This caused a significant limitation in the movement command rates that could be achieved, sometimes as low as 2 Hz.

Robotic Operating System (ROS) use is popular for real-time SLAM. The Atom board [6] uses Ubuntu 18.04 LTS (Bionic Beaver). Ubuntu 18.04 is only compatible with ROS Melodic [7]. ROS Melodic is limited to Python version 2.7 [8]. The code for the depth camera and tracking camera in this research is based on Python 3.x

or above for saving and demonstrating the pictures. This caused limitations in using Python code for real-time SLAM.

1.6 Documents Overview

This thesis divided into the following five chapters: Introduction, Background and Literature Search, Methodology, Results and Analysis, Conclusion. And the following are brief descriptions of the chapters.

Chapter 1: Introduction - Describes the motivation, problem statement, research objective, methodology, assumption, and limitations.

Chapter 2: Background and Literature Search - Paper reviews of the processing of Visual Odometry (VO) and Simultaneous Localization And Mapping (SLAM) and thesis using the T265 and D435.

Chapter 3: Methodology - This chapter will discuss architectural view, hardware description, software description, and experimental test plan.

Chapter 4: Results and Analysis - This chapter shows the object detection analysis, route and SLAM analysis, statistical analysis, qualitative analysis, and DOE analysis of the experimental results.

Chapter 5: Conclusion - Focuses on answering the three investigative questions.

II. Background and Literature Search

2.1 Visual Odometry and Simultaneous Localization And Mapping

Visual Odometry (VO) is used to improve the positioning and orientation capability of a robot, vehicle, or human using camera images. Early odometry used data from rotary encoder on a wheel of a vehicle. However, wheel odometry has errors from wheel slip in uneven terrain or other adverse conditions. Visual Odometry has advantages over wheel odometry because it is not affected by wheel slip. The VO processing is summarized below (see Figure 1).

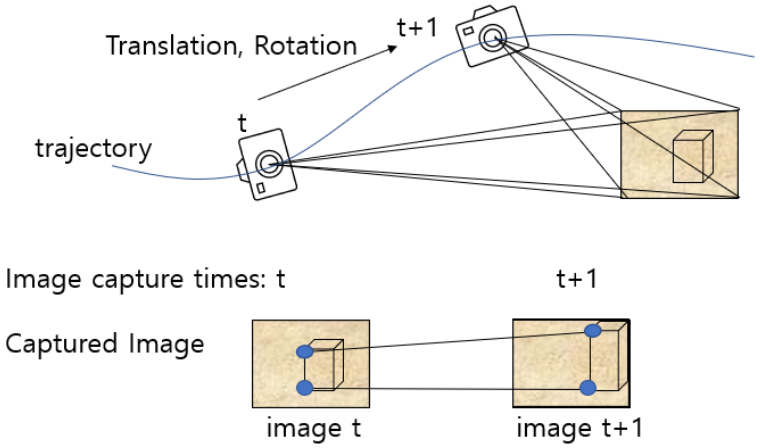


Figure 1. Visual Odometry

For VO, images are taken in a timed sequence. The VO algorithm then detects features in the image frame. and matches those features with those from the previous frames. Using the movement of the features in the image frames, the third step consists of computing the relative motion of the camera between the time instants t and t+1. The camera position is computed by concatenation of the movement with the previous pose. Finally, an iterative refinement can be done over the last frames to obtain a more accurate estimate of the local trajectory [9]. VO uses the sequenced images with time from monocular or binary cameras. The images are used

to determine the translation and rotation. The translation and rotation are used to estimate a camera trajectory. It is possible to calculate from the 'image t' the position of camera using Equation 1.

$$X^C = R_W^C(X^W - C^W) \tag{1}$$

The position of the object in image (X^C) is rotated by the world frame to camera frame rotation matrix (R_W^C) after subtracting the position of the camera (C^W) in world frame from the position of the object in world frame (X^W). The camera trajectory can be calculated by rearranging Equation1 for the position of the camera in the world frame, adding a homogeneous solution, and substituting the object coordinates of the camera frame and the object coordinate of the world frame [9] [10]. Typical accuracy for VO is about from 0.5 percent to 2 percent of distance traveled. For VO to work effectively, there should be sufficient illumination of the environment and a static scene with enough texture to allow apparent motion to be extracted. Furthermore, consecutive frames should be captured by ensuring that they have sufficient scene overlap. The VO uses Random sample consensus (RANSAC) to speed up its ability to capture objects [9] [10]. For relative 3-D positioning, most of the current research uses a stereo camera. The stereo camera has two cameras with a baseline distance between them, which take pictures at the exact same time. The two camera images can be compared by constructing epipolar lines and planes as shown in Figure 2. The relative 3-D positions of the features in the images are directly measured by triangulation at every robot location.

Simultaneous Localization And Mapping (SLAM) means the robot locates itself in unknown environments and scans surroundings. The main difference between VO and SLAM is that VO mainly focuses on local consistency and aims to incrementally estimate the path of the camera from one position to the next. SLAM aims to obtain

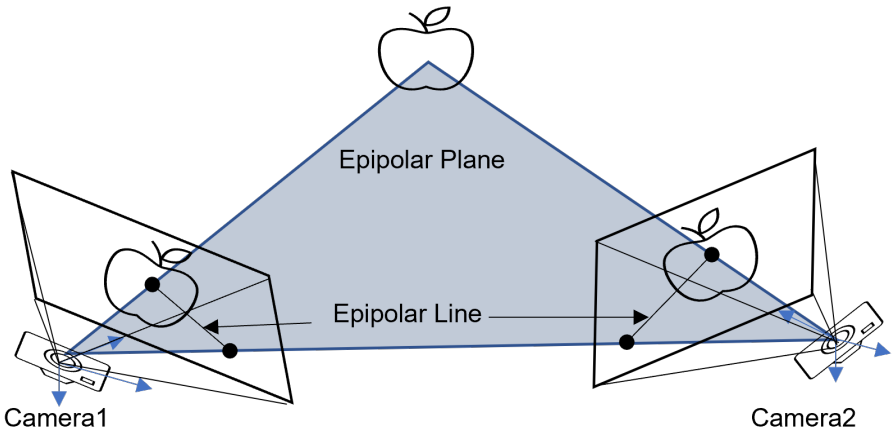


Figure 2. Stereo Camera and Epipolar Line

a globally consistent estimate of the camera trajectory and map. Global consistency is achieved by realizing that a previously scanned region has been traversed (closed-loop). This will decrease the drift errors. Recent advancements with SLAM use rich visual information from passive low-cost video sensors compared to LASER scanners. However, the trade-off is a higher computational cost and the requirement for more sophisticated algorithms for processing the images and extracting the necessary information. Due to recent advances in CPU technology, the real-time implementation of the required complex algorithms is no longer an insurmountable problem [9] [11].

2.1.1 Limitation and Application

Robust feature detection and continuous tracking of the object are important for the accurate calculation of camera position. For feature detection the code developed for this research used a library of OpenCV Findcontour [12]. However, the camera missed objects closest to the vehicle because of the properties of the random sampling. At a certain time, the Atom board calculated with no objects or objects which is not closest object. So depth-based code was added for correcting this problem.

Other researchers have used SLAM with Robot Operating System (ROS). We had tried to build ROS with another vehicle. But there were many errors in building the

code with the vehicle as well as board computer limitations. Therefore we changed to the Traxxas vehicle and Pixhwak autopilot which was used in another thesis. The Atom board computer was retain for the new configuration. The Atom board computer only supports ROS melodic which is not compatible with Python version 3.x or above. The RealSense camera is only used in python version 3.x or above. Python code was written without ROS for the camera to work and save the data. The scanned SLAM data is saved as point cloud data, but it is not used directly for navigation in the experiment. Its processing speed was very slow when used to avoid objects and navigate, so it only was used to analyze and create maps after the test.

2.2 Coupling of Localization and Depth Data for Mapping using Intel RealSense T265 and D435i Cameras

Tsykunov, et.al., used 2 cameras, t265 and D435i to navigate an autonomous robot in unknown environment. Tsykunov, et.al. built a 3D map to perform the real-time localization and path planning using two cameras. For this purpose they used an algorithm that fused data from two cameras into a 3D occupancy map. For the real-time localization, they compared different point cloud (PC) alignment methods and frame transformations to obtain different trajectories. They then created the whole scene by combining the depth data and PC data. They used an overlap of 80 percent and two methods, Iterative closest point (ICP) solution family and RSME, to achieve their point cloud [13].

2.2.1 Limitation and Application

Tsykunov, et.al focused on building a 3D map for movement, allowing vehicle navigation. Current research requires a way to align the point cloud data for the 3D map in real-time.

2.3 Autonomous Spatial Exploration with Small Hexapod Walking Robot using Tracking Camera Intel RealSense T265

Bayer and Faigl used a small hexapod walking robot with T265 and D435 cameras to navigate through harsh and unknown terrain. To reach the waypoints, they used ground mapping for terrain avoidance and a coordinate system for navigation to the goal. This paper focused on the development of computationally efficient solutions for the full navigation system including localization, mapping, planning, and decision-making in exploration missions [14].

2.3.1 Limitations and Application

Bayer and Faigl did not comment on the transformation between the two data frames associated with the D435 and T265 cameras. The difference between the attached locations of the cameras on the vehicle can contribute errors if not properly accounted for.

III. Methodology

3.1 Overview

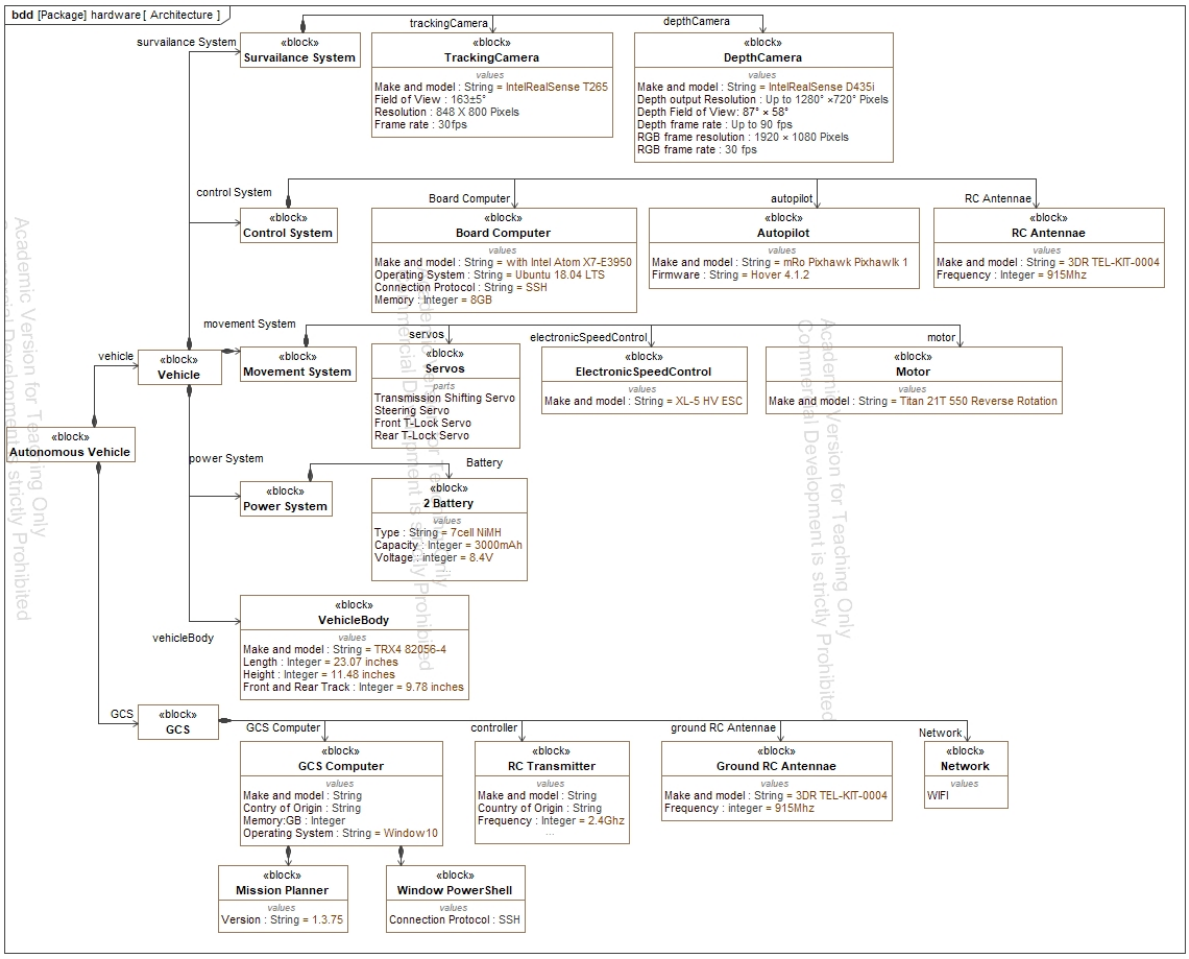


Figure 3. Architecture View

The overall architecture view of the vehicle for this research is shown in Figure 3. This research uses a four wheel ground vehicle that follows a planned route and avoids obstacles in its path using a depth camera and a tracking camera. The planned route is a simple set of ordered waypoints specified in the reference frame of the tracking camera. Finally, when the Atom board recognized the vehicle reached the final waypoint, the computer stops the vehicle and calculates the positional error from

the vehicle's current position to the final waypoint. The calculated distance is always positive because the vehicle approach is nondirective.

3.2 Hardware Description

The vehicle for this research was previously built for another thesis. Additions to the existing vehicle included an attached board computer, an additional battery to power the board computer, a depth camera, and a tracking camera. The vehicle is shown in Figure 4.

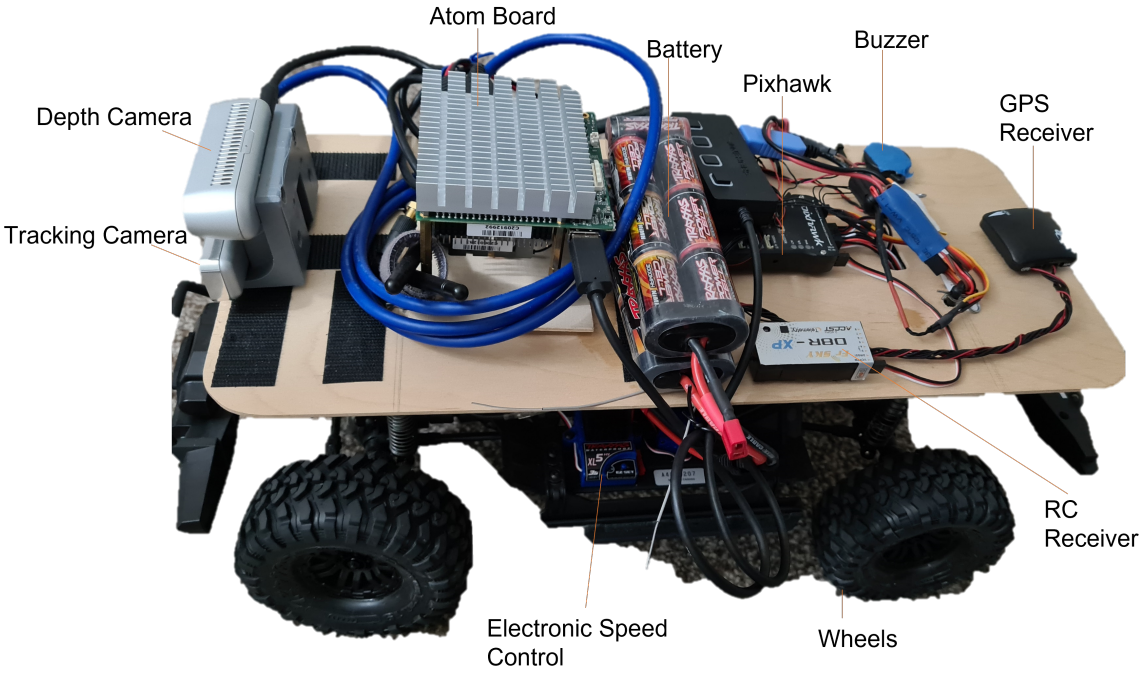


Figure 4. Vehicle Hardware

The depth camera and the tracking camera are fixed using a camera mount in the front of the vehicle aligned with the vehicle's longitudinal (forward) axis. Because the Pixhawk autopilot IMU data was not being used these experiments, the offset distance between the autopilot and the tracking camera did not need to be accounted

for. The tracking camera has its own internal IMU. The camera mount designed by the Intel RealSense team was 3D printed. The mount for both cameras is given with an extrinsic matrix for referencing and/or transforming their data [15]. The depth camera, Intel D435i, has an IR projector, left and right stereo cameras, and an RGB camera. Both stereo cameras take pictures for the depth information used to produce point clouds of objects in the image [5]. The tracking camera, Intel T265, has two fisheye cameras for generating coordinates and SLAM. The ATOM board produces move commands using Python code developed as part of this research, and the move commands are sent to the Pixhawk autopilot through the USB interface [16].

The Python code is initiated using a ground control computer via Wi-Fi using SSH protocol. When the move command reaches the Pixhawk, the Pixhawk holds the move commands until two conditions are solved. The first condition is controlled by the manual safety switch which must be armed. The second condition is the vehicle control mode must be manual or guided for activation [16]. The manual control mode was used for this research. The vehicle mode can be changed by the ground control computer using RF modems at 915 MHz or an RC transmitter and receiver pair operating at 2.4 GHz. The vehicle condition can be monitored using the Mission Planner application at the ground computer.

3.3 Software Description

3.3.1 Overview

Some of the code used in this thesis was posted on GitHub [17]. Beyond that existing code, eight functions were developed in Python for showing images and saving data from the tracking and depth cameras, and making a move command for vehicle movement. The software functions for these processes are shown in Figure 5. The move command functions required a connection between the ATOM board and the

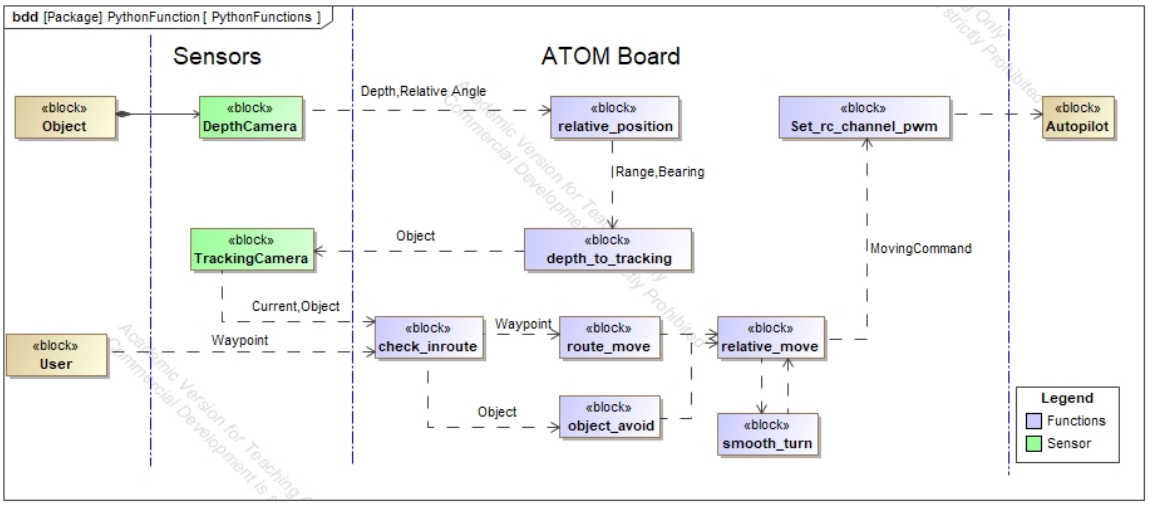


Figure 5. Python Functions

Pixhawk autopilot, and this was accomplished using Dronekit and Pymavlink libraries [16] [18]. All commands to the autopilot are in the form of a pulse width modulated (PWM) signal. Three commands are required for movement: a turn command which moves the steering servo, a transmission command which ensures the vehicle stays in low gear, and a throttle command which goes to the motor speed controller. A test was conducted to check the vehicle start and stop movements.

Code for image frame processing of the depth and tracking cameras had to be developed. This involved displaying the video from both cameras, and capturing and saving pictures from the cameras for evaluating after experiments. These routines are based on Python code samples provided by the Intel RealSense team [19]. The stored data includes image frames from both cameras, point cloud files from the tracking camera, time, position coordinates, velocity, acceleration, vehicle heading, current goal's coordinate, object's coordinate, and coordinate of an object's edge. This code used NumPy, Pyrealsense2, OpenCV, Time, NumPy, matplotlib, PLY, and JSON libraries.

Additional code was developed to grab objects from the images, draw a box around the objects, extract depth data from the grabbed objects, and transform this infor-

mation to the tracking camera coordinate system. This code will be discussed further in the "Object Detection Code" section of this chapter.

The vehicle was operating in a manual control mode; therefore, code needed to be written to facilitate autonomous waypoint following. Initially this involved moving to a commanded waypoint and stopping when it reaches a threshold distance from that waypoing, with the threshold set to 0.3 meters for this research. Too small of a threshold can be incompatible with the turn radius of the vehicle. If there is more than one coordinate goal, the coordinate goal is changed to the next waypoint when the distance threshold to the current goal is met. The RELATIVE MOVE function makes move commands utilizing the RELATIVE POSITION and SMOOTH TURN functions. The RELATIVE POSITION function returns relative angle and distance from a certain point to vehicle position and vehicle's heading by using the vehicle's coordinated position and a certain point. The SMOOTH TURN function returns the vehicle's steering angle relative to the vehicle's heading. In the RELATIVE MOVE function, the turn direction and wheel steering angle is determined using data from RELATIVE POSITION and SMOOTH TURN functions.

Finally, object avoidance code was developed and tested. A standoff distance and direction for avoidance maneuvers had to be established. This code shall be discussed further in the Object Avoidance Code section of this chapter. Because the depth camera attempts to grab all things in the image frame regardless of how close they are, an IN ROUTE function was developed to compensate for that feature. The IN ROUTE function checks the current position and current goal to determine if an object is within 50 centimeters of the path from the current position to the current goal.

3.3.2 Object Detection Code

The main steps of the VO processing are as follows: first, taking sequenced images; second, feature detection; third, feature matching or tracking; fourth, motion estimation; and fifth, local optimization [9]. The object detection code covers all VO processing steps. The vehicle can avoid objects when it detects and tracks them from image to image continuously. This research started with code to detect colored balls, a feature based code[20][21]. The feature based code sometimes missed objects from image to image, so depth based code was added. The feature code and depth based code was combined to make the algorithm more reliable.

Feature Based Code

Feature Based Code is based on the color ball code [20] [21]. After taking pictures with the depth camera, the images take four steps of image processing in the color ball code to find the object's outline and draw a box on the detected object. The first step is to store the frameset for later processing at step 3. Second, set the color and extract the depth data corresponding to the color using the 'bitwise' function of the Opencv operation by comparing the images from the color frame in the frameset. From step one to step two are VO's feature detection and feature matching or tracking. Third, proceed with the image processing to find the absolute difference between step 1 frameset and step 2 frameset. This step is the same as VO's motion estimation step. In this step, it uses five functions of Opencv; absdiff, cvtColor, GaussianBlur, threshold, and dilate function. The absdiff function is calculating the difference between images [22]. The cvtColor function (COLOR BGR2GRAY) changes the image to gray because the next function requires gray scale images [23]. The GaussianBlur function is blurring images with various low pass filters for getting smooth images [24]. The threshold function is to obtain a distinct image regardless of the amount of

light within the image frame [25]. The dilate function is useful for filling in the image of objects to remove gaps [26]. The last step is to find the outline of an object from the step three result using the findContours library [12].

Two thing were changed from the above steps. First, color was deleted in order to detect objects regardless of their color. Considering operational environments, it is not possible to designate an object as having one specific color. The color ball code is based on the set color range with RGB for image processing of OpenCV [12] for detecting objects. The ball code rejects the other object based on the color range. However, the image processing of openCV finds the object and extracts the four points on the border of the object from the image frames with or without color. Therefore, it is possible to detect all color objects. The second modification was to add Otsu's Binarization threshold method for the third step. Otsu's Binarization enhanced the detection features [25]. It was a found in earlier test, that Otsu's method provided more continuous tracking of objects, and a better match of the actual objects outline.

Depth Based Code

The feature-based code grabs all the objects but draws the contoures of only one object. When multiple objects are detected in the image frame, the object chosen to draw contours of can change from image to image. It is useless for tracking the object and avoiding the object. For this reason, it required the addition of the depth based code. The depth based code uses the shortest depth length inside of 3 meters in the image frame. The code determines the height and width of an object by finding pixels with depth information less than or equal to twenty percent of the shortest depth in the frame. Three filters were added for improving the depth accuracy. The three filters are a Spatial Filter, Temporal Filter, and Hole Filling filter. The Spatial Filter is a fast implementation of Domain-Transform Edge Preserving Smoothing. The Temporal Filter implements basic temporal smoothing and hole-filling. It is meaningless

when applied to a single frame, so it captures several consecutive frames. The Hole Filling filter offers an additional layer of depth extrapolation [27].

Comparison of Feature and Depth Based Object Detection

Object detection uses two methods which are the depth based method and the feature-matching method. Both methods attempt to draw boxes around objects in the image. The Depth based method is more stable when grabbing the object, but it takes more time when drawing the box and does not draw precise contours. The feature-matching method is less stable when grabbing the objects, but it is faster and produces a more precise contour. When using only the feature based method, obstacles were sometimes missed.

Data Transfer Between Depth and Tracking cameras

When the main code is started, the tracking camera establishes the origin position and generates its own coordinate system, and the depth camera starts VO processing. When the detection of features and extraction of depth data from the image frame is completed, the data is ready to transfer and use to generate move commands. Transferring data from depth camera to tracking camera is performed by the DEPTH TO TRACKING function. This function includes generating relative angle and distance and then generating coordinates in the tracking camera coordinate system. The relative angle and depth are calculated using this pixel point position, depth camera pixels, and field of view. This is facilitated by the extrinsic camera matrix provided by the Intel RealSense team for the camera mount [15]. The matrix, which is 3 by 4, allows transformation from the tracking camera frame to the depth frame ($T^{depth}_{tracking}$).

The matrix consist of rotation matrix ($R^{depth}_{tracking}$), which is a 3 by 3 matrix, and translation matrix ($t^{depth}_{tracking}$), which is a 3 by 1 vector. The translation matrix is the relative distance from the axis center of the tracking camera, on each axis, to the equivalent axis center of the depth camera. The rotation matrix is made of the tracking and depth camera axis direction. The depth camera and tracking camera both are using a right-hand rule coordinate frame, but the directions are different from each other. For the depth camera and IMU coordinate frame, the positive x-axis points to the right, the positive y-axis points down, and the positive z-axis points forward. For the tracking camera coordinate frame, the positive x-axis points to the right, the positive y-axis points up, and the positive z-axis points backward. For the tracking camera IMU frame, the x-axis and z-axis directions are opposite but the y-axis direction is the same as the tracking camera coordinate frame [28]. The transformation matrix from the depth camera to tracking camera ($T^{tracking}_{depth}$) is defined by adding the inverse matrix of rotation matrix ($R^{tracking}_{depth}$) and the translation matrix from the depth camera to the tracking camera ($t^{tracking}_{depth}$).

$$_{3\times4}T^{tracking}_{depth} = \left[{}_{3\times3}R^{tracking}_{depth} |_{3\times1} t^{tracking}_{depth} \right] \tag{2}$$

3.3.3 Object Avoidance Code

Initially, the Object Detection code was catching too many objects, many of which were not along the path. To correct for this, a 3 meter range was established for object avoidance. If the detected object is further than 3 meters from the vehicle, the code draws the box on the detected object and gets the depth information, but it waits before it generates an avoidance command. The code determines and saves the coordinates of detected objects and the edges of the object when the detected object is within 3 meters. The code checks the current detected target and previously

detected target interference with the vehicle movement using the IN ROUTE function. If either one of them interferes with movement, avoid the obstacle first.

The object avoidance code uses the object's center coordinate and the object's edge coordinate for avoidance. The coordinates of the center of the object and the object's edge are generated from the Object Detection code. Whether to grab coordinates of the left or right edge of an object depends on where the object's center is located in the image frame. When the center of the object is located on the left side of the camera frame, the code saves coordinates of the center and left edge coordinate of the object. And then calculate the avoid clearance point using depth data, coordinate of object edge, and a pad for minimum clearance. To avoid the object, the vehicle turns right until the left edge of the object in the camera frame meets the offset limit. When the left offset limit is reached, the vehicle is rotated in the opposite direction to place the detected object within the image frame. The offset limits have both side bumpers for compensating calculation time limitations. The turn directions are shown in Table 1. The vehicle continues with an avoidance turn and a recover turn for

Object location	**Left**	**Left limit**	**Center**	**Right limit**	**Right**
Turn Direction	Turn left	Turn right	Turn right	Turn left	Turn right

Table 1. Turn Directions

placing an object within the image frame. When the x or z coordinate of the vehicle reaches the x or z coordinate of the object, the vehicle breaks out of the avoidance turn to continue movement towards the current goal waypoint. If the vehicle loses the object, the code checks to determine if the coordinates of the previously detected object is on the path by using the IN ROUTE function. If the previously detected object interferes with the vehicle's movement, the vehicle moves to avoid the object

by the required clearance distance calculated from the object edge.

3.4 Experimental Test Plan

3.4.1 Evaluation Metrics

The evaluation metric for this research is the distance from the final vehicle position to the final goal waypoint. The vehicle follows the given route and avoid obstacles when they are detected and attempts to stop at the final waypoint. The Atom board saves the current position using the coordinates when the tracking camera frame is updated. From this, the error distance between those coordinates and the final waypoint is calculated.

3.4.2 Design Variables

The vehicle is tested indoors with four independent variables: distance, path circuits, path shape, and obstacle avoidance. The path shapes are triangular and square. These routes were chosen to provide variation in the turn angles required. The presence of obstacles means when the vehicle detects objects, the vehicle must avoid them and then return to the following the desired path. The vehicle is tested with two different path leg lengths, and different numbers of circuits of the path. Path leg length will be kept short to ensure battery voltage does not very widely during a trial or across different trials.

3.4.3 DOE Matrix

We used the Design Of Experiment (DOE) matrix for analyzing and assessing the results. The DOE matrix helps us to understand the influential variables and interactions among variables. The DOE matrix allows us to analyze and quantify the

effect of the outputs of the variable. In this research, the DOE matrix has 4 variables, with each variable having 2 values, total of 16 factors.

IV. Results and Analysis

4.1 Overview

Using indoor experiments, the distance from the vehicle to the final waypoint is used to measure accuracy of navigation around a closed path. For these measurements we set 4 path variables; path shape, number of circuits, path leg length, and presence of objects. Each variable has 2 values for a total of 16 factors. Each combination of factors was tested with 6 repetitions. The experiments were analyzed based on object detection, route and SLAM analysis, statistical analysis, qualitative analysis and the DOE matrix for evaluating accuracy. The DOE matrix with statistical results is shown as Table 2. A more complete set of results are shown in Appendix A.

4.2 Object Detection Analysis

The camera has to track continuously the object which is closest to the vehicle in order to avoid it. Feature based and depth based code are used for tracking objects. The feature based code was not able to track continuously the objects during early tests. The depth based code tracked objects continuously. In Figure 6, The red box is the actual object boundary, which was added after the experiments to improve understanding of code behavior. The green box is the detected object. The green box with depth information on the top is produced by the feature based code, and the green box with depth information on the bottom is produced by the depth based code. The feature based code catches the floor as an object in the top left picture of Figure 6, detects another part of the floor in the top right picture, detects another spot in the bottom left picture, and grabs the light reflection in the bottom right picture. In all of these images, the boxes detected by the feature based code do not overlap the object. The depth based code is able to continuously track the object.

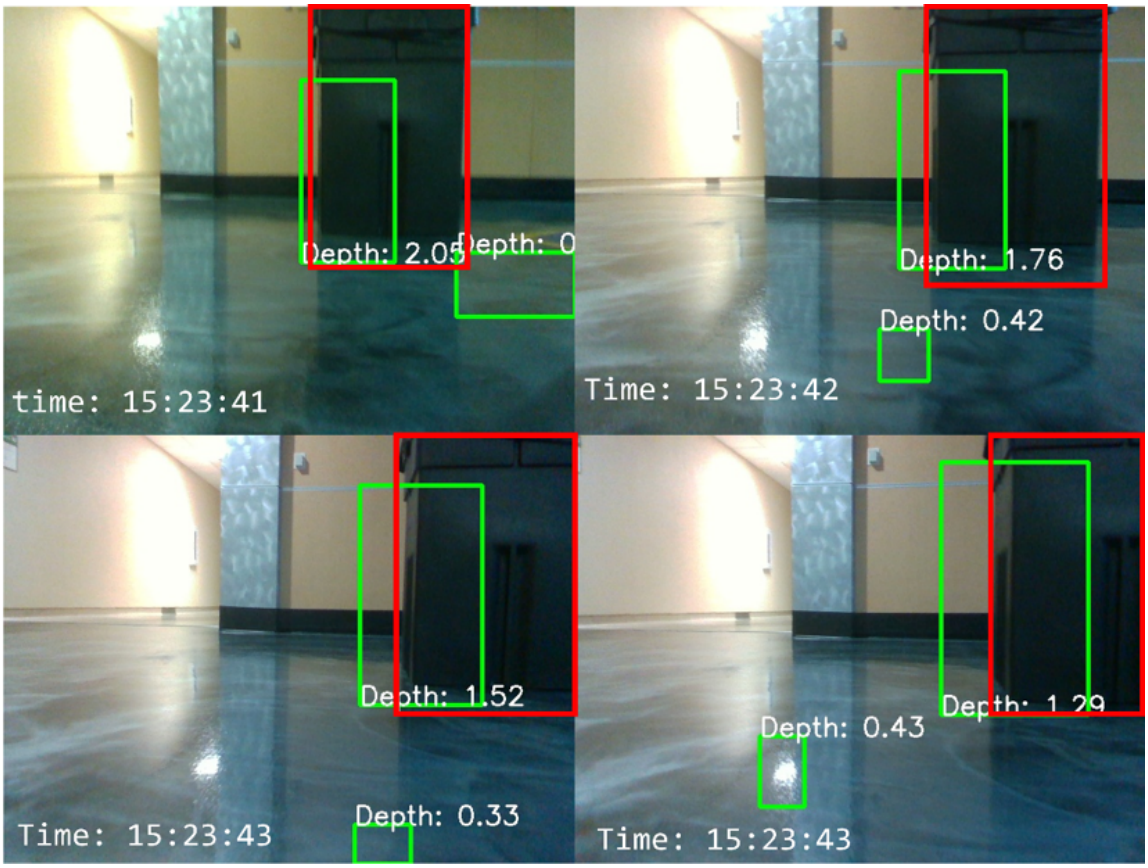


Figure 6. Feature Detection

The green box from the depth based code does not perfectly align with the real object, but it is mostly within the object boundary. The closer the camera is to the object, the better the green box size fits the actual object outline. Even for the image with the smallest green box, upper left of Figure 6, the coordinate location of the detected object is enough to avoid it. At the indoor test before the lobby experiment and without vehicle movement, the green box from depth based code had matched the real object's contour. The other green box from the feature based code was better matched the with the real object's contour but sometimes not stable from image to image. The detection abilities of both codes were weak in the presence of bright lights and reflections.

4.3 Route and SLAM Analysis

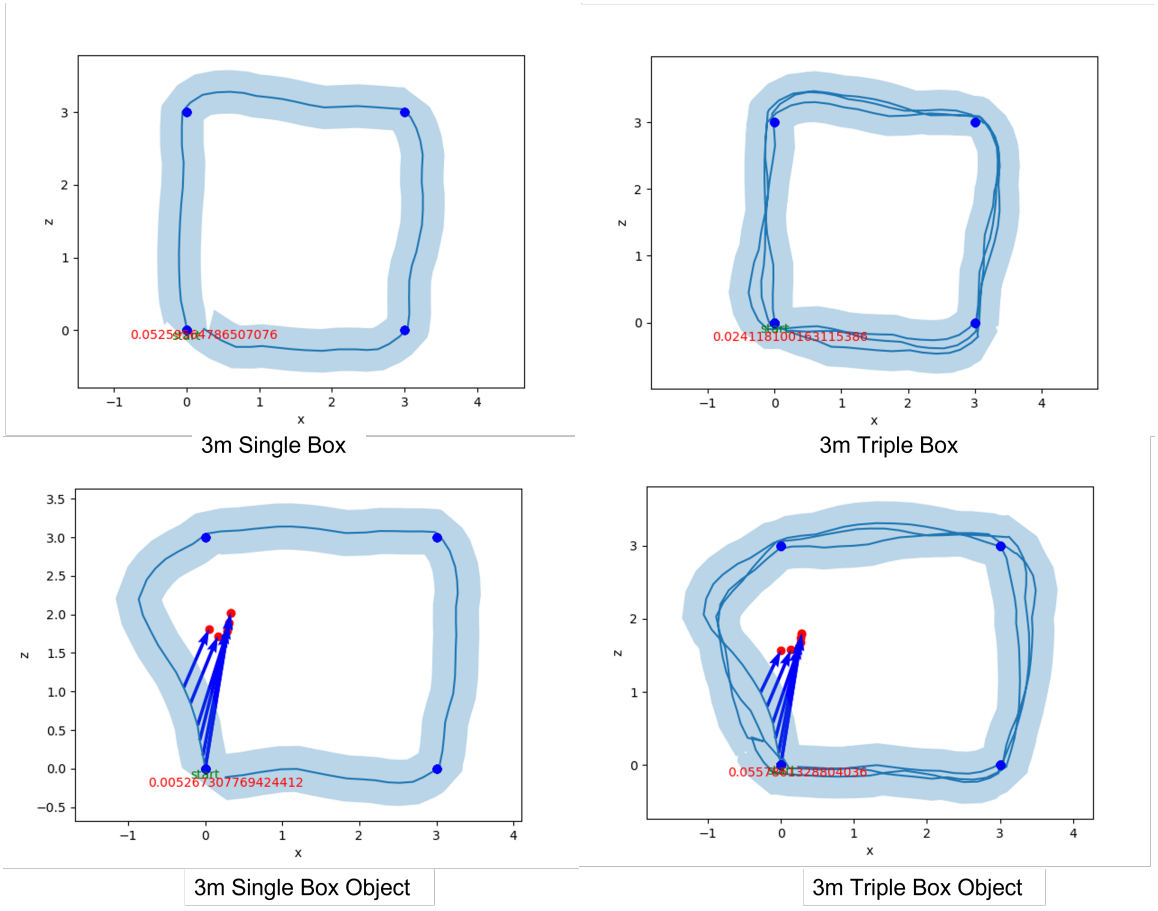


Figure 7. 3 Meters Box

Figure 7 shows the 3 meter box routes base on the computer generated position information. The other routes are shown in Appendix B. The upper two pictures are without enroute objects and the bottom two pictures are with an enroute object. In the bottom pictures, the blue arrow points from the vehicle to the object when the vehicle detects an object within a set distance of its path, left or right 50 centimeters. The red dots are detected edges of the object. The light blue shade depicts the +/- 30 cm corridor on either side of the intended path. The vehicle was able to follow the route in order and properly stop at the starting point, the final goal. In the upper left pane of Figure 7, the vehicle started at the bottom left and proceeded in a clockwise

direction, returning to the starting point. The red text is the distance from the final goal to the vehicle when the vehicle stopped. For cases when the camera detects an object, it saves coordinates of points on the boundary of the detected object. If another object is not detected, it checks the saved points to determine if the object will interfere with the intended vehicle path from its current position to the next waypoint. If the saved object points interfere with the movement of the vehicle, the vehicle uses the set clearance to define a path around the object. This is shown in the bottom right pane of Figure 7.

The vehicle should gain more accurate position information using SLAM updates when the vehicle makes a closed-loop path [9]. This corrects for the drift errors of the inertial measurement unit of the tracking camera which increase with time. The corrections resulting from the closed loop path should more than compensate for the drift errors associated with the inertial measurement unit, thus improving accuracy for closed loop paths. However, this requires the building of global map to realize the improved accuracy for close loop paths. For reason associated with memory limitation and computing power a global map was not constructed for these experiments.

4.4 Statistical Analysis

As shown in Table 2 , the vehicle positioning error at the final waypoint is 6.88 centimeters on average. The maximum positioning error is 10.49 centimeters which occurs for the 3 meter box pattern with three circuits and the presence of obstacles. The minimum is 4.91 centimeters for the 5 meter triangle pattern with one circuit. Analyzing the skewness column of Table 2, it is seen that most of the values are positive. Positive skewness means that the mean value of the data set is greater than the median value of the data set. And the mode value of the data is less than the median of data set. There are three cases that resulted in negative skewness values.

Factor	Minimum (cm)	Maximum (cm)	Mean (cm)	Std deviation	Skewness
3m/Single Box	0.18	13.07	7.25	4.37	-0.52
3m/Triple Box	2.41	19.20	8.57	6.11	1.19
3m/Single Triangle	0.17	19.50	6.67	7.36	1.28
3m/Triple Triangle	0.58	14.63	5.20	4.91	1.83
3m/Single Box Object	0.29	15.29	6.45	6.64	0.70
3m/Triple Box Object	5.58	19.82	10.49	5.09	1.44
3m/Single Triangle Object	3.41	15.55	7.74	5.27	0.99
3m/Triple Triangle Object	2.27	12.31	7.04	4.18	0.04
5m/Single Box	1.00	15.16	6.24	5.52	0.79
5m/Triple Box	0.17	18.56	8.43	6.39	0.53
5m/Single Triangle	1.52	15.65	4.91	5.47	2.09
5m/Triple Triangle	0.10	9.92	5.20	3.74	-0.28
5m/Single Box Object	0.19	15.89	7.31	6.55	0.07
5m/Triple Box Object	1.39	19.39	8.70	6.86	0.84
5m/Single Triangle Object	0.03	14.64	7.70	5.98	-0.39
5m/Triple Triangle Object	5.69	11.93	8.66	2.54	0.17

Table 2. Statistic Description

The negative skewness means that the mean of the data set is less than the median of the data set. In graphical terms, positive skew indicates that the tail is on the right. In cases where one tail is long. The standard deviation has an average of 5.44. This indicates that the distribution does not follow the standard normal deviation.

4.5 Qualitative Analysis

Qualitative data gathered during the experiment provided ideas regarding possible experimental errors and what should be changed if the test was to be repeated. All experiments were conducted in the same area, a building lobby, in order to remove a possible confounding variable associated with varying frictional coefficients of the floor. The lobby was not very large, so there were times when the camera mistook the wall for an object. In order to avoid this, the route should stay at least 50 centimeters

from the wall. The lobby is not large enough to accommodate a 5 meter box pattern while maintaining the required standoff distance from the wall. For this reason, the box pattern was changed to a 5 x 3 meter rectangle. For the triangular pattern it was necessary to align the vehicle with the first leg of the triangle to accommodate the limited turn radius of the vehicle relative to the leg length available.

4.5.1 Quantitative data

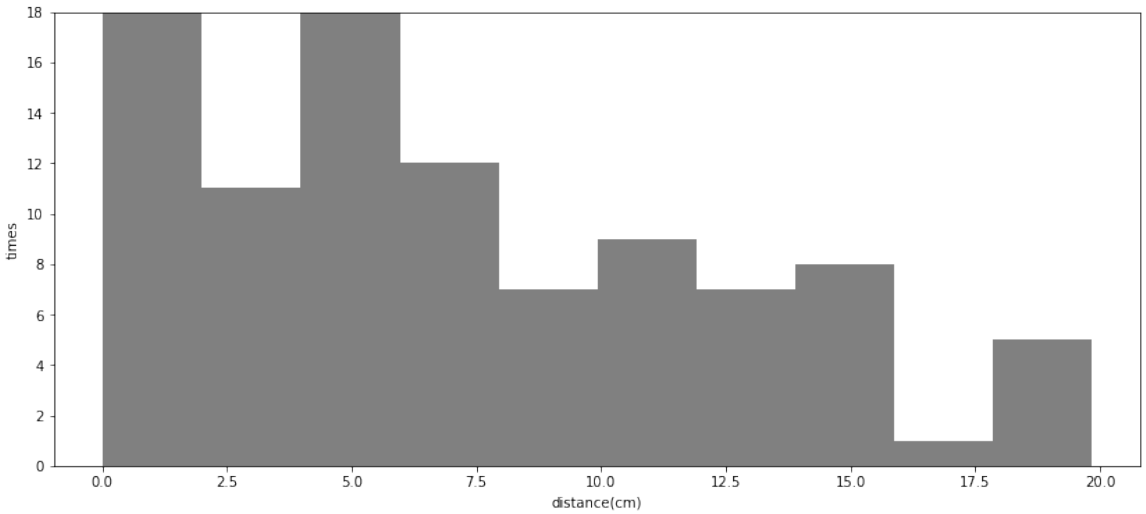


Figure 8. Histogram

The histogram presented in Figure 8, shows the positional errors are concentrated from 2 to 5.5 centimeter. Considering the average of final position error, 6.88 centimeters, the distance frequently lies below the average. This shows the distribution of the experiment's result does not follow the standard normal distribution. Frequent lower-than-average values indicate that most of the skewness values are positive as would be expected based on the small amounts of error and the fact that the error is only measured in a positive sense.

Symbol	Variables	(+)	(–)
A	Length	5 meters	3 meters
B	Circuit	Triple	Single
C	Route	Triangle	Box
D	Object	Yes	No

Table 3. DOE matrix index

factor	A	B	C	D	Y-BAR (cm)
1	-1	-1	-1	-1	7.25
2	-1	1	-1	-1	8.57
3	-1	-1	1	-1	6.67
4	-1	1	1	-1	5.20
5	-1	-1	-1	1	6.45
6	-1	1	-1	1	10.49
7	-1	-1	1	1	7.74
8	-1	1	1	1	7.04
9	1	-1	-1	-1	6.24
10	1	1	-1	-1	8.43
11	1	-1	1	-1	4.91
12	1	1	1	-1	5.20
13	1	-1	-1	1	7.31
14	1	1	-1	1	8.70
15	1	-1	1	1	7.70
16	1	1	1	1	8.66
(+)	7.14	7.79	6.64	8.01	
(–)	7.42	6.78	7.93	6.56	
delta	0.28	-1.00	1.29	-1.45	

Table 4. DOE matrix

4.6 DOE Analysis

The DOE matrix index for the experiment is shown in Table 3. A value of -1 indicates the low value for the variable, whereas a value of +1 indicates the high value for the variable. Using these indices, the results of the DOE analysis are shown in the Table 4. All experimental distance results were calculated by the code and

appear in the final column of the table. The last column of Table 4, Y-BAR, is the experiments' average distance from the waypoint for the test point associated with the set of factors for a given row. For instance, the second row of Table 4, factor 1 row, are all '-1'. It is interpreted by the Table 3 as 'test at 3 meters in length of one leg of the box pattern route with a single circuit and no objects'. For the last three rows of the Table 4, the (–) row is the experiment's distance results average of the column corresponding to the -1 values, the (+) row is the experiments' distance results average of the column corresponding to the +1 values, the delta row is the (–) row minus the (+) row. The average positional error associated with (–) of symbol A, 3 meter leg length, is larger than (+), 5 meter leg length. This result appears to be related to steering error, and the ability of the vehicle to correct for these errors given a longer leg length. The single circuit had lower final error than the three circuit trials, and the triangle route had lower positional error than the square route. This was an unexpected results and can not be explained from the observed data. Had a global map been constructed, it would be expected that the three circuit trials would have reduced error versus the single circuit trials. Finally, the positional error was lower when no objects were encountered than it was with detected objects.

The absolute value of the delta row represents how the positional error was impacted by the factor represented in the column. If the absolute value of delta for a specified column is larger than deltas associated with other columns/factors, it suggests the factor associated with the specified column has a greater impact on the experimental results. The most impactful independent variable in our experiments is object avoidance. The absence of an object on the intended vehicle route resulted in a lower positional error result than the cases where an object was present. When the vehicle followed a route without an object, the positional error was 1.45 centimeters less than that achieved for the route with an object. The least impactful variable

is the leg length of the path. The trials showed that the longer path length only improved the positional error by .28 centimeters.

V. Conclusion

5.1 Overview

The objective of this research was to demonstrate autonomous navigation using VO and SLAM from depth and tracking cameras combined with a commercial autopilot on a ground vehicle without GPS signals in an unknown area. Three investigative questions are used to outline this objective.

How can depth and tracking cameras be integrated with a low cost, commercial autopilot to support autonomous navigation without the aid of GPS? The depth and tracking cameras were integrated using Python code. The data from each camera can be transferred to another camera by using a transition matrix, which was defined for this research. An Atom processor board generates the movement command using data from the cameras, and it sends it to a Pixhawk autopilot using Mavlink and Dronekit. The vehicle was able to avoid objects and follow a specified set of waypoints around a closed path.

How do factors such as path geometry, distance, path circuit and obstacle avoidance impact the accuracy of navigation? The presence or absence of objects was demonstrated to have the greatest impact on final positional error upon arrival at the terminal waypoint. The leg length of the route was demonstrated to be the least impactful factor on positional error. It should be noted, however, that the average positional error for all test points was approximately 7 cm.

Is the demonstrated navigation system appropriate for vision-based guidance of SUAV using low-cost componentry? The time for calculating one cycle of code takes about 0.5 sec. This will be a limitation for utilization on a SUAV if the code can not be made to run faster because the SUAV requires a more frequent update of the movement commands for path following or obstacle avoid-

ance. This thesis does, however, demonstrate how to combine a depth camera, a tracking camera, and an autopilot. Further research will find a way to decrease the computational times for using on a SUAV.

5.2 Recommendation for Further Research

Further research must consider the impact of uneven lighting and sunshine. The capability of the depth camera depends on the brightness of the light. The depth camera sometimes catches the bright lights or reflections, causing it to miss an actual object. The test area floor used in this research is slightly reflective (shiny). The depth camera detected objects well on cloudy days or when the sun was high because the sunlight did not reflect much. Other times the floor reflects sunlight, degrading the depth capability. The depth camera would see the sunlight as an object and take action to avoid it.

It takes, on average, about 0.5 to 0.6 seconds to compute the entire code once. Considering the vehicle's velocity, 30 to 33 centimeters per second, when one cycle of code is complete, the vehicle moves 15 to 17 centimeters. During this latency period, the vehicle is traveling blindly, and is not able to determine where it is relative to a waypoint. Vehicle velocity could be reduced but this would required additional memory to save images for a given trial. Computing time needs to be decreased using shorter lines of code or upgrading the processor board. Adapting the code for parallel processing would also improve processing time.

Both the depth data based code and the feature based code have their drawbacks. The depth based code depends on the brightness of light. The feature based code is not stable for certain objects. It is possible that adding code for color range or restricting object shapes will help detect objects.

The computer board saved PLY files, which include point cloud data, and JSON

files with position data of the vehicle. A single PLY file shows point cloud data at a certain time. By combining PLY files over all time steps, it should be possible to make a whole scanned 3D map for more accurate navigation. The current research effort was not able to do that in the time available for the effort. In this research, only compiled 3 points of positional data and point cloud data like Figure 9 using Meshlab program [29].

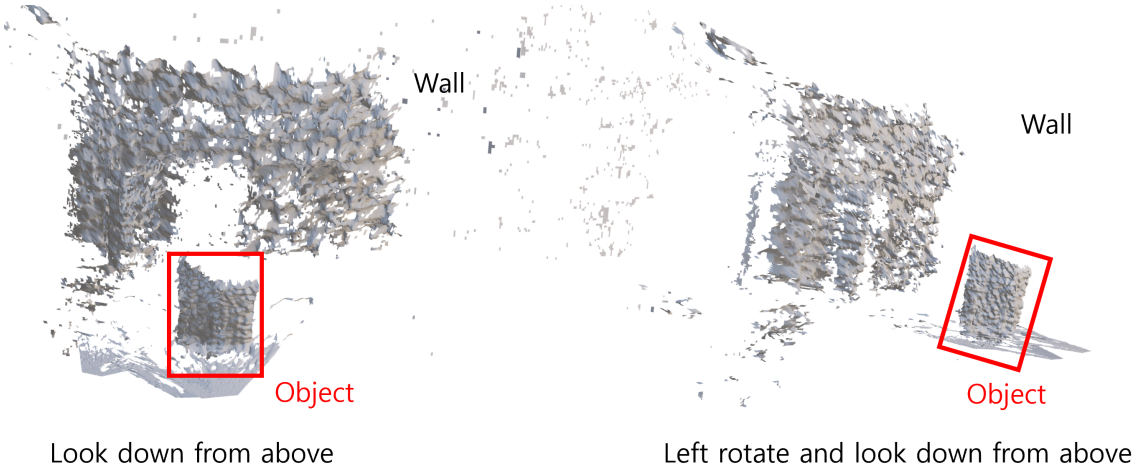


Figure 9. SLAM data

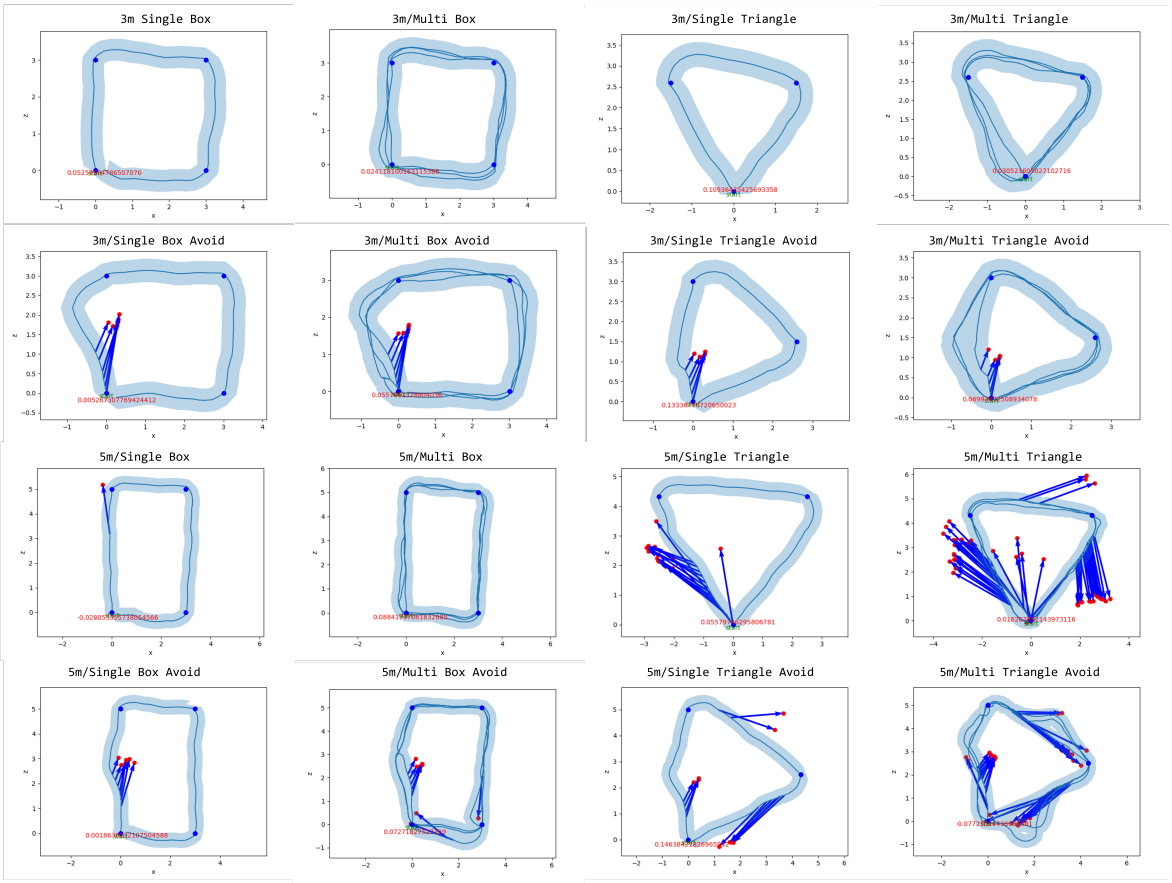
Appendix A. Distance Results

(cm)

Factors	1	2	3	4	5	6	Mean
3m/Single Box	5.26	7.49	9.98	13.07	7.50	0.18	7.25
3m/Triple Box	4.49	11.75	5.62	2.41	7.94	19.20	8.57
3m/Single Triangle	5.44	10.94	19.50	1.61	2.35	0.17	6.67
3m/Triple Triangle	5.76	14.63	3.05	3.78	3.38	0.58	5.20
3m/Single Box Object	0.53	0.29	4.05	15.29	4.37	14.15	6.45
3m/Triple Box Object	6.63	11.78	9.87	5.58	9.28	19.82	10.49
3m/Single Triangle Object	4.40	13.34	4.50	5.23	3.41	15.55	7.74
3m/Triple Triangle Object	2.49	7.11	6.99	12.31	11.06	2.27	7.04
5m/Single Box	1.44	8.99	8.02	1.00	2.81	15.16	6.24
5m/Triple Box	4.72	12.20	18.56	0.17	8.84	6.11	8.43
5m/Single Triangle	15.65	2.50	5.58	2.62	1.52	1.57	4.91
5m/Triple Triangle	0.10	9.92	1.62	8.06	6.00	5.48	5.20
5m/Single Box Object	15.89	0.38	12.21	0.19	10.62	4.59	7.31
5m/Triple Box Object	14.59	1.39	4.33	5.24	7.27	19.39	8.70
5m/Single Triangle Object	11.08	7.33	0.03	1.21	14.64	11.91	7.70
5m/Triple Triangle Object	9.24	6.29	5.69	7.73	11.10	11.93	8.66

Total Average = 6.88 cm

Appendix B. Total Routes



Bibliography

1. N. Boston. (2021) Mass. company sends robot to help florida building collapse rescue efforts. [Online]. Available: https://www.nbcboston.com/news/local/mass-company-sends-robot-to-help-florida-building-collapse-rescue-efforts/2415473/ [Accessed: 2021-6-26]

2. GPS.gov. (2021) Gps accuracy. [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/ [Accessed: 2021-11-2]

3. wiki. (2021) Ackermann steering geometry. [Online]. Available: https://en.m.wikipedia.org/wiki/Ackermann_steering_geometry [Accessed: 2021-10-21]

4. Traxxas. (2021) Trx4. [Online]. Available: https://traxxas.com/products/landing/trx-4/ [Accessed: 2021-11-1]

5. I. Corporation. (2021) Intel realsense d400 series product family datasheet. [Online]. Available: https://dev.intelrealsense.com/docs/intel-realsense-d400-series-product-family-datasheet [Accessed: 2021-11-2]

6. A. T. Inc. (2021) Up squared specifications. [Online]. Available: https://up-board.org/upsquared/specifications/ [Accessed: 2021-12-6]

7. C. C. A. 3.0. (2021) Ubuntu install of ros melodic. [Online]. Available: http://wiki.ros.org/melodic/Installation/Ubuntu [Accessed: 2020-3-25]

8. . C. L. Ubuntu and C. are registered trademarks of Canonical Ltd. (2021) Psa for ros users: Some things to know as python 2 approaches eol. [Online]. Available: https://ubuntu.com/blog/

psa-for-ros-users-some-things-to-know-as-python-2-approaches-eol [Accessed: 2019-10-28]

9. D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

10. C. C. A.-S. License. Visual odometry. [Online]. Available: https://en.wikipedia.org/wiki/Visual_odometry [Accessed: 2021-6-2]

11. K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An overview to visual odometry and visual slam: Applications to mobile robotics," *Intelligent Industrial Systems*, vol. 1, no. 4, pp. 289–311, 2015.

12. doxygen. (2021) Structural analysis and shape descriptors. [Online]. Available: https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0 [Accessed: 2021-12-21]

13. E. Tsykunov, V. Ilin, S. Perminov, A. Fedoseev, and E. Zainulina, "Coupling of localization and depth data for mapping using intel realsense t265 and d435i cameras," *arXiv preprint arXiv:2004.00269*, 2020.

14. J. Bayer and J. Faigl, "On autonomous spatial exploration with small hexapod walking robot using tracking camera intel realsense t265," in *2019 European Conference on Mobile Robots (ECMR)*. IEEE, 2019, pp. 1–6.

15. Intel. (2021) Intel RealSense Developer Documentation. [Online]. Available: https://dev.intelrealsense.com/docs/depth-and-tracking-cameras-alignment [Accessed: 2021-11-1]

16. A. D. Team. (2021) Rover. [Online]. Available: https://ardupilot.org/rover/index.html [Accessed: 2021-11-1]

17. H. Kim. (2022) Thesiscode. [Online]. Available: https://github.com/hongseok-kim-afit/GroundVehilceDepthTracking [Accessed: 2022-2-21]

18. I. 3D Robotics. (2021) Dronekit python. [Online]. Available: https://github.com/dronekit/dronekit-python/ [Accessed: 2021-2-18]

19. I. RealSense. (2021) Python. [Online]. Available: https://dev.intelrealsense.com/docs/python2 [Accessed: 2021-4]

20. M. Aboulhair. (2021) Tracking depth ball realsensel515. [Online]. Available: https://github.com/michael-ab/Tracking_Depth_ball_RealSenseL515/blob/master/realsense_tracking_ball.py [Accessed: 2020-7-9]

21. ——. (2021) Realsense l515 - simple object tracking + depth. [Online]. Available: https://www.youtube.com/watch?v=eeKkrWj55Bg [Accessed: 2020-9-1]

22. doxygen. (2022) Operations on arrays. [Online]. Available: https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga6fef31bc8c4071cbc114a758a2b79c14 [Accessed: 2022-1-19]

23. ——. (2022) Color space conversions. [Online]. Available: https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html [Accessed: 2022-1-19]

24. eastWillow. (2016) Smoothing images. [Online]. Available: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html [Accessed: 2016]

25. e. OpenCV. (2016) Image thresholding. [Online]. Available: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html [Accessed: 2021-7-13]

26. doxygen. (2022) Morphological transformations. [Online]. Available: https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html [Accessed: 2022-1-19]

27. I. R. Team. (2021) depth filters. [Online]. Available: https://github.com/IntelRealSense/librealsense/blob/jupyter/notebooks/depth_filters.ipynb [Accessed: 2021-7-13]

28. I. RealSense. (2021) How-to:getting imu data frome d435i and t265. [Online]. Available: https://www.intelrealsense.com/how-to-getting-imu-data-from-d435i-and-t265/ [Accessed: 2019-2-4]

29. P. Cignoni and A. Muntoni. (2022) Meshlab. [Online]. Available: https://www.meshlab.net/ [Accessed: 2022-1-25]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2022	Master's Thesis	Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Ground vehicle navigation with depth camera and tracking camera

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hongseok, Kim, Major, ROKAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of System Engineering (AFIT/ENV)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENV-MS-22-M-217

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Munitions Directorate

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RW

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The aim of this research is to provide autonomous navigation of a 4 wheel vehicle using commercial, off-the-shelf depth and tracking cameras. Some sensitive operations need accuracy within a few inches of navigation ability for indoor or outdoor scenarios where GPS signals are not available. Combination of the Visual Odometry(VO), Distance-Depth(D-D), and Object Detection data from the cameras can be used for accurate navigation and object avoidance. The Intel RealSense D435i, a depth camera, generates depth measurements and the relative position vector of an object. The Intel RealSense T265, a tracking camera, generates its own coordinate system and grabs coordinate goals. Both of them can generate Simultaneous Localization and Mapping (SLAM) data. The cameras share their data to provide a more robust capability. Combining the Intel cameras with a Pixhawk autopilot, it was demonstrated that the vehicle can follow a desired path and avoid objects along that path.

15. SUBJECT TERMS

Visual Odometry(VO), Simultaneous Localization and Mapping (SLAM), Visual aided navigation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. David Jacques, AFIT/ENV
U	U	U	UU	52	19b. TELEPHONE NUMBER *(include area code)* (937) 255 6363 x3329; david.jacques@au.af.edu