2018 International Conference on Unmanned Aircraft Systems (ICUAS)
Dallas, TX, USA, June 12-15, 2018

Autonomous Collision Avoidance System for a Multicopter using Stereoscopic Vision

Erwin Perez, Alexander Winger, Alexander Tran, Carlos Garcia-Paredes, Niran Run, Nick Keti, Subodh Bhandari, and Amar Raheja

Abstract—**This paper presents the use of stereoscopic vision as a means of sensing and detecting obstacles and other aircraft for collision avoidance system for a multicopter. A ZED stereo camera is mounted on a DJI S900 Hexacopter UAS to generate depth maps. A NVIDIA Jetson TX1 board is used for onboard processing of the depth maps and collision avoidance algorithm. The board communicates with the Pixhawk autopilot, which transmits data to the ground control station via XBee radios. By using the ZED software development kit (SDK), it is possible to obtain depth maps directly from the camera and use them in the implementation of obstacle avoidance. The algorithm that is used partitions the depth maps into multiple sections in order to find the section of the image that represents the objects furthest away. This selected section is the section that is free of obstacles. The hexacopter can maneuver itself to that selected section autonomously, thus avoiding obstacles in its path. The developed algorithm and flight test results are discussed.**

I. Introduction

Unmanned aerial systems (UASs) are one of the fastest growing sectors of the aerospace industry. UASs have potential to replace manned aircraft for many applications such as surveillance of disaster-hit areas, search and rescue, border patrol, precision agriculture, package delivery, traffic monitoring, power line and pipeline inspections, etc. However, despite these potentials, the use of UASs for commercial application is still limited. One of the reasons for this limited application is the lack of collision avoidance capabilities. Developing and implementing these capabilities can help integrate UASs more seamlessly into the National Airspace System (NAS) with fewer safety concerns.

Many solutions have been proposed for collision detection and avoidance system for UASs. Automatic dependent surveillance-broadcast (ADS-B) transponders, LIDARs, camera vision using optical flow are some of the sensors or methods for collision detection [1-6]. These sensors and methods are useful, but either are expensive, heavy or have other limitations. For examples, LIDARs are expensive and heavy, and may not be suitable for small UASs. ADS-B transponders work only for cooperative UASs i.e. the UASs that also are equipped with ADS-B transponders [1-2].

This paper presents the use of stereoscopic vision as a collision avoidance system for UASs. Stereoscopic vision provides a more affordable and lightweight solution for collision detection. However, the stereoscopic vision system must be able to detect and avoid obstacles sufficiently fast enough, meeting the equivalent level of safety (ELoS) requirement, which requires that UASs be able to detect the obstacles and take corrective measures at least four seconds before the collision [7]. The presented method uses a C++ library and an open source library, OpenCV, to develop collision detection and avoidance algorithms. A combination of special computer algorithms and open source libraries were used for various parts of this project. C++ was used for the algorithms while the Open Source Computer Vision (Open CV) and Stereolabs libraries were used for the disparity images. Stereoscopic vision uses two cameras to generate two-dimensional images while simultaneously utilizing a computer algorithm known as Block Matching to generate a sense of depth perception in a blended image known as a disparity map [8, 9].

The rest of the paper is organized as follows. Section II talks about the stereoscopic vision. Hardware used for the project including the unmanned aerial vehicle platform is presented in the third section. Section IV presents the communication between the onboard processor and the autopilot. Collision avoidance algorithm is discussed in Section V. Flight test results are presented in the sixth section followed by the conclusion and future work in the last section.

II. Stereoscopic Vision

Stereoscopic vision uses two cameras located a set distance apart, mimicking the human eyesight. The ideal stereovision system, shown in Fig. 1, shows how the two image planes are coplanar to each other and the projection points, p_l and p_r, are on the same row. In stereoscopic vision, it is important that the projection points are leveled on the same pixel row, otherwise an error in data may propagate. The current setup uses a special stereo camera that encompasses both camera lenses on a single camera module.

Erwin Perez is an undergraduate student in the Department of Aerospace Engineering at Cal Poly Pomona (e-mail: erwin.p721@yahoo.com).

Alexander Winger is an undergraduate student in the Department of Computer Science at Cal Poly Pomona (e-mail: ajwinger@cpp.edu).

Alexander Tran is an undergraduate student in the Department of Aerospace Engineering at Cal Poly Pomona (e-mail: avt@cpp.edu).

Carlos Garcia-Paredes is an undergraduate student in the Department of Aerospace Engineering at Cal Poly Pomona (e-mail: Paredes@cpp.edu)

Niran Run is an undergraduate student in the Department of Aerospace Engineering at Cal Poly Pomona (e-mail: nrun@cpp.edu).

Nick Keti is an undergraduate student in the Department of Aerospace Engineering at Cal Poly Pomona (e-mail: nketi@cpp.edu).

Subodh Bhandari is a Professor in the Department of Aerospace Engineering at Cal Poly Pomona, Pomona, CA 91768 (phone: 909-869-2612; e-mail: sbhandari@cpp.edu).

Amar Raheja is a Professor in the Department of Computer Science at Cal Poly Pomona (e-mail: raheja@cpp.edu).

A. *Image Rectification*

The purpose of image rectification is row-alignment, which means making the corresponding projection points to be on the same pixel level by introducing epipolar geometry [8-10].

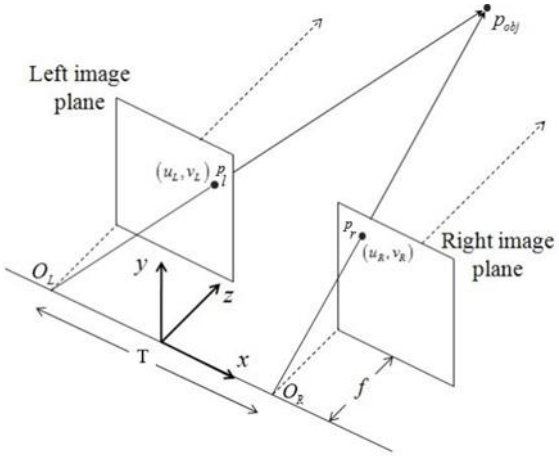


Figure 1. Stereoscopic vision geometry.

In epipolar geometry, shown in Fig. 2, the two rectangles represent the image plane of the left camera and right camera, respectively. In this image, the image planes are not perfectly aligned in a parallel and coincident manner and the correction for this is done through the use of epipolar geometry [11]. In the image, X represents the points captured by both cameras and O_L and O_R are optical centers. X is projected onto the left image plane as point X_L and projects on the right image plane as point X_R. Connecting the points O_L and O_R, we obtain a line called the baseline. The intersecting points of the baseline and two image planes are called epipoles, e_L and e_R. The lines formed by X_R and e_R is called epipolar line. A second epipolar line is formed from X_L and e_L. The process of image rectification aims at making both epipolar lines parallel in order to make X_R and X_L on the same pixel row, and thus reducing the propagation of error [8].

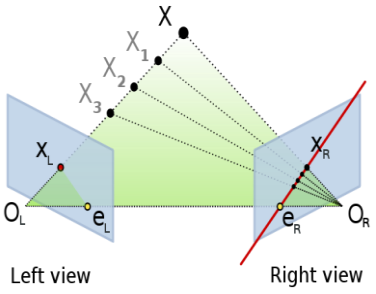


Figure 2. Epipolar geometry.

B. *Disparity Map Generation*

Given an object located at point P, represented in Fig. 3, in the physical world, the projection of this point onto each of the two camera planes will appear to be in a different location due to the distance between the two cameras. These two projection points, x^l and x^r, seen in Fig. 3, represent the respective portrayal of point P onto the left and right cameras, respectively. The difference in location, *d,* known as disparity, will be different depending on the distance, Z, of the object. For our applications, point Z represents the distance from the UAS, at point T, to a potential collision. The distance f represents the focal distance of the particular cameras in use. Also, (u_L, v_L) and (u_R, v_R) are points on the left and right image planes, respectively, which are obtained as follows.

$$\begin{bmatrix} u_L \\ v_L \end{bmatrix} = \frac{f}{z}\begin{bmatrix} x-(-\frac{T}{2}) \\ y \end{bmatrix} = \frac{f}{z}\begin{bmatrix} x+\frac{T}{2} \\ y \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} u_R \\ v_R \end{bmatrix} = \frac{f}{z}\begin{bmatrix} x-\frac{T}{2} \\ y \end{bmatrix} \quad (2)$$

Using Eq. (1) and Eq. (2), we have

$$u_L - u_R = \frac{fT}{z} \quad (3)$$

and Z, which represents the distance to any tracked object at a given point P.

$$z = \frac{fT}{u_L - u_R} \quad (4)$$

If the observed point is closer to the image planes, the disparity will be large [9], and vice versa. Hence, by using this method, disparity is utilized for determining the relative distance between objects and the cameras based on the disparity, or the calculated shift of a point, on each camera plane [14]. As previously mentioned, stereoscopic vision is a mimicry of human eyesight. By closing one eye, one can see that an object appears to be in a specific location. By switching the closed eye, the object will appear to shift to a different location. By repeating this experiment with the object at a much further distance, the shift of the object will be much smaller. This is the concept of disparity [13, 15].

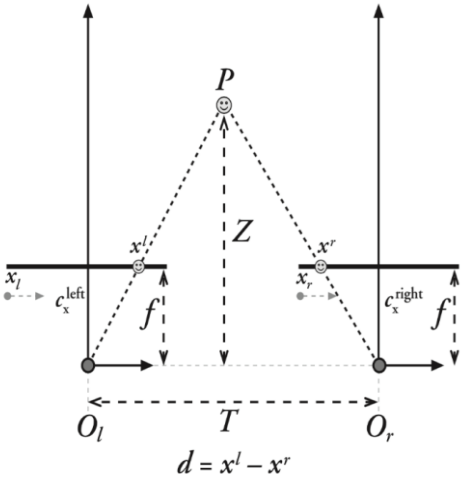
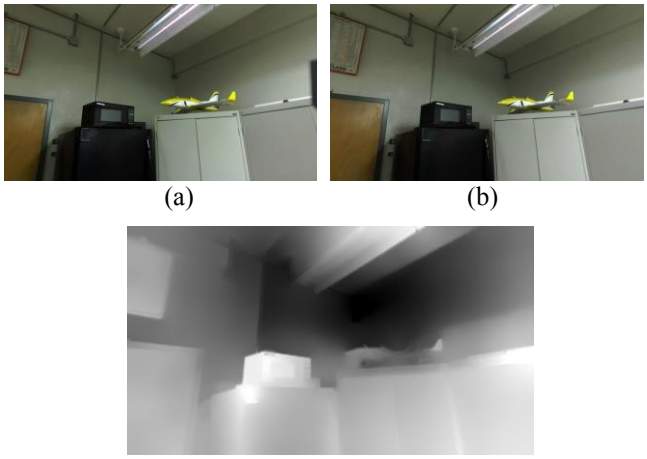


Figure 3. Disparity map generation.

Using this concept, a disparity map is generated as a grayscale image. The algorithm used in disparity map generation is generally known as block matching [8, 9]. The block matching algorithm can be found in open-source libraries and modified for different project applications. The algorithm generates a disparity map by finding a pixel in one of the camera images, locating the matching pixel in the other camera image, and calculating the disparity between those individual pixels. The disparity is quantified by a numerical

value ranging from 0 to 255. In this method, a value of 0 represents black and 255 represents white. A pixel with a higher numerical value is appointed to pixels with a larger disparity, which corresponds to pixels of objects that are close to the camera frame, and will appear a shade of white in the disparity map. The objects that are further away will have a lower numerical value and will appear a shade of gray. The further an object is, the darker it will appear in the disparity map. This process is repeated for the entire image and a blended grayscale image is created from both of the camera images. This concept is portrayed in Fig. 4, where Fig. 4(a) and 4(b) are the left and right camera images, respectively, and 4(c) is the disparity map generated from blending the former two.



(a) (b)

(c)

Figure 4. Left and right image capture with disparity map.

III. HARDWARE

A. UAV Platform

The UAV platform used for this project is the DJI S900 Spreading Wings Hexacopter, which is shown in Fig. 5. The hexacopter was chosen due to its light weight and good flying qualities. The UAV weighs around 7 lbs. without payload, and its six rotors allow it to fly for about 20 minutes while fully loaded, allowing enough time to generate the disparity maps. It can carry a payload of more than 9 lbs. The UAV is equipped with a Pixhawk autopilot, NVDIA Jetson TX1 processor, and ZED stereo camera. The UAV has two rails centered under the rotor hub, which is essential for the mounting of the hardware used. Two 3-D printed mounts were built to house the camera and flight computer.

Figure 5. DJI S900 Spreading Wings hexacopter.

The onboard computer and its mount are centered under the rotor hub and the ZED stereo camera is in front of it. Fig. 6 below shows the UAV in a flying field prior to the flight tests.

Figure 6. Equipped hexacopter ready for flight tests.

B. Stereoscopic Camera

The ZED Stereo Camera, shown in Fig. ,7 is being used for this project. There are various video resolutions that can be handled by the ZED camera, and include 2.2K at 15 FPS (frames per second), 1080p at 30 FPS, and 720P at 60 FPS. Although the video mode is easy switchable within the ZED camera, the current frame rate of 60 FPS is used due to the number of decisions that could be made each second through each frame. The ZED's two camera lenses are spaced 4.7 inches apart, which allows for an effective range of 5 to 60 feet for usable stereo depth information. The spacing of the camera is done in a way to imitate the way human eye function and gives it the capability of large scale depth perception, 3-dimensional mapping, and 6-axis position tracking. The camera requires a host computer, which is important as a higher processing capability is required to process the quadrant selection algorithm simultaneously with the ZED stereo camera. A CUDA-capable (Compute Unified Device Architecture) host computer, Jetson TX1, with a NVIDIA GPU is used in order to process the images. The Software Development Kit (SDK) that comes with the camera runs on Linux and is responsible for image rectification.

Figure 7. ZED stereo camera.

Position tracking is the ability to estimate the object's position relative to its surroundings. The ability to track the 6-axis position (3 rotational and 3 translational) makes the camera well suited for the project. As the camera moves around its surrounding, the new positions and orientations are reported, by the ZED API, to the host computer, and is called 6-degrees-of-freedom (6-DoF) pose. The amount that's reported per second gives the camera its framerate. The higher the framerate, the more positions and orientations it can track and calculate. The position is reported in an X (forward positive), Y (right positive), and Z (down positive) vector space. The orientation is represented in a ϕ, θ, and ψ vector space, which represent roll, pitch, and yaw, respectively. The

Binocular vision, which the two eyes give, give the ZED stereo camera its depth perception. However, the ZED does this by capturing high-resolution 3D video and estimates its depth by comparing the number of pixels displaced in the left and right of the image. Furthermore, by continuously scanning its surroundings, it is creating new 3D maps. The map is updated at the same rate as the framerate, and the map data is saved in a fixed reference coordinate frame. One thing to note is the accuracy and range relationship. Increasing the range decreases the accuracy. Which one is preferred and chosen is mission dependent, and a balance is always suggested between the two.

C. Communication Hardware

FPV radio telemetry modules are used for sharing information between the UAV and ground station. The system consists of air and ground modules as shown in Fig. 8. The telemetry modules utilize the 915MHz bands and provide a full-duplex link using HopeRF's HM-TRP modules running custom and open source firmware. The system is capable of the air data rates of up to 250kbps with the range of approximately one mile.

Figure 8. FPV radio telemetry modules.

D. Flight Controller

The Pixhawk flight controller, seen in Fig. 9, is the onboard flight controller for the hexacopter. The Pixhawk consists of a PX4-FMU controller and PX4-IO, which are optimized to provide control and automation using ArduPilot flight software. It consists of a 3-axis Micro L3GD20H gyroscope for orientation, a 3-axis accelerometer for position, and compass for heading data. A MS5611 barometric pressure sensor is used for determining altitude. The Pixhawk receives messages from the on-board computer, which it then processes, and sends the appropriate commands to the hexacopter.

Figure 9. Pixhawk PX4 flight controller.

E. Flight Computer

The Jetson TX1 Module, mounted on the developer kit as shown in Fig. 10, is used for stereovision algorithm. The TX1 module is built around the NVIDIA Maxwell™ architecture, which features 256 CUDA cores. The developer kit is responsible for processing the images received by the Zed Stereo Camera and generating the disparity map. It then applies an algorithm to the disparity map, which analyzes the disparity map, chooses the best direction of travel, and sends the appropriate commands to the flight controller.

Figure 10. Jetson TX1 developer kit.

IV. COMMAND AND CONTROL

The following paragraphs describe the communication between the flight computer, flight controller, and ground control station for the command and control of the vehicle.

A. MAVLink

Micro Air Vehicle Link or MAVLink, is a light weight library used to communicate with small unmanned aircraft. It is suited for applications that are resource constrained such as those with limited memory, RAM, and communication bandwidths. Though other interfaces such as C-UART for communication between the Pixhawk and the Jetson TX1 Module are possible, MAVLink is currently used for this project. MAVLink creates the bridge for waypoint protocols, mission protocols, parameter protocols, camera protocols, and command protocols to be transmitted. Messages are defined by XML files and converted to the appropriate source language. This allows messages to be transmitted from the on-board computer to the Pixhawk via an FTDI to USB cable adapter. The information passed through MAVLink include the ability to change the velocity, position, and altitude of the hexacopter. In addition, MAVLink can also be used to define a new waypoint to any list of waypoints that the UAS is following. Thus, a connection is established between the Pixhawk and Jetson, with MAVLink sending and receiving messages to get the hexacopter to avoid obstacles.

B. Mission Planner

Mission Planner software is used to set initial waypoints for the desired mission as well as to update waypoints required for collision avoidance in real-time. Mission Planner is a versatile software used in the ground station due to its full featured application already compatible with the Pixhawk. Mission Planner can be used as a configuration utility or as a dynamic control supplement for the autonomous aircraft [10].

Also, it is possible to plan, save, and load autonomous missions into the autopilot straight from the Mission Planner by simple point and click waypoint entry on interfaces such as Google Maps or any other mapping tool. Fig. 11 shows a snapshot of the interface.

Figure 11. Mission Planner interface.

With the telemetry hardware on the Pixhawk, it is possible to monitor the aircraft's status while it is in operation. It also records the telemetry logs. This allows to further view and analyze telemetry logs post-flight to ensure proper actions were executed by the UAS.

C. *DroneKit*

DroneKit is an Application Program Interface (API) that uses MAVLink messages to communicate between a companion computer such as the Jetson and a flight controller such as the Pixhawk. In essence, it acts as a third party "interpreter" for communication. DroneKit makes it easier to create the connection between the Jetson and the Pixhawk by using predefined functions along with the customizable functions that send MAVLink messages for the vehicle movement. The main issues with using the DroneKit lie in its exclusive compatibility with Python and Android platforms. Despite this issue, DroneKit is still a viable option because it is possible to create a Python interpreter in the C++ program that would allow Python scripts to run in the current code. These Python scrips would then call upon all of DroneKit's functionality.

Just like with MAVLink, there are many ways to send the commands to the Pixhawk through the use of DroneKit functions. One such function is the send_ned_velocity() function which generates the MAVLink message to change position, velocity, and acceleration based on the UAS's local frame. There are also functions to set a specific position for the UAS to move to.

V. Collision Avoidance Algorithm

A. *Overview*

Fig. 12 shows a flow chart depicting the breakdown of the processes needed to detect and avoid obstacles using a stereoscopic camera. First, the camera is initialized to ensure that it is ready and able to capture images. Next, the appropriate location of the sections for the disparity maps is pre-determined. The size and dimensions of these sections are calculated prior to any image capture because they are overlaid on the image. Hence, regardless of what is being depicted in the image, the appearance and dimensions of the sections would remain the same. From here, the ZED SDK is used to capture a depth image. Using the image along with the overlaid sections, the algorithm sums up the numerical values associated with the pixel density in each section, and a section with the lowest pixel density can be selected. This section is used to maneuver the UAS to if there is an obstacle in the current path of the UAS.

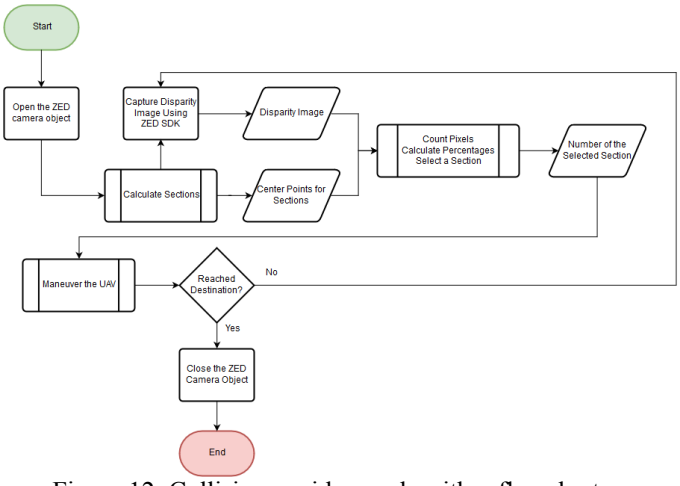


Figure 12. Collision avoidance algorithm flowchart.

B. *Section Calculations*

The algorithm uses 289 overlapping sections (17 rows x 17 columns) that are each 628 x 252 pixels in size. These sections are overlapping to give the UAS the best possible path to avoid an obstacle. When fewer number of sections are used, there may be portions of the images that are free of obstacles, but each section has some obstacles. Therefore, obstacle free selection is not possible.

The sections are numbered from 0-288, starting from the top left, and then moving to the right along the rows. The center location of each section can be saved using an array along with variables for the width and height of each section. The array can be used as an overlay for the section selection once a depth map is captured. The center coordinates of each section are saved. These coordinates are the ones that the vehicle is commanded to move to for obstacle free path.

C. *Section Selection*

To be able to determine which of the 289 sections mentioned above are free of obstacles, each section is assigned a counter. These counters keep track of how many pixels are closer than the given threshold. The threshold is the distance away from the ZED camera that an object can be before it is recognized as a threat. During flight tests, this threshold was set to thirty feet. This distance allowed for a UAS traveling at 5 mph to detect and avoid an obstacle within 4 seconds. The algorithm moves along each row and column, getting the value at each pixel. If the pixel is considered to be a threat, then every section that the pixel falls in has its counter increased by 1. Fig. 13 shows two sections (green is section 1 and blue is section 2). If the pixel at (x1, y) is seen

as a threat, the counter for section 1 is increased because it is only in that section, whereas if the pixel at (x2, y) is seen as a threat, the counter for both the sections 1 and 2 would be increased because the pixel falls in both sections.

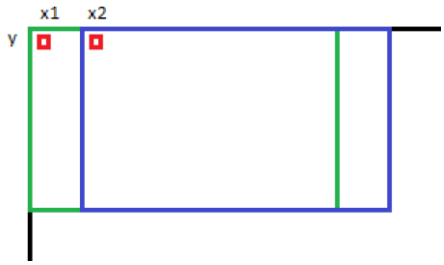
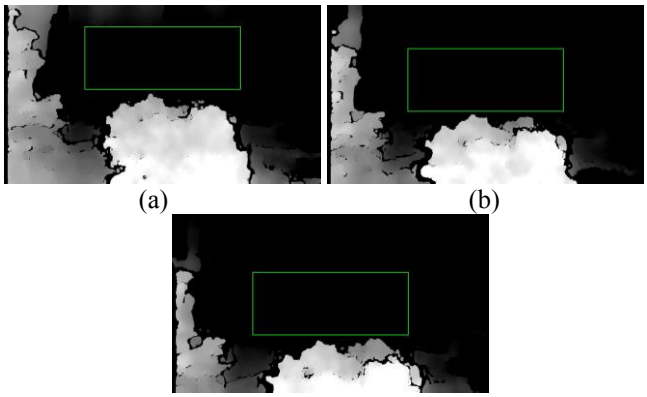


Figure 13. Two sets of pixels in separate sections.

Once all of the pixels have been checked, the percent of pixels within the given threshold is calculated for each section. The section with the lowest percentage is the section that is selected to be clear of the obstacles. If there are two sections that have the same percentages, then the section that is closest to the center of the overall image is selected. Fig. 14(a) to 14(c) is a progression of three sequentially generated disparity maps. These images show that the selected section is the closest obstacle-free section to the center of the overall image. As the UAS increases its altitude, the section starts to move towards the center of the image [12].



(a) (b)

(c)

Figure 14. Section selection as the UAS increases its altitude.

Once the section with the lowest percentage has been selected, the percentage is compared to a percentage threshold that is set to be at 15%. This percentage threshold is used to make sure that a section which is heavily populated with obstacles is not selected just because it has the lowest percentage. If the selected section has a percent higher than the percentage threshold, then all sections can be rejected. The number of the selected section is sent to the movement function so that the UAS can move to the obstacle free area. In the event that there is not a section that is clear, then a value of -1 is sent to the movement function instead.

D. Vehicle Movement Command

Once a section number has been received, a command is sent to the Pixhawk that will command the vehicle to move as desired. If the center section is selected, then the UAS will continue on its current path because there are no obstacles currently in its way. When a section other than the center is selected, a new, temporary position is selected, and the vehicle alters its original course. The movement command utilizes the go-to GPS coordinate function within the DroneKit API. The function is given a horizontal and vertical distance based on the position of the selected section. Trigonometric equations are used to calculate the appropriate distance to move in each direction. When the DroneKit movement function receives these distances, it calculates a new GPS coordinate based on the current position and heading of the UAS. The UAV then yaws in order to alight with this new GPS point, before moving to the location. If no sections are selected, then the vehicle will hover in place, and yaw until a new path can be chosen. When the UAS is moving to the calculated position, if another new position is determined to be a better solution, the UAS will adjust to moving to this better point. This allows for the program to continuously calculate the best route based on the UAS's current position.

E. OpenCV

Open Source Computer Vision library (OpenCV) was used to show the depth images produced by the ZED Stereo Camera SDK. This is important for testing the algorithm while in the lab or during flight tests. OpenCV allows for the visualization of the images. Functions for drawing geometric shapes, provided by OpenCV, were used to produce the rectangle on the image that indicates what section was being selected.

VI. RESULTS

Originally, two Point Grey Chameleon cameras along with an Intel NUC as the on-board computer were used. This system only could generate disparity maps at a rate of five decisions per second. The ZED stereo camera and the SDK provided by Stereolabs is able to generate and make decisions at a rate of about fourteen decisions per second, a 280% increase in efficiency. To be able to use Jetson TX1 with NVIDIA graphics processing unit, Jetson Development Pack (JetPack) was used. This automates the installation of the development environment by containing host and target tools, APIs, and other packages, along with making it easy to flash the latest OS images onto the Jetson system. Originally, the number of sections was 3x3 and the partition lines that were used were all drawn on the image. Fig. 15 shows what this looked like. These sections were of the same sizes and were not overlapping.

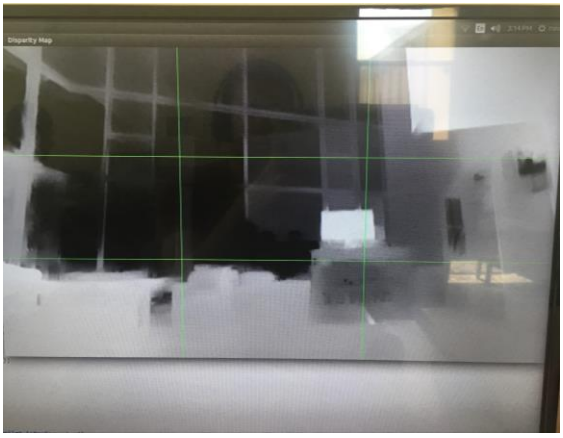


Figure 15. Original partitioning of a disparity image.

Partitioning in this way results in no section being selected as each section is shown to have obstacle despite some of the sections having no obstacles. This is due to the partitioning lines crossing through these open areas. This is seen in Fig. 16 where there is an open area along the top of the image. With 3 x 3 sections, the partitioning lines fall through the open space, and therefore, there are obstacles in each of the two sections.

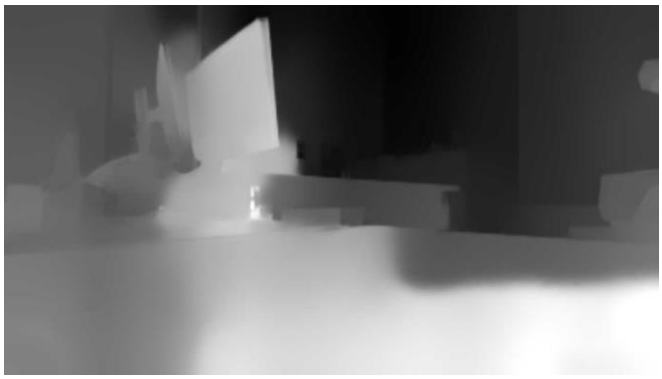


Figure 16. 3x3 sections with no section selected.

With the overlapping sections, there is a larger chance of a section being seen as free of obstacles. Fig. 17 shows the same scene as Fig. 16, with the same open area selected as a section this time, but there are 17 x 17 possible sections to choose from. Because of the overlapping sections, there is now a section located where there is an area free of obstacles. However, the algorithm is slower since it requires more section calculations.

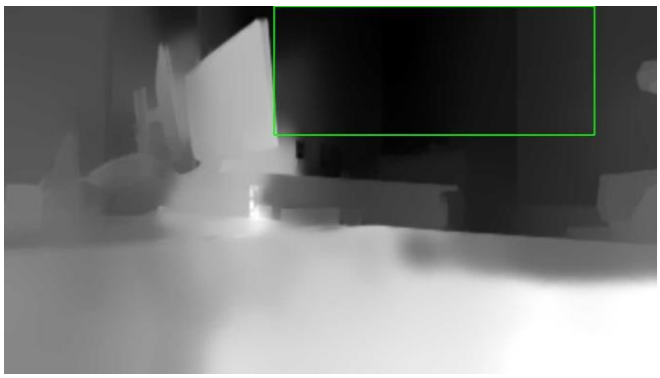


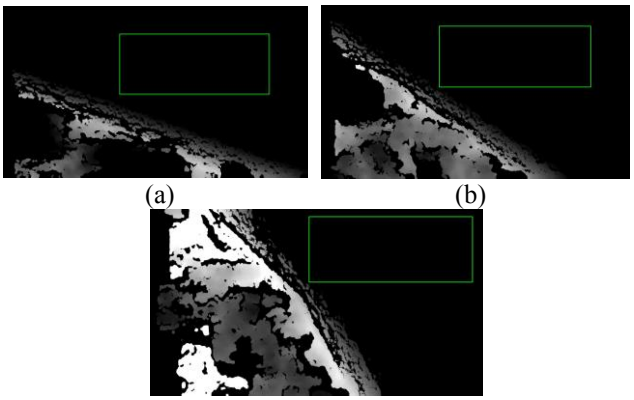
Figure 17. 17 x 17 sections with an empty section selected.

The number of sections used was determined by finding the number of decisions made per second. A decision is defined as an image captured by the camera that resulted in a section selection. Table 1 shows how many decisions were made per second for each tested number of sections. For each trial, there were 60 seconds worth of data points gathered. The table also shows that as more sections are used, number of decisions per second decreases, indicating a tradeoff between speed and accuracy. Based on the data, 17 x 17 was selected for the number of sections to be used.

Table 1. Number of decisions per second for a varying number of sections.

Number of Rows x Columns	Decisions per sec.			Average Decisions per sec.
	Trial One	Trial Two	Trial Three	
3x3	23.1	24.1	24.9	24.0
5x5	21.6	23.1	23.0	22.6
7x7	17.9	21.3	21.7	20.3
9x9	17.3	20.0	19.3	18.8
11x11	15.2	18.0	18.4	17.2
13x13	16.6	16.8	17.2	16.9
15x15	15.5	15.5	15.5	15.5
17x17	13.6333	14.7833	14.8667	14.4278
19x19	10.9833	10.9833	10.9833	10.9833
21x21	9.48330	10.1833	10.0500	9.90553
23x23	9.05000	9.01667	9.1333	9.06666
25x25	9.48333	9.20000	9.3000	9.32778
27x27	7.96667	8.03333	8.0500	8.01667

Several ground and flight tests were performed to test the hexacopter's obstacle avoidance capabilities. Ground tests were conducted to check the obstacle detection and section selection algorithm while the ZED was stationary and unarmed. During flight tests, the hexacopter was flown manually at low altitudes with the ZED Stereo Camera pointed toward various obstructions such as trees, bushes, and nearby vehicles. The goal of this flight test was to capture depth images while the UAS was moving to check if the algorithm was as efficient as when it was stationary. Fig. 18 shows a few of the depth images that were captured and the sections that were selected.



(a) (b)

(c)

Figure 18. Depth images and sections selected during first flight test.

The object that is seen in the bottom left corner of Fig. 18 (a) is the ground. It is seen to move closer and closer to the center of the image. The sections that were selected were the correct sections as they were free of obstacles and were avoiding the ground as it approached the center of the image. The section selection algorithm was able to correctly identify obstacles in an outdoor environment.

Two additional sets of flight tests were conducted. During the first set of flight tests, the UAS was manually flown around different obstacles. The UAS in flight can be seen in Fig. 19. Numerous depth images were captured. Upon post-processing, the depth images revealed that the correct sections were selected, confirming that the algorithm worked properly during a full flight. Fig. 14 above shows the depth images that were captured during one such flight test.

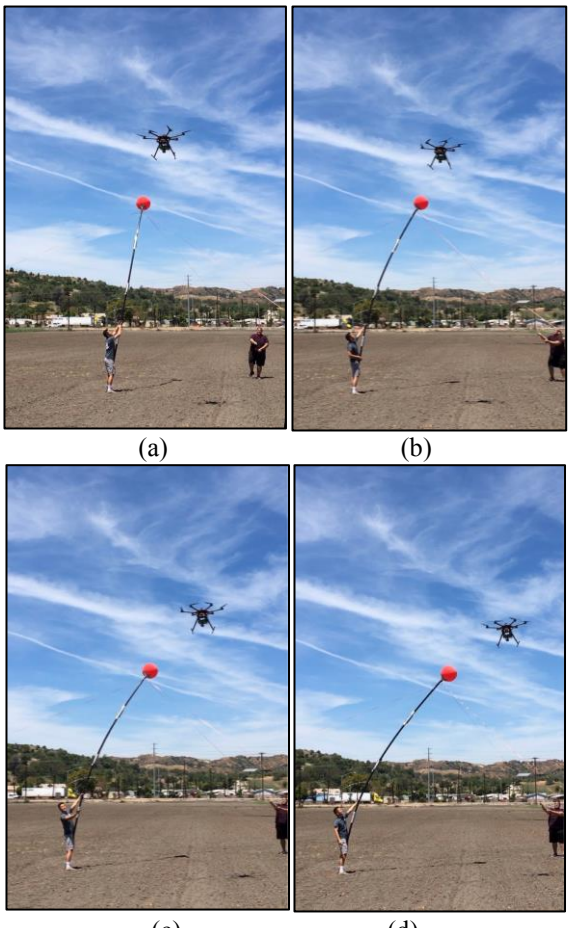
Figure 19. UAS in manual flight for data collection.

Flight test images reveal that the ZED Stereo Camera creates images with clarity such that obstacles can be distinguished. This clarity helps in the section selection as the edges of obstacles are more accurately identified, ensuring that the obstacles the UAS detects are more likely to be in their true location and not falsely represented by incorrectly marked pixels.

The second set of flight tests primarily focused on exploring the viability of using DroneKit as an alternative method of sending commands to the vehicle via MAVLink. To examine this, a basic takeoff function in DroneKit was first called using a python script. The script called for the vehicle to elevate to a low altitude of 1 foot above the ground and hover for 2 seconds. This simple command was used to verify that the DroneKit messages in our Python script were successfully able to connect the Jetson TX1 to the Pixhawk and direct it to perform a simple takeoff maneuver. After successfully taking off at the low altitude, it was established that the connection between DroneKit and the vehicle was made and further commands could be added to the script.

To follow up with this simple test, several Python scripts were written to test if the UAS would be able to move to a specific location. This was done by testing the vehicle's ability to move forward and backward, roll left and right, yaw side to side, change elevation and perform combinations of these movements entirely autonomously. Each maneuver was programmed in separate Python scripts through the use of DroneKit's waypoint commands to determine how the UAS moved in each direction. Specifically, a simple_goto command was used to set a waypoint using x, y, and z distances relative to the frame of the vehicle. The vehicle would then calculate and execute the maneuver needed to approach each waypoint defined in the script. Ultimately, each test was successful and the vehicle was able to interpret each movement command within the expected timespan and autonomously maneuver in all directions. Furthermore, the vehicle showed no signs of error or instability during the maneuvers and was capable of maneuvering at a steady velocity of 1 meter per second. Lastly, a significant achievement from these tests were the success of heading changes maneuvers, which eliminate the possibility of obstacles directly adjacent to the vehicle not being detected due to the limited field of view of the forward-facing ZED camera. Ultimately, this collection of flight tests demonstrated the ability to use DroneKit as a method of efficiently communicating the MAVLink commands to the vehicle's flight controller to maneuver the vehicle.

The latest flight test verified the ability of the system to autonomously detect obstacles and proceed to avoid them. The test setup involved placing five distance markers 10 ft. apart. A 20 ft. tall pole with a large balloon affixed at the top as seen in Fig. 20 was moved along the linear coned path towards the hovering UAS, which hovered at an altitude of 20 feet.



(a) (b)

(c) (d)

Figure 20. UAS autonomous collision avoidance

As the balloon approached within a pre-set threshold of 10 feet from the UAS, the UAS began detecting the balloon as a potential obstacle and moved away laterally. The balloon was the moved laterally, following the cone. The UAS continued to move away from the balloon obstacle.

As seen in Fig. 20, the balloon apparatus was stationed in one spot slightly to the left of the UAS, was gradually moved right into the direct line of sight of the ZED camera. As the obstacle moved closer toward the vehicle's direct line of sight, the UAS responded by maneuvering laterally to the right, a. This initial autonomous collision avoidance test verified the system's functionality in a safe and controlled manner. Although the UAS autonomously took-off and avoided the balloon obstacle, the UAS operator was always ready to switch the flight mode to manual flight and take control of the UAS in the event of any unexpected movements. Several test runs were conducted successfully without requiring the operator to take control of the vehicle.

VII. Conclusion and Future Work

From the results of the aforementioned flight tests, it is promising that obstacle detection and avoidance can be performed efficiently, at an affordable cost, and with a minimal footprint for small UASs. Using the ZED Stereo Camera, depth maps were successfully generated to analyze the hexacopter's environment and detect obstacles and their relative distances from the UAS. Moreover, the method of partitioning the image captured by the cameras into a grid of 17 rows x 17 columns of sections has been an effective solution for allowing the algorithm to make decisions based on the disparity map. By integrating the image partition and the section selection algorithm developed during the course of this project, a reliable and efficient method of recognizing and selecting alternative flight paths to avoid the obstacles has been developed. This is corroborated by flight test results which indicated that an average of 14.4 decisions are made per second for selecting a section to navigate toward. The use of 17 rows x 17 columns has proven to serve as the most balanced partitioning dimension for the section selection algorithm with respect to efficiency and accuracy.

In addition, numerous flight tests have demonstrated the successful implementation of the section selection algorithm and the DroneKit's movement commands to avoid obstacles. During the most recent flight test, the UAS detected a balloon as an obstacle and successfully responded by maneuvering laterally in order to avoid the obstacle.

Future work will focus primarily on further developing and optimizing the collision avoidance algorithm. The scope of this optimization phase will include improving the efficiency and accuracy of the collision avoidance algorithm with respect to how many decisions can be made per second and how accurately the vehicle can detect various obstacles. With the movement commands successfully integrated with the collision avoidance algorithm, several options for how the UAS will respond to obstacles will be explored, including the use of waypoint navigation, changes in elevation, changes in heading, or any combination of these maneuvers. These options will be compared in a trade study for the optimum solution for efficient and safe collision avoidance.

Although the current utilization of 17 x 17 quadrants for the section selection algorithm has proven to be efficient, there is still room for improvements to accuracy and speed. Currently, the section selection algorithm utilizes a pixel density threshold of 15% for determining the ideal section to move. This means that any section with a total pixel density above 15% would be actively avoided. This threshold was selected to eliminate ambiguous "noise" from the environment. However, it has demonstrated some small levels of inaccuracies in obstacle detection, particularly for very small obstacles. More tests will have to be conducted to find the ideal threshold while still eliminating noise. In addition, methods of accounting for small obstacles will be explored as well. Another area of focus in future work will be the use of multithreading. Rather than having the algorithm analyze pixels one thread at a time, the ability to analyze multiple threads of pixel would greatly improve decision-making speeds and efficiency in theory. Furthermore, another parameter that will be examined for optimization is the frequency at which images are captured. This frequency could play a role in how quickly decisions could be made and how well the decision-making algorithm would be able to operate at higher flight velocities. Moreover, implementing an adaptive system that actively throttles the pixel density threshold as a function of flight velocity could also possibly improve the efficiency of the section selection process. This method could be an improvement over the current method of utilizing a single, hard-coded threshold to use indefinitely.

The refined and optimized algorithm will then be tested in more complex scenarios that will include multiple obstacles in the UAS's flight path.

References

[1] F. Martel., R. R. Schultz, et al., "Unmanned Aircraft Systems Sense and Avoid Avionics Utilizing ADS-B Transceiver," *AIAA Infotech@Aerospace Conference*, Seattle, WA, 6-9 Apr. 2009.

[2] N. Curtis-Brown, I. Guzman, T. Sherman, J. Tellez, E. Gomez, and S. Bhandari, "UAV Collision Detection and Avoidance using ADS-B Sensor and Custom ADS-B Like Solution," *Proceedings of Infotech@Aerospace Conference*, Grapevine, TX, 9-13 Jan. 2017.

[3] J. Redding, J. N. Amin, J. D. Boskovic, Y. Kang, K. Hedrick, J. Howlett, and S. Poll, "A Real-Time Obstacle Detection and Reactive Path Planning System for Autonomous Small-Scale Helicopters," *AIAA Guidance, Navigation, and Control Conference,* Hilton Head, SC, August, 2007.

[4] M. Whalley, G. Schulein, C. Theodore, and M. Takahashi, "Design and Flight Test Results for a Hemispherical Ladar Developed to Support Unmanned Rotorcraft Urban Operations Research," *Proceedings American Helicopter Society 64th Annual Forum,* Montreal, Canada, April 29-May 1, 2008.

[5] S. Griffiths, J. Saunders, A. Curtis, B. Barber, T. McLain, and R. Beard, "Obstacle and Terrain Avoidance for Miniature Aerial Vehicles. *Advances in Unmanned Aerial Vehicles*, pp. 213-244, Springer Netherlands, 2007.

[6] B. Richards, M. Gan, S. Bhandari, et al., "Collision Avoidance System for UAVs using Computer Vision," *Proceedings of AIAA Invotech@Aerospace*, Kissimmee, FL, 5-9 Jan. 2015.

[7] R. Schaeffer, "A Standards-Based Approach to Sense-and-Avoid Technology," *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*, Sept. 2004.

[8] A. Tan, J. Banuelos, K. Patel, S. Bhandari, et al. "Flight Test Results of the Collision Avoidance System for Unmanned Aerial Systems using Stereoscopic Vision*", Proceedings of AIAA Infotech@Aerospace*, Grapevine, TX, 9-13 Jan. 2017.

[9] T. Srinivasan, J. Gray, M. Torstenbo, S. Bhandari, et al., “Collision Avoidance System using Stereoscopic Vision for Unmanned Aerial Systems,” *Proceedings of Infotech@Aerospace,* San Diego, CA, 4-8 Jan. 2016.

[10] H. Alvarez, L. M. Paz, J. Sturm, and D. Cremers. "Collision Avoidance for Quadrotors with a Monocular Camera," *The 14th International Symposium on Experimental Robotics*, Marrakech and Essaouira, Morroco, 15-18 Jun. 2014.

[11] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, "A Fast Stereo Matching Algorithm Suitable for Embedded Real-time Systems," *Computer Vision and Image Understanding*, Vol. 114, No. 11, pp. 1180-1202, November 2010.

[12] M. Solh and G. Alregib, "Hierarchical Hole-Filling for Depth-Based View Synthesis in FTV and 3D Video." *IEEE Journal of Selected Topics in Signal Processing IEEE*, Vol. 6, No. 5, pp. 495-504, June 2012.

[13] R. J. Carnie, R. A. Walker, and P. I. Corke, “Computer Vision Based Collision Avoidance for UAVs.” *Proceedings of 11th Australian International Aerospace Congress*, Melbourne, Australia, 2005.

[14] X. Lai, H. Wang, Y. Xu, "A Real-time Range Finding System with Binocular Stereo Vision," *International Journal of Advanced Robotic System*, 2012.

[15] D. Tzovaras, N. Grammalidis, and M. G. Strintzis. "Disparity Field and Depth Map Coding for Multiview 3D Image Generation." *Signal Processing: Image Communication* 11.3 (1998): 205-30.