Autonomous Monocular Obstacle Detection for Avoidance in Quadrotor UAVs

Panos Valavanis
University of South Florida, valavanisp@gmail.com

Autonomous Monocular Obstacle Detection for Avoidance in Quadrotor UAVs

by

Panos Valavanis

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Alfredo Weitzenfeld, Ph.D.
Dmitry Goldgof, Ph.D.
Sriram Chellappan, Ph.D.

Date of Approval:
May 16, 2019

Keywords: Computer Vision, Drone, Contour Detection, Image Processing

ACKNOWLEDGMENTS

This research has been partially supported by SOFWERX, a public facing intermediary for a partnership agreement between USSOCOM and DEFENSEWERX. I would like to thank my manager, John Coglianese, who supervised the efforts, provided financial support, and provided laboratory space for the development of the software package.

I express my sincere gratitude and thanks to my major professor and advisor Dr. Alfredo Weitzenfeld for his guidance and support throughout my studies at USF and for supervising the thesis. I thank wholeheartedly the committee members, Drs. Dmitry Goldgof and Sriram Chellappan for helping me complete this research.

I acknowledge the help of Drs. M. Mahoor, M. J. Rutherford and K. P. Valavanis (all from the University of Denver) for allowing me to visit their laboratories and see how they tackle UAV navigation and control challenges. I also thank Fred Haramboure, my manager at Lockheed Martin, who supported while completing this thesis.

Last, but not least, I thank my parents Areti and Kimon, and my sister Stella for their unconditional love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Unmanned Aircraft Systems (UAS) research and development and applications have witnessed unprecedented levels of growth in the past two decades. Although military applications have dominated the market, it is anticipated and expected that civilian and public domain applications will be dominant in the future. Consequently, gradual and timely integration of unmanned aviation into the National Airspace System (NAS) is a real challenge, and roadmaps towards achieving full integration are already in place in the US, Canada, Australia, South Africa and European Union (EU). However, before complete integration of manned-unmanned aviation, additional challenging problems need to be addressed and solved, which include, among others, flight control systems, sense-detect-and-avoid and see-and-avoid systems, communication spectrum, standards, unmanned aviation safety and reliability, etc.

One, distinct, major bottleneck to achieving full integration of manned-unmanned aviation is the mid-air collision avoidance problem. Although the manned aviation collision avoidance framework is fully developed and implemented, and technology standards continue to improve, unmanned-manned and unmanned-unmanned mid-air collision avoidance systems have not reached implementation maturity, and wide or universal acceptance (by Federal authorities, industry and other players). The lack of universally acceptable standards adds to the difficulty of solving the problem, although there exist recommendations for system design and performance requirements. Regardless, solving the mid-air collision avoidance problem is a hard prerequisite to integration into the NAS.

This research tackles the specific problem of real-time obstacle detection using monocular vision, which is then used for collision avoidance. Focus is on small-scale, Class-I

unmanned aerial vehicles (UAVs) with Maximum Takeoff Weight (MTOW) of less than 25 lbs. This class of UAS constitutes the first family of unmanned systems that will be allowed to fly in (restricted) civilian airspace. Emphasis is put on multi-rotor (quadrotor) UAVs, because they have demonstrated promise and potential for a wide spectrum of civil and public domain applications due to their (mostly) cost-effectiveness, ability to takeoff and land without requiring any infrastructure (runway), and ability to hover, fly in low altitudes, in confined spaces, close to the ground, ceiling and walls, as well as because of their robust flying agility and flexibility (even when a motor fails, for example).

The focus of the research is on introducing and developing an on-board the quadrotor cost-effective monocular vision-based system and support technologies for mid-air obstacle detection that allows for: i.) robust detection of obstacles in real-time, while flying, and, ii.) ways in which this detection can be used for obstacle avoidance when moving within the environment. A possible extension of this research will allow for the ability to plan paths, trajectories, create, and store and update environment maps (with obstacles) offline and on-line as the quadrotor flies within the environment.

There is a plethora of commercially available multi-rotor/quadrotor prototypes with different functionalities and onboard (visual and nonvisual) sensors. Most of such systems are RC-operated/controlled (by a human operator), while some follow visual navigation based on an onboard camera. Regardless, their technology is rather restrictive and proprietary to the manufacturer and the application they have been designed for. In addition, there are many University lab-prototypes, the framework of which is mostly 3-D printed, with off-the-shelf-components (motors, sensors, autopilots) and other dedicated technologies and onboard sensors depending on application requirements.

As opposed to most existing prototypes, the methodology and support technologies proposed in this thesis are hardware agnostic (independent of specific hardware configuration). Although for implementation and testing purposes the ARDrone 2.0 quadrotor is used, the underlying software library may be used on any prototype quadrotor

equipped with a monocular camera. Demonstration of the proposed approach considers static and dynamic environments, single and multiple obstacles, and it is evaluated in realistic simulated world environments using ROS, Gazebo and the TUM Simulator package. Interfaces are developed to allow for portability and easy integration, so that the overall ensemble quadrotor (onboard monocular vision system interface) is modular, portable and independent of specific camera and quadrotor type.

Obtained results contribute to: i.) a thorough understanding of advantages, disadvantages and applicability limitations of using only monocular vision for obstacle detection and avoidance; ii.) real-time applicability, and, iii.) applicability restrictions due to limitations of existing support technologies, i.e., ROS, Gazebo and the TUM Simulator package. A major contribution of the developed system is that it may be used as an open-source educational tool for students, practitioners and end users who are interested in unmanned aviation.

CHAPTER 1: INTRODUCTION

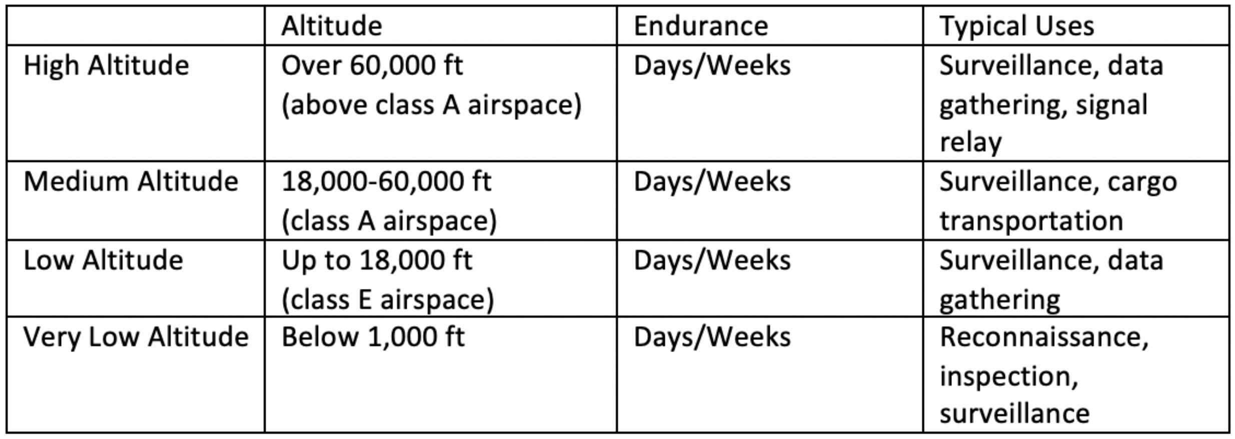
1.1 Introduction and Rationale

Unmanned Aircraft Systems (UAS) research and development have witnessed unprecedented levels of growth in the past two decades. UAS market growth projections are revised on a regular basis with the aim to 'capture' as accurately as possible the economic and societal impact of unmanned aviation. The March 2013 AUVSI report on the economic impact of UAS in the US [2] provides the first comprehensive and supported by data analysis. In addition, the Teal Group predicts that the civil UAS production, worldwide, will increase substantially [3], and that the military UAV production will reach $90 billion over the next decade [4]. To be more specific, according to the Teal Group's 2018 World Civil UAS Market Profile and Forecast, the non-military UAS production will total $88.3 billion in the next decade, up from $4.4 billion worldwide in 2018 to $13.1 billion in 2027. It is also projected that commercial use will surpass the consumer UAS market in 2024, becoming the largest segment of the civil market [3][4].

When focusing on civil and public domains, unmanned aviation applications cover a very wide spectrum, which includes: precision agriculture, forest protection and early fire detection, environment monitoring, cartography, mapping, surveillance, reconnaissance, infrastructure inspection, traffic monitoring, search and rescue, emergency response, early anomaly detection, border and coastal patrol, as well as a plethora of other applications like aerial filming and photography [5].

Consequently, gradual and timely integration of unmanned aviation into the National Airspace System (NAS) is a real challenge, and roadmaps towards achieving full integration are already in place in the US, Canada, Australia, South Africa and European Union (EU). However, integration into the airspace system depends on the classification of the UAS. Defense agencies (like the U.S. DoD) have their own preferred classification standard, while civilian agencies and civilians classify UAS by size, weight, range, endurance, flying altitude, etc. Table 1.1 below shows a broad classification of UAS along with potential uses. A rather accurate classification of UAS is also provided in [1], which is depicted in Table 1.2. Regardless, before complete integration of manned-unmanned aviation, additional challenging problems need to be addressed and solved, which include, among others, flight control systems, sense-detect-and-avoid and see-and-avoid systems, communication spectrum, standards, unmanned aviation safety and reliability, etc. [6][7].

Table 1.1: Broad UAS classification with potential uses



	Altitude	Endurance	Typical Uses
High Altitude	Over 60,000 ft (above class A airspace)	Days/Weeks	Surveillance, data gathering, signal relay
Medium Altitude	18,000-60,000 ft (class A airspace)	Days/Weeks	Surveillance, cargo transportation
Low Altitude	Up to 18,000 ft (class E airspace)	Days/Weeks	Surveillance, data gathering
Very Low Altitude	Below 1,000 ft	Days/Weeks	Reconnaissance, inspection, surveillance

Timewise, the Class-I UAS/UAVs with MTOW <= 25 lbs. will be the first to be allowed to fly in restricted civilian space. Thus, it is logical to focus on Class-I UAS and develop reliable methods, techniques, tools and technology to improve safety and reliability during flight. The mid-air collision avoidance problem is one of the most important ones that needs to be addressed and solved, and this is not possible without first developing a robust obstacle detection methodology, applicable in real-time. This challenge has provided motivation and rationale for the thesis.

Table 1.2: Accurate classification of UAVs (Taken from [1])

			Mass	Range	Flight Alt	Endurance
Tactical		Micro	<5kg	<10km	250m	1 hour
	Mini	Mini	<20/25/30/150	<10	150/250/300	<2
	CR	Close Range	25-150	10-30	3000	2-4
	SR	Short Range	50-250	30-70	3000	3-5
	MR	Medium Range	150-500	70-200	5000	6-10
	MRE	MR Endurance	500-1500	>500	8000	10-18
	LADP	Low Alt. Deep Penetration	250-2500	>250	50-9000	.5-1
	LALE	Low Alt. Long Endurance	15-25	>500	3000	>24
	MALE	Med. Alt. Long Endur.	1000-1500	>500	5/8000	24-48
Strategic	HALE	High Alt. Long Endur.	2500-5000	>2000	20000	24-48
	Strato	Stratospheric	>2500	>2000	>20000	>48
	Exo	Exo-stratospheric	TBD	TBD	>30500	TBD
Special Task	UCAV	Combat UAV	>1000	+/- 1500	12000	+/- 2
	LET	Lethal	TBD	300	4000	3-4
	DEC	Decoys	150-500	0-500	50-5000	<4

1.2 Problem Statement

The problem of mid-air collision avoidance is a challenging one. It is a prerequisite to timely and gradual integration of unmanned aviation into the NAS. The challenge becomes even greater given that no 'standard technology' or 'standards' have been developed.

Mid-air obstacle detection and mid-air collision avoidance methodologies, techniques, tools and support technologies are classified in two main, broad, categories:

1. Sense-detect-and-avoid systems, SDAA (non-visual based). Such systems are mostly based on RADAR or LiDAR technologies.

2. See-and-avoid systems, SAA (vision-based). SAA systems are derived and implemented following either monocular or stereo vision methodologies, each having advantages and disadvantages.

Federal authorities like the FAA (Federal Aviation Administration) recommend 'dual' systems onboard UAVs, that is, a combination of SDAA-SAA, which should function independent of each other and in unison. However, depending on the class of the UAV, effective payload may limit onboard sensors due to size and maximum takeoff weight (MTOW) restrictions. To be exact, Class-I UAVs with MTOW <= 25 lbs. (USA classification) limit considerably flexibility of choosing onboard sensor modalities.

Although when using UAVs in military domain applications cost may not be a factor, for civilian and public domain use of UAVs (i.e., search-and-rescue, emergency response, precision agriculture, traffic monitoring, infrastructure inspection, fire detection, etc.) cost is an additional factor (to size, range, endurance and payload limitations) when building such complete unmanned systems.

In this research, the focus is on developing, implementing and testing a low-cost monocular vision system for low-altitude multi-rotor UAVs flying above urban or rural areas, while detecting obstacles reliably, and with a high degree of accuracy. Urban/rural areas, under nominal conditions, are structured, or at worst, semi-structured environments. For example, buildings, roads, infrastructure is static, humans, cars, etc., are dynamic. Emphasis is given on how effective, fast and applicable in real-time the proposed method is, as well as, determining its applicability and limitations.

For proof-of-concept demonstration and evaluation of the proposed methodology, several case studies are conducted using the commercially available Parrot ARDrone 2.0 and ROS, Gazebo and the TUM Simulator package. The proposed monocular vision system will be considered installed onboard the quadrotor, which will be commanded to hover and maintain its position, as well as fly at a fixed altitude - the z component of the (x, y, z) Cartesian coordinates is fixed. Simulated experiments will be based on real

ARDrone, camera, ROS and Gazebo constraints. Studies will show how the quadrotor detects objects, from a single object to multiple objects, and how each object is classified within the image. The robustness of this method will directly tie in to applicability for obstacle avoidance, so the quadrotor can safely navigate within an environment.

It is expected that the end-result will be a functional, modular software-based tool (ARDrone 2.0, ROS, Gazebo, TUM Simulator) that will be expandable based on future work and used for educational purposes and for laboratory studies in computer vision, UAV collision avoidance and path planning. In addition, this research will reveal implementation limitations of existing support packages like ROS and Gazebo.

1.3 Proposed Solution

The proposed solution centers on developing and implementing a software-based modular and cost-effective algorithmic solution to the Class-I UAS obstacle detection problem. Results from obstacle detection can be used for obstacle avoidance. Combined, obstacle detection and avoidance solves the mid-air collision avoidance problem. It derives and tests a lightweight sophisticated algorithm for obstacle detection using monocular vision, which is real-time implementable. The testbed platform is the Parrot ARDrone 2.0, while the world environment (simulated world) is created using the Robot Operating System (ROS), Gazebo simulation environment, and the TUM simulator package. The algorithmic approach is provided in terms of the following steps, given the testbed dynamics model:

- Acquire frames from the monocular camera onboard the quadrotor
- Run the acquired frames through an edge detection algorithm
- Apply contour detection to the processed edge detected frame
- Filter the obstacles detected in the scene to only focus on the obstacle interesting the quadrotors path

1.4 Summary of Contributions

Thesis contributions are summarized as follows:

- Expanding and simplifying preexisting technologies used for mid-air obstacle detection, without affecting performance. This means having time efficient software which can run in real-time.
- After obstacle detection, determination of whether the detected obstacle constitutes a threat to the UAV/quadrotor, meaning, determination of whether the object is in the flight path of the UAV, or its trajectory is in a collision course with the UAV trajectory.
- Recommendation on how the aforementioned obstacle detection algorithm can be further developed into a collision avoidance algorithm.
- Delivering an open-source software tool for educational and research purposes, to be used by other groups, and as educational tool in related courses.

1.5 Thesis Outline

The thesis is organized as follows: Chapter 2 presents related work and relevant literature review, summarizing advantages and disadvantages of other approaches. Chapter 3 presents in detail the problem statement, while Chapter 4 describes the proposed methodology to address and solve the problem under consideration. Chapter 5 focuses on proof-of-concept demonstrations and includes several case studies and results. Chapter 6 offers discussion, conclusions and future work. Appendices include support material and other data as needed.

CHAPTER 2: LITERATURE REVIEW

This Chapter reviews and summarizes closely related research to the topic of the thesis. It discuses different tools, techniques, and methodologies used for obstacle detection and UAV path planning, and reports on different hardware setups that have been implemented.

Stereo vision-based and monocular-vision based approaches are presented first, followed by a summary of the most closely related simultaneous localization and mapping (SLAM) approaches. Selected non-vision-based approaches are also discussed for completion purposes. Then, higher level techniques for object detection, and some relevant path planning techniques are summarized. The Chapter concludes by summarizing different simulated environment set ups.

2.1 Vision Based

Vision-based systems and their implementation rely on stereo or monocular vision. The basis of stereo vision relates to reconstructing a 3D image using multiple image sources. These sources vary from using two cameras, also known as binocular vision, or multiple cameras. Cameras are setup to acquire the same image from different angles (vantage points) and fuse acquired images together to form a 3D and/or depth image. Images provide accurate representations of the surrounding area with detailed information, such as distance between objects, scale of objects, motion within the field of view (FOV), etc. Stereo vision systems are usually coupled with other concepts, like simultaneous localization and mapping (SLAM), and optical flow, to name the most commonly used approaches, due to robust representation of the perceived 3D environment ? details are provided in subsequent sections. The construction of a 3D scene from the camera feed is based on what the lenses see/observe in 2D. This, when appropriate allows for simplification as the need for a 3D representation of the environment is not required, for example see [8] that discusses obstacle avoidance using UAVs, where a single 2D passive vision sensor is used to generate measurements of the surrounding environment, and an Extended Kalman Filter (EKF) is implemented to estimate the relative position of detected obstacles. Avoidance actions depend on a collision cone approach that is derived and used as criteria to examine if there is an obstacle in the UAV path. A minimum-effort guidance method is implemented to track multiple targets. Images are processed in real-time, and an object is said to be within the critical section of a generated collision cone if its relative velocity vector intersects the UAVs? relative velocity vector.

Unlike stereo vision, monocular vision relies on using a single camera. For monocular vision approaches to function and provide the same or similar results as stereo vision, the monocular vision system must first acquire an image of the scene, then capture a new/second image of the scene after moving to a different calculated position. This resembles how a stereo vision system functions, thus, resulting in having the same effect as binocular vision. What makes monocular vision challenging, is its limitation to static scenes. If the scene is dynamic, the potential for errors in calculation and reconstruction of the 3D scene and depth images is high.

Reported research in [9] extends the monocular vision approach by incorporating depth estimation within the image. Depending on camera parameters (that are unique to each device), distance can be estimated between the detected obstacles in the center of the image and the sides of the image. The relationship of the camera parameters between the actual 3D space and the perceived 2D space may be used to find the converging lines of the image, which are considered to be the center of the frame. This calculated center is, then, used to estimate actual distances between objects in the frame. This technique is very effective, but it may suffer from low camera resolution or if the camera orientation changes. A similar approach is reported in [10], which estimates distances between objects and the camera by calculating the Euclidian distance between them. The algorithm works by first computing obstacle position in 3D space, and then back projecting the obstacle feature points. Research in [11] also uses distance estimation along with the monocular system, but for forward collision warning systems in cars. However, this method is highly depended on a-priori information, car width for example. If this is known, then, based on the pinhole method for object location and size estimation, the width of the detected object is inversely proportional to the width of the car.

Research in [10], and [11] uses the concept of virtual horizon estimation; however, this concept is basically the focus of [12] and it is used for obstacle detection. An algorithm is presented, which is optimized for monocular camera systems located very close to the ground. Inverse perspective mapping is used for obstacle detection that is extended to appearance-based object segmentation. Virtual horizon estimation and inverse perspective mapping are run on a virtual image, which is constructed from what is considered to be the first frame of the algorithm; results are, then, compared with the second following image. The differences between these images help calculate object position and distance estimation. The main drawback of this approach is its limitation to detect and avoid free-hanging obstacles. Another concept reported in [12] focuses on distance estimation, which is also reported in [13] and [14]. The former uses circle-based Hough Transforms to detect the size of pixels, and then compares frames (frame by frame) to find changes in the images. The latter uses line-based Hough Transforms along with the pinhole camera model to calculate distance to an obstacle. The downside of this approach is the need for a-priori information of the environment.

Reported research in [15] uses a monocular camera system, however, it does not rely on it. This technique is called Appearance Based Obstacle Detection, and it is implemented based on three key assumptions: obstacles differ in appearance from the ground, the ground is relatively flat, and there are no overhanging obstacles. Appearance Based Obstacle Detection is based on pixel classification, meaning that any pixels that are not within a specific color range are classified as objects. In this way, path planning becomes an easier task to achieve. The limitation is that if any of the assumptions are not true, the algorithm is prone to large errors.

All previous papers include robust algorithms, but most are computationally intensive and rely on external hardware for computations rather than onboard the UAV computations to achieve the final goal of obstacle detection. This limitation dictates that they are not suitable for use in this thesis as the objectives are simplification and efficiency of implementation with all computations occurring in real-time, onboard the UAV (quadrotor) to achieve obstacle detection.

2.2 Extended Vision-Based Systems

Construction of a 3D environment using computer vision techniques and approaches is a well-known and widely researched area in robotics. However, when navigating in an unknown (dynamic) environment with no pre-determined goal (i.e., end location to be reached), the 3D environment construction may be insufficient for the robot or robots to complete assigned missions. In such cases simultaneous localization and mapping (SLAM) is a must! That is, construct a 3D map of the environment as the robot navigates autonomously within the environment, and update/mark the robots' location as the map expands. The work in [16] uses SLAM to explore an unknown environment; a front facing stereo camera system that is composed of two monocular cameras acquires images that are used for localization and mapping, while a bottom facing camera is used for optical flow and pose.

3D representations of an environment may be computationally costly; in highly dynamic environments pertinent algorithms may be difficult to implement. To overcome this limitation, optical flow algorithms offer a viable alternative solution. Optical flow is the change of structured light in an image, that is, change in relative motion between the scene and the camera lens [17]. This technique has proven to be very robust, as shown in [18] where optical flow is used to measure the relative motion between the robot and a specific object. Obtained information is then used for navigation purposes as demonstrated via an example: if within the image, the left side has greater optical flow values than the right side, the robot must turn right to avoid collision with the object creating the large optical flow. A more robust approach is presented in [19] in which stereo vision and optic flow are combined in one implementation. Two fisheye lenses are used for vision, segmenting what the lenses acquire into two separate algorithms: the center of the lens is used for stereo vision to detect obstacles, and the edges of the lens are used for optic flow calculations to keep the robot/UAV centered. This allows for priority to change between optic flow and stereo vision as needed. That is, if no obstacles are detected in front of the UAV, optic flow takes over to keep the UAV centered in its path; if an obstacle is detected, stereo vision takes over to avoid the obstacle.

Like [19], work in [20] also uses optic flow for positioning and navigation at the expense of additional hardware (overhead) onboard the UAV. Fusion of ultrasonic, infrared, inertial and pressure sensors is used that allows for more accurate readings and computations. The obvious drawback of the need for additional hardware (and added cost) compared to other commercially available off-the-shelf UAVs. A second configuration that uses more hardware than typically found in commercial UAVs is the one found in [21] that includes two stereo vision systems and an inertial measurement unit (IMU). The use of two stereo vision systems allows for implementing fast vision algorithms as one set is used for object detection and the other for localization. The acquired information is used to create a 3D map of the environment, which then allows for deriving a path planning algorithm.

As optical flow algorithms may be costly performance-wise, RADAR- and LiDAR-based approaches and techniques offer viable alternative solutions. In [22] a laser range finder is used along with video and image processing techniques for navigation purposes. The algorithm assumes a static environment and uses LiDAR to create a map of the robots? surroundings. This approach allows for a more robust environment representation and for a more lightweight optical flow model to be used - optical flow is only used for position estimation. Although robust, this algorithm depends on the constraint of having a static environment; when the environment is dynamic, this approach is likely to fail. Other approaches may be found in [23], which imitates a sonar system through a monocular camera using a technique called Visual Sonar. The algorithm first detects obstacles and does partial identification of obstacles, followed by creating and updating a radial map of its environment. The robot computes the open areas it can move to while avoiding obstacles using the radial map. Obstacles are classified by their pixel values (colors), compared to the ground pixel values. This technique will not work in an environment where obstacles are very similar in color to the ground. Visual sonar is also found in [24] where it is used to store data found from image frames acquired by the camera. Data is used to segment the frame into different regions representing free space, ground, and obstacles.

Two additional techniques have incorporated GPS and motion capture systems. In [25] GPS is used to provide noisy estimates of the current state of the UAV and a shortest path between two points within a known environment. Following this, a vision system (onboard the UAV) is used to provide more accurate data for path computation; combined, this results in a collision-free course. Most systems, however, steer away from using GPS due to limitations within GPS-denied environments and because of connectivity issues. The use of motion capture systems is may also be limited in scope, depending on the specific application. In [26] a motion capture system is used, VICON, for map and motion estimation, path planning, optical flow, guidance, and control. This is achieved by placing special tracking dots on the UAV used, MAV, and then tracking these dots through the camera system. This technique is robust but suffers from limited applicability due to requirements for specialized equipment.

There also exist more sophisticated approaches to study and solve the problem of obstacle detection and avoidance. In [27] obstacle avoidance is accomplished by using a collision avoidance law based on the conventional proportional navigation law. The objective is to maintain a predefined safe distance between the aircraft and a detected obstacle. To achieve this objective, a collision avoidance vector is defined first for the aircraft. Then, a second vector is defined that registers the heading angle of the aircraft and is compared to the heading angle and heading vector of the detected obstacle. This results in a collision vector between the aircraft and the obstacle, and depending on when/how these vectors intersect, the aircraft is guided away from the obstacle. This all takes places while the aircraft is in avoidance mode, and once collision avoidance has been achieved for a given object, the UAS switches back to navigation mode as it continues navigating. The approach is mostly used in missile guidance systems, but it can be used for autonomous navigation in aircrafts as well. Another higher level approach is reported in [28], which incorporates task assignment and path planning to a set of fixed-wing aircrafts. This is achieved through the use of two algorithms and a search tree. The first algorithm is an exhaustive search algorithm that improves over time and produces a minimum cost solution, while the second algorithm is a greedy algorithm, which provides a quick solution to the problem of obstacle avoidance. The tree is a decision tree that holds the solutions of the exhaustive search algorithm. This solution addresses the difficulties encountered from a single person operating multiple UAV's and the growing need for multiple UAV's in use for different functions. The solution is split into two parts: motion planning and task assignment. For motion planning, the problem is formulated into a decision tree with having the targets position, vehicles initial configurations, and obstacles vertices be tree nodes. Each branch of the tree represents a described path, whether it is direct without any obstacles, or requires the UAV to avoid obstacles. Following this, task assignments are produced. Task assignments are represented in the tree with each node holding a "lost benefit" cost. This cost is the amount of time it takes for the vehicle to reach the desired target. The greedy algorithm presents an upper bound for the cost, eliminating any unnecessary explorations in the tree. The exhaustive search then comes in and updates this bound until an optimal solution is found.

All reviewed research offers insight on the plethora of algorithms available for obstacle detection and avoidance, extending beyond just the reconstruction of a 3D environment. The main drawback is that most techniques utilize specialized hardware configurations that are not found in commercially available off-the-shelf UAVs. However, in this thesis, information and knowledge gained by reviewing pertinent techniques will be used to derive a simple and effective robust algorithm that is easily implementable in off-the-shelf UAVs (quadrotors) with the absolute minimum of hardware overhead.

2.3 Path Planning

Autonomous navigation is a two-step problem: obstacle detection, and path planning. There exist different algorithms, approaches, and solutions to path planning. Rapidly-exploring Random Trees (RRT) are reported in [29], which extends the RRT algorithm to use a KD tree (KD trees are a data structure that are based on a nearest neighbor approach to storing spatial data). This allows for improved planning efficiency and quality of the generated paths. The idea behind this approach is that similar states will arise again, meaning, the world is changing, but not by much.

Another extension to the RRT algorithm is found in [30], which includes the A-Star algorithm for path planning to efficiently determine the cost of each randomly generated way-point. A very similar implementation can be found in [31], which uses these methods in a game design environment. A basic implementation is found in [32], which uses the RRT algorithm to compute multiple paths, and the runs Dijkstra's path planning algorithm to find the shortest path.

Lastly, [33] uses stereo vision to represent the environment in a probabilistic roadmap, and uses this in conjunction with the D* Lite Algorithm for path planning and obstacle detection. Through this method, obstacles are detected using stereo vision, which then builds a 3D occupancy map. The path planning is done using the probabilistic road maps, and D* Lite is used to search for a shortest collision free path.

2.4 Simulated Environments

Four main stages of the development cycle which are found in any development approach are design, implementation, testing, and release. The aforementioned sections fall within the design and implementation stages. The testing stage is what bridges the gap between the implementation and released product, and it can be the most time consuming and tedious stage. It is in this stage which bugs and errors will be found, and a step backwards in the development cycle must be taken to correct the implementation.

Testing of projects can be very time consuming, expensive, and in many cases, not possible. This is especially true for military systems, space systems, disaster relief solutions, etc. This is where the use of simulations comes into play; simulation environments are in abundance, many with very specific applications for each system. Gazebo, [34], is a fully featured robotics simulator which allows for fast and efficient testing of algorithms. The simulated world can be configured to match any scenario needed, represent any real world environment, and include all relevant laws of physics needed. To extend upon Gazebo's functionality, it also allows for custom plugins to be created and ported into the simulation environment. This allows for very specific test cases to be handled and controlled exactly as desired.

Unlike Gazebo, there are simulators which are built for a specific purpose. Subsim is one of these simulators [35] [36]. The main purpose of this simulator is for submarine games and warfare. It features a fully featured physics engine and configuration environment. A similar approach is taken with the X-Plane simulator, which solely focuses on aircrafts [37]. X-Plane also allows for custom aircraft creation, custom scenery creation, and custom code plugins [38]. Similar to X-Plane is Flightgear, which also allows for custom creations to be imported and used into the world [39].

Switching back to more general purpose simulators, Unity 3D is a game design software package which allows for full customization of the environment, models, physics, etc. [40]. It is mostly used for game development, however, its features allow it to be used for more than just game oriented purposes. Unity 3D supports many types of algorithms and approaches, from simple proportional-integral-derivative (PID) controllers, to machine learning. This allows for any test cases desired to be programmed and implemented within the simulator created in Unity 3D.

For the purpose of this thesis, Gazebo has been chosen due to its simplicity and its ability to be ran on the Robot Operating System (ROS).

2.5 Remarks

The presented literature review only covers a small field of the voluminous research that exists within obstacle detection and obstacle avoidance. Papers mentioned above were hand picked for this review due to how closely they trelate to the subject of the thesis. However, the provided review reveals the need to develop the proposed obstacle detection system.

CHAPTER 3: TECHNICAL PROBLEM STATEMENT

The solution to the problem of obstacle detection for mid-air collision avoidance is twofold. It requires accurate modeling and sensor-based navigation/control of the UAV (or UAVs) and, accurate modeling and representation of the world environment. When a simulated environment is considered, then, the simulated environment must reflect reality and account for real-world constraints for the results to count. In this thesis, ROS, Gazebo and the TUM Simulation library are used as tools. ROS is an industry standard in robotics and includes multiple libraries that are optimized for development. It serves as the underling development environment to implement and test the proposed solution. Gazebo is a real-world simulation environment that is interfaced (pre-installed) with ROS. Gazebo is used for visualization purposes - to 'see' how the proposed solution is implemented and tested. The ROS-Gazebo ensemble allows for an accurate controlled environment with realistic constraints in which multiple test case studies are conducted to evaluate performance of the proposed methodology. The interface between the UAV/quadrotor and the simulated environment is achieved through supplemental technologies provided by the TUM Simulation library that facilitate a simple run-execute work test bench, which can also be ported to run on a physical UAV (quadrotor).

SAA systems have limitations, some of which are not found in SDAA ones. Two key factors that differentiate the two approaches relate to hardware and processing power. Off-the-shelf monocular systems (SAA systems) are limited by processing power. This limitation dictates the need for fast, but efficient, algorithms to overcome computational burdens. The proposed solution to the problem of obstacle detection for mid-air collision avoidance addresses this problem and overcomes this limitation.

The rest of this Chapter provides an in-depth analysis of the technologies used, followed by a technical description of the problem statement and configuration to be adopted.

3.1 ROS

ROS is a framework created to ease the development of robotics software. In its core, it is a compilation of libraries and tools that promote collaborative software development [41]. For a stable instalment and execution, ROS must be installed on a Linux distribution. Based on the Linux distribution, ROS has separate distributions for system specific requirements. The distributions used for the proposed configuration are Ubuntu 14.04 and ROS Indigo.

The quadrotor of choice is the Parrot ARDrone 2.0. In order to start developing using this platform, the UAV drivers must be downloaded. A pre-existing library is the 'ardrone_autonomy' library, and it has been created specifically for this type of development within ROS. It is based on the official SDK of the ARDrone, so all components of the drone are accessible to the programmer. Once installed within the ROS ecosystem, the next library used is the OpenCV library, 'cv_bridge'. OpenCV is a collection of computer vision algorithms that have been implemented and optimized for peak performance on a system. The cv_bridge library converts ROS images, one of the pieces of data utilized, to OpenCV images. This allows for direct use of the vision library within the code without any extra steps required, meaning, no external accesses to files and drivers are required. Proper setup of these components is the main prerequisite to begin development for a solution to the problem stated above.

The capabilities ROS presents are attributed to its modularity; it is created in a way that allows for it to be distributed and modular. This allows for the user to only use desired components of ROS, instead of the full framework - one of the main advantages ROS provides. However, all this modularity comes with limitations. The core limitation and disadvantage of ROS is user friendliness [42]. It is a very complex system with multiple steps required to begin development, with insufficient documentation to support these steps and development.

3.2 Gazebo

Gazebo is an environment specifically created for robotics simulations, featuring a robust and accurate physics engine, high-quality graphics, and programmable user and graphic interfaces. Modeling of robots, environments, regression testing, AI training, and algorithm testing, are all possible within this virtual environment [34]. Gazebo is a standalone program which comes preinstalled with ROS, however, to fully utilize all of its features and achieve desired execution, it must be interfaced within the ROS framework. Once this is complete, robotics models can be programmed and simulated.

Gazebo can be a very powerful tool when setup properly, and a simple API interface means models and sensors can be easily imported into the world. This is one of the main advantages provided from this simulator. This simplicity and ease of use however, is matched by one limiting factor: heavy computation within Gazebo is very taxing on the hardware. This can be reduced on systems with special implementations to take advantage of GPU modules or development environments which use libraries to alleviate computations Gazebo has to perform, but these limitations are still to be heavily considered on everyday systems.

3.3 TUM

TUM simulator is a ROS package developed at the Technical University of Munich, which takes advantage of ROS and ardrone_autonomy libraries, allowing for a direct interface between the ARDrone platforms and Gazebo and ROS. It acts as an extension to Gazebo, allowing for direct and seamless simulation of the drone's components. Current defaults are set to the ARDrone 2.0. However, these parameters can be adjusted.

The TUM simulator library is a very powerful tool, with its main drawback being broad applicability.

The simple interface with the quadrotor makes for simple development, but extending its application is a very difficult task. The other main limitation of this library is its extension to other platforms. It is created on the ARDrone framework, so extension to other platforms would require a complete redesign of the library.

3.4 Technical Description of the Problem Statement

ROS, Gazebo and the TUM Simulator library are the support software tools that are configured and integrated in one ensemble using Ubuntu 14.04 (Linux based). The ROS framework is installed on top of Ubuntu 14.04 and it will be the base for the robotics development environment. The Gazebo simulator is then properly interfaced within the ROS framework to allow for full access to the robotics components of any models used. The ardrone_autonomy and cv_bridge libraries are installed to allow for proper installation of the TUM Simulator package. This set up with its constituent components properly installed and configured provide the set-up simulated environment. Figure 3.1 provides a visual representation of the aforementioned system and the hierarchy of the systems within each other. Following this, Figure 3.2 displays how each piece of software, library, and package relates to each other.

3.5 Remarks

This chapter presented an in depth discussion of the problem statement, its components, and the technologies used. Pros and cons to each piece of software, package, and environment were discussed, along with how they are used. The following chapter presents a detailed, step by step solution to the problem of obstacle detection for avoidance, followed by extensions into obstacle avoidance.

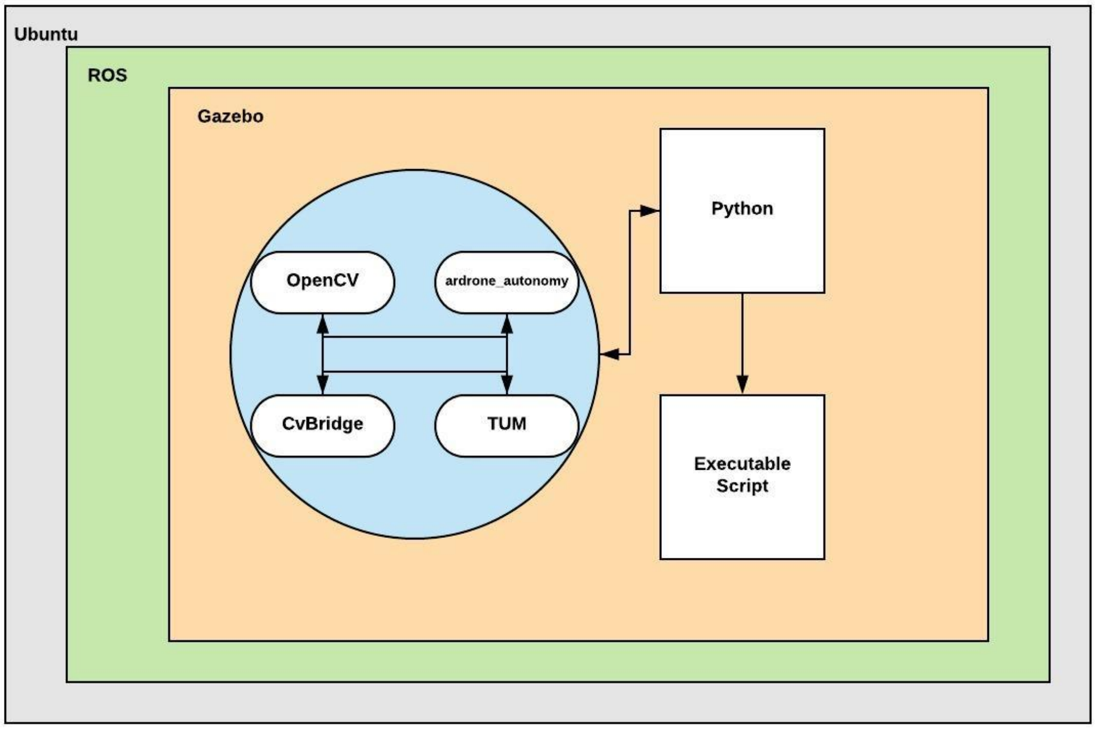


Figure 3.1: Proposed hierarchy

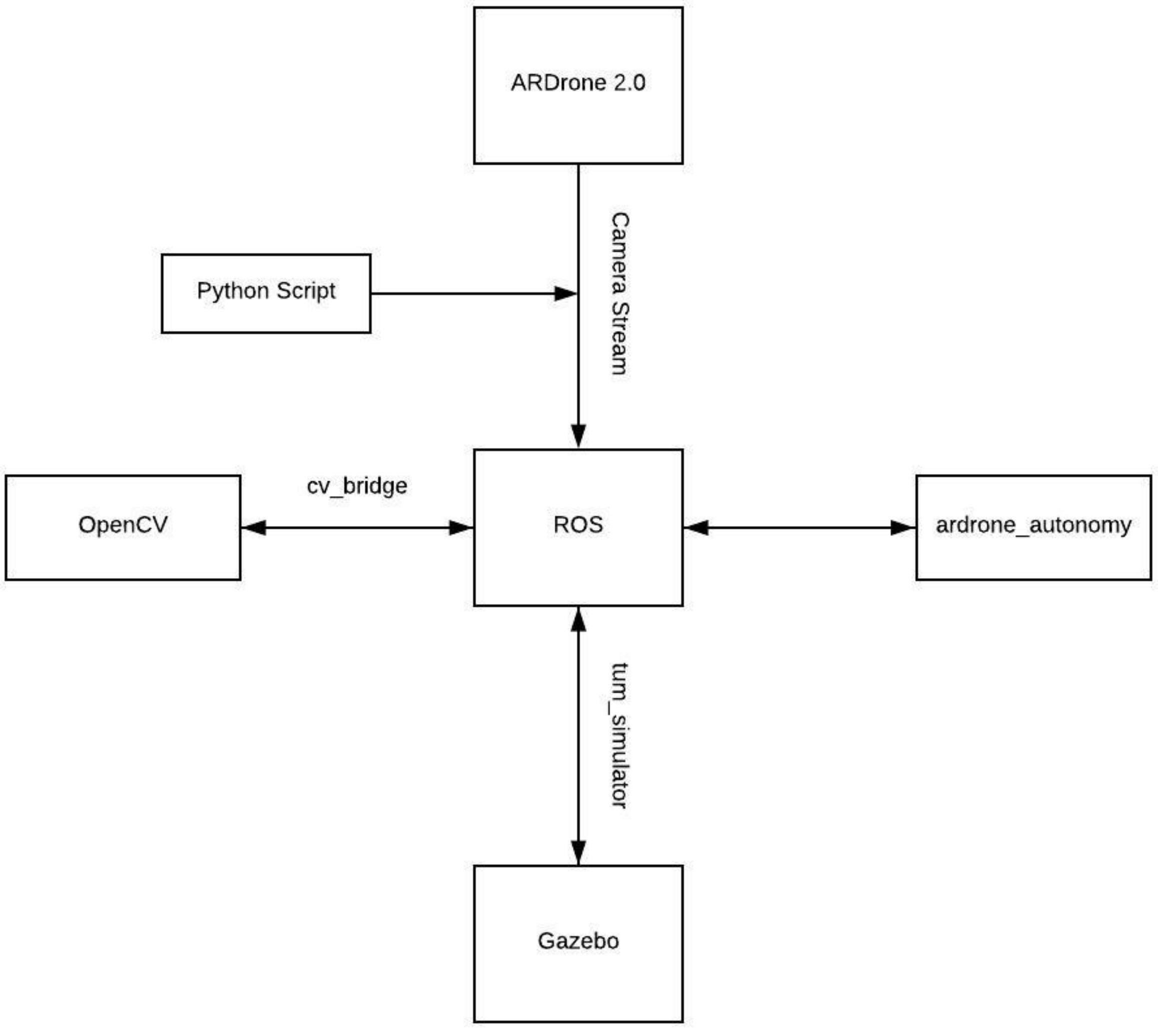


Figure 3.2: Visual representation of the interfacing libraries

CHAPTER 4: PROPOSED METHODOLOGY

This Chapter presents the in-depth description of the proposed cost-effective methodology to solve the problem of obstacle detection for avoidance using monocular vision. The simulated environment configuration (using available software libraries, packages and tools) is presented, followed by the derived algorithms and procedures to solve the problem. The proposed solution involves breaking the overall problem into subproblems: i.) environment setup, ii.) frame acquisition followed general obstacle detection, and iii.) filtering to focus on object intersecting quadrotors path. Then, integrated implementation results in the solution of obstacle detection for avoidance.

At first, the proper setup of the development environment is presented along with its corresponding capabilities and limitations. This is followed by the specifics for obstacle detection. The following section presents ways in which obstacle avoidance can be achieved based on the aforementioned obstacle detection algorithm. The last Section summarizes the functional / operational capabilities of the overall configuration.

4.1 Fundamentals of the Simulated Environment

The simulated environment is configured using the following components:

- Ubuntu 14.04: This is the main operating system (OS) that is used. Ubuntu is the OS of choice due to its high degree of configurability, which facilitates installing all constituent components to develop the simulate environment and implement the proposed solution to obstacle detection and avoidance.

- ROS Indigo: ROS, the Robot Operating System, has been specifically created for development (implementation and testing) of robotics applications. It supports a plethora of UAV pertinent libraries making it the obvious choice. Moreover, its wide applicability to diverse applications, makes it a suitable platform for comparative studies. The specific version of ROS used in this thesis, Indigo, is the stable released version for Ubuntu 14.04.

- Gazebo: Gazebo is chosen for its ease of model portability, and ease of connectivity between software modules, tools, and libraries developed within ROS and virtualization. Testing on physical hardware can be very time consuming and expensive. Virtualization is a cost-effective and efficient method of testing, usually done within a specific software (Gazebo in this case) rather than on physical hardware.

- TUM Simulator: The TUM Simulator is used because of the development libraries it provides, and because of its effective connectivity between UAV drivers and ROS.

- OpenCV: OpenCV is used for implementing image processing algorithms. It includes algorithms that are optimized for fast implementation, which makes them suitable for run-time/time-complexity applications.

- Python 2.7: Python is preferred because of its ease of use as a scripting language. To be more specific, Version 2.7 is chosen because it is supported by the aforementioned OS, software, and related libraries.

- ardrone_autonomy: ardrone_autonomy is used because it allows for direct interfaces between the user/programmer and the Parrot ARDrone?s hardware.

- cvbridge: cvbridge is a package that connects OpenCV with ROS, and allows for direct access to video streams produced by the UAV under consideration.

The commercially available (prototype) UAV that is integrated and will be functioning within the simulated environment is the Parrot ARDrone 2.0. This UAV has a video resolution of 720p (1280x720), H.264 Video Format, a frame rate of 30 frames per second, it includes a 3-axis stabilization, and wireless video transmission support. Its dimensions are 23 x 0.5 x 23 inches, and it has a weight of 4 pounds.

Next, a step-by-step guide is presented to detail how the adopted simulated environment is installed and configured. The numerical list describes the action performed, and the bulleted subsections present the command which is entered into the terminal.

1. Install a working version of Ubuntu 14.04. This is done so the development environment has a base operating system.

    - Ubuntu can be found here: https://www.ubuntu.com

2. Download and install ROS Indigo. This will act as the main development environment for the proposed solution.

    - sudo apt-get install ros-indigo-desktop-full

3. Create a ROS workspace. ROS workspaces are folders which hold all information for that specific project. Many workspaces can exist within a ROS system. The following two commands create a folder and turn it into a workspace.

    - mkdir -p /catkin_ws/src
    - cd /catkin_src
    - catkin_init_workspace

4. Build the workspace. This is equivalent to compiling a program; a workspace is built so that it be recognized by ROS as a workspace. The following commands navigate the user into the created workspace and build it.

    - cd /catkin_ws/
    - catkin_make

5. Create a package to store development files, dependencies, and libraries. A package is a folder within a ROS workspace. The following commands navigate the user to the proper folder and create the new package to be used.

   - cd /catkin_ws/src
   - catkin_create_pkg <package_name> [dependency] [depencency] . . .

6. Install ROS ARDrone drivers so that the UAV?s hardware will be directly accessible. The following commands navigate the user to the proper folder, downloads and installs the required files, and re-builds the workspace with all the new additions.

   - cd /catkin_ws/src
   - git clone https://github.com/AutonomyLab/ardrone_autonomy.git -b indigo-devel
   - cd /catkin_ws
   - rosdep install –from-paths src -i
   - catkin_make

7. Install full version of Gazebo. This done so that all features of Gazebo are available for use.

   - sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control

8. Connect Gazebo and ARDrone drivers through TUM Simulator. The following commands navigate the user to the proper file, downloads and installs all required files, and builds the workspace once more.

   - roscd
   - git clone https://github.com/tum-vision/tum_simulator.git
   - export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:'pwd'/tum_simulator

- rosmake cvg_sim_gazebo_plugins
- rosmake message_to_tf

9. Launch a test world
   - roslaunch cvg_sim_gazebo ardrone_testworld.launch
10. Update the file 'cmakelists' to include 'cvbridge' in order to use OpenCV

These commands give a basic outline of how to setup and configure the development environment. Following these commands, a workspace named tum_simulator_ws was created. Inside this workspace, a package named opencv was created inside the src folder. Within the src folder of the opencv package, the python scripts holding the algorithms for the solution are implemented. In order for the python scripts to become executable files, the following command must be ran:

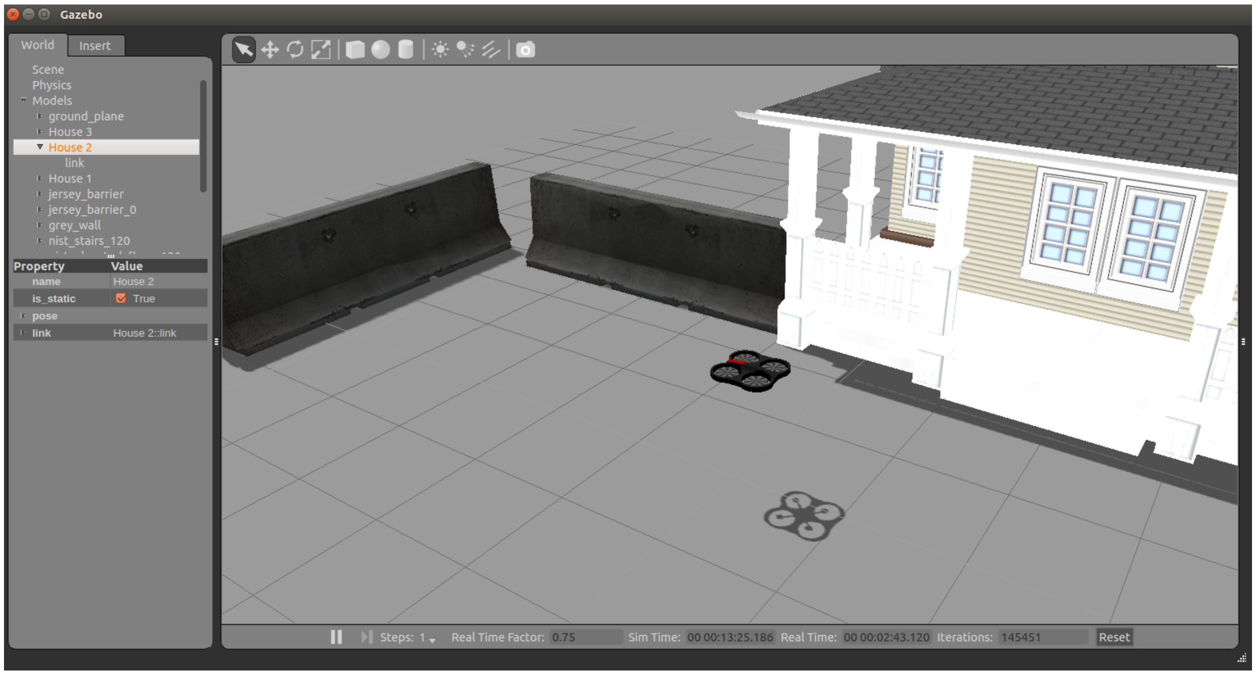
- chmod +x <filename>.py

Once this has been done, the following commands allow for development and execution of the scripts:

1. export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:'pwd'/tum_simulator
   - export package path if needed
2. catkin_make
   - build package if needed
3. . /tum_simulator_ws/devel/setup.bash
   - set bash dependency
4. roslaunch cvg_sim_gazebo ardrone_testworld.launch
   - start Gazebo with a launch file to run the scripts

5. rosrun <package_name> <file_name>.py
    - run the script on the opened gazebo world

Figure 4.1 depicts an illustration of the world being used within Gazebo.



Property	Value
name	House 2
is_static	True
pose	
link	House 2::link

Real Time Factor: 0.75 Sim Time: 00 00:13:25.186 Real Time: 00 00:02:43.120 Iterations: 145451

Figure 4.1: Gazebo world

4.2 Obstacle Detection

Challenges related to obstacle detection that must be addressed and overcome include, among others, free-floating obstacles, ground-based obstacles, nongeometrically-shaped obstacles, static and dynamic obstacles, as well as randomly moving dynamic obstacles (i.e., unpredictable trajectories). Such challenges impose pertinent constraints on algorithm derivation and affect real-time algorithm applicability, which is a function of and depends on the execution time of the derived obstacle detection algorithm(s). For example, a "slow" obstacle detection algorithm (high computational complexity) may not capture rapid changes in an unknown environment, when changes occur because of fast obstacle movements. When this is the case, onboard the UAV monocular vision algorithms may not fully process (in time) the required information / data to detect obstacles and subsequently avoid them.

There exist several approaches to solve the problem of obstacle detection, as can be observed in chapter 2; however, in this thesis, contour detection is chosen. Contour detection is based on detecting the outline of the object within the environment. It is preferred because it is based upon edge detection, which has been optimized in run time and complexity, therefore, allowing for near real-time run implementations. Further expanding on the advantages of contour detection, the edges and lines of the outlined obstacle can be used for avoidance applications, including but not limited to, optical flow, bug algorithms, and image segmentation for movement applications. More details on these applications can be found in the sections to follow.

The following algorithm explains the proposed solution to the problem of obstacle detection for avoidance. It works on the video stream produced by the UAV's camera. The video stream is composed of 30 frames per second, and each frame will undergo the processing described in the explanation of the proposed solution. Upon acquisition of the frame, the image first undergoes a greyscale operation. This is done so only one channel is needed for computations, rather than three channels which are needed to represent a color image (red, green blue).

Once the image has undergone a grayscale operation, it is then smoothed to alleviate any extraneous noise which would produce incorrect results. Following this, the image has the Canny Edge Detector kernel applied to it. The resulting image is a binarized black and white image, where all edges are white, and the rest of the image is black. This can be observed in Figure 4.2.

This allows for clear outlines of everything found within the scene. Once this step is complete, an OpenCV predefined contour detection function is applied to the image. As this function runs and analyzes the frame, it creates a tree-like structure with nodes

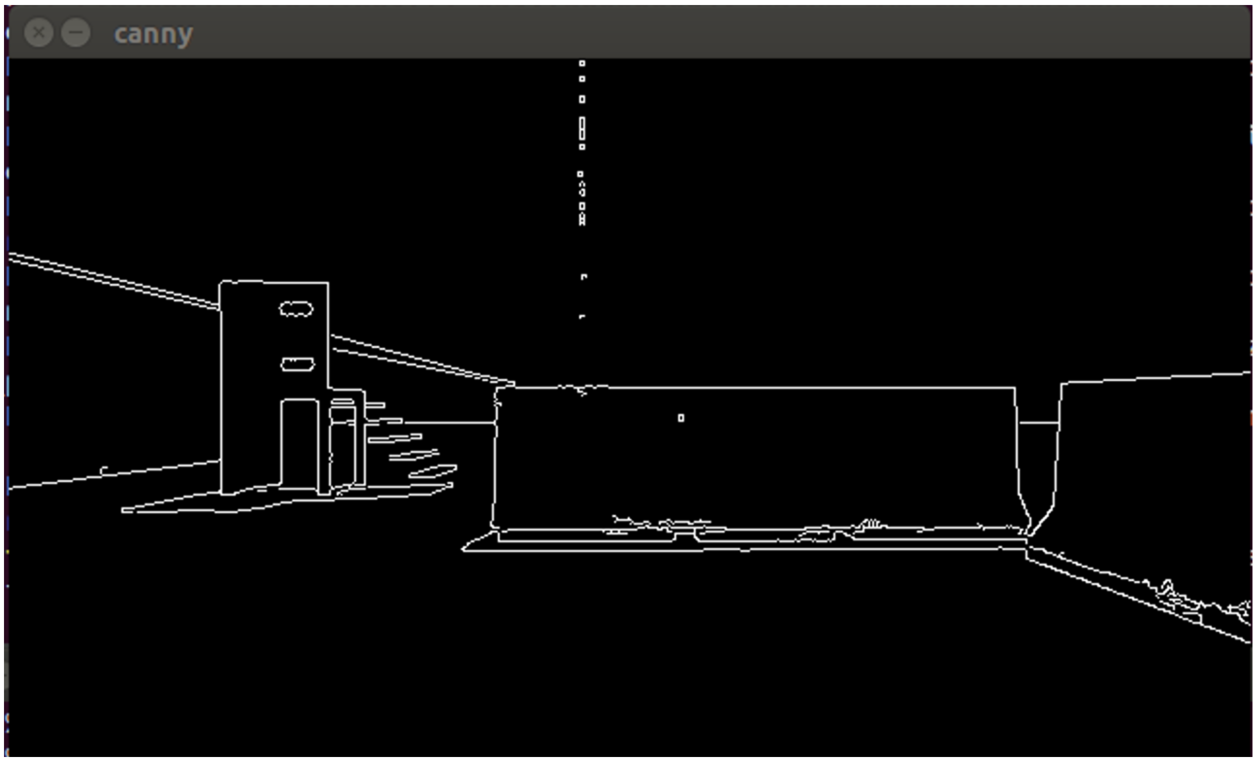


Figure 4.2: UAV point of view after edge detection

at different levels. These different levels represent the different types of obstacles in the image. For this application, 'type' refers to how the object is classified in the image. For example, the first node of the tree represents all obstacles detected in the frame as a single obstacle (the detected obstacles are grouped together). The second node represents these detected obstacles as individual standalone obstacles rather than grouping them together. The third node of the tree represent contours found within each detected obstacle. Figure 4.3 shows all nodes of the tree being displayed, meaning, all detected objects have bounding boxes showing. For the purpose of obstacle detection, only if an obstacle intersects the path of the quadrotor will it be highlighted. The proposed algorithm focuses on the second node of the tree, holding the information for the detected objects within the whole image. Once this information is acquired, a check is done to see if the detected obstacle intersects the direct straight-line path of the quadrotor. If an obstacle intersects the path, only that obstacle is highlighted. This can be observed in Figure 4.4.

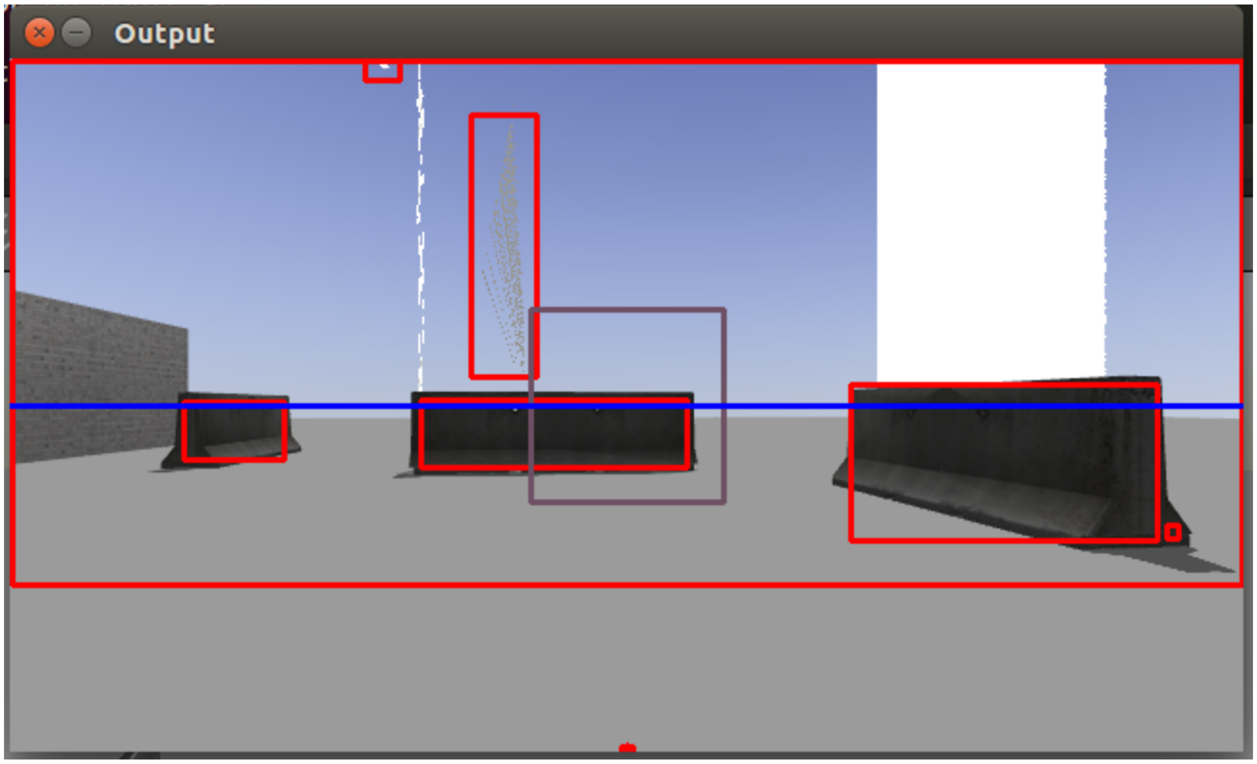


Figure 4.3: All detected objects within the image with bounding boxes visualized This means all nodes of the contour detection tree are displayed. Quadrotor position is the red dot at the bottom of the image

Once the obstacle is highlighted, a second check is done to see how close the obstacle is to the quadrotor. This is done by checking the relative location of the detected obstacle in the scene to the bottom of the quadrotor's FOV. Since we know the dimensions of the FOV, a predetermined value can be defined to act as a threshold in determining if the quadrotor should keep moving forward or hover in place. Figure 4.5 depicts this algorithm in a visual medium.

The approach mentioned above is applied to the live video stream from the UAV, allowing for real-time obstacle detection. This approach works on both static and dynamic obstacles. Static obstacles are a simple case for detection due to being stationary and easily tracked. Dynamic obstacles however can still be tracked due to the efficient nature of the algorithm, meaning, it is real-time. Similar to static obstacles, if a dynamic obstacle

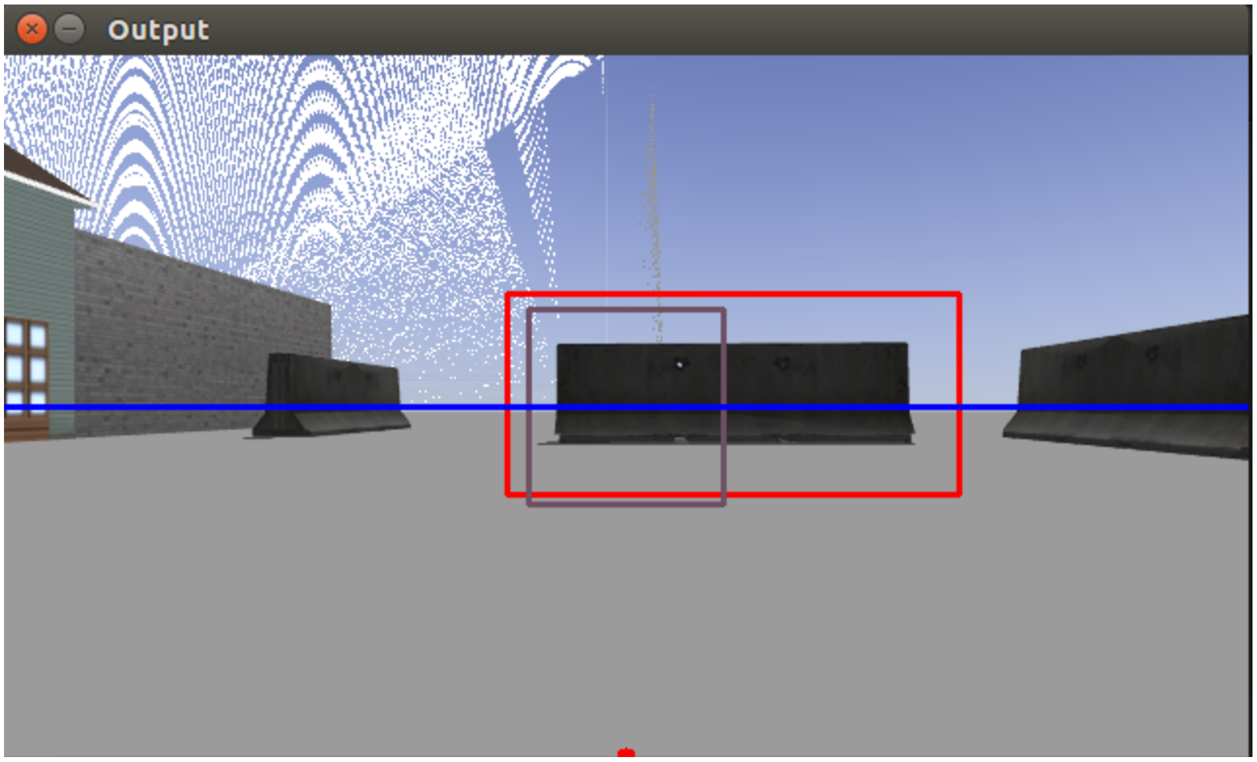
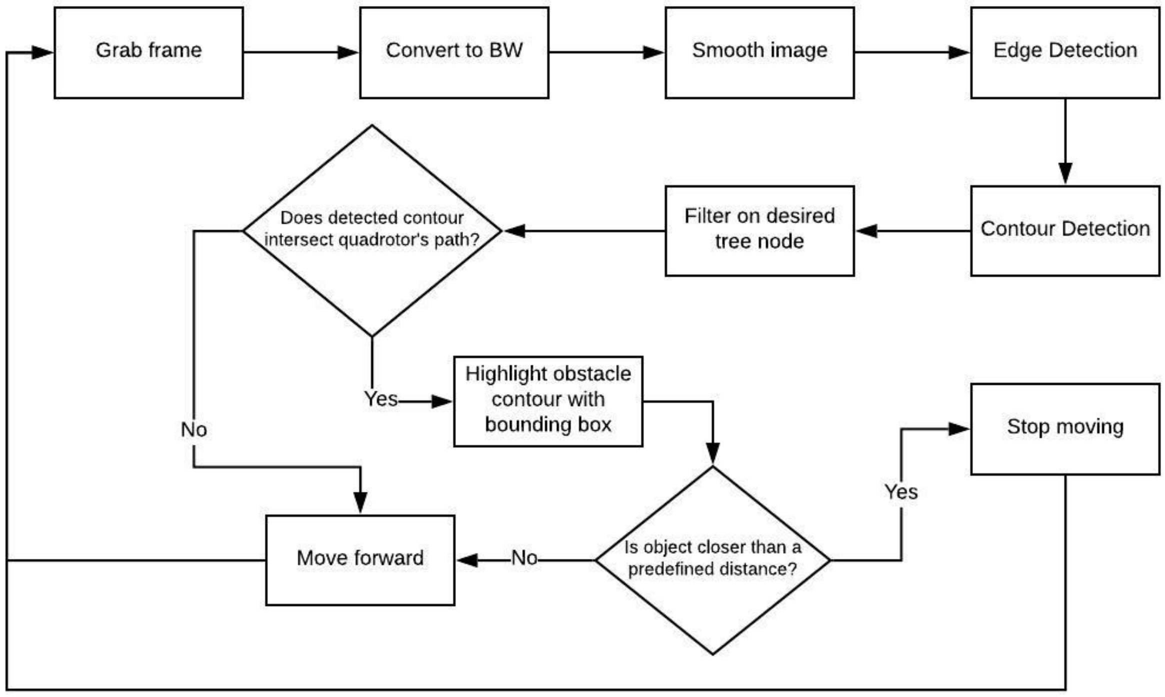


Figure 4.4: Obstacle intersecting direct path of the quadrotor
This is an image filtered to only display obstacles that intersect the quadrotors path; specifically, obstacles found within the second node of the tree. Quadrotor position is the red dot at the bottom of the image.

intersects the path of the quadrotor, the quadrotor will highlight it and react accordingly. This can be observed in Figure 4.6.

The advantage this approach has is that it is efficient, and allows for real time detection of obstacles. Efficiency is achieved through the separability of the convolution properties which all used algorithms hold. For example: instead of using one 2D filter for a computation, separability allows for two 1D filters to be used. Analyzing this complexity, if we consider a square image of N*N, the original filter would have a time complexity of O(N^2). The separable filter would have a time complexity of O($N + N$) = O($2N$), which through Bog-Oh rules simplifies to just O(N). This is an order of magnitude faster than before. This methodology is applied to the grayscale filter, smoothing filter, and Sobel Edge Detection filter.



Grab frame
Convert to BW
Smooth image
Edge Detection
Contour Detection
Filter on desired tree node
Does detected contour intersect quadrotor's path?
Yes
No
Highlight obstacle contour with bounding box
Stop moving
Yes
Is object closer than a predefined distance?
No
Move forward

Figure 4.5: Obstacle detection algorithm

4.3 Obstacle Avoidance Application

Obstacle avoidance is coupled with obstacle detection. Regardless of the specific method followed and/or implemented, detection and tracking, optical flow, SLAM, object classification, etc., prerequisite to obstacle avoidance is knowledge of the surrounding environment acquired either through detection or received a-priori (mostly when the surrounding environment is static). This knowledge, data or information, in general, serves as the input to the obstacle avoidance algorithm.

The obstacle detection approach is the cornerstone for avoidance algorithms because of its simple and robust nature. Contour detection builds on top of edge detection so that all contours falling within the same category (i.e., within the same node in the tree), represent the objects in the scene that need to be avoided. The edges of these obstacles can be tracked to implement an optical flow algorithm for avoidance.

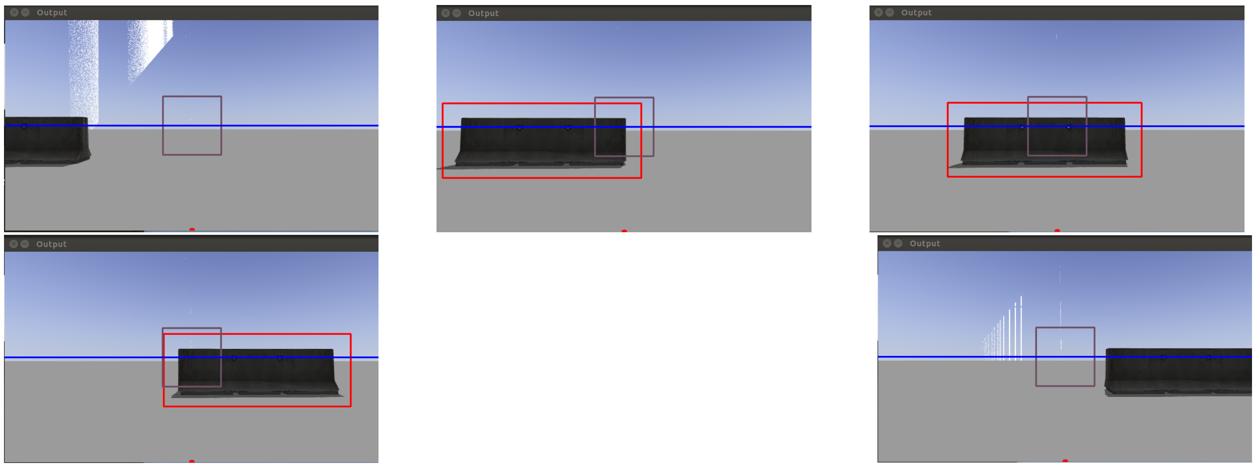


Figure 4.6: Dynamic obstacle in motion

Another avoidance method that may be used is expanding the parameter space of the obstacle, decreasing the parameter space of the UAV, and moving towards the edge of the obstacle falling nearest to the goal. What this means is the following: if a bounding rectangle for a detected object falls directly on the obstacle contours, the expanded parameter space of the obstacle will have the bounding box increase evenly from all sides, occupying more space in the frame. This difference can be observed in Figures 4.7 and 4.8:

By expanding the parameter space of the obstacles and decreasing the parameter space of the quadrotor, the quadrotor can act as a simple point in space.

One more method for avoidance which can be used based on the aforementioned detection algorithm is a quad-tree based, segmentation approach. More on quad trees can be found in [43] and [44]. This approach has the following characteristics:

- Quadrotor location is the root of the quad tree.
- Segment the frame into 4 quadrants.
- Each quadrant is a new node in the tree.
- If there is a quadrant that has no obstacles, move towards this direction.
- If all quadrants have an obstacle, further segment each quadrant into four more quadrants.

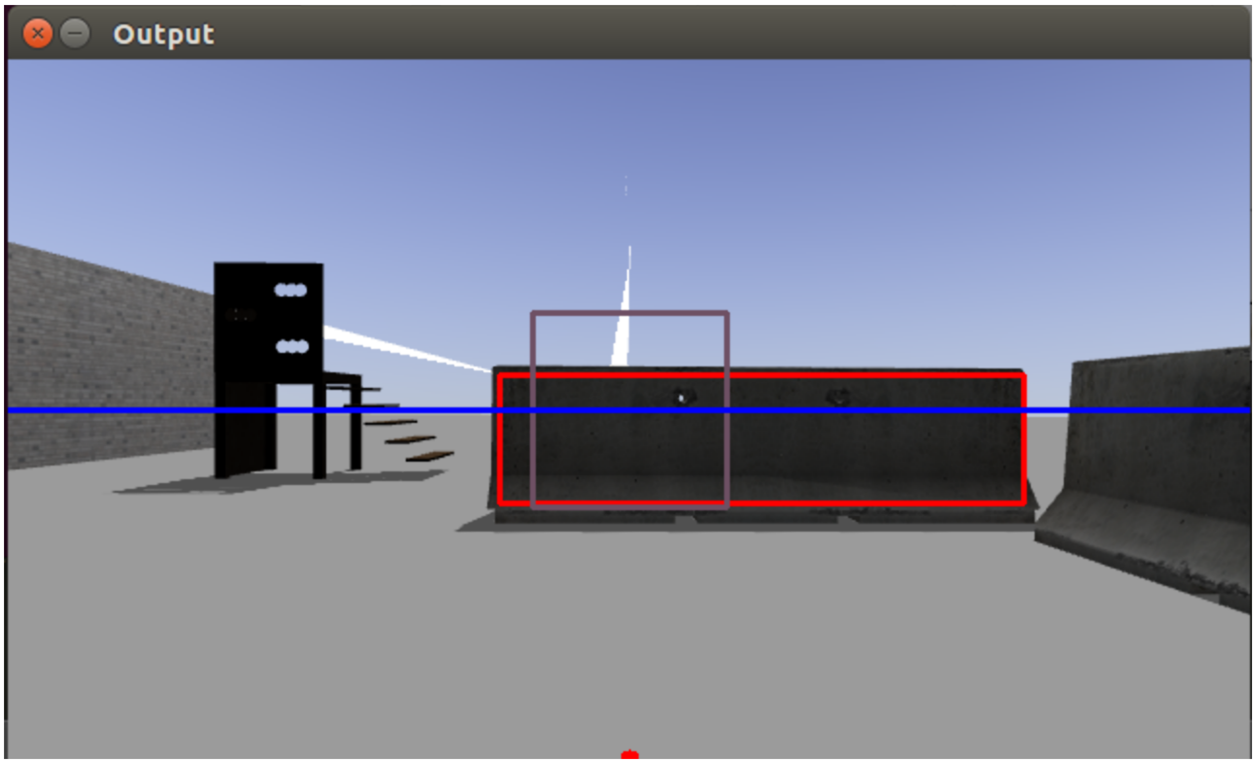


Figure 4.7: Object with its normal parameter space
This detected obstacle is a result of the algorithm filtering along the second node of the contour detection tree and due to its intersection with the direct path of the quadrotor. Quadrotor position is the red dot at the bottom of the image.

- Repeat this process until a reachable point is found, where the quadrotor can safely navigate towards.

This can be further optimized by only segmenting quadrants which are ’least dense’ with obstacles, meaning, quadrants which have more available free space for the drone to navigate to. One more optimization to add to the latter description can be to add a heuristic value to segment along the less dense quadrants and the furthest reachable point, meaning, furthest possible achievable location for the quadrotor.

Any of the avoidance methods mentioned above can be used to solve the problem of autonomous navigation; however, selection of the approach will be based on hardware, desired motion model, static and/or dynamic obstacles, indoor/outdoor application, etc. For example, the second method mentioned (expanding parameter space of obstacles) can be a very robust avoidance method for environments with well defined obstacles, but

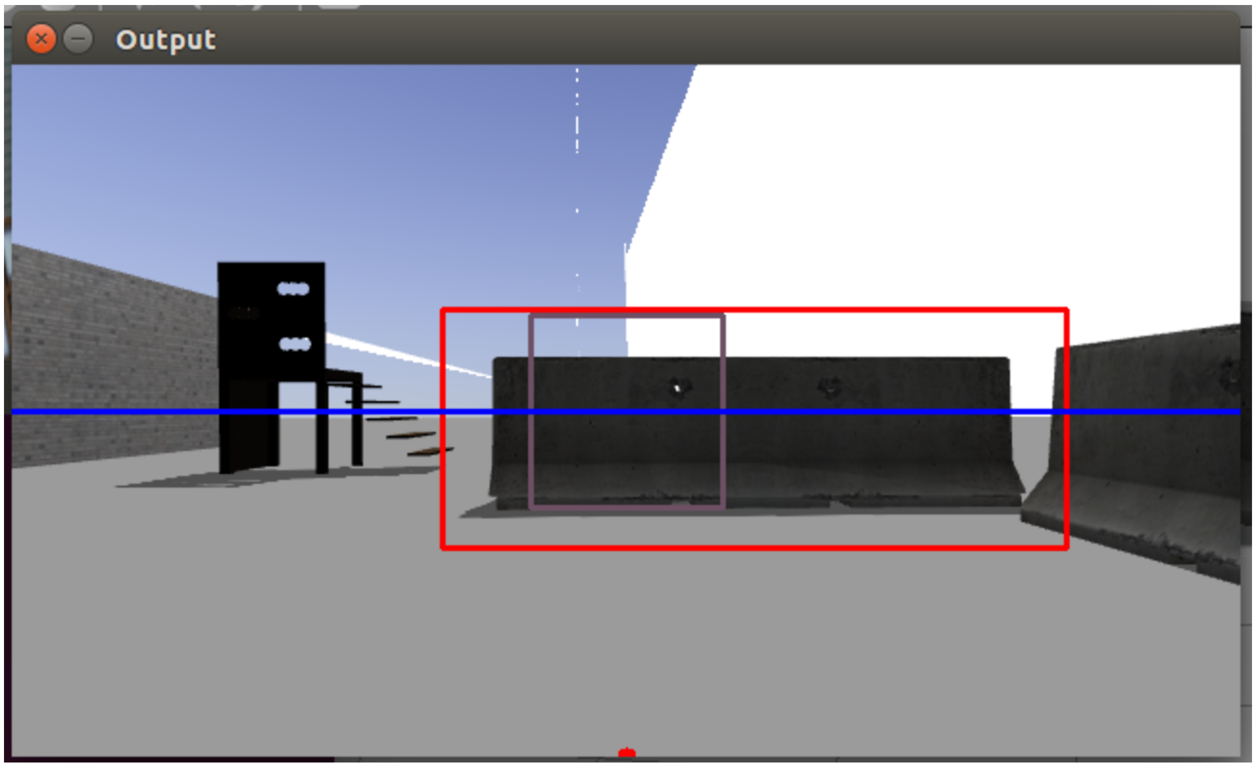


Figure 4.8: Object with its extended parameter space
This detected obstacle is a result of the algorithm filtering along the second node of the contour detection tree and due to its intersection with the direct path of the quadrotor. Quadrotor position is the red dot at the bottom of the image.

it might not have such favorable results in environments where the obstacles lack great definition. Optical flow can perform better under these conditions because it tracks relative motion of detected points. The quad tree method is a little more hardware intensive, therefore it might not act as a real-time algorithm for navigation, or it cannot be used with fast moving dynamic obstacles; fast moving in this case refers to objects which move at a faster rate than the comparable refresh rate of the quadrotor's camera.

4.4 Remarks

Through testing and modifications, the developed algorithm proved to be a robust implementation for obstacle detection. The flexibility of being able to be used on both static and dynamic obstacles will be very useful if the detection algorithm is used for an avoidance algorithm. Simplicity in the implementation files allows the user to easily change or modify the algorithm to better suit their needs.

CHAPTER 5: RESULTS AND DISCUSSIONS

5.1 Results

Initial test cases revealed limitations of the detection algorithm, which led to re-design. Limitations were related to cases of not detecting obstacles that fell half-inside and half-outside of the FOV of the quadrotor. This occurred because such obstacles were classified into the first node of the tree structure from the contour detection, rather than the second node, because the whole/full obstacle was not observed; this resulted in considering the visible part of the obstacle within the quadrotor's field of view a part of the general background of the image. Once this issue was corrected, proper detection of obstacles was achieved.

An important aspect of the algorithm that was tested repeatedly is which specific obstacle the UAV must focus on and avoid. This is an important aspect of the algorithm because it eliminates the need for extra processing, slowing down the overall system. Through only focusing on an obstacle/obstacles which are in the direct path of the quadrotor, less processing power is required, thus making the overall system more lightweight and with faster execution times. This is achieved through a simple intersection calculation between a predefined "danger" area close to the UAV, meaning that if the obstacle is within this area, then, it is marked/outlined in a red bounding box. Figure 5.1 shows an example of this concept, while Figure 5.2 shows all detected objects displayed. This is deemed essential, mostly for the user to be able to understand and follow the UAV decision making process. The UAV continues moving towards the obstacle (forward) until it reaches a predefined distance away from it, at which time the UAV stops moving and hovers in place. It is at this point where the avoidance algorithm takes control to navigate the UAV around the obstacle.

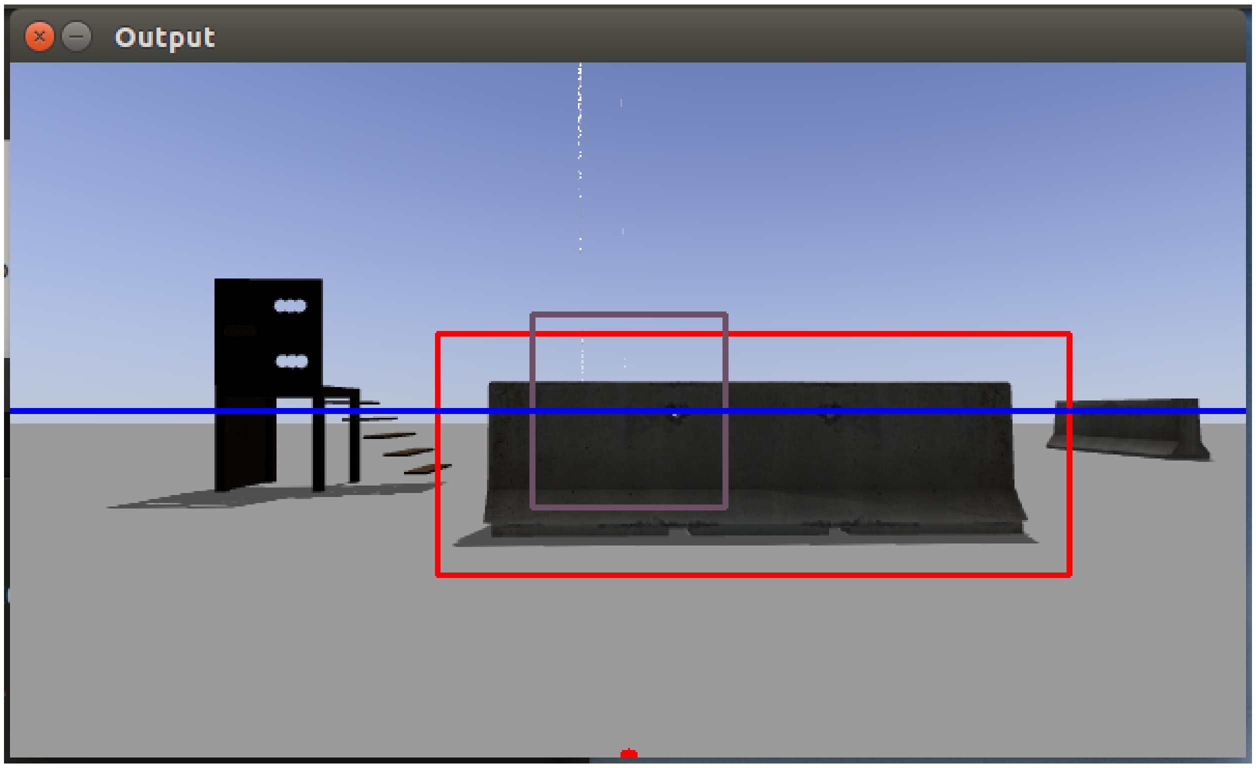


Figure 5.1: Single object in focus for UAV
This is the result of the algorithm filtering the scene to only focus on obstacles that intersect its direct path. Quadrotor position is the red dot on the bottom of the image.

Several tests are conducted with both dynamic and static obstacles. The obstacle detection algorithm demonstrates reliability and accuracy in detecting and highlighting the obstacles and in commanding the UAV to stop from moving further (after algorithm criteria are met). This reliability is a direct effect of having no difference in detecting static or dynamic obstacles; the same process is in effect, and a bounding box is displayed around the dynamic obstacle once it intersects the quadrotor's path. This bounding box will remain around the dynamic obstacle as it moves until the obstacle is no longer in the direct path of the quadrotor. To assist the user with what state the drone is in, print statements are displayed for observation. These print statements state whether the drone is mov-

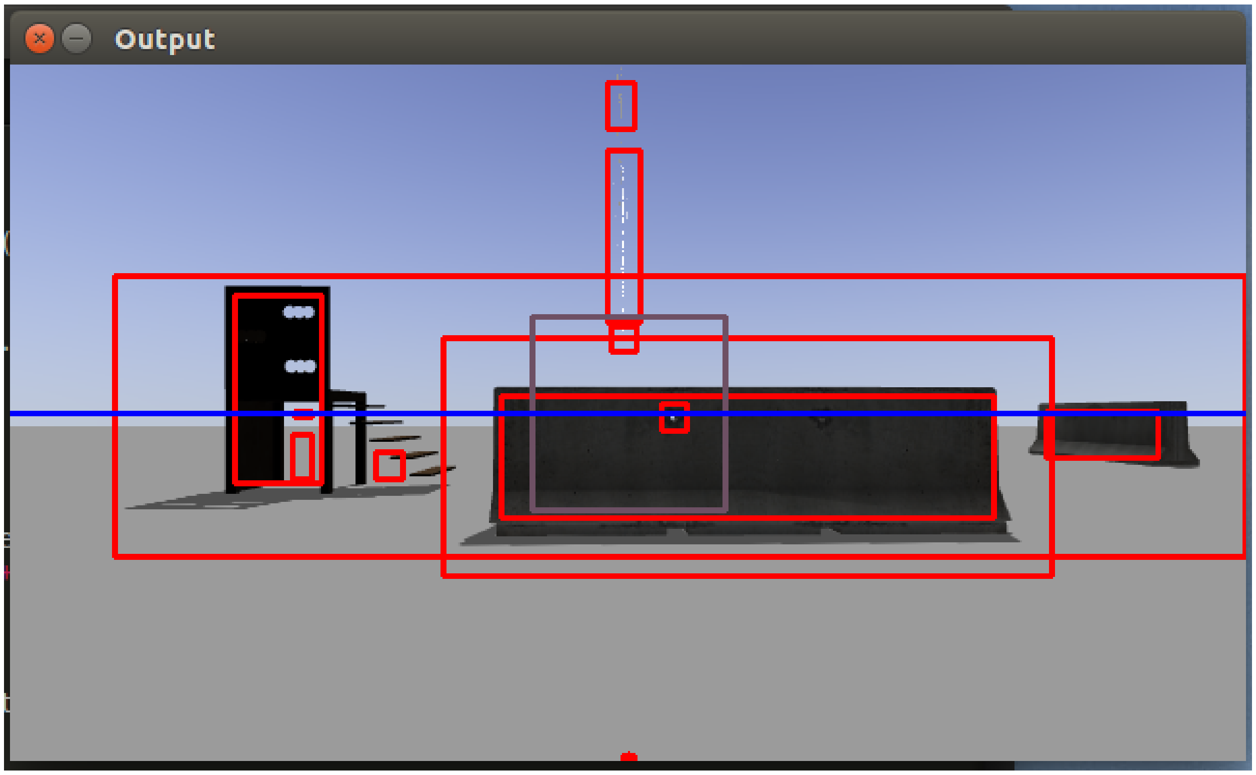
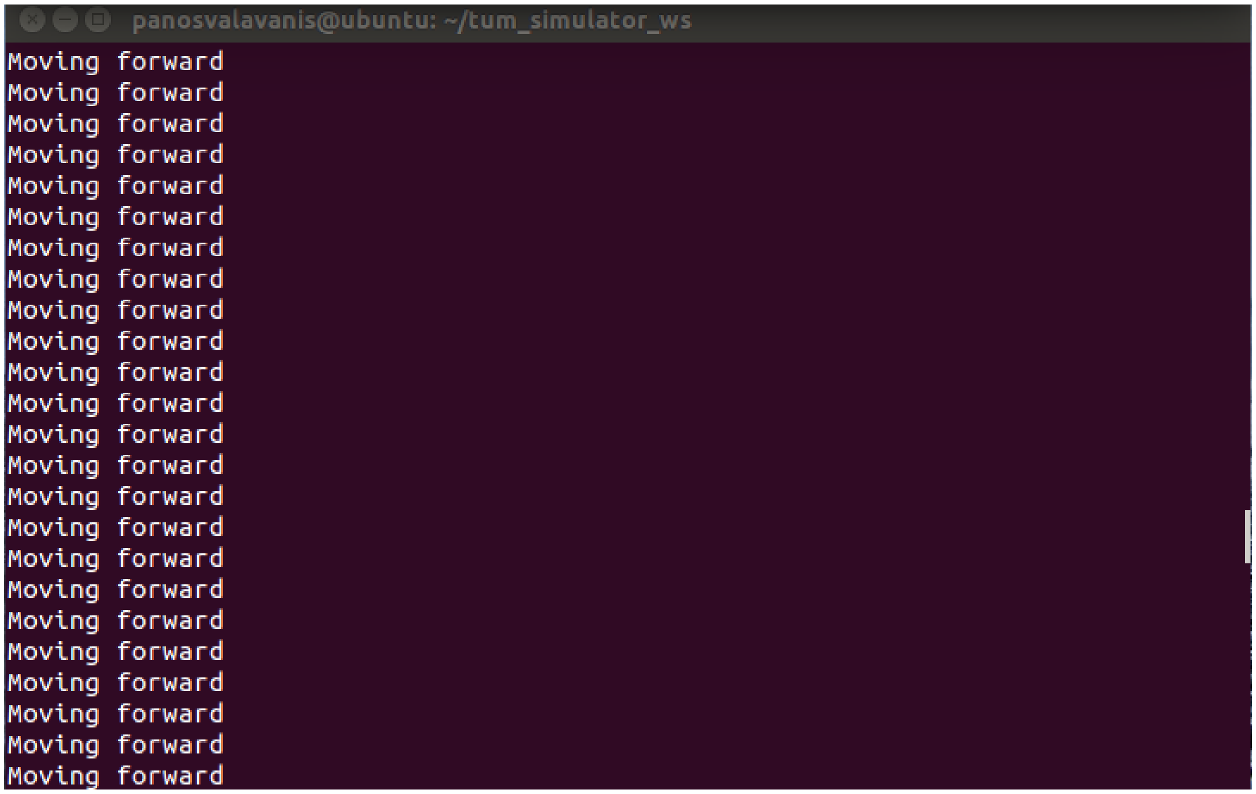


Figure 5.2: All detected objects and contours displayed
This is the algorithm displaying all information/obstacles it has detected in the scene.
Quadrotor position is the red dot on the bottom of the image.

ing, stopping, has detected an obstacle in its direct path, or has no obstacles in front. Figures 5.3, 5.5 and **??** show examples of what the statements display.

5.2 Discussions

The development environment (ROS, Gazebo, Ubuntu) played a major role in the flow and ease of development. Upon normal circumstances working from a Linux workstation, meaning, a dedicated desktop computer, the development limitations would have far fewer. For this thesis, development took place on a MacBook Pro, within a virtual machine having Ubuntu installed. This presented the following issues: slowdown of processes within the virtual machine, slower algorithm execution, "buggier" execution runs, meaning, programs would crash unexpectedly. Learning to cope this these issues, development and completion of the thesis was achieved.



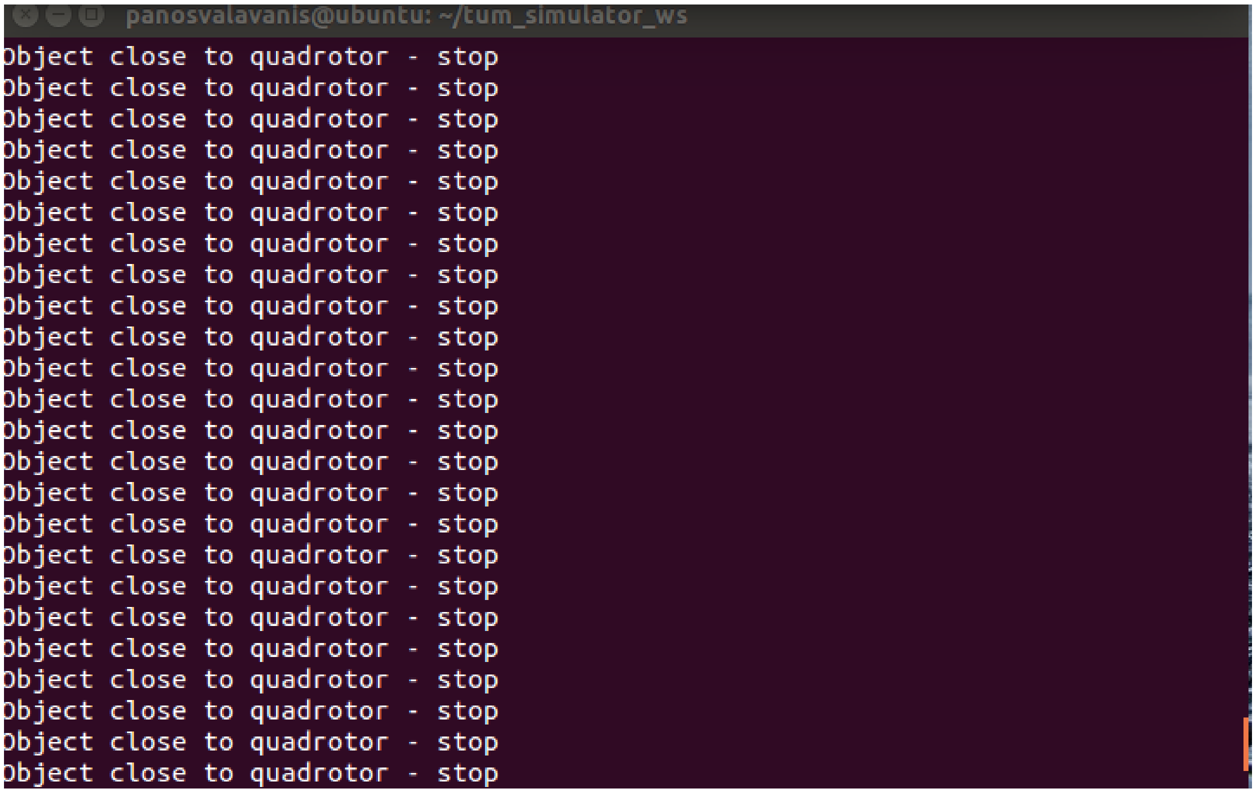
```
panosvalavanis@ubuntu: ~/tum_simulator_ws
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
Moving forward
```

Figure 5.3: No obstacle in the distance
This is an example of a print statement for when there is no obstacle intersecting the quadrotors path.

Researching the subject of autonomous obstacle detection and avoidance brought many ideas to light, however, it also showcased how much more work there is to do for a full integration of UAV's into the NAS. Without robust, proper obstacle detection in all environments, weather conditions, various altitudes, etc., proper avoidance cannot be achieved.

5.3 Utilization as an Educational Tool

This software package, along with each associated support library (OpenCV, TUM, etc.), may be used as an educational tool to complement teaching lectures in courses related to robotics, UAVs, image processing, computer vision, and any other field which could deem it beneficial. There are no minimum system requirements for the software package, however, Core i5 processor with a 256GB hard drive (preferably solid state) and

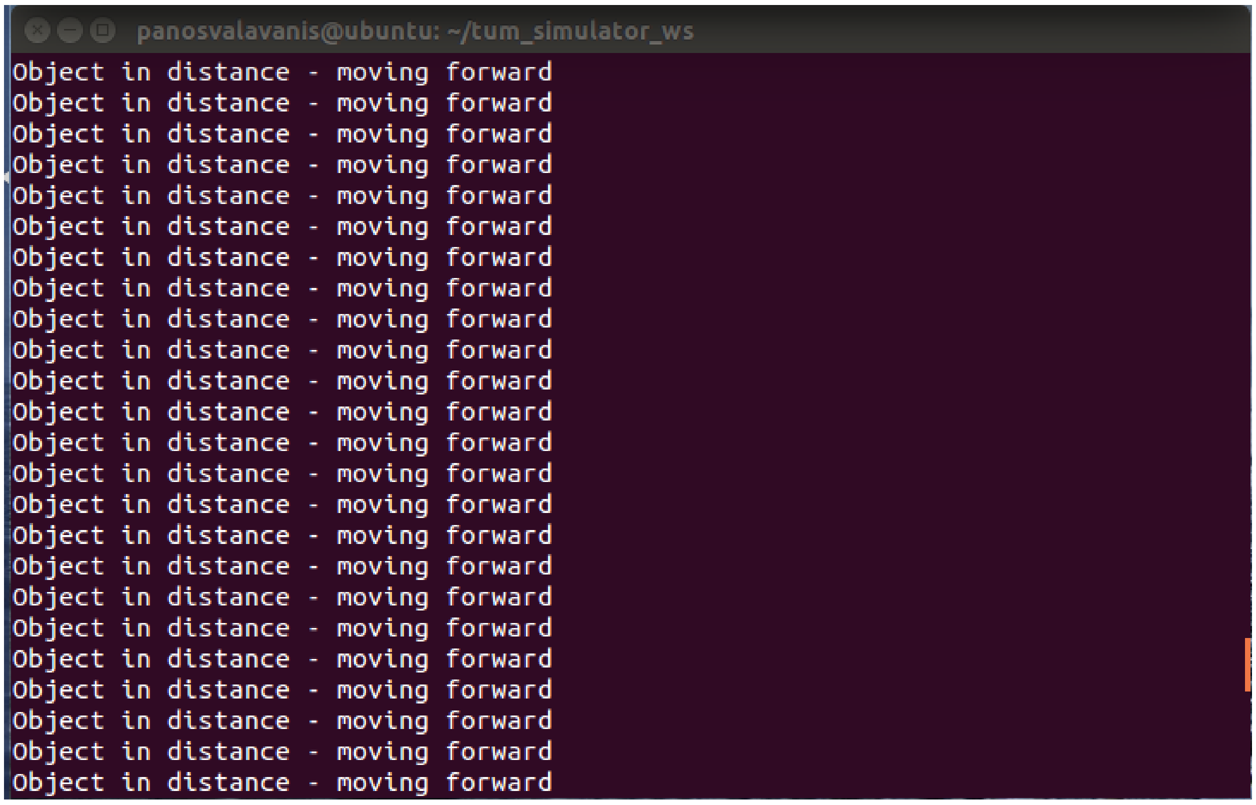


```
panosvalavanis@ubuntu: ~/tum_simulator_ws
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
Object close to quadrotor - stop
```

Figure 5.4: Print statement for object in the near distance
This is an example of a print statement for when the quadrotor is in motion and has detected an obstacle closer than the predefined distance away.

8 GB of RAM are recommended as a minimum configuration. Installation is covered in detail in section 4.1.

Once the system is installed and configured, scripts to control the quadrotor can be added, modified, executed, etc. Models can also be added to Gazebo, different simulated worlds can be created to match specific use cases; the whole software package becomes a ’playground’ for development.



```
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
Object in distance - moving forward
```

Figure 5.5: Print statement for object in the far distance
This is an example of a print statement for when there is an object in the quadrotor's path, but not close enough to stop yet.

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

6.1 Summary of Results

Chapter 5 explained the issues encountered and how they were overcome to produce a successful algorithm in obstacle detection for avoidance. This method proved to be a robust approach to this issue, yielding successful results in the runs tested. Success was measured on how accurately the quad rotor detected obstacles, both static and dynamic, and how it was able to maintain focus on these obstacles as it was moving, or as the obstacles were moving. Changing the parameters in the code for which object to focus on or to display all objects found in the scene proved how robust contour detection can be for obstacle detection. The flexibility, along with the user friendliness of the code, are both valuable assets to this thesis. Going through the script files, the implemented algorithm can be understood upon the initial read.

Furthermore, OpenCV's optimization of vision based algorithms allow for the development and implementation of efficient algorithms which can execute obstacle detection in real-time. This, along with only processing required information for detection, allowed for an algorithm fast enough for the UAV to encounter dynamic obstacles which intersect its movement path and react accordingly.

The results obtained from the detection algorithm present flexibility in how the algorithm can be extended to implement avoidance. As mentioned in previous sections, there are many different approaches which can be taken for an obstacle avoidance algorithm. The avoidance algorithm to be used is solely dependent upon the type of desired results, the environment in which the UAV will be in, and the type of data wanted for navigation. An efficient, all around recommended approach for avoidance, is optical flow. This is due to the detected contours of the objects from the detection algorithm, which can be superimposed with points to track relative motion to the quadrotor.

6.2 Contributions

Upon completion of this thesis, the main contribution at hand is the delivery of a modular, open source software tool which can be used for educational and research purposes. Having an efficient algorithm for obstacle detection allows for the expansion of this work to implement an efficient algorithm for obstacle avoidance. Due to the simplicity of the approach and format of the data of the detected obstacles, many different approaches can be taken for obstacle avoidance, as described in sections 4.3 and 6.4. This, due to its flexibility, promotes the modularity of the developed software tool. Lastly, a high end workstation is not needed to run this software tool (although desired). The software tool can run on workstations with lower-end hardware; however, the tradeoff will be on accuracy of the visual representation of what the tool is doing, meaning, instead of the video stream being real time, it will seem 'choppy' and drop some frames; for example, it will display 20 frames per second rather than 24 (which is considered real time). One way to help a lower end system run close to real time is to eliminate display of the print statements, and eliminate the window which displays the quadrotor's FOV. Movement of the quadrotor will still be seen within the Gazebo simulated world.

6.3 Discussions

Upon completion of this thesis, there are some key takeaways:

- Use the latest version of Ubuntu: This thesis uses Ubuntu 14.04 due to compatibility with the TUM package and ROS Indigo. Using Ubuntu 16.04 would better interface with the latest version of the virtual machine used for development, yielding a more streamlined development experience.

- Use the latest version of ROS: This thesis uses ROS Indigo due to its compatibility with the TUM package. ROS Kinetic, which is the following version, would yield a better development environment due to improvements and bug fixes presented in ROS Indigo.

- Don't emphasize such reliability on the TUM package: It is because of the TUM package why the specific versions of Ubuntu and ROS were chosen. Not relying on this package for development would have yielded a more streamlined development environment using the latest software.

6.4 Future Research

Expanding upon this thesis, the first step would be to implement obstacle avoidance based on the implemented obstacle detection algorithm. This would bring this research one step closer in assisting with a solution to the problem of autonomous UAV's being integrated in the NAS. Without this, proper integration cannot be achieved.

There are many ways in which the developed obstacle detection algorithm can be extended to implement obstacle avoidance, some of which are described in section 4.3. Through implementation of any of these methods, a first step to a fully autonomous system will be achieved. Implementing an algorithm which uses all described avoidance implementations can prove to be a very robust approach for accurate obstacle avoidance. As a first step, however, optical flow is recommended for avoidance (as discussed in the previous section). The detected contours of the objects present an easy interface for tracking points to be superimposed on the obstacle and tracked. These points would show the relative motion of the detected obstacle in relation to the quadrotor, allowing the quadrotor to react accordingly to its surroundings. From here, through meticulous case studies and enhancements, robust obstacle detection and avoidance can be achieved.

Further expanding on avoidance, SLAM can be implemented so the quadrotor can create a map of the environment it is navigating. This brings forth the issues of computation power and memory limits; however, extra hardware can be used/added to achieve this goal. This means that the quadrotor system is no longer an ’off the shelf’ system. If this path is to be chosen, a stereo camera can be mounted to the quadrotor for distance estimation. Supporting this, LiDAR can also be used, and the data acquired by the system can be processed and fuzed for a detection and avoidance algorithm.

REFERENCES

[1] P. van Blyenburgh, March "Avionics‘06 Conference Amsterdam, The Netherlands March 9, 2006." [Online]. Available: http://www.dror-aero.com/link/uav/uav-sys-global-review.pdf. [Accessed: 120-Dec-2019].

[2] AUVSI, “The economic impact of unamnned aircraft systems integration in the united states,” tech. rep., Association for unmanned vehicle systems international, 2013.

[3] "Teal Group Predicts Worldwide Civil Drone Production Will Soar Over the Next Decade", Home - Teal Group, 16-Jul-2018. [Online]. Available: http://www.tealgroup.com/index.php/pages/press-releases/54-teal-group-predicts-worldwide-civil-drone-production-will-soar-over-the-next-decade. [Accessed: 05-May-2019].

[4] *Drone Military Production Increase*, November 2018.

[5] "Best Multi Rotor UAV Drones for Aerial Photography and Filming, EPFILMS Best Pro Video Cameras Latest 4K Camcorders and 8K, 13-Jan-2019." [Online]. Available: http://epfilms.tv/best-multirotor-drones-top-quadcopters-aerial-filming. [Accessed: 31-May-2019].

[6] G. J. V. K. P. Valavanis, *Handbook of Unmanned Aerial Vehicles (UAVs).* Springer, 2016.

[7] L. A. P. K. Dalamagkidis, K. P. Valavanis, “On integrating unmanned aircraft systems in to the national airspace system: Issues, challenges, operational restrictions, certification, and recommendations,” *International Series on Intelligent Systems, Control and Automation: Science and Engineering*, vol. 54, 2012.

[8] E. N. J. Yoko Watanabe, Anthony J. Calise, “Vision-based obstacle avoidance for uavs,” *American Institute of Aeronautics and Astronautics*, August 2007.

[9] A. Joglekar, D. Joshi, R. Khemani, S. Nair, and S. Sahare, “Depth estimation using monocular camera,” *International Journal of Computer Science and Information Technologies*, vol. 2, no. 4, pp. 1758–1763, 2011.

[10] S. W. Suman Saha, Ashutosh Natraj, “A real-time monocular vision-based frontal obstacle detection and avoidance for low cost uavs in gps denied environment,” *IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, pp. 189–195, 2014.

[11] S.-Y. H. Ki-Yeong Park, “Robust range estimation with a monocular camera for vision-based forward collision warning system,” *The Scientific World Journal*, p. 9, January 2014.

[12] D.-I. D. C. Tae-Jae Lee, Dong-Hoon Yi, “A monocular vision sensor-based obstacle detection algorithm for autonomous robots,” *MDPI Sensors*, p. 19, March 2016.

[13] K. S. M. N. A. Wahab, N. Sivadev, “Target distance estimation using monocular vision system for mobile robot,” *IEEE Conference on Open Systems*, pp. 11–15, September 2011.

[14] Z. Zhou, T. Chen, D. Wu, and C. Yu, “Corridor navigation and obstacle distance estimation for monocular vision mobile robots,” *International Journal of Digital Content Technology and its Applications*, vol. 5, pp. 192–202, March 2011.

[15] I. N. Iwan Ulrich, "Appearance-based obstacle detection with monocular color vision," *American Association for Artificial Intelligence*, 2000.

[16] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor mav," *EEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4557–4564, 2012.

[17] F. Raudies, "Optic flow," Scholarpedia. [Online]. Available: http://www.scholarpedia.org/article/Opticflow. [Accessed: 31-May-2019].

[18] A. K. Kahlouche Souhila, "Optical flow based robot obstacle avoidance," *International Journal of Advanced Robotic Systems*, vol. 4, no. 1, pp. 13–16, 2007.

[19] S. Hrabar, G. S. Sukhatme, P. Corke, K. Usher, and J. Roberts, "Combined optic-flow and stereo-based navigation of urban canyons for a uav," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[20] S. M. Nils Gageik, Michael Strohmeier, "An autonomous uav with an optical flow sensor for positioning and navigation," *International Journal of Advanced Robotic Systems*, vol. 10, p. 9, July 2013.

[21] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing," *2011 IEEE International Conference on Robotics and Automation*, 2011.

[22] A. Ferrick, J. Fish, E. Venator, and G. S. Lee, "Uav obstacle avoidance using image processing techniques," *IEEE*, vol. 12, pp. 73–78, 2012.

[23] M. V. Scott Lenser, "Visual sonar: Fast obstacle avoidance using monocular vision."

[24] J. Hoffmann, M. Jungel, and M. Lotzsch, *A Vision Based System for Goal-Directed Obstacle Avoidance*, vol. 3276, pp. 418–425. Springer, 06 2004.

[25] B. Sinopoli, M. Micheli, G. Donato, and T. J. Koo, “Vision based navigation for an unmanned aerial vehicle,” *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1757–1764, May 2001.

[26] S. Ahrens, D. Levine, G. Andrews, and J. P. How, “Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments,” *2009 IEEE International Conference on Robotics and Automation*, pp. 2643–2648, 2009.

[27] H. B. Su-Cheol Han, “‘proportional navigation-based optimal collision avoidance for uass’,” *2nd International Conference on Autonomous Robots and Agents*, pp. 76–81, January 2004.

[28] T. S. Yoav Gottlieb, “Uavs task and motion planning in the presence of obstacles and prioritized targets,” *Sensors*, pp. 29734–29764, November 2015.

[29] M. V. James Bruce, *Real-Time Randomized Path Planning for Robot Navigation*, p. 6. Springer, 2003.

[30] Y.-J. Tsai, C.-S. Lee, C.-L. Lin, and C.-H. Huang, “Development of flight path planning for multirotor aerial vehicles,” *Aerospace*, vol. 2, pp. 171–188, April 2015.

[31] P. H. Kourosh Naderi, Joose Rajamäki, “Rt-rrt*: A real-time path planning algorithm based on rrt*,” *ACM*, pp. 113–118, November 2015.

[32] B. D. Newton, “Executing rrt paths with the ar.drone quadrotor,” December 2012.

[33] S. Hrabar, “3d path planning and stereo-based obstacle avoidance for rotorcraft uavs,” *Intelligent Robots and Systems*, pp. 807–814, October 2008.

[34] O. S. R. Foundation, Osrf, "Why Gazebo?,” gazebo. [Online]. Available: http://gazebosim.org/. [Accessed: 31-May-2019].

[35] T. Braunl, "SubSim." [Online]. Available: robotics.ee.uwa.edu.au/auv/subsim.html. [Accessed: 31-May-2019].

[36] N. R. Stevens, "submarine game forums, news, patches, and mods,” Subsim. [Online]. Available: http://www.subsim.com/index.php. [Accessed: 31-May-2019].

[37] “Plane 11 Flight Simulator | More Powerful. Made Usable.,” X. [Online]. Available: https://www.x-plane.com/. [Accessed: 12-Mar-2019].

[38] “Plane Developer Site,” X. [Online]. Available: https://developer.x-plane.com/. [Accessed: 12-Mar-2019].

[39] "FlightGear Flight Simulator." [Online]. Available: https://www.flightgear.org/. [Accessed: 31-May-2019].

[40] U. Technologies, “Unity,” Unity. [Online]. Available: https://unity.com/. [Accessed: 12-Mar-2019].

[41] “About ROS,” ROS.org. [Online]. Available: http://www.ros.org/about-ros/. [Accessed: 12-Mar-2019].

[42] "Is ROS For Me?", ROS.org. [Online]. Available: http://www.ros.org/is-ros-for-me/. [Accessed: 31-May-2019].

[43] J. Kang, "An interactive explanation of quadtrees." [Online]. Available: https://jimkang.com/quadtreevis/. [Accessed: 31-May-2019].

[44] C. Kingsford, "Quad Trees." [Online]. Available: https://www.cs.cmu.edu/ ckingsf/bioinfo-lectures/quadtrees.pdf. [Accessed: 31-May-2019].

APPENDIX A: NOMENCLATURE

Distribution:	A bundle/version of software that is ready for use
Drivers:	Files that enable communication between software and hardware
Frame:	A standalone image from a video sequence
Framework:	A package containing common code with basic functionality, which can be overridden to contain specific functionality
FOV:	Field Of View
Gazebo:	Simulation software
Library:	Collection of implemented techniques and algorithms
LiDAR:	Light Detection And Ranging
Linux:	Operating system
MTOW:	Maximum Take-Off Weight
NAS:	National Air Space
OpenCV:	Open Computer Vision, library containing image processing and computer vision algorithms
Package:	A collection of files and information about those files
PID	Proportional-integral-derivative
Platform:	A base group of technologies used for development of other technologies
Quadrotor:	Multirotor aerial vehicle comprised of four rotors
RADAR:	RAdio Detection And Ranging
ROS:	Robot Operating System
SAA:	See And Avoid

SDAA:	Sense Detect And Avoid
TUM:	Technical University of Munich, package developed for ARDrone quadrotor
UAS:	Unmanned Aircraft System
UAV:	Unmanned Aerial Vehicle
Ubuntu:	Linux based operating system distribution