Development and Testing of a Simulation Environment for an Unmanned Aerial Vehicle

by

Shashank Sunil

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Applied Science

In

Aerospace Engineering
Carleton University
Ottawa, Ontario, Canada
December 2013

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Development and Testing of a Simulation Environment for an Unmanned Aerial Vehicle

Submitted by Shashank Sunil
in partial fulfilment of the requirements for the degree of
M.A.Sc. Aerospace Engineering

_____

Jeremy Laliberte, Thesis Supervisor

_____

Carleton University
2013

Abstract

This thesis describes the development of a simulation environment for the GeoSurv II Unmanned Aerial Vehicle (UAV) developed by the Unmanned Aircraft System Technologies Project which is a collaborative research program between Carleton University and Sander Geophysics Ltd (SGL). Unmanned Aircraft System Technologies Project continues to evolve and therefore, many of the avionics sub systems has been carried out by the previous students. The problem considered in this thesis is that of integrating the existing systems and two new sub-systems developed in this research, into one integrated autonomous simulation environment. The solution for this problem included developing a concept for an autonomous system but more importantly to develop a simulation environment in which the sub-systems can be tested individually or as whole system when required. The new simulation environment developed in this research includes all aspects of the UAV system, from external environmental factors to the flight control responses caused by the avionics and the aerodynamics of the UAV. The framework is also able to produce stereoscopic data that can be used to test various image processing algorithms. Lastly, a communication protocol was developed which bridged the gap between the simulation framework and the real world aircraft.

Acknowledgement

I am heartily thankful to my supervisor, Jeremy Laliberte whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. I would also like to thank him for his in-depth review of my thesis, which greatly contributed to the improvement of this document.

I owe my deepest gratitude to my friends Shahaboddin Owlia and Rytis Verbickas for their extensive insight on obstacle avoidance and detection algorithms, usage of Simulink and Matlab and also with the help with experimental setup for experiments performed in this research.

I would like to thank the research group under my supervisor for their support and suggestions provided during weekly meetings. I am also grateful towards Mr. Brian Wattie and the volunteers who helped and supported me during the flight tests.

Not the least, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

Table of Contents

List of Tables

List of figures

This page is intentionally left blank.

List of Acronyms

ACRONYMS	DEFNITION
3DMLW	3D Markup language for Web
ADSB	Automatic dependent surveillance-broadcast
AGL	Altitude above ground level
ATB	Aircraft test bed
CG	Centre of gravity
CIC	Computer-In-Control
CRD	Collaborative Research and Development
CRF	Camera Reference Frame
DCM	Directional Cosine Matrix
DOF	Degree of freedom
FOV	Field of View
GA	General Aviation
GCS	Ground Control System
MSL	Mean-sea-level altitude
MTOW	Maximum take-off weight
NSERC	Natural Sciences and Engineering Research Council of Canada
PIC	Pilot-In-Command
PPM	Pulse position modulation
PWM	Pulse width modulation

RC	Radio control
RGB	Red Green Blue
SDK	Software development package
SFOC	Special Flight Operation Certificate
SGL	Sander Geophysics Limited
STPC	Serial to PPM Conversion
UAVs	Unmanned Aerial Vehicles
VRML	Virtual Reality Modeling Language

1.0 Introduction

This section provides a thesis overview, and a brief literature review along with the contributions made during this research project.

1.1 Thesis Overview

This thesis is comprised of 7 chapters. Chapter 1 provides the literature review of prior research that has been conducted on different aspects of unmanned aerial vehicles (UAVs) and also highlights the contributions made during this specific research project.

Chapter 2 provides a detailed literature review of requirements for the use of UAVs and also provides a brief summary of the Canadian UAV industry.

Chapter 3 discusses the developments performed within the larger Carleton University research group in the field of UAV technology development. These developments are based on the industry requirements mentioned in chapter 2, while following the regulatory requirements and the limitations placed on Canadian UAV operators.

Chapter 4 discusses the establishment of a simulation model of a UAV, which considers all possible aspects of UAV performance. The research described in chapter 4 is highly dependent on the past and ongoing research performed within Carleton University which is mentioned in chapter 3.

Chapter 5 identifies the need for a 3D virtual environment in order to consider some of the aspects of the UAV simulation encountered in chapter 4. In Chapter 5 a description of the

successful solution of the problems encountered while creating the simulation framework in chapter 4 is presented.

Chapter 6, as in chapter 5 also describes the solution to a problem encountered during Simulink framework creation in chapter 4. Chapter 6 describes a technique to bridge the gap between the simulation world and the actual aircraft. A communication protocol between the two elements was established and tested.

Finally chapter 7 summarizes all the results from the previous chapters and also provides possible future research topics to develop a more complete and robust system.

1.2 Literature Review

Creating and simulating models as part of the design and modification process for new systems is a technique used for rapid testing and verification. Changes and modifications are made to the model, evaluating its intended function against the requirements. Thus, modeling allows faster verification and iteration of the design change process without having to conduct expensive tests with prototypes of the system.

There are fewer publications that describe a single model which comprises of the modeling, simulation and flight test of various systems found in Unmanned Aerial Vehicles (UAVs), when compared to those written about individual systems found in UAVs. The few relevant documents focus on only one aspect of the UAV. For example, the research performed by Fong See Yan of UTeM, examines the effect of variation of angle of attack of elevator deflection on the lift coefficient, drag coefficient and pitching moment coefficient. This research focuses on the derivative constants that are required to develop a non-linear model for longitudinal

stability [Fong, 2009]. Research performed by Leonine Kunzwa of the University of Hertfordshire, focuses on defining, estimating and analysing of both the longitudinal and lateral derivatives. These derivatives were used in the already available non-linear model of the Aerosonde UAV in the Aerosim Blockset in Simulink. Thereby modifying the Aerosonde model to the required UAV model [Kunzwa, 2011].

A number of research projects have been performed on obstacle detection and obstacle avoidance. These examined various sensors such as camera, laser and sonic sensors used for obstacle detection along with a wide variety of algorithms used for both detection and avoidance. For example, Yeonsik Kang of the University of California, used a combination of cameras and Kalman Filter algorithms to deal with obstacles [Kang, 2011]. Though there are multiple papers describing obstacle detection research, only few actually deal with UAVs and even fewer dealt with fixed wing UAVs – the majority of research was on rotary wing UAVs such as quadcopters. Most of the fixed wing UAV research found during this literature review focuses on directly implementing algorithms on a functional UAV rather than testing the algorithm in collaboration with the existing non-linear Simulink model.

Another group of research focuses on using hardware-in-the-loop set-ups to test the hardware. For example Adiprawita Widyawardana, Ahmad Adang Suwandi and Semibiring Jaka of Institut Teknologi Bandung, shows that many aspects of the UAV autopilot hardware can be tested without risking the valuable airframe and payload [Adiprawita, 2007]. Research performed by Chao Yun, Xiao-min Li and Zong-gui Zheng of Mechanism Engineering College,

shows the use of Simulink to perform hardware-in-the-loop testing of UAV autopilot systems while using the Aerosim blockset [Chao, 2013].

As mentioned earlier, all of this prior research focuses only on one aspect of the UAV at the time such as stability derivatives, nonlinear models, obstacle avoidance and detection and hardware-in-the-loop simulation. There is research considering other possible sub-systems of a UAV such as target tracking [Wheeler 2006] and co-operative control between UAVs [Vires, 2013] , but there was hardly any (if any) research that considers integrating all these sub-systems within Simulink to test the interdependent functioning of the whole system.

Hence, the research described in this thesis takes on the challenge of integrating the available systems and also generates new techniques to tackle problems faced during the integration process.

1.3 Contributions

Based on the literature review, three main contributions were made by this research as follows:

- **Development of simulation framework**: Chapter 4 discusses the development of a simulation framework that considers all the relevant developments being performed within the university. In this contribution various available models are brought together into a single model. For most cases the various sub models have been considered as a black box as the detailed workings are beyond the scope of this project. It should be noted that only a limited understanding of these blocks were required to use them for the development of the simulation framework.

- **Development of 3D environment:** Chapter 5 recognises the need to include a 3D environment within the simulation framework in order to test obstacle avoidance and detection algorithms. In this section, a sub model is created within the simulation framework, which generates a video output that can be used to test image processing algorithms.
- **Development of communication protocol:** Chapter 6 bridges the gap between the created simulation framework and the aircraft test bed, by developing a communication interface between the two platforms with the help of a microcontroller.

2.0 Unmanned Aerial Vehicles (UAV)

Unmanned Aerial Vehicles (UAVs) are aircraft with no human pilot on board. They are either controlled by human pilots on the ground or pre-programmed to fly missions autonomously or some combination of the two. There are different types of UAVs, for example, a UAV could be a helicopter, multi-rotor or fixed wing airplane [Austin, 2010]. There are many more types of UAV that are used based on the mission requirements. Every UAV configuration has its own advantages or disadvantages, depending on the particular mission and payload.

2.1 UAV Autonomous Systems

An autonomous system is defined as a system in which high level goals are assigned to the system, and the low-level tasks that are required to accomplish that goal are performed by the system itself. Therefore, an autonomous system requires minimal input from its user once it starts its mission [Winnefeld, 2013]. Some of the features that could make an autonomous UAV system are listed below.

- *Path Planning*: The ability of a system to navigate via a particular flight path as per user requirements [Butenko, 2003].
- *Collision avoidance*: The capacity of the system to sense obstacle(s) and avoid it in time to prevent a collision. A complex system would be able to avoid moving obstacles too [Weibel, 2004]. This is required for integration into airspace with other traffic [Transport Canada-2, 2013].
- *Sensor networks*: The system should be able to gather information with available sensors on board. The sensors should be not only used for successful flight operations but also for other data gathering based on the mission requirements.
- *Co-operative control*: The capability for a group of UAVs to work together to accomplish a defined task [Redding, 2010].
- *Human factors*: In order to operate the UAV, the user does not need be an experienced pilot. Thus, anyone with no UAV experience should be able to learn to operate the UAV quickly. Therefore, making UAV a simple tool that can be used on a field. The operator might require a license or a specific training, but the goal here is to consider the human factors and develop an easy to use design for UAV operation [Gawron, 1998].
- *Guidance/navigation/control (GNC)*: The system should be able to maintain the user specified path based on GPS co-ordinates, with presences of environmental factors such as wind [Winnefeld, 2013].

2.2 Canadian Developments

The UAV industry in Canada is growing; there are companies that deal with specific components of UAV, such as the company L-3 Wescam which attaches its targeting systems to UAVs [Mann,

2012]. Companies such as Brican Flight Systems Inc. design and manufacture Remotely Piloted Aircraft systems that deliver high quality digital information to scientists and researchers which require data from areal platforms [Members, 2013]. Other companies such as the ING Robotic Aviation, not only develop UAV systems but also provide services as, for instance, infrastructure inspection, aerial surveying, environmental monitoring and training & simulation [ING, 2013]. There are also various other established companies such as MMIST, MDA, Thales that are running UAV programs for various mission requirements. Many other are start-up companies being established throughout the country.

On May 17th 2013 the “Minister of the Economic Development Agency of Canada for the Regions of Quebec and Minister of Intergovernmental Affairs, announced that Aéroport d’Alma has been granted financial assistance to build the infrastructure needed to group all of the activities of the Unmanned Aerial System Centre of Excellence under one roof” [Quebec Economic Development Program, 2013]. This program is estimated to be over $4.3 million.

With more and more UAVs being deployed, Transport Canada is developing new regulations with a goal to “normalize” UAV operations within civil airspace.

Section 101.01 of the Canadian Aviation Regulations (CARs) states "Unmanned Air Vehicle" means a power driven aircraft, other than a model aircraft, that is operated without a flight crew member on board. In order to operate a UAV one must apply for Special Flight Operation Certificate (SFOC). Section 623.65 outlines information that should be submitted when making an application for a SFOC. The application should be a detailed document on the functioning and capabilities of the UAV along with the intended use of the UAV. The application should

demonstrate that the UAV can be operated safely, and ensures no hazard is created to persons or property on the surface [Transport Canada, 2013]. This is mostly guaranteed by equipping safety systems, such as Ignition Cut-off system into the UAV.

3.0 Background to the Present Project

This section discusses the developments being done within Carleton University in the field of unmanned aerial vehicles (UAV). Section 3.1 describes the *Unmanned Aircraft System Technologies Project* in which students are responsible for designing and building UAVs. Section 3.2 and 3.3 describes the applied and theoretical work being performed by graduate and undergraduate students, respectively.

3.1 *Unmanned Aircraft System Technologies Project*

The *Unmanned Aircraft System Technologies Project* is funded jointly by Sander Geophysics Limited (SGL) and the Natural Sciences and Engineering Research Council (NSERC) under a Collaborative Research and Development (CRD) Grant. Its objective is to develop a UAV that is capable of performing high resolution geophysical surveys. This project requires an aircraft to fly at low altitudes above the ground in order to perform high resolution geophysical surveys. Also, in order for the project to be profitable form an economic aspect, the aircraft should be airborne for as long as possible. A long endurance allows the UAV to survey a large area of land in a shorter time span, when compared to that of a lesser endurance UAV. A longer endurance UAV would also avoid excessive fuel consumption by avoiding multiple transits to complete the survey. Therefore, these requirements dictate that the UAV fly at low altitude and also provide high flight endurance [Owlia, 2013]. For this particular reason a fixed wing aircraft was chosen

as aircraft of this type configuration has the ability to fly at low altitude, while providing ample flight endurance (approximately 8 hours) as compared to rotary wing (e.g. helicopter) type platforms.

To date, the aircraft has been designed, built, and several low and high-speed taxi tests have been performed with the help of graduate and undergraduate students of Carleton University. The current iteration of the developed aircraft is named as GeoSurv II.

The aircraft designed is a twin-boom pusher aircraft with a wingspan of 4m, with two magnetometers installed on its wing tips to measure magnetic field gradients of the terrain the UAV is flying over. The aircraft is expected to have 8 hours of endurance. The magnetometers used are the same instruments used by SGL in their current manned aircraft.

Figure 1: GeoSurv II

Alternatively, aircraft test beds (ATBs) have also been developed by the team to test various avionics and autonomy sub-systems. Details of the ATBs are given in section 3.2.

As mentioned earlier an autonomous UAV must perform its function with minimal input from the user. Furthermore, the GeoSurv II needs a higher level of autonomy, as it is expected to fly at lower altitude where it can encounter a higher number of obstacles (such as trees, power lines, telecommunication towers, etc.) - when compared to that of a high altitude UAV.

Hence, research has been taken or is ongoing by graduate students at Carleton University on different aspects of the aircraft. For example, research on low cost composite airframe [Polowick, 2013], [Maley, JA. 2008], [Lares, A. 2012], obstacle detection [Owlia, 2013], obstacle avoidance [Verbickas, 2012], hardware-in-the-loop simulations [Hojeij, 2013], and magnetic interference of ferromagnetic sources on the aircraft [Caron M., 2013], [Forrester, 2011], are all part of the GeoSurv Project.

Details of autonomous systems that related to this document are mentioned in the subsequent sections 3.2 and 3.3.

3.2 Air Vehicle Hardware Development

As part of this thesis, an aircraft test bed was developed to experimentally test the developing avionics system. The test bed was developed by integrating enhanced avionics into an off-the-shelf RC model aircraft. A detailed description of the test bed is mentioned below.

3.2.1 ATB-10

Within the GeoSurv II project, a series of Avionics Test Beds (ATBs) were developed in the past by undergraduate students to assess the performance of various avionics components. The current autonomous test bed inherited into this research is the ATB-10 as discussed in the following section.

3.2.1.1 Senior Telemaster

The model aircraft used for the test bed is a Senior Telemaster, shown in Figure 2. The Telemaster is a high-wing tail dragger with a wingspan of 2.41m and a length of 1.60m. The aircraft typically has a maximum take-off weight of 2.6 kg and a cruise speed of 60-70 knots at 80% throttle [Senior Telemaster, 2013].

Figure 2: Senior Telemaster/ ATB-10 [Senior Telemaster, 2013]

The model has three sets of control surfaces: ailerons, elevators and a rudder [Senior Telemaster, 2013]. For propulsion, the aircraft is equipped with an electric motor that is powered by a 5000 mAh Li-Po battery. With the available battery, the test bed is capable of flying for approximately 20 minutes. Please note that this is considered as ample time for testing purposes. Additionally, the batteries can be replaced with fully charged once if required.

A nine channel 2.4GHz transmitter and receiver, shown in Figure 3, is used to transmit actuator and throttle commands by the pilot [DX8, 2013].

Figure 3: Spektrum DX8 Transmitter [DX8, 2013]

The Senior Telemaster was chosen for its large load capacity, and low cruise speed. These characteristics made the aircraft easier to control, and therefore, a suitable test bed candidate for autonomy research. Furthermore, due to its large load capacity, the test bed could carry a large payload consisting of additional equipment and sensors.

3.2.1.2 Telemetry

The telemetry link used for ATB-10 was a pair of MicroHard R n920-ENC radio modems. The radio modem, shown in Figure 4, operates at 900 MHz using frequency-hopping spread spectrum and are claimed to have a range of over 100 km. However, at this stage of the project the aircraft is not expected to travel beyond a distance greater than the line of sight. Furthermore, they weigh 220 g and are capable of transmitting data wirelessly at 19200-230400 bps [Nano n920-ENC, 2013]. Detailed specs can be found in the appendix A1. Figure 4 shows the back view of the modem.

Figure 4: MicroHard R n920-ENC radio modem [Nano n920-ENC, 2013]

The n920-ENC was selected due to its small size and weight which allowed easy placement of the modem on the ATB-10 without drastic effect on its maximum take-off weight (MTOW). Also the low power consumption of 1W is other criteria considered while selecting a suitable modem [Nano n920-ENC, 2013].

3.2.1.3 Autopilot

This section provides a detailed description of the selected autopilot for the ATB-10.

MicroPilot MP2028 autopilot

The autopilot used on ATB-10 is an off-the-shelf MP2028 commercial autopilot manufactured by MicroPilot. The MP2028, shown in Figure 5, weighs 28 g and measures 10 cm by 4 cm [MicroPilot, 2006]. Unlike the other autopilot MicroPilot provides a comprehensive technical support service which can be helpful if encountered with any problem. Thus, MicroPilot is used in the Unmanned Systems Technologies Project.

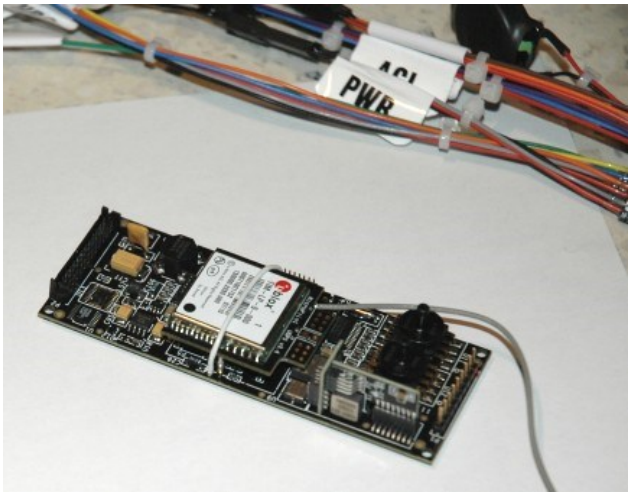


Figure 5: MP2028

The autopilot is equipped with high and a low-level controller. The high-level controller computes parameters such as required heading, altitude, etc. which are determined by the user/pilot. The low-level controller then computes the actuator inputs required to perform the flight commands. Unlike the high level commands, the low level commands cannot be manipulated directly by the user [MicroPilot, 2006].

The control laws are determined by PID feedback loop controllers, which are the type of controller that most commercial autopilots use. The MicroPilot uses a combination of twelve feedback loops to fly the UAV. Table 1 describes these twelve PID controllers.

Table 1: PID controllers [MicroPilot, 2006]

Name	Description
Aileron from bank	Controls the ailerons to minimize the difference between desired bank and actual bank.
Elevator from pitch	Controls the elevator to minimize the difference between desired pitch and actual pitch.

Rudder from Y accelerometer	Controls the rudder to minimize the difference between the desired value for Y accelerometer and the actual value. This is the feedback loop that coordinates turns.
Rudder from heading	Controls the rudder to minimize the difference between desired heading and actual heading. This feedback loop is used during take-off to keep the aircraft on the correct heading.
Throttle from speed	Controls the throttle to minimize the difference between desired speed and actual speed. This feedback loop is used during final approach and when the option to control speed via throttle and altitude via elevator is selected.
Throttle from altitude	Controls the throttle to minimize the difference between desired altitude and actual altitude. This feedback loop is used when the option to control altitude via throttle and speed via elevator is selected.
Pitch from altitude	Controls desired pitch to minimize the difference between desired altitude and actual altitude.
Pitch from AGL	Controls desired pitch to minimize the difference between desired altitude and actual altitude as measured by the AGL board. This feedback loop is enabled during landing and controls the flare.
Pitch from airspeed	Controls the desired pitch to minimize the difference between desired airspeed and actual airspeed. This feedback loop is enabled during climb and during level flight when the option to control altitude via throttle is selected.
Bank from heading	Controls the desired angle of bank to minimize the difference between the desired heading and the actual heading. This feedback loop is enabled any time your MicroPilot Autopilot is navigating.
Heading from cross-track error	Controls the desired heading to minimize the distance between your MicroPilot Autopilot and the line defined by the previous waypoint and the next waypoint. This feedback loop is enabled when the “fromTo” command is being run.
Pitch from descent	Controls desired pitch to minimize the difference between desired descent rate and actual descent rate.

Here, it should be noted that all the controllers are not enabled together. Depending on the task, a particular set of controllers are enabled in a precise sequence to achieve the required goal. To demonstrate the complexity of the system, an example of levelled flight is described in Figure 6. The enabled feedback loops for a level flight is shown in bold in Figure 6. This example assumes that the MicroPilot is configured to use the elevator to control altitude and the throttle to control airspeed. If the MicroPilot were configured to use the elevator to control airspeed then the pitch from airspeed feedback loop would be enabled instead of the pitch from altitude and throttle from altitude would be enabled instead of throttle from airspeed [MicroPilot, 2006].

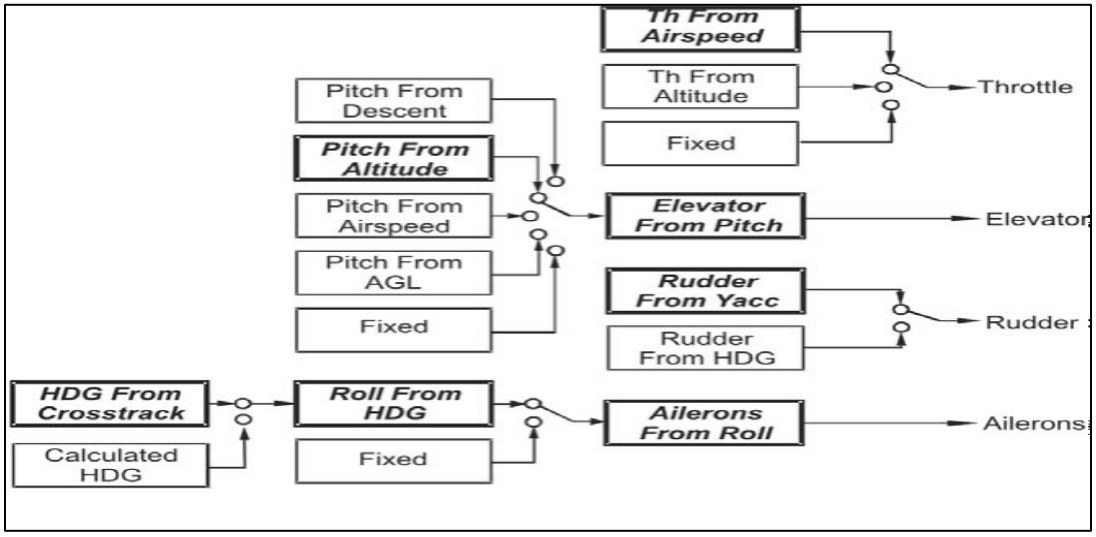


Figure 6: Set of PID controllers used for level flight [MicroPilot, 2006]

After tuning the PID controllers in the autopilot by following the instructions provided by the manual, the autopilot is able to maintain a stable flight. Furthermore, the aircraft can be set to hold altitude, hold airspeed, navigate using GPS waypoint and is able to perform other features. The autopilot is also equipped with sensors such as GPS, accelerometers and gyroscopes which

are essential for the above mentioned tasks. Additionally the data from these sensors can be recorded for further analysis.

Once the autopilot is installed in the aircraft, the operator can choose between two flight modes: Pilot-In-Command (PIC) and Computer-In-Control (CIC). In PIC mode, the commands received from the transmitter are used to guide the aircraft. In contrast, when the autopilot is switched to CIC mode, the autopilot performs the flight mission that is has been commanded and commands received from the transmitter are ignored. The flight mode is chosen by the operator and gives the operator the ability to engage and disengage the autopilot at will.

The MP2028 is sold with its supporting software named Horizon which can be installed in the Ground Control System (GCS). Horizon allows the operator to not only monitor the UAV but also acts as a data acquisition system which can be used to collect relevant data. Horizon also allows complete control of the UAV to the pilot .Therefore, the pilot can control the UAV via the radio control or Horizon based on the experimental requirements.

The communication between the autopilot and GCS is established through a standard serial port with the help of the modem described in section 3.2.1.1.

It should be noted that, for this particular thesis, the MicroPilot is not being used for stabilised flight, but as a data acquisition system, and thus the MicroPilot is always on PIC mode.

3.2.2.4 On-Board Avionics and GCS setup

The avionics setup within the ATB-10 is shown in Figure 7. The description of most the components have been mentioned in the above section. Additional ATB-10 is powered by three separate batteries. The description of the batteries are mentioned below,

1. Spektrum NiMh 2150mAh 6.0V for the receiver
2. Great Hobbies Li-Po 3200mAh for the servos, MicroPilot and the Modem
3. Turnigy Li-Po 5000mAh for the 80 amp motor

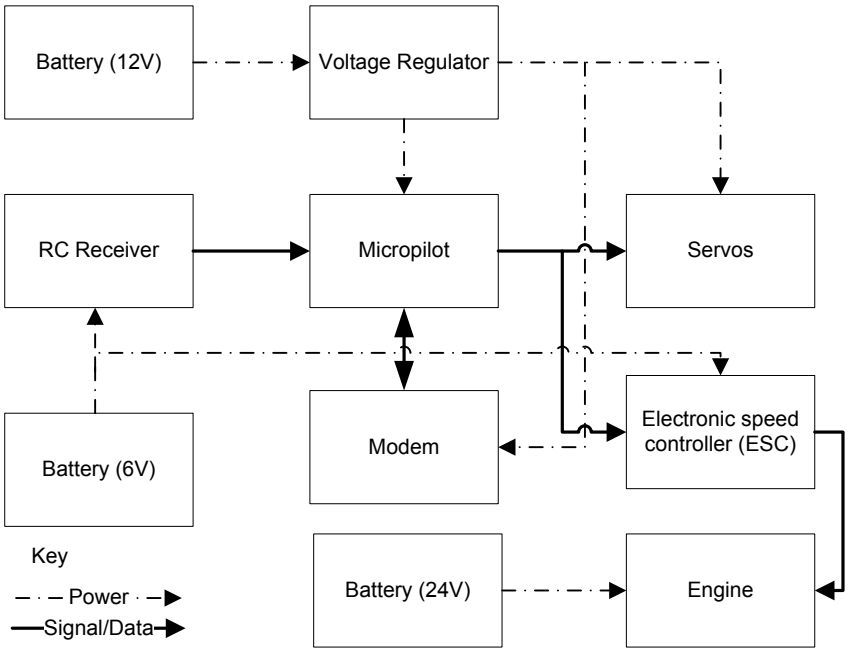


Figure 7: Avionics Setup

3.2.1.4 Special Flight Operation Certificate

The ATB-10 is operated under a Special Flight Operations Certificate (SFOC) issued by Transport Canada [Transport Canada, 2013]. This certificate permits the use of the Senior Telemaster for

this research provided all conditions outlined in the SFOC are met. These conditions can be found in the SFOC which is attached in appendix A2.

3.3 Theoretical Developments

There are various developments at Carleton University by other research related to this project. However, only those directly associated with the work described in this thesis have been listed below. It should be noted that the theories selected in the following section have been already selected by pervious students working under this project and that these and NOT selected by the author of this thesis. Therefore, more description of these theories may be acquired from the previous respective research work.

3.3.1 Obstacle Avoidance

Since the GeoSurv II is expected to fly at an altitude less than 50m above ground level (AGL), the obstacles that are expected to be encountered are power line, telecommunication towers and trees. In the research performed till date, including the present project assumes that the GeoSurv is not expected to encounter other aircraft in its vicinity. For the foreseeable future, UAV operations in Canada must be segregated from manned aircraft so this is a reasonable assumption. Also the GeoSurv is not expected to fly in cluttered or built-up urban environments but in open and empty lands with fewer obstacles. The latest research done on obstacle avoidance is by Owlia [Owlia, 2013]. According to his research, the GeoSurv II obstacle avoidance maneuvers are calculated using potential flow theory and panel method. This technique is borrowed from fluid mechanics and is used to generate evasive maneuver path when obstacles are encountered. His research considers the GeoSurv's flight handling

constraints and is able to produce feasible maneuvers for the UAV. Currently the research considers only a single obstacle; however, his research can be modified to accommodate multiple obstacles [Owlia, 2013]. This research does not require a specific autopilot; it is assumed that a flight management system will send the appropriate signals to the flight controls on the aircraft.

There are various other techniques such as road map methods, rapid exploring random tree, artificial potential fields, harmonic potential functions and model predictive control which are used for obstacle avoidance. A road map is a network of straight lines that do not intersect obstacles and connect the starting point to the goal of a robot. To create such a network, the operational space of the robot is divided into an obstacle free subspace and an obstacle subspace. A network of straight lines within the obstacle free subspace is created [Shanmugavel, 2011]. This network is created using various algorithms such as Dijkstra and A* which are mentioned by the works done by LaValle [LaValle, 2006].

Rapidly Exploring Random Tree (RRT) is a probabilistic method to determine an obstacle free path in a cluttered environment. Initially, the starting point of the path is used to initiate a “tree". This process is followed by generating a random point and the nearest node on the tree to this random point is found. A new node is then created on the line connecting the random point and the nearest node from the previous step. Finally, if this point does not coincide with any obstacle it is added to the tree. This process is done in an iterative manner until the goal is reached [Mujumdar, 2011] [LaValle, 1998].

The work done by Khatib considers a repulsive Artificial Potential Field (APF) that was assigned to an obstacle present in the operational space of a robotic manipulator. An attractive APF was generated to the goal and a repulsive APF was generated from the obstacle [Khatib, 1986]. However, it was found that, if the goal is close to an obstacle, the repulsive force generated by the obstacle can be larger than the attractive force of the goal and as a result the robot will miss the goal point [Khosla, 1988].

Kim's work suggested the usage of harmonic potential functions to eliminate the limitations caused by APF method [Kim, 1992].

In the model predictive control, an optimal control input sequence is found over a finite receding horizon for a given system with respect to some predefined cost function. The cost function is the deviations from the reference input, high control inputs and forbidden states. The receding horizon is a predictive time horizon over which the cost function is minimized. A new optimal control sequence is determined over a new finite horizon starting at every time step [Richalet, 1993] [Mujumdar, 2011].

Tough all these techniques are promising, this research project is to built upon the word done by previous students. Hence, Owlia's algorithm was used for this research project

Owlia's thesis derives the required equation to calculate the maneuver and implements it in Simulink [Owlia, 2013]. However, the Simulink model created during this research is able to perform only a *"pitch-up"* maneuver. The model does not perform the required *"fly-around"* maneuver or implement decision logic to select between *"fly-around"* and *"pitch-up"* maneuver.

The current research implements the selection of *"pitch-up"* and *"fly-around"* maneuver. This research also executes *"fly-around"* maneuver by heavily modifying the existing "*pitch-up*" model from Owlia's research. Details of these contributions are mentioned in chapter 4, section 4.2.3.

3.3.2 Obstacle Detection

Accurate obstacle detection is critical for obvious reasons as obstacles need to be first detected properly before it can actually be avoided. There are various methods that can be used to achieve obstacle detection. For example, active range sensors that include laser, radar or sonar based solutions are available. Laser scanners and rangefinders are the most commonly used sensors for UAVs under laser based solutions. It should be noted that, laser rangefinders provide focused results, therefore, returns a single range estimate in a single direction [Griffiths, 2007] [Saunders, 2005]. Special pan/tilt rigs are constructed to obtain a sweep [Geyer, 2006]. These devices have distance measuring rage capabilities up to 400m. Laser scanners are alternative solutions to laser rangers as it provides a wide field of view. However, it usually limited to a 2D scanning plane. These laser sensors are usually lighter, when compared to traditional off-the-shelf stereoscopic based solutions. These smaller laser sensors have the maximum range that is limited to around 50m [Hrabar, 2012]. Furthermore, custom 3D laser radar scanner solutions have been used on UAVs that were able to identify obstacles at a range of 150m [Scherer, 2007]. Alternatively, obstacle avoidance feasibility have been tested with the use of doppler radar systems on UAVs [Viquerat, 2008], the portable system that can be mounted on UAVs only provide a limited range of 10-15m.

All these have been already considered by graduate students working at Carleton University. The most recent research considers acquiring obstacle vision by focusing on using a stereoscopic vision in order to determine the exact coordinates of the obstacle in 3D space [Verbickas, 2012]. A stereoscopic solution seemed to be the most cost effective solution to detect obstacles. A basic understanding of the stereoscopic vision concept is described below.

3.3.2.1 Stereoscopic Vision

A stereoscopic vision system can use two cameras to acquire the required data. Using the data acquired, one is able to render objects in a more realistic approach. i.e. more details of the environment can be extracted since the viewer gains depth perception of the environment. Therefore, the obstacle's co-ordinates can be plotted in a 3D co-ordinate system [Meer, 2013]. The process of acquiring data from two cameras to detect an obstacle is illustrated below with help of an example.

3.3.2.2 Stereoscopic Vision Example

First, a pair of image is compared with each other as shown in figure below. The two images are sparingly different from each other as the camera location while taking the picture is different. The cameras are located besides each other with a specific amount of distance between them. Therefore, if the location of the cameras were provided in a Cartesian system, two out of three axes would have the same units, for example, the co-ordinates would look like (**4**,**2**,3) and (**4**, **2**,1). The top picture is taken from the left camera and the bottom is taken from the right camera. The disparity between the pictures in demonstrated in Figure 8 [Szeliski, 2013], i.e. the location of pixels composing the images is offset when the images are compared to each other. This is due to the change in location of the cameras responsible for capturing these images.

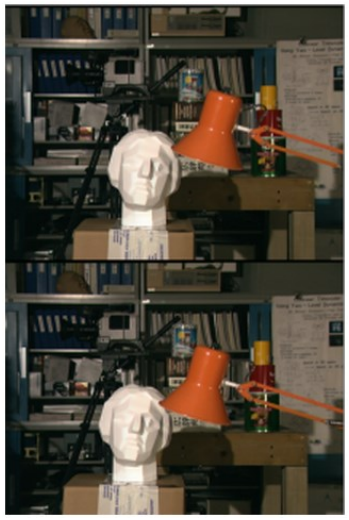


Figure 8: Comparing Right image with Left Image [Tsukuba, 2013]

Figure 9 represents images from Figure 8 place overlapping each other. Figure 9 demonstrates that the closer objects are misaligned more while compared to the further located object. For example, the lamp is more distorted than the statue head [Meer, 2013].

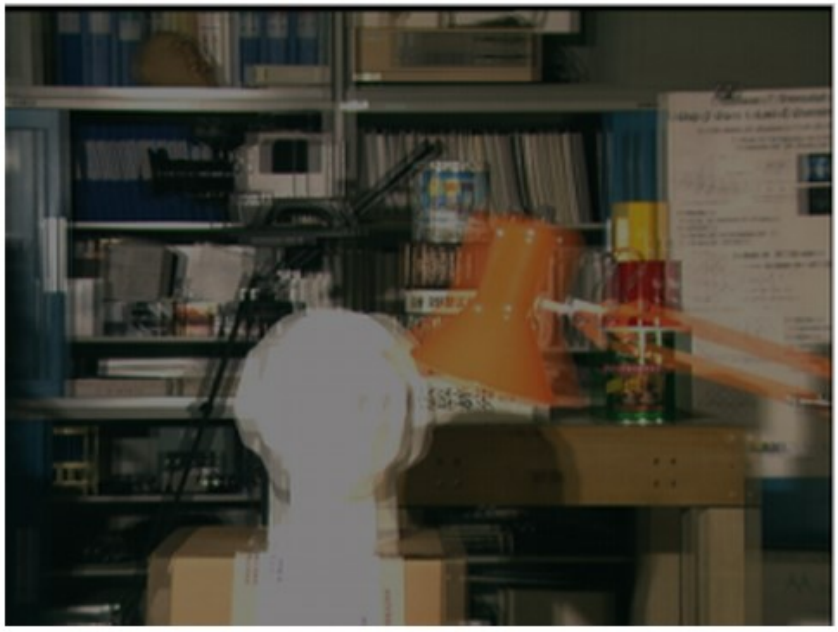


Figure 9: Overlapping Images [Tsukuba, 2013]

Various methods are available to extract information from the image disparity, for this example slide and subtract operation from right-to left (R-L) and left to right (L-R) were performed. Distorted pixels are removed by using the disparity from R-L or L-R pass based on the lowest matching difference values acquired from slide and subtract operation. The disparities are divided into four possible ranges. Figure 10 shows orange, yellow, light blue, dark blue as the four ranges. The red color indicates closer objects (pixel) and the blue ones represent the furthest object (pixel) [Vision, 2013].

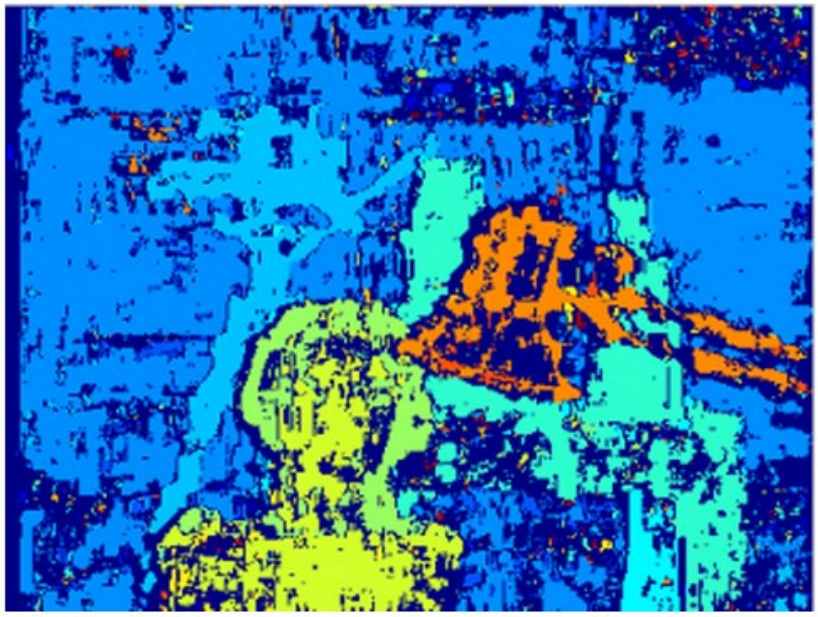


Figure 10: Disparity in terms of intensity value [Vision, 2013].

This example is the most basic example, for the UAV obstacle detection system, the system should be able to not only detect the object but also provide 3D coordinates of the object with minimal error so that the obstacle avoidance system can act accordingly.

3.3.3 Hardware-in-the-loop Simulation

There are several approaches that can be taken to implement hardware-in-the-loop simulation. The research works of this topic by the students under this project are in its initial stages. Therefore, a brief description of some of the simulation architectures that were explored by other research are mentioned below.

In the most common method, the data that were expected to be produced via IMU, GPS and air data probe (pitot tube sensor) can be simulated and sent to the avionics unit with the help Simulink. The avionics would respond to these data/signals and respond to it by sending signals to the respective components of the avionic system, such as the servo. In this scenario, the Simulink model generates the sensor data based on the models created to emulate the atmosphere, environment and motion of the atmosphere. However, this method generally requires expensive state of the art equipment to produce real-time results. Thus, Simulink is not able to provide a true real-time execution without the state of the art equipment [Mueller, 2007].

The second method is to model the sensors and the avionics response within Simulink. In this method, no actual hardware is connected to the final hardware-in-the-loop system i.e., the sensor behaviour is analysed by passing various possible data into the sensors, and the output is recorded. The same is done to gather possible response by the avionics response due to various sensor inputs. Thus, this technique focuses on the various test cases required to develop a full functional hardware-in-the-loop model. Assuming the test cases cover all possible scenarios, the avionics unit can essentially be modeled within Simulink [Hojeij, 2013]. Currently

the research being performed by the student at Carleton University is based on this second approach. The research is in its initial stage and thus not much is known on the model being created in Simulink. However, it has been confirmed that the model will be incorporating the Aerosim blockset, which is also being used in this research [Hojeij, 2013]. Thus uniformity is maintained between research projects within Carleton University, allowing easier integration in the future.

3.3.4 Integrating existing physical and theoretical development

The research described in section 3.3 is currently being carried out independently. In order for the project to move forward it is paramount that all these different systems be integrated into one on-board system that should be used on the GeoSurv Avionics Platform. Therefore, this present thesis makes its contribution by integrating the sub-systems. Thus, creates an autonomous avionics platform for the GeoSurv II.

However, since none of the sub systems are yet complete, it is not sensible to develop the actual physical platform but instead to develop a simulation platform that can handle the above mentioned various sub-systems before actually testing it on the GeoSurv. This simulation platform should be adaptable to not only different UAVs, but also be able to incorporate possible future systems such as secondary obstacle detection system (e.g. Automatic dependent surveillance-broadcast (ADSB)).

Although this thesis project is oriented towards development of a system for the GeoSurv, the proposed system should also be easily adaptable to any other UAV. The simulation environment should be able to load different algorithms, hence providing a hassle free testing

environment for the various available sub-systems. Chapter 4 discusses the developed simulation environment.

During the development phase of the virtual environment, two problems were encountered. The first one was the lack of data within the virtual environment to test an obstacle detection algorithm, and the second problem was the inability of the virtual environment to communicate directly with the ATB-10 for field testing purposes. The solutions to these two problems are the remaining contributions by this thesis and, are described in chapter 5 and 6, respectively.

4.0 Virtual Simulation Framework for UAVs

The successful certification of new autonomous UAVs can be enabled by flight testing which involves debugging of both the hardware and software before deploying the UAV in the field [Perhinschi, 2010]. Flight testing can be expensive and poses a safety risk, especially while dealing with aircraft such as GeoSurv which is a novel, untested platform. Although these cost and risk can be reduced with a help of using a test bed such as the ATB-10, it would be even more effective if the autonomous system could be tested before it was implemented on the ATB-10.

There are various software that can be used in order to build a simulation framework. Some of the most commonly used software are C++ libraries such as JSBSim, Aviones and Simulink. JSBSim is an open source flight dynamics model. Being a library most of the work done would be script based and less of dynamic modeling. This can be demonstrated in the research done

by James M. Goppert of Prudue University, USA [Goppert, 2012]. The convenience of dynamic programing is demonstrated in section 4.1. There are other UAV flight simulator software such as Aviones that are open source and fairly developed, and hence have comparatively less documentation, when compared to that of fully developed software such as Simulink and JSBSim.

Alternatively, Matlab/Simulink can be used to create the simulation framework. Matlab has proven to be the most popular by a number of researches. Matlab, in general has been considered as the most demanding computational software in the world [Goppert, 2012]. Although there are many projects investigating on UAVs using Matlab, most of them focus on only one aspect of the UAV. As mentioned earlier, this new research attempts to bridge the gap between prior Carleton research projects by integration of various models into one; thereby, creating a fully functional simulation environment.

This chapter discusses the creation of a simulation framework that is capable of predicting the outcomes of the autonomous system. After performing this research project, it was found out that, the simulation framework has adequate flexibility to handle the autonomy's sub-systems as an individual entity or a group of sub-systems as a single entity. Based on the simulation results from the framework, necessary modifications can be made to the aircraft before actually performing any flight tests on the ATB-10 or GeoSurv II.

4.1 Matlab/Simulink

Construction of a simulation framework is a complex process as all the required details of the aircraft's flight stability and performance have to be considered. For example, the forces and

moments due to the environment, aerodynamics, propulsion and other effects should be accounted for building a non-linear UAV flight mechanics model [Hostmark, 2007].

The simulation framework was created using Matlab/Simulink. As mentioned earlier, there are alternative programs that can be used to achieve the same results provide by the simulation framework created with Matlab. However, Matlab has features/functions that are already built into it. These feature/functions can be used readily as opposed to creating the same available features/functions from scratch. For example, matlab provides disparity function that calculates the disparity between two images. Besides, available C++ functions can be loaded into Matlab easily.

Simulink block is a dynamic interaction environment which can afford us dynamic modeling, simulation and analysis. Majority of time a Simulink user does his/her work with less script writing and more usage of mouse clicks which emphasis on visual modeling. Simulink contains many basic modeling such as aerospace module, computer vision, data acquisition and many more. Aerosim toolbox is a specific toolbox aimed at UAV simulation researching [Kai-bo, 2009].

Aerospace toolbox is another toolbox that is available in Simulink which provides the same capabilities as the Aerosim toolbox. However, unlike Aerospace toolbox, the Aerosim blockset is already equipped with a working model of one or more UAVs. There are also a number of documents that are provided by various sources available online. The large number of successfully published documents using Aerosim blockset itself demonstrates the ease and reliability of using the blockset. Consequently the required simulation framework can be created faster with the help of Matlab [Hostmark, 2007].

4.2 Simulation Framework

This section describes the simulation framework. The simulation framework created in this research deals with non-linear aerodynamic modelling, obstacle avoidance and detection, hardware-in-the-loop simulation, interfacing with flight gear and serial interface with the ATB-10. This research integrates the existing models into one model and determines whether the created simulation framework is a viable option to test the individual sub-systems and the avionics system as a whole. The processing speed of the framework was not examined, as the objective of the framework is to provide a testing environment. Therefore, as long as the Simulink provided solutions within few minutes, it was deemed as acceptable. The current configuration of the fully functional model with the PC running Intel Core i5 processor at 2.3 GHz provides results within 2-3 minutes. Where, the simulation time was set to 60 sec. It should be noted that the simulation time and actual clock time are not the same. For example, if running a simulation for 60 s usually does not take 60 s as measured on a clock. The amount of time it actually takes to run a simulation depends on many factors including the complexity of the model, the step sizes, and the computer speed. This research project not only integrates the existing model but also add additional components. Figure 11 shows the block diagram of the simulation framework.

Each of these blocks exists as a Simulink model. These blocks are either available from the Simulink library or were created from scratch. Figure 11 represents the flow of data from each block. The sections below describe each of these blocks in detail. All the blocks, except for the "*Non-linear Model*", are created through this research or other researches done within Carleton University. The "*Non-linear Model*" block is acquired from the Aerosim library. The Simulink

model diagram representing the above block diagram is shown in Appendix E2. The *VRML* block is described in chapter 5 and the *Serial Interface* block is described in chapter 6.

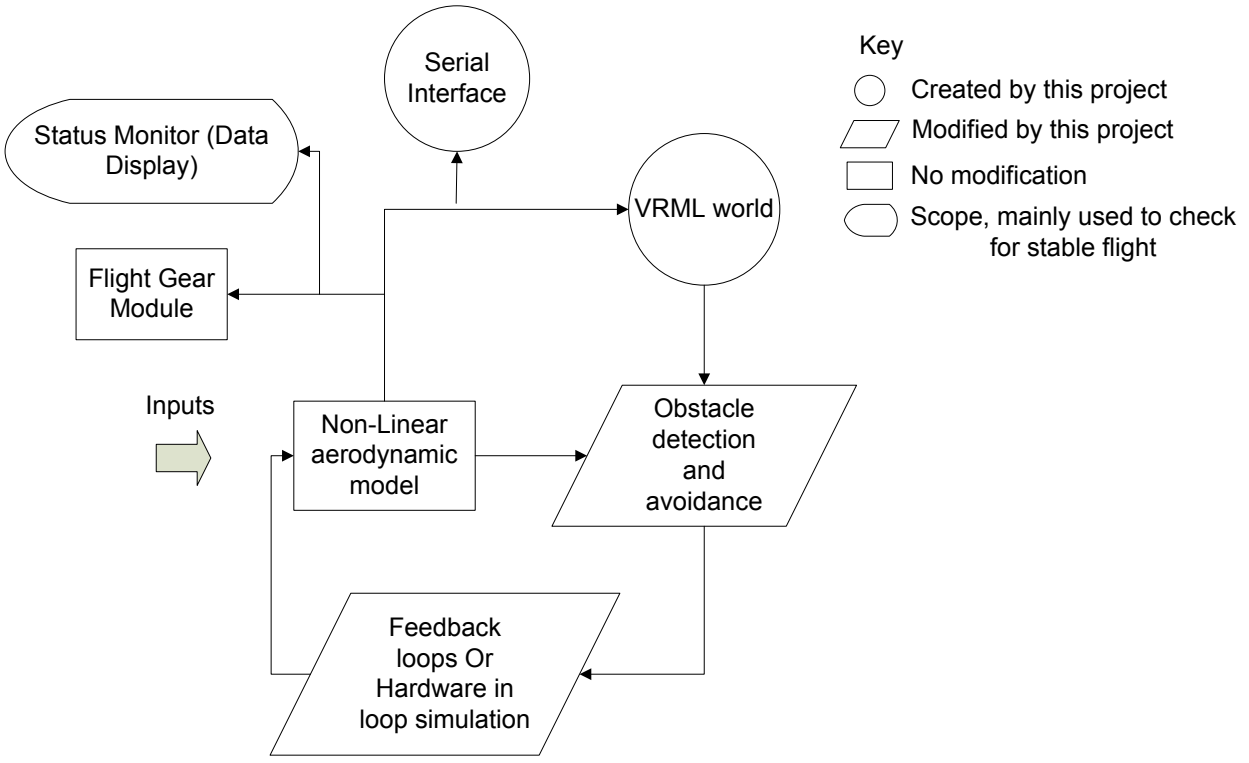


Figure 11: Architecture of the Simulation Framework

The simulation framework allows the testing of a single algorithmic or a set of algorithms. It should not be confused with the model that will be loaded onto an aircraft. The model that is loaded onto an aircraft would most likely be a C++ based code. Although, Matlab/Simulink is capable of producing the required C++ code, the simulation framework as a whole will never be converted to C++, only parts of it that are required will be converted and loaded on to the on-board computer. This thesis does not deal with implementation of real-time systems on the aircraft.

4.2.1 Aerosim Blockset (Non-Linear simulation)

While performing non-linear simulation of a UAV, one has to consider building the non-linear model of the particular UAV along with the control loops that achieve stability of the aircraft. Control loops are usually PID feedback controllers that are able to read data from sensors and control the control surfaces of the aircraft which allows a stable flight. Most of these control loops are provided by the Aerosim blockset. The Aerosim Blockset is equipped with two fully functional non-linear aircraft demonstrations along with a few examples of control loops.

The non-linear model is composed of eight sub-models namely Aerodynamics, General Aviation (GA) Propulsion System, Aircraft Inertia, Atmosphere, Total Acceleration, Total Moment, Equation of Motion and Earth Model. A brief description of each of these models is provided in this section. Due to these various internal models, the output of the whole system is not proportional to its input. Hence, the term non-linear model is used to define this model.

For this research project, the “Non-Linear Model” is considered as a black box where one provides the required input to acquire the required output values. The actual working of the model is beyond the scope of this research. Also the model is based on the Aerosonde UAV, therefore, a basic understanding of the block was acquired in order to convert the model for the GeoSurv or the ATB-10. However, these modifications were not implemented to the “Non-Linear Model” due to the lack of required parameters and time constraints. Thus, this section describes the model, and suggests the required modifications that need to be made in the future.

The inputs and the outputs of the nonlinear block that are used in this research are shown in Table 2. It should be noted that there are far more outputs than what is shown in this table. These additional outputs that are generated by the "Non-linear model" are not considered as they are not used by the simulation framework.

Table 2: Inputs and outputs of non-linear block

Inputs	**Outputs**
Controls = the 7x1 vector of aircraft controls [flap elevator aileron rudder throttle mixture ignition]. Aerodynamic controls are in radians, throttle is from 0 to 1, mixture is fraction air/fuel flow, ignition is 0 (engine off) or 1 (engine on).	**States** = the 15x1 aircraft state vector [Velocities(m/sec), Angular rates(rad), Quaternions, Position(latitude, longitude, altitude), Fuel mass, Engine speed (rad/s)]
Winds = the 3x1 vector of background wind speed components in navigation frame [North East Down]	**Sensors** = the 18x1 vector of sensor outputs [GPS Position, GPS Groundspeed, Accelerometers, Gyros, Air data, Magnetometer]
	VelW = the 3x1 vector of aircraft velocity in wind axes [Airspeed(m/s), sideslip(rad), angle of attack(rad)]
	Euler = the 3x1 vector of Euler angles (bank(rad), pitch(rad), yaw(rad))

A demo model provided by Simulink is shown in Figure 12. This particular model is designed to maintain a steady bank angle. The various output data shown in Table 2 can be monitored from the various display blocks as shown in the figure. Additionally, scopes can be attached before the simulation to generate graphs of the various data. For example, this model monitors the

bank angle. This model has a PI controller that has enabled a stable bank angle as shown in Figure 13. Thus, the model has stable lateral dynamics due to the inclusion of the PI controller.

It should be understood that this project focuses on usage of pre-existing blocks in order to setup the overall simulation framework. The working of the model is confirmed by monitoring the following parameters:

Altitude: Maintain the current altitude at all times.

Bank, Pitch and Yaw angles: A stabilised graph of these parameters would look like Figure 12.

Airspeed: Maintaining the pre-determined airspeed at all times.

By observing these parameters, the required modification to achieve stable flight can be determined. It should be noted that when obstacle avoidance is included, the aircraft will make the required maneuvers, this can be observed with these parameters. Additionally, the model would be made to return back to stable path after a successful maneuver. Sample graphs demonstrating the stabilised flight by the completed simulation framework is shown in Appendix E1.

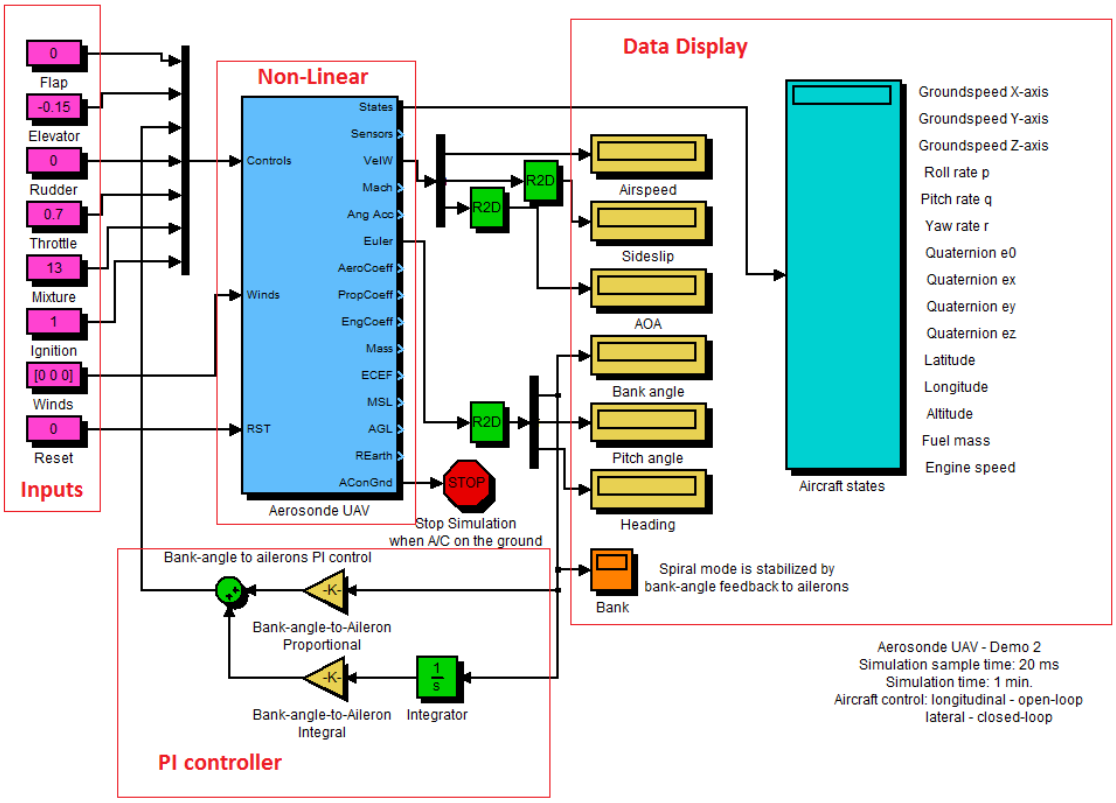


Figure 12: Demo Model

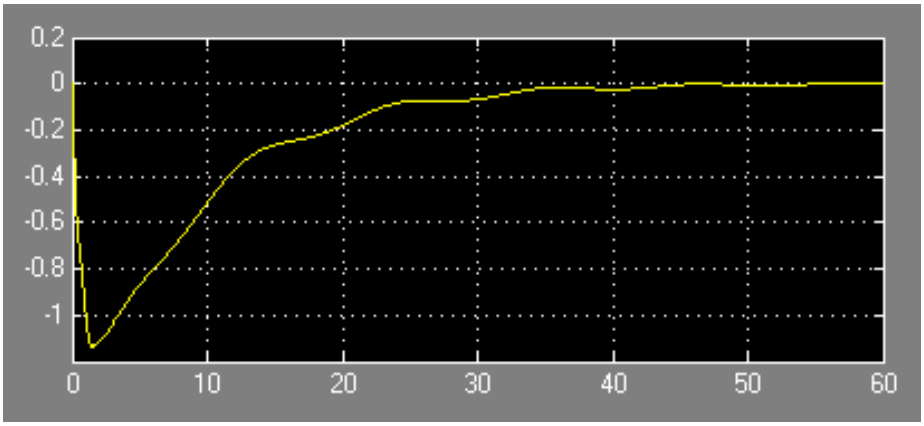


Figure 13: Bank angle vs. time

4.2.1.1 Equations of Motion

Building a basic aircraft simulation model starts with establishing the equations of motion for a six degree of freedom (6-DOF) model. The 6-DOF EOM block implements the six degrees of freedom: three translational modes and three rotational modes. The 6-DOF considers the forces and moments acting on the aircraft as these are the only factors responsible for its resulting motion. The forces and moments are caused by various factors such as aerodynamics, the propulsion system and the external environment. Each factor is detected with its own block.

The "*Equations Of Motion*" block consists of six other sub-blocks namely Forces, Moments, Kinematics, Navigation, Euler Angles, Body Inertial DCM. Refer Appendix E2 for the inter connectivity of these sub-blocks. Appendix E3 provides the inputs and outputs of the Equation of Motion Block.

Block Functions

Throughout the model three different co-ordinate systems are used for convenience. They are *local horizon co-ordinate system*, *body co-ordinate system* and *wind co-ordinate system* [Stevens, 2003].

The *local horizon coordinate system* is a Cartesian coordinate system. Its origin is located on pre-selected point of interest and its X_H, Y_H, Z_H axes align with the north, east and down direction respectively [Stevens, 2003]. Figure 14 demonstrates the *local horizon coordinate system.*

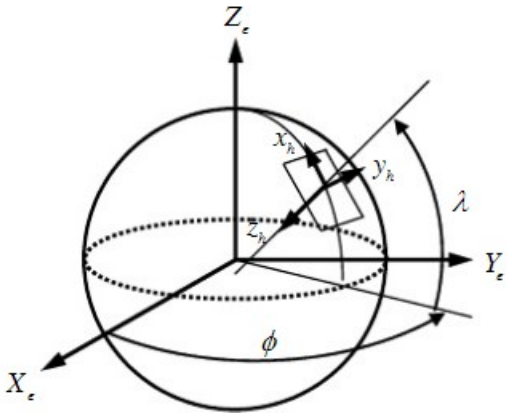


Figure 14: Local horizon coordinate system [Wibowo, 2007].

The *body coordinate system* is a Cartesian coordinate system which represents the aircraft body. Its origin is attached to the aircraft centre of gravity. The positive X_B axis lies along the symmetrical axis of the aircraft in the forward direction, its positive Y_B axis is perpendicular to the symmetrical axis of the aircraft to the right direction, and the positive Z_b is perpendicular to the X_b Y_b plane obeying the right-hand orientation [Stevens, 2003]. Figure 15 shows the *body coordinate system.*

Figure 15: Body coordinate system *[Wibowo, 2007].*

The *wind co-ordinate system* is also a Cartesian coordinate system. The origin is attached to the centre of gravity while its axes define the direction and the orientation of flight path. The positive X_W axis coincides to the aircraft velocity vector. The Z_W axis lies on the symmetrical plane of the aircraft, perpendicular to the X_w axis and positive downward. And the last, positive y_w axis is perpendicular to the $X_W Z_W$ plane following the right-hand orientation [Stevens, 2003].

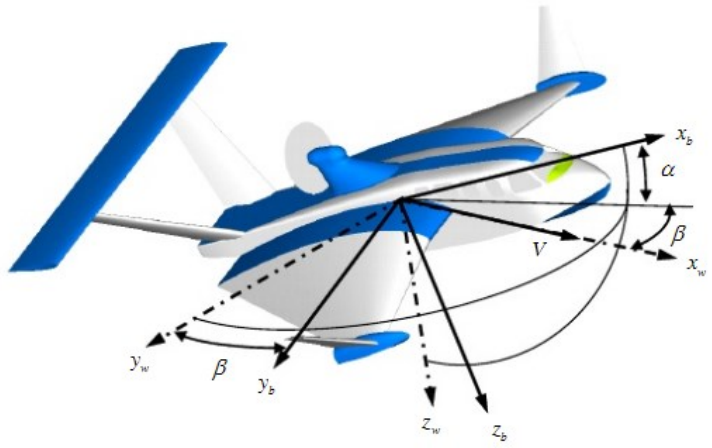


Figure 16: Wind co-ordinate system [Wibowo, 2007].

The *local horizon co-ordinate system* is considered as the referential frame as most aircraft fly in low altitude relative to the earth's surface. Therefore, transformations are usually done from the remaining co-ordinate systems to the *local horizon co-ordinate system* [Stevens, 2003].

Transformation of body axes to the local horizon frame

The transformation is carried out by using Euler angles (Pitch (Θ), Yaw (Ψ), Bank (ϕ)). The transformation of *local horizon coordinate system* to *body coordinate system* can be expressed as shown in the equation below [Jenie, 2006].

$$C_b^h = \begin{matrix} cos\theta cos\Psi & cos\theta sin\Psi & -sin\theta \\ sin\varphi sin\theta cos\Psi - \cos\varphi\sin\Psi & sin\varphi sin\theta cos\Psi + \cos\varphi\sin\Psi & cos\Psi\sin\varphi\cos\theta \\ cos\varphi sin\theta cos\Psi + \sin\varphi\sin\Psi & cos\varphi sin\theta sin\Psi - \sin\varphi\cos\Psi & cos\Psi\cos\varphi\cos\theta \end{matrix} \quad \textbf{(1)}$$

Transformation of Wind axes to the Local horizon frame

The transformation of *wind co-ordinate system* to *body coordinate system* can be expressed as shown in the equation below [Jenie, 2006].

$$C_b^w = \begin{matrix} cos \propto cos\beta & -cos \propto cos\beta & -sin\alpha \\ sin\beta & cos\beta & 0 \\ sin\alpha cos\beta & -sin\alpha sin\beta & cos\alpha \end{matrix} \quad \textbf{(2)}$$

Where α is the angle between X_B and X_W and β is the angle between Y_B and Y_W as shown in Figure 16.

The wind axis provides the aircraft's velocity vector. The velocity vector can in turn be used to determine the flight path and speed of the aircraft.

Directional Cosine Matrix (DCM)

The DCM provides the angle θ between any two vectors in 3 dimensional space. The above mentioned C_b^w C_b^h are considered as DCM as they provide the angle between two vectors in their respective co-ordinate systems [Wibowo, 2007].

Quaternions

Quaternions can be used to find DCM, and the Euler angles. The formulae required for calculation and used in simulation blocks are shown below: [Wibowo, 2007]

$$C_b^h = DCM = \begin{matrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{matrix} \quad \textbf{(3)}$$

$$\varphi = \arctan\frac{2(q_2q_3 - q_0q_1)}{q_0^2 + q_1^2 - q_2^2 - q_3^2} \quad \textbf{(4)}$$

$$\theta = \arcsin[-2(q_2q_3 - q_0q_1)] \quad \textbf{(5)}$$

$$\Psi = \arctan\frac{2(q_1q_2 - q_0q_3)}{q_0^2 + q_1^2 - q_2^2 - q_3^2} \quad \textbf{(6)}$$

$$q_0 = \pm cos\frac{\varphi}{2}cos\frac{\theta}{2}cos\frac{\Psi}{2} + sin\frac{\varphi}{2}sin\frac{\theta}{2}sin\frac{\Psi}{2} \quad \textbf{(7)}$$

$$q_1 = \pm sin\frac{\varphi}{2}cos\frac{\theta}{2}cos\frac{\Psi}{2} - cos\frac{\varphi}{2}sin\frac{\theta}{2}sin\frac{\Psi}{2} \quad \textbf{(8)}$$

$$q_2 = \pm cos\frac{\varphi}{2}sin\frac{\theta}{2}cos\frac{\Psi}{2} + sin\frac{\varphi}{2}cos\frac{\theta}{2}sin\frac{\Psi}{2} \quad \textbf{(9)}$$

$$q_3 = \pm cos\frac{\varphi}{2}cos\frac{\theta}{2}sin\frac{\Psi}{2} + sin\frac{\varphi}{2}sin\frac{\theta}{2}cos\frac{\Psi}{2} \quad \textbf{(10)}$$

Kinematic equation

Kinematic equation provides the relation between euler angles and angular velocity. All the equations shown below are based on the body coordinate system.

$$\omega_b = [P \quad Q \quad R]^T \quad \textbf{(11)}$$

$$\dot{\varphi} = P + Qsin\varphi tan\theta + Rcos\varphi tan\theta \quad \textbf{(12)}$$

$$\dot{\theta} = Qcos\varphi - Rsin\varphi \quad \textbf{(13)}$$

$$\dot{\Psi} = Q\frac{sin\varphi}{cos\theta} + R\frac{\cos\varphi}{\cos\theta} \quad \textbf{(14)}$$

$$\begin{bmatrix}\dot{\varphi}\\ \dot{\theta}\\ \dot{\Psi}\end{bmatrix}=\begin{bmatrix}1 & \sin\varphi\tan\theta & \cos\varphi\tan\theta\\ 0 & \cos\varphi & -\sin\varphi\\ 0 & \frac{\sin\varphi}{\cos\theta} & \frac{\cos\varphi}{\cos\theta}\end{bmatrix}\begin{bmatrix}P\\ Q\\ R\end{bmatrix} \quad \textbf{(15)}$$

Translational Motion and Angular motion

Based on Newton's second law, the net forces acting on the aircraft are summed together to determine the translation motion.

$$\sum \Delta F = \frac{d(mV_t)}{dt} = m\left(\frac{dV_b}{dt} + \omega_b \times V_b\right) + \dot{m}V_b \quad \textbf{(16)}$$

Where,

$$\sum \Delta F = [F_X \quad F_Y \quad F_Z] \quad \textbf{(17)}$$

$$V_b = [U \quad V \quad W]^T \quad \textbf{(18)}$$

$$\omega_b = [P \quad Q \quad R]^T \quad \textbf{(19)}$$

After decomposition, the three scalar force equations becomes

$$F_X = \dot{m}(U + QW - R) + \dot{mU} \quad \textbf{(20)}$$

$$F_Y = m(\dot{V} + RU - PW) + \dot{mV} \quad \textbf{(21)}$$

$$F_Z = m(\dot{W} + PV - QU) + \dot{mW} \quad \textbf{(22)}$$

Therefore, translational acceleration is found out to be

$$\dot{U} = \frac{F_X - \dot{m}U}{m} - QW + RV \quad \textbf{(23)}$$

$$\dot{V} = \frac{F_Y - \dot{m}V}{m} - RU + PW \quad \textbf{(24)}$$

$$\dot{W} = \frac{F_Z - \dot{m}W}{m} - PV + QU \quad \textbf{(25)}$$

The forces caused in the translational acceleration equations are due to aerodynamics, control surface propulsion, and earth's gravity. Hence, the following equation holds true

$$F_X = F_{A_X} + F_{C_X} + F_{P_X} + F_{G_X} \quad \textbf{(26)}$$

$$F_Y = F_{A_Y} + F_{C_Y} + F_{P_Y} + F_{G_Y} \quad \textbf{(27)}$$

$$F_Z = F_{A_Z} + F_{C_Z} + F_{P_Z} + F_{G_Z} \quad \textbf{(28)}$$

The net moments are also summed together in order to determine the net angular rates and angular accelerations. The net moment of an aircraft is the net moments experience by the aircraft as shown in the following equation.

$$\sum \Delta M = \frac{dH_I}{dt} = I\dot{\omega}_b + \omega_i \times (I\omega_b) + \dot{I}\omega_b \quad \textbf{(29)}$$

ω = Angular velocity, **I** = aircraft inertia

Angular moments are represented as,

$$\dot{P} = \frac{M_x I_{zz} + M_z J_{xz}}{\Delta} + \frac{J_{xy}\left(I_{xx} - I_{yy} I_{zz}\right) PQ}{\Delta} - \frac{[I_{zz}\left(I_{zz} - I_{yy}\right) + J_{xz}^2]\, QR}{\Delta} \quad \textbf{(30)}$$

$$\dot{Q} = \frac{M_y}{I_{yy}} + \frac{(I_{zz} + I_{xx})PR}{I_{yy}} \; \frac{J_{xz}\,(P^2 - R^2)J}{I_{yy}} \quad \textbf{(31)}$$

$$\dot{R} = \frac{M_x J_{xz} + M_z I_{xx}}{\Delta} + \frac{[I_{xx}\left(I_{xx} - I_{yy}\right) + J_{xz}^2]\, PQ}{\Delta} - \frac{J_{xz}\left(I_{xx} - I_{yy} + I_{zz}\right) QR}{\Delta} \quad \textbf{(32)}$$

Where, Δ is the determinate of I

And,

$$I = \begin{bmatrix} I_{xx} & -J_{yx} & -J_{zx} \\ -J_{xy} & -I_{yy} & -J_{zy} \\ -J_{xz} & -J_{yz} & I_{zz} \end{bmatrix} \quad \textbf{(33)}$$

Similar to transitional acceleration, rotational acceleration can be expressed in terms of aerodynamic, control surface and propulsion moments as follows

$$M_X = M_{A_X} + M_{C_X} + M_{P_X} \quad \textbf{(34)}$$

$$M_Y = M_{A_Y} + M_{C_Y} + M_{P_Y} \quad \textbf{(35)}$$

$$M_Z = M_{A_Z} + M_{C_Z} + M_{P_Z} \quad \textbf{(36)}$$

Forces, Moments, Kinematics, Navigation, Euler Angles and *Body-inertial DCM From Quaterions* blocks shown in appendix E2 work together to execute the equations mentioned above to provide the outputs mentioned in Appendix E3.

4.2.1.2 Aerodynamics

This block determines the aircraft's response to any deviation from a steady rectilinear symmetric flight. A set of six derivatives of aerodynamic coefficients in total are used to determine the aerodynamic forces and moments [Wibowo, 2007].

These are the *Lift coefficient, Drag coefficient, Side force coefficient, Pitch moment coefficient, Roll moment coefficient* and *Yaw moment coefficient.* The aerodynamic coefficients can be acquired from experimental results or any other alternate source such as Digital Datcom.

Currently the aerodynamic coefficients have been programmed using the demonstration model for the Aerosonde UAV.

Insides of the Aerodynamic block can be found in appendix E2. There are 10 blocks within the aerodynamic block that are dedicated to calculate the Lift Coefficient, Drag Coefficient, Side Force Coefficient, Pitch Moment Coefficient, Bank Moment Coefficient, Yaw Moment Coefficient, Aerodynamic forces and Aerodynamic Moment. The remaining two blocks Wind-axis Velocities and Dynamic pressure provides the required inputs in the right format after unit conversions. The two support blocks also provide additional constants that are required to perform the calculation. The internal calculation that occurs within the block is beyond the scope of this research. The inputs and the outputs of the block are listed below in Appendix E3.

4.2.1.3 General Aviation (GA) Propulsion System

This block contains models of piston engine and fixed pitch propeller models. These two models are based on the available engine and propeller in the Aerosonde UAV. The major contribution of this block is the force and moment provided by the propulsion system. Appendix E3 shows the outputs and inputs of the block.

4.2.1.4 Aircraft Inertia

The *Aircraft Inertia* block integrates the fuel flow rate and updates the inertia parameters of the airplane when provided with the fuel quantity data. Appendix E3 shows the inputs and outputs of the block. This block contains interpolation table. Using the data tables the inertia data is acquired via interpolation. The data in the interpolation table is dependent on the aircraft and the engine used. However, if the piston engine is to be replaced by an electric DC motor, this

block could be replaced with a *constant* block. This is because the aircraft inertia would no longer be affected by the change in fuel weight as a DC electric motor runs on battery.

4.2.1.5 Atmosphere

The Atmosphere block comprises of Standard atmosphere block, Background Wind block, Turbulence block and Wind shear block. Appendix E2 shows the configuration of these blocks. The Standard atmosphere block computes the static pressure, temperature, density and speed of sound at current MSL altitude, using the 1976 Standard Atmosphere look-up tables. The Background Wind block computes the background wind velocity and acceleration in body axes, given the wind velocities in navigation frame, and the navigation-to-body frame transformation matrix. The Turbulence block provides von Karman turbulence models. The Wind Shear block computes the body angular rates due to wind and turbulence. The inputs and outputs of these blocks are shown in Appendix E3.

4.2.1.6 Total Acceleration

The total acceleration block computes the total acceleration applied to the airframe due to aerodynamics and propulsion. The inputs and the outputs are provided in Appendix E3. The forces here are summed up and divided by the mass to provide the acceleration. Appendix E2 provides the block diagram of this block.

4.2.1.7 Total Moment

The Total Moment block computes the total moment applied to the airframe with respect to the current CG location. The forces that generate moments about the CG are the aerodynamics and the propulsion. The inputs and the outputs are provided in Appendix E3.

4.2.1.8 Earth Model

This block provides models for earth's gravity, atmosphere, magnetic field and wind. The model provides a WGS-84 geoid model of the Earth. It computes the mean-sea-level altitude relative to the WGS-84 geoid. It also provides the magnetic field components at current aircraft location. It computes the altitude above ground (AGL) and sets a flag if the AGL is negative (used for ground contact detection). It also computes the coordinates of the vehicle in the Earth-Centred Earth-Fixed frame, given the coordinates in geographic frame and local Earth radii. Appendix E3 shows the input and output of the block and Appendix E2 shows the sub-models being used with this model.

4.2.1.9 Required modification

This section describes the changes required to the non-linear model in order to incorporate the GeoSurv or the ATB-10. The required changes are not performed due to time constrains. As mentioned earlier the stability of the Aerosonde model is checked while creating the simulation framework. Two major modifications are required to convert the Aerosonde model to the ATB-10. The first modification was to replace the fuel engine with an electric motor and the second modification to replace the aerodynamic coefficients.

When adapting this model to an electric fixed wing aircraft the *GA Propulsion System* block must be modified. The ATB-10, unlike a gas powered aircraft (Aerosonde), the centre of gravity in an electric aircraft remains constant due to the absence of a fuel tank. Hence, the piston engine model requires be replaced by a brushless DC motor. For simplicity, a normal DC motor could be considered. A DC motor block was created using *Simscape* blockset. Figure 17 shows the layout of the DC block. A DC motor follows the following equations [Movellan, 2010]:

$$V = L\frac{dI}{dt} + RI + k_b\theta \quad \textbf{(37)}$$

$$M\ddot{\theta} = k_T I - v\dot{\theta} - \tau \quad \textbf{(38)}$$

Where,

V = voltage applied to the motor

I = current through the motor windings

R = the motor winding resistance

k_T = the motor's torque constant

M= the rotor's moment of inertia

L = the motor inductance

τ = torque applied to the rotor by an external load.

v = the motor's viscous friction constant

$\dot{\theta}$= the rotor's angular velocity

k_b = the motor's back electro magnetic force constant

These two equations are implemented in the model using Simscape library. This block equates the change in throttle value to changes in the DC motor voltage. Thus, changing the torque of the DC motor based on the voltage levels. The only parameter required for the non-linear model to function is the output torque of the motor. Though the model appeared to function successfully, this technique requires determination many of the parameters shown in the above equations. These parameters are usually not readily available. Hence, a motor is required to be characterised before implementing this model.

Alternately, one can build a DC motor model by acquiring the performance data such as voltage vs. torque or efficiency vs. current from the motor manufactures. This data is usually readily available. These values can be loaded into a “look up table” function block. The “look up table” function block interpolates the required data from the provided performance data. To determine torque, the following equations are used

$$n = \frac{P_{out}}{P_{in}} \quad \textbf{(39)}$$

$$P_{out} = n \times V \times I \quad \textbf{(40)}$$

$n = efficiency, V = voltage, I = current$ (all of which are known)

$\tau = \frac{P_{out}}{RPM}$ (RMP is generated by Simulink) **(41)**

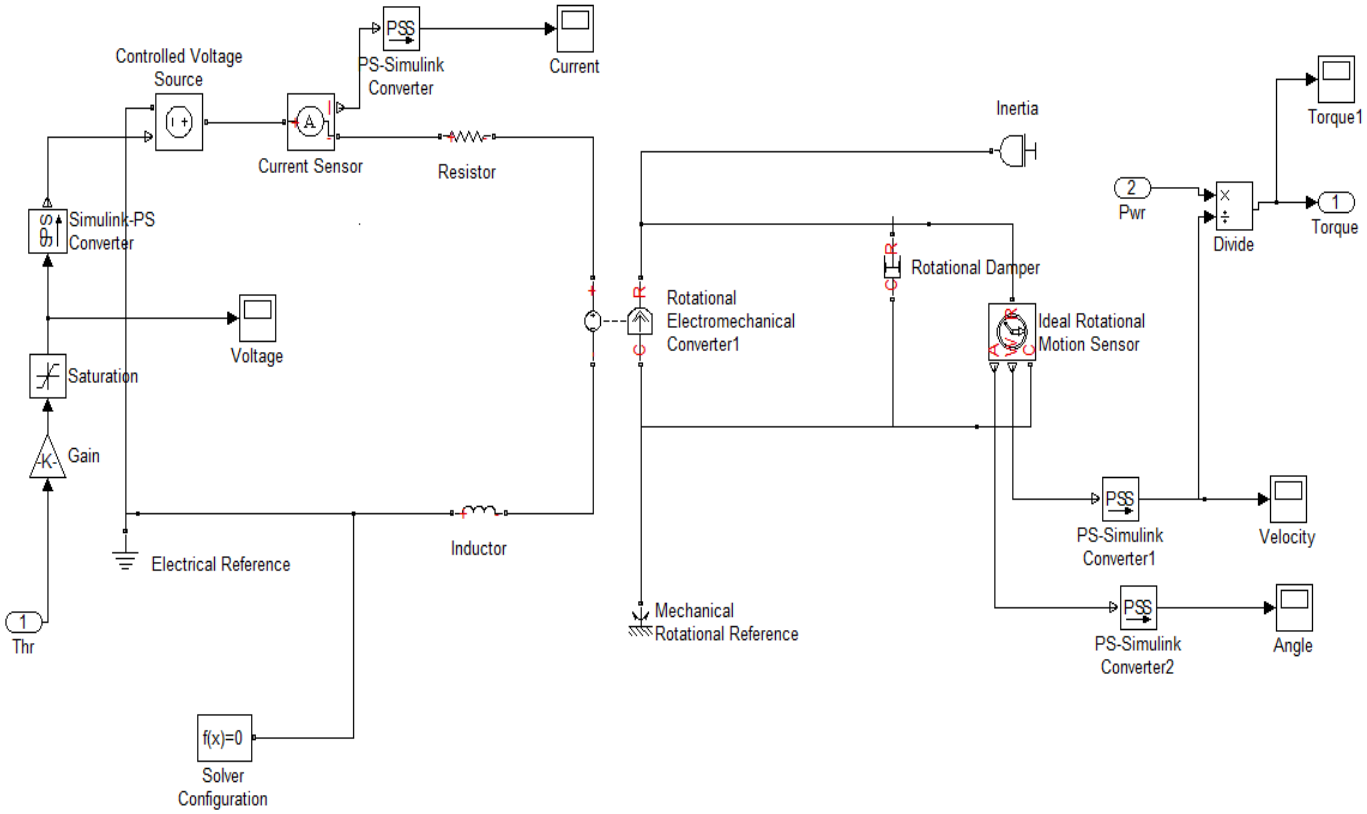


Figure 17: Proposed DC Model [Mathworks,2013]

Figure 18 shows a sample data specs from a Turnigy–C2830-1050 brushless DC motor.

Note that for GeoSurv motor modification is not required. However, the propeller coefficients need to be modified along with the aerodynamic constants, to convert the Aerosonde model to ATB-10 to the GeoSurv model

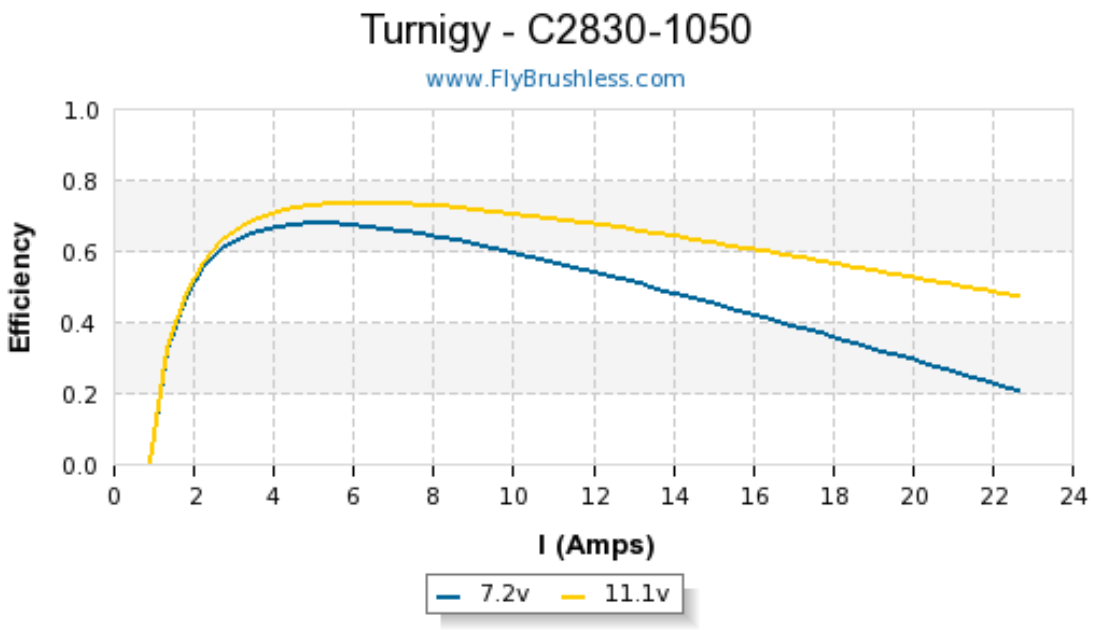


Figure 18: Data spec of NTP Prop Drive [Turnigy, 2013]

.4.2.2 Feedback Loops

This section describes the feedback loops mentioned in Figure 11 which is same as the *feedback controllers* block seen in Figure 19. This block contains proportional-integral-derivative controller (PID controller) and proportional-integral controller (PI controller). Figure 19 shows the inside of the *feedback controller* block. The PID controllers stabilise the aircraft by providing feedback between the following:

1. Throttle command from airspeed sensors
2. Elevator from pitch angle
3. Aileron angles from bank angles
4. Pitch angle from altitude sensors

5. Heading correction from course command

6. Pitch correction from path command

7. Banking angles from heading angles

These controllers are used to perform three separate functions. The first controller maintains the airspeed, the second set of controllers maintains the pitch angle, and the third set of controllers maintains the bank angle.

The 1st PID controller is in loop with the non-linear model as shown in Figure 19. The PID controller measures the airspeed from the non-linear model and compares with the required airspeed (arbitrary value, 23m/s), to provide the modified airspeed required to maintain the aircraft at 23m/s.

The second set controller maintains the pitch angle by sending elevator deflection angle to the non-linear model. Unlike the 1st controller, this system has obstacle avoidance block along with the non-linear block within the feedback loop.

The third set of controller maintains the bank angle by sending ailerons deflection angle to the non-linear model. Similar to the 2nd set of controllers, the system has both obstacle avoidance and non-liner block within the feedback loop.

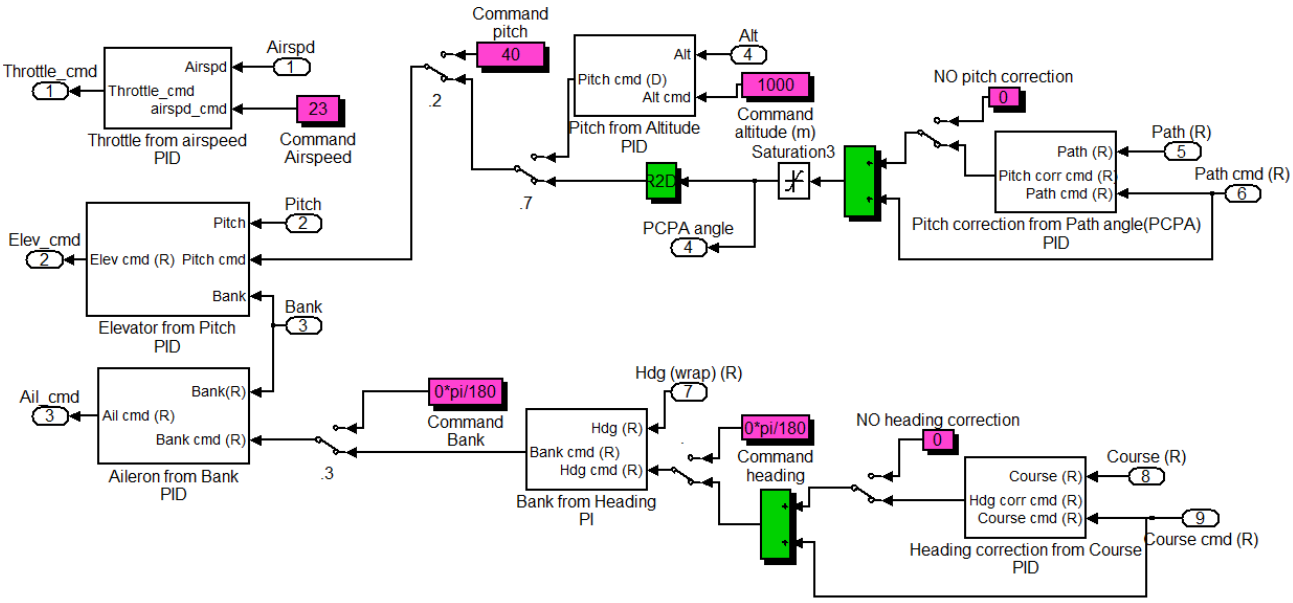


Figure 19: Inside of feedback controllers block

Figure 20 shows a generic PID controller. A PI controller is same as the PID controller when the derivative gain (Ki) is set to zero.

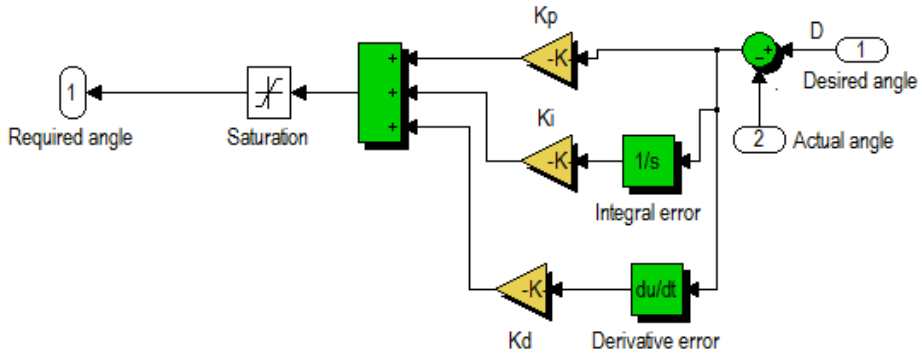


Figure 20: Building a PID controller

A PID controller calculates the error between the input signal and a required signal by comparing both the signal with a *sum* block. Once the error is known, the new signal is then passed through proportional (P), integral (I) and derivative (D) values. Based on a current rate

of change, these values represent the present error, past accumulated error and the future error, respectively. Finally, the weighted sum of these three actions is used to determine throttle command, elevator angle, aileron angle, pitch angle, heading and pitch correction angles.

Heuristic evaluation was performed in order to attain the P, I and D values. That is the gains for P, I and D were manually determined by trial and error. Initially, I and D gains are set to zero. The P gain (Kp) was then increased until the output of the controller oscillated. This value was adjusted to achieve the approximate required output. Here, the required output was a stable flight, therefore, airspeed, aircraft altitude, bank angle, pitch angle and yaw angle were monitored during each attempt to run the simulation with a new gain value. This step was followed by adjusting I gain (Ki), which brings the response of the PID controller closer to the required output response. Lastly, if required the rate at which the controller responses to an input to provide a required output can be adjusted by using D gain (Kd) [Zhong, 2006].

The determined gains to achieve a stable flight is shown in Table 3

Table 3: PID gains

Controller	**Kp**	**Ki**	**Kd**
Throttle command from airspeed sensors	0.02	0.5	0
Elevator from pitch angle	0.34	0.12	0.06
Aileron angles from bank angles	0.02	0.2	0.3
Pitch angle from altitude sensors	0.17	0.5	3
Heading correction from course command	0.08	0.05	0
Pitch correction from path command	0.03	1	0
Bank from heading	1	0	0.1

It should be noted here that the controllers from 1 to 4 were created to replicate the feedback controllers available in an autopilot. With integration of hardware-in-the-loop simulation of MicroPilot, these feedback controllers will be removed. Since the development of hardware-in-the-loop block is not completed yet, this research will be using the available PID controllers. Selection 4.2.4 explains the availability of the hardware-in-the-loop block and its integration into the simulation framework.

As mentioned earlier the simulation frame work does run successfully without any errors, furthermore a stable flight is also observed. Appendix E1 shows the stability graphs.

4.2.3 Obstacle avoidance and detection

The *Obstacle avoidance and detection* block consist of three sub-blocks namely, *Obstacle detection, Obstacle avoidance, Pitch vs. Bank Decision.* Each of these sub-blocks are explained in detail below. Figure 21 shows the arrangement of the mentioned sub-components in the *Obstacle avoidance and detection* block.

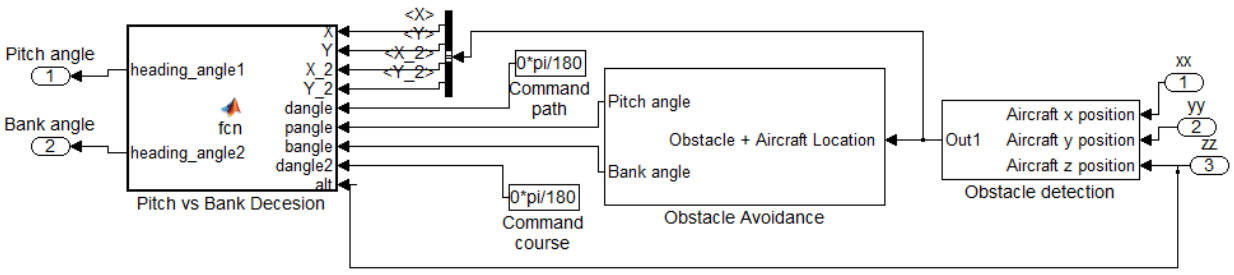


Figure 21: Inside the Obstacle avoidance and detection block

4.2.3.1 Obstacle detection

Ideally this block is responsible for image processing. Once the video data from cameras mounted on the aircraft is processed, this block is responsible for providing the X, Y and Z co-ordinates of a potential obstacle in flat Earth coordinate system, which assumes the Z-axis is downward positive and the X, Y co-ordinates are derived from Latitude and longitude values. However, this algorithm for obstacle detection is beyond the scope of this project and is being dealt by other students within the GeoSurv project [Owlia, 2013]. Once an algorithm is created it can be programmed into the *Obstacle detection* block.

Hence, for the time being a "pseudo" obstacle was provided within the block by providing the following user specified data.

X, Y and Z co-ordinates

The obstacle boundary is determined by a set of 6 co-ordinates in its respective plane as shown in Figure 22. These co-ordinates are required by the obstacle avoidance block which is explained in the next section.

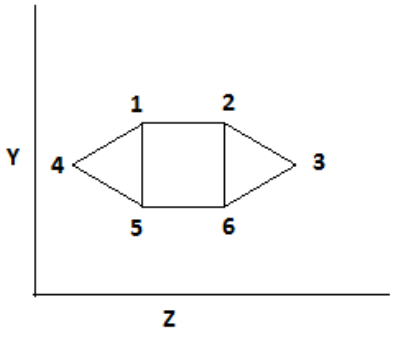


Figure 22: YZ plane

Y, Z co-ordinates of the YZ plane of the obstacle provides information on "*pitch-up*" maneuver and X, Z co-ordinates of the XZ plane of the obstacle provides information on *"fly-around"* maneuver. The co-ordinates are acquired with the help of "*LLA to Flat Earth"* block, which has the input of latitude, longitude and altitude (LLA) and outputs the required X, Y, Z flat earth co-ordinates.

Aircraft Location

The location of the aircraft is passed through the block as one can expect this information to be used in the obstacle detection algorithm to extrapolate X co-ordinates based on stereoscopic disparity. Also, the aircraft location is required by the *VRML* block which will be placed within the *obstacle detection* block. The *VRML* block is explained in detail in chapter 5.

Goal (sink) strength and position

Sink strength and position are parameters introduced in potential flow theory; Owlia's research explains the use of these parameters in the "*panel method*", which is essentially the method used in modeling a steady flow path around an obstacle. The working theory of this code is beyond the scope of this research. However, the sink strength and position are dependent on obstacle co-ordinate. Thus by providing the obstacle co-ordinates, one is cable of running the model without determining these parameters. The model is currently programed to automatically determine the sink parameters with the obstacle co-ordinates [Owlia, 2013].

Figure 23 shows the inside of the *obstacle detection* block. As mentioned earlier, instead of an algorithm detecting the obstacle and sending the required parameters to the obstacle

avoidance block, 'pseudo' values are sent. Here, X, Y represents Z and Y co-ordinates of the obstacle in the YZ plane (*"pitch-up"* maneuver). X_2, Y_2 represents X, Y co-ordinates of the obstacle in the YX plane (*"fly-around"* maneuver). xx, yy, zz represents the X, Y, Z position of the aircraft in flat earth co-ordinates. *xg ,yg ,xg_2, yg_2* represents the goal position *and lambda_g, lambda_g1* represents goal strength of its respective goals

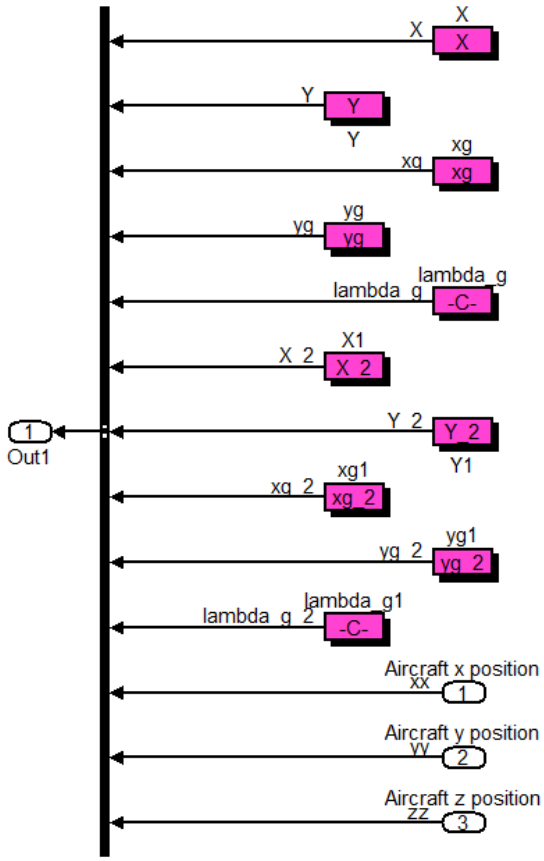


Figure 23:Inside Obstacle detection block

4.2.3.2 Obstacle avoidance

The obstacle avoidance algorithm is obtained from the prior research performed within the GeoSurv Project. Owlia's research developed an algorithm to provide a flight path to avoid obstacles. The algorithm was able to execute the new flight path by providing heading angles.

The algorithm used two dimensional potential flow in order to avoid obstacles. According this theory, when a uniform fluid flow encounters an obstacle, fluid particles deviate from their original course to avoid it. This concept can be used for obstacle avoidance, by commanding the aircraft in a way that it would follow the trajectory of a fluid particle (pathline) encountering an obstacle. Since the fluid particles do not flow through obstacles, the aircraft is able to reach its goal without collision. Figure 24 demonstrates the pathline followed by the fluid.

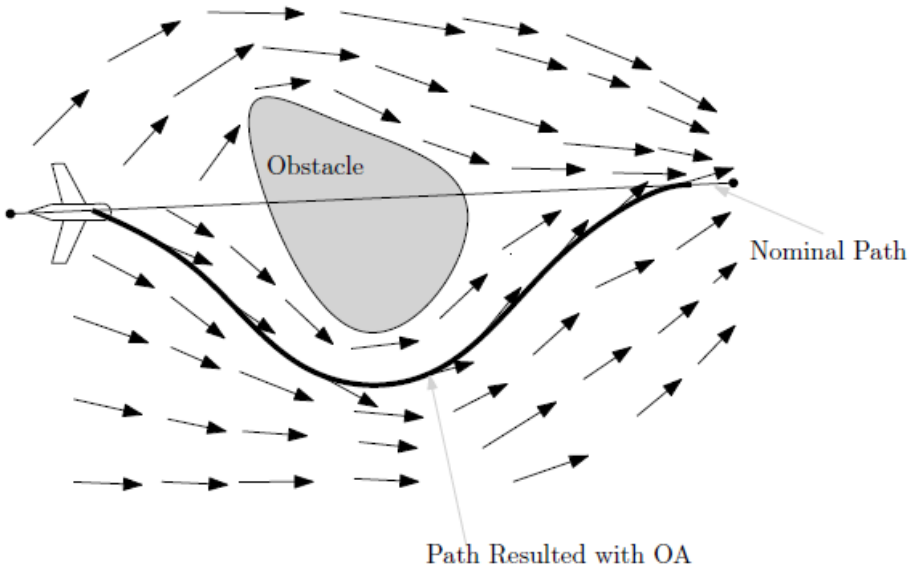


Figure 24: Obstacle avoidance vector field [Owlia, 2013]

Owlia's research derives the potential function of three 2D flow primitives that are restricted to a 2D plane: uniform flow, sinks and sources, shown in Figure 25. A source is a flow primitive that emits fluid particles radially and a sink absorbs flow radially. A combination of these basic primitives can be used to represent more complex flows.

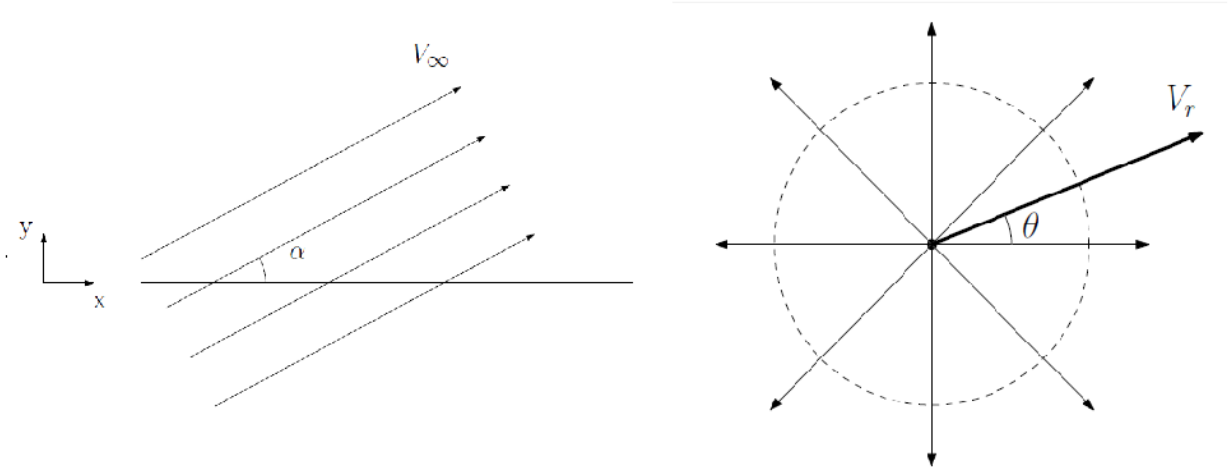


Figure 25: uniform flow (left), sinks and sources (right) [Owlia, 2013]

The potential functions of the primitives are shown below.

$$\Phi_u = (\mathrm{V}_\infty \cos \propto)x + (V_\infty \sin \propto)y \quad \textbf{(42)}$$

$$\Phi_S = \frac{\Lambda}{2\pi} \ln r \quad \text{, } \Lambda = \text{source/sink strength (m}^2\text{/s)} \quad \textbf{(43)}$$

Modeling an obstacle with the flow primitives uses the concept of panels. Where, the panels are created by an infinite number of infinitesimally small sources and sinks. **Λ** is chosen such that the velocity normal to each panel is forced to zero on the panel itself. Hence, there is no flow across the panel and thus avoid the obstacle. The following equation shows the potential function of a single panel, where, λ is the source strength per unit of depth and unit of length of each panel (m/s)

$$\mathrm{d}\Phi = \frac{\lambda_j}{2\pi} \ln(r)\, ds \quad \textbf{(44)}$$

Figure 26 shows a closed obstacle with n panels is immersed in a uniform flow.

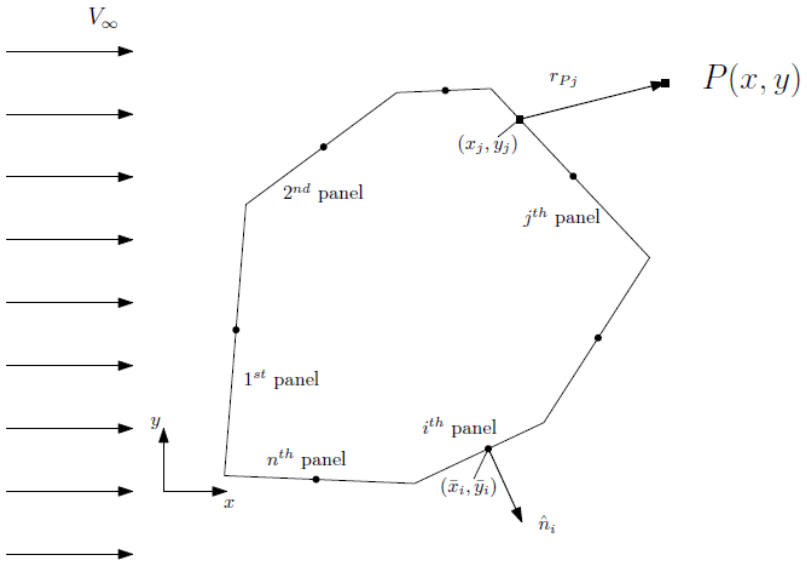


Figure 26: Modeling obstacle with panel [Owlia, 2013]

The potential flow of the ith panel of length S is shown as follows

$$\Phi_j = \frac{\lambda_j}{2\pi} \int_0^{S_j} \ln rp_j \, ds_j \qquad \textbf{(45)}$$

Where ,

$$rp_j = \sqrt{(x - x_j)^2 + (y - y_j)^2} \qquad \textbf{(46)}$$

The following equation holds true for the potential due to all the panel fields or also known as potential field of the obstacle.

$$\Phi_o\,(\mathrm{P}) = \sum_{j=1}^{n} \frac{\lambda_j}{2\pi} \int_0^{S_j} \ln rp_j \, ds_j \qquad \textbf{(47)}$$

The potential field of the obstacle is used to generate the velocity vector, and the velocity vector at each point can be used as a course/path angle command to direct the aircraft away

from the obstacle. The above relations were further developed by Owlia's research up to an extent of creating the working algorithm within Simulink. The algorithm also considers the acceptable range of maneuvers that should be performed by the GeoSurv. Therefore, the algorithm does not produce velocity vector that could potentially stall the aircraft [Owlia, 2013].

The detailed workings of the algorithm are beyond the scope of this project. However, sufficient understanding from the above mentioned theory allows performing a *"pitch-up"* maneuver and *"fly-around"* maneuver. Hence, the algorithm was used as an existing function, where one could input data such as location of the co-ordinates and the function would output a heading angle which allows the aircraft to follow the pathline.

It was decided that the aircraft would either perform a "*fly above*" the obstacle or *"fly-around"* the obstacle. Since the algorithm created by Oliwa's research is limited to a 2D plane, it can only perform one possible maneuver. Therefore, in this research, in order to execute the two required maneuvers two 2D planes were considered. The first plane is the YZ plane for *"pitch-up"* maneuver and the second plane is the YX plane for *"fly-around"* maneuver. X, Y and Z data of the obstacle is acquired from the obstacle detection block, ideally with the help of stereoscopic image processing. Figure 27 shows the occurrence of the model containing Oliwa's algorithm twice, "*Obstacle avoidance (Pitch)"* and "*Obstacle avoidance (Bank)".* The double occurrence of the algorithm, implicates the consideration two planes (YZ, YX). The modified Simulink model is now capable of providing both pitch angle heading angles.

Along with the XYZ data goal strength and location data prevents execution of pitching the aircraft down and also helps determine to head to the right or left while performing the *"fly-*

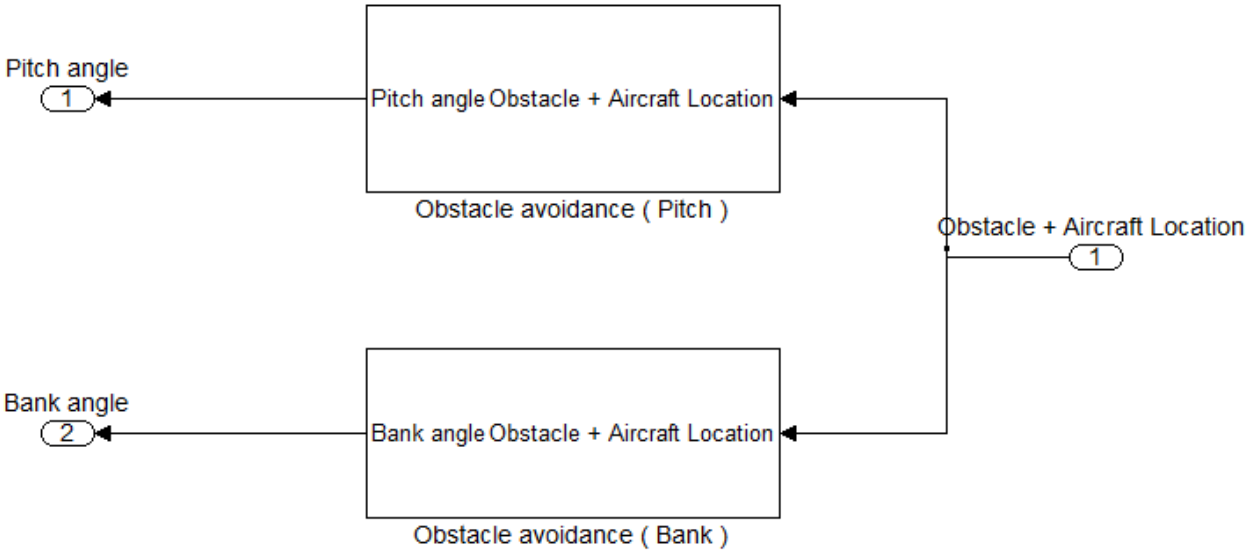


Figure 27: Sub-components of obstacle avoidance block

Both the pitch model and bank model are logically implementing the same process; the inside of a block is shown in Figure 28. Owlia's algorithm is loaded as a Matlab function. Additionally there are switch blocks which saturate the bank or pitch angle and hence prevents execution of any drastic or excessive flight altitude angles. The angle limit is set to 30^{o} and can be changed according to user requirements or aircraft handling limitations.

Now that the modified model can perform two possible maneuvers, it is important to make sure that model as a whole executes only one selected maneuver for a given obstacle. The selection between pitch and bank was performed by the "*Pitch vs. Bank Decision*" block as shown in Figure 21. The working of this block is described in the next section.

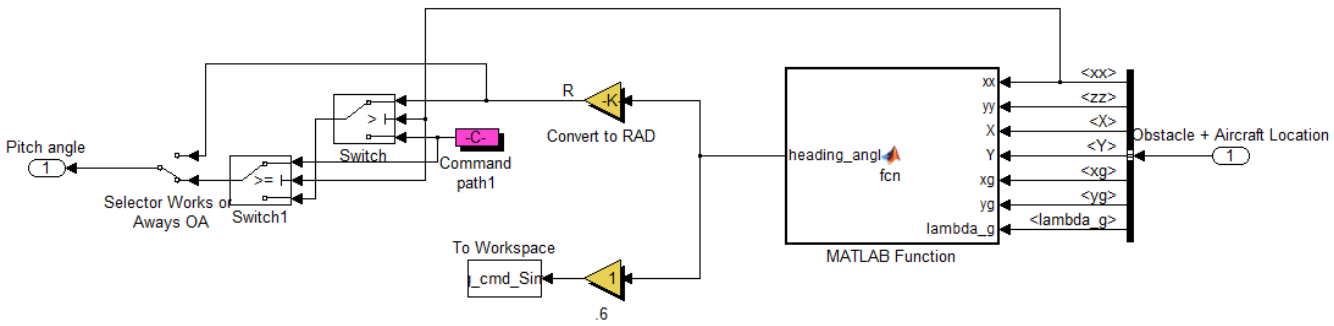


Figure 28: Implementation of OA algorithm

In this section, this thesis contributes to adding a second maneuver by adding a second instance of the previously developed algorithm. Apart from the above mentioned Simulink modifications, the variables used by the duplicate version of the code in matlab were modified. This change in variables allowed successfully execution of both the code simultaneously during simulation.

4.2.3.3 Pitch vs. Bank Decision

Once the pitch angle and bank angle is acquired it is passed into the *Pitch vs. Bank Decision* block. This block selects weather the aircraft should perform a pitch up maneuver or a bank maneuver. This block calculates the distance required to travel by both the maneuvers and compares to find the maneuver with the shortest distance.

Assuming the aircraft travels in same speed performing both the maneuvers as the throttle is controlled by the autopilot; it would imply the aircraft avoids the obstacle quickly by performing the maneuver with the shortest distance. Hence, the maneuver with the shortest distance was decided to be the appropriate path to be executed. However, it should be noted that this method does not consider the amount of fuel required to perform a maneuver. The

aircraft might require more fuel to perform one maneuver over the other. The selection of the maneuver could be based on fuel conception. An assumption is made that the aircraft holds ample fuel and fuel consumption is not parameter that is accounted while selecting the maneuver. Thus, the time taken to complete the maneuver influences the selection of the maneuver. The block diagram in Figure 29 shows the logic behind the maneuver decision. The complete working code is shown in appendix D1

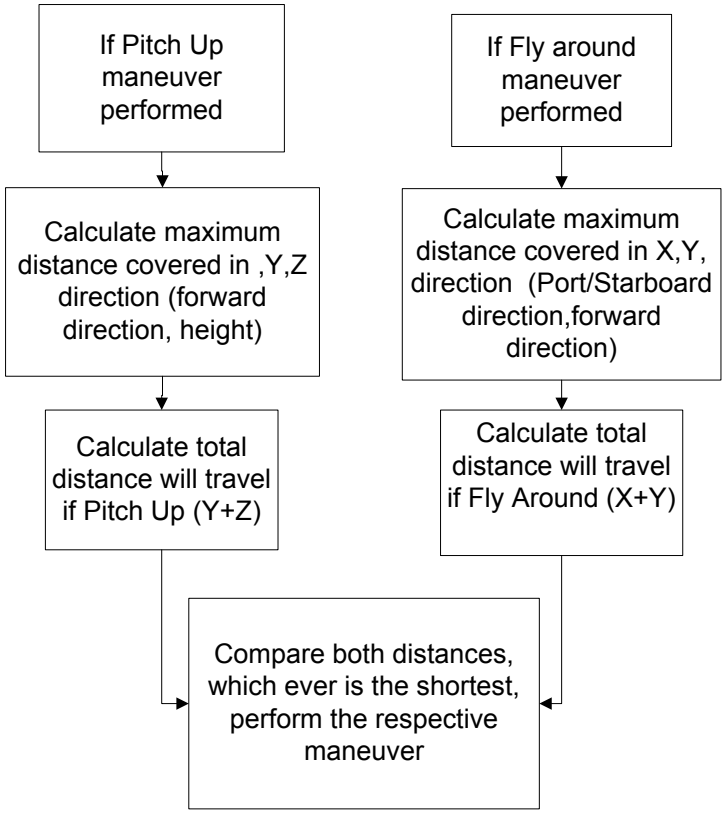


Figure 29: Pitch vs. Bank Decision

4.2.4 Hardware-in-the-loop Simulation

The hardware-in-the-loop block is in its early stages of development. The GeoSurv Project currently has other MASc students modeling the sensors such as the altimeter, accelerometer and pressure. The transfer functions being developed can be used to estimate the reaction of

the PID controllers embedded within the MicroPilot. Once the block is developed, the current *feedback loops* block can be replaced with the hardware-in-the-loop block. However, one can expect more inputs and outputs to the hardware-in-the-loop block when compared to the existing *feedback loops* block. For example, the values that need to be sent to the hardware sensors are additional inputs to the block, and would have the aileron control signal as output of the block.

4.2.5 FlightGear

FlightGear is an open source software that provides multi-platform flight simulation. The F*lightGear* block links the Simulink environment to the FlightGear environment. The F*lightGear* block is located in the *navigation and FlightGear* block. This block provides position and attitude information to the FlightGear Flight Simulator 0.9.8 for visual output of the aircraft behaviour. The FlightGear block requires the aircraft position, euler angles and the air speed to successfully observe the aircraft dynamics in FlightGear simulation environment. Supplementary information can also be provided to the block for a smoother performance; however, these are not compulsory data that are required for *FlightGear* to function. Table 4 lists all possible inputs for this block.

Table 4: FlightGear Parameters

Inputs	Description
Position	The 3x1 vector of geographic position in [rad rad m]
Euler	The 3x1 vector of Euler angles in radians
Airspeed	Scalar number representing airspeed, in m/s
VClimb	The climb rate, in m/s
AGL	The altitude above ground, in meter
VelNED	The 3x1 vector of groundspeed, in m/s
Rates	The 3x1 vector of angular rates, in rad/s

Acc	The 3x1 vector of body accelerations, in m/s^2
Omega	The engine speed, in rad/s
FuelFlow	The instantaneous fuel flow, in grams per hour
OutofFuel	Set it to 1 to disable engine noise in FlightGear, otherwise set it to 0
Controls	The 7x1 vector of aircraft controls [flap elevator aileron rudder throttle mixture ignition]; the first 4 aerodynamic controls should be normalized to [-1, 1].

5.0 3D Virtual Environment for UAV Simulations

As mentioned earlier, currently other students are working with stereoscopic vision systems for the GeoSurv II [Verbickas, 2012]. Since the algorithm is still under development, the working of the algorithm is not mentioned in this thesis. However, one of the main problems faced by any obstacle detection system is the lack of suitable video data to test the system/algorithm. Conventionally this data is gathered during the test stages from cameras that are mounted on to the wings of the UAV. These cameras record the required video during flight. The recorded video is later used to test the functionality of the image processing algorithm.

Substantial resources can be spent while performing flight tests to gather the required video data. Therefore, the primary objective of the work described in this chapter was to extract video data from a 3D virtual environment; a 3D environment which would be created based on the mission requirements of the UAV. Since Simulink is being used in all other aspect of the autonomous system, in order to maintain uniformity, the virtual video data is to be generated in the Simulink environment. Therefore, VRML (Virtual Reality Modeling Language) was used in creating the 3D environment as VRML is compatible with Simulink. Section 5.1 discusses the implementation of VRML within the available Simulink model.

5.1 Implementation of VRML

Appendix F describes VRML programing. Once the VRML file has been created, it can be used in conjunction with Simulink to provide a video output for the obstacle detection block. This section describes the various environments created for testing purposes. This section also describes the proposed Simulink architecture of the obstacle detection and avoidance with respect to the entire virtual framework created in chapter 4.

Additionally, this section explains the validity of the proposed system with the help of experimental results i.e., In order to check the efficiency of the VRML environment, the results obtained by a proven system needs to be compared with that of the proposed system. The proven system decided upon was using the XBOX-360 Kinect [Webster, 2013]. The goal of the experiment is to capture the stereoscopic data produced by the Kinect and compare it with the stereoscopic data produced by the Simulink model using a VRML environment. It should be noted that the Kinect does not work on stereoscopic principles but rather uses a camera and structured light range sensors to map its surroundings. Section 5.2.1 describes the working of the Kinect in detail.

Two VRML were created namely kinectreplica.vrml and uav.vrml. The kinectreplica is a replica of the actual environment used for exterminations performed with the Kinect.

The uav.vrml model is meant to be implemented in the UAV simulation framework in Simulink. The immediate two sections below describe both these models.

5.1.1 VRML environment for testing 3D environment

Since the Kinect was the most precise, affordable and fully functional 3D sensor for indoor use, it was considered to be the best alternative to any two camera system. Also predefined Kinect blocksets are available in Simulink for simulation development. After performing a literature review it was found out that almost all the image processing researches use the Kinect for gathering data, for example a research done by Wham deals with tracking the movement of the human body with a Kinect [Wham, 2012], another study by Stanley deals with examining the ability to predict user attention using features of body posture and head pose [Stanley, 2013], another study by Sanabria deals with 3D modeling of objects by importing stereoscopic data produced by the Kinect into AUTOCAD [Sanabria, 2013]. All these research work demonstrates the high quality stereoscopic data obtained by the Kinect. Hence, one could consider the data provided by the Kinect as the best possible result one could expect from a two camera system for the price range. The research conducted by Verbickas also uses the Kinect to acquire stereoscopic data [Verbickas, 2013]. Thus it was decide that Kinect would be used to acquire stereoscopic data.

This first requirement to perform experiments with the Kinect was to decide upon a location where the experiment would be conducted. In order to avoid creating complex features within the VRML environment, the selected location was made sure to not have any complex feature. Therefore, the corridor of the Canal Building at Carleton University was selected. An obstacle was placed in the corridor as shown in Figure 30.

Figure 30: Selected testing environment

The dimensions of the room and the location of the obstacle are show in Figure 31.

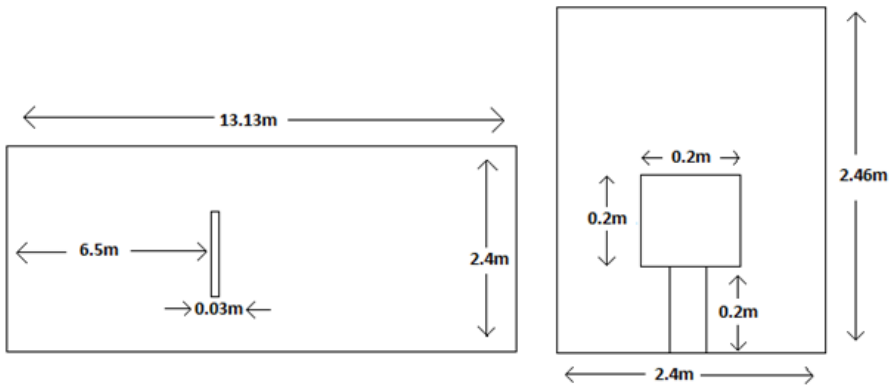


Figure 31: Room and Obstacle Dimension

The Kinect was placed on a chair, which was placed in front of the obstacle. The chair was rolled towards the obstacle to acquire the required moving stereoscopic image. Figure 32 shows the setup of the Kinect and laptop on the chair. Section 5.1.1.1 describes the working of the Kinect and the description of data acquired from this experiment. Section 5.1.3.2 describes the means of importing required data from the Kinect to Simulink.

Figure 32: Kinect Setup

Figure 33 shows the virtual environment created based on the actual environment. The environment was created by programming in VRML language in VrmlPad. The code can be found in Appendix B7. The developed VRML world was opened in Simulink, which is shown in Figure 33. It should be noted here that the dimensions of the room, door and obstacle are all matched to real world dimensions.

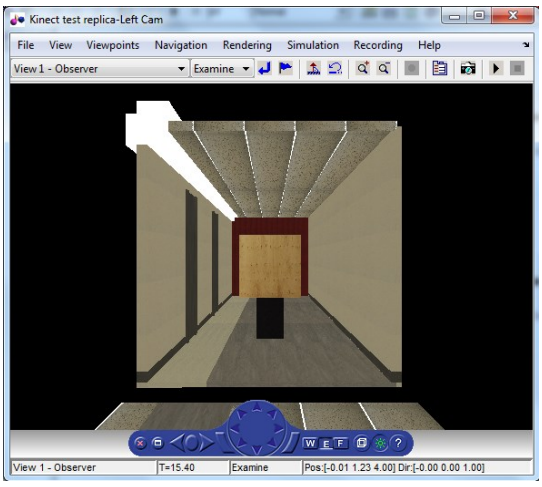


Figure 33: 3D Virtual Environment to test VRML

5.1.2.1 Functions of the Kinect

The Kinect has a normal RGB camera, an infrared camera and depth camera. The RGB camera captures images around normal video speeds of 30 frames per second, and records at a 640x480 resolution. The Kinect emits infrared light which reflects off objects in its field of view (approximately 4m). The depth camera views the scattered infrared signal to build a 3D map of the room. The distortion parameters of the structured light field are used to provide depth in XYZ coordinates [Microsoft, 2013].

Combination of both the results from the RGB camera and the infrared camera can then be used effectively for various purposes. It should be noted that unlike in a regular stereoscopic camera where two RGB cameras are used, the Kinect uses only one RGB camera. Figure 34 shows the scattered light being detected by the sensor. The sent and received data is compared to acquire the depth map, which is the stereoscopic data that is required by this project.

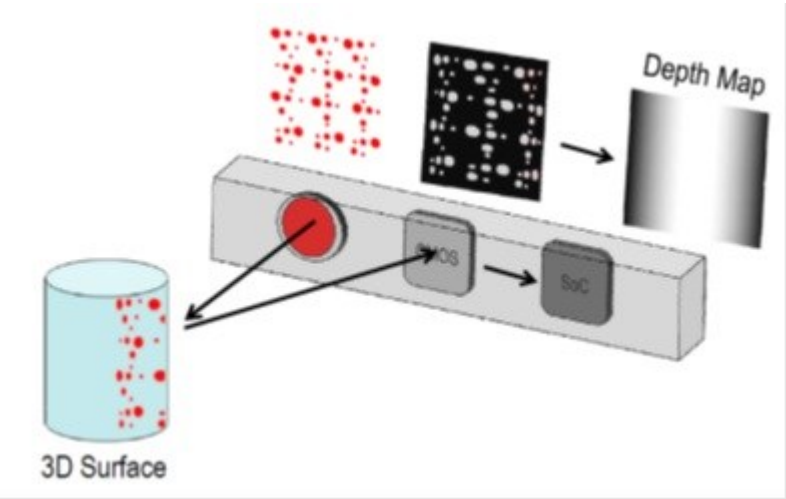


Figure 34: Working of the Kinect [Kinect 2, 2013]

Infrared cameras can be used only indoors as external light sources will interfere with the infrared camera sensors and hence provide incorrect results. Thus Kinect will not be used on fixed wing UAVs as it does not provide the required range and also will not work well outdoors due to the interference from the sunlight.

However, the goal of the experiment is to compare with the best possible results available from a "stereoscopic system" with that of VRML simulation. Besides the mentioned drawbacks, the Kinect provides the most accurate results indoors within 4m range when compared to results gained from outdoors [Ryu, 2011]. As mentioned earlier the Kinect device is preferred by many researches. For example, researchers at the University of Surrey are preparing for a new mission that could dock tiny satellites together in space with the help of Kinect sensors [Webster, 2013]. Researchers at MIT have built a prototype wearable mapping system that can wirelessly transfer data in real-time which works with the help of the Kinect [Hardesty, 2013]. The US Air Force is developing a Kinect therapy system which is aimed to provide affordable veterans physical theory [Peck, 2013].Countless more researches related to image processing are performed but none implement the Kinect on a UAV for reasons mentioned before. The high quality stereoscopic data, the online software support and the affordable price has persuaded researchers to select Kinect for their research. Hence, the Kinect was selected as the device to be used for comparing the VRML simulation results.

5.1.2 VRML environment for the UAV

For the UAV environment, an open space was created with random obstacles as shown in Figure 35. The cameras viewing the environment were set to have 6 degrees of freedom (DOF).

Thus the camera moves in all possible directions (6-DOF), hence representing a 6-DOF video display from a video camera attached to a UAV.

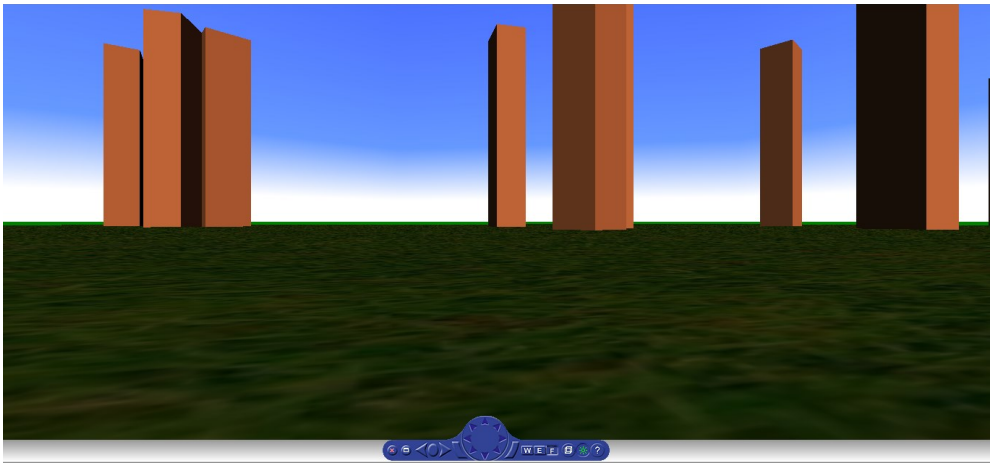


Figure 35: 3D Environment for UAV

The VRML code used to develop this environment can be found in Appendix B8.

5.1.3 Block diagram/schematic for VRML block

This section explains the three different Simulink models in different scenarios. The first model implements the kinectreplica.vrml in Simulink. The second model helps in recording information provided by the Kinect hardware. Lastly, the third model integrates VRML environment in the UAV Simulink framework.

5.1.3.1 Model 1: Simulink Model used for testing the VRML environment

This section describes the model used for testing the VRML environment. The blocks used in this model are mainly from the *computer vision* blockset and *Simulink 3D animation* blockset. The blocks that are used are described below.

VR To Video: The VR to video block can be found in the 3D animation blockset of Simulink. This block allows writing/changing values in the VRML world. The block also outputs a video data. The video output data is same as the selected viewpoint output or the view from camera defined in VRML.

Colour Space Conversion: The colour space conversion box is obtained from the Computer vision systems blockset of Simulink. This block converts colour information between colour spaces.

Video Viewer: The Video block is obtained from the computer vision systems blockset of Simulink. Video viewer block allows viewing a binary, intensity, or RGB image or a video stream.

The different blocks are used in conjecture with each other to create the model shown in Figure 36.

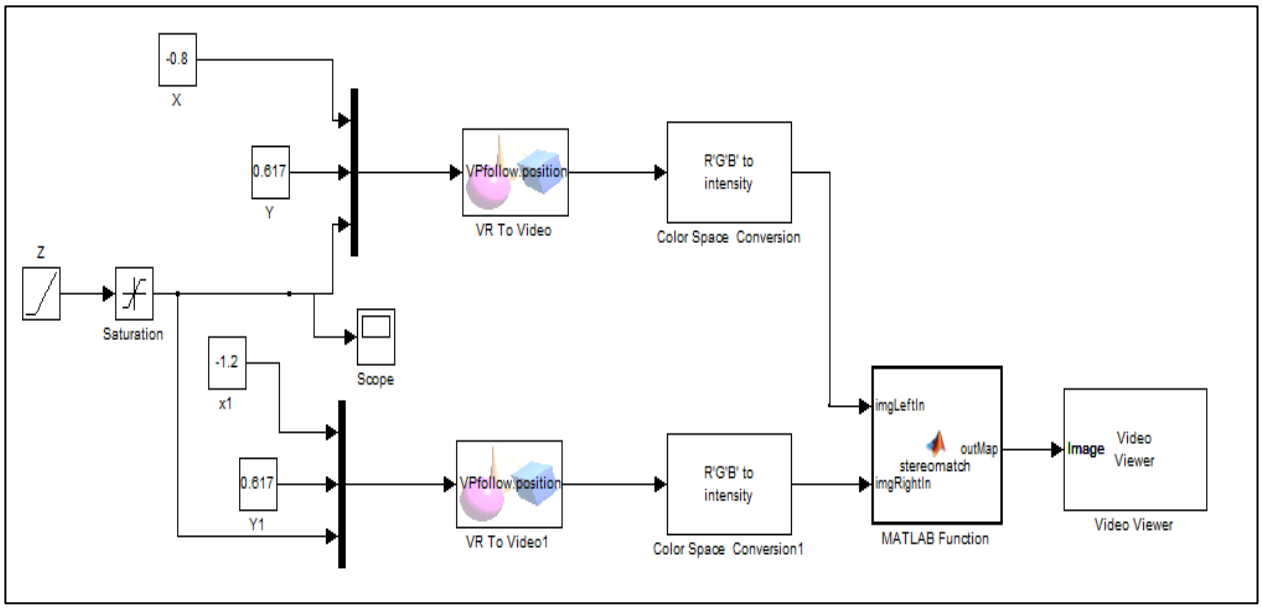


Figure 36: Model 1 schematic

Analysis of the model:

The model has two environments built into it, specifically “VR to Video” and “VR to Video1”. These blocks are loaded with the same environment. The locations of the observing viewpoints within these environments are varied. This is because, as mentioned earlier stereoscopic vision requires comparison of two or more images. Therefore, at least two video sources are required for comparing images. In order to have two camera output, two VRML environments are required. The two environments are a copy of each other; however, the location of the cameras within the environment will be slightly offset. The cameras offset will be in such a way that one of the cameras would appear as a left view camera and the other camera would appear to be right view camera.

These blocks accept the X, Y and Z co-ordinates of the selected viewpoint. All the parameters of the cameras such as the altitude (Y axis), orientation of the camera and the z axis would remain the same for both the cameras. These parameters only change when required to simulate the moving motion of the camera.

While running this particular model only the Z value is changed by using the ramp function. The altitude (Y) and the right and left displacement of the camera are held constant. Thus, the simulation model simulates the camera to move in only one direction.

Figure 37 shows the right camera and left camera view during the starting of the simulation. Figure 38 shows the difference after 5 sec of simulation. And Figure 39 shows after 15 sec. The object gets closer to the camera as time passes.

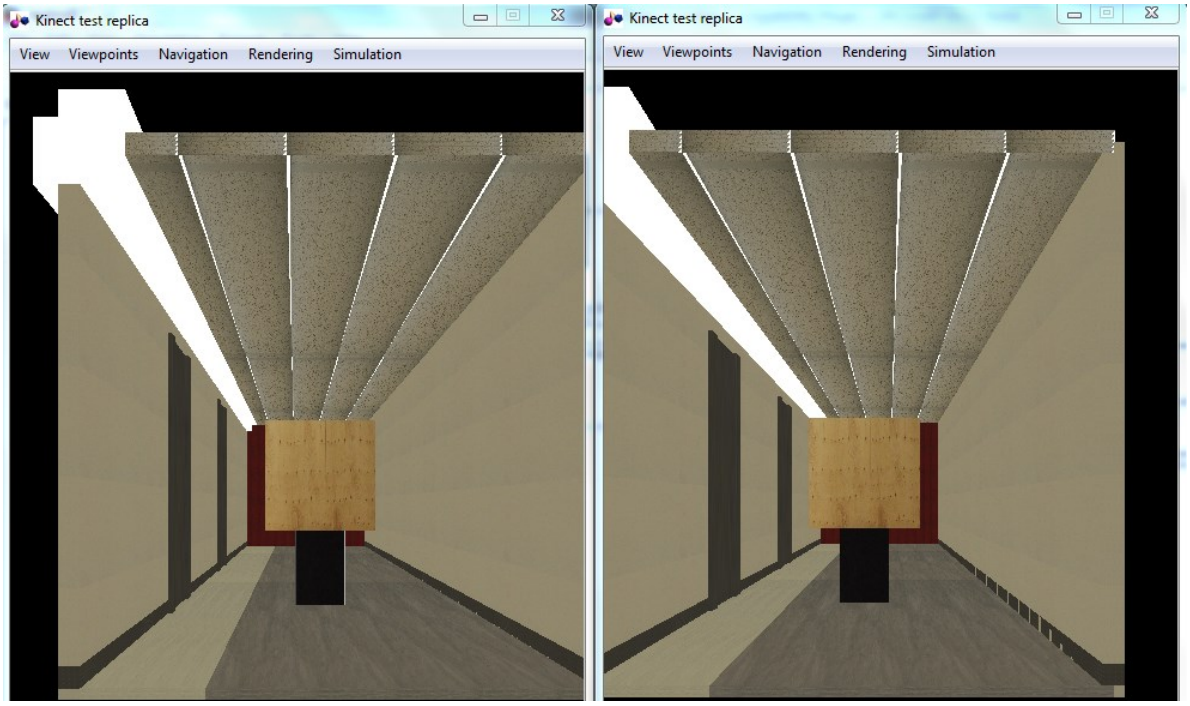


Figure 37: Cameras positioned at 0th second

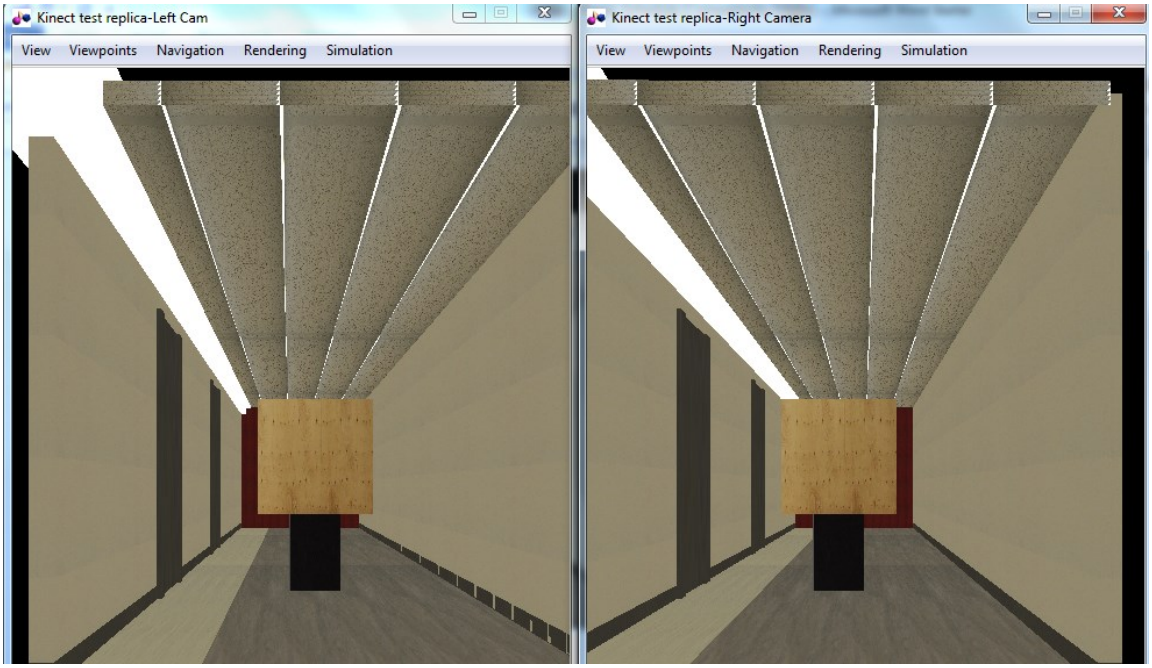


Figure 38: Cameras positioned at 5th second

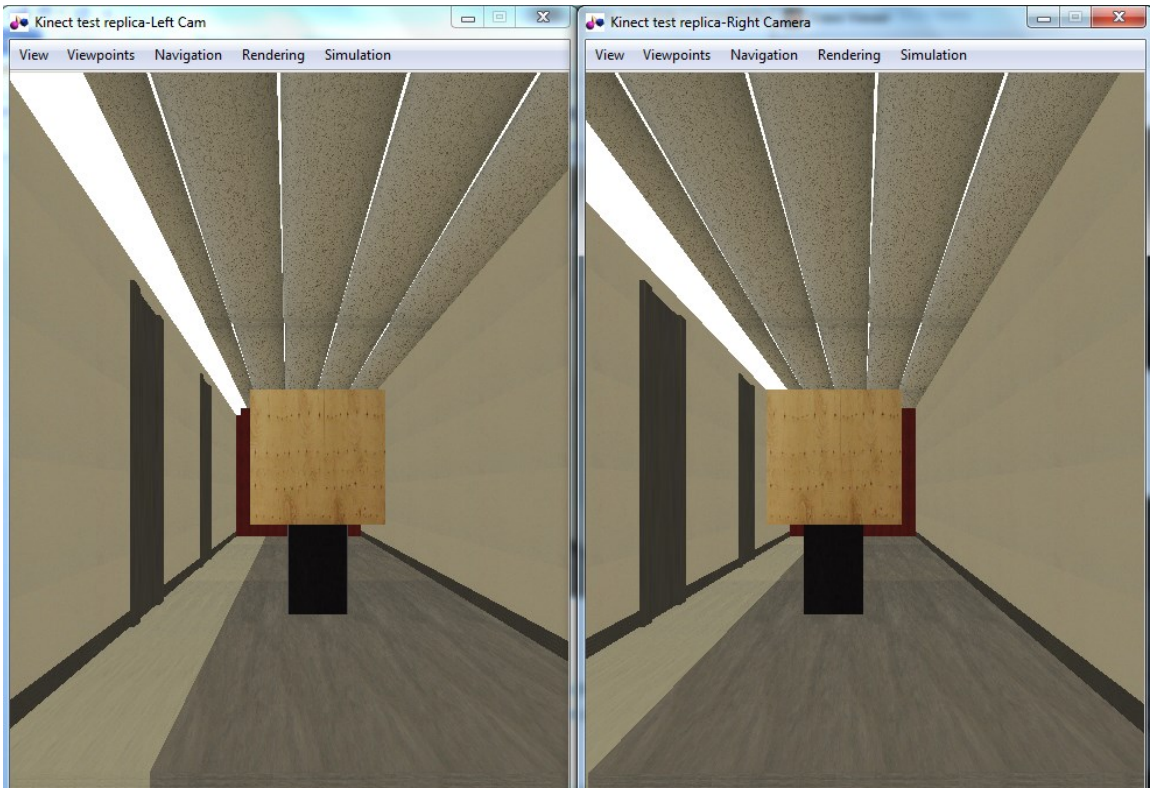


Figure 39: Cameras positioned at 15th second

By comparing the three sets of figures, it is observed the wooden object placed in the centre appears to get closer and closer since the camera is moving towards the object.

The video is fed to the respective colour space block, where the video is converted to grey scale in order to reduce noise while image processing. The Matlab function block compares the left and right video and calculates the disparity. The disparity is represented in the form of an image with different intensity, where white colour denotes the closest object and dark colour denotes free space.

The video viewer outputs the intensity map, which can be later used for comparing with data provided by the actual Kinect trials. Figure 40 show the intensity map produced after 5 sec of simulation run time.

Figure 40: Intensity map from two video at the 5th sec.

From the intensity map it is possible to recognize distinct features such as the object, door, floor etc. The code used in the Matlab function block is the disparity function which is inbuilt into Matlab. Appendix B1 shows the usage of disparity function in order to obtain the intensity map. Appendix B2 describes the parameters of the disparity function. These parameters are set based on trial and error, where on keeps trying various values in order to acquire a satisfactory intensity map as shown in Figure 40. The final selected values of the parameter are shown in Table 5.

Table 5: Disparity function parameters

contrastThresh	*blockWndSize*	*dispRange*	*uniqueThresh*	*textureThresh*	*distanceThresh*
0.9	15	[0, 64]	15	0.002	0

5.1.3.2 Model 2: Model used for Kinect data acquisition

This section describes the model used for acquiring data from the Kinect. The Kinect library provides with the required blocks for extracting data from the Kinect. The blocks used for this model is mention below.

NID IMAQ: The block acquires all the data from Kinect device. The block outputs the SYNC signal.

NID DEPTH: This block acquires the sync signal and returns the depth data from Kinect device. The depth device is returned in the form of XYZ coordinates and RGB image. The brighter image is closer to the device and black color denotes that the depth rate is lost.

Figure 41 shows the model with the NID IMAQ and NID Depth block.

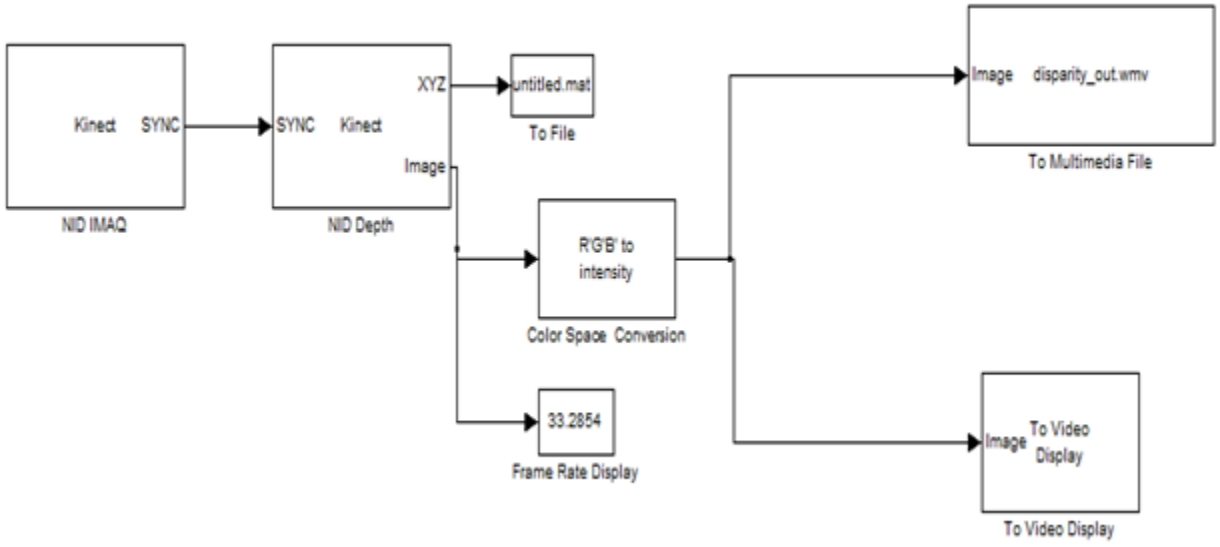


Figure 41: Model 2 schematic

Analysis of the model: Figure 41 shows the functions of the model. The NID IMAQ transfers the data to the NID depth block. The NID depth block provides two data as mentioned earlier, The

XYZ coordinates are saves into Matlab workspace (untitled.mat). Alternately, disparity map in the form of intensity map is saved in a video file (disparity_out.wmv). A detailed description is provided in section 5.2.1.

5.1.3.3 Model 3: Model used for UAV simulation

A new block was created in order to incorporate the VRML model into the Simulink framework. The block accepts the location of the aircraft in X, Y, Z co-ordinates along with the bank, pitch and yaw angles. Figure 42 shows the inputs from the simulation framework to the VRML world.

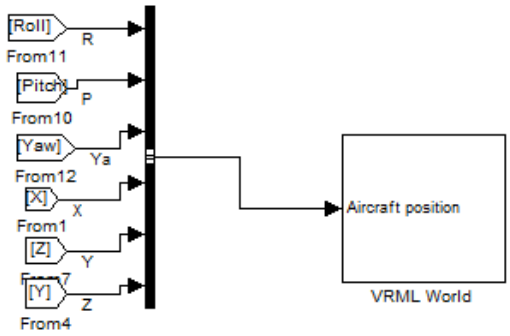


Figure 42: All required parameters for the VRML World

The X, Y and Z co-ordinates are passed on to the respective left and right VR to Video block with the left X coordinate being offset by a few centimeters. The offset can be set based on the determined physical hardware configuration of the cameras mounted on the aircraft.

Unlike the test models which had only 3-DOF freedom, the UAV model considers 6-DOF. This is accomplished with the help of bank, pitch and yaw angles generated from the Simulink framework. Similar to the previous test model (Model 1), two VRML environments were incorporated into the UAV model to provide the left and right camera video output. However,

since the previous model had only 3-DOF, it accepts only change in X, Y and Z co-ordinates. The UAV model provides 6-DOF by including the variables associates with the “rotation function” that is available in the VRML world. The bank, pitch and yaw angles were converted to the required “rotation” parameters of the camera in the VRML world by creating a new function. This function was responsible for selecting the axis in which the camera rotation would take place. Section 5.1 had explained the parameters associated with “*rotation*”.

Based on the available parameters, it should be noted that these parameters accepts rotation only in one axis at a single instance in time. Therefore, the axis with the largest angle was selected as the axis in which the rotation of the camera would occur. The logic behind the created function that determines the required angles is shown in Figure 43. The working code can be found in appendix B6

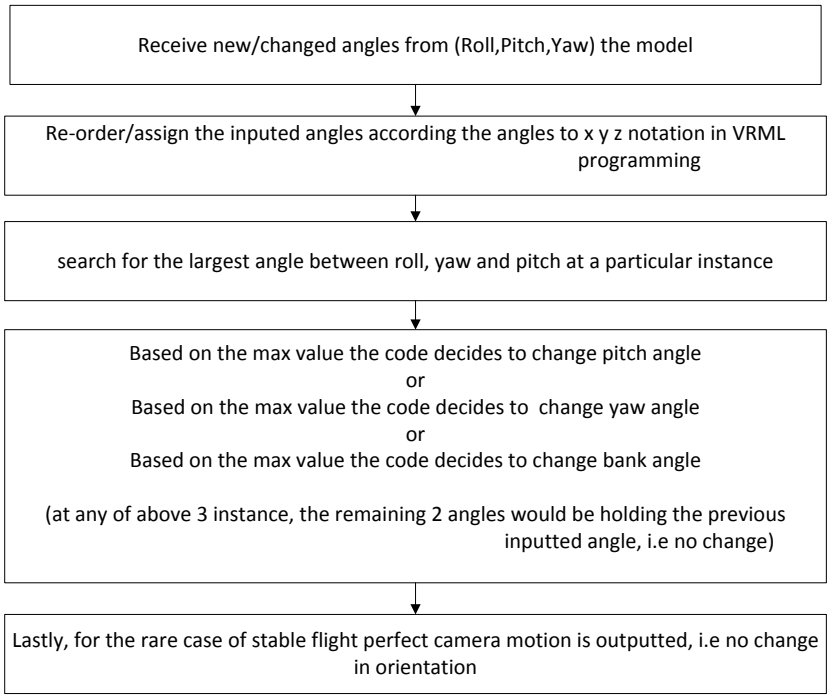


Figure 43: Orientation selection logic

The Figure 44 shows the architecture of the model. It provides a better understanding of how the six input variables (X, Y, Z, bank, pitch and yaw) are either transferred or modified before these variables are sent to the VR to video blocks. Similar to the previous test model, the VR to video blocks produce the video, it is then passed through the colour space box, the Matlab function and the video viewer. These blocks follow the same principles as explained in the test model.

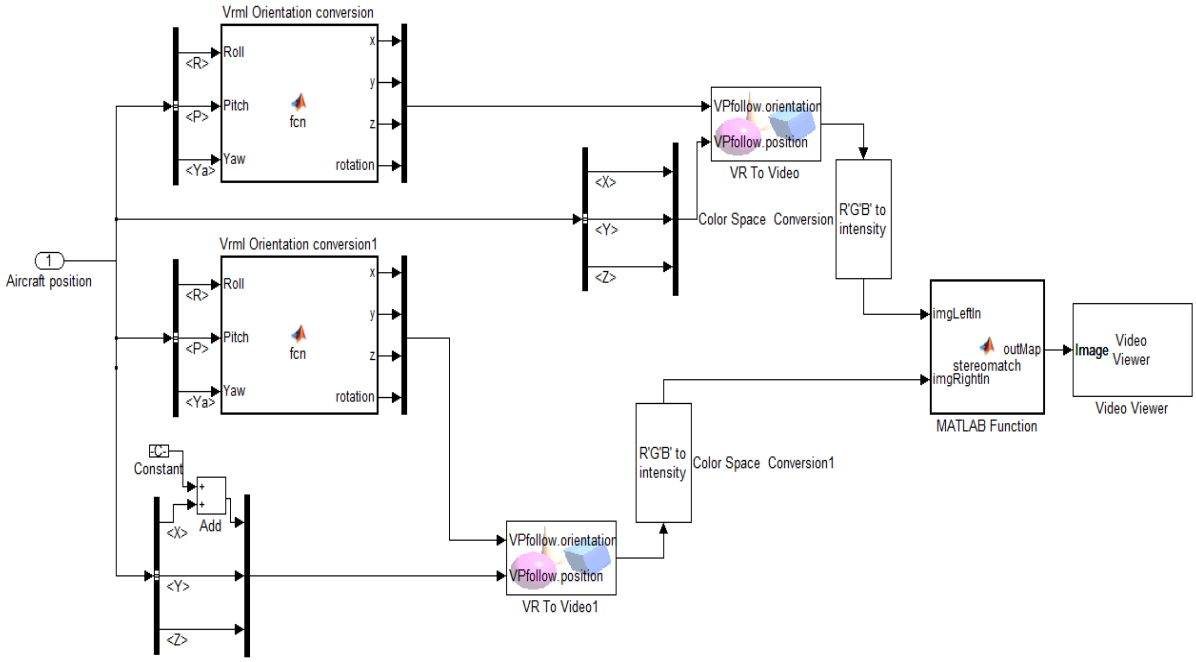


Figure 44: Model 3 schematic

These blocks are there to reduce noise, calculate disparity and view the intensity map. Figure 45 shows the left cam and right cam during runtime and Figure 46 shows the intensity map produced after comparing the left and the right cam. The stereomatch function is shown in the appendix B1. This function compares every pixel in a frame to its corresponding pixel in its corresponding frame. This function uses the disparity function provided by the Matlab library.

The parameters used within the disparity function were the same as the ones used to acquire Figure 40.

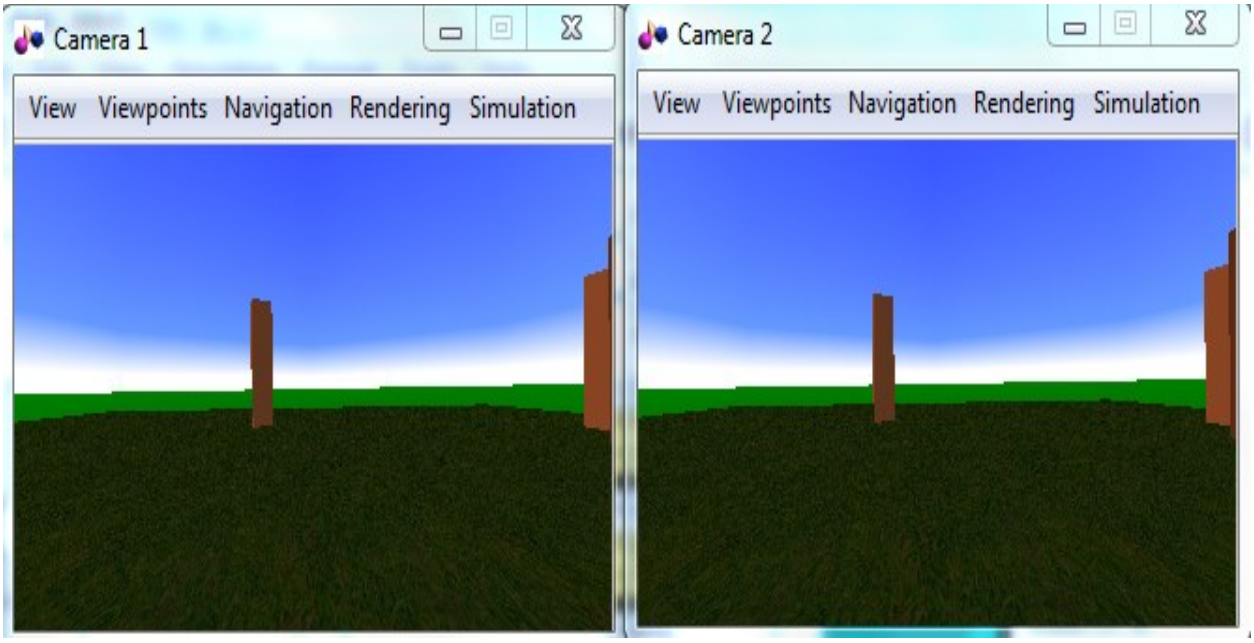


Figure 45: VRML world from video viewer

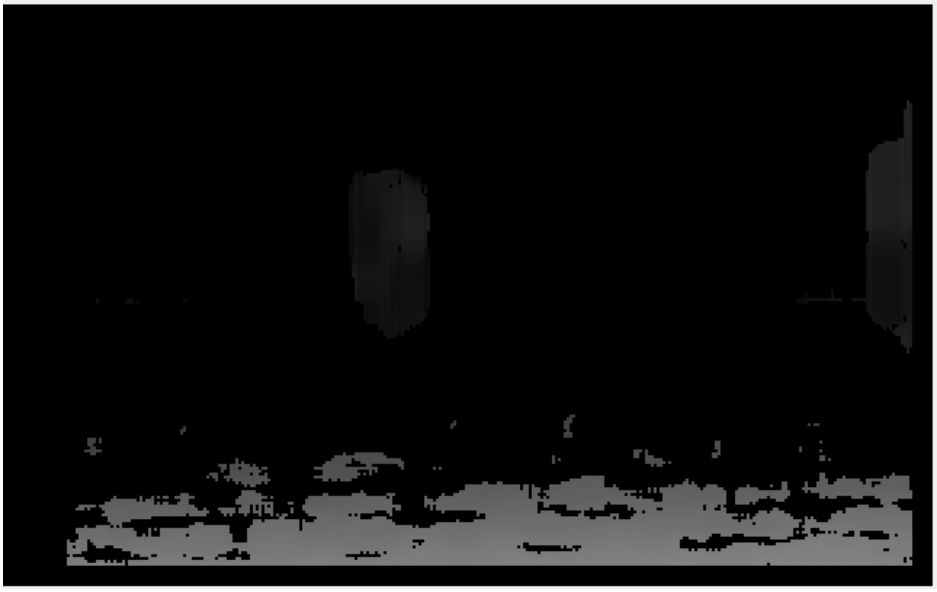


Figure 46: Intensity map created from data gathered from Figure 45

5.2 Observation and results

Section 5.2.1 describes functions of the Kinect and section 5.2.2 discusses the results obtained after analysing the vide data from the simulation and experimental trials.

5.2.1 Data acquired via the Kinect

Section 5.1.3.2 describes the model (model type 2) which is used to acquire data from the Kinect. The NID DEPTH block within the model 2 acquires the sync signal and returns the depth data and video data from Kinect device.

The depth data is returned in the form of XYZ coordinates. Where, each pixel from a still image is given an X, Y and Z value. Therefore, a still image of a video is represented by a 640x480x3 matrix. The entire depth data capture during the experiment is stored in the untitled.mat file. Based on the duration of the experiment, the matrix stored in the mat file varies. For example if the experiment was run for 10 sec, the based on the frames captured per second, which is 30 frames, the matrix stored would be 300 x 640 x 480 x 3. Thus based on the time of the experiment once can acquire the approximate relevant frame for image processing.

Simultaneously RGB video is also recorded. The process is the same as above with the exception of replacing the X, Y and Z values with R, G and B (contour intensity) values. The RGB video is converted to an intensity scaled video. Where the brighter image is closer to the device and black color denotes that the depth rate is further. This data is stored in disparity_out.wmv. Therefore, instead of saving a .mat file as shown in untitled.mat the data is converted in a video file for visual convenience. The video file can be converted back to the required matrix with help of matlab functions as shown in appendix B5. The disparity_out.wmv is the required

stereoscopic data that is used to compare with the stereoscopic data provided by the simulation environment. The untitled.mat file is used later for a single instance of comparing the calculated distance of the camera to the object from the simulation environment and the actual measured distance of the sensor to the object in the real world. However, untitled.mat is not necessary for this research as the distance can be measured physically in real world.

5.2.2 Analysis of the video data

In order to compare the results, the data acquired from both the simulation and the Kinect hardware are passed through the same image processing techniques.

Three different techniques are used in order to compare the data provided by the Kinect and the simulation. The three techniques can be described as:

- Connected components
- 2D map and
- Frame reprojection.

These techniques are described below along with the results observed from the respective technique. All these techniques are performed by Matlab using the video data gathered during simulations and experimentations. The code created for each of these techniques can be found in appendix B3-B5.

5.2.2.1 Connected Component

In this technique an algorithm was created which detects a disparity within a particular range. After detecting the particular range of disparity the algorithm searches for a cluster of pixels

with the same disparity range. The algorithm then determines whether the cluster is large enough to be considered as an obstacle or small enough to be considered as noise. The obstacles are then highlighted by drawing boxes around the region using the connected component function available in Matlab. The working of the algorithm is represented in a block diagram as shown in Figure 47.

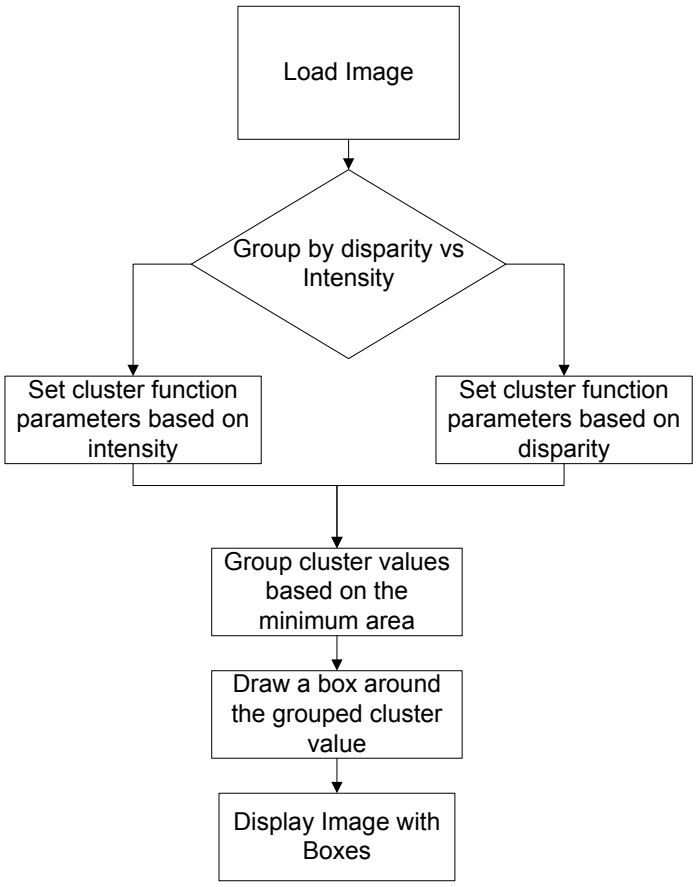


Figure 47: Working of Connected Component algorithm

A random set of screens were imported from the video file recorded during Kinect experimentation and compared to the respective screens acquired during simulation. Therefore, in reality while comparing the screens, the number of boxes that appear from the Kinect data and from the simulations should be similar.

However, it should be noted that the simulation environment is not an exact replica of the real environment. The simulation environment can be made more detailed, for example, every square foot of the floor/wall/ceiling could be made to look different based on the real life environment. The current virtual environment has the texture of the features repeated from a single image file.

Having different texture within a feature would mean increasing the probability of detecting a disparity among the pixels. A monotonous texture could potentially lead to lack of measurable disparity.

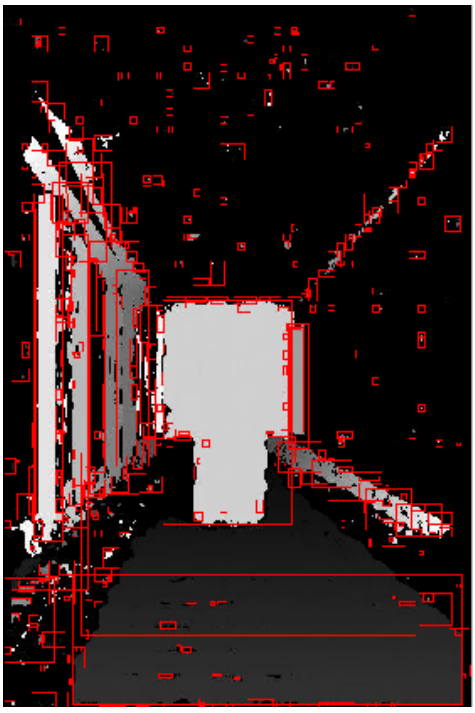


Figure 48: Intensity image acquired via simulation

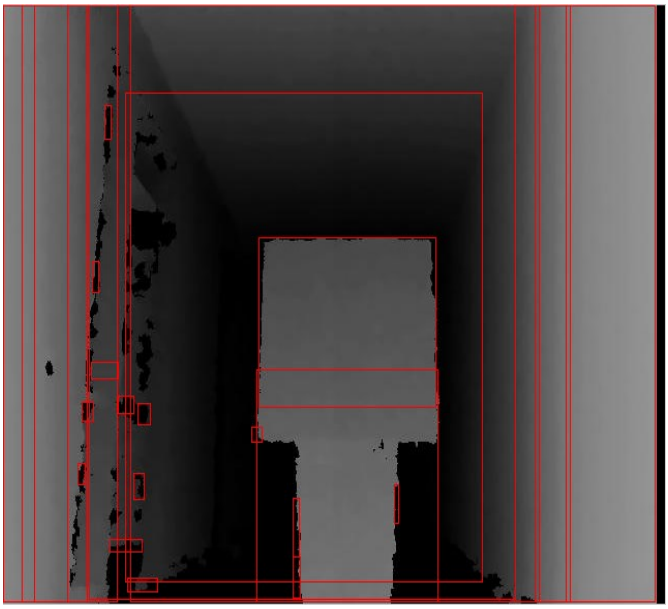


Figure 49: Intensity image acquired experimentally via the Kinect

Figure 48 and Figure 49 provides the results of the connected component algorithm. Figure 48 shows the stereoscopic data from simulation and Figure 49 shows the stereoscopic data from the Kinect. While comparing the Figure 48 and Figure 49, the major difference observed is the number of boxes. As mentioned earlier and also mentioned in Figure 47, the boxes represent the cluster of similar disparity value in a particular region. Hence, the number of boxes is being used as a measurement utility to compare the number of obstacles/disparity region detected.

It should be noted that the box is drawn only if the area of this particular region is greater than the provided threshold area. A trial and error approach was used to determine the threshold area. The threshold area is to be set in such a way that the boxes are around the potential obstacle and avoids boxing around errors. The threshold area was found out to be 20 pixel^2. Setting the threshold is based on mission requirements by estimating the expected size of an obstacle that is to be observed on an image. In this case, detecting the obstacle and ignoring

any errors caused by the rolling motion of the chair wheels were considered. It can be observed that a group of boxes in the simulation map are being represented as one box in the Kinect intensity map. As mentioned earlier, this is mainly because of the lack of complexity in the virtual environment.

Also, it should be noted that the distance from the camera to the obstacle is not the same for both the figures. The image from the simulation is the image from the start of the simulation which is 6.52m. On the other hand the Kinect is around 3.8m away from the obstacle. The difference in distance of the first image acquired in the Kinect and simulation world is predominantly because of the fact that the light source in the environment interferes with the infrared light produced by the Kinect. The Kinect is designed to be used within a 4m range. Consequently, the Kinect cannot detect an obstacle which is at 6.52m away from the Kinect. Instead of comparing both the images at 4 m, it was decided that we compare at the minimum range at which the obstacle can be detected to demonstrate that the simulation results are able to detect obstacles at a larger distance while maintaining a similar accuracy.

If the Kinect were used in a UAV, one can expect inaccurate results due to range and interference with sunlight. On the other hand the stereoscopic vision acquired via two ordinary cameras are expected to be as/more inaccurate as the Kinect due to the fact that these cameras experiences vibrations, displacement due to bending of the wing, delay in data acquisition due to wireless communication and many more unknown factors [Verbickas, 2012]. Hence, a source of noise is to be injected into the system to replicate the performance of a real-time stereoscopic system.

However, the main goal of the experiment to determine whether the VRML simulation can be used for creating video data that is comparable to the real world video data and if this video can be used for testing a future obstacle detection algorithm. Thus, allowing VRML simulation data to be used in testing obstacle detection and obstacle avoidance algorithms. The system can be made more robust by adding the required noise signal based on the disturbances experienced by the camera mounted on the UAV. The noise signal can either be modeled from a real-time obstacle detection system or can be standard noise models such as White Gaussian noise model that is available in the Simulink library.

5.2.3.2 2D Map

In this technique, an algorithm was created which produces an estimated top view of the corridor, i.e. a 2D map of the room is generated from the video file of the Kinect data and simulation data.

The algorithm firstly reads the video file and extracts multiple frames making the video and stores these frames in an array. A single line of horizontal pixel is extracted from a frame. The horizontal line is located at around 200 pixels from the bottom. The selected horizontal line can be located anywhere as long as the line includes the obstacle. Figure 50 shows the horizontal line of pixel selected from a single frame. The horizontal line comprises of only one pixel in vertical length, the Figure 50 shows an exaggerated vertical length.

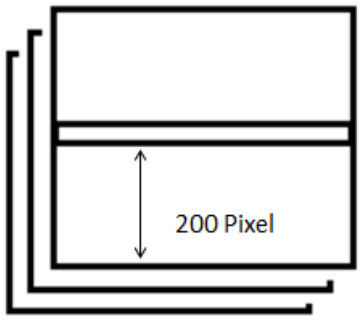


Figure 50: Dimension in pixel in a set of frames acquired via video

Similar horizontal lines are acquired from all the frames at the same location. These horizontal lines represent the features present in the room at a particular instance in time. If these lines were stacked up together, they would provide an approximated top view of the room, as the pixels in horizontal line provides approximate 3D information of its location in real space. This information is based on each pixel's intensity value. Hence, these horizontal lines can be used to represent a 2D map of the room. Figure 51 shows the first horizontal line being placed in the 2D image to be created.

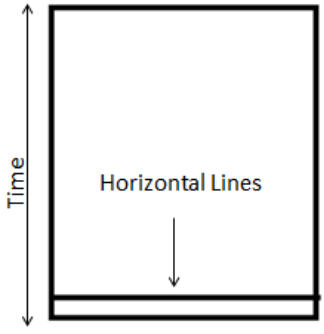


Figure 51: Creation of 2D map from a video

For the 2D mapping method, multiple simulations were run on the virtual environment at different speed interval. The speed of the forward motion of the camera was changed to run

the simulation in 7sec, 17sec, 24sec, 42sec and 55 sec. This was achieved by increasing or decreasing the slope function of the Z co-ordinate in model 1. This demonstrated the effects of simulation speed on the quality of data acquired via simulation. Figure 52 shows the 2D map attained via Kinect data which is a 7 sec video.

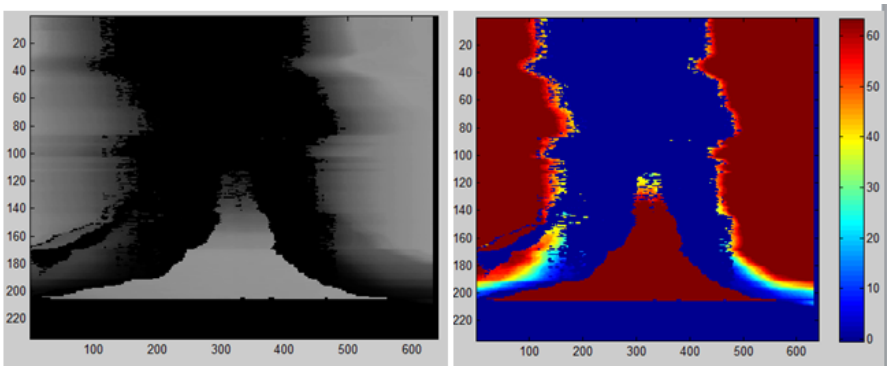


Figure 52: 2D map created from Kinect video (7sec)

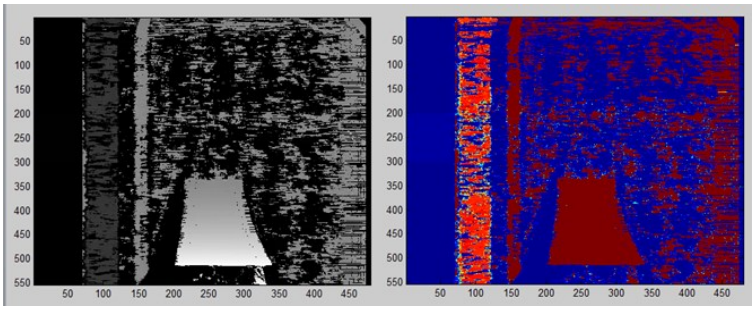


Figure 53: 2D map created from simulation video (55sec)

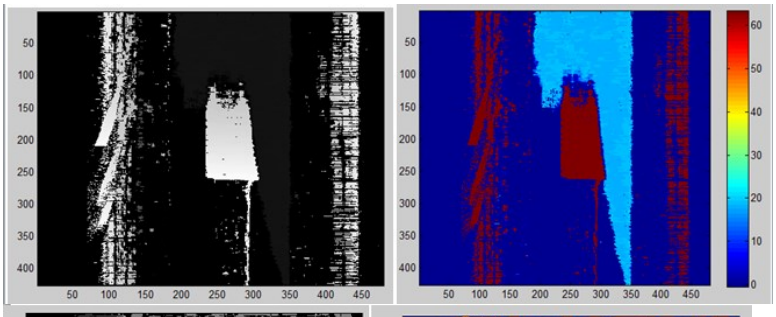


Figure 54: 2D map created from simulation video (42sec)

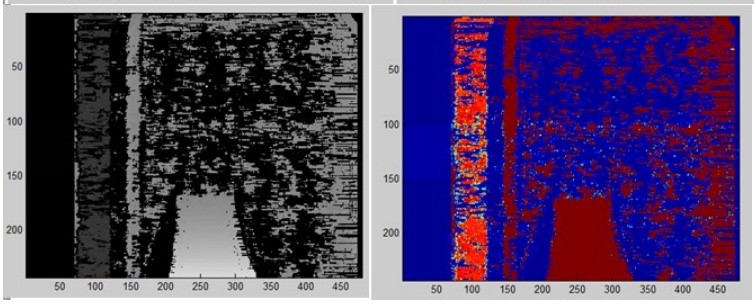


Figure 55: 2D map created from simulation video (24sec)

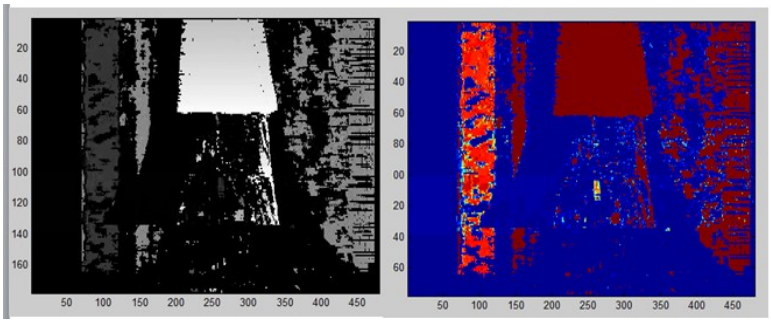


Figure 56: 2D map created from simulation video (17sec)

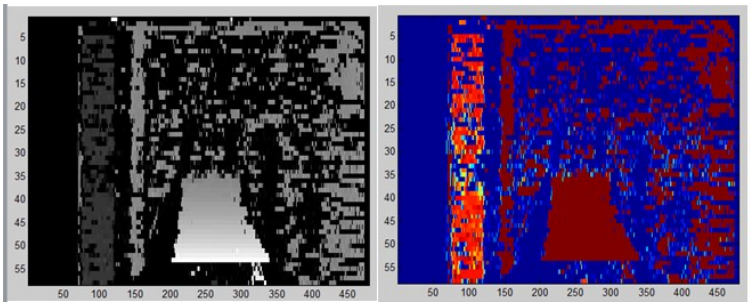


Figure 57: 2D map created from simulation video (7 sec)

The 2D maps represent the corridor without any errors. The 2D map attained from the Kinect data appears to be accurate, and the actual run time to get the Kinect video was 7 sec. In order to replicate the 2D map produced by the Kinect, the simulation needed to be run for around 55sec. Thus if one were to replicate the exact data via simulation, the motion of the camera within the 3D environment needs to be slower than what is observed in real time. However, speeding up the simulation does provide a 2D map, but with less clarity as demonstrated in Figure 54 to Figure 57.

Thus, real-time motion in the 3D environment does provide reliable data, higher definition data can be gathered by slowing the simulation speed by slowing the motion in the 3D environment. Please note that this technique unlike connected components and reprojection which are algorithms that are used to detect obstacles is not meant to be used for obstacle detection.

This technique was developed by this research to analyse the video quality by comparing the available two video data from Kinect and simulation environment.

5.2.2.3 Reprojection

In this technique the disparity values are used to attain 3D information. Before explaining this technique, background information on the Pinhole Camera model is provided below.

Pinhole Camera

The pinhole camera model provides a mathematical relationship between the co-ordinates of a 2D image and the actual 3D world. Where the 2D image is acquired on the screen of a pinhole camera, where the aperture of the camera is a point and does not have an optical lens to focus light. Since the pin hole camera does not include the effects on the acquired image provided by a the lens of a camera, it makes the pinhole camera model an approximate model for mapping of a 3D scene from a 2D image. The basic terminology and the ideology behind a pinhole camera model are shown in Figure 58.

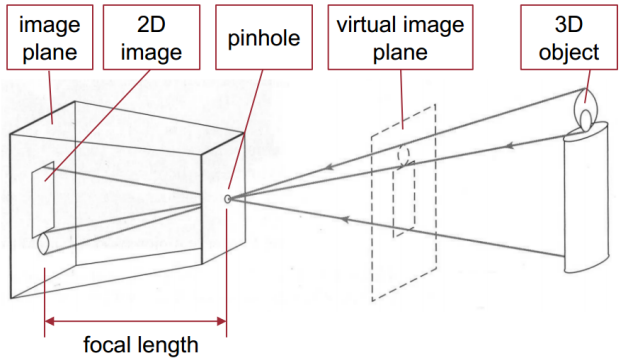


Figure 58: Terminology used in a pinhole camera model [Kheng, 2012]

By using a pinhole camera model, simple equations can be used to describe the perspective projection which occurs of 3D objects onto the 2D image plane. Assuming (X,Y,Z) to be the

coordinates of a point in 3D space(P_o), one can project point P_i (u,v) on the image plane, given a focal length f and centre of projection O. The image plane is a distance f from the centre of projection along the optical axis:

$$u = \frac{Xf}{Z} \quad \text{and} \quad v = \frac{Yf}{Z} \qquad \textbf{(48)}$$

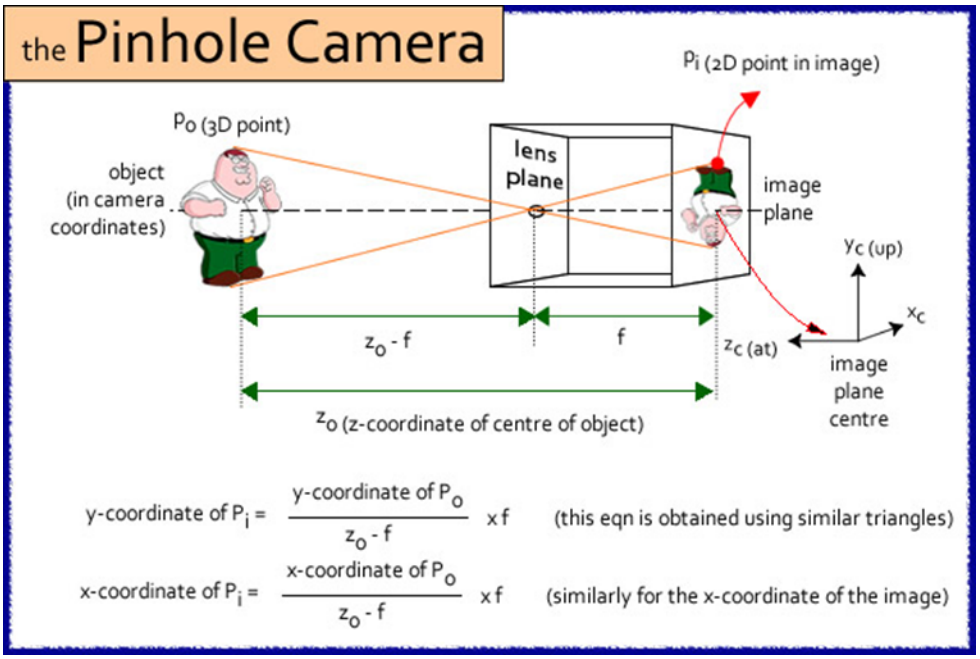


Figure 59: Relationship between image and actual object [Pushypanda, 2013]

Since the points (x,y)/(u,v) are measured in the camera reference frame (CRF) they also have physical units. Since images features are typically referred to by their pixel location a conversion must be made from the CRF to image coordinates x_{img} and y_{img}. The conversion can be written as follows:

$$x_{img} = s_x * x + c \qquad \textbf{(49)}$$

$$y_{img} = s_y * y + c_y \qquad \textbf{(50)}$$

The parameters s_x and s_y are the horizontal and vertical scaling factors respectively and c_x and c_y the principal points. The scaling factors describe the physical dimensioning of pixels while the principal points describe the origin offset of the camera reference frame with respect to the image plane [Verbickas, 2012].

Generally, gathering the required variables for the above equations would require camera calibration, however, while using the simulation environment, these parameters are specified while creating the environment. The field of view of the camera is set in the VRML script. The field of view is considered to be satisfactory when one is able to observer most of the environment via the camera.

Parameters	Source
Field of View= 57.29^o	VRML Script
Baseline= 40cm	Simulink Model

In order to reproject data into a 3D plot, it would be better to use the actual disparity data used to create the intensity map rather than using the intensity map itself. This is because, a range of disparity values are allocated to a particular intensity, therefore, reducing resolution of the data during the process. The stereomatch function which is responsible to output only disparity map previously but was now modified to output a 2^{nd} set of data into a matrix in matlab as a .mat file. The 2^{nd} set of data contained the disparity values.

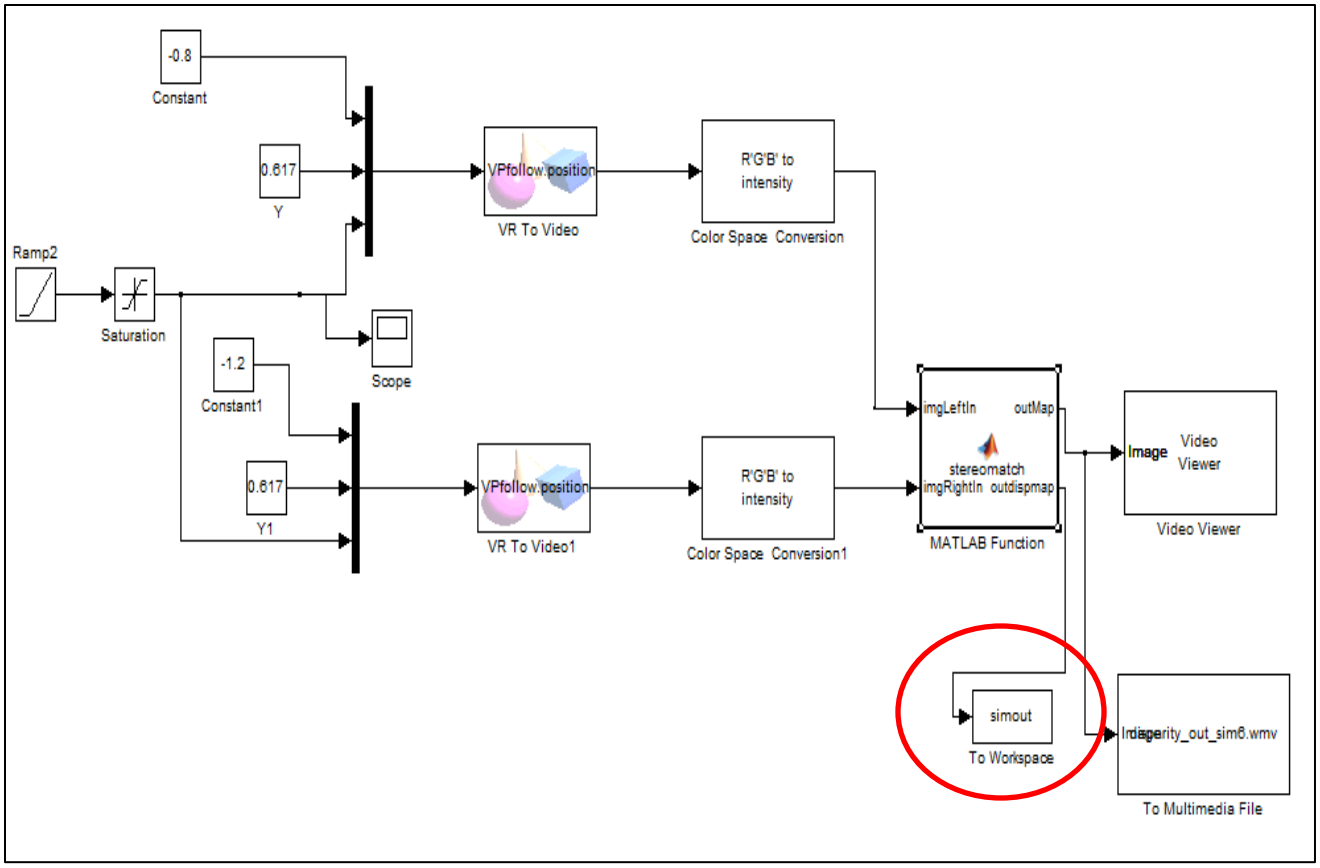


Figure 60: Model 1 with To Workspace block

Figure 60 shows an additional block 'To workspace' which sends out the disparity value into Matlab as simout.mat.

After obtaining a disparity for the pixels in an image from the simulation file, it is possible to reproject the pixels into 3D space. To accomplish this, the re-projection Q matrix is used [Bradski, G., 2008].

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & 1/T & (c_x - c'_x)/T \end{bmatrix} \quad \textbf{(51)}$$

This matrix is a compact version of the camera reference frame centered 3D to 2D projection equations described above and as mentioned all parameters can be acquired during virtual environment creation. The terms c_x and c_y are the principal point co-ordinates for the reference image (left or right). T is the distance between 2 cameras. f is the focal point, which is assumed to be the same for both the camera. The focal point is acquired from field of view (FOV) parameter that was mentioned earlier. The relationship between FOV and f is shown below.

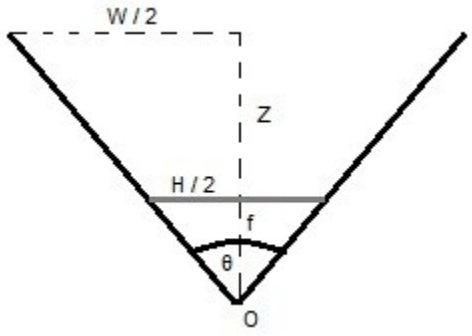


Figure 61: Relation between FOV and f

$$\frac{W}{2Z} = \tan\left(\frac{\theta}{2}\right) = \frac{H}{2f} \quad \textbf{(52)}$$

$$f = \frac{ZH}{W} \quad \textbf{(53)}$$

Where, θ is the FOV angle. H is the vertical dimension of the screen in pixels. Where, the vertical and horizontal dimension is equal to 640pixels. Z is the maximum depth in minimum disparity.

$$\frac{W}{2Z} = \tan\left(\frac{\theta}{2}\right) \quad \textbf{(54)}$$

Multiply by $\frac{2Z}{H}$ on both sides

$$\frac{Z}{W} = 2\tan\left(\frac{\theta}{2}\right) \quad \textbf{(55)}$$

Hence, **(56)**

$$f = \frac{H}{2\tan\left(\frac{\theta}{2}\right)}$$

Therefor f was determined to be 206 pixels.

The term c_x' is the horizontal coordinate of the principal point in the opposite image. The principal points will have no disparity for an object that is at infinity. In the case where the principal points are set to intersect at infinity, the following equation holds true.

$$c_x = c_x' \quad \textbf{(57)}$$

The re-projection matrix allows computation of physical 3D coordinates to feature points using the following expression:

$$\begin{bmatrix} X_h \\ Y_h \\ Z_h \\ W_h \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} \quad \textbf{(58)}$$

Where X_h, Y_h, Z_h and W_h are the homogeneous coordinates of a pixel having image coordinates (X,Y) and disparity d. The 3D coordinates (X, Y, Z) of the feature point can then be recovered since:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_h/W_h \\ Y_h/W_h \\ Z_h/W_h \end{bmatrix} \quad \textbf{(59)}$$

Being able to compute the depth to pixels we need to take into account two limitations to any stereoscopic camera system: range accuracy and range resolution.

Range accuracy is a measure of how close our estimate of disparity or depth is to the ground truth depth for a particular 3D point. This accuracy is affected by a number of factors, including errors introduced during the calibration process as well as errors in the stereo correspondence process.

Range resolution on the other hand is a fundamental limitation of a stereoscopic system at discerning how finely range measurements will vary at a particular depth, for a given pixel or feature point, based on variations in the disparity estimate. To compute range resolution Δz, we can consider Δd to be the corresponding disparity delta which then allows us to write the following expression for Δz: [Verbickas, 2012].

$$\Delta z \approx \frac{z^2}{f * T} \Delta d \qquad \textbf{(60)}$$

A function named reproject was created by this research project, which can be seen in appendix B4. This function calculates the obstacle co-ordinates based on the disparity values saved in matlab. The co-ordinates are attained from the equation shown earlier. This function also creates a 3D plot of the obstacle co-ordinates, hence, recreating the 3D environment. This 3D plot can be compared to its corresponding dimensions in real world. Figure 62 shows the reprojected co-ordinates from the disparity values acquired from the first frame shot of the simulation.

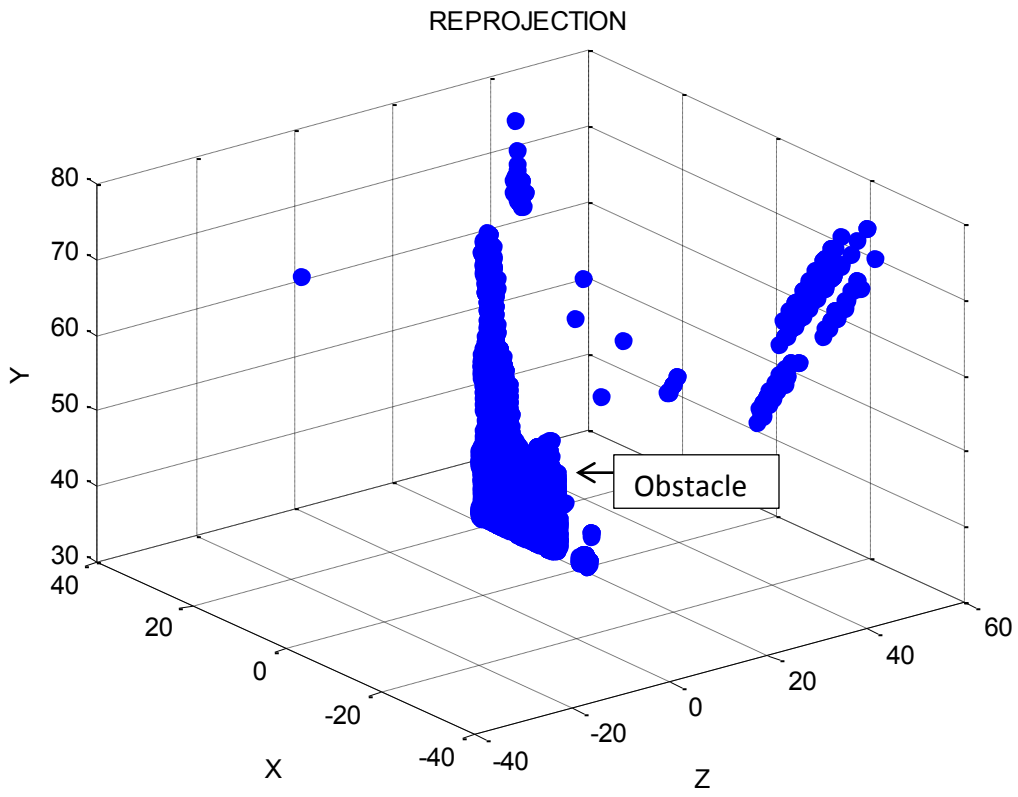


Figure 62: Re-projected 3D map from simulation data

The co-ordinates dimensions were initially in pixels. The equivalent dimension in meters can be calculated by determining $\frac{W}{H}$. The relation was found out to be 0.347pixel/ft (1.14 pixel/meter). Figure 62 shows that the obstacle is at 20ft (6.09m), In reality the obstacle is around 21.3ft (6.5m). This range accuracy error is marginal and is acceptable as the reprojection matrix is used for approximation rather than detecting accurate range. An advanced version of the code would use "Dynamic Programing" to acquire better stereoscopic data [Veksler 2005] [Park 2000]. This technique accompanied by concepts such as "*Reprojection Error*" and "R*ectification Error*" could provide more accurate results [Bradely, 2010].

These techniques approximate the errors using statistical models and makes the required

adjust to the acquired co-ordinates to provide precise distance information. These techniques of “Dynamic Programing”, “*Reprojection Error*” and “R*ectification Error*” are beyond the scope of this research. The algorithm used by to acquire the 3D map is most basic in nature using only the re-projection matrix. The objective here is to show that the VRML provides valid video data for various type of image processing.

Obstacle detection code to handle these errors would be more complex and require a high safety margin so that the aircraft would have enough time and space to maneuver over/around the obstacle using the re-projection matrix. Besides, the aircraft being GeoSurv which is larger in size compared to UAVs with similar size as that of ATB-10 requires higher safety margins to perform maneuvers. This is because a larger UAV will require comparatively more air space and more power to perform the maneuver. In that perspective, provided a high enough safety margin, a basic code as the one used here would be adequate.

6.0 Connectivity

This chapter describes the integration of the Simulink frame work with the ATB-10. After the development of a functional Simulink framework, which incorporates hardware-in-the-loop, obstacle detection and avoidance, and the aerodynamic responses of the aircraft, the framework can linked to the ATB-10 to compare the results obtained during simulations and real time test flights. The ideology on how the link has been established between the framework and the ATB-10 is explained in section 6.1, the subsequent sections describes the theory and techniques used to make this idea a reality.

6.1 Methodology

There are many ways in which one can send navigation commands to the autopilot. The company selling the MicroPilot, also sells a software development kit (SDK) that enables linking Simulink to the autopilot. However, if the SDK were to be used, it would require advanced programming skills. Also, the Horizon ground control software cannot be used simultaneously with SDK.

Alternatively, there are various developments being made in autopilot systems and hence there are alternative autopilots available in the market. Therefore, it would not be a farfetched idea to assume that the current autopilot could be replaced at some future time in the development of the GeoSurv II.

Consequently it was decided to not to depend on the SDK but rather develop a basic technique that would establish a link between Simulink and the ATB-10. This would allow modifications to be made within the Simulink framework and the ATB-10 without affecting the communication link between the two.

In order to develop a link which is independent of Simulink and ATB-10 modifications, one has to find components within Simulink and ATB-10 that would likely be untouched under most possible modifications. The list of components that are always available and untouched are listed below.

ATB-10: Servo, Radio Control Receiver/ Transmitter and batteries

Simulink: The 'T*o Instrument*' block in the Instrumentation blockset

The navigation signal is sent from Simulink to the radio control frequency. One possible method to do this was to purchase a transmitter that can be linked to a computer, which could transmit signal at a particular frequency that the receiver in the ATB-10 would recognize. This would imply purchasing a different set of transmitters every time the Radio Control system is changed/upgraded within the ATB-10. Not only this would be expensive, it would also mean losing time in developing an interface within the Simulink framework for the new transmitter. Besides, having two transmitters in the same frequency would most likely induce errors on the receiver module and could henceforth cause the ATB-10 to crash.

Consequently another robust technique was discovered and implemented. In order to understand this technique, one must understand how a 'master/pupil' relation works on a model aircraft radio controller. The Figure 63 below explains the 'master/pupil' mode.

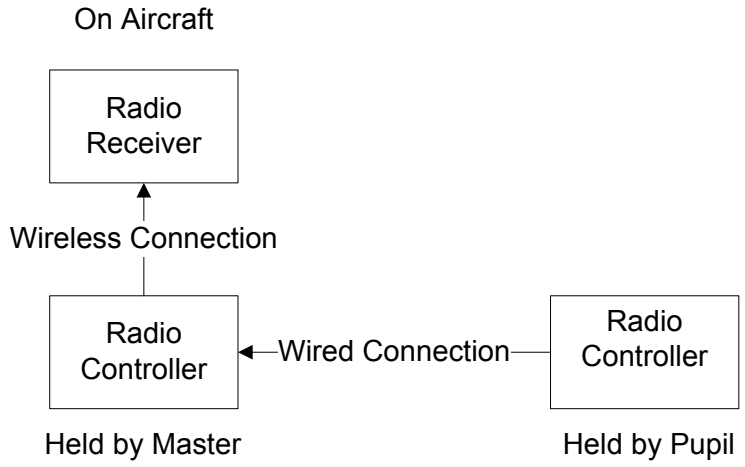


Figure 63: Master/Pupil (Teacher/Student) Mode

The master/pupil mode is used between two radio controllers where one is held by the teacher and the other is held by the pupil or student. The teacher's controller the required signals to

the aircraft. Both the RC are connected via a 2.5mm audio cable. The student's radio controller transmits the signals to the teacher's controller and then the teacher's controller transmits the signal to the aircraft.

The teacher's controller is usually equipped by a "*dead man's switch"* which controls whether or not the student's signal is sent to the aircraft so that as long as the teacher holds the transmit button down the student is in control of the aircraft and if not the teacher takes over control of the aircraft.

There are many other setting in the teacher's controller that allow one to adjust the sensitivity of the signal ported in from the student's controller. For example, a student's full throttle could be set to 50% of the throttle. Hence, allowing only the teacher access to full throttle.

Now, it is possible to replace the student's controller by a computer running Simulink. If one understands the type of signals sent by the RC via the audio cable, one can replace the RC with a computer. The types of signals used are identified and mentioned in section 6.1.1. Under this method, the pilot is always in command to avoid any catastrophic crashes caused by signals sent by the computer. Figure 64 shows the modified method (the main ideology for establishing link between Simulink and aircraft).

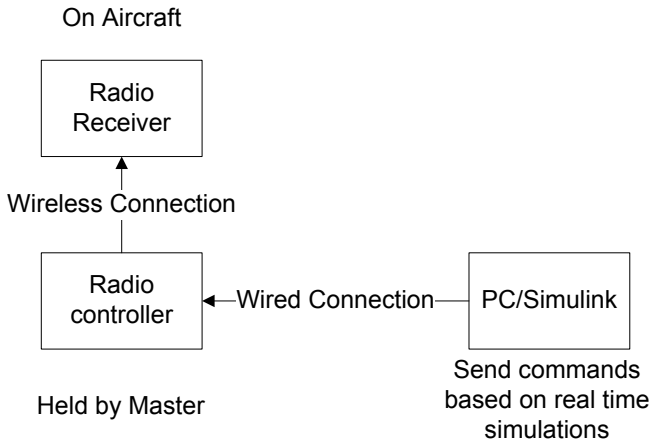


Figure 64: Modified method

6.1.1 Types of signals used between RC transmission

All communication between the transmitter and the receiver are performed by pulse position modulation (PPM) signals. PPM signals are a sequence of pulse width modulation (PWM) signals. The section below describes both these signals in details.

Pulse Width Modulation

A pulse is an electric signal which has a beginning and an end for a particular amount of time. In digital electronics, this would imply turning the switch to high/on for a short amount of time and low/off for the remaining amount of time. Figure 65 shows example of a generic PWM pulse.

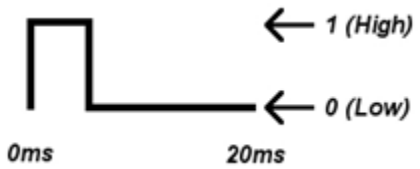


Figure 65: PWM

PWM signals include a consecutive number of pulses. The finite time during which each pulse is on is known as “Time ON” .The time it takes for the signal to turn on is known as “Time OFF”. While dealing with PWM, the pulses repeat in a period which remains constant. The required information from a pulse is gathered from the ratio of Time ON and Time OFF during the set period. This ratio is known as the “Duty Cycle”. Figure 66 explains the workings of the “Duty Cycle”.

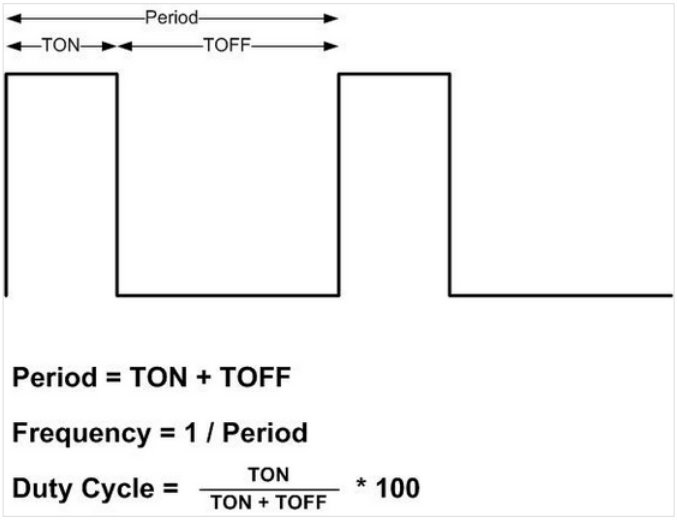


Figure 66: Duty cycle [Avayan, 2013]

Servos respond to these PWM signals. Based on the duty cycle, the servo receives the information to turn its arm to an absolute angle. Almost all servos work on the principal that has the duty cycle which is proportional to the angle of rotation. However, the servo settings can be changed to change its reactions based on different duty cycles. An example relation between the PWM and the servo is shown in Figure 67.

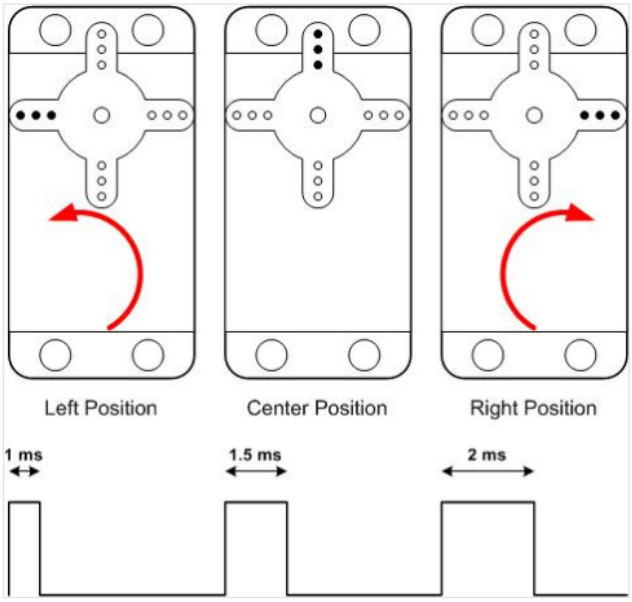


Figure 67: Servos responding to PWM signal [Avayan, 2013]

A radio controlled aircraft usually has more than one servo. Therefore, one PWM signal is not sufficient for complete control. Hence, RC transmitters and receiver do not directly communicate with PWM so that multiple servos can be controlled by one transmitter. These devises uses another type of signal which are known as pulse position modulation (PPM). The workings of PPM are described below.

Pulse Position Modulation

A PPM signal is basically a stack of PWM signals sent together at set frequency. Based on the number of channels the radio controller has, the PPM signal will contain the same number of PWM signals [Ince, 2003]. The radio controller used by the ATB-10 has 8 channels, therefore, it sends out 8 PWM signal in one PPM signal. The 8 PWM signals are positioned in one frame, which is essentially one period of the PPM signal. The frames used in radio controller are 20ms long and in some rare cases 25ms long.

If PWM signals are sent individually, which is never the case; it would take 20ms x 8 = 160ms. However, with PPM signal it takes only 20ms to send the same data as the eight PWM signals are compressed into one PPM signal. A diagram of a partial PPM signal is shown in the Figure 68 below.

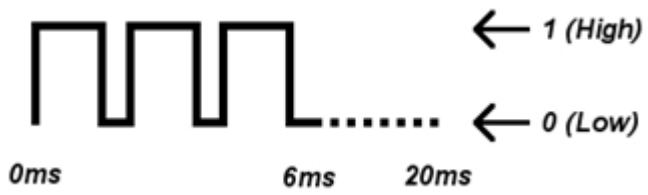


Figure 68: PPM Pulse

6.2 Establishing connection between computer and the radio control

The computer cannot send PWM or PPM signals directly. A computer generally uses serial communication or Ethernet communication. Both connection types can be used for our purpose provided; there is a device in between the computer and the radio controller that can convert serial data into PPM signals. Using a microcontroller such as a Beagle board or Arduino can help convert the Ethernet or serial signals to PPM signals.

Using Ethernet signal would require using TCP/IP protocols. Since Ethernet is generally used to handle larger amounts of data, it was decided that serial communication would be used to send PPM signals. Since the Ethernet port was not required from a Beagle board, it was decided that a simpler and less expensive microcontroller such as the Arduino UNO would serve the purpose. Another reason for selecting Arduino was that, the Arduino language would be used regardless of using Arduino or Beagle board. The Arduino code was used due to its simple

programming structure and the author's familiarity with the Arduino from other projects. Figure 69 shows the proposed setup.

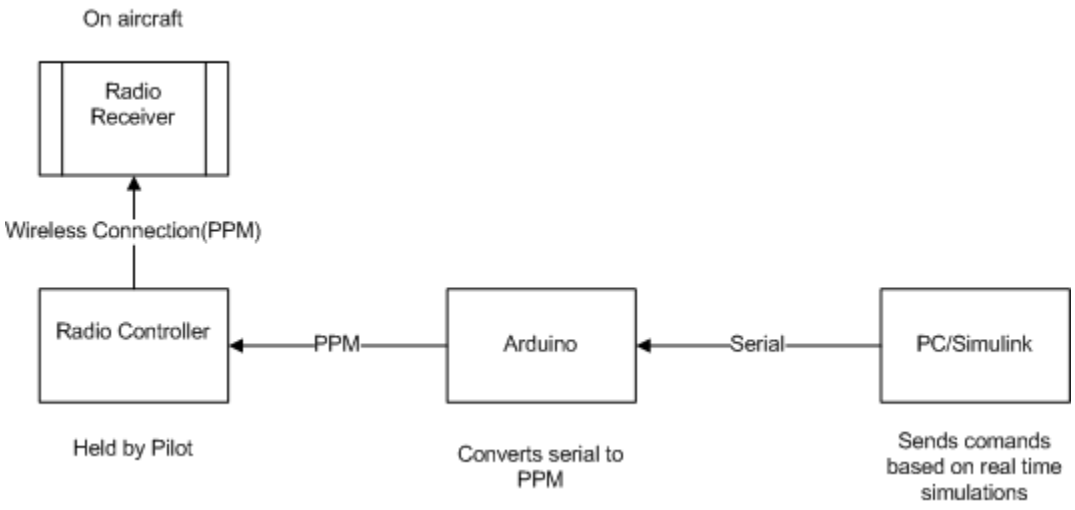


Figure 69: Setup diagram with Arduino

The subsequent section shows how the proposed setup from Figure 69 was accomplished. These sections cover both the hardware and software work done in order to establish the link mentioned in the above Figure 69.

6.2.1 Setup required in computer

The serial link required between the computer and Arduino is achieved by a serial cable with one end of class Type A and of class Type B. Type A connects to the computer and Type B connects to the Arduino.

Simulink can be used to send serial signal using the '*To Instrument*' block in the '*Instrument Control Toolbox*'. The '*To Instrument*' block configures and opens an interface to an external instrument, initializes the instrument, and sends data to the instrument. The configuration and

initialization happen at the start of the model execution. The block sends data to the instrument during model run time.

The block has only one input port which intakes the data that needs to be sent to the instrument. This data type must be double precision based on the default requirements of the block. If the data required by the instrument is to be converted, the '*To Instrument*' data will convert before sending it via serial port. Figure 70 and Figure 71 shows the required parameter set for successful interface between Simulink and Arduino. The Port number must synchronize with the port number of the Arduino. Under the hardware configuration the values are mostly generic. Description of these parameters can be bound in Table 6.

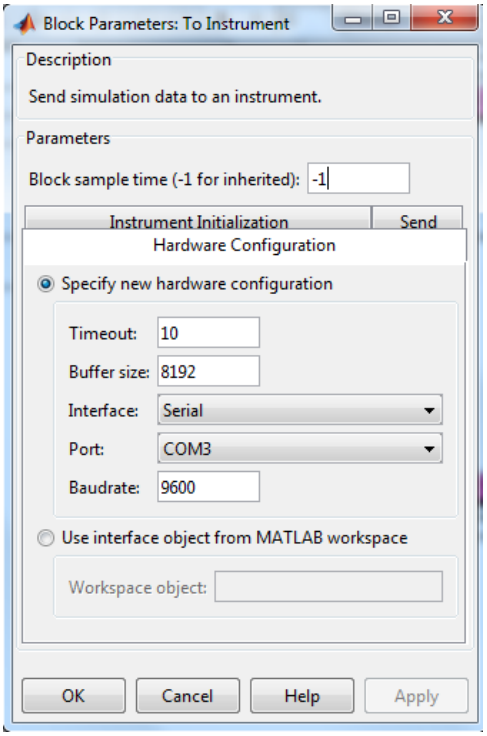


Figure 70: To Instrument block parameter

Table 6: Hardware configuration parameters

Parameters	Description
Timeout	Time in seconds, allowed to complete the query operation, Default 10sec
Buffersize	The total number of bytes that can be stored in the software output buffer during a read operation, Default 8192
Interface	Select the type of hardware interface to the instrument. Multiple options available, select serial since Arduino deals with serial interface
Port	The port number for the serial interface. Select accordingly
Baudrate	The rate at which bits are transmitted for the serial, 9600 selected. Selection base on Arduino compatibility, which is determined during Arduino programming

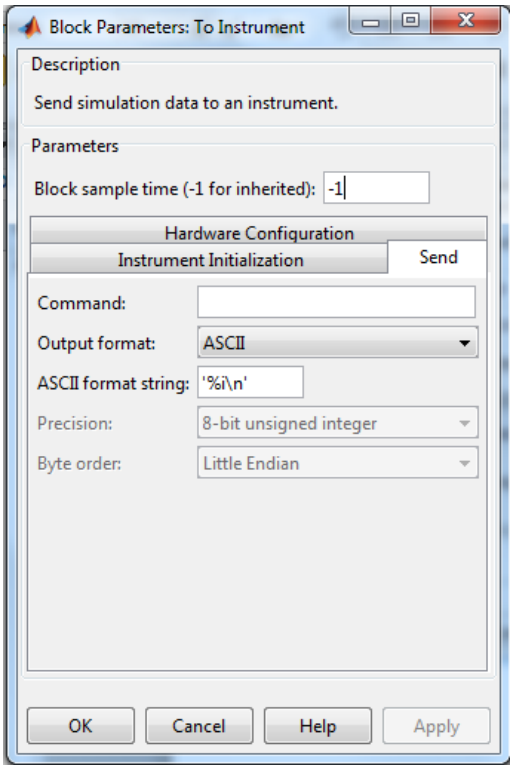


Figure 71: To Instrument block parameter

Under the send tab the most important field is the ASCII format string. Based on the software code written in the Arduino, the Arduino expects an integer value to be received. The integer

values were chosen so that the required angle values from 0° - 180° could be sent. The Arduino expects a return key character at the end of the integer. This integer and return key can be assigned by typing "%i\n" in the ASCII format string. Here "I" implies for integer and "\n" implies the return signal.

6.2.2 Setup Required in Arduino

The wire used in sending the PPM signal from Arduino to the radio control is same as that of an audio jack wire. The audio jack wire has three terminals; however, two of them are ground and remaining one is the terminal that has a high/low value while compared to the ground voltage value. Therefore, there are actually only two terminals to be dealt with, one being ground and the other one being high or low signal.

The two signals are sent out by the Arduino using the ground pin and a digital pin. Any digital pin can be selected; the pin that was selected was pin number 12. Figure 72 shows the 2.5mm audio pin and Figure 73 shows the pins in the Arduino.

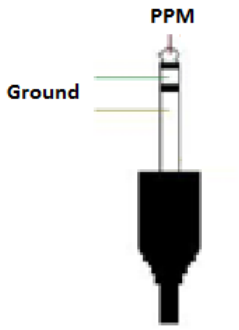


Figure 72: 2.5mm audio pin

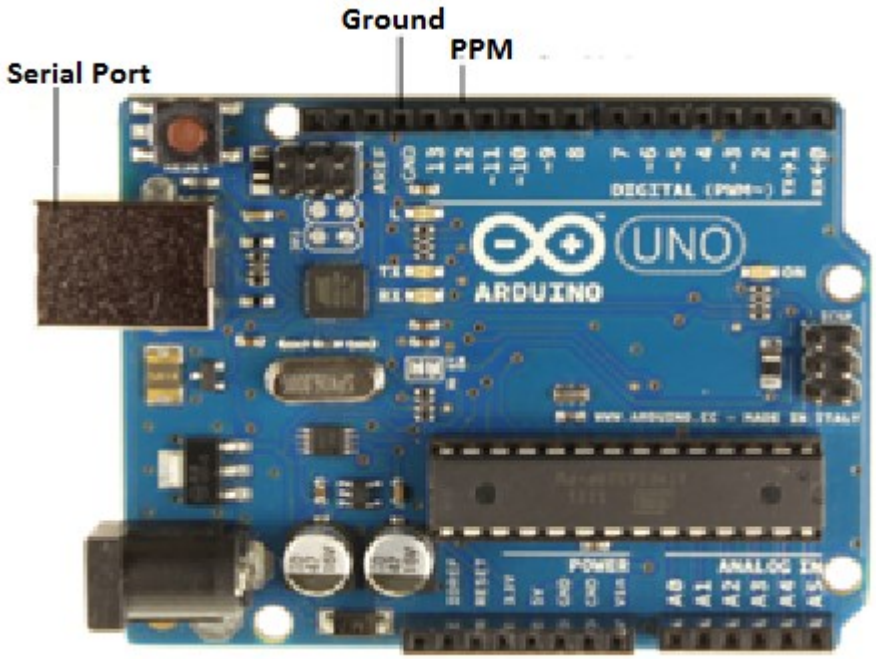


Figure 73: Arduino

Two sets of code were written in the Arduino for two different purposes. The first code named PPM_IN was used to measure the maximum and minimum Time ON of a PWM signal in a one frame of a PPM signal sent by a receiver. The second code which was named PPM_OUT was used to send out PPM signal to the radio control based on the angle inputs from the computer.

The two codes have different hardware setups in regards to the transmitter. For this purpose code 1 will be considered to be used for Setup 1 and code 2 for Setup 2. The PPM_IN and PPM_OUT are described in detail below.

PPM_IN

In order to understand the code one must know the following facts.

- The radio controller used by the test bed has 8 channels. Therefore, a PPM signal should have 8 PWM in one PPM frame.
- Each PWM signal in a PPM signal usually has minimum of 3000 µs Time ON.
- As mentioned earlier the single frame is 20000 µs long.
- After the last channel there's an End Of Frame (or Beginning Of Frame) pause

Details of the software code:

The code reads in the values from port 4. It observes for PPM signal that has a minimum of 3000 µs "Time ON". However, the detected signal cannot be assigned to its respective channel. After the last channel there's an "End Of Frame" (or "Beginning Of Frame") pause, this information can be used to assign each PWM to its respective channel. The last pulse will therefore, have a value either greater than 2000 µs ("End of frame") or less than 100 µs ("Beginning Of Frame"). Once the last pulse is identified, the last pulse can be used to assign values using the equation shown below.

$$Channel\ (i) = \frac{lastpulse\ time\ +\ \ pulse\ time\ (i)}{2} \qquad \textbf{(61)}$$

Finally the values are transmitted to the PC using serial port. The logic behind the code is shown in Figure 74. The complete working code is shown in appendix C1

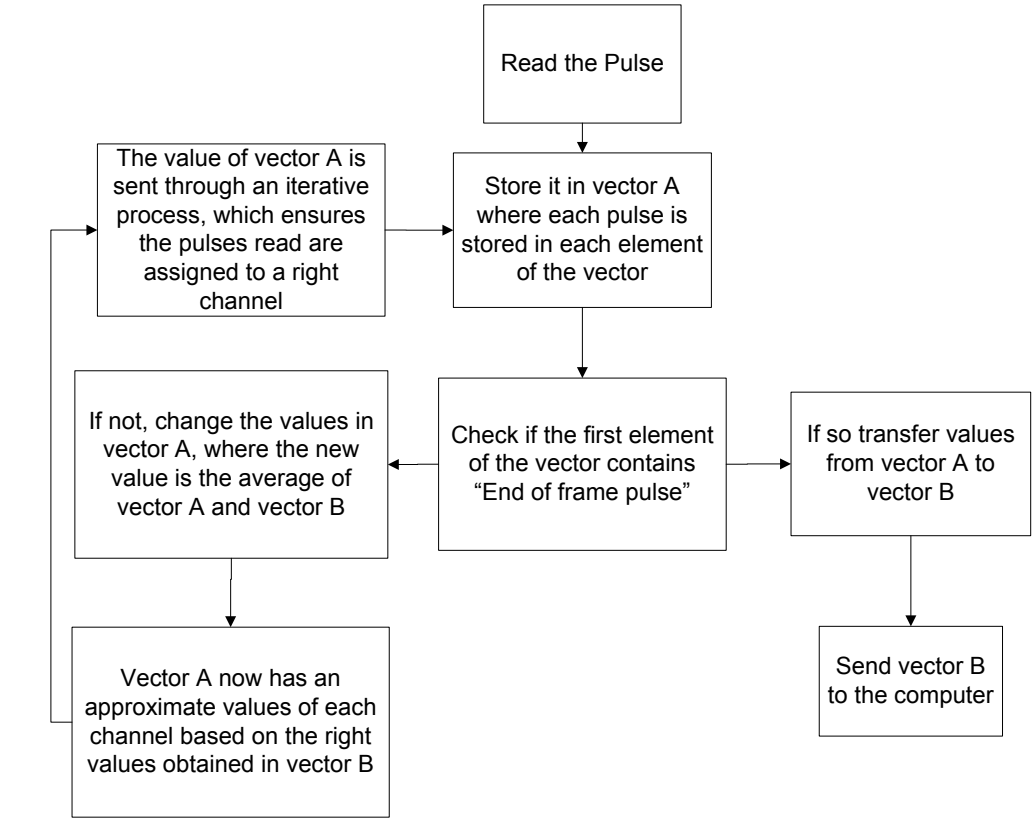


Figure 74: PPM_IN Logic

A screen shot of the output in the serial monitor software, which is inbuilt into the Arduino interface software is shown in Figure 75.

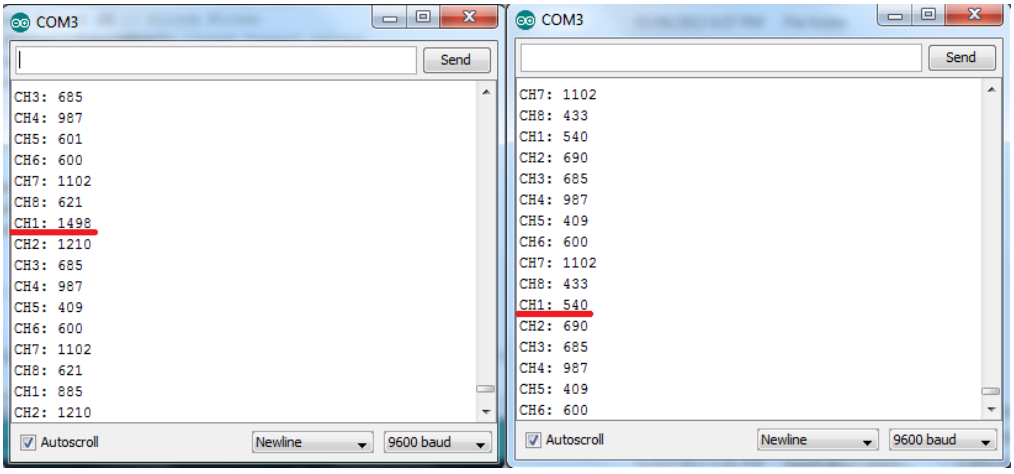


Figure 75: Screenshot from serial monitor of Arduino interface software

After the physical connections between RC, Arduino and the computer had been established, the Arduino is up loaded with the PPM_IN.pde file. This allows the reading of the signals sent by the RC through its trainer port. During this time, the radio control joysticks are moved around to determine the maximum and minimum time “ON/OFF” for a particular radio channel. The signals from the joysticks are encoded in channel 1- 4. The Radio control currently under use was determined to have minimum of 500µs and maximum of 1500 µs Time ON setup. Figure 75, shows the maximum and minimum Time ON observed. Due to the slow clock speed of the processor in the Arduino, the rate at which Arduino measures the “Time ON” and “Time OFF” is not accurate. This leads to only approximate data being collected.

An accurate data can be acquired with the help of an oscilloscope. Here, the oscilloscope is connected to the trainer port. The joystick associated to channel two is changed to neutral, maximum and minimum position to acquire the respective pulse width length. In the Figure 76, which is the graph acquired from an oscilloscope confirms that the maximum pulse width is 1400 µs and the total frame size is 20msec.

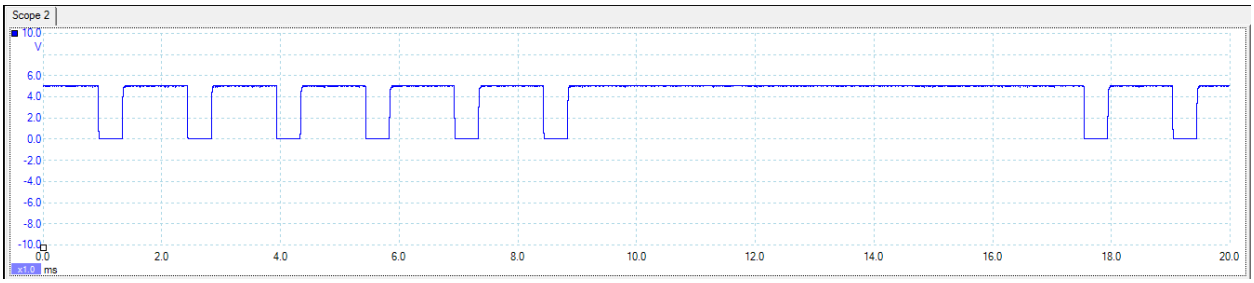


Figure 76: Max pulse width in channel 2

The neutral PPM was found out to be 1080 µs. This can be observed in channel 2 in Figure 77.

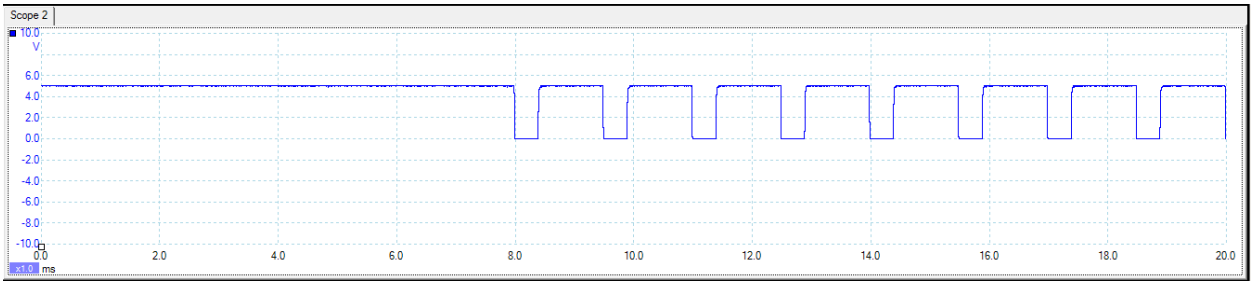


Figure 77: Neutral pulse width in channel 2

The minimum pulse length was found out to be 680 µs. This can be observed in channel 2 in Figure 78.

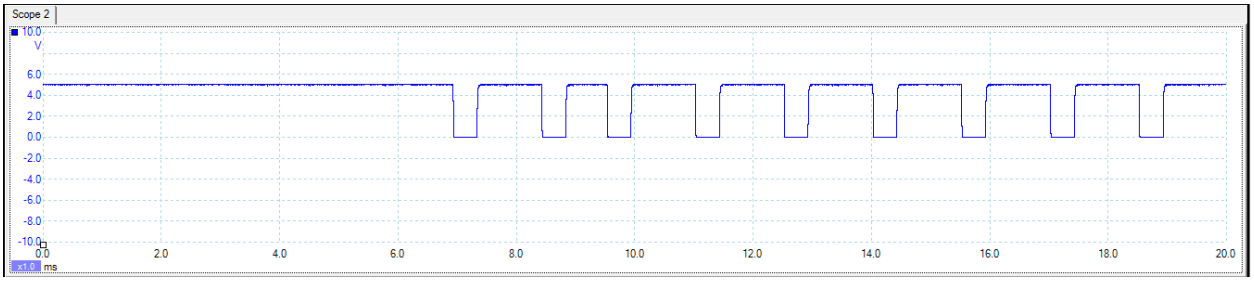


Figure 78: Minimum pulse width in channel 2

PPM_OUT

The PPM out code comprises of two parts. The first part reads the servo angle data sent from the computer in uint8 format and the second part uses the angle information to send the respective PPM signal out of the digital port equipped in the Arduino.

The second part of the code is capable of sending a full PPM signal, however, for test case scenarios; all the servos except one are given neutral deflection signal, w.r.t control surfaces. Therefore, only a single channel will be in use during test procedures. The test procedures will be explained in detail in section 6.3.

Part 1

When data is being transmitted in integer format via serial port, only single digit numbers can be transmitted. The part 1 of the code deals with reading more that single digit numbers.

In serial communication characters/Integers are constantly flowing in, the challenge is to distinguish the stream of numbers into the actual numbers required for servo angle deflection.

This is made sure that the return key symbol '/n' be used at the end of an integer. Simulink outputs the integer followed by '/n' to the serial port based on the configuration described in section 6.2.1. The logic followed by part 1 is shown in Figure 79 .

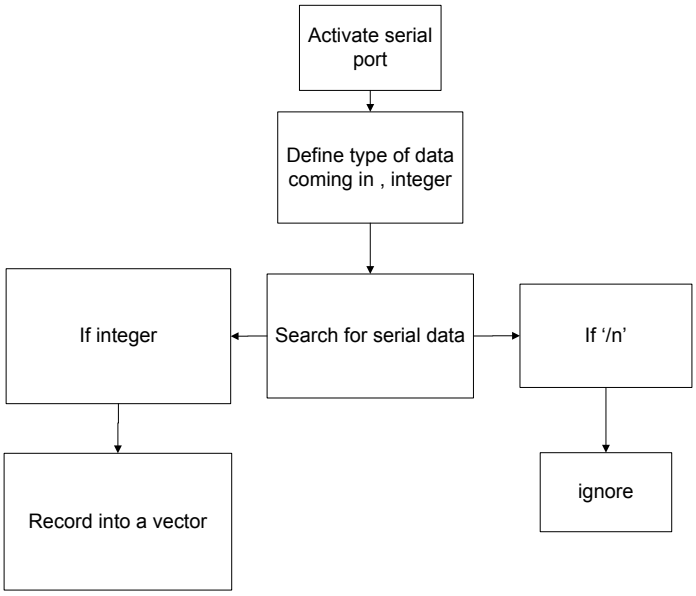


Figure 79: PPM out, part 1 logic

While working with Arduino serial monitor software language by default the '/n' symbolizes end of data. Therefore, the functions in Arduino library as already equipped to handle '/n' without any additional coding. The complete working code is shown in appendix C2

Part 2

The second part is responsible for generating the required PPM signal based on the serial input received. As mentioned earlier, all the channels in the PPM signal are at a constant value except for the channel 2. This is because during the test case scenario the aircraft is required to do only one *"pitch-up"* maneuver and thus needing control over only one set of elevator servos. However, the code developed is capable of providing full PPM signal. Sections of this code are commented out for testing purposes.

The code requires data collected previously either via PPM_IN code and via the PPM_OUT code itself by trial and error. The required data and source are listed below in Table 7.

Table 7: Parameters for the code

Data	Source
Width of start pulse = 300µs	PPM_OUT
Minimum width of a pulse = 680µs	PPM_IN/Oscilloscope
Maximum width of a pulse = 1400 µs	PPM_IN/Oscilloscope
Frame length= 20 ms	PPM_OUT/Oscilloscope (Standard for all transmitters)
Number of channels= 4	PPM_OUT (Transmitter reads only the first 4 channels)

Those with sources PPM_OUT, implies that the numbers were determined by attaining success by trial and error, where success is when a successful PPM signal is output by the microcontroller.

The successful PPM output by the microcontroller can be verified by observing the "monitor" option available in the radio controller. In this option, one can observe the signal being transmitted to the receiver. Figure 80 shows a screen shot of an instance.

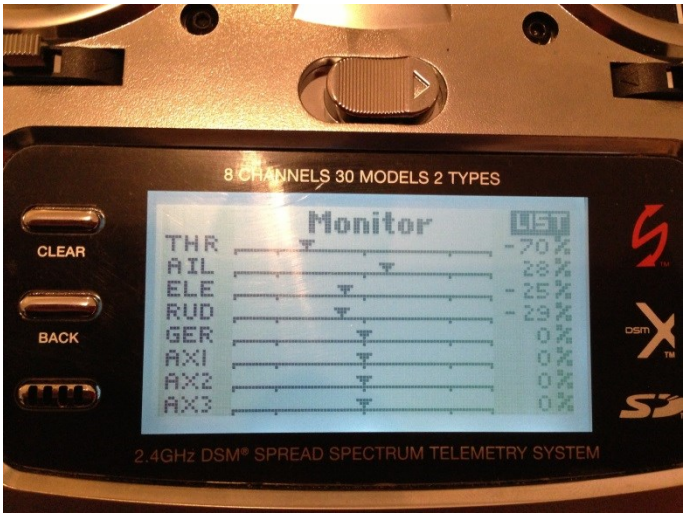


Figure 80: Monitor screen in DX8

The logic behind the code is shown in Figure 81. The complete working code is shown in appendix C2. The relation between angle generated by Simulink and the pulse width is set by the following equations.

$$Conversion\ factor = Highest\ pulse\ width - least\ pulse\ width - starting\ pulse\ width \quad \textbf{(62)}$$

$$Required\ width = (Conversion\ factor \times deflection\ angle) + minimum\ pulse\ width \quad \textbf{(63)}$$

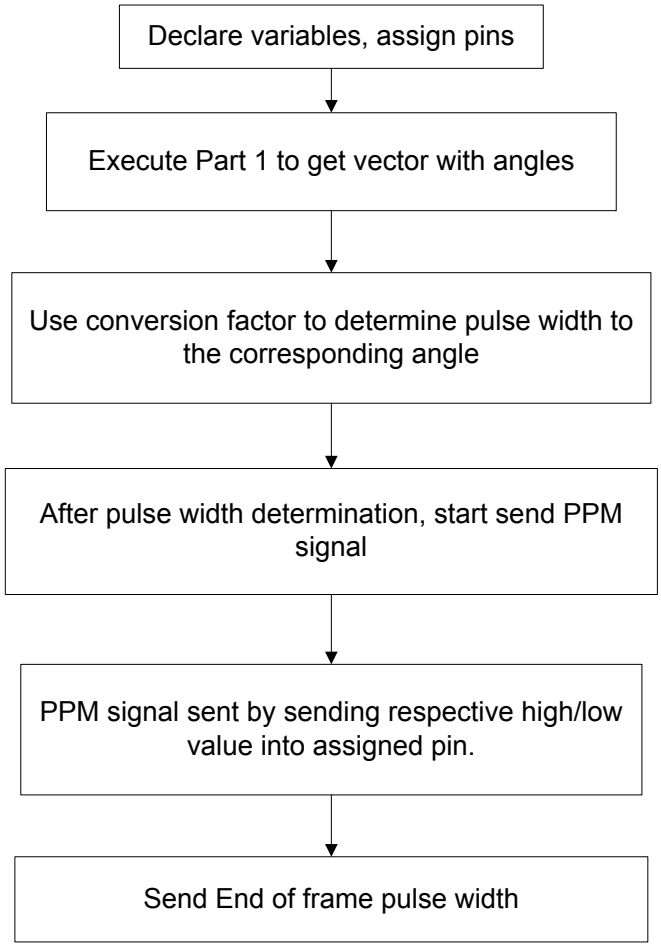


Figure 81: PPM out, part 2 logic

6.3 Testing Procedures

The connectivity setup was tested in two phases. Both phases of the test were performed with the help of the available aircraft test bed. Phase one involves observing the applicable control surface of the aircraft on ground and phase two involves performing maneuvers while the aircraft is inflight. Figure 82 provides a brief description of both phases by using flowchart. The sections below describes the aircraft test bed and both the test phase. Lastly, section 6.4 provides the results obtained from the tests.

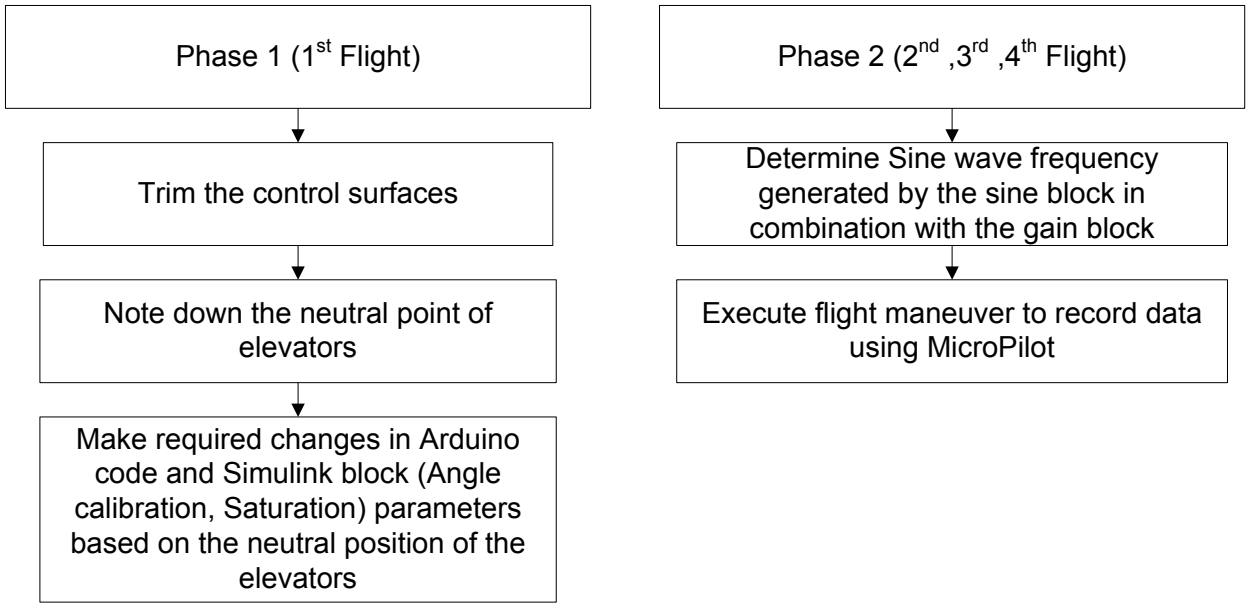


Figure 82: Flow chart of test procedure

6.3.1 Phase One

Phase one involves testing the connectivity setup between the PC and the ATB-10 by observing the control surface on the aircraft. This involves testing of both the Arduino interface and also the Simulink file that sends the required elevation angle to the elevator.

In order to test the connectivity, it was decided that the aircraft should be able to perform particular maneuvers using the connectivity interface. Ideally, performing the obstacle avoidance maneuver by the aircraft is the best case scenario. However, the Simulink model that provides the obstacle avoidance maneuver is sophisticated and hence takes a substantial amount of time to begin execution of the model. The model also runs slowly after execution, making it difficult to modify the model during flight.

It should be noted that ATB-10 has a limited endurance of only 20 minutes due to engine battery constrains. Also not all of the 20 minutes can be used to perform the required

maneuvers based on the location of the aircraft. The aircraft needs enough space to perform the maneuvers which is decided by the pilot. The pilot makes the required decision by checks if there is enough altitude to recover from any trouble caused by the maneuver. Therefore, time is spent to set up the aircraft for each maneuver and several passes may be needed to have a suitable flight path in the judgement of the pilot. It should be noted here that the actual maneuver take around 8 sec to 30 sec. This making 20 min ample time perform test procedures. If more time is required additional batteries are available on ground which can be replaced with the depleted battery. Additional, charging equipment is available on site to charge all the depleted batteries.

Due to the lack of sufficient time during flights to perform the modifications to the existing model, an alternative model needed to be created that could be modified quickly during the 20 minutes endurance window. The new model performs a *"pitch-up"* and down maneuver in a sin wave pattern. The Simulink model is shown in Figure 83.

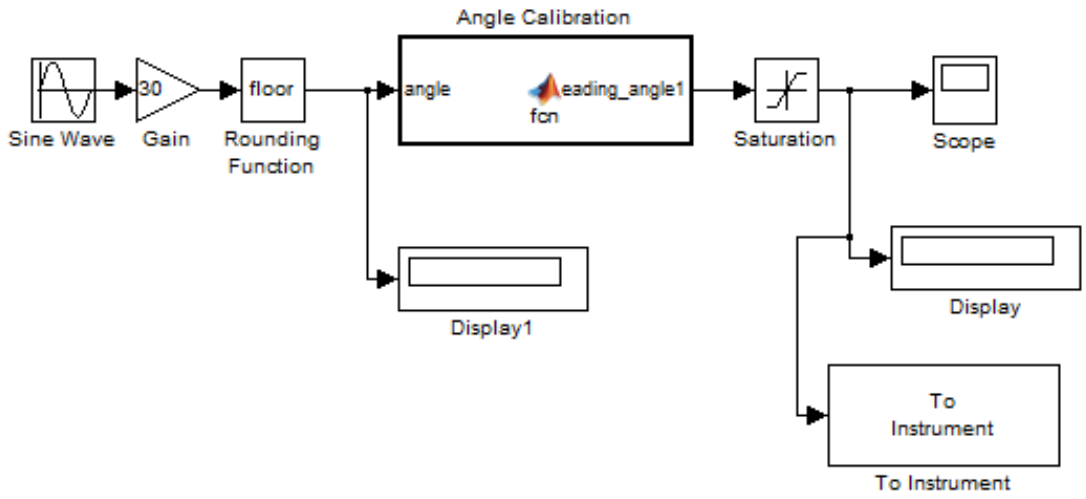


Figure 83: Schematic for Arduino interface

The model outputs a sine wave which is passed through a gain to achieve the required amplitude. The signal is then rounded and passed through a function named angle calibration. The pseudo code of the angle calibration function is shown in the next section. The angle calibration function offsets the values based on the neutral location of the control surface. Based on the default factory setting of the servo, 90° of the servo would imply 0° on the control surface. Likewise +5° deflection angle on the elevators would imply 95° on the servo and -5° on the control surface would imply 85° on the servo. However, the neutral position of the servo would change based on the setup of the servo on the aircraft. The neutral position of the servo would also change based on the centre of gravity of the aircraft, the control surfaces are usually adjusted during flight to set the true neutral position of the control surface which is performed by phase two.

Once the signal is passed through the angle calibration, it is passed through a saturation block which enables output of only a certain range of angles. This makes sure that the aircraft does not perform any extreme maneuvers.

Finally the angles are sent to the Arduino interface using a '*To Instrument*' block. The display block allows examination of the signal.

Once the link is established and the Simulink file is ready for execution, phase one involves testing the link and the Simulink file. This can be done by examining the elevators and check if they move in a sine wave pattern when the Simulink file is executed.

6.3.2 Phase two

Phase two involves testing the link and Simulink file while the ATB-10 is in flight. When the trainer button is activated the aircraft should perform a *"pitch-up"* maneuver followed by pitch down maneuver in a sine wave pattern. Based on the available flight time in a trainer mode, the frequency of the sine requires to be modified so that at least one or more cycle of sine wave can be observed. Also the saturation block should be set based on the maximum and minimum height for the *"pitch-up"* and down maneuver.

The neutral value in the angle calibration function also needs to be set based on the trim values set during the first flight. That is once the trim values are set during the first flight, the neutral angle can be determined on ground. This can be achieved by comparing the neutral elevator location with the trainer mode on and off. While the trainer mode is on the Simulink is made to send an angle close to 90° and while trainer mode is off the elevators move back to neutral location. By a process of iteration the neutral angle can be determined by making sure that the elevators do not move while toggling between the trainer switch. The neutral angle was determined to be 120°.

The acquired neutral angle can be passed into the angle calibration function. The full working code of the angle calibration function is shown below.

If angle sent via Simulink is positive
Then
    servo angle is the angle value +120
Else if the angle sent via Simulink is negative
Then
    servo angle is 120 + the angle value

6.4 Observation and results

This section describes the observation and modifications made during both the phases and also the results acquired during the testing process.

6.4.1 Phase One

During phase one it was found that the speed at which Simulink generates new angle is sometimes not fast enough due to resources being used by background software by the PC. This creates an occasional lag which is definitely much less than a quarter of a second. During the lag between each angle command sent the Arduino considered the input as 0° deflection. Therefore, the elevator's deflection angle rapidly changes 0° which will appear as vibration to one's naked eyes. Thus the transitions between angles were not as smooth at times.

Hence, a new logic was implemented into the Arduino code which ignores a sudden 0° input. This filters out the unwanted 0° read by the Arduino. This was achieved by comparing the consecutive angles received from the serial signal. While comparing the angles, if the difference between the new angle and the old angle is too large, the now angle is ignored. The pseudo code of the logic is shown below. The complete working code is shown in appendix C2.

If absolute value of the difference previous angle and current angle < 20
Then
    Transmit new servo angles
Else
    Transmit old angel

6.4.2 Phase two

During phase two the neutral angle was found out to be 120°. Therefore, when the servo is at 120^{o}, the aircraft flies in a straight line. The elevator deflection angle is considered as 0^{o} when the servo angle is at 120^{o}. This relationship is exploited later to represent all data in terms of elevator angle deflection.

Frequencies of 4 Hz to 11 Hz were attempted. The required frequency was found out to be 10 Hz. Therefore, at 10Hz the aircraft will perform around 3 sine patterned maneuver in the given 8sec window provided by the pilot.

The elevator input generated by the MicroPilot was monitored along with altitude measurements acquired via GPS sensor and pressure sensor. The graphs acquired are shown in the appendix C3. The first two trials were used to determine the correct frequency of the sine wave generated by the Simulink model and also to establish a protocol to implement the right sequence between enabling the trainer switch and the Simulink model.

It took around 7 sec for the Simulink to execute the model that transmits the serial signal, which was determined to be late as the window in which the pilot was ready to enable the trainer switch was missed. Therefore, the model was running at all times during the flight and the pilot enabled the trainer port according to his judgement. This made the aircraft to follow a sine wave path. This sine wave pitch maneuver was observed in the later end of trial 3 and all of trial 4. However, it should be noted that since the Simulink model is running at all times the aircraft caught different part of the sine wave for its starting point. In this new scenario the aircraft performed steep maneuvers initially and gradually followed the expected sine pattern.

Since we are dealing with a small range of elevator deflection angle, the pilot deemed the minor steep maneuvers safe as he was confident that he would be able to recover the aircraft and avoid any crashes.

The data recoded by the MicroPilot was analysed using a data viewer program and plotted as shown in MicroPilot displays altitude in ft*-8 units as shown in Table 8. This data has been converted to feet in Figure 84.

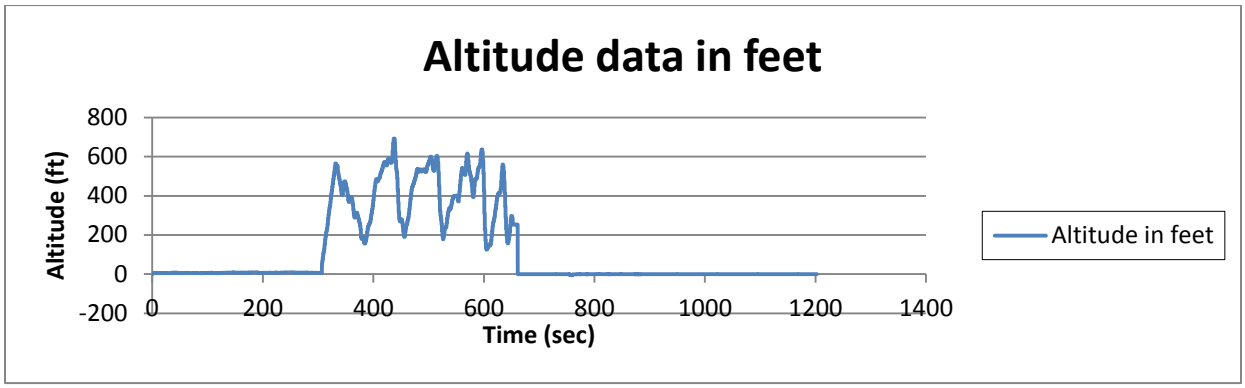


Figure 84: Altitude in feet

The above figure compares the altitude acquired via the on board pressure sensor and the GPS sensor. The required data was stored in the standard telemetry file.

The standard telemetry file is created when the link between horizon and the MicroPilot is terminated. The telemetry file is a text file containing data gathered during period of successful telemetry link. The format of the file is shown in Table 8: Data field ID [MicroPilot, 2006]Table 8. The rate at which the data is recorded is dependent on the speed of the data link. These tables allowed successful extraction of data into Microsoft Excel.

Table 8: Data field ID [MicroPilot, 2006]

column	data
1	time (hh:mm:ss) from the PC's clock
2	GPS position East in degrees and decimal degrees
3	GPS position North in degrees and decimal degrees
4	Pitch in radians times 1024
5	Roll in radians times 1024
6	airspeed in feet per second
7	gps speed in feet per second
8	altitude in feet*-8
9	rate of change of altitude in feet*-8 per second
10	heading in degrees times 100
11	error field that contains error indicator
12	status bitfield
13	main battery voltage in volts times 100
14	servo battery voltage in volts times 100
15	throttle postion
16	status2 bitfield

Figure 85 shows the successful execution of the desired flight path. For example, during the time period between 360sec and 425sec, the sine wave pattern generated is due to the commands sent by Simulink to the aircraft through Arduino board. Figure 85 shows pilot's and Simulink's contribution to altitude. The arrow shows the sine pattern generated by Simulink. The distortion of sine wave can be explained by the pilot's control over the ailerons. The pilot was allowed to provide stability to the aircraft using the remaining control surfaces while the trainer port was activated according to his discretion.

It should be noted that the data being analyzed in this section is acquired from the 4th trial. The data acquired from the previous trial can be found in appendix C1. The data from the 4th trial was examined in detail mainly due to the fact that the required frequency was established only in the 4th trial.

At the end of the flight test, the pilot lost all communication between the radio control and the aircraft due to an unexpected shutdown of MicroPilot which led to a crash. The data gathered

after crash showed fluctuation of voltage levels while the Pilot was performing steep maneuver. The voltage fluctuations lead to rebooting of the MicroPilot, which is the reason for losing control of the aircraft. The aircraft have to undergo some minor repairs before it is flight worthy again. This provides another reason to focus more on the 4th trial.

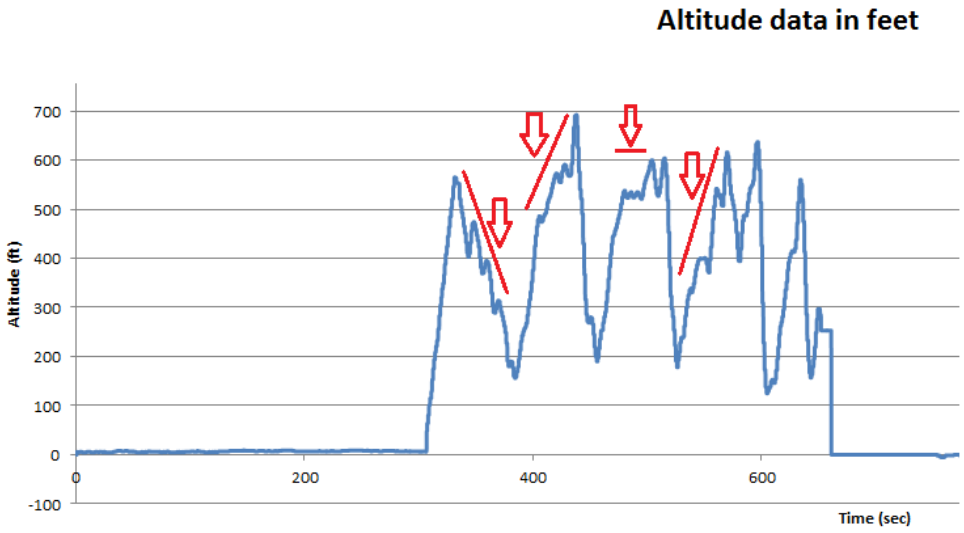


Figure 85: Trial 4 altitude data (Simulink contribution highlighted)

Figure 86 is a screen shot from the MicroPilot log viewer. It shows the commands sent to the elevator servos. The data is recorded in a separate log file named the sensor log file. The sensor log file is created via horizon. The contents of the sensor log file are determined by the user while enabling the log file. The recording of log file can be enabled or disabled at any point after the MicroPilot is armed.

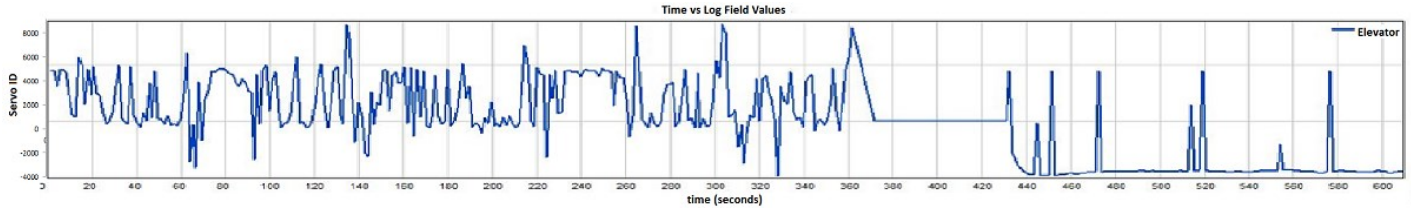


Figure 86: Elevator data for trial 4

In regards to this research, the log file was enabled once the aircraft is airborne and ready to execute the required maneuvers. Since the Simulink controlled the elevator servos, the log file was made to record the elevator servo commands. The MicroPilot records servo angle in Servo ID unit. This unit ranges from +32,767 to -32,767. Since the servo has a total arrange of 180^{o} the following relation holds true.

$$1^{o} = 364 \text{ servo ID} \quad \textbf{(64)}$$

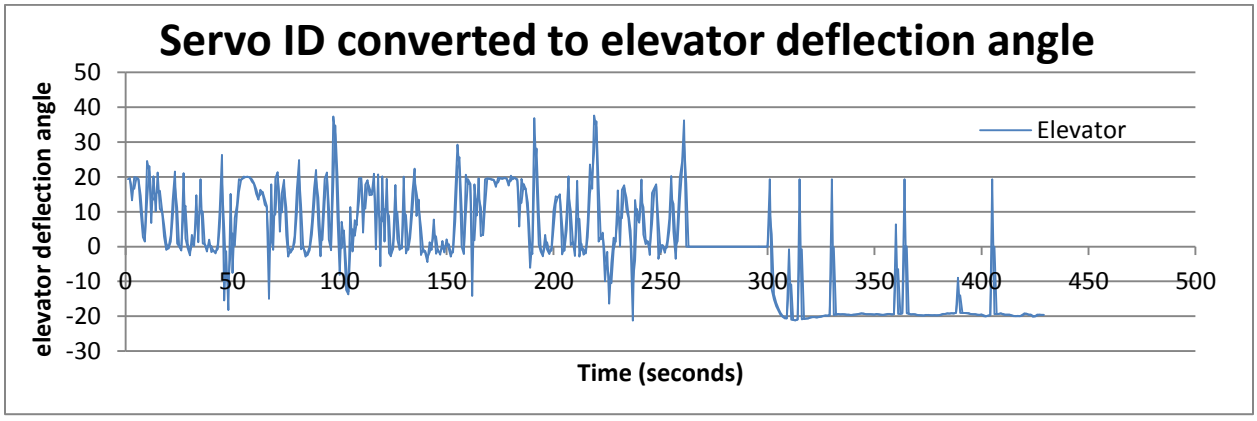


Figure 87: Log file (servo id converted to elevator deflection angle)

A portion of data (83sec - 100sec) was extracted from the data log file and re-plotted as shown in Figure 88. The sensor log files records around half the number of data points per second when compared to that of the telemetry log file. Therefore, it should be noted that the data resolution is low.

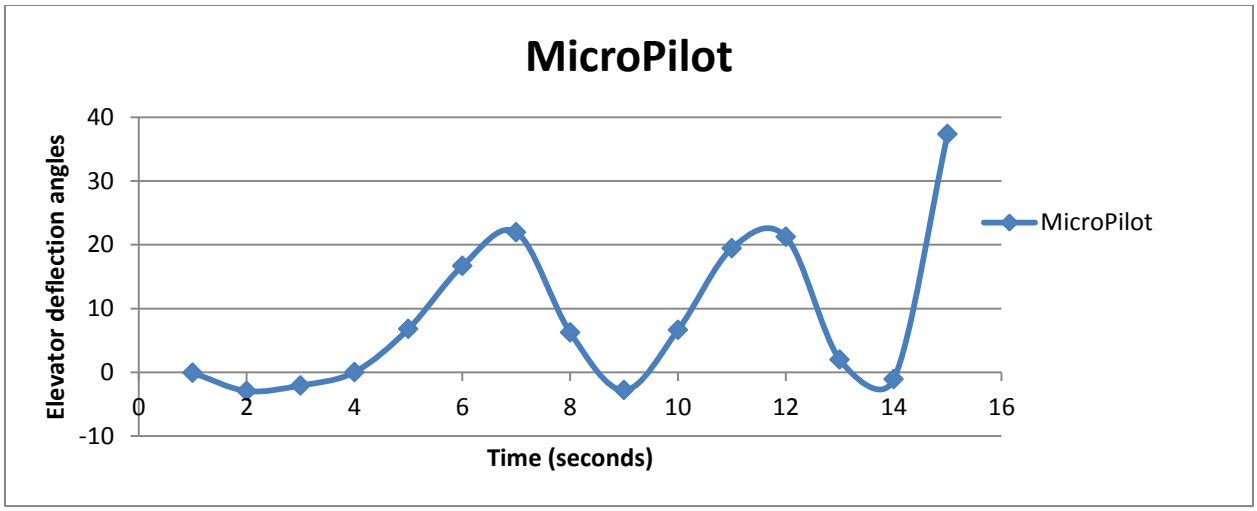


Figure 88: Specific altitude data from trial 4

During the last trial, it was decided that the angles sent by Arduino need to be recorded. The angles were recorded into a vector immediately after it read from the serial port. Due to the lack of memory in the Arduino not all the values that were read can be stored onboard. Hence, a few select angles were recorded instead. The Arduino was programed to compare the angles, and if there was more than a 10^{o} change between the previously recorded angle and the present angle, it was then recorded. Hence, around 10 data points were eliminated per second. This data was plotted to acquire the graph shown in Figure 89.

However, it was after the final trial that a discrepancy on the sine pattern as seen in Figure 89. This discrepancy is due to the same error detected in phase one of the testing (section 6.4.1). In phase one it was shown that the intermittent lag of Simulink processing produced false value (0^{o}) during the lag. Although this issue was fixed in phase one, the issue was overlooked while recording the data during the last trial. The receiving of false value lead to generating additional sine waves. The false 0^{o} has been circled in Figure 89. In order to compare Figure 88 and Figure

89, the data recorded by the Arduino needs to be corrected manually. This was done by replacing the 0^{o} angles with the ideal angles from the log file.

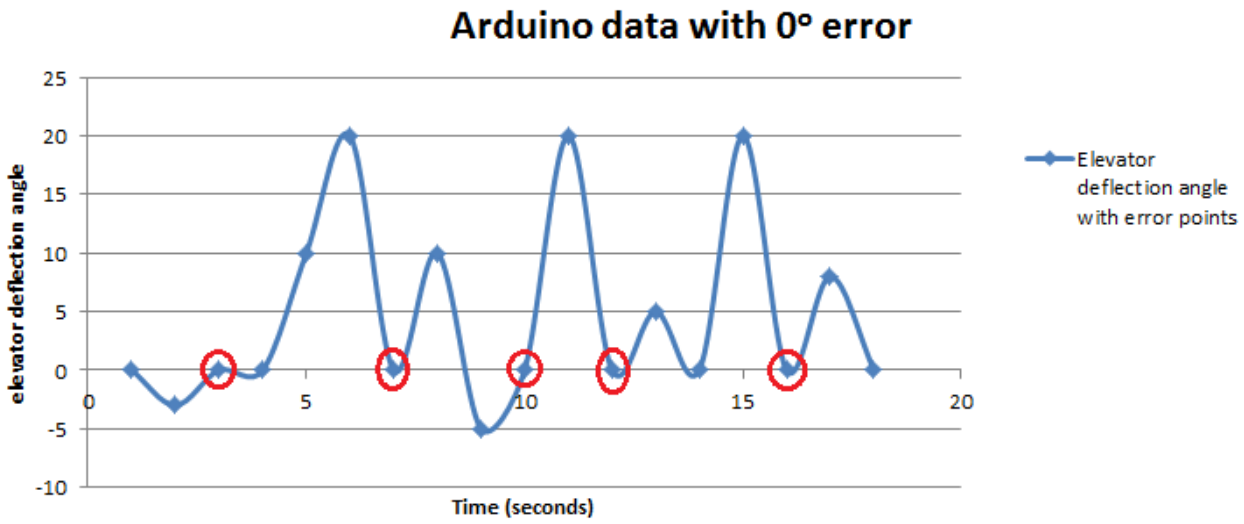


Figure 89: Arduino data with 0^{o} error

Once the data was Arduino data was modified, the modified data, the non-modified data and the MicroPilot data were all plotted into one graph. This graph is shown in Figure 90.

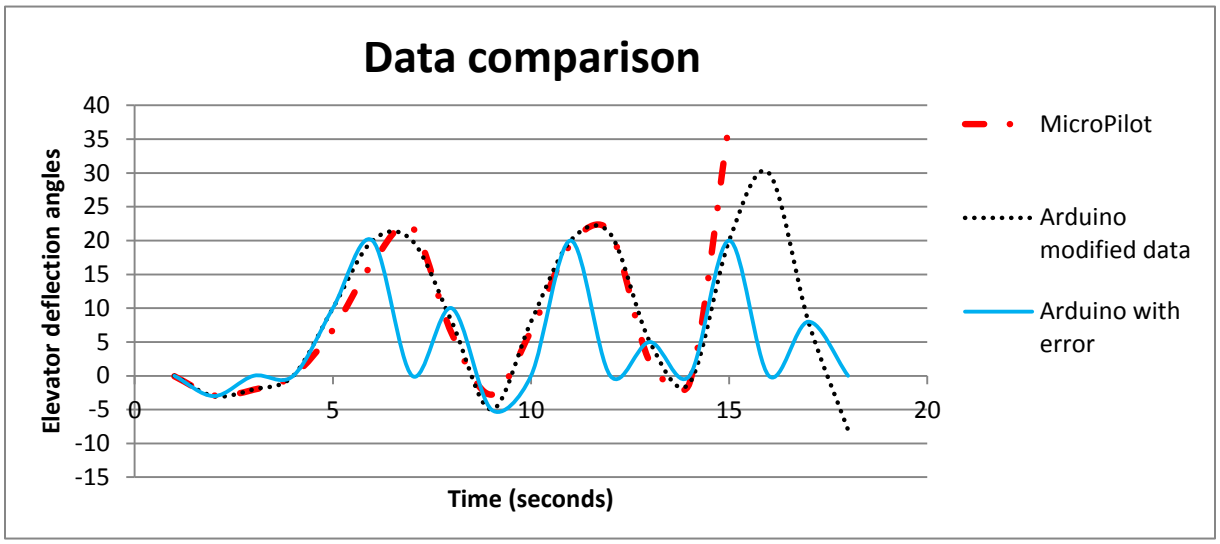


Figure 90: Data comparison

Figure 90 shows that the transmitted data and the received data are almost the same. Where the transmitted data is the modified Arduino data and the received data is the MicroPilot data. The correlation coefficient between these signals was found with the help of the correlation function available in Microsoft excel. The correlation coefficient was found out to be 0.871. It should be noted that excel uses the following equation to calculate the coefficient.

$$Correlation\ coefficient\ = \frac{Covariance(x,y)}{\sqrt{Variance\ (x)} \times \sqrt{Variance\ (y)}} \quad \textbf{(65)}$$

Where,

Covariance(x,y) is the sample covariance between x and y:

$$Covariance(x,y) = \frac{1}{n-1} \times \sum_i (x_i\ - \bar{x}\ )\ (y_i\ - \bar{y}\ ) \quad \textbf{(66)}$$

Variance(x) is the sample variance of x:

$$Variance(x)\ = \frac{1}{n-1} \times \sum_i (x_i\ - \bar{x}\ )^2 \quad \textbf{(67)}$$

Variance(y) is the sample variance of y:

$$Variance(y) = \frac{1}{n-1} \times \sum_i (y_i\ - \bar{y}\ )^2 \quad \textbf{(68)}$$

A correlation coefficient of 1 demonstrates a total positive correlation, 0 demonstrates no correlation and -1 demonstrates a negative correlation [Stigler, 1989] [Rodgers, 1998].

Since the commands are sent from Simulink to avoid the obstacle, the angles are sent to the elevator in advance due to the fact that obstacle avoidance algorithm considers a flight envelop which includes a buffer distance for the aircraft to perform a maneuver safely. The aircraft

could avoid the obstacle successfully as the delay in the command would be delayed within 5-10 milliseconds range. Hence, the lack of a 1 correlation coefficient is acceptable.

Lastly, the GPS coordinates were plotted using Google Earth with the help of Australian Transport Safety Bureau (ATSB) KML Creator. The KML format is a file that provides the required data to Google Earth. The procedure to extract data from the standard telemetry log file and using it in Google Earth with the help of ATSB KML Creator is described in the appendix C4.

Figure 91 shows the total flight path followed by the ATB-10 during its 4th trial. The figure also shows the elevation profile graph at the bottom.

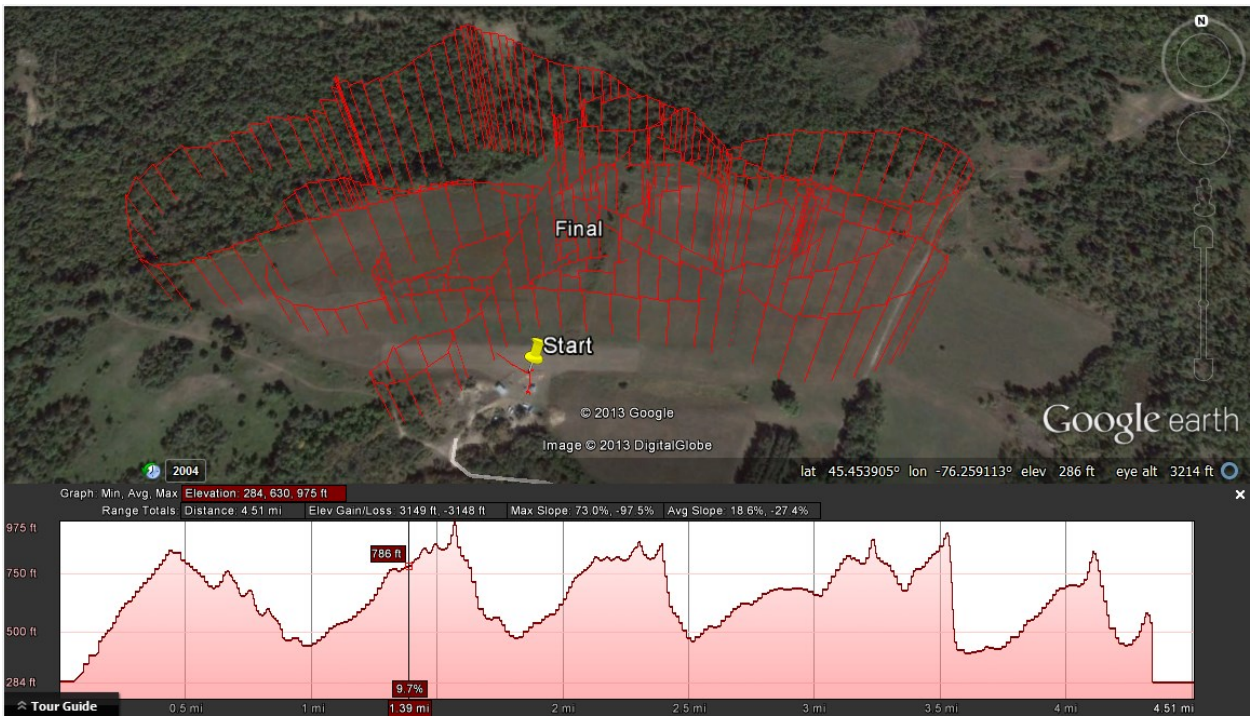


Figure 91: Full Flight Path in trial 4

Figure 92 shows an isolated part of the total flight path in order to clearly represent the section of interest. The pitching maneuvers shown visible in the selected isolated path are performed by Simulink via the Arduino and not by the pilot.

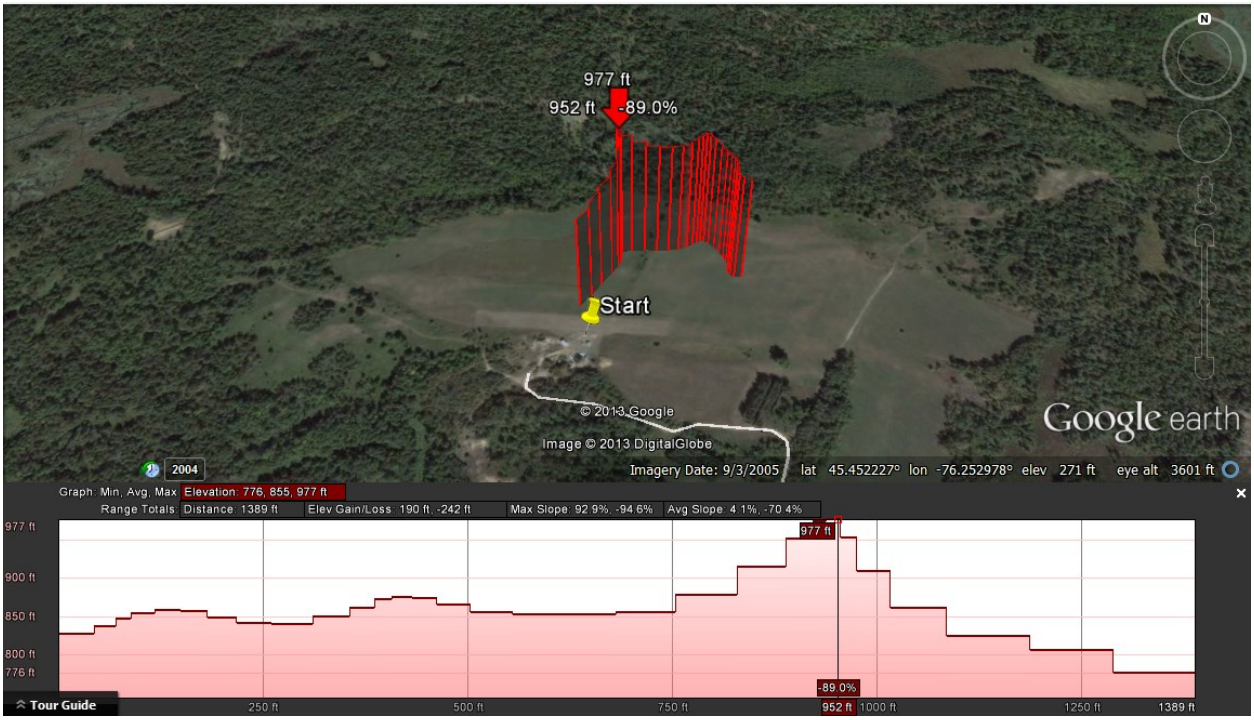


Figure 92: Selective flight path executed by Simulink

By observing the elevation profile in Figure 92, one can observe two periods of the sine wave pattern maneuver performed by the ATB-10 within in the shown 152m (500ft) of the distance traveled.

7.0 Conclusions

In this research project a simulation framework was developed that can be used not only for the Unmanned Aircraft Systems Technologies NSERC CRD Project, but also for any other fixed wing UAV activity. Partial modification of the non-linear model from the Aerosim Blockset and incorporation of other various other models was also demonstrated. The data generated by the non-liner model was successfully transmitted to the feedback loop, obstacle avoidance and detection, VRML environment and the Arduino connectivity blocks.

The feedback loop and the obstacle avoidance and detection block were modified from existing models created by Owlia [Owlia, 2013]. The modification done was to incorporate more than one possible flight path to avoid obstacles. The modified model handles both the pitch-up maneuver and fly around maneuver.

The VRML block developed in this research, provide real-time simulated video data to the obstacle avoidance and detection block. Various versions of this block were created for testing purposes. A variation of the block was also incorporated to the available UAV model itself. The testing of the model was done by comparing the results produced by various algorithms. Algorithms results for the video data produced using the Kinect Device and the VRML model were compared to establish the viability of the created VRML environment. Three algorithms were created for this purpose, namely, Connected Components, 2D Map and Reprojection. These algorithms proved that the created VRML block was viable.

The connected components algorithm demonstrated that the video acquired by the VRML environment is comparable to that of a real world video data. However, the video data

acquired by VRML environment can be considered as “too perfect”, hence injecting noise into the video could be used to mimic the quality of real world video collected by cameras on actual UAVs.

The 2D map algorithm demonstrated that the speed of running the simulation affects the resolution of the video data. For the experiment performed with the Kinect, the simulation had to run for 55sec to provide the same data which was acquired in 7sec. Thus, the simulation time needs to be calibrated based on researcher’s requirements.

The reprojection algorithm once again demonstrated that the video data is adequate for image processing. It also gave an insight on the level of programming required to reconstruct a 2D data to 3D. The complexity of the developed reprojection algorithm was considered to be adequate based on the mission requirement of GeoSurv II. Thus, the usage of the reprojection algorithm to demonstrate the adequacy of the video obtained by the simulation environment is justified.

Lastly, the Arduino Connectivity block was created to establish a communication link between the Simulink model and the ATB-10. Several flight tests were performed to test this newly developed system. The results after the flight test demonstrated a successful communication between the Simulink model and the ATB-10. It should be noted that the developed model currently can handle only one control surface (elevator). However, instructions are given for the future development of fully functional model.

7.1 Future Work

This section outlines the possible future developments that could be attempted and implemented, based on the work done in the present project in three key areas.

7.1.1 Simulation Framework

Currently the simulation framework is lacking the required stability and control derivatives of both the ATB-10 and the GeoSurv. Based on the literature review, one could estimate that two or more additional students would be needed to develop the Longitudinal and Lateral derivatives of these two aircraft. Under the same proposed research, one would need to familiarise oneself with new framework described in this thesis, especially with the PID controllers to establish stable flight with the new stability and control derivatives.

7.1.2 3D Environment

The current 3D environment created via VRML although serves the purpose of producing viable simulated video camera outputs, the quality of the 3D environment and its video output can be significantly increased by using a real-time 3D graphics engine such as UNIGINE. For example, the UNIGINE engine is able to produce real-time 3D environment shown in Figure 93.

In order to use a 3D engine such as this, one should check the compatibility with Simulink and ensure sufficient processing capabilities on the selected computer. For example, the UNIGINE requires a minimum of 6-core processor running at 3.0 Ghz along with a superior graphics card [UNIGINE, 2013]. Considering the fact that, Simulink and image processing occurs simultaneously, one should expect to use a more powerful computer processor, more likely a 12-core processor.

Figure 93: Image from video rendered by UNIGINE [UNIGINE, 2013]

7.1.3 Serial to PPM Conversion (STPC)

Under this topic, there is less work to be done. The STPC requires to consideration of all four channels (throttle, ailerons, elevator and rudder), not just the elevators. The PPM_OUT program is already capable of transmitting a four Channel data; however, it is not able to read four channel data input from the serial port from Simulink. Also the *"To Instrument"* block is currently not configured to transmit a four channel data to the serial port.

Hence, one should be able to develop a fully functional STPC by updating the Arduino program and the *"To Instrument"* block in the simulation framework.

References

Adiprawita, W. *et al.* 2007. *"Hardware In The Loop Simulator in UAV Rapid Development Life Cycle",* International Conference on Intelligent Unmanned System (ICIUS), Paper No. ICIUS2007-A006

Aerosim blockset, *"Aerosim User's Guide"*, [Manual], Ver. 1.01, Unmanned Dynamics LLC..

Austin, R. 2010. "*Introduction to Unmanned Aircraft Systems (UAS)"* in Unmanned Aircraft Systems, 1st ed., United Kingdom: WILEY, pp. 1-15.

Avayan, Last visited Dec 2013, *"Understanding PWM",* http://ebldc.com/?p=48

Bradley, D 2010. "*Binocular Camera Calibration Using Rectification Error*", Seventh Canadian Conference on Computer and Robot Vision, pp 183-190.

Bradski, G. *et al*. 2008. "*Learning OpenCV: Computer vision with the OpenCV library"*. O'Reilly Media, Incorporated,

Butenko *et al.* 2003. "*Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments*" in "Cooperative Control: Models, Applications and Algorithms", ch 6, pp 96.

Caron M. *et al* 2013. Aeromagnetic surveying using a simulated unmanned aircraft system. Accepted for publication in *Geophysical Prospecting*.

Chao, Y. *et al*. 2013. "*Design of UAV Simulator Based on MAN-IN-LOOP Simulation Platform"* International Journal of Science, Environment and Technology, Vol. 2, No 3, 449 – 456

David M., "*RE: Internship Positions*" Personal communication (December 2012).

DX8. Last visited April 17th 2013, "DX8", http://www.spektrumrc.com/Products/Default.aspx?ProdId=SPM8800.

Fong, S. Y 2009. *Longitudinal aerodynamic performance of an aircraft model, project* report, University Teknikal Malaysia Melaka.

Forrester, R. W. 2011. *Magnetic signature control strategies for an unmanned aircraft system.* MASc thesis, Department of Mechanical and Aerospace Engineering, Carleton University.

Gawron V.J 1998, "Human factors issues in the development, evaluation, and operation of uninhabited aerial vehicles". AUVSI '98: Proceedings of the Association for Unmanned Vehicle Systems International, Huntsville, AL, 431-438.

Gerstmann, P. 2000, "*Building games in VRML*", M.A.Sc Thesis, The Ohio State University.

Geyer, M. S. et al. 2006, *"3D Obstacle Avoidance in Adversarial Environments for Unmanned Aerial Vehicles*."

Goppert, J. M. 2012. *An adaptable, low cost test-bed for unmanned vehicle systems research,* MSAA thesis, Department of Aeronautics and Astronautics, Purdue University.

Griffiths, S *et al*. 2007,"*Obstacle and terrain avoidance for miniature aerial vehicles*", Advances in Unmanned Aerial Vehicles: 213-244.

Hardesty, L. Last visited November 5th 2013, "*Automatic building mapping could help emergency responders*", MIT News, http://www.mit.edu/newsoffice/2012/automatic-building-mapping-0924.html.

Hojeij, H. 2013. Personal communication, MASC candidate, Department of Mechanical and Aerospace Engineering, Carleton University.

Hostmark, J. B. 2007. *"Modelling Simulation and Control of Fixed-wing UAV:CyberSwan"*, M.SC Thesis, Norwegian University of Science and Technology.

Hrabar S. 2011. "*An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance*", Journal of Field Robotics.

Ince, E. A. 2003. "*Pulse Time Modulation*", Class lecture, Dept. of Electrical and Electronics Eng., Eastern Mediterranean University.

ING. Last visited March 2013, "*Services*", http://www.ingrobotic.com/services/.

Jenie, S. D. et al 2006. "*Automatic Flight Control System*", Class lecture, Dept. of Aeronautics and Astronautics, Malaysian Institute of Aviation Technology, Malaysia.

Kai-bo, B. *et al*. 2009. , "*Flying navigation, controls and MATLAB simulation Technology*",

Kang, Y. 2011. "*A robust lane recognition technique for vision-based navigation with a multiple clue-based filtration algorithm",* International Journal of Control, Automation and Systems, Volume 9, Issue 2, pp 348-357.

Khatib, O. 1986, "*Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*" The International Journal of Robotics Research 5(1), 90-98. ISSN0278-3649.

Kheng, L. W. 2012, *"Camera Models and Imaging"* Class lecture ,*CS4243*, National University of Singapore

Khosla, P. *et al*. 1998, **"***Superquadric artificial potentials for obstacle avoidance and approach*" in "Proceedings. 1988 IEEE International Conference on Robotics and Automation" pages 1778-1784. IEEE Comput. Soc. Press.

Kim, J.-O. *et al*. 2011, "*Real-time obstacle avoidance using harmonic potential functions*", IEEE Transactions on Robotics and Automation 8(3), 338-349.

Kinect. Last visited June 2nd 2013, "*Kinect*", http://www.xbox.com/en-US/kinect.

Kinect-2. Last visited November 10th 2013, "*How the Kinect works",* Depth Biomechanics, http://www.depthbiomechanics.co.uk/?p=100.

Kunzwa, L. F. 2011. "*UAV Modeling Simulation Stability and Control", Project* Report, School of Engineering and Technology, University of Hertfordshire.

Lares, A. 2012**.** *Insert Design and Manufacturing for Foam-Core Composite Sandwich Structures.* MASc thesis. Department of Mechanical and Aerospace Engineering, Carleton University.

LaValle, S. M 2006**. "***Planning Algorithms*", Cambridge University Press. ISBN9780521862059**.**

LaValle, S. M. 1998**, "***Rapidly-exploring random trees: A new tool for path planning***"** Technical report, Department of computer Science, Iowa State University, Ames, IA, USA.

Maley, J. A. 2008**.** *An investigation into low-cost manufacturing of carbon epoxy composites and a novel "mouldless" technique using the vacuum assisted resin transfer moulding (VARTM) method.* MASc thesis, Department of Mechanical and Aerospace Engineering,

Carleton University. (OCE)

Matt W. et al. 2006*, "Cooperative tracking of moving targets by a team of autonomous UAVs",* IEEE Xplore *1-4244-0378-2/06/*

Mathworks . Last visited *October 2013, "DC Motor Model",* http://www.mathworks.com/help/physmod/elec/ug/example--modeling-a-dc-motor.html

Meer, P. 2013. "*Stereo Vision*", Class Lecture, Faculty of Electrical and Computer Engineering, Rutgers University, New Jersey, USA.

Members. Last visited September 2013, "*Fostering Success in Unmanned Vehicle system*", http://www.unmannedsystems.ca/content.php?doc=112, January 2013.

MicroPilot 2006. "*MicroPilot Autopilot Installation & Operation"*, [Manual], Manitoba: MicroPilot Inc..

Microsoft. Last visited *October 7th 2013,* "*Kinect for Windows features", http://www.microsoft.com/en-us/kinectforwindows/discover/features.aspx.*

Movellan, J. R. *et al.* 2010, *"DC Motors",* Tutorial, Machine Perception Laboratary

Mueller, E. R. 2007."*Hardware-in-the-loop Simulation Design for Evaluation of Unmanned Aerial Vehicle Control Systems*", AIAA Modeling and Simulation Technologies Conference and Exhibit, South Carolina, AIAA 2007-6569.

Mujumdar, A. *et al*. 2011 **"***Evolving philosophies on autonomous obstacle/collision avoidance of unmanned aerial vehicles*." Journal of Aerospace Computing, Information, and Communication 8, 17-41.

Nano n920-ENC. Last visited April 17th 2013, "*Nano n920-ENC*", http://www.microhardcorp.com/n920-ENC.php.

Park, C. S *et al*. "*A robust stereo disparity estimation using adaptive window search and dynamic programming search*." Pattern Recognition, 2000.

Peck, M. Last visited November 5th 2013, "Microsoft Wants to Kinect with Pentagon**",** http://www.defensenews.com/article/20121217/TSJ01/312170003/Microsoft-Wants-Kinect-Pentagon.

Perhinschi, M. G *et al*. 2010. *"Integrated Simulation Environment for Unmanned Autonomous Systems—Towards a Conceptual Framework,"* Modelling and Simulation in Engineering, vol. 2010, Article ID 736201, doi:10.1155/2010/736201.

Pinhole camera model. Last visited November 12th 2013, "*Multi-view video coding"* http://www.epixea.com/research/multi-view-coding-thesisse8.html

Polowick, C. 2013. *Optimizing Vacuum Assisted Resin Transfer Moulding (VARTM) Processing Parameters to Improve Part Quality*, MASc thesis, Department of Mechanical and Aerospace Engineering, Carleton University.

Pushypanda. Last visited November 11th 2013, "*HOW IMAGES ARE OBTAINED*", http://pushypanda.blogspot.ca/2010/11/how-images-are-obtained.html.

Quebec Economic Development Program. Last visited July 2013, "*Government of Canada funds development of Unmanned Aerial System Center of Excellence*", http://nouvelles.gc.ca/web/article-eng.do?nid=743129 ,17th may 2013.

Redding, J. A. G. *et al*. 2010. "*An intelligent cooperative control architecture*," in American Control Conference.

Richalet, J. 1993 "*Industrial applications of model based predictive control*", Automatica 29(5), 1251-1274.

Rodgers, J. L. *et al*. 1988. "*Thirteen ways to look at the correlation coefficient*", The American Statistician, 42(1):59–66.

Ryu, D. *et al*. 2011**,** *"Multiple intensity differentiation for 3-d surface reconstruction with mono-vision infrared proximity array sensor,"* IEEE Sensors Journal, vol. 11, no. 12, pp. 3352 –3358.

Sanabria, J. 2013, *"Kinect Architecture The Microsoft Kinect Camera and its use for Architectural 3D Modeling",* Honors Thesis, John Nevins Andrews Scholars Andrews University Honors Program.

Saunders, J. *et al*. 2005, "*Static and dynamic obstacle avoidance in miniature air vehicles*", In Proc. Infotech@ Aerospace Conf.

Scherer S. *et al*. 2007. "*Flying fast and low among obstacles*" In Robotics and Automation, 2007 IEEE International Conference on, pp. 2023-2029.

Senior Telemaster. Last visited April 14th 2013, "*SeniorTelemaster*", http://www.rcuniverse.com/magazine/pdfconvert.cfm?id=708.

Shanmugavel, M. *et al*. 2011, *"Cooperative path planning of unmanned aerial vehicles*", American Institute of Aeronautics and Astronautics; Wiley, West Sussex, U.K. ISBN 9781600867798.

Stanley, D. 2013. *"Measuring Attention using Microsoft Kinect:",* Masters Thesis, Department of Computer Science, College of Computing and Information Sciences Rochester Institute of Technology.

Stevens, B. L. *et al*. 2003. "Aircraft Control and Simulation", John Wiley and Sons Inc..

Stigler, S. M. 1989. "*Francis Galton's Account of the Invention of Correlation*" in Statistical Science, Volume 4, Number 2, pp 73–79.

Szeliski, R. *et al*. Last visited May 15th 2013, "*Symmetric Sub-pixel Stereo Matching*", http://research.microsoft.com/pubs/75689/Szeliski-ECCV02.pdf.

Tan, Kian-Huat *et al*. 1999, "Understanding Machine Operations and Manufacturing Using VRML", Am. Soc. of Engineering Educ, (ASEE) 1999 Conf., June 20 - 23, 1999 .

Tanz, Jason. Last visited June 2nd 2013, "Kinect Hackers Are Changing the Future of Robotics", http://www.wired.com/magazine/2011/06/mf_kinect/.

Transport Canada. Last visited September 7th 2013, "Unmanned Air Vehicle (UAV)", http://www.tc.gc.ca/eng/civilaviation/standards/general-recavi-brochures-uav-2270.htm 3rd May 2010.

Transport Canada-2. Last visited September 7th 2013, "*UAV Working Group*", http://www.tc.gc.ca/eng/civilaviation/standards/general-recavi-uavworkinggroup-2266.htm, 29th February 2012.

Tsukuba. Last visited May 15th 2013,"*Middlebury Stereo Evaluation - Version 2*", http://vision.middlebury.edu/stereo/eval/#references, June 2 2009.

Turnigy. Last visited November 2013, "*Turnigy - C2830-1050*", http://www.flybrushless.com/motor/view/462.

UNIGINE. Last visited November 2013, "UNIGINE Engine", http://unigine.com/products/unigine/.

Veksler, O 2005. "*Stereo Correspondence by Dynamic Programming on a Tree*", IEEE Computer Society Conference on Computer Vision and Pattern Recognition Volume II, pp. 384-390.

Verbickas, R. 2012. *Convolutional Neural Network-based Vision Systems for Unmanned Aerial Vehicles*, MASc thesis, Department of Systems and Computer Engineering, Carleton University.

Viquerat, A. *et al*. 2008, "*Reactive collision avoidance for unmanned aerial vehicles using doppler radar*", In Field and Service Robotics, pp. 245-254, Springer Berlin/Heidelberg.

Vision. Last visited May 15th 2013, "*3D Vision with Stereo Disparity*", http://www.shawnlankton.com/2007/12/3d-vision-with-stereo-disparity/, Dec 19th 2007.

Vries E.D. et al. 2013, "*Cooperative tracking of moving targets by a team of Autonomous UAVs",* 49th AIAA Aerospace Sciences Meeting

VRML. Last visited November 2013, "*A Beginner's Guide to VRML",* http://www.sv.vt.edu/classes/vrml/vrml_primer_index.html.

Webster, A. Last visited November 2013, "*Nanosatellites equipped with Kinect could join together like space LEGO*", The Verge, http://www.theverge.com/2012/5/28/3048138/nanosatellites-kinect-join-together/in/2380283.

Weibel, R. E. *et al. 2004.* "*Safety considerations for operations of different classes of UAVs in the NAS*", AIAA's 4th Aviation Technology and Operations (ATIO) Forum.
Wham, R. M. 2012 *"Three-Dimensional Kinematic Analysis Using the",* Honors thesis Project, Knoxville Trace: Tennessee Research and Creative Exchange, University of Tennessee.

Wibowo, S. S. 2007. "Aircraft Flight Dynamics, Control and Simulation Using Matlab and Simulink", Course Notes, University of Kuala Lumpur-Malaysian Institute of Aviation Technology.

Winnefeld, J. A. *et al.* Last visited September 2013, "*Unmanned Systems Integrated Roadmap FY2011-2036*", http://www.defenseinnovationmarketplace.mil/resources/UnmannedSystemsIntegratedRoadmapFY2011.pdf.

Zhong, J. 2006. *"PID Controller Tuning: A Short Tutorial",* (lecture), Department of

Mechanical Engineering, Purdue University.

Appendix A1 (Nano Series specs)

Nano Series - n920

microhard SYSTEMS INC.

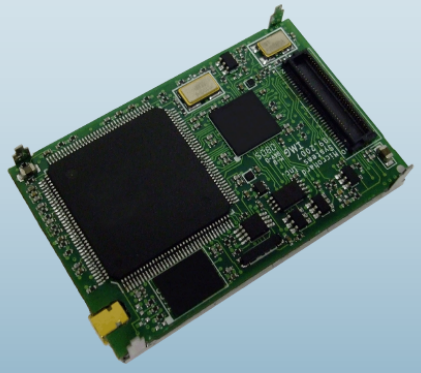
Miniature 900 MHz Wireless Modem

For size conscious consumers, consider the Nano Series with its small footprint and design flexibility. The Nano Series offers the reliability, features, and performance of our larger modems, yet can fit almost anywhere! Fully compatible with MHX series radio modems!

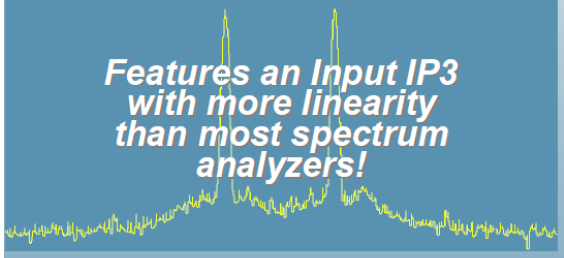
Applications

- *Robotics*
- *Remote Telemetry, DGPS*
- *Electric, Oil & Gas Sensors/Detection*
- *Display Signs*
- *Small Enclosure Communication Devices*
- *Industrial Communications*

Weighs only 18 grams!



2" X 1.25" X 0.25"



The Nano Series features robust, high speed, low latency, secure data communications. The n920 has full serial and diagnostics data capabilities and is 'radio compatible' with 910/910A and 920/920A modems! The n920 offers excellent noise figure, superior interference rejection, agile frequency synthesis, digital modulation, and matched filter detection. The n920 can support 1.2Mbps and higher!

Features of the n920

- Supports up to 1.2Mbps (higher rates available, contact Microhard for details)
- Quad Filter Stage provides Extreme Noise & Interference Rejection
- Supports Point-to-Point, Point-to-Multipoint, Store and Forward Repeater, TDMA
- Maximum allowable transit power (1W)
- Low Power consumption in Sleep and Sniff modes
- 32 bits of CRC, selectable Forward Error Correction with retransmit
- Separate diagnostics port—transparent remote diagnosis and online network control
- Very Small Footprint
- MHX910 and MHX920A compatible interface card available

Interface Options

Enclosed Solution

Motherboard Solution

Rev 1.03

n920 Specifications

Frequency		902-928 MHz
Spreading Method		Frequency Hopping / DTS
Band Segments		Selectable via Freq Restriction
Forward Error Detection		Hamming BCH Golay Reed-Solomon
Error Detection		32 bits of CRC, ARQ
Encryption		Optional (see –AES option)
Range		60+ miles (100+ km)
Sensitivity	**n920T** **n920F** **n920S**	-100 dBm @ 10^{-4} -108 dBm @ 10^{-6} -116 dBm @ 10^{-6}
Output Power		100mW - 1W (20-30dBm)
Serial Interface		TTL
Serial Baud Rate		- Up to 230.4 kbps asynchronous - Up to 3.2 Mbps synchronous
Link Rate		19.2kbps to 1.3824 Mbps (higher rates available, contact Microhard for details)
Operating Modes		Point-to-Point, Point-to-Multipoint, Store & Forward Repeater, Peer-to-Peer
Signals Interface		RxD1, TxD1, RTS, CTS DCD, DSR, DTR, RxD2, TxD2, RSSI LEDs, Tx/Rx LEDs, Reset, Config, Wake-up, RSmode
Input IP3 (Antenna Connector)		+12 dBm
RF Selectivity		
Adjacent Channel		60 dB
Alternate Channel		75 dB
Out of Band		>90 dB
Remote Diagnostics		VSWR, Battery Voltage, Temperature, RSSI, Packet Statistics
Core Voltage		
	OEM	3.3VDC Nominal (+/- 0.3V)
	Enclosed	7-30VDC

Power Consumption (3.3V +/- 0.3V)	Sleep	< 1mA
	Idle	20mA
	Rx:	140mA to 280mA
	Tx :	1000mA to 1500mA
(12V with Development board/Enclosed Unit)	Sleep	< 1mA
	Rx:	50mA to 95mA
	Master Avg Tx/Rx:	140mA
	Slave Avg Tx/Rx:	110mA
	Tx :	300mA to 450mA
Connectors:		
OEM		
Antenna		MMCX
Data		60 Pin OEM Header
Enclosed		
Antenna		RP-TNC Female Bulkhead
Data		Female DB9 x2
Environmental		-40°C - +85°C (All units are fully tested over the entire temperature Range)
Weight		
OEM		Approx. 18 grams
Enclosed		Approx. 220 grams
Dimensions		
OEM		Approx. 1.25" x 2.0" x .25" (32mm x 51mm x 6.35mm)
Enclosed		Approx. 2.25" x 3.85" x 1.70" (57mm x 98mm x 43mm)
Approvals		FCC Part 15.247 IC RSS210

Order Options

MHS113000(OEM)/MHS114000(ENC) **MHS113060(OEM)/MHS114060(ENC)** **MHS113020(OEM)MHS114020(ENC)** **MHS113040(OEM)**	n920S - Slow n920LC - Low Cost (115 kbps) n920F - Fast n920T - Turbo
-AES	128-bit AES Encryption (NOT AVAILABLE for export outside of Canada and USA.)
-AES256	256-bit AES Encryption (NOT AVAILABLE for export outside of Canada and USA.)
-2W	Up to 2 Watts Output Power. (Government, Export Only. Not for commercial / Industrial use) Contact Microhard for details.
-C1D2	Class 1 Div 2 (for use in hazardous environments)
-869	869.4 MHz - 869.65 MHz Operation

Contact Information

Microhard Systems Inc.
150 Country Hills Landing N.W.
Calgary, AB, Canada T3K 5P3

Email: info@microhardcorp.com
Tel: (403) 248-0028
Fax: (403) 248-2762

microhard SYSTEMS INC.

www.microhardcorp.com

Rev 1.03

Appendix A2 (SFOC)

Transport Canada / Transports Canada

4900 Yonge Street
4th Floor
Toronto, Ontario
M2N 6A5

UNCLASSIFIED
Your file / Votre référence

Our file / Notre référence
5812-15-16

February 19, 2013

Carleton University
Department of Mechanical and Aerospace Engineering
AVIONICS TEST BED 10 RC Aircraft
C/O Jeremy Laliberte
1125 Colonel By Drive
Ottawa, Ontario
K1S 5B6

Attention: Jeremy Laliberte, AVIONICS TEST BED 10 RC UAV Operations Manager

Certificate Number 5812-15-16-2013-2

Dear Mr. Laliberte:

Pursuant to section 603.67 of the *Canadian Aviation Regulations* this constitutes your Special Flight Operations Certificate (SFOC) for the operation of the Senior Telemaster unmanned air vehicle (UAV) systems as described in your SFOC application dated February 10, 2013.

This Certificate is issued to **Carleton University**. It may be suspended or cancelled at any time by the Minister for cause, including failure on the part of the Certificate holder, its servants or agents to comply with the provisions of the *Aeronautics Act* and the *Canadian Aviation Regulations* (CARs). This Certificate is not transferable and is valid from **March 1, 2013 until February 28, 2014** or until it is suspended or cancelled.

This Certificate is valid for the operation of the "**ATB10" Senior Telemaster** unmanned air vehicle, at the **Arnprior Remote Control Club** (ARCC) field, Arnprior, Ontario, for the purpose of conducting test flights.

Nothing in this Certificate shall be held to relieve the Certificate holder from requirements to comply with the provisions of such Canadian Aviation Documents as may have been issued to him pursuant to the *Aeronautics Act* or the *Canadian Aviation Regulations*.

Issued under the authority of the Minister pursuant to the *Aeronautics Act*, this document certifies that the Certificate holder is adequately equipped and able to conduct a safe operation, subject to the observance and performance by the certificate holder of the following conditions:

1. Except where otherwise referred to in this Certificate, the Certificate holder shall comply with the applicable provisions of the *Aeronautics Act* and the *Canadian Aviation Regulations* (CARs).

2. The Certificate holder shall maintain an adequate management organization that is capable of exercising supervision and operational control over persons participating in the UAV operation.

3. The Certificate holder shall conduct the operation of the UAVs in a safe manner.

4. The Certificate holder shall have subscribed for adequate liability insurance covering risks of public liability at the levels described in subsection 606.02(8) of the *Canadian Aviation Regulations* for the period of each UAV's operation.

5. The Certificate holder shall adhere to the security plan for the Arnprior Remote Control Club area of operation in accordance with the data provided in Carleton University's SFOC application dated January 13, 2013, or as otherwise agreed upon in writing between Carleton University, and Transport Canada.

6. The normal and emergency procedures for the Arnprior Remote Control Club area of operation shall be conducted in accordance with data provided in Carleton University's SFOC application dated January 13, 2013, or as otherwise agreed upon in writing between Carleton University, and Transport Canada.

7. The Certificate holder is responsible for ensuring that the Arnprior Remote Control Club have been advised of the proposed respective operations and have no objections.

8. Throughout flight operations, the Certificate holder shall ensure that the UAVs are flown over areas that would permit a safe landing on the surface without hazard to persons or property in the event of any emergency requiring immediate landing.

9. Only one UAV shall be operated in-flight at any one time at a given location.

10. The UAVs shall give way to manned aircraft.

11. Operations of the UAV shall only be conducted in VFR weather conditions as required for the airspace the operation is to be conducted in.

12. The UAVs shall remain clear of cloud during flight operations.

13. The UAVs shall only be operated during daylight hours.

14. The UAV pilot **or** the crew member assigned with observer duties, shall maintain continuous unaided visual contact with the UAV when in operation.

15. The operation of the UAVs shall be limited to a maximum altitude of 800 feet AGL at Arnprior Remote Control Club field.

16. When operations of the UAVs are to be conducted at an altitude greater than 500 feet above ground level (AGL), the certificate holder shall contact the appropriate Flight Information Centre (FIC) by phone (London 1-866-541-4104), to arrange for a "Notice to Airmen" (NOTAM) to be issued advising other aircraft of the UAV operation.

17. Notwithstanding the requirements of paragraph 602.14(2)(b) of the ***Canadian Aviation Regulations***, the UAVs shall not be operated (including take-offs, landings, and flight demonstrations) at a lateral distance of less than 30 metres away from inhabited

structures such as buildings, vehicles or vessels, or from other persons who are not associated with the operation. In addition the UAV shall not be operated at a distance less than 30 metres away from any livestock.

18. A crew member assigned and trained to perform duties as an observer associated with collision avoidance, such as continuously monitoring the UAV and the airspace (e.g. for other air traffic, clouds, obstructions, and terrain) both around and sufficiently beyond the UAV, shall be present for all flight operations.

19. The observer(s) shall maintain constant communication with the UAV pilot at all times while the UAVs are being operated.

20. The UAVs shall only be operated by Carleton University, Department of Mechanical and Aerospace Engineering Staff and Students, who are members of the ATB10 UAV Team, and are authorized by the Operation Manager or Deputy Operation Manager.

21. The Certificate holder shall not require any person to operate the controls of the UAVs if either the person or the Certificate holder has any reason to believe that the person is suffering or is likely to suffer from fatigue so that they are unfit to perform their duties.

22. No person shall operate the UAVs within eight hours after consuming an alcoholic beverage or while under the influence of alcohol or while using any drug that impairs the person's faculties to the extent that the safety of the operation is endangered in any way.

23. Flight of the UAVs over spectators is prohibited.

24. The UAVs shall not carry a payload that can be jettisoned, disperses products, drops objects or is an explosive device.

25. The UAVs shall be operated in accordance with the Model Aeronautics Association of Canada Safety Guidelines, and in accordance with the Club Field Rules for the location they are operating at.

26. The Certificate holder shall report to this office, on the first working day following, details of any of the following occurrences:

    a. injuries to persons requiring medical attention resulting from the operation of a UAV authorized in this SFOC;
    b. unintended contact between the UAVs and persons, livestock, vehicles, vessels or other structures;
    c. unanticipated damage incurred to a UAV, control station, payload or communications links resulting from the operation of a UAV authorized in this SFOC;
    d. any time a UAV flies outside of the bounds of operation as described in this SFOC; and
    e. any other incident (resulting from the UAVs operation) that results in a Canadian Aviation Daily Occurrence Report (CADORS).

27. The Certificate holder shall not operate the UAVs following any of the occurrences listed in condition **25**, until such time as this office approves its further operation in writing.

28. All persons connected with this operation shall be familiar with the contents of this SFOC, and the contents of Carleton University's SFOC application dated January 13, 2013.

29. A copy of this Certificate and a copy of Carleton University's SFOC application dated January 13, 2013, shall be on site any time the UAVs are in operation.

Yours truly,

Clifford Frank Jr.
Technical Team Lead - Flight Operations
Civil Aviation, Ontario Region
for Minister of Transport

Appendix B1 (Matlab Code for Intensity Map)

```
function [outMap, outdispmap]  = stereomatch(imgLeftIn, imgRightIn)

coder.extrinsic('disparity', 'find','reproject');

contrastThresh = 0.9;
blockWndSize = 15;
dispRange = [0, 64];
uniqueThresh = 15;
textureThresh = 0.002;
distanceThresh = 0; %[ 0 ]; %[];

dispMap = zeros(size(imgLeftIn,1), size(imgLeftIn,2), 'single');

dispMap = disparity(imgLeftIn, imgRightIn, ...
                    'ContrastThreshold', contrastThresh, ...
                    'BlockSize', blockWndSize, ...
                    'DisparityRange', dispRange, ...
                    'UniquenessThreshold', uniqueThresh, ...
                    'TextureThreshold', textureThresh, ...
                    'DistanceThreshold', distanceThresh);

flattenedDispMap = dispMap(:, :);
dispMinRange = flattenedDispMap < dispRange(1);
dispMap(dispMinRange) = 0;

dispMaxRange = flattenedDispMap > dispRange(2);
dispMap(dispMaxRange) = 0;
dispMap = (255.0/(dispRange(2)-dispRange(1)))*dispMap - dispRange(1);
 outdispmap=dispMap;
reproject(dispMap,[20,30]);
outMap = cast(dispMap, 'uint8');
```

Appendix B2 (Parameter Description)

Property	Description
`contrastThresh`	The contrast threshold defines a range of contrast values
`blockWndSize`	The function sets the block to a square, of size BlockSize-by-BlockSize pixels.
`dispRange`	A two-element vector defining the range of disparity
`uniqueThresh`	A non-negative integer defining the minimum value of uniqueness. When the function sets a pixel to a low uniqueness value, the disparity computed for it is less reliable. Increasing this parameter will result in marking more pixels unreliable because of the higher threshold.
`textureThresh`	The texture threshold defines the minimum texture value for a pixel to be considered reliable. The lower the texture value, the less reliable the computed disparity is for the pixel. When you increase this parameter, more pixels with low texture value are marked as unreliable.
`distanceThresh`	A non-negative integer defining the maximum distance for left-to-right image checking between two points. When you decrease the value of the distance threshold, you increase the reliability of the disparity map

Appendix B3 (Connected Components)

```
inImgPath = 'C:\Users\Shacks\Desktop\test4.jpg';
loadImg = imread(inImgPath);



if ( size(loadImg, 3) == 1 )
   inImg = loadImg;
   drawImg(:,:,1) = loadImg;
   drawImg(:,:,2) = loadImg;
   drawImg(:,:,3) = loadImg;
else
   inImg = rgb2gray(loadImg);
   drawImg = loadImg;
end

shapeInserter = vision.ShapeInserter('Shape', 'Rectangles', ...
                        'BorderColor', 'Custom', ...
                        'CustomBorderColor', uint8([255 0 0]));

boundingBoxes = {};
minArea = 50;

groupingType = 1;         % 0: disparity, 1:intensity

% parameters
disparityMin = 0;
disparityMax = 16;
disparityDelta = 1;

imgPixValMin = 0;
imgPixValMax = 255;
imgPixDelta = 30;

ignoreHighLow = 'low';
numBins = -1;
startBin = -1;
endBin = -1;
binDelta = -1;
```

```
% begin
if ( groupingType == 0 )
   numBins = (disparityMax - disparityMin)/disparityDelta;
   binDelta = disparityDelta;
   startBin = disparityMin;
   endBin = disparityMin+disparityDelta;
elseif ( groupingType == 1 )
   numBins = (imgPixValMax - imgPixValMin)/imgPixDelta;
   binDelta = imgPixDelta;
   startBin = imgPixValMin;
   endBin = imgPixValMin+imgPixDelta;
else
   disp('');
   return;
end

bwimg = zeros(size(inImg,1), size(inImg,2));
for levelIdx=1:numBins
   ind = [];
   if ( strcmp(ignoreHighLow, 'high') )
      ind = find(inImg >= startBin & inImg < endBin);
   else
      ind = find(inImg > startBin & inImg <= endBin);
   end

   if ( ~isempty(ind) )
      % create binary image using only the pixels in the current range
      bwimg(ind) = 1;

      % find all 8 connected neighbours
      CC = bwconncomp(bwimg, 8);

      % process all intensity clusters
      for neighIdx=1:CC.NumObjects
         if ( length(CC.PixelIdxList{neighIdx}) > minArea )
            leftmostHorz = 999;
            rightmostHorz = -1;
            topmostVert = 999;
            bottommostVert = -1;

            for neighPixIdx=1:length(CC.PixelIdxList{neighIdx})
               [r,c]=ind2sub(size(bwimg), CC.PixelIdxList{neighIdx}(neighPixIdx));
               if ( c < leftmostHorz )
                  leftmostHorz = c;
```

```
            end

            if ( c > rightmostHorz )
               rightmostHorz = c;
            end

            if ( r < topmostVert )
               topmostVert = r;
            end

            if ( r > bottommostVert )
               bottommostVert = r;
            end
          end

          drawImg = step(shapeInserter, drawImg, ...
                    int32([ leftmostHorz, topmostVert, ...
                          rightmostHorz-leftmostHorz, ...
                          bottommostVert-topmostVert ]));
          %boundingBoxes = boundingBoxes; ...
          %              [ topmostVert, leftmostHorz, bottommostVert, rightmostHorz ] };
        end
      end

      % cleanup
      bwimg(ind) = 0;
    end

    startBin = startBin + binDelta;
    endBin = endBin + binDelta;
end

imshow(drawImg);
```

Appendix B4 (Matlab code for 2D mapping)

```
clc;
clear;

carobj=VideoReader('disparity_out.wmv');

nFrames=carobj.NumberOfFrames;

M=carobj.Height; % no of rows

N=carobj.Width; % no of columns

video=zeros(M,N,nFrames,'uint8'); % creating a video 3d matrix

for k= 1 : nFrames

im= read(carobj,k);

im=im(:,:,1);          % all three layers will have same image

video(:,:,k)=im;
end

analyse=zeros(nFrames,N,'uint8');

for k= 1 : nFrames
analyse(k,:)=video(460,:,k);
end


if ( size(analyse, 3) == 1 )
   inImg = analyse;
   drawImg(:,:,1) = analyse;
   drawImg(:,:,2) = analyse;
   drawImg(:,:,3) = analyse;
else
   inImg = rgb2gray(analyse);
   drawImg = analyse;
end

image(analyse)
```

Appendix B5 (Matlab code for Reprojection)

```
function [] = reproject(dispMap, dispRange)

coder.extrinsic('scatter3');

Xpos = zeros(size(dispMap,1)*size(dispMap,2), 1);
Ypos = zeros(size(dispMap,1)*size(dispMap,2), 1);
Zpos = zeros(size(dispMap,1)*size(dispMap,2), 1);

% fill disparity map based on the loaded disparity map

% assume field of view is given and also assume its the same for both
% cameras (for now)
cameraFOVHorz = 57.29 * (pi/180);           % horizontal camera FOV in rads
cameraFOVVert = cameraFOVHorz;              % vertical camera FOV in rads

% 2 assumptions for the principal points:
%   1) assume the optical centre is at the centre of the image (ideal camera) for both cameras
%   2) assume that the optical axes are aligned for both cameras so that
%      the horizontal difference of the principal points between the two
%      cameras is 0 (and so Q(4,4) below is (cx - cx')/T = 0)
principalX = size(dispMap, 2)/2.0;
principalY = size(dispMap, 1)/2.0;

cameraBaseline = 2;                         % in the simulators 'physical' units of distance

% camera matrix
cameraMatrix = [ size(dispMap,2)/(2*tan(cameraFOVHorz/2)), 0, principalX;
            0, size(dispMap,1)/(2*tan(cameraFOVVert/2)), principalY;
            0, 0, 1 ];

% reprojection from homogeneous pixel coordinates to homogeneous world coordinates
% so we have [ Xh, Yh, Zh, Wh ] = Q*[ x_pixel, y_pixel, disparity_pixel, 1 ]
% and the 3d pixel position is [ X, Y, Z ] = [ Xh/Wh, Yh/Wh, Zh/Wh ]
Q = [ 1, 0, 0, -cameraMatrix(1,3);
    0, 1, 0, -cameraMatrix(2,3);
    0, 0, 0, cameraMatrix(1,1);      % take the horizontal
    0, 0, 1/cameraBaseline, 0 ];     % principal points are aligned

% project all which have a disparity to their 3D positions
nextPoint = 1;
for vert=1:size(dispMap,1)
```

```
    for horz=1:size(dispMap,2)
        % check if theres a valid disparity here
        if ( dispMap(vert, horz) > dispRange(1) && ...
             dispMap(vert, horz) <= dispRange(2) )
            XYZW = Q*[ horz; vert; dispMap(vert, horz); 1];

            Xpos(nextPoint) = XYZW(1)/XYZW(4);
            Ypos(nextPoint) = XYZW(2)/XYZW(4);
            Zpos(nextPoint) = XYZW(3)/XYZW(4);
            nextPoint = nextPoint + 1;
        end
    end
end

% if we found anything plot it
if ( nextPoint > 1 )
    % truncate
    Xpos = Xpos(1:(nextPoint-1));
    Ypos = Ypos(1:(nextPoint-1));
    Zpos = Zpos(1:(nextPoint-1));

    scatter3(Ypos, Xpos, Zpos, 'fill');
    title('REPROJECTION');
xlabel('Z');
ylabel('X');
zlabel('Y');
else
    disp('no valid points found in disparity map...');
end
```

Appendix B6 (Orientation for 6-DOF)

```
function [x,y,z,rotation]= fcn(Roll,Pitch,Yaw)

x=0;
y=0;
z=0;
rotation=0;



   comparison=[abs(Pitch) abs(Roll) abs(Yaw)]; % ordered
according to x y z notation at vrml programming
   temp= max(comparison); % serching for the max value



if (temp == comparison(1))  %Based on the max value the code
decides for changing pitch angle
    x=1;
    y=0;
    z=1;
    rotation=comparison(1);

elseif  (temp == comparison(2)) %Based on the max value the code
decides for changing yaw angle
    x=0;
    y=1;
    z=0;
    rotation=comparison(2);

elseif (temp == comparison(3)) %Based on the max value the code
decides for changing bank angle
   x=0;
   y=0;
   z=1;
   rotation=comparison(3);

else    % for rare case of stable flight perfect camera motion
is outputed
   x=0;
   y=1;
   z=0;
   rotation=0;

end
```

Appendix B7 (VRML- Room/Corridor)

```
#VRML V2.0 utf8

WorldInfo {
  title "Kinect test replica"
  info "$Author: shacks $"
}
NavigationInfo {
  type "EXAMINE"
}
DEF View1 Viewpoint {
  description "View 1 - Observer"
  position 0 1.234 0
  orientation 0 1 0 3.14
  fieldOfView 1
}
DEF Camera_car Transform {
  translation 1 0.25 1
  rotation 0 1 0 0
  children DEF VPfollow Viewpoint {
    description "View 2 - Driver"
    position -1.1 0.617 4
    orientation 0 1 0 3.14
    fieldOfView 7.85398
  }
}
DirectionalLight {
  intensity 5000
  direction 0.9 -0.3 -0.35
  color 1 0.8 0.7
}
SpotLight {
  #         floating point angle in radians = 90 degree
  #         floating point angle in radians = 45 degree
  location 1.203 2.234 13.56
  cutOffAngle 0.985398
  beamWidth 1.5708
  ambientIntensity 5
}
PointLight {
  radius 1
  location 1.203 2.234 13.56
```

```
  color 0 0 1
}
Background {
  skyColor [0 0 0, 0 0 0, 0 0 0 ]
  skyAngle [0, 0 ]
  groundColor [1 1 1, 1 1 1 ]
  groundAngle 0
}
DEF floor Transform {
  children Shape {
    geometry Box {
      size 300 0.0001 80
    }
    appearance Appearance {
      textureTransform TextureTransform {
        #          rotation 0.95
        scale 80 40
      }
      texture ImageTexture {
        url "floor.jpg"
      }
      material Material {
        diffuseColor 0.3 0.4 0.5
      }
    }
  }
}
DEF Wall1 Group {
  children Shape {
    geometry Box {
      size 13.136 2.468 0.1
    }
    appearance Appearance {
      textureTransform TextureTransform {
        scale 15 5
      }
      texture ImageTexture {
        url "wall.jpg"
      }
      material Material {
        diffuseColor 0.4 0.23 0.13
      }
    }
  }
```

```
}
DEF Wall2 Group {
  children Shape {
    geometry Box {
      size 2.407 2.468 0.1
    }
    appearance Appearance {
      textureTransform TextureTransform {
        scale 15 5
      }
      texture ImageTexture {
        url "wall_2.jpg"
      }
      material Material {
        diffuseColor 0.4 0.23 0.13
      }
    }
  }
}
DEF Wall3 Group {
  children Shape {
    geometry Box {
      size 2.407 0.1 13.136
    }
    appearance Appearance {
      textureTransform TextureTransform {
        scale 5 5
      }
      texture ImageTexture {
        url "wall_3.jpg"
      }
      material Material {
        diffuseColor 0.4 0.23 0.13
      }
    }
  }
}
DEF Floor1 Group {
  children Shape {
    geometry Box {
      size 1.805 0.1 13.136
    }
    appearance Appearance {
      textureTransform TextureTransform {
```

```
      scale 5 5
     }
     texture ImageTexture {
      url "_floor1.jpg"
     }
     material Material {
      diffuseColor 0.4 0.23 0.13
     }
    }
   }
}
DEF Floor2 Group {
  children Shape {
   geometry Box {
    size 0.601 0.1 13.136
   }
   appearance Appearance {
    textureTransform TextureTransform {
     scale 5 5
    }
    texture ImageTexture {
     url "_floor2.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
DEF Box1 Group {
  children Shape {
   geometry Box {
    size 0.1 0.9 0.9
   }
   appearance Appearance {
    texture ImageTexture {
     url "obj.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
```

```
DEF Box2 Group {
  children Shape {
    geometry Box {
      size 0.1 1 0.4
    }
    appearance Appearance {
      texture ImageTexture {
        url "obj2.jpg"
      }
      material Material {
        diffuseColor 0.4 0.23 0.13
      }
    }
  }
}
DEF Obstacles Group {
  children [
    Transform {
      translation -0.3 0 13.56
      rotation 0 0 0 0
      children USE Floor1
    }
    Transform {
      translation 0.9 0 13.56
      rotation 0 0 0 0
      children USE Floor2
    }
    Transform {
      translation -0 1.234 13.56
      rotation 0 0 0 0
      children USE Wall2
    }
    Transform {
      translation -0 2.468 13.56
      rotation 0 0 0 0
      children USE Wall3
    }
    Transform {
      translation -1.203 1.234 13.56
      rotation 0 1 0 1.57
      children USE Wall1
    }
    Transform {
      translation 1.203 1.234 13.56
```

```
    rotation 0 1 0 1.57
    children USE Wall1
   }
   Transform {
    translation 0 0.15 8.614
    rotation 0 1 0 1.57
    children USE Box2
   }
   Transform {
    translation 0 1.1 8.614
    rotation 0 1 0 1.57
    children USE Box1
   }
  ]
}
```

Appendix B7 (VRML- Field)

```
#VRML V2.0 utf8
WorldInfo {
  title "Kinect test replica"
  info "$Author: shashank $"
}
NavigationInfo {
  type "EXAMINE"
}
DEF View1 Viewpoint {
  description "View 1 - Observer"
  position 0 1.234 0
  orientation 0 1 0 3.14
  fieldOfView 1
}
DEF Camera_car Transform {
  translation 1 0.25 1
  rotation 0 1 0 0
  children DEF VPfollow Viewpoint {
    description "View 2 - Driver"
    position -0.8 0.617 3
    orientation 0 1 0 3.14
    fieldOfView 7.85398
  }
}
SpotLight {
  #  floating point angle in radians = 90 degree
  location 1.203 2.234 8.56
  direction 0 1 1
  #  floating point angle in radians = 45 degree
  cutOffAngle 0.985398
  beamWidth 0.570796
  intensity 5
  ambientIntensity 500
}
SpotLight {
  #  floating point angle in radians = 90 degree
  location 1.203 2.234 12.56
  #  floating point angle in radians = 45 degree
  cutOffAngle 0.985398
  beamWidth 3.14
  intensity 5
```

```
  ambientIntensity 500
}
Background {
  skyColor [0 0 0, 0 0 0, 0 0 0 ]
  skyAngle [0, 0 ]
  groundColor [1 1 1, 1 1 1 ]
  groundAngle 0
}
DEF floor Transform {
  children Shape {
    geometry Box {
      size 300 0.0001 80
    }
    appearance Appearance {
      textureTransform TextureTransform {
        #  rotation 0.95
        scale 80 40
      }
      texture ImageTexture {
        url "floor.jpg"
      }
      material Material {
        diffuseColor 0.3 0.4 0.5
      }
    }
  }
}
DEF Wall1 Group {
  children Shape {
    geometry Box {
      size 13.136 2.468 0.1
    }
    appearance Appearance {
      textureTransform TextureTransform {
        scale 15 5
      }
      texture ImageTexture {
        url "wall.jpg"
      }
      material Material {
        diffuseColor 0.4 0.23 0.13
      }
    }
  }
```

```
}
DEF Wall_1 Group {
  children Shape {
    geometry Box {
      size 13.136 0.1 0.1
    }
    appearance Appearance {
      textureTransform TextureTransform {
        scale 15 5
      }
      texture ImageTexture {
        url "blackwall.jpg"
      }
      material Material {
        diffuseColor 0.4 0.23 0.13
      }
    }
  }
}
DEF Wall_2 Group {
  children Shape {
    geometry Box {
      size 13.136 0.3 0.3
    }
    appearance Appearance {
      textureTransform TextureTransform {
        scale 15 5
      }
      texture ImageTexture {
        url "light.jpg"
      }
      material Material {
        diffuseColor 0.4 0.23 0.13
      }
    }
  }
}
DEF Wall2 Group {
  children Shape {
    geometry Box {
      size 2.407 2.468 0.1
    }
    appearance Appearance {
      textureTransform TextureTransform {
```

```
      scale 15 5
     }
     texture ImageTexture {
      url "wall_2.jpg"
     }
     material Material {
      diffuseColor 0.4 0.23 0.13
     }
    }
   }
}
DEF Wall3 Group {
  children Shape {
   geometry Box {
    size 2.407 0.1 13.136
   }
   appearance Appearance {
    textureTransform TextureTransform {
     scale 5 5
    }
    texture ImageTexture {
     url "wall_3.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
DEF Floor1 Group {
  children Shape {
   geometry Box {
    size 1.805 0.1 13.136
   }
   appearance Appearance {
    textureTransform TextureTransform {
     scale 5 5
    }
    texture ImageTexture {
     url "_floor1.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
```

```
   }
  }
}
DEF Floor2 Group {
  children Shape {
   geometry Box {
    size 0.601 0.1 13.136
   }
   appearance Appearance {
    textureTransform TextureTransform {
     scale 5 5
    }
    texture ImageTexture {
     url "_floor2.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
DEF Door Group {
  children Shape {
   geometry Box {
    size 0.5 4 0.1
   }
   appearance Appearance {
    texture ImageTexture {
     url "blackwall.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
DEF Door_1 Group {
  children Shape {
   geometry Box {
    size 0.1 4 0.15
   }
   appearance Appearance {
    texture ImageTexture {
     url "blackwall.jpg"
```

```
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
DEF Box1 Group {
  children Shape {
   geometry Box {
    size 0.1 0.9 0.9
   }
   appearance Appearance {
    texture ImageTexture {
     url "obj.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
DEF Box2 Group {
  children Shape {
   geometry Box {
    size 0.1 1 0.4
   }
   appearance Appearance {
    texture ImageTexture {
     url "obj2.jpg"
    }
    material Material {
     diffuseColor 0.4 0.23 0.13
    }
   }
  }
}
DEF Obstacles Group {
  children [
   Transform {
    translation -0.3 0 13.56
    rotation 0 0 0 0
    children USE Floor1
   }
```

```
Transform {
 translation 0.9 0 13.56
 rotation 0 0 0 0
 children USE Floor2
}
Transform {
 translation -0 1.234 13.56
 rotation 0 0 0 0
 children USE Wall2
}
Transform {
 translation -0 2.468 13.56
 rotation 0 0 0 0
 children USE Wall3
}
Transform {
 translation -1.203 1.234 13.56
 rotation 0 1 0 1.57
 children USE Wall1
}
Transform {
 translation 1.203 1.234 13.56
 rotation 0 1 0 1.57
 children USE Wall1
}
Transform {
 translation -1.203 0.1 13.56
 rotation 0 1 0 1.57
 children USE Wall_1
}
Transform {
 translation 1.203 0.1 13.56
 rotation 0 1 0 1.57
 children USE Wall_1
}
Transform {
 translation 1.103 2.558 13.56
 rotation 0 1 0 1.57
 children USE Wall_2
}
Transform {
 translation 1.213 2.438 13.56
 rotation 0 1 0 1.57
 children USE Wall_2
```

```
    }
    Transform {
     translation 1.203 0.15 8.614
     rotation 0 1 0 1.57
     children USE Door
    }
    Transform {
     translation 1.203 0.15 8.864
     rotation 0 1 0 1.57
     children USE Door_1
    }
    Transform {
     translation 1.203 0.15 8.364
     rotation 0 1 0 1.57
     children USE Door_1
    }
    Transform {
     translation 1.203 0.15 10.614
     rotation 0 1 0 1.57
     children USE Door
    }
    Transform {
     translation 1.203 0.15 10.864
     rotation 0 1 0 1.57
     children USE Door_1
    }
    Transform {
     translation 1.203 0.15 10.364
     rotation 0 1 0 1.57
     children USE Door_1
    }
    Transform {
     translation 0 0.15 8.614
     rotation 0 1 0 1.57
     children USE Box2
    }
    Transform {
     translation 0 1.1 8.614
     rotation 0 1 0 1.57
     children USE Box1
    }
  ]
}
```

Appendix C1 (PPM_IN)

```
// defining variables
 channumber                        // define how many channels in the  radio
 filter 10                        //define glitch filter
 channel[channumber];             // define integer array read channel values
 lastReadChannel[channumber]; // define integer array  for last  values read
 conta=0;                         // define counter

// establishing serial port to transmit values
  Serial.begin(9600);              // Serial Begin
  pinMode(4, INPUT);               // Pin 4 as input
  pinMode(13, OUTPUT);             // Led pin 13

// Logic to gather PWM signals in the PPM
  If  Time off  > 3000 µs
  {
    // Read the pulses of the channels based on the number of channels
   {
     channel[i]=Time Off
   }

// logic to distinguish the end of frame pulse or the beginning of frame pulse
  for(int i = 0; i <= channumber-1; i++)          //Iterate based on the number of channels
   {
    if((channel[i] > 2000) || (channel[i] <100))// If channel > max range, change the value to the last
pulse
    {
     channel[i]= lastReadChannel[i];             // identifying the last pulse
    }
    else
    {
    channel[i]=(lastReadChannel[i]+channel[i])/2; //Average the last pulse with the current pulse
    conta++;                                        // increment counter
    }}}

 // Filtering any potential glitches along with transmitting data to PC
// The filter makes sure that values are printed to PC , only if there are valid ppm signal read by Arduino.

if(conta > filter)                          // If counter is > than filter, then prints values
   {
    for(int i = 0; i <= channumber-1; i++) //Cycle to print values
    {
      Serial.print("CH");                  // Print 'CH'
      Serial.print(i+1);                   // Print Channel number
      Serial.print(": ");                  //
      Serial.println(channel[i]);
```

```
      lastReadChannel[i]=channel[i];
    }
    delay(400); //Delay
    conta=0;// Restart couter.
  }
}
```

Appendix C2 (PPM_OUT)

Part 1

```
// Defining variables
unsigned long loopTime, currentTime;
const int serStrLen = 30;
char serInStr[ serStrLen ];  // array that will hold the serial input string

//Serial Read
uint8_t readSerialString()         // Defining function name so that can be used later in part 2
// Searching if the serial port is active
// wait a little for serial data
//
  memset( serInStr, 0, sizeof(serInStr) );          // set it all to zero
  int i = 0;
  while(Serial.available() && i<serStrLen ) {
    serInStr[i] = Serial.read();                    // The required number/angle  is saved in this array
    i++;
  }
  return i;                                         // return number of chars read
}
```

Part 2

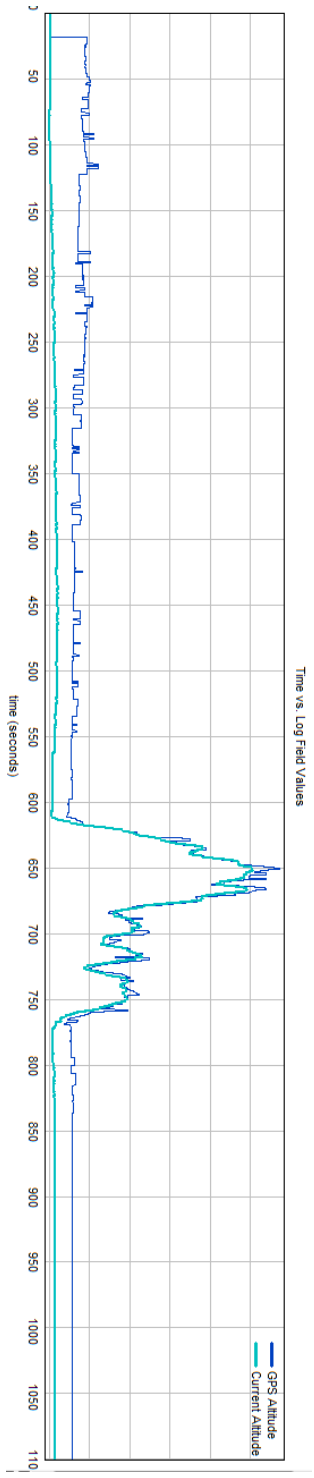
```
// Declaring Variables
// Control pin for trainer interface
servoPin = 12
// A pulse starts with a low signal of fixed width (0.3ms), followed by a high signal for the remainder of
the pulse. Total pulse width is proportional to servo position (1 to 2ms)
pulseStart = 300       // pulse start width in microseconds
// pulse minimum width minus start in microseconds
pulseMin = 200
// pulse maximum width in microseconds
pulseMax = 1600
// Conversion factor is the constant that determines the relation between servo angle and frame length.
 Conversion Factor = pulseMax - pulseMin - pulseStart
conversionFactor = 4.5
// A frame is a succession of pulses, in order of channels, followed by a synchronisation pulse to fill out
the frame. A frame's total length is fixed (20ms)
frameLength = 20
// The time in milliseconds of the last frame
lastFrame = 200
// Number of channels to send (keep below frameLength/pulseMax)
channelNumber = 4


// Values to send on channels (duration of pulse minus start, in  microseconds)
channel[3]
```

```
//Counter Variables
 i, j

// Setup serial connection and PPM signal output port
void setup() {


//PPM output
 pinMode(servoPin, OUTPUT);  // Set servo pin as an output pin


 //Serial Reading
Serial.begin(9600)        // connect to the serial port while declaring baud rate
  Serial.flush()

//Setting default PPM signal incase of no input
  for ( i = 0; i < channelNumber; i = i + 1 ) {channel[i] = pulseMin;}
  Serial.println("Trainer_PPM_Interface ready");
}


void loop() {
//  Acquiring angle data in char format using function developed in part 1
     if( readSerialString() )
     {
     Serial.println(serInStr);
     }

  // Save the time of frame start
  lastFrame = millis();
//Returns the number of milliseconds since the Arduino board began running the current program. This
number will overflow (go back to zero), after approximately 50 days.

  // This for loop generates the pulse train of the PPM, one per channel

 for ( i = 0; i < channelNumber; i = i + 1 ) {
    digitalWrite(servoPin, LOW);   // Initiate pulse start
    delayMicroseconds(pulseStart); // Duration of pulse start
    digitalWrite(servoPin, HIGH);  // Stop pulse start
    delayMicroseconds(channel[i]); // Finish off pulse
  }
    digitalWrite(servoPin, LOW);   // Initiate synchronisation pulse
    delayMicroseconds(pulseStart); // Duration of start of synchronisation pulse
    digitalWrite(servoPin, HIGH);  // Stop synchronisation pulse start



  // Calculate pulse durations from servo positions
```

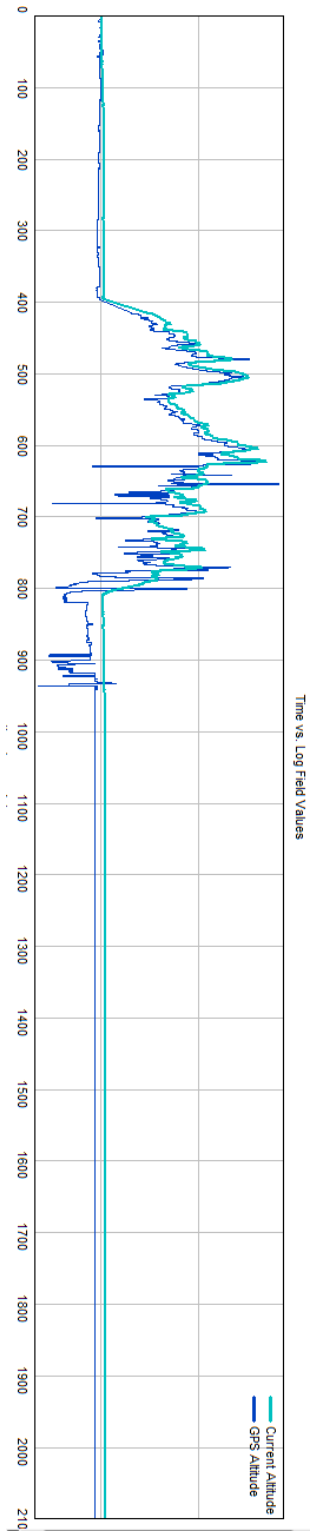
```
 //Channel 2
   channel[2] = abs(atoi( serInStr ));  //atoi converts char format to integer format.
                                        // abs takes the absolute angles, the reason for usage of absalut angles
will be mention in test procedure in section xxxxx
   channel[2] = int(channel[2]*conversionFactor)+pulseMin;

//Latest edition for ignoring lag between serialread and ppm output (explained in 6.4.1)
if (abs(prevserInStr - ((atoi( serInStr )))) < 20  ){
 channel[2] = abs(atoi( serInStr ));
 channel[2] = int(channel[2]*conversionFactor)+pulseMin;
 prevserInStr=abs(atoi( serInStr ));
 Serial.println(abs(prevserInStr - ((atoi( serInStr )))));
 Serial.println("execute");
 }

//If more channels needs to be sent
  //Channel 1
      //channel[1] = abs(1);
       // channel[1] = int(channel[1]*conversionFactor)+pulseMin;
  //Channel 3
      //channel[2] = abs(2);
      //channel[2] = int(channel[2]*conversionFactor)+pulseMin;
  //Channel 4
     // channel[3] = abs(3);
     // channel[3] = int(channel[3]*conversionFactor)+pulseMin;
  //Channel 5
     // channel[4] = abs(4);
     //channel[4] = int(channel[4]*conversionFactor)+pulseMin;
  //Channel 6
     // channel[5] = abs(5);
     // channel[5] = int(channel[5]*conversionFactor)+pulseMin;
  //Channel 7
     // channel[6] = abs(6);
    // channel[6] = int(channel[6]*conversionFactor)+pulseMin;
 //Channel 8
    // channel[7] = abs(7);
    // channel[7] = int(channel[7]*conversionFactor)+pulseMin;
  //if (j==0) {
  // Serial.println(channel[0]+pulseStart);
  //}

  // We're ready to wait for the next frame
  // Some jitter is allowed, so to the closest ms
  while (millis() - lastFrame < frameLength) {
    delay(1);
  }
}
```

Appendix C3 (Altitude Data)

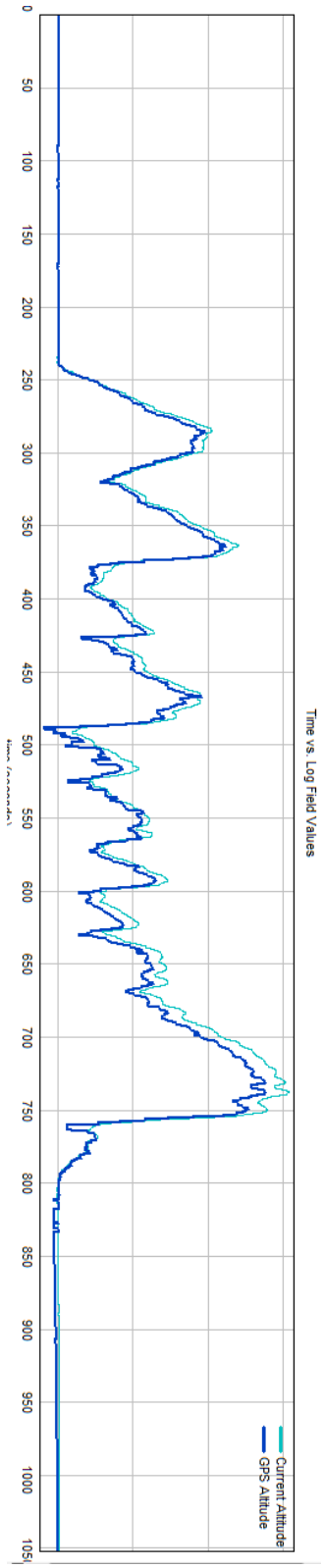
Trial 1



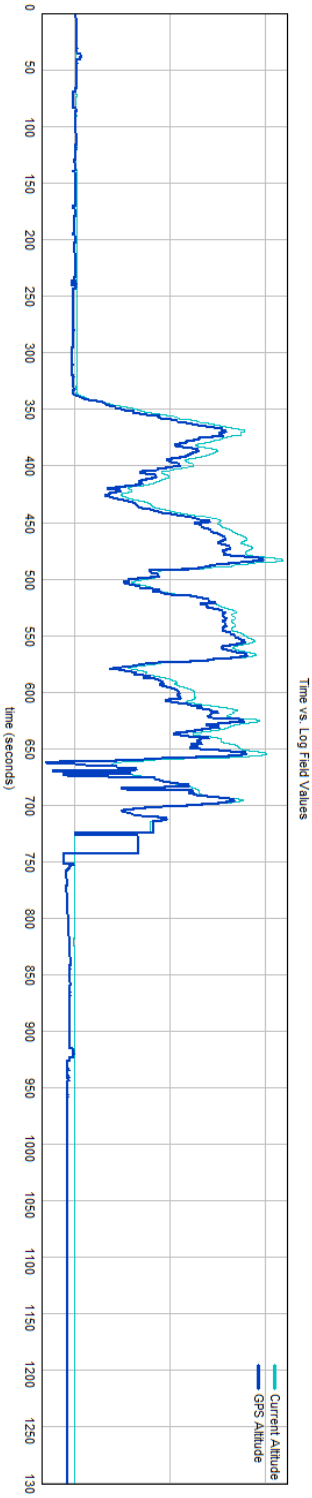
Trial 2


Time vs. Log Field Values
Current Altitude
GPS Altitude
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000

Trial 3


Time vs. Log Field Values
Current Altitude
GPS Altitude
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000

Trial 4


Time vs. Log Field Values
time (seconds)
Current Altitude
GPS Altitude
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000
1050
1100
1150
1200
1250
130

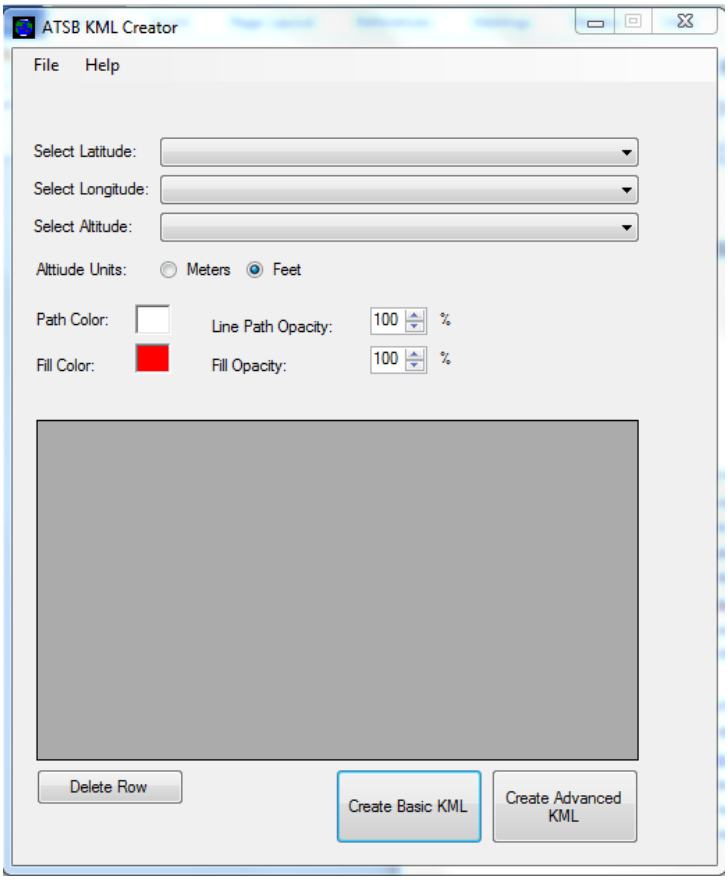
Appendix C4 (KML SETUP)

Setup of KML File.

1. Open the standard telemetry log file in excel. The log file is a text file and hence the file requires to be converted to excel format. Microsoft excel is capable of converting text file into an excel file. This procedure is straightforward, one has to simply follow the instructions provided by excel while attempting to open the log file.
2. In order to create the KML file, the latitude, longitude and altitude data is required. This data can be found in coloum 2, 3 and 8 respectively from the datalog file. Pls note that the altitude data from the log file is in feet*-8 format (refer table 14) and hence needs to converted into feet or meter. For the sake of avoiding any confusion, it is recommended that one copy the data from the standard telemetry file and paste it into a new excel file. Based on requirement, one need to copy only the relevant data i.e extract data based on the time scale.
3. Save the new excel file and open it in the ATSB KML creater. The KML Created can be downloaded from the following link:

   http://www.atsb.gov.au/utilities/atsb-kml-creator.aspx

   The software can be run directly after downloading the .exe file without any prior installation. Once opened the required excel file containing the data can be selected from the file tab in the left corner of the interface as shown below



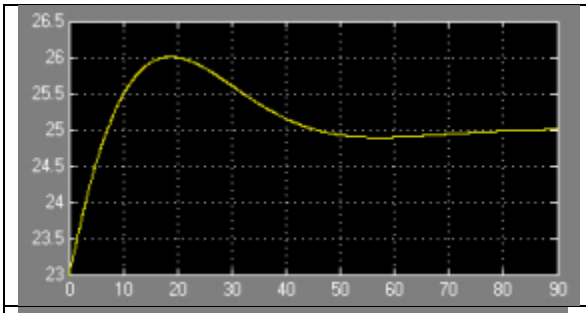
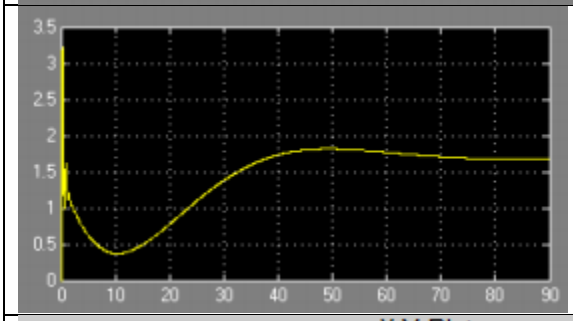
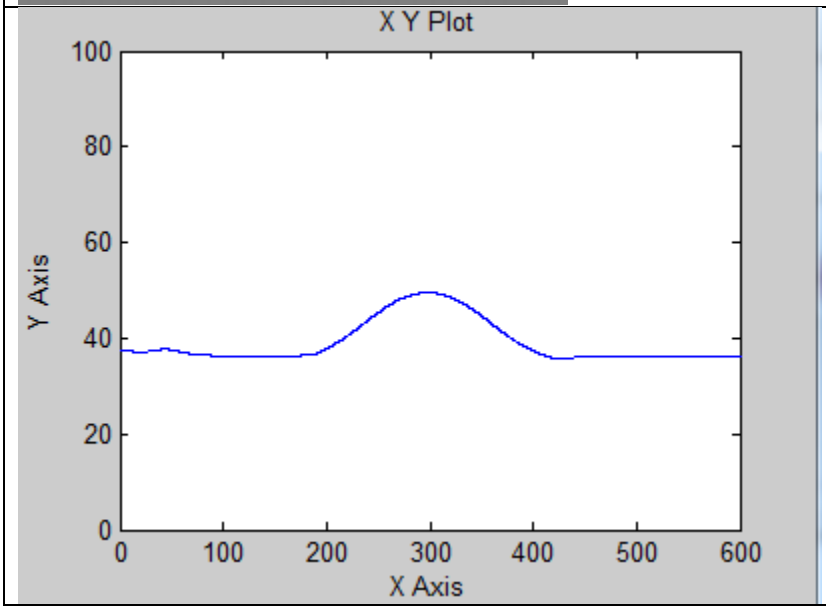
4. Once the excel file is selected, a pop up window will ask for *the number of header lines*. This is the number of rows to avoid which contain identifiers such as headings of altitude, latitude and longitude column. This number is based on the creation of the excel file.
5. After step 4, the columns of data will be displayed in the dark box above the Create Basic KML.
6. Each column is required to be allocated in the *Select Latitude, Select Longitude and Select Altitude* drop down lists. The contents within this list is acquired from on *the number of header lines* provided earlier in step 4.

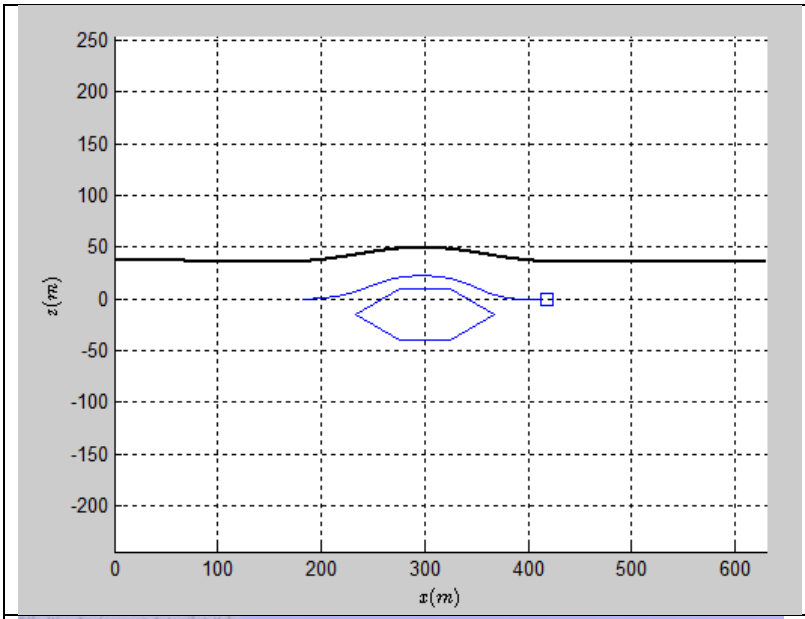
7. Additional relevant information can be set such as the line colour and units with the KML creator but can also be set in Google Earth.
8. Click the Create Basic KML file to create the file.
9. Open KML file with google earth. The aircraft path can be seen in google Earth.

Appendix D1 (Pitch vs Bank Decision)

```
Function[heading_angle1,heading_angle2]=fcn(X,Y,X_2,Y_2,dangle,pangle, bangle,dangle2,alt)


a= max(Y)- min(Y);     //Maximum distance covered in Y direction during both maneuver
b= max(X)- min(X);    //Maximum distance covered in X direction during both maneuver
B_2= max(X_2)- min(X_2); //Maximum distance covered in Z direction during both maneuver

//Comparing distances

if (((a/2)+b) > (max(Y_2) + B_2 - alt))      // If bank angle is greater

  heading_angle1=pangle;      // send new pitch angle to avoid obstacle
  heading_angle2=dangle2;     // sending default angle to avoid bank maneuver

elseif (((a/2)+b) < (max(Y_2) + B_2 - alt))     // If pitch angle is greater

  heading_angle1=dangle;     // sending default angle to avoid pitch maneuver
  heading_angle2=bangle;     // send new bank angle to avoid obstacle

else
  heading_angle1=dangle;
  heading_angle2=dangle2;
end
```

Appendix E1 (Stable Flight)

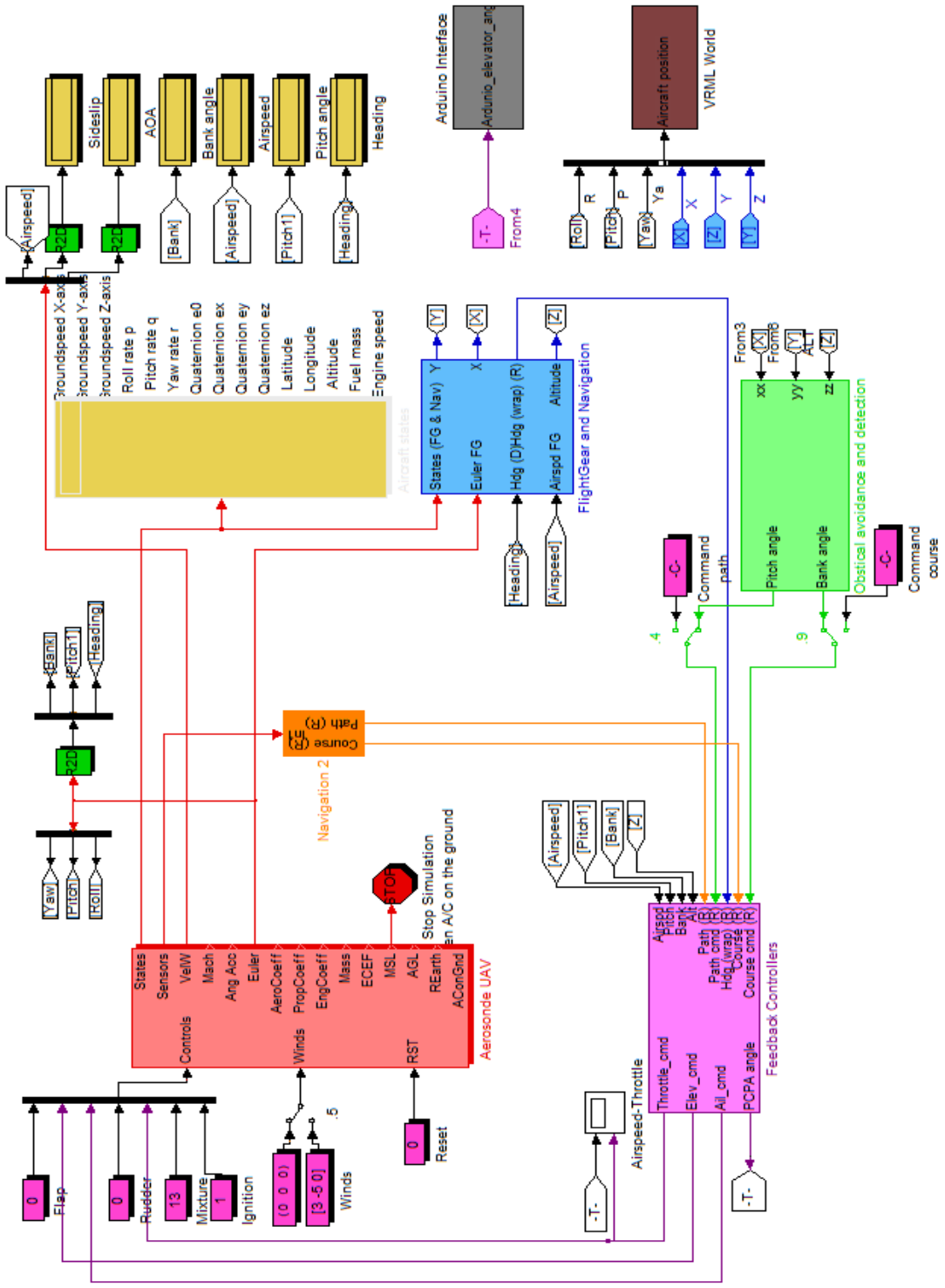
	Stable Flight: Airspeed
	Stable Flight: Pitch Angle
	Stable Flight: Obstacle avoidance maneuver , XY plane implementing fly around maneuver.

	Stable Flight: Obstacle avoidance maneuver , ZX plane implementing pitch up maneuver maneuver.
	Stable Flight: straight and level flight observed via FlightGear

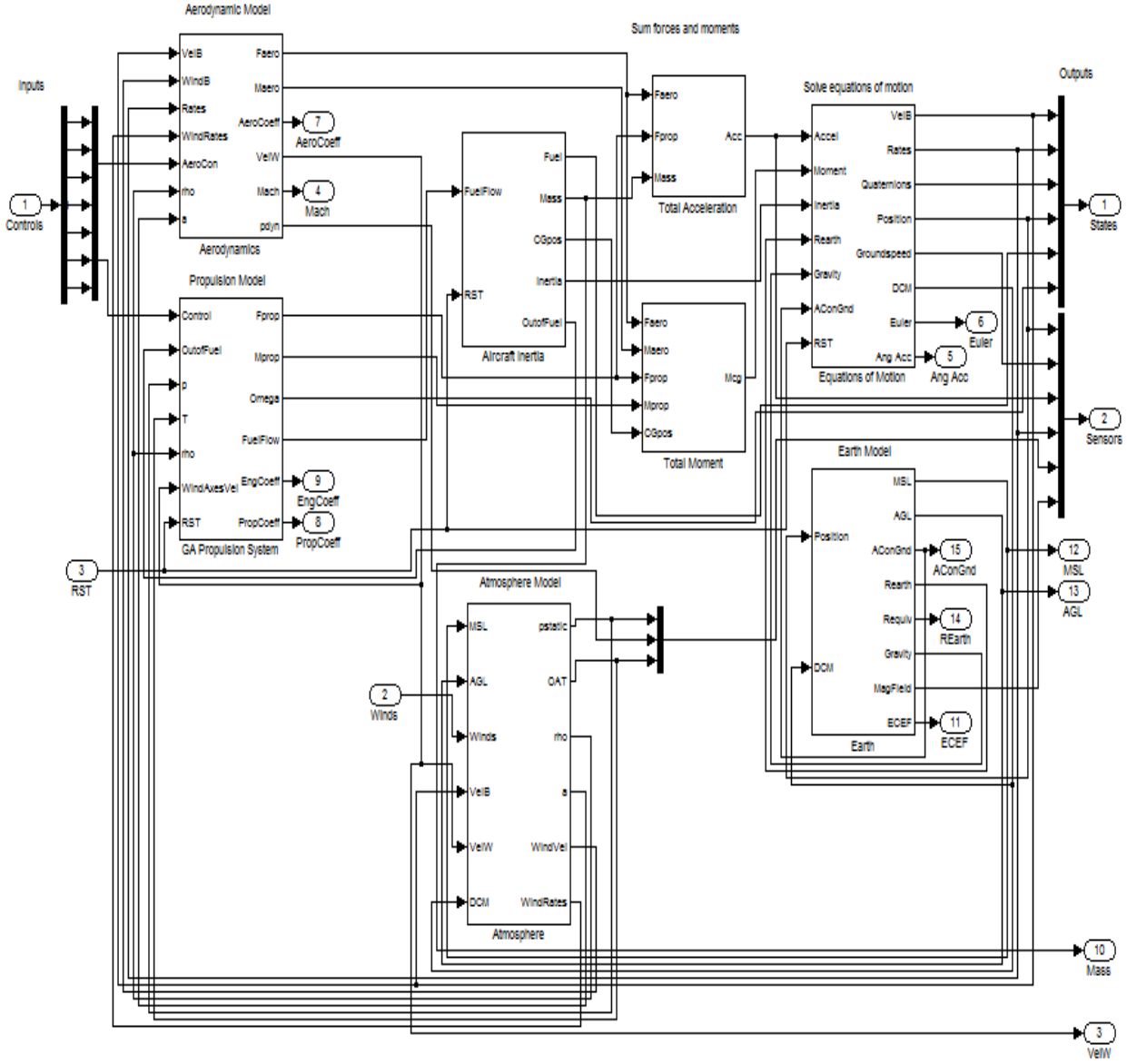
	Stable Flight: Aircraft performing bank angle which is observed via FlightGear

Appendix E2 (Models)

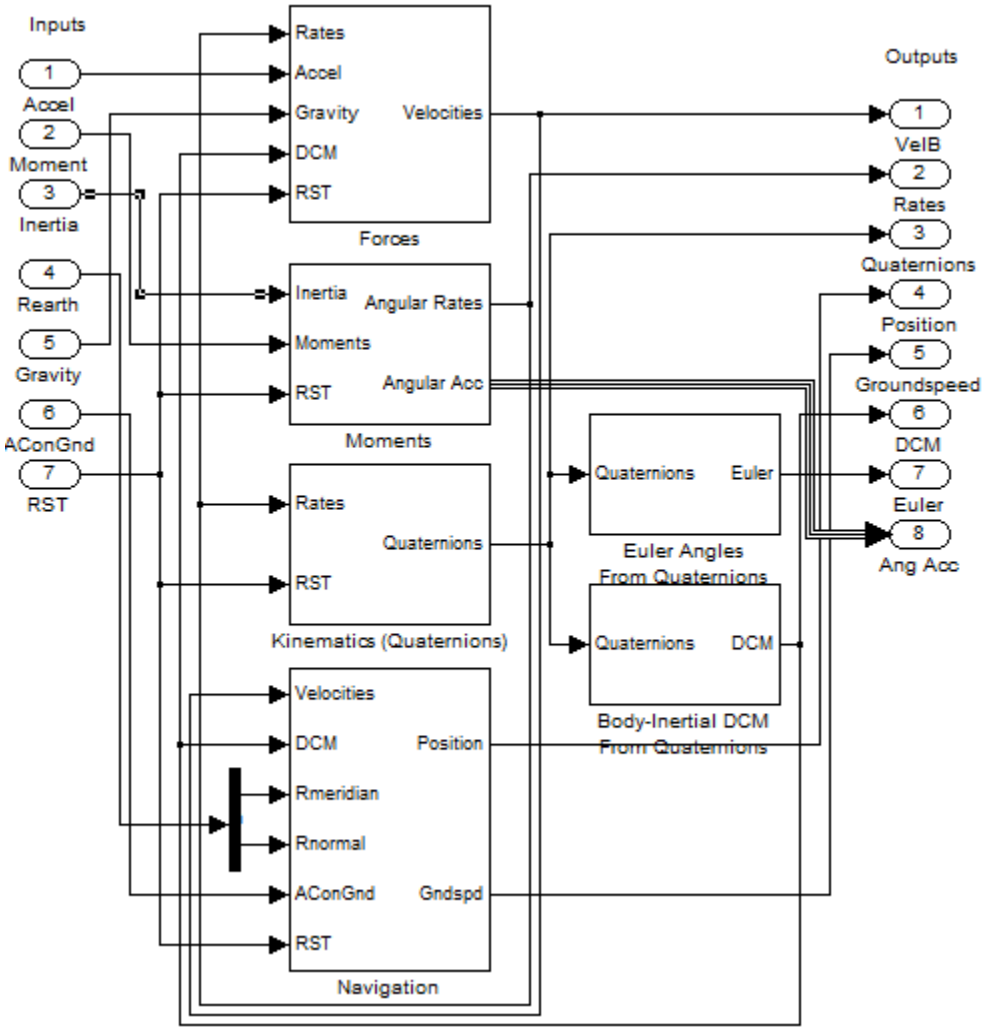
Completed simulation framework



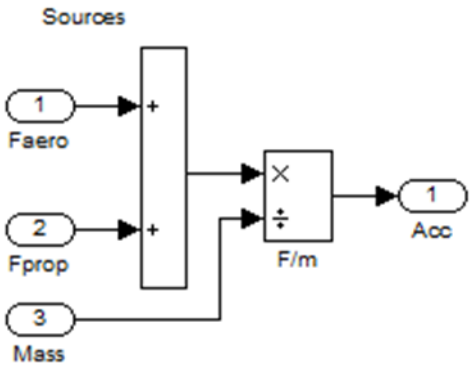
Inside non-linear model of Aerosonde UAV



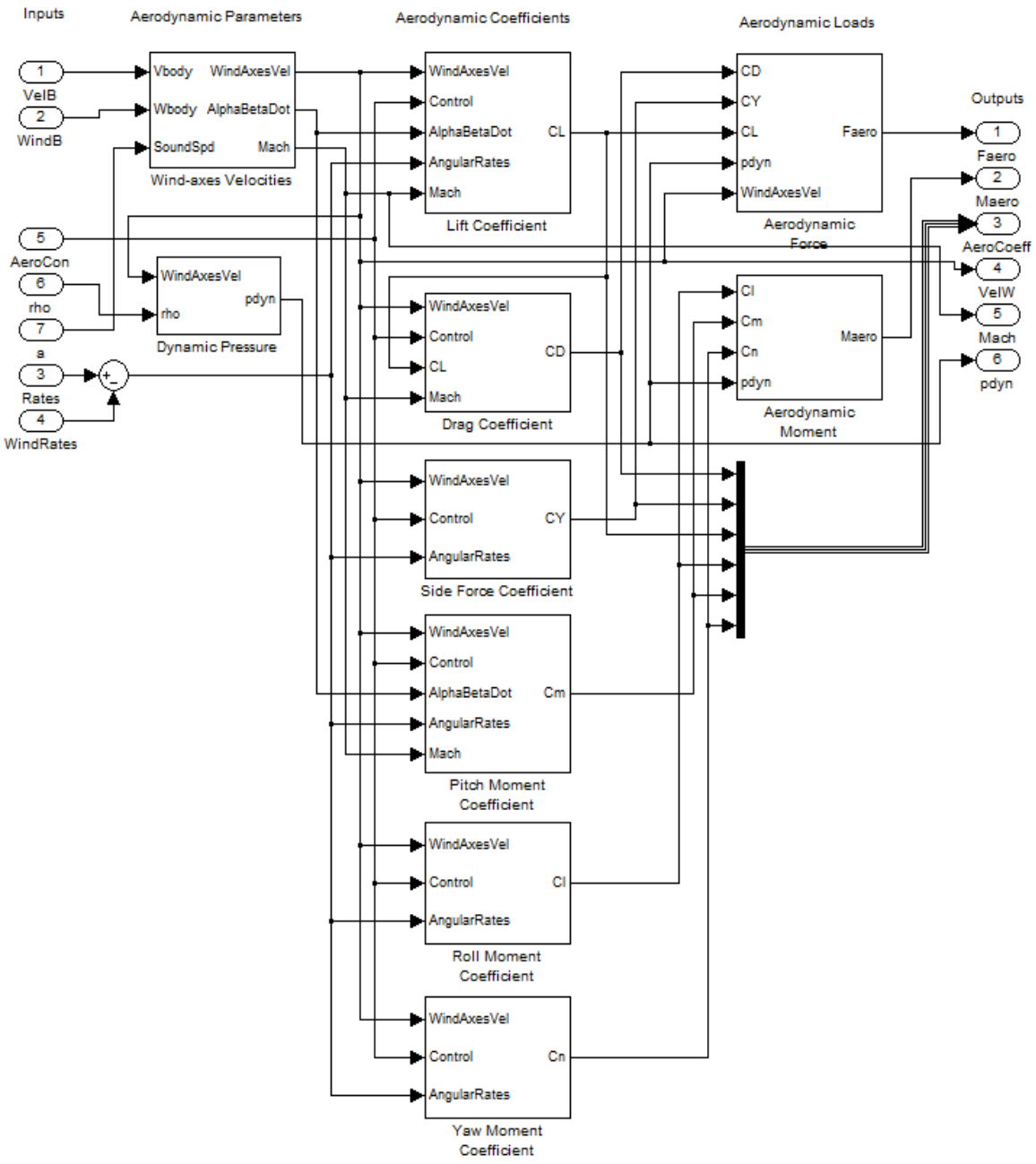
Inside of Equation of Motion Block



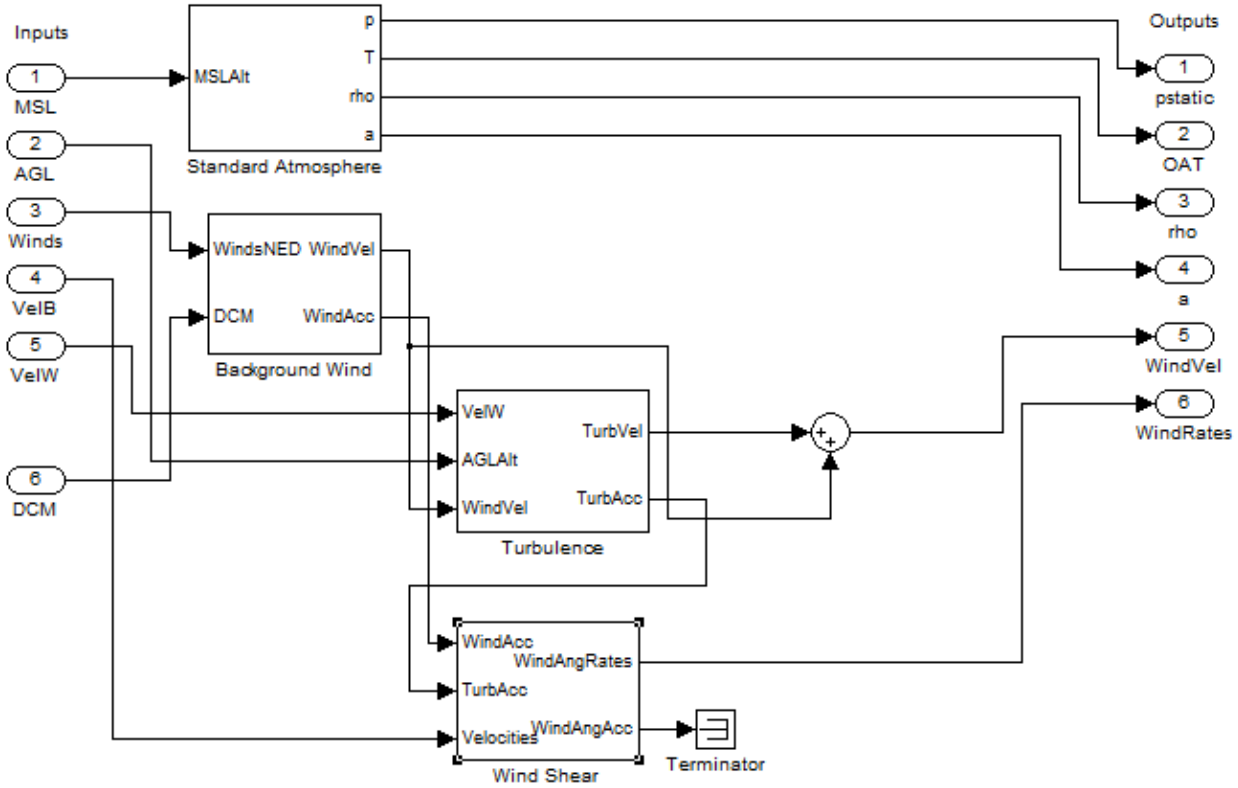
Inside of Total Acceleration Block



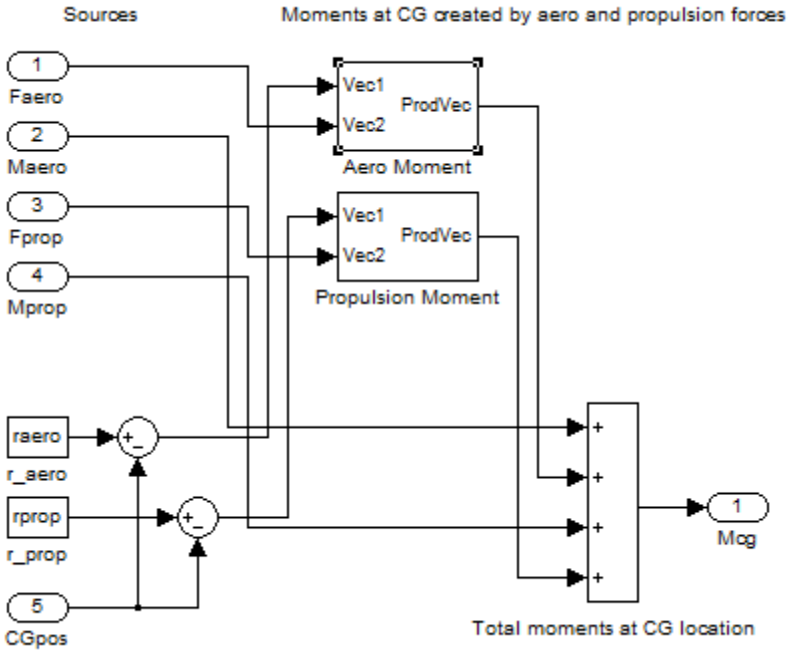
Inside of Aerodynamics Block



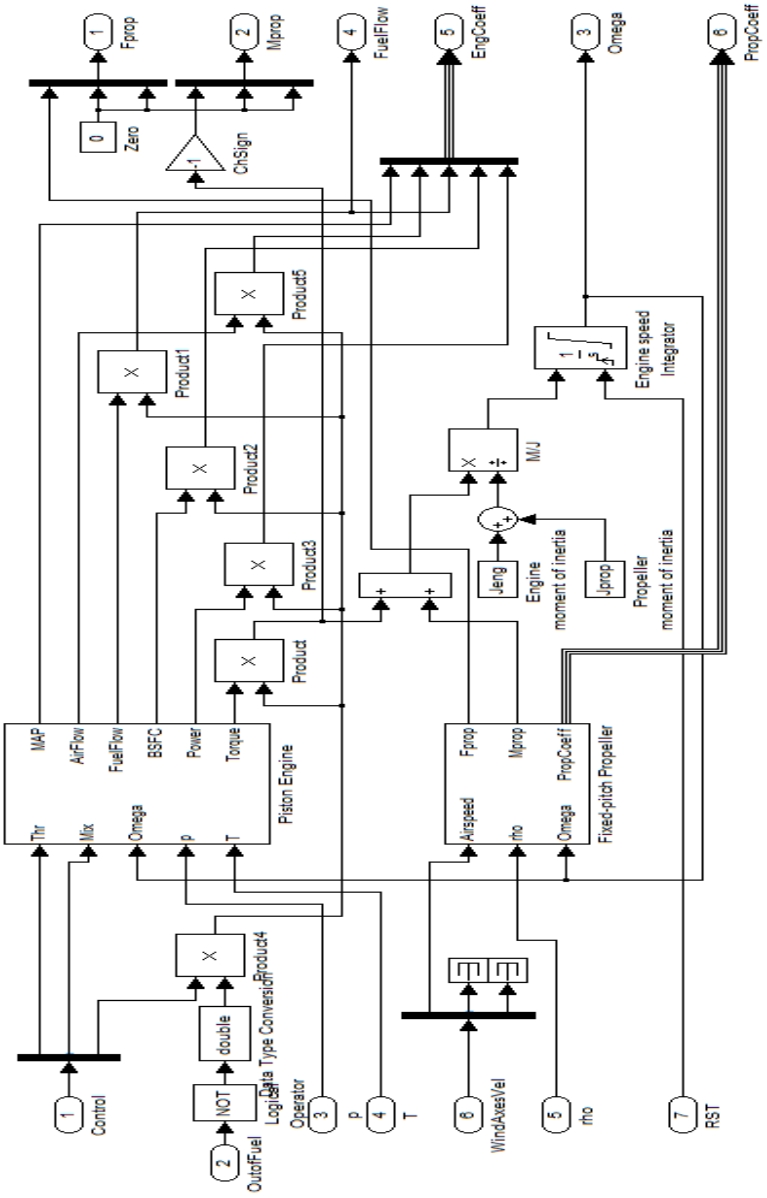
Insides of the Atmosphere Block



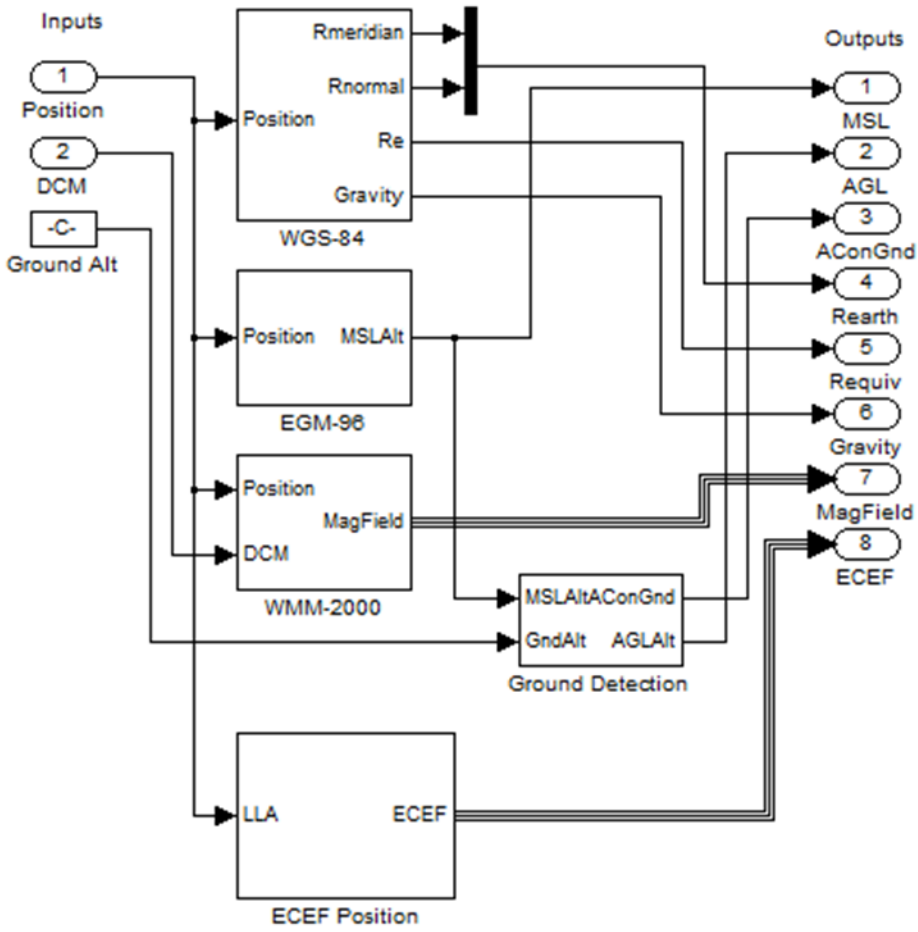
Inside of Total Moment Block



Inside of Propulsion system block



Inside of Earth Model Block



Appendix E3 (Inputs, Outputs)

Inputs and outputs of Equation of motion block

Inputs	**Outputs**
Acceleration	Velocities in Body frame
Moment	Angular rates
Inertia	Quaternions
Earths equivalent radius	Position (Latitude, Longitude, altitude)
Gravity	Ground Speed
The aircraft on ground flag	DCM
Reset	Angular Acceleration

Inputs and outputs of Aerodynamics block

Input	**Output**
Velocities in Body frame	Aerodynamic forces
Velocities in Wind frame	Aerodynamic moments
Angular rates	Aerodynamic Coefficients
Wind rates	Aircraft velocities in wind axes [airspeed sideslip angle-of-attack]
Control surface and throttle	Mach
Air density	Dynamic Pressure
Speed of sound	

Inputs and outputs of GA Propulsion System

Input	**Output**
Throttle mixture ignition	The 3x1 vector of propulsion system forces applied to the airframe, at the propeller hub location
The out-of-fuel flag (0 or 1)	The 3x1 vector of propulsion system moments applied to the airframe
Static pressure at current altitude (Pa)	The engine shaft rotation speed (rad/s)
Air temperature at current altitude (K)	The mass fuel flow (kg/s)
Air density at current altitude, (kg/m^3)	The 5x1 engine coefficient vector [map airflow fuelflow bsfc power] (kpa kg/s kg/s g/(w*hr) w)
Airspeed Sideslip Angle-	The 3x1 propeller coefficient vector (J

of-attack] [m/s rad rad]	CT CP)
Reset flag	

Inputs and outputs of Aircraft Inertia [Aerosim Blockset]

Input	Output
Fuel flow	Current aircraft mass;
Reset flag	Current aircraft CG location;
	Current aircraft moments of inertia.
	Flag that gets set to 1 if the aircraft is out of fuel.

Inputs and outputs of Atmosphere Block [Aerosim Blockset]

Input	Output
Mean-sea-level altitude	Static pressure
Altitude of the aircraft above ground	Outside air temperature
3x1 vector of wind speed components [North East Down]	Air density
3x1 vector of aircraft velocities in body axes	Speed of sound
3x1 vector of aircraft velocities in wind axes [Airspeed Sideslip Angle-of-attack]	3x1 vector of wind velocity in body axes
DCM	3x1 vector of angular rates due to wind

Inputs and outputs of Total Acceleration Block

Input	Output
3x1 vector of aerodynamic forces	3x1 vector of accelerations in body frame
3x1 vector of propulsion forces	
Current aircraft mass	

Inputs and outputs of Earth Model Block

Inputs	Outputs
Position location [Latitude Longitude Altitude]	Mean-sea-level altitude
DCM	Altitude of the aircraft above ground

	Aircraft-on-ground flag (0 if aircraft above ground, 1 if aircraft on the ground)
	2 x 1 earth meridian radius and Earth normal radius
	Earth equivalent radius
	Normal gravity
	3x1 vector of magnetic field components in body axes
	3x1 vector of ECEF position (X, Y, Z).

Appendix E4 (Constants to be modified)

Aerodynamic constant variables

AERODYNAMICS CONSTANTS	Variable Name	Unit
Aerodynamic force application point (usually the aerodynamic Centre)[x y z]	rAC	m
Aerodynamic parameter bounds		
Airspeed bounds	VaBnd	m/s
Sideslip angle bounds	BetaBnd	rad
Angle of attack bounds	AlphaBnd	rad
Aerodynamic reference parameters		
Mean aerodynamic chord	MAC	m
Wind span	b	m
Wing area	S	m2
Aerodynamics derivatives		
Lift coefficient		
Zero-alpha lift	CL0	rad
alpha derivative	CLa	rad
Lift control (flap) derivative	CLdf	rad
Pitch control (elevator) derivative	CLde	rad
alpha-dot derivative	CLalphadot	rad
Pitch rate derivative	CLq	rad
Mach number derivative	CLM	rad
Drag coefficient		
Lift at minimum drag	Clmind	rad
Minimum drag	CDmin	rad
Lift control (flap) derivative	CDdf	rad
Pitch control (elevator) derivative	CDde	rad
Bank control (aileron) derivative	Cdda	rad
Yaw control (rudder) derivative	CDdr	rad
Mach number derivative	CDM	rad
Oswald's coefficient	osw	rad
Side force coefficient		
Sideslip derivative	Cybeta	rad
Bank control derivative	Cyda	rad
Yaw control derivative	Cydr	rad
Bank rate derivative	Cyp	rad

Yaw rate derivative	Cyr	rad
Pitch moment coefficient		
Zero-alpha lift	Cm0	rad
alpha derivative	Cma	rad
Lift control (flap) derivative	Cmdf	rad
Pitch control (elevator) derivative	Cmde	rad
alpha-dot derivative	Cmalphadot	rad
Pitch rate derivative	Cmq	rad
Mach number derivative	CmM	rad
Bank moment coefficient		
Sideslip derivative	Clbeta	rad
Bank control derivative	Clda	rad
Yaw control derivative	Cldr	rad
Bank rate derivative	Clp	rad
Yaw rate derivative	Clr	rad
Yaw moment coefficient		
Sideslip derivative	Cnbeta	rad
Bank control derivative	Cnda	rad
Yaw control derivative	Cndr	rad
Bank rate derivative	Cnp	rad
Yaw rate derivative	Cnr	rad

PROPELLER CONSTANTS	Variable Name	Unit
Propulsion force application point (usually propeller hub) [x y z]	rHub	m
Advance ratio vector	J	1x16 vector
Coefficient of thrust look-up table CT = CT(J)	CT	1x16 vector
Coefficient of power look-up table CP = CP(J)	CP	1x16 vector
Propeller radius	Rprop	m
Propeller moment of inertia	Jprop	kg*m^2

Appendix F (Virtual Reality Modeling Language)

Virtual Reality Modeling Language (VRML) is an international standard for describing a 3D virtual environment on the World Wide Web. VRML is used for various purposes, examples of which include web-based entertainment, distributed visualization, 3D user interface, Interactive simulation for education and many more. VRML is commonly used in all fields to create virtual models. Hence, off-the-shelf VRML models and systems are readily available [Gerstmann, 2000].

The usage of VRML is mainly used in industrial applications. For example, Design Phase, Prototype Making, Prototype Simulation, Testing, Visualization and Manufacturing could be accomplished with the help of VRML [Tan, 1999]. There are also few researches that incorporate VRML into the UAV field. However, most of this research uses VRML to observe the flight dynamics of the UAV, hence being used as an alternative to FlightGear. They also use VRML to map the trajectory of the UAV. Majority of the demos provided by Matlab also focuses on creating dynamic system behaviour of models in a virtual reality environment. Nevertheless, the *Vehicle Dynamic Simulation* model, which is a demo model provided in Simulink library, deals with an additional component which is acquiring the driver's view of a moving car. This is done by creating a moving camera within the virtual environment. In this research, the concept of a moving camera is used to acquire a suitable video for image processing.

There are various alternative to VRML such as COLLADA, O3D and 3DMLW. The 3D Markup language for Web (3DMLW) is open source software which can be used to develop 3D models and publish them on the web. The 3D object can be created and published with no advanced

coding or programming skills. O3D is also an open source software developed and published by Google that is used to develop 3D interactive graphic application that can be opened in the web browser. A user of O3D requires the ability to write java scripts, which would make programing in VRML or 3DMLW a simpler option for this research. There are other alternatives that can be used to create 3D environments, however, only VRML is readily compatible with Simulink. Thus, VRML is the ideal candidate for this research.

This section will introduce the reader to VRML concepts and terminology. It will also introduce the reader to the programing aspects of the language.

A typical VRML scene is composed of nodes. Nodes describe the physical elements such as shapes, light and sound within the environment. Generally, once a node is created, these nodes are replicated, if they are required. For example, if a box is defined in a room and if there were to be or more that one of the same box in the room, the initially created box can be called again during programming.

The children syntax can be used to make the box appear at a particular co-ordinate along with its orientation. This abstraction of data can be performed using “children” syntax. The sample code below shows how multiple instances of a node can be defined.

Example:

```
DEF Obstacles Group {
 children [
  Transform {
   translation 450 0 -222
   rotation 0 1 0 1.57
   children USE Box1
  }
  Transform {
```

Syntax

```
<Grouping Node> {
    children [
        <other nodes>
    ]
    <field>
    <field>
     .
     .
```

```
    translation 360 0 412                    .
    rotation 0 1 0 0.47                  }
    children USE Box1
}
  Transform {
    translation 547 0 -317
    rotation 0 1 0 1.97
    children USE Box1
```

There are various fields depending on the node, for example, an object has appearance, texture and material. As the names suggests fields defines the characteristics of the nodes.

The above concept of “children” only defines the set group of nodes but does not implement in the 3D environment. In order to place the object in the environment one must use the “Transform” syntax. The transform node applies scales, rotations and translations to its children. An example of transform syntax is shown below.

```
Transform {
    translation -9 0 10
    rotation 0 1 0 1.17
    children USE Box1
  }
```

The translation and rotation describes the location of the object and the children syntax provides parameters for defining the object.

To understand the translation and rotation, one must understand the co-ordinate system in the VRML world. VRML uses right handed co-ordinate system, where distances are measured in metres. Figure 94 shows the right handed co-ordinate system.

All objects by default are built at the origins of world co-ordinate system. Using the position syntax one can place the object anywhere in the co-ordinate system. However, while using the

"*Transform*", "*Position*" cannot be used but instead "*Translation*" should be used as shown in the previous syntax example. The numbers associated after are "Position" and "*Translation*" syntax are the location of the local origin at X, Y, Z axis.

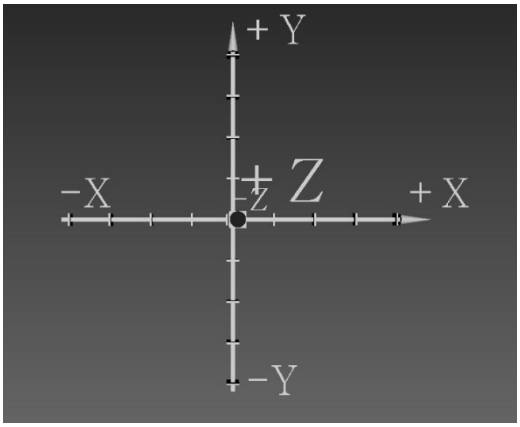


Figure 94: Co-ordinates of VRML world [VRML, 2013]

"*Rotation*" rotates the object around the axis. The numbers associated after "*Rotation*" syntax is the axis to be rotated and the amount of rotation in radians. The three examples below show different possible scenarios.

rotation 1 0 0 1.57 #the camera is rotated around the X axis 90 degrees

or

rotation 0 1 0 -1.57 #the camera is rotated around the Y axis -90 degrees

or

rotation 0 1 0 3.14 #the camera is rotated around the Y axis 180 degree

The two concepts of rotation and translation above mentioned are very important in animation aspects of the VRML world as they provide 3-DOF and 6-DOF i.e. changing the rotation translation parameters with the help of Simulink during simulation allows 6-DOF of the defined

camera within the VRML world. This allows complete freedom of movement of an object created in the 3D environment. Next section shows how the objects in the VRML world can be moved by merely changing the values under "*Rotation*" and "*Translation*" fields.

This thesis only covers the basic elements of the VRML programming capabilities as most of the programing/logic that involves animation is implemented using Simulink. It is simpler and more effective to develop programming logic using Simulink blocksets rather than composing multiple pages of VRML code, since the author is more familiar with graphical programming over typical object oriented programming.