WILD Goblin Sensor Pod Design, Development, and Integration

December 15, 2014

Submitted by:

Lillian Walker, lgwalker@wpi.edu, WPI box number: 1421

Daniel Zaleski, drzaleski@wpi.edu, WPI box number: 1909

Advised by:

WPI Advisor: Professor Fred Looft

MIT Lincoln Laboratory Supervisor: Bryce Remesch

Table of Contents

Table of Figures

Table of Tables

Table of Equations

Section 1. Introduction

Unmanned Aerial Vehicles (UAVs) have proven to be critical components of many military and civilian missions, both at home and abroad. One major mission area UAVs contribute significant value to is Intelligence, Surveillance and Reconnaissance (ISR). ISR capabilities are essential for exploring terrain, gathering intelligence, and finding objects of interest. The information UAVs can provide while flying over unknown territories can make the difference between mission success and failure.

Several inherent traits of UAVs make them ideal for ISR missions. Since UAVs do not have a pilot, the risk of human causalities in reconnaissance missions decreases with the increased use of UAVs for high risk missions and geographic areas. UAVs are usually smaller than a light aircraft (e.g. a small private airplane), so they are capable of quick and agile maneuvers through the air that a normal manned aircraft would not be able to easily accomplish. Finally, UAVs are often smaller and relatively quiet and, as a result, are less observable from the ground than large piloted aircraft.

The Wing and Internal Launcher Deployed (WILD) Goblin is a small UAV in development at Lincoln Laboratory that is intended for use for ISR missions including autonomous reconnaissance. This UAV is composed of a BAE Systems Coyote airframe as well as specially selected sensors used for tracking objects of interest and developing sparse point clouds of objects.

The purpose of our capstone project was to, first, design a sensor pod turret assembly which can house the specific set of sensors that were selected for the WILD Goblin's missions, and second, to develop a software package that can be used with the Goblin's turret assembly to control and stabilize the turret that the sensors will be mounted in. While many Commercial Off-The-Shelf (COTS) UAV turrets and sensor packages are available, they are not optimized for the WILD Goblin's restricted size, weight, and power (SWaP) constraints. As a result, this project was focused on a custom turret design and software stabilization/control solution designed specifically to meet the WILD Goblin system requirements.

Section 2. Background

Modern reconnaissance unmanned vehicles, like the WILD Goblin, are equipped with advanced sensor technology which enables them to gather valuable intelligence and provide many advantages over manned aircraft. Since UAVs do not need to accommodate a pilot, with the corresponding weight, power, volume, safety, and other requirements, UAVs allow for a wider variety of aircraft sizes, designs, operation areas, and deployment capabilities.

2.1. History of Unmanned Aerial Vehicles

Ever since manned flight was proven to be possible, inventors have been seeking to implement and improve unmanned flight. During the Civil War, balloons, such as shown in Figure 1, were used for manned reconnaissance (surveying enemy terrain) as well as unmanned bomb dropping vehicles. Both the Union and the Confederacy would launch these balloons and rely on wind patterns to bring the balloon down behind enemy lines and, hopefully, destroy a target of value [1]. Charles Perley designed an aerial bomber in 1863, a hot air balloon equipped with a mechanical timer which could be preset to drop a bomb at a specific time after launch. These bombers were used by both Union and Confederate forces, but had limited success as they were inaccurate, because they relied on wind patterns, and dangerous, due to the potential for premature explosions [2].

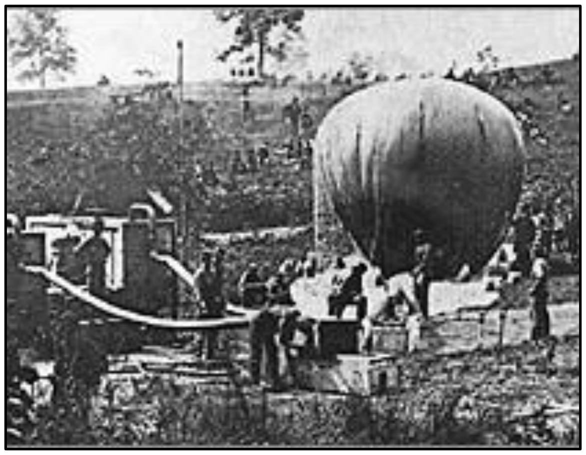


Figure 1: Civil War Balloon [2]

Many of the first unmanned aerial vehicles were the result of researchers adapting existing aircraft to fly autonomously using mechanical devices. In 1917, Elmer Sperry invented an automatic gyroscopic stabilizer which could keep an airplane level during flight [2]. The same device could also be used to cause the aircraft to crash after a length of time, sending it crashing into a target. Near the end of World War I, branches of the United States military began investing in research into UAVs like Sperry's to be used as disposable bomb delivery aircraft. Figure 2 shows the Kettering "Bug," purchased by the United States Army, made by Charles F. Kettering and Orville Wright, which had a counter on the propeller that cut power to the engine after a certain number of propeller revolutions. The end of World War I, however, brought an early end to most of the interest in UAV research [3].

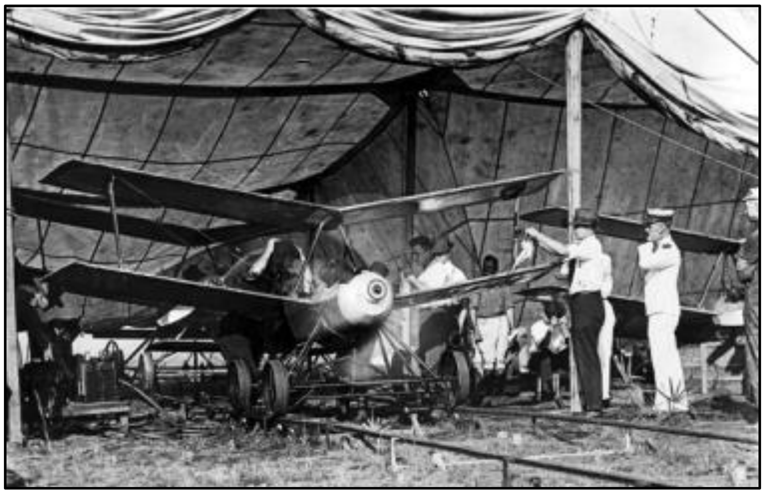


Figure 2: The Kettering "Bug" [4]

During World War II, UAV research experienced a resurgence as military branches in several countries began using unmanned aircraft for target practice. These remote control UAVs were flown by ground or ship based operators who attempted to elude anti-aircraft personnel for practice. In the 1930's, the British military developed the Queen Bee, shown in Figure 3, a multi-use remote control UAV which was used to train the Royal Navy anti-aircraft gunners. The Queen Bees were wooden biplanes that had a range of 300 miles and could fly as high as 17,000 feet [2]. After seeing the British Navy's use of UAVs, the US military invested in similar technology which was produced by Radioplane Company, a toy aircraft manufacturer. These target UAVs were essentially larger versions of remote control toy aircraft with wingspans up to 12 feet 3inches and lengths up to 9 feet 3inches. Both the US Army and US Navy invested in these target UAVs for training [3].

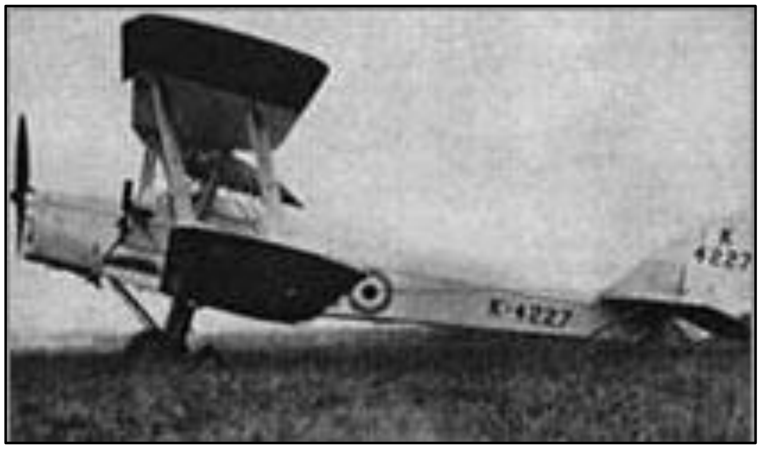


Figure 3: The Queen Bee [2]

UAVs were also used by both the Allies and Germany during World War II as remote-control bomb carrying vehicles. In 1944, Germany employed unmanned rockets, the V-1 and V-2, during the bombing of London. The V-1 carrying a 2,000 pound warhead was launched from a ramp and could be preprogrammed to fly a specified distance before dropping its bomb [2]. The US forces also developed UAVs for remote attacks during World War II by repurposing older planes to be remote controlled. For example, during Operation Aphrodite, pilots would takeoff in converted B-17 or other bombers filled with 25,000 pounds of explosives, then bail out while a remote operator piloted the UAV towards a target. These missions were rarely successful, however, as on many missions the remote control malfunctioned, the plane was shot down, or the payload exploded prematurely [3].

In the 1950's and 60's the United States began using UAVs for reconnaissance. Some reconnaissance UAVs were the same ones used for target practice during World War II, but were modified to carry a variety of sensors. The Firebee, built by Ryan Aeronautical Company in 1948, was the first to be adapted for reconnaissance missions in 1959. One model of the Firebee, the Q-2C, could travel at heights of 60,000 feet at a range of 800 miles. These UAVs were equipped with conventional film cameras, making recovery of the cameras necessary for the intelligence to be useful. Many of the Firebee models were designed to be recoverable and some models had over a 90 percent return rate [3].

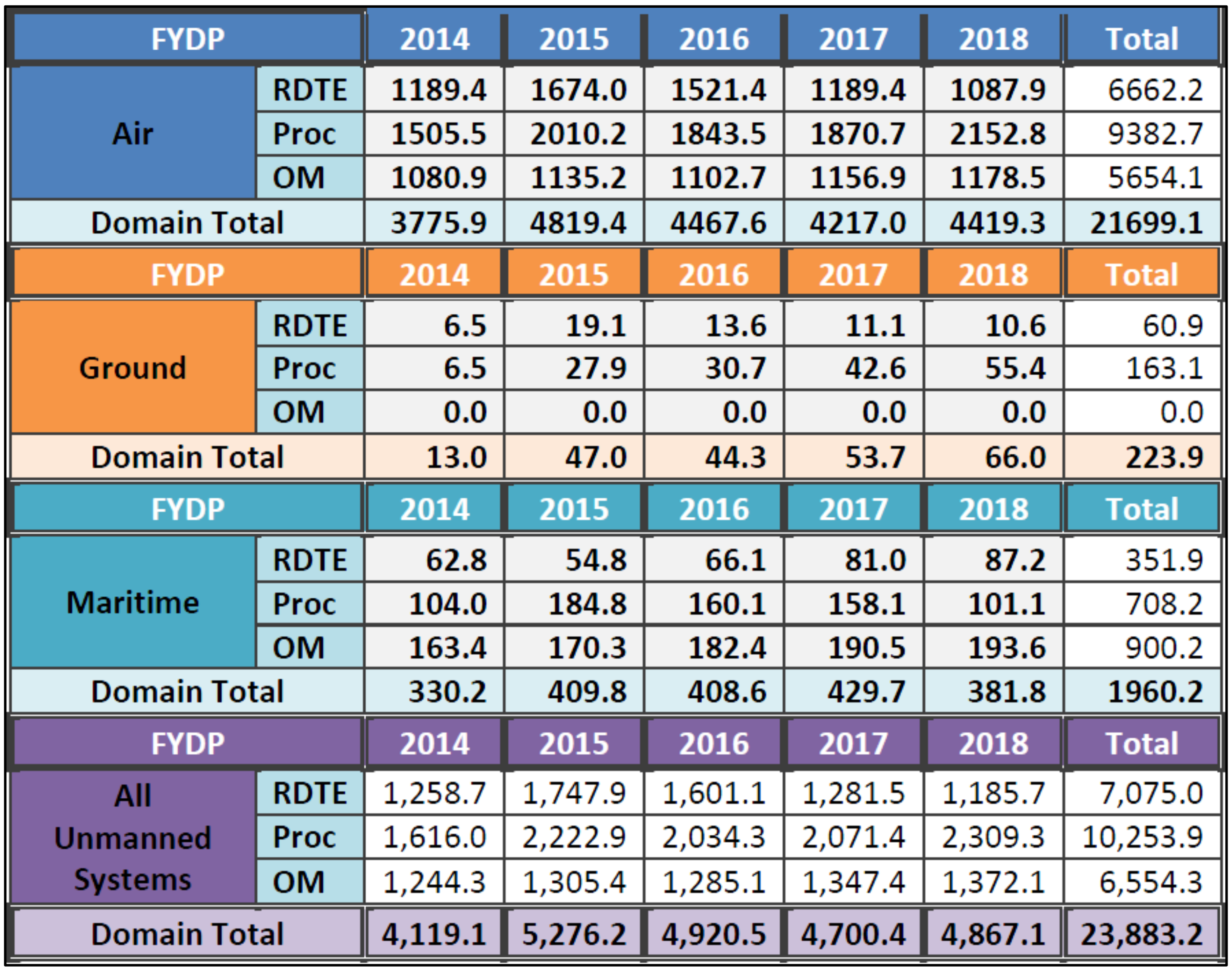
The Q-2C was followed by the Ryan Fire Fly 147A and 147B, which were designed to fly missions over Russia undetected. These "stealth" UAV were equipped with radar absorbing materials and other devices to disguise the aircraft from radar. Ryan Aeronautical continued to design many more models of the Fire Fly to fit a wide range of mission scenarios, high and low altitudes, and ranges. The Fire Fly continued to be used in Vietnam for reconnaissance, taking the place of human operators during dangerous missions, including day and night missions to take photographs of terrain and enemy locations [3].

2.2. Modern UAVs

In the last 15 years of the 20th century and up to modern day, UAVs have constantly been developed and implemented for various missions. New technology for UAVs continues to be created, leading to yet more uses for unmanned aerial vehicles.

Uses of UAVs

Unmanned systems (US) are very common in today's military and are used for a wide variety of missions. Each mission has specific operating conditions which determine the requirements for the unmanned system to be used. Unmanned systems are used not only for airborne platforms, but also for ground based platforms and both underwater and surface sea platforms [5]. As shown in Figure 4, air platforms receive the majority of funding for unmanned systems; for financial years 2014-2018, Unmanned aerial systems (UAS) account for 90.9% of the 23.9 billion dollars in funding designated for unmanned systems, likely because UAS are the most versatile, able to operate from ground or sea bases and used by all branches of the military. Unmanned ground systems (UGS) receive 0.9% of the budget and unmanned maritime systems (UMS) receive 8.2%.



FYDP		2014	2015	2016	2017	2018	Total
Air	RDTE	1189.4	1674.0	1521.4	1189.4	1087.9	6662.2
	Proc	1505.5	2010.2	1843.5	1870.7	2152.8	9382.7
	OM	1080.9	1135.2	1102.7	1156.9	1178.5	5654.1
Domain Total		3775.9	4819.4	4467.6	4217.0	4419.3	21699.1
FYDP		2014	2015	2016	2017	2018	Total
Ground	RDTE	6.5	19.1	13.6	11.1	10.6	60.9
	Proc	6.5	27.9	30.7	42.6	55.4	163.1
	OM	0.0	0.0	0.0	0.0	0.0	0.0
Domain Total		13.0	47.0	44.3	53.7	66.0	223.9
FYDP		2014	2015	2016	2017	2018	Total
Maritime	RDTE	62.8	54.8	66.1	81.0	87.2	351.9
	Proc	104.0	184.8	160.1	158.1	101.1	708.2
	OM	163.4	170.3	182.4	190.5	193.6	900.2
Domain Total		330.2	409.8	408.6	429.7	381.8	1960.2
FYDP		2014	2015	2016	2017	2018	Total
All Unmanned Systems	RDTE	1,258.7	1,747.9	1,601.1	1,281.5	1,185.7	7,075.0
	Proc	1,616.0	2,222.9	2,034.3	2,071.4	2,309.3	10,253.9
	OM	1,244.3	1,305.4	1,285.1	1,347.4	1,372.1	6,554.3
Domain Total		4,119.1	5,276.2	4,920.5	4,700.4	4,867.1	23,883.2

Figure 4: 2014-2018 President's Budget for Unmanned Systems ($ Mil) [6]

Figure 5 shows a breakdown of UAS flight hours since 1996. While the Air Force is responsible for the majority of flight hours, the Army has a comparable number of flights and the Navy and Marine Corps are also strongly represented.

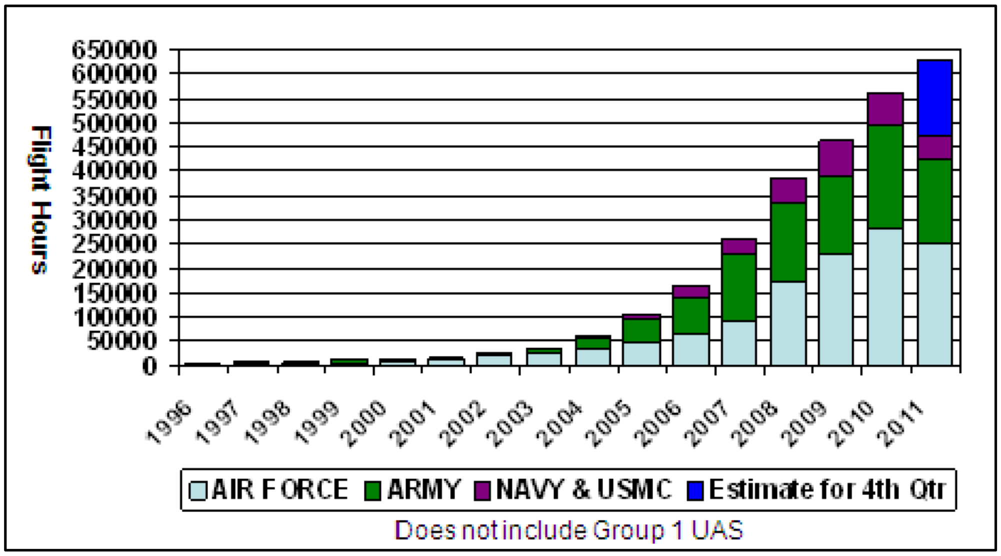


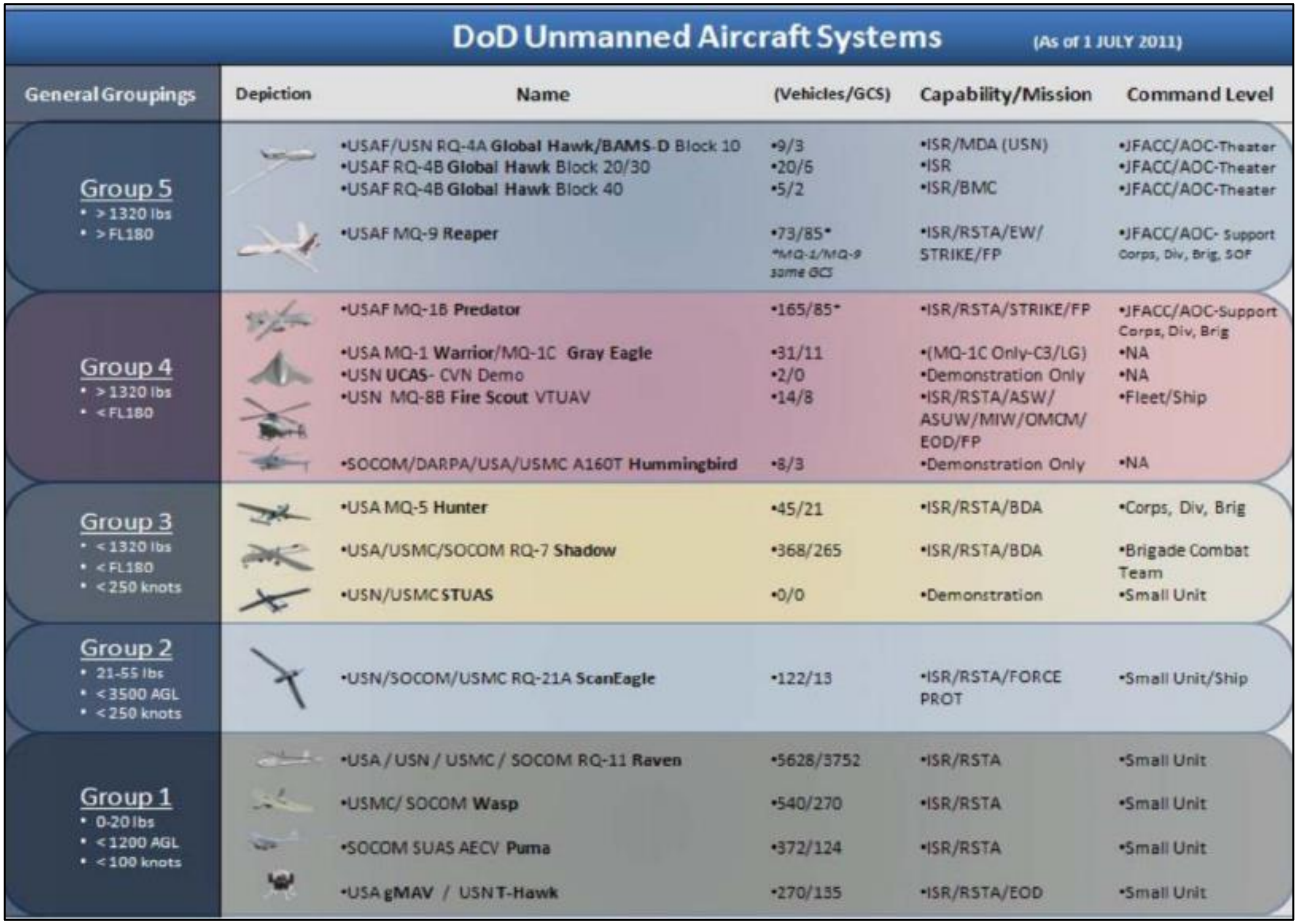
Figure 5: UAS Flight Hours (1996-2011) [5]

UAV Capabilities

Each UAV system is a factor of many components which are determined during design based on the eventual mission area of the UAV. These components include UAV size, launch method, recovery method, payload capability and level of autonomy.

Vehicle Design

The wide range of mission requirements for UAVs calls for a diverse selection of vehicles with different payload capacities, speeds, ranges, and sizes. Figure 6 shows a chart from the Department of Defense (DoD) Unmanned Systems Integrated Roadmap of current Department of Defense UAV projects sorted into groups according to weight, speed, and other factors [5].



DoD Unmanned Aircraft Systems (As of 1 JULY 2011)

General Groupings	Depiction	Name	(Vehicles/GCS)	Capability/Mission	Command Level
Group 5 • >1320 lbs • >FL180		•USAF/USN RQ-4A **Global Hawk/BAMS-D** Block 10	•9/3	•ISR/MDA (USN)	•JFACC/AOC-Theater
		•USAF RQ-4B **Global Hawk** Block 20/30	•20/6	•ISR	•JFACC/AOC-Theater
		•USAF RQ-4B **Global Hawk** Block 40	•5/2	•ISR/BMC	•JFACC/AOC-Theater
		•USAF MQ-9 **Reaper**	•73/85* *MQ-1/MQ-9 same GCS	•ISR/RSTA/EW/ STRIKE/FP	•JFACC/AOC- Support Corps, Div, Brig, SOF
Group 4 • >1320 lbs • <FL180		•USAF MQ-1B **Predator**	•165/85*	•ISR/RSTA/STRIKE/FP	•JFACC/AOC-Support Corps, Div, Brig
		•USA MQ-1 **Warrior**/MQ-1C **Gray Eagle**	•31/11	•(MQ-1C Only-C3/LG)	•NA
		•USN **UCAS**- CVN Demo	•2/0	•Demonstration Only	•NA
		•USN MQ-8B **Fire Scout** VTUAV	•14/8	•ISR/RSTA/ASW/ ASUW/MIW/OMCM/ EOD/FP	•Fleet/Ship
		•SOCOM/DARPA/USA/USMC A160T **Hummingbird**	•8/3	•Demonstration Only	•NA
Group 3 • <1320 lbs • <FL180 • <250 knots		•USA MQ-5 **Hunter**	•45/21	•ISR/RSTA/BDA	•Corps, Div, Brig
		•USA/USMC/SOCOM RQ-7 **Shadow**	•368/265	•ISR/RSTA/BDA	•Brigade Combat Team
		•USN/USMC **STUAS**	•0/0	•Demonstration	•Small Unit
Group 2 • 21-55 lbs • <3500 AGL • <250 knots		•USN/SOCOM/USMC RQ-21A **ScanEagle**	•122/13	•ISR/RSTA/FORCE PROT	•Small Unit/Ship
Group 1 • 0-20 lbs • <1200 AGL • <100 knots		•USA / USN / USMC / SOCOM RQ-11 **Raven**	•5628/3752	•ISR/RSTA	•Small Unit
		•USMC/ SOCOM **Wasp**	•540/270	•ISR/RSTA	•Small Unit
		•SOCOM SUAS AECV **Puma**	•372/124	•ISR/RSTA	•Small Unit
		•USA **gMAV** / USN **T-Hawk**	•270/135	•ISR/RSTA/EOD	•Small Unit

Figure 6: DoD UAS [5]

Small Unmanned Aerial Vehicles (SUAVs) are of particular interest as vehicles for tactical reconnaissance. Figure 7 shows how prevalent Group 1 and 2 UAVs are in the Department of Defense, making up 89% of the DoD inventory. These aircraft can be as small as the Black Hornet Nano, a rotary wing vehicle which can fit in the palm of your hand [7]. The small size of SUAVs and more limited payload mean that for many scenarios, these aircraft cost far less than a Group 4 or 5 UAV while still providing the essential instruments and sensors that are needed for a mission. These UAVs can be inexpensive enough to be expendable, so they do not endanger ground forces involved in retrieving the SUAV. SUAVs can also often be deployed at a lower organizational level, meaning that a small squad of soldiers could be equipped with useful reconnaissance tools.

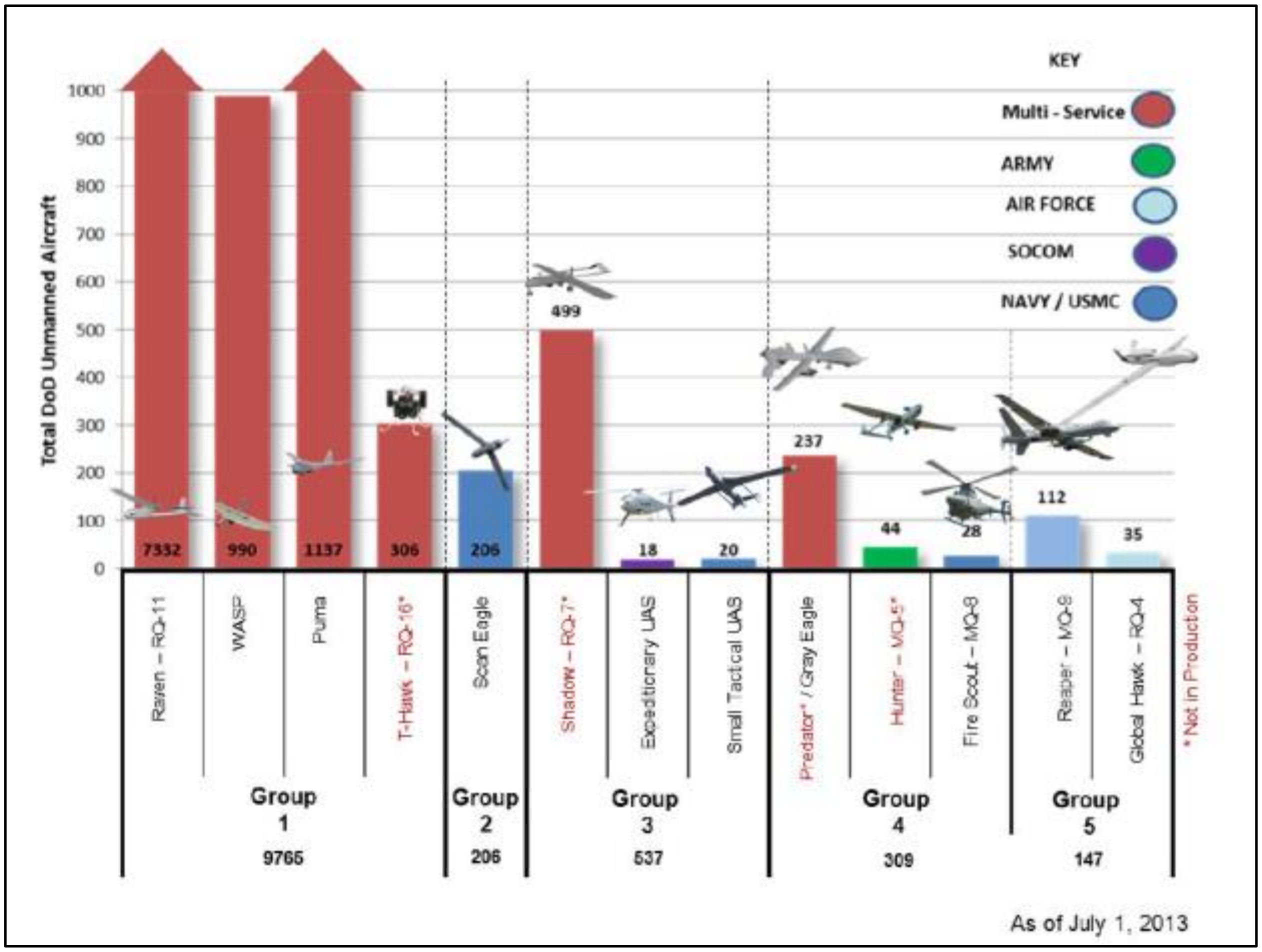


Figure 7: Inventory of DoD UAS [6]

Launch Methods

Depending on their size, UAVs can be launched in a variety of ways. Some UAVs can be launched much like other large aircraft using runways, although this requires space for takeoff. Others can be hand launched, which is useful for Group 1 aircraft which are light. Vertical launches are also used for helicopter like aircraft with rotary wings. Catapults, which can be free standing, car-top, or even hand held, can be used to launch UAVs up to Group 4. Finally, UAVs can be air launched by another aircraft. Some SUAVs use this technique and are typically jettisoned or dropped from an aircraft inside a protective casing such as a sonobuoy, flare magazine, etc. and then deploy after they are dropped or jettisoned [8].

Recovery Methods

Recovery is not always an option or even a design choice in UAV missions. When a UAV is launched into a high risk area for a reconnaissance mission it may not be possible to retrieve the UAV if it has a short battery life or is not near a friendly base [8]. Thus, some UAVs are designed to be disposable and do not have a recovery method.

Some UAVs are equipped with landing gear and can be landed on a runway like a conventional aircraft. Another common way of landing and recovering an aircraft is parachute deployment. The UAV deploys a parachute and is retrieved from the ground, sea, or even plucked from the air [8].

Autonomy

Autonomy, in other words the capability to perform actions without the direct intervention of a human, is a key characteristic of UAVs. As seen in Table 1, UAVs vary in their level of autonomy; ranging from human operated, or remote control, in which a computer controls only low level functions, to fully autonomous, in which a computer is in control of the entire system for often long periods of time. Increasing autonomy can have many advantages in a UAV system depending on the mission. In a wide spread mission involving many UAVs, autonomy can lead to manpower savings since all modern aircraft require a team of people to operate, maintain, observe and interpret the data received, even for a remote controlled vehicle. In a more autonomous system, one pilot could oversee several aircraft and delegate a computer to control some of the tasks associated with flying each aircraft, such as stabilization or navigation [9].

Level	Name	Description
1	Human Operated	A human operator makes all decisions. The system has no autonomous control of its environment although it may have information-only responses to sensed data.
2	Human Delegated	The vehicle can perform many functions independently of human control when delegated to do so. This level encompasses automatic controls, engine controls, and other low-level automation that must be activated or deactivated by human input and must act in mutual exclusion of human operation.
3	Human Supervised	The system can perform a wide variety of activities when given top-level permissions or direction by a human. Both the human and the system can initiate behaviors based on sensed data, but the system can do so only if within the scope of its currently directed tasks.
4	Fully Autonomous	The system receives goals from humans and translates them into tasks to be performed without human interaction. A human could still enter the loop in an emergency or change the goals, although in practice there may be significant time delays before human intervention occurs.

Table 1: Four Levels of Autonomy [5]

2.3. WILD Goblin

The Wing and Internal Launcher Deployed (WILD) Goblin is a UAV in development in Group 106 at Lincoln Laboratory. The idea behind the WILD Goblin UAV is to have a small UAV (SUAV) that will be relatively inexpensive, disposable, and equipped with a sensor pod to perform completely autonomous reconnaissance, target identification, and target tracking.

Some sample mission areas that the WILD Goblin is designed for and will prove useful include:

- Border patrols
- Searching for camouflaged and concealed targets
- Identifying targets in heavy clutter
- Inspecting pipelines, roads, transmission lines, and bridges

Coyote Airframe

Shown in Figure 8, the aircraft design of the WILD Goblin is based off of the BAE Systems Coyote airframe. As shown in Figure 8, the body of the UAV is three feet long and fits into a standard A-size sonobuoy, also shown, which is 36 inches long and 4.875 inches in diameter[1]. In addition to the standard flight components that make up the WILD Goblin such as foldable wings and control surfaces, a motor and its associated electronics for propulsion, and other material and control systems required for flight, the front 5 inches of the UAV body as well as the internal space of the nose cone is left empty to allow a custom sensor payload to be installed.

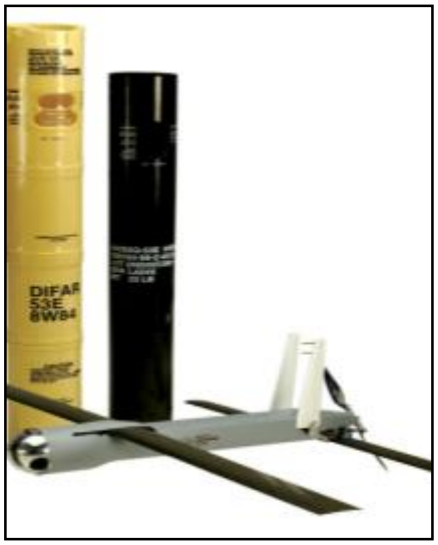


Figure 8: BAE Systems Coyote and its Sonobuoy Tube [9]

Mission Outline

The WILD Goblin is designed to be air-launched from an airplane at high altitudes (50,000 feet) and high speed. The UAV is intended to be dropped from the aircraft inside the sonobouy tube, and between 50,000 feet and 20,000 feet, deploy a parachute to stabilize the sonobuoy during fall. While falling, the WILD Goblin is designed to warm up inside the tube since the temperature at 50,000 feet is about -55 degrees Fahrenheit, which is too cold for the electronics onboard the UAV to work properly.

At about 20,000 feet, the WILD Goblin is ejected from the sonobuoy tube with the parachute remaining attached to the airframe, not the tube. Shortly after, the parachute detaches and the wings of the WILD Goblin expand and the UAV begins to glide with a 20:1 glide ratio, meaning that the UAV will

[1] The astute reader will notice the discrepancy in the diameter dimensions given in this section. Even though the sonobuoy tube is 4.875 inches in diameter, our project mentor at Lincoln Laboratory made the requirement for our nose cone design to be less than 5.375 inches in order to be able to perform various tests with a design, such as a ground launch, even if the designed sensor pod will not fit inside the sonobuoy tube.

glide 20 feet forward for every foot it falls. During this gliding phase, the UAV does not produce any noise or heat signature for enemies to detect.

Eventually the WILD Goblin's propulsion system will start and maintain the WILD Goblin at its mission altitude (altitude at which it typically performs) between 100 and 300 feet with a cruise speed of around 60 knots. The Coyote has the capability of travelling at a burst speed of 120 knots, but sustained flight at this speed greatly reduces the overall time of flight, as more power must be drawn from the batteries to keep the UAV at that speed.   While the battery is sized to maintain the WILD Goblin in normal flight (60kts) for about an hour, the battery would only last about 10 minutes if the UAV were to constantly travel at 120 knots.

Sensor Payload

Typical operation of the WILD Goblin first involves using acoustic sensors mounted on the belly of the WILD Goblin to listen for a certain sound or a potential target of interest (e.g. the rumbling of a pick-up truck's engine). Once a sound is heard, the Goblin uses the long-wave infrared camera or short-wave infrared camera to obtain a picture of the target (e.g. heat signatures on the target or headlights from the target). If the target is identified as a potential target of interest, the Goblin approaches and uses a laser rangefinder to determine how far away the target is and develops a point cloud image of the target. In Figure 9, a sample point cloud image is shown. A point cloud image is usually a 3 dimensional picture of an object made up of tiny dots, or points. The position of each point is based off of distance data received from a laser rangefinder or other distance sensor.

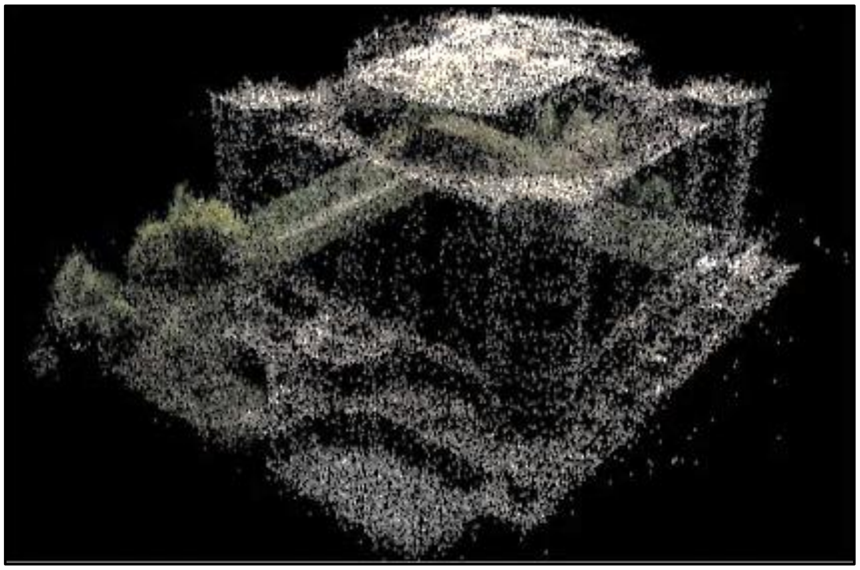


Figure 9: Example of a Sparse Point Cloud of a Building [10]

UAV Turret Scans

The word turret in this project refers to a mechanical housing for various sensors that are implemented in UAVs in order to move the UAVs' sensors in a desired way. Depending on the mission task of a UAV, a UAV’s sensor turret typically has various controls and is able to perform several movement patterns, sometimes referred to as scans.

Figure 10 shows an example of a raster scan. A raster scan is primarily performed by the turret so the sensors can image stationary objects when the UAV is orbiting those objects. The basic movement of a raster scan consists of moving to the right with a desired width and then down a certain amount (usually a small amount) and then moving to the left the same desired width. This process continues until the desired height is reached, then the turret moves back to the starting position.

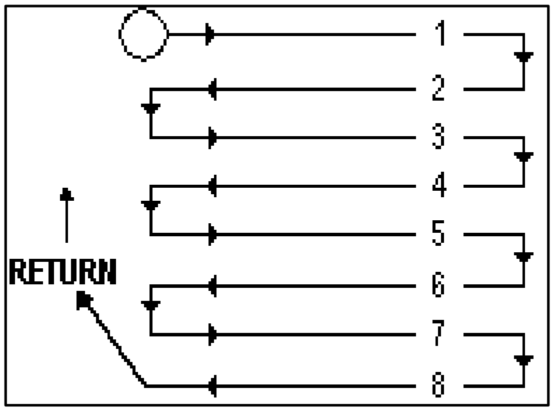


Figure 10: Raster Scan [11]

Figure 11 shows another type of scan that turrets can perform. Turrets perform spiral scans because spiral scans enable laser rangefinders to shoot many points in a short amount of time in a focused area on the desired target. Typically, spiral scans start from the middle of the spiral and gradually move further from the center of the spiral as more revolutions are completed, until a desired radius is reached.

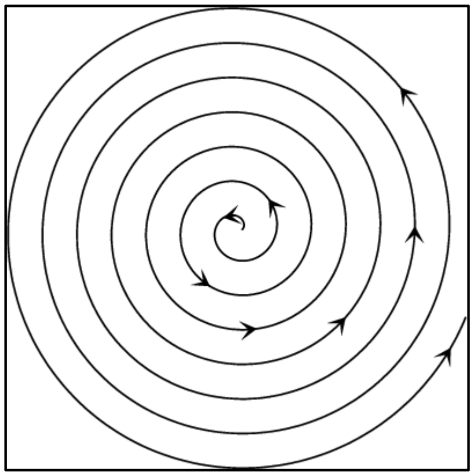


Figure 11: Spiral Scan [12]

Figure 12 shows an example of a Figure-8 clover scan (a clover with 2 leaves) with arrows representing the motion of how a turret can move to complete a clover scan. A clover scan is used to replicate a convoy eye-scanner (human behavior) because it allows the sensors to look around at the

UAV’s surroundings and then focus back at the middle to know where the UAV is flying. There are other types of clover scans that are common including a 4-leaf clover scan, which is useful because it allows the turret to move in such a way that the sensors have a better vertical view of the surroundings when compared to a Figure-8 scan.

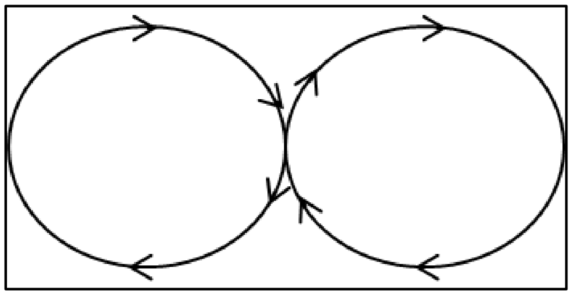


Figure 12: Figure-8 clover Scan

Previous Work

The development of the WILD Goblin has been divided into several development phases. Phase 0 involved acquiring the Coyote airframe as well as the sensors to be used. During this phase some of the intended capabilities were demonstrated to show project feasibility.

Phase 1 of the Goblin project, which is in progress, has been divided into several parts including:

- Integrating sensors into a hardware package
- Developing software to manage operations
- Developing tracking and detection algorithms.

In 2013, a WPI senior design project team was tasked with designing a sensor turret to be mounted in the nose cone of the Goblin UAV. This project team was able to demonstrate the feasibility of a small volume sensor turret for the WILD Goblin which could produce point to point motion covering a limited field of view (FOV). While the 2013 senior design project produced a prototype sensor turret, it did not perform to the desired level of expectation. Projects have also been undertaken by other groups of students to write tracking software for operation.

After the completion of Phase 1 of the WILD Goblin project, Phase 2 will involve final demonstration of the system. Phase 2’s system demonstration will include:

- Flight operations
- Communications via Iridium satellite
- Launch tests, both from the ground, then dropped from a plane at high altitudes

2.4. Sensor Capabilities

Three sensors have been chosen to facilitate the WILD Goblin's search, find, and identify missions: a laser rangefinder, a short-wave infrared (SWIR) camera, and a long-wave infrared (LWIR) camera, each performing a specific function.

Laser Rangefinder

A laser rangefinder is used to determine the distance to a particular point on an object or target, and as mentioned previously, the data can be used to develop a sparse point cloud of the object. Shown in Figure 13, the laser rangefinder that was chosen to be used in the WILD Goblin sensor pod was the Jenoptik DLEM-SR laser rangefinder. The laser for this sensor is a 1.55 micron diode laser. This laser rangefinder's maximum range is 5000 meters and its maximum measuring rate is 25 Hz (meaning it can send out a maximum of 25 pulses in 1 second) and the accuracy is less than 1 meter [13] . A faster repetition rate can increase the accuracy of the measurement because more laser points will be able to hit the target, and if the target is moving, then having more points on the target closer together will allow for a more accurate point cloud to be created.

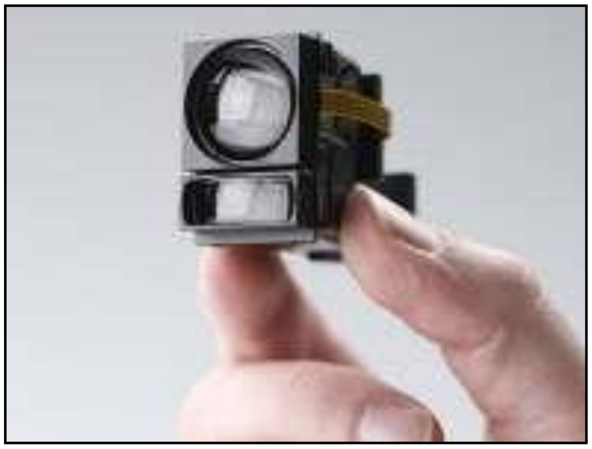


Figure 13: Jenoptik DLEM-SR Laser Rangefinder [13]

Laser rangefinders all follow similar operational steps:

1. The laser rangefinder firsts sends out a laser pulse.
2. That pulse then hits and bounces off of the object in front of the sensor and returns back to the laser rangefinder.
3. Flight time from the pulse being emitted to the pulse being received is used in measuring the distance to the object [14].

IR Cameras

Figure 14 shows the wavelengths of the electromagnetic spectrum with a breakdown of the infrared section of the spectrum: short-wave infrared (SWIR), medium-wave infrared (MWIR), and long-wave infrared (LWIR). Cameras that utilize wavelengths of the infrared spectrum are able to capture images which express different qualities than standard digital camera images that utilize the visible light spectrum. The qualities of SWIR cameras and the qualities of LWIR cameras are elaborated on in the following sections.

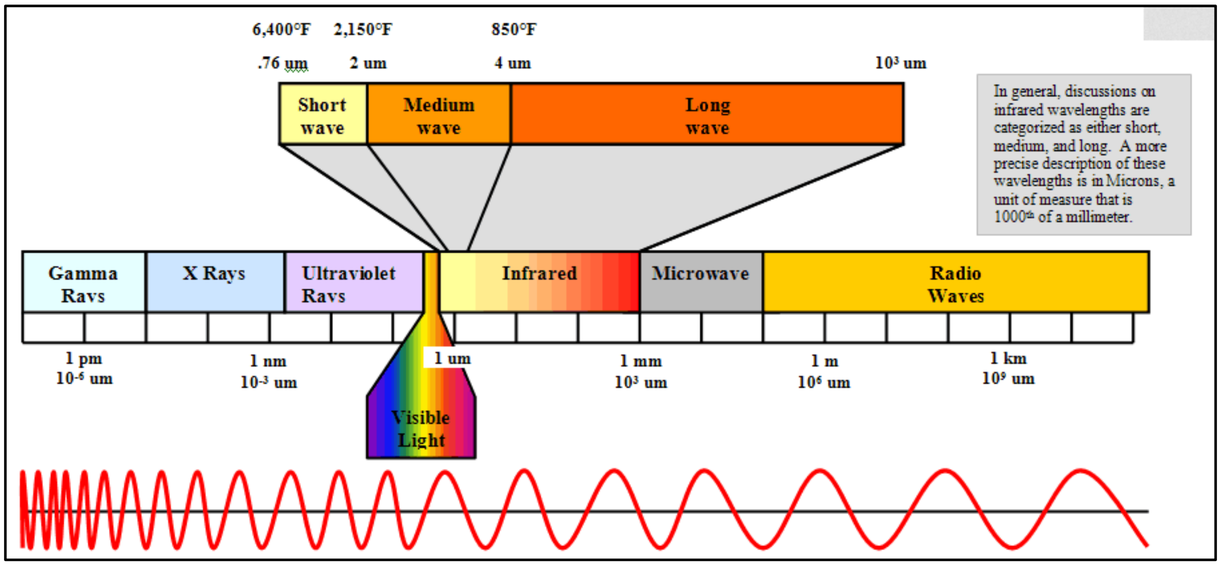

6,400°F
2,150°F
850°F
.76 um
2 um
4 um
10³ um
Short wave
Medium wave
Long wave
In general, discussions on infrared wavelengths are categorized as either short, medium, and long. A more precise description of these wavelengths is in Microns, a unit of measure that is 1000th of a millimeter.
Gamma Rays
X Rays
Ultraviolet Rays
Infrared
Microwave
Radio Waves
1 pm 10-6 um
1 nm 10-3 um
1 um
1 mm 10³ um
1 m 10⁶ um
1 km 10⁹ um
Visible Light

Figure 14: Electromagnetic Spectrum based on Wavelength [15]

Short-wave Infrared (SWIR) Camera

SWIR cameras are useful since standard digital cameras that utilize the visible wavelength of light can be obstructed by weather related phenomena and are unable to produce a clear image at night without external illumination. SWIR cameras can be used for producing clear images of objects that are in fog, mist, etc. or, as shown in Figure 15, used for night vision [16]. In Figure 15, on the left, a visible wavelength camera (e.g. a standard digital camera) was used, and only illuminated a small part of the captured image. However, on the right of Figure 15, a SWIR camera was used and a much more detailed image can be seen even though the image was taken at night, thus demonstrating the usefulness of SWIR cameras for night vision.

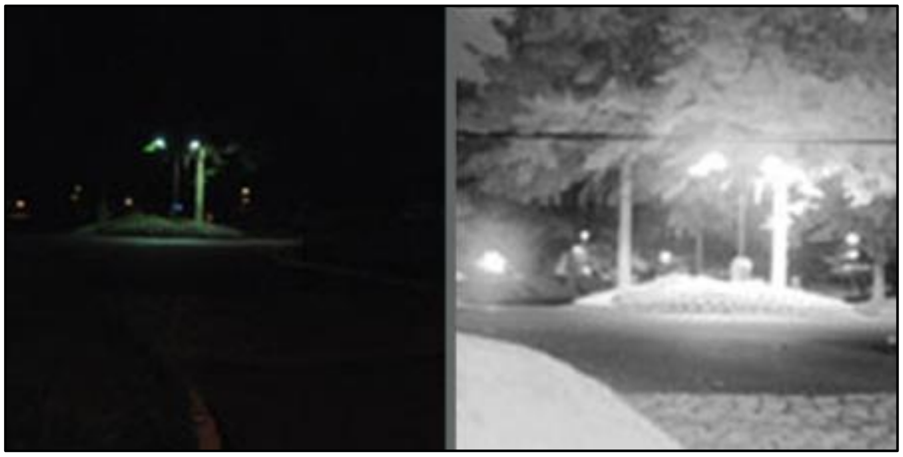


Figure 15: SWIR camera being used for night vision (visible wavelength camera used on the left, SWIR camera used on the right) [17]

Also, SWIR cameras detect and image targets through tree tops and canopies as long as the IR light is able to pass through the covering. Other applications of SWIR cameras include metal detection and edge detection of objects. Metal detection is useful because it allows the Goblin to distinguish between natural objects, such as trees, and man-made metal objects, such as cars. Edge detection is important because a color thermal camera may only produce a blob of the target so the SWIR camera can use its edge detection capabilities to produce a more detailed image of a target in its FOV.

Shown in Figure 16, the SWIR camera that will be used in the target tracking sensor pod of the WILD Goblin is the Sensors Unlimited Micro-SWIR. Its standard spectral response is 0.9-1.7 micron and its digital output frame rate is 30 frames per second (fps) [18].

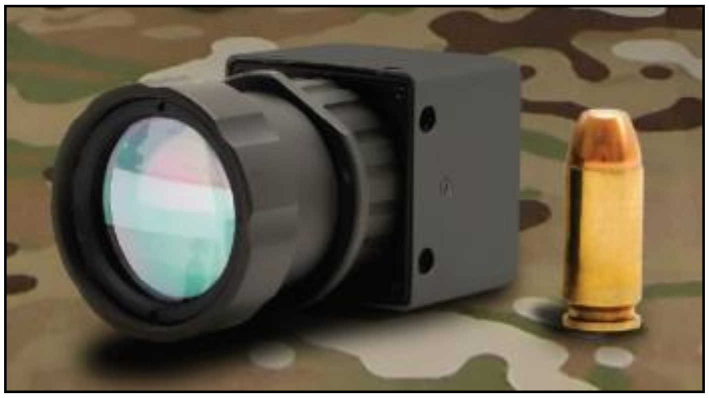


Figure 16: Sensors Unlimited Micro-SWIR [19]

Long-wave Infrared (LWIR) Camera

LWIR cameras are typically known as “thermal cameras” because they are able to sense the waves of energy (heat) that objects naturally emit and make them visible to humans [20]. An LWIR camera can be passive, meaning that the camera does not need to illuminate the scene to sense the objects’ radiated energy [21]. Long-wave infrared cameras convert the heat they detect into an electronic signal. The electronic signal is then able to be processed and, as shown in Figure 17, a thermal image is made based on that signal [22].

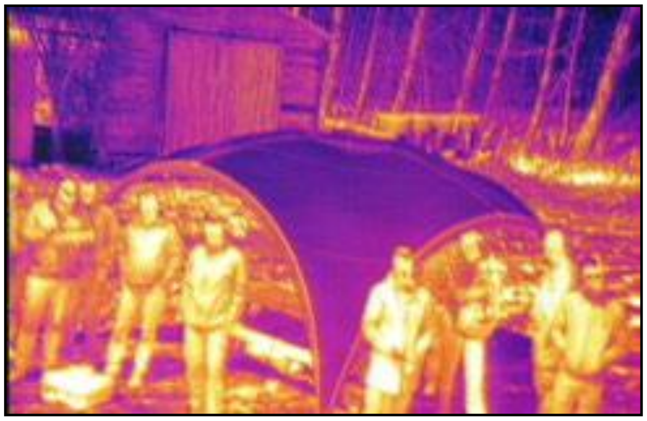


Figure 17: Thermal Image produced by a FLIR Quark LWIR camera [23]

Figure 18 shows the FLIR Quark, the long-wave infrared camera selected for the WILD Goblin. The spectral band of this camera is 7.5 –13.5 micron and its NEdT (noise equivalent differential temperature, which “represents approximately the minimum temperature difference which the camera can resolve” [24]) is less than 50 mK at f/1.0 (f/1.0 is regarding the focal length of the lens).

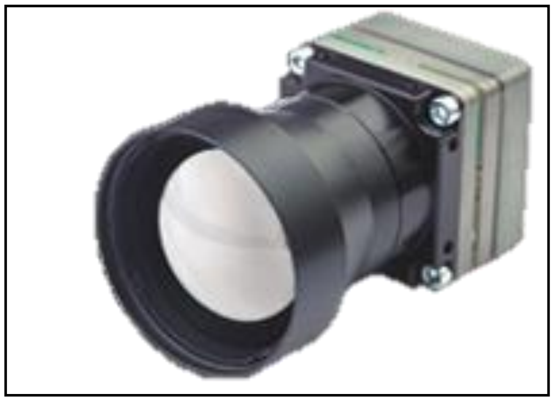


Figure 18: FLIR Quark LWIR camera [25]

Stabilization Sensors

The WILD Goblin UAV will require a stabilization system in order to decouple the field of view (FOV) of the sensors from the motion of the aircraft. The aircraft will perform various movements while flying and it is important that the sensors stay locked on the desired target regardless of the UAV's motion or orientation. The undesired motions of the UAV caused by turbulence or sudden wind gusts, for example, must be measured in order to know how much the sensor pod needs to be adjusted to keep the target in the sensors' FOV. Typically in UAVs a device called an inertial measurement unit (IMU) is used for stabilization and for maintaining the orientation of the UAV while flying.

There are usually three microelectromechanical systems (MEMS) that make up an IMU: a triaxial accelerometer, a triaxial gyroscope, and a magnetometer. For this project, the focus will be on implementing a triaxial accelerometer and a triaxial gyroscope in order to stabilize the turret.

The accelerometers are used to measure acceleration and are stable over long periods of time, but return inaccurate values if they are subjected to rapid motions. Acceleration is measured by g-force and MEMS accelerometers measure gravity's force on the three axes of the accelerometer in order to detect tilt [26]. Sensitivity of accelerometers is measured in mV/g, meaning that there are certain voltages that are output by the accelerometers due to the g-force on each of the three axes. The measurement range for accelerometers is also given in g-force (e.g. ±3g), where a low-g range is less than 20 g (i.e. motion a human is capable of generating), and a high-g range is more than 20 g and is used to detect motions that humans cannot generate (e.g. machine movements). Appendix A presents sample data sheets for a triaxial accelerometer's measurement range and sensitivity.

MEMS gyroscopes (also called gyros) measure angular velocity and are useful in detecting rapid motions over short time spans. Gyro sensitivity can be expressed in mdps/digit (milli degrees per second / digit). For example if the gyro is a 16 bit gyroscope and is programmed use a 500 degree per second measurement range (i.e. the range is from -500 dps to 500 dps) then the sensitivity would be (500dps*2)/ 2^{16} = sensitivity = 15.3 mdps/digit. However, datasheets for gyros tend to show a larger number than the actual calculated sensitivity in order to accommodate for manufacturing tolerances. Appendix A also includes a sample data sheet for a triaxial gyroscope's measurement range and sensitivity. Since gyros measure angular velocity, in order to acquire the angular positions of gyros (i.e. how much the gyros rotated), the gyros' velocities must be integrated over the time the gyros rotated. Due to these velocity measurements constantly being summed, the position values calculated from the angular velocity data tend to drift and become inaccurate over long periods of time.

Sensor Fusion Algorithms

In order to combine the readings from the accelerometers and gyroscopes on the IMU, a sensor fusion algorithm must be implemented in the software to ensure accurate orientation and direction readings.

Two possible sensor fusion algorithms that could be used are a complementary filter or a Kalman filter. A complementary filter includes a low-pass and high-pass filter to combine data from the accelerometers and gyroscopes. The low-pass filter is used to filter out vibrations of the accelerometers (since accelerometers are inaccurate during rapid motions) and a high-pass filter is used to filter out the slow drift of the gyros. By comparison, a Kalman filter uses properly weighted and measured inputs (e.g. gyro data and accelerometer data) and estimates a predicted output. Complementary filters are easy to use and less computationally expensive than Kalman filters. However, Kalman filters are usually more accurate than complementary filters since Kalman filters attempt to predict the next output from the inputs.

2.5. Summary

The goal of the WILD Goblin project is to design a small UAV capable of performing autonomous reconnaissance missions such as border patrol, searching for concealed targets, and investigating targets using point clouds. The success of the mission of the WILD Goblin UAV hinges on its implementation of a specific set of sensors used together for object finding, tracking, and identification. The small size of the UAV makes implementing the three sensors, the LWIR camera, SWIR camera, and laser rangefinder, an important component of Phase 1 of the WILD Goblin project. This report describes the design, development and integration of the WILD Goblin sensor pod.

Section 3. Project Statement

3.1. Introduction

The purpose of this section is to describe the specific goals, objectives and requirements for this senior project. These goals and objectives will guide the design and development of the sensor turret.

3.2. Project Statement

The purpose of our capstone project was to, first, design a sensor turret assembly which can house a set of sensors that will be used for the WILD Goblin's missions (e.g. target tracking), and second, to develop a software package that can be used with the turret assembly to control and stabilize the sensors mounted to the turret platform. While many Commercial Off-The-Shelf (COTS) UAV turrets exist, they are not optimized for the WILD Goblin's required size, weight, and power (SWaP) constraints.

3.3. Project Objectives

There were 3 main objectives for this project:

- Design and prototype a sensor turret
- Implement point to point movement, scan geometry, and turret stabilization on a test turret
- Integrate control software with the final turret prototype

3.4. Requirements

Table 2 shows the requirements for the system as well as the traceability for each requirement.

3.5. Summary

The purpose of this section was to describe the specific problem statement, objectives, and requirements for this senior project. The next section, Methodology, will describe the methods used to carry out the goals and objectives according to the requirements.

Table 2: System Requirements and Tracability

ID	Category	Requirement	Need Traceability
R01	Weight	The turret assembly without sensors shall weight less than 17 ounces.	Limited battery life, aerodynamics
R02	Size	The outer diameter of the turret assembly shall be less than 5.375" in diameter.	Size constraint of sonobouy used for launch.
R03	Size	The complete length of the turret assembly shall be no longer than 5".	Size constraint of sonobouy used for launch.
R04	Torque - Tilt Platform	The torque supplied to actuate the tilt platform shall be more than 10 oz-in, with a goal of at least 20 oz-in.	Based on approximate calculations of turning point, center of gravity and weight
R05	Torque - Roll Platform	The torque supplied to actuate the roll platform shall be more than 20 oz-in with a goal of at least 50 oz-in.	Based on approximate calculations of turning point, center of gravity and weight
R06	Speed - Tilt Platform	The tilt mechanism will be able to tilt at a rate of at least 15 degrees/second.	Mission parameters
R07	Speed - Roll Platform	The roll mechanism will be able to roll at a rate of at least 15 degrees/second.	Mission parameters
R08	Acceleration - Tilt Platform	The tilt mechanism will be able to accelerate at a rate of at least 15 degrees/second^2.	Mission parameters
R09	Acceleration - Roll Platform	The roll mechanism will be able to accelerate at a rate of at least 15 degrees/second^2.	Mission parameters
R10	Field of Regard - Tilt Platform	The tilt mechanism shall be able to tilt at minimum 45 degrees with a goal of 90 degrees.	Identify objects of interest directly below, identify aerial obstacles
R11	Field of Regard - Roll Platform	The roll mechanism shall be able to roll a minimum of 90 degrees with a goal of 180 degrees.	Maximize field of view of sensors
R12	Sensor Capacity	The turret assembly shall be able to hold the Jenoptik Laser Rangefinder, the Flir QUARK, and the Sensors Unlimited MicroSWIR.	Required sensor payload, established in Phase I
R13	Accuracy	The accuracy of the mechanism shall be less than 1 degree (the actual position of the turret should be within 1 degree of the specified position).	Mission parameters
R14	Repeatability	The repeatability of the mechanism shall be less than 1 degree.	Mission parameters

Section 4. Methodology

The purpose of the section is to describe the steps that were followed to develop the final sensor pod for the WILD Goblin UAV. The methods are divided into two primary pieces: the mechanical design and the turret control. These two aspects of the project were developed in parallel and merged during two integration tests at the midpoint and end of the term.

4.1. Mechanical Design Process

The mechanical design process consisted of design, analysis, prototyping, and testing phases. During the design phase, requirements were developed and used to design a SolidWorks model of the turret. The resulting model was analyzed for weaknesses, excessive stresses, etc. to ensure performance requirements were met. The design was prototyped using manufacturing tools available at Lincoln Laboratory as well as WPI. Once manufactured, the prototype was tested using verifications appropriate for each requirement.

SolidWorks was used for modeling the turret assembly due to the availability of SolidWorks on the Lincoln Laboratory system and WPI campus, as well as familiarity with the software. In addition, SolidWorks has many built in tools for design including interference checker and center of gravity calculator.

SolidWorks simulation tools were also used for more computationally intensive analysis including Finite Element Analysis. Analysis required complicated math which was done with the readily available software.

Prototyping took place in the Technology Office Innovation Laboratory (TOIL), which is a general learning and experimentation resource for Lincoln Laboratory staff. A description of the various machines and tools available can be seen in Appendix E: TOIL Capabilities.

Testing of the assembly took place as a part of the integration testing at the midpoint of the term and in the final week. A Gantt chart showing the scheduling of these milestone tests is shown in Appendix B: Gantt Chart. To verify that the assembly met all of the requirements, the assembly was tested using the methods outlined in Appendix D: Verification Plan.

4.2. Turret Control Algorithm Development

Modular Programming

In order to develop the turret control algorithms that were used to move the sensor pod in the final turret assembly, a modular programming method was implemented. Modular programming allowed each aspect or piece of the final code to be written independently so that if progress was halted in one module, the development of another module was not hindered.

Programming Environment

The test turret came with an Arduino Duemilanove board so the Arduino integrated development environment (IDE) was used to program the turret. Programs in the Arduino IDE were written in Clanguage, are easily transferable to other Arduino compatible boards, and have useful built in libraries, such as a math library and servo library.

Serial Commands

A serial commands module was developed in order to accomplish various desired tasks with the turret. These serial commands include:

1. Point to point turret movement
2. Status command
3. Heartbeat Trigger
4. Home command

Scans

A scan module was then written which controlled both servos, tilt and roll, in specific preprogrammed patterns. The scan program module incorporated the previous module, serial commands. These scans include:

1. Raster Scan
2. Spiral Scan
3. Clover Scan

The various applications for each scan are described in Section 2: Background of this report and the performance of each scan is evaluated in Section 6: Software Results of this report.

Inertial Measurement Unit (IMU)

For the IMU program module, a specific sensor fusion algorithm known as a complementary filter was developed in order to combine the accelerometer readings (analog) and gyroscope readings (I^2C) and to accommodate for the errors expected with accelerometers and gyroscopes. These errors were discussed in greater detail in Section 2: Background of this report. The ability of this IMU program to return accurate angle values is discussed in Section 6: Software Results of this report.

Target Tracking

The 2013 senior project's target tracking code was modified to work with our turret design. In MATLAB, several actions were programmed that sent various commands to the turret's microcontroller, depending on the desired action. Also the 2013 senior project code was changed to incorporate the short-wave infrared camera along with the long-wave infrared camera.

4.3. Integration Tests

Twice throughout the term, integration tests occurred during which the software and hardware components of the turret were tested together. The specific tests that were performed during both integrations of the hardware and software are listed in Appendix D: Verification Plan and the results of the tests are discussed in the Section 5: Design of the Sensor Turret.

4.4. Summary

The purpose of this section was to describe the design and testing processes for the hardware and software of the WILD Goblin UAV sensor pod. The design of the sensor turret solution consisted of two parts: the mechanical and the turret control. These two parts were developed simultaneously and integrated during two milestone tests. Design choices and the results of the tests are discussed in Section 5: Design of the Sensor Turret.

Section 5. Design of the Sensor Turret

5.1. Introduction

The purpose of this section is to describe the results of our methodologies in the design of the sensor turret. The first section, Mechanical Results, describes the development of the mechanical design including the improvements made to the turret design during each iteration and the shortcomings of each iteration. The Integration Tests section describes the results of the integration tests.

5.2. Mechanical Results: Iterations

This section describes each of the iterations of the sensor turret design, the improvements made during each iteration, and the problems and challenges encountered with each iteration. The sensor turret assembly consists of 6 major components: the sensor housing, the roll ring, the roll mechanism, the tilt mechanism, the turret housing, and the nose cone. Table 3 describes the iterations of the sensor turret design by components, as well as the issues encountered in each iteration.

Table 3: Iterations of the Sensor Turret

Iteration	Description of Item in Iteration						Issues Found
	Sensor Housing	Roll Ring	Roll Mechanism	Tilt Mechanism	Turret Housing	Nosecone	
1	Three pockets (one for each of the three sensors).	0.2 inch thick ring. Pocket for micro servo.	Internal gear (70 teeth) driven by pinion gear (10 teeth) attached to a 180 degree position servo.	Complex Motion. Rack gear driven by a pinion on a micro servo linearly actuates the sensor housing vertically. As the sensor housing raises or lowers it rotates about a roll bar.	Retain 2013 senior project turret housing. Cylindrical turret housing holds all components produced for the project. Fits into the cylindrical body of the Goblin. Holds roll ring. Keyway in front of housing is used to insert roll ring.	Cut in half to expose the sensors. Anchored to the Goblin Body, does not move with roll or tilt mechanism.	● Sensor Housing did not fit the sensors. ● Roll Mechanism requires a 1:1 ratio to reach the desired range of motion (180 degrees) ● Turret Housing: Keyway causes hold on roll ring to be loose. ● Roll Ring: Servo collides with pinion gear driving the roll mechanism
2	No Change	Ring was mirrored so that the servo no longer collided with the gears of the roll mechanism.	Changed gear ratio of gear train to be as close to 1:1 as certain conditions would allow.	No Change	Used two interlocking rings which formed a grove when locked together to hold the roll ring. The rings straddle a ridge in the existing Goblin body, holding them in place.	No Change	● Sensors did not fit into the sensor housing. ● Servo did not fit into the servo housing, likely due to an inaccurate servo CAD model.
3	Redesigned to fit sensors. Thinned or removed walls between sensors to decrease size of the sensor housing. Added screw holes for all sensors.	Removed micro servo mount, leaving two vertical bars to attach brackets to hold the sensor housing.	Reduced the number of teeth of the gear driving the internal gear, decreasing the range of the roll mechanism. Increased the gear ratio of the first two gears by changing their pitch in order to compensate.	Since larger pocket sizes for the sensors made the sensor housing longer, the roll rods would no longer fit on either side of the roll ring and so the tilt mechanism was changed.	Separated the tube side locking ring into two pieces (roll servo housing and tube side locking ring).	No Change	●Tilt mechanism did not have full 90 degrees of motion. ● Roll ring collided with the gear driving the internal gear, needed to cut a piece out of the vertical bar of the roll ring.
4	No Change	Added the brackets directly to the vertical bars of the roll ring rather than using machine screws to attach the brackets to the vertical bar.	No Change	Resynthesized four bar linkage to increase range of mechanism.	No Change	No Change	● Too much modularity on the design made the assembly very difficult to put together.
5	Redesigned sensor housing to allow more space for the SWIR lens. Moved the Laser Rangefinder to the top of the housing.	Moved the brackets to attach to the outside of the sensor housing.	No Change	Redesigned tilt mechanism to use gears rather than a four bar.	Combined the locking rings, servo housing, and Goblin body into one turret housing to reduce the number of pieces in the assembly.	Changed nose cone to attach to roll ring rather than Goblin body, allowing the nose cone to rotate with the roll mechanism. Changed the cut of the nose cone to cover more of the opening.	● Gears were all 3D printed and did not mesh well. ● Gear attached to tilt servo collided with nosecone in fully retracted position.
6	Altered the sensor housing to make it machinable by increasing wall thickness and adding fillets to inside corners.	Reduced the size of the brackets.	Changed from a position servo to a continuous turn servo to increase the range of motion of the mechanism, which allowed for a gear ratio other than 1:1. Performed parts selection to find aluminum gears for the roll mechanism.	Changed the gear mounted on the servo so that it did not collide with the nose cone. Change the ratio of the gears to increase the torque of the mechanism.	Realigned the holes for the shafts holding the gears for the new gearing.	No Change	● Roll servo does not line up well with the shaft causing grinding and vibration. ● gears for tilt mechanism are plastic and do not attach well to the micro servo.
7	Changed the micro servo mount to accommodate a new micro servo.	No Change	Realigned servo and shaft.	Used new micro servo with standard spline size as well as a gear which attaches directly to the standard spline.	Extended the length of the turret housing to add another sensor. Added ledges with holes to allow shelves for electronics to be added .	No Change.	

Iteration 1

In the first iteration of the sensor turret design, we attempted to keep some parts of the 2013 senior project design. Figure 19 shows the first design iteration with colored parts representing the pieces which were carried over from the last design (red) and which were new to the design (blue).

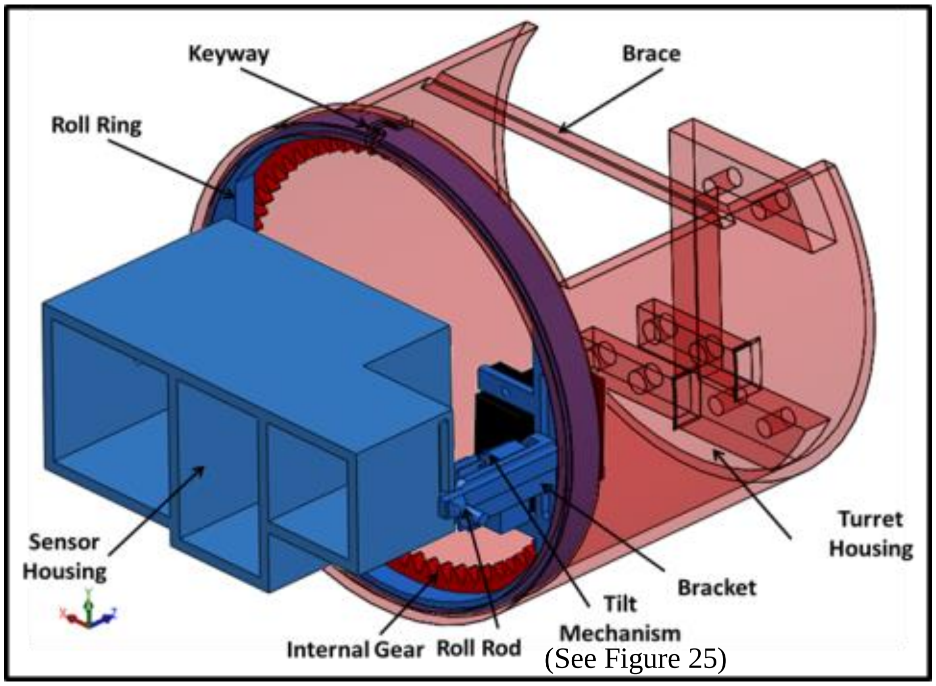


Figure 19: SolidWorks Model of Iteration 1 (Front)

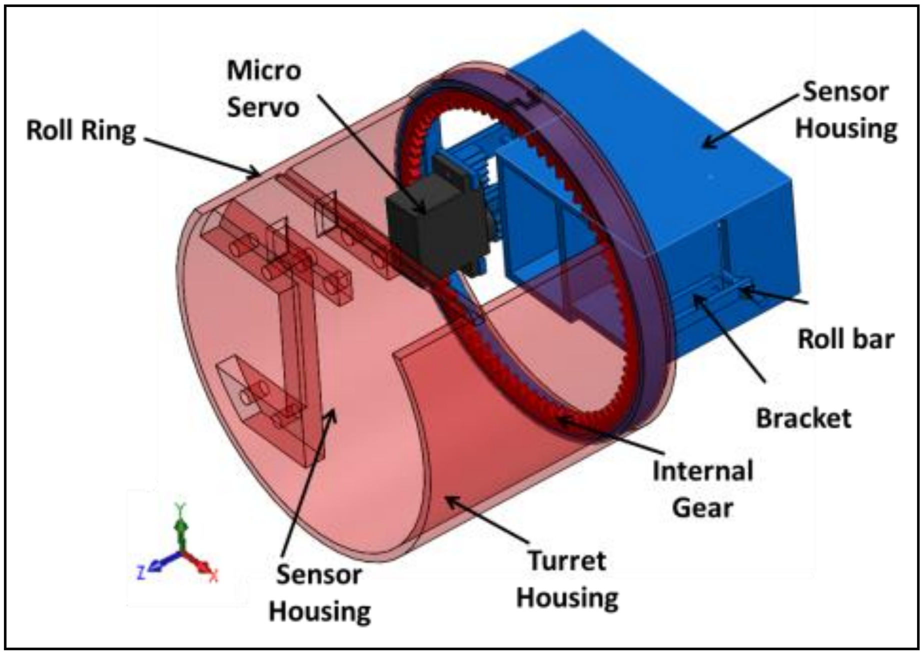


Figure 20: SolidWorks Model of Iteration 1 (Back)

One part of the previous design which was retained was the roll mechanism, consisting of a large internal gear driven by a smaller spur gear. Use of the internal gear requires less space within the tube, which has two benefits: it allows more or larger electronic components to be added to the payload and makes cabling for the cameras simpler as the moving parts which could get in the way are kept to the edge of the tube. Another part of the design which was retained was the cylindrical turret housing (shown in Figure 19 and Figure 20 as transparent red). This housing contains the roll and tilt mechanisms, servos, and the camera housing.

Another commonality between the 2013 project and the first iteration was the shape and function of the nose cone, shown in Figure 21. The Coyote is designed to have a domed nose cone which must have a hole cut into it for the cameras to see out. The lower half of the nose cone is removed to expose the sensors, while the top half of the mechanism is covered. The nose cone is attached to the outer body of the Goblin and does not move with either mechanism.

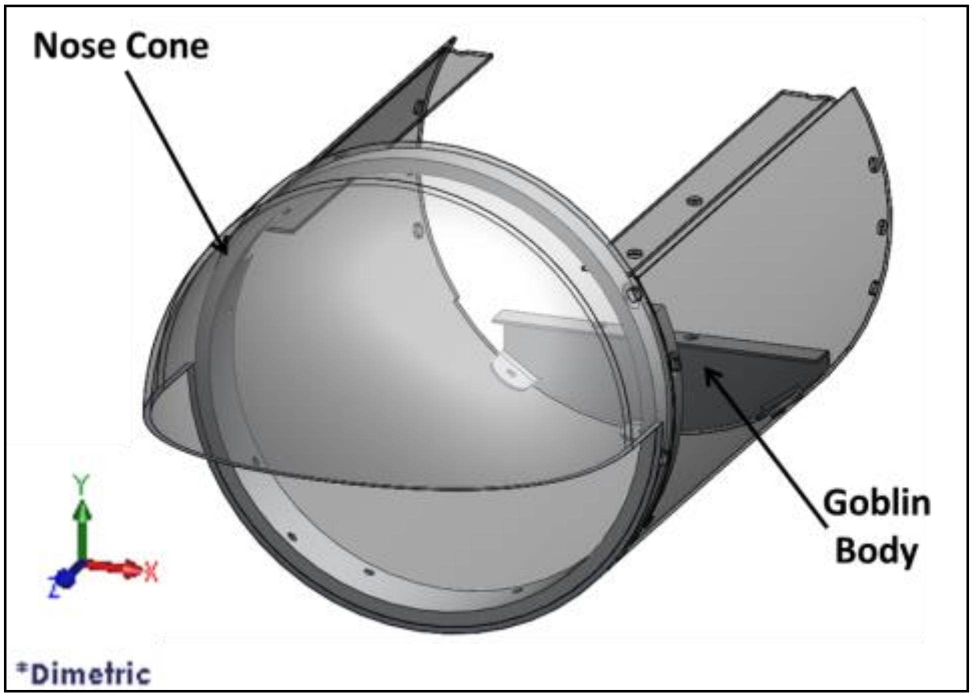


Figure 21: Body and Nose Cone of Goblin with Half Cut Nose Cone

One part of the 2013 assembly which could not be reused in this project was the sensor housing, as a new requirement for the sensor turret was to accommodate three cameras rather than two. In 2013, the turret was designed to hold only the Jenoptik Laser Rangefinder and the FLIR Quark. The sensor package was expanded to include the Sensors Unlimited Micro SWIR camera for this project. Figure 22 shows SolidWorks models of each camera for size comparison.

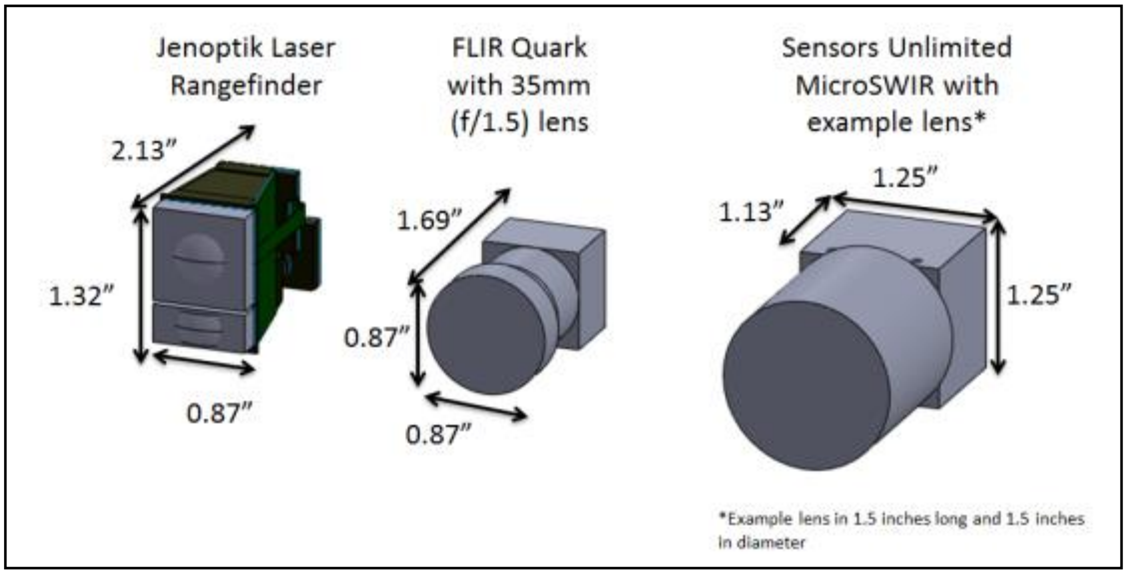

Jenoptik Laser Rangefinder
FLIR Quark with 35mm (f/1.5) lens
Sensors Unlimited MicroSWIR with example lens*
2.13″
1.32″
0.87″
1.69″
0.87″
0.87″
1.13″
1.25″
1.25″
*Example lens in 1.5 inches long and 1.5 inches in diameter

Figure 22: Size Comparison of Three Sensors

Figure 23 shows the sensor housing designed for the first iteration. The housing has three pockets, one for each sensor, which are open in the back to accommodate the cables for the sensors.

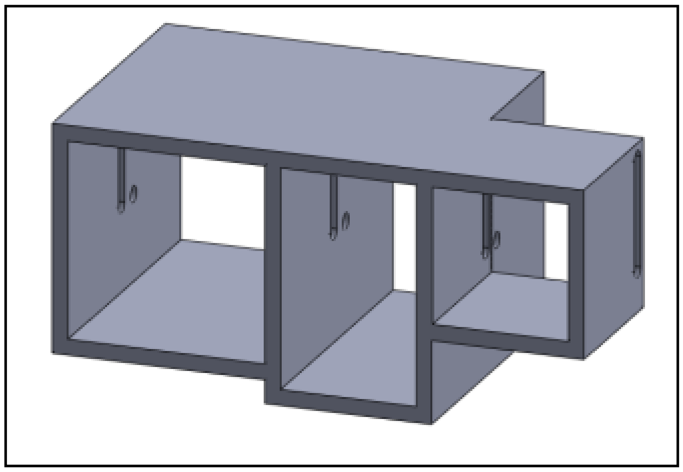


Figure 23: Sensor Housing for Iteration 1

Also in the first iteration, a new tilt mechanism was designed. The previous design used a four-bar mechanism, shown in Figure 24. According to the 2013 MQP report, the four-bar was able to move the cameras through a range of 45 degrees. While this range is sufficient, we wanted to expand the range of the tilt motion without using a four-bar, as four-bar mechanisms tend to be space inefficient.

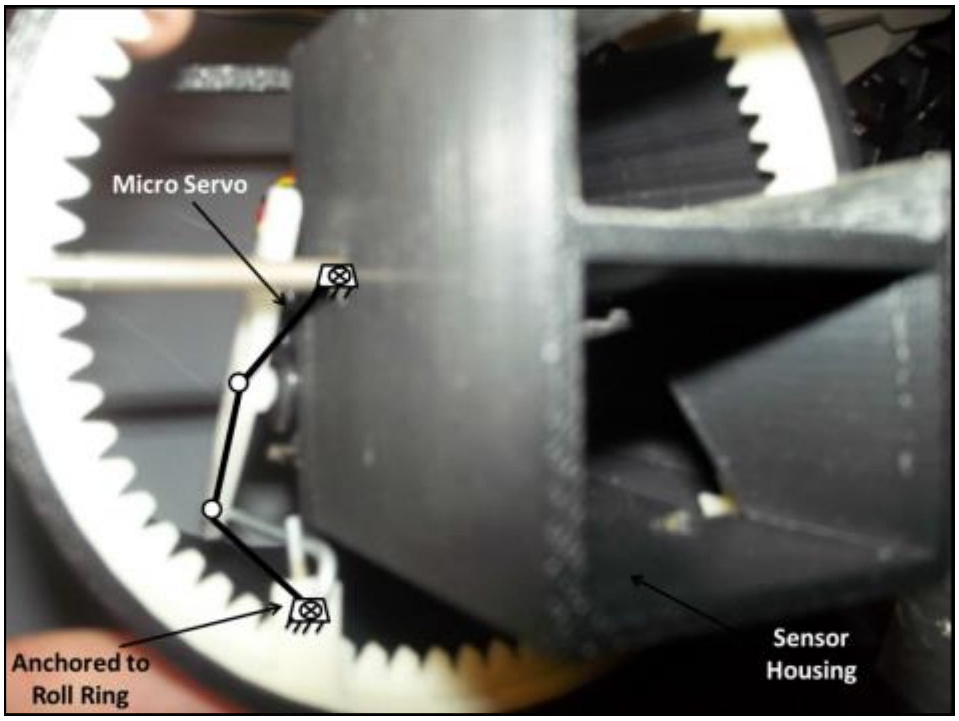


Figure 24: 2013 Senior Project Four-bar Linkage

In order to allow the cameras to see directly below the aircraft without being obstructed by the body of the aircraft, the point of rotation needed to be moved outside of the body of the aircraft and into the nose cone. In addition, the cameras needed to be able to see out of the front of the aircraft without being obscured by the nose cone. As Figure 21 shows, the nose cone was cut to cover the upper half of the tube only, leaving the lower half open for the cameras to look out. The first mechanism considered for the tilt motion was a mechanism which used a combination of linear and rotational motion, a complex motion, in order to accommodate both of these positions.

Figure 25 shows the complex motion mechanism used in the first iteration of the turret design. The sensor housing was held by a bracket on either side, and the bracket attached to a slot in the roll ring. A rack gear was attached to the bracket and is driven by a pinion mounted on the spline of a micro servo. In order to rotate the sensor housing, a rod with a 90 degree bend was attached normally to the roll ring, and the end of the rod was inserted into a slot on the sensor housing. When the pinion was rotated by the servo, the rack and bracket were raised or lowered and the sensor housing was rotated around the rod while also rising in the tube

This complex motion positioned the sensors in the lower half of the tube when in the horizontal position, and held the sensor housing outside of the body of the aircraft so the sensors could see directly below the aircraft in the vertical position. Figure 26 shows the horizontal position, an intermediate position, and the vertical position of the sensor housing.

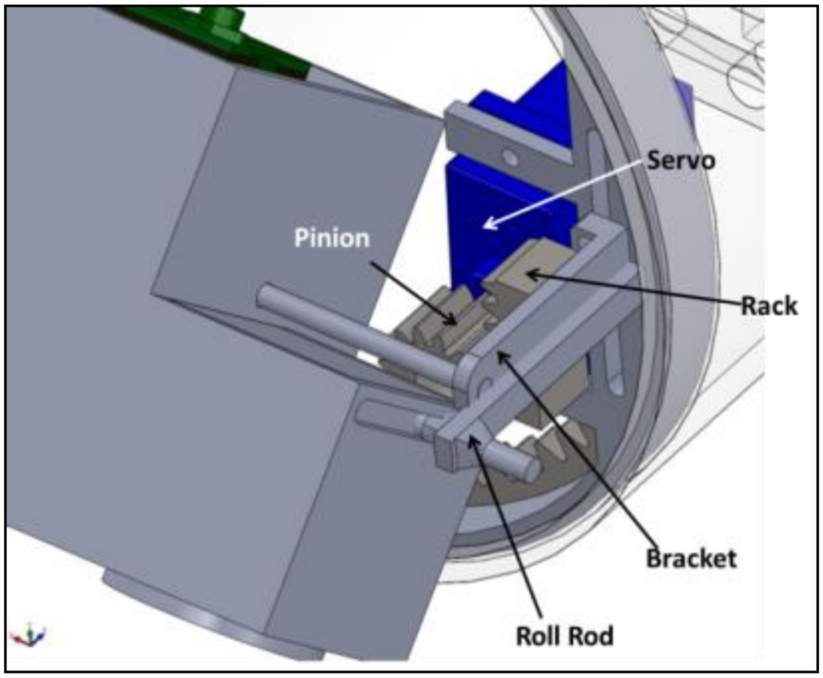


Figure 25: Complex Motion Tilt Mechanism

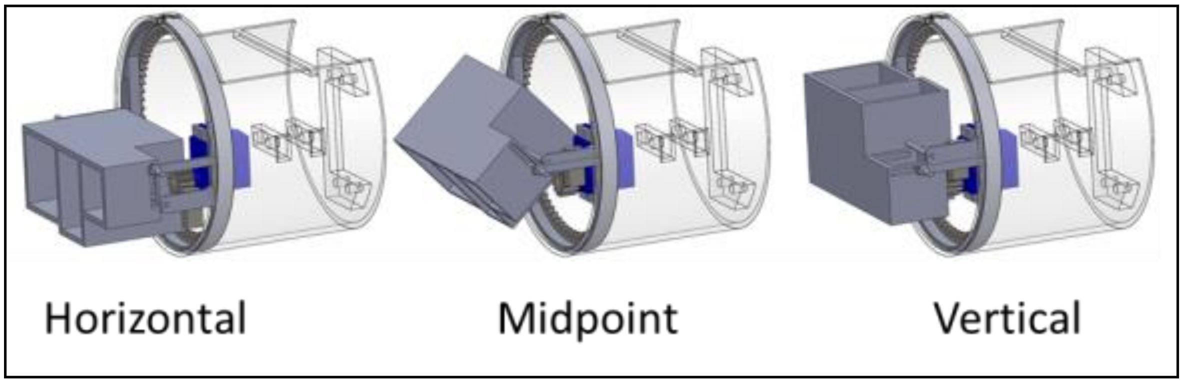


Figure 26: Three Positions of the Complex Tilt Mechanism

Upon analysis and assessment of iteration 1, we identified several opportunities for improving the design of the sensor turret. One of the parts which was problematic was the cylindrical turret housing, shown in Figure 27, which had been reused from the 2013 project. This cylindrical housing has a groove in the front which keeps the roll ring from falling out while rolling. A keyway is cut into the top of the tube to allow the roll ring to be inserted or removed from its groove. Adding the keyway, however, loosened the hold the groove needed to have on the roll ring in order to ensure a smooth rolling motion. In addition, the turret housing, which was supposed to be inserted into the body of the payload volume had no mechanism to anchor it in place.

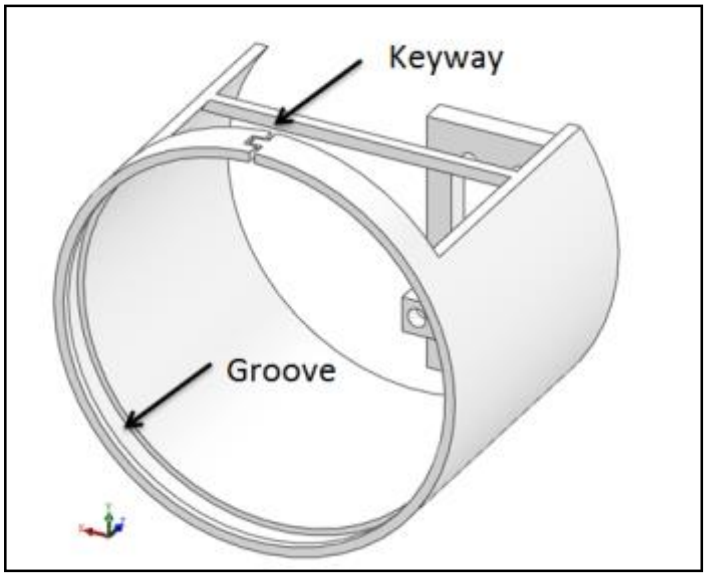


Figure 27: Keyway and Groove of the Turret Body

Another area of concern was the gearing of the roll mechanism. In the 2013 design, the 70 tooth internal gear was driven by a 10 tooth gear, which was driven by a second stage of 10 tooth gears to transmit power from the servo to the internal gear. Overall, the mechanism had a gear ratio of 7:1, so if driven by a position servo the mechanism would have a range of less than 180 degrees.

Iteration 2

In the second iteration, all parts which were retained from the 2013 project were replaced. Figure 28 and Figure 29 show two views of the SolidWorks model for this iteration. The tilt mechanism was also moved to the opposite side of the tube opening so as not to interfere with the roll mechanism. The length of the rack driven by the pinion was lengthened to extend the range of the tilt motion. The roll servo was moved to the lie along the bottom of the tube, connected more securely to the turret housing than in the previous iteration, making the servo more stable and less susceptible to vibrations.

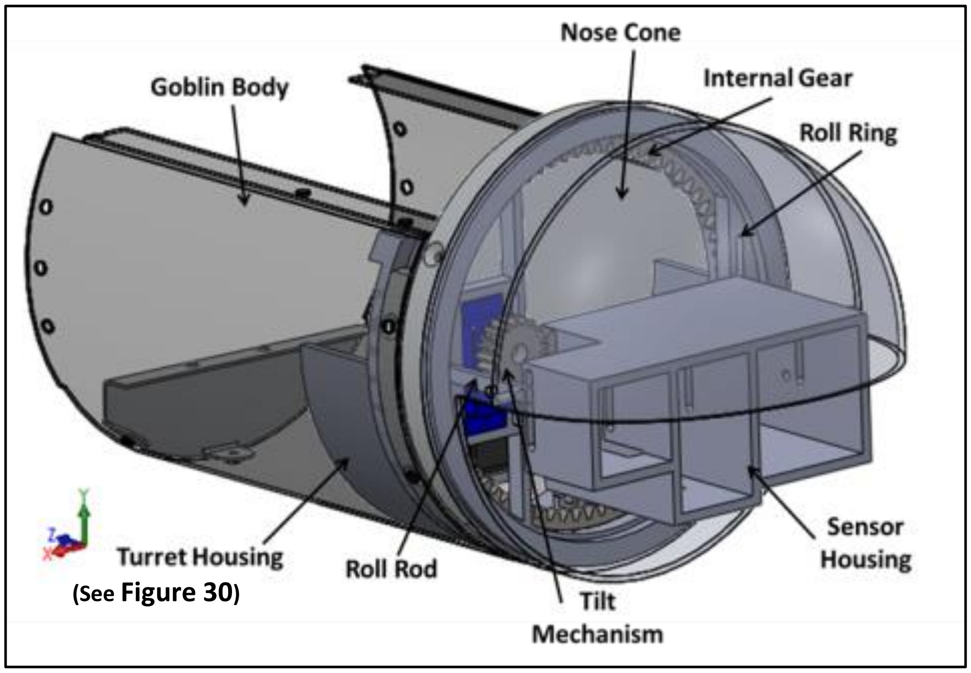


Figure 28: SolidWorks Model of Iteration 2 (Front)

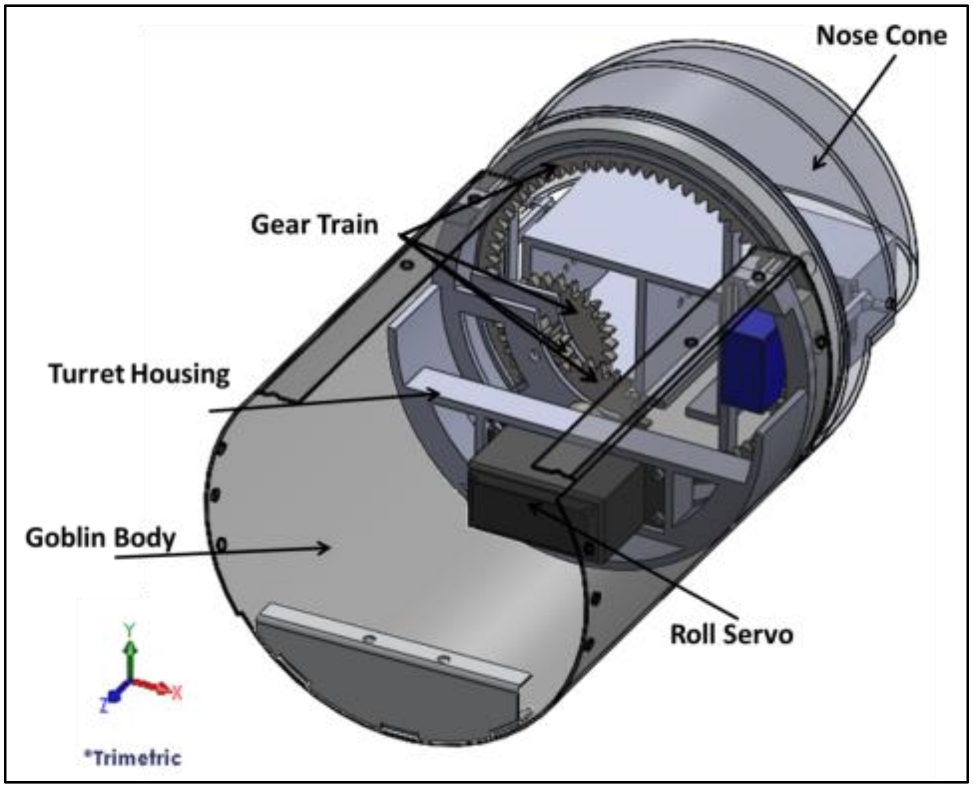


Figure 29: SolidWorks Model of Iteration 2 (Back)

In the second iteration, we needed to design a part or set of parts to hold the roll ring tightly without a keyway and anchor the turret housing in the body of the Goblin. The solution we used involved two interlocking rings which straddled the ridge between the tube and nose cone. Figure 30 shows an exploded view of the interlocking rings and the ridge.

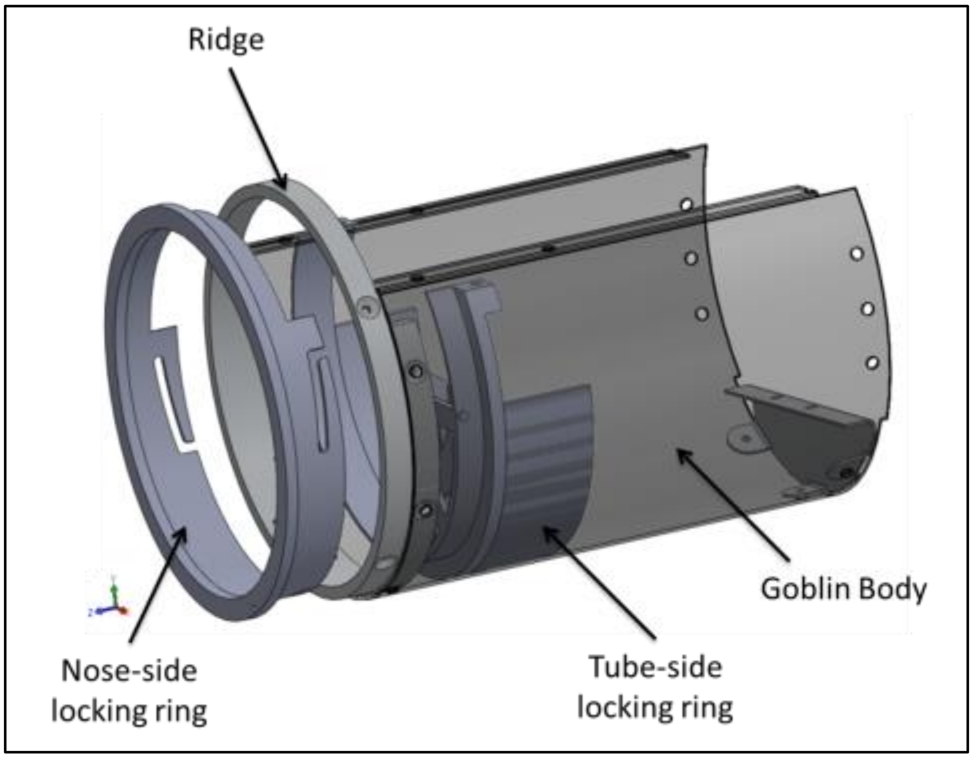


Figure 30: Exploded View of Locking Rings

Figure 31 shows how the rings locked together. The tube-side locking ring has three bumps along its rim which fit into three curved slots along the rim of the nose-side locking ring.

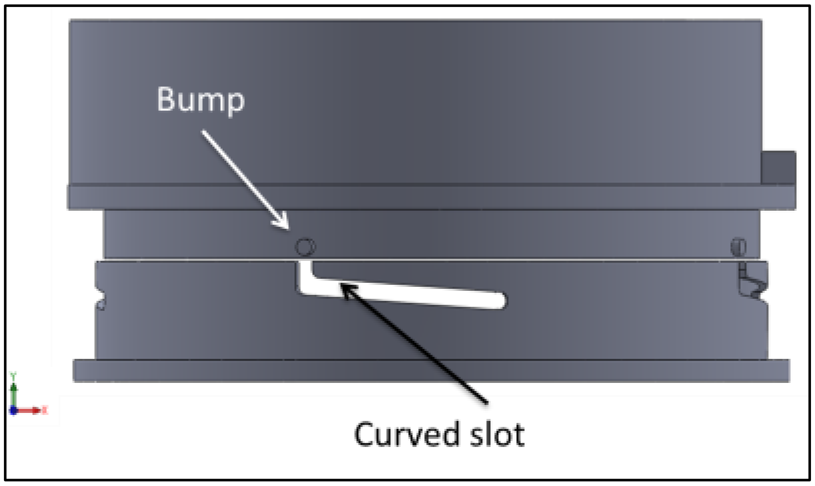


Figure 31: Locking Rings

When locked together, the locking rings formed a groove for the roll ring to fit in, which is shown in purple in Figure 32. In addition to holding the roll ring tightly, the interlocking rings secured the entire assembly in the tube by straddling the fixed ridge in the payload volume.

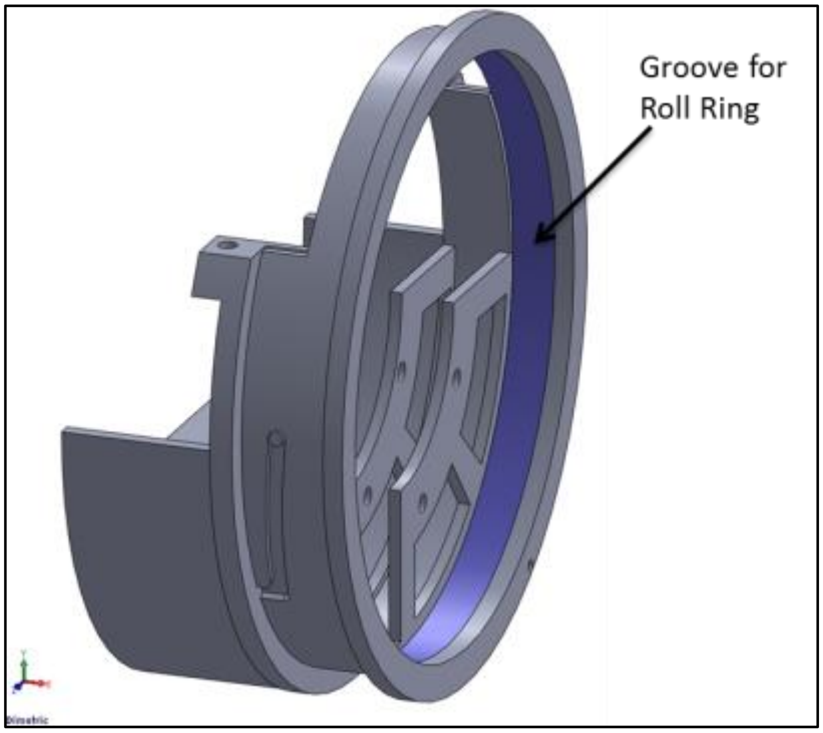


Figure 32: Groove for Roll Ring (shown in purple)

Another goal of this iteration was to determine the gear configuration needed to drive the roll mechanism. The roll mechanism consisted of a large internal gear and three spur gears as shown in Figure 33. Since the position servo driving the roll mechanism has a range of 180 degrees and the desired range of the mechanism is 180 degrees, the gear ratio between the servo and the spur gear should be 1 or lower. A gear ratio lower than 1:1, however, would reduce the resolution of the mechanism as for every degree of input motion there would be more than one degree of output motion.

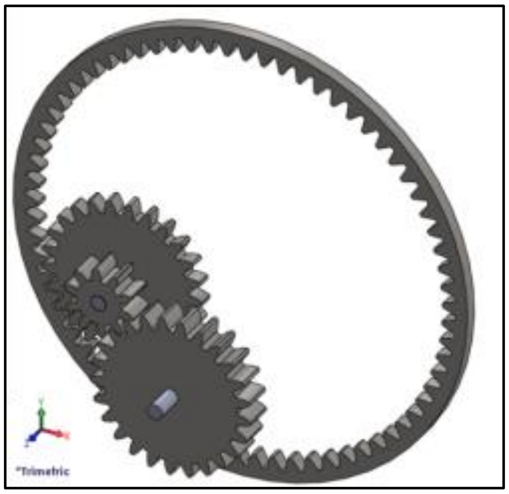


Figure 33: Gearing for Iteration 2

Figure 34 shows a diagram of the gear train for the roll mechanism. Gear 1 (N_1) is driven by the roll servo and N_i is the internal gear. Equation can be used to find the gear ratio of the gear train using the number of teeth on each gear.

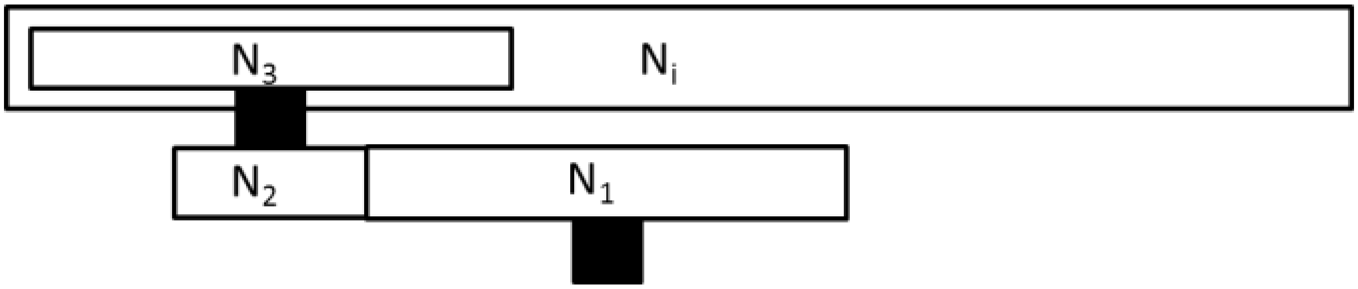


Figure 34: Gear Labels

$$Gear\ ratio = \frac{N_1}{N_2} \times \frac{N_3}{N_i} \tag{1}$$

Since the WILD Goblin is limited in space, the gears for the gear train were chosen to reduce gear size. The minimum gear tooth size considered was 10 teeth, as that is the minimum number of teeth generally available for 20 pitch gears which were used during this iteration, and the maximum number of teeth on a spur gear was 25, which with a pitch of 20 has a 1.25" diameter.

Equation (1) solves Equation (2) with these parameters and optimizes for the largest gear ratio.

$$Gear\ Ratio = \frac{N_1}{10} \times \frac{N_3}{70} = \frac{N_1 \times N_3}{700} = \frac{25 \times 25}{700} = 0.89 \tag{2}$$

Equation (3) can be used to calculate the output range from the input range and gear ratio.

$$Gear\ Ratio \times Input\ Range = Output\ Range \tag{3}$$

Equation (4) calculates the output range for the gear train designed above.

$$0.89 \times 180\ degreees = 160\ degrees \tag{4}$$

This range of 160 degrees meets the minimum requirement for output range of 90 degrees, though it does not reach the ideal goal of 180 degrees. Figure 35 shows a gearing diagram, annotated with the number of teeth on each gear.

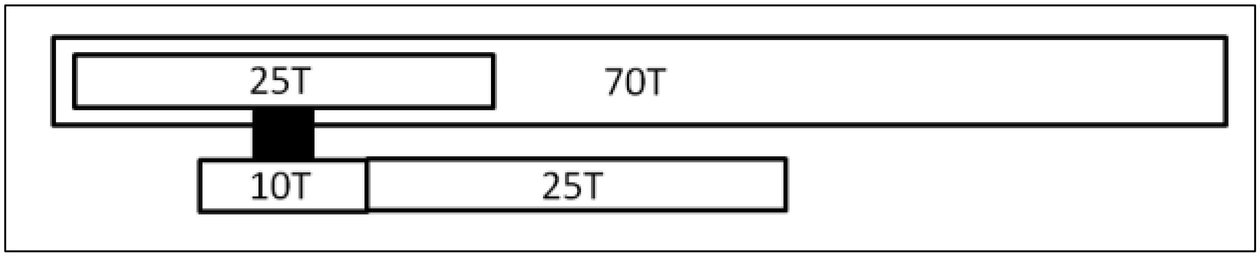


Figure 35: Diagram of Gearing Annotated with Tooth Counts for Each Gear

This iteration was the first to be prototyped during this project using the 3D print capabilities in TOIL. As a result, many of the results of this iteration relate to the fit of various components and the tolerances of the machines used. The pockets in the sensor holder, for instance, were all too small to fit the sensors. The servo also did not fit into its housing because the SolidWorks model was made using a "standard servo" CAD model used in the 2013 senior project SolidWorks assemblies, which was not the same size as any of the servos we had available. In future iterations, servo size must be decided on in advance, or a more easily swappable servo housing must be designed. The servo housing in this iteration was in the same SolidWorks part as the tube-side locking ring. The entire piece took several hours to print on the Stratasys and required a large amount of support material as it contained several overhangs, making reproduction of the part many times for different servos infeasible.

Iteration 3

In the third iteration, shown in Figure 36, several parts had to be changed in order to resize the pockets holding the sensors and servos. The effects of increasing pocket size, however, required a change to the tilt mechanism.

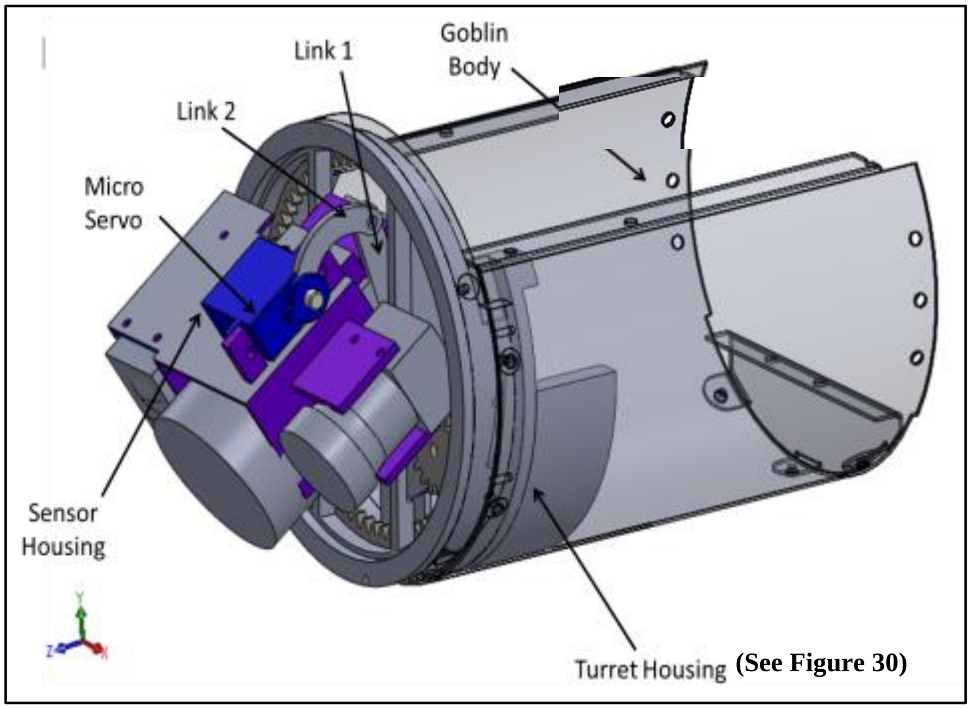


Figure 36: SolidWorks Model of Iteration 3 (Front)

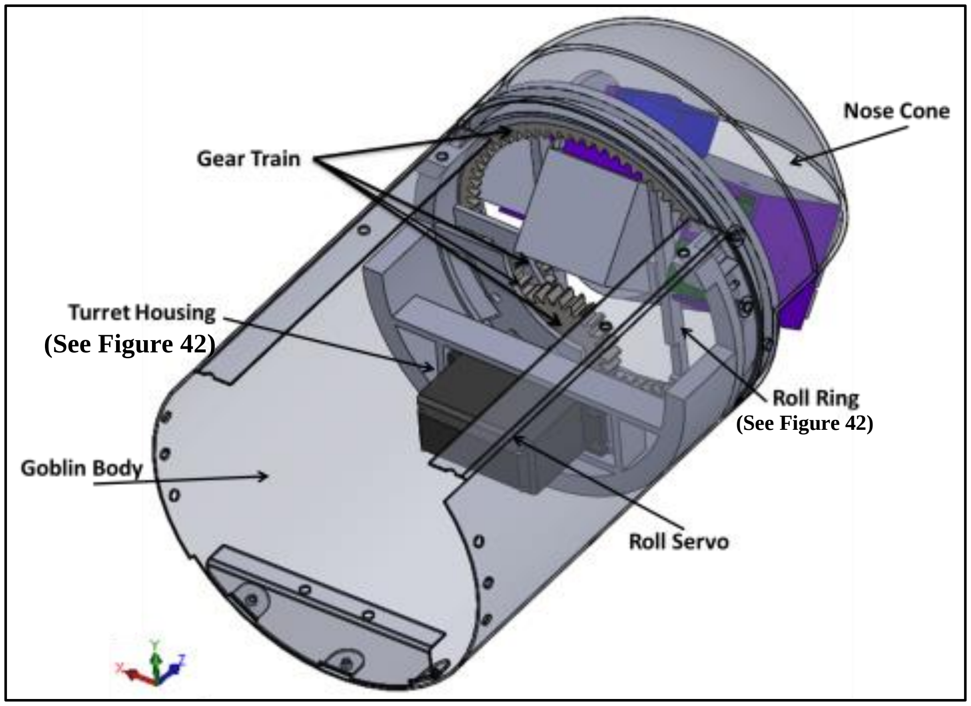


Figure 37: SolidWorks Model of Iteration 3 (Back)

The sensor housing was redesigned to increase the pocket size for each sensor. The sensors were also reordered as shown in Figure 38 such that the SWIR camera was the middle sensor in the housing, as the length of the SWIR camera with lens and cable required that the camera be located at the largest point of the roll ring.

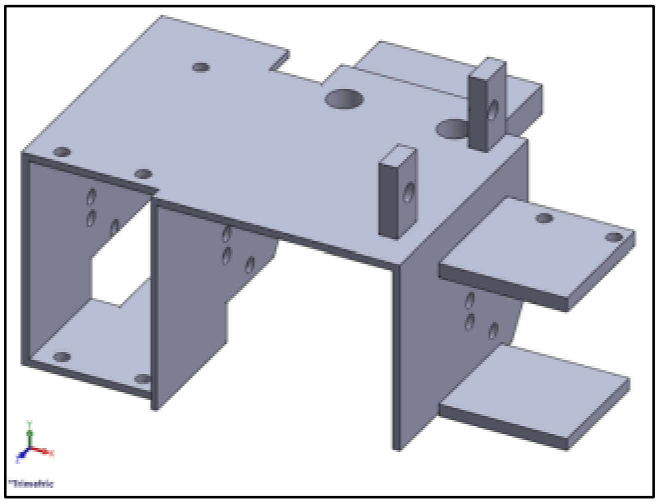


Figure 38: Sensor Housing for Iteration 3

Increasing the sizes of the pockets of the sensor housing made the housing too large to use the complex tilt mechanism as there was no room for the roll rods or the micro servo. For this reason, we redesigned the tilt mechanism to be a four-bar mechanism, shown in Figure 39, driven by a micro servo located on top of the sensor housing. Figure 40 shows the sensor housing in three positions, horizontal, midpoint, and vertical.

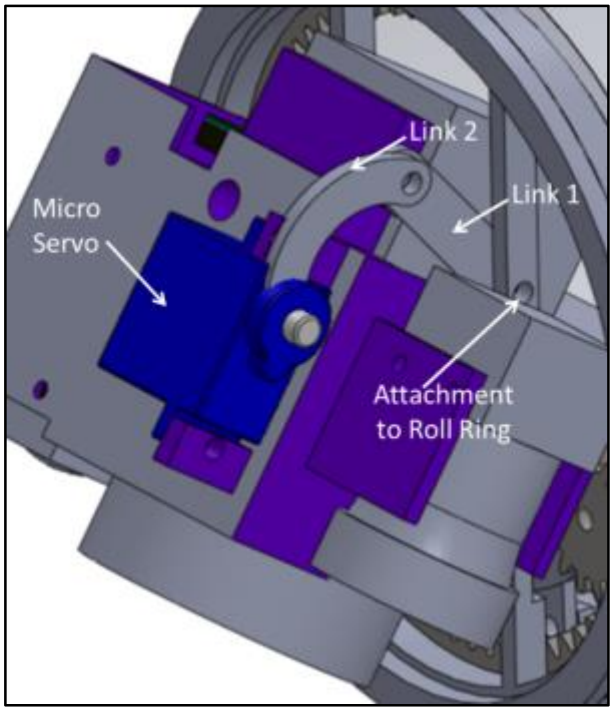


Figure 39: Iteration 3: Four-bar Linkage and Servo

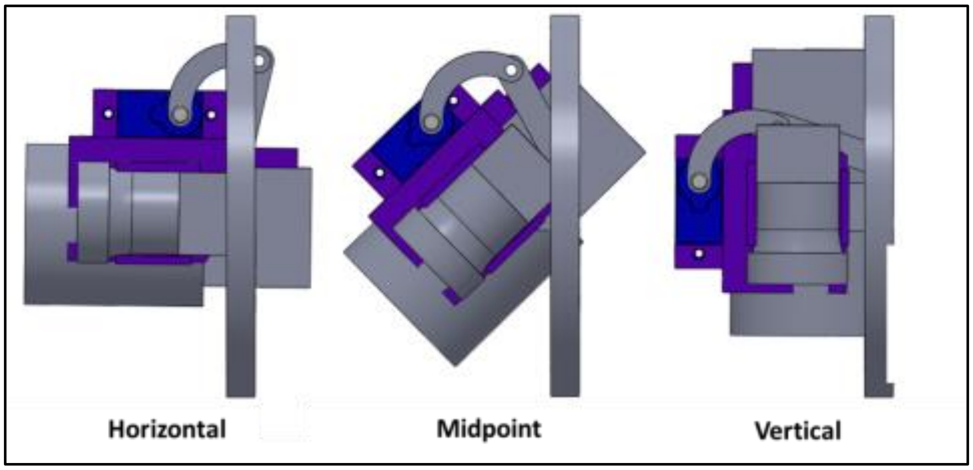


Figure 40: Three Positions of the Four Bar Tilt Mechanism

Figure 41 shows a kinematic diagram of the four-bar linkage. The point the cameras rotate about is O_2. The other anchor is located on one of the vertical bars of the roll ring.

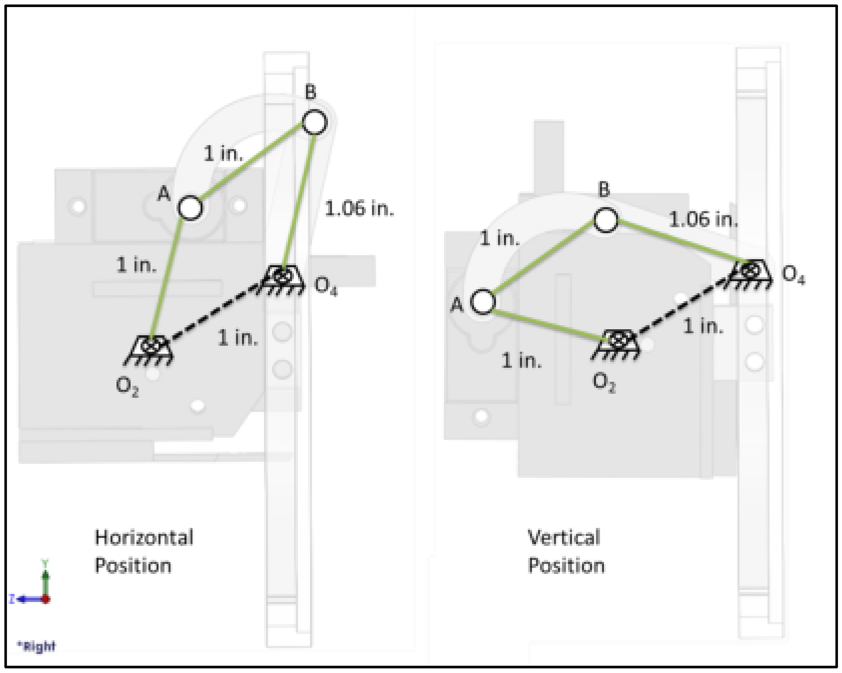


Figure 41: Two Extreme Positions of the Four-bar Linkage

The gear train mechanism also needed to be reevaluated as the spur gear driving the internal gear was very large (1.25” diameter) and collided with the sensor housing. The diameter of this gear needed to be reduced to 0.75”, so the 25 tooth 20 pitch gear was replaced with a 16 tooth 20 pitch gear. This greatly reduced the gear ratio of the mechanism, such that the output of the roll mechanism would be reduced to 103 degrees:

$$\frac{N_1}{N_2} \times \frac{N_i}{N_3} = \frac{25}{10} \times \frac{16}{70} = 0.57 \tag{5}$$

$$Gear\ Ratio \times servo\ range = 0.57 \times 180 = 103\ degrees$$

In order to increase the gear ratio between the first two gears while adhering to the requirements about maximum gear diameter (1.25”) and minimum tooth number, the pitch of gears 1 and 2 was changed to 32 teeth per inch. At this pitch the minimum tooth number for gear 2 was 12. The tooth count on gear 1 at the maximum diameter of 1.25” is given in Equation (6).

$$32 \frac{teeth}{inch} * 1.25\ inches = 40\ teeth \tag{6}$$

This tooth count from Equation (6) can be used in Equation (5) to calculate the gear ratio and output range of the roll mechanism:

$$\frac{N_1}{N_2} \times \frac{N_i}{N_3} = \frac{40}{12} \times \frac{16}{70} = 0.76$$

$$0.76 \times 180\ degrees = 137\ degrees$$

Another goal of this iteration was to make the locking ring and servo housing assembly more modular so that the roll servo could be changed without having to reprint the entire tube-side locking ring. As shown in Figure 42, the locking ring on the tube side was remade into two pieces, the servo housing and the locking ring, attached together using two machine screws in the locations indicated in Figure 43.

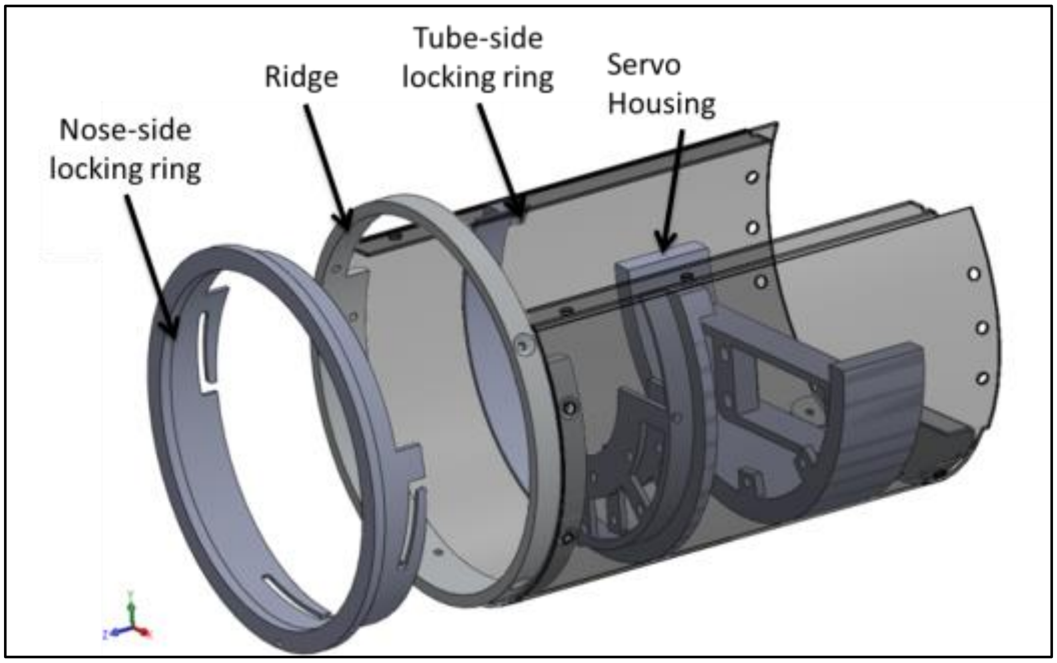


Figure 42: Exploded View of Locking Rings, Ridge, and Servo Housing

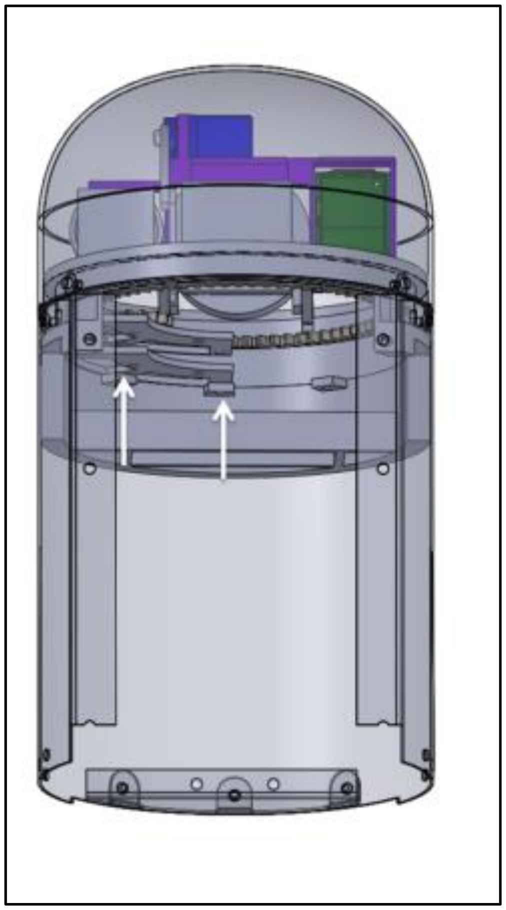


Figure 43: Locations for Screws (at arrows) Connecting the Servo Housing and Tube Side Locking Ring

Once produced, testing showed that the four-bar mechanism was not able to reach the full range suggested by the SolidWorks model due to the fasteners used to hold the mechanism together.

Finally, a piece of the roll ring needed to be cut away, as shown in Figure 44, so that the roll ring did not get caught on the gear axle behind it.

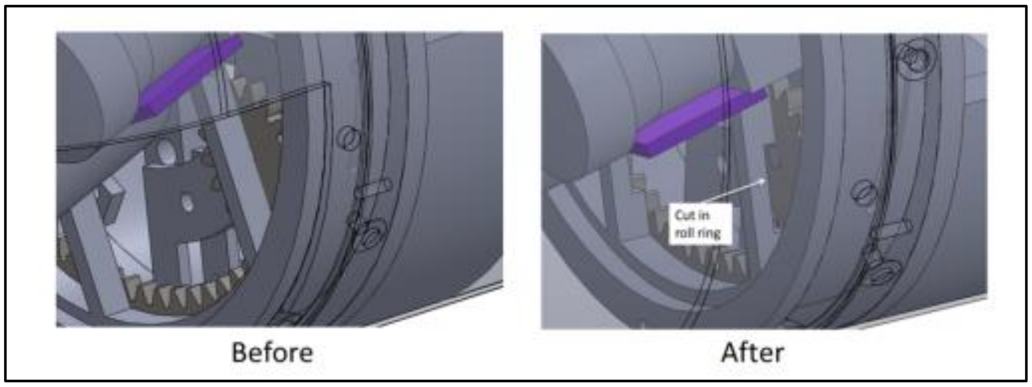


Figure 44: Cut in Roll Ring

Iteration 4

In iteration 4, shown in Figure 45 the four-bar mechanism was resynthesized to improve the range. The anchor points and link lengths determined for this iteration are shown in Figure 47.

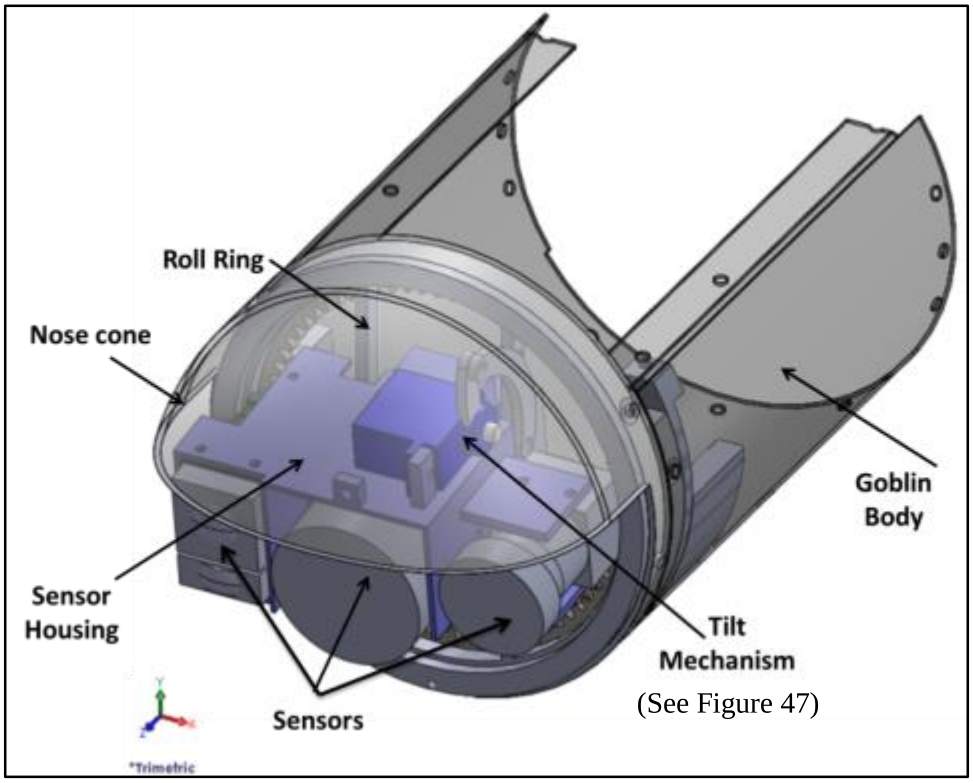


Figure 45: SolidWorks Model of Iteration 4 (Front)

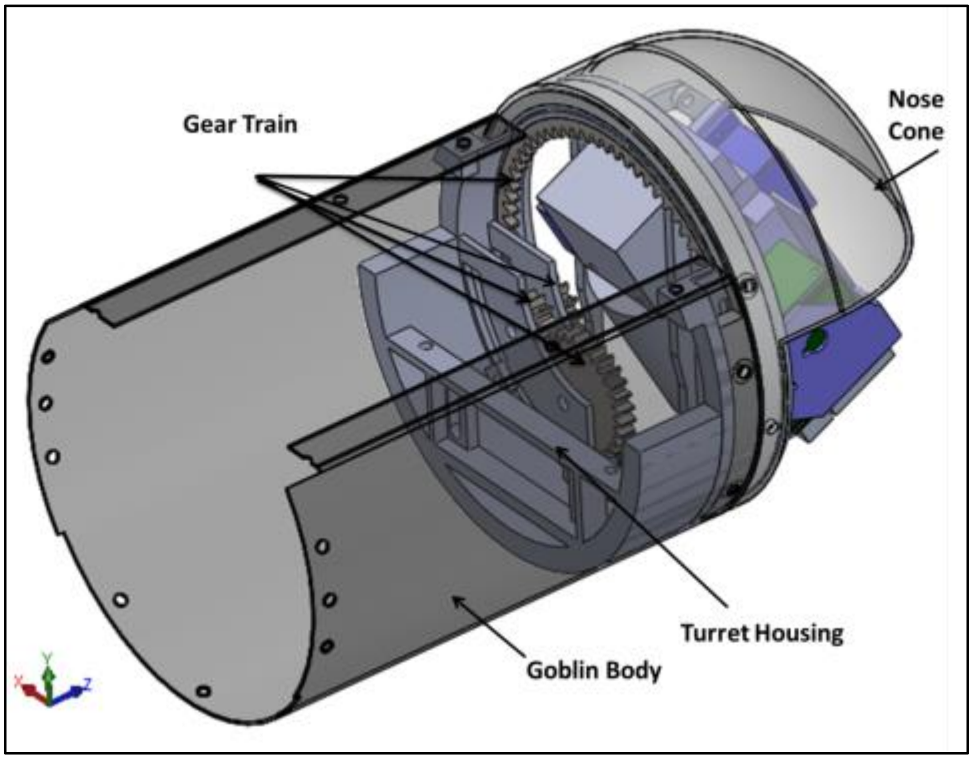


Figure 46: SolidWorks Model of Iteration 2 (Back)

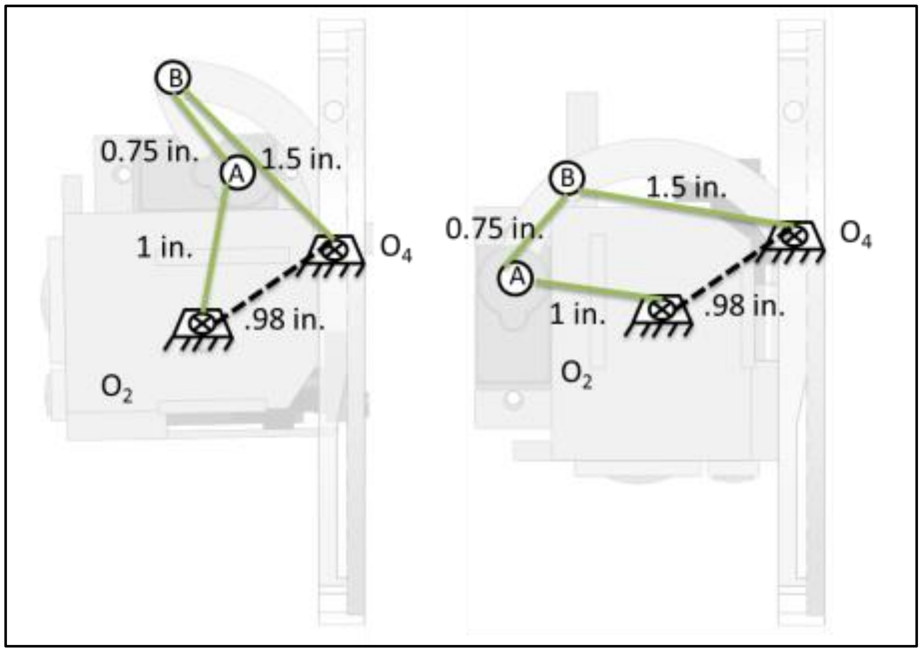


Figure 47: Anchor Points and Link Lengths for Iteration 4 Four-bar

While assembling the turret, it became apparent that the assembly required too many parts and fasteners. It was very difficult for a single person to put together the turret as several of the screw holes were difficult to reach. The modularity behind the detached servo housing in Iteration 3 became a disadvantage when assembling the turret.

Iteration 5

In Iteration 5, shown in Figure 48, the turret housing was changed to be more securely attached to the aircraft. The sensor housing was also altered to allow larger lenses to be used. The tilt mechanism was redesigned to use gears rather than a four-bar mechanism.

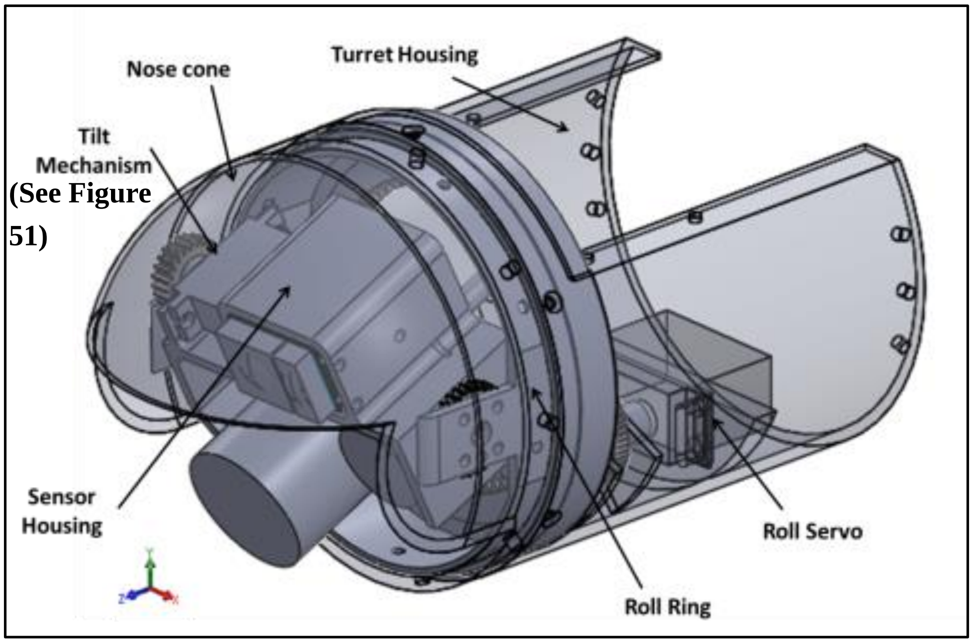


Figure 48: SolidWorks Model of Iteration 5 (Front)

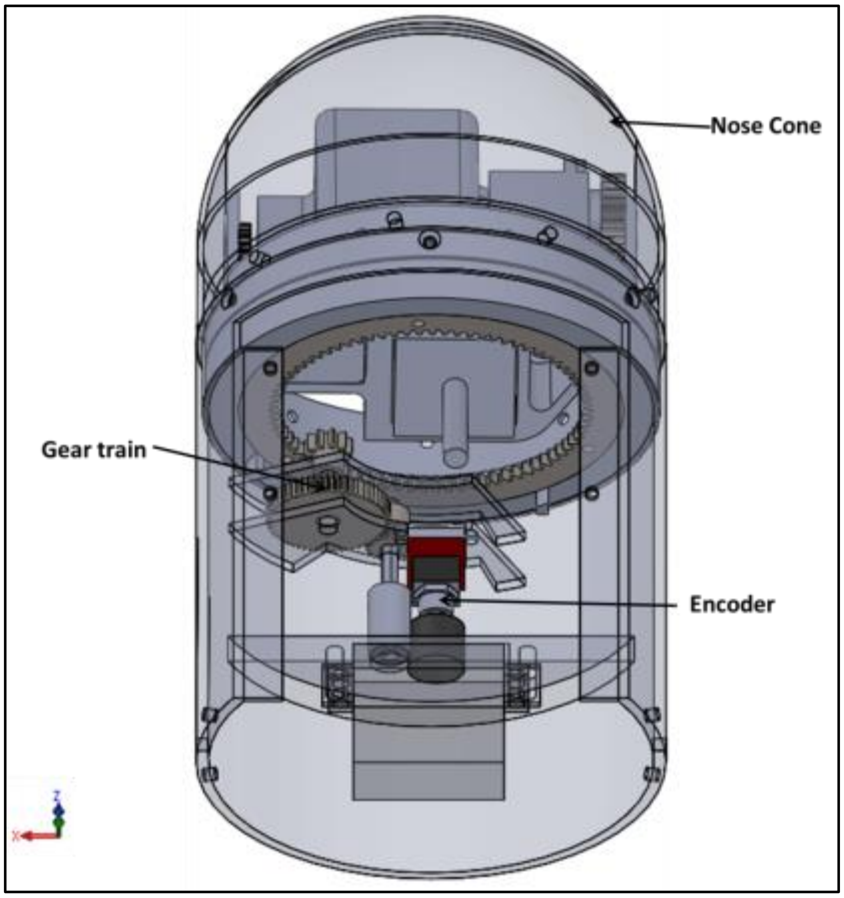


Figure 49: SolidWorks Model of Iteration 5 (Back)

After fitting the turret housing into the WILD Goblin, it was obvious that the fifth iteration design should be well anchored in the body of the WILD Goblin and require fewer pieces than the previous iteration. To do this, we combined the servo housing and tube side locking ring into one piece, which took the place of the metal tube body of the Goblin. The new turret housing will be directly attached to the rest of the Goblin by six screws rather than be inserted into the existing metal body.

Also in this iteration we changed the setup of the camera housing so that there was more room for lenses for the SWIR. In the previous iteration, there was very little space allocated for the SWIR lens, which was limited in diameter to 1.35 inches. In the iteration 5 sensor housing, shown in Figure 50, the Laser Rangefinder was moved to the top of the housing to allow more space for SWIR lenses. This configuration did mean that the nose cone would obscure the Laser Rangefinder for some positions of the turret, but the ability to use larger SWIR lenses was more valued than an always operational rangefinder. Several set of SWIR camera holes allowed the camera to be moved up and down in the housing to accommodate lenses up to 1.75 inches in diameter.

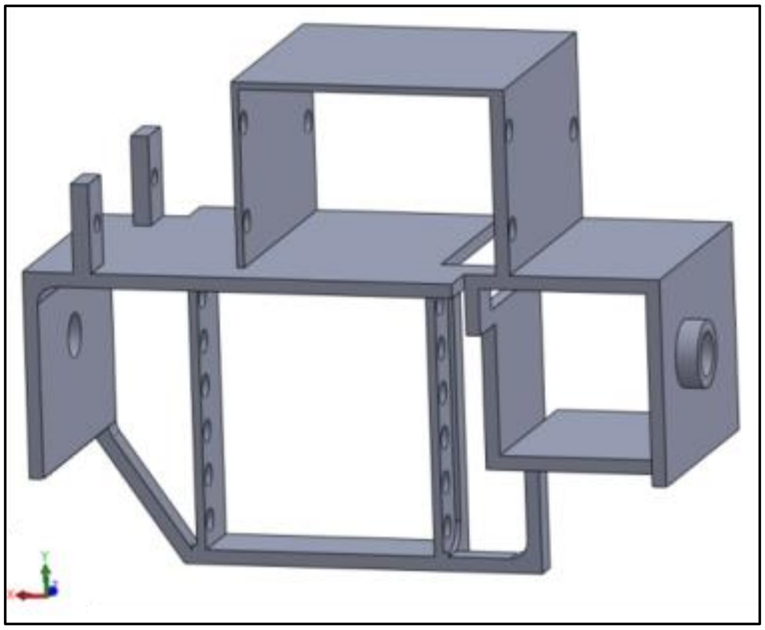


Figure 50: Sensor Housing for Iteration 5

The tilt mechanism was redesigned in this iteration from a four-bar mechanism to geared rotation, which is shown in Figure 51. Gear A in the figure is anchored to the bracket on the roll ring, which is transparent in the figure. The small cylinders on either side of the camera housing shown in Figure 50 insert into the centers of the two Gear A's on either side of the roll ring and can rotate freely inside of them. Gear B is attached to the micro servo on top of the camera housing. Figure 52 shows the position of the mechanism in three positions, to illustrate the motion of the mechanism. This mechanism is geared 1.667:1, increasing the torque supplied by the micro servo.

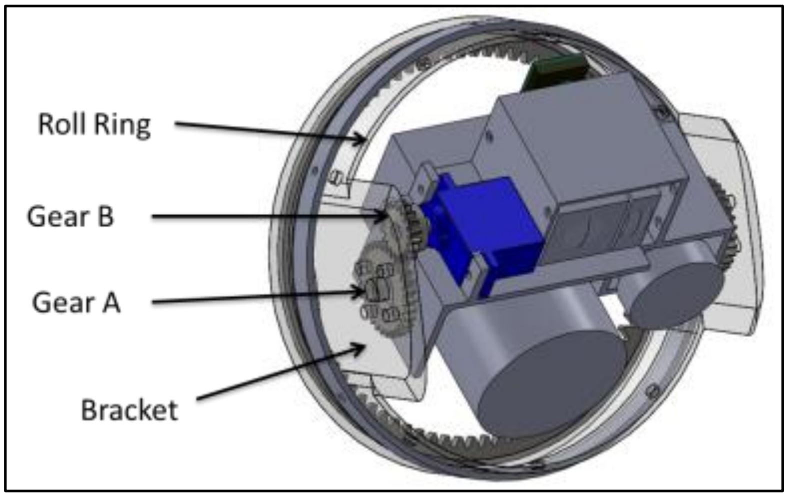


Figure 51: Iteration 5 Geared Roll Mechanism

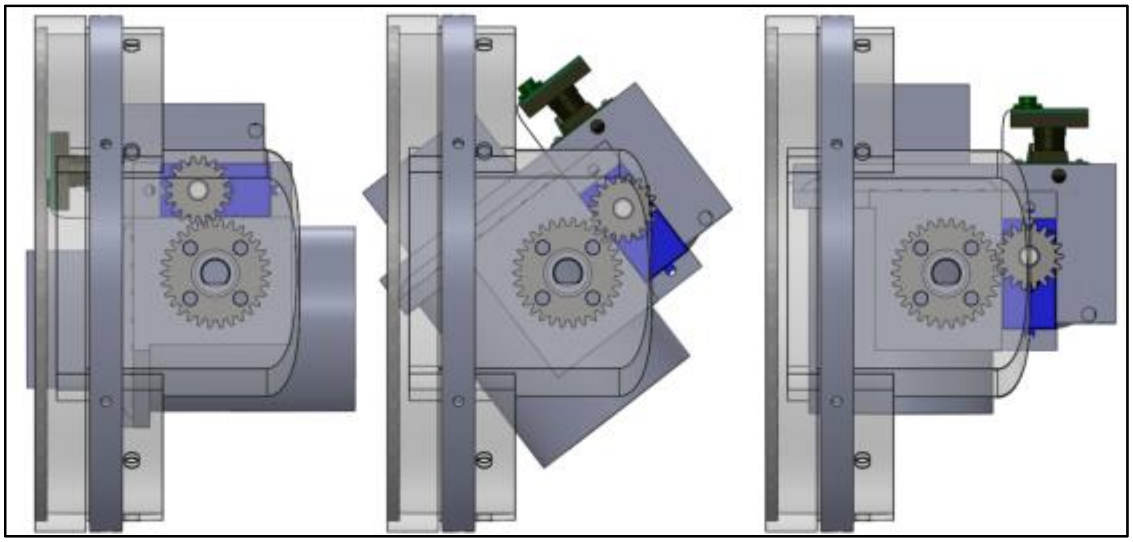


Figure 52: Three Positions of Geared Tilt Mechanism

Finally, part of the roll ring redesign involved attaching the half nose cone to rotate with the roll ring. This increased the field of view for the turret as the sensors were no longer blocked when they rolled past the nose cone.

Most of the issues found in this iteration related to the fit of components. The holes in the nose cone did not line up with the holes in the roll ring. In addition, the sensor holder did not leave enough space for the tilt servo.

Iteration 6

In iteration 6, shown in Figure 53 and Figure 54, the few fit problems from the previous iteration were rectified. The design was also modified to use purchased metal gears rather than 3D printed gears.

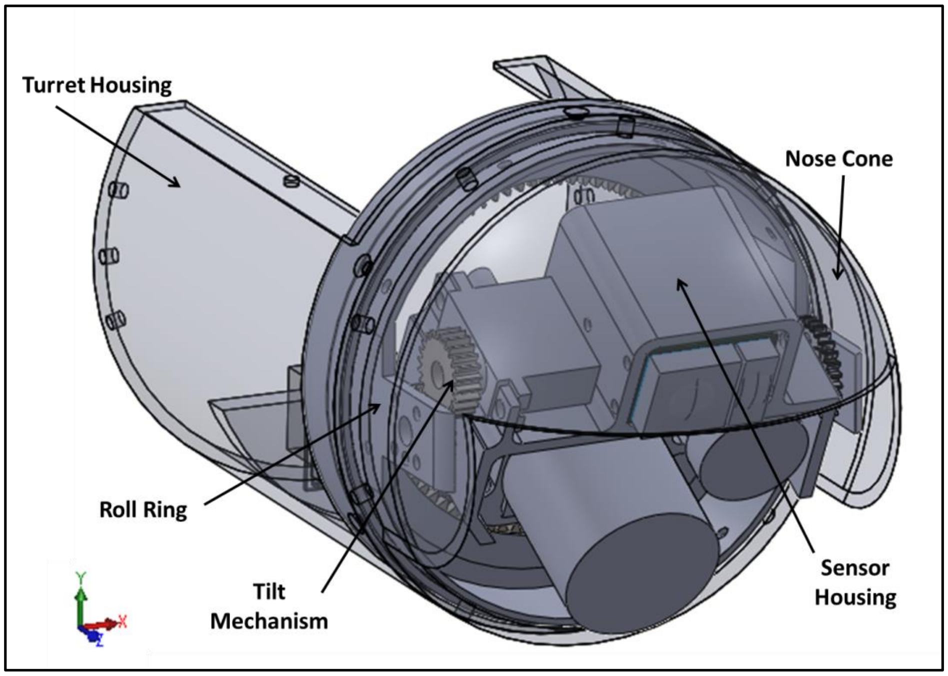


Figure 53: SolidWorks Model of Iteration 6 (Front)

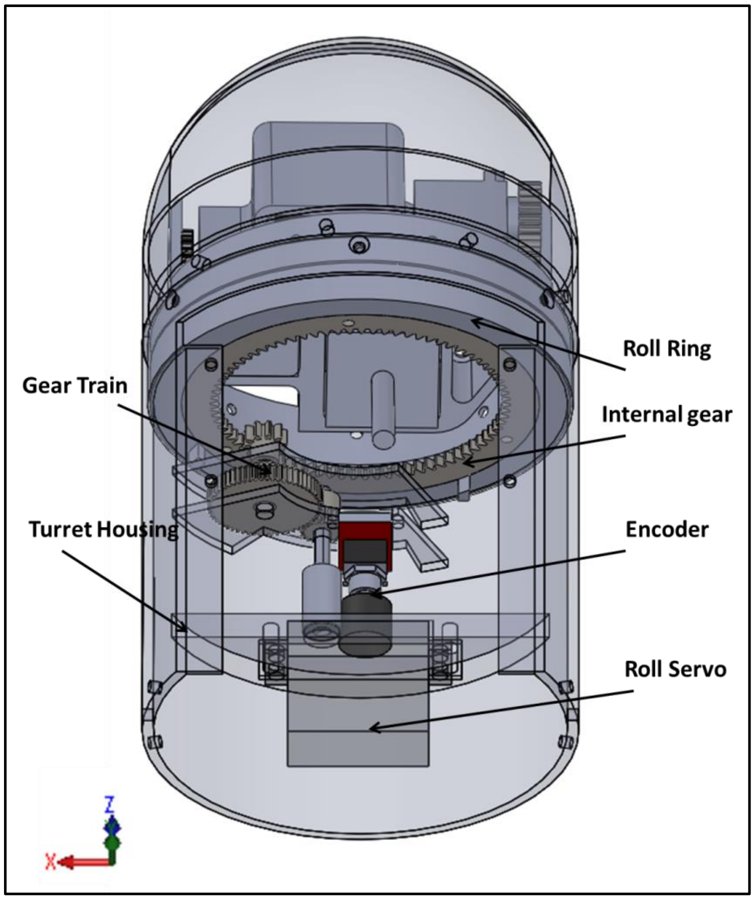


Figure 54: SolidWorks Model of Iteration 6 (Back)

First, our sponsor asked that we design a way for the roll mechanism to be driven by a micro or standard servo. Figure 55 shows the turret housing as well as the piece we designed to allow the servos to be swapped. The servo converter fits into the servo housing in the turret body and has a hole for the micro servo to mount in which lines up with the drive shaft.

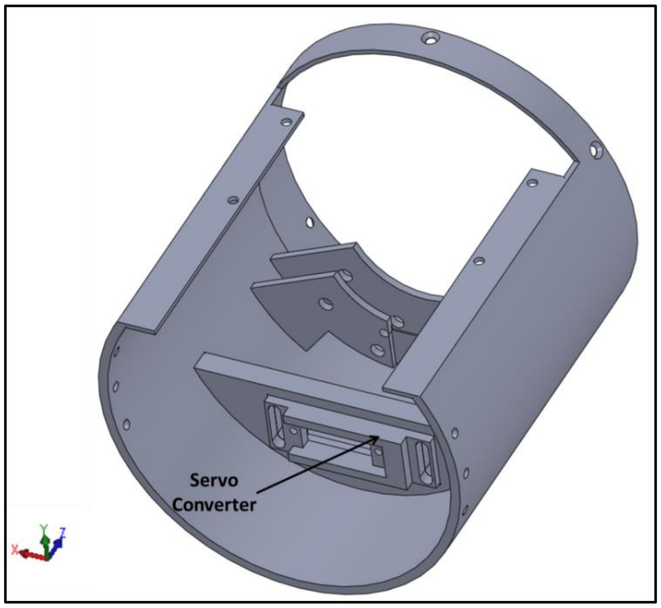


Figure 55: Servo Converter Instered into Servo Housing

Also in the iteration, the sensor housing was changed, as shown in Figure 56, so that it could be machined. Our sponsor wants a machined aluminum sensor holder in order to protect the sensors from damage caused by impact or stress which could deform a plastic servo housing and damage the sensors.

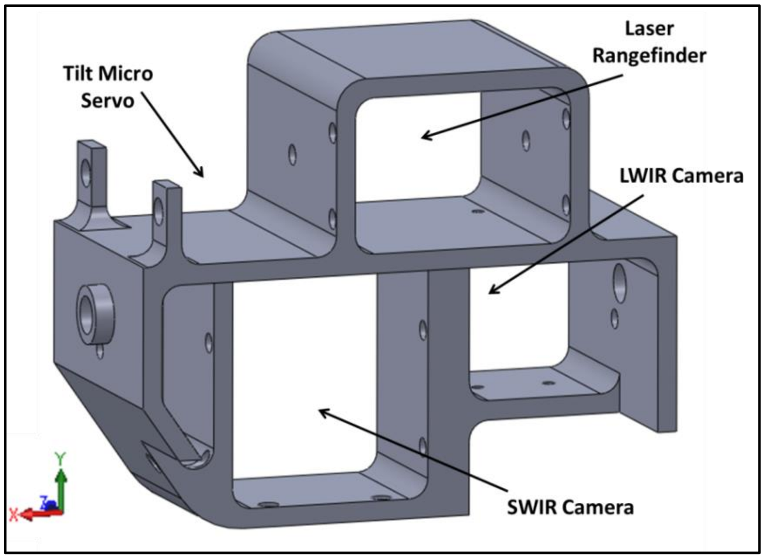


Figure 56: Sensor Housing

In order to mill the sensor mount out of aluminum, input from the TOIL manager, David Scott, was used to improve the machinability of the design. The walls, previously between 0.08” and 0.04” in thickness were increased to 0.125”. Fillets were also added to the design with a radius of 0.125”, as that is the smallest diameter end mill available in TOIL. A 3D printed model of the camera housing to be milled was made in order to test the fit of sensors and the fit of the housing in the assembly.

Also during this iteration, the position servo previously used to drive the roll mechanism was replaced with a continuous turn servo. Use of the continuous turn servo eliminated the need for a gear ratio of 1:1 for the roll mechanism. We were provided with a rotary encoder to use for feedback and it was suggested that we use the encoder shaft and the shaft coupler attached to the roll servo as friction wheels to spin the encoder. The diameter of the drive shaft (D_{shaft}) was 0.42 inches, and the diameter of the encoder ($D_{encoder}$) shaft was 0.6 inches.

$$\frac{D_{encoder}}{D_{shaft}} = \frac{0.6}{0.42} = 1.43$$

In order to have a 1:1 ratio between encoder ticks and output degrees, the gear train from the drive shaft to the output gear needed to be:

$$\frac{24\ ticks}{360\ degrees} \times R_{encoder} = 21.42$$

The constraints for gear selection became the required torque, speed, and accuracy for the roll mechanism, which can be seen in Appendix C. Since the resolution of the encoder was 24 ticks per revolution[2], the gear ratio between the encoder and the roll mechanism had to be at least 1:15 in order to achieve a relationship of one encoder tick per degree. In order to simplify calculations for the micro controller, the goals for the relationship between the encoder and the output motion was to find an integer value for degrees per encoder tick, preferably 1.

Four gears were required to complete the roll mechanism: three spur gears and an internal gear. Iteration 5 used a 20 pitch gear with 80 teeth, making the pitch diameter 4 inches. The two internal gears available for purchase close to this size were a 4 inch pitch diameter (PD), 4.75 inch outer diameter (OD) and a 3.25 inch PD, 3.998 OD gear. The 3.25 PD gear was selected for use as it did not require increasing the diameter of the roll ring and therefore the entire assembly. This gear had a pitch of 24 and 78 teeth.

The gear to drive the internal gear needed to be a small diameter gear in order to not collide with the cameras as they rolled. Since the internal gear chosen for the gear train had a pitch of 24 teeth, the

[2] It was originally thought that the encoder had 24 ticks per revolution as that was the feature stated on the product page. However, in testing it was found that the encoder actually had 4 times as many ticks per revolution, so in the rest of the paper 96 should be the number of encoder ticks per revolution.

gear to drive the internal gear was required to be 24 pitch. The smallest available 24 pitch gear was a 10 tooth gear, so this gear was selected for the mechanism.

Finally, once the gear ratio of the first pair of gears was chosen and the ratio between the encoder and drive shafts was known, the required gear ratio of the remaining gears could be determined. For a compound gear train, consisting of pairs of gears N_1 and N_2, N_3 and N_4…, where the even numbered gears are the output gears, Equation (7) can be used to determine the total gear ratio:

$$\frac{N_2}{N_1} \times \frac{N_4}{N_3} \times \ldots = R \tag{7}$$

Since this gear train consists of three pairs of gears from the encoder to the output motion, the remaining gear ratio could be determined.

$$\frac{N_2}{N_1} \times \frac{N_{internal}}{N_3} = 21.42$$

$$\frac{N_2}{N_1} \times \frac{78}{10} = 21.42$$

$$\frac{N_2}{N_1} = 2.75$$

A set of two 48 pitch gears with 22 teeth and 60 teeth was selected for N_1 and N_2 as the ratio was very close to the required ratio:

$$\frac{N_2}{N_1} = \frac{60}{22} = 2.73$$

Thus the gear ratio from the drive shaft to the output gear was:

$$\frac{60}{22} \times \frac{78}{10} = 21.27$$

The relationship between encoder ticks and degrees of output was:

$$\left(\frac{N_2}{N_1} \times \frac{N_4}{N_3} \times \frac{D_{shaft}}{D_{encoder}}\right) \times \frac{24\ ticks}{360\ degrees}$$

$$\frac{60}{22} \times \frac{78}{10} \times \frac{0.42}{0.6} \times \frac{24}{360} = 0.99 \frac{ticks}{degree}$$

Selection of an aluminum internal gear required changes to the roll ring. Previously, the internal gear had been printed in the same part as the roll ring, but the aluminum gear required a method attachment. Three holes were added to the roll ring, as shown in Figure 57, and three holes were drilled into the aluminum gear using the milling machine in TOIL. The gear was attached to the roll ring using three machine screws and nuts.

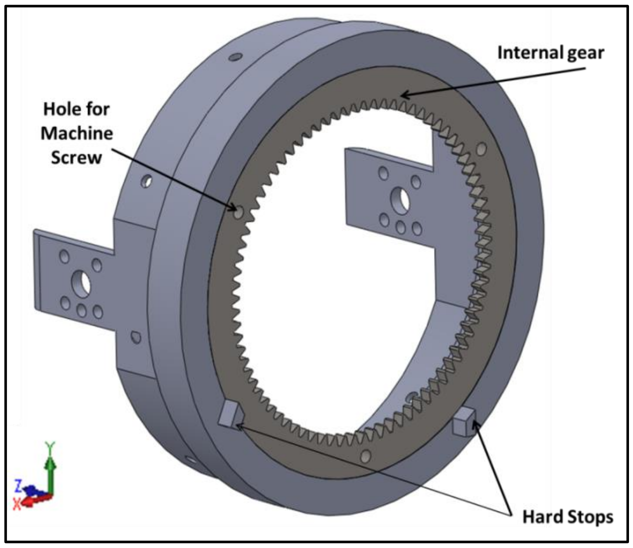


Figure 57: Roll Ring and Internal Gear

One of the issues with this iteration was the fit of the continuous turn servo. The fit of the servo in the housing was slightly off, which resulted in a grinding noise when the mechanism was used. However no deflection or deformation was observed and the model was used in the first iteration test.

Iteration 7

The final iteration incorporated some additional requests by our sponsor, addressed the servo fit issue from iteration 6, and incorporated the encoder for the continuous turn servo. Figure 58 and Figure 59 show the model of this iteration.

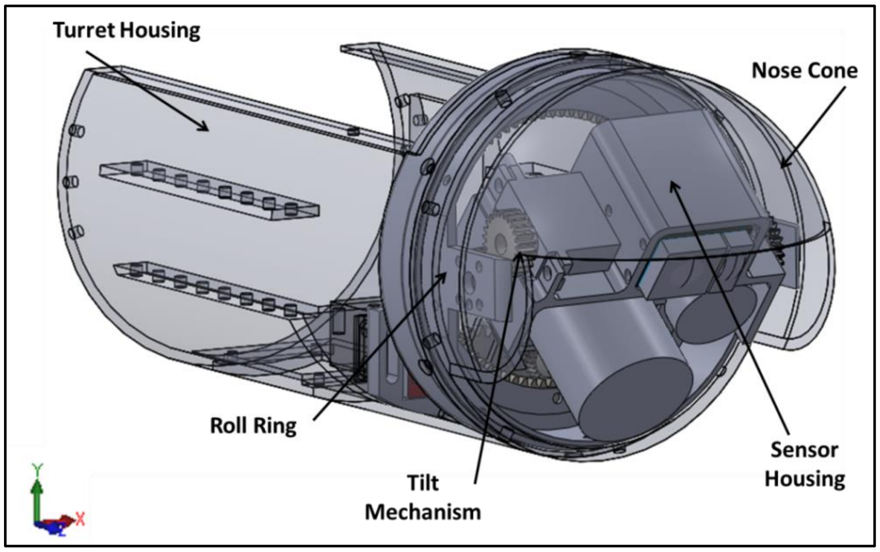


Figure 58: SolidWorks model of Iteration 7 (Front)

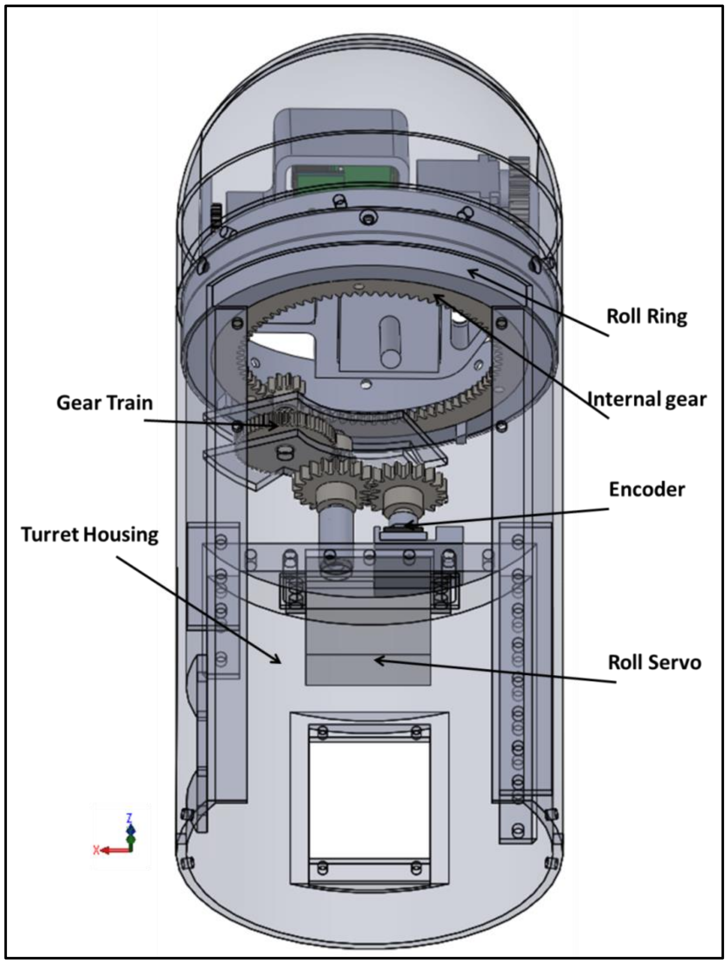


Figure 59: SolidWorks model of Iteration 7 (Back)

Figure 60 and Figure 61show the final turret design prototype. Figure 62 shows the final turret prototype with all three of the sensors held in the sensor housing.

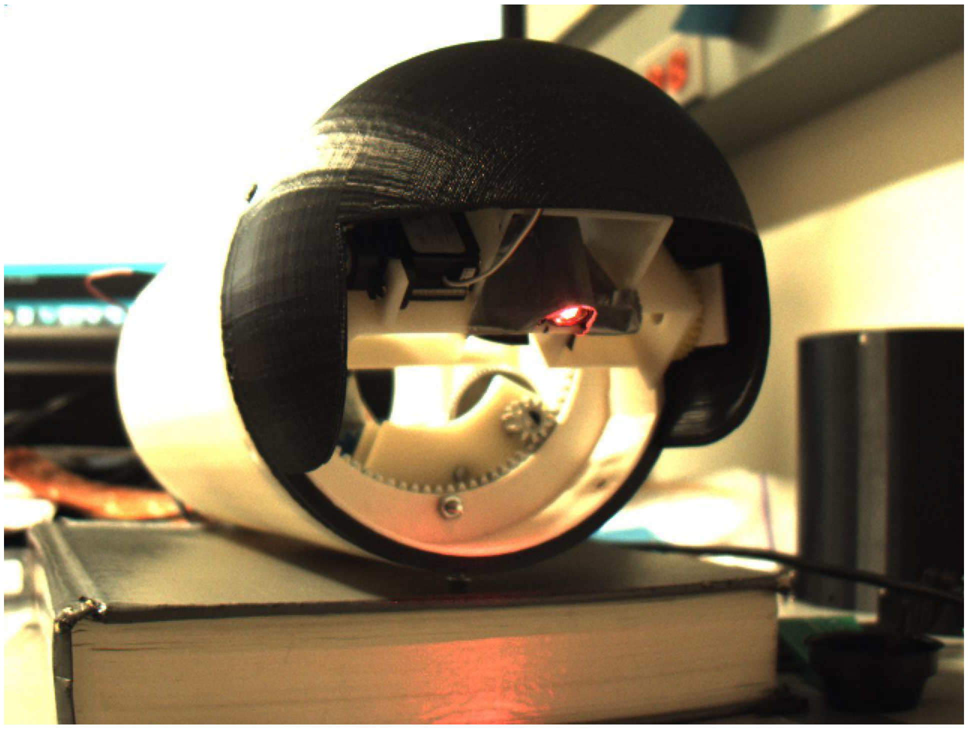


Figure 60: Final Turret Design, Front View

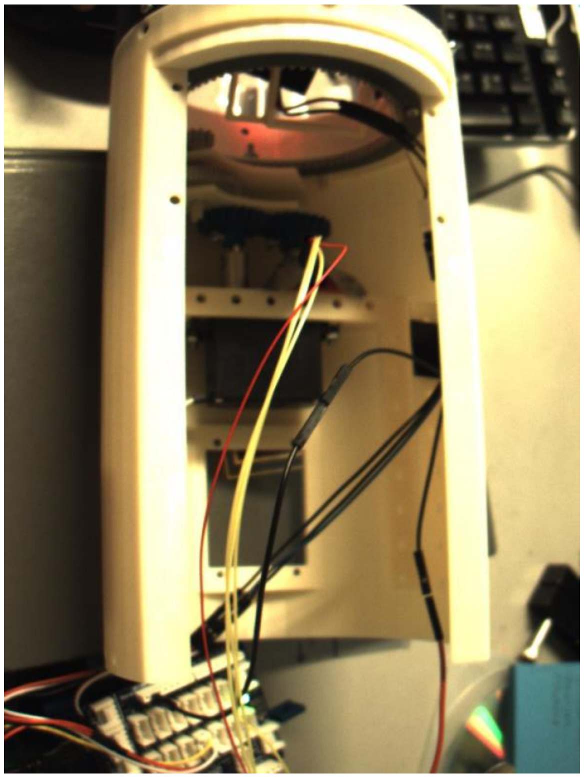


Figure 61: Final Turret Design, Top View

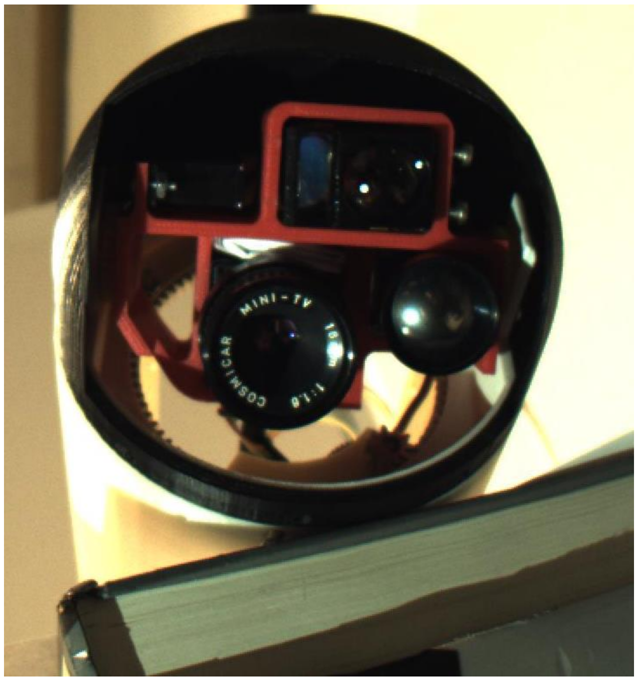


Figure 62: Final Turret Design with Sensors

The improvements in this iteration were isolated to the turret housing. Our sponsor requested that the housing be lengthened beyond the original 5 inch requirement in order to accommodate more electronics and an additional sensor. The length of the housing was extended to 8 inches and the requirement was edited to reflect the change. The additional sensor was a FLIR Rangefinder to be used as a laser altimeter. The sensor was designed to mount such that when the Goblin is flying, the sensor would be pointed straight down towards the ground, as shown in Figure 63. In addition, a mount for the electronics for the sensor was added to the side wall of the turret housing.

The second change to the turret housing was the addition of shelves for electronics. To accomplish this, racks were added to the sides of the turret housing, as shown in Figure 63 to which electronics or shelves could be mounted. There were many benefits of adding racks for shelves to be attached to rather than built in shelves. The racks did not block access to the roll servo, which would have been covered by shelves. As there are two layers of racks, one or both can be added depending on the size of the electronics needed.

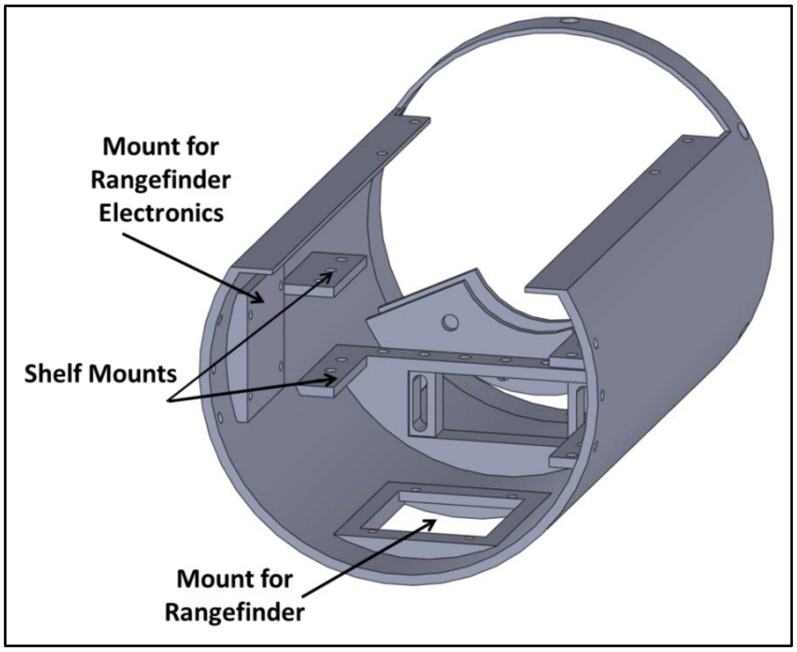


Figure 63: Turret Housing with Space for Rangefinder, Electronics, and Shelves

Finally, the encoder was mounted in the turret housing to give feedback for the roll motion. The mount for the encoder attached to the front of the servo housing, as shown in Figure 64. The mount fit into the opening in the servo housing and was attached to the slot in the housing.

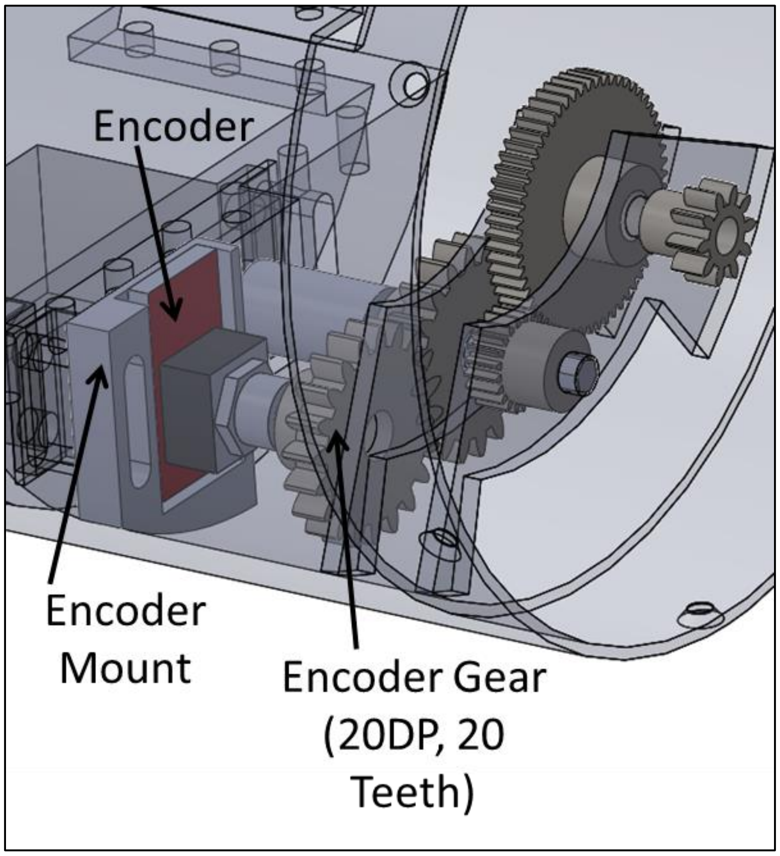


Figure 64: Final Encoder Mount

The encoder and drive shaft were to be geared in order to provide feedback to the roll mechanism. In order to have the required 1 degree of accuracy for the output of the roll mechanism, the ratio of encoder ticks to output motion in degrees needed to be at least 1:1. A ratio greater than 1:1 (i.e. more encoder ticks per degree of output roll motion) would result in a more precise mechanism, as increments smaller than one degree of output could be measured.

As shown in Equation (8), the gear ratio between the drive shaft and the output gear was 27.273:1. The encoder used for feedback had a resolution of 3.75 degrees per encoder tick.

$$\frac{21.273\ degrees\ input}{1\ degree\ output} \times \frac{1\ tick}{3.75\ degrees\ of\ encoder\ motion} = 1 \qquad (8)$$

$$\frac{degrees\ of\ encoder\ rotation}{degrees\ input} = \frac{1}{5.67}$$

As we were unlikely to reach a ratio of 1:5.67 in the space allotted for the encoder and gearing, this was not a large concern. It was decided that to simplify calculations on the microcontroller, the mechanism should use a 1:1 gear ratio between encoder and drive shaft. As shown in Figure 64, a 20 pitch, 20 tooth gear was attached to the shaft of the encoder and another 20 pitch 20 tooth gear was added to the drive shaft to drive the encoder gear.

With the 1:1 gearing, the output angle of the roll motion could be determined using Equation (9):

$$degrees\ output = \frac{encoder\ ticks}{5.67} \tag{9}$$

5.3. Integration Tests

During this project, two integration tests will occur. The first will occur at the midpoint of the project and the second will occur at the conclusion. The purpose of the integration tests is to integrate the hardware and software components of the project, as the final deliverable for the project is a system consisting of both. During the first integration test, whatever software and hardware is ready for testing will be used to test all available requirements according to the Verification Plan in Appendix D. During the second integration test, all requirements will be tested in order to be able to show that the sensor turret meets all requirements.

First Integration Test

At the time of integration test 1, not all requirements were able to be tested. The encoder module had not been completed, so the roll mechanism could not be tested. The accelerometer was also unavailable as it was being used in the test turret and there was no scale available to test torque. Table 4 shows the results of the test:

Table 4: Results of First Integration Test

ID	Category	Requirement	Verification Test	Result	Pass/Not Pass
R02	Size	The outer diameter of the turret assembly shall be less than 5.375" in diameter.	Measurement of diameter	4.875"	Pass
R03	Size	The complete length of the turret assembly shall be no longer than 5".	Measurement of body	5"	Pass
R06	Speed - Tilt Platform	The tilt mechanism shall be able to tilt at a rate of at least 15 degrees/second.	Use a timer and do several trials.	37.57 degrees	Pass
R10	Field of Regard - Tilt Platform	The tilt mechanism shall tilt at minimum 45 degrees with a goal of 90 degrees.	Use mounted laser pointer and measure change in position and convert to angle	90 degrees	Pass
R12	Sensor Capacity	The turret assembly shall hold the Jenoptik Laser Rangefinder, the Flir QUARK, and the Sensors Unlimited MicroSWIR.	Put sensors in camera housing. Check for appropriate fit.	All Sensors Were Fit	Pass
R13	Accuracy	The accuracy of the mechanism shall be less than 1 degree (the actual position of the turret should be within 1 degree of the specified position).	Use the field of regard test. Calculate "ideal" reading and compare to actual data. Repeat 20 times.	5.31 degrees	Not Pass
R14	Repeatability	The repeatability of the mechanism shall be less than 1 degree.	Compare the different trials of the accuracy test for repeatability	0.34 degrees	Pass

Seven tests were performed during this integration test. Of these seven, two were not passed by the system. The accuracy of the mechanism, calculated as the average error in each movement, was 5.27 degrees, as opposed to the requirement of 1 degree. In addition, the repeatability of the mechanism was 1.98 degrees, rather than the 1 degree required.

The accuracy and repeatability tests were performed by moving the tilt mechanism through a range at increments of 5, 10, and 15 degrees from 90 degrees to 30 degrees and back. As shown in Figure 65, the angle was determined by attaching a laser pointer to the sensor holder, measuring the position of the laser on a flat wall, and using this value to find the angle of the mechanism.

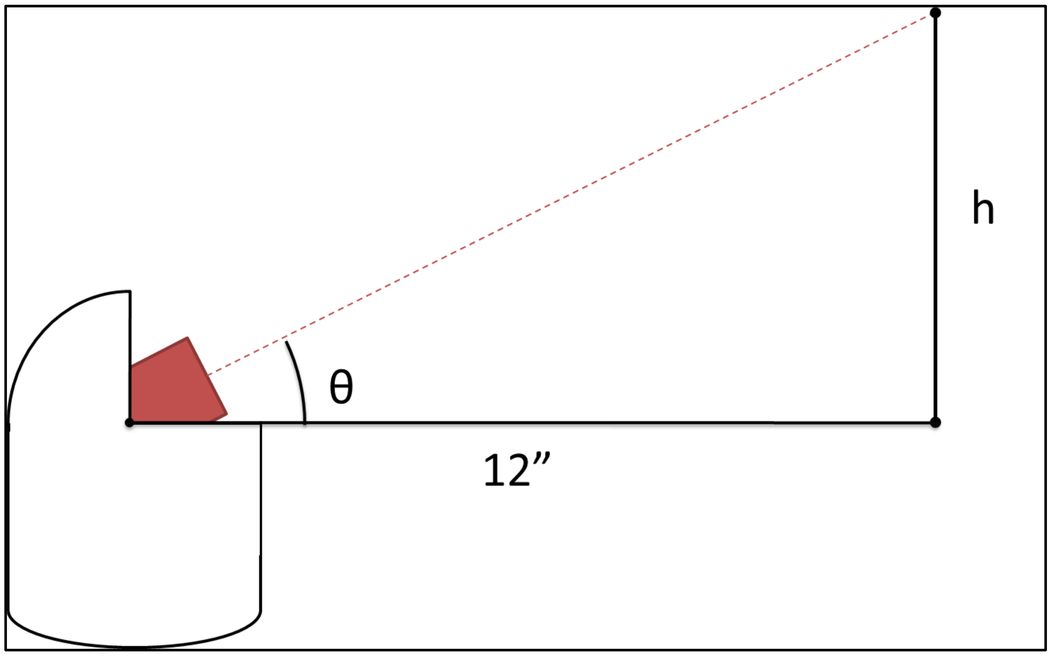


Figure 65:Test Setup for Accuracy and Repeatability of Tilt Mechanism

When the angles calculated during the test are separated into those taken when the mechanism was moving up (from 90 to 30 degrees) or down (from 30 to 90 degrees), and compared only to those readings in the same set, the repeatability is 0.34 degrees, far below the requirement of 1 degree. This pattern is indicative of hysteresis in the system; the motion output is dependent on the previous motion, n this case whether the mechanism is raising or lowering. Figure 66 shows a graph of the measurements when the mechanism is raising and lowering.

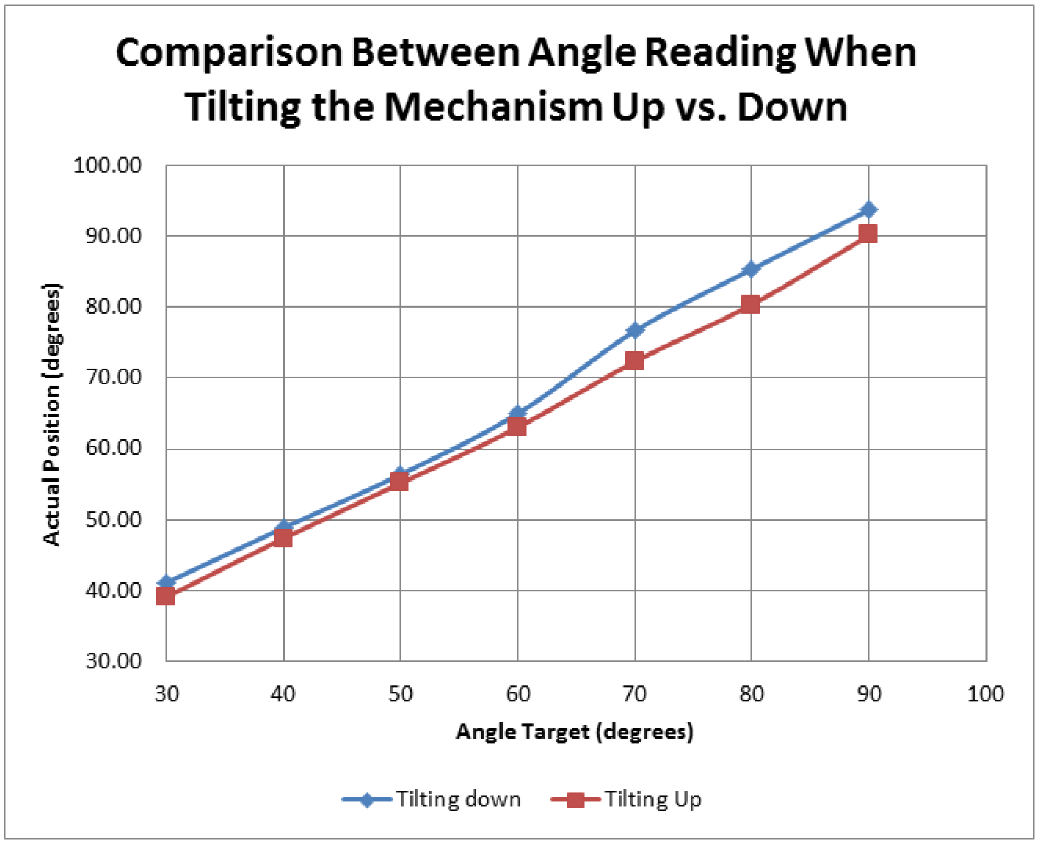


Figure 66: Comparison Between Angle Reading When Tilting the Mechanism Up vs. Down

If the hysteresis can be mediated, the mechanism could be calibrated to have more accurate position. One option for mediating the hysteresis is to pick one motion, raising or lowering, and have the mechanism always complete a motion moving in the same direction. For example, the mechanism is commanded to move from 45 degrees to 70 degrees, it could move past 70 degrees to 75 degrees and then back to 70 degrees so that 70 degrees from 45 degrees is the same as 70 degrees from 75 degrees.

Second Integration test

During the second integration tests, more of the tests were performed on the turret assembly. At the time of testing, the IMU was not able to be used to test the acceleration, and a spring scale could not be found to test torque. The results of the tests performed are shown in Table 5 below. The implications of these results are discussed in the next chapter.

Table 5: Results of Second Integration Test

ID	Category	Requirement	Verification Test	Result	Pass/Not Pass
R02	Size	The outer diameter of the turret assembly shall be less than 5.375" in diameter.	Measurement of diameter	4.875"	Pass
R03	Size	The complete length of the turret assembly shall be no longer than 5".	Measurement of body	10.875"	Pass*
R06	Speed - Tilt Platform	The tilt mechanism shall be able to tilt at a rate of at least 15 degrees/second.	Options: 1. Use a timer and do several trials. 2. Limit switches and software timer. 3. Use the already implemented gyro. Mount very close to the turning center of the tilt motion.	68.3 degrees/second	Pass
R07	Speed - Roll Platform	The roll mechanism shall roll at a rate of at least 15 degrees/second.	Options: 1. Use a timer and do several trials. 2. Limit switches and software timer. 3. Use the already implemented gyro. Mount very close to the turning center of the roll motion.	18.8 degrees/second	Pass
R10	Field of Regard - Tilt Platform	The tilt mechanism shall tilt at minimum 45 degrees with a goal of 90 degrees. (Figure 2)	Use mounted laser pointer and measure change in position and convert to angle	90 degrees	Pass
R11	Field of Regard - Roll Platform	The roll mechanism shall roll a minimum of 90 degrees with a goal of 180 degrees. (Figure 3)	Use mounted laser pointer and measure change in position and convert to angle	160 degrees	Pass
R12	Sensor Capacity	The turret assembly shall hold the Jenoptik Laser Range finder, the Flir QUARK, and the Sensors Unlimited MicroSWIR.	Put sensors in camera housing. Check for appropriate fit.	All Sensors Were Fit	Pass
R13	Accuracy	The accuracy of the mechanism should be 1 degree (the actual position of the turret should be within 1 degree of the specified position).	Use the FoR test. Calculate "ideal" reading and compare to actual data. Repeat 20 times.	Tilt: 0.84 degrees Roll: 0.44 degrees	Pass
R14	Repeatability	The repeatability of the mechanism should be 0.1 degrees.	Repeat FoR Test 50 times.	Tilt: 0.31 degrees Roll: 0.44 degrees	Pass*

5.4. Summary

The purpose of this section was to describe the results of the design iterations as well as the results of the integration tests. There were seven design iterations during the project, each of which consisted of six major components. The integration tests were performed in order to ensure that the sensor turret met the mechanical requirements, performance requirements, and could implement the control software. The results of the first integration test showed that the tilt mechanism had unsatisfactory accuracy and repeatability. A software solution was implemented before the second integration tests in order to improve the performance of the sensor turret during the second integration test.

Section 6. Software Results

6.1. Introduction

Various program modules were created for the different stages of the prototyping process. Figure 67 shows the flowchart of the different program modules that were programmed using the Arduino IDE and how the modules interact with one another. Each module is explained in greater detail later in this section.

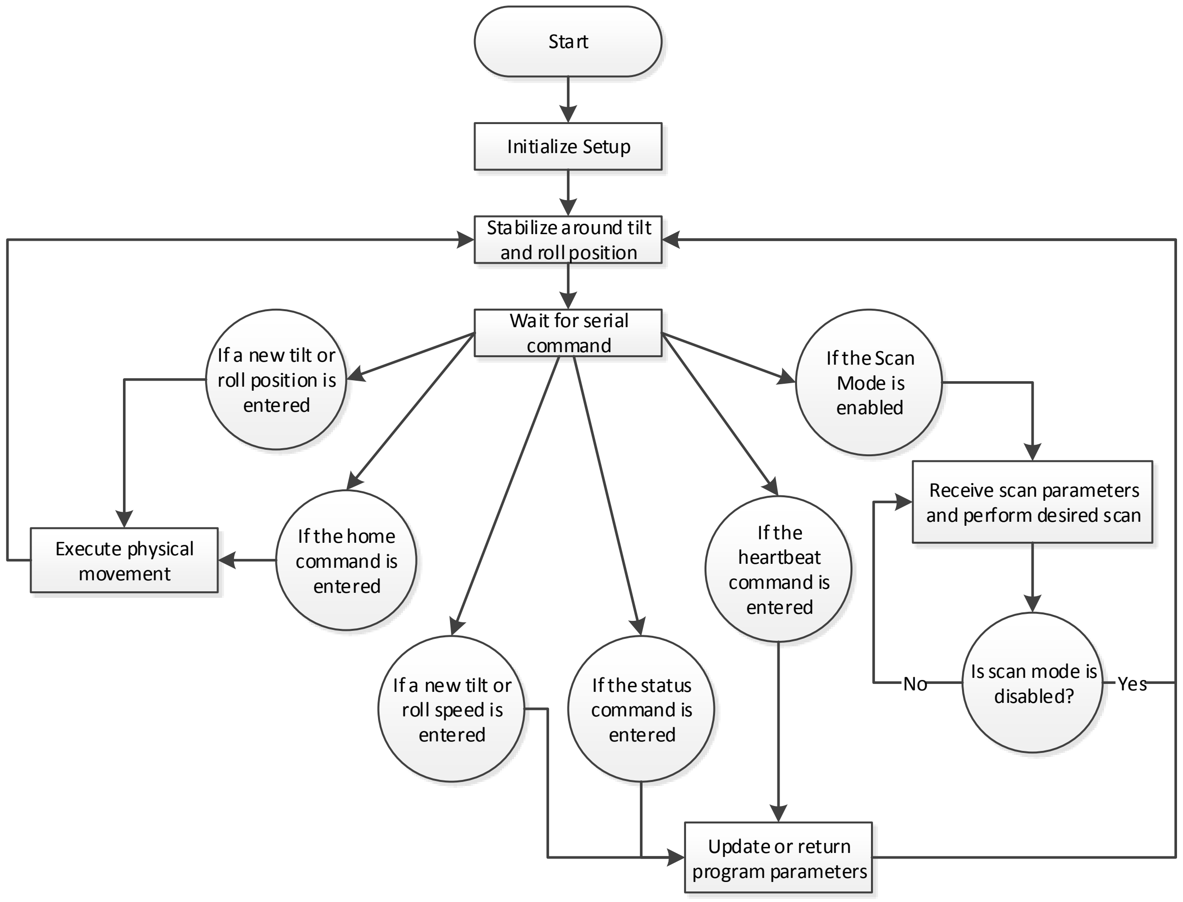

Start
Initialize Setup
Stabilize around tilt and roll position
Wait for serial command
If a new tilt or roll position is entered
If the home command is entered
If a new tilt or roll speed is entered
If the status command is entered
If the heartbeat command is entered
If the Scan Mode is enabled
Receive scan parameters and perform desired scan
Is scan mode is disabled?
No
Yes
Execute physical movement
Update or return program parameters

Figure 67: Flowchart of Program Modules

The different modules included a serial command module which included position and speed change of the servos, a status command, a home command, and a heartbeat trigger. Another module was a scans program which enabled the turret to move in preprogrammed patterns including a raster scan, spiral scan, and clover scan. A stabilization program module was developed in order to stabilize the turret, which was eventually used to decouple the sensor view from the motion of the aircraft in the final assembly. The stabilization module and scan program were then combined to stabilize the turret while the various scans were being performed. Target tracking was the final stage of the modular programming process, which first involved optical target tracking (i.e. just using the sensors) and then optomechanical target tracking (i.e. target tracking with the sensors moving in the turret assembly).

6.2. Test Turret Setup

In the test turret, all servos were Hitec brand servos and all had metal gearing on the inside, as opposed to plastic gearing. The old (first) test turret tilt servo and roll servo were both the HS-322HD 180 degree position servos. The new turret's tilt servo was the HS-626MG and the new roll servo was the HS-322HD, which are both 180 degree position servos. Figure 68 shows a picture of the test turret.

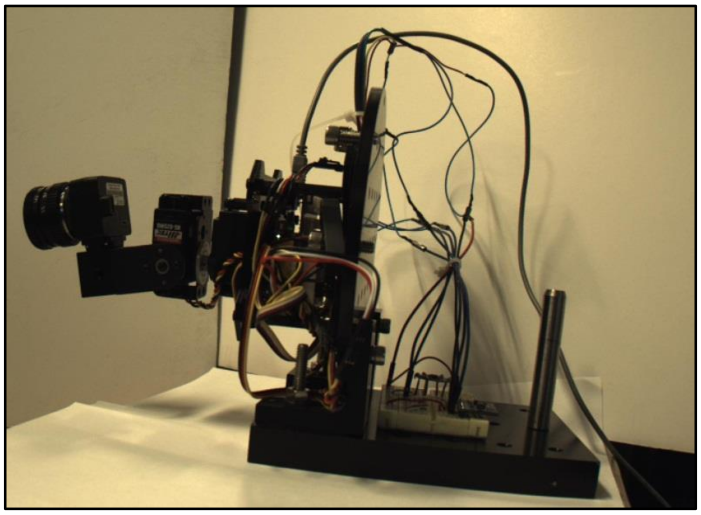


Figure 68: Test Turret Setup

The position of the servos are controlled by a pulse width, where the different pulse widths sent from the microcontroller (described more in the following section) correspond to an angle (e.g. for the HS-322HD, a pulse width of 600 milliseconds corresponds to -90 degrees and a pulse width of 2400 milliseconds corresponds to 90 degrees, but in the setup we mapped the 600 ms pulse to 0 degrees and 2400 ms pulse to 180 degrees). A new test turret was needed about halfway through the project because the older test turret's servos became worn down mechanically over long and repetitive usage so the movements of the older servos were sluggish and did not perform the desired tasks efficiently.

6.3. Arduino Capabilities

The Arduino Duemilanove was the Arduino board that came mounted to the test turret assembly along with a compatible motor shield. The Arduino Duemilanove Arduino board was primarily chosen due to its availability and knowledge we had about programming in the Arduino IDE. The Arduino Duemilanove has an Atmel ATmega328 microprocessor that has a clock speed of 16Mhz and is capable of storing programs up to 30 kB (the amount of flash on the ATmega328 is 32 kB, but around 2 kB is used for the Arduino IDE bootloader). The Arduino IDE also offers a simple serial interface, called the Serial Monitor, which allowed for a string of serial data to be entered and proved useful when implementing the serial commands that are discussed in the following parts of this section.

6.4. Serial Commands

Point-to-point movement using serial commands

The first requirement for turret control was to do basic turret movement using serial commands. The serial monitor in the Arduino IDE was used to send a string of serial commands to the microcontroller, which then controlled the turret actions. A number from 0 degrees to 180 degrees followed by either a "p" (to roll) or a "t" (to tilt) moved the corresponding servo to the specified angle. For example, "45p66t" entered in the serial monitor moved the roll servo to 45 degrees and the tilt servo to 66 degrees. Both servos moved simultaneously to a position to archive the desired fluidic motion requirement of the servos.

If the angle entered was greater than the preprogrammed maximum servo angle or less than the minimum servo angle, then the new angle positions were defaulted to the maximum servo angle or minimum servo angle, respectively. For example if the maximum angle for the roll servo was programmed to be 180, and the command 222p was entered, the roll servo would only move to the 180 degree position.

Ability to change the speed of position servos

The next serial commands that were implemented were servo speed change options. The speed of the servos needed to be variable because the turret will eventually be used for target tracking, and not all targets will be moving at the same speed. Changing the speed of each servo individually will allow the turret to accurately track the target. The command entry into the serial monitor for the speed change was similar to the position commands, but for the speed command an "r" after a number changed the tilt speed to that degree per second speed and an "o" after a number changed the roll speed to that speed. For example, the entered command "70r88o" changed the speed of the tilt servo to 70 degrees per second (dps) and changed the speed of the roll servo to 88 degrees per second. The speeds of the servos ranged from 0 degrees per second (stopped) to the no load speed of the servos, which can be entered as a variable in the code so servos can be swapped.

In order to change the speed of the position servos in the test turret, there needed to be a delay in between each degree step of the servos that would make the servo move at the desired speed in degrees per second. Table 6 explains the variables used in the speed change equations. The word "Angle" in Equation (10), Equation (11), Equation (12), and Equation (13) is replaced by "tilt" in the NewDelayTimeTilt function and by "roll" in the NewDelayTimeRoll function in order to produce the new delay times needed to change the speed of the tilt servo and roll servo, respectively.

The speed change functions take in the desired servo speed in dps and the function returns the new delay time for needed for the desired speed.

Table 6: Variables for Speed Change Equations

Variable Name	Use
noDelTimeAngle	Time it would take the servo to move 1 degree with no speed change
NLSpeedAngle	The preprogrammed no load speed from the specific servo based on the data sheet from the servo
delTimeAngle	Time needed to move to 1 degree at desired speed (i.e. servoSpeedAngle)
servoSpeedAngle	Desired servo speed
difDelTimeAngle	Difference in the delay time and the no delay time
newDelTimeAngle	Delay needed for each step in milliseconds to move servo at desired speed

$$noDelTimeAngle = \frac{1.0}{NLSpeedAngle} \tag{10}$$

$$delTimeAngle = \frac{1.0}{servoSpeedAngle} \tag{11}$$

$$difDelTimeAngle = delTimeAngle - noDelTimeAngle \tag{12}$$

$$newDelTimeAngle = difDelTimeAngle \times 1000.0 \tag{13}$$

Interrupts were implemented in the code in order to accurately produce the delay needed to control the speed of the servos. In the interrupt service routine (ISR), there is a segment of code that checks to see if a certain amount of time passes and if a certain amount of time has passed (the delay time), a flag (either “rollNow” or “tiltNow”) is triggered to a 1 which allows the servos to move in the main loop of the code. At the end of the movement, the “rollNow” or “tiltNow” variable is set to 0 and is not set to 1 again unless the certain delay time passes.

The default speeds for the servos were programmed to be 60 dps for each servo. If the entered speed is greater than the preprogrammed maximum servo speed (i.e. speed at no load), then the servo speed is defaulted to the no load speed. For example if the maximum speed for the test turret’s roll servo was programmed to be 316 dps, and the command 432o was entered, the servo speed would be adjusted to 316 dps. If the speed entered was 0 dps, then the servo would stop in that current position. If the servo is commanded to move to a new position from a stop (0 dps), then the stopped servo speed is set to the default servo speed (60 dps) and the servo moves to the newly desired location.

The calculations for the maximum no load speed for the two servos used in the test turret are shown below in Equation (14) and Equation (15).

Tilt Servo: HS-626MG: The operating speed at no load with 4.8V applied is 0.18sec/60°. Using this information:

$$Speed\ at\ no\ load = \frac{60°}{0.18sec} \cong 333\ degrees\ per\ second \tag{14}$$

Roll Servo: HS-322HD: The operating speed at no load with 4.8V applied is 0.19sec/60°. Using this information:

$$Speed\ at\ no\ load = \frac{60°}{0.19sec} \cong 316\ degrees\ per\ second \tag{15}$$

Having a speed option allows for the speed to be scaled in order to accommodate any gearing to obtain the desired end effector movement rate in degrees per second in the final assembly.

The serial command program also includes an option for a heartbeat, which sends the message “BEAT” over serial to the computer to let the user know that the system is active and waiting for a command or action. The “z” command triggers the heartbeat on and the “x” command triggers the heartbeat off. The “BEAT” message prints after a certain number of milliseconds using the counter for millisecond ticks in the ISR.

The status command, “s”, sends the current status of the turret to the serial monitor. The status includes the current positions of the roll and tilt servos, the current speed of the roll and tilt servos, and whether the heartbeat is active. If the turret moves from one point to another, the angle values of both the roll and the tilt servos update each time after the “s” command is entered. The same also goes for the speed of the servos, meaning that if the speed change command is entered, then the new speed appears.

The home command, “h”, returns the turret to a desired home position. This home command option will be useful for centering the sensor’s FOV when the UAV isn’t performing any scans or tracking, so the turret will be ready to be used from the home position when the turret needs to begin scanning for targets. The home position is preprogrammed so if it is desired to change the coordinates of the home position then the coordinates need to be changed in the code and then that code would have to be reuploaded. If any or both of the servos are stopped (i.e. servo speed set to 0 dps), and the “h” command is entered, then the servo speeds of the stopped servos are set to the default servo speed of 60 dps to move to the home position.

The program is able to effectively parse all command strings entered into the serial monitor so any number of these commands can be executed. Figure 69 shows the function that is responsible for parsing a string of serial commands. The function readSerialString() first checks to see if a number was entered and stores that value as “angle”. It then checks for a letter to use as an indication of a command and executes the corresponding action. The last two commands that can be entered are the “q” and “w” command that enable and disable scan mode, respectively. These commands are described more in the following section.

```
void readSerialString () {
  int endTiltAngle = newTiltPos;
  int endRollAngle = newRollPos;
  if (Serial.available()){
    ch = Serial.read();
    if(ch >= '0' && ch <= '9') {angle = angle * 10 + ch - '0';}
    else if(ch == 'P'| ch == 'p') {        //roll position precedes P
      if (angle > 180) {angle = 180;}
      if (angle < 0) {angle = 0;}
      if (degSpeedRoll == 0) {degSpeedRoll = defSpeedRoll;}
      newRollPos = angle;
      angle = 0;}
    else if(ch == 'T'| ch == 't') {        //tilt position precedes T
      if (angle > 180) {angle = 180;}
      if (angle < 0) {angle = 0;}
      if (degSpeedTilt == 0) {degSpeedTilt = defSpeedTilt;}
      newTiltPos = angle;
      angle = 0;}
   else if(ch == 'H'| ch == 'h') {          //home command
      if (degSpeedRoll == 0) {degSpeedRoll = defSpeedRoll;}
      if (degSpeedTilt == 0) {degSpeedTilt = defSpeedTilt;}
      newTiltPos = tiltCenter;
      newRollPos = rollCenter;
      angle = 0;}
    else if(ch == 'O'| ch == 'o') {          // Roll Servo Speed precedes O
      if (angle > NLSpeedRoll) {angle = NLSpeedRoll;}
      if (angle < 0) {angle = 0;}
      degSpeedRoll = angle;
      angle = 0;}
    else if(ch == 'R'| ch == 'r') {          // Tilt Servo Speed precedes R
      if (angle > NLSpeedTilt) {angle = NLSpeedTilt;}
      if (angle < 0) {angle = 0;}
      degSpeedTilt = angle;
      angle = 0;}
    else if(ch == 'S'| ch == 's') {   // Status Check
      Serial.print("STATUS");
      Serial.print("\n");
      Serial.print("Roll Position = ");
      Serial.print(rollServo.read());
      Serial.print("\n");
      Serial.print("Roll Speed = ");
      Serial.print(degSpeedRoll);
      Serial.print("\n");
      Serial.print("Tilt Position = ");
      Serial.print(tiltServo.read());
      Serial.print("\n");
      Serial.print("Tilt Speed = ");
      Serial.print(degSpeedTilt);
      Serial.print("\n");
      if (trigHB == 1) {
        Serial.print("Heartbeat ON");
        Serial.print("\n"); }
      if (trigHB == 0) {
        Serial.print("Heartbeat OFF");
        Serial.print("\n");}}
    else if(ch == 'Z'| ch == 'z') {   //used for heartbeat trigger on
      trigHB = 1;
      angle = 0;}
    else if (ch == 'X' | ch == 'x') {  //used for heartbeat trigger off
      trigHB = 0;
      angle = 0;}
    else if (ch == 'Q' | ch == 'q') {   //used to trigger Scan Mode on
      trigScanMode = 1;
      Serial.println("Scan Mode Enabled");
      Serial.println("");
      angle = 0;}
    else if (ch == 'W' | ch == 'w') { //used to trigger Scan Mode off
      trigScanMode = 0;
      Serial.println("Scan Mode Disabled");
      Serial.println("");
      angle = 0;}}}
```

Figure 69: readSerialString() Function for Parsing a String

6.5. Scans

The turret control program also has the ability to execute scan patterns. The term "scan" refers to moving both servos in a specific pattern that is preprogrammed, in order to demonstrate how the sensors will move in the final assembly to accomplish a certain task (e.g. when target tracking). If the "q" command is entered, the program enters scan mode and waits for a scan command. Entering the "w" command into the serial monitor causes the program to exit scan mode.

The three scans that the turret could perform include a raster scan, a spiral scan, and a clover scan. Once in scan mode, the turret waits for a serial command and corresponding parameters to be entered. Once the scan is complete, the turret returns to its starting position.

A raster scan is used primarily on stationary objects when the UAV is orbiting. Each raster scan is composed of a desired height (controlled by the tilt servo) and width (controlled by the roll servo). Once scan mode is enabled, the command ".raster,HEIGHT,WIDTH" executes a raster scan with the desired height (i.e. HEIGHT in the command example) and desired width (i.e. WIDTH in the command example). The raster scan is made up of for loops and while loops that perform the raster motion: The turret rolls one direction the number of degrees specified as WIDTH, down one degree, and then rolls back the number of degrees specified as WIDTH. This process repeats until the starting tilt position minus the current tilt position is equal to the desired height.

A spiral scan is used on moving objects because it allows the laser rangefinder, to fire multiple points in a short amount of time in a focused area on the target. A sample spiral scan command has the form ".spiral,RADIUS,DENSITY". The spiral scan command parameters are the desired radius of the spiral (i.e. RADIUS in the command example) and the density (i.e. number revolutions/ DENSITY in the command example).

A spiral scan uses parametric equations to determine the x and y coordinates of each spiral point. A loop is used to increase a variable "t" by a certain amount each execution of the loop then uses "t" in Equation (16) and Equation (17) to calculate the x and y coordinates of a point on the spiral at that value. A change in the amplitude value in equations Equation (16) and Equation (17) affects the end radius of the spiral.

$$x = Amplitude * t * \sin(t) \tag{16}$$

$$y = Amplitude * t * \cos(t) \tag{17}$$

Many tests were performed in MATLAB in order to observe how changing the different variables in the spiral equation affected the resulting spiral's radius and density. Table 7 explains the variables used in the spiral scan calculations below. The x and y coordinates corresponded to the desired roll and tilt servo angles, so the x equation was used to control the roll servo position and the y equation was used to control the tilt servo position by acting as an offset to the position the servos are at when the spiral scan begins (i.e. newRollPos in Equation (18) and newTiltPos in Equation (19)). The original spiral equations were modified to be able to change the ending radius of the spiral and the number of revolutions of the spiral. After modifications were made, we found that changing "desDensity" (in Equation (18) and Equation (19)) changes the amount of revolutions of the spiral (we used the word density to describe this number) and that changing "spiAmp" changes the resulting radius of the spiral in Equation (18) and Equation (19).

Table 7: Variables for Spiral Scan Calculations

Variable Name	Use
x	Resulting x position at a certain time in a spiral scan with a specific density and radius
spiAmp	Used to change the final radius of the spiral
desDensity	Desired spiral density (number of revolutions)
t	Time
newRollPos	Starting roll position of the clover scan

y	Resulting y position at a certain time in a spiral scan with a specific density and radius
newTiltPos	Starting tilt position of the clover scan

$$x = spiAmp \times \left(\frac{10.0}{desDensity * 2.0}\right) \times t \times \sin(t) + newRollPos \quad (18)$$

$$y = spiAmp \times \left(\frac{10.0}{desDensity * 2.0}\right) \times t \times \cos(t) + newTiltPos \quad (19)$$

The following plots, Figure 70 and Figure 71, are a result of implementing Equation (18) and Equation (19) in MATLAB, keeping “spiAmp” constant (so the end point radius doesn’t change), and only changing the “desDensity”. The “t” variable must be iterated to a final value of desDensity*2*π, for Figure 70 and Figure 71 to have a varying density and same amplitude. The text boxes on Figure 70 and Figure 71 reveal that the end y position stays constant, which proves the ability to change the density of a spiral without changing the ending radius.

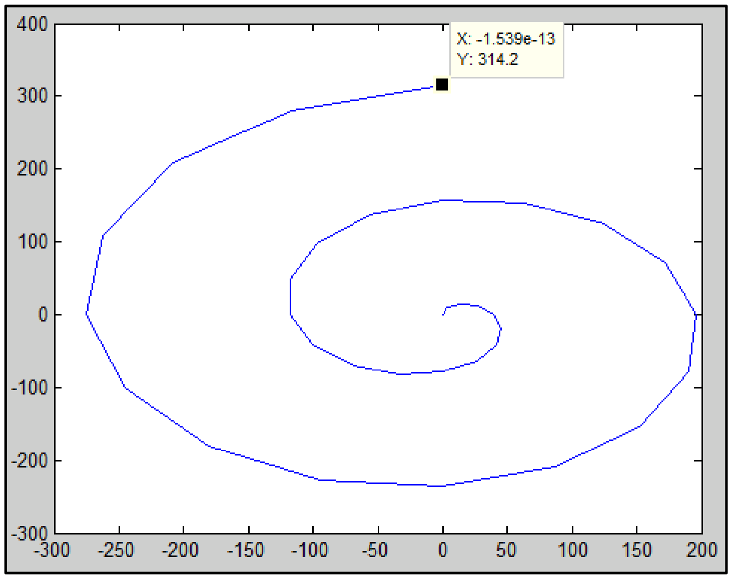


Figure 70: Spiral with Density of 2 (spiral goes around 2 times)

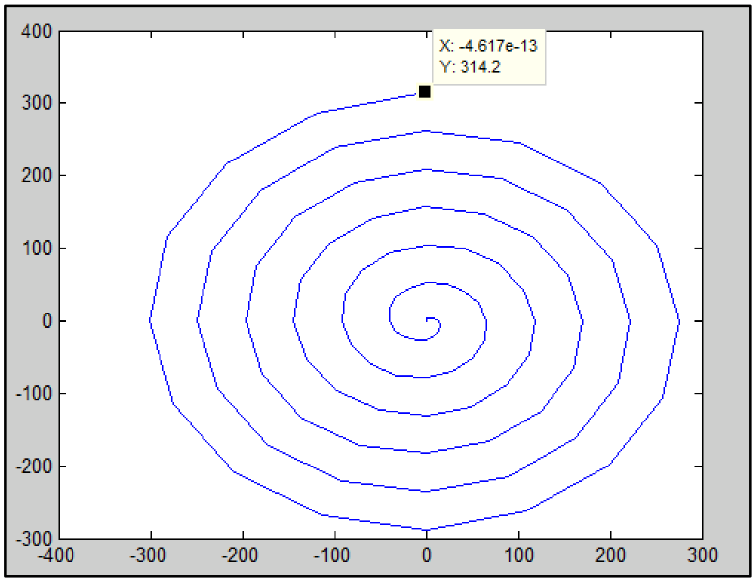


Figure 71: Spiral with Density of 6 (spiral goes around 6 times)

A clover scan is used to replicate a convoy eye-scanner (human behavior) because it allows the sensors to look around at the UAV's surroundings and then focus back at the middle to know where the UAV is flying. The clover scan command has the form ".clover,RADIUS,LEAVES", where RADIUS is the desired radius of the clover and LEAVES is the number of leaves (e.g. a Figure 8 clover scan has 2 leaves).

A clover scan, like the spiral scan, uses two parametric equations and a loop that iterates a variable "t" by a certain amount each loop iteration. The new value of "t" is used in Equation (20) and Equation (21) for the x and y coordinates of a clover:

$$x = Amplitude * \cos(t) * (1 + \cos(t)) \qquad (20)$$

$$y = Amplitude * \sin(t) * (1 + \cos(t)) \qquad (21)$$

Multiple tests were performed in MATLAB in order to see how changing the variables in the clover equation affected the resulting clover's number of leaves and number of revolutions. Table 8 explains the variables used in the clover scan equations below. The x and y coordinate equations correspond to the desired roll and tilt servo angles, respectively, so Equation (22) and Equation (23) were used to control those servos. The following equations are the resulting equations that are used in the Arduino code to obtain the desired x and y values that produced a clover while the variable "t" increased until "t" reached (2*π)+(π/numLeaves). We found that changing the "numLeaves" value changed the number leaves the clover had and that changing the "desClovRadius" changed the resulting radius of the clover (from clover center to farthest point left or clover center to farthest point right)

Table 8: Variables Used for Clover Scan

Variable Name	Use
x	Resulting x position at a certain time in a clover scan with a certain number of leaves
newRollPos	Starting roll position of the clover scan
desClovRadius	Desired clover radius
t	Time
numLeaves	Desired number of leaves in clover scan
y	Resulting y position at a certain time in a clover scan with a certain number of leaves
newTiltPos	Starting tilt position of the clover scan

$$x = newRollPos + \left(\frac{desClovRadius}{2}\right) \times \cos(t) \times (1 + \cos(numLeaves \times t)) \tag{22}$$

$$y = newTiltPos + \left(\frac{desClovRadius}{2}\right) \times \sin(t) \times (1 + \cos(numLeaves \times t)) \tag{23}$$

Since the clover scan is based off of a time variable “t”, increasing the number of leaves with keeping the maximum “t” value the same causes the servos’ positions to not always reach the exact middle point of the clover for leaves more than 8. This was due to the clover scan still having to be completed in that maximum “t” amount of time, and the servos had a limited maximum speed. If the maximum “t” for the loop was increased as the number of leaves increased, then the servos’ positions would reach the center of the clover scan after each leaf was completed. Since it was only desired by our supervisor to have a Figure-8 scan (a clover scan with 2 leaves) and a 4-leaf clover scan, and the servos’ positions would reach the center of the clover scan after each leaf with a maximum “t” value of (2*π)+(π/numLeaves), having the” t” variable as a modifiable parameter in the serial monitor wasn’t necessary.

Figure 72 is the result of setting the number of leaves to 2 and showing how the end radius of the clover can be changed. The red clover has half the end radius of the blue clover, but still has the same number of leaves. Figure 73 shows the result when the number of leaves was changed to 4 and shows how the end radius of the clover can be changed without affecting the number of leaves (i.e. the red clover has half the end radius of the blue clover, but still has 4 leaves).

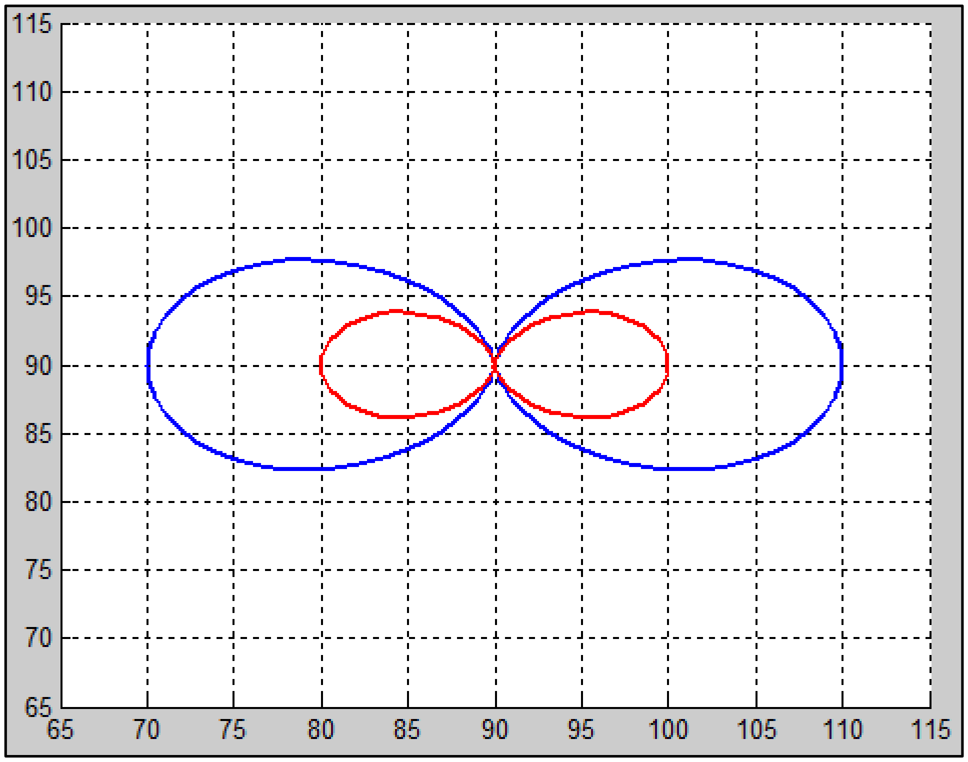


Figure 72: Clover Scans with 2 Leaves (Figure-8 Scan) Showing the Ability to Change the End Radius of the Clover

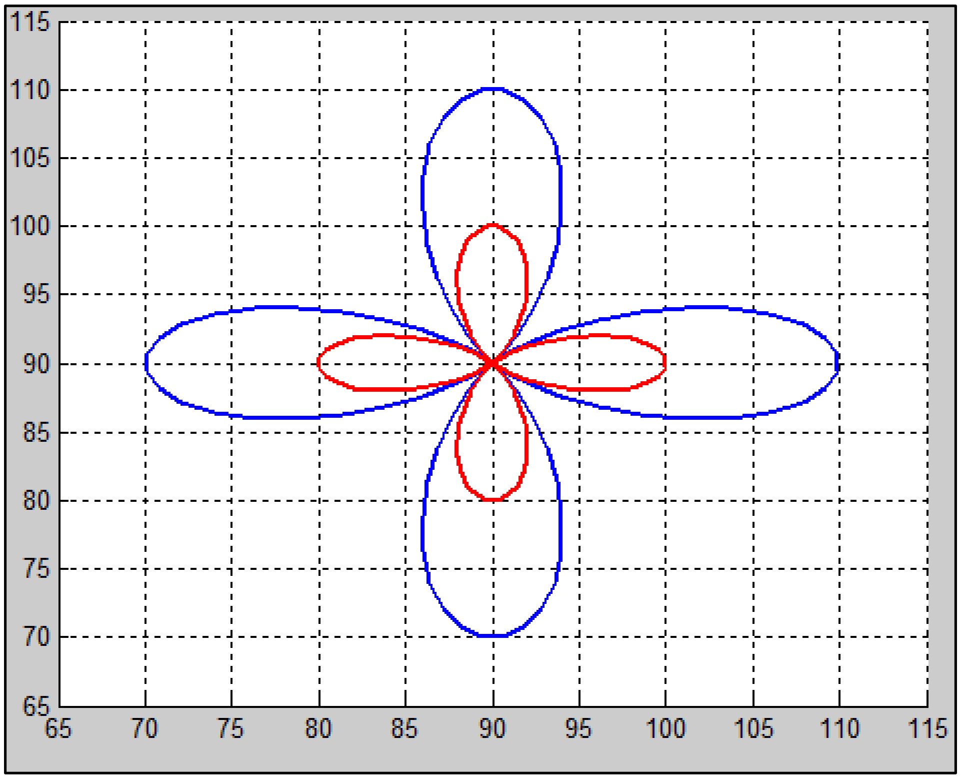


Figure 73: Clover Scans with 4 Leaves Showing the Ability to Change the End Radius of the Clover

The time at which every point occurred during each scan needed to be easily calculable because the point times are scan data that will be useful in target tracking and point cloud creation. For example, if there was a unique occurrence in the thermal camera's FOV during a raster scan (such as a dark red spot), the program will be able to make note at which time that red spot was observed. The other sensors will then be able to orient themselves to perform further scans on that target of interest.

Obtaining the time for the raster scan points involved drawing out several raster scans on a sheet of paper by hand that were a certain width and a certain height. The heights' and widths' units were represented by dots, in order to symbolize each degree movement the respective servo had to move (i.e. roll servo for width and tilt servo for height), to produce the desired scan. Figure 74 shows a sample raster scan with the dots representing position and the number representing the point numbers which were used in the point-time (time at which the point occurred) calculation. The first row and first column are row 0 and column 0, respectively, which are important when counting the rows and columns for the point number calculations.

	Col0	Col1	Col2	Col3	Col4
Row0	● 0	● 1	● 2	● 3	● 4
Row1	● 9	● 8	● 7	● 6	● 5
Row2	● 10	● 11	● 12	● 13	● 14

Figure 74: Sample Raster Scan Point Numbers

Table 9 and Table 10 explain the variables used in the equations for finding the time at which a point in a raster scan occurred. Equation (24) is used if the desired point is in an even row or the first row (row 0) and Equation (26) is used if the desired point is in an odd row:

Table 9: Variables Used in Raster Scan Point Time Calculations

Variable Name	Use
pointNum	Point number found for a specific x and y coordinate of a raster scan
pointRollPos	Desired roll position to know the time at which the point occurred
startingRollPos	Starting roll position
startingTiltPos	Starting tilt position
pointTiltPosSpi	Desired tilt position to know the time at which the point occurred
desWidth	Desired width of the raster scan

$$pointNum = (pointRollPos - startingRollPos) + ((startingTiltPos - pointTiltPos) \times (desWidth + 1)) \qquad (24)$$

For example, in Equation (24) if the following variables were given the following values:

Variable	Value
pointRollPos	92
pointTiltPos	88
startingRollPos	90
startingTiltPos	90
desWidth	4 (a raster scan that is 5 points wide means the roll servo only has to move 4 times, so the “desWidth” that the user enters is 4)

Then the point number is found in Equation (25):

$$pointNum = (92 - 90) + \big((90 - 88) \times (4 + 1)\big) = 2 + (2 \times 5) = 12 \qquad (25)$$

(which corresponds to point 12 in Figure 74).

If the desired point is in an odd row the following equation is used to find its point number:

$$pointNum = (((((startingTiltPos - pointTiltPos) - \left(\frac{(startingTiltPos - pointTiltPos) + 1}{2}\right)) + 1) \times ((desWidth + 1) \times 2)) - ((pointRollPos - startingRollPos) + 1)) \qquad (26)$$

For example, in Equation (26) if the following variables were given the following values:

Variable	Value
startingRollPos	90
startingTiltPos	90
pointTiltPos	89
pointRollPos	91
desWidth	4

Then the point number is found in Equation (27):

$$pointNum = (((((90 - 89) - \left(\frac{(90 - 89) + 1}{2}\right)) + 1) \times ((4 + 1) \times 2)) - ((91 - 90) + 1)) = (((1) - (1) + 1) \times (5 \times 2)) - ((1) + 1)) = 10 - 2 = 8 \qquad (27)$$

(which corresponds to point 8 in Figure 74).

Since we know the total number of points of the raster scan (height*width), we can take the total time for the raster scan, divide it by the total number of points (found by using Equation (28)), to get the time between each point (Equation (29)) and multiply it by the point number to obtain the time at which any point in the raster scan occurred (Equation(30)).

Table 10: Variables Used to Calculate the Point Time of a Raster Scan

Variable Name	Use
totalNumPoints	The total number of points in the raster scan
desHeight	The desired height of the raster scan
timePerPoint	The amount of time in between each point of the raster scan
totalRasterTime	The total amount of time it took the raster scan to complete
timeOfPoint	The time at which the specific point in the raster scan occurred

$$totalNumPoints = (desHeight + 1) \times (desWidth + 1) \qquad (28)$$

$$timePerPoint = \frac{totalRasterTime}{totalNumPoints} \qquad (29)$$

$$timeOfPoint = timePerPoint \times pointNum \qquad (30)$$

Next an equation was found to find the point number of a point in a spiral scan. Table 11 and Table 12 explain the variables used in the equations for finding the time at which a point in a spiral scan occurred. We discovered that since the spiral is based on a time (“t”) variable, and a spiral path revolves outward from the middle, at any given time, the radius to any point in a spiral will be different. As a result, we developed Equation (31) to get the specific point number of a spiral scan at any time. The “round” function was used in the code in order to return a whole number after the calculation for the point number was completed.

Table 11: Variables Used in Spiral Scan Point Number Equations

Variable Name	Use
ptNumSpi	Point number found for a specific x and y coordinate of a spiral scan
desDen	Desired number of revolutions for the spiral
sAmp	Amplitude of the spiral. This variable changes the end radius of the spiral
RSpi	Desired roll position to know the time at which the point occurred
stRoll	Starting roll position
TSpi	Desired tilt position to know the time at which the point occurred
stTilt	Starting tilt position

$$ptNumSpi = \left(\left(\frac{2 \times desDen}{10 \times sAmp}\right) \times \sqrt{((RSpi - stRoll)^2 + (TSpi - stTilt)^2)} \times \left(\frac{16.0}{\pi}\right)\right) \quad (31)$$

In the program, a timer starts when a spiral scan begins and the timer stops when the spiral scan end. In order to get the total amount of time it took to complete the spiral scan since we know the total number of points of the spiral scan (determined by the "t" variable shown in Equation (32)), we can take the total time for the spiral scan, divide it by the total number of points (shown in Equation (33)) and multiply it by the point number to obtain the time at which any point in the spiral scan occurred (shown in Equation (34)).

Table 12: Variables Used in Spiral Scan Point Number Equations

Variable Name	Use
totalNumPointsSpi	Total number of points in the spiral scan
t	Time
timePerPointSpi	The amount of time in between each point of the spiral scan
totalSpiralTime	The total amount of time it took the spiral scan to complete
timeOfPointSpi	The time at which the specific point in the spiral scan occurred

$$totalNumPointsSpi = t \times \frac{16.0}{\pi} \quad (32)$$

$$timePerPointSpi = \frac{totalSpiralTime}{totalNumPointsSpi} \quad (33)$$

$$timeOfPointSpi = timePerPointSpi \times pointNumSpi \quad (34)$$

The point number calculation for the clover scan took more time than the previous two scans. We developed several MATLAB scripts that would allow us to test to see if our equations would work to acquire the point number in a clover scan. Figure 75 and Figure 76 show attempts at trying to get an equation that corresponds to a point number of the clover scan. We tested out equations by plotting actual point we were trying to find as a red dot and the point we calculated with a blue dot. However, since the red dot doesn't overlap the blue dot in the Figure 75 nor Figure 76, the attempted equation would not work to calculate the point number of the clover scan.

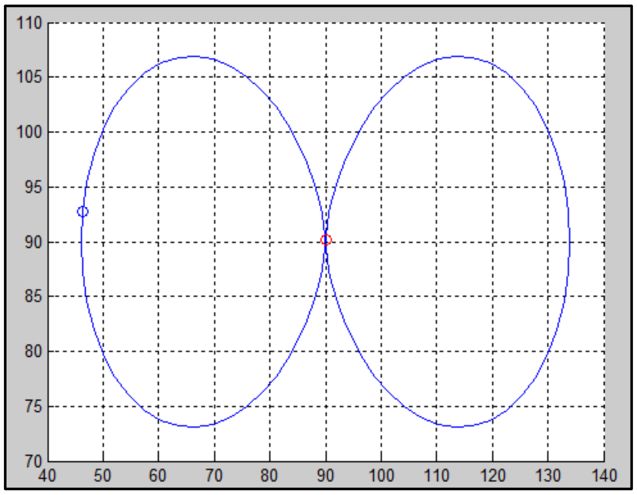


Figure 75: Clover Scan with 2 Leaves Showing that Red and Blue Dot do not Overlap

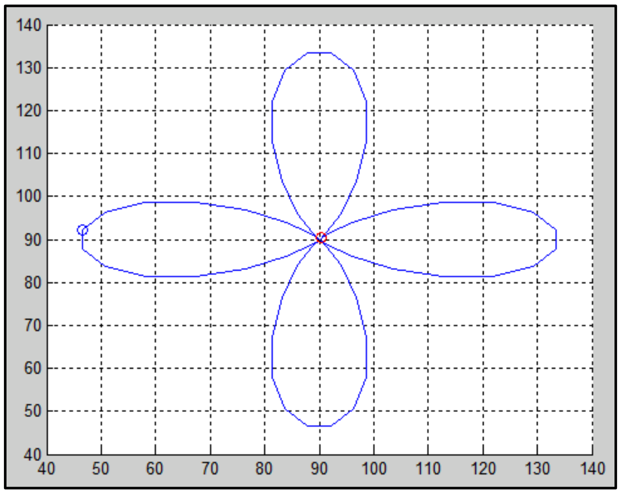


Figure 76: Clover Scan with 4 Leaves Showing that Red and Blue Dot do not Overlap

After several iterations and tests, we developed Equation (36) (which uses "tInt" from Equation (35)) to obtain the point number of any point that occurred in a clover scan of any number of leaves. In Figure 77 and Figure 78 the red dot overlaps the blue dot, proving that the equation produces the correct point number for any time in the clover scan for any number of leaves.

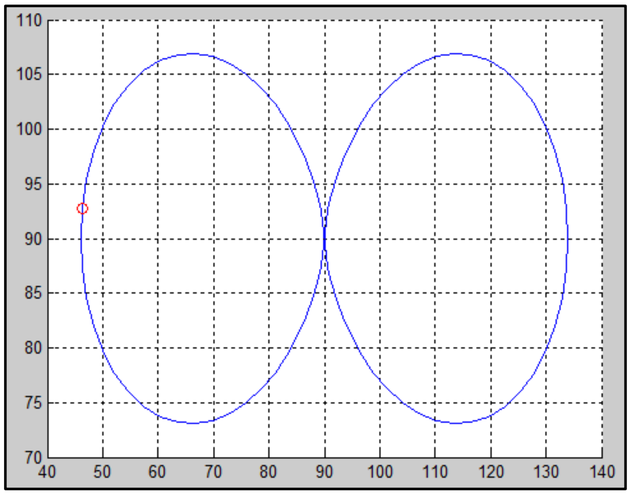


Figure 77: Clover Scan with 2 Leaves Showing that Red and Blue Dot do Overlap

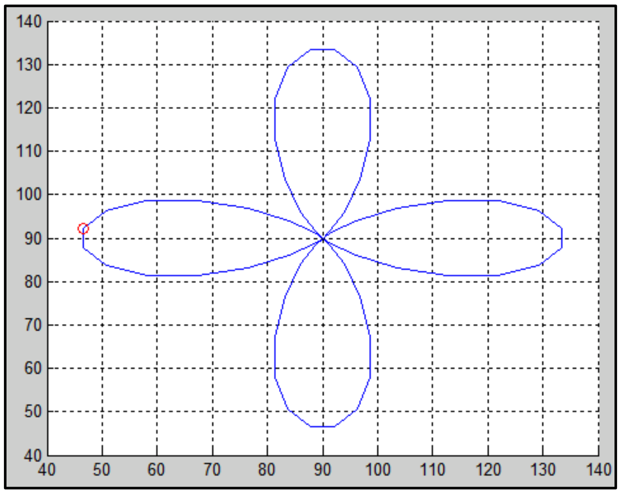


Figure 78: Clover Scan with 4 Leaves Showing that Red and Blue Dot do Overlap

Table 13 and Table 14 describe the variables used to calculate the time at which points in a clover scan occurred.

Table 13: Variables Used in Clover Scan Point Number Equations

Variable Name	Use
tInt	Angle value from positive x axis (center of clover is the origin) to desired point on clover
numLeaves	Desired number of leaves for the clover scan
pointNumClov	Point number found for a specific x and y coordinate of a clover scan
piPieces	Affects the smoothness (resolution) of the clover scan (larger piPieces corresponds to a smoother the clover scan). It is a preprogrammed number so it cannot be changed via serial monitor. It must be changed in the code then that code must be reuploaded.

$$tInt = atan2\big((pointTiltPosClov - startingTiltPos), (pointRollPosClov - startingRollPos)\big) \qquad (35)$$

If "tInt" was less than π/numLeaves, then 2π was added to "tInt" in the program.

$$pointNumClov = round\left(\left(\left(tInt \times 2.0 \times \frac{piPieces}{2.0 \times \pi}\right) - \left(\frac{piPieces}{numLeaves}\right)\right)\right) \qquad (36)$$

As in the spiral scan calculations, we first used a timer to obtain the amount of time the clover scan took to complete. We knew the total number of points of the clover scan by using Equation (37), so we could take the total time for the clover scan, divide it by the total number of points (shown in Equation (38)) and multiply by the point number (shown in Equation (39)) to obtain the time at which any point in the clover scan occurred.

Table 14: Variables Used in Clover Scan Point Number Equations

Variable Name	Use
totalNumPointsClov	Total number of points in the clover scan
numTimesAround	Represents the number of times the scan will perform the desired number of leaves with the specified radius before the scan finishes. By default, the numTimesAround variable is set to 1. The numTimesAround variable must be changed in the code and that code must be reuploaded to change the number of times the clover scan will go around.
timePerPointClov	The amount of time in between each point of the clover scan
totalCloverTime	The total amount of time it took the clover scan to complete
timeOfPointClov	The time at which the specific point in the clover scan occurred

$$totalNumPointsClov = \left(numTimesAround \times (2.0 \times \pi)\right) \times \left(\frac{piPieces}{\pi}\right) \tag{37}$$

$$timePerPointClov = \frac{totalCloverTime}{totalNumPointsClov} \tag{38}$$

$$timeOfPointClov = timePerPointClov \times pointNumClov \tag{39}$$

6.6. Encoder with Continuous Rotation Servo

In order to achieve the desired roll of the sensor housing in the final turret assembly, a continuous rotation servo with an encoder (to keep track of the amount of turns the servo performed) was required because a normal 180 degree position servo would not roll the sensor housing the desired angle (160 degrees) due to the limited space that was allowed for gearing. The encoder was an absolute encoder (its value reset to 0 after the code reran) that was programmed using the following equations so 9.5 encoder rotations corresponded to 160 degrees output.

Using Equation (7), the gear ratio , or the ratio between input motion from the servo to output motion of the mechanism, was:

$$\frac{N_2}{N_1} \times \frac{N_{internal}}{N_3} = R = \frac{input\ range}{output\ range}$$

$$\frac{60}{22} \times \frac{78}{10} = 21.27$$

We used this gear ratio in Equation (40) to calculate the number of revolutions the encoder would complete in order to obtain 160 degrees output.

$$\frac{1\ degree\ output}{21.27\ degrees\ input} \times degrees\ input = 160 \tag{40}$$

$$degrees\ input = 160 \times 21.27 = 3403\ degrees$$

$$\frac{3403}{360} = 9.453 \approx 9.5\ encoder\ turns$$

The continuous rotation servo to encoder gearing was 1:1, so that 9.5 complete 360 degree turns of the continuous rotation servo will turn the rotary encoder 9.5 times, which resulted in 160 degrees output measurement from the encoder. Table 15 shows the variables used in the encoder calculations so that 9.5 complete encoder turns will output 160 degrees. The encoder had 4 counts per tick (there were 24 discrete ticks in the encoder) and Equation (41) shows the amount of encoder counts per 360 degree turns of the encoder. Equation (42) shows the calculation to get the amount of degrees per count in order for the desired 160 degrees to correspond to 9.5 rotations of the encoder. Equation (43) shows the calculation that outputs the degree measurement based off of the encoder count value.

Table 15. Variables Used in Encoder Degree Calculations

Variable Name	Use
EncoderCountsPerRev	Number of encoder counts per revolution (360 degrees)
ScaledEncoderDegree	Degree measurement based off of the encoder count value
EncoderCountValue	The number of counts of the encoder

$$EncoderCountsPerRev = \frac{4\ counts}{tick} \times \frac{24\ ticks}{rev} = 96\ counts\ per\ rev \tag{41}$$

$$\frac{160°}{9.5\,rev * 96\,counts\,per\,rev} = 0.1754°\,per\,count \tag{42}$$

$$ScaledEncoderDegree = 0.1754°\,per\,count \times EncoderCountValue \tag{43}$$

6.7. Stabilization

IMU

The ADXL335 triaxial MEMS accelerometer and the L3G4200D triaxial MEMS gyroscope were chosen as the components of the inertial measurement unit for the stabilization system that was implemented in our final assembly due to their low cost and availability. The accelerometer chip offered the desired sensitivity of ±3g and the gyro offered sensitivity ranges up to 2000 dps.

The accelerometer breakout board was wired using a breadboard, and wires from the X, Y and Z axes outputs of the accelerometer breakout board were wired to the A0, A1, and A2 Arduino analog pins, respectively. The advantage to using analog interface for the accelerometer was that it allowed for data to be sent from the accelerometer to the Arduino for processing using the analogRead(XX) command (XX would be replaced by the analog pin name, such as A0). The accelerometer orientation values were sent to the Arduino board in mV/deg so the degree measurement for each axis of the accelerometer can be easily obtained using Equation (44) and Equation (45), and Equation (46) (variables for these equations are described in Table 16)

Table 16: Variables Used to Calculate Accelerometer Orientation

Variable Name	Use
accXval	X-axis accelerometer data after accounting for a zero value
aX	Used to represent the Arduino analog pin A0 in our code
zeroValue[0]	An average of 100 accelerometer x-axis values that accumulated in the setup of the accelerometer when the accelerometer is stationary (in order to have a known 0 degree roll and 0 degree tilt orientation).
accYval	Y-axis accelerometer data after accounting for a zero value
aY	Used to represent the Arduino analog pin A1 in our code
zeroValue[1]	An average of 100 accelerometer y-axis values that accumulated in the setup of the accelerometer when the accelerometer is stationary (in order to have a known 0 degree roll and 0 degree tilt orientation).
accZval	Z-axis accelerometer data after accounting for a zero value
aZ	Used to represent the Arduino analog pin A2 in our code
zeroValue[2]	An average of 100 accelerometer z-axis values that accumulated in the setup of the accelerometer when the accelerometer is stationary (in order to have a known 0 degree roll and 0 degree tilt orientation).
accXangle	Resultant roll angle based off of the 3 axes of the accelerometer
accYangle	Resultant tilt angle based off of the 3 axes of the accelerometer

$$accXval = analogRead(aX) - zeroValue[0] \tag{44}$$

$$accYval = analogRead(aY) - zeroValue[1] \tag{45}$$

$$accZval = analogRead(aZ) - zeroValue[2] \tag{46}$$

The individual angles were able to be combined using Equation (47) and Equation (48) in order to get the overall roll offset and tilt offset angles:

$$accXangle = \left(atan2\left(-accXval, \sqrt{accYval^2 + accZval^2}\right) + \pi\right) * RAD\_TO\_DEG \tag{47}$$

$$accYangle = \left(atan2\left(accYval, \sqrt{accXval^2 + accZval^2}\right) + \pi\right) * RAD\_TO\_DEG \tag{48}$$

The gyro uses I^{2}C communication in order to pass angular velocity values to the Arduino for processing. I^{2}C requires certain registers of the MEMS chip to be set-up in a specific way in order to get the desired values at the desired sensitivity (we used 2000 dps for this project). Table 17 describes the variables used to calculate the gyroscope orientation. In Figure 79, the integration of gyro rates (velocity data) to obtain the angles that the gyro rotated around the axes is shown.

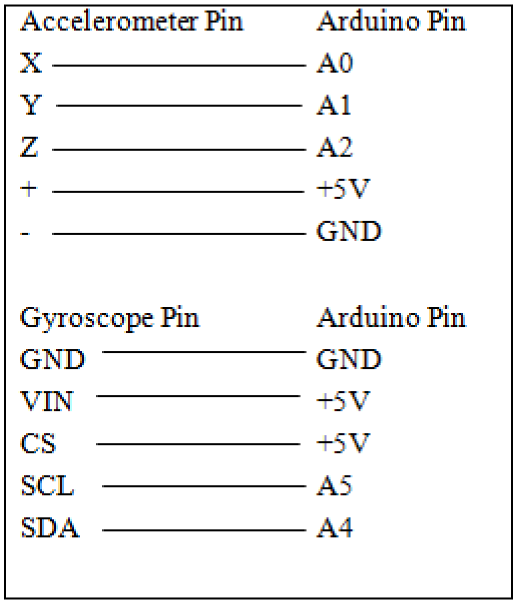
Table 17: Variables Used to Calculate Gyroscope Orientation

Variable Name	Use
millis()	Keeps track of the amount of milliseconds that passed since the code began running
timer	Used to store the millis() value at the beginning of the integration
sampleTime	Number of milliseconds that need to pass in order to update the gyro data.
gyro.read()	Read the values from the gyro, updates the gyro variables
gxrate	Rate at which the gyro is rotating about the x-axis
gyro.g.x	Raw velocity data from x-axis of gyro
gxdc_offset	Offset of the x-axis of the gyro due to the DC voltage
gxangle	The angle the gyro rotated about the x-axis
gxprev_rate	The gyro’s previous velocity about the x-axis
gyrate	Rate at which the gyro is rotating about the y-axis
gyro.g.y	Raw velocity data from y-axis of gyro
gydc_offset	Offset of the y-axis of the gyro due to the DC voltage
gyangle	The angle the gyro rotated about the y-axis
gyprev_rate	The gyro’s previous velocity about the y-axis
gzrate	Rate at which the gyro is rotating about the z-axis
gyro.g.z	Raw velocity data from z-axis of gyro
gzdc_offset	Offset of the z-axis of the gyro due to the DC voltage
gzangle	The angle the gyro rotated about the z-axis
gzprev_rate	The gyro’s previous velocity about the z-axis

```
if(millis()-timer > sampleTime){
   timer = millis();
   gyro.read();
   gxrate=(((int)gyro.g.x-gxdc_offset)/100)-6;
   gxangle += ((double)(gxprev_rate+ gxrate) * sampleTime) / 2000;
   gyrate=((int)gyro.g.y-gydc_offset)/100+1;
   gyangle += ((double)(gyprev_rate+ gyrate) * sampleTime) / 2000;
   gzrate=((int)gyro.g.z-gzdc_offset)/100;
   gzangle += ((double)(gzprev_rate+ gzrate) * sampleTime) / 2000;
   gzprev_rate = gzrate;
   gxprev_rate = gxrate;
   gyprev_rate = gyrate;
```

Figure 79: Gyro Integration Code

Wiring diagrams of the IMU to the Arduino are shown in Figure 80 and Figure 81, where the red board is the gyroscope breakout board and the purple board is the accelerometer breakout board.



Accelerometer Pin	Arduino Pin
X	A0
Y	A1
Z	A2
+	+5V
-	GND

Gyroscope Pin	Arduino Pin
GND	GND
VIN	+5V
CS	+5V
SCL	A5
SDA	A4

Figure 80: Arduino to IMU Pin Connections

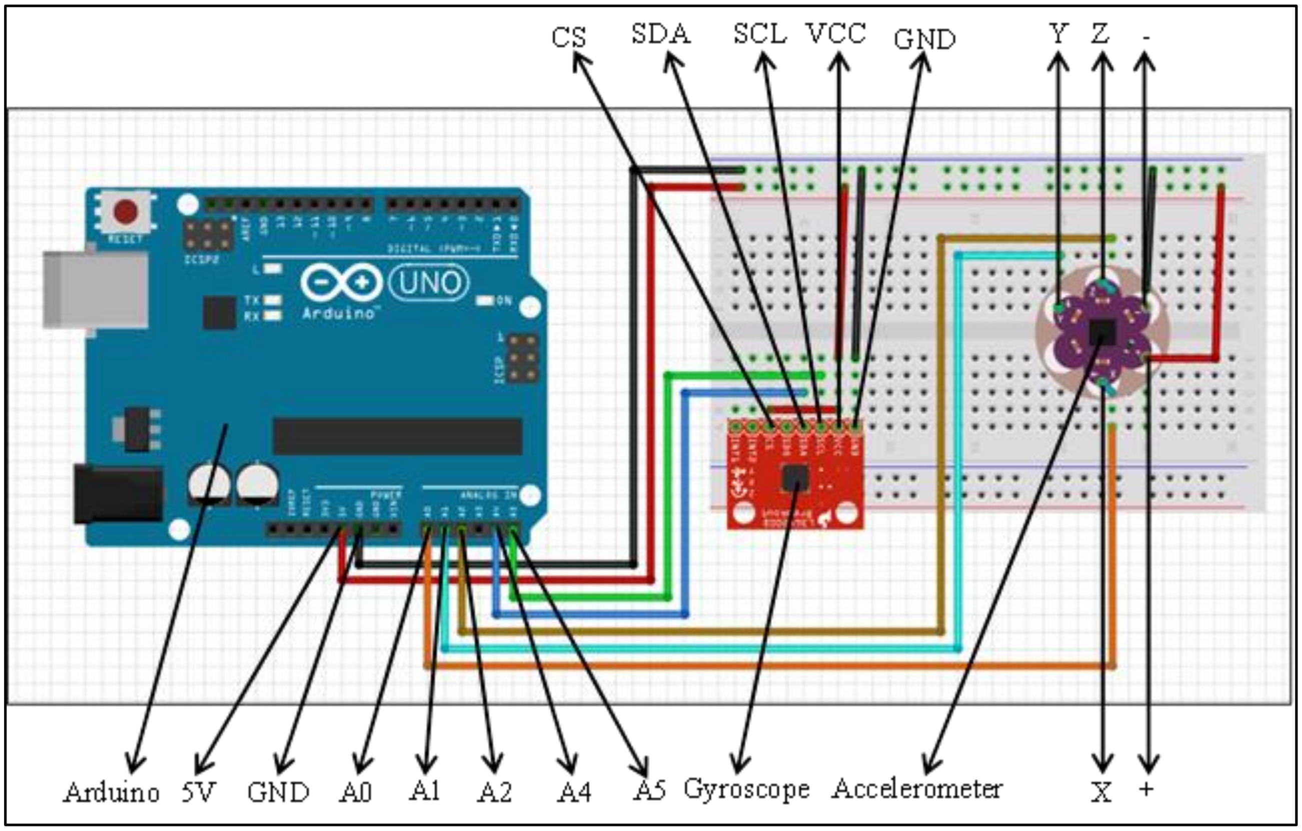


Figure 81: Arduino to IMU Wiring Diagram[3]

Sensor Fusion Algorithm

In order to combine the accelerometer readings and gyroscope readings, we implemented 2 sensor fusion algorithms, a Kalman filter and a complementary filter. The Kalman filter was implemented using a Kalman filter library that calculated the resultant angles based on weighted gyroscope and accelerometer readings. For comparison, a complementary filter was also implemented, after which it was determined that the complementary filter generated accurate values without the larger number of calculations that the Kalman filter required. Also, the Kalman filter would also be more difficult for other people who work with this software we developed for stabilization to understand and tune sensor values if different accelerometer chips or gyros were to be used.

[3] The astute reader will notice that the diagram shows an Arduino Uno, but the test turret has an Arduino Duemilanove, however, the circuit design software used for this diagram does not have a Duemilanove component. The Uno does have the same pin out as the Duemilanove.

In order to compare the computational speed of the complementary and the Kalman filter, several tests were performed on both filters. In each test, tilt and roll angles of the IMU that were generated over a 5 second period and were displayed in the Arduino's Serial Monitor, first, when a complementary filter was implemented and then when a Kalman filter was implemented. Table 18 shows the numbers of angle calculations that were printed out for a 5 second period and a 1 second period. The stationary tests involved holding the IMU still for the 5 second duration and reading out the angle measurements. The smooth motion tests involved slowly rolling and tilting the IMU during the 5 second test in order to simulate a smooth banking turn the UAV could perform while flying. The shaking tests involved rapidly shaking the IMU to simulate turbulence the UAV could encounter while flying. The results show that the complementary filter has a larger output bandwidth because it was able to produce about twice as many values as the Kalman filter in all of the corresponding tests (e.g. The complementary filter stationary test resulted in 550 angle calculations per second and the Kalman filter stationary test resulted in only 284 angle calculations per second).

Table 18. Number of Angle Calculations for Each Filter Test

Test	Number of Roll and Tilt Angle Calculations in 5 seconds	Number of Roll and Tilt Angle Calculations in 1 second
Complementary Filter Stationary Test = CST	2750	550
Complementary Filter Smooth Motion Test = CSM	2729	545.8
Complementary Filter Shaking Test = CSH	2417	483.4
Kalman Filter Stationary Test = KST	1420	284
Kalman Filter Smooth Motion Test = KSM	1292	258.4
Kalman Filter Shaking Test = KSH	1195	239

Figure 82 graphically shows the numbers of angle calculations for a 1 second period when the IMU was stationary.

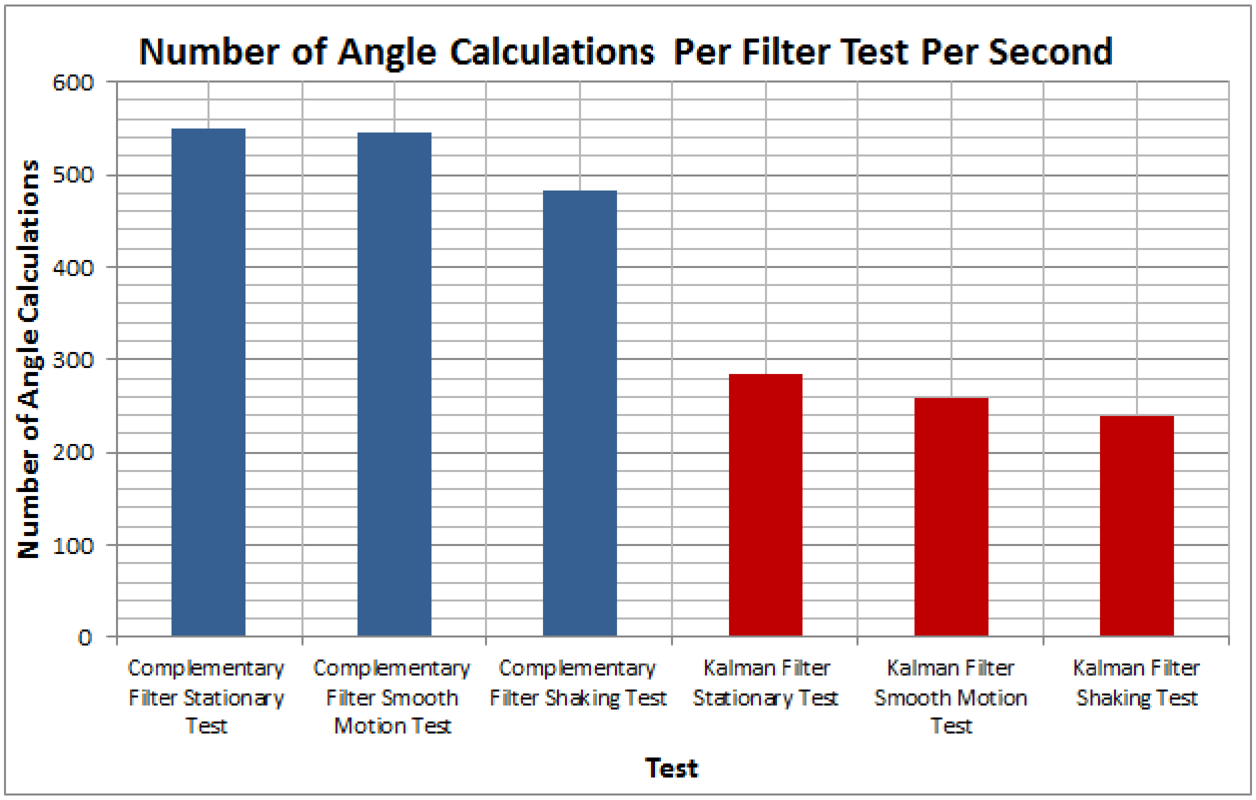


Figure 82: Number of Angle Calculations Per Filter Test in 1 Second

Figure 83 and Figure 84 show a line graph of the stationary tests of the complementary and Kalman filter tilt and roll angles, respectively. The moments of distribution for Figure 83 and Figure 84 are shown in Table 19. Figure 83 and Figure 84 show that both filters were accurate in achieving the desired 90 degrees (shown with a straight black line) for the majority of the angle calculations for the tests. In Figure 83 and Figure 84, the complementary filter results are in blue and the Kalman filter results are in red. The Kalman filter line is lower around the 90 degree mark than the complementary filter line because the Kalman filter produced significantly fewer values than the complementary filter, as shown in Figure 83 and Figure 84.

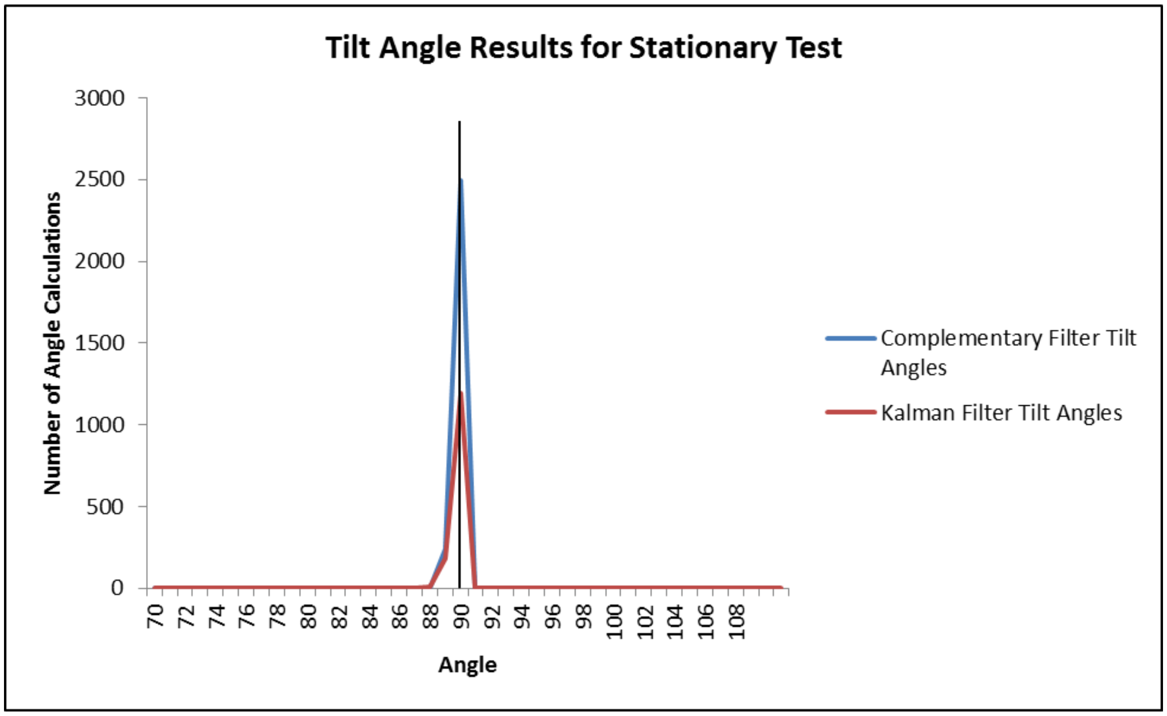


Figure 83: Tilt Angle Results for Stationary Test After 5 Seconds

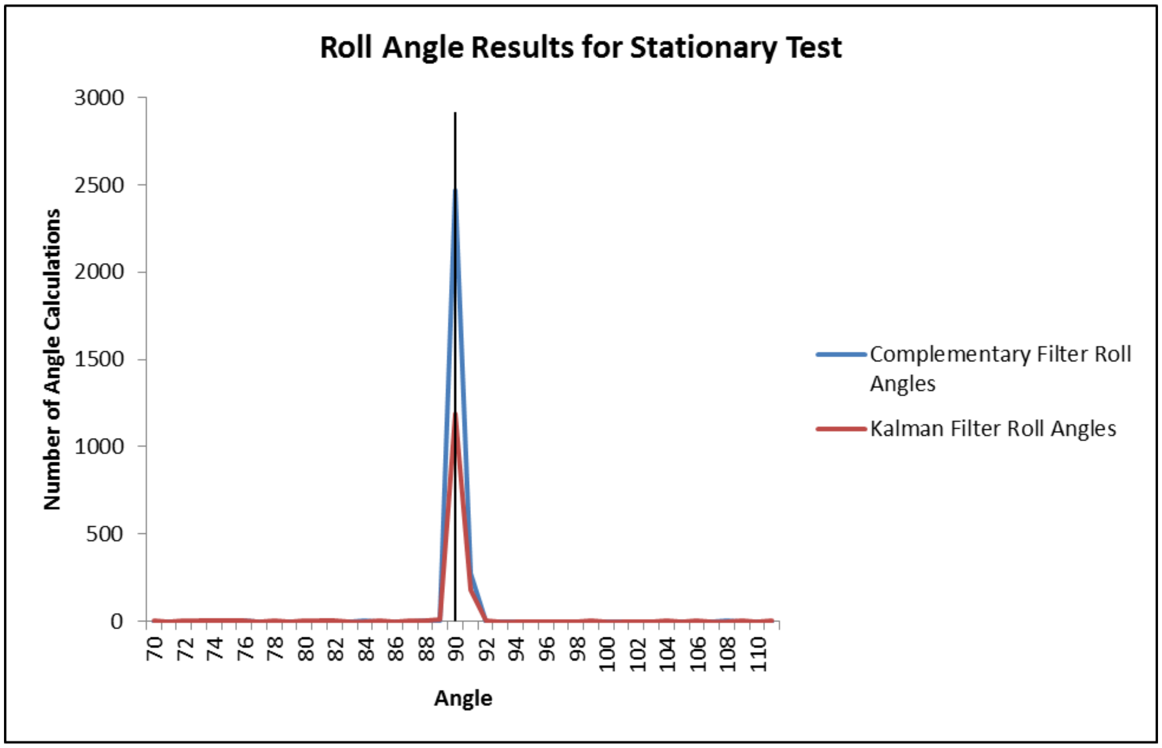


Figure 84: Roll Angle Results for Stationary Test After 5 Seconds

Figure 85 and Figure 86 show a line graph of the stationary tests of the complementary and Kalman filter tilt and roll angles, respectively, with the y axis scaled in order to see the random values that were calculated around the desired angle for each test. In Figure 85 and Figure 86, the complementary filter results are in blue and the Kalman filter results are in red. Figure 85 and Figure 86 both reveal that the Kalman filter produced more random values even though the IMU was stationary during the tests for each filter. The arrows at the top of Figure 85 and Figure 86 represent that the number of angle calculations goes beyond the limit on the y axis of Figure 85 and Figure 86.

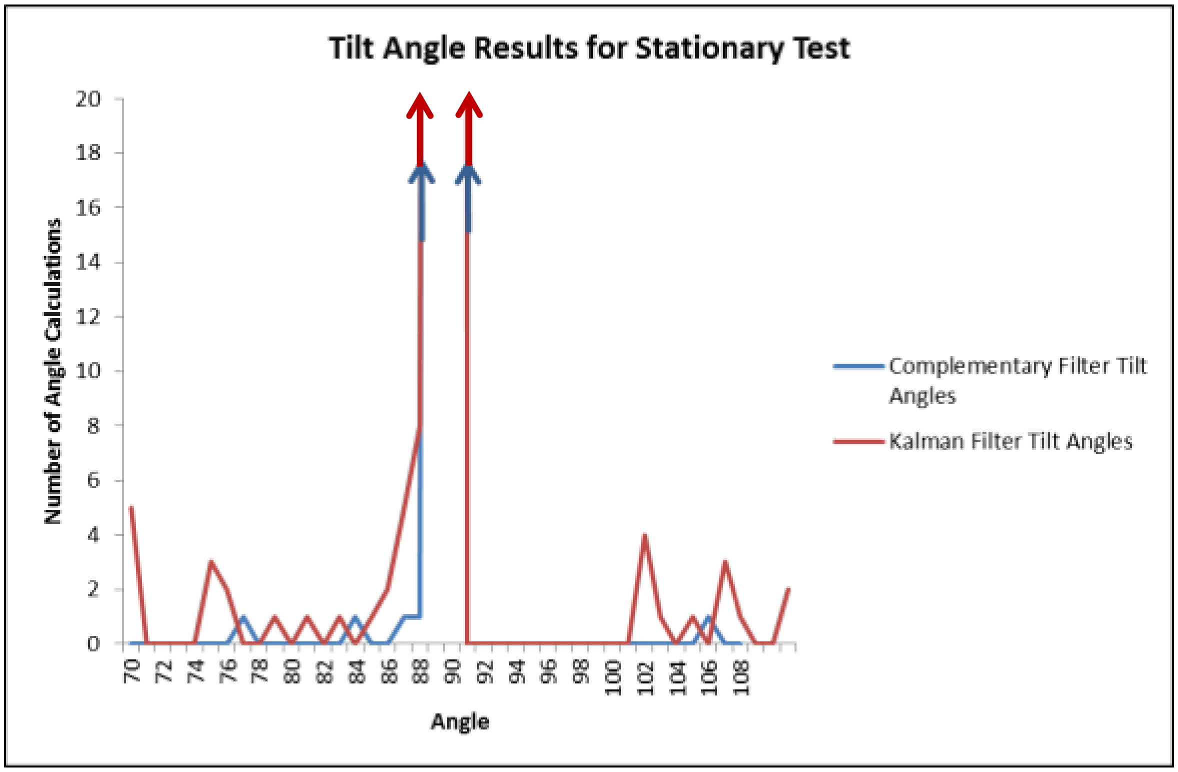


Figure 85: Tilt Angle Results for Stationary Test After 5 Seconds with Scaled Axes

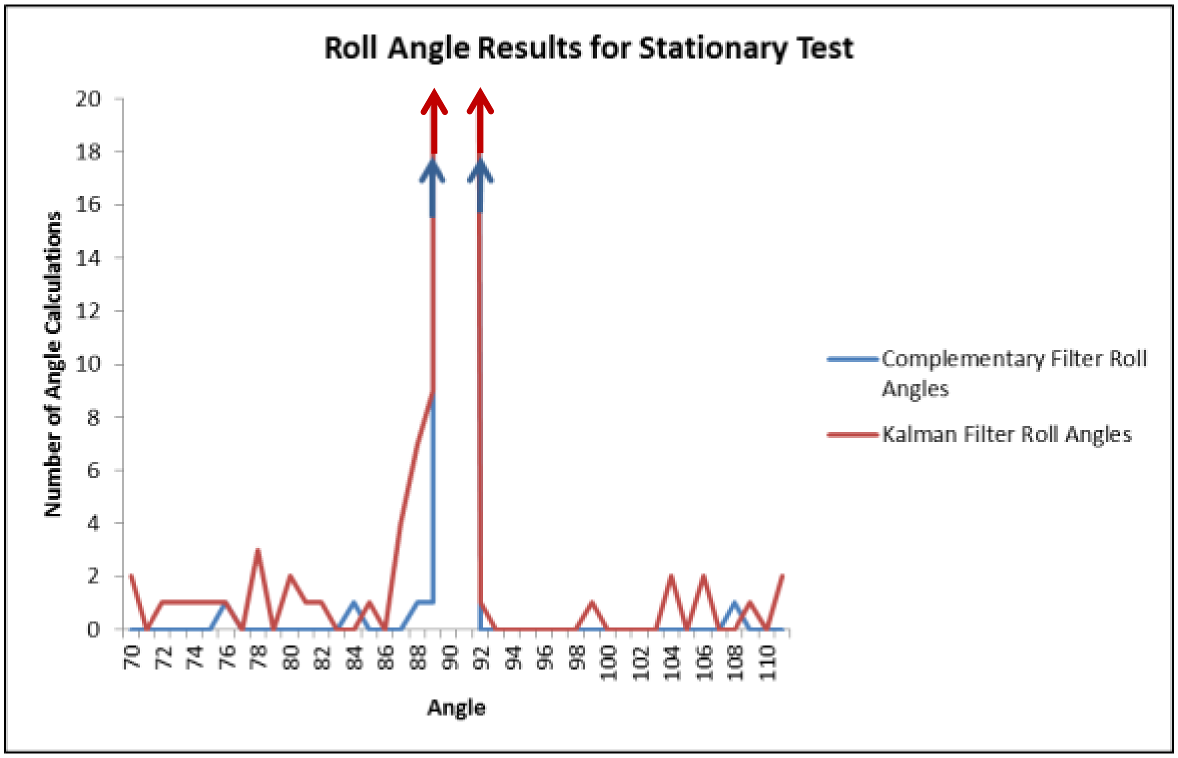


Figure 86: Roll Angle Results for Stationary Test After 5 Seconds with Scaled Axes

The previous test results prove that the complementary filter is not only faster for calculating the angle outputs of the IMU, but also is more accurate and produces fewer random values that are not equal to the desired angle, than the Kalman filter. Table 19 shows statistical calculations that were performed on the roll and tilt angle data from the complementary filter and Kalman filter stationary tests. The mean for each test was around the desired 90 degrees, which was expected. The Kalman filter results have a higher standard deviation, in both roll and tilt angle, than the complementary filter, meaning that more Kalman filter calculations were farther away from the average (which can also be seen in Figure 85 and Figure 86). The complementary filter tests have a higher skewness in both the roll and tilt angle data meaning that the results are less symmetric than the Kalman filter roll and tilt angle data. The kurtosis values of the complementary filter roll and tilt angles are higher than the Kalman filter roll and tilt angle’s kurtosis values, meaning that there is a larger rate of change in the standard deviation concentrated around the mean of the data for the complementary filter.

Table 19. Moment of the Distribution Calculations for Roll and Tilt Angles for Complementary and Kalman Filter Stationary Tests

Angle Data	Mean (1st moment)	Standard Deviation (2nd moment)	Skewness (3rd moment)	Kurtosis (4th moment)
CST Roll Angle	90.097	0.542	6.628	604.813
CST Tilt Angle	89.908	0.503	4.566	544.674
KST Roll Angle	90.050	2.056	-1.850	84.351
KST Tilt Angle	89.831	2.180	-0.052	61.246

Also, a complementary filter was chosen over a Kalman filter to combine the accelerometer and gyro readings as the complementary filter is much less computationally expensive and easier to tune to achieve the desired output. Equation (49) and Equation (50) show the angle calculations of the resultant roll and tilt angle after combining the gyro and accelerometer data using a complementary filter, respectively.

$$compAngleX = \big((gyrobiasX) \times (compAngleX + gyroNewRollAngle) + (1.00 - gyrobiasX) \times (accXangle)\big) \quad (49)$$

$$compAngleY = \big((gyrobiasY) \times (compAngleY + gyroNewTiltAngle) + (1.00 - gyrobiasY) \times (accYangle)\big) \quad (50)$$

Once calculated using the complementary filter, compAngleX and compAngleY were mapped using the Arduino map command in order to scale the compAngleX and compAngleY to values that allow the servos to perform accurate stabilization control. Also, in order to eliminate the random motion that used to occur because of the randomness of the sensors, the turret was programmed to only move if the sensor reading was above a sensitivity threshold for two readings in a row. This action filters out the one random value (that was above or below the sensitivity threshold) that tended to occur with the MEMS components at random times. Figure 87 shows the IMU stabilization code when the sensitivity threshold was not used and Figure 88 shows a plot of the IMU stabilization code when the sensitivity threshold was implemented. Figure 87 and Figure 88 were gathered in close to real time by having the Arduino send over the coordinates of the roll and tilt servo through serial into MATLAB. The green line represents the roll angle and the blue line represents the tilt angle in Figure 87 and Figure 88. After running several tests, the average roll jitter was 7 degrees (peak-to-peak) and the average tilt jitter was also 7 degrees (peak-to-peak).

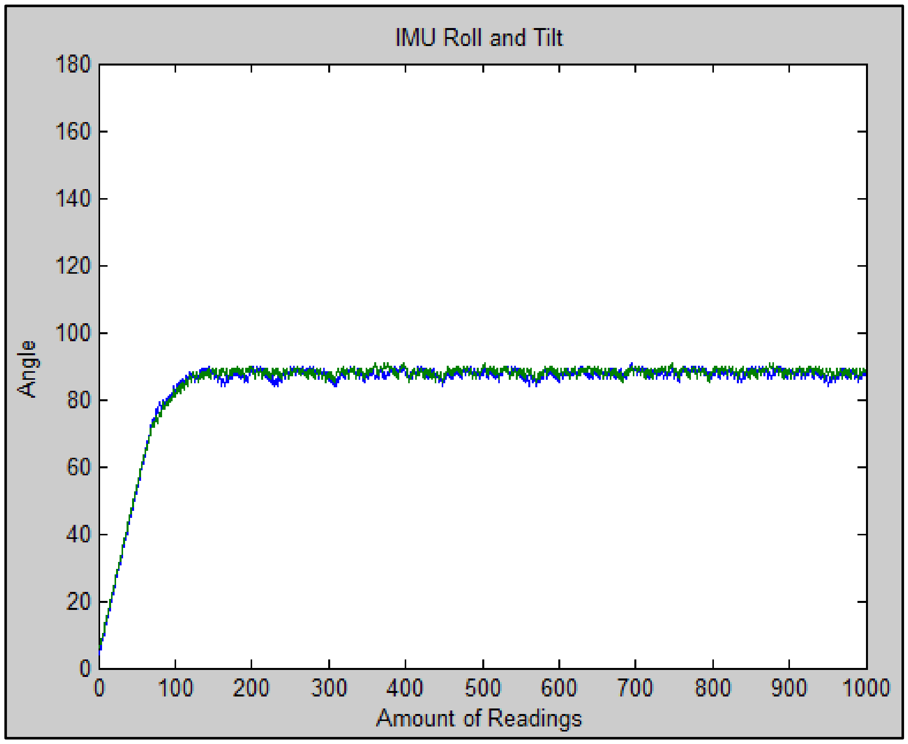


Figure 87: IMU Roll and Tilt with Jitter

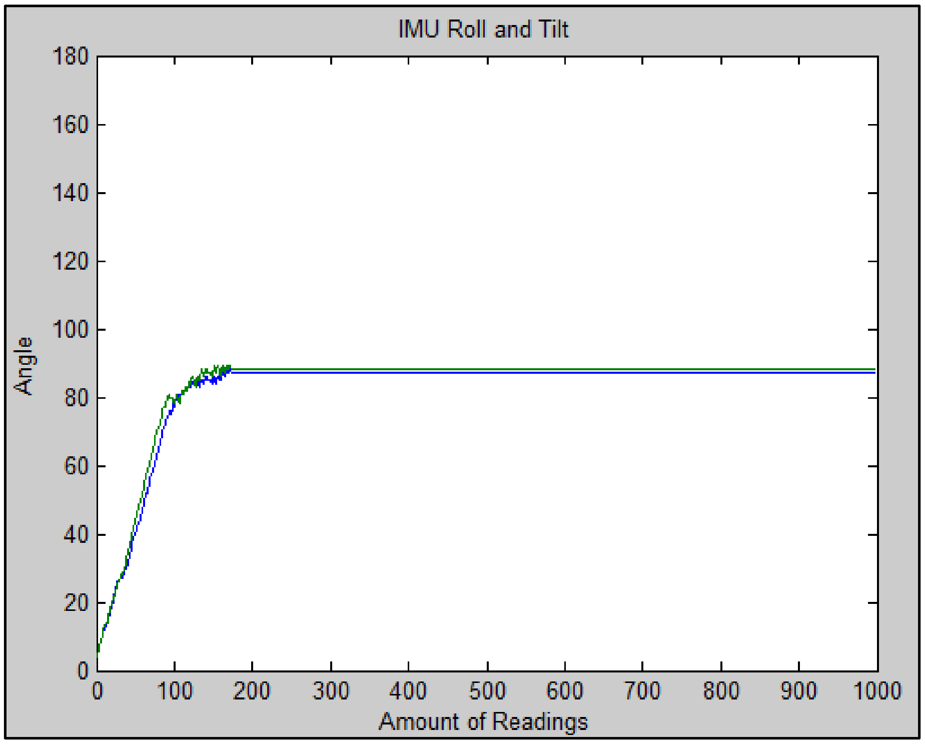


Figure 88: IMU Roll and Tilt with no Jitter

Stabilize Around One Point

After complementary filters were implemented to return an X (roll) and a Y (tilt) value based off of the combined data from the accelerometer and gyro, a function was written to stabilize around any given point. The IMU would start at a home position of 0 degrees tilt and 0 degrees roll and any roll of the IMU to the left would make turret the roll right (to cancel out the movement of the IMU), any IMU roll to the right would make turret the roll left, any IMU tilt up would make turret the tilt down, and any IMU tilt down would make turret the tilt up. A roll and a tilt of the IMU could be done simultaneously in order to simulate various turbulence effects or UAV banking turns the Goblin could encounter while flying.

6.8. Summary

The purpose of this section was to describe the results of the various software modules that were designed for this project: serial commands, scans patterns, and IMU-based stabilization. The following section describes the performance of the mechanical assembly and turret control software.

Section 7. Discussion

7.1. Mechanical Component Breakdown

Each iteration of the sensor turret was composed of six major parts: the roll mechanism, the tilt mechanism, the roll ring, the turret housing, the sensor housing, and the nose cone. This section discusses the development of each major component as well as the strengths and weaknesses of the final component.

7.1.1. Sensor Housing

The sensor housing is a key component of the sensor turret. As the design progressed, the sensor housing went through several iterations, each of which was evaluated using the same static simulation in SolidWorks to estimate load forces corresponding to the weight of each sensor, shown in Table 20.

Table 20: Simulation Load Forces for Each Sensor

Sensor	Weight (according to website)	Modeled Force ($F = m \times 9.8$)
Jenoptek DLEM SR [13]	<40 g	0.4 N
FLIR Quark [25]	28 g	0.3 N
Micro 640CSX SWIR [18]	60g + 45g*=105g	1 N

A static simulation of the 2013 sensor housing is shown in Figure 89, displacement on the left and stress on the right. While this sensor housing would not have worked for this project, as another sensor was added, the original housing was analyzed as a comparison to the iterations developed for this senior project. The maximum displacement caused by the sensor load was 3.566e-3 mm and the maximum stress was 119,838 N/m2.

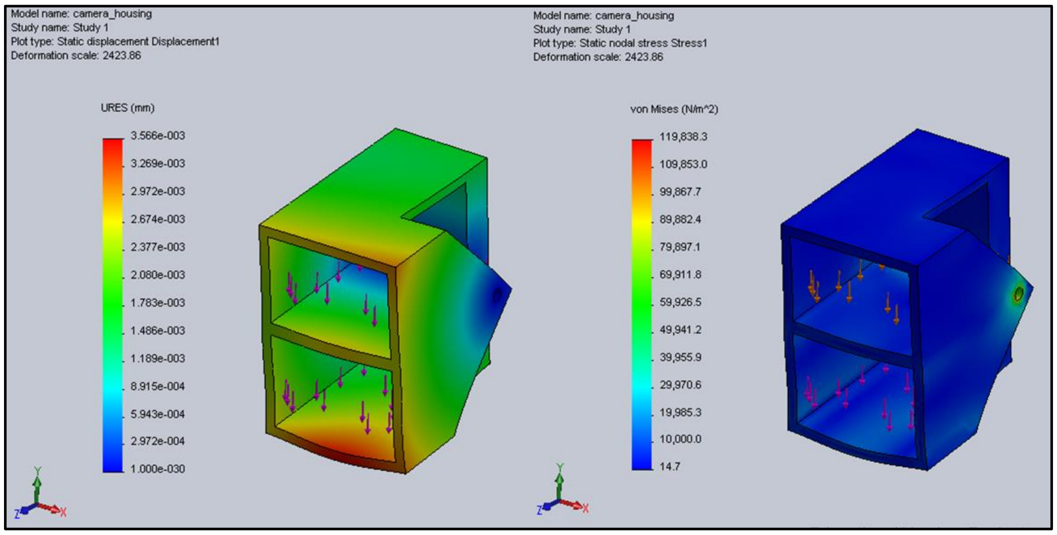


Figure 89: 2013 Sensor Housing SolidWorks Static Simulation (Displacement Left, Stress Right)

The first iteration of the sensor holder was designed to hold all three sensors side by side. Figure 90 shows a static analysis performed in SolidWorks on this first sensor housing; the left model shows the displacement of the sensor housing and the right shows the stress in the part due to the sensor load. The maximum stress experienced by the part under the loads stated above was 505435 N/m^2 and the maximum displacement caused by the loads was 8.65×10^{-3}mm. While the maximum stress and displacement for the load was acceptable, the spaces for the sensors were too small.

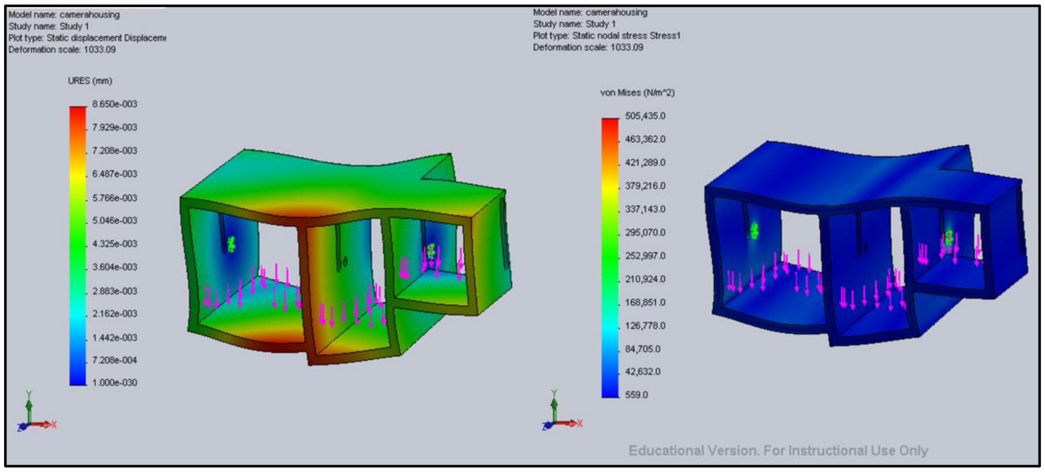


Figure 90: Iteration 1 Sensor Housing SolidWorks Static Simulation (Displacement Left, Stress Right)

The second iteration of the sensor housing is shown in Figure 91. The housing has much thinner walls than the previous housing in order to keep the same configuration of sensors while still fitting inside the small (4.875") diameter of the turret housing. Figure 91 shows the displacement (left) and the stress (right) caused by the loads in simulation. The stress and displacement were much higher in the iteration, though the sensors did fit in the housing. The maximum displacement caused by the load was 1.88e-1 mm and the maximum stress in the part was 1239130 N/m^2. One of the major problems with this iteration was the wall thickness. The narrowest wall was 0.04 inches thick, which did not make for a very solid structure to hold the sensors or to protect them from impact.

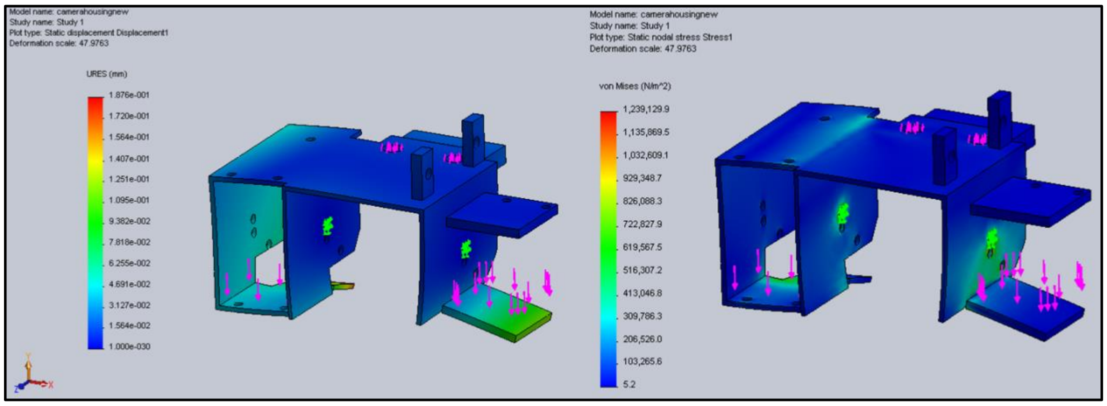


Figure 91: Iteration 2 Sensor Housing SolidWorks Static Simulation (Displacement Left, Stress Right)

In the third iteration of the sensor housing, the sensors were rearranged into a different configuration in order to allow a larger SWIR lens to be used and to accommodate thicker walls. Figure 92 shows the static simulation for the part with displacement from the load on the left and stress from the load on the right. The maximum displacement in the part was 7.74e-2 mm and the maximum stress in the part was 729887 N/m^2, which was a significant improvement from the previous sensor housing.

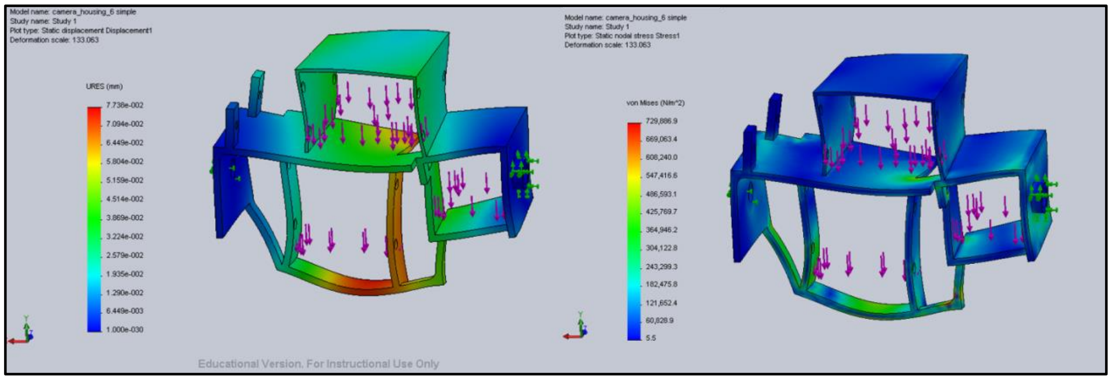


Figure 92: Iteration 3 Sensor Housing SolidWorks Static Simulation (Displacement Left, Stress Right)

In the fourth and final iteration of the sensor housing, the walls from the previous housing were increased to 0.125 inches from 0.08 inches to strengthen them and in preparation for machining the part out of aluminum. As shown in Figure 93 this dramatically reduced the displacement in the part caused by the sensor load, shown on the left, as well as the stress, shown on the right. When 3D printed out of ABS plastic, the maximum displacement was 4.55e-3 mm and the 162428 N/m^2.

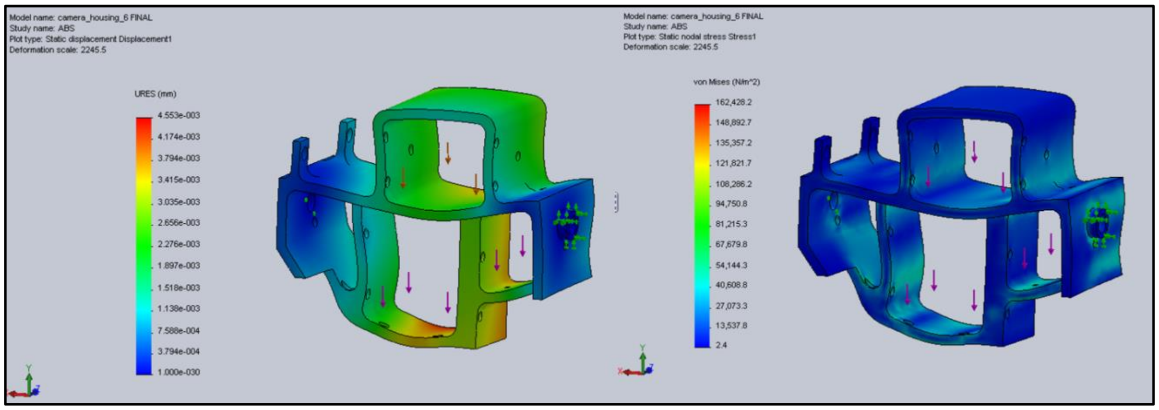


Figure 93: Final Iteration Sensor Housing (ABS) SolidWorks Static Simulation (Displacement Left, Stress Right)

Figure 94 shows the same static simulation of the sensor loads on the sensor housing machined out of aluminum. This did not significantly reduce the stress, which was 166233 N/m^2 for the aluminum sensor housing, though it did reduce the displacement which was 1.34e-4 mm. It is likely that in practice there would be more significant difference between the displacement and stress caused in the parts as the SolidWorks simulation assumes the part is solid ABS, as opposed to hollow 3D printed ABS.

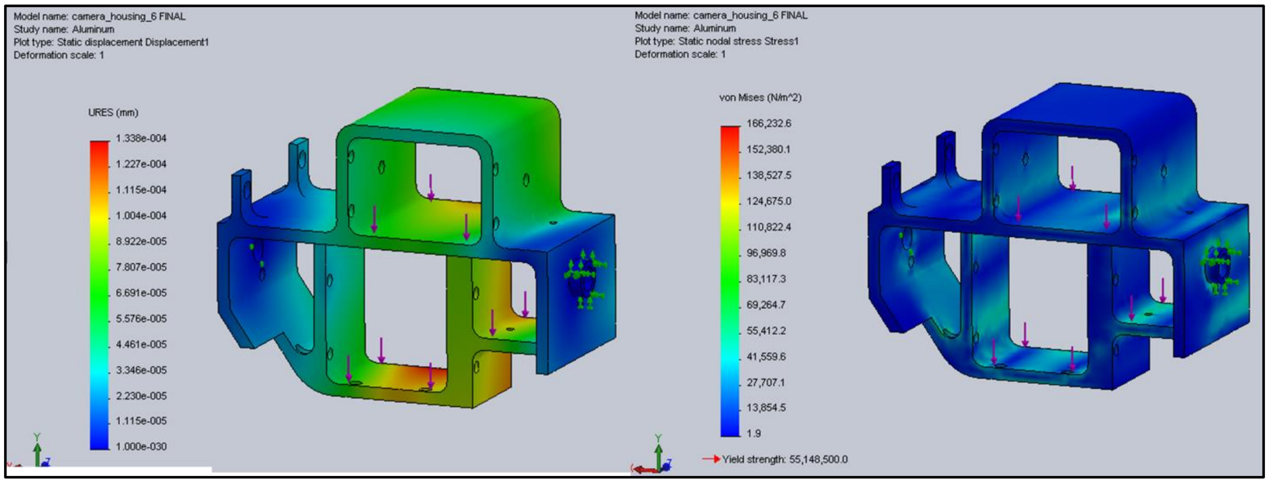


Figure 94: Final Iteration Sensor Housing (Aluminum) SolidWorks Static Simulation (Displacement Left, Stress Right)

Figure 95 shows a comparison of the maximum displacement and stress in each of the sensor housings. While the 2013 and the first iteration had relatively low stress and displacement, neither of these housings would fit all of the sensors. The second iteration had the highest maximum stress and displacement caused by the sensor loads, and as the design progressed the maximum stresses and displacements decreased. In the final iteration, the aluminum housing, the maximum stress increased slightly, but the maximum displacement decreased.

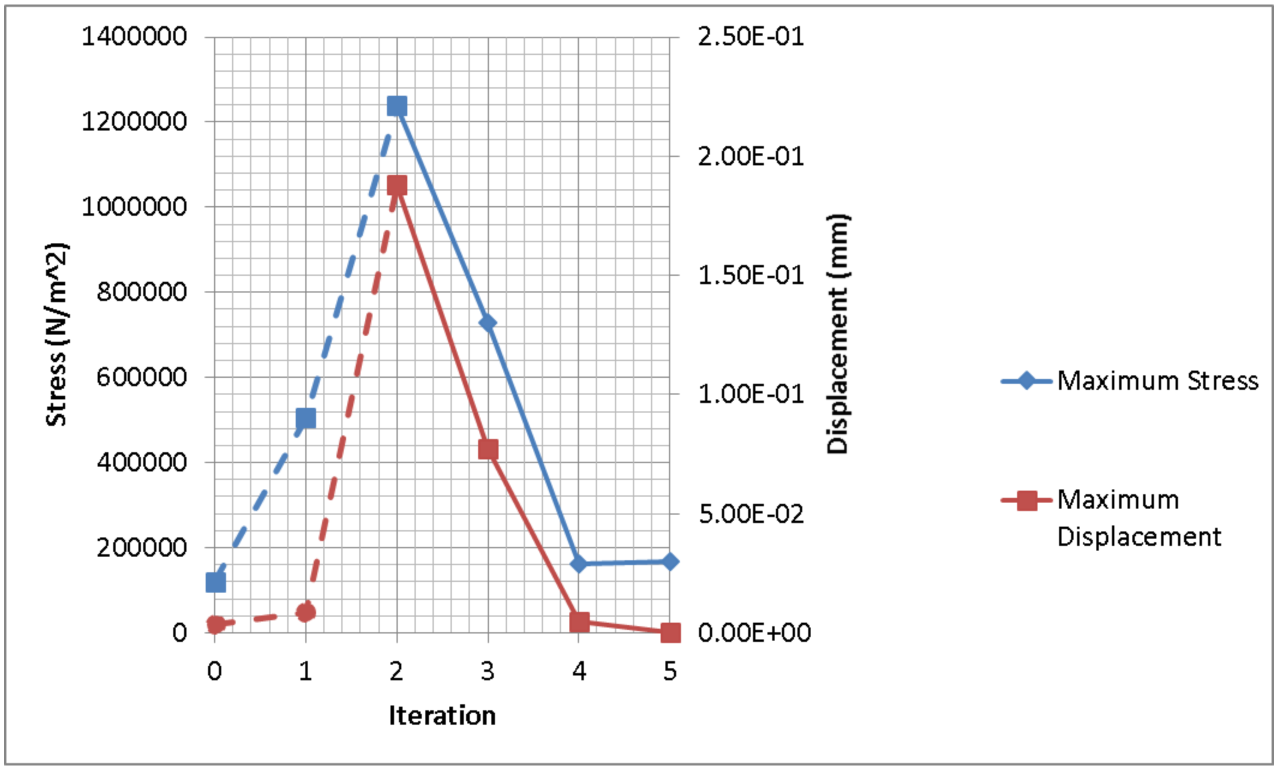


Figure 95: Maximum Stress and Displacement due to Sensor Load in Each Sensor Housing Iteration

The chart in Figure 95 shows the improvement of the sensor housing as the design developed. The first two iterations are connected with dashed lines to indicate that these sensor housings were not able to hold all three sensors. The 2013 design, while it did have the lowest maximum stress and displacement, was not acceptable for this project as it was only designed to fit two sensors. In the simulation of the 2013 sensor housing, only the loads from the rangefinder and the LWIR camera were used, as there was no space for the SWIR camera in the model.

What is surprising about the differences between maximum stress and displacement between iterations is the relatively small difference between the final sensor housing made out of ABS vs. Aluminum. However, as this is only a simulation, it does not take into account the production methods used. The 3D printing process produces a part with many layers which would be weaker than a housing produced from a solid material such as aluminum. The ABS version of the housing is also hollow, build using only two outer layers of plastic and a small amount of infill.

7.1.2. Turret Housing

The turret housing consisted of a part or several parts which held the roll ring in place as it was driven by the roll servo. The 2013 design utilized a keyway in order to allow the roll ring to be inserted and removed from the turret housing. Figure 96 and Figure 97 show the effect of a 0.5 N-m torque on the grove for the roll ring. The torque load caused a maximum displacement of 5.78e-1 mm and a stress of

3,566,884 N/m^2. The keyway caused the turret housing to deform significantly so that the roll ring was not held tightly and could be in danger of sliding out.

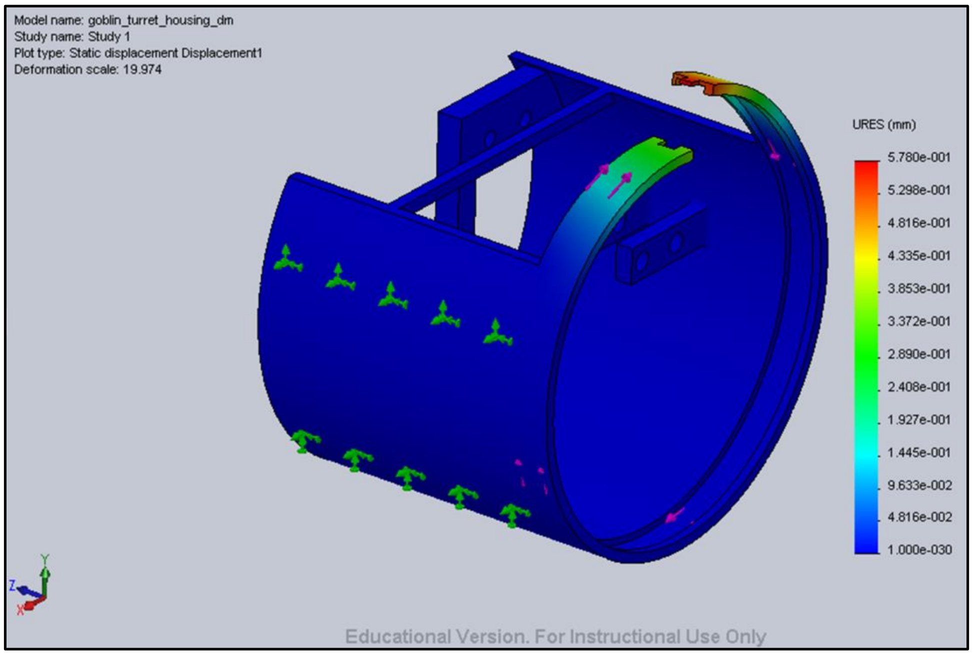


Figure 96: Displacement caused by a 0.5 Nm Torque applied at the groove

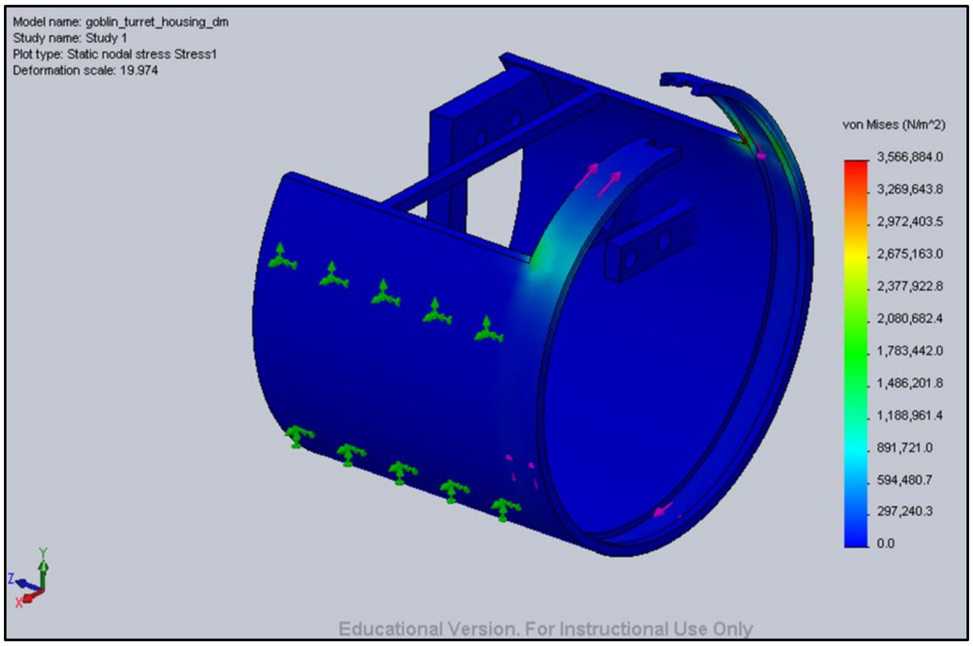


Figure 97: Stress caused by a 0.5 Nm Torque applied at the groove on the 2013 Turret Housing

Figure 98 and Figure 99 show the locking rings design for the turret housing under the effects of the same torque. The maximum displacement in this design was 1.755e-5 mm, and the maximum stress was 10,267 N/m^2, which was a significant improvement from the previous turret housing. However, it was found that the slots used to lock the rings together broke easily during assembly, so another design was necessary.

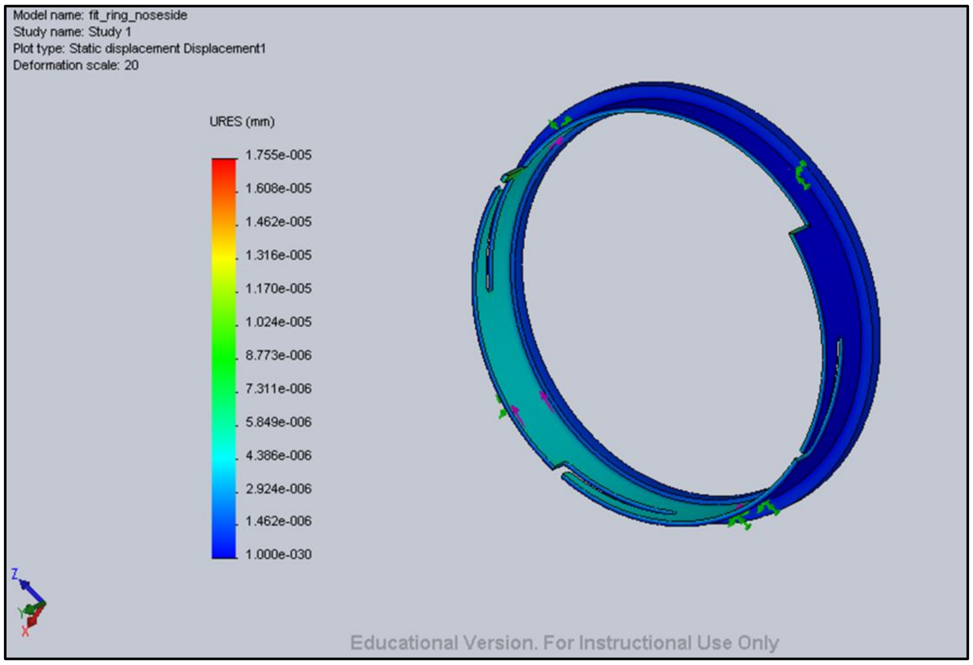


Figure 98: Displacement caused by a 0.5 Nm Torque applied at the groove on the Locking Rings Turret Housing

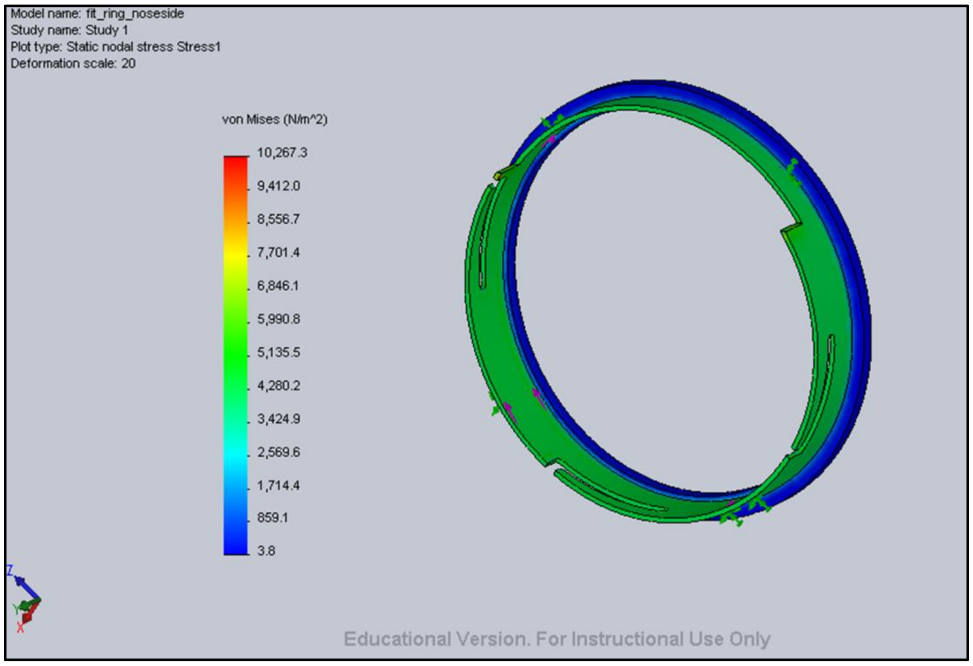


Figure 99: Stress caused by a 0.5 Nm Torque applied at the groove on the Locking Rings Turret Housing

Once it was determined that the turret housing should consist of fewer pieces, the turret assembly was redesigned to be one piece. The simulated displacement, Figure 100, and stress, Figure 71, caused in the part by the same 0.5 Nm torque are higher than those in the locking rings design, but still significantly better than those in the initial turret housing design. In testing, this one piece design was far easier to assembly than the previous designs, and simplicity was worth the tradeoff for increased maximum displacement and stress.

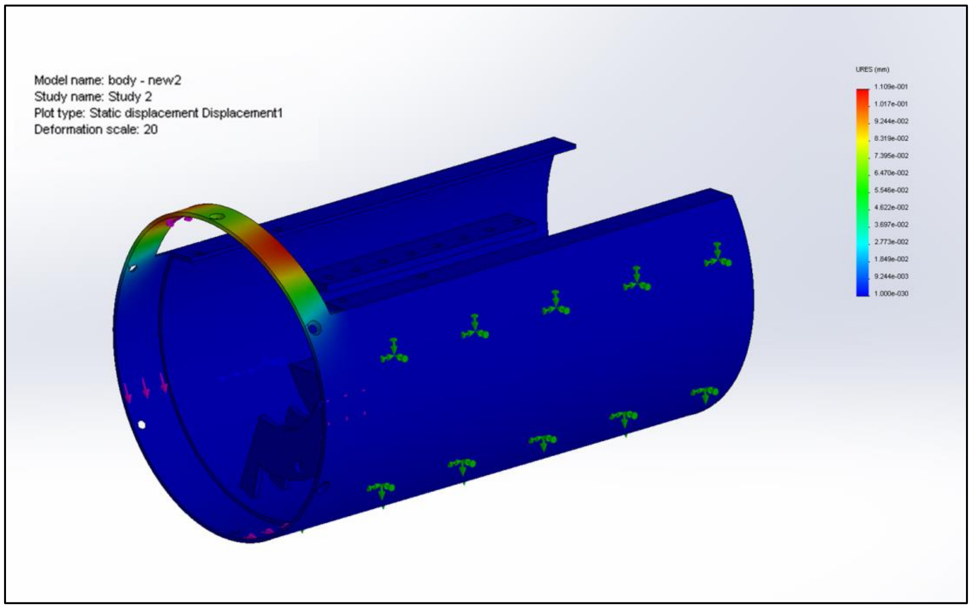


Figure 100: Displacement caused by a 0.5 Nm Torque applied at the groove on the Final Turret Housing

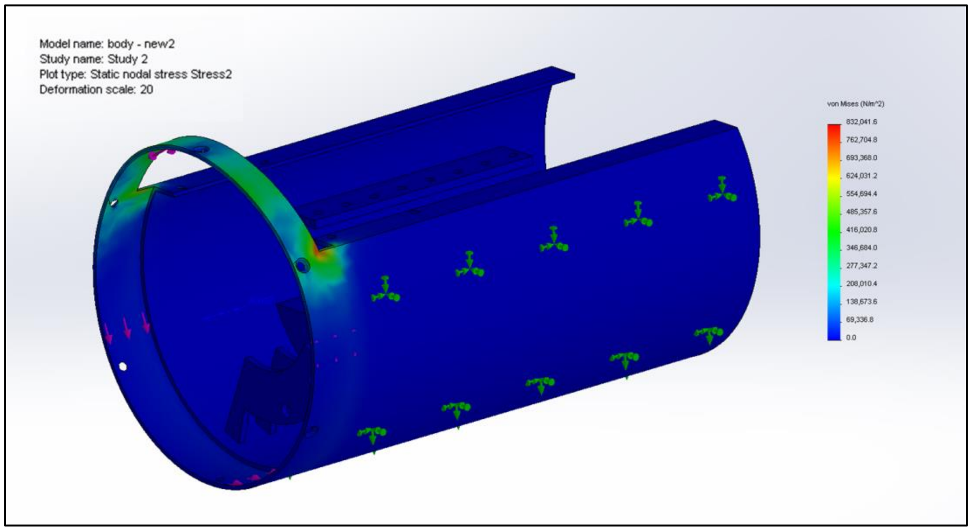


Figure 101: Stress caused by a 0.5 Nm Torque applied at the groove on the Final Turret Housing

7.1.3. Tilt Mechanism

The final tilt mechanism for the design succeeded in reducing complexity, both in the number of components and the amount of calculation and calibration required and achieving the required range of motion.

Table 21 shows a breakdown of the parts and fasteners needed for each mechanism.

Table 21: Parts Required for Each Title Mechansim

Mechanism	Parts
Complex	6 parts, 2 fasteners
Four bar	4 parts, 3 fasteners
Geared	2 parts, 2 fasteners

The large number of parts required by the complex motion mechanism made both production and assembly complicated. The fasteners in the four bar mechanism were problematic as they were constantly becoming loose and slipping or falling out. The final geared mechanism was simpler in both number of parts and fasteners, reducing the number of points of failure in the design: it consisted of two gears, one stationary and held by two screws and one specially ordered gear which fit on the end of the standard spline.

The geared mechanism was also the most simple for calculation of sensor housing angle. The complex mechanism would have required a lot of modeling and math which due to its large number of parts and possible points of failure would likely not have been very accurate. A look up table of values may have improved the accuracy but would likely have required a lot of calibration and testing. Similarly, the four bar mechanism, while easy to model in MATLAB would have required the micro controller to perform complicated trigonometric equations every time the tilt angle needed changed. The math to calculate the necessary servo values for desired output sensor housing angles was easy to implement on the micro controller as the servo angle only needed to be offset by the servo position at the 0 degree position and scaled by the gear ratio of the gear train, all of which are simple math for a micro controller.

7.1.4. Roll Mechanism

The roll mechanism for the sensor turret underwent one major change during the design process; originally driven by a position servo, the mechanism was later adapted to use a continuous turn servo. The position servo was originally chosen due to its simplicity in that it did not require a separate sensor for feedback. However, the constraints on the gears used in the system due to available gears and to maximum allowable gear diameter made the position servo system infeasible.

While it added complexity by adding another sensor for feedback, the continuous turn servo allowed the mechanism to reach the desired range of motion for the mechanism. As described in Chapter 5, the resolution of the mechanism was about 5 encoder ticks per degree, so the mechanism could be made more precise by implementing measurement of roll angles down to a fifth of a degree.

The accuracy of this mechanism was improved in software by programming the servo to always turn a few encoder ticks farther than necessary and them turning back when the mechanism was moving clockwise. This ensured that the mechanism always reached a position from the counterclockwise direction, eliminating the effects of the hysteresis on accuracy.

7.1.5. Roll Ring

The roll ring, which includes the rolling ring itself as well as the brackets to hold the sensor housing, improved in static stability over time.

In the first iteration the brackets were mounted in slots and held in position by the rack and pinion. Figure 102 shows a static simulation of the turret housing at the lowest tilt position with displacement on the left and stresses on the right. The maximum displacement caused by the 2N simulated sensor load was 0.27 mm and the maximum stress caused was 2.60e6 N/m^2. This stress and displacement was high due to the rather thin and delicate brackets and rolling bar.

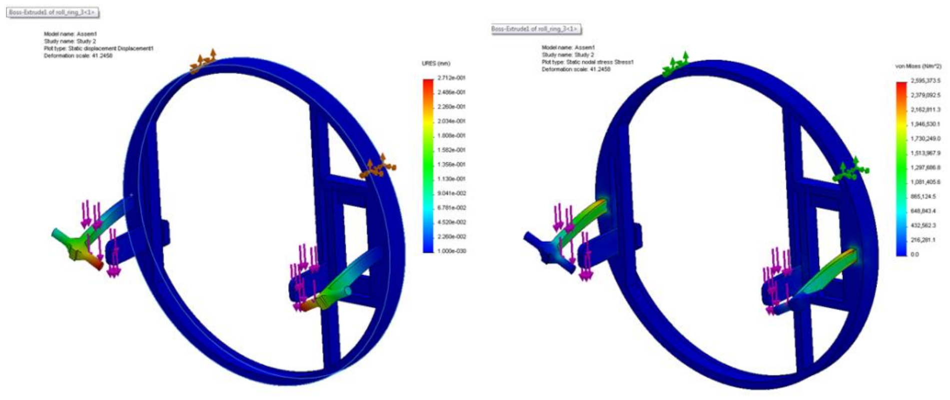


Figure 102: Roll Ring, Iteration 1, SolidWorks Static Simulation, 2N Total Force (Displacement Left, Stress Right)

In the 3rd iteration, once the four bar mechanism was implemented, the sensor housing was held by stationary brackets mounted to posts in the roll ring, the maximum displacement and stress are shown in Figure 103. The maximum displacement caused by the same 2N sensor load was reduced to stress was reduced to 4.9e-2 mm and the maximum stress was reduced to 1.2e6 N/m^2.

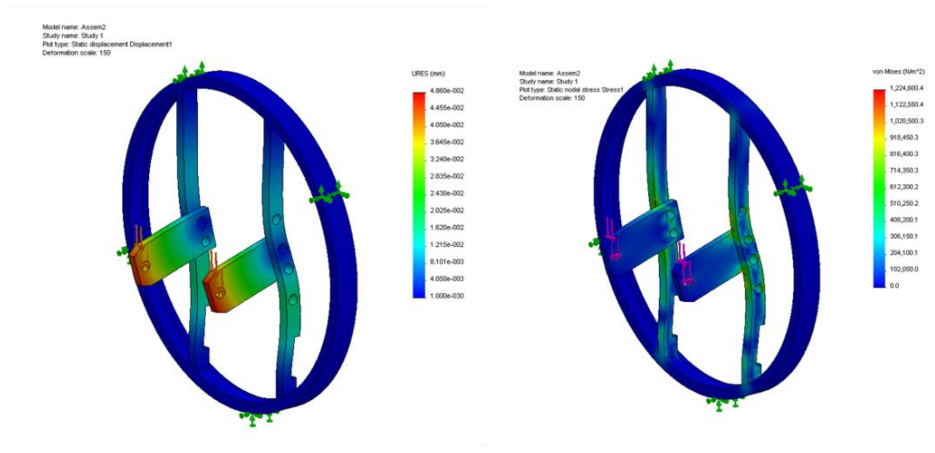


Figure 103: Roll Ring, Iteration 3, SolidWorks Static Simulation, 2N Total Force (Displacement Left, Stress Right)

In the 4th iteration the roll ring was changed so that the brackets were built into the roll ring in an effort to reduce the number of pieces in the design. However, as shown in Figure 104, the maximum displacement and stress were not significantly decreased, the maximum displacement increased to 5.9e-2 mm (a change of +22% from iteration 3) and the maximum stress decreased to 1.0e6 (a change of -15% from iteration 4).

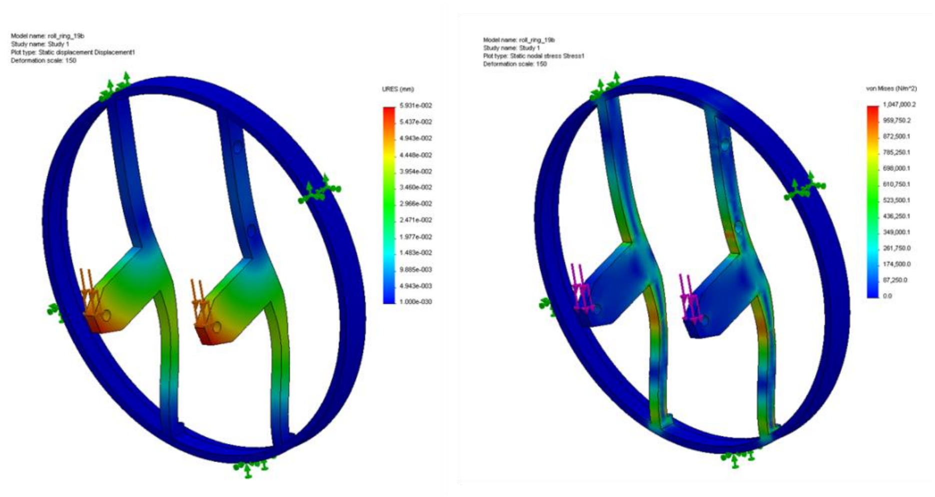


Figure 104: Roll Ring, Iteration 4, SolidWorks Static Simulation, 2N Total Force (Displacement Left, Stress Right)

In the later iterations with the geared tilt mechanism used a roll ring with brackets on either side of the sensor housing. A gear with hole in the middle was attached to either bracket and a cylinder on either side of the sensor housing was inserted into the gear. Figure 105 shows a static analysis of this mechanism, for which the maximum displacement caused by the 2N sensor load was 2.7e-3 mm and the maximum stress was 9.9e4 N/m^2, a significant improvement from previous iterations. Another benefit of this setup was that there were no additional fasteners required to mount the sensor housing.

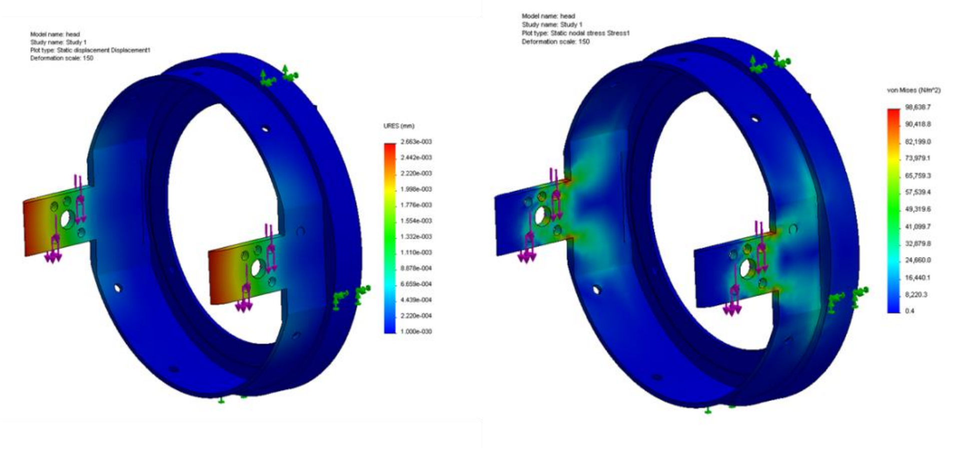


Figure 105: Roll Ring, Iteration 7, SolidWorks Static Simulation, 2N Total Force (Displacement Left, Stress Right)

Figure 106 shows the change in the maximum displacement and stress in the part over iterations due to the same 2N sensor load. Both displacement and stress decreased significantly by the end of development, though displacement did rise between iterations 3 and 4. By the end of development, the roll ring was very sturdy and could easily hold a heavier sensor load if, for example, the SWIR lens weight changed significantly.

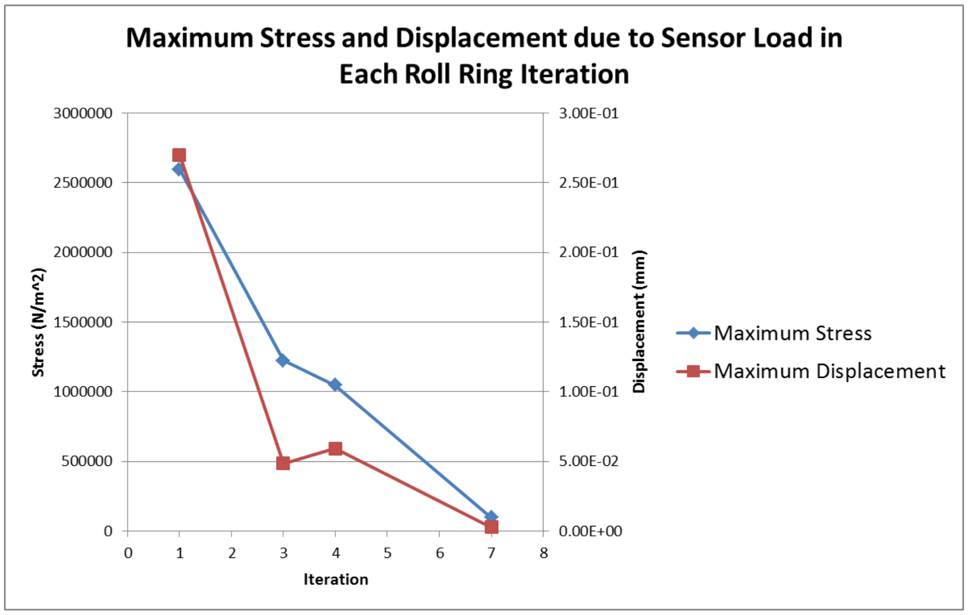


Figure 106: Maximum Stress and Displacement due to Sensor Load in Each Roll Ring Iteration

7.1.6. Nose Cone

Finally, the rotating nose cone increased the field of view of the sensors by allowing the sensors to see out of the aircraft despite roll angle. In previous designs when the nose cone was stationary the cameras would be blocked for part of the roll range. The rotating nose cone design allowed the window of the covering to continuously move with the sensors without requiring another mechanism as it was attached to the roll ring. An extension of this improvement would be to design a space efficient mechanism to open and close the window according to the tilt mechanism. There would be several challenges to this extension, including the shape of the nose cone and the lack of space for additional servos and mechanisms.

7.2. Control Software Discussion

This section describes the development of the software, the challenges faced while implementing each major module, as well as the strengths and weaknesses of the final modules.

7.2.1. Point-to-point movement

The point to point movement via serial commands did not work as desired at first. The roll servo would move and then the tilt servo would move, instead of both servos moving simultaneously to achieve fluidic motion. Interrupts were needed in order to achieve the desired smooth servo movement as well as create the specific delay in between each servo position change to achieve the desired servo speed change. The complicated process of parsing the serial commands into servo commands was addressed by creating a specific format for serial commands, as described in the Control Software Results section. The command format developed (a numerical value followed by a letter indicating the command) was tested when the code was used for integration. One partner developed the command line interface and the other used the interface in testing. While it took some time to learn the commands, the commands were generally straight forward and easy to pick up with few errors.

Several fail safes were added to the code to ensure that user error did not cause damage to the turret. If the user entered an angle that was larger than the maximum angle the servo could move to (which was 180 degrees in the test turret servos), the program limited the angle to the preprogrammed maximum angle. Also, if the user entered an angle that was smaller than the minimum angle the servo could move to (which was 0 degrees in the test turret servos), the program limited the angle to the preprogrammed minimum angle.

7.2.2. Scans

After a point to point movement command structure was developed, we also developed a structure for scan commands. This involved parsing a string of words (as opposed to just a single letter command in the serial commands point-to-point program module) and 2 parameters for each of the scans, separated by a comma. The parameters were necessary components of the scan structure, as the ability to change certain bounds of the scans (such as the radius of a clover scan and the number of leaves of a clover scan), would allow the scan module to address many situations the UAV could encounter while flying.

As with any new piece of software, there was a learning curve in order to use the scans program module because there was a certain order that the scan command needed to be entered into the Arduino Serial Monitor. In order to help the user to remember the order of parameters, print statements were used to remind the user of the correct format.

The most challenging part of developing the scan module was modifying the parametric equations of the spiral and clover scans in order to only change 2 values in each of the equations to create the desired scan (i.e. in order to get the radius and number of leaves for a clover to change). Several previous equation attempts were not successful in producing the desired scan by always changing the same variable. For example, an older clover scan equation might have produced a clover scan of 2 leaves successfully and was capable of producing clovers with different radii, but the equation would not be able to then produce a clover scan of 4 leaves. A separate equation could have been made to create a clover scan with 4 leaves, but in order to optimize the size and efficiency of the code, we wanted to find one equation that could work for a clover with any number of leaves with any radius. We also found a similar equation for the spiral scan which also proved very useful when programming the scans.

The scans program module was versatile enough so that any new set of x and y parametric equations could be added in order to make the turret move in almost any preprogrammed pattern. For example, the equations for a sine wave could be implemented in the code and then the turret would be able to perform a sine wave scan.

The scans program module also limited the servo angles to the preprogrammed maximum and minimum angles of the turret (180 degrees and 0 degrees, respectively, in the test turret) so that the servos did not try and move outside of that range and potentially damage the turret set up. For example, shown in Figure 107, if the clover scan started close to the edge of the lower limit of the roll servo (i.e. 0 degrees), then the roll servo would stop at the 0 degree positon and the tilt servo would travel straight up along the 0 degree roll servo position if the equation returned any values less than 0 degrees. The shape of the clover scan would no longer be a round leaf, but the turret would remain undamaged.

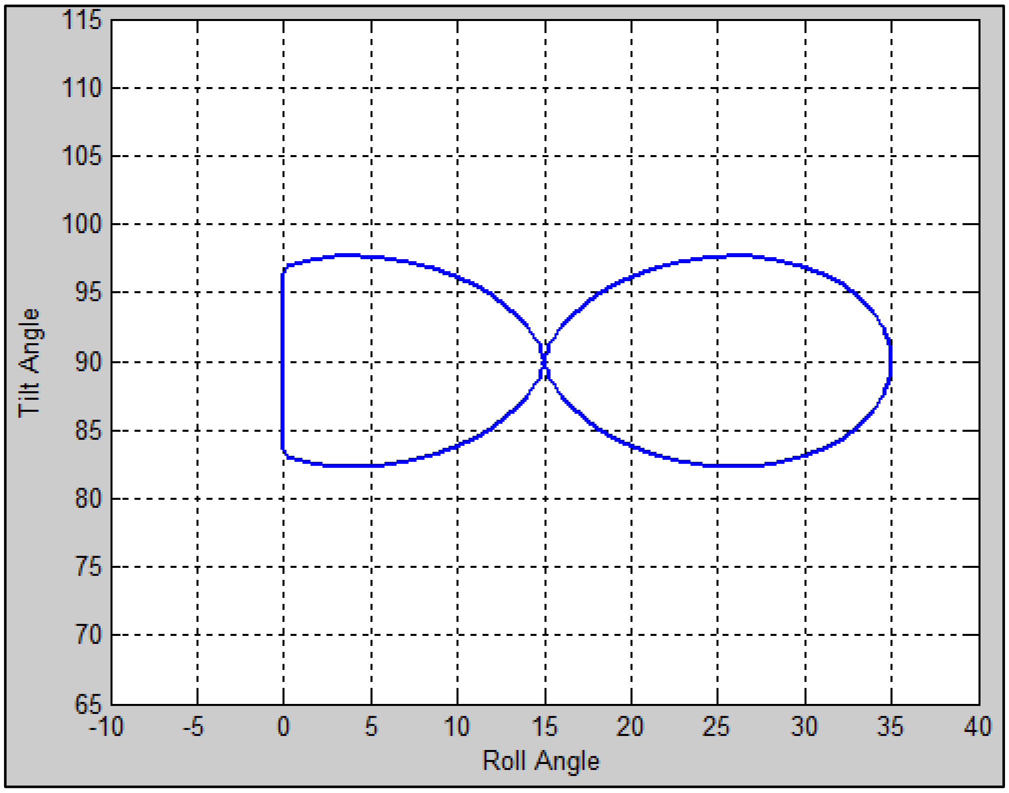


Figure 107: Clover Scan Showing an Instance when the Calculated Roll Angle is Less than the Preprogrammed Lower Limit of the Roll Servo (0 Degrees in this Case)

7.2.3. IMU

Based on the results from the tests involving a complementary and Kalman filter (described in Section 6: Software Results), a complementary filter was chosen to be used to accurately combine the accelerometer angle readings and the gyroscope readings. The complementary filter was more accurate (i.e. produced fewer random values) than the Kalman filter when the IMU was stationary and the complementary filter also produced more values than the Kalman filter, representing that the complementary filter has a larger output bandwidth.

Table 19 shows the calculations for the 4 moments of distribution which are typically used to describe data readings from a sensor in order to compare sensors. The mean (1st moment) for each of the tests were all close to the desired angle which was 90 degrees for the tests, which means that each filter was capable of producing the desired angle for the majority of the calculations.

The complementary filter tests had a smaller standard deviation (2nd moment) in both roll and tilt, than the Kalman filter, meaning that the Kalman filter produced more angle calculations farther away from the mean of the results, which it not a desirable trait for a filter that will be used in a stabilization system.

The skewness (3rd moment) of the complementary filter was larger than the skewness of the Kalman filter meaning that the Kalman filter data results were more symmetric about the mean for that set of data, than the complementary filter's data results. The large skewness for the complementary filter could be due to the data plot resulting in a larger spike of values than the Kalman filter. Since the complementary filter data spike was so narrow, a slight shift to the left or the right (due to values that weren't exactly equal to the desired value) would cause the symmetry around the mean to be greatly changed thus greatly increasing the absolute value of the skewness. A positive skewness means the peak value of data results lies to the left of the mean and a negative skewness means the peak of data results lies to the right of the mean. A skewness of 0 represents a symmetric bell-shaped curve, with the majority of the results being the mean of the data.

For all of the tests, the kurtosis (4th moment) values of the complementary filter test results were also larger than the kurtosis values of the Kalman filter results. This means that the slope (rate of change) of the standard deviation concentrated around the mean is higher for the complementary filter than the rate of change of the Kalman filter's standard deviation. It makes sense that the complementary filter kurtosis values for both the roll and the tilt were larger than the Kalman filter kurtosis values because the complementary filter had a smaller standard deviation (more data points that are closer to the mean), so there was a smaller area to notice a rate of change over, compared to the Kalman filter. Therefore, any small rate of change in the standard deviation would greatly increase the kurtosis of the complementary filter data.

7.2.4. Improvements

For improvements, the IMU program should be integrated into the scans program module so that the turret can stabilize as scans are being performed. In our development process, due to the limited time for the project, we had to stop the integration of the scans and IMU modules because the encoder with a continuous rotation servo had a higher priority, since the final assembly would not be able to move if the encoder and continuous rotation servo code was not completed. Also, if the servos had a finer resolution (all the position servos we used had a 1 degree resolution, besides the encoder and continuous rotation servo set up which had 0.20 degrees resolution per step) then more precise servo control and scans could be accomplished. The serial commands and scan commands still need to be interfaced with MATLAB in order to eventually control the turret with the target tracking software that uses the MATLAB Computer Vision Toolbox.

7.3. Integration Tests

The purpose of the integration tests was to first, to check that the sensor turret met the requirements established at the start of the project. Second, the tests ensured that the control software written on the test turret worked on the custom designed sensor turret.

The results of the first integration test showed that the accuracy and repeatability of the sensor turret was below the required 1 degree for the tilt mechanism. There was an average of 5.27 degrees of error in accuracy during this test, and the average variance between the results of different trials was 3.13 degrees. However, the repeatability decreased to 0.34 degrees when readings were only compared to readings from motions in the same direction, as described in Section 5. This pointed to a hysteresis, which we determined could be solved in software by overshooting the target tilt when moving up, and then readjusting the position so that the mechanism always reaches the target point moving down. The software solution to the hysteresis was picked as a mechanical solution would have required additional parts, which increase the weight of the system, or additional servos, which increase the power.

For the tilt mechanism, the changes to eliminate the hysteresis did not significantly improve the repeatability; the changes actually increased the average variance between readings of opposite direction movements to over 3 degrees from 2 degrees in the first iteration test. Rather than pursue another software fix for the hysteresis, we decided to look at the calibration of the mechanism.

For the first integration test, the position of the tilt servo was calculated using the known gear ratio between the servo gear and the bracket gear as well as the known offset of the tilt mechanism (i.e. the servo position which produced a 0 degree output position. The gear ratio was 23/30, and the offset was 10 degrees of servo position, so the equation used was.

$$ServoPosition = \left(\frac{23}{30}\right) \times OutputAngle + 10$$

Rather than use this ideal calculation, we performed a calibration test. We moved the tilt mechanism through the 30 to 90 degree range at increments of 10 degrees, recording the position of the laser pointer, as described previously, to calculate the resulting angle. Shown in Figure 108, we plotted the servo values and output angles using Excel and found the best fit line of the points.

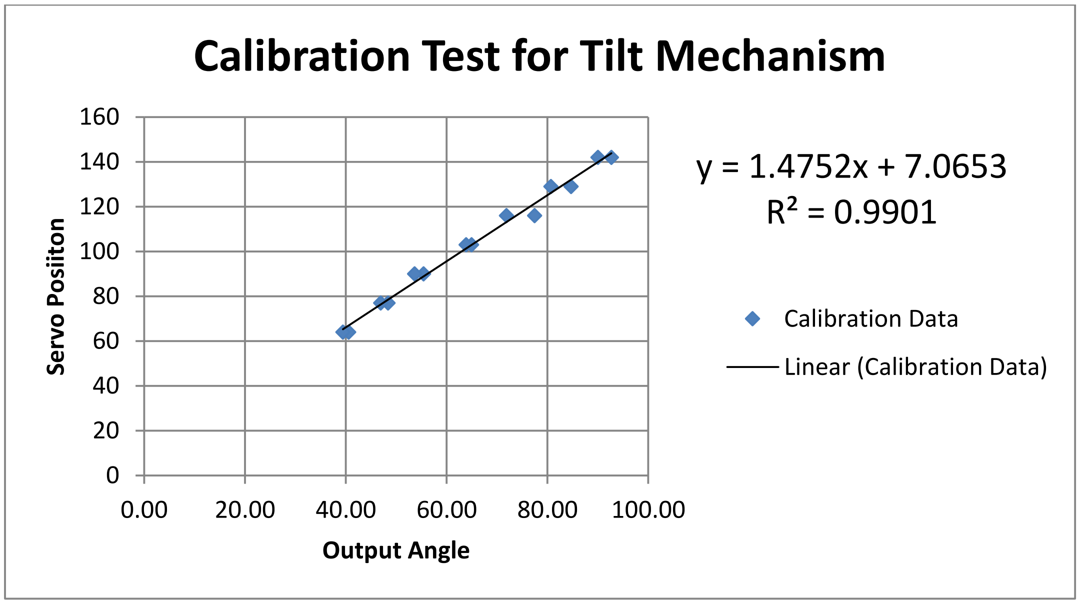


Figure 108: Test Data for Tilt Mechanism Calibration

This equation greatly improved the accuracy for the mechanism for the second integration test, from an average of 5.27 degrees of error to an average of 0.53 degrees of error. However, the repeatability for opposite direction motion only improved slightly from the hysteresis eliminating code. This method of calibration should be used for decreasing the average error, and more testing should be done with the overshooting and hysteresis eliminating code.

Preliminary testing with the roll mechanism was performed before the second integration test in order to determine what the expected error would be and whether the overshoot technique should be implemented. This testing showed that the roll mechanism had acceptable accuracy (average of 0.75 degrees of error), though the repeatability was high (an average of 1.42 degrees of variance), and the mechanism had the same hysteresis problem as the tilt mechanism (when comparing only motions from the same direction, the average variance was 0.46 degrees) with no overshoot. When overshoot was implemented, the accuracy improved to 0.73 degrees of average error and an overall variance of 0.18 degrees of average variance. These tests were performed only on increments of 10 degrees on a range of 30 to 90 degrees roll motion.

During the second integration test, the accuracy was high, but the repeatability was not as high as was expected from the preliminary tests. The average accuracy for the 5, 10, and 15 degree increment tests was 0.44 degrees, and the average overall variance was 1.10 degrees. While the overshoot did drastically improve the repeatability for 10 degree increment motions, the repeatability for other increments was not as good. This could mean that the overshoot amount should be dependent on the total change in motion. Further testing should be performed on the roll mechanism to determine the best overshoot amount to have accurate and precise point to point motion.

7.4. Summary

The purpose of this section was to present conclusions about the final mechanical design and control software for the sensor turret. Each major component was discussed and the improvements over time identified. Areas for expansion and improvement were also recognized.

Section 8. Conclusion

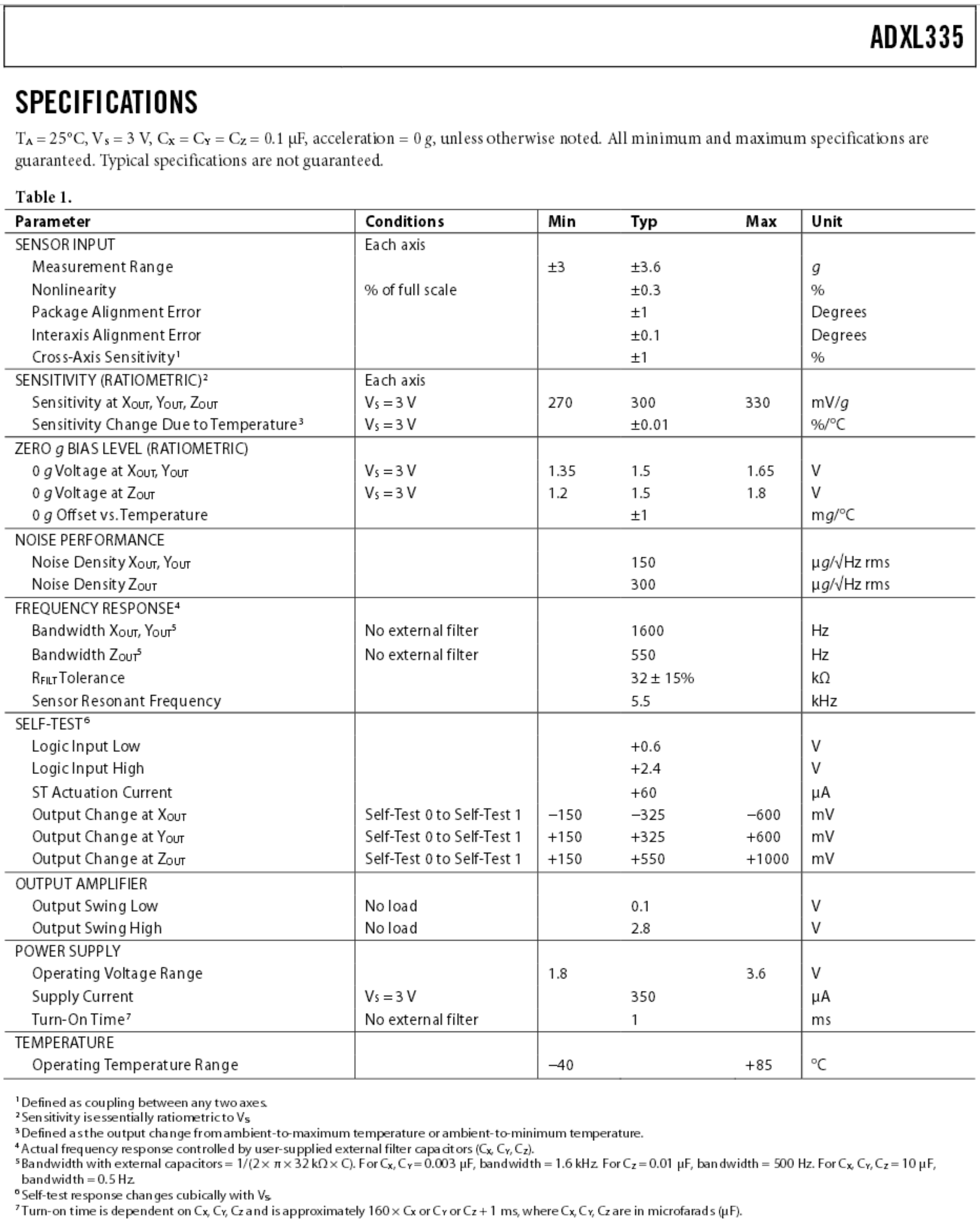
The goal of this project was to design and develop a sensor turret for the WILD Goblin UAV to house three mission-specific sensors and orient those sensors according to a serial interface. The project was divided into two parts, mechanical design and control software design, which were worked on simultaneously. The software and hardware was integrated to provide one functional robotic system.

The turret control software was capable of adjusting turret positon and speed of the servos. A home command, heartbeat command, and status command was also integrated into the code to increase the functionality of the turret assembly when it is mounted into the WILD Goblin UAV. A scan program was also created in the turret control software which allowed the user to enter a desired scan, either a raster scan, a spiral scan, or a clover scan, as well as 2 parameters for each of the scans. An IMU, which consisted of a tri-axial accelerometer and a tri-axial gyroscope, was successfully used to stabilize the turret, which will eventually be used to decouple the sensors' field of view (FOV) from the motion of the flying UAV so the sensors stay locked onto the target in a robust manner.

The sensor turret itself was developed through a series of integrations and prototyped using additive manufacturing capabilities available at MIT Lincoln Laboratory. The final turret assembly was able to use the point to point and scanning software written for the test turret and demonstrate these capabilities in several integration tests. The full range of motion of the turret exceeded the requirement for the system, rolling 160 degrees and tilting 90 degrees, giving more functional capabilities to the final product.

Appendix A. Datasheets

ADXL335 Triaxial Accelerometer Datasheet



ADXL335

SPECIFICATIONS

$T_A = 25°C$, $V_S = 3$ V, $C_X = C_Y = C_Z = 0.1$ µF, acceleration = 0 *g*, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range		±3	±3.6		*g*
Nonlinearity	% of full scale		±0.3		%
Package Alignment Error			±1		Degrees
Interaxis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[1]			±1		%
SENSITIVITY (RATIOMETRIC)[2]	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	$V_S = 3$ V	270	300	330	mV/*g*
Sensitivity Change Due to Temperature[3]	$V_S = 3$ V		±0.01		%/°C
ZERO *g* BIAS LEVEL (RATIOMETRIC)					
0 *g* Voltage at X_{OUT}, Y_{OUT}	$V_S = 3$ V	1.35	1.5	1.65	V
0 *g* Voltage at Z_{OUT}	$V_S = 3$ V	1.2	1.5	1.8	V
0 *g* Offset vs. Temperature			±1		m*g*/°C
NOISE PERFORMANCE					
Noise Density X_{OUT}, Y_{OUT}			150		µ*g*/√Hz rms
Noise Density Z_{OUT}			300		µ*g*/√Hz rms
FREQUENCY RESPONSE[4]					
Bandwidth X_{OUT}, Y_{OUT}[5]	No external filter		1600		Hz
Bandwidth Z_{OUT}[5]	No external filter		550		Hz
R_{FILT} Tolerance			32 ± 15%		kΩ
Sensor Resonant Frequency			5.5		kHz
SELF-TEST[6]					
Logic Input Low			+0.6		V
Logic Input High			+2.4		V
ST Actuation Current			+60		µA
Output Change at X_{OUT}	Self-Test 0 to Self-Test 1	−150	−325	−600	mV
Output Change at Y_{OUT}	Self-Test 0 to Self-Test 1	+150	+325	+600	mV
Output Change at Z_{OUT}	Self-Test 0 to Self-Test 1	+150	+550	+1000	mV
OUTPUT AMPLIFIER					
Output Swing Low	No load		0.1		V
Output Swing High	No load		2.8		V
POWER SUPPLY					
Operating Voltage Range		1.8		3.6	V
Supply Current	$V_S = 3$ V		350		µA
Turn-On Time[7]	No external filter		1		ms
TEMPERATURE					
Operating Temperature Range		−40		+85	°C

[1] Defined as coupling between any two axes.
[2] Sensitivity is essentially ratiometric to V_S.
[3] Defined as the output change from ambient-to-maximum temperature or ambient-to-minimum temperature.
[4] Actual frequency response controlled by user-supplied external filter capacitors (C_X, C_Y, C_Z).
[5] Bandwidth with external capacitors = 1/(2 × π × 32 kΩ × C). For C_X, C_Y = 0.003 µF, bandwidth = 1.6 kHz. For C_Z = 0.01 µF, bandwidth = 500 Hz. For C_X, C_Y, C_Z = 10 µF, bandwidth = 0.5 Hz.
[6] Self-test response changes cubically with V_S.
[7] Turn-on time is dependent on C_X, C_Y, C_Z and is approximately 160 × C_X or C_Y or C_Z + 1 ms, where C_X, C_Y, C_Z are in microfarads (µF).

2 Mechanical and electrical characteristics

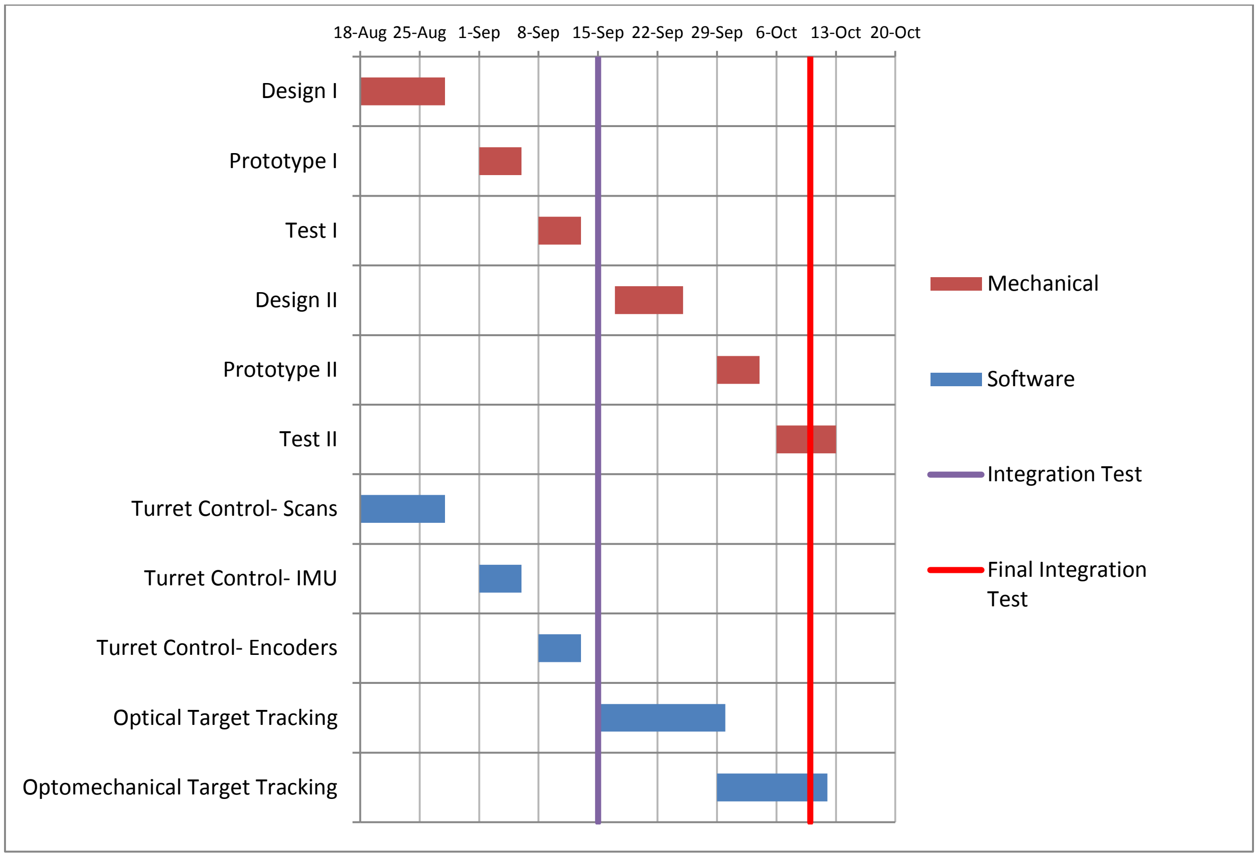
2.1 Mechanical characteristics

Table 4. Mechanical characteristics @ Vdd = 3.0 V, T = 25 °C, unless otherwise noted[1]

Symbol	Parameter	Test condition	Min.	Typ.[2]	Max.	Unit
FS	Measurement range	User-selectable		±250		dps
				±500		
				±2000		
So	Sensitivity	FS = 250 dps		8.75		mdps/digit
		FS = 500 dps		17.50		
		FS = 2000 dps		70		
SoDr	Sensitivity change vs. temperature	From -40 °C to +85 °C		±2		%
DVoff	Digital zero-rate level	FS = 250 dps		±10		dps
		FS = 500 dps		±15		
		FS = 2000 dps		±75		
OffDr	Zero-rate level change vs. temperature[3]	FS = 250 dps		±0.03		dps/°C
		FS = 2000 dps		±0.04		dps/°C
NL	Non linearity[4]	Best fit straight line		0.2		% FS
DST	Self-test output change	FS = 250 dps		130		dps
		FS = 500 dps		200		
		FS = 2000 dps		530		
Rn	Rate noise density	BW = 50 Hz		0.03		dps/ sqrt(Hz)
ODR	Digital output data rate			100/200/ 400/800		Hz
Top	Operating temperature range		-40		+85	°C

1. The product is factory calibrated at 3.0 V. The operational power supply range is specified in *Table 5*.
2. Typical specifications are not guaranteed.
3. Min/max values have been estimated based on the measurements of the current gyros in production.
4. Guaranteed by design.

Appendix B. Gantt Chart



This Gantt Chart shows how the mechanical design process and turret control software design process will be integrated. The mechanical design process will consist of at least two design iterations as shown in the chart, each consisting of a Design, Prototype, and Test phase. The Software is developed modularly, so as each module is completed the next is begun. The hardware and software will be combined during the integration tests.

Appendix C. Requirements

ID	Category	Requirement	Need Traceability
R01	Weight	The turret assembly without sensors shall weight less than 17 ounces.	Limited battery life, aerodynamics
R02	Size	The outer diameter of the turret assembly shall be less than 5.375" in diameter.	Size constraint of sonobouy used for launch.
R03	Size	The complete length of the turret assembly shall be no longer than 5".	Size constraint of sonobouy used for launch.
R04	Torque - Tilt Platform	The torque supplied to actuate the tilt platform shall be more than 10 oz-in, with a goal of at least 20 oz-in.	Based on approximate calculations of turning point, center of gravity and weight
R05	Torque - Roll Platform	The torque supplied to actuate the roll platform shall be more than 20 oz-in with a goal of at least 50 oz-in.	Based on approximate calculations of turning point, center of gravity and weight
R06	Speed - Tilt Platform	The tilt mechanism will be able to tilt at a rate of at least 15 degrees/second.	Mission parameters
R07	Speed - Roll Platform	The roll mechanism will be able to roll at a rate of at least 15 degrees/second.	Mission parameters
R08	Acceleration - Tilt Platform	The tilt mechanism will be able to accelerate at a rate of at least 15 degrees/second^2.	Mission parameters
R09	Acceleration - Roll Platform	The roll mechanism will be able to accelerate at a rate of at least 15 degrees/second^2.	Mission parameters
R10	Field of Regard - Tilt Platform	The tilt mechanism shall be able to tilt at minimum 45 degrees with a goal of 90 degrees. (Figure 2)	Identify objects of interest directly below, identify aerial obstacles
R11	Field of Regard - Roll Platform	The roll mechanism shall be able to roll a minimum of 90 degrees with a goal of 180 degrees. (Figure 3)	Maximize field of view of sensors

R12	Sensor Capacity	The turret assembly shall be able to hold the Jenoptik Laser Rangefinder, the Flir QUARK, and the Sensors Unlimited MicroSWIR.	Required sensor payload, established in Phase I
R13	Accuracy	The accuracy of the mechanism should be 1 degree (the actual position of the turret should be within 1 degree of the specified position).	Mission parameters
R14	Repeatability	The repeatability of the mechanism should be 0.1 degrees.	Mission parameters

Appendix D. Verification Plan

ID	Category	Requirement	Verification Test
R01	Weight	The turret assembly without sensors shall weigh less than 17 ounces.	Scale available in TOIL
R02	Size	The outer diameter of the turret assembly shall be less than 5.375" in diameter.	Measurement of body
R03	Size	The complete length of the turret assembly shall be no longer than 5".	Measurement of diameter
R04	Torque - Tilt Platform	The torque supplied to actuate the tilt platform shall be more than 10 oz-in, with a goal of at least 20 oz-in.	Calculate expected torque. Find stall torque using spring scale. Attach spring scale to one of the nose mounting holes.
R05	Torque - Roll Platform	The torque supplied to actuate the roll platform shall be more than 20 oz-in with a goal of at least 50 oz-in.	Calculate expected torque. Find stall torque using spring scale. Attach spring scale to one of the nose mounting holes.
R06	Speed - Tilt Platform	The tilt mechanism shall be able to tilt at a rate of at least 15 degrees/second.	Options: 1. Use a timer and do several trials. 2. Limit switches and software timer. 3. Use the already implemented gyro. Mount very close to the turning center of the tilt motion.
R07	Speed - Roll Platform	The roll mechanism shall roll at a rate of at least 15 degrees/second.	Options: 1. Use a timer and do several trials. 2. Limit switches and software timer. 3. Use the already implemented gyro. Mount very close to the turning center of the roll motion.
R08	Acceleration - Tilt Platform	The tilt mechanism shall accelerate at a rate of at least 15 degrees/second^2.	Use accelerometer already implemented. Mount accelerometer close to turning center of tilt platform. Look at the acceleration response using serial and MATLAB.

R09	Acceleration - Roll Platform	The roll mechanism shall accelerate at a rate of at least 15 degrees/second^2.	Use accelerometer already implemented. Mount acceleromter close to turning center of roll platform. Look at the acceleration response using serial and MATLAB.
R10	Field of Regard - Tilt Platform	The tilt mechanism shall tilt at minimum 45 degrees with a goal of 90 degrees. (Figure 2)	Use mounted laser pointer and measure change in position and convert to angle
R11	Field of Regard - Roll Platform	The roll mechanism shall roll a minimum of 90 degrees with a goal of 180 degrees. (Figure 3)	Use mounted laser pointer and measure change in position and convert to angle
R12	Sensor Capacity	The turret assembly shall hold the Jenoptik Laser Rangefinder, the Flir QUARK, and the Sensors Unlimited MicroSWIR.	Put sensors in camera housing. Check for appropriate fit.
R13	Accuracy	The accuracy of the mechanism should be 1 degree (the actual position of the turret should be within 1 degree of the specified position).	Use the FoR test. Calculate "ideal" reading and compare to actual data. Repeat 20 times.
R14	Repeatability	The repeatability of the mechanism should be 0.1 degrees.	Repeat FoR Test 50 times.

Appendix E. TOIL Capabilities

The TOIL has tools for both additive and subtractive manufacturing, and each of these machines presents several material options. The 3D printers in the TOIL use two materials: ABS (Acrylonitrile Butadiene Styrene) or PLA (Polylactic Acid), each of which is useful for its own purpose. ABS requires a heated chamber to reduce warping and cracking which can destroy the part while PLA is not adversely affected by temperature change. ABS is also more flexible and stronger than PLA, which is more brittle and stiff. Sheet acrylic is also available for use on the laser cutter. Acrylic is not as easily deformable as ABS. Aluminum can be machined in the TOIL and would be the most reliable material for a long-term continuous use WILD Goblin sensor pod.

The TOIL has three different types of 3D printers available for additive manufacturing. The Stratasys Dimension printer produces ABS parts and uses a support material during production. The support material is removed using a water based solution in a heated tank. The use of support material makes the Stratasys ideal for complicated pieces with overhangs, holes, or moving pieces. The printer also manufactures the piece in a heated build chamber, which is ideal for additive manufacturing with ABS. While it is the most reliable machine and has the best product, the Stratasys is not always available as it is used for many projects by Lincoln Laboratory staff. Also, the process of removing support material can take hours to days for a very complicated piece with a lot of support material.

The TOIL also has several models of the MakerBot 3D printers: the MakerBot Replicator 2X, the MakerBot Replicator 5th generation, and the MakerBot Replicator Z18. The Replicator 2X prints in ABS while the Replicator 5th generation and Replicator Z18 use PLA. The MakerBot software has an option to use the ABS and PLA to construct supports for the part which can be removed after printing to allow for manufacturing overhangs. Four of these machines are available in the TOIL, and at least one is usually available for use despite the machines being shared and occasional maintenance. While the MakerBots are good for rapid additive manufacturing, they do have a relatively high failure rate which makes their use hit-and-miss.

Finally, a Cubify 3D printer is available. The 2013 senior design project team used a Cubify printer due to proximity to their workplace. It has been observed that this machine has a much coarser product, making it less ideal for detailed parts. It is also very difficult to remove items from the bed of the printer. In addition, the Cubify does not seem to manufacture overhangs well.

The laser cutter can be used for subtractive manufacturing. The machine can cut parts out of sheet acrylic, wood, cardboard, or many other flat materials. The laser cutter is very quick and accurate, but the tolerances of the laser beam can be difficult to work with when making a part with precise dimensions, such as a gear.

The machine shop is also available for use with prescheduled appointments with the lab manager. The machine shop has a manual and CNC milling machine, a manual and CNC lathe, a drill press, and a band saw. Time and skill are the constraints with the machine shop as the space is not always available. While machining aluminum parts would be the most desired for a final WILD Goblin sensor pod, the time constraints of the project may render such parts infeasible.

Appendix F. Executive Summary

WILD Goblin Sensor Pod Design, Development, and Integration

Executive Summary

December 15, 2014

Submitted by:

Lillian Walker, lgwalker@wpi.edu, WPI box number: 1421

Daniel Zaleski, drzaleski@wpi.edu, WPI box number: 1909

Advised by:

WPI Advisor: Professor Fred Looft

MIT Lincoln Laboratory Supervisor: Bryce Remesch

Introduction

The Wing and Internal Launched Deployed (WILD) Goblin is a small unmanned aerial vehicle (UAV) project in development at MIT Lincoln laboratory that will be used to perform autonomous reconnaissance. The WILD Goblin will utilize a laser rangefinder, a short-wave infrared (SWIR) camera, and long-wave infrared (LWIR) camera in order to perform target tracking and identification.

The purpose of this capstone project was to design a sensor turret system that could house the 3 sensors the WILD Goblin will utilize in its missions (laser rangefinder, SWIR camera, and LWIR camera) as well as orient this specific set of sensors in a two degree of freedom (pitch and roll) gimbal mechanism using software we developed to control the turret.

Previous student groups have worked on parts of the WILD Goblin hardware and software. A 2013 WPI senior project team prototyped a sensor turret that held only 2 sensors and met some of the requirements of the system, showing that the project was feasible and providing a basic design for the turret mechanism. This team also developed software to move the turret through its range given coordinates through a MATLAB interface and to track targets using the MATLAB Computer Vision Toolbox.

Design of the Sensor Turret

The design of the sensor turret came about through several iterations which are described in Table 22. Each iteration consisted of six major components, each of which is described in the table. The turret housing is the piece or assembly of pieces which holds all of the other components produced for this project inside of the body of the Goblin. The sensor housing holds the three sensors, the Jenoptik Laser Rangefinder, the MicroSWIR Camera and the FLIR Quark. The Roll Ring fits into a groove of some sort and is rotated by the roll mechanism. The Roll Mechanism, located in the turret housing, consists of a servo and a gear train and outputs the roll motion, one of the two degrees of freedom of the turret. The tilt mechanism outputs the other degree of freedom: tilt motion. The nosecone is responsible for covering all parts of the sensor turret which do not need to be exposed.

Iteration	Description of Item in Iteration						Issues Found
	Sensor Housing	Roll Mechanism	Tilt Mechanism	Turret Housing	Nosecone	Roll Ring	
1	Three pockets (one for each of the three sensors).	Internal gear (70 teeth) driven by pinion gear (10 teeth) attached to a 180 degree position servo.	Complex Motion. Rack gear driven by a pinion on a micro servo linearly actuates the sensor housing vertically. As the sensor housing raises or lowers it rotates about a roll bar.	Retain 2013 senior project turret housing. Cylindrical turret housing holds all components produced for the project. Fits into the cylindrical body of the Goblin. Holds roll ring. Keyway in front of housing is used to insert roll ring.	Cut in half to expose the sensors. Anchored to the Goblin Body, does not move with roll or tilt mechanism.	0.2 inches thick. Pocket for micro servo.	● Sensor Housing did not fit the sensors. ● Roll Mechanism requires a 1:1 ratio to reach the desired range of motion (180 degrees) ● Turret Housing: Keyway causes hold on roll ring to be loose. ● Roll Ring: Servo collides with pinion gear driving the roll mechanism
2	No Change	Changed gear ratio of gear train to be as close to 1:1 as certain conditions would allow.	No Change	Used two interlocking rings which formed a grove when locked together to hold the roll ring. The rings straddle a ridge in the existing Goblin body, holding them in place.	No Change	Ring was mirrored so that the servo no longer collided with the gears of the roll mechanism.	● Sensors did not fit into the sensor housing. ● Servo did not fit into th servo housing, likely due to an inaccurate servo CAD model.
3	Redesigned to fit sensors. Thinned or removed walls between sensors to decrease size of the sensor housing. Added screw holes for all sensors.	Reduced the number of teeth of the gear driving the internal gear, decreasing the range of the roll mechanism. Increased the gear ration of the first two gears by changing their pitch in order to compensate.	Since larger pocket sizes for the sensors made the sensor housing longer, the roll rods would no longer fit on either side of the roll ring and so the tilt mechanism was change.	Separated the tube side locking ring into two pieces (roll servo housing and tube side locking ring).	No Change	Removed micro servo mount, leaving two vertical bars to attach brackets to hold the sensor housing.	●Tilt mechanism did not have full 90 degrees of motion. ● Roll ring collided with the gear driving the internal gear, needed to cut a piece out of the vertical bar of the roll ring.
4	No Change	No Change	Resynthesized four bar linkage to increase range of mechanism.	No Change	No Change	Added the brackets directly to the vertical bars of the roll ring rather than using machine screws to attach the brackets to the vertical bar.	● Too much modularity on the design made the assembly very difficult to put together.
5	Redesigned sensor housing to allow more space for the SWIR lens. Moved the Laser Rangefinder to the top of the housing.	No Change	Redesigned tilt mechanism to use gears rather than a four bar.	Combined the locking rings, servo housing, and Goblin body into one turret housing to reduce the number of pieces in the assembly.	Changed nose cone to attach to roll ring rather than Goblin body, allowing the nose cone to rotate with the roll mechanism. Changed the cut of the nose cone to cover more of the opening.	Moved the brackets to attach to the outside of the sensor housing.	● Gears were all 3D printed and did not mesh well. ● Gear attached to tilt servo collided with nosecone in fully retracted position.
6	Itered the sensor housing to make it machinable by increasing wall thickness and adding fillets to inside corners.	Changed from a position servo to a continuous turn servo to increase the range of motion of the mechanism, which allowed for a gear ratio other than 1:1. Perfromed parts selection to find aluminum gears for the roll mechanism.	Changed the gear mounted on the servo so that it did not collide with the nose cone. Change the ratio of the gears to increase the torque of the mechanism.	Changed the holes for the shafts holding the gears for the new gearing.	No Change	Reduced the size of the brackets.	● Roll servo does not line up well with the shaft causing grinding and vibration. ● gears for tilt mechanism are plasic and do not attach well to the micro servo.

Table 22: Integrations

Figure 109: Final Turret, Front View and Figure 110: Final Turret, Top View shows the final product, produced in TOIL using several 3D printers as well as the machine shop.

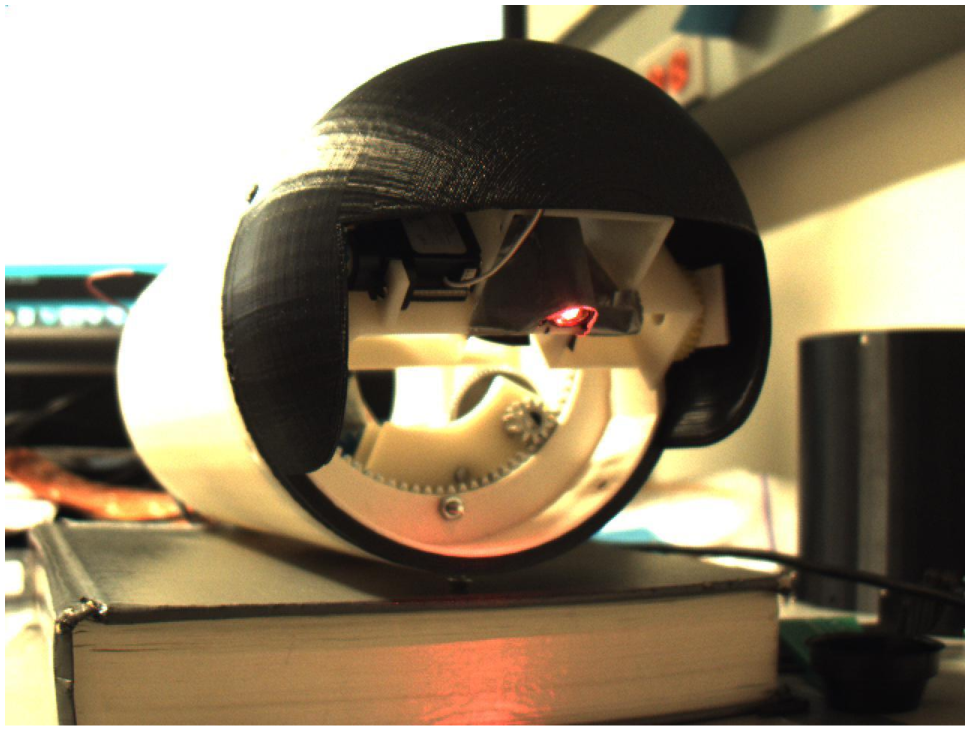


Figure 109: Final Turret, Front View

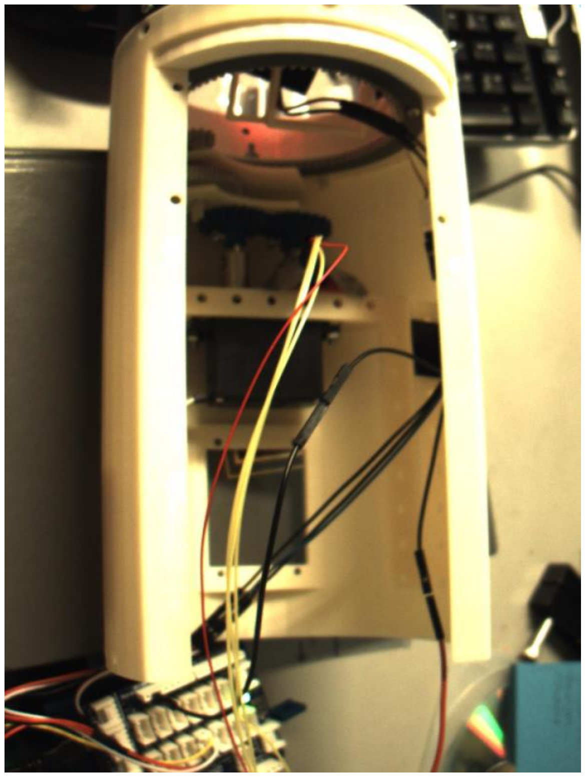


Figure 110: Final Turret, Top View

Software

We programmed an Arduino Duemilanove using the Arduino IDE in order to control a test turret that consisted of a roll and tilt 180 degree position servos. The test turret was able to perform several actions based on command inputs from a user. Figure 111 shows an overall flowchart of the software that was developed for the test turret.

- The roll servo and tilt servo positon were able to be changed
- The roll servo and tilt servo speed were able to be changed
- The status of the turret (including the current position and speeds of the servos as well if the heartbeat was active or not) was able to be viewed by the user at any time
- A home command that moved the servos to a preprogrammed default home position was implemented
- A scan mode was implemented.

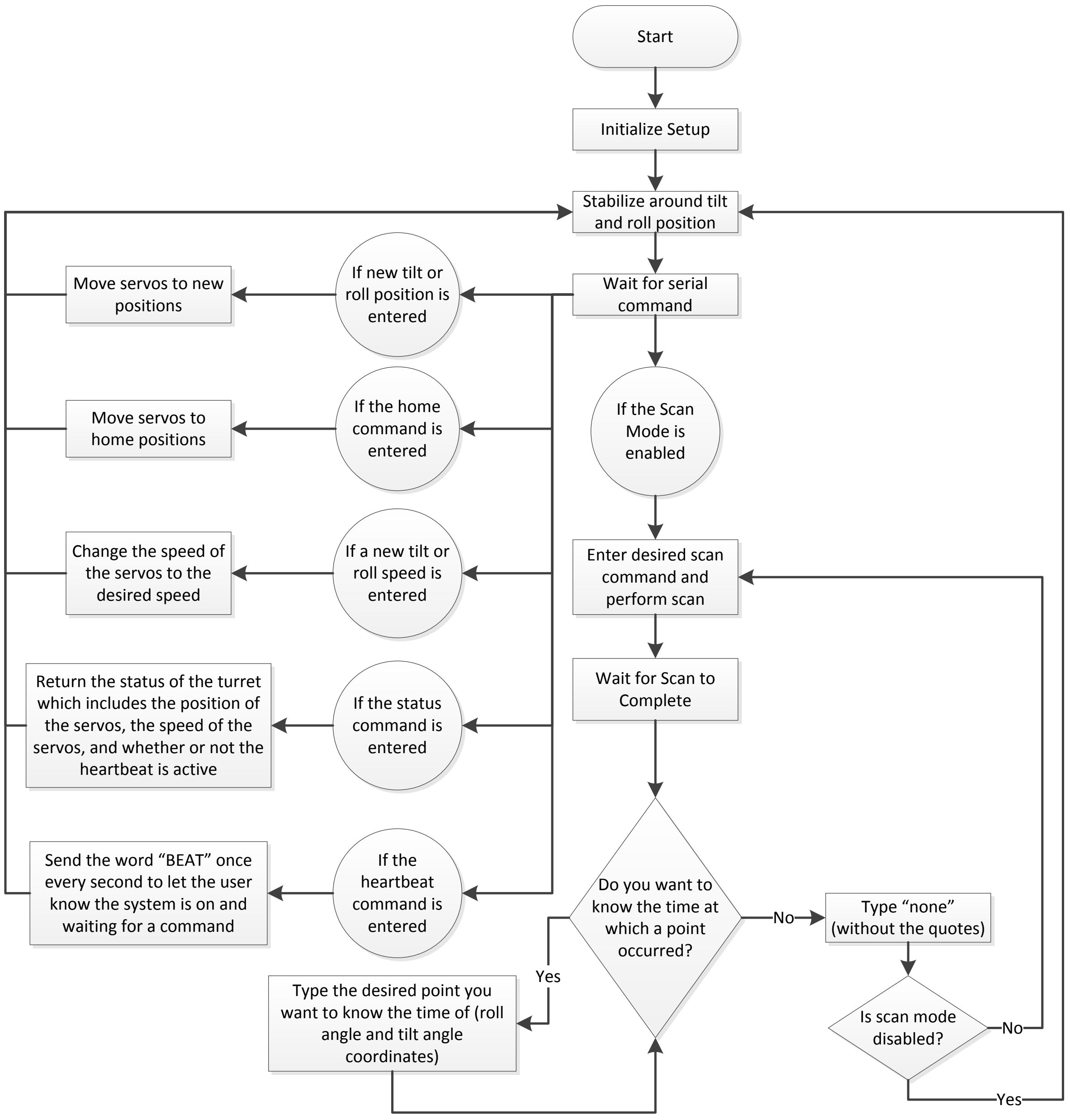

Start
Initialize Setup
Stabilize around tilt and roll position
Wait for serial command
If new tilt or roll position is entered
Move servos to new positions
If the home command is entered
Move servos to home positions
If a new tilt or roll speed is entered
Change the speed of the servos to the desired speed
If the status command is entered
Return the status of the turret which includes the position of the servos, the speed of the servos, and whether or not the heartbeat is active
If the heartbeat command is entered
Send the word "BEAT" once every second to let the user know the system is on and waiting for a command
If the Scan Mode is enabled
Enter desired scan command and perform scan
Wait for Scan to Complete
Do you want to know the time at which a point occurred?
Yes
Type the desired point you want to know the time of (roll angle and tilt angle coordinates)
No
Type "none" (without the quotes)
Is scan mode disabled?
No
Yes

Figure 111. Overall Flowchart of Turret Control

The scans program module consisted of a raster scan, a spiral scan, and a clover scan. For each of the scans, the user was able to change 2 parameters:

- For a raster scan, the user could change the height and width of the raster scan
- For a spiral scan, the user could change the end radius of the spiral as well as the density (i.e. the number of revolutions) of the spiral
- For a clover scan, the user could change the end radius of the clover as well as the number of leaves of the clover (e.g. a 2 leaf clover scan is a Figure-8)

Also, after each scan was completed, the program asked the user if they would like to know the time at which any point occurred throughout the scan, and the program provided the coordinates of the roll servo angle and tilt servo angle for the user to choose from. If the user wanted to know the time of a certain point, then they entered the roll servo angle and tilt servo angle of that desired point and the time at which that point occurred in the scan would be displayed. If the user did not wish to know the time at which a certain pointed occurred, then they typed “none” (without the quotes) and then the user was able to enter another scan with new parameters or disable scan mode.

During point-to-point movements, the turret was being stabilized by an inertial measurement unit (IMU) composed of an accelerometer and gyro. The roll and tilt of the IMU was used to represent the types of movements the UAV would encounter while flying (such as banking turns or turbulence). The stabilization was implemented so when the IMU rolled or tilted one direction a certain amount of degrees, the turret would move that same amount of degrees in the opposite direction to cancel out the IMU movement.

After several tests, it was proven that a complementary filter produced fewer random angle calculations than a Kalman filter when the IMU was stationary and the complementary filter also performed more angle calculations in the same amount of time as a Kalman filter (meaning the complementary filter has a higher output bandwidth than the Kalman filter. Thus, complementary filters (one for roll angle and one for tilt angle) were used to accurately combine the accelerometer readings and gyroscope readings in order to produce a resultant roll angle of the IMU and a resultant tilt angle of the IMU.

Conclusion

This capstone project involved the design and development of a turret that housed 3 sensors and integrated hardware and software to perform desired actions that will be useful for the WILD Goblin UAV. The sensor turret was successfully able to reach the required positions and orientations using software to control the roll and pitch mechanism.

Appendix G. B Term MQP Addendum

Introduction

This report describes the additional work that was performed at WPI following the completion of the main body of this report: Design and Development of the WILD Goblin Sensor Turret at MIT Lincoln Laboratory. The purpose of this extension is to provide MATLAB software which can be used to control the sensor turret in response to video input. To accomplish this we used the MATLAB Computer Vision System Toolbox library including a tracking script written during the 2013 senior project.

Background

There were two desired outcomes from this extension: interfacing the existing tracking software with the turret commands and developing stabilization software. Tracking targets is essential for the success of WILD Goblin missions. In order to keep targets of interest within the field of view of the sensors, targets must be tracked mechanically. While the MATLAB software is appropriate for tracking a target within a video, the target could leave the field of view of the sensor if the turret is not moved in response to the motion of the target. For this reason, the turret must be moved in response to the video feed.

The stabilization software is disconnected from target tracking; no turret commands result from video stabilization. A target is usually acquired by the WILD Goblin without knowledge of its identity. In order to determine what the UAV is looking at or tracking, the WILD Goblin will eventually have identification software. This software will require relatively stable images of the target. While some stabilization is provided mechanically through the use of the IMU and the embedded stabilization code on the Arduino, MATLAB stabilized video would be beneficial for identification software.

The MATLAB tracking software was developed as an extension of the 2013 senior project. The software presents a frame of the video to the user who selects a target by drawing a box around the target. The script then tracks the target using feature matching; identifying features in the user-drawn bounding box, finding those features in subsequent frames, and transforming the box in response to feature point movement.

For this project, the video used for testing tracking was thermal video. We took several videos representative of scenarios the WILD Goblin could be used in. Since vehicles are one of the objects to be identified by the UAV, we took and tested with several videos of a car from various vantage points. We also took videos with various backgrounds, as the temperature and color differences between target and background could be stark or very similar, in which case the target could be more easily lost. Finally, we also took videos with the camera moving, the target moving, and both the camera and target moving. This covers situations in which the background is changing relative to the target and in which the frame is changing with the background.

The following goals and deliverables were defined for this project addendum:

- Implemented video stabilization in MATLAB
- Wrote MATLAB code to compose turret commands to send to the Arduino based on video input
- Collected several videos which can be used to test the tracking and stabilization code

Methodology

For this extension we used the MATLAB Computer Vision System Toolbox, which was available at WPI. The software written is compatible with MATLAB 2013b and later.

Figure 112 shows a diagram of the proposed MATLAB code structure. The MATLAB software consists of a tracker, code to convert the output of the tracker into turret commands, and a simulation to show the MATLAB user. For the tracker, either the tracking script written in the 2013 project or a custom tracker was used, depending on the performance of the 2013 tracker using available video. The code to compose the turret commands from the tracker data created strings and sent the strings over serial to the Arduino. The simulation consists of an output panel showing the location of the object being tracked, an indication of the orientation of the turret, as well as other output which could be useful to the user.

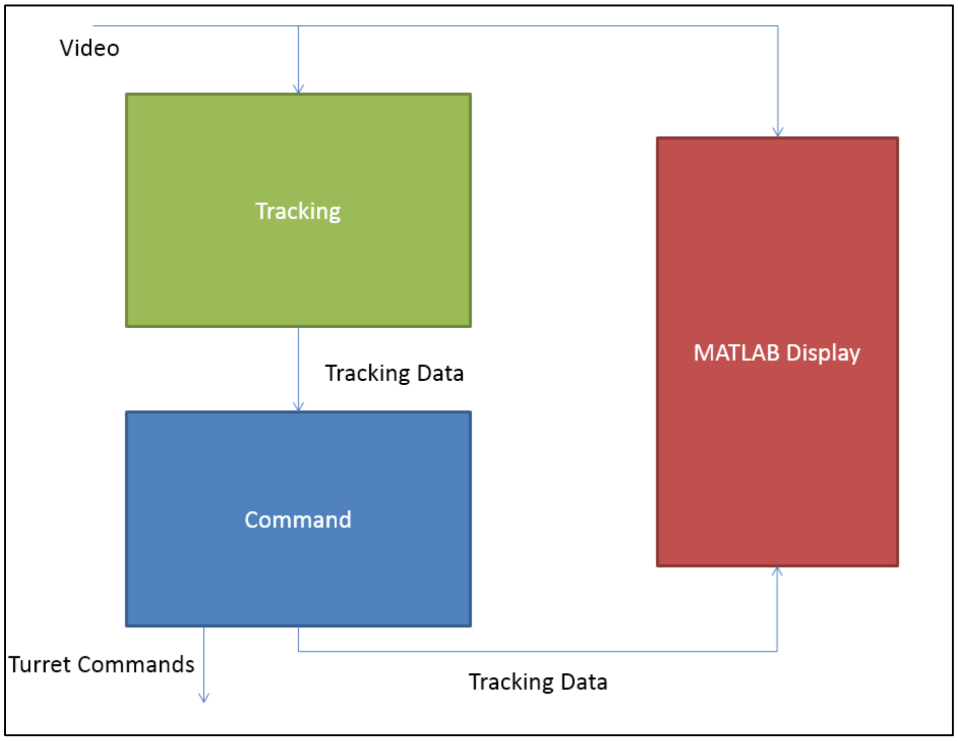


Figure 112: MATLAB code structure for the Tracking software

The video stabilization code was created by modifying some MATLAB sample code (link in the Appendix XXX with the code) involving video stabilization. The video stabilization code works by first inputting a thermal .mov video into the MATLAB file that has a resolution of 640 by 480. Then once the run button is pressed in MATLAB, the code finds the centroid of the hottest object, places an inner and outer stabilization box around that centroid, and keeps track of how much that point moves with respect to its previous frame.

For collecting thermal video, we used the FLIR One, available through our advisor, as we could not bring the FLIR Quark 2.0 back to WPI for testing. This camera uses the iPhone 5S, which one team member owned, to record videos and store them. Table 23 shows a comparison of the two devices.

Table 23: Comparison of FLIR One and FLIR Quark 2

Quality	FLIR Quark 2 with 35mm lens (f/1.5)	FLIR One
Cost	Unknown	$349.99
Field Of View	18° x 14 ° [1]	
Resolution	640 x 512 [1]	640 x 480 [2]
Frame Rate	60 Hz [3]	9 Hz [4]
Scene Range Temperature	-40 – 320 °F	32 – 212 °F [2]
Sensitivity	<50 mK at f/1.0 [5]	0.18 °F [2]

Results

The final MATLAB code consisted of three modules with some overlap. It was determined that a display panel would be useful for operation so the simulation and turret command modules were not completely separated. Figure 113 shows the final code structure.

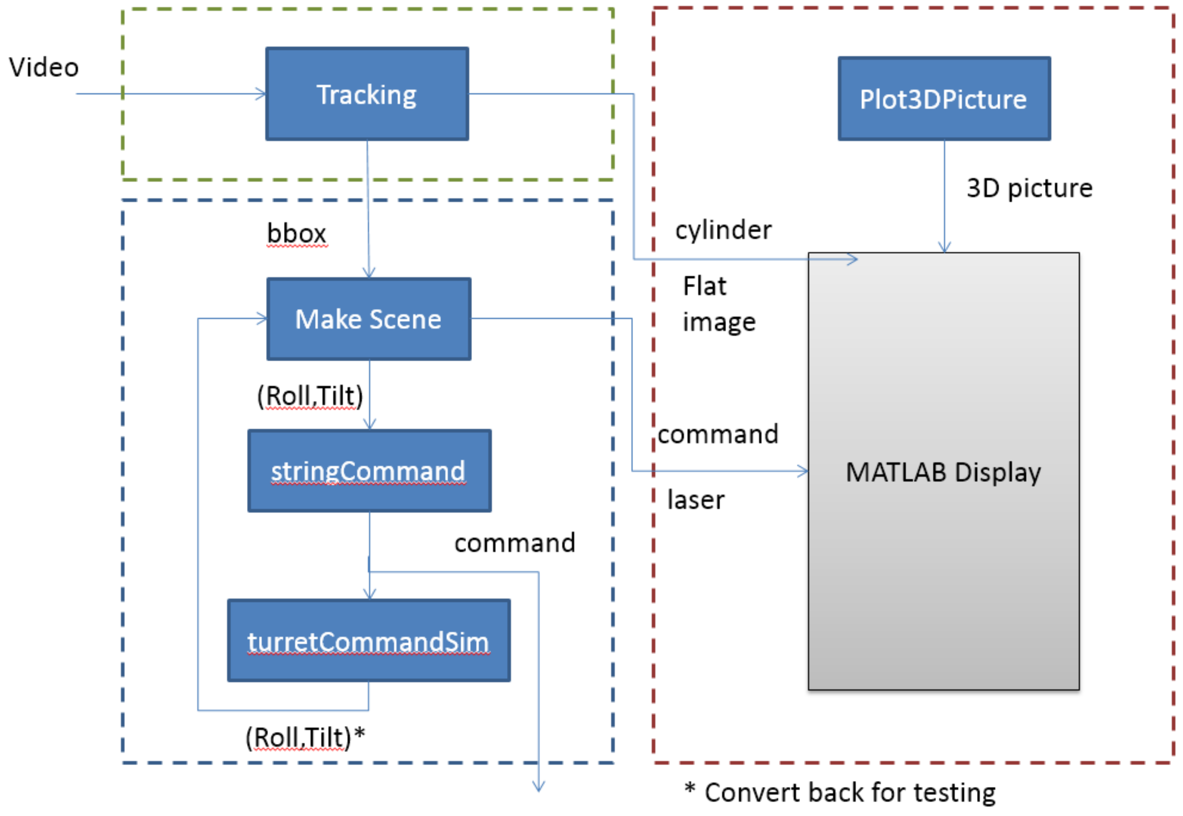


Figure 113: Final MATLAB Tracking Software Structure

Testing showed that the 2013 tracking code was usable for our purposes. While that code was originally tested with a video we did not have access to outside of Lincoln Laboratory, the program worked well with a few adjustments made for the videos we collected with the FLIR One. We collected videos inside and outside. Our inside videos used very hot objects, such as a lamp or soldering iron, against a cold featureless background, such as a blank wall. The outside videos were taken on a field on the WPI campus from which a road was visible.

The tracking software uses user input to identify a target in the first frame of the video. The frame is displayed and the user draws a rectangle containing the target to be tracked. Feature points are identified in the rectangle, or bounding box, and identified again in subsequent frames. The box is transformed according to the identification of the feature points in order to track the object from frame to frame. If too few feature points are identified, the target has been lost.

The command composition software uses the bounding boxes from the tracking code to pick a point to aim the turret at. The MakeScene function takes the points for a bounding box and finds the location of the center of the box. The pixel location is used calculate the roll and tilt positions of the turret. Figure 114 shows the variables used to calculate the tilt and roll angles, or φ and θ.

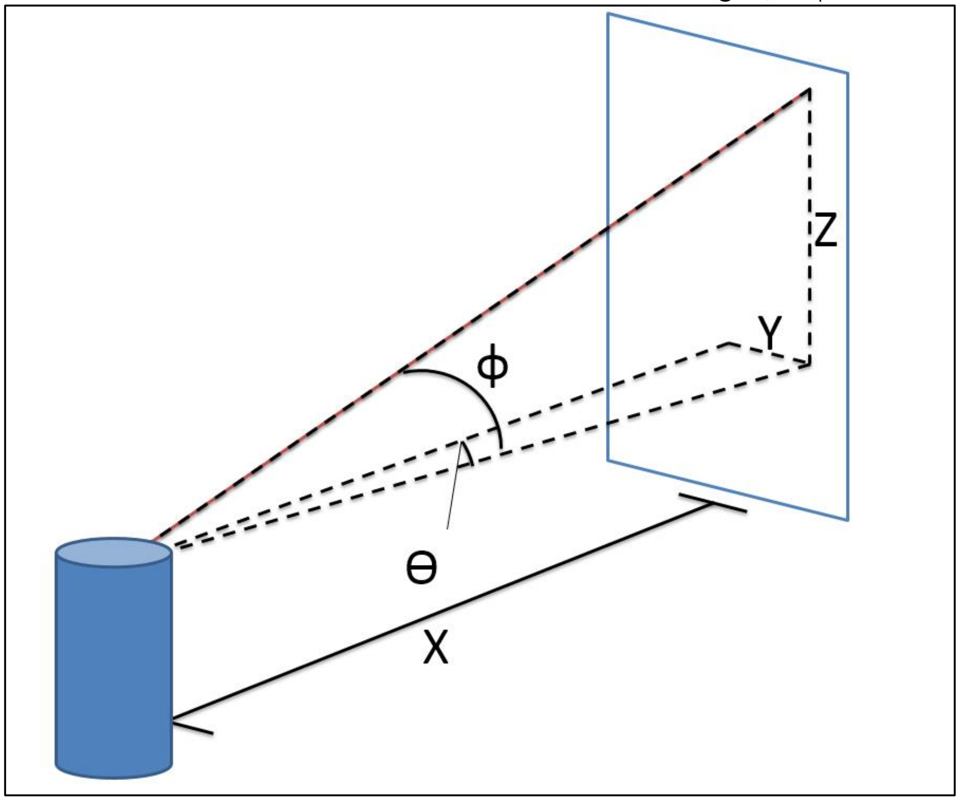
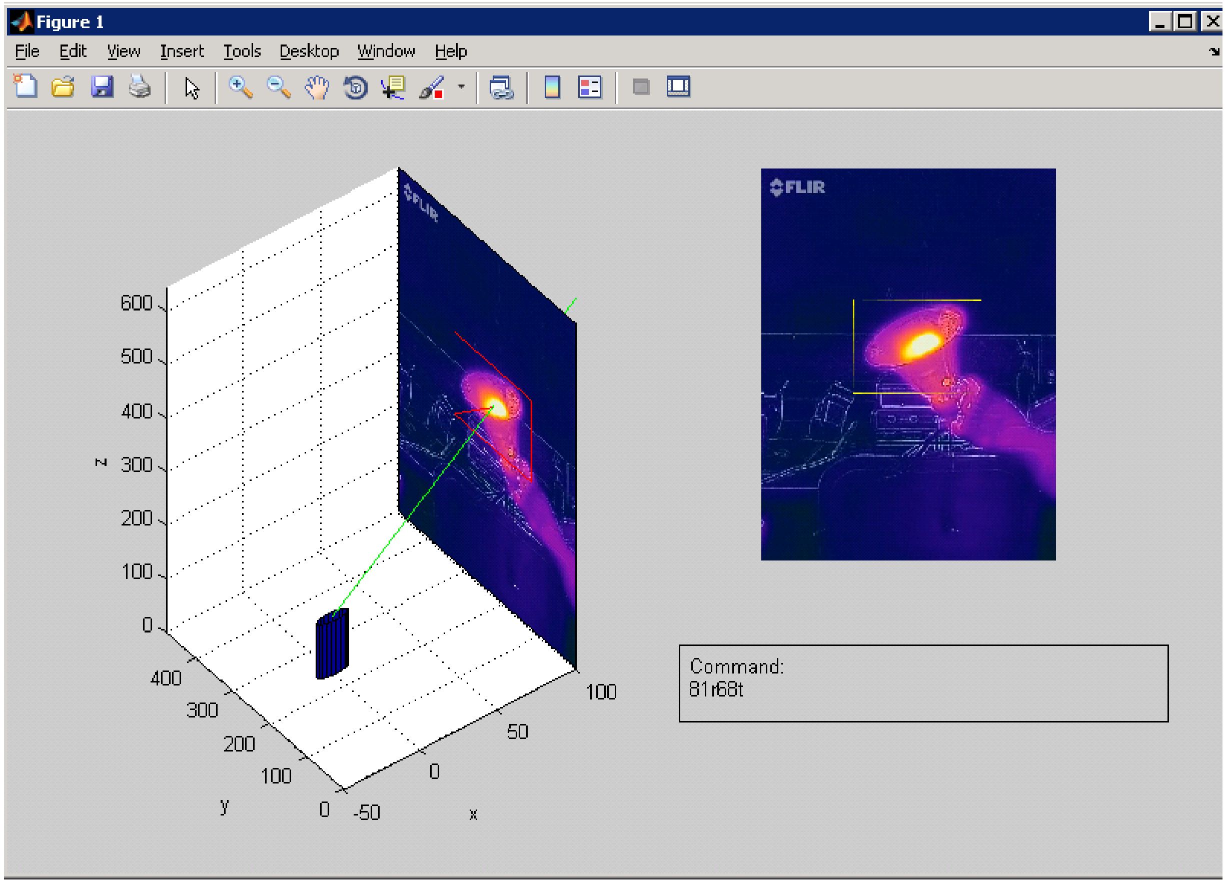


Figure 114: Variables used to calculate Roll and Tilt

$$\theta = \operatorname{atan}\left(\frac{y}{x}\right)$$
$$\theta = \frac{360 \times \theta}{2 \times \pi}$$
$$w = \sqrt{(x^2 + y^2)}$$
$$\phi = \operatorname{atan}\left(\frac{z}{w}\right)$$
$$\phi = \frac{360 \times \phi}{2 \times \pi}$$

The values of φ and θ, shown above, are inserted into a turret command string. The format of the turret command is “φtθr” (ex. “30t60r”). This string is what would be sent over serial to the Arduino in the actual turret setup.

The simulation software was originally intended to test the turret control and tracking software since the turret was not available for testing, but could also be used for user or tester feedback. Figure 115 shows a screenshot of the final user feedback panel showing the simulation. The figure on the right is a 3D plot showing the frame the tracker is currently processing, the bounding box around the target, the cylinder representing the turret, and a 3D line plot representing a laser. This shows that the turret is aiming at the correct point in the image. On the right side of the window, the frame of the video currently being processed as well as the bounding box is shown. Finally, on the lower right there is a textbox showing the command string generated by the turret simulation.



Command:
81r68t

Figure 115: User Feedback Panel

Stabilization

As an addition to the work completed on the project in the previous terms we also spent time learning about and implementing video stabilization using MATLAB and the Computer Vision System Toolbox. The program, located in Appendix XXX, is designed to take in any video file that is 640 by 480 pixels and convert it to 320 by 240 pixels. The two videos (shown in Figure 116 with the original unstabilized video on the left and the stabilized video on the right) are then displayed in an organized intuitive way, so the user can compare the original video to the resulting stabilized video.

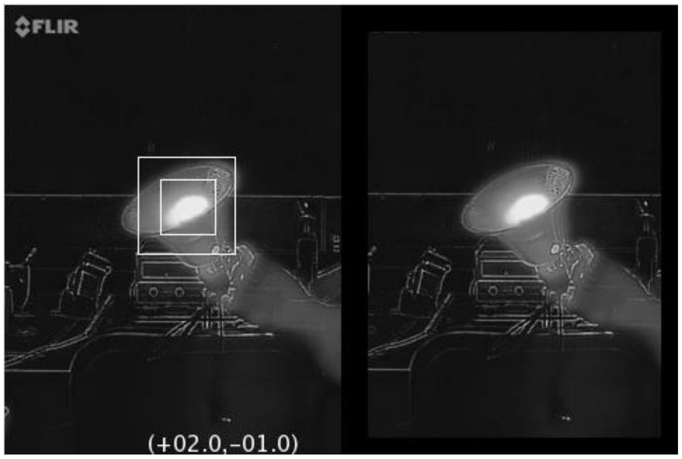


Figure 116. Video Frame Displaying the Unstabilized Video (Left) with the White Stabilization Boxes and the Stabilized Video (Right)

There are several variables in the MATLAB code that can be changed by the user in order to influence the output stabilized video. The user is able to define the point to be stabilized around, the size of the area that defines the stabilized point (inner stabilization box on the left in Figure 116), and how much the stabilization boxes are allowed to move from one video frame to the following video frame and still remain locked onto the same position to stabilize around (outer stabilization box on the left in Figure 116).

The program first reads in the desired frame of the video and marking the point to be stabilized around, based on the previously mentioned variables. The video then continues to read in the video, frame by frame, and compares the position of the small and large box in the current frame to the position of the small and large box in the previous frame, using point feature matching. The difference in positions in the large and small boxes is then displayed as an offset in the bottom of the left video in Figure 116 an x and y position. The offset is how far the stabilized video needs to adjust either its horizontal or vertical borders for an x or y offset, respectively, in order for the stabilized video to make the focus object (i.e. the object or part of an object with the boxes around it) appear as if it was not moved in the video frame, thus giving the appearance of a stabilized video.

The next piece of code that was added was centroid finding code in order to focus on a similar type of point between all of the videos, as opposed to choosing random points to stabilize around. Figure 117 shows a sample centroid finding image from the video frame displayed in Figure 116. We first load a thermal video into the MATLAB program then convert the desired frame (i.e. the frame that will have its centroid found) to a binary frame, consisting only of black and white. This makes the really bright spots (i.e. the hot spots) show as white and the rest of the frame turn black (depending on the threshold the user provides). Then the centroid finding code looks for white blobs that are at least a certain size, defined by the user, and stores the x and y position of that centroid. We were then able to use this centroid vector’s x and y positions to make the video stabilize around the center of the largest hot object that was in the desired initial frame of the video. As the video progresses through its frames, the 2 stabilization boxes are able to stay focused on the original hot object and stabilize around that object by shifting the border of the stabilized video by the required horizontal or vertical offset.

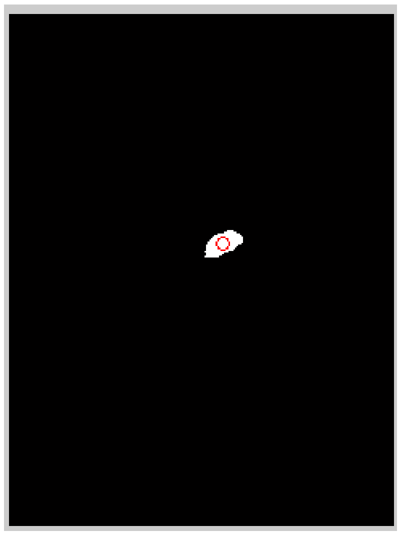


Figure 117. Sample Centroid Finding Result

Figure 118 shows a demonstration of the video stabilization with 3 different video frames overlaying each other. On the left of Figure 118, three distinct positions of the 2 stabilization boxes focused on the same object in each frame are shown. In Figure 118 on the right, the resulting stabilized videos are also overlapping, however the image that the 2 stabilization boxes were focusing around appears unmoved in all three stabilized video frames, which validates video stabilization. Figure 119 shows the same three video frames that are in Figure 118, but instead of being overlapped, Figure 119 displays the video frames how they were produced from the MATLAB script. Figure 119 still demonstrates the video stabilization because on the right side of Figure 119, one can see that the bright (hot) object stays in relatively the same position between all 3 frames, while the stabilization boxes move around with the rest of the frame in the left of Figure 119.

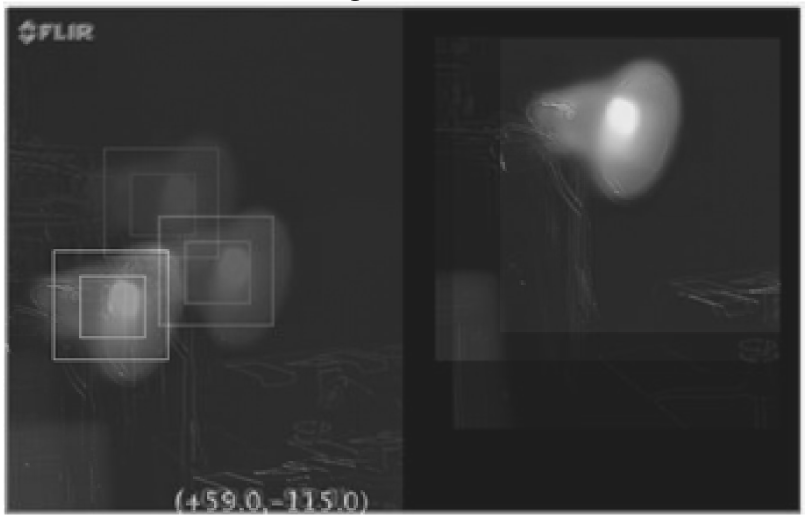


Figure 118. Overlapping Frames of a Video Showing the Stabilized Video Result on the Right

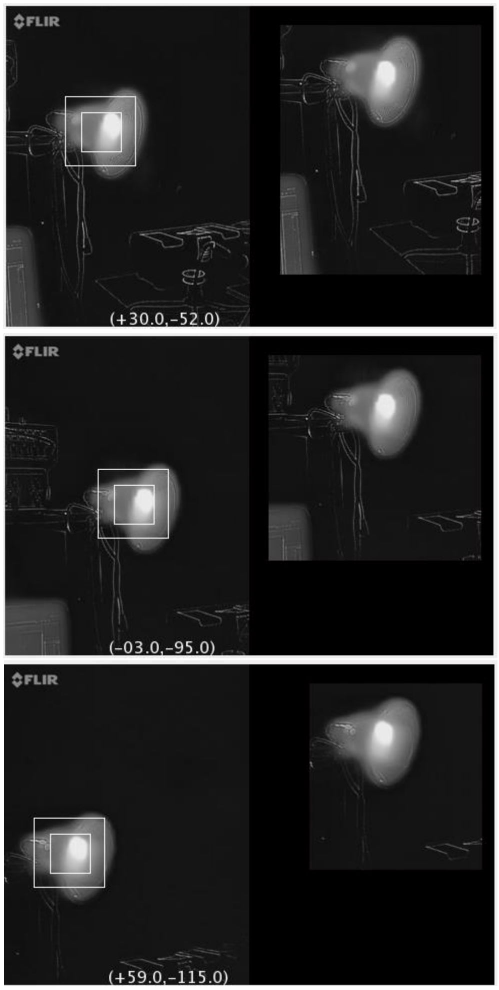


Figure 119. Video Frames Demonstrating Video Stabilization

In order to simulate the various scenarios the WILD Goblin will encounter while flying, several test videos were recorded to display the robustness of this video stabilization. The first video involved keeping the hot object stationary while only moving the camera (e.g. the video examples shown in Figure 118 and Figure 119). This could represent the UAV flying over a stationary object, but still needing to stabilize the video for better object identification.

Another scenario for testing was keeping the camera mainly stationary and only moving the hot object in the FOV of the camera. This scenario could represent the UAV flying, with its cameras aimed almost straight ahead, and seeing some target of interest off in the distance that constantly changes orientation, while the cameras stay stationary on the flying UAV. The process of video stabilization is the same as the previous example, and Figure 120 shows the moving object, with the boxes focused around the hottest part of the video frame on the left, and the resultant stabilized video on the right.

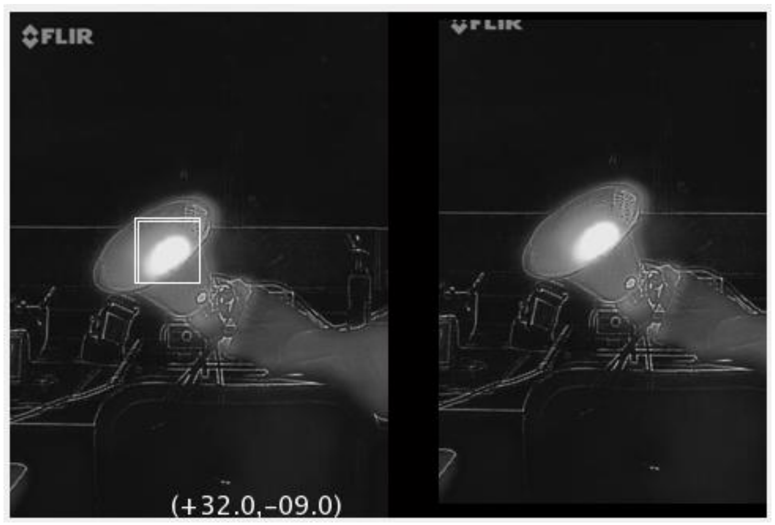


Figure 120. Initial Frame of Moving Object, Stationary Camera Test Video

Discussion

The tracking software will be useful for transforming video input into turret commands for the Arduino. While some assumptions were made to compensate for the lack of equipment to test on, the final product demonstrates using video feedback to control the turret.

The tracker picks a target based on user input; identifying the target in a frame by drawing a box. While this is not what is intended for the final system, as the final system will autonomously identify targets, it is useful for testing. The stabilization code does find the tracking target autonomously by finding the hottest object in the frame and drawing a box around it, but the tracking was left as user input in order to see what kinds of objects and features work well as targets.

Using multiple videos for testing containing different targets, we were able to see what made a good target in a video. For example in the videos of the lamp, most of the feature points that were picked were not associated with a particular color (that is, they were not associated with hot objects), but were located on defined lines in the image or stark temperature contrasts. Figure 121 shows the points that were picked when the lamp was selected as the target. The feature points are located anywhere but the bright light bulb. This is likely because of the nature of FLIR One videos: they combine the visual and thermal images in every frame into a hybrid video which contains the color scale of thermal and dark lines indicating features of the video.

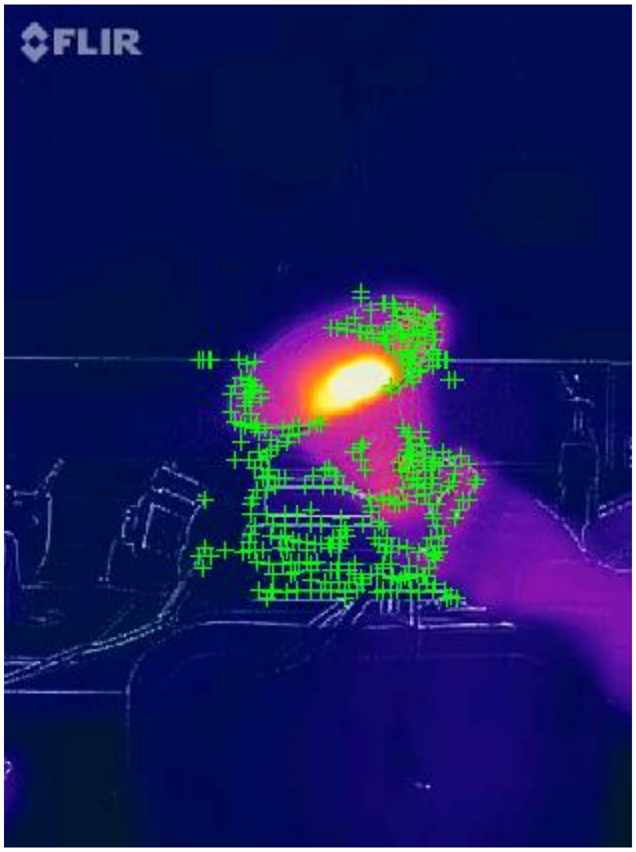


Figure 121: Feature Points on a Lamp

The main difference between the two video examples used for testing the stabilization code (i.e. moving camera with stationary object and moving object with a stationary camera) is the change in the background as video stabilization is being performed. In the first example (shown in Figure 118 and Figure 119), everything in the FOV of the camera changed from frame to frame, including the hot object of interest. In the second video (with the stationary camera, shown in Figure 120), the only changes between the frames of the video were the position and orientations of the hot object. The rest of the frame (i.e. the background) remained mostly unchanged throughout the video, thus showing the robustness of the video stabilization in various scenarios that the UAV could encounter.

We encountered some problems when using some thermal videos in which the desired object to be stabilized around did not have enough contrast with the warm background as the object moved farther away from the FLIR One camera. However, when there was a clear distinction between the hottest object and the coldest objects throughout the unstabilized video, the output video successfully stabilized around the part of an object that was in the 2 stabilization boxes in the original video.

Another issue with this stabilization code was realized when the hot object to be stabilized around moved close to the edges of the video frames. If the search border (the maximum allowable displacement) was set too large in the code and the object moved too close to the edge of the video, the 2 stabilization boxes would no longer be focused around the same initial hot object as before. The stabilized video can only shift its borders a limited amount before the object of interest goes out of frame. As shown in Figure 122, if the large amount of shifting caused the stabilization boxes (i.e. the position of the stabilization boxes in the stabilization video) to be hit by the expanded borders in the stabilization video, the boxes would lose the hottest object in the video frame.

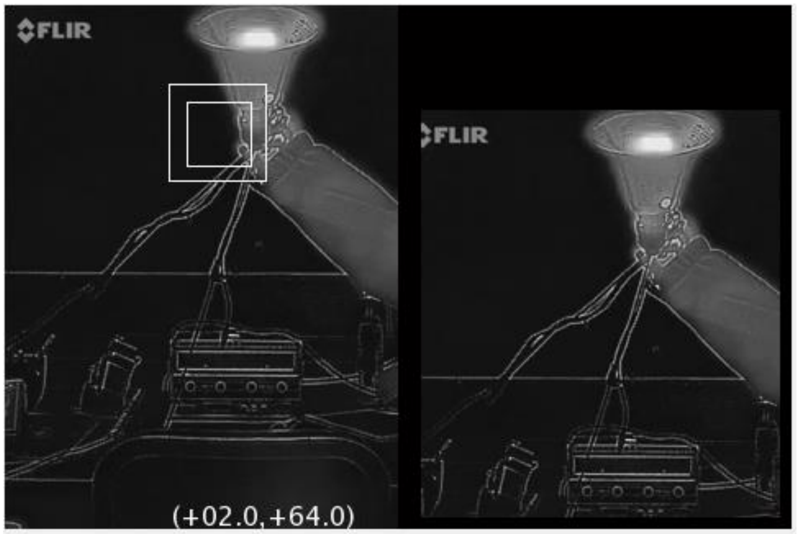


Figure 122. Stabilization Boxes Losing Object of Interest Close to Video Frame Borders

Shown in Figure 123, if the inner stabilization box was large enough, however, the stabilized video was able to compensate for the large displacement of the stabilized video borders and the inner and outer stabilization boxes were able to refocus on (or get close to) the hot object in the video frame.

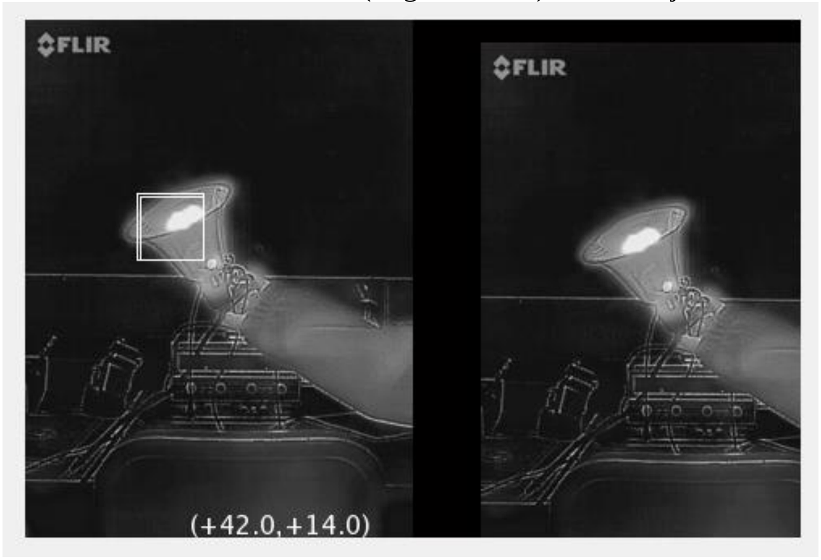


Figure 123. Stabilization Boxes Refocusing Near Initial Object of Interest in Video Frame

Conclusion

The software developed for this senior project extension in MATLAB will allow the WILD Goblin turret to move in response to a video input. We were successfully able to demonstrate a simulated target tracking scenario that used the same type of commands which can be used to move the turret. We were also able to implement video stabilization which reduced the change in position of a target frame to frame. Video stabilization will allow for the object of interest to be easily identifiable, even if the cameras are shaking while the UAV is flying. The mechanical stabilization (developed last term) will be able to compensate for the large motion of the UAV in order to stabilize the cameras. The video stabilization will adjust the video accordingly so that small camera movements do not distort the object of interest. The target tracking code will be able to send commands to the turret so it can autonomously track an object of interest.

Addendum Works Cited

[1] "Quark Product Specification".

[2] "FLIR Explore," [Online]. Available: http://www.flir.com/flirone/explore.cfm.

[3] "FLIR Quark Feature Comparison," FLIR, [Online]. Available: http://www.flir.com/cvs/cores/view/?id=64110.

[4] J. Carroll, "FLIR ONE infrared camera for the iPhone officially launches," Vision Systems Design, [Online]. Available: http://www.vision-systems.com/articles/2014/07/flir-one-infrared-camera-for-the-iphone-officially-launches.html.

[5] "Quark 2 Uncooled Cores," FLIR, [Online]. Available: http://www.flir.com/cvs/cores/view/?id=51266&collectionid=549&col=51275.

Appendix H.   Video Stabilization Code

```
clc
 Based off code found at http://www.mathworks.com/help/vision/examples/video-stabilization.html
% Input video file which needs to be stabilized.
filename = 'VIDEONAMEHERE.MOV';
hVideoSource = vision.VideoFileReader(filename, ...
                                      'ImageColorSpace', 'Intensity',...
                                      'VideoOutputDataType', 'double');
hTranslate = vision.GeometricTranslator( ...
                              'OutputSize', 'Same as input image', ...
                              'OffsetSource', 'Input port');
hTM = vision.TemplateMatcher('ROIInputPort', true, ...
                             'BestMatchNeighborhoodOutputPort', true);
hVideoOut = vision.VideoPlayer('Name', 'Video Stabilization');
hVideoOut.Position(1) = round(0.4*hVideoOut.Position(1));
hVideoOut.Position(2) = round(1.5*(hVideoOut.Position(2)));
hVideoOut.Position(3:4) = [1000 400];
%Read in video and find the centroid of video frame 2 from the hottest
%(brightest) object
centroids = [];
obj = VideoReader(filename);
video = read(obj);
hblob = vision.BlobAnalysis;
hblob.AreaOutputPort = false;
hblob.BoundingBoxOutputPort = false;

%look for a blob that is at least the following number of pixels
hblob.MinimumBlobArea =5;
framenumber=2
frame = video(:,:,:,framenumber);
frame=imresize(frame, 0.5);
pic = rgb2gray(frame);

%Changes the threshold of which pixels are deemed white or black, typical
%range = 0.7 to 0.9
contrastWeight  = 0.9
pic = im2bw(pic,contrastWeight);
imshow(pic)
centroid = step(hblob,pic)
figure(1)
hold on
imshow(pic)
scatter(round(centroid(:,1)), round(centroid(:,2)))

%Set dimensions of inner stabilization box
tempsizex=40 tempsizey=40

%Set the dimensions of the outer stabilization box which defines the
%maximum horizontal and vertical placement
searchborderx = 15 searchbordery= 15

%Use the calculated centroid to position the inner and outer stabilization
%boxes around the hottest part of the second video frame to stabilze around
pos.template_orig = [round((centroid(:,1))-tempsizex/2) round((centroid(:,2))-tempsizey/2)];
pos.template_size = [tempsizex tempsizey];   % [width height]
pos.search_border = [searchborderx searchbordery];   % max horizontal and vertical displacement
pos.template_center = floor((pos.template_size-1)/2);
pos.template_center_pos = (pos.template_orig + pos.template_center -1 );
fileInfo = info(hVideoSource);
W = fileInfo.VideoSize(1)/2; % Width in pixels
H = fileInfo.VideoSize(2)/2; % Height in pixels
BorderCols = [1:pos.search_border(1)+4 W-pos.search_border(1)+4:W];
BorderRows = [1:pos.search_border(2)+4 H-pos.search_border(2)+4:H];
sz = fileInfo.VideoSize;
TargetRowIndices = ...
  pos.template_orig(2)-1:pos.template_orig(2)+pos.template_size(2)-2;
TargetColIndices = ...
  pos.template_orig(1)-1:pos.template_orig(1)+pos.template_size(1)-2;
SearchRegion = pos.template_orig - pos.search_border - 1;
Offset = [0 0];
Target = zeros(18,22);
firstTime = true;

while ~isDone(hVideoSource)
    input = step(hVideoSource);
    %Resize the video (from 480 by 640 to 240 by 320)
    input=imresize(input,.5);
    % Find location of Target in the input video frame
    if firstTime
      Idx = int32(pos.template_center_pos);
      MotionVector = [0 0];
      firstTime = false;
    else
      IdxPrev = Idx;

      ROI = [SearchRegion, pos.template_size+2*pos.search_border];
      Idx = step(hTM, input, Target, ROI);

      MotionVector = double(Idx-IdxPrev);
    end

    [Offset, SearchRegion] = updatesearch(sz, MotionVector, ...
        SearchRegion, Offset, pos);

    % Translate video frame to offset the camera motion
    Stabilized = step(hTranslate, input, fliplr(Offset));

    Target = Stabilized(TargetRowIndices, TargetColIndices);

    % Add black border for display
    Stabilized(:, BorderCols) = 0;
    Stabilized(BorderRows, :) = 0;
    TargetRect = [pos.template_orig-Offset, pos.template_size];
    SearchRegionRect = [SearchRegion, pos.template_size + 2*pos.search_border];
    % Draw rectangles on input to show target and search region
    input = insertShape(input, 'Rectangle', [TargetRect; SearchRegionRect],...
                        'Color', 'white');
    % Display the offset (displacement) values on the input image, shown in
    % the bottom of the left video
    txt = sprintf('(%+05.1f,%+05.1f)', Offset);
    input = insertText(input(:,:,1),[100 300],txt,'FontSize',16, ...
                    'TextColor', 'white', 'BoxOpacity', 0);
    % Display video
    step(hVideoOut, [input(:,:,1) Stabilized]);
end
```

Works Cited

[1] J. Garamone, "From U.S. Civil War to Afghanistan: A Short History of UAVs," 16 April 2002. [Online]. Available: http://www.defense.gov/news/newsarticle.aspx?id=44164.

[2] "Time Line of UAVs," November 2002. [Online]. Available: http://www.pbs.org/wgbh/nova/spiesfly/uavs.html.

[3] J. D. Blom, "Unmanned Aerial Systems: A Historical Perspective," Fort Leavenworth, 2010.

[4] J. Stamp, "Unmanned Drones Have Been Around Since World War I," 12 February 2013. [Online]. Available: http://www.smithsonianmag.com/arts-culture/unmanned-drones-have-been-around-since-world-war-i-16055939/?no-ist. [Accessed 8 September 2014].

[5] "Unmanned Systems Integrated Roadmap: FY 2011-2036".

[6] Department of Defense, "Unmanned Systems Integrated Roadmap FY 2013-2038," 2013.

[7] "UAV Study v2.01," 2013.

[8] "Unmanned Aerial Vehicle Options Assessment," Massachusetts Institute of Technology Lincoln Laboratory, 2013.

[9] Defense Industry Daily, "BAE Acquires UAV Make ACR for $14.7M," 8 June 2009. [Online]. Available: http://www.defenseindustrydaily.com/BAE-Acquires-UAV-Maker-ACR-for-147M-05488/. [Accessed 26 August 2014].

[10] S. Gienow, "Full Campus Scan with Octo," Ecosynth, 4 February 2014. [Online]. Available: http://ecosynth.org/profiles/blogs/full-campus-scan-with-octo. [Accessed 3 September 2014].

[11] J. Lemons, "Radar," May 2003. [Online]. Available: http://www.geocities.ws/jasonlemons/radar/topic1.htm. [Accessed 26 August 2014].

[12] J. G. Mangum, D. T. Emerson and E. W. Greisen, "The On The Fly imaging technique," Astronomy & Astrophysics, 8 May 2007. [Online]. Available: www.aanda.org/articles/aa/full/2007/41/aa7811-07/aa7811-07.fig.html. [Accessed 26 August 2014].

[13] Jenoptik, "DLEM SR," 2014. [Online]. Available: http://www.jenoptik.com/en-diode-laser-rangefinder-dlem-sr. [Accessed 2 September 2014].

[14] Nikon, "Laser Rangefinders," 2008. [Online]. Available: http://nikon.com/about/technology/life/sportoptics/laser/index.htm. [Accessed 2 September 2014].

[15] Protherm, "Infrared Basics," [Online]. Available: http://www.pro-therm.com/infrared_basics.php. [Accessed 11 September 2014].

[16] Xenics Infrared Solutions, "Night Vision," 2014. [Online]. Available: http://www.xenics.com/en/infrared_imaging_applications/infrared_camera_detection_for_security_applications/application_-_night_vision.asp. [Accessed 36 August 2014].

[17] Sensors Unlimited, "SWIR Night Vision Camera Systems for Vehicle Navigation," 2014. [Online]. Available: http://www.sensorsinc.com/applications/military/night-vision-systems. [Accessed 3 September 2014].

[18] Sensors Unlimited, "Micro 640CSX SWIR Camera," 2014. [Online]. Available: http://www.sensorsinc.com/products/detail/microswir-camera. [Accessed 14 September 2014].

[19] Photonics Online, "Micro-SWIR™ Camera for Military Applications: SU640CSX," 2014. [Online]. Available: http://www.photonicsonline.com/doc/micro-swir-camera-for-military-applications-0001. [Accessed 3 September 2014].

[20] Industrial Precision Instruments, "How do Infrared Cameras work?," 2014. [Online]. Available: http://www.ipi-infrared.com/start-here/how-do-infrared-cameras-work2. [Accessed 2 September 2014].

[21] C. Douglass, "IR Spectal Bands and Performance," FLIR Systems, Inc., 2014. [Online]. Available: http://gs.flir.com/surveillance-products/surveillance-technology/imaging-technotes/IR_Spectral_Bands. [Accessed 7 September 2014].

[22] FLIR Systems, Inc., "How Does and IR Camera Work?," 2014. [Online]. Available: http://www.flir.com/thermography/americas/us/view/?id=55706. [Accessed 2 September 2014].

[23] FLIR Systems, Inc., "Sky-Watch Integrates FLIR Quark 640 into Small UAV," 2014. [Online]. Available: http://www.flir.com/cvs/cores/view/?id=61163. [Accessed 2 September 2014].

[24] FLIR Systems, Inc., "How is NEDT measured?," 2014. [Online]. Available: http://flir.custhelp.com/app/answers/detail/a_id/128/~/how-is-nedt-measured%3F. [Accessed 3 September 2014].

[25] FLIR Systems, Inc., "Quark 2 Uncooled Cores," 2014. [Online]. Available: http://www.flir.com/cvs/cores/view/?id=51266&collectionid=549&col=51275. [Accessed 3 September 2014].

[26] Analog Devices, Inc., "The Five Motion Senses: Using MEMS Inertial Sensing to Transform Applications," 2014. [Online]. Available: http://www.analog.com/en/content/over_five_motion_senses/fca.html. [Accessed 11 September 2014].

[27] J. K. Oestergaard, "AAI RQ-7 Shadow 200," Aeroweb, 22 May 2014. [Online]. Available: http://www.bga-aeroweb.com/Defense/RQ-7-Shadow.html. [Accessed 2 September 2014].

[28] Marine Corps, "RQ 7B Shadow," 2014. [Online]. Available: http://www.marines.com/operating-forces/equipment/aircraft/rq-7-shadow#features. [Accessed 2 September 2014].

[29] M. H. Ettenberg and M. D. S., "SWIR Imaging," Photonics, 2014. [Online]. Available: http://www.photonics.com/EDU/Handbook.aspx?AID=25134. [Accessed 3 September 2014].

[30] J. Merchant, "Infrared Temperature Measurement Theory and Application," Omega Engineering, Inc., 2014. [Online]. Available: http://www.omega.com/techref/iredtempmeasur.html. [Accessed 4 September 2014].

[31] T.-R. Hsu, "Chapter 2 Working Principles of MEMS of Microsystems," San Jose State University, 2008. [Online]. Available: http://www.engr.sjsu.edu/trhsu/ME189_Chapter%202.pdf. [Accessed 4 September 2014].

[32] J. Esfandyari, R. De Nuccio and G. Xu, "Introduction to MEMS gyroscopes," SolidState Technology, 2014. [Online]. Available: http://electroiq.com/blog/2010/11/introduction-to-mems-gyroscopes/. [Accessed 2 September 2014].

[33] P. Jain, "Magnetometers," EngineersGarage, 2012. [Online]. Available: http://www.engineersgarage.com/articles/magnetometer. [Accessed 4 September 2014].

[34] "Our Heritage," [Online]. Available: http://www.northropgrumman.com/AboutUs/OurHeritage/Pages/default.aspx.

[35] L. G. D. Deptula, "Air Force Unmanned Aerial System (UAS) Flight Plan 2009 - 2047".

[36] "BAE Acquires UAV Maker ACR for $14.7M," Defense Industry Daily, 8 June 2009. [Online]. Available: http://www.defenseindustrydaily.com/BAE-Acquires-UAV-Maker-ACR-for-147M-05488/. [Accessed 26 August 2014].

[37] "Night Vision," Xenics Infrared Solutions, [Online]. Available: http://www.xenics.com/en/infrared_imaging_applications/infrared_camera_detection_for_security_applications/application_-_night_vision.asp. [Accessed 26 August 2014].

[38] J. Dorich and D. Mulcahy, "GOBLIN Eyes: Sensor Turret Target Tracking for Small Unmanned Air Vehicles," 2013.

[39] Massachusetts Institute of Technology, "Modular Programming," 16 February 2001. [Online]. Available: http://web.mit.edu/16.070/www/year2001/Modular_Programming.pdf. [Accessed 8 September 2014].