Spring 2019

Automatic Range Finding Bow Sight

Garrett Gill
gsg9@zips.uakron.edu

Dillon Denny
dmd141@zips.uakron.edu

Cory Verba
cwv5@zips.uakron.edu

Automatic Range Finding Bow Sight

Project Design Report

Dillon Denny
Garrett Gill
David King
Cory Verba

Faculty Advisor: Dr. Veillette

04/26/2019

The goal of the Automatic Range Finding Bow Sight project is to integrate a rangefinder into the sight system of a compound bow and autonomously adjust a single sight pin based on the calculated arrow trajectory to assist an archer in striking the bullseye of their target. To do this, the user inserts the average projectile speed of their bow, aims at their target, and presses a button to initiate the range finding sequence. The anticipated location of impact at the measured distance is then calculated. The sight pin then adjusts to the corresponding location. The user then uses it to aim the bow and assist them in striking the bullseye of their target. My primary roles in this senior design project have been to design and program the range finding system, determining the arrow trajectory at varying speeds, and assisting in calculating the necessary sight pin movements to compensate for the arrow trajectory.

The major component that is used to measure the distance between the user and the target is the TeraRanger Evo 60m time of flight sensor. This sensor operates by emitting a focused beam of light from an LED and measuring the time that it takes for the beam to project out to the target and reflect back to the sensor. This measured length of time is then divided by two and multiplied by the speed of light to determine the distance from the user to the target in millimeters. Being that the sensor is a "free running" device, it continually executes this function until it receives a trigger reading command from the master device. In this case, the master device is a PIC microcontroller and data is transferred between the microcontroller and sensor via I2C communication protocol. Once the user presses the corresponding button, the microcontroller sends a trigger reading command to the sensor to which the sensor sends three data bytes back. These three data bytes include the distance measurement in millimeters and the checksum. The distance bytes are then stored in a single variable and then converted to yards to be displayed and used to determine the sight pin location.

The arrow trajectory calculations were developed through the use of an online ballistics calculator, and then tested and verified by shooting multiple bows of different speeds at varying distances and comparing the results to the calculations. Using these results and measurements taken from a presighted in bow, my group member and I were able to calculate the resolution at which the sight pin is required to adjust per yard the user is away from the target.

Along with these roles, I also assisted my fellow group members with various aspects of their responsibilities such as drawing wiring blueprints, assisting in the design of the case, and general program troubleshooting.

- Dillon Denny

Student# 2899285

My design contributions for the project Automatic Range Finding Bow Sight were the user interface for operator control and 3D design/printing. The goal of our project was to allow a user to enter the speed that their bow shoots their chosen arrow weight, then allow them to obtain the range to their target using an onboard rangefinder. Once both the speed and range were obtained the data was used to adjust the sight pin to the appropriate bullseye location.

The user interface of our design consisted of a 16x2 LCD display and four push-buttons. Communication with the screen was performed using Parallel Master Port and the four buttons were wired to the processor so that when they were pressed, they tied the input to ground and performed the necessary action. Four screens were programmed for the LCD one that displayed speed, one that displayed the last range obtained by the rangefinder, a “main” or home screen and one that displayed the new range and drop when the rangefinder was triggered. The push-buttons operations consisted of raising and lowering the speed in increments of 10 (2 buttons), toggling between the displays and triggering the rangefinder.

The 3D design of the case and mounting components was done using an open source CAD software designed for creating 3D printable STL files. The components that were printed consisted of the case that housed all of the components, the mounting plate for the motor and cable guides and all of the components necessary to attach the project to a compound bow. The case dimensions specified for the project were 5”x5”x5”, both the length and height are right at the 5” mark but the width was reduced to ~2”. All of the hardware to mount the sight to a bow was designed to match the dimensions of a store-bought sight and allows it to be universally used on a wide range of compound bows. When shooting the bow using the sight, zeroing the bow was quick and easy with three screws for adjustment and should hold zero once mounted.

My final contributions to this project were the organization of written elements and helping with necessary testing.

- Garrett Gill
Student# 2735113

My design contributions for the Automatic Range Finding Bow Sight project were writing up the code to handle the linear interpolation of data from the table of ranges, speed, and pin sight movement, as well as consolidating everyone's individual together into one working code. I also designed the schematics and routed the PCB, as well as soldered all the components onto the board. The goal of our project was to allow a user to enter the speed that their bow shoots their chosen arrow weight, then allow them to obtain the range to their target using an onboard rangefinder. Once both the speed and range were obtained, the data was used to adjust the sight pin to the appropriate bullseye location.

The lookup table was handled using arrays. One array contained six values at increments of 10 starting at 0. This was used for the distance in yards for our lookup table. Nine arrays were needed to cover the nine different bow speed options that we had obtained data for. These arrays contained the data on how far the arrow would drop in inches, depending on the distance. The last group of nine arrays were needed to reflect how many microseconds were needed in the PWM signal to step the motor the correct amount. As for the actual linear interpolation code, the speed of the arrow set is used to assign the correct pin sight array and motor position array through the use of case statements. Then, the code runs through the distance array comparing the distance measured from the range finder to main distance array to what values in the group of arrays to use. The code then runs a standard linear interpolation to obtain the arrow drop based on the pin sight location array and how far to step the motor based on the motor position array. As for consolidating the codes, it was mostly just making sure variables matched up, but also creating functions to call for each button press to make reading the code easier.

The schematics were done in EAGLE. Any parts not found in the library were typically obtained via Mouser. The circuit boards were printed from OSH Park. All of the surface mount parts were soldered using solder paste and then heating them in a reflow oven. The through hole parts were soldered by hand. A few errors were made visible only after populating the board. Some were just components being one package size off of the part available, which was an easy fix, while other errors were only resolvable via sky wiring and cutting traces. All troubleshooting of PCB errors was done entirely by me and I was the sole person to verify that any of the systems on the PCB worked as intended.

- Cory Verba
Student# 3014329

Table of Contents

List of Figures

List of Tables

Abstract:

With the growing popularity of archery, a system designed to help an archer achieve the best shot possible becomes necessary. The Automatic Range Finding Bow Sight is a system that consists of a single pin sight controlled by a servo motor. The servo motor controls the sight pin by rotating a drive wheel that will move the pin up or down. The sight pin movement is based on a reading obtained by a built-in rangefinder and speed data input by the user. This data will then be compared to programmed lookup tables in order to find the appropriate pin position. After the sight pin has been adjusted, the user should be confident they will hit their target.

- Rangefinder will quickly obtain distance when triggered
- The servo motor will accurately adjust the sight pin up or down
- Interpolation between lookup table rows will ensure accurate movements

Authors: DD, GG, DK, CV

Needs Statement:

According to Randy Ulmer, an author for Peterson's Bow Hunting Magazine, the use of laser rangefinders is becoming more commonly used in order to position and gauge distances as a guideline for when a target comes into sight. In the article, Ulmer describes the difficulties of sighting a bow through the use of the human eye, increases at distance, elevation change, aiming cross ravines or canyons, and aiming in low light situations. The above situations could provide the difference between a kill shot and unnecessarily injuring an animal. A solution that could automatically position the sight upon receiving the input signal from the rangefinder would be helpful at ensuring a more accurate sight without the use of multiple devices or worrying about errors due to distance, elevation, or low light [1].

Authors: DD, GG, DK, CV

Objective Paragraph:

The objective of this project is to design and prototype a bow sight that will automatically account for the distance of a target. The device will need to be able to accurately adjust the pin sight within a few seconds after acquiring the distance to the target. This will prevent the need for multiple tools and reduce unnecessary error. The device will be weatherproof, meaning it will be able to withstand rain, snow and temperatures ranging from -30° F to 150°F so that it can be used for varying hunting seasons and in different global locations. The device will need to be able to withstand the shock from the shot of the arrow. The device will also need to be compact, no larger than 5"x5"x5" in order to not add too much weight or obstruct the user's view. The sight will also need to be able to mount to the original sight location on the bow.

Authors: DD, GG, DK, CV

Research Survey:

With the growing popularity of bow hunting, the need for additional tools such as a rangefinder in conjunction with the mounted sighting system is becoming more common. The purpose of the proposed model is to provide a quick and optimal solution to the sighting system of a compound bow used for hunting by combining two separate tools into one device that can be activated before each shot. The model would allow the rangefinder to be integrated into the sighting system itself, therefore allowing the sight to measure the distance for the user. By utilizing a single pin sight, the system will have the ability to adjust the sight based on the measured distance. This sight system will eliminate any need to "sight in" the bow, saving the user a large amount of time. Having a sight that moves according to the calculated arrow flight path will lead to a more accurate shot placement reducing the chance of unnecessarily maiming the target.

Limitations of current technology are that it requires the use of a manual rangefinder and one of several different types of optics to be "sighted in" or manually adjusted to user defined distances to account for the projectile drop of the arrow. According to Bob Robb, the three most common types of sights used for hunting are the fixed-pin, moveable-pin and pendulum sights. Fixed-pin sights have several pins that can be adjusted to specific distances. This in turn makes it easy to quickly adjust to one of the other pins if the target is moving. The movable-pin sight is a sight with a single pin that can be adjusted for different distances. The proposed concept plans to utilize this type of sight and move the pin electronically using a motor. The pendulum sight is best used by tree-standers because it employs the use of a hinge allowing the sight to swing freely to compensate for the height of the tree stand [2].

All three of the above sights require that the user "sight" them in to specific user defined distances before use. According to Todd Kuhn from Outdoor Life, the normal process of "sighting in" a bow begins by, "starting at 20 yards and moving back 1 yard with each successive shot." This is done until the arrow no longer strikes the target area. He then states, "Write this distance down. Now you have a range of yardages at which your 20-yard pin is capable of delivering an arrow. Repeat this process for each pin" [3]. This method requires the user to take multiple test shots to learn what the drop of the arrow is and then adjust the sight pins to specific shot distances. This takes a lot of the user's time and requires that every test shot be perfect to get the pins adjusted properly.

Another problem with the current technology is that the distance between the user and the target is often difficult to determine. This requires the user to either judge the distance by eye or use a distance measuring tool such as a rangefinder. In a hunting application, it quickly becomes a rather tedious task to measure the distance to the target, put down the rangefinder, pick up the bow, aim, and then shoot, and often there is not enough time to go through this entire process. Therefore, most archers opt to judge the distance by eye which often leads to an improperly placed shot.

An existing technology that allows the rangefinder to be mounted directly to the sight system is the Dead-On Range Finder. This device attaches a manual rangefinder into the outer hoop of the multi-pin sight eliminating the need for a separate rangefinder. While reducing the time to find the range of your shot, the Dead-On Rangefinder still requires that the sight be "sighted in" prior to use. The use of this device also requires the user to be able to line up a "belly line" and a "back line" in order to measure the distance to the target which could be difficult if the target is not broadside or if the target is moving. Manual measurement means increased chance of error due to relying on the user's judgement [4].

While plenty of technology exists today that uses parts of our concept, none have managed to do everything proposed. Most sight systems use multiple pin sights to determine where to aim depending on distances. One example of this is patent US6634111B2 filed by Paul M. Lorroco for "Multiple pin sight for an archery bow" [5]. This patent is for ranged sights having at least two pin sights to assist in aiming. The proposed concept will have a single pin, which will be adjusted through the use of a servo motor based on the data read back from the rangefinder. The pin sight adjusting automatically sets this concept apart from others. As mentioned previously, the Dead-On Range Finder is a rangefinder made to go on a sight, but the user still has to "sight in" beforehand and it does not adjust the pin sights depending on the distance.

Another system that was designed was the Auto-Correcting Bow Sight [6]. This system was designed by Timothy M. Gorsuch and James A. Buckley and had the ability to "perform situation-specific aim evaluations and corrections to correct or compensate for situation-specific shooting and environmental factors, at a given time and on a per-shot basis” [6]. This system utilized multiple measurement devices, such as a rangefinder, an inclinometer, and an anemometer to calculate the arrow’s flight path. An electronic, translucent display would then illuminate with an LED dot to be used as the sight indicator. The dot would then be adjusted to compensate for the distance of the target, angle of the shot, and windage. Unfortunately, the product was abandoned in 2012 before being marketed. This system operates in a very similar manner to the proposed model with one major difference. The model proposed will utilize a mechanical sight system that will be programmed to adjust the single pin sight on the y-axis to compensate for the calculated arrow flight path rather than using an electronic displayed sight.

“Laser bow sight apparatus” filed by Blair J. Zykan and Jeremy G. Dunne is one of the closest patents to the proposed design, except it utilizes a multiple pin sight. The “Laser bow sight apparatus” has a rangefinder that utilizes a laser to determine the distance to the target by reading the light reflected from the laser. This information is then processed by a central processing unit to determine which sight pin LEDs the system will light up [7]. The design will use similar technology to determine distance but instead will use the processed information to power a motor and adjust the single pin sight accordingly.

Another system that measures distance electronically and adjusts where to aim is the ArcAid system which is intended to aid an amateur archer in hitting their target with each shot. “The Arc Aid interface shows you how to shoot to hit the target by providing continuous feedback on your actions. In this way, the user will be able to learn how to master the bow in a step-by-step way.” This system uses multiple sensors to monitor the angle of the bow, the tension on the string and the distance to the target. By combining the feedback from these sensors, the ArcAid is able to display “two arcade style bars indicate the angle of the bow and the force on the string. Both bars and the bullseye on the smartphone are divided in color zones to give the user visual feedback of where the arrow will probably hit. As with other (arcade) games you have to try to get everything into the green zone when launching for a perfect shot” [8]. Similarly, to the ArcAid, the proposed device will also have to be programmable so that the bow’s arrow speed and arrow weight can be entered so that the projectile drop can be accurately accounted for based on the distance to the target. Where this device will differ is it will be purposed as a hunting device and not as a training device. Use as a hunting device will mean the proposed device will need to be rugged as well as capable of mounting to a variety of compound bows. The ability to be universally mounted will require the device to contain all of the necessary sensors and the sight in a compact design that will remain as unobtrusive to the overall use of the bow as possible.

Despite being thousands of years old, bow hunting is still extremely popular and continues to see growth today. This growth realizes the need for additional equipment and tools and as such complex and user-friendly sighting systems are becoming more and more popular while traditional multi pin sighting systems are becoming outdated. The designed system employs the use of existing technology to not only save time in sighting in the bow but also to help hunters shoot more accurately in the field to help prevent maiming the target. The system consists of a rangefinder integrated into a single pin sight that will automatically adjust the position of the pin based on data received from the rangefinder. Technology similar to the proposed system exists but none have successfully done what this system intends to do. Other technology is limited in various ways. Robustness, the need to sight in and the continued use of multi pin sights are all areas where other systems have failed. This design intends to not only solve these problems, but also become a marketable and reliable system.

Authors: DD, GG, DK, CV

Marketing Requirements:

- Must be easily programmable to the user's settings.
- Mounting must be convenient and non-obstructive.
- Must be robust and durable.
- Must have a power storage device capable of multiple uses.
- Must quickly and accurately measure distance and adjust sight position
- Sight must be easily seen in low light conditions.

Authors: DD, GG, DK, CV

Objective Tree:

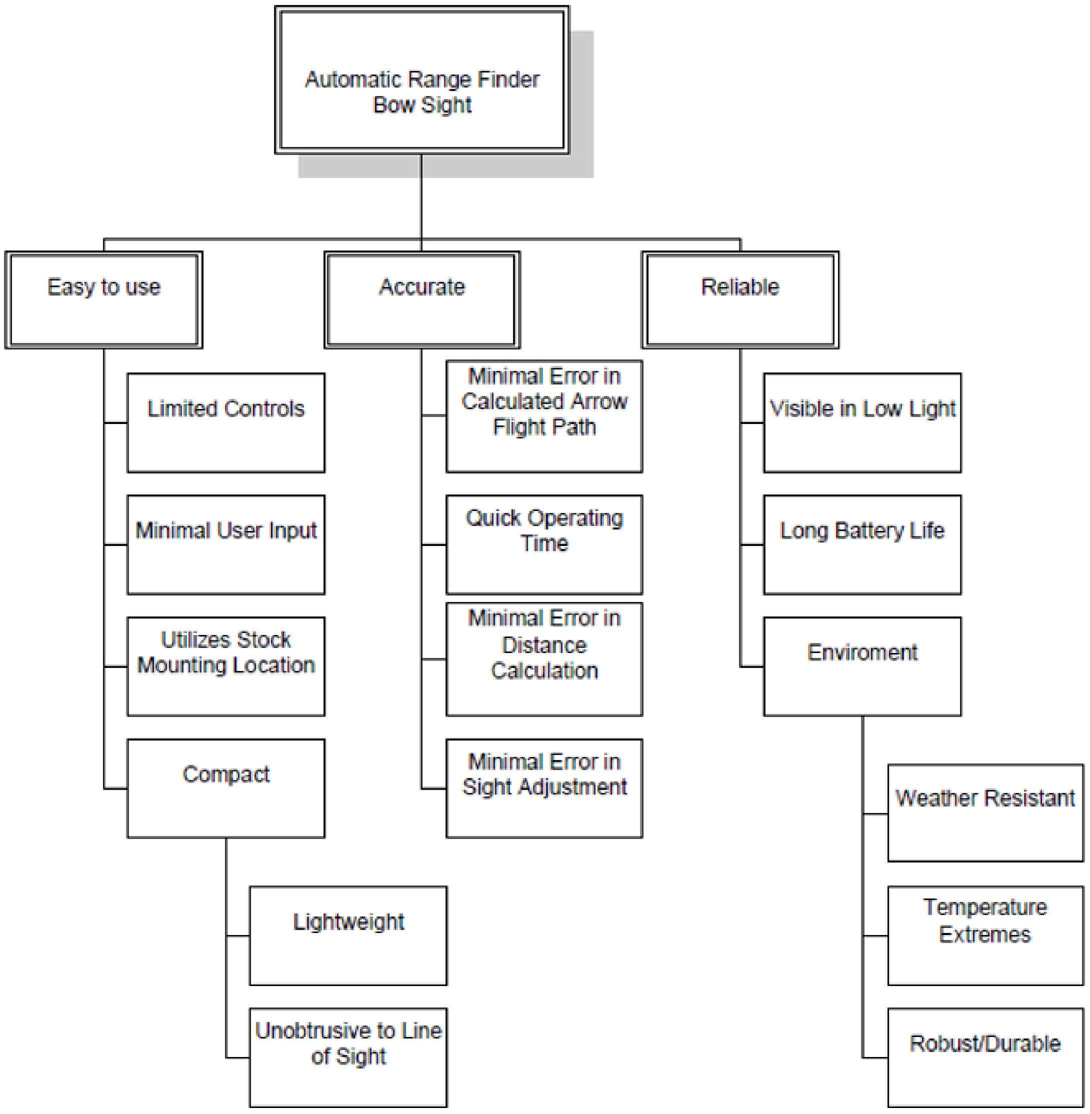

Automatic Range Finder Bow Sight
Easy to use
Accurate
Reliable
Limited Controls
Minimal User Input
Utilizes Stock Mounting Location
Compact
Lightweight
Unobtrusive to Line of Sight
Minimal Error in Calculated Arrow Flight Path
Quick Operating Time
Minimal Error in Distance Calculation
Minimal Error in Sight Adjustment
Visible in Low Light
Long Battery Life
Enviroment
Weather Resistant
Temperature Extremes
Robust/Durable

Figure 1: Objective Tree

The Objective Tree shows that the three main objectives that need to be accomplished are ease of use, accuracy of the rangefinder and sight adjustment, and overall reliability. Under easy to use, some minor objectives would be making sure the rangefinder has limited controls, requires minimal user input, utilizes the existing stock mounting location on the bow, and is compact. The focus on compactness would be the rangefinder being lightweight and not be obtrusive to the user's line of sight. Under accurate, the minor objectives would be minimizing the error in the sight adjustment, the error in distance calculation and the error in the arrow's flight path, as well as making sure the rangefinder still has a quick operating time. Under reliable, the minor objectives would be making sure the rangefinder can have a long battery life, is visible in low light, and can withstand the environment, such as remaining operable despite extreme temperatures and any kind of weather.

Authors: DD, GG, DK, CV

Engineering Requirements:

Engineering Requirements		
Requirement	**Justification**	**Marketing Requirement**
Rangefinder must be accurate within ±2.5 yards of target distance.	This is necessary to accurately predict the trajectory of the arrow.	5
The rangefinder should be accurate between 10 and 50 yards.	This is the range necessary to ensure that the rangefinder is accurate for the entire range of sight pin adjustment.	5
The sight pin should adjust between 15 to 45 yards.	This is the standard range an archer may shoot from.	5
Sight pin needs to display to the user the position to hit the target within ±6 inches.	This will give the user the best chance of accurately hitting the target.	5
The sight movement will be divided into a minimum of 50 increments.	This will allow the sight pin to be adjusted by 1 mm with each position.	5
The sight must utilize the stock bow sight mounting location.	This allows the sight to be mounted to a variety of bows as an aftermarket attachment.	2
User interface must be backlit.	This will allow for use in low light conditions like dusk or dawn.	6
User interface must be easy to use and intuitive	Limited user interface will provide minimal error from the user.	1
Must hold enough power to be used for 3 hours including sleep time.	This will allow it to last the duration of a typical hunting trip.	4
Rangefinder and sight pin need to complete their operations within 10 seconds.	This will allow the user to line up the sight right before taking the shot and reduce the chance of the target moving.	5
The sight must be durable enough to withstand the adverse conditions that can be experienced while hunting.	Will need to be able to withstand weather changes and being bumped against other items.	3
Marketing Requirements: • Must be easily programmable to the user's settings. • Mounting must be convenient and non-obstructive. • Must be robust and durable. • Must have a power storage device capable of multiple uses. • Must quickly and accurately measure distance and adjust sight position • Sight must be easily seen in low light conditions.		

Authors: DD, GG, DK, CV

Engineering Analysis:

Speed of the Bow: The speed of the bow will be calculated with the use of a chronometer. This device will measure the time that it took the arrow to pass between two sensors at a specific distance and calculate the speed of the arrow. Using a chronometer will ensure that we are not relying on the average speed given by the bow manufacturer, but instead using the actual speed with the users desired arrow weight.

Distance to Target: For the engineering requirement of ±2.5 yards of target distance, a range finding sensor will be used that will provide input to the processor for further calculations.

Range of Sight Pin Movement: The range of the sight pin will be the distance from the bottom of the sight ring to the top of the sight ring. Assume this distance is variable **D**, which for this design will be 2 inches or 50.8 mm.

Sight Pin Resolution: The variation in sight pin position was found by measuring the sight pin positions of a currently sighted in compound bow. The measured bow shot arrows at 340 feet per second and contained 3 sight pins that were sighted in at distances of 20, 30, and 40 yards. An arrow being shot from this bow was found to drop 9 inches from 20 yards to 30 yards. From 30 to 40 yards, the arrow was found to drop 11 inches. Therefore, the average drop of an arrow can be found to be about 10 inches per 10 yards equating to a drop of about 1 inch per yard. Thus, a linear approximation of the sight pin position can be used to maintain ±6" accuracy out to 45 yards. The vertical distance between the 20 and 30 yard pins was measured and found to be approximately 8.7 mm and the distance between the 30 and 40 yard pins was found to be approximately 10.3 mm. The average between these two measurements equates to roughly 9.5 mm. Therefore, the change in vertical position of the sight pin equates to approximately 0.95 mm per yard. This value can be considered the pin movement resolution factor and is represented as variable **X**. This computation will be repeated to find the pin movement resolution factor for various bow speeds.

Resolution of Servo Required: The number of positions the motor needs to have in order to travel the whole range of the sight pins movement can be calculated as shown in Eq.1 assuming **N** is the variable for number of locations necessary within 180 degrees of rotation.

$$N = 2 * \left({}^{D}\!/_{X}\right) \tag{1}$$

Size of the Drive Wheel: The drive wheel attached to the motor will have to have a circumference equal to two times **D**. This will ensure that when the motor rotates 180 degrees, the sight pin moves from the bottom to the top of the sight ring. The radius of the wheel can be found using Eq.2.

$$r = {}^{D}\!/_{\pi} \tag{2}$$

Interpolation Between Two Positions: A linear interpolation will need to be performed on the distances in the lookup table to calculate the correct pin sight location. **Y** is the calculated pin sight location. **X** is the distance measured from the rangefinder. $\mathbf{X_1}$ is the closest distance on the lookup table that is less than **X** and $\mathbf{X_2}$ is the closest distance on the lookup table that is greater than **X**. $\mathbf{Y_2}$ is the pin sight position that corresponds to the distance $\mathbf{X_2}$ and $\mathbf{Y_1}$ is the pin sight position that corresponds to $\mathbf{X_1}$

$$y = y_1 + (x - x_1)\frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{3}$$

Torque will be assumed negligible

Table 1: Calculations and conversions for engineering analysis

Attribute	Variable	Formula	Value	Unit	Value	Unit
RF Accuracy	R	------	2.5	yds	2.286	m
Speed of Light	C	------	3.28E+08	yds/sec	3.00E+08	m/s
Frequency	F	1/S	6.56E+07	Hz	6.56E+01	MHz
Distance	D	------	2	in	50.8	mm
Resolution	X	------	0.0393701	in	1	mm
Positions	N	2*(D/X)	101.5999451	locations	101.5999451	locations
Radius	r	D/pi	0.6366197731	in	16.17014224	in
Diameter	d	2*r	1.273239546	in	32.34028447	in

Energy Storage Device Power: The power required of the energy storage device will be determined based on the max draws by the processor, rangefinding emitter and receiver, the motor and LCD display. The calculations related to power are shown in Table 2.

Amp Hours:

$$\text{Ah} = 3\text{hours} * 0.602\text{A} = 1.806\text{Ah} \tag{4}$$

Spec over by ~20%:

$$1.806\text{A}/0.8 = 2.25\text{Ah} \tag{5}$$

Table 2: Power Calculations

Component	V	Max I (A)	Min I (A)	Max P (W)	Min P (W)
R.F. Emitter & Receiver	5.25	0.33	0.09	1.7325	0.4725
Processor	3.3	0.1	0.00000005	0.33	0.000000165
LCD Display	2.4	0.0079	0.0038	0.01896	0.00912
	5.5	0.0141	0.0069	0.07755	0.03795
Servo motor	6	0.15	0.008	0.9	0.048
MCP73830	6	0.000015	0.0000005	0.00009	0.000003
Switching Regulator	5	0.00003	0.000001	0.00015	0.000005
		Total:		Total:	
		0.602	A	3.05925	W

Pin Position Calculation: Using data collected from a bow with a speed of 340fps and comparing it to an online ballistics' calculator, to help ensure both sources are accurate, the drop patterns of various bow speeds were determined and collected. This data was then graphed to visually confirm the assumptions that bows with higher speeds would drop less over a given distance. This was confirmed.

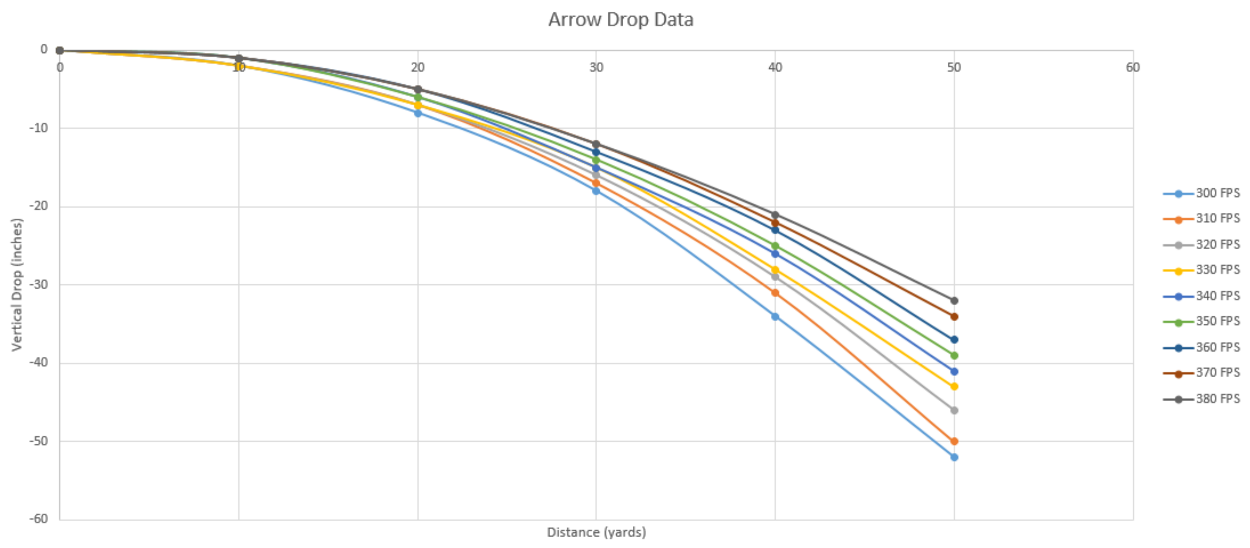


Figure 2: Arrow drop relative to bow speed and distance

Next the data was used to calculate the appropriate pin positions for the sight system. Correct motor positions were calculated in the following table. Using the speed and drop data, the necessary angle of release to compensate for projectile drop was calculated for each speed and distance in 10fps and 10 yard increments. The pin sight will be physically zeroed at the 15 yard distance for a given bow. This, was used to calculate the height adjustment from the ground at the target, from the zeroed position and scaled back to determine the distance the motor must move from the zeroed position. This distance was converted into mm and then into microsecond adjustments to be used by the motor in the form of a pulse width modulated signal with (M/20,000)*100% duty cycle.

Table 3: Arrow drop conversion to motor positions

	Drop [in]	Dist. [yds]	Dist. [in]	Angle of Release	Height Adjust from Ground	Zero Height Comp.	Adjusted Height	Height Adjust [mm]	Motor Position
	Drop [in]	Dist. [yds]	Dist. [in]	Θ	X [in]	X0	X'	X'[mm]	M [us]
300	5	15	540	0.009258	0.268518	0.268518	0	0	2300
300	8	20	720	0.011110	0.322222	0.268518	0.053703	1.364074	2252
300	18	30	1080	0.016665	0.483333	0.268518	0.214814	5.456296	2107
300	34	40	1440	0.023606	0.684722	0.268518	0.416203	10.57157	1925
300	52	50	1800	0.028880	0.837777	0.268518	0.569259	14.45918	1788
310	4.5	15	540	0.00833	0.241666	0.241666	0	0	2300
310	7	20	720	0.009721	0.281944	0.241666	0.040277	1.023055	2264
310	17	30	1080	0.015739	0.456481	0.241666	0.214814	5.456296	2107
310	31	40	1440	0.021524	0.624305	0.241666	0.382638	9.719027	1956
310	50	50	1800	0.027770	0.805555	0.241666	0.563888	14.32277	1793
320	4.5	15	540	0.00833	0.241666	0.241666	0	0	2300
320	7	20	720	0.009721	0.281944	0.241666	0.040277	1.023055	2264
320	16	30	1080	0.014813	0.42962	0.241666	0.187962	4.774259	2131
320	29	40	1440	0.020136	0.584027	0.241666	0.342361	8.695972	1992
320	46	50	1800	0.025549	0.741111	0.241666	0.499444	12.68588	1851
330	4.5	15	540	0.00833	0.241666	0.241666	0	0	2300
330	7	20	720	0.009721	0.281944	0.241666	0.040277	1.023055	2264
330	15	30	1080	0.013887	0.402777	0.241666	0.161111	4.092222	2155
330	28	40	1440	0.019441	0.563888	0.241666	0.322222	8.184444	2010
330	43	50	1800	0.023884	0.692777	0.241666	0.451111	11.45822	1894
340	3.5	15	540	0.006481	0.187962	0.187962	0	0	2300
340	6	20	720	0.00833	0.241666	0.187962	0.053703	1.364074	2252
340	15	30	1080	0.01388	0.402777	0.187962	0.214814	5.456296	2107
340	26	40	1440	0.018053	0.523611	0.187962	0.335648	8.525462	1998
340	41	50	1800	0.02277	0.660555	0.187962	0.472592	12.00385	1875
350	3.5	15	540	0.006481	0.187962	0.187962	0	0	2300
350	6	20	720	0.00833	0.241666	0.187962	0.053703	1.364074	2252

350	14	30	1080	0.012962	0.375925	0.187962	0.187962	4.774259	2131
350	25	40	1440	0.017359	0.503472	0.187962	0.315509	8.013935	2016
350	39	50	1800	0.021663	0.628333	0.187962	0.44037	11.18540	1904
360	3	15	540	0.005555	0.161111	0.161111	0	0	2300
360	5	20	720	0.006944	0.201388	0.161111	0.040277	1.023055	2264
360	13	30	1080	0.012036	0.349074	0.161111	0.187962	4.774259	2131
360	23	40	1440	0.015970	0.463194	0.161111	0.302083	7.672916	2028
360	37	50	1800	0.020552	0.596111	0.161111	0.435	11.049	1909
370	3	15	540	0.005555	0.161111	0.161111	0	0	2300
370	5	20	720	0.006944	0.201388	0.161111	0.040277	1.023055	2264
370	12	30	1080	0.011110	0.322222	0.161111	0.161111	4.092222	2155
370	22	40	1440	0.015276	0.443055	0.161111	0.281944	7.161388	2046
370	34	50	1800	0.018886	0.547777	0.161111	0.386666	9.821333	1952
380	3	15	540	0.005555	0.161111	0.161111	0	0	2300
380	5	20	720	0.006944	0.201388	0.161111	0.040277	1.023055	2264
380	12	30	1080	0.011110	0.322222	0.161111	0.161111	4.092222	2155
380	21	40	1440	0.01458	0.422916	0.161111	0.261805	6.649861	2064
380	32	50	1800	0.017775	0.515555	0.161111	0.354444	9.002888	1981

Authors: DD, GG, DK, CV

Level 0 Hardware Diagram:

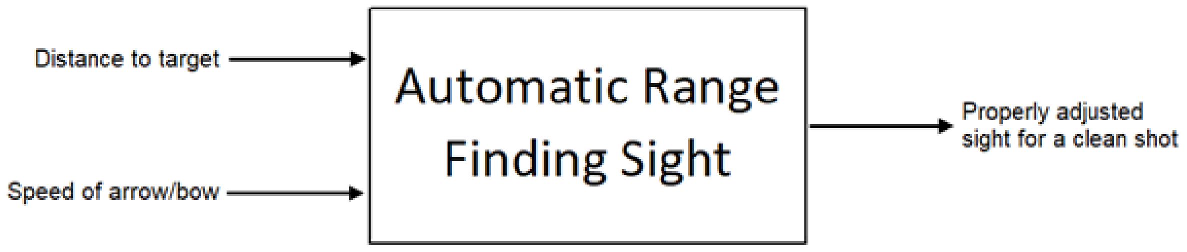

Distance to target
Speed of arrow/bow
Automatic Range Finding Sight
Properly adjusted sight for a clean shot

Figure 3: Level 0 Hardware Block Diagram

The above figure is a basic Level 0 diagram of our concept. The main idea is that two inputs, distance to target, and the speed of the arrow on the users bow, will enter our proposed system and the output will be the sight properly adjusting for a clean shot.

Authors: DD, GG, DK, CV

Level 1 Hardware Diagram:

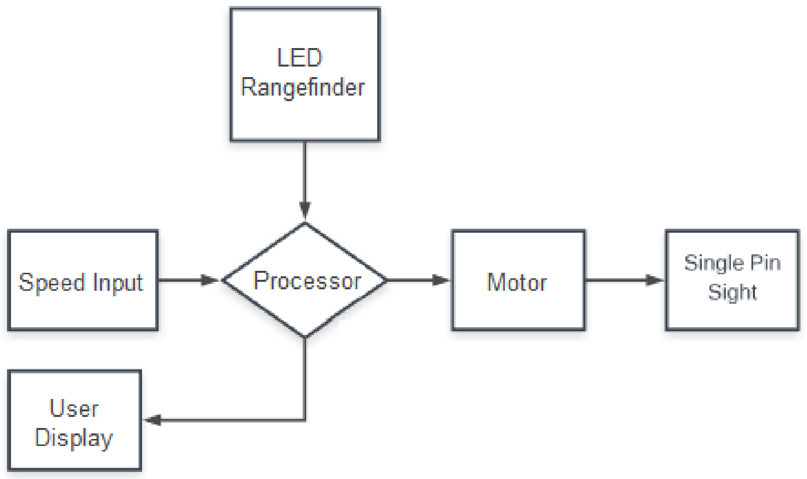


Figure 4: Level 1 Hardware Block Diagram

The above figure shows a more detailed Level 1 block diagram used to represent the system being proposed. The sight will work using two inputs, one provided by the operator and the other by triggering the rangefinder. Once the two inputs have been gathered the processor will then gather the current position of the motor and run the program to calculate the appropriate sight adjustment. Once the appropriate sight position is calculated the sight pin will be moved by the motor to provide the perfect shot.

Authors: DD, GG, DK, CV

Level 2 Hardware Diagram:

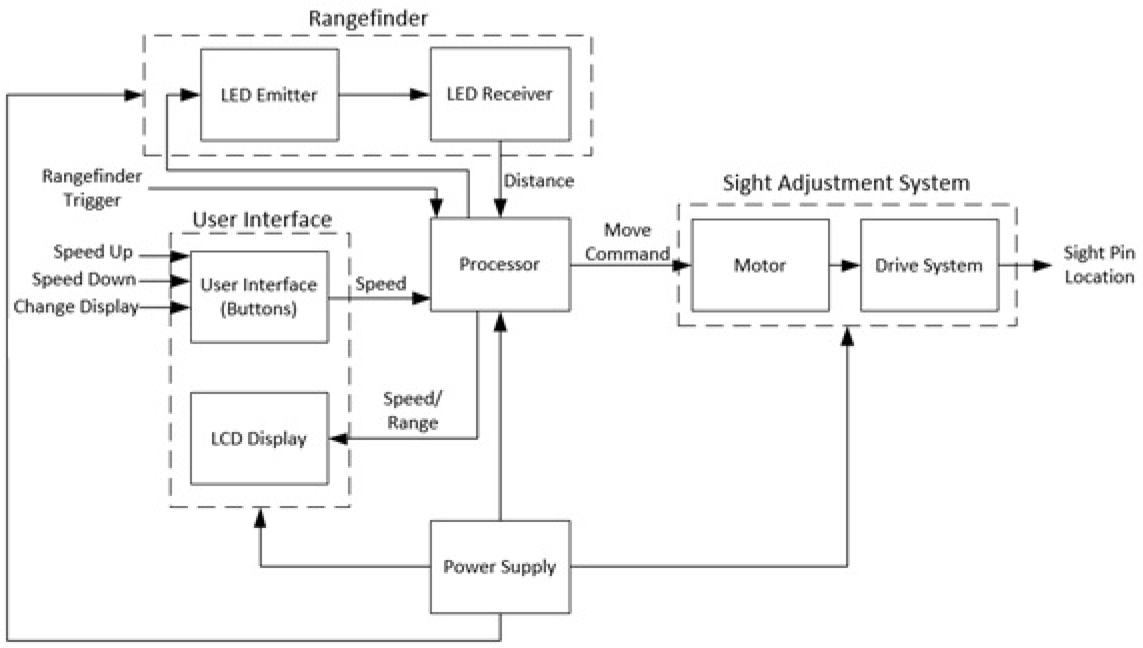


Figure 5: Level 2 Hardware Block Diagram

The above figure shows a more detailed Level 2 block diagram used to represent the system being proposed. This diagram breaks down the rangefinder, sight adjustment system and user interface into separate components. For the rangefinder, a trigger signal controlled by the user will send a signal to the processor, then the emitter projects a beam of light to the target which reflects and is captured by the receiver module. The time between the emission and reception of the signal is be calculated and then passed to the processor as the acquired distance. The sight adjustment system consists of three components – the motor, drive system and the sight pin. The motor will receive a move command (either up or down) from the processor and will begin to rotate the drive system as necessary. The drive system will then move the sight pin to the necessary location for the perfect shot. The user interface will consist of several buttons for user input and a backlit display. The buttons will be used to adjust the speed of the bow up or down, to trigger the rangefinder and to toggle through the displays. The display will be used to show the speed being input to the user and also return the range obtained from the rangefinder.

Author: GG

Table 4: Hardware Diagram User Interface functional requirements

Module	User Interface
Designer	DT03
Inputs	- Display signal from processor
Outputs	- Bow speed increase - Bow speed decrease - Change display
Functionality	Receives inputs from the processor and displays correct speed of the bow. Sends outputs to the processor to modify bow speed. Must be durable, visible in low light conditions and have resistance to various weather conditions.

Table 5: Hardware Diagram User Interface (Buttons) functional requirements

Module	User Interface (Buttons)
Designer	DT03
Inputs	- Increase bow speed set point - Decrease bow speed set point - Scroll Displays - Trigger rangefinder
Outputs	- Bow speed increase - Bow speed decrease - Cycle to next display - Calculate distance of target
Functionality	Receives inputs from the user to increase or decrease bow speed, or trigger the rangefinder to obtain a distance. Sends the input from the user to the processor to make the necessary adjustments.

Table 6: Hardware Diagram LCD Display functional requirements

Module	LCD Display
Designer	DT03
Inputs	- Speed set point from processor - Distance obtained from processor
Outputs	- Speed value - Distance value
Functionality	Receives the user inputs from the processor and displays them.

Table 7: Hardware Diagram Rangefinder functional requirements

Module	Rangefinder
Designer	DT03
Inputs	- Pulse/Control signal - Receive light beam return time
Outputs	- Distance based on laser return time
Functionality	Receives inputs from the processor and emits and receives a beam of light. Transmits light received information to the processor. System must be capable of at least a minimum of 15 yards and maximum of 45 yards.

Table 8: Hardware Diagram Rangefinder Emitter functional requirements

Module	Rangefinder Emitter
Designer	DT03
Inputs	- Signal from user trigger
Outputs	- Focused LED light signal
Functionality	Receives inputs from the user and sends light signal out towards target. Emitter must be capable of at least a minimum of 15 yards and maximum of 45 yards.

Table 9: Hardware Diagram Rangefinder Receiver functional requirements

Module	Rangefinder Receiver
Designer	DT03
Inputs	- Reflected infrared light signal
Outputs	- Distance based on signal return time
Functionality	Receives reflected light signal that has returned from the target. Determines distance based on the time between the signal being sent and the signal being received.

Table 10: Hardware Diagram Motor and Sight functional requirements

Module	Motor and Sight
Designer	DT03
Inputs	- Motor control signal (Steps or Location)
Outputs	- Motor position to processor - Drive wheel to adjust pin sight position
Functionality	Receives inputs from the processor and moves motor accordingly and through use of a drive wheel, moves the pin sight into the correct position. Processor stores the last position the motor was sent to for reference during next position change.

Table 11: Hardware Diagram Servo Motor functional requirements

Module	Servo Motor
Designer	DT03
Inputs	- Motor control signal (Steps or Location)
Outputs	- Drive wheel to adjust pin sight position
Functionality	Receives inputs from the processor and moves motor accordingly and through use of a drive wheel, moves the pin sight into the correct position.

Table 12: Hardware Diagram Drive System functional requirements

Module	Drive System
Designer	DT03
Inputs	- Motor turning output
Outputs	- Sight pin adjustment
Functionality	Converts the motor’s turns into vertical movement to adjust the sight pin up or down.

Table 13: Hardware Diagram Processor functional requirements

Module	Processor
Designer	DT03
Inputs	- Bow speed increase (UI) - Bow speed decrease (UI) - Bow speed save/enter (UI) - Laser received signal (Rangefinder) - Motor feedback (Motor) - Power (Undecided DC voltage)
Outputs	- UI display - Motor adjustments - Laser control/pulse
Functionality	Receive inputs from various parts of the system, perform calculations on said systems and deliver the appropriate output signals to the UI and motor. Must function quickly enough to utilize the rangefinder signal to calculate distance to target within ± 1 yard.

Table 14: Hardware Diagram Power Supply functional requirements

Module	Power Supply
Designer	DT03
Inputs	- Power Switch - 3.7 V Lithium Ion Battery - DC-DC Converter
Outputs	- 5 V with at least 1.8Ah of charge
Functionality	Provide power to various components and step the voltage up to 5 V

<u>Level 1 Software Diagram:</u>

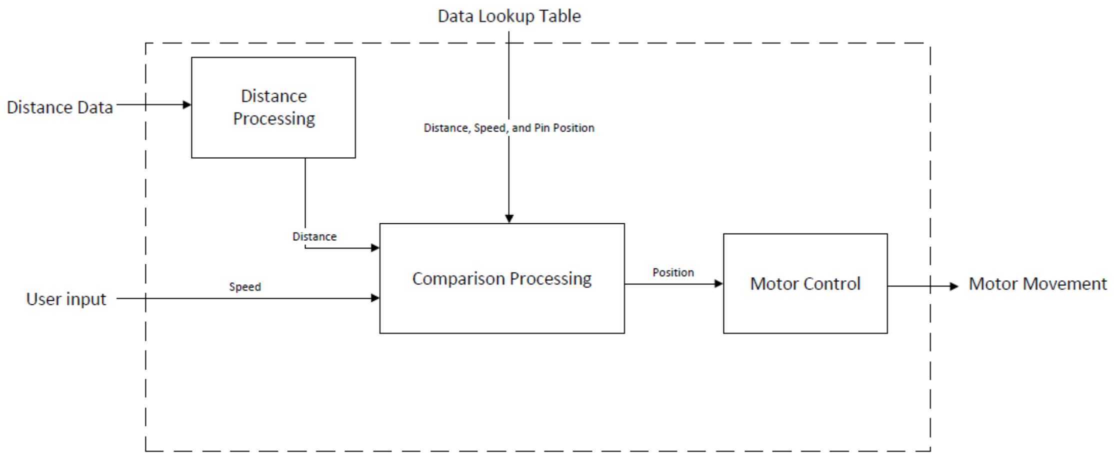


Figure 6: Level 1 Software Block Diagram

The above figure shows a more detailed Level 1 block diagram used to represent the software being proposed. This system will use a Distance Processing system to gather the distance data from the laser rangefinder before sending the distance signal to the Comparison Processor. The Comparison Processing system will then analyze the speed input and the distance to find the appropriate sight pin location in the lookup table. Next, the position found by the Comparison Processor will be passed to the Motor Control which will adjust the pin sight to the appropriate position.

Authors: DK, CV

Level 2 Software Diagram:

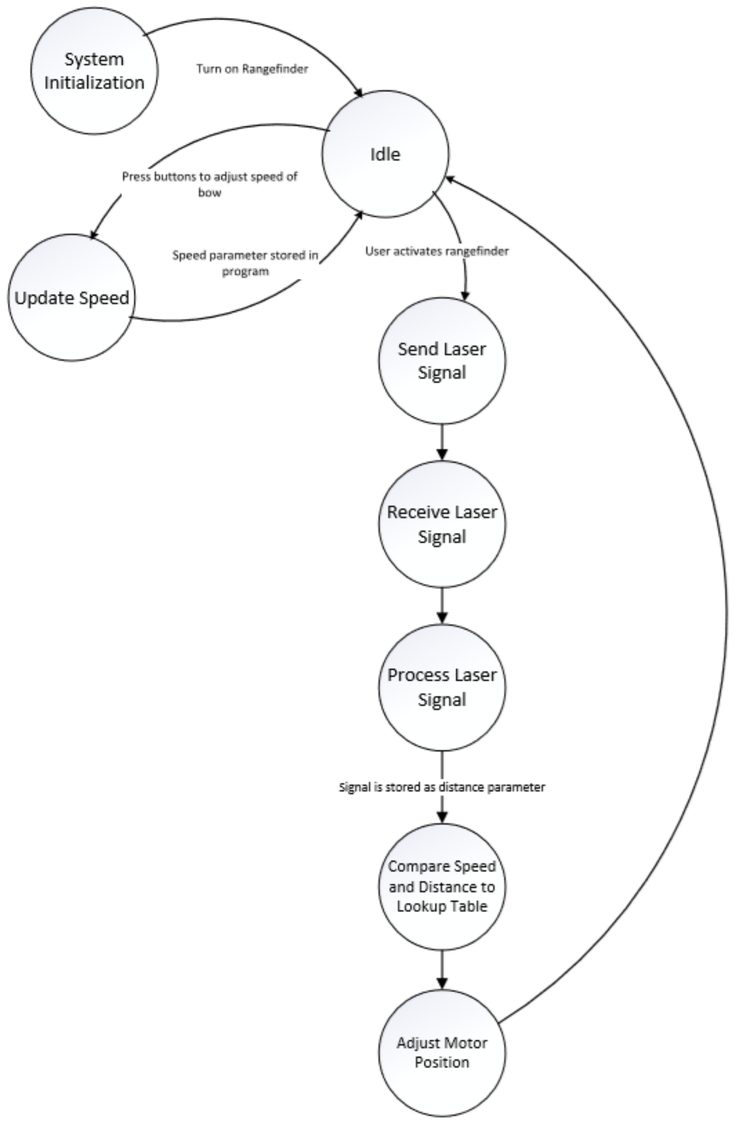

System Initialization
Turn on Rangefinder
Idle
Press buttons to adjust speed of bow
Update Speed
Speed parameter stored in program
User activates rangefinder
Send Laser Signal
Receive Laser Signal
Process Laser Signal
Signal is stored as distance parameter
Compare Speed and Distance to Lookup Table
Adjust Motor Position

Figure 7: Level 2 Software Diagram

The above figure shows a more detailed Level 2 State diagram used to represent the software being proposed. The program starts by initializing all of the variables and functions. The program will then sit idle until one of two things occur. If the operator presses a button to adjust the speed of the bow, the program will have to process that change and update the variable used for speed. Then, it goes back to idling until either the speed is changed again, or the operator presses the button to trigger the rangefinder. The program will send a LED light signal out and wait for it to return. When the signal returns, it is processed into useable data that is stored as the distance parameter. Then, the program looks at a distance vs pin sight lookup table that is relevant to the speed programmed. The pin sight position is determined by interpolating between the two closest table entries for pin sight distance. The program will signal to the motors to move the correct number of steps that correspond to the pin sight distance calculated. Finally, the system goes back to idle until either the operator presses the button again or changes the speed of the bow.

Authors: CV

Table 15: Software Diagram Comparison Processing functional requirements

Module	Comparison Processing
Designer	DT03
Inputs	-Distance -Speed -Data pulls from Lookup Table
Outputs	-Position
Functionality	Compares inputs to data table and averages information as needed to calculate an accurate position.

Table 16: Software Diagram Distance Processing functional requirements

Module	Distance Processing
Designer	DT03
Inputs	-Distance Data
Outputs	-Distance
Functionality	Interprets distance data and outputs the information as data which can be read.

Table 17: Software Diagram Motor Control functional requirements

Module	Motor Control
Designer	DT03
Inputs	-Position
Outputs	-Motor Movement
Functionality	Moves motor to correct position.

User Interface:

The user interface of the Automatic Rangefinding Bow Sight consists of a 16x2 LCD display, 4 pushbuttons and a power on/off switch. The LCD display is used to display to the user the speed setpoint they have entered and the distance returned by the rangefinder. The 4 pushbuttons will be used to increase speed, decrease speed, toggle through the displays and trigger the rangefinder. The buttons that increase and decrease speed allow the user to raise or lower the bow speed at set increments of 5. This increment was chosen because standard speeds given for bows are given in increments of 5 and when the user measures the speed of their bow and arrow combination, the speed obtained should be in a similar increment. The rangefinder trigger button will be used to collect a distance reading and will be mounted near the shooters hand. The last button that will be a part of the user interface is the button that will be used to toggle through the various displays. This will allow the user to view their current speed setting and distance the pin sight is set at. The last component of the user interface is a master on/off switch that will allow the user to power the system on or off when necessary.

Authors: GG

Drive System:

The drive system will consist of a servo motor, drive wheel, cable system, free spinning top wheel, and a single pin sight. The servo motor will function as the driving force behind the system. It will be bidirectional and should be accurate to 1mm increment changes. The servo motor will be controlled through the processor and powered by the high and low voltage rails. To accomplish the accuracy needed, a motor with many positions should be utilized. The difference in the sight pin’s minimum and maximum positions should be roughly 2 inches. To achieve this, the motor will need a minimum of about 100 positions in 180 degrees of rotation (or half of a rotation). To ensure that the system will maintain accuracy and hold the correct position during movement, the drive cable will be attached to the drive wheel. This means only half of the drive wheel will actually be used, hence the need for 100 positions per half rotation. Once the motor receives a movement signal from the processor, the motor will apply movement to the drive wheel, which will rotate the cable about the free spinning wheel and move the pin into the correct position. The motor will then shut down and have a certain amount of holding torque to help prevent unintentional movement.

Authors: DK

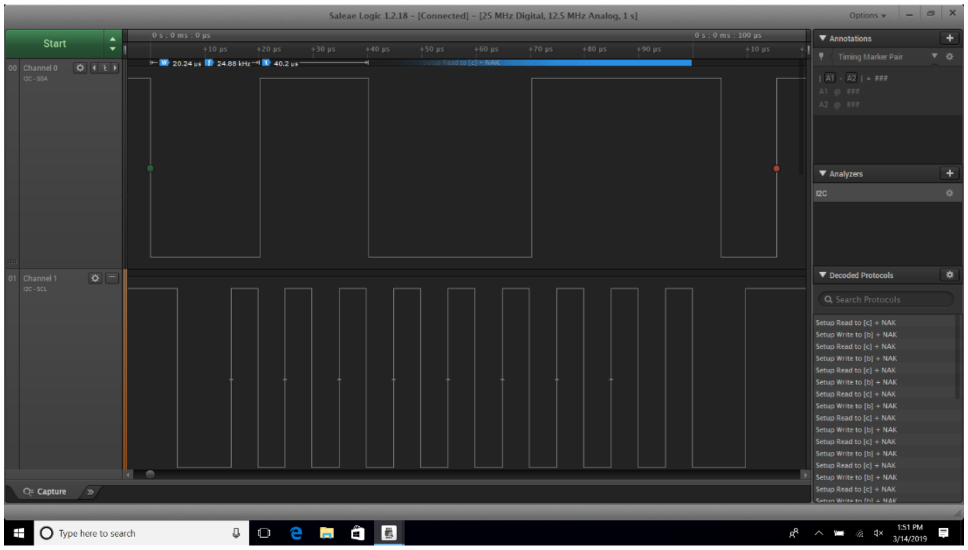
Rangefinder:

The rangefinding system of the Automatic Rangefinding Bow Sight is a crucial aspect of the project. Without an accurate distance measurement, the sight system would be unable to dictate the proper location to set the pin sight up for a precise shot placement. Therefore, the basic operation of the rangefinder is to measure the distance from the user to the target. To accomplish this task, a time of flight sensor is utilized. A time of flight sensor operates by emitting a beam of light, capturing the reflected light, and measuring the time it took for the beam to travel from the sensor to the target, and back. The beam of light is projected over a linearly increasing area to allow the sensor to more accurately determine the distance in the event that something slightly obstructs the target. The distance of the target is then calculated by dividing the total time by two and then multiplying the resulting value by the speed of light in yards per second. This data is then sent to the processor to be utilized.

The specific model of time of flight sensor chosen for this project is the TeraRanger Evo 60m sensor. This sensor uses focused LEDs to project a beam of light to the target and an optic system to capture the reflected beam. To initiate the range finding sequence, a button mounted to the handle of the bow and is pressed by the user. The button press signal will be sent to the processor. The processor then communicates with the sensor to begin the distance measurement. The range finding function within the TeraRanger is then executed multiple times. Any bad readings or outlying data is ignored, and an average of the most pertinent data is calculated. This is be done to reduce the likelihood of encountering an error in the distance measurement. Once the distance of the target has been found, the processor then begins calculating the necessary location of the pin sight.

The I2C communication protocol is used to send data between the PIC and the TeraRanger sensor and proves to be the most difficult aspect of the rangefinder design. In order for the sensor to send data to microcontroller, the I2C communication protocol must be initialized and started. Once initiated the PIC sends two bytes to the sensor to begin the data transfer. The first byte sent (0x62), is the 7-bit address followed by the write bit ‘0’ initializing the “Trigger Reading” command. Immediately following, the actual “Trigger Reading” command (0x00) is sent to the sensor. The I2C communication is then stopped and after a wait of 400 microseconds is restarted. The byte (0x63) is then sent to the sensor. This byte includes the 7-bit address followed by the write bit ‘1’, thus telling the sensor to begin transmitting data back to the microcontroller. Immediately, the microprocessor executes three “Get Byte” commands, reading the two distance bytes and the checksum byte sent back from the sensor. The I2C communication is then stopped after the data transfer is complete.

During the design process, issues arose with the transmission of data between the TeraRanger and the PIC. To alleviate these issues, the Saleae Logic analyzer was connected to an Arduino microcontroller that operated properly. The first byte of the communication waveform from the Arduino simulation was then captured and can be seen below in Figure 8.



```
Setup Read to [c] + NAK
Setup Write to [b] + NAK
Setup Read to [c] + NAK
Setup Write to [b] + NAK
Setup Read to [c] + NAK
Setup Write to [b] + NAK
Setup Read to [c] + NAK
Setup Write to [b] + NAK
Setup Read to [c] + NAK
Setup Write to [b] + NAK
Setup Read to [c] + NAK
Setup Write to [b] + NAK
Setup Read to [c] + NAK
Setup Write to [b] + NAK
Setup Read to [c] + NAK
```

Figure 8: Arduino Simulation – First Byte

After analyzing the Arduino's waveform, it was found that the delay between commands in the PIC microcontroller were too short. Being too short, this did not allow the data from each command to clear out of the processor before the next command was executed, thus leading to corrupted data. Once the delays were set properly through a series of trial and error, the first data byte was able to be sent properly from the PIC to the sensor and seen below in Figure 9.

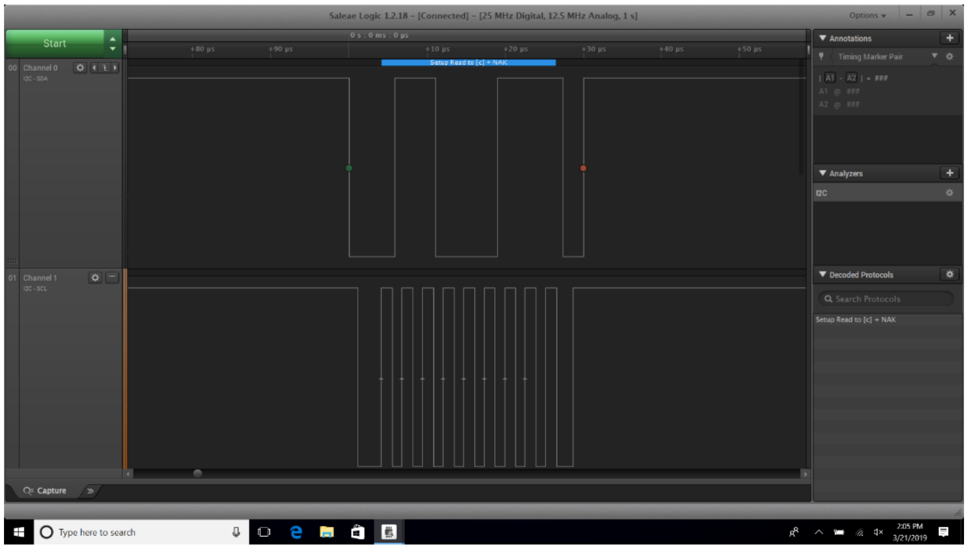
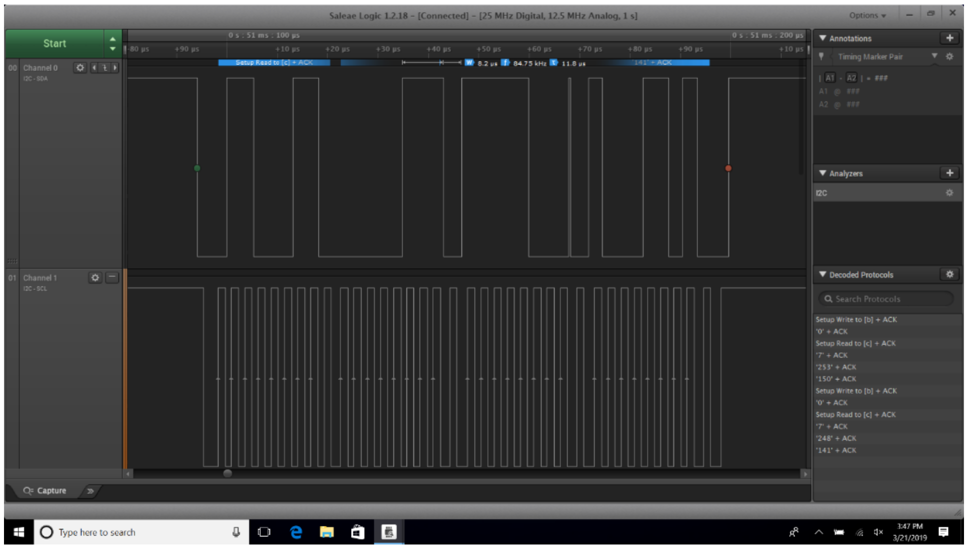


Figure 9: PIC Simulation – First Byte

Once the delays were set properly, the entire sequence as described above was able to be executed. This was verified by connecting the Saleae Logic Analyzer between the PIC and TeraRanger and viewing the send byte command followed by the three byte reads. Looking at Figure 10, one can verify that the first byte is the “Trigger Reading” command (0x63) being sent to the TeraRanger. The second and third bytes, are the distance data bytes that are sent from the TeraRanger back to the PIC and equates to the distance measured by the sensor from the desktop to the ceiling. This was found to be 2040 millimeters (0x07F8). The fourth byte shown below is the CRC8 checksum byte (0x8D) sent from the sensor to the microcontroller.


Saleae Logic 1.2.18 - [Connected] - [25 MHz Digital, 12.5 MHz Analog, 1 s]
Start
Channel 0
Channel 1
Annotations
Analyzers
I2C
Decoded Protocols
Capture

```
Setup Write to [b] + ACK
'0' + ACK
Setup Read to [c] + ACK
'7' + ACK
'253' + ACK
'150' + ACK
Setup Write to [b] + ACK
'0' + ACK
Setup Read to [c] + ACK
'7' + ACK
'248' + ACK
'141' + ACK
```

Figure 10: PIC Simulation – Trigger Reading Followed by Three Byte Reads

Once the communication issues between the PIC and TeraRanger were resolved, the distance data was formatted and converted to yards so that is could be referenced by the lookup tables.

Authors: DD

Battery Management:

The battery management system will consist of three primary parts. A lithium battery to power the system, a charging circuit to charge the lithium battery, and a DC to DC boost converter to step the low voltage of the battery up to a usable 5V for the rest of the system.

The battery chosen is a 3.7V lithium ion battery that is capable of 2Ah. This battery was chosen because it has the capability to run the system at full load for the specified 3-hour duration. The total draw of the system will be approximately 1.806Ah. This allows the battery to power the system while never fully draining the battery, avoiding the potential of ruining the cell.

The charge circuit for this system will be designed around a Microchip MCP73830 battery management controller. This device will provide the necessary charge algorithms for the single cell lithium ion battery and will also regulate the charge voltage to 4.2V ±0.75%. The charge circuit designed will be used in conjunction with a 500mA micro-USB wall charger.

Most components in the system are powered off of 5 volts, therefore, in order to use the 3.7V battery in this system, a DC to DC boost converter is required to step the voltage up to the required voltage. A DC switching regulator will be used to step the voltage up and a voltage regulator circuit will then be added after the converter to ensure that the output voltage remains at a constant 5VDC.

Authors: DK, GG

Processor:

The processor that was selected was the PIC24FJ128GA010 General Purpose 128 KB Flash Microcontroller. This is the same processor used in ESI labs. Already having experience using it made it a great choice when having to choose processors. This PIC24 has all the features needed – UART, PWM, flash memory, sleep mode, and plenty of I/O pins, further solidifying the choice to pick this processor. The Integrated Development Environment that will be used will be MPLAB X IDE. This is Microchip's own IDE to be used on their microcontrollers. It has plenty of features to help debug any code related problems encountered during design. Ultimately, the only requirements for the processor was to have all of the specifications we needed and since this PIC24 managed to cover everything, it turned out to be the best option to use.

Authors: CV

Microcontroller Code:

```
#include "config.h"
#include "buttons.h"
#include <stdio.h>
#include "xc.h"


char SPDchar[8], DISTchar[8], ARRchar[8]; // Create an array for 8 characters
int displaycount = 1; // Display Number for switch statement
double speed = 340;
int speed1 = 340;
float distance = 40;
unsigned int distancemm;
float distancearray[] = { 0, 10, 20, 30, 40, 50 };
float pinsightlocation[6];
float motorposition[6];
float dx, dy,dy2, arrowdrop, motorsteps;

    unsigned int Byte1;
    unsigned int Byte2;
    unsigned int Byte3;
    unsigned int HighByteDist;
    unsigned int LowByteDist;
    unsigned int CheckSumByte;


float case300[6] = { 0, -2, -8, -18, -34, -52 };
float case310[6] = { 0, -2, -7, -17, -31, -50 };
float case320[6] = { 0, -2, -7, -16, -29, -46 };
float case330[6] = { 0, -2, -7, -15, -28, -43 };
float case340[6] = { 0, -1, -6, -15, -26, -41};
float case350[6] = { 0, -1, -6, -14, -25, -39};
float case360[6] = { 0, -1, -5, -13, -23, -37};
float case370[6] = { 0, -1, -5, -12, -22, -34};
float case380[6] = { 0, -1, -5, -12, -21, -32};

float case300steps[6]= { 2300 , 2300 , 2251, 2106, 1925, 1787};
float case310steps[6]= { 2300 , 2300 , 2263, 2106, 1955, 1792};
float case320steps[6]= { 2300 , 2300 , 2263, 2130, 1991, 1850};
float case330steps[6]= { 2300 , 2300 , 2263, 2155, 2010, 1894};
float case340steps[6]= { 2300 , 2300 , 2251, 2106, 1997, 1874};
float case350steps[6]= { 2300 , 2300 , 2251, 2130, 2016, 1903};
float case360steps[6]= { 2300 , 2300 , 2263, 2130, 2028, 1908};
float case370steps[6]= { 2300 , 2300 , 2263, 2155, 2046, 1952};
float case380steps[6]= { 2300 , 2300 , 2263, 2155, 2064, 1981};
```

```
void ms_delay(int N) {
   T1CON = 0x8030; // Timer Enabled with prescale of 256
   TMR1 = 0; //CLR Timer1
   while (TMR1 < 62.5 * N) //Timer1 incrementing
   {
   }
}

void us_delay(int us){
   T1CON = 0x8010;
   TMR1 = 0;                  // Clear Timer 1
   while(TMR1<(2*us)){}       // 2 TMR1 counts per ?

}

void InitPMP(void) {
   // PMP initialization. See my notes in Sec 13 PMP of Fam. Ref. Manual
   PMCON = 0x8303; // Following Fig. 13-34. Text says 0x83BF (it works) *
   PMMODE = 0x03FF; // Master Mode 1. 8-bit data, long waits.
   PMAEN = 0x0001; // PMA0 enabled
}

void InitLCD(void) {
   // PMP is in Master Mode 1, simply by writing to PMDIN1 the PMP takes care
   // of the 3 control signals so as to write to the LCD.
   PMADDR = 0; // PMA0 physically connected to RS, 0 select Control register
   PMDIN1 = 0b00111000; // 8-bit, 2 lines, 5X7. See Table 9.1 of text Function set
   ms_delay(1); // 1ms > 40us
   PMDIN1 = 0b00001100; // ON, cursor off, blink off
   ms_delay(1); // 1ms > 40us
   PMDIN1 = 0b00000001; // clear display
   ms_delay(2); // 2ms > 1.64ms
   PMDIN1 = 0b00000110; // increment cursor, no shift
   ms_delay(2); // 2ms > 1.64ms
} // InitLCD

char ReadLCD(int addr) {
   // As for dummy read, see 13.4.2, the first read has previous value in PMDIN1
   int dummy;
   while (PMMODEbits.BUSY); // wait for PMP to be available
   PMADDR = addr; // select the command address
   dummy = PMDIN1; // initiate a read cycle, dummy
   while (PMMODEbits.BUSY); // wait for PMP to be available
   return ( PMDIN1); // read the status register
} // ReadLCD
// In the following, addr = 0 -> access Control, addr = 1 -> access Data
#define BusyLCD() ReadLCD( 0) & 0x80 // D<7> = Busy Flag
#define AddrLCD() ReadLCD( 0) & 0x7F // Not actually used here
#define getLCD() ReadLCD( 1) // Not actually used here.

void WriteLCD(int addr, char c) {
   while (BusyLCD());
   while (PMMODEbits.BUSY); // wait for PMP to be available
   PMADDR = addr;
   PMDIN1 = c;
} // WriteLCD
// In the following, addr = 0 -> access Control, addr = 1 -> access Data
#define putLCD( d) WriteLCD( 1, (d))
#define CmdLCD( c) WriteLCD( 0, (c))
#define HomeLCD() WriteLCD( 0, 2) // See HD44780 instruction set in
#define ClrLCD() WriteLCD( 0, 1) // Table 9.1 of text book

void I2Cinit(int BRG) {
   I2C1BRG = BRG;              // See data sheet, Table 16.1 pg. 139
   while (I2C1STATbits.P);     // Check buss idle, see 5.1 of I2C document
                               // It works, not sure its needed.
   I2C1CONbits.A10M = 0;       // 7-bit address mode (Added 8-14-17)
   I2C1CONbits.I2CEN = 1;      // enable module
}

void I2CStart(void) {
   //us_delay(10);             // delay to be safe
   I2C1CONbits.SEN = 1;        // Initiate Start condition pg. 21 of I2C man
   while (I2C1CONbits.SEN);    // wait for Start condition complete sec. 5.1
   //us_delay(10);             // delay to be safe
}

void I2CStop(void) {
   //us_delay(12);             // delay to be safe
   I2C1CONbits.PEN = 1;        // see 5.5 pg. 27 of Microchip I2C manual
   while (I2C1CONbits.PEN);    // This is at hardware level, I suspect fast
   //us_delay(10);             // delay to be safe
}

void I2Csendbyte(char data) {
   while (I2C1STATbits.TBF);   // wait if buffer is full
   I2C1TRN = data;             // pass data to transmission register
   us_delay(23);               // delay to be safe
}
```

```
char I2Cgetbyte(void) {
   I2C1CONbits.RCEN = 1;       // Set RCEN, Enables I2C Receive mode
   while (!I2C1STATbits.RBF);  // wait for byte to shift into I2C1RCV register
   I2C1CONbits.ACKEN = 1;      // Master sends Acknowledge
   us_delay(2);                // delay to be safe
   return (I2C1RCV);
}


void putsLCD(char *s) {
   while (*s) putLCD(*s++); // See paragraph starting at bottom, pg. 87 text
} //putsLCD


void SetCursorAtLine(int i) {
   int a;
   if (i == 1) {
      CmdLCD(0x80); //Sets DRAM Address for HD44780 instruction set
      //To enable DRAM, hex 0x80 is translated to binary: 1000 0000

} else if (i == 2) {
   CmdLCD(0xC0);//Sets DRAM Address for HD44780 instruction set
   //To enable DRAM, hex 0xC0 is translated to binary: 1100 0000
} else {
   TRISA = 0x00; // hex for zero, sets PORTA<0:7> for output
   for (a = 1; a < 20; a++) //Loop to flash LEDs for 5 seconds at 2Hz
   {
      PORTA = 0xFF; //Turns LEDs on
      ms_delay(2000); //Delays half of the cycle // On For 2 Seconds
      PORTA = 0x00; //Turns LEDs off
      ms_delay(1000); //Delays half of the cycle // Off For 1 Seconds
   }
  }
}

void servoRotate()
{
   unsigned int i;
   for (i = 0; i < 50; i++) {  // 50Hz
      PORTAbits.RA4 = 1;      // RA4 High
      us_delay(motorsteps);           // Desired Position
      PORTAbits.RA4 = 0;
      us_delay(20000- motorsteps);
   }
}
```

```
void button1press(void){
   switch(displaycount)
   {
      case 1:
         ClrLCD(); // clear display
         ms_delay(5); // 2ms > 1.64ms
         SetCursorAtLine(1);
         putsLCD(" Auto Bow Sight "); // Puts message on first line of LCD
         SetCursorAtLine(2);
         putsLCD("  DD GG  DK CV  "); // Puts message on second line of LCD
         displaycount++;
         break;

      case 3:
         ClrLCD(); // clear display
         ms_delay(5); // 2ms > 1.64ms
         sprintf(DISTchar, "%0.1f", distance);
         ms_delay(32);
         SetCursorAtLine(1); //Set to first line.
         putsLCD("RANGE:"); //Floating Point Temperature on First line.
         SetCursorAtLine(2); //Set to start of second line
         putsLCD(DISTchar);
         putsLCD(" yards");
         displaycount=1;
         break;

      case 2:
         ClrLCD(); // clear display
         ms_delay(5); // 2ms > 1.64ms
         sprintf(SPDchar, "%0.1f", speed);
         ms_delay(32);
         SetCursorAtLine(1); //Set to first line.
         putsLCD("SPEED:"); //Floating Point Temperature on First line.
         SetCursorAtLine(2); //Set to start of second line
         ms_delay(5);
         sprintf(SPDchar, "%0.1f", speed);
         ms_delay(32);
         putsLCD(SPDchar);
         putsLCD(" FPS");
         displaycount++;
         break;
   }
}

void button3press (void)
{
   switch(displaycount-1)
   {
      case 2:
         if ((speed <=380) & (speed >=310))
            {
               speed = speed - 10;
               speed1=speed;
               ms_delay(32);
               ClrLCD();
               ms_delay(5); // 2ms > 1.64ms
               sprintf(SPDchar, "%0.1f", speed);
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("SPEED:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD(SPDchar);
               putsLCD(" FPS");
            }
            else
            {
               ClrLCD();
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("SPEED:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD("Error: Minimum");
               ms_delay(1000);
               ClrLCD();
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("SPEED:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD(SPDchar);
               putsLCD(" FPS");
            }
         break;

      case 0:
            ClrLCD();
            ms_delay(32);
            SetCursorAtLine(1);
            putsLCD("Distance:"); // Puts message on first line of LCD
            SetCursorAtLine(2);
            putsLCD("Error");
            ms_delay(1000);
            ClrLCD();
            ms_delay(32);
            SetCursorAtLine(1);
            putsLCD("Distance:"); // Puts message on first line of LCD
            SetCursorAtLine(2);
            putsLCD(DISTchar);
            putsLCD(" yards");

         break;


      case 1:
         ClrLCD();
         ms_delay(32);
         SetCursorAtLine(1);
         putsLCD(" Auto Bow Sight "); // Puts message on first line of LCD
         SetCursorAtLine(2);
         putsLCD("Error");
         ms_delay(1000);
         ClrLCD();
         ms_delay(32);
         SetCursorAtLine(1);
         putsLCD(" Auto Bow Sight "); // Puts message on first line of LCD
         SetCursorAtLine(2);
         putsLCD("  DD GG  DK CV  ");
         break;
   }
}

void button2press (void)
{
   switch(displaycount-1)
   {
      case 2:
         if ((speed <=370) & (speed >=300))
            {
               speed = speed + 10;
               speed1=speed;
               ms_delay(32);
               ClrLCD();
               ms_delay(5); // 2ms > 1.64ms
               sprintf(SPDchar, "%0.1f", speed);
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("SPEED:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD(SPDchar);
               putsLCD(" FPS");
            }
            else
            {
               ClrLCD();
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("SPEED:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD("Error: Maximum");
               ms_delay(1000);
               ClrLCD();
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("SPEED:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD(SPDchar);
               putsLCD(" FPS");
            }
         break;
      case 0:

         {
               ClrLCD();
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("Distance:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD("Error");
               ms_delay(1000);
               ClrLCD();
               ms_delay(32);
               SetCursorAtLine(1);
               putsLCD("Distance:"); // Puts message on first line of LCD
               SetCursorAtLine(2);
               putsLCD(DISTchar);
               putsLCD(" yards");
         }
            break;

      case 1:
             ClrLCD();
             ms_delay(32);
             SetCursorAtLine(1);
             putsLCD(" Auto Bow Sight "); // Puts message on first line of LCD
             SetCursorAtLine(2);
             putsLCD("Error");
             ms_delay(1000);
             ClrLCD();
             ms_delay(32);
             SetCursorAtLine(1);
             putsLCD(" Auto Bow Sight "); // Puts message on first line of LCD
             SetCursorAtLine(2);
             putsLCD("  DD GG  DK CV  ");
             break;
      }
}

void button4press (void)
{
   int i;
     switch (speed1)
   {

       case 300:
         for (i = 0; i < 6 - 1; i++)
         {
         pinsightlocation[i]=case300[i];
         motorposition[i] = case300steps[i];
         }
         break;

       case 310:
         for (i = 0; i < 6 - 1; i++)
         {
         pinsightlocation[i]=case310[i];
         motorposition[i] = case310steps[i];

         }
         break;

       case 320:
         for (i = 0; i < 6 - 1; i++)
         {
         pinsightlocation[i]=case320[i];
         motorposition[i] = case320steps[i];
         }
```

```
      break;

    case 330:

      for (i = 0; i < 6 - 1; i++)
      {
      pinsightlocation[i]=case330[i];
      motorposition[i] = case330steps[i];
      }
    break;


    case 340:
      for (i = 0; i < 6 - 1; i++)
      {
      pinsightlocation[i]=case340[i];
      motorposition[i] = case340steps[i];
      }
      break;

    case 350:
      for (i = 0; i < 6 - 1; i++)
      {
      pinsightlocation[i]=case350[i];
      motorposition[i] = case350steps[i];
      }
      break;

    case 360:
      for (i = 0; i < 6 - 1; i++)
      {
      pinsightlocation[i]=case360[i];
      motorposition[i] = case360steps[i];
      }
      break;

    case 370:
      for (i = 0; i < 6 - 1; i++)
      {
      pinsightlocation[i]=case370[i];
      motorposition[i] = case370steps[i];
      }
      break;

    case 380:
      for (i = 0; i < 6 - 1; i++)
```

```
        {
        pinsightlocation[i]=case380[i];
        motorposition[i] = case380steps[i];
        }
        break;

    }



      I2CStart();            // place module into start condition - send slave device address
      I2Csendbyte(0x62);     // Initiate Write Command via address
      I2Csendbyte(0x00);     // Initiate Write Command via address
      I2CStop();

      us_delay(400);

      I2CStart();
      I2Csendbyte(0x63);     // Trigger Reading Command
      Byte1 = I2Cgetbyte();   // Get Data Byte 1 - First Byte of Distance
      Byte2 = I2Cgetbyte();   // Get Data Byte 2 - Second Byte of Distance
      Byte3 = I2Cgetbyte();   // Get Data Byte 3 - Checksum Byte
      I2CStop();             // place module into stop condition

      ms_delay(50);          // 50 ms Delay

      I2CStart();            // place module into start condition - send slave device address
      I2Csendbyte(0x62);     // Initiate Write Command via address
      I2Csendbyte(0x00);     // Initiate Write Command via address
      I2CStop();

      us_delay(400);


      I2CStart();
      I2Csendbyte(0x63);     // Trigger Reading Command
      HighByteDist = I2Cgetbyte();   // Get Data Byte 1 - First Byte of Distance
      LowByteDist = I2Cgetbyte();   // Get Data Byte 2 - Second Byte of Distance
      CheckSumByte = I2Cgetbyte();   // Get Data Byte 3 - Checksum Byte
      I2CStop();             // place module into stop condition


      HighByteDist=0x008E;
      LowByteDist=0x00D0;
      distancemm = 256 * HighByteDist + LowByteDist;
```

```
     distance = distancemm/914.4;

     ms_delay(50);          // 50 ms Delay



/* number of elements in the array */
static const int count = sizeof (distancearray) / sizeof (distancearray[0]);



/* find i, such that distancearray[i] <= x < distancearray[i+1] */
for (i = 0; i < count - 1; i++)
  {
   if (distancearray[i + 1] > distance)
  {
   break;
  }
  }

/* interpolate */
   dx = distancearray[i + 1] - distancearray[i];
   dy = pinsightlocation[i + 1] - pinsightlocation[i];
   dy2 = motorposition[i + 1] - motorposition[i];
   arrowdrop = pinsightlocation[i] + (distance - distancearray[i]) * dy / dx;
   motorsteps = motorposition[i] + (distance - distancearray[i]) * dy2 / dx;



     ClrLCD();
     ms_delay(32);
     SetCursorAtLine(1);
     putsLCD("RANGE: "); // Puts message on first line of LCD
     sprintf(DISTchar, "%0.1f", distance);
     putsLCD(DISTchar);
     putsLCD(" yd");
     SetCursorAtLine(2);
     putsLCD("Drop: "); // Puts message on first line of LCD
     sprintf(ARRchar, "%0.1f", arrowdrop);
     putsLCD(ARRchar);
     putsLCD(" in"); // Puts message on first line of LCD



     servoRotate();
```

```
        ms_delay(1000);
        ms_delay(1000);
        ms_delay(1000);
        button1press();
        button1press();
        button1press();
}





main(void) {
//    char LCDchar[8]; // Create an array for 8 characters
//    int display = 0; // Display Number for switch statement
//    double speed = 340, distance = 0;
    T1CON = 0x8010;          // TMR1 on, Prescale 1:8, Tcy as clock
    T2CON = 0x8030;          // Timer 2 on, Prescale 1:256, Tcy as clock
    PORTAbits.RA4 = 0;       // Clear RA4
    TRISAbits.TRISA4 = 0;   // Set RA4 to Output

    I2Cinit(37);             // Initialize I2C module with 400kHz Baud Rate
    initButtons(0x0009);

    Byte1 = 1;
    Byte2 = 2;
    Byte3 = 3;
    HighByteDist = 0;
    LowByteDist = 0;
    CheckSumByte = 0;



    ms_delay(32); // At least 30ms for LCD Internal Initialization
    InitPMP(); // Initialize the Parallel Master Port
    InitLCD(); // Initialize the LCD

    SetCursorAtLine(1);
    putsLCD(" Auto Bow Sight "); // Puts message on first line of LCD
    SetCursorAtLine(2);
    putsLCD("  DD GG  DK CV  "); // Puts message on second line of LCD
    while (1)
       {
      if(getButton(0x0001))
      {
         button1press();
         ms_delay(300);
      }

      if (getButton(0x0002))
      {
         button2press();
         ms_delay(200);
      }

      if (getButton(0x0004))
      {
         button3press();
         ms_delay(200);
      }

      if (getButton(0x0008))
      {
         button4press();
         ms_delay(500);
      }


       }
} // main
```

The above code is currently a mix of both pseudocode and actual code. The code starts off by initializing all of the functions needed. This includes initializing the LCD screen, setting up the UART functions, and the millisecond and microsecond delays.

Inside the main function, the initialization functions are called, and the values for the motor and last recorded speed are read from flash memory. The processor then instructs the LCD to write the speed out. If none of the buttons are pressed within 10 seconds, an interrupt sequence starts which puts the MPU to sleep. The device is woken up whenever another input is issued. If the speed up button is pressed, the speed parameter is increased by five and then sent to the LCD screen. The value stored in the flash memory is also updated so if the rangefinder is powered off, it can remember what speed it was last at. If the speed down button is pressed, the speed parameter is decreased by five and that number is sent to the LCD screen. Once again, the value in the flash memory is updated.

The biggest function of the code is when the button to fire the LED is pressed. When this happens, the processor sends a signal via UART to the TeraRanger EVO. This causes the rangefinder to emit an LED signal onto a target and receive it as it bounces back. The rangefinder then stores that value in millimeters and must be sent back to the processor via UART. Once the measurement is processed, it will be divided by 1000 to convert the distance to meters since millimeters is too high of a resolution for use case.

Once the distance is recorded, the lookup tables are accessed. Each speed has its own separate lookup table for distance and pin position. A function is called that looks up the specific speed's look up table and finds the two closest distances to the distance measured. The related pin sight positions are then interpolated to find what position to move the pin sight. This information is then sent to the motor, which moves the pin sight into the correct location.

Authors: DK, CV

<u>Lookup Table:</u>

Table 18: Lookup Table Example

Bow Speed [mps/yps/fps]	Distance [m/yds/ft]	Pin Position [mm/cm/in]
X	15	a
X	25	b
X	35	c
X	45	d
Y	15	A’
Y	25	B’
Y	35	C’
Y	45	D’

The lookup tables used to adjust the sight will consist of 3 main columns – bow speed, distance to the target, and appropriate pin position. To determine the values of the table, various tests will be performed on existing bows of different speeds to collect data on their shooting characteristics. The speed of each bow will be determined by using a chronograph before operation and then input using the user interface. Once speed has been input to the system, each bow being tested will be fired from various distances (found by using the rangefinder) in order to observe their drop before hitting the target. Observing the drop of the arrow and where it hits the target will allow the sight adjustments to be fine-tuned. Once the lookup tables have been calculated, they will be programmed into the processor. To use the tables, the processor will find the given user speed in the table and the measured distance before finally sending the signal for pin position to the drive system. In cases where values fall between two rows, the processor will be able to interpolate the two rows of the table to ensure the accuracy of the system.

Authors: DK, CV

Mechanical Sketch:

Mount Sketch:

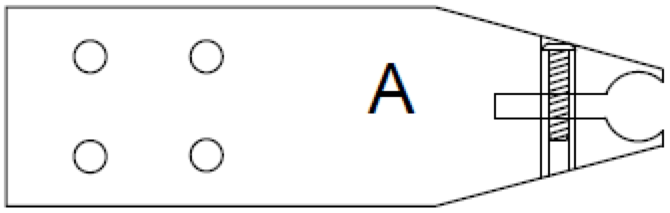


Figure 11: Mount Sketch of design

A. Mounting bracket designed to fit in the stock mounting location. On the right is where the sight will clamp onto the mount.

Front view of Control-Box and Sight:

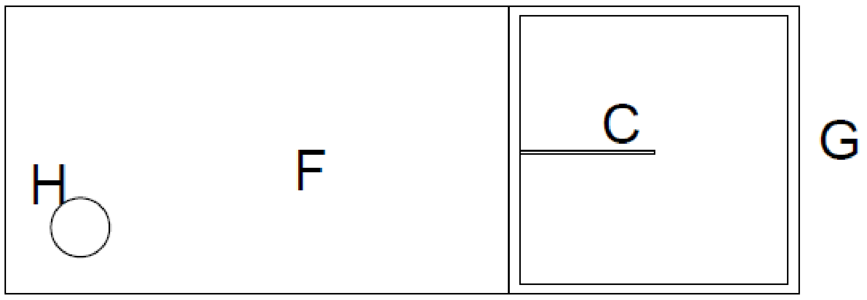


Figure 12: Front view of Control-Box and Sight

C. Single sight-pin that will be mechanically adjusted, up and down based on range.

F. Control box/Housing – contains the processor, rangefinder circuitry, drive motor and user interface display and buttons.

G. Sight retaining ring - serves as a guide for the sight-pin as it is adjusted up or down.

H. Rangefinder lens

Right Side View of Control-Box:

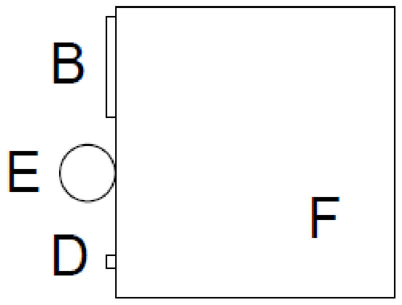


Figure 13: Right Side View of Control-Box

B. User interface display

D. User interface pushbuttons

E. Sight guide

F. Control-Box/Housing

Rearview of Control-Box and Sight:

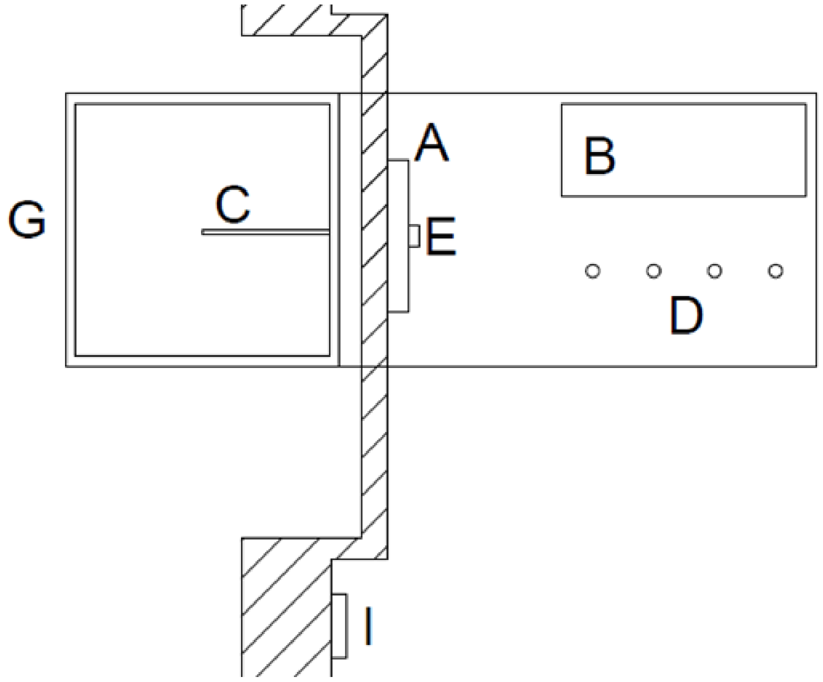


Figure 14: Rearview of Control-Box and Sight

A. Mount

B. User interface display - used to program in the speed of the bow and the weight of the arrow.

C. Pin-Sight

D. User interface pushbuttons - one will serve to select speed, one for activating the rangefinder, and the other two will be to adjust the values up or down.

E. Sight guide - attaches to the mount and allows the sight to be adjusted left or right as necessary.

G. Sight retaining ring

I. Rangefinder trigger – pushbutton that allows the user to trigger the rangefinder while aiming the bow.

Front View of Drive System

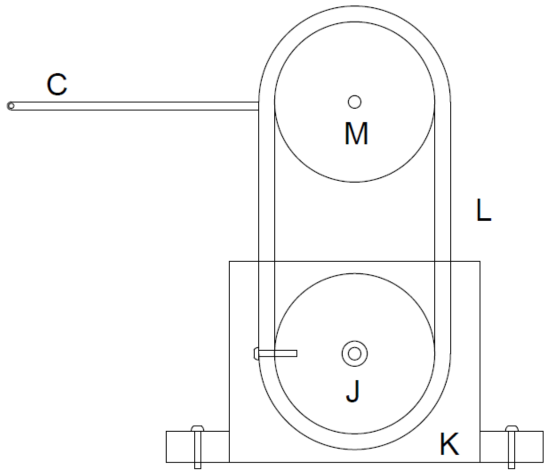


Figure 15: Front View of Drive System

C. Pin-Sight

J. Motor cable guide – cable guide attached to the motor spindle that the guide cable will mount to.

K. Servo motor – motor that will drive the pin-sight up or down based on necessary adjustment.

L. Sight pin guide cable – cable that the sight pin will be attached to.

M. Top cable guide – will guide the cable as it is moved by the motor.

Side View of Drive System

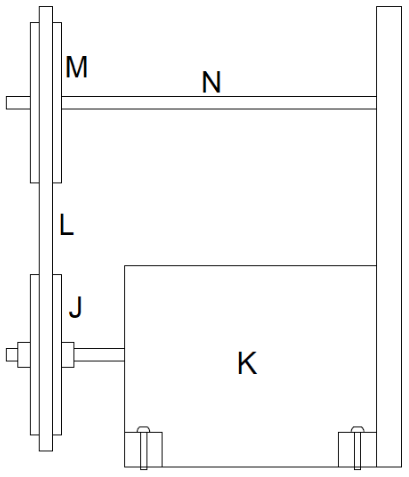


Figure 16: Side View of Drive System

J. Motor cable guide

K. Servo motor

L. Sight pin guide cable

M. Top cable guide

N. Top cable guide mount – holds the top cable guide to the housing.

Rendered View of Final Case Design

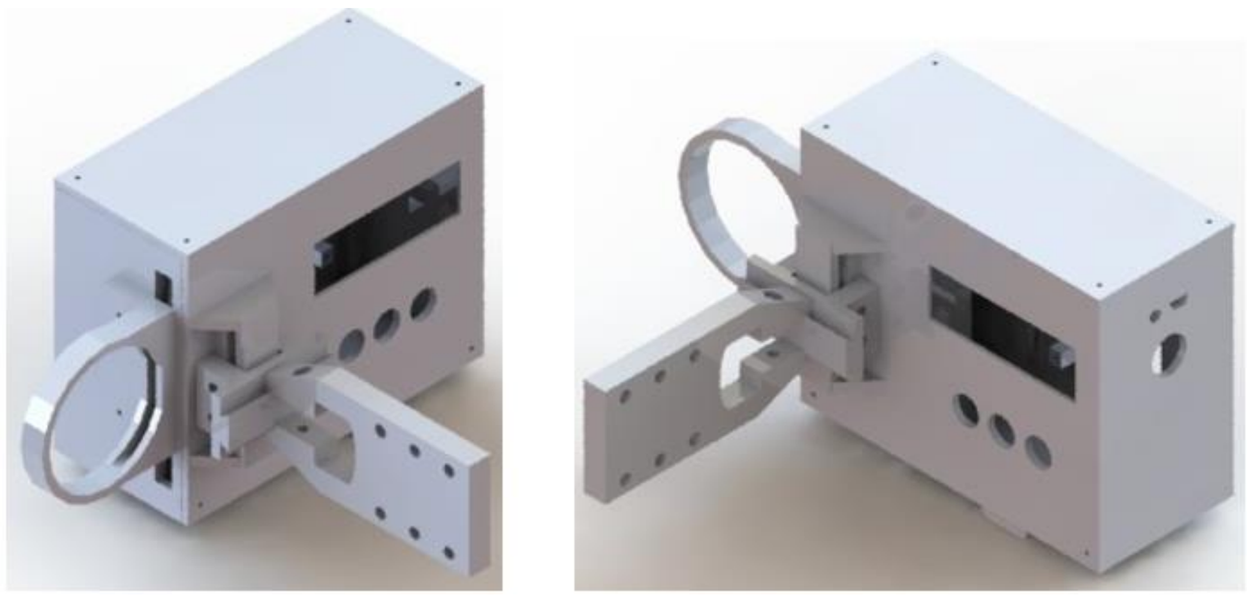


Figure 17: Rendered View of Final Case Design

Authors: GG, DK

Electrical Sketches:

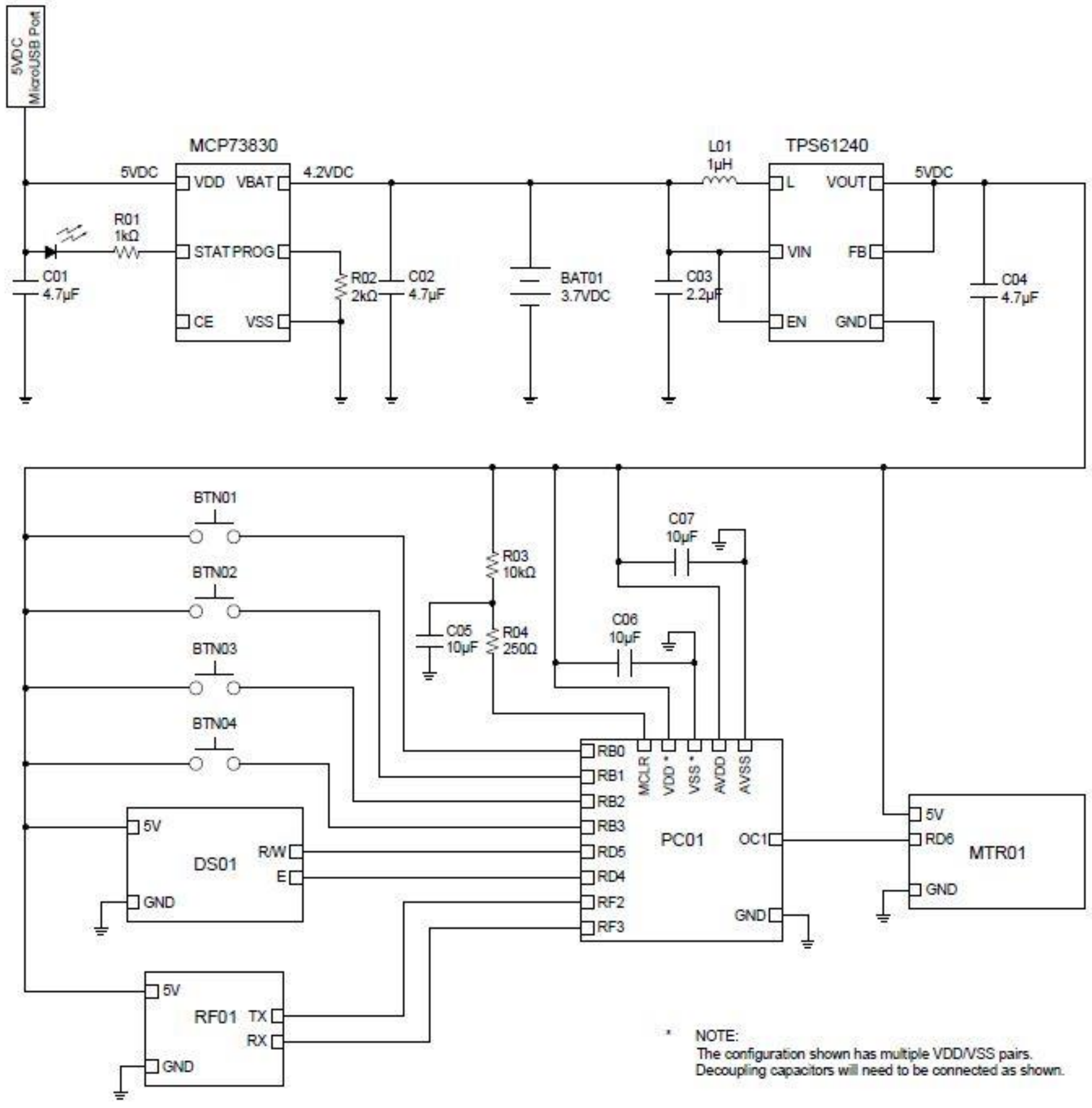


Figure 18: Electrical circuit sketch

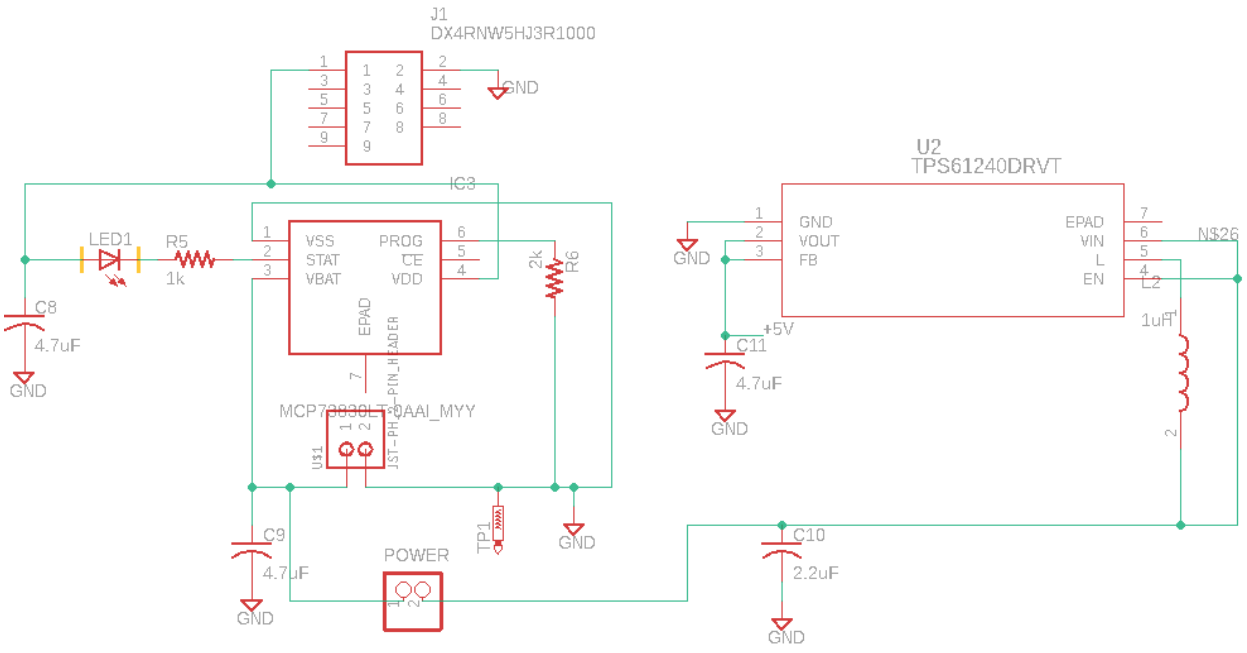


Figure 19: Electrical schematic for charging and DC-DC boost

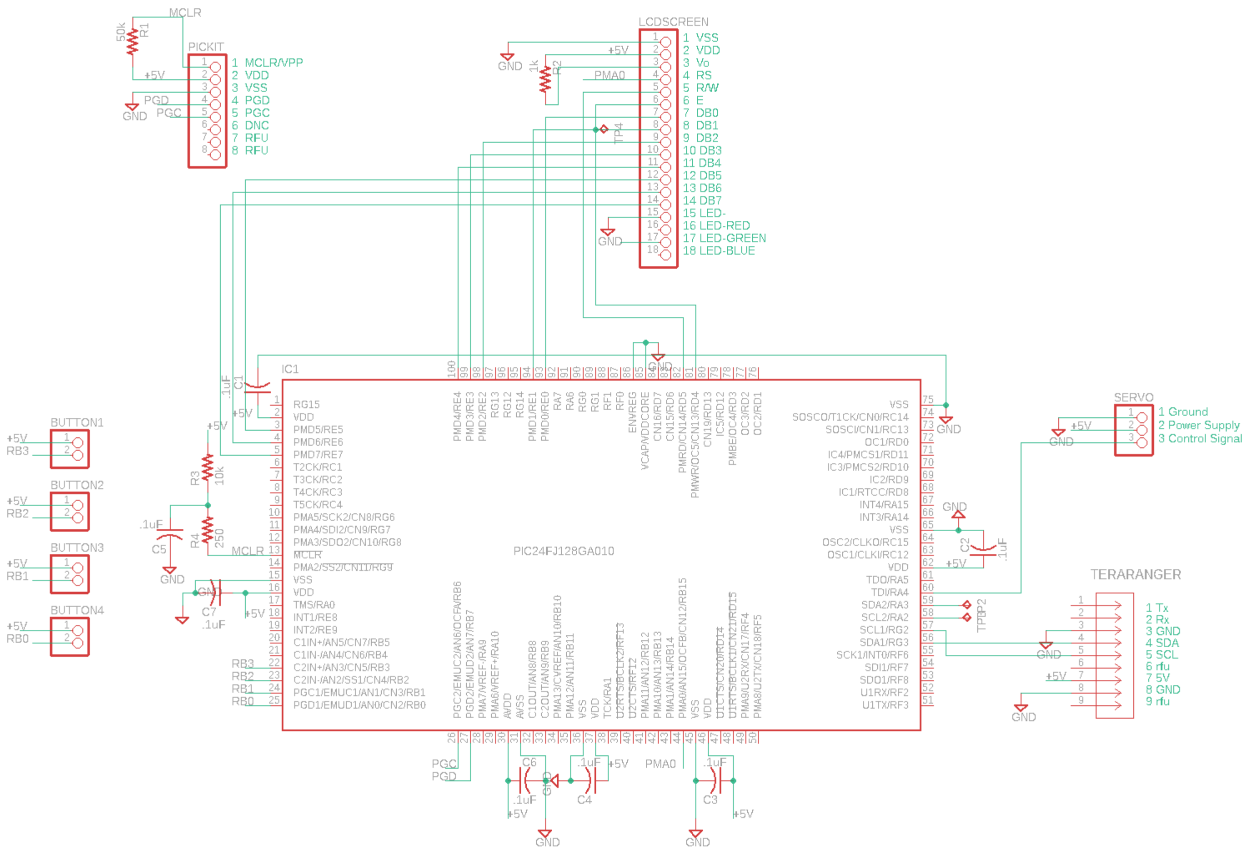


Figure 20: Electrical schematic for processor connection

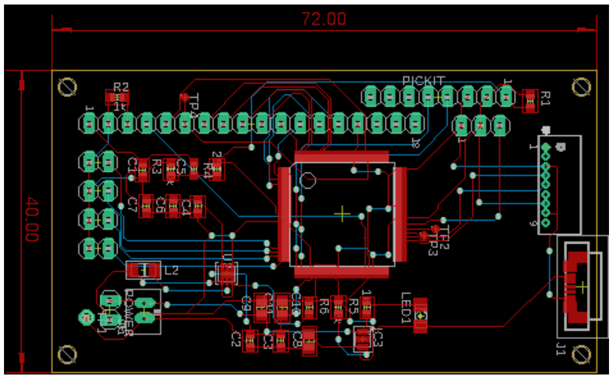


Figure 21: Final PCB Design

The anticipated overall wiring sketch and schematics of the entire system are shown above in Figure 18. Within these schematics, the power circuitry and communication interconnections between the microcontroller, user interface system, time of flight sensor, and servo motor are shown. The entire system will be powered by a 3.7V Lithium Ion battery that will be stepped up to 5V through the use of a DC/DC boost converter. This circuitry can be seen in the upper right of the sketch and in Figure 19. A 5V, 500mA charger will be used to charge the system by plugging it into a Micro USB connector. This charge current will then be directed to the charge algorithm controller which will be used to maintain appropriate voltage and current, so that the battery will not be damaged. This charging system circuitry can be seen in the upper left corner of the sketch and in Figure 19. All of the communication wiring can be viewed in the lower half of the sketch and in Figure 20. The microprocessor will be connected to the LCD’s read/write and enable pins so that data can be written to and received from the display. The momentary on push buttons will be connected to the multipurpose I/O pins on the microprocessor. The microprocessor will also communicate with the TeraRanger time of flight sensor via UART connections. It will also transmit data to the servo motor via PWM signals. Authors: DD, CV

Parts List:

Table 19: Parts List

Qty.	Refdes	Part Num.	Description
2	MTR01	Hitec HS-425BB	Servo Motor
1	DS01	NHD-0216K1Z-NS(RGB)-FBW-REV1	LCD Display
3	CS01	MCP73830T-2AAI/MYY	Charge Algorithm Controller
3	VR01	TPS61240YFFR	Switching Voltage Regulator
1	Micro USB Port	DX4RNW5HJ3R1000	Waterproof Micro-USB Female Connector
1	RF01	TR-EVO-I2C	TeraRanger Rangefinder (LED Emitter/Receiver)
3	PC01	PIC24FJ128GA010-I/PF	PIC24FJ128GA010-I/PF
1		1X18 IDC INTERFACE CABLE	1x18 IDC Interface Cable
2	BAT01	2011	Adafruit 3.7V 2Ah Single Cell Battery
1		DF13-9S-1.25C	DF13 Connector
4	BTN01, BTN02, BTN03, BTN04	SB4011NOM	Momentary Pushbutton
1		CXCP242CP18	ON/OFF Power switch
3	R01	RGT1608P-102-B-T5	1 kOhm Resistor
3	R02	AC0603FR-102KL	2 kOhm Resistor
9	C01, C02, C04	CL10A475MQ8NNNC	4.7 uF Capacitor
3	C03	06036D225KAT4A	2.2 uF Capacitor
3	L01	LQM31PN1R0M00L	1 uH Inductor
6	C05, C06, C07, C08, C09, C10	C0603T104J5RACTU	0.1 uF Capacitor 20V Ceramic
10	C05, C06, C07, C08, C09, C10	CL10B104KB8NNNC	CAP CER 0.1UF 50V X7R 0603
6	C11	CL10A106MQ8NNNC	CAP CER 10UF 6.3V X5R 0603
1	R03	RGT1608P-103-B-T5	10 kOhm Resistor
1	R04	TNPW0603250RBEEN	250 Ohm Resistor
1		IK - 416014	Slik Tip Points (125 Grain)
1		CR150514E106	LED lights
2			PIC16/32 Development Board
2		PNM0603E5002BST5	RES SMD 50K OHM 0.1% 0.15W 0603
1		HA-ZC5V	5V 500mA Wall Charger
1		7P6EV4	USB to micro-USB Cable
1		B07JJ5H3NT	Screws for Assembly
2			PIC16/32 PIC24FJ128GA010 100 Pin board
1		AMAB011752-10	AmazonBasics ABS 3D Printer Filament, 1.75mm, Black, 1 kg Spool
1		B07DWCP38J	Laser Sight Rechargeable
1		PG164140	MPLAB PICkit 4 In-Circuit Debugger
2		PNM0603E5002BST5	RES SMD 50K OHM 0.1% 0.15W 0603
2		RNCF0603TKY250R	RES 250 OHM 0.01% 1/10W 0603
5		RNCP0603FTD10K0	RES 10K OHM 1% 1/8W 0603
5		ERA-3AEB202V	RES SMD 2K OHM 0.1% 1/10W 0603
1		KIT 2221 - 3 Pk	Cable Guide (guitar string)

Table 19 above outlines the main components to build the electrical schematic shown in Figure 18.

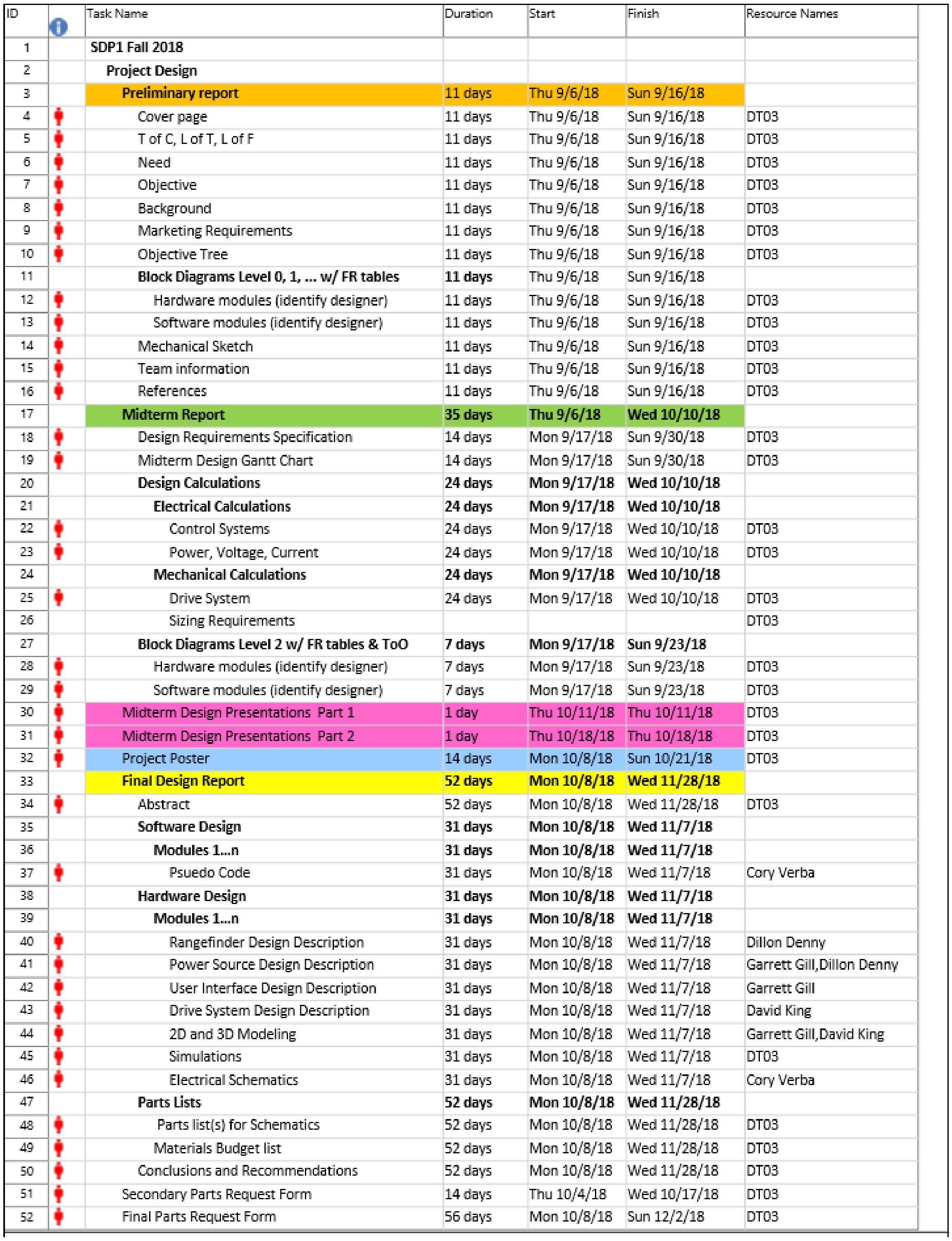
<u>Materials Budget:</u>

Table 20: Revised Material Cost

			Unit	Total
Qty.	Part Num.	Description	Cost	Cost
2	Hitec HS-425BB	Servo Motor	$12.21	$24.42
1	NHD-0216K1Z-NS(RGB)-FBW-REV1	LCD Display	14.75	14.75
3	MCP73830T-2AAI/MYY	Charge Algorithm Controller	0.92	2.76
3	TPS61240YFFR	Switching Voltage Regulator	1.16	3.48
1	DX4RNW5HJ3R1000	Waterproof Micro-USB Female Connector	3.10	3.10
1	TR-EVO-I2C	TeraRanger Rangefinder (LED Emitter/Receiver)	149.00	149.00
3	PIC24FJ128GA010-I/PF	PIC24FJ128GA010-I/PF	4.58	13.74
1	1X18 IDC INTERFACE CABLE	1x18 IDC Interface Cable	11.14	11.14
2	2011	Adafruit 3.7V 2Ah Single Cell Battery	12.50	25.00
1	DF13-9S-1.25C	DF13 Connector	0.31	0.31
4	SB4011NOM	Momentary Pushbutton	3.13	12.52
1	CXCP242CP18	ON/OFF Power switch	7.79	7.79
3	RGT1608P-102-B-T5	1 kOhm Resistor	0.55	1.65
3	AC0603FR-102KL	2 kOhm Resistor	0.10	0.30
9	CL10A475MQ8NNNC	4.7 uF Capacitor	0.10	0.90
3	06036D225KAT4A	2.2 uF Capacitor	0.35	1.05
3	LQM31PN1R0M00L	1 uH Inductor	0.40	1.20
6	C0603T104J5RACTU	0.1 uF Capacitor 20V Ceramic	1.40	8.40
10	CL10B104KB8NNNC	CAP CER 0.1UF 50V X7R 0603	0.10	1.00
6	CL10A106MQ8NNNC	CAP CER 10UF 6.3V X5R 0603	0.18	1.08
1	RGT1608P-103-B-T5	10 kOhm Resistor	0.55	0.55
1	TNPW0603250RBEEN	250 Ohm Resistor	0.78	0.78
1	IK - 416014	Slik Tip Points (125 Grain)	14.99	14.99
1	CR150514E106	LED lights	6.36	6.36
2		PIC16/32 Development Board		
2	PNM0603E5002BST5	RES SMD 50K OHM 0.1% 0.15W 0603	3.07	6.14
1	HA-ZC5V	5V 500mA Wall Charger	10.46	10.46
1	7P6EV4	USB to micro-USB Cable	7.99	7.99
1	B07JJ5H3NT	Screws for Assembly	12.69	12.69
1	AMAB011752-10	AmazonBasics ABS 3D Printer Filament, 1.75mm, Black, 1 kg Spool	18.99	18.99
1	B07DWCP38J	Laser Sight Rechargeable	36.99	36.99
1	PG164140	MPLAB PICkit 4 In-Circuit Debugger	47.95	47.95
2	PNM0603E5002BST5	RES SMD 50K OHM 0.1% 0.15W 0603	3.07	6.14
2	RNCF0603TKY250R	RES 250 OHM 0.01% 1/10W 0603	2.15	4.30
5	RNCP0603FTD10K0	RES 10K OHM 1% 1/8W 0603	0.10	0.50
5	ERA-3AEB202V	RES SMD 2K OHM 0.1% 1/10W 0603	0.35	1.75
1	KIT 2221 - 3 Pk	Cable Guide (guitar string)	13.00	13.00
			Total	$473.17

Table 20 above shows the current state of the budget after choosing the components that will be used to build the main electrical components of the project shown in Figure 18.

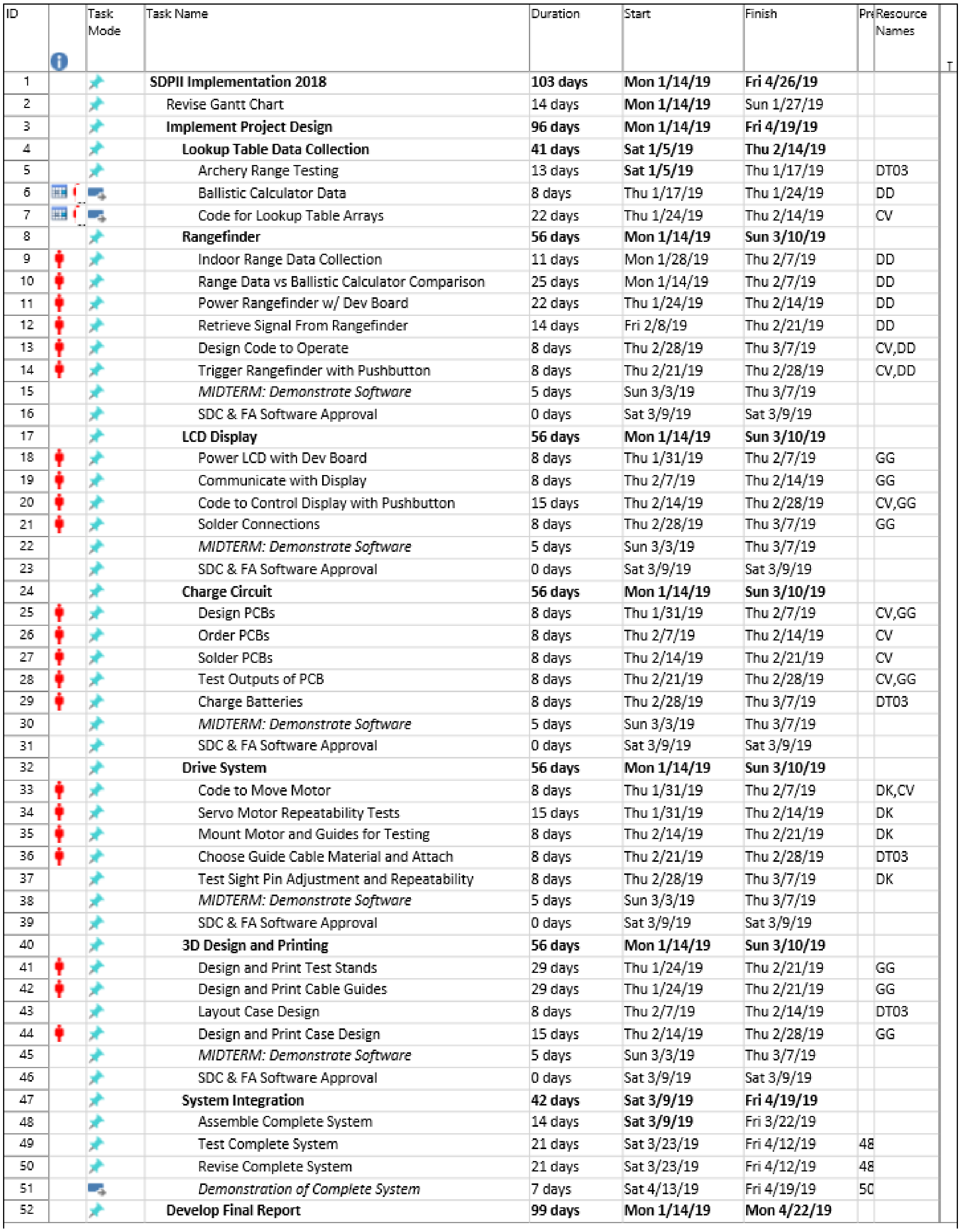
Gantt Chart Fall 2018:



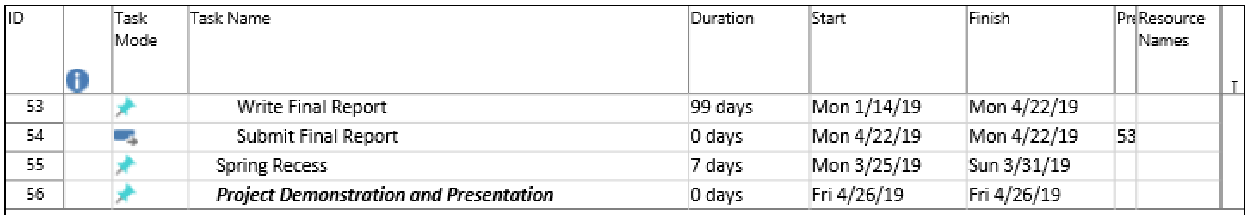
ID	ℹ	Task Name	Duration	Start	Finish	Resource Names
1		**SDP1 Fall 2018**				
2		**Project Design**				
3		**Preliminary report**	11 days	Thu 9/6/18	Sun 9/16/18	
4	●	Cover page	11 days	Thu 9/6/18	Sun 9/16/18	DT03
5	●	T of C, L of T, L of F	11 days	Thu 9/6/18	Sun 9/16/18	DT03
6	●	Need	11 days	Thu 9/6/18	Sun 9/16/18	DT03
7	●	Objective	11 days	Thu 9/6/18	Sun 9/16/18	DT03
8	●	Background	11 days	Thu 9/6/18	Sun 9/16/18	DT03
9	●	Marketing Requirements	11 days	Thu 9/6/18	Sun 9/16/18	DT03
10	●	Objective Tree	11 days	Thu 9/6/18	Sun 9/16/18	DT03
11		**Block Diagrams Level 0, 1, ... w/ FR tables**	**11 days**	Thu 9/6/18	Sun 9/16/18	
12	●	Hardware modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18	DT03
13	●	Software modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18	DT03
14	●	Mechanical Sketch	11 days	Thu 9/6/18	Sun 9/16/18	DT03
15	●	Team information	11 days	Thu 9/6/18	Sun 9/16/18	DT03
16	●	References	11 days	Thu 9/6/18	Sun 9/16/18	DT03
17		**Midterm Report**	**35 days**	**Thu 9/6/18**	**Wed 10/10/18**	
18	●	Design Requirements Specification	14 days	Mon 9/17/18	Sun 9/30/18	DT03
19	●	Midterm Design Gantt Chart	14 days	Mon 9/17/18	Sun 9/30/18	DT03
20		**Design Calculations**	**24 days**	**Mon 9/17/18**	**Wed 10/10/18**	
21		**Electrical Calculations**	**24 days**	**Mon 9/17/18**	**Wed 10/10/18**	
22	●	Control Systems	24 days	Mon 9/17/18	Wed 10/10/18	DT03
23	●	Power, Voltage, Current	24 days	Mon 9/17/18	Wed 10/10/18	DT03
24		**Mechanical Calculations**	**24 days**	**Mon 9/17/18**	**Wed 10/10/18**	
25	●	Drive System	24 days	Mon 9/17/18	Wed 10/10/18	DT03
26		Sizing Requirements				DT03
27		**Block Diagrams Level 2 w/ FR tables & ToO**	**7 days**	**Mon 9/17/18**	**Sun 9/23/18**	
28	●	Hardware modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18	DT03
29	●	Software modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18	DT03
30	●	Midterm Design Presentations Part 1	1 day	Thu 10/11/18	Thu 10/11/18	DT03
31	●	Midterm Design Presentations Part 2	1 day	Thu 10/18/18	Thu 10/18/18	DT03
32	●	Project Poster	14 days	Mon 10/8/18	Sun 10/21/18	DT03
33		**Final Design Report**	**52 days**	**Mon 10/8/18**	**Wed 11/28/18**	
34	●	Abstract	52 days	Mon 10/8/18	Wed 11/28/18	DT03
35		**Software Design**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**	
36		**Modules 1...n**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**	
37	●	Psuedo Code	31 days	Mon 10/8/18	Wed 11/7/18	Cory Verba
38		**Hardware Design**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**	
39		**Modules 1...n**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**	
40	●	Rangefinder Design Description	31 days	Mon 10/8/18	Wed 11/7/18	Dillon Denny
41	●	Power Source Design Description	31 days	Mon 10/8/18	Wed 11/7/18	Garrett Gill,Dillon Denny
42	●	User Interface Design Description	31 days	Mon 10/8/18	Wed 11/7/18	Garrett Gill
43	●	Drive System Design Description	31 days	Mon 10/8/18	Wed 11/7/18	David King
44	●	2D and 3D Modeling	31 days	Mon 10/8/18	Wed 11/7/18	Garrett Gill,David King
45	●	Simulations	31 days	Mon 10/8/18	Wed 11/7/18	DT03
46	●	Electrical Schematics	31 days	Mon 10/8/18	Wed 11/7/18	Cory Verba
47		**Parts Lists**	**52 days**	**Mon 10/8/18**	**Wed 11/28/18**	
48	●	Parts list(s) for Schematics	52 days	Mon 10/8/18	Wed 11/28/18	DT03
49	●	Materials Budget list	52 days	Mon 10/8/18	Wed 11/28/18	DT03
50	●	Conclusions and Recommendations	52 days	Mon 10/8/18	Wed 11/28/18	DT03
51	●	Secondary Parts Request Form	14 days	Thu 10/4/18	Wed 10/17/18	DT03
52	●	Final Parts Request Form	56 days	Mon 10/8/18	Sun 12/2/18	DT03

Authors: DD, GG, DK, CV

Gantt Chart Spring 2019:



ID		Task Mode	Task Name	Duration	Start	Finish	Pre	Resource Names
1			**SDPII Implementation 2018**	**103 days**	**Mon 1/14/19**	**Fri 4/26/19**		
2			Revise Gantt Chart	14 days	**Mon 1/14/19**	Sun 1/27/19		
3			**Implement Project Design**	**96 days**	**Mon 1/14/19**	**Fri 4/19/19**		
4			**Lookup Table Data Collection**	**41 days**	**Sat 1/5/19**	**Thu 2/14/19**		
5			Archery Range Testing	13 days	**Sat 1/5/19**	Thu 1/17/19		DT03
6			Ballistic Calculator Data	8 days	Thu 1/17/19	Thu 1/24/19		DD
7			Code for Lookup Table Arrays	22 days	Thu 1/24/19	Thu 2/14/19		CV
8			**Rangefinder**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**		
9			Indoor Range Data Collection	11 days	Mon 1/28/19	Thu 2/7/19		DD
10			Range Data vs Ballistic Calculator Comparison	25 days	Mon 1/14/19	Thu 2/7/19		DD
11			Power Rangefinder w/ Dev Board	22 days	Thu 1/24/19	Thu 2/14/19		DD
12			Retrieve Signal From Rangefinder	14 days	Fri 2/8/19	Thu 2/21/19		DD
13			Design Code to Operate	8 days	Thu 2/28/19	Thu 3/7/19		CV,DD
14			Trigger Rangefinder with Pushbutton	8 days	Thu 2/21/19	Thu 2/28/19		CV,DD
15			*MIDTERM: Demonstrate Software*	5 days	Sun 3/3/19	Thu 3/7/19		
16			SDC & FA Software Approval	0 days	Sat 3/9/19	Sat 3/9/19		
17			**LCD Display**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**		
18			Power LCD with Dev Board	8 days	Thu 1/31/19	Thu 2/7/19		GG
19			Communicate with Display	8 days	Thu 2/7/19	Thu 2/14/19		GG
20			Code to Control Display with Pushbutton	15 days	Thu 2/14/19	Thu 2/28/19		CV,GG
21			Solder Connections	8 days	Thu 2/28/19	Thu 3/7/19		GG
22			*MIDTERM: Demonstrate Software*	5 days	Sun 3/3/19	Thu 3/7/19		
23			SDC & FA Software Approval	0 days	Sat 3/9/19	Sat 3/9/19		
24			**Charge Circuit**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**		
25			Design PCBs	8 days	Thu 1/31/19	Thu 2/7/19		CV,GG
26			Order PCBs	8 days	Thu 2/7/19	Thu 2/14/19		CV
27			Solder PCBs	8 days	Thu 2/14/19	Thu 2/21/19		CV
28			Test Outputs of PCB	8 days	Thu 2/21/19	Thu 2/28/19		CV,GG
29			Charge Batteries	8 days	Thu 2/28/19	Thu 3/7/19		DT03
30			*MIDTERM: Demonstrate Software*	5 days	Sun 3/3/19	Thu 3/7/19		
31			SDC & FA Software Approval	0 days	Sat 3/9/19	Sat 3/9/19		
32			**Drive System**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**		
33			Code to Move Motor	8 days	Thu 1/31/19	Thu 2/7/19		DK,CV
34			Servo Motor Repeatability Tests	15 days	Thu 1/31/19	Thu 2/14/19		DK
35			Mount Motor and Guides for Testing	8 days	Thu 2/14/19	Thu 2/21/19		DK
36			Choose Guide Cable Material and Attach	8 days	Thu 2/21/19	Thu 2/28/19		DT03
37			Test Sight Pin Adjustment and Repeatability	8 days	Thu 2/28/19	Thu 3/7/19		DK
38			*MIDTERM: Demonstrate Software*	5 days	Sun 3/3/19	Thu 3/7/19		
39			SDC & FA Software Approval	0 days	Sat 3/9/19	Sat 3/9/19		
40			**3D Design and Printing**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**		
41			Design and Print Test Stands	29 days	Thu 1/24/19	Thu 2/21/19		GG
42			Design and Print Cable Guides	29 days	Thu 1/24/19	Thu 2/21/19		GG
43			Layout Case Design	8 days	Thu 2/7/19	Thu 2/14/19		DT03
44			Design and Print Case Design	15 days	Thu 2/14/19	Thu 2/28/19		GG
45			*MIDTERM: Demonstrate Software*	5 days	Sun 3/3/19	Thu 3/7/19		
46			SDC & FA Software Approval	0 days	Sat 3/9/19	Sat 3/9/19		
47			**System Integration**	**42 days**	**Sat 3/9/19**	**Fri 4/19/19**		
48			Assemble Complete System	14 days	**Sat 3/9/19**	Fri 3/22/19		
49			Test Complete System	21 days	Sat 3/23/19	Fri 4/12/19	48	
50			Revise Complete System	21 days	Sat 3/23/19	Fri 4/12/19	48	
51			*Demonstration of Complete System*	7 days	Sat 4/13/19	Fri 4/19/19	50	
52			**Develop Final Report**	**99 days**	**Mon 1/14/19**	**Mon 4/22/19**		



ID	ℹ	Task Mode	Task Name	Duration	Start	Finish	Pre	Resource Names
53			Write Final Report	99 days	Mon 1/14/19	Mon 4/22/19		
54			Submit Final Report	0 days	Mon 4/22/19	Mon 4/22/19	53	
55			Spring Recess	7 days	Mon 3/25/19	Sun 3/31/19		
56			***Project Demonstration and Presentation***	0 days	Fri 4/26/19	Fri 4/26/19		

Authors: DD, GG, DK, CV

Team Information:

Dillon Denny, Electrical Engineering. ESI: Yes

Garrett Gill, Electrical Engineering. ESI: Yes

David King, Electrical Engineering. ESI: Yes

Cory Verba, Electrical Engineering. ESI: Yes

Conclusions and Recommendations:

The final design of the Automatic Range Finding Bow Sight met the expectations set forth. The system consisted of a drive system with a tethered cable system between two guide wheels, a four button user interface with two line LCD, and a rangefinder that operated from 15 to 45 yards. The operator of the bow will first input the speed at which their bow shoots. Then, they will trigger the rangefinder to find the range of their target. Finally, the data collected from the user and the rangefinder is processed and compared against a lookup table that has been programmed into the processor in order to move the sight pin to the proper location.

During testing, the bow was fired at a range of 15 and 25 yards. Using these distances, it was possible to observe the sight pin adjust up or down, and verify that the shot would hit the intended target. After sighting in the bow to the minimum range of 15 yards, a bullseye was hit at the increased distance of 25 yards. With further testing and more time, this design could continue to be improved.

Authors: DD, GG, DK, CV

References

[1] B. Robb, "Selecting A Bow Sight," *NBS Nation's Best Sports Fish & Hunt,* pp. 72-76, 2007.

[2] T. Kuhn, "SIGHT PIN SELECTION," *Outdoor Life,* p. B2, August 2012.

[3] Lateral Solutions, LLC., "Dead-On Range Finder | Bow Mounted Bow Hunting Rangefinder," Lateral Solutions, LLC., [Online]. Available: https://www.deadonrangefinder.com/. [Accessed 28 2 2018].

[4] P. M. Lorocco, "Multiple pin sight for an archery bow". United State of America Patent US6634111B2, 28 06 2001.

[5] T. M. Gorsuch and J. A. Buckley, "Auto-correcting bow sight". United State of America Patent US20100115778A1, 09 11 2009.

[6] B. J. Zykan and J. G. Dunne, "Laser bow sight apparatus". United State of America Patent US6073352A, 19 03 1998.

[7] J. Vervaeke, J. Ameye, S. van Esh, J. Saldien and S. Verstockt, "ArcAid interactive archery assistant," in *Intelligent Technologies for Interactive Entertainment (INTETAIN), 2015 7th International Conference*, Turin, 2015.

[8] R. Ulmer, "Estimating Range The Old-Fashioned Way," 28 October 2010. [Online]. Available: http://www.bowhuntingmag.com/tactics/tactics_bh_range_0609/. [Accessed 3 21 2018].

Appendix:

Hitec HS-425BB Servo Motor:
https://cdn.sparkfun.com/datasheets/Robotics/Hitec-HS-425BB-Servo-Specsheet.pdf

3.7V 2Ah Li-Polymer Battery:
https://cdn-shop.adafruit.com/datasheets/LiIon2000mAh37V.pdf

16x2 LCD Display:
https://www.hawkusa.com/sites/hawk-dev.ent.c-g.io/files/hawk_item/NWHVN/Series%20LCD%20Character/NHD-0216K1Z-NS%28RGB%29-FBW-Rev1/spec/NHD-0216K1Z-NS_RGB_FBW-REV1.pdf

TeraRanger Evo 60m Time of Flight Sensor:
https://www.robotshop.com/media/files/pdf2/teraranger-evo-60m-specification-sheet.pdf

TPS6124 Switching Voltage Regulator:
https://www.digikey.com/product-detail/en/texas-instruments/TPS61241YFFR/296-24520-1-ND/2062926

Charge Algorithm Controller:
https://www.mouser.com/datasheet/2/268/20005049D-607242.pdf

Momentary Pushbutton:
http://spec_sheets.e-switch.com/specs/29-KS01Q01.pdf

Microchip Processor PIC24FJ128GA010:
https://www.mouser.com/datasheet/2/268/39747F-254584.pdf

Waterproof Micro-USB Connector:
https://www.mouser.com/datasheet/2/206/jaeelectronics_DX4RNW5HJ3%20SJ114803-1170919.pdf

DF13 Connector:
https://www.mouser.com/datasheet/2/185/DF13_catalog-939190.pdf