Payload Mechanism

Abstract

This project report covers the results and findings from researching, designing, manufacturing, and assembling a payload mechanism used to lift and lower objects. The report is divided into four main sections: the introduction, methods, results and discussion, and conclusion. The introduction explains the inspiration behind the project, why it was chosen, the requirements it needed to meet, and states the question I'm trying to answer, which is how light, compact, and strong can a lifting mechanism be made. The methods section goes into detail about the reasoning and calculation used behind every major choice, from the gear design to the electrical components, and it lists the parts used to satisfy the requirements of a General Mechatronics System (Figure 0). The results and discussion section covers the findings and setbacks during assembly, including tolerance issues, parts breaking due to being too thin, and spots where I had to add extra support to keep components secure. The conclusion wraps everything up by summarizing what worked, what didn't, and what I would change or improve in later versions.

Introduction

As part of the final project for Introduction to Mechatronics (ME 133), we were assigned to build a project that complies with 3 requirements. The first requirement is that it must incorporate each component from the General Mechatronic System. This means it has to include actuators, sensors, input and output signal conditioning interfacing, digital control architectures, and a user interface. The second requirement is that it must incorporate a method for which the

mechanism can be controlled. Lastly, the project should be complex enough and different from labs done in the past. Given all these requirements, I have decided that I will be building a miniature hoist mechanism. The inspiration behind this decision is heavily influenced by the fact that I was tasked by a drone club with researching and developing a mechanism that will allow an airborne drone to deliver a payload safely to the designated location for SUAS 2026. Given that the requirements for that project satisfies the requirements required by the final project, it seems like a no-brainer. Additionally, this project aims to try to answer the question of how compact, light, and strong can I make a lifting mechanism. The smallest electronically driven lifting mechanisms found online are rated for hundreds of pounds but are just too heavy and bulky to be used on a drone where every extra gram makes a difference. Though the payload mechanism takes inspiration from hoists found in machine shops, it will be challenging to determine what components we can do without and still ensure it remains safe and reliable. Furthermore it will take a lot of research and development to find the appropriate gear reduction given the respective motor specifications and then finally integrating it into a compact and lightweight form factor. That being said, in the past I have used Solidworks to model and manufacture Custom Car Stands for the FSAE Highlander Racing team. I have experience cading for manufacturing and tolerancing parts for FDM printing which will be our principal tool for creating the payload mechanism parts.

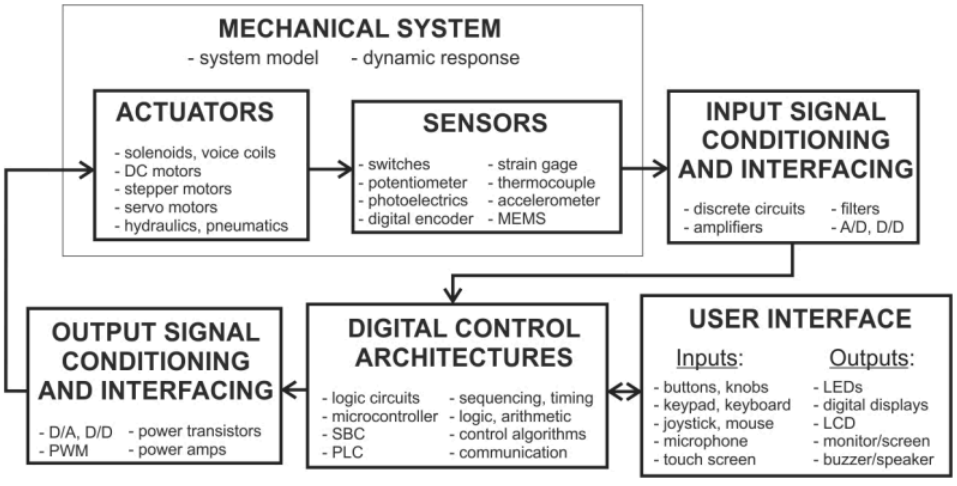


Figure 0: The General Mechatronics System

Methods

Requirements

The first step in building this project was outlining the requirements the hoist needed to meet. After researching different hoists and their specifications, I decided that a lifting and lowering speed of around 80 feet per minute would be a good target. The system also needed to lift a load of roughly 600 grams. For SUAS 2026, 80 fpm makes sense because it means the payload can be raised or lowered about 150 feet in roughly two minutes. That speed can be improved later, but for now I'm focusing on making the mechanism reliable, compact, and as light as possible. Anything in the ballpark of 50–85 feet per minute is acceptable for this first iteration

ME Components

Gears

Originally I was going to use **Spur Gears** (also called Involute Gears) because of their simple design and ease of making. However, given the high speeds at which these gears will be spinning, there are other gear designs that are better suited for these speeds. This all comes down to the way in which the teeth mesh with each other. When the teeth of Spur gears mesh, the entirety of the tooth touches at once which results in a louder noise when running at higher speeds and also more wear because the tooth doesn't get to slowly load the gear but rather "shocks" the tooth. In other words, the entirety of the load gets shifted to the next gear all at once rather than gradually.

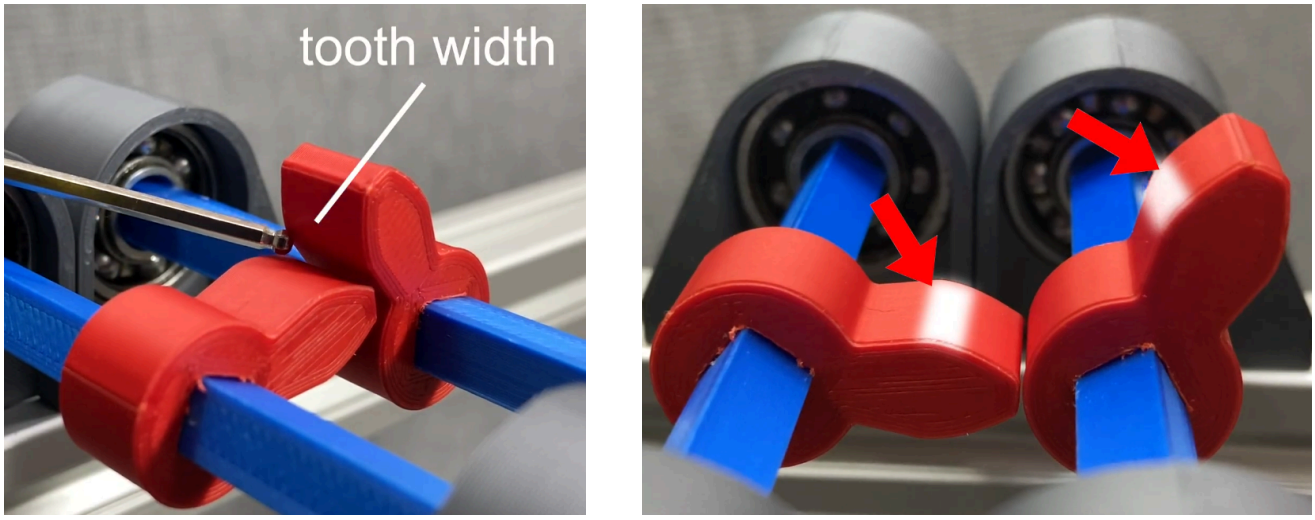


Figure 2: Spur Gear Meshing Process

With Helical and Herringbone Gear, the teeth being angled 15-25 degrees allows for the load to gradually be loaded onto each tooth. This subtle but important modification is what allows for smoother and quieter meshing at higher speeds while greatly reducing the wear and tear on the plastic gears.

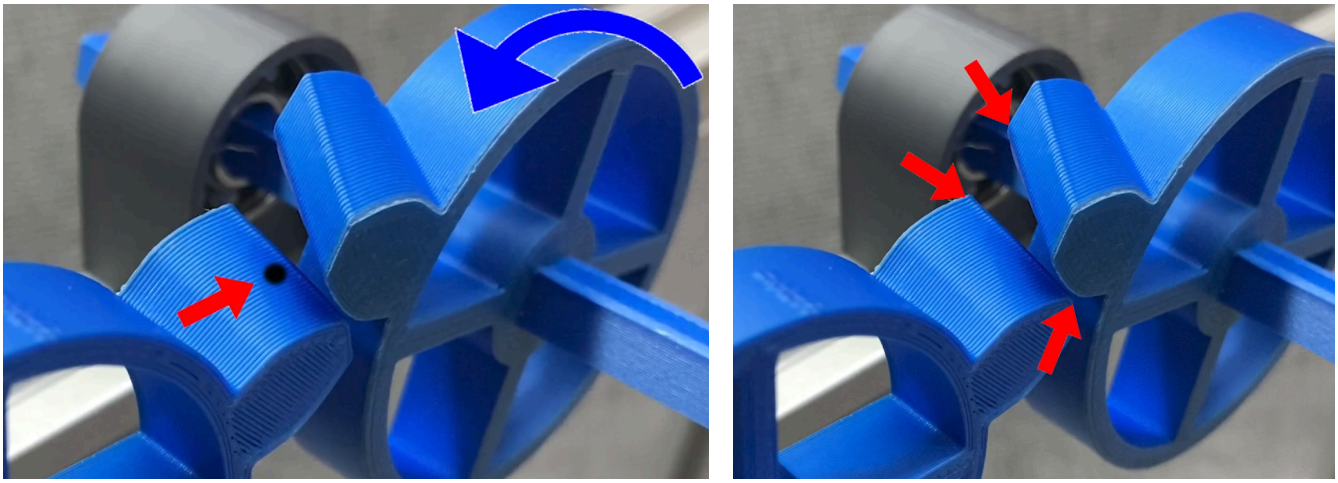


Figure 3: Helical Gear Meshing Process

Body

Having a rigid frame for all the components to be mounted and secured onto is very important because we want to fully maximize the torque that the motor, through the gearbox, will generate. Since everything will be made using an FDM printer, it is important that the body is designed with 3D printing in mind, which requires careful consideration of rigidity and overall structural integrity. The plan is to have two 5 mm stainless steel shafts run through the bores of the gears, and to reduce friction and allow the gears to spin as freely and smoothly as possible, I will be countersinking sealed bearings with an 8 mm outer diameter and 5 mm inner diameter into the gears so they sit flush with the gear face. Because the bearing ID and shaft OD are the same, I will need to sand down the shaft slightly to slide the bearings on. Since I want the gears to spin around the shaft and not with the shaft, I will fix the shaft to the body using M3 heat set inserts and M3 Allen bolts to act as a collar and prevent rotation. This same approach will be used to fix the input gear onto the rotation of the motor shaft. To keep the motor secured to the body, I will also make a motor mount that restricts the motor's movement in all six degrees of freedom. Furthermore, the drum that the rope will wind onto must be fixed to the rotation of the final gear in the gear reduction assembly, which I will achieve using a key slot, or keyway. This locking method involves cutting a channel in both the gear's bore and the shaft surface to fit a machine key, allowing the gear and shaft to lock together, preventing slippage and ensuring they rotate as a single unit.

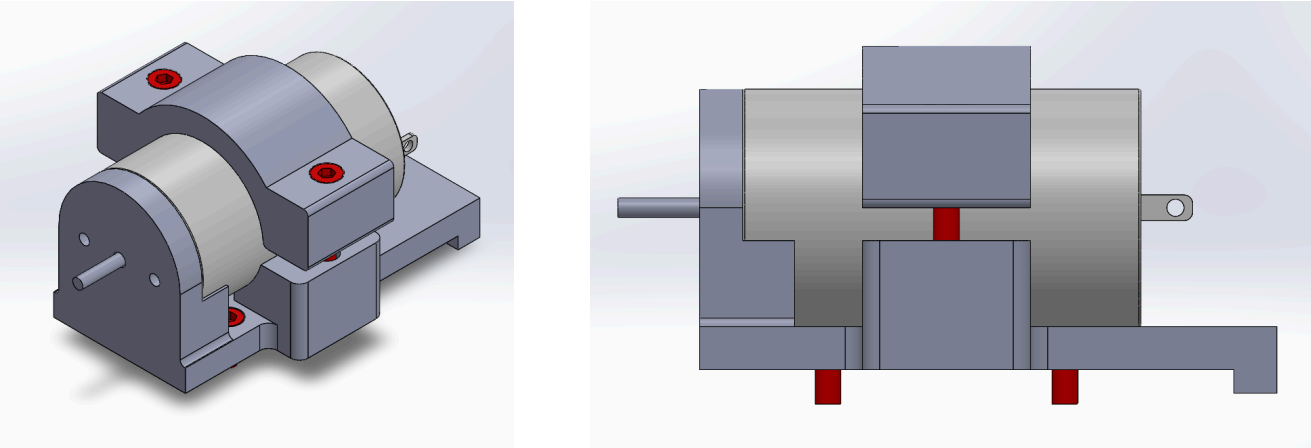


Figure 4: Motor Mount and Top Piece

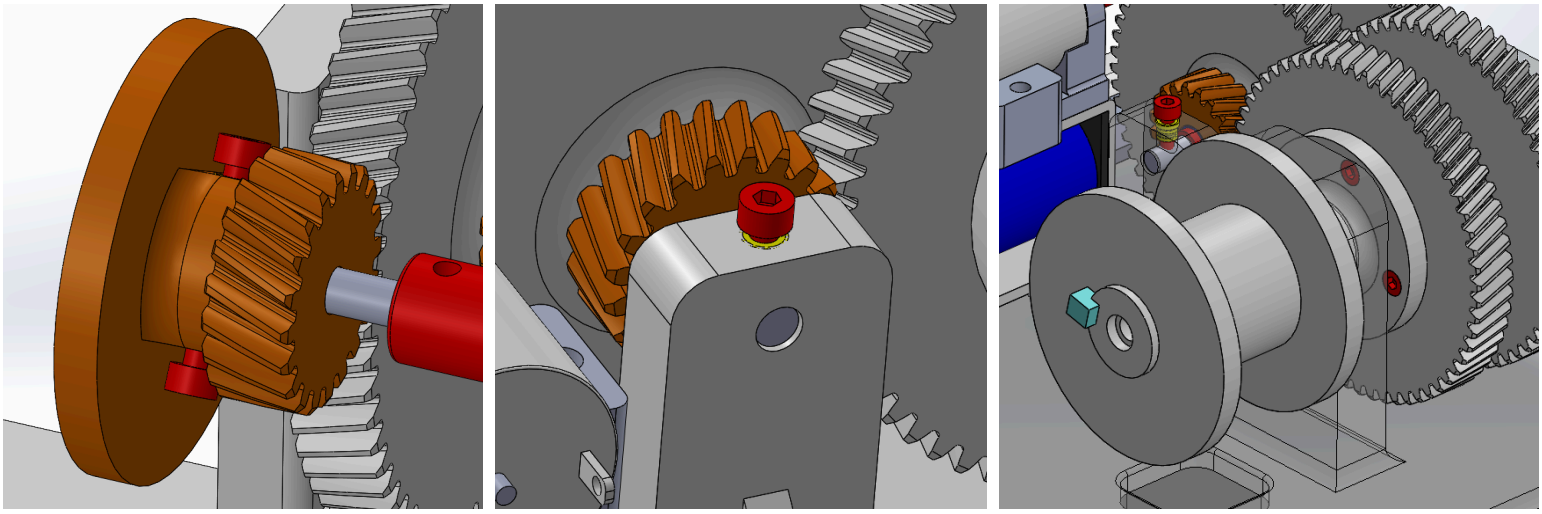
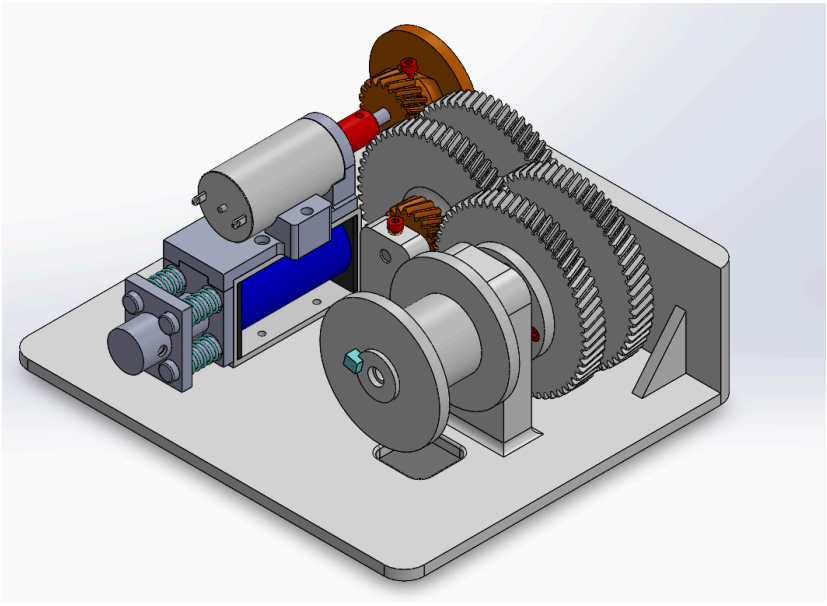


Figure 5: Input Gear, 5mm Shaft, and Drum Locking Method



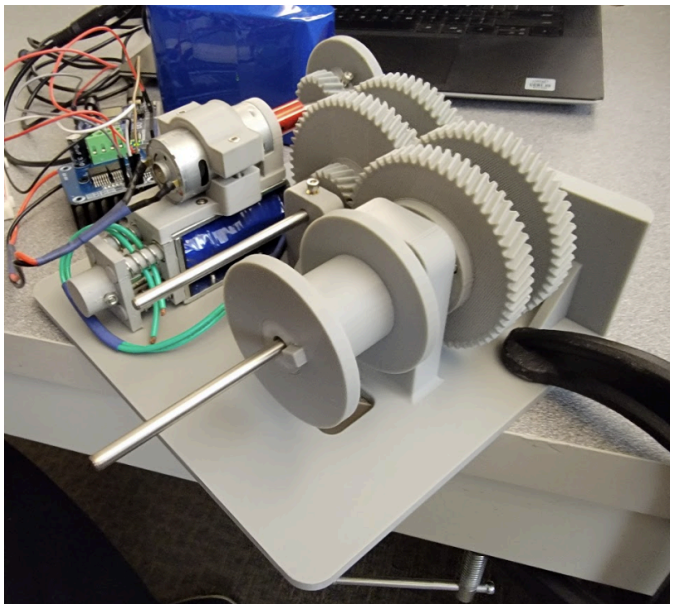


Figure 6: Final Assembly (more photos at end)

EE Components

Actuator

DC motors are commonly used in mechatronics projects because of their efficiency, their ability to run for long periods with minimal heating, and their low cost. The only downside to these motors is that they do not perform very well in lifting scenarios. There is an inverse correlation between speed and torque in motors, which states that the faster a motor spins the lower its torque, and the slower it spins the higher its torque. This can work for short periods, but you risk damaging or overheating the motor because this type of application requires high amounts of current to flow into the motor continuously, which DC motors are not designed to handle. However, since the lifting capability and the ability to prevent overheating or damage under load will ultimately depend on the gearbox, the motor of choice is the AUTOTOOLHOME 6–12V Mini DC Motor High Torque Gear for Traxxas R/C and Power Wheels PCB DIY Electric Drill. Its low price of $6.89 and its accessible specifications, such as operating current, speed (RPM), voltage range, and maximum efficiency range, make it a good fit for this project.

Performanece			
Voltage	Deprating Range	V/DC	6V-18V
	Voltage	V/DC	12V
AT NO LOAD	Speed	r/min	12000
	Current	A	0.24
AT MAXINUM EFFICIENCY	Efficiency	%	no
	Speed	r/min	10386
	Current	A	1.55
	Torque	g.cm	96
	Output	W	10.23
AT MAX POWER	Output	W	22
	Speed	r/min	6000
	Current	A	5.12
	Torque	g.cm	357
AT STALL	Torque	g.cm	714
	Current	A	10

Figure 7: DC Motor Data Sheet

Sensors

A Ponteniometer serves as a sensor in this circuit because it converts a physical input of the user into an electrical signal which then the microcontroller can use.

Figure 8: Potentiometer

Input Conditioning and Interfacing

Through the use of a potentiometer in conjunction with an Arduino UNO R4 WIFI, we are able to meet the requirement of having an Analog-to-Digital Conversion. The potentiometer serves as an important tool in this scenario because it allows us to mechanically control and set the value of the voltage running through the circuit and into the microcontroller. This function is what provides the analog part of the conversion due to the ability to provide a range in which the voltage can be set and changed from by the turn of a knob. However, for it to truly be an Analog-to-Digital converter there has to be a way for the turning of the knob to correspond to a digital value which the microcontroller can read and control the speed of the motor through. The

Arduino performs this conversion by reading the analog signal sent by the potentiometer and converts it into a number between 0 and 1023.

Output Conditioning and Interfacing

Each number in the 0-1023 scale corresponds to a digital reading of the potentiometer's analog voltage which then allows for the control of the speed once it has been mapped to a PWM value ranging from 0-255. PWM stands for Pulse Width Modulation and is a common method for controlling the voltage supplied to DC Motors because of its compatibility with the idea of having more control over the speed of the motor. Each PWM value corresponds to a duty cycle between 0 and 100 which is what actually controls the speed of the motor. As an example, a PWM value of 255 corresponds to a 100% duty cycle which means that a continuous stream of 5V is always being supplied.

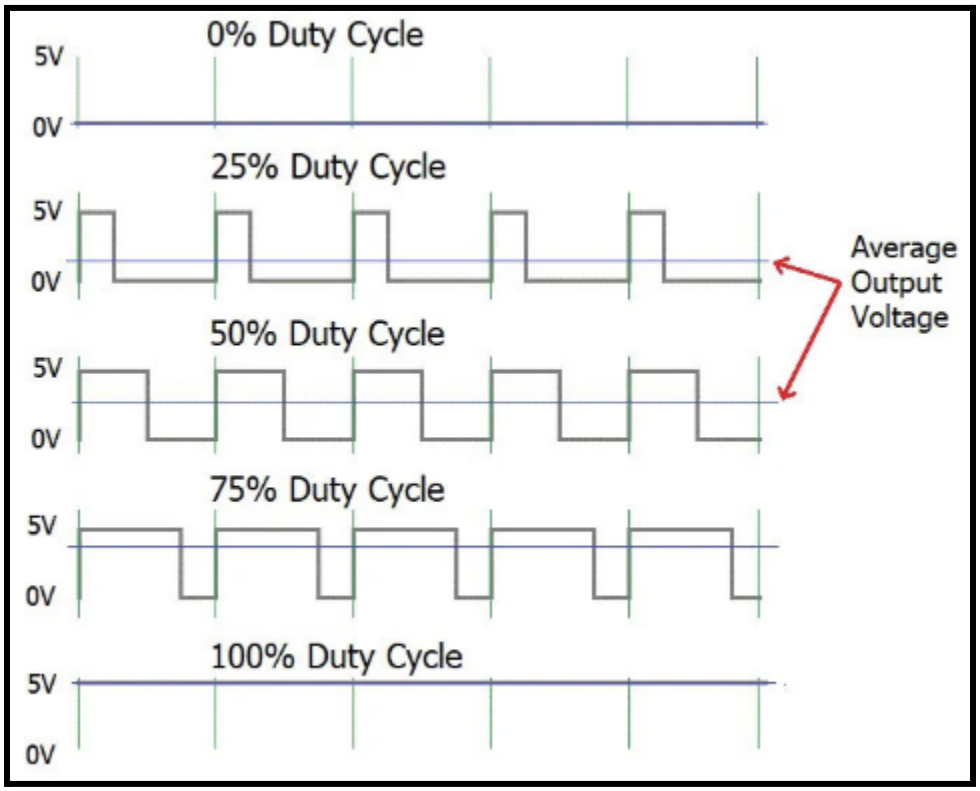
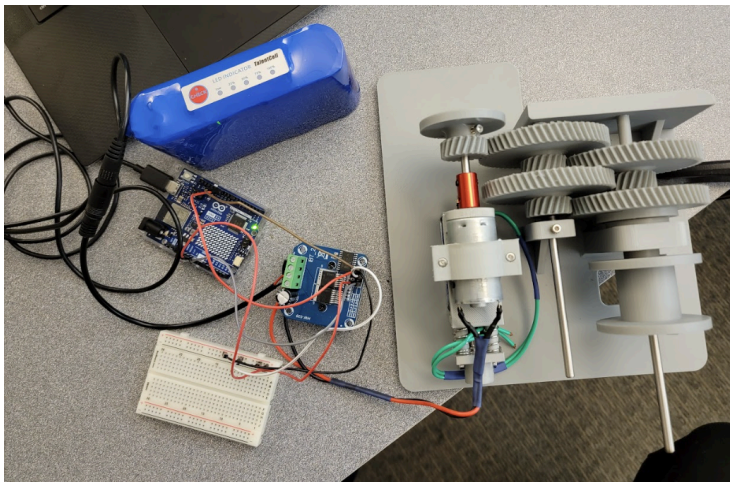


Figure 9: PWM Value to Duty Cycle Relationship

Digital Control Architecture

An Arduino UNO R4 WiFi is the microcontroller I'm using, and it handles all the logic needed for the system to run. In this setup, its main job is to read the input coming from the potentiometer, which shows up as a changing voltage, and converts that voltage into a digital value between 0 and 1023. Once it has that number, the Arduino uses the code written for the circuit to turn that value into a PWM signal in the 0 to 255 range. That PWM output is what actually lets me control the speed of the DC motor. Since the motor needs either 6V or 12V to operate, it can't be powered directly from the Arduino's 5V output, so an external power source is required. For this project, I'm using a TalentCell 12V 6Ah LiFePO4 battery pack, which provides a steady 12V, has a capacity of 6000 mAh, and can supply up to 5A. To safely handle the higher voltage and current and to allow direction control of the motor, I'm using a BTS7960 43A motor driver, which supports supply voltages from 5V to 27V. This combination lets the Arduino read the user input, process it, and control the motor using the proper power and hardware.



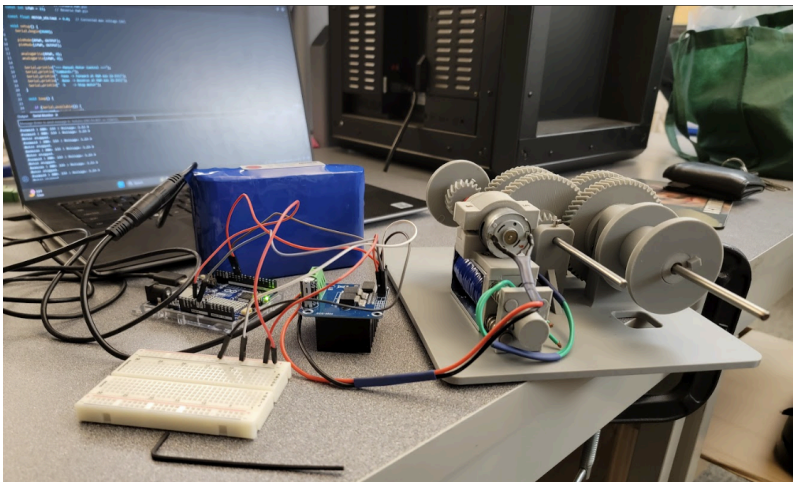


Figure 10: Connection Layout for the Arduino UNO R4 WiFi, Power Supply, and Motor Driver

User Interface

The user interface is simple, since the code lets the computer act as the control system. The user just types F#, R#, or S into the serial monitor. F and R set the direction of the motor, either forward or reverse, while S acts as a stop command, which is essentially the same as sending F0 or R0. The number that follows F or R is the PWM value, which can range from 0 to 255. A value of 255 runs the motor at its maximum speed, and a value of 0 brings it to a stop. This setup makes controlling the motor very straightforward, but I do plan on improving it as the project develops. The long-term goal is to move toward wireless control.

```
const int RPWM = 10;     // Forward PWM pin
const int LPWM = 11;     // Reverse PWM pin

const float MOTOR_VOLTAGE = 9.0;   // Corrected max voltage (9V)

void setup() {
  Serial.begin(9600);

  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);

  analogWrite(RPWM, 0);
  analogWrite(LPWM, 0);

  Serial.println("=== Manual Motor Control ===");
  Serial.println("Commands:");
  Serial.println("  Fxxx -> Forward at PWM xxx (0-255)");
  Serial.println("  Rxxx -> Reverse at PWM xxx (0-255)");
  Serial.println("  S    -> Stop motor");
}

void loop() {

  if (Serial.available()) {
    char cmd = Serial.read();

    if (cmd == 'F') {
      int pwm = Serial.parseInt();
      pwm = constrain(pwm, 0, 255);

      float duty = pwm / 255.0;
      float approxVoltage = duty * MOTOR_VOLTAGE;

      analogWrite(RPWM, pwm);
      analogWrite(LPWM, 0);
      Serial.print("Forward | PWM: ");
      Serial.print(pwm);
      Serial.print(" | Voltage: ");
      Serial.print(approxVoltage);
      Serial.println(" V");
    }

    else if (cmd == 'R') {
      int pwm = Serial.parseInt();
      pwm = constrain(pwm, 0, 255);

      float duty = pwm / 255.0;
      float approxVoltage = duty * MOTOR_VOLTAGE;

      analogWrite(RPWM, 0);
      analogWrite(LPWM, pwm);

      Serial.print("Reverse | PWM: ");
      Serial.print(pwm);
      Serial.print(" | Voltage: ");
      Serial.print(approxVoltage);
      Serial.println(" V");
    }

    else if (cmd == 'S') {
      analogWrite(RPWM, 0);
      analogWrite(LPWM, 0);
      Serial.println("Motor stopped.");
    }
  }
}
```

Figure 11: Arduino Program for Controlling the DC Motor

Calculations

Gear-Reduction

As seen in Figure 1, running the motor at roughly 10386 rpm is what puts it under the maximum efficiency range and so I will set that as my input angular velocity from which I have to reduce to a final output angular velocity of roughly 128 rpm. Using the following equation

$$Feet\ per\ Minute\ =\ Revolutions\ per\ Minute\ x\ \pi\ x\ Dimater$$

with an angular velocity of 128 rpm with a drum diameter of 63.5mm, I get a linear velocity of 83.5 feet per minute. To get this output velocity, I have to obtain a gear ratio of 81:1. I obtained this value from the following equation

$$Gear\ Ratio\ =\ \frac{\omega_1}{\omega_2} = \frac{n_1}{n_2} = \frac{d_2}{d_1} = \frac{T_2}{T_1}$$

where ω1 and ω2 are the angular velocity in radians/sec for the driver and driven gear, n1 and n2 are the gear speeds in rpm, d1 and d2 are the driver and driven gear pitch diameters, and T1 and T2 are the number of teeth on the driver and driven gear respectively. Using the gear speeds relation, I divided 10,386 by 128 to get a gear ratio of 81. This means that the driver gear has to make 81 revolutions before the output gear makes 1 revolution. In order to make the design more compact for drone applications, I will be using 5 individual gears, including the driver gear, to obtain this gear ratio. I will be working with 2 different types of pitch diameters, 25mm and 75mm. The pitch diameter is the main value that determines gear ratios as seen in the above equation. In order to get the desired ratio, I will be using a method called compounding which allows me to easily increase the gear ratio when using multiple smaller gears. Originally meshing

a 25mm PD gear with a 75mm PD gear will only give me a gear reduction (ratio) of 3:1. However, using the compounding effect, if I mesh the 75mm gear with another 25mm 4 more times, I can get a final gear ratio of 81:1 because compounding allows for a multiplication of the gear ratios.

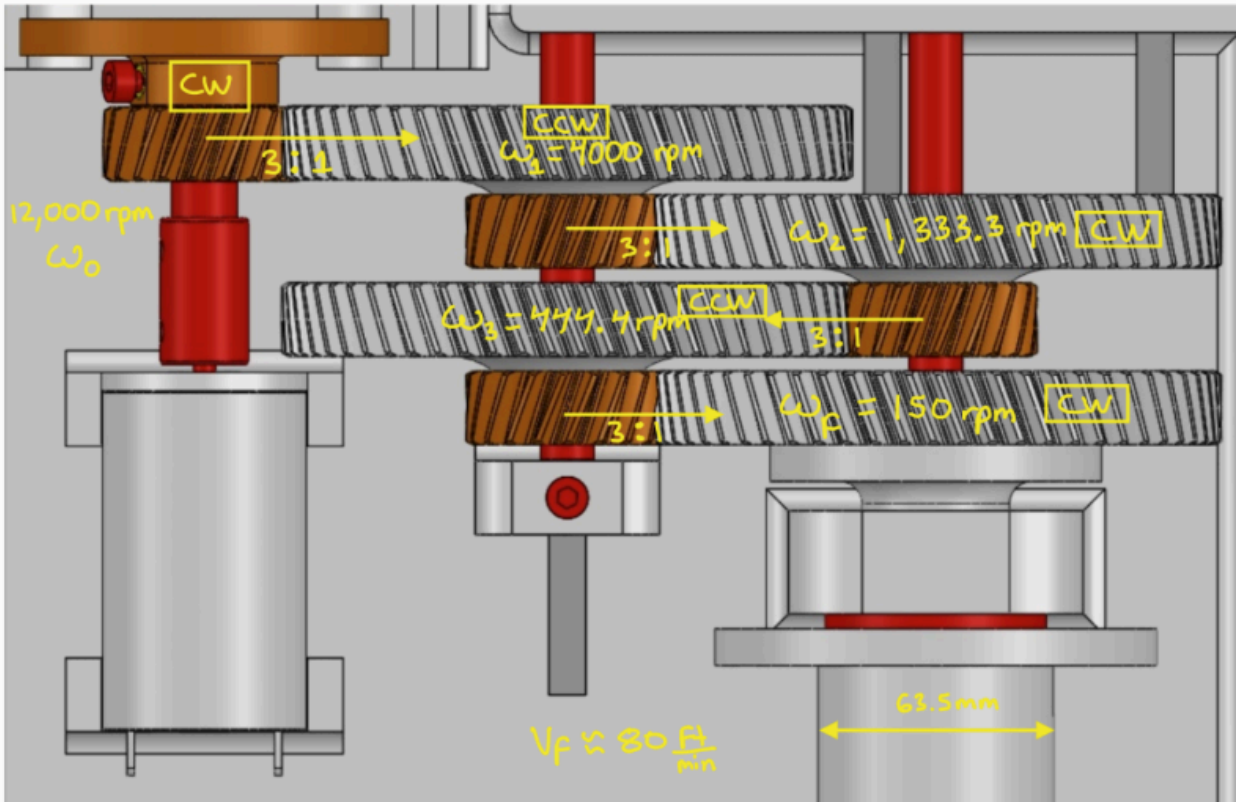


Figure 12: Compounding Gears in the Payload Assembly

Results and Discussion

The CAD model turned out to be very close to the final printed product, which was encouraging, but the assembly process proved to be more troublesome than expected. I didn't leave enough room for my hands or tools, and fitting the drum shaft almost didn't work at all. The motor

mount also caused issues at first, as it couldn't properly secure the motor. To fix this, I added an extra support piece that sits on top of the motor and screws into the base. However, the main motor mount body where the bolts for the top piece screwed in was too thin, so when I started tightening the bolts, the plastic began to shear and deform because the heat-set inserts were pulling out along with the material. To raise the motor higher, I had to screw the motor mount body into the main body, but since I didn't countersink the bolts, the main body no longer sat flat on a table. The motor shaft itself was too short, so I used a shaft coupler to extend the existing shaft and also make it thicker at the point where the input gear sits. This created a minor problem with concentricity, as the coupler and shaft didn't always spin perfectly aligned with the motor, but it wasn't severe enough to prevent operation. A similar issue occurred when attaching the input gear; originally one Allen bolt caused the gear to flex and misalign with the shaft, so I fixed it by adding a second bolt from the opposite side. I also ran into tolerance issues with the bearing and shaft, which required sanding down the metal shafts to make them fit. Despite all these challenges, none were impossible to fix or required a complete reprint. The final assembly is not as compact or light as I had hoped, but this can be greatly improved in future iterations. The gears could be smaller since my printer can still produce detailed small parts, and the batter I chose is overkill, so a smaller battery could easily meet the system's basic needs.

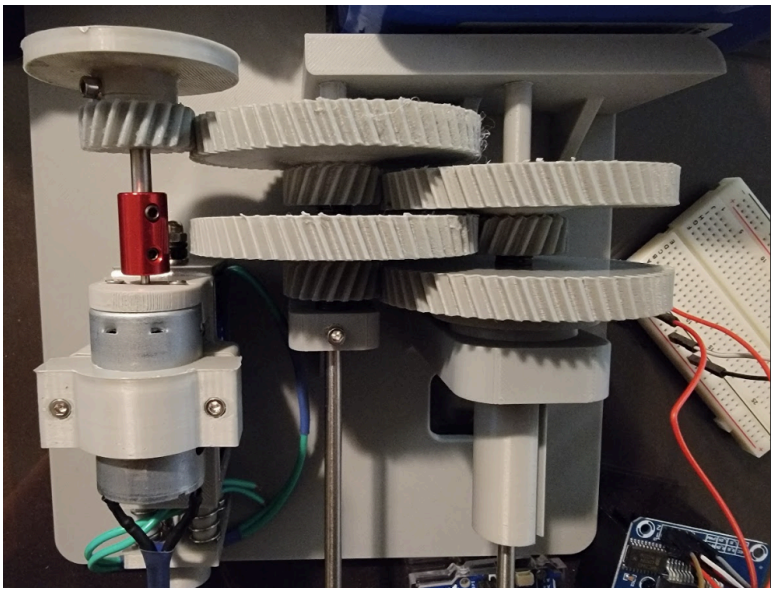


Figure 13: First iteration of the Input Gear (notice how bolt also split apart the gear)

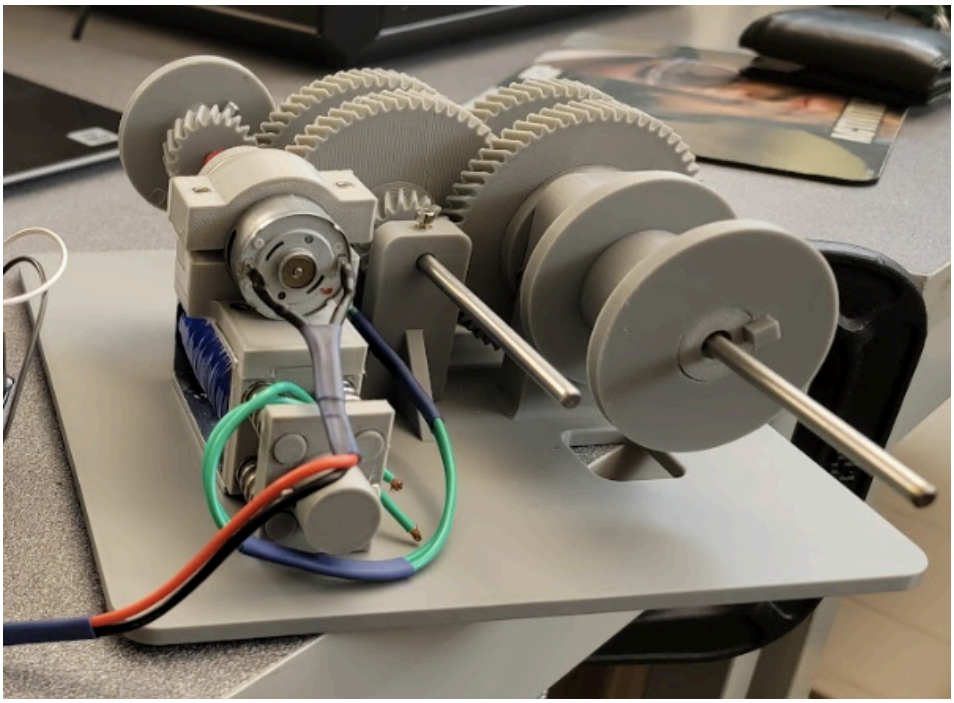


Figure 14: Second Iteration of the Input Gear with Two Allen Bolts (2nd bolt not visible in the picture)

Conclusion

The question I was aiming to answer was how compact, light, and strong I could make a payload hoist mechanism while still meeting all the requirements of a general mechatronic system. The process started with CAD in Solidoworks, then calculating the appropriate gear reductions to achieve the desired speed, and then finally printing and assembling all the parts. The payload mechanism worked by first having the Arduino read the potentiometer's analog voltage. Then it would turn it into a digital value and use that to output a PWM signal, which is what actually controls the motor speed.

The assembly ended up being more challenging than expected. For starters, there wasn't enough room for tools such as allen keys or my hands to add the needed spacers between the gears, the main motor mount body was too thin where the heat set inserts were pressed in and started to

deform when tightening the bolts for the top motor mount piece. Furthermore, the motor shaft was too short, so I had to use a shaft coupler to extend it. The input gear didn't line up properly at first but luckily reprinting the gear with a second allen bolt location on the opposite side fixed that. Bearings and shafts weren't perfectly toleranced either, so I had to sand the shaft down to fit. None of these problems were impossible to fix, and the mechanism ended up working relatively well as seen in the videos, even if the assembly isn't as compact or light as I'd like.

That being said, there's still a lot of room for improvement. The gears can be made smaller since my printer can print them while still retaining the same amount of detail. The battery I used was way overkill, so a smaller one could be used without affecting performance. Overall, this project showed how cading, gear calculations, 3D printing, and digital motor control can all come together to make a functioning hoist mechanism, and it really highlighted how important iteration and prior experience come into play when trying to make something small, light, and reliable.