**TCES430 – Microprocessors**

Final Project Report

Group Members:

*James Brewer*

*Igor Gonchar*

*Zach Martinez*

*Jake Nasonov*

June 10, 2015

Table of Contents:

Project Objective: 2

Description of System: 2

Hardware: 4

Button Interface 4

User Interface (UART) 4

Thermistor/Potentiometer Readings 5

Analog to Digital Converter 6

ADC Averaging System 7

Pulse Width Modulation 8

Variable/Constant Speed for DC Motor 8

DC Motor Control 8

Stepper Motor Control 9

Software: 10

Analysis: 11

Experiment: 12

Conclusion: 17

List of Figures:

Figure 1: 3

Figure 2: 3

Figure 3: 9

Figure 4: 13

# Project Objective:

The objective of this report was to measure the work accomplished by different motor types in a system under varying conditions, and to answer the question “Which motor is better for this application?” Our target was to produce a device that could control both DC & Stepper motors under varying loads. Our device should switch from a constant speed mode to a variable speed mode by the push of a toggle button. In the variable speed mode, the velocity of the motor varied according to a potentiometer reading. We then measured the work accomplished by both motors to numerically analyze their efficiencies. A secondary objective was to further analyze the varying interrupt latencies in our complex system.

# Description of System:

*As taken from our project instructions, the system will:*

* Provide a UART user interface consisting of a keyboard and monitor.
* Provide a button to switch modes between constant and variable speed for the motors.
* Control a bench-attached motor (of two types) that, through a pulley, lifts a weight a specific distance.
* Provide a potentiometer to an Analog to Digital Converter interface that will be used to control motor speed in the variable speed mode.
* Incorporate a Pulse Width Modulation interface to trigger the DC motor.
* Monitor the velocity of the DC motor via the Quadrature Encoder hardware, and produce the appropriate software interface.

The physical board that was used for this project was the Texas Instruments Tiva C Series TM4C1294 Connected Launchpad.

*A more visual description of our system is shown below in Figures 1 & 2.*

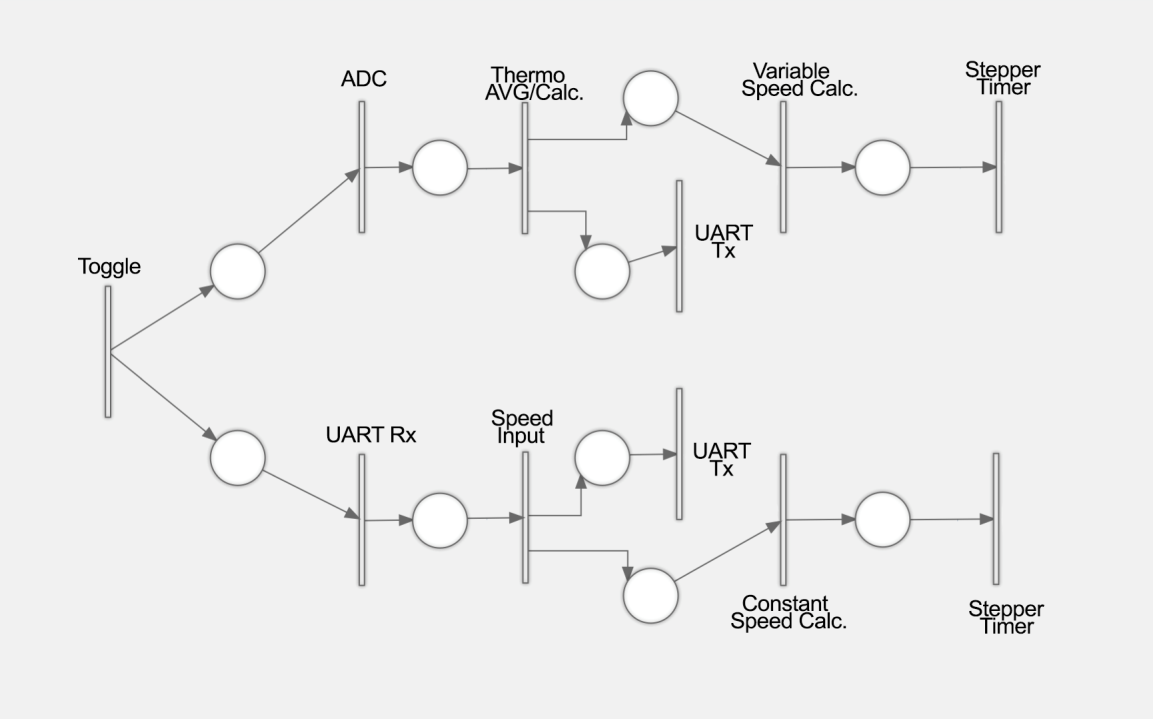
Stepper Motor: 

Figure 1: A Petri-Net diagram demonstrating the data path for our Stepper Motor.

DC Motor:

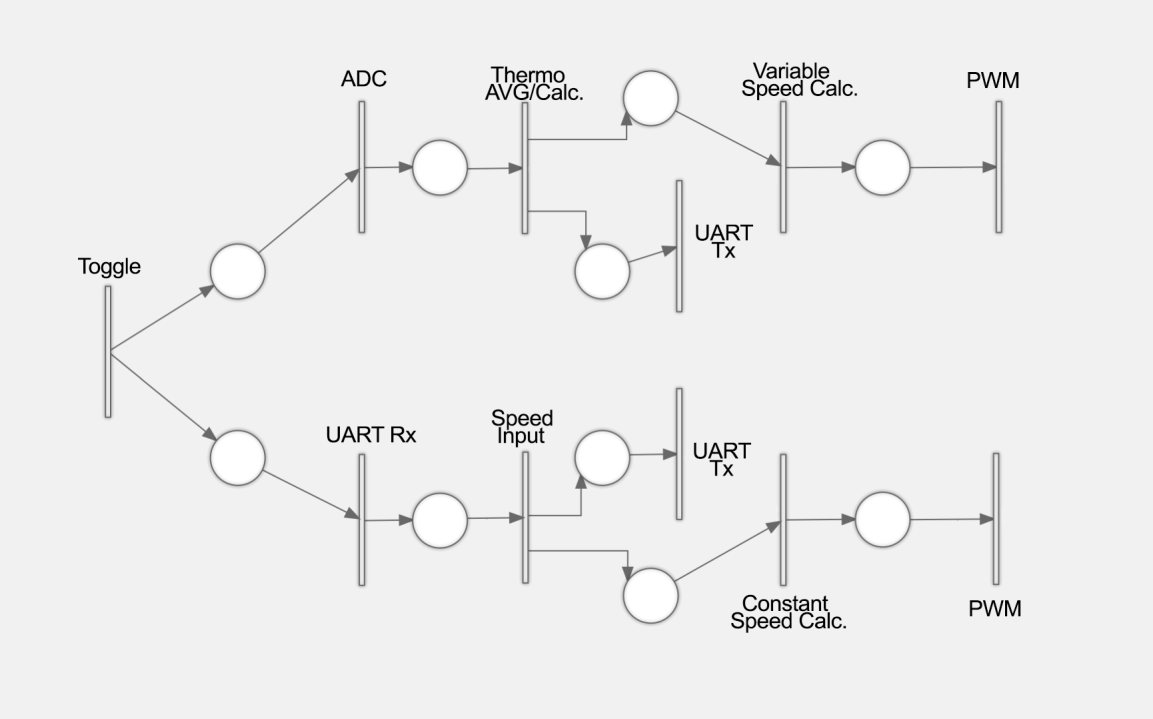


Figure 2: A Petri-Net diagram demonstrating the data path for our DC Motor.

*We will describe the hardware components of our system in more detail, following the order presented in the figures above, with the first system element being the toggle button interface.*

# Hardware:

## Button Interface

The first component in our system was a toggle button input. The button served as a toggle between two different operating modes for our motors. The software you either operate in a “constant mode” or in a “variable mode”. To accomplish this, we spent a great deal of time manually configuring the GPIO ports and a SysTick-enabled interrupt interface for the button. We configured the button to be falling-edge triggered (allowing the user to hold it, without experiencing any irregular behavior). As we began debugging our circuit, we noticed that our external button input would sense a voltage drop/rise from our voltage source, and toggle unpredictably. Therefore, we decided to switch over to the Tiva microcontroller’s onboard buttons. This gave us a much cleaner response and worked perfectly every time. We also considered the possibility of “button debounce” interfering with our system, however, that never came up as an issue in our testing.

## User Interface (UART)

For our console’s output system, we manually configured the UART interface instead of implementing the built-in driver code. This gave us a large amount of flexibility when it came to setting up our interrupts. We implemented several basic functions that built off of each other for ease of programming in our main project, starting with printing characters and building up to printing lines, strings, integers and doubles.

The type of system we implemented was that of the UART interrupt example in the textbook. As the UART brings values in from an outside source, it takes the values from the hardware FIFO and places them in a software FIFO to be used in the program. We limited user input to digits ‘0’ through ‘9’ and ‘Enter’. That way, it would only let users enter a number and not interfere with the program.

To handle the entering of numbers and the conversion to an actual integer variable, we added another software queue that, after pressing ‘Enter’ once, would convert each character to its number digit then multiply it by the power of 10. This was needed to put the digits in their proper decimal place.

Originally we never intended to provide double printing capability, but for low RPM values we needed to see the fractions. We were able to quickly implement a function that would take a double, cast a copy as an integer, and then multiply the decimal left by a power of 10 to get a decimal value from it. Then we would print the casted integer, a period, and then the multiplied decimal.

## Thermistor/Potentiometer Readings

Initially, we arranged our system to incorporate a thermistor as our variable-mode input for the motors. We set up the thermistor using a voltage divider circuit and used the internal analog to digital converter (ADC) system of the Tiva board to quantify the voltage into a digital value. Along with the thermistor, we could use a constant resistance of about to get the maximum range of voltage (0.380 V) in a simple voltage divider. To find this resistance value, we measured the voltage of the thermistor at two different readings: room temperature and figure-tip temperature.

Towards the end of the lab we were allowed to use a potentiometer, which made some things a lot easier. The biggest impact of this change was that the variable input signal became much cleaner and more stabilized. We just replaced the thermistor with a potentiometer and used the same circuit. We did modify the resister values again to better optimize the potentiometers input. Refer to the *Graph 1. Voltage Divider Range* (below) for a demonstration of how we came up with the best the resistor values for the potentiometer.

Graph 1: The range of the voltage compared to varying constant resistances for the potentiometer.

## Analog to Digital Converter

After we initialized the ADC, we began working on calculating a function that takes in the ADC value and converts it to the motors RPM. The function is shown below:

The ADC’s range was from 0 to 3100 but we want to use a % range, which is why we came up with this equation. We calculated that the motor would start rotating at its *Base RPM* of 160. From the first part of the function we get the values from 0 to 1 (double values) meaning that if we multiply it by 1000 we will get the percent at which the motor is spinning. Subtracting the base RPM from the 1000 we get our Offset RPM value, which tends to be 840.

For the potentiometer we had to use a slightly different equation:

We are taking the Average Value (Refer to ADC Averaging System) and divide it into the maximum range of the potentiometer, which is 3140, and multiply it by 1000, which is the percentage value. The general structure of our code for this part of the project was taken from our textbook and modified; the setup for the internal ADC will follow that of the example in section 8.5 of the textbook.

## ADC Averaging System

Initially we chose to use the exponential smoothing equation given in lecture, but in tests implementing the function, it didn’t operate as we intended. The ADC values continued to have a high amount of noise. Instead we chose to use a queue-style averaging system, inspired by the multiple-access queue system described in Section 8.5.5 of the textbook. Essentially it passes recorded values through an array like a queue, and then averages them. When keeping the potentiometer still, we wanted the values to stay constant rather than fluctuate with noise. Using an array of 10 values gave us a good balance between speed and consistency of voltage readings. When the motor was turned off, we only see a change of ±1 in the digital value. Sometimes when running the motors (especially the stepper motor at low speeds) we would see a much more intense amount of noise, sometimes from the minimum and maximum range of values.

When we tested the stepper motor system, we noticed minor amounts of voltage at ground, which was disconcerting. The power supply current reading was jumping around as well, which we inferred was causing the problem. We attempted to rectify the problem by increasing the number of connections to ground, essentially creating a larger “drain” for current. This drastically reduced our problem, and in combination with the averaging system, gave us much more consistent values.

## Pulse Width Modulation

The PWM (Pulse Width Modulation) is a function of the Tiva board, which uses a Generator Block to send a controlled duty cycle through one of the pins of the Tiva board for use with motor driven modules. Each generator block has two PWM output signals, which can be operated independently or as a pair of signals with dead band delays inserted (Texas Instrument Inc., 417). There is also a control block that determines the parity of the signals and which signals goes through which pins.

In general, the PWM is used to set a certain number of highs and lows in a given frequency range, such that the duty cycle of the frequency is set to that of the number of highs over the total frequency range. Thus, a DC motor can be controlled with this setup so that a desired control speed can be acquired. The output power of the DC motor depends entirely on the duty cycle of the PWM module, ranging from 0-100% of the duty cycle.

## Variable/Constant Speed for DC Motor

Constant speed for the DC motor switches the duty cycle of the PWM based on user input. The range used on our DC motor is 2-1000 where 2 is 0% and 1000 is 100%. The user value given by the user is passed into a formula: (59999\*value/1000) which gives us our duty cycle. The reason why we use 59999 instead of a frequency range of 60000 is because there cannot be a duty cycle of 100%, as such evidence by the example code given by Valvano where he states that the Cycle and the Duty Cycle cannot be the same. For variable speed it is essentially the same concept, just passing through the ADC average value instead of the user value.

## DC Motor Control

Integrating the DC motor took some initial testing and documentation to understand the PWM. First, we set up the PWM module so that we can test some values on the oscilloscope. From there we tested several duty cycles to understand how the PWM will work. After finishing that, we looked over more of the documentation for the PWM and did some more testing with new formulas and found out how to interact with the DC motor.

Lifting the bottle was the next step in our process of learning PWM. We did several tests with and without the weight attached. The documentation states that there needs to be at least .8v for the motor to spin on a 3.3v power supply. To expand on this, we converted to 5V and found that we needed at least a 25% duty cycle to spin the motor. When a bottle is attached, however, there is more force keeping the motor from spinning, so therefore we needed a higher value around 60% to have the motor spinning.

It is also important to note that we spent a great deal of time optimizing our pulley system to achieve the most efficient results. The way we had it initially set up involved the DC motor to rest on top of a gear that pulled the pulley upward. However, we found that this setup placed a great deal of weight on the gears, and involved a much larger force to overcome that initial static friction. We later implemented the gears in a different configuration (with the weight of the motor being supported by a platform), and found that our system became significantly more efficient.

## Stepper Motor Control

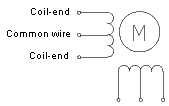


Figure 3: The coil/wire configurations in a unipolar stepper motor.

The stepper motor was a slightly more convenient device because of its inherent digital interface and ease of use. The Tiva Microcontroller is able to control both the position and velocity of the stepper motor in an open loop fashion. The stepper motor that we used for this project is a unipolar motor. A unipolar stepper motor has one winding with center tap per phase (as shown in the diagram above). Each section of windings is switched ON/OFF for each direction of magnetic field and only half of the electromagnets are energized at a time. In order to correctly activate the drive transistors, we ran our voltage signal through a L293 chip. The center tap was connected to the +V power source and the L293 chip controlled the four ends of the phases.

We initially resorted to a standard full-step sequence to run the stepper motor. In this method, the direction of the current in one of the coils is reversed in each step. We found this to cause the motor to jerk often (especially under low speeds). So we switched to a half-step sequence. In this method, the coil goes through a non-current state between reversals. This significantly smoothed out the motors operation. To spin the motor at a constant speed, the software outputs a sequence of 5-4-6-2-10-8-9-1 separated by a fixed time in-between. To calculate the motor speed, we used the following equation.

If is the time between outputs in seconds, and the motor has *n* steps per revolution, the motor speed will be in RPM.

# Software:

Interrupt Service Routines

When we initially designed our system, we intended to operate primarily via properly timed interrupt service routines. We quickly found that to be a much more challenging approach, however, we stuck with it nonetheless. Some modules already included basic service routines that we were able to expand to fit the project needs.

The ADC service routine simply activated an outside function. The toggle button service routine simply changed a boolean value when triggered. The UART service module was more than likely the most complex because it had to deal with three separate triggers, and had to handle incoming user key presses.

In terms of priority we set up the following order for both motor types: Motor Toggle > ADC > UART. For the stepper motor, we set the SysTick priority to be the highest, so the motor operation would take precedence.

Interface/Processing Modules

In the main program, a major part of the code involved printing to the UART. Instead of simply having it print out lines indefinitely, we wanted a cleaner display so we used a console refreshing system. We set up a while loop in the main program that would count, and after a certain count (determining the refresh rate) it would go through and print the necessary lines and data based on the mode of the motor. This gave us an extremely easy way to debug and display values in semi-real time. We had to keep the refresh rate fairly low because the capability of the Tiva board UART communication is somewhat limited in speed, and if set too fast, can’t keep up to print to the console accurately.

We connected values in the main program to the various modules through pointers, including a user input integer (for the constant speed mode), and a boolean (for the toggle button). This gave us a clean, user accessible interface.

For the variable mode ADC readings, we passed a function as a pointer to the ADC module which would process the reading each time the ADC interrupt was triggered.

# Analysis:

The approach the team took to integrate the various components into a single prototype consisted of a modular design. Each hardware/software component had an individual programming file and appropriate header file. We then integrated all of these files into a single project, and made sure to include them into the main file. This approach allowed us to be much more organized (when compared to writing all the code in a single main file). The modular coding technique saved a great deal of time during the debugging process, as each component could easily be accessed and modified as necessary.

Thankfully, with the modular approach, integrating the software modules went very smoothly. We were able to easily identify proper placement of the various functions from each module, and if we needed extra functionality from any of the modules it was very easy to add an extra function to that module’s code.

During the integration however, we added and tested one module at a time to ensure that it would work before adding the others in. We started with the UART module so we could read values without having to pause the program in CCS. Then, we added the ADC module to start taking readings within the overall system. The next module we added was the toggle button, so we could test switching modes. Then in each program we added the respective motor module(s) so we could test the speed/power changing functions.

# Experiment:

Our experimental setup consisted of the two motors, two Gatorade bottles, and lots of Legos. We constructed platforms for the motors to rest on, as well as groundings to secure the motors in place. Then we constructed and attached a pulley system to each motor at approximately table height. The photograph below demonstrates our experimental setup well.

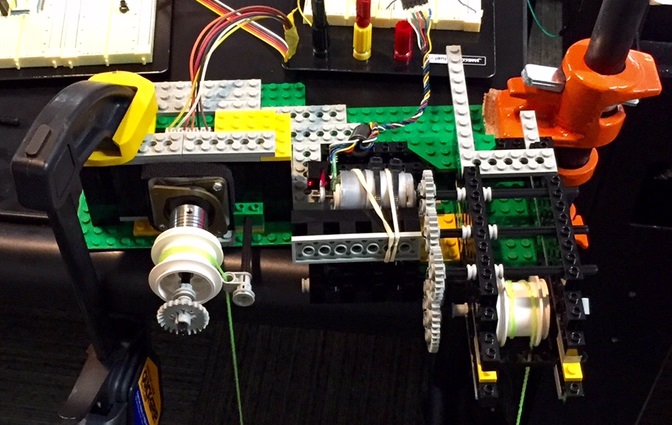


Figure 4: A photograph of our experimental setup

The data collection method that we use involved multiple trials with varying power and weight settings. On our two motors, we tested three different power outputs (80%, 90%, 100%), and multiple weight adjustments (50g, 60g, 75g, 100g, 110g). Overall, we tested approximately 36 different trials and configurations.

First we analyzed the stepper motor. We measured the height of the motor from the ground to be 0.4826 meters. This was used to calculate the work done by the motor when lifting the weight. The equation that we used to calculate the work,

(1)

Where 9.81m/s2 is the constant of gravity, and weight is measured in grams.

Next we needed to analyze the power that was being produced by the motor. To do this, we timed the duration of our motor’s lift, and divided our calculated work by that time.

(2)

Meanwhile, we always kept an eye out on our voltage source to see the intensity of current that was being passed through. We found that the stepper motor required less and less current as it rotated faster and faster (more on that later). To find the power drain in our system, we multiplied the current by the voltage applied (both of which are displayed on the power source). The knowledge in power drain allowed us to calculate the final motor efficiency.

(3)

The data table for our stepper motor is shown below, and to better visualize the data, we created a graph that compares the efficiency of the motor to the weights attached to the motor.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Weight | WORK | Time (s) | Power | Amps | PowDrain | EFC |
| 50 | 0.2367153 | 8.5 | 0.027848859 | 0.33 | 1.65 | 2% |
| 60 | 0.28405836 | 8.5 | 0.033418631 | 0.31 | 1.55 | 2% |
| 75 | 0.35507295 | 8.5 | 0.041773288 | 0.32 | 1.6 | 3% |
| 100 | 0.4734306 | 8.5 | 0.055697718 | 0.27 | 1.35 | 4% |
| 115 | 0.54444519 | 8.5 | 0.064052375 | 0.2 | 1 | 6% |
| 50 | 0.2367153 | 10 | 0.02367153 | 0.27 | 1.35 | 2% |
| 60 | 0.28405836 | 10 | 0.028405836 | 0.25 | 1.25 | 2% |
| 75 | 0.35507295 | 10 | 0.035507295 | 0.23 | 1.15 | 3% |
| 100 | 0.4734306 | 10 | 0.04734306 | 0.22 | 1.1 | 4% |
| 115 | 0.54444519 | 10 | 0.054444519 | 0.21 | 1.05 | 5% |
| 50 | 0.2367153 | 7.75 | 0.03054391 | 0.31 | 1.55 | 2% |
| 60 | 0.28405836 | 7.75 | 0.036652692 | 0.3 | 1.5 | 2% |
| 75 | 0.35507295 | 7.75 | 0.045815865 | 0.3 | 1.5 | 3% |
| 100 | 0.4734306 | 7.75 | 0.061087819 | 0.24 | 1.2 | 5% |
| 115 | 0.54444519 | 7.75 | 0.070250992 | 0.19 | 0.95 | 7% |

Table 1: This table demonstrates our experimental data for the stepper motor.

Graph : This graph allows for a visual representation of our data.

In some initial research of stepper motors, we found that stepper motors typically operate with better efficiency as the speed increases and more torque is added. Once we started the experiment and visualized our data, we can clearly see that expectation was true. At lower weight, the difference in efficiency was minimal, but as we increased the weight the difference in efficiency was much more noticeable. At the higher end of weight trials, the efficiency differs by 1% or more, which is somewhat significant given the margins.

Oddly enough however, the middle of the range for weights at a lift time of 8.5s dropped lower than the 10s lift time. One thing we noticed while running the motor is that after it continues to operate for some time, the current tended to saturate from the power supply, which is why we typically turned off the motor between trials. For a lower efficiency to come up like this though, it could be the case that the current in the motor coils might not have dissipated enough between the trials.

With those observations, we inferred that the cause of greater efficiency at higher speeds was due to less current saturation in the motor coils. At slow speeds, the current through coils has more time to saturate, and at high speeds the current doesn’t stay on each coil as long. When at a rest state and a held position, the current draw jumped dramatically, meaning the coils had saturated completely, and acted more like short circuits.

In terms of how the weight affected the motor, we suspected that at higher speeds the torque on the motor was reduced, which meant there was less resistance from lifting the weight.

The experimental process for the DC motor is very similar to that of the stepper motor.

Table 2: This table demonstrates the experimental data of the DC motor.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Weight | Work | % | Time | Power | Amps | PowDrain | EFC |
| 50 | 0.2117979 | 80 | 3.26 | 0.064968681 | 0.215 | 1.075 | 6.0% |
| 60 | 0.25415748 | 80 | 5.61 | 0.045304364 | 0.245 | 1.225 | 3.7% |
| 75 | 0.31769685 | 88.7 | 7.2 | 0.044124563 | 0.32 | 1.6 | 2.8% |
| 100 | 0.4235958 | 80 | 10.4 | 0.040730365 | 0.4 | 2.8 | 1.5% |
| 50 | 0.2117979 | 90 | 2.26 | 0.093715885 | 0.25 | 1.25 | 7.5% |
| 60 | 0.25415748 | 90 | 2.85 | 0.089178063 | 0.285 | 1.425 | 6.3% |
| 75 | 0.31769685 | 90 | 6.28 | 0.05058867 | 0.32 | 1.6 | 3.2% |
| 100 | 0.4235958 | 90 | 12.66 | 0.033459384 | 0.41 | 2.46 | 1.4% |
| 50 | 0.2117979 | 100 | 1.44 | 0.147081875 | 0.295 | 1.475 | 10.0% |
| 60 | 0.25415748 | 100 | 2.01 | 0.126446507 | 0.305 | 1.525 | 8.3% |
| 75 | 0.31769685 | 100 | 3.15 | 0.100856143 | 0.36 | 1.8 | 5.6% |
| 100 | 0.4235958 | 100 | 3.46 | 0.122426532 | 0.46 | 2.76 | 4.4% |
| 110 | 0.46595538 | 100 | 8.03 | 0.058026822 | 0.49 | 2.94 | 2.0% |

And again, to better visualize the data, the following graph represents our results for the DC motor.

Graph 3: This graph represents the data for the DC motor.

The results from the DC motor seemed much more apparent from our expectations. As the weight increased, the efficiency dropped significantly, while increasing the power given to the motor improved the efficiency.

We especially noticed that the motor struggled when increasing weight at the lower speeds, and even slightly improving the power to have the weight lift still showed a drop in efficiency. We couldn’t even test the 110g weight under 100% power.

We determined that the key factor in how well a DC motor operates is the torque on the motor. We especially noticed this when setting up the gear system. If the gears couldn’t move as freely, or there was too much friction, the motor was very sensitive to those changes. Standing alone, the DC motor needed 25% power to even start turning. When adding gears, the motor needed upwards of 40-55% power to start turning.

Even still, the efficiency of the DC motor at low weight and low speed was noticeably better than the stepper motor in those conditions.

# Conclusion:

When analyzing the results from both of the motors and comparing them, there are some noticeable differences in how they operate in certain conditions. Our results show the motors act opposite of each other.

Graph 4: This graph compares efficiency of both motor types.

At low weights and faster speed/higher available power, the DC motor is more efficient. At more weight and faster speeds, the stepper motor is more efficient. Still, the stepper motor could handle high weight at even the slowest speed, which shows that its operation is much less dependent on torque. The DC motor is highly dependent on torque, or lack thereof, in order to run smoothly if at all. When analyzing other points of data, such as the current draw, the stepper motor generally uses less than the DC motor.

Considering the use of motors to perform work in a system, having a greater efficiency when weight is introduced is a much more desirable factor. The DC motor has the highest efficiency when it is simply running on its own, which isn’t the intended use of a motor.

We believe that the stepper motor has a better range of operation, and can handle much greater demands without failing. Higher efficiency as more work is being done is definitely a logical choice. Therefore, that makes the stepper motor our better recommendation of the two.