

5-2018

Baseline Data from Servo Motors in a Robotic Arm for Autonomous Machine Fault Diagnosis

Jacob Brown

Follow this and additional works at: <http://scholarworks.uark.edu/meeguht>

 Part of the [Electro-Mechanical Systems Commons](#), [Other Engineering Commons](#), and the [Other Mechanical Engineering Commons](#)

Recommended Citation

Brown, Jacob, "Baseline Data from Servo Motors in a Robotic Arm for Autonomous Machine Fault Diagnosis" (2018). *Mechanical Engineering Undergraduate Honors Theses*. 72.
<http://scholarworks.uark.edu/meeguht/72>

This Thesis is brought to you for free and open access by the Mechanical Engineering at ScholarWorks@UARK. It has been accepted for inclusion in Mechanical Engineering Undergraduate Honors Theses by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, ccmiddle@uark.edu.

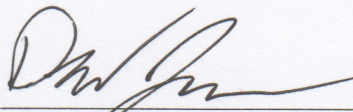
Baseline Data from Servo Motors in a Robotic Arm
for Autonomous Machine Fault Diagnosis

A thesis submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Mechanical Engineering

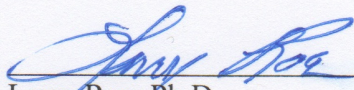
by

Jacob Brown

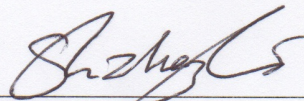
May 2018
University of Arkansas



David Jensen, Ph.D.
Thesis Director



Larry Roe, Ph.D.
Committee Member



Zhenghui Sha, Ph.D.
Committee Member

ABSTRACT

Fault diagnosis can prolong the life of machines if potential sources of failure are discovered and corrected before they occur. Supervised machine learning, or the use of training data to enable machines to discover these faults on their own, makes failure prevention much easier. The focus of this thesis is to investigate the feasibility of creating datasets of various faults at both the component and system level for a servomotor and a compatible robotic arm, such that this data can be used in machine learning algorithms for fault diagnosis. The faults induced at the component level in different servomotors include: low lubrication, no lubrication, two gears chipped, and four gears chipped. Each fault was also examined at 180, 135, 90, and 45-degree swings of the servo arm. Component level data was obtained using an Arduino microcontroller and a feedback wire in each servomotor to obtain the actual position of the servo arm, which allowed for the calculation of the difference in actual and theoretical position and the speed of the servo arm at the various faults. System level data was obtained using OptiTrack's motion tracking software, Motive, to track the position of two reflective markers on the hand of the robotic arm. At the component level, the low lubrication and no lubrication faults did not exhibit a large difference from the normal servomotor, whereas the servomotors with the gears chipped exhibited significant differences when compared to the normal servomotor. When evaluating the difference in position and speed of the servo arm at larger degree sweeps it was more evident that failure occurred, as opposed to the data at smaller degree sweeps. At the system level, the error was not as visible in the data as there wasn't much distinction between the speeds of the robotic arm's hand when the servomotors with faults were placed in it. The results of this work indicate that servomotors can be used to create fault behavior datasets at the component and system level that are usable for machine learning.

ACKNOWLEDGEMENTS

I would like to thank the Mechanical Engineering Department at the University of Arkansas for providing the equipment for the project and Dr. David Jensen, Dr. Zhenghui Sha, and Mr. Charlie DeStefano for helping with the experiment and with writing the thesis.

TABLE OF CONTENTS

INTRODUCTION 5

BACKGROUND..... 8

APPROACH..... 10

ANALYSIS 24

RESULTS 29

DISCUSSION 31

CONCLUSION 32

REFERENCES 33

APPENDIX 36

INTRODUCTION

The goal of this thesis is to determine the feasibility of gathering baseline data from a HITEC Servo, which could be used for autonomous machine fault diagnosis in the future. Before getting into the depth of this thesis it is best to define some key terms that will be used throughout. The first basic term is failure, which is defined by Smith (2017), as “non-conformance to some defined performance criterion.” A machine fault, however, as defined by Jayaswal et. al (2008) is “any change in a machinery part or component which makes it unable to perform its function satisfactorily.” Fault diagnosis is critical for all machines because if faults aren’t discovered and corrected, then machine failure can occur. Timing is also critical for fault diagnosis because undetected faults can upset production deadlines and cause heavy financial losses for companies that are dependent on these machines [3]. As such, the two main goals for fault diagnosis are to “prevent future failures and to ensure safety, reliability, and maintainability of machines” [4]. If the symptoms of the failure precede the actual failure itself, then one potential fix is machine learning [5]. Florez-Revuelta et. al (2016) define machine learning as a component of Artificial Intelligence that focuses on learning from data. Insights from data can permit the use of data-driven decision-making for the machine, which if applied to fault diagnosis, could be used to prolong machine failure [6]. There are two main categories of machine learning: supervised and unsupervised learning [6]. The difference between the categories is that supervised machine learning requires training data. There are many different algorithms within the two categories as well. Common supervised learning algorithms include Decision Trees, Naïve Bayes Classification, Least Squares Regression, Logistic Regression, Support Vector Machines, and Ensemble Methods [7]. Decision Trees are decision support tools, which are “tree like” models that narrow possible outcomes based on the answers to yes or no

questions [7]. Naïve Bayes Classification is a family of probabilistic classifiers based on Bayes Probability Theorem [7]. Least Squares Regression is a method of comparing data using an error metric that compares the data about a best-fit line, with the most popular type being a linear regression model [7]. Logistic Regression measures the relationship between a dependent variable and one or more independent variables by estimating probabilities using a cumulative logistic distribution [7]. Support Vector Machines is a binary classification algorithm that separates data into groups using straight lines [7]. Ensemble Methods create a set of classifiers and then classifies new data based on predictions [7]. Although this thesis is focused on examining the feasibility of obtaining training data for supervised learning, unsupervised learning is also beneficial. One of the most prominent types of unsupervised learning is clustering, where data is grouped into “clusters” based on similarity [7]. There are many different types of clustering such as k-means, centroid-based, connectivity-based, density-based, probabilistic, neural networks, and deep learning [7]. As defined by Nielsen (2017) a neural network is a “biologically-inspired programming paradigm, which enables a computer to learn from observational data” and deep learning is a set of techniques used for learning in neural networks. This thesis is focused more on supervised learning, or machine learning where training data is available. There are many machine learning dataset repositories online, such as the UC Irvine Machine Learning Repository, which have easily accessible training data [9]. There are not many existing datasets for failure data of machines at the system level, however. Though some do exist, such as research by Wienke et. al (2016) in which the authors created a dataset for a robotic system data based on various component faults. The focus of this thesis differs from that of the abovementioned research since the focus is on the feasibility of datasets being created

to look at failure data at both the component and the system level of a machine. The system is a robotic arm, with the faults being induced at the component level in the servomotors.

BACKGROUND

There does exist research on determining failure in machines at the system level with a variety of different approaches. Prognostic Health Management for electronics is one such method of assessing the life of an electronic, which uses “the integration of system state and the assessment of product survivability in deployed systems using non-destructive assessment of underlying damage” [11]. This Prognostic Health Management is a method that constantly evaluates the current “health” of the electronic in question in order to try to prevent failure [12]. According to The Electronic Prognostics and Health Management Research Center at the University of Maryland main approaches of Prognostic Health Management implementation include Built-in-Testing, use of expendable devices, “monitoring and reasoning of parameters that are precursors to impending failure”, and “modeling of stress and damage in electronic parts and structures utilizing exposure conditions” [12]. Built-in-testing as defined by Vichare et. al (2006) is an on-board hardware-software diagnostic means to identify and locate faults, and has various levels including circuit, module, and system level testing. The paper by Vichare et. al (2006) focused on electronics, so when referring to the use of expendable devices the paper mentioned fuses and other easily replaceable electronic devices. A failure precursor as defined by Vichare et. al (2006) is “a change in a measurable variable that can be associated with subsequent failure,” such as monitoring motor temperature in order to determine if the motor is about to fail. The measuring of stress and damage to electronic parts involves monitoring exposure conditions including usage, temperature, vibration, and radiation. Additionally, NASA has their own prognostic dataset repository on failure, with 16 different datasets on failure in various mechanical systems [13]. However, these datasets differ from the data in this thesis in that the systems tested were run until failure, whereas in this thesis the failure was induced prior

to testing. Similar to research done by Wienke et. al (2016) I will be inducing the failures. However, instead of inducing the failure at the component level and evaluating system performance as done in the previously mentioned paper, I induced the failure at the component level and evaluated the data at both the component and the system level. More specifically, I induced failures in a servomotor and then evaluated the performance of both the servomotor and the robotic arm that the servomotor was a part of. In order to design the experiment, it was important to identify the types of failure that I was anticipating. The major failure types as defined by Nandi et. al (2005) include stator faults with regard to function and connection, broken rotor bars, static and dynamic air-gap irregularities, bent shafts, shorted rotor field winding, and bearing failures, and gearbox failures.

APPROACH

There are a variety of reasons why servomotors fail, which can be grouped into two larger categories: mechanical and electrical. Mechanical faults are something physically wrong with the motor such as wear on the gears and bearings, lack of, or contaminated, lubricant where the gears are, out of balance rotors, contamination in the motor, excessive vibration, improper installation, moisture in the motor, etc. [14,15,16]. Electrical faults include faulty wiring, electrical degradation, a bad power supply, a low supply voltage, faulty electronic connections, etc. [14, 16]. The servomotor examined in this thesis is a HITEC HS-422 Deluxe Dual Ball Bearing Servo. The faults that were examined in this thesis include a lower voltage, low lubrication, no lubrication, chipped gear teeth, and chunks of gear teeth missing. These faults were chosen in order to diversify the causes of failure such as failures due to maintenance and mechanical errors. Improper lubrication typically results from an error in the maintenance of the motor and chipped gear teeth typically results from a mechanical fault such as an overloaded motor. See Tables 1 and 2 for matrices showing all of the faults that were tested at the component and system level, respectively. A new servomotor was used for each fault. Data collected from the servomotor and the robotic arm included position versus time and the speed of the servomotor and hand of the robotic arm. Datasets were generated for a normal servomotor and for each of the aforementioned fault types at both the component and system level. The number of trials necessary for machine learning varies. Previous experiments like that of Wienke et. al (2016) had a total of 10 trials for baseline data and 33 trials of fault data. A paper by Indira et. al (2010) focused on determining the sample size necessary for machine learning fault diagnosis using power analysis and found that at a sample size of about 10, the classification accuracy for machine learning is already at 85% and does not vary much, even with additional

trials. The total number of cycles analyzed for this thesis was 19 cycles for each component level failure and 20 cycles for each system failure.

Table 1: A matrix of all of the failure modes examined at the component level in this thesis

Not Tested	Tests				
Failure Modes	5V	3.3V	135-degree cycle	90-degree cycle	45-degree cycle
Normal (Control)	Tested	Tested	Tested	Tested	Tested
Low Lubrication	Tested	Tested	Tested	Tested	Tested
No Lubrication	Tested	Tested	Tested	Tested	Tested
2 Gear Teeth Chipped	Tested	Tested	Tested	Tested	Tested
4 Gear Teeth Chipped	Tested	Tested	Tested	Tested	Tested
Many Teeth Chipped	Not Tested	Not Tested	Not Tested	Not Tested	Not Tested

Table 2: A matrix of all of the failure modes examined at the system level in this thesis

	Tests					
Failure Modes	4 Second Cycle		2 Second Cycle		8 Second Cycle	
Normal (Control)	Tested	Tested	Tested	Tested	Tested	Tested
Low Lubrication	Tested	Tested	Tested	Tested	Tested	Tested
No Lubrication	Tested	Tested	Tested	Tested	Tested	Tested
2 Gear Teeth Chipped	Tested	Tested	Tested	Tested	Tested	Tested
4 Gear Teeth Chipped	Not Tested	Not Tested	Not Tested	Not Tested	Not Tested	Not Tested
Many Teeth Chipped	Not Tested	Not Tested	Not Tested	Not Tested	Not Tested	Not Tested

Experimental Setup

Before testing began on the servomotors, each servomotor needed to be modified to include a feedback wire that attaches to the potentiometer wire on the servomotor's circuit board [18]. Following the steps from the instructables site, see reference 18, the bottom cover of the servomotor was removed. A voltmeter was then used to determine which wire was the potentiometer wire and a feedback wire was soldered onto this wire [18]. The side of the

servomotor was then clipped in order to allow adequate room for the feedback wire to exit the side of the servomotor as seen in Figure 1 below.

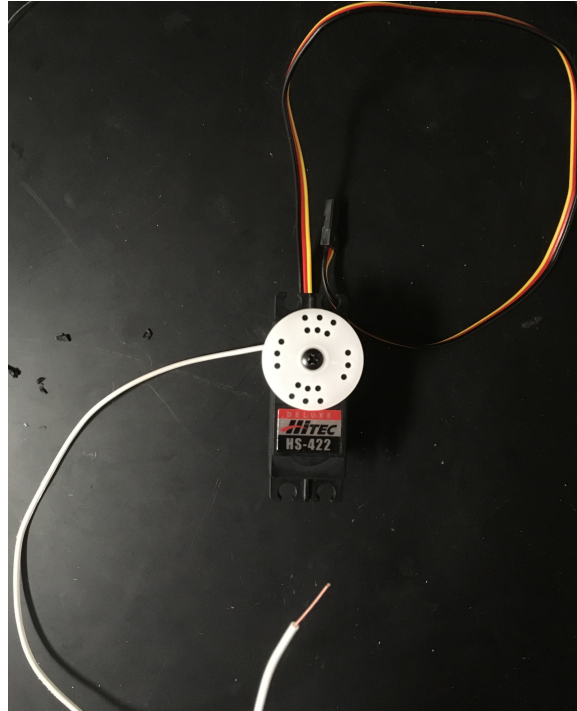


Figure 1: Image of a servomotor after the feedback wire, the white wire, was attached. The black, red, and yellow wire is the power cable for the servomotor.

Testing of the servomotor was conducted using an Arduino MEGA 2560 Microcontroller as seen in Figure 4. Before testing the servomotor, Arduino code was completed that made the servomotor complete a sweep. The basic code for the sweep was obtained from the Arduino Tutorial website [18]. This code enabled the servomotor to perform a basic sweep function, rotating 180 degrees with two-degree increments. This code was then modified to output the position of the servo arm, the current time of the code in milliseconds, and a column with either a positive or negative “1” indicating a positive or negative swing, respectively. Code was then

added from Trossen Robotics Community that outputted the actual servomotor position using the feedback wire from the potentiometer after calibration was accomplished at the beginning of the code [20]. Code for a servo arm swing from 0 degrees to 180 degrees can be seen in Figures 2 and 3 below. As seen in Table 1, each fault was tested at a 180, 135, 90, and 45-degree servo arm swing. Figures 15 through 20 in the Appendix show code for the 135, 90, and 45-degree swings that were also tested at each fault. The last test that was done at each fault was a 180 servo arm swing at a lower voltage of 3.3 Volts, which was achieved by simply moving a wire from the 5V pin to the 3.3V pin on the microcontroller seen in Figure 4.



```

Servo_Test_180_Degrees §
// Sweep
// by BARRAGAN <http://barraganstudio.com>
|
#include <Servo.h>
#include <VarSpeedServo.h>

VarSpeedServo myservo1; // create servo object to control a servo

int analogpin1 = 1; // pot sensor from servo 1

int calVal[] = {10, 100}; // initial calibration values
int angle = 0;
int pos = 0; // variable to store the servo position
int act_pos = 0;
unsigned long minutes = 60000; //Milliseconds in one minute
int pos_swing = 1;
int neg_swing = -1;

void setup()
{
  myservo1.attach(2); // attaches the servo on pin 2 to the servo object
  // myservo2.attach(3); // attaches the servo on pin 2 to the servo object
  // myservo3.attach(4); // attaches the servo on pin 2 to the servo object

  Serial.begin(9600);
  analogReference(EXTERNAL); // Use the external analog reference for voltage

  Serial.println("calibrating");
  myservo1.write(0); //Go to 0 degrees
  delay(1000); // wait 1 second for servo to reach the position
  calVal[0] = analogRead(analogpin1); // Calculate the Equivalent Minimum
  myservo1.write(180); //Go to 180 degrees
  delay(1000);
  calVal[1] = analogRead(analogpin1); // Calculate the Equivalent Maximum
  Serial.print("Cal values: "); //Print out the Calibration Values
  Serial.print(calVal[0]);
  Serial.print(",");
  Serial.println(calVal[1]);
}

```

Figure 2: Part 1 of the code for the 180-degree servo arm swing.

```

void loop()
{
  if (millis() < minutes * 3) //continue for 4 minutes
  {
    for(pos = 0; pos < 180; pos += 2) //goes from 20 degrees to 180 degrees
    {
      // in steps of 2 degrees
      myservo1.write(pos,180); // tell servo to go to position in variable 'pos'

      act_pos = analogRead(analogpin1);
      angle=map(act_pos, calVal[0], calVal[1], 0, 180);

      Serial.print(pos);
      Serial.print("\t");
      Serial.print(angle);
      Serial.print("\t");
      Serial.print(pos_swing);
      Serial.print("\t");
      Serial.println(millis());

      delay(50) ; // Delay for servo to reach the position
    }

    for(pos = 180; pos>=0; pos-= 2) // goes from 180 degrees to 0 degrees
    {
      myservo1.write(pos,180); // tell servo to go to position in variable 'pos'

      act_pos = analogRead(analogpin1);
      angle=map(act_pos, calVal[0], calVal[1], 0, 180);
      Serial.print(pos);
      Serial.print("\t");
      Serial.print("\t");
      Serial.print(angle);
      Serial.print("\t");
      Serial.print(neg_swing);
      Serial.print("\t");
      Serial.println(millis());

      delay(50) ; // waits 15ms for the servo to reach the position
    }
  }
  else
  {
    //Servo stops after 3 minutes have been reached
  }
}

```

Figure 3: Part 2 of the code for the 180-degree servo arm swing.

The servomotor was then attached to the breadboard and microcontroller similar to the setup in the Trossen Robotics Community instructions [20]. The voltage source was first divided using 1K and 2K-ohm resistors and is then connected to the AREF source on the microcontroller in order to divide the voltage by an internal 32K resistor within the microcontroller. The voltage division increased the range of feedback from the servomotor, which made the data acquired from the feedback wire more accurate [18]. As seen in Figure 4 the servomotor is connected to the microcontroller as follows: one end of a red wire is connected from the ground next to the AREF pin to column J and row 41 of the breadboard, a 1K-ohm resistor is then connected next to the ground wire with one end in column I and row 41 of the breadboard and the other end in column I and row 44. To complete the voltage divider the 2K-ohm resistor has one end in column H and row 44 and the other end in column H and row 49, a yellow wire is then connected to the AREF pin on the microcontroller and the other end in column G and row 49. A white wire is then connected next to the AREF wire in column F and row 49 with the other end connected to the 5V pin on the microcontroller. With the voltage divider connected, the servomotor was then connected to the microcontroller. As seen in the code above in Figure 2, the servomotor is “attached” to pin 2, so another yellow wire was connected to pin 2 on the microcontroller with the other end in column A and row 9 as seen in Figure 5 below. Above the yellow wire is a red wire to provide the input voltage for the servomotor with one end on the “Vin” pin on the microcontroller and the other end in column A and row 8 on the breadboard. The servomotor’s ground is then connected to the breadboard in column A and row 7 and then connected to the “GND” pin on the microcontroller above the “Vin” pin. The servomotor is then connected to the breadboard in column E in the same color arrangement as the other wires in column A. As seen in the code above in Figure 2 the analog pin is set equal to 1, so the feedback

wire from the servomotor is connected to analog pin 1 on the microcontroller as seen in Figure 4 below.

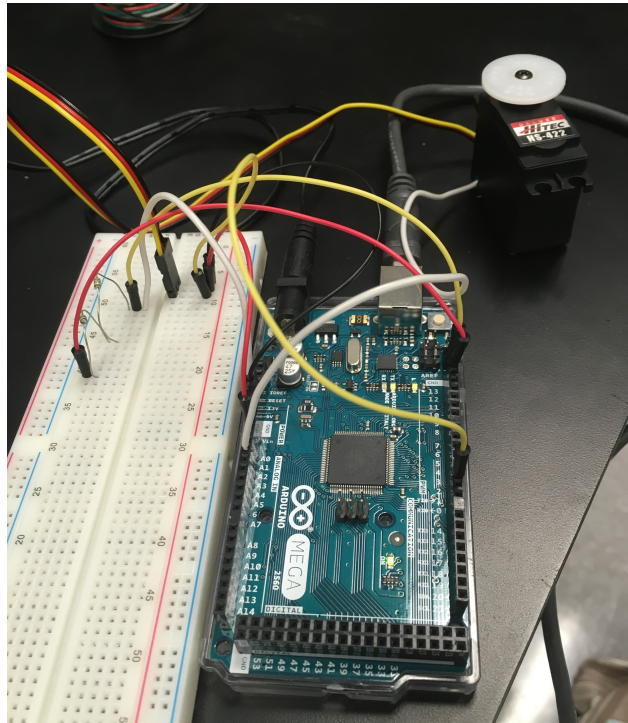


Figure 4: A top view of the servomotor connected to the microcontroller.

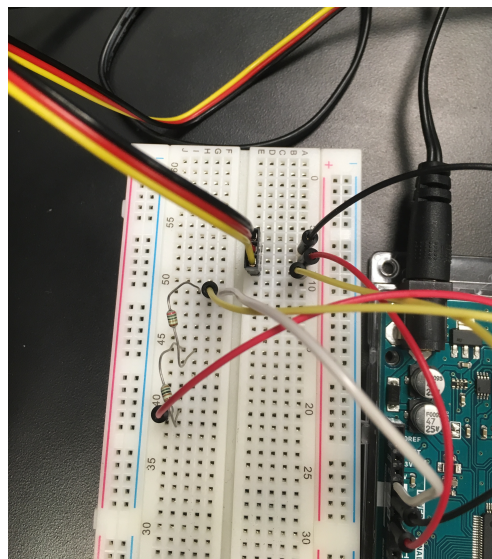


Figure 5: A top view of the breadboard.

In order to create the faults in the servomotors the top plate was first removed by first removing the four screws on the bottom of the servomotor. Figure 6 below depicts a normal servomotor with no faults induced. In order to achieve the low lubrication fault some of the lubrication was wiped away. The servomotor with no lubrication can be seen in Figure 7 below. Figures 8, 9, and 10 show servomotors with 2 gear teeth, 4 gear teeth, and many gear teeth chipped off, respectively, with the red arrows indicating the teeth that were chipped off. As seen in Tables 1 and 2 the servomotor with many gear teeth chipped off was not tested at the component or system level because it was too damaged to run.

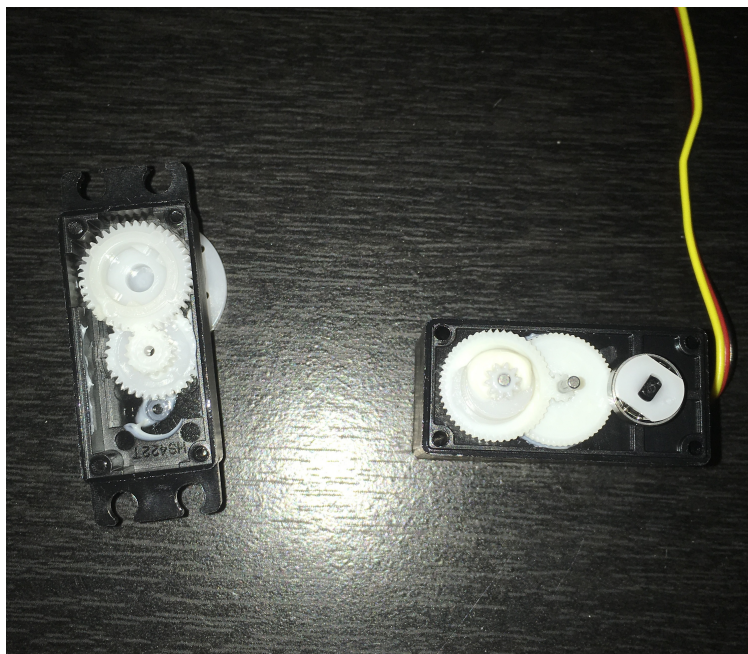


Figure 6: A normal servomotor with the top plate removed.

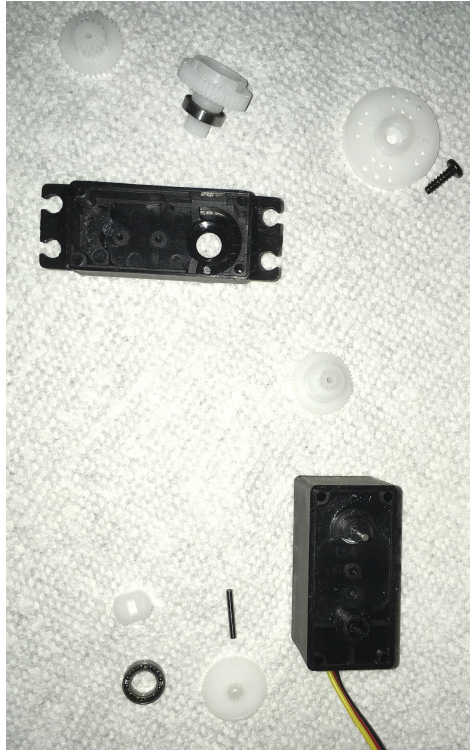


Figure 7: The servomotor with no lubrication.

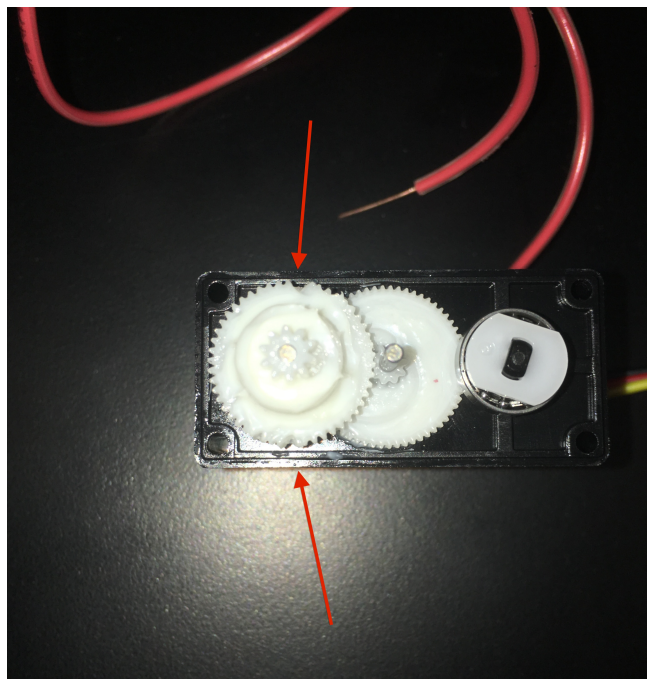


Figure 8: The servomotor with 2 gear teeth chipped.

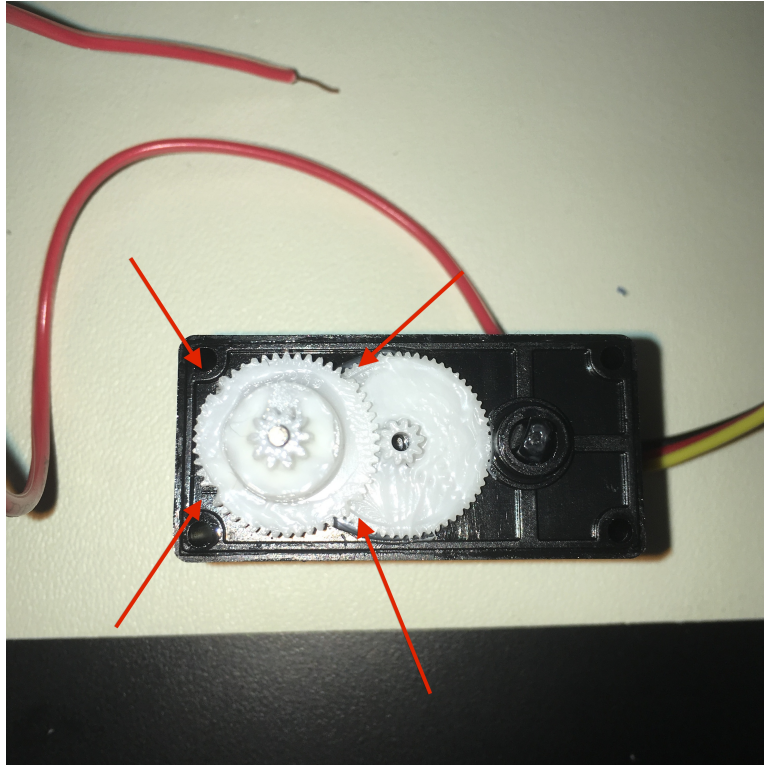


Figure 9: The servomotor with 4 gear teeth chipped.

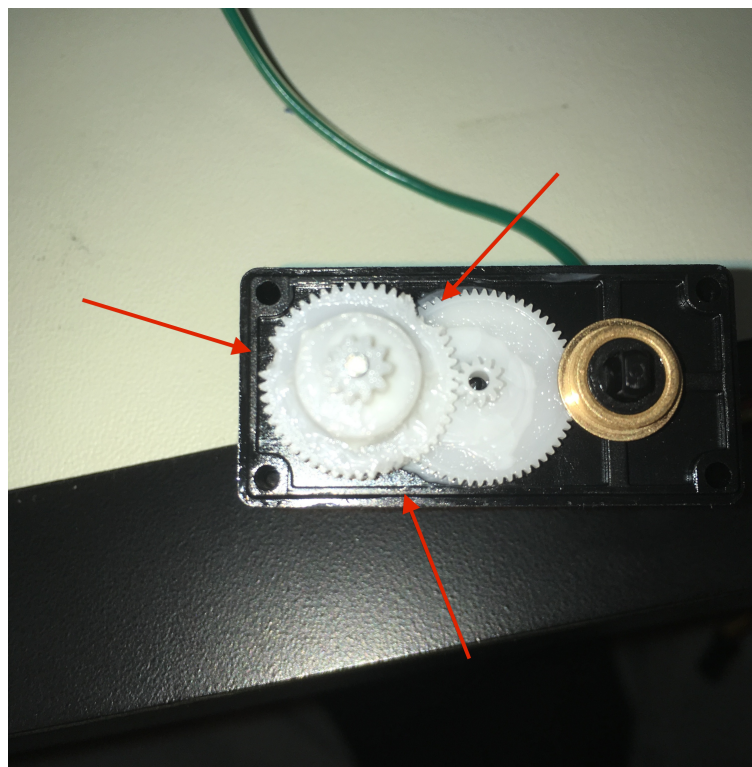


Figure 10: The servomotor with many gear teeth chipped.

The HS-422 Servomotor is in control of the “hand” of the robotic arm. In order to test whether a fault has occurred the speed of the hand at the various faults mentioned in Table 2 was determined. In order to track the position of the hand, OptiTrack Motion Capturing Software was used with six Prime 13W cameras positioned above the robotic arm. Before testing could begin calibration was necessary. In order to calibrate, the aim assist was first turned on and then each camera was switched into a gray scale video type and the gain, exposure, and LED settings were adjusted so the cameras had the best focus [21]. After each camera was adjusted, the cameras were switched back to object mode and all markers were removed and then any remaining extraneous reflections were “masked”, or removed, using the mask visible button in the Motive program [22]. The calibration options then were adjusted to ensure the correct OptiWand was selected and the “Start Wanding” button was clicked, and then the wand was waved in front of the cameras until enough samples were acquired to complete the wanding process. The final part of calibration was setting the ground plane, which was done by placing the calibration square on the ground selecting all of the markers of the calibration square and clicking the “Set Ground Plane” button [22]. After calibration was complete, the layout was switched to capture, and a reflective marker was placed on each “claw” on the hand of the robotic arm, as seen in Figure 11 below. Figure 12 is a screen capture of the Motive software with the two markers on the robotic arm. The robotic arm was then connected to a laptop computer and the Program SSC-32 Servo Sequencer was used to control the robotic arm, with each servomotor having the ability to be individually adjusted before adding a new frame. Figure 13 below shows the robotic arm connected to a laptop with the SSC-32 Servo Sequencer Program open.

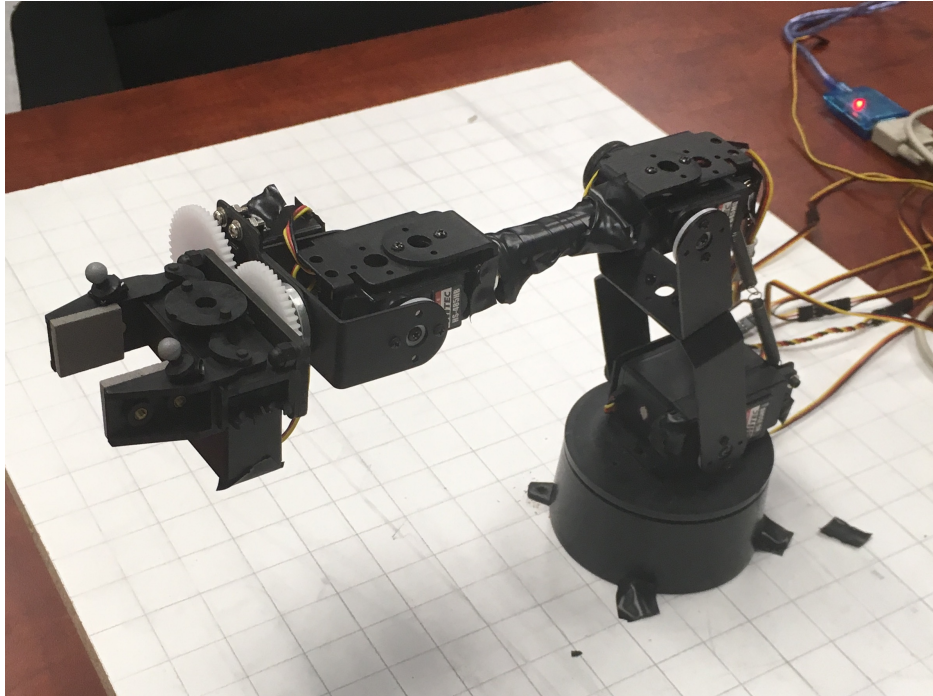


Figure 11: The robotic arm with reflective markers seen on the “hand”.

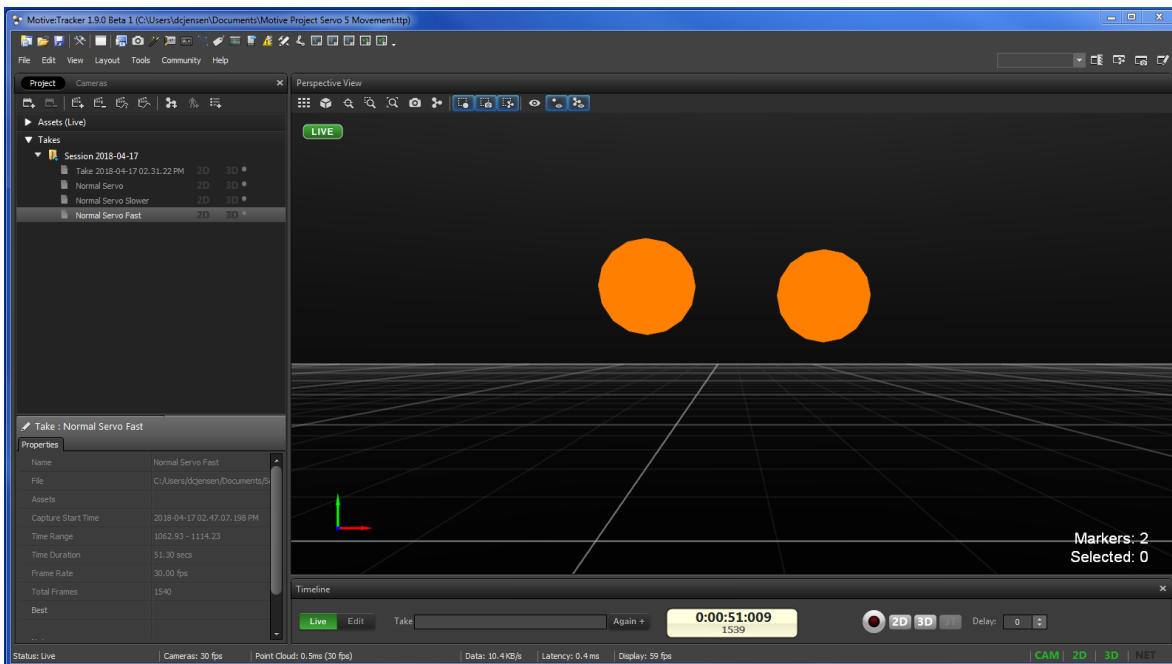


Figure 12: The Motive software with the two markers in the middle.

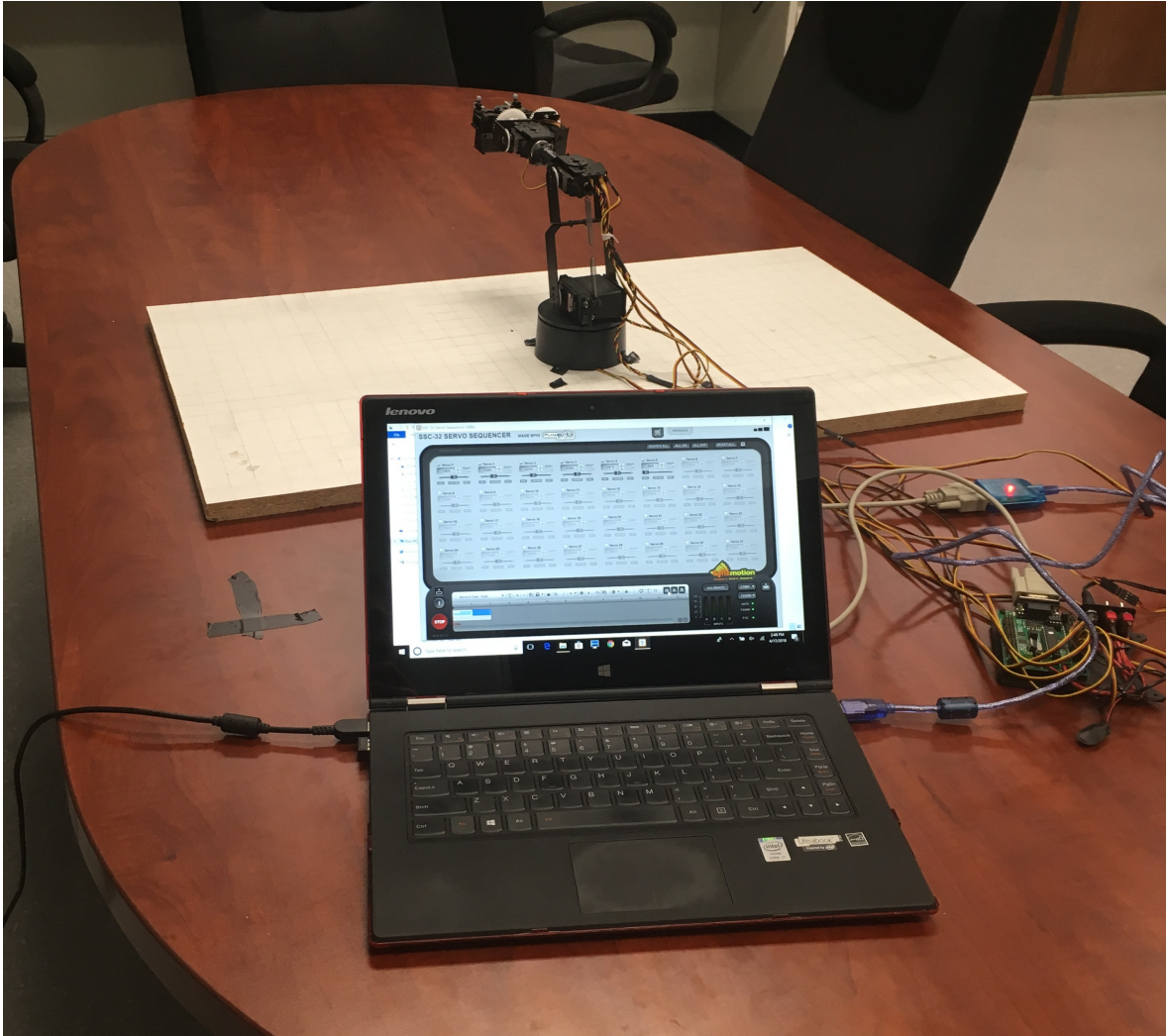


Figure 13: The robotic arm connected to a laptop with the SSC-32 Servo Sequencer Program.

ANALYSIS

At the component level the Arduino outputted four columns: one with the theoretical position, one with the actual position, one with the direction of the swing (either positive or negative “1”), and one with the time in milliseconds. The code ran for a total of 3 minutes. A column was added at the beginning to depict the current cycle, which is defined as the servo arm completing its sweep and then returning to its original position. Another column added was the difference in position between the theoretical and actual columns. The column after this is the adjusted difference in position, which is simply used to eliminate the “noisy” data, or data that is an outlier. If the difference in position is greater than 7, then an “if statement” function in excel was used to set the value equal to 7. The number of data points modified was monitored and can be seen in Table 10. Two more columns were added to display the time in seconds and minutes. The average difference in position, standard deviation of the difference in position, sample size, and number of adjusted cells for each test can be seen in Tables 3 through 7. The next column, J, is the difference between the current actual position and the previous actual position in order to obtain the total distance traveled by adding up all of the values in this column. However, due to “noisy” data in the actual position column, column J was also adjusted as seen in column K, where any difference between the current and previous actual position over 7 is reduced to a difference of 2, which corresponds to a normal difference in position, since the degrees increment by 2 degrees. Table 8 shows the speed of the servomotor in degrees per second when comparing the various faults at the different servo arm sweeps. Table 8 is the average speed of the 19 cycles that were completed for each fault; the values of the speed for each cycle can be found in the attached excel sheet with the dataset. Table 11 in the Appendix has the number of cells that were adjusted for each of the faults.

Table 3: The results for the difference in theoretical and actual position for each fault undergoing a 180-degree cycle

5 Volt - 180 Degree Cycle	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Average Difference in Position:	2.970	3.201	2.989	5.764	5.898
Standard Deviation, sigma:	1.299	1.293	1.255	1.910	1.841
Sample Size, n:	3439	3439	3439	3439	3439
Number of Adjusted Cells:	149	206	184	2254	2371

Table 4: The results for the difference in theoretical and actual position for each fault undergoing a 180-degree cycle at 3.3V

3.3 Volt - 180 Degree Cycle	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Average Difference in Position:	2.979	3.219	2.990	5.868	5.899
Standard Deviation, sigma:	1.125	1.260	1.171	1.824	1.830
Sample Size, n:	3439	3439	3439	3439	3439
Number of Adjusted Cells:	130	212	170	2300	2366

Table 5: The results for the difference in theoretical and actual position for each fault undergoing a 135-degree cycle

5 Volt - 135 Degree Cycle	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Average Difference in Position:	2.686	2.818	2.725	3.752	3.968
Standard Deviation, sigma:	1.166	1.212	1.222	1.904	1.921
Sample Size, n:	2539	2539	2539	2539	2539
Number of Adjusted Cells:	96	128	112	394	488

Table 6: The results for the difference in theoretical and actual position for each fault undergoing a 90-degree cycle

5 Volt - 90 Degree Cycle	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Average Difference in Position:	2.314	2.474	2.426	2.933	3.113
Standard Deviation, sigma:	1.074	1.250	1.237	1.809	1.760
Sample Size, n:	1639	1639	1639	1639	1639
Number of Adjusted Cells:	52	76	67	108	152

Table 7: The results for the difference in theoretical and actual position for each fault undergoing a 45-degree cycle

5 Volt - 45 Degree Cycle	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Average Difference in Position:	2.003	2.080	2.016	2.537	2.637
Standard Deviation, sigma:	0.909	0.820	0.831	1.422	1.701
Sample Size, n:	739	739	739	739	739
Number of Adjusted Cells:	1	3	3	35	47

Table 8: The speed of the servomotor in degrees per second for each fault

	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Average Speed at 5V (180 Degree Cycle):	40.156	39.478	40.149	25.540	19.581
Average Speed at 3.3V (180 Degree Cycle):	40.041	39.955	39.966	25.419	17.849
Average Speed for a 135 Degree Cycle:	31.476	30.353	31.162	32.181	30.481
Average Speed for a 90 Degree Cycle:	20.563	20.065	20.372	18.744	18.664
Average Speed for a 45 Degree Cycle:	9.905	9.887	9.858	9.340	8.648

Figure 11 shows the actual position of the normal servomotor versus the theoretical position for a 180-degree cycle at 5V for only the first cycle. Figures 21 through 25, in the Appendix, compare the actual position versus time for each of the faults at each of the 180-degree cycle 3.3V, 135-degree cycle, 90-degree cycle, and 45-degree cycle. The actual position is averaged across each cycle for each of these Figures. Figures 21 and 22 show that the 2 gear and 4 gear fault's actual position varied greatly from the normal servomotor in the 180-degree cycle. Figures 23 through 25 show that there is still some variability of the actual position between the chipped gear faults for the 135, 90, and 45-degree cycles, with a few spikes in position, but the chipped gears actual position is much closer to that of the normal servomotor when compared to the 180-degree cycles seen in Figures 21 and 22. In the Appendix, Figures 26 through 30 show the average difference in position versus time for one cycle for the 180-degree cycle at 5V, 180-degree cycle at 3.3V, 135-degree cycle, 90-degree cycle, and 45-degree cycle.

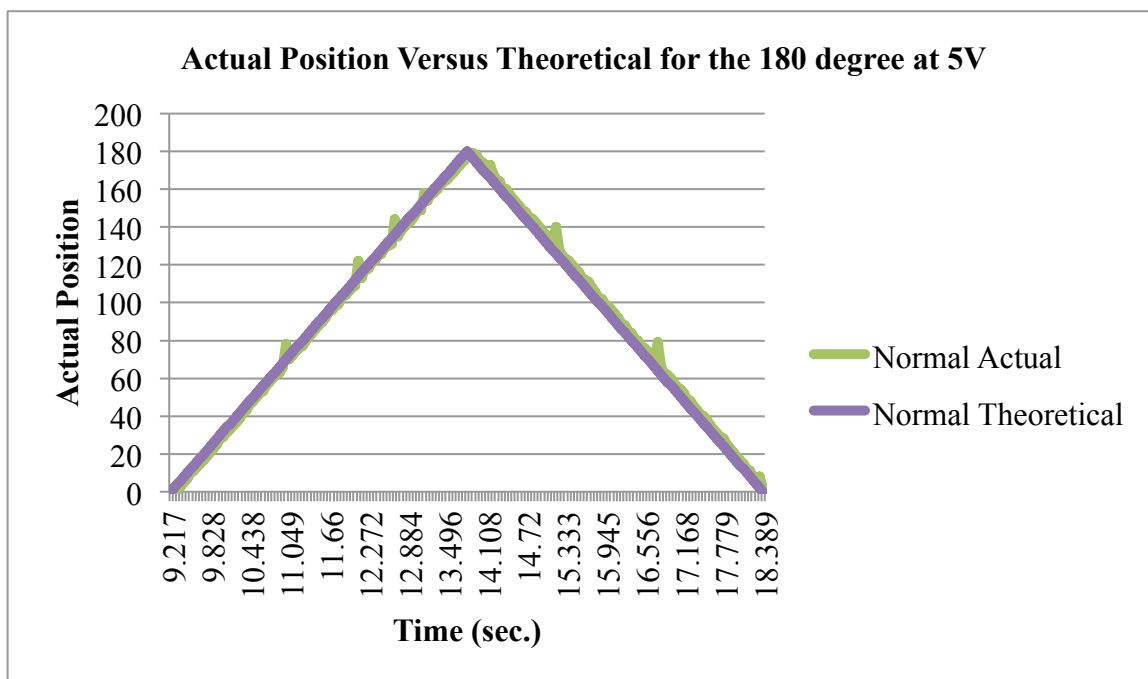


Figure 14: A graph of actual position versus the theoretical for the 180-degree cycle at 5V.

The dataset from the robotic arm consists of the frame, the current running time for the arm in seconds, the current cycle number, and the x, y, and z coordinates of each marker on the “hand” of the arm. The total distance between the markers was then computed using the distance formula [23]:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (1)$$

The speed was then computed for each cycle by dividing the total distance traveled throughout the cycle in meters by the total time for each cycle. This was repeated at three different speeds for each fault as described in Table 2 above. In the Appendix, Tables 12 through 15 consist of data on the speeds of the normal, low lubricant, no lubricant, and 2 gear chipped servomotors. Table 9 below has the average speed of the hand at each of the faults. The 4 gear chipped servomotor data did not output well from the Motive software and thus the speed wasn't computed.

Table 9: The speed of the hand with each servomotor in the robotic arm

	Normal	Low Lubrication	No Lubrication	2 Gear
Average Speed (m/s) of 4 Second Cycle:	1.151	1.160	1.108	1.122
Average Speed (m/s) of 2 Second Cycle:	1.142	1.160	1.115	1.094
Average Speed (m/s) of 8 Second Cycle:	1.154	1.157	1.106	1.116
Average Speed (m/s):	1.149	1.159	1.110	1.111

RESULTS

As seen in Tables 3 through 7 above, at the component level, the normal servomotor had the lowest difference in position followed by the low lubrication, no lubrication, 2 gear and 4 gear. At the lower cycles, such as the 45-degree cycle, the normal and improperly lubricated servomotors had very small differences between the average differences in position; however, the gear teeth faults still had a significant difference in position. As seen in Table 8, the speed differences were also apparent between normal servomotor and the chipped gear teeth servomotors at the 180-degree and 135-degree swings. However, at lower degree swings, the speed differences were not as pronounced. The majority of the error in the 2 gear and 4 gear servomotors can be attributed to the gears getting stuck in a certain position and not being able to continue rotating. Another indication of failure as mentioned in the introduction is a higher temperature, and both chipped gear servomotors were warm to the touch after running. At the system level the speed at the different faults did not vary much, with the normal servomotor having a speed of about 1.15 meters per second, and the servomotor with low lubrication having a slightly higher speed of 1.16 meters per second. The servomotor with no lubrication and the servomotor with 2 gear teeth chipped both had a slower average speed than the normal servomotor of 1.110 and 1.111 meters per second, respectively. The 180-degree cycle at the lower voltage of 3.3V exhibited similar values in terms of speed and difference in position to that of the 5V test, as seen in Tables 3, 4, and 8, indicating that the 3.3V wasn't a low enough voltage to induce any failure.

The number of cells that were adjusted as seen in Tables 10 and 11, in the Appendix, indicates that the normal servomotor had the least amount of adjusted cells with the low lubrication and no lubrication not being too far behind, but the 2 and 4 gear tooth chipped

servomotors had many more cells adjusted for both the difference in position and the difference between current and previous position, which was likely due to the servo arm getting stuck in certain positions.

DISCUSSION

While the faults did not show a large error at the system level, it was evident that at the component level the faults examined in this thesis showed a sign of some error in both the speed and the difference in position. If this topic was further researched, many improvements could be made to get better data. One improvement would be to repeat the experiment multiple times to ensure the data is precise and consistent. This would include soldering more servomotors in order to create multiple servomotors with the same fault and being able to test each of these servomotors to ensure the data from each fault is consistent. Another improvement would be to test more faults such as a higher voltage, moisture in the servomotor, contaminants in the motor, chipping a different gear, and a faulty ball bearing. Alterations could also be made in the Arduino code such as different degree increments, bigger servo arm swings, and more time for the servomotor to complete more cycles. Better motion tracking would also be better for testing at the system level since the OptiTrack cameras were not able to focus well at the markers on the robotic arm. Another improvement would be to test a different type of servomotor, such as one that is on a joint of the arm that moves more often, such as the “elbow” of the arm. The HS-422 was only used in the hand of the arm, whereas, a different model HITEC Servo would be compatible with another area of the arm.

CONCLUSION

The datasets attached to this thesis include both the raw data and the formatted data used to compare the faults to the normal servomotor. At the component level, the lower voltage, low lubrication, and no lubrication faults were not significantly different from the normal servomotor. The servomotors with the gear teeth chipped exhibited failure when compared to the normal servomotor at higher servo arm cycles. At the system level, however, the faults were not visible in the data because the speed of the robotic arm's hand when the servomotors with faults were placed in the robotic arm did not have any significant difference when compared to the speed with the normal servomotor in the hand. As a result of this experiment, inducing faults in the servomotors could be used to create datasets for evaluating component level faults, which could be incorporated into machine learning algorithms. However, due to the lack of variability in the data for the system level failures, the faults examined in this thesis would not be easily detected using machine learning algorithms at a system level. If further testing were to be done, such as testing new faults, then the servomotors could be used to create datasets that would evaluate the system level faults.

REFERENCES

1. Smith, David J.. (2017). Reliability, Maintainability and Risk - Practical Methods for Engineers (9th Edition) - 2.1 Defining Failure and Failure Modes. Elsevier. Online version available at: <https://app.knovel.com/hotlink/pdf/id:kt011CFIO2/reliability-maintainability/defining-failure-failure>
2. Pratesh Jayaswal, A. K. Wadhvani, and K. B. Mulchandani, "Machine Fault Signature Analysis," International Journal of Rotating Machinery, vol. 2008, Article ID 583982, 10 pages, 2008. doi:10.1155/2008/583982
3. S. Nandi, H. A. Toliyat and X. Li, "Condition monitoring and fault diagnosis of electrical motors-a review," in *IEEE Transactions on Energy Conversion*, vol. 20, no. 4, pp. 719-729, Dec. 2005.
doi: 10.1109/TEC.2005.847955
4. Pratesh Jayaswal, A. K. Wadhvani, and K. B. Mulchandani, "Machine Fault Signature Analysis," International Journal of Rotating Machinery, vol. 2008, Article ID 583982, 10 pages, 2008. doi:10.1155/2008/583982
5. A. Chigurupati, R. Thibaux and N. Lassar, "Predicting hardware failure using machine learning," *2016 Annual Reliability and Maintainability Symposium (RAMS)*, Tucson, AZ, 2016, pp. 1-6.
doi: 10.1109/RAMS.2016.7448033
6. Florez-Revuelta, Francisco Charaoui, Alexandros Andre. (2016). Active and Assisted Living - Technologies and Applications - 9.3 Active Daily Living. Institution of Engineering and Technology. Online version available at:
<https://app.knovel.com/hotlink/pdf/id:kt0110RUB9/active-assisted-living/active-daily-living>
7. Le, James. "The 10 Algorithms Machine Learning Engineers Need to Know." *KDnuggets Analytics Big Data Data Mining and Data Science*, KDnuggets, 11 July 2016, www.kdnuggets.com/2016/08/10-algorithms-machine-learning-engineers.html
8. Nielsen, Michael A. "Neural Networks and Deep Learning." *Neural Networks and Deep Learning*, Determination Press, Dec. 2017, neuralnetworksanddeeplearning.com/index.html
9. Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA: University of California, School of Information and Computer Science.
10. Wienke, Johannes, et al. "A Data Set for Fault Detection Research on Component-Based Robotic Systems." *Towards Autonomous Robotic Systems Lecture Notes in Computer*

Science, 2016, pp. 339–350., doi:10.1007/978-3-319-40379-3_35,
<https://pdfs.semanticscholar.org/7551/f90b70f67743a50d5d6b6b17f639c8104a82.pdf>

11. “Prognostic Health Management for Electronics.” *NSF Center for Advanced Vehicle and Extreme Environment Electronics*, Cave3 At Auburn University, cave.auburn.edu/rsrch-thrusts/prognostic-health-management-for-electronics.html.
12. N. M. Vichare and M. G. Pecht, "Prognostics and health management of electronics," in *IEEE Transactions on Components and Packaging Technologies*, vol. 29, no. 1, pp. 222-229, March 2006.
doi: 10.1109/TCAPT.2006.870387
13. “PCoE Datasets.” *NASA*, NASA, ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/, <https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/>
14. “Common Servo Motor Failures.” Fletcher Moorland,
<https://fletchermoorland.com/common-servo-motor-failures/>
15. “Why Motor Fails? Troubleshooting Your Motors.” *Servo Motor*,
www.servomotor.co/why-motor-fails-troubleshooting-your-motors-852763.html
16. York, Jade. “7 Common Reasons for a Servo Motor Breakdown.” *THE MACHINE RUNNER - a RepairZone Blog*, 3 May 2016, blog.repairzone.com/7-common-reasons-for-a-servo-motor-breakdown/.
17. V. Indira, R. Vasanthakumari, V. Sugumaran, Minimum sample size determination of vibration signals in machine learning approach to fault diagnosis using power analysis, *Expert Systems with Applications*, Volume 37, Issue 12, 2010, Pages 8650-8658, SSN 0957-4174, <https://0-doi-org.library.uark.edu/10.1016/j.eswa.2010.06.068>.
18. Instructables. “Servo Feedback Hack (Free).” *Instructables.com*, Instructables, 5 Nov. 2017, www.instructables.com/id/Servo-Feedback-Hack-free/
19. “Sweep.” *Arduino*, 18 Aug. 2015, www.arduino.cc/en/Tutorial/Sweep.
20. “Tutorial: Get Position Feedback from a Standard Hobby Servo.” *Trossen Robotics Community RSS*, 29 July 2009, forums.trossenrobotics.com/tutorials/how-to-diy-128/get-position-feedback-from-a-standard-hobby-servo-3279/
21. “Aiming and Focusing.” *Aiming and Focusing - NaturalPoint Product Documentation Ver 2.0*, OptiTrack, 1 Mar. 2018,
v20.wiki.optitrack.com/index.php?title=Aiming_and_Focusing#Aim_Assist_Button
22. “Calibration.” *Calibration - NaturalPoint Product Documentation Ver 2.0*, OptiTrack, 8 Feb. 2018, v20.wiki.optitrack.com/index.php?title=Calibration

23. Lambers, Jim. "Three-Dimensional Coordinate Systems." University of Southern Mississippi, 2010, <http://www.math.usm.edu/lambers/mat169/fall09/lecture17.pdf>

APPENDIX

```

// Sweep
// by BARRAGAN <http://barraganstudio.com>

#include <Servo.h>
#include <VarSpeedServo.h>

VarSpeedServo myservo1; // create servo object to control a servo
                        // a maximum of eight servo objects can be created
//Servo myservo2;
//Servo myservo3;

int analogpin1 = 1; // pot sensor from servo 1
//int analogpin2 = 2; // pot sensor from servo 2
//int analogpin3 = 3; // pot sensor from servo 3

int calVal[] = {10, 100}; // initial calibration values
int angle = 0;
int pos = 0; // variable to store the servo position
int act_pos = 0;
unsigned long minutes = 60000; //Milliseconds in one minute
int pos_swing = 1;
int neg_swing = -1;

void setup()
{
  myservo1.attach(2); // attaches the servo on pin 2 to the servo object
  // myservo2.attach(3); // attaches the servo on pin 2 to the servo object
  // myservo3.attach(4); // attaches the servo on pin 2 to the servo object

  Serial.begin(9600);
  analogReference(EXTERNAL); // Use the external analog reference for voltage

  Serial.println("calibrating");
  myservo1.write(22.5); //Go to 20 degrees
  delay(1000); // wait 1 second for servo to reach the position
  calVal[0] = analogRead(analogpin1); // Calculate the Equivalent Minimum
  myservo1.write(157.5); //Go to 160 degrees
  delay(1000);
  calVal[1] = analogRead(analogpin1); // Calculate the Equivalent Maximum
  Serial.print("Cal values: "); //Print out the Calibration Values
  Serial.print(calVal[0]);
  Serial.print(",");
  Serial.println(calVal[1]);
}

void loop()
{
  if (millis() < minutes * 3) //continue for 4 minutes
  {
    for(pos = 22.5; pos < 157.5; pos += 2) //goes from 20 degrees to 160 degrees
    {
      // in steps of 2 degrees
    }
  }
}

```

Figure 15: Part 1 of the code for the 135-degree servo arm swing.

```

{
    myservo1.write(pos,157.5);          // in steps of 2 degrees
                                        // tell servo to go to position in variable 'pos'

    act_pos = analogRead(analogpin1);
    angle=map(act_pos, calVal[0], calVal[1], 22.5, 157.5);

    Serial.print(pos);
    Serial.print("\t");
    Serial.print(angle);
    Serial.print("\t");
    Serial.print(pos_swing);
    Serial.print("\t");
    Serial.println(millis());

    delay(66.67) ;                      // Delay for servo to reach the position
}

for(pos = 157.5; pos>=22.5; pos-= 2) // goes from 180 degrees to 0 degrees
{
    myservo1.write(pos,157.5);          // tell servo to go to position in variable 'pos'

    act_pos = analogRead(analogpin1);
    angle=map(act_pos, calVal[0], calVal[1], 22.5, 157.5);
    Serial.print(pos);
    Serial.print("\t");
    Serial.print("\t");
    Serial.print(angle);
    Serial.print("\t");
    Serial.print(neg_swing);
    Serial.print("\t");
    Serial.println(millis());

    delay(66.67) ;                      // waits 15ms for the servo to reach the position
}
}
else
{
    //Servo stops after 3 minutes have been reached
}
}

```

Figure 16: Part 2 of the code for the 135-degree servo arm swing.

```

// Sweep
// by BARRAGAN <http://barraganstudio.com>

#include <Servo.h>
#include <VarSpeedServo.h>

VarSpeedServo myservo1; // create servo object to control a servo
                        // a maximum of eight servo objects can be created
//Servo myservo2;
//Servo myservo3;

int analogpin1 = 1; // pot sensor from servo 1
//int analogpin2 = 2; // pot sensor from servo 2
//int analogpin3 = 3; // pot sensor from servo 3

int calVal[] = {10, 100}; // initial calibration values
int angle = 0;
int pos = 0; // variable to store the servo position
int act_pos = 0;
unsigned long minutes = 60000; //Milliseconds in one minute
int pos_swing = 1;
int neg_swing = -1;

void setup()
{
  myservo1.attach(2); // attaches the servo on pin 2 to the servo object
  // myservo2.attach(3); // attaches the servo on pin 2 to the servo object
  // myservo3.attach(4); // attaches the servo on pin 2 to the servo object

  Serial.begin(9600);
  analogReference(EXTERNAL); // Use the external analog reference for voltage

  Serial.println("calibrating");
  myservo1.write(45); //Go to 20 degrees
  delay(1000); // wait 1 second for servo to reach the position
  calVal[0] = analogRead(analogpin1); // Calculate the Equivalent Minimum
  myservo1.write(135); //Go to 160 degrees
  delay(1000);
  calVal[1] = analogRead(analogpin1); // Calculate the Equivalent Maximum
  Serial.print("Cal values: "); //Print out the Calibration Values
  Serial.print(calVal[0]);
  Serial.print(",");
  Serial.println(calVal[1]);
}

void loop()
{
  if (millis() < minutes * 3) //continue for 3 minutes
  {
    for(pos = 45; pos < 135; pos += 2) //goes from 20 degrees to 160 degrees
    {
      // in steps of 2 degrees

```

Figure 17: Part 1 of the code for the 90-degree servo arm swing.

```

for(pos = 18; pos <= 135; pos += 2) // goes from 18 degrees to 135 degrees
{
  myservo1.write(pos,135);          // tell servo to go to position in variable 'pos'

  act_pos = analogRead(analogpin1);
  angle=map(act_pos, calVal[0], calVal[1], 45, 135);

  Serial.print(pos);
  Serial.print("\t");
  Serial.print(angle);
  Serial.print("\t");
  Serial.print(pos_swing);
  Serial.print("\t");
  Serial.println(millis());

  delay(100) ;                      // Delay for servo to reach the position
}

for(pos = 135; pos >= 45; pos -= 2) // goes from 180 degrees to 0 degrees
{
  myservo1.write(pos,135);          // tell servo to go to position in variable 'pos'

  act_pos = analogRead(analogpin1);
  angle=map(act_pos, calVal[0], calVal[1], 45, 135);
  Serial.print(pos);
  Serial.print("\t");
  Serial.print("\t");
  Serial.print(angle);
  Serial.print("\t");
  Serial.print(neg_swing);
  Serial.print("\t");
  Serial.println(millis());

  delay(100) ;                      // waits 15ms for the servo to reach the position
}
}
else
{
  //Servo stops after 3 minutes have been reached
}
}

```

Figure 18: Part 2 of the code for the 90-degree servo arm swing.


```

// Sweep
// by BARRAGAN <http://barraganstudio.com>

#include <Servo.h>
#include <VarSpeedServo.h>

VarSpeedServo myservo1; // create servo object to control a servo
                        // a maximum of eight servo objects can be created
//Servo myservo2;
//Servo myservo3;

int analogpin1 = 1; // pot sensor from servo 1
//int analogpin2 = 2; // pot sensor from servo 2
//int analogpin3 = 3; // pot sensor from servo 3

int calVal[] = {10, 100}; // initial calibration values
int angle = 0;
int pos = 0; // variable to store the servo position
int act_pos = 0;
unsigned long minutes = 60000; //Milliseconds in one minute
int pos_swing = 1;
int neg_swing = -1;

void setup()
{
  myservo1.attach(2); // attaches the servo on pin 2 to the servo object
  // myservo2.attach(3); // attaches the servo on pin 2 to the servo object
  // myservo3.attach(4); // attaches the servo on pin 2 to the servo object

  Serial.begin(9600);
  analogReference(EXTERNAL); // Use the external analog reference for voltage

  Serial.println("calibrating");
  myservo1.write(67.5); //Go to 20 degrees
  delay(1000); // wait 1 second for servo to reach the position
  calVal[0] = analogRead(analogpin1); // Calculate the Equivalent Minimum
  myservo1.write(112.5); //Go to 160 degrees
  delay(1000);
  calVal[1] = analogRead(analogpin1); // Calculate the Equivalent Maximum
  Serial.print("Cal values: "); //Print out the Calibration Values
  Serial.print(calVal[0]);
  Serial.print(",");
  Serial.println(calVal[1]);
}

void loop()
{
  if (millis() < minutes * 3) //continue for 3 minutes
  {
    for(pos = 67.5; pos < 112.5; pos += 2) //goes from 20 degrees to 160 degrees
    {
      // in steps of 2 degrees
    }
  }
}

```

Figure 19: Part 1 of the code for the 45-degree servo arm swing.

```

{
    myservo1.write(pos,112.5);          // in steps of 2 degrees
                                        // tell servo to go to position in variable 'pos'

    act_pos = analogRead(analogpin1);
    angle=map(act_pos, calVal[0], calVal[1], 67.5, 112.5);

    Serial.print(pos);
    Serial.print("\t");
    Serial.print(angle);
    Serial.print("\t");
    Serial.print(pos_swing);
    Serial.print("\t");
    Serial.println(millis());

    delay(200) ;                        // Delay for servo to reach the position
}

for(pos = 112.5; pos>=67.5; pos-= 2) // goes from 180 degrees to 0 degrees
{
    myservo1.write(pos,112.5);          // tell servo to go to position in variable 'pos'

    act_pos = analogRead(analogpin1);
    angle=map(act_pos, calVal[0], calVal[1], 67.5, 112.5);
    Serial.print(pos);
    Serial.print("\t");
    Serial.print("\t");
    Serial.print(angle);
    Serial.print("\t");
    Serial.print(neg_swing);
    Serial.print("\t");
    Serial.println(millis());

    delay(200) ;                        // waits 15ms for the servo to reach the position
}
}
else
{
    //Servo stops after 3 minutes have been reached
}
}

```

Figure 20: Part 2 of the code for the 45-degree servo arm swing.

Table 10: The number of adjusted cells in the difference in position for each fault due to the data being “noisy”.

	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Number of Adjusted Cells at 5V (180 Degree Cycle):	149	206	184	2254	2371
Number of Adjusted Cells at 3.3V (180 Degree Cycle):	130	212	170	2300	2366
Number of Adjusted Cells for a 135 Degree Cycle:	96	128	112	394	488
Number of Adjusted Cells for a 90 Degree Cycle:	52	76	67	108	152
Number of Adjusted Cells for a 45 Degree Cycle:	1	3	3	149	149

Table 11: The number of adjusted cells in the difference between the current and previous position to compute the total degrees traveled for each fault due to the data being “noisy”.

	Normal	No Lubrication	Low Lubrication	2 Gear	4 Gear
Number of Adjusted Cells at 5V (180 Degree Cycle):	330	425	374	465	522
Number of Adjusted Cells at 3.3V (180 Degree Cycle):	336	425	353	374	559
Number of Adjusted Cells for a 135 Degree Cycle:	176	264	202	474	515
Number of Adjusted Cells for a 90 Degree Cycle:	70	118	97	124	145
Number of Adjusted Cells for a 45 Degree Cycle:	2	6	8	42	51

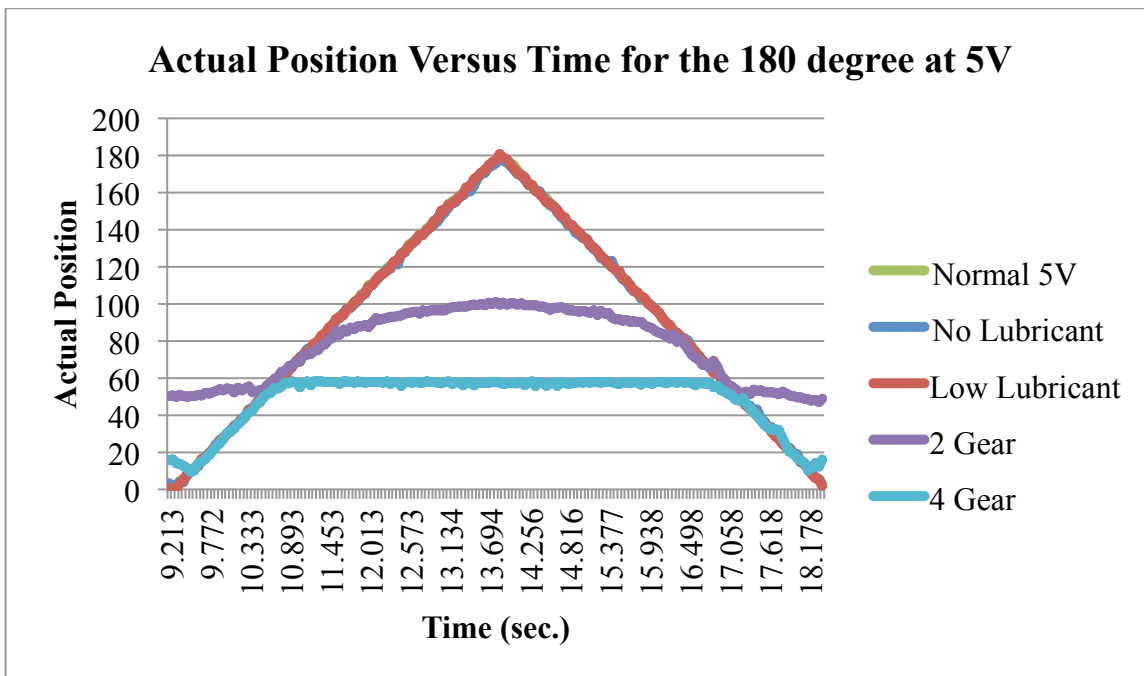


Figure 21: A graph of the actual position versus time with each of the faults for the 180-degree cycle at 5V.

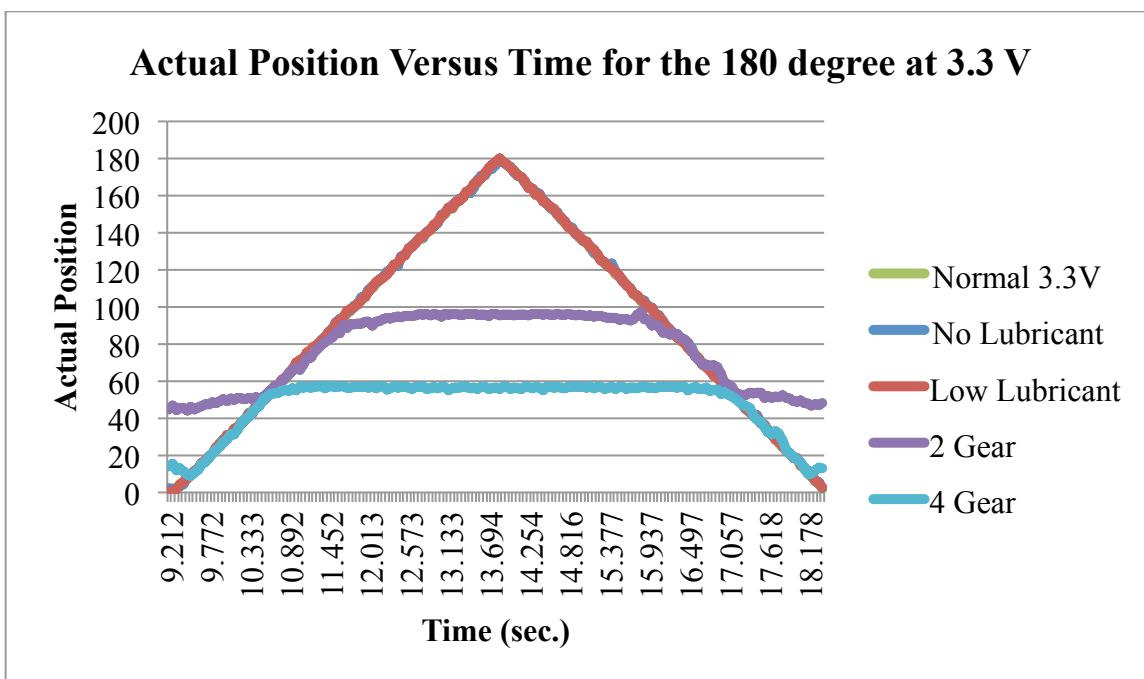


Figure 22: A graph of the actual position versus time with each of the faults for the 180-degree cycle at 3.3V.

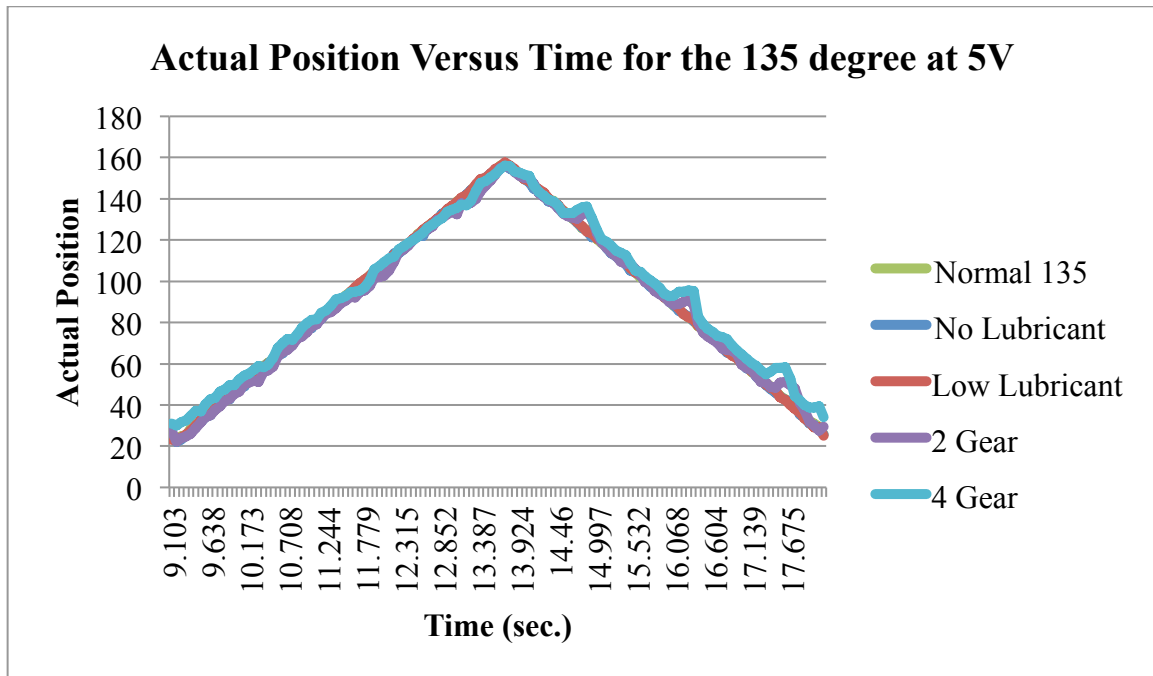


Figure 23: A graph of the actual position versus time with each of the faults for the 135-degree cycle at 5V.

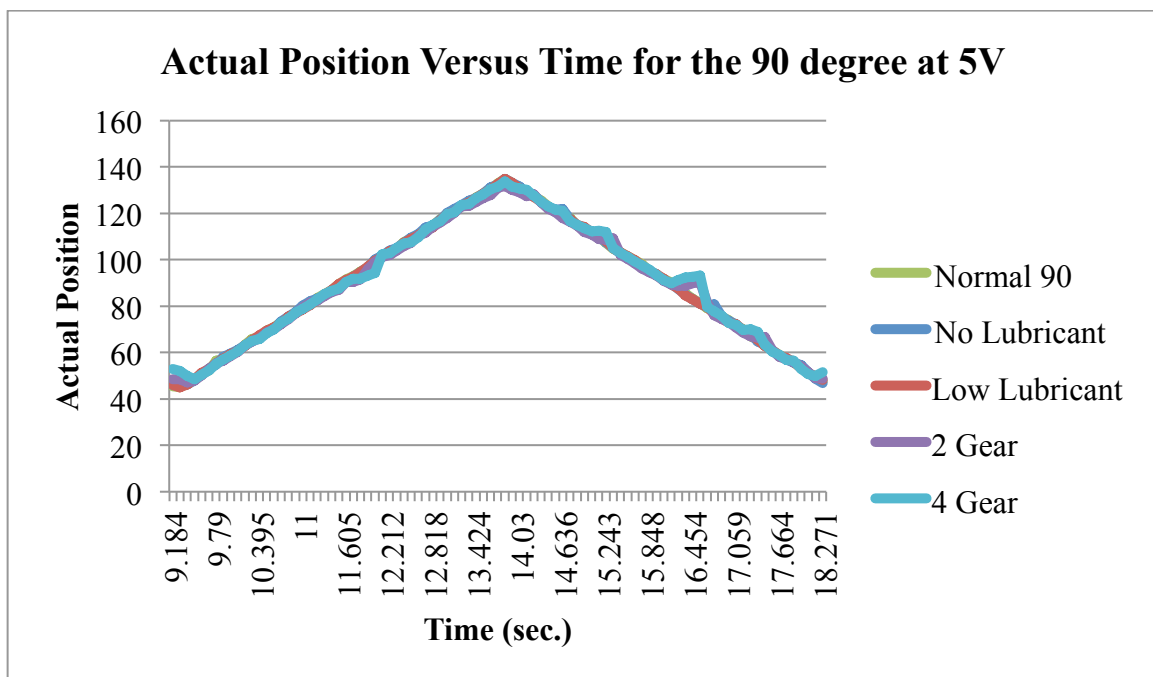


Figure 24: A graph of the actual position versus time with each of the faults for the 90-degree cycle at 5V.

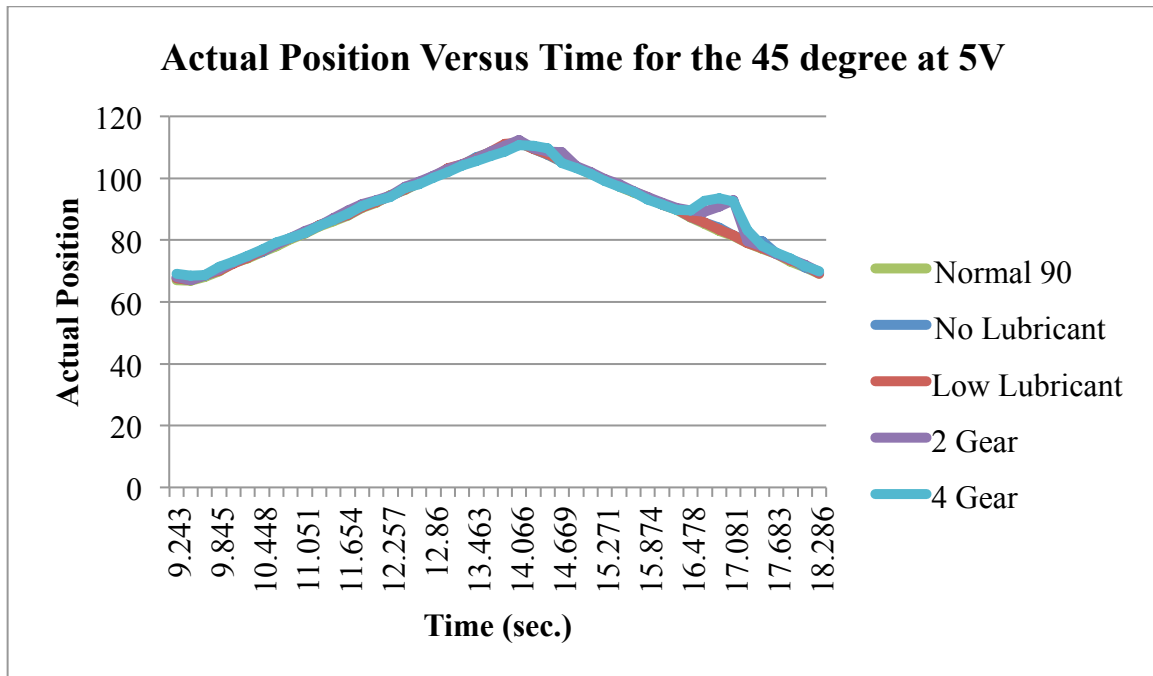


Figure 25: A graph of the actual position versus time with each of the faults for the 45-degree cycle at 5V.

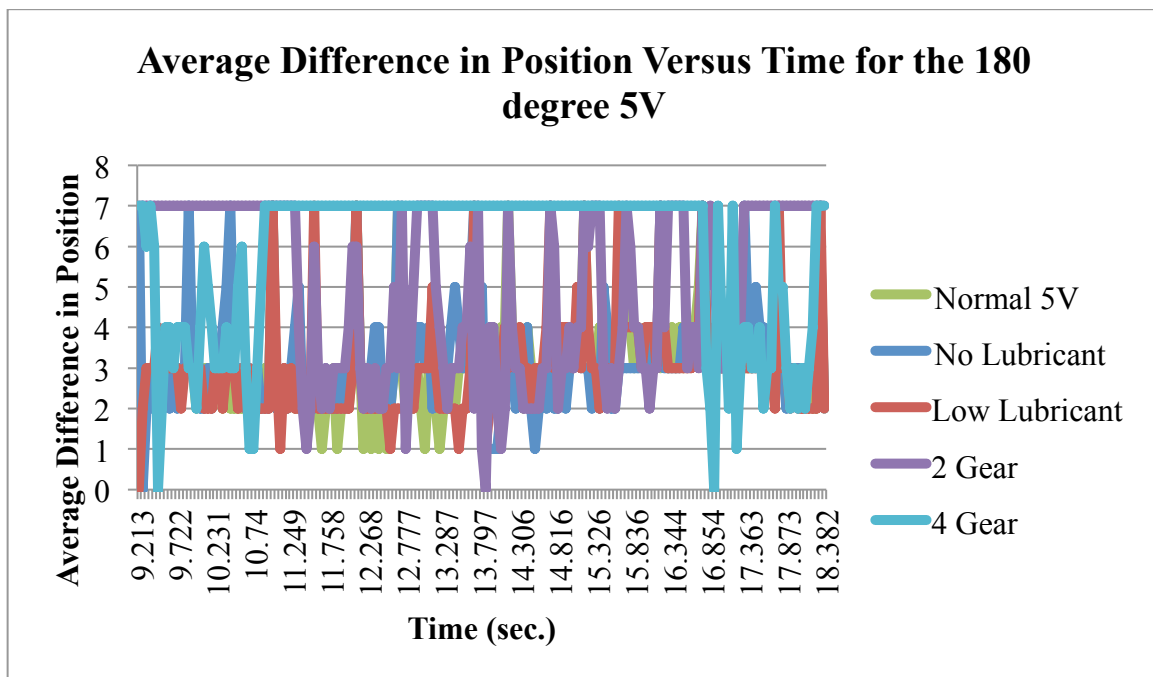


Figure 26: A graph of average difference in position versus time for the 180-degree cycle at 5V.

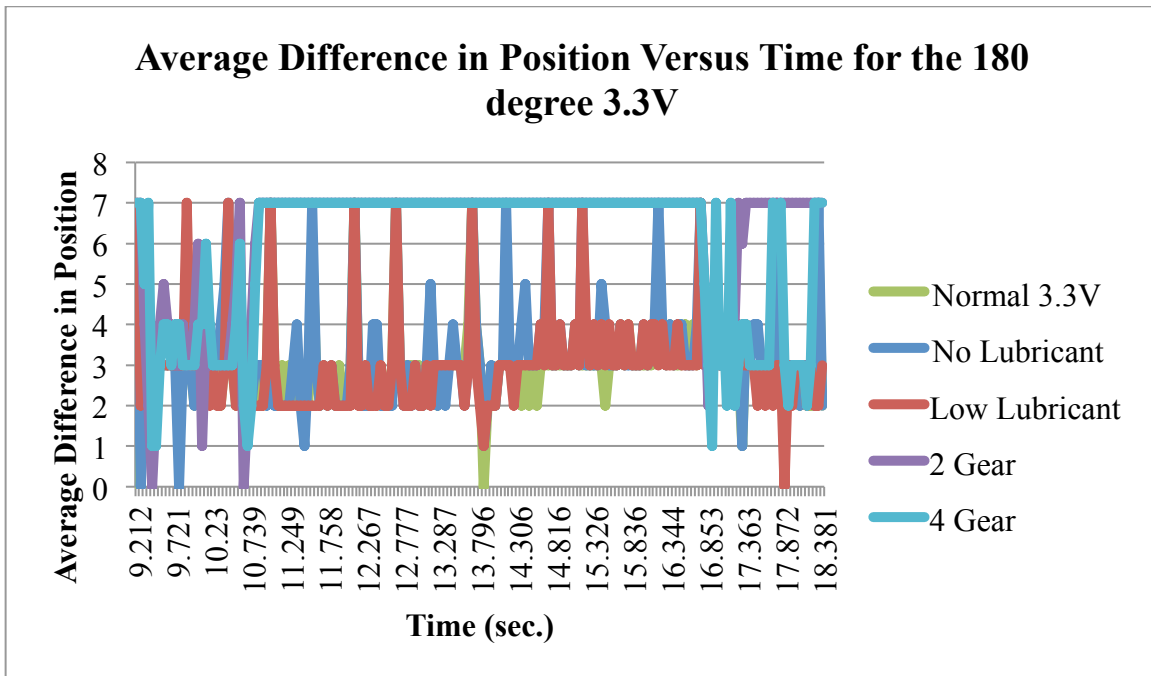


Figure 27: A graph of average difference in position versus time for the 180-degree cycle at 3.3V.

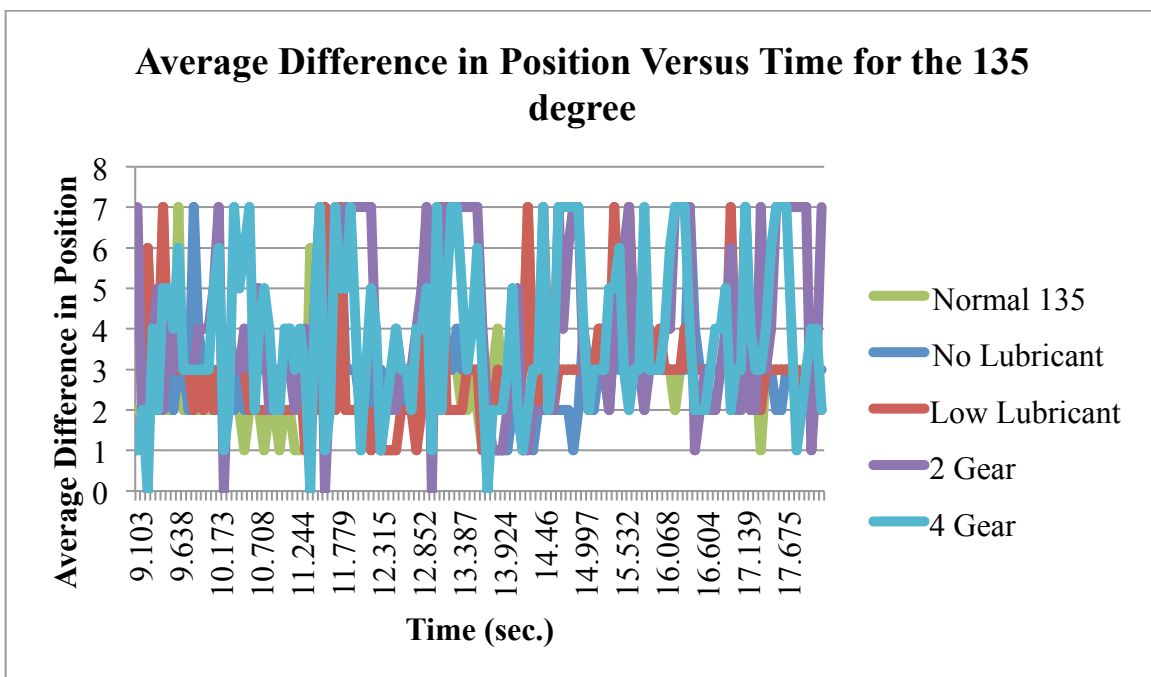


Figure 28: A graph of average difference in position versus time for the 135-degree cycle at 5V.

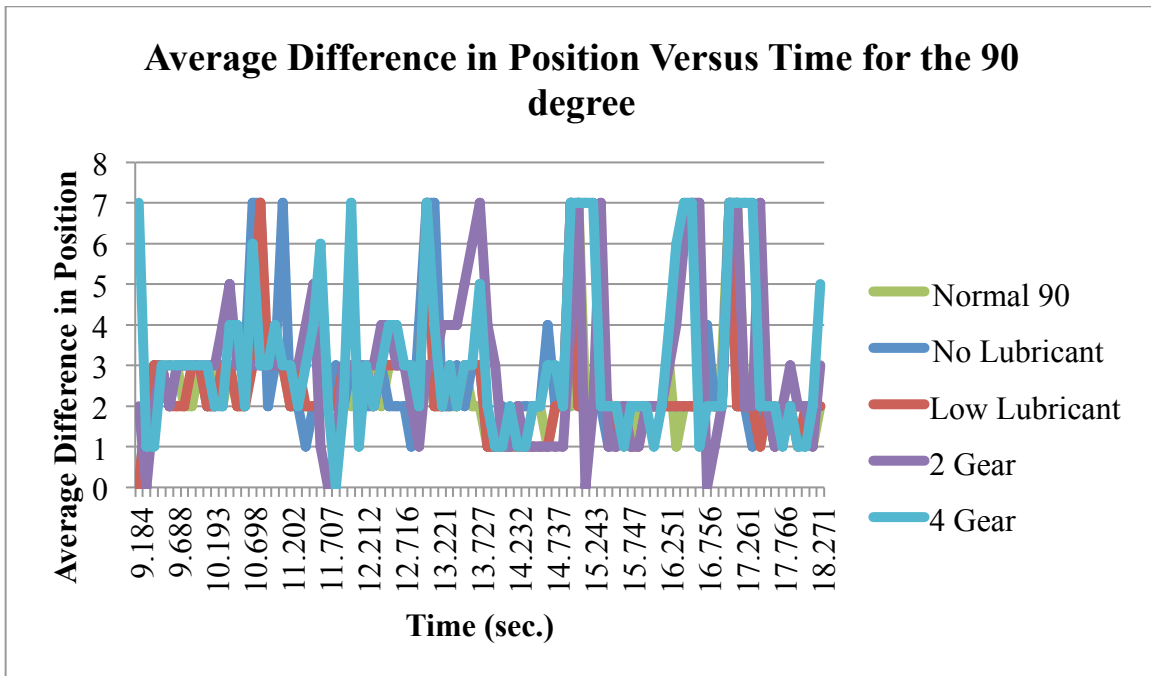


Figure 29: A graph of average difference in position versus time for the 90-degree cycle at 5V.

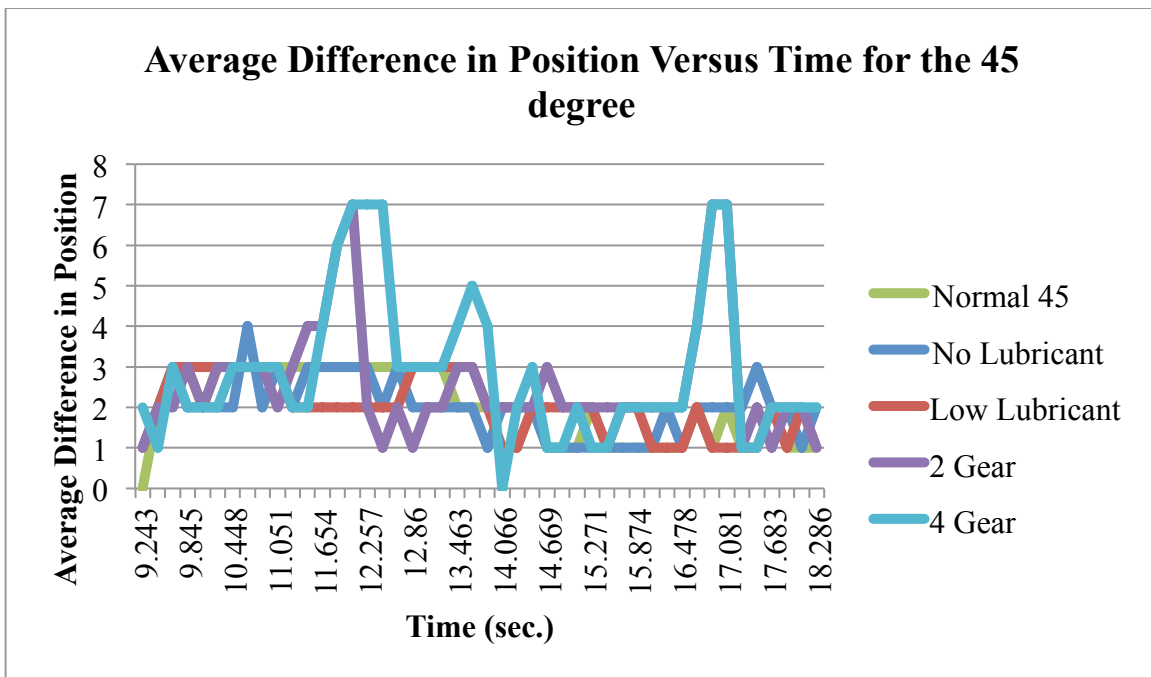


Figure 30: A graph of average difference in position versus time for the 45-degree cycle at 5V.

Table 12: The speed of the hand with the normal servomotor in the robotic arm.

Cycle	4 Second Cycle			2 Second Cycle			8 Second Cycle		
	Time (s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)
1	4	4.620	1.155	2	2.288	1.144	8	9.208	1.151
2	4	4.604	1.151	2	2.279	1.140	8	9.294	1.162
3	4	4.612	1.153	2	2.280	1.140	8	9.213	1.152
4	4	4.596	1.149	2	2.273	1.136	8	9.222	1.153
5	4	4.636	1.159	2	2.288	1.144	8	9.235	1.154
6	4	4.607	1.152	2	2.296	1.148	8	9.230	1.154
7	4	4.597	1.149	2	2.280	1.140	8	9.228	1.154
8	4	4.590	1.147	2	2.288	1.144	8	9.231	1.154
9	4	4.596	1.149	2	2.280	1.140	8	9.238	1.155
10	4	4.613	1.153	2	2.280	1.140	8	9.239	1.155
11	4	4.611	1.153	2	2.280	1.140	8	9.235	1.154
12	4	4.597	1.149	2	2.296	1.148	8	9.222	1.153
13	4	4.607	1.152	2	2.288	1.144	8	9.247	1.156
14	4	4.597	1.149	2	2.280	1.140	8	9.189	1.149
15	4	4.587	1.147	2	2.280	1.140	8	9.223	1.153
16	4	4.610	1.153	2	2.280	1.140	8	9.183	1.148
17	4	4.597	1.149	2	2.280	1.140	8	9.244	1.156
18	4	4.610	1.152	2	2.303	1.152	8	9.236	1.154
19	4	4.607	1.152	2	2.288	1.144	8	9.239	1.155
20	4	4.620	1.155	2	2.287	1.144	8	9.264	1.158
		Average:	1.151		Average:	1.142		Average:	1.154

Table 13: The speed of the hand with the low lubricant servomotor in the robotic arm.

Cycle	4 Second Cycle			2 Second Cycle			8 Second Cycle		
	Time (s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)
1	4	4.639	1.160	2	2.300	1.150	8	9.251	1.156
2	4	4.653	1.163	2	2.331	1.165	8	9.297	1.162
3	4	4.661	1.165	2	2.300	1.150	8	9.269	1.159
4	4	4.661	1.165	2	2.329	1.165	8	9.268	1.158
5	4	4.614	1.154	2	2.263	1.132	8	9.255	1.157
6	4	4.651	1.163	2	2.315	1.157	8	9.279	1.160
7	4	4.622	1.156	2	2.366	1.183	8	9.272	1.159
8	4	4.642	1.161	2	2.358	1.179	8	9.234	1.154
9	4	4.628	1.157	2	2.320	1.160	8	9.271	1.159
10	4	4.656	1.164	2	2.271	1.135	8	9.243	1.155
11	4	4.641	1.160	2	2.338	1.169	8	9.242	1.155
12	4	4.639	1.160	2	2.323	1.161	8	9.242	1.155
13	4	4.641	1.160	2	2.330	1.165	8	9.263	1.158
14	4	4.633	1.158	2	2.350	1.175	8	9.233	1.154
15	4	4.641	1.160	2	2.300	1.150	8	9.240	1.155
16	4	4.640	1.160	2	2.291	1.146	8	9.238	1.155
17	4	4.628	1.157	2	2.337	1.169	8	9.284	1.160
18	4	4.653	1.163	2	2.300	1.150	8	9.268	1.158
19	4	4.624	1.156	2	2.360	1.180	8	9.230	1.154
20	4	4.648	1.162	2	2.316	1.158	8	9.282	1.160
		Average:	1.160		Average:	1.160		Average:	1.157

Table 14: The speed of the hand with the no lubricant servomotor in the robotic arm.

Cycle	4 Second Cycle			2 Second Cycle			8 Second Cycle		
	Time (s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)
1	4	4.447	1.112	2	2.238	1.119	8	8.862	1.108
2	4	4.415	1.104	2	2.231	1.115	8	8.860	1.107
3	4	4.396	1.099	2	2.240	1.120	8	8.837	1.105
4	4	4.451	1.113	2	2.227	1.113	8	8.836	1.104
5	4	4.443	1.111	2	2.232	1.116	8	8.835	1.104
6	4	4.439	1.110	2	2.207	1.104	8	8.841	1.105
7	4	4.432	1.108	2	2.278	1.139	8	8.839	1.105
8	4	4.377	1.094	2	2.246	1.123	8	8.836	1.104
9	4	4.459	1.115	2	2.239	1.119	8	8.882	1.110
10	4	4.418	1.105	2	2.207	1.103	8	8.838	1.105
11	4	4.386	1.096	2	2.231	1.115	8	8.837	1.105
12	4	4.426	1.106	2	2.232	1.116	8	8.867	1.108
13	4	4.449	1.112	2	2.230	1.115	8	8.832	1.104
14	4	4.418	1.104	2	2.231	1.116	8	8.862	1.108
15	4	4.440	1.110	2	2.182	1.091	8	8.862	1.108
16	4	4.441	1.110	2	2.231	1.115	8	8.850	1.106
17	4	4.472	1.118	2	2.272	1.136	8	8.838	1.105
18	4	4.457	1.114	2	2.222	1.111	8	8.840	1.105
19	4	4.448	1.112	2	2.208	1.104	8	8.827	1.103
20	4	4.417	1.104	2	2.232	1.116	8	8.840	1.105
		Average:	1.108		Average:	1.115		Average:	1.106

Table 15: The speed of the hand with the 2-gear tooth servomotor in the robotic arm.

Cycle	4 Second Cycle			2 Second Cycle			8 Second Cycle		
	Time (s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)	Time(s)	Distance (m)	Speed (m/s)
1	4	4.511	1.128	2	2.221	1.111	8	8.925	1.116
2	4	4.484	1.121	2	2.139	1.069	8	8.978	1.122
3	4	4.498	1.124	2	2.220	1.110	8	8.903	1.113
4	4	4.502	1.126	2	2.204	1.102	8	8.960	1.120
5	4	4.484	1.121	2	2.180	1.090	8	8.932	1.116
6	4	4.484	1.121	2	2.165	1.083	8	8.879	1.110
7	4	4.483	1.121	2	2.171	1.086	8	8.957	1.120
8	4	4.491	1.123	2	2.212	1.106	8	8.947	1.118
9	4	4.479	1.120	2	2.213	1.107	8	8.873	1.109
10	4	4.468	1.117	2	2.171	1.085	8	8.952	1.119
11	4	4.490	1.122	2	2.195	1.098	8	8.879	1.110
12	4	4.474	1.118	2	2.204	1.102	8	8.872	1.109
13	4	4.478	1.119	2	2.180	1.090	8	8.877	1.110
14	4	4.474	1.119	2	2.183	1.092	8	8.875	1.109
15	4	4.490	1.123	2	2.205	1.103	8	8.948	1.119
16	4	4.496	1.124	2	2.156	1.078	8	8.953	1.119
17	4	4.495	1.124	2	2.212	1.106	8	8.957	1.120
18	4	4.472	1.118	2	2.198	1.099	8	8.960	1.120
19	4	4.495	1.124	2	2.139	1.070	8	8.960	1.120
20	4	4.476	1.119	2	2.205	1.102	8	8.911	1.114
		Average:	1.122		Average:	1.094		Average:	1.116