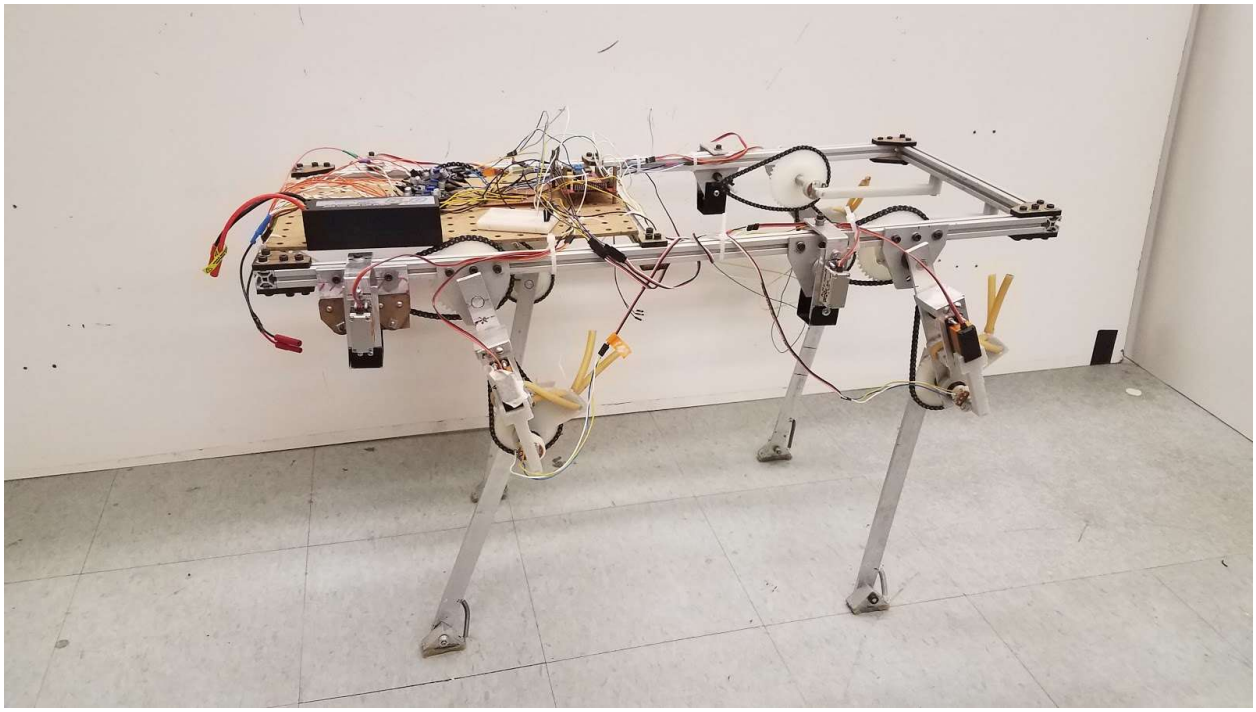
RoboDog:
A Low-Cost Electrically Actuated Quadruped



Written by:
Yunda Li & Michael Pickett

Advisor:
Professor Popovic

Co-Advisor:

Professor Radhakrishnan

Abstract

Legged robots allow for travel over the variable terrain commonly found on Earth, which is more difficult for traditional wheeled systems. The MIT Leg Lab made great progress with both bipedal and quadruped robots in the 90s. Currently, advances have been made with legged robots through companies like Boston Dynamics. These robots provided the inspiration for Robodog. Legged robots pose a variety of challenges from mechanical design to control, and as a result, WPI has not produced many successful legged robots. For our project, we designed a quadrupedal robot to accomplish a basic walking gait. It stands at 21 inches and weighs 18.9 lbs. Robodog was designed and manufactured from the ground-up, and is the first step toward cheaper quadrupeds. This was accomplished using servo motors and a parallel elastic system to decrease motor load and increase efficiency. After extensive experimentation our quadruped is capable of successfully walking.

TABLE OF CONTENTS

Acknowledgements

We would like to thank our advisor, Professor Marko Popovic, for his advice, motivation, and insight into the field of legged robotics. We would also like to thank our co-advisor, Professor Pradeep Radhakrishnan, whose advice and resources were very helpful in the completion of this project. We'd like to thank the various members of Popovic Labs, especially Matthew Bowers, for inspiration and lab help, as well as Chinmay Harmulkar, for help with sensors and 3D printer usage. Also in Popovic Labs, we'd like to thank Elina Saint-Elme and Casey Kracinovich for their help. In the faculty, we'd like to thank James Loiselle for the assistance with CNC machining and Professor Erica Stults for help with 3D printing pieces. Finally, we'd like to thank Caleb Wagner for helping every step of the way, and helped ease the heavy workload of this project.

Executive Summary

In this Major Qualifying Project (MQP), we designed, built, tested, and refined a quadrupedal robot for all terrain usage. The robot was meant to be capable of walking over small obstacles and uneven terrain like that of some planets in our Solar System. This report details our work in getting a robotic quadruped around the size of a medium sized dog to walk, while still keeping costs reasonably low. First, the various research done before the project was started is presented, which includes the workings of previous quadrupeds, and research into the various gaits of 4 legged animals. After this is a summary of the goals set at the beginning of the project. The next section details the various designs for the mechanical systems, electronics, and coding involved in the development of Robodog, and then the various experimentation done to refine and improve these designs. Finally, the various methods used in the construction and assembly of

the final robot are discussed, and the report ends with various steps any successors could make in improving this project.

Robodog is built from mostly aluminum, in order to keep costs low, have a light body, and still have strong support. The frame is made of Aluminum 80/20, chosen for its easy configurability, and the legs are made of Aluminum 6061, chosen for its workability.

The dog is actuated by servo motor, as well as a parallel elastic system on the lower knee joint. This elastic system was key in ensuring that there was enough torque to get such a large robot to walk. The elastic system charges as the leg lifts, and releases as the leg steps. Using this system allowed the dog to work with lower cost motors, and makes these motors more efficient. This allows the motors to save energy during its running, but also when at rest, since the elastics allow the Robodog to stand without power. Robodog was able to successfully walk, and with the usage of a parallel elastic system, two walking gaits were achieved, the walking trot and the lateral sequence.

Introduction

The motivation for the study of legged robots is inspired by the limitations of wheeled robots. Wheeled robots tend to need a flat surface to run, but flat surfaces are rare in the unexplored places on earth, and on the other planets of the Solar System. Another issue with legged robots is cost, where for many quadrupeds, the cost of which makes them unfeasible. The advent of a low cost robotic quadruped could result in advances in not just space exploration, but also in general everyday usage in carrying objects, shipping, and other general utility

Wheeled robots and their limitations are apparent when dealing with nonuniform surfaces. For example, the surface of the Moon and Mars have many craters and slopes throughout their surfaces. In everyday life on Earth, there are stairs, hills, and terrain to navigate,

which can be difficult for wheeled robots. Legged robots take inspiration from living things that navigate such obstacles easily.

This project was undertaken by two senior undergraduate students at Worcester Polytechnic Institute(WPI). Every part of the process of creating a working quadruped was done, from design, to testing and experimentation, to construction and manufacturing. This project can be seen as a first step in the interest of building low cost robotic quadrupeds with parallel elastics systems, and ideally future work can be done to improve upon the work done here. This report details the various processes and challenges encountered, so that future work done on this concept can learn from the insight gathered from this project.

Background

Gait Research:

In order to control a legged robot, one must look to nature and way other legged creatures move. By looking at the movement of quadrupeds in different gaits, determining a control pattern was possible. Quadrupeds have various gaits, both symmetrical, meaning that any given side's (left or right) pair alternate, and asymmetrical, meaning that the pairs move together [1,2]. For this project, focus was placed on the walking gaits due to limited motor torque.

The first gait in the symmetrical gaits section of Figure 1 shows the walking trot. This gait requires the diagonal leg pairs to alternate phases with a short time where all four legs are in contact. The second walking gait is the lateral sequence gait which is more complicated, but allows for potential for three legs on the ground at a given time with worst case being two legs on the ground. It should be noted that gait diagrams like Figure 1 are just a general rule of thumb and actual contact times vary depending on leg lengths and various other variables [2]. This diagram served as a basis for RoboDog's control scheme.

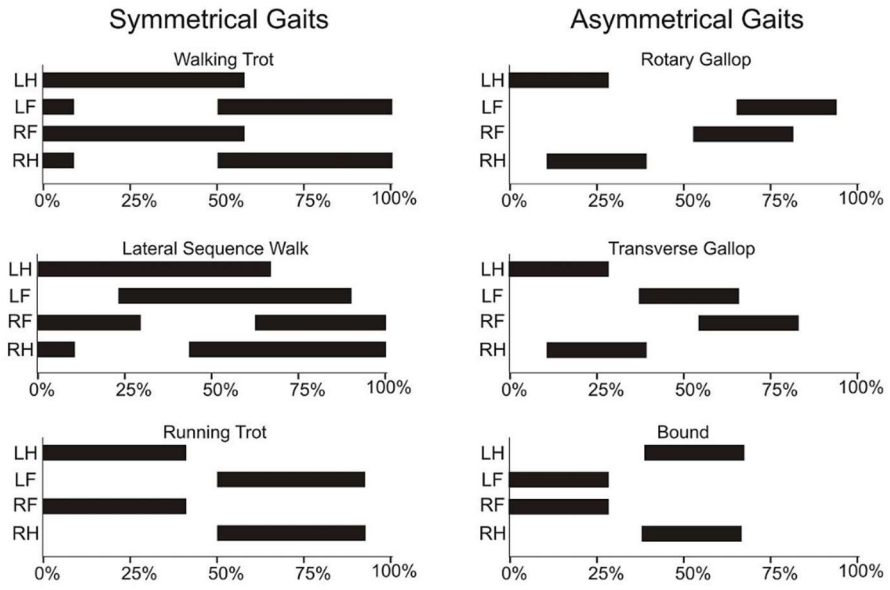


Figure 1: Gait Timing Graphs [1]

To help with understanding the movement of canines, videos of canines were observed at various speeds [3]. These videos were analyzed using a program called Tracker. This program allows for frame to frame motion analysis and provide information on leg angles during impact and lift off, as well as, foot pathing. Figure 2 shows an example of this software being used on the dog video.

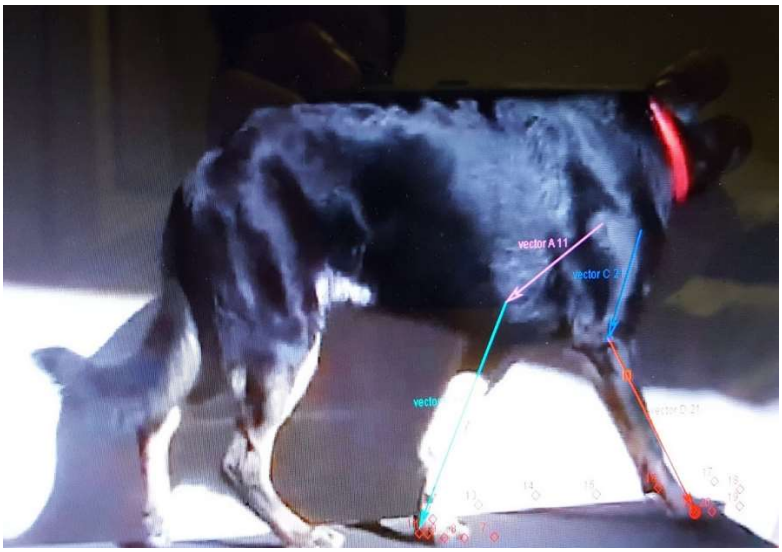


Figure 2: Dog walking on treadmill and being tracked via Tracker software

Other Legged Robots:

Design inspiration for this project came from the multitude of legged robots that came before RoboDog. MIT Leg Lab's design process had a big impact on the design process that was followed for RoboDog. Starting off with a single leg to ensure proper specifications and results was inspired by the Leg Lab and a circular track from the single leg was even constructed. Though RoboDog's single leg was never meant for independent travel, it served as an interesting point of research to determine the leg's fitness for the resulting quadruped.

More recently, Boston Dynamics various quadrupeds inspired the design choices that were made as the project progressed. Boston Dynamics has created quadrupeds like BigDog,

Wild Cat, Spot and Mini-Spot. Given the planned size and budget of RoboDog, Spot and Mini-Spot served as main focal points. Particularly, the leg orientation of Spot and Mini-Spot inspired RoboDog's final design. It became clear that have inverted back legs would be the best approach for RoboDog in terms of clearance. Videos of Spot and Mini-Spot also assisted in understanding the controls and the general movement of quadruped robots.

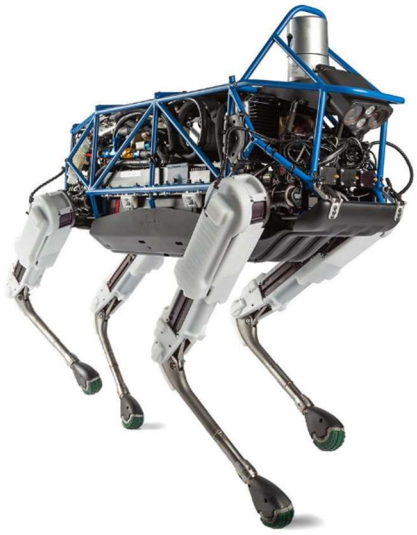


Figure 3: Boston Dynamic's Spot [4]

DH Parameters:

During the development of RoboDog's various leg designs, it was necessary to analyze the inverse kinematics of the legs. Normally a simple 2-DoF doesn't require DH parameters to calculate the Inverse Kinematics, but by using Work-Energy Equivalence the torque our motors would ideally need to output could be determined. Below is an outline of DH parameters, Work-Energy Equivalence and how it pertained to this project.

DH parameters are a tool used to convert from one coordinate plane to another. This is most helpful with multi-DoF system as it allows one to calculate the movement of a given joint

or end-effector with respect to its or any other coordinate frame. This allows one to look at a desired output and figure out the necessary joint angles to get that output.

For RoboDog's 2-DoF leg DH parameters were used to transform from the hip joint to the foot in order to understand the movement of the leg with respect to the hip joint. With DH parameters matrices for transformation were created (See Figure 3 for an example matrix). The matrices consist of a rotational and translational element. The various DH parameter as as follows: d is the distance along the previous z axis to the common normal, θ is the angle about the previous z axis to align with the new x axis, a is the length of the common normal, and **α** is the angle about the new x axis to align the previous z axis with the new z axis [5]. With multiple matrices between various coordinates we can multiply them to get the relation from one to another. After obtaining the matrix the Work-Equivalence theorem can be used.

$$
{}^{n-1}T_n = \left[\begin{array}{cccc}
\cos\theta_n & -\sin\theta_n & 0 & a_{n-1} \\
\sin\theta_n \cos\alpha_{n-1} & \cos\theta_n \cos\alpha_{n-1} & -\sin\alpha_{n-1} & -d_n \sin\alpha_{n-1} \\
\sin\theta_n \sin\alpha_{n-1} & \cos\theta_n \sin\alpha_{n-1} & \cos\alpha_{n-1} & d_n \cos\alpha_{n-1} \\
\hline
0 & 0 & 0 & 1
\end{array}\right]
$$

Figure 4: Transformation Matrix with DH parameters [5]

The Work-Equivalence theorem is based off the conservation of energy. The work done linearly will be equal the rotational work done. Given a desired force matrix and the needed torque matrix can be determined [6]. This is done by using the Jacobian of the system, as shown by the equation below, where F_q, J^T, and F_x are the torque, jacobian and force matrices respectively.

$$F_q = J_{ee}{}^T * F_x$$

The Jacobian can be calculated from the transformation matrix obtained from the DH parameters and consists of a linear and angular part. The linear results from the partial derivatives of the

translational components of the transformation matrix and the angular is the partial derivatives of the angular components (See Figure 4). With this, the leg design can be properly analyzed.

$$\mathbf{J}_v(\mathbf{q}) = \begin{bmatrix} \frac{\partial x}{\partial q_0} & \frac{\partial x}{\partial q_1} \\ \frac{\partial y}{\partial q_0} & \frac{\partial y}{\partial q_1} \\ \frac{\partial z}{\partial q_0} & \frac{\partial z}{\partial q_1} \end{bmatrix}$$

$$\mathbf{J}_\omega(\mathbf{q}) = \begin{bmatrix} \frac{\partial \omega_x}{\partial q_0} & \frac{\partial \omega_x}{\partial q_1} \\ \frac{\partial \omega_y}{\partial q_0} & \frac{\partial \omega_y}{\partial q_1} \\ \frac{\partial \omega_z}{\partial q_0} & \frac{\partial \omega_z}{\partial q_1} \end{bmatrix}$$

$$\mathbf{J}_{ee}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_v(\mathbf{q}) \\ \mathbf{J}_\omega(\mathbf{q}) \end{bmatrix}$$

Figure 5: Jacobian [6]

Objectives:

The objectives for this project were to create a quadruped robot capable of 1.5 mph walking, while standing at 21 in. This robot would hopefully be capable of walking over variable terrain, like gravel, and up and down inclines of up to 30 degrees. RoboDog was also meant to be very cheap and be fairly simple compared to other quadruped robots, which usually have custom motors and sensor rigs.

Design and Experimentation

Leg Design 1:

Figure 6: Initial Leg Design

The initial design was a simple 4 bar linkage. The goal of this first design was to get a motion that resembled the output of the stepping motion of a dog. The design was inspired by other legged linkage designs. After testing with a preliminary design in Working Model, a wooden model was built, to test various leg lengths, and observe the output. This model was powered by hand, turning the input link. Later, two stoppers were added to test the feasibility of using stoppers to raise the leg.

Next this leg design was modeled in CAD. This was done to simulate the motion of the leg with a motor, and easily see the difference in output with various motor speeds. In CAD, collision with the included stoppers was difficult to simulate, but CAD programs were able to generate valuable mass, material properties, and motion. From these results, Aluminum 6061 was chosen as the material for the leg, due to this material being relatively strong, lightweight, and inexpensive. A model was built with the aluminum, and motion of the leg was tested with a motor and collar.

Leg Design 2:

The next designs made use of springs in conjunction with the motor to store and release energy. The first attempt to integrate springs with the leg design involved adding a torsion spring at the input link of the initial design. This spring would be oriented in the opposite direction of the motor output, so that as the motor turned it would charge the spring, and the release in spring energy would cause the leg to walk. Since torsion springs were unavailable at the time, this idea was simulated in CAD.

Leg Design 3:

After performing testing with prior leg designs, it was determined that having independent control of the elbow joint would be worth the tradeoff of having additional complexity. In order to get elbow control, it was originally thought a direct connection to a precision servo would be a sufficient, but would result in the servo operating at stall torque for extended periods of time. Professor Popovic suggested a device that could be charged by a servo and retracted by a spring system. Accordingly, the spring force would be used in the most demanding areas to protect the servo from having to actuate. The implementation of this design would require two per leg, one for the upper leg and one for the lower leg. An alternative design based on these properties involved a piston-based system that ran to various pulleys.

Final Design:

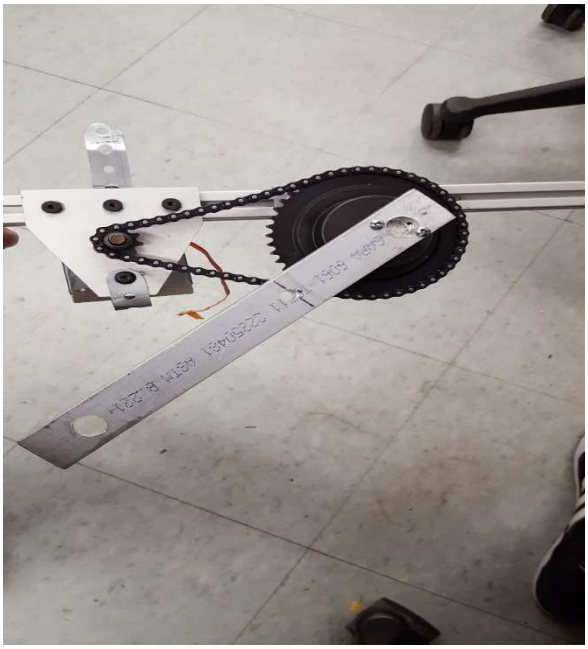


Figure 7: Upper chain sprocket

The piston design was ultimately determined to be too difficult to implement as the piston could only move in linearly in one direction while requiring the upper leg to rotate around an axis. Thus pushing and pulling on one end of the upper leg with the piston would not be possible, since the path that the end of the leg would traverse would be curved. This issue was also compounded with problems supporting the piston.

The final model implemented a simple design, where each joint of the leg was actuated by a servo geared up via sprocket and chain. Initially, a 1:5 gear was used for the upper hip joint, and a 1:3 gear ratio for the knee joint. This was eventually changed after the results of various experimentation. A list of material used in the design and test process can be found in Appendix D.

Mount Design:

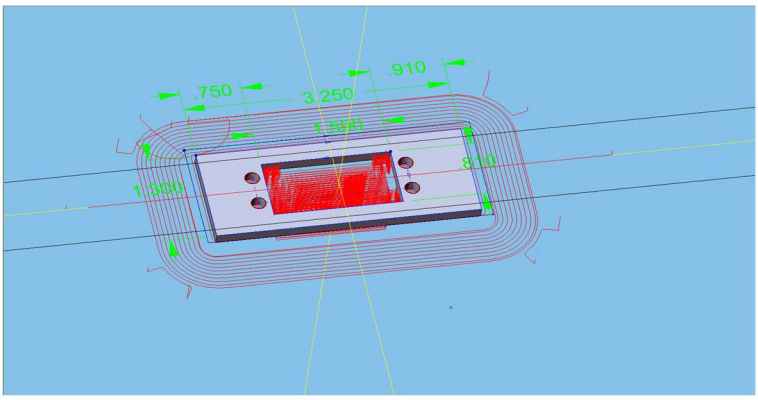


Figure 8: CAM tool paths for machined mounts

With the aforementioned final design of the leg, the servomotor the would need to be mounted to the upper leg. This mount would need to be secure so as to resist reactionary forces from the drive train. The initial motor mount was put together using various scrap, and supported by wood and duct tape. This was a temporary measure to use in the testing of the leg. The final model would machine the mounts for the motors in order to have a consistent mount for each motor, as well as to make them as secure as possible. The mounts were machined on a Haas MiniMill, and consisted of a plate with a rectangular pocket and holes to screw into the attached screw holes on the servo. Eight of the described plates were machined to fit the RJX hobby servos. The RJX hobby servos were later switched out for Savox servos. Due to a similar servo size being ordered, the plates were quickly modified to fit the new servo screw holes by filing the ridges on the screw hole ridges.

With the plates completed, the servos themselves could be mounted to an attachment piece, which then had to be mounted to the legs. For the lower leg mounts, the plate needed to be mounted onto the upper leg, while for the upper leg the plate had to be mounted to the frame of the body. The lower leg mount needed to be especially secure since the piece that the plate would

be mounted on would be moving. The system would have to make sure that nothing came loose, or would shake during the movement of the leg. To this end, two thicker brackets of aluminum were drilled and tapped, such that they could be screwed into the plate to the brackets and to the leg. Special care was considered while dimensioning the location of the mount on the leg as the chain was both difficult to adjust and the size of the chain and teeth meant that the sprockets would have be at certain distances from each other, otherwise the chain would catch on the sprocket, or be too loose. As we also found with the prototype leg, we would only have one chance to get this dimension correct, since if we made a mistake, we would have to make the holes larger to space the chain and sprockets properly. This results in a too loose of fit of the shafts on the leg, which causes it to wiggle.

Feet Design:

The Hydrodog used simple rubber caps as feet. These caps were initially used as a placeholder, with the intention of replacing these caps for the final design, since they didn't have much friction with the ground, and had a small area in which it contacted the ground. These feet were mostly used for force testing with the prototype leg. Eventually, a design was developed for a larger foot at the base. This was done with a triangular prism, held with a pin to the bottom of the lower leg, to allow for rotation. This is similar to an ankle joint, which allows the rotation of the foot forward and backward. This prism was made of aluminum, which doesn't have much friction with the ground, so some different material would need to be added to the bottom of the foot. Tests were conducted on various nonmetal materials found in the lab, to see which would have the best friction with the ground. To do this, strips of materials such as rubber and latex were cut out at the same dimensions as the bottom of the feet. Then, a weight was placed onto the strip, and how much force it took to pull the strip and weight to slide on the ground was

measured. The more force it took to pull it along, the greater friction there was between the material and the ground. After these tests the material with the best friction with the ground to use was latex, from latex tubing around the lab. A section of this tubing was cut out, and then flattened, and superglued to the bottom of the foot.

With the triangular piece pinned to the bottom of the leg, the foot simply hangs in place, but without a way to restrict its rotation, the foot would flip over when in use. The next iteration of the foot design was inspired by the achilles heel that humans have. As a human walks, the weight of the person is shifted forward, and the back of the foot is pulled by the achilles heel. To mimic this pull, a spring was placed on the back end of the foot, to be connected to the back of the leg. As the leg goes through its pull through phase, the foot is on the ground and rotates backwards relative to the leg. This increases the distance between the back end of the foot and the leg. By putting a spring here, the motion of the leg charges the spring, and as foot lifts again, the spring pulls the foot to rotate forward.

This accomplishes several things. Since the spring will try to return to its initial length, this ideally prevents the foot from rotating too far backwards. This also ensures that the foot stays at the same orientation when the foot isn't on the ground, and that the foot will always return to the same orientation after lifted off the ground. The original intention of the design was to help to shift the weight of the foot forward, and it does accomplish this to a degree, however the springs available and at this size were not strong enough to make much of a difference in shifting weight. As a result, the spring is mostly used as a positioning tool.

This foot design was novel for legged robots, and was used in most of the tests run. Later, it was observed that perhaps these feet gripped too well on the surfaces that Robodog was tested on, and so made it difficult to lift the leg again from the ground. Observing other legged robots

seemed to show that most legged robots had small feet, especially compared to the size of the leg. In an effort to reduce friction, the feet were replaced with tennis balls, that the legs were slotted into. These ended up working much better on the tile floors in the lab. The previous foot design may have potential for future work on this project.

Single Leg Experimentation:

Figure 9: Prototype Leg

Rigorous testing was done for the initial prototype leg. The first leg was a way to prepare and figure out issues concerning assembly of the legs for the final robot. This leg also was used for force testing to determine final components necessary, testing the pathing of the leg, as well as basic walking tests.

Having a prototype leg allowed tests to be run for the forces that the leg can produce. At the end of the lower leg, at the foot, the leg needs to push down with greater force than the weight of the robot, divided by the number of legs are on the ground. At any time, the fewest number of legs in contact with the ground is two, so any single leg needs to be able to support

half the weight of the entire robot. Through testing, the original design with a 3:1 gearing at the elbow joint, was found to not have enough force to support the entire robot. The lower joint was more important in supporting the robot, since the lower leg was significantly longer than the upper leg, which meant that the moment arm was much longer, and that it would take more torque at the lower joint to counteract the normal force from the floor at the foot. This joint was switched to a roughly 4.5:1 gearing at the elbow joint. This would provide more torque at the joint, in exchange for a lower range of motion and less speed.

Several tests were done to try and measure the forces the foot could generate. For force downward, the "body" of the single leg was secured and then powered the foot to push down during pull through phase of a walking gait on a force sensor. This test had issues, since it was difficult to have the leg pull through at a point where data could be collected. The leg would often pull past where the force sensor was, and the small surface area of the initial foot used, it was difficult to orient the sensor so all of the force generated by the leg was recorded. A different method was tried, where the leg was run to stand, and kept powered, and then the force sensor was pushed against the bottom of the foot until the motor started to give. This method also didn't work well, since the set screws that held the shafts in place started to slip before the servos would give way. This led to the use of a thread locking fluid to secure the screws. In the end, leg and body was flipped upside down, and the leg tried to lift a weight upwards.

Building a prototype leg gave an idea of the weight of each leg, and therefore allowed a better estimation for the total weight of the dog, since the majority of the weight would be in the legs. The body would mostly be an aluminum 80/20 frame, with some electronics on it, which overall weighs much less than the components of the legs. The leg initially weighed around 6 lbs, which was difficult for the leg to lift itself with. Measures were taken to reduce the weight as

much as possible, and the biggest impact to this was the changing the material of the sprockets. The large upper sprocket weighed about 1 lb, and the lower sprocket weighed about 0.7 lbs. These were switched to use nylon sprockets, each of which weighed less than 0.1 lbs. Just by switching the sprockets the weight of a single leg was reduced by around 1.5 lbs. These nylon sprockets didn't have holes for set screws, which were drilled and tapped in the machine shop afterwards. For the large 40 and 45 tooth sprockets, the screws would hold, since there was much more space for more threads. However, when attempting to replace the smaller 9 tooth sprockets with nylon sprockets, the sprockets were too small, and so even when drilling and tapping holes the screws wouldn't hold.

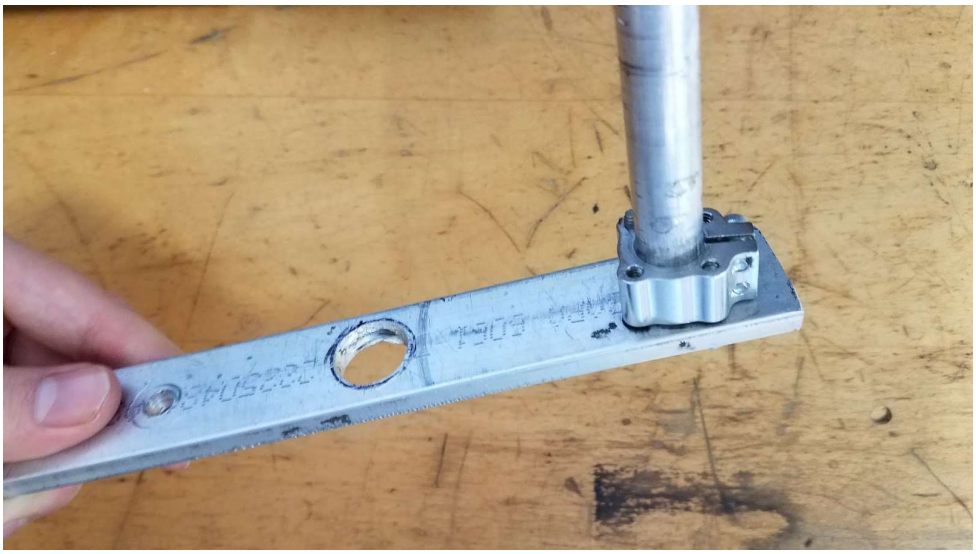


Figure 10: Example of the clamp that was replaced

Assembly of the single leg led to important knowledge and experience regarding putting the legs together. The first leg had many improvised or suboptimal systems, which were replaced. For example, the first leg attached the shafts to the leg bars with clamp hubs, that were tightened by screws. These were hard to attach, partially due to difficulties with placing the screw holes to mount the hubs. These screw holes had to be precisely placed to allow the hub to fit in line with the shaft. This was difficult to do with the limited tools in the lab, and since at the

time we didn't have full access to the machine shop. Another major issue with the clamp hubs that these would slip. For the rest of the duration of testing, the clamps were superglued to the shafts to maintain the connection without slip. For the final legs, the shafts were instead welded to the bars of the leg.

Another major insight gathered regarding assembly had to do with the chain, as well as the distances of the sprockets to each other. Since the chain has certain spacing in between the chains, and the sprockets' teeth were spaced accordingly, the chain could only be certain lengths so that the chain fit with the teeth without being loose or catching on a tooth. This became an important issue when building the other legs. When drilling the holes to put the shafts in, the placement and angle of the hole had to be precise to make sure the sprockets and chain fit properly. This was one of the reasons we decided to be trained to use the machine shop, to use the drill press for more precise holes.

This switch in gearing created issues when assembling the leg. The chain was sized to fit the previous gearing and distance between the two sprockets, but when the new gearing was introduced, the chain had to be resized, and didn't fit properly with the distance between the two sprockets. Since the holes had already been drilled into the aluminum, there was no way to change the distance without widening the hole. This meant that when running the leg, the shaft wasn't secured, and thus could wobble while operating. A piece was added to support the shaft and keep it level, and eventually added a bearing to keep the shaft stable.

Elastic System Experimentation:

The use of elastic systems in robotics is common. When elastic elements are discussed, the two main systems are series elastics and parallel elastics. The main difference is that a series elastic does work in one direction on its own and is charged in the other direction. A parallel

elastic works in line with a motor or other actuator and results in both contributing to work at the same time (usual in the same direction for one phase and opposite direction for the other). The difference in performance of series elastics versus parallel elastics has be researched, but results have not shown anything definitive if one is better than the other [7]. Most likely the best answer is a combination of both in a system.

For RoboDog's legs a parallel elastic system was used for the joint that would be doing the most work when lifting the weight, the lower knee joint. The general idea behind this system would be for when the leg lifts up, this would charge a spring. As the leg steps through, the spring would release its energy, and help the leg step through. We first built a rig to test this concept, and went through various ideas to actually implement this in the robot.

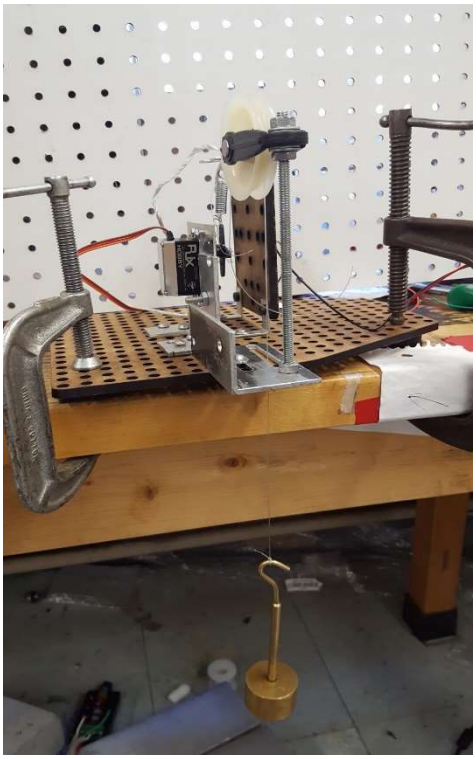


Figure 11: Parallel Spring System Testing

After deciding to implement a parallel elastic system, we decided to build a rig to test this concept. The servomotor spline is attached on one end to a pulley with a weight on one end, so

that when the servo runs, it raises the weight. On the other end, a spring is attached to a fixed point. When the weight is lowered, the spring is charged, and when the servo raises the weight, the spring releases, to help the servo lift the weight. We tested in two ways, once with a just a simple lowering and lifting up once, and then again while cycling lifting and lowering the weight. We measured the effectiveness of the spring in helping the servo by looking at the current draw of the servomotor. We found that with the spring, the current draw of the servo was much less, which was a good sign for the effectiveness of a parallel elastic system. Unfortunately, when we cycled the weight, it was difficult to track differences in the current draw.

After this, we had to decide how to implement an elastic system to a joint. We had a few ideas on how to do this. It seemed that the elastic system would be most valuable on the lower knee joint, where the moment arm was longest, and thus required the most torque. Our first idea was to have a drum, in which at one point we would attach a pulley with a spring. This drum would be fixed to the shaft, so as the leg lifts, the drum would turn, and charge the spring. We eventually decided that we didn't have enough space on the leg to put a drum. Next, we decided to try using latex tubes as an elastic element instead of springs, since it would be easier to deal with for construction. With the latex tubes, we first tried attaching one end along the back side of the lower leg, and then attaching to the upper leg. This method wouldn't in much extra torque at the joint, since the moment arm would be too small. Finally, we decided to essentially use a portion of a drum, in a sort of "pie" shape. This we decided to make ourselves by 3D printing, rather than ordering a drum. The added benefit of 3D printing, meant that we could save space by designing the pies to attach over the half inch collars that keep the legs on the shaft. The set screw for these collar would be replaced by longer screws, that function simultaneously as set

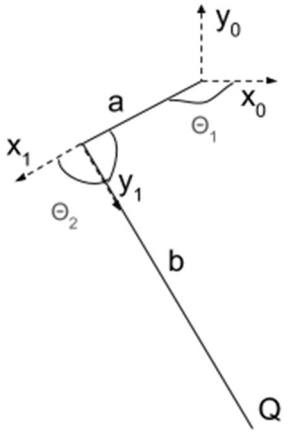
screws and to keep the pies in place. These pies would have a hole as far out on the pie as possible, to make the moment arm as long as possible, and this hole would have a latex tube looped through, the other end which attaches to the upper leg. This became the general design for the implementation of the elastic system.

Electronics and Sensors:

With the design described above an electronics system needed to be created that allowed us to optimize the performance of our system while still being fairly simple and inexpensive. For ease of use, a prototyping board like the Arduino Uno was used for the prototype leg. Eventually the processor was swapped for a Spider Board because of the extra analog ports needed for the potentiometers. The Spider Board also has the memory and power of a Arduino Mega, which came in handy when writing code. Rather than have to worry too much about memory usage, more processor-heavy code could be written. Deciding on the right motors was the one bigger decisions made on this project.

Motor choice was based around a few key factors. First the leg needed to be capable of lifting at least half the body weight of the dog because that would be the worst case scenario in a Lateral Sequence Gait. With the various leg designs made creating enough force to do this required an incredible amount of torque. Upon settling on the 2-DoF design of the final leg the high torque servos already available in lab were chosen. The RJX Hobby Servos chosen were convenient, fast and had a considerable amount of torque. To justify the use of these motors the Work Equivalency calculations were done, which is briefly outlined in the background of this paper. The aim for forces was 12N of force in the x direction for propulsion and 24N of force in the y direction for counteracting gravity. This vertical force was calculated based off of a estimated body weight for the dog. The estimation was made using the current weight of the

prototype leg. Force in the x direction was less exact and quite excessive, but it was for a safety factor. The angles were determined from tracking software placed on a dog video that determined the angle of the leg segment at the beginning of the contact phase. Lengths of the segments were based off the prototype legs dimensions. Below shows the calculations and matrices constructed to calculate the necessary motor torques.



Link	θ	α	a	d
0	0	0	0	0
1	θ_1	0	a	0

Figure 12: Coordinate frames and the DH Parameters

$^0T_1$ =

$\cos\theta_1$	$-\sin\theta_1$	0	$a*\cos\theta_1$
$\sin\theta_1$	$-\cos\theta_1$	0	$a*\sin\theta_1$
0	0	1	0
0	0	0	1

Q =

$b*\cos\theta_2$
$b*\sin\theta_2$
0
1

$^0T_1 * Q$ =

$b*\cos(\theta_1+\theta_2) + a*\cos\theta_1$
$b*\sin(\theta_1+\theta_2) + a*\sin\theta_1$
0
1

$J_{ee}(q)$ =

$-b*\sin(\theta_1+\theta_2) - a*\sin\theta_1$	$-b*\sin(\theta_1+\theta_2)$
$b*\cos(\theta_1+\theta_2) + a*\cos\theta_1$	$b*\cos(\theta_1+\theta_2)$
0	0
0	0
0	0
1	1

F_x =

-12 N
-24N

θ_1 = 33.4 deg

θ_2 = -69 deg

a = 0.18m

b = 0.38m

$\tau = J_{ee}^T(q) * F_x$

τ =

-12.456 Nm

-10.068 Nm

Given the gear ratios of 1:5 in the upper joint and 1:4.5 in the lower these number are very close to the 2 Nm output of the RJX motors. With the budget restriction, the choice to stick with these motors was made. The thought was that the force in the upper joint was excessive due to the -12N pull in the x direction. Even with the hope that these calculations were excessive it seemed that after we built the dog the RJX motors weren't enough to get the job done. This was most likely due to losses in the system and the fact that these RJX motors had already been worn out on a previous lab project. After a bit of research, the Savox motors of a similar size were found, which meant we would not have to adjust our motor mounts, but had a 1.4 more Newton-meters. The only tradeoff was that these Savox motors ran a bit slower. With this the decision was made to just replace the lower motors as those needed the most torque when it came to upward motion with the long forearm segment.

In order to power these motors a four cell LiPo battery was used. To use this battery Buck Converters to step down the voltage for each motor were used because the RJX motors required

8.4V and the Savox motors required 7.4V. The Buck Converters chosen worked well for the prototype leg so there was little reason to move toward a different converter. These converters allowed the incoming 16.3V to be brought down to the desired output for every motor as well as the Spider Board.

Finally, sensors were needed for our system to work well. Since using the servos had built in potentiometers controlling their movement was trivial. It only took sending the desired angle to the servos to make them move to that angle. Even with this convenience, knowing the actual angle of the leg joint is important. The actual angle reports what the angle is even though the desired angle sent to the motor might be different. This gives the control system how far a given leg has made it through its phase and serves as an input for our control system. Simple 10K Ohm potentiometers were used to calculate this angle and calibration code was written and used that gave the potentiometer output as a motor angle. This not only monitored the legs during movement, but made it possible for the dog to be placed in a desired position, to record the necessary motor outputs from the potentiometers and then run it in code to get the same result.

Code Design:

Due to the constraints mentioned before, the two gaits used for testing the quadruped were the trot gait and the lateral sequence gait. Both of the gaits were programmed based off of the timing diagram shown in Figure 1. Using a timing structure that alternates the point to point movement of the legs at given times. This timing structure cycles to keep movement forward. PID control is used to increase the time between timing phases. The input for this control is the actual angle of the legs versus the desired angle. Extra time is given based off the leg needing the most time to catch up, while the others remain in their desired positions.

To move to their desired positions, the legs were calibrated so that a given "xy" position could be requested and the processor would be able to determine the necessary motor outputs to achieve it (See Appendix C for Potentionmeter Calibration Code). The equation for this were determined through calculations made in MatLab (See Appendix B) whose inputs were determined by measure the leg's maximum ranges of motion. A full look at the code can be seen in Appendix A.

Construction, Testing, Results

Machining Parts:

For cutting and drilling the lengths of the legs, there were the various tools in the Washburn machine shops for this purpose. As mentioned earlier, when drilling holes for the mounts, precision was essential, or else the chain and sprocket system wouldn't work properly. The lower servo mounts had pieces that were cut by bandsaw, to ensure a clean cut, and for convenience. The most important piece was the plate that the servomotor was screwed into. This piece was machined with a CNC Mill. We did every part of the CNC process, from the making the CAD, to using the CAM to define the tool paths, to assembling and loading the tools, and finally operating the machine.

Only the single plate was machined, due to issues with fixturing parts with angles in it. When holding a workpiece in a vice for machining, there is a large force applied by the vice onto the piece. With an angled piece, this concentrates the stress from the vice onto the angle, and with thin pieces this can cause deformation. By just machining the one single plate, the rest of the mount would have to be assembled, so it wouldn't look as neat, and allow more imperfections. The plate was the most important part though, and it was essential for this part to

be precise so that the motor would be held securely, and prevent losses from the motor moving in its mount.

3D Printing Parts:

For sensor mounts, strong, light parts that could be made quickly were necessary, so that multiple designs could be tested. For this the sensor mounts and part of the elastic system were 3D printed, with a PLA 3D printing filament. For the pot mounts the pieces were attached, so that one end holds an end of the potentiometer still and the other end follows the rotation of the joint. For the elastic system, the 3D printed piece was essentially a substitute for a drum, and so a means to increase the moment arm and get more torque out of the elastic system. There wasn't enough space on the shaft of the lower joint to actually add a separate drum, so the printed piece was fitted over the collars that held the legs in place.

The pot mounts consisted of 2 pieces. One piece was used to hold the potentiometer itself, and then attach to the shaft of the lower joint. These pieces had to be dimensioned to fit the shaft, since glue and leaving a hole for a set screw weren't options. The printed pieces would be too thin, and the material would hold a threading poorly. Most glues also didn't work well when trying to glue a PLA piece and an aluminum piece. In order to get a proper fit, tolerancing fit tables were used to help get a good idea of how to dimension the 3D printed piece. The size of the shaft is measurable, and so for the piece that needs to fit over the shaft can simply be the difference in clearance, since the assumption is made that the printer is very accurate with the tolerance of its printed pieces. For these pieces, a Locational Clearance fit was used, after some experimentation with some various other fits. A Locational Clearance fit provides a snug fit for stationary parts, but can still be disassembled and reassembled easily. This was ideal because often times the pieces around the lower joint would need to be adjusted. A similar approach was

also used to fit the potentiometer itself to the mount, while accounting for the lip on the potentiometer for the wires.

In order to keep the turning end of the potentiometer still, the end would have to held with some still part on the robot. First the printed part for the potentiometer was fitted to the semicircular shape of the potentiometer. The shape of the other end depended on which joint that the potentiometer was on. For the lower joint, this piece was attached to the servo motor that powered the lower joint, which was mounted to the upper leg. Since there would be no forces on this piece, the piece was simply fitted to the size of the motor with two prongs. The piece was held by the friction of the fits. For the upper joints, the piece was attached to the frame of the robot itself. To prevent any complications, this piece was made as simply as possible, a long rod with a hole at the end to screw it to the aluminum 80/20 frame. An added benefit was by printing the same piece twice, the positioning of the legs could be adjusted to the same place, the same distance from the end of the frame. Later, brackets were added to maintain stability of this piece, since it is only attached by screw at one end, this allows it to rotate along the axis of the screw. Adding brackets helps to restrict this rotation, and ensures that the piece stays in place during the running of the leg.

For the elastic system, there wasn't enough space on the lower shaft to place a drum, as previously planned. The compromise was made, so that a piece was made to fit over a piece that was already there in previous designs, the collars that held the legs in place on the shafts. These collars were held by set screw, and so these set screws were replaced, and also used to hold the piece in place on the collar. This way, the printed piece was held secure to the collar and thus the shaft, and the screw would still work as a set screw. In terms of attaching the elastic system, that posed some challenges as well. After deciding to use latex tubes for the elastic system, one end

could be looped to the mount for the servo motor. The other end would have to be attached to the printed piece. After testing a few designs, the way it was done was to add a hole to the printed piece, to loop the tube through. Then, to attach two ends of the latex tube, a zip tie was used. A groove was added to the piece so that the loop that the latex tube was in would be fairly uniform. These parts worked well, and always held, even through rigorous stress testing.

Elastic System Results:

The parallel elastics used for RoboDog provide the necessary torque for RoboDog to stand without power. In addition, the elastics allowed for much more efficient standing. During testing standing would normally pull 0.51 amps, but with the parallel elastic system, the current draw was around 0.01-0.04 amps. Full cost of travel (COT) calculation were not completed, but it seems reasonable from previous small scale testing that current draw during high stress situations (contact phase), would be reduced. This amp draw may be shifted to the lifting phase of a given leg, but still keeps the motor further away from stall torque current draw.

Control Testing:

The trot gait that RoboDog was capable of producing was not ideal; however, it proved capability of the robotic system. During the trot gait, RoboDog made short and quick movements of the feet and shuffled forward. An increase in the distance of the feet travel could potentially make this trot gait better, but the vertical height might need to be reduced to increase the useful workspace. Due to the servo motors in use, angles of motion were limited and thus so was the workspace. Graphs of these workspaces can be found below in Figure 13.

RoboDog's lateral sequence gait was more promising as the quadruped was capable of making more natural looking steps forward. This gait led to some balance issues, which could be

potentially be fixed with an Inertial Measurement Unit (IMU). Aside from the balance issues the lifting of the front paws was limited. This resulted in the front feet not being capable of moving over rougher terrain. A redistribution of the center of gravity toward the center of the quadruped and/or stronger motors could help to fix this.

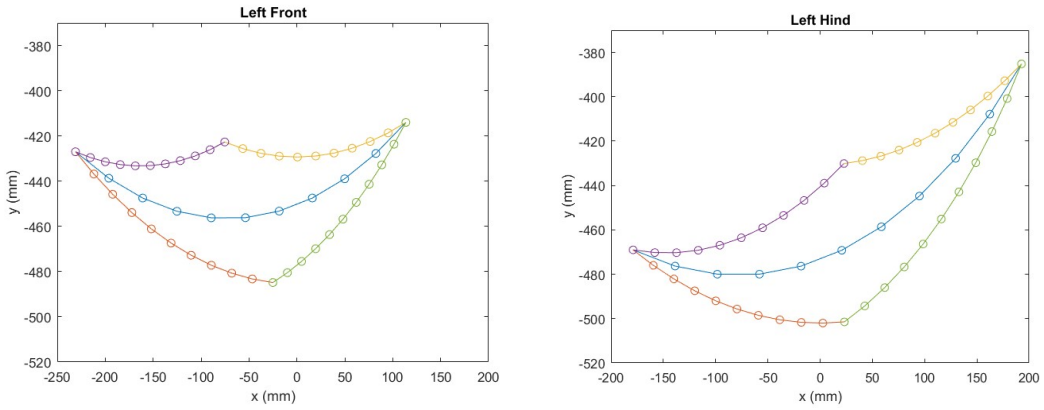


Figure 13 a: Left Side leg workspaces

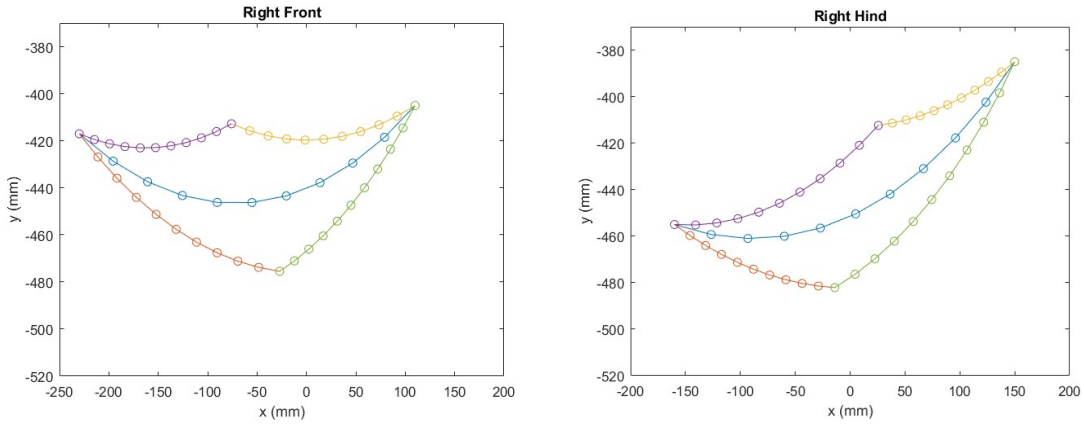


Figure 13 b: Right Side leg workspaces

Conclusion

As a result of this project, it has been found that parallel elastics have been capable of helping motors with less desirable torque outputs. Without these parallel elastics, RoboDog was only just capable of lifting itself up, while with them it could stand without power. It has also been found that it is possible to create a quadruped of this size capable of walking while still only

spending a fraction of the cost used for more refined quadrupeds like Spot and MIT Cheetah. Unfortunately, RoboDog did not meet all the goals that were originally set out. The dimensions were met, but the performance was not. While being capable of walking, RoboDog was not capable of going over coarse gravel due to motor strength and weight distribution, which lead to less than desirable foot clearance in the front.

For future work, the use of an IMU for balancing would be advised. This will allow for some much more interesting control choices. In addition to an IMU, stronger motors that are not limited in their rotation like the current servos should be explored. These new motors will allow for more gait options, foot pathing and a much easier time with controls. As the calculations showed, stronger motors were probably needed and the replacement of solely the lower leg joint might have only alleviated some of the force issues. Simply replacing the upper motors with more Savox motors with better controls might solve a majority of the issues RoboDog faces, but more exact calculations of the required torques should be completed before that choice is made. For controls, the use of a cosine function for the “x” coordinates and a sin function for the “y” coordinates of the foot pathing could be used to create a decent spline curve. These function could be stretched and altered through their amplitudes and potential through their periods. Though not specifically used, the following is for those interested in more detailed control of legged robots [8].

References

1. Wikipedia."Gait". *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Gait [Accessed: Web. 26 Apr. 2017].
2. Nunamaker, David, and Peter Blauner. "Normal And Abnormal Gait". *Cal.vet.upenn.edu*. [Online]. Available: http://cal.vet.upenn.edu/projects/saortho/chapter_91/91mast.htm.[Accessed: Web. 26 Apr. 2017].
3. Grahund."Jackie on the treadmill in slow motion—part 1",*Youtube.com*, 5-September-2011.[Online]. Available: https://www.youtube.com/watch?v=zrqOAwCPowg.[Accessed: Web. 26 Apr. 2017].
4. D. Prindle, "Watch Google's new robo-dog recover from an unexpected kick," *Digital Trends*, 11-Feb-2015. [Online].Available: http://www.digitaltrends.com/cool-tech/boston-dynamics-spot-robot/. [Accessed: 26-Apr-2017].
5. Wikipedia."Denavit–Hartenberg Parameters". *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters [Accessed: Web. 26 Apr. 2017].
6. T. DeWolf, "Robot control part 2: Jacobians, velocity, and force," *StudyWolf*, 18-Mar-2017. [Online]. Available: https://studywolf.wordpress.com/2013/09/02/robot-control-jacobians-velocity-and-force/. [Accessed: 26-Apr-2017].
7. Grimmer, Martin et al. "A Comparison Of Parallel- And Series Elastic Elements In An Actuator For Mimicking Human Ankle Joint In Walking And Running - IEEE Xplore Document". 14-May-2012. *Ieeexplore.ieee.org*. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6224967&tag=1 [Accessed: Web. 26 Apr. 2017].
8. Pierre-Brice Wieber, Russ Tedrake, and Scott Kuindersma. Modeling and control of legged robots. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics, 2nd Ed*, chapter 48. Springer, 2015.

Appendix A: Arduino Code

```
//FinalCodeWithPID.ino

#include <Servo.h>
#include <PID.h>
#include <Math.h>

//servo pins
#define LH_upperPin 24
#define LH_lowerPin 25
#define LF_upperPin 41
#define LF_lowerPin 40
#define RH_upperPin 8
#define RH_lowerPin 6
#define RF_upperPin 5
#define RF_lowerPin 4

bool startLF = false;
bool startLH = false;
bool startRF = false;
bool startRH = false;

int flubb = 10; //variable to give pull phase an extra kick until control maybe
//if we want it to
double upY = 30; //in mm
double downY = -30; //in mm
double upX = 30; //in mm
double downX = -30; //in mm


//number of elements in point to point movement with standing as first element
const int arrayPoints = 4;
class TimeWiz {
  private:
  public:
    double lift = 1.5;
    double pull = 7;
    unsigned long alottedTimeLift = 0; //ms
    unsigned long alottedTimePull = 0; //ms
    unsigned long lastTime = 0;
    unsigned long timeSet = 0; //the original time when walking begins
    double unit = 700; //ms
    unsigned long runTime; //ms how long the specific move has been running


    void initialize(){
      this->alottedTimeLift = this->liftTime();
      this->alottedTimePull = this->pullTime();
    }

    //returns the lift time
    unsigned long liftTime(){
      return (unit*lift);
    }
    //returns the pull time
    unsigned long pullTime(){
      return (unit*pull);
    }
```

```
};
//angle of upper and lower with respect to the legs
class Leg {
  private:
  public:
    String name = "";
    double upperAngle = 90; //standing? position
    double lowerAngle = 90; //standing? position


    double standingX = 0;
    double standingY = 0;

    //create dimensions for leg in mm
    double upperLength = 0;
    double lowerLength = 0;

    //conversion for the actual to servo
    double lowerConversion = 0;
    double upperConversion = 0;
    double lowerInterc = 0;
    double upperInterc = 0;

    //previous angles
    double previousLower = 0;
    double previousUpper = 0;
    //previous x & y
    // double previousX = 0;
    // double previousY = 0;

    //standing boolean
    bool standing = true;

    //create servo objects
    Servo upperServo;
    Servo lowerServo;

    //boost increment for back legs
    double increment = 0;

    //calibration value for the pot offset
    double upperPotCalibration = 0;
    double upperPotInt = 0;
    double lowerPotCalibration = 0;
    double lowerPotInt = 0;

    //Time creation
    TimeWiz Time;

    //array of movements
    double upperArray[arrayPoints] = {0};
    double lowerArray[arrayPoints] = {0};

    //PID variables
    double input;
    double output = 0;
    double setPoint = 0;
    //constants
    int kp = 1;
    int ki = 0;
    int kd = 0;
```

```
    //PID direction
    bool motorDir = true; //Default is direct

    //PID controller
    PID pid = this->initializePID();

    //counter for part of legs step movement
    double counter = 0;
    //potPin
    int upperPotPin = 0;
    int lowerPotPin = 0;
    //state of leg is cycle (used in switch case)
    int state = 0;

    /*runs calculation to get the radian values

      ArrayPos conform to 0 being standing, 1 being state0, 2 being state1, 3 being state3

    */
    double calculations(double x, double y, int arrayPos){
      double a = this->upperLength;
      double b = this->lowerLength;

      double theta1 = 2*atan((2*a*y + (pow(a,2)*sqrt((- pow(a,2) + 2*a*b - pow(b,2) + pow(x,2)
+ pow(y,2))*(pow(a,2) + 2*a*b + pow(b,2) - pow(x,2) - pow(y,2))))/(- pow(a,2) + 2*a*b -
pow(b,2) + pow(x,2) + pow(y,2)) + (pow(b,2)*sqrt((- pow(a,2) + 2*a*b - pow(b,2) + pow(x,2) +
pow(y,2))*(pow(a,2) + 2*a*b + pow(b,2) - pow(x,2) - pow(y,2))))/(- pow(a,2) + 2*a*b - pow(b,2)
+ pow(x,2) + pow(y,2)) - (pow(x,2)*sqrt((- pow(a,2) + 2*a*b - pow(b,2) + pow(x,2) +
pow(y,2))*(pow(a,2) + 2*a*b + pow(b,2) - pow(x,2) - pow(y,2))))/(- pow(a,2) + 2*a*b - pow(b,2)
+ pow(x,2) + pow(y,2)) - (pow(y,2)*sqrt((- pow(a,2) + 2*a*b - pow(b,2) + pow(x,2) +
pow(y,2))*(pow(a,2) + 2*a*b + pow(b,2) - pow(x,2) - pow(y,2))))/(- pow(a,2) + 2*a*b - pow(b,2)
+ pow(x,2) + pow(y,2)) - (2*a*b*sqrt((- pow(a,2) + 2*a*b - pow(b,2) + pow(x,2) +
pow(y,2))*(pow(a,2) + 2*a*b + pow(b,2) - pow(x,2) - pow(y,2))))/(- pow(a,2) + 2*a*b - pow(b,2)
+ pow(x,2) + pow(y,2)))/(pow(a,2) + 2*a*x - pow(b,2) + pow(x,2) + pow(y,2))); //in radians
      double theta2 = 2*atan(sqrt((- pow(a,2) + 2*a*b - pow(b,2) + pow(x,2) +
pow(y,2))*(pow(a,2) + 2*a*b + pow(b,2) - pow(x,2) - pow(y,2)))/(- pow(a,2) + 2*a*b - pow(b,2)
+ pow(x,2) + pow(y,2))); //in radians
      Serial.print("Theta1: ");
      Serial.println(theta1);
      Serial.print("Theta2: ");
      Serial.println(theta2);
      double endLowerAngle = theta2*this->lowerConversion + this->lowerInterc;
      Serial.print("Lower After Conversion: ");
      Serial.println(endLowerAngle);
      double endUpperAngle = theta1*this->upperConversion + this->upperInterc;
      Serial.print("Upper After Conversion: ");
      Serial.println(endUpperAngle);
      //adjusts the array
      *(upperArray + arrayPos) = endUpperAngle;
      *(lowerArray + arrayPos) = endLowerAngle;
    }
    /*
    @param
    double endUpperAngle
    double endLowerAngle

    Takes the upper and lower motor outputs from calculations.

    */
    void moveDistance(double endUpperAngle, double endLowerAngle, char side){
```

```
      bool initLift = true;

      //standing will just be the angles set here
      if(this->standing){
        this->lowerAngle = endLowerAngle;
        this->upperAngle = endUpperAngle;
        this->previousLower = endLowerAngle;
        this->previousUpper = endUpperAngle;
        // this->previousX = x;
        // this->previousY = y;
      }


    }
    /*@return
        returns a pid controller that is used to calculate control of a given leg
    */
    PID initializePID(){
        //MotorOutput and PID controller
        MotorOutput motorOut(&this->output,this->motorDir);
        PID pid(&this->input,this->kp,this->ki,this->kd,&this->setPoint,motorOut);
        return pid;
    }

    //drive legs to predetermined angles
    void driveLeg() {
      //handles incrementation thats over capability
      if (this->upperAngle > 180) {
        this->upperAngle = 180;
      }
      else if (this->upperAngle < 10) {
        this->upperAngle = 10;
      }

      // Serial.print("Upper Angle: ");
      // Serial.println(this->upperAngle);
      //handles incrementation thats over capability (ADAPTED FOR NEW SERVOS)
      if (this->lowerAngle > 160) {
        this->lowerAngle = 160;
      }
      else if (this->lowerAngle < 10) {
        this->lowerAngle = 10;
      }

      //drive lower servo
      this->lowerServo.write(this->lowerAngle);
      //drive upper servo
      this->upperServo.write(this->upperAngle);

      // Serial.print("Lower Angle: ");
      // Serial.println(this->lowerAngle);

    }

    //gives angle percentage based off the start and end angles
    void movePercentage(double startAngleLower, double endAngleLower, double startAngleUpper,
double endAngleUpper, double stepAmount, double count) {


      double movePercL = (double)(startAngleLower - endAngleLower) / stepAmount;
```

```
      double movePercU = (double)(startAngleUpper - endAngleUpper) / stepAmount;

      this->lowerAngle = startAngleLower - (double)(movePercL * count);
      this->upperAngle = startAngleUpper - (double)(movePercU * count);



    }

    //reads the potentiometer and returns an appropriate angel for it
    //NEEDS TO BE FIXED FOR THE TWO POTCALIBRATION AND INT NUMBERS
     double angleRead(int x){
        double val = analogRead(x);
        double mappedVal = map(val,0,1023,0,270);
        return mappedVal;
    }


};

Leg LF; //front left
Leg LH; //back left
Leg RF; //front right
Leg RH; //back right


double liftStep = 3; //just because i have only to steps in lift now
double pullStep = 7; //4/02 thoughts are maybe to tweak this and counter timer thing for
separate legs

//intialize lastTime
unsigned long lastTime = 0;
bool flipTrot = true; //for standing trot



bool initializeTimeMove = false; //allows to set the first initial timeSet so that future
moves can use millis without issue

int legCycle = 0; //states

double LHupperPull = 0;
double LHlowerPull = 0;
double LHupperLift = 0; //mind have intermediate lift for all lifts but this will be final
before pull
double LHlowerLift = 0;

double LFupperPull = 0;
double LFlowerPull = 0;
double LFupperLift = 0;
double LFlowerLift = 0;

double RFupperPull = 0;
double RFlowerPull = 0;
double RFupperLift = 0;
double RFlowerLift = 0;

double RHupperPull = 0;
double RHlowerPull = 0;
double RHupperLift = 0;
double RHlowerLift = 0;
```

```
double sampleTime = 200; //for .1 seconds (HAS ABOUT 20% propogated error at worst over time)

void setup() {
  Serial.begin(9600);
  //give names to help with final output
  LF.name = "LF";
  LH.name = "LH";
  RF.name = "RF";
  RH.name = "RH";

  LH.upperConversion = -413.8;
  LH.lowerConversion = 271.7;
  LF.upperConversion = -382.6;
  LF.lowerConversion = 353.6;
  RH.upperConversion = 549.6;
  RH.lowerConversion = -288;
  RF.upperConversion = 380.2;
  RF.lowerConversion = -362.3;

  LH.upperInterc = -885.6; //decrease by 10
  LH.lowerInterc = -239.7; //decrease by 10
  RH.upperInterc = 1448;
  RH.lowerInterc = 451.1;
  RF.upperInterc = 1096;
  RF.lowerInterc = 560.4;
  LF.upperInterc = -880.5;
  LF.lowerInterc = -349.4;

  LH.upperLength = 174;
  LH.lowerLength = 377;
  RH.upperLength = 170;
  RH.lowerLength = 370;
  RF.upperLength = 170;
  RF.lowerLength = 375;
  LF.upperLength = 175;
  LF.lowerLength = 370;


  //potentiometers
  LH.upperPotPin = A10;
  LH.lowerPotPin = A15;
  LF.upperPotPin = A8;
  LF.lowerPotPin = A11;
  RH.upperPotPin = A7;
  RH.lowerPotPin = A6;
  RF.upperPotPin = A9;
  RF.lowerPotPin = A13;

  LH.upperPotCalibration = 85.00;
  LH.upperPotInt = -21750.00;
  LH.lowerPotCalibration = 3.85;
  LH.lowerPotInt = -340.00;

  LF.upperPotCalibration = 6.54;
  LF.upperPotInt = -1016.54;
  LF.lowerPotCalibration = 4.05;
  LF.lowerPotInt = -241.35;

  RH.upperPotCalibration = 7.39;
  RH.upperPotInt = -751.30;
```

```
  RH.lowerPotCalibration = 4.41;
  RH.lowerPotInt = -272.35;

  RF.upperPotCalibration = 5.86;
  RF.upperPotInt = -816.55;
  RF.lowerPotCalibration = 4.05;
  RF.lowerPotInt = -345.76;


  //states for movement in state functions of legs
  // LH.state = 3;
  // LF.state = 0;
  // RH.state = 3;
  // RF.state = 3;

  LH.lowerAngle = 50; //brought up to let right back come down
  RH.lowerAngle = 150;
  RF.lowerAngle = 130;
  LF.lowerAngle = 80; //brought down
  LH.upperAngle = 140;
  LF.upperAngle = 130;
  RF.upperAngle = 50;
  RH.upperAngle = 60; //went up

  LHupperPull = 0;
  LHlowerPull = 0;
  LHupperLift = 0; //mind have intermediate lift for all lifts but this will be final before
pull
  LHlowerLift = 0;

  LFupperPull = 0;
  LFlowerPull = 0;
  LFupperLift = 0;
  LFlowerLift = 0;

  RFupperPull = 0;
  RFlowerPull = 0;
  RFupperLift = 0;
  RFlowerLift = 0;

  RHupperPull = 0;
  RHlowerPull = 0;
  RHupperLift = 0;
  RHlowerLift = 0;

  // RH.counter = 6; //based off excel they should have completed this much movement
  // RF.counter = 4;
  // LF.counter = 1;

  // RH.Time.runTime = 4200; //6*unit
  // RF.Time.runTime = 2800; //4*unit
  // LF.Time.runTime = 700; //1*unit

  //need to get them to this pose pause then let it run?? USE pots and PID to control speed of
the gait (-- counter for everyone based off PID??)

  LH.pid.setMinMax(10,180); //set for servo motors
  LF.pid.setMinMax(10,180);
  RH.pid.setMinMax(10,180);
  RF.pid.setMinMax(10,180);
```

```
  //ATTACH all the motors
  LH.upperServo.attach(LH_upperPin);
  LH.lowerServo.attach(LH_lowerPin);

  LF.upperServo.attach(LF_upperPin);
  LF.lowerServo.attach(LF_lowerPin);

  RH.upperServo.attach(RH_upperPin);
  RH.lowerServo.attach(RH_lowerPin);

  RF.upperServo.attach(RF_upperPin);
  RF.lowerServo.attach(RF_lowerPin);
}

void loop() {

  unsigned long now = millis();
  unsigned long timeChange = (now - lastTime);


  //standing for x milliseconds
  standing(5000);

  //trot for x milliseconds
//  trotWalk(18000);
//  standing(100000);

  LH.standing = false;
  LF.standing = false;
  RH.standing = false;
  RF.standing = false;

  LHupperPull = 150;// from 180
  LHlowerPull = 60;// from 10
  LHupperLift = LH.upperArray[1]-10; //mind have intermediate lift for all lifts but this will
be final before pull
  LHlowerLift = LH.lowerArray[1]+40; //30 for more lift

  LFupperPull = 88.7 + 30;
  LFlowerPull = 96.9 - 20;
  LFupperLift = 75.6;
  LFlowerLift = 155.6;

  RFupperPull = 138.6 - 40;
  RFlowerPull = 134 + 30;
  RFupperLift = 180;
  RFlowerLift = 30.9;

  RHupperPull = 10;
  RHlowerPull = 150;// from 160
  RHupperLift = RH.upperArray[1]+10;
  RHlowerLift = RH.lowerArray[1]-30; //30 for more lift

  double sort = 0;
  double threshHold = 5; //5 degree leeway

  if (timeChange >= sampleTime) {

    Serial.println("Start of Cycle");
    Serial.println(timeChange);
```

```
switch (legCycle){
  case 0:

    sampleTime = 200; //reset time
    //LH
    Serial.println("Made it to 0");
    LH.movePercentage(LHlowerLift,LHlowerPull,LHupperLift,LHupperPull,pullStep,1);
    //LF lift 2
    LF.lowerAngle = LFlowerLift;
    LF.upperAngle = LFupperLift;
    //RF
    RF.movePercentage(RFlowerLift,RFlowerPull,RFupperLift,RFupperPull,pullStep,5);

    //RH
    RH.movePercentage(RHlowerLift,RHlowerPull,RHupperLift,RHupperPull,pullStep,7);

    LH.driveLeg();
    LF.driveLeg();
    RF.driveLeg();
    RH.driveLeg();

    LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
    LH.setPoint = LH.lowerAngle;

    if(abs(LH.input - LH.setPoint) <= threshHold){
      LH.output = 0;
    }
    else{
      LH.pid.outputControl();
    }


    LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
    LF.setPoint = LF.lowerAngle;

    if(abs(LF.input - LF.setPoint) <= threshHold){
      LF.output = 0;
    }
    else{
      LF.pid.outputControl();
    }

    RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
    RH.setPoint = RH.lowerAngle;

    if(abs(RH.input - RH.setPoint) <= threshHold){
      RH.output = 0;
    }
    else{
      RH.pid.outputControl();
    }

    RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
    RF.setPoint = RF.lowerAngle;


    if(abs(RF.input - RF.setPoint) <= threshHold){
      RF.output = 0;
    }
```

```
        else{
          RF.pid.outputControl();
        }

        if(LH.output>RH.output){
          sort = LH.output;
        }
        else{
          sort = RH.output;
        }

        if(sort < RF.output){
          sort = RF.output;
        }
        if(sort < LF.output){
          sort = LF.output;
        }

        sampleTime += sort;

        break;
      case 1:
        sampleTime = 200; //reset time
        //LH
        Serial.println("Made it to 1");
        LH.movePercentage(LHlowerLift,LHlowerPull,LHupperLift,LHupperPull,pullStep,2);
        //LF lift 3
        LF.lowerAngle = LFlowerLift-10;
        LF.upperAngle = LFupperLift;
        //RF
        RF.movePercentage(RFlowerLift,RFlowerPull,RFupperLift,RFupperPull,pullStep,6);

        //RH lift 1
        RH.lowerAngle = RHlowerLift; //might need to be higher not sure depends but dont move
the uppe yet

        LH.driveLeg();
        LF.driveLeg();
        RF.driveLeg();
        RH.driveLeg();

        LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
        LH.setPoint = LH.lowerAngle;
        if(abs(LH.input - LH.setPoint) <= threshHold){
          LH.output = 0;
        }
        else{
          LH.pid.outputControl();
        }

        LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
        LF.setPoint = LF.lowerAngle;
        if(abs(LF.input - LF.setPoint) <= threshHold){
          LF.output = 0;
        }
        else{
          LF.pid.outputControl();
        }

        RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
        RH.setPoint = RH.lowerAngle;
```

```
        if(abs(RH.input - RH.setPoint) <= threshHold){
          RH.output = 0;
        }
        else{
          RH.pid.outputControl();
        }

        RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
        RF.setPoint = RF.lowerAngle;
        if(abs(RF.input - RF.setPoint) <= threshHold){
          RF.output = 0;
        }
        else{
          RF.pid.outputControl();
        }


        if(LH.output>RH.output){
          sort = LH.output;
        }
        else{
          sort = RH.output;
        }

        if(sort < RF.output){
          sort = RF.output;
        }
        if(sort < LF.output){
          sort = LF.output;
        }

        sampleTime += sort;

        break;
      case 2:
        sampleTime = 200; //reset time
        //LH
        LH.movePercentage(LHlowerLift,LHlowerPull,LHupperLift,LHupperPull,pullStep,3);
        //LF
        LF.movePercentage(LFlowerLift,LFlowerPull,LFupperLift,LFupperPull,pullStep,1);

        //RF
        RF.movePercentage(RFlowerLift,RFlowerPull,RFupperLift,RFupperPull,pullStep,7);

        //RH lift 2
        RH.lowerAngle = RHlowerLift; //might need to be higher not sure depends but dont move
the uppe yet
        RH.upperAngle = RHupperLift;

        LH.driveLeg();
        LF.driveLeg();
        RF.driveLeg();
        RH.driveLeg();

        LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
        LH.setPoint = LH.lowerAngle;
        if(abs(LH.input - LH.setPoint) <= threshHold){
          LH.output = 0;
        }
        else{
          LH.pid.outputControl();
```

```
        }

        LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
        LF.setPoint = LF.lowerAngle;
        if(abs(LF.input - LF.setPoint) <= threshHold){
          LF.output = 0;
        }
        else{
          LF.pid.outputControl();
        }

        RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
        RH.setPoint = RH.lowerAngle;
        if(abs(RH.input - RH.setPoint) <= threshHold){
          RH.output = 0;
        }
        else{
          RH.pid.outputControl();
        }

        RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
        RF.setPoint = RF.lowerAngle;
        if(abs(RF.input - RF.setPoint) <= threshHold){
          RF.output = 0;
        }
        else{
          RF.pid.outputControl();
        }

        if(LH.output>RH.output){
          sort = LH.output;
        }
        else{
          sort = RH.output;
        }

        if(sort < RF.output){
          sort = RF.output;
        }
        if(sort < LF.output){
          sort = LF.output;
        }

        sampleTime += sort;

        break;
      case 3:
        sampleTime = 200; //reset time
        //LH
        LH.movePercentage(LHlowerLift,LHlowerPull,LHupperLift,LHupperPull,pullStep,4);
        //LF
        LF.movePercentage(LFlowerLift,LFlowerPull,LFupperLift,LFupperPull,pullStep,2);

        //RF lift 1
        RF.lowerAngle = RFlowerLift;

        //RH lift 3
        RH.lowerAngle = RHlowerLift+10; //might need to be higher not sure depends but dont
move the uppe yet
        RH.upperAngle = RHupperLift;
```

```
  LH.driveLeg();
  LF.driveLeg();
  RF.driveLeg();
  RH.driveLeg();

  LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
  LH.setPoint = LH.lowerAngle;
  if(abs(LH.input - LH.setPoint) <= threshHold){
    LH.output = 0;
  }
  else{
    LH.pid.outputControl();
  }

  LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
  LF.setPoint = LF.lowerAngle;
  if(abs(LF.input - LF.setPoint) <= threshHold){
    LF.output = 0;
  }
  else{
    LF.pid.outputControl();
  }

  RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
  RH.setPoint = RH.lowerAngle;
  if(abs(RH.input - RH.setPoint) <= threshHold){
    RH.output = 0;
  }
  else{
    RH.pid.outputControl();
  }

  RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
  RF.setPoint = RF.lowerAngle;
  if(abs(RF.input - RF.setPoint) <= threshHold){
    RF.output = 0;
  }
  else{
    RF.pid.outputControl();
  }

  if(LH.output>RH.output){
    sort = LH.output;
  }
  else{
    sort = RH.output;
  }

  if(sort < RF.output){
    sort = RF.output;
  }
  if(sort < LF.output){
    sort = LF.output;
  }

  sampleTime += sort;

  break;
case 4:
  sampleTime = 200; //reset time
  //LH
```

```
LH.movePercentage(LHlowerLift,LHlowerPull,LHupperLift,LHupperPull,pullStep,5);
//LF
LF.movePercentage(LFlowerLift,LFlowerPull,LFupperLift,LFupperPull,pullStep,3);

//RF lift 2
RF.lowerAngle = RFlowerLift;
RF.upperAngle = RFupperLift;

//RH
RH.movePercentage(RHlowerLift,RHlowerPull,RHupperLift,RHupperPull,pullStep,1);

LH.driveLeg();
LF.driveLeg();
RF.driveLeg();
RH.driveLeg();

LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
LH.setPoint = LH.lowerAngle;
if(abs(LH.input - LH.setPoint) <= threshHold){
  LH.output = 0;
}
else{
  LH.pid.outputControl();
}

LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
LF.setPoint = LF.lowerAngle;
if(abs(LF.input - LF.setPoint) <= threshHold){
  LF.output = 0;
}
else{
  LF.pid.outputControl();
}

RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
RH.setPoint = RH.lowerAngle;
if(abs(RH.input - RH.setPoint) <= threshHold){
  RH.output = 0;
}
else{
  RH.pid.outputControl();
}

RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
RF.setPoint = RF.lowerAngle;
if(abs(RF.input - RF.setPoint) <= threshHold){
  RF.output = 0;
}
else{
  RF.pid.outputControl();
}

if(LH.output>RH.output){
  sort = LH.output;
}
else{
  sort = RH.output;
}

if(sort < RF.output){
  sort = RF.output;
```

```
  }
  if(sort < LF.output){
    sort = LF.output;
  }

  sampleTime += sort;

  break;
case 5:
  sampleTime = 200; //reset time
  //LH
  LH.movePercentage(LHlowerLift,LHlowerPull,LHupperLift,LHupperPull,pullStep,6);
  //LF
  LF.movePercentage(LFlowerLift,LFlowerPull,LFupperLift,LFupperPull,pullStep,4);

  //RF lift 3
  RF.lowerAngle = RFlowerLift+10;
  RF.upperAngle = RFupperLift;

  //RH
  RH.movePercentage(RHlowerLift,RHlowerPull,RHupperLift,RHupperPull,pullStep,2);

  LH.driveLeg();
  LF.driveLeg();
  RF.driveLeg();
  RH.driveLeg();

  LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
  LH.setPoint = LH.lowerAngle;
  if(abs(LH.input - LH.setPoint) <= threshHold){
    LH.output = 0;
  }
  else{
    LH.pid.outputControl();
  }

  LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
  LF.setPoint = LF.lowerAngle;
  if(abs(LF.input - LF.setPoint) <= threshHold){
    LF.output = 0;
  }
  else{
    LF.pid.outputControl();
  }

  RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
  RH.setPoint = RH.lowerAngle;
  if(abs(RH.input - RH.setPoint) <= threshHold){
    RH.output = 0;
  }
  else{
    RH.pid.outputControl();
  }

  RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
  RF.setPoint = RF.lowerAngle;
  if(abs(RF.input - RF.setPoint) <= threshHold){
    RF.output = 0;
  }
  else{
    RF.pid.outputControl();
```

```
        }

        if(LH.output>RH.output){
          sort = LH.output;
        }
        else{
          sort = RH.output;
        }

        if(sort < RF.output){
          sort = RF.output;
        }
        if(sort < LF.output){
          sort = LF.output;
        }

        sampleTime += sort;

        break;
      case 6:
        sampleTime = 200; //reset time
        //LH
        LH.movePercentage(LHlowerLift,LHlowerPull,LHupperLift,LHupperPull,pullStep,7); //
might need to speed up and have this complete already
        //LF
        LF.movePercentage(LFlowerLift,LFlowerPull,LFupperLift,LFupperPull,pullStep,5);

        //RF
        RF.movePercentage(RFlowerLift,RFlowerPull,RFupperLift,RFupperPull,pullStep,1);

        //RH
        RH.movePercentage(RHlowerLift,RHlowerPull,RHupperLift,RHupperPull,pullStep,3);

        LH.driveLeg();
        LF.driveLeg();
        RF.driveLeg();
        RH.driveLeg();

        LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
        LH.setPoint = LH.lowerAngle;
        if(abs(LH.input - LH.setPoint) <= threshHold){
          LH.output = 0;
        }
        else{
          LH.pid.outputControl();
        }

        LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
        LF.setPoint = LF.lowerAngle;
        if(abs(LF.input - LF.setPoint) <= threshHold){
          LF.output = 0;
        }
        else{
          LF.pid.outputControl();
        }

        RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
        RH.setPoint = RH.lowerAngle;
        if(abs(RH.input - RH.setPoint) <= threshHold){
          RH.output = 0;
        }
```

```
  else{
    RH.pid.outputControl();
  }

  RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
  RF.setPoint = RF.lowerAngle;
  if(abs(RF.input - RF.setPoint) <= threshHold){
    RF.output = 0;
  }
  else{
    RF.pid.outputControl();
  }

  if(LH.output>RH.output){
    sort = LH.output;
  }
  else{
    sort = RH.output;
  }

  if(sort < RF.output){
    sort = RF.output;
  }
  if(sort < LF.output){
    sort = LF.output;
  }

  sampleTime += sort;

  break;
case 7:
  sampleTime = 200; //reset time
  //LH lift 1
  LH.lowerAngle = LHlowerLift;

  //LF
  LF.movePercentage(LFlowerLift,LFlowerPull,LFupperLift,LFupperPull,pullStep,6);

  //RF
  RF.movePercentage(RFlowerLift,RFlowerPull,RFupperLift,RFupperPull,pullStep,2);

  //RH
  RH.movePercentage(RHlowerLift,RHlowerPull,RHupperLift,RHupperPull,pullStep,4);

  LH.driveLeg();
  LF.driveLeg();
  RF.driveLeg();
  RH.driveLeg();

  LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
  LH.setPoint = LH.lowerAngle;
  if(abs(LH.input - LH.setPoint) <= threshHold){
    LH.output = 0;
  }
  else{
    LH.pid.outputControl();
  }

  LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
  LF.setPoint = LF.lowerAngle;
  if(abs(LF.input - LF.setPoint) <= threshHold){
```

```
    LF.output = 0;
  }
  else{
    LF.pid.outputControl();
  }

  RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
  RH.setPoint = RH.lowerAngle;
  if(abs(RH.input - RH.setPoint) <= threshHold){
    RH.output = 0;
  }
  else{
    RH.pid.outputControl();
  }

  RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
  RF.setPoint = RF.lowerAngle;
  if(abs(RF.input - RF.setPoint) <= threshHold){
    RF.output = 0;
  }
  else{
    RF.pid.outputControl();
  }

  if(LH.output>RH.output){
    sort = LH.output;
  }
  else{
    sort = RH.output;
  }

  if(sort < RF.output){
    sort = RF.output;
  }
  if(sort < LF.output){
    sort = LF.output;
  }

  sampleTime += sort;

  break;
case 8:
  sampleTime = 200; //reset time
   //LH lift 2
  LH.lowerAngle = LHlowerLift;
  LH.upperAngle = LHupperLift;

  //LF
  LF.movePercentage(LFlowerLift,LFlowerPull,LFupperLift,LFupperPull,pullStep,7);

  //RF
  RF.movePercentage(RFlowerLift,RFlowerPull,RFupperLift,RFupperPull,pullStep,3);

  //RH
  RH.movePercentage(RHlowerLift,RHlowerPull,RHupperLift,RHupperPull,pullStep,5);

  LH.driveLeg();
  LF.driveLeg();
  RF.driveLeg();
  RH.driveLeg();
```

```
  LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
  LH.setPoint = LH.lowerAngle;
  if(abs(LH.input - LH.setPoint) <= threshHold){
    LH.output = 0;
  }
  else{
    LH.pid.outputControl();
  }

  LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
  LF.setPoint = LF.lowerAngle;
  if(abs(LF.input - LF.setPoint) <= threshHold){
    LF.output = 0;
  }
  else{
    LF.pid.outputControl();
  }

  RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
  RH.setPoint = RH.lowerAngle;
  if(abs(RH.input - RH.setPoint) <= threshHold){
    RH.output = 0;
  }
  else{
    RH.pid.outputControl();
  }

  RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
  RF.setPoint = RF.lowerAngle;
  if(abs(RF.input - RF.setPoint) <= threshHold){
    RF.output = 0;
  }
  else{
    RF.pid.outputControl();
  }

  if(LH.output>RH.output){
    sort = LH.output;
  }
  else{
    sort = RH.output;
  }

  if(sort < RF.output){
    sort = RF.output;
  }
  if(sort < LF.output){
    sort = LF.output;
  }

  sampleTime += sort;

  break;
case 9:
  sampleTime = 200; //reset time
  //LH lift 3
  LH.lowerAngle = LHlowerLift-10;
  LH.upperAngle = LHupperLift;

  //LF lift 1
  LF.lowerAngle = LFlowerLift;
```

```
//RF
RF.movePercentage(RFlowerLift,RFlowerPull,RFupperLift,RFupperPull,pullStep,4);

//RH
RH.movePercentage(RHlowerLift,RHlowerPull,RHupperLift,RHupperPull,pullStep,6);

LH.driveLeg();
LF.driveLeg();
RF.driveLeg();
RH.driveLeg();

LH.input = LH.angleRead(LH.lowerPotPin)*LH.lowerPotCalibration + LH.lowerPotInt;
LH.setPoint = LH.lowerAngle;
if(abs(LH.input - LH.setPoint) <= threshHold){
  LH.output = 0;
}
else{
  LH.pid.outputControl();
}

LF.input = LF.angleRead(LF.lowerPotPin)*LF.lowerPotCalibration + LF.lowerPotInt;
LF.setPoint = LF.lowerAngle;
if(abs(LF.input - LF.setPoint) <= threshHold){
  LF.output = 0;
}
else{
  LF.pid.outputControl();
}

RH.input = RH.angleRead(RH.lowerPotPin)*RH.lowerPotCalibration + RH.lowerPotInt;
RH.setPoint = RH.lowerAngle;
if(abs(RH.input - RH.setPoint) <= threshHold){
  RH.output = 0;
}
else{
  RH.pid.outputControl();
}

RF.input = RF.angleRead(RF.lowerPotPin)*RF.lowerPotCalibration + RF.lowerPotInt;
RF.setPoint = RF.lowerAngle;
if(abs(RF.input - RF.setPoint) <= threshHold){
  RF.output = 0;
}
else{
  RF.pid.outputControl();
}

if(LH.output>RH.output){
  sort = LH.output;
}
else{
  sort = RH.output;
}

if(sort < RF.output){
  sort = RF.output;
}
if(sort < LF.output){
  sort = LF.output;
}
```

```
        sampleTime += sort;

        break;
    }

    lastTime = now;
    if(legCycle == 9){
      Serial.println(legCycle);
      legCycle = 0;
    }
    else{
      legCycle++;
      Serial.println("++");
    }
  }



}
void trotWalk(int x){
  while(millis() <= x){//millis() <= x is real code
    unsigned long now = millis();
    unsigned long timeChange = (now - lastTime);

    if(flipTrot && timeChange > 500){
      LF.moveDistance(LF.upperArray[0],LF.lowerArray[0],'L');
      LF.lowerAngle -= 20;
      LF.upperAngle += 20;
//      LF.lowerAngle += 20; //cuz it needs to be up more

      LH.moveDistance(LH.upperArray[1],LH.lowerArray[1],'L');
//      LH.lowerAngle -= 20; //extra forward boost
      LH.lowerAngle += 10; //more lift


      RF.moveDistance(RF.upperArray[1],RF.lowerArray[1],'R');
//      RF.lowerAngle -= 20; //extra lift

      RH.moveDistance(RH.upperArray[3],RH.lowerArray[3],'R');
      RH.lowerAngle = 160;
      RH.upperAngle = 10;

      flipTrot = !flipTrot;

      lastTime = now;
    }
    else if(!flipTrot && timeChange > 500){
      LF.moveDistance(LF.upperArray[1],LF.lowerArray[1],'L');

//      LF.lowerAngle += 10; //cuz it needs to be up more

      LH.moveDistance(LH.upperArray[3],LH.lowerArray[3],'L');
      LH.lowerAngle = 10;
      LH.upperAngle = 180;

      RF.moveDistance(RF.upperArray[0],RF.lowerArray[0],'R');
      RF.lowerAngle += 20;
      RF.upperAngle -= 20;
```

```
      RH.moveDistance(RH.upperArray[1],RH.lowerArray[1],'R');
      RH.lowerAngle -= 10; //more lift

      flipTrot = !flipTrot;

      lastTime = now;
    }

    LH.driveLeg();
    LF.driveLeg();
    RH.driveLeg();
    RF.driveLeg();
  }
}
void bowDown(int x){
   while (millis() <= x) { //millis() <= 5000
    //calculate all movement (scratch for loop if im planning different movement for each leg)


    LF.moveDistance(LF.upperArray[0],LF.lowerArray[0],'L');
    LF.upperAngle = 180;
    LF.lowerAngle = 160;

    LH.moveDistance(LH.upperArray[0],LH.lowerArray[0],'L');

    RF.moveDistance(RF.upperArray[0],RF.lowerArray[0],'R');
    RF.upperAngle = 10;
    RF.lowerAngle = 30;


    RH.moveDistance(RH.upperArray[0],RH.lowerArray[0],'R');


    LH.driveLeg();
    LF.driveLeg();
    RH.driveLeg();
    RF.driveLeg();

    startLF = true;
    startRH = true;
    startRF = true;
  }
}

void standing(int x){
  while (millis() <= x) { //millis() <= 5000
    //calculate all movement (scratch for loop if im planning different movement for each leg)

    Serial.println("LH"); //gonna be outside of workspace
    LH.standingX = -60;
    LH.standingY = -480;
    Serial.println("0");
    LH.calculations(LH.standingX,LH.standingY,0);
    Serial.println("1");
    LH.calculations(LH.standingX,LH.standingY+upY,1); //makes equilateral triangle????
    Serial.println("2");
    LH.calculations(LH.standingX+upX,LH.standingY,2);
    Serial.println("3");
    LH.calculations(LH.standingX+downX-20,LH.standingY,3);
```

```
    Serial.println("RH");
    RH.standingX = -60;
    RH.standingY = -480;
    Serial.println("0");
    RH.calculations(RH.standingX,RH.standingY,0);
    Serial.println("1");
    RH.calculations(RH.standingX,RH.standingY+upY,1); //makes equilateral triangle????
    Serial.println("2");
    RH.calculations(RH.standingX+upX,RH.standingY,2);
    Serial.println("3");
    RH.calculations(RH.standingX+downX-20,RH.standingY,3);

    Serial.println("RF");
    RF.standingX = 0;
    RF.standingY = -460;
    Serial.println("0");
    RF.calculations(RF.standingX,RF.standingY,0);
    Serial.println("1");
    RF.calculations(RF.standingX,RF.standingY+upY,1); //makes equilateral triangle????
    Serial.println("2");
    RF.calculations(RF.standingX+25,RF.standingY,2);
    Serial.println("3");
    RF.calculations(RF.standingX-50,RF.standingY,3); //4/02: -50 for front legs

    Serial.println("LF");
    LF.standingX = 0;
    LF.standingY = -450;
    Serial.println("0");
    LF.calculations(LF.standingX,LF.standingY,0);
    Serial.println("1");
    LF.calculations(LF.standingX,LF.standingY+upY,1); //makes equilateral triangle????
    Serial.println("2");
    LF.calculations(LF.standingX+25,LF.standingY,2);
    Serial.println("3");
    LF.calculations(LF.standingX-50,LF.standingY,3);



    LF.moveDistance(LF.upperArray[0],LF.lowerArray[0],'L');

    LH.moveDistance(LH.upperArray[0]-20,LH.lowerArray[0]+50,'L');

    RF.moveDistance(RF.upperArray[0],RF.lowerArray[0],'R');

    RH.moveDistance(RH.upperArray[0],RH.lowerArray[0],'R');


    LH.driveLeg();
    LF.driveLeg();
    RH.driveLeg();
    RF.driveLeg();

    startLF = true;
    startRH = true;
    startRF = true;
  }
}

void potReading(){
  while(1){
//     double LHupper = LH.angleRead(LH.upperPotPin);
```

```
//     double LHupperCal = LHupper*LH.upperPotCalibration + LH.upperPotInt;
//     Serial.print("LH Upper Required Motor Value: ");
//     Serial.println(LHupperCal);
//     double LHlower = LH.angleRead(LH.lowerPotPin);
//     double LHlowerCal = LHlower*LH.lowerPotCalibration + LH.lowerPotInt;
//     Serial.print("LH Lower Required Motor Value: ");
//     Serial.println(LHlowerCal);
//     Serial.println("");

//     double LFupper = LF.angleRead(LF.upperPotPin);
//     double LFupperCal = LFupper*LF.upperPotCalibration + LF.upperPotInt;
//     Serial.print("LF Upper Required Motor Value: ");
//     Serial.println(LFupperCal);
//     double LFlower = LF.angleRead(LF.lowerPotPin);
//     double LFlowerCal = LFlower*LF.lowerPotCalibration + LF.lowerPotInt;
//     Serial.print("LF Lower Required Motor Value: ");
//     Serial.println(LFlowerCal);
//     Serial.println("");

     double RHupper = RH.angleRead(RH.upperPotPin);
     double RHupperCal = RHupper*RH.upperPotCalibration + RH.upperPotInt;
     Serial.print("RH Upper Required Motor Value: ");
     Serial.println(RHupperCal);
     double RHlower = RH.angleRead(RH.lowerPotPin);
     double RHlowerCal = RHlower*RH.lowerPotCalibration + RH.lowerPotInt;
     Serial.print("RH Lower Required Motor Value: ");
     Serial.println(RHlowerCal);
     Serial.println("");

//    double RFupper = RF.angleRead(RF.upperPotPin);
//    double RFupperCal = RFupper*RF.upperPotCalibration + RF.upperPotInt;
//    Serial.print("RF Upper Required Motor Value: ");
//    Serial.println(RFupperCal);
//    double RFlower = RF.angleRead(RF.lowerPotPin);
//    double RFlowerCal = RFlower*RF.lowerPotCalibration + RF.lowerPotInt;
//    Serial.print("RF Lower Required Motor Value: ");
//    Serial.println(RFlowerCal);
//    Serial.println("");



  }

}
```

Appendix B: MatLab Scripts

```
function [theta1,theta2,sym1,sym2] = mathLeg(lenU,lenL,lenX,lenY)
%use this to solve the system of equations to get the angle in radians for a
given length and position
    syms u;
    syms l;
    syms a;
    syms b;
    syms x;
    syms y;

    eqn1 = b*cos(u + l) + a*cos(u) == x;
    eqn2 = b*sin(u + l) + a*sin(u) == y;
    sol = solve([eqn1,eqn2],[u,l]);
    sym1 = sol.u;
    sym2 = sol.l;
    theta1 = subs(sol.u,[a,b,x,y],[lenU,lenL,lenX,lenY]);
    theta2 = subs(sol.l,[a,b,x,y],[lenU,lenL,lenX,lenY]);


function y = workspaceCalc(a,b,xMin,xMax,yMin,yMax,xAmp,zAmp,zPos,xPos,label)
%use to graph a workspace using min max radians and graphs a spline using
cosine and sin
    %x is the theta2
    %y is the theta1

    yRange = (yMax-yMin)/10;
    xRange = (xMax-xMin)/10;
    t1 = yMin:yRange:yMax;
    t2 = xMin:xRange:xMax;
    t3 = [xMin,xMin,xMin,xMin,xMin,xMin,xMin,xMin,xMin,xMin,xMin];
    t4 = [yMin,yMin,yMin,yMin,yMin,yMin,yMin,yMin,yMin,yMin,yMin];
    t5 = [xMax,xMax,xMax,xMax,xMax,xMax,xMax,xMax,xMax,xMax,xMax];
    t6 = [yMax,yMax,yMax,yMax,yMax,yMax,yMax,yMax,yMax,yMax,yMax];

    %vary both
    M1 = b*cos(t1+t2) + a*cos(t1);
    M2 = b*sin(t1+t2) + a*sin(t1);
    %vary upper with lower min
    N1 = b*cos(t1+t3) + a*cos(t1);
    N2 = b*sin(t1+t3) + a*sin(t1);
    %vary upper with lower max
    J1 = b*cos(t1+t5) + a*cos(t1);
    J2 = b*sin(t1+t5) + a*sin(t1);
    %vary lower with upper min
    G1 = b*cos(t4+t2) + a*cos(t4);
    G2 = b*sin(t4+t2) + a*sin(t4);
    %vary lower with upper max
    P1 = b*cos(t6+t2) + a*cos(t6);
    P2 = b*sin(t6+t2) + a*sin(t6);

    x = xAmp*cos(0:0.1:2*pi) + xPos;
    z = zAmp*sin(0:0.1:2*pi) + zPos;
    plot(M1,M2,'-o',N1,N2,'-o',J1,J2,'-o',G1,G2,'-o',P1,P2,'-o',x,z);
    title(label);
```

```
axis([-250 200 -520 -370]);
xlabel('x (mm)');
ylabel('y (mm)');
y = true;
```

Appendix C: Pot Calibration Code

```
Pot Calibration
//ServoTest.ino


#include <Servo.h>

Servo lower;
Servo upper;
Servo upper2;
Servo lower2;

#define potPinUpper A8
#define potPinLower A11

double potCalUpper = 0;
double potCalLower = 0;
double potIntUpper = 0;
double potIntLower = 0;

double upperAngle = 10;
double lowerAngle = 160;

void setup() {
  Serial.begin(9600);
  // put your setup code here, to run once:
 // myServo.attach(8);
  upper.attach(41);
//  upper2.attach(24);
  lower.attach(40);
//  lower2.attach(25);
}

void loop() {
   upper.write(upperAngle);
   lower.write(lowerAngle);
   delay(3000);
   double upperPot1 = angleRead(potPinUpper);
   double lowerPot1 = angleRead(potPinLower);
   double upperAngle2 = 180;
   double lowerAngle2 = 10;
   upper.write(upperAngle2);
   lower.write(lowerAngle2);
   delay(3000);
   double upperPot2 = angleRead(potPinUpper);
   double lowerPot2 = angleRead(potPinLower);
   double y1 = (upperAngle - upperAngle2);
   double y2 = (lowerAngle - lowerAngle2);
   double x1 = (upperPot1 - upperPot2);
   double x2 = (lowerPot1 - lowerPot2);
   potCalUpper = (y1/x1);
   potCalLower = (y2/x2);

   potIntUpper = (upperAngle - (potCalUpper*upperPot1));
   potIntLower = (lowerAngle - (potCalLower*lowerPot1));

   Serial.print("Pre-calibrated Upper Value: ");
   Serial.println(upperPot1);
   Serial.print("Calibrated Upper Value: ");
   double calibratedU = ((upperPot1*potCalUpper)+potIntUpper);
```

```
    Serial.println(calibratedU);
    Serial.print("Motor Upper Value: ");
    Serial.println(upperAngle);
    Serial.print("Upper Calibration Value: ");
    Serial.println(potCalUpper);
    Serial.print("Upper Intercept Value: ");
    Serial.println(potIntUpper);

    Serial.println("");

    Serial.print("Pre-calibrated Lower Value: ");
    Serial.println(lowerPot1);
    Serial.print("Calibrated Lower Value: ");
    double calibratedL = ((lowerPot1*potCalLower)+potIntLower);
    Serial.println(calibratedL);
    Serial.print("Motor Lower Value: ");
    Serial.println(lowerAngle);
    Serial.print("Lower Calibration Value: ");
    Serial.println(potCalLower);
    Serial.print("Lower Intercept Value: ");
    Serial.println(potIntLower);

    Serial.println("");


//    upper2.write(180);
//    lower2.write(10);

//  upper2.write(180);
//  lower2.write(120);

//  delay(100);
//  upper.write(180);
//  delay(100);
//    for(int i = 0; i<180; i++){
//      upper.write(i);
//      lower.write(abs(i-180));
//      delay(10);
//    }
//
//
//    for(int i = 180; i>0; i--){
//      upper.write(i);
//      lower.write(abs(i-180));
//      delay(10);
//    }
//

}

double angleRead(int potPin){
    double val = analogRead(potPin);
    Serial.print("Analog Read: ");
    Serial.println(val);
    double mappedVal = map(val,0,1023,0,270);
    return (mappedVal);//*potCalibration + potInt);
  }
```

Appendix D: Bill of Materials

Item	Producer/Website	Item #
#25 Roller Chain	McMaster-Carr	6261K171
9 Tooth Finished-Bore Sprocket for ANSI Roller Chain	McMaster-Carr	2737T1
1/2in Bore Size 28 Tooth Finished-Bore Sprocket for ANSI Roller Chain	McMaster-Carr	2737T21
1/2in Bore Size 45 Tooth Finished-Bore Sprocket for ANSI Roller Chain	McMaster-Carr	2737T31
Aluminum Mounting Hub for 1/4in shaft	pololu.com	#1993
0.500in Bore Size Dual Pinch Bolt, Tapped Clamping Hubs	ServoCity.com	545340
10k Potentiometers	digikey	
Set Screw Shaft Collars	McMaster-Carr	9414T11
Buck DC Converter	Amazon	
Machinable Bore Sprocket #25 10 Tooth Nylon	McMaster-Carr	60425K13
Machinable Bore Sprocket #25 45 Tooth Nylon	McMaster-Carr	60425K34
Machinable Bore Sprocket #25 40 Tooth Nylon	McMaster-Carr	60425K33
Multipurpose 6061 Aluminum, Rectangular Bar	McMaster-Carr	

Standard Hub Horns	**ServoCity.com**	**525132**
Aluminum Mounting Hub for 1/4in shaft	**pololu.com**	**#1993**
#25 Roller Chain	**McMaster-Carr**	**6261K171**
9 Tooth Finished-Bore Sprocket for ANSI Roller Chain	**McMaster-Carr**	**2737T1**
Buck DC Converter	**Amazon**	