Doggo

A Quadrupedal Arduino Robot

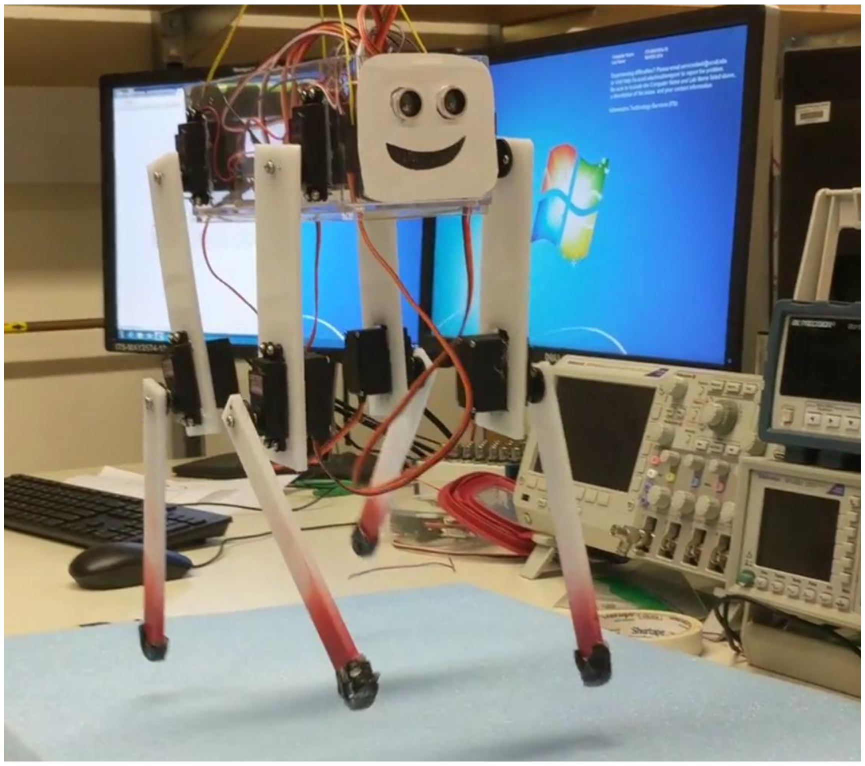
By Corlin Palmer

Motivation and Concept

While not exactly my initial project concept, the goal of the final robot design was to create an unconventional walking robot – here one that has long legs and is inherently unstable. This posed a challenge over other walking robot designs with proportionally smaller legs and lower centers of gravity.

Functional Definition

Doggo consists of a rectangular body housing an Arduino microcontroller, an ultrasonic distance sensor, a battery pack, a voltage regulator, and four servos, each attached to a 'thigh' that holds another servo attaching the 'shin' part of the leg. The final robot is pictured below:



Doggo's function is to follow a pre-programmed gait to achieve forward locomotion. The Ultrasonic Distance Sensor that makes up his eyes is used to detect and avoid obstacles by turning, or to order Doggo to stop moving when it comes too close to an obstacle.

Mechanical Considerations

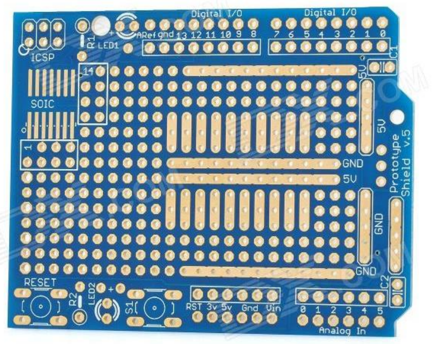
The mechanics of assembling the robot were relatively straight forward. I cut four thigh pieces and four shin pieces from the plastic sheets using a table saw. In the shin pieces, I rounded the ends using a bench grinder and applied insulating putty for extra traction. In the thigh pieces, I cut a rectangular hole for a servo and four holes to screw it on. I did the same to cut holes for the servos in the chassis.

This design has very long legs and considerable instability in the joints due to play in the servo horns and flexibility of the plastic. I decided not to change anything in the design, however, and instead decided to proceed ahead with making the lanky unstable robot walk. I did add hot glue to every piece to improve rigidity, but overall the frame held up well and did not give me any issues nor require any fundamental changes.

Electrical Considerations

Powering the robot had some interesting considerations: The servos require 6v to run their fastest, and very high current (> 1.5A each under load), while the Arduino needs 7v - 16v on its V_{in} pin to run. I ended up using two 4v batteries (described in the parts section) for 8v of power, running directly into the Arduino and also through a voltage regulator (pictured below) transforming the power to 6v for the servos. The voltage regulator was very convenient to have, as it allowed me to monitor the charge on the batteries and supplied steady voltage to the servos – a dropping voltage from connecting batteries directly would cause the servos to change behavior over time. Finally, I added an on/off switch to the positive wire of the battery pack. I estimate the bot could walk for around 30 minutes before needing recharging.

Wiring 8 servos, with three pins each, to one Arduino, was a challenge. Luckily, I was able to take advantage of the channels on the Arduino prototype board, pictured below. I soldered 8 connectors spanning the 5v, GND, and another row of three, attaching one servo to each connector. I cut small jumper wires to connect those directly to pins 4, 5, 6, 7, 8, 10, 11, and 12. I removed the pin that attached the 5v line of the prototype board to the Arduino and instead ran the power directly to the 6v output of the voltage regulator. Another small row on the board was used to power the 5v needed for the ping sensor.



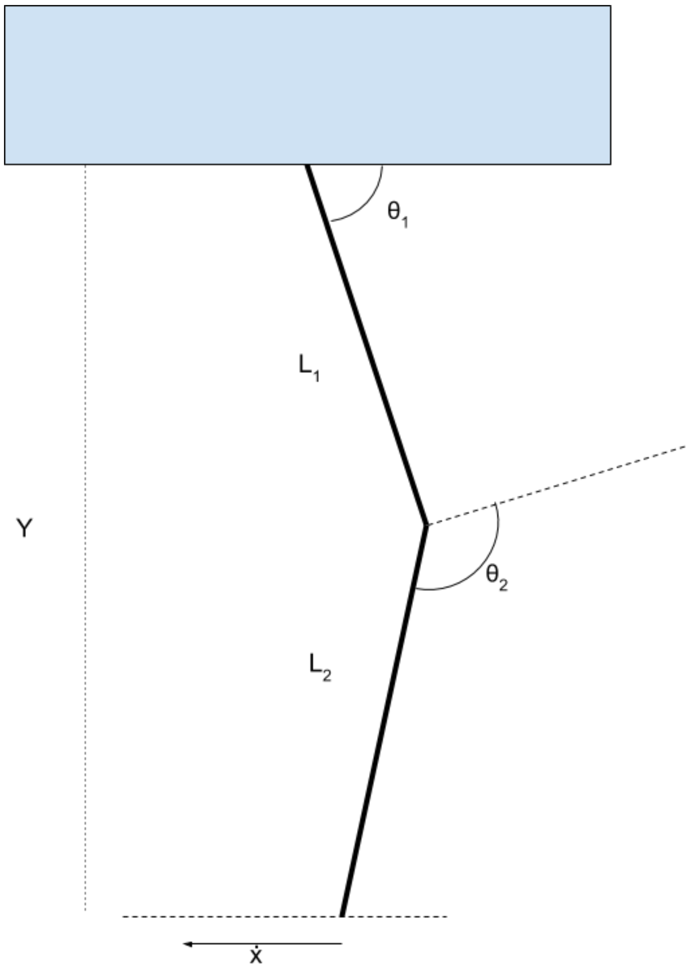
Walking Math

The movement of the Arduino had to follow two basic rules to realize the best possible walking pattern: The height of the foot had to remain constant (when not lifted), and the speed on the foot along the floor had to remain constant, to avoid fighting the momentum of the bot. The math for this operation is described below.

The height of both the legs together is:

$$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2 - 90°)$$

Where L_1 is the length of the thigh and L_2 is the length of the leg, with θ_1 and θ_2 being the angles (centered at 90°) of the hip and knee servos, respectively.



So, to angle the knee such that Y is always constant:

$$\theta_2 = \sin^{-1}\left(\frac{Y}{L_2} + \frac{L_1 \sin(\theta_1)}{L_2}\right) - \theta_1 + 90°$$

The x position of the foot is:

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2 - 90°)$$

In terms of just the thigh rotation this is:

$$x = L_1 \cos(\theta_1) + \sqrt{1 - \left(\frac{Y}{L_2} + \frac{L_1 \sin(\theta_1)}{L_2}\right)^2}$$

For movement, we can consider the thigh rotation to be a function of time

$\theta_1 = \theta(t) = \theta, \quad \frac{d}{dt}\theta_1 = \dot{\theta}$

Solving this out, the speed at which the robot moves in terms of the rotation of the leg is:

$$\dot{\mathrm{X}} = -\mathrm{L}_1\, \dot{\theta} \sin(\theta) + \frac{L_1\, \dot{\theta}\, \cos(\theta) \left(\frac{Y}{L_2} + \frac{L_1 \sin(\theta)}{L_2}\right)}{L_2 \sqrt{1 - \left(\frac{Y}{L_2} + \frac{L_1 \sin(\theta)}{L_2}\right)^2}}$$

This needs to be solved under the boundary conditions that $\theta(0) = \theta(pi)$ and θ and $\dot{\theta}$ remain within reasonable values.

With the help of Mathematica (and Clayton Anderson), we approximated the solution, however upon plugging it into the simulation the movement was not reasonable. Instead I opted to use numerical optimization to find the best fit based on some rating factors in a simulation, including how constant the speed of the foot is and how fast the servo turned. This is described in the next section.

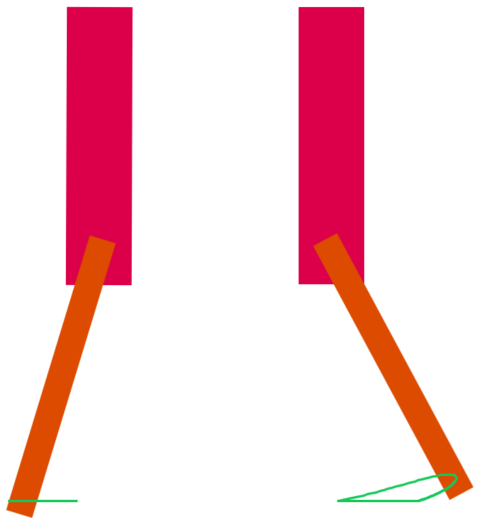
Software

The first software I wrote for the project was a Genetic Optimizer that can optimize the values in an array of length N by the following steps:

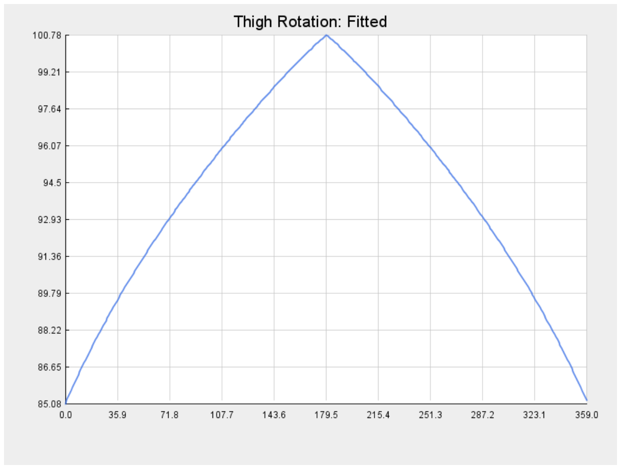
- Create an 'initial population' of N-length arrays containing random values
- Test the fitness / error metric for each member of the population
- Randomly choose two arrays from among the ones with best fitness
- Average the values of the two arrays and add a small random number to each value
- Test the fitness of the child array, and if it's an improvement on the worst member of the population, insert the child array into the population by comparing itself each existing member:
    - If its values are close to those of another member, replace it
    - If it is unique, remove the last member and insert where appropriate
- Repeat steps 3-5 until the best member of the population reaches a desired fitness or a maximum number of iterations have been made.

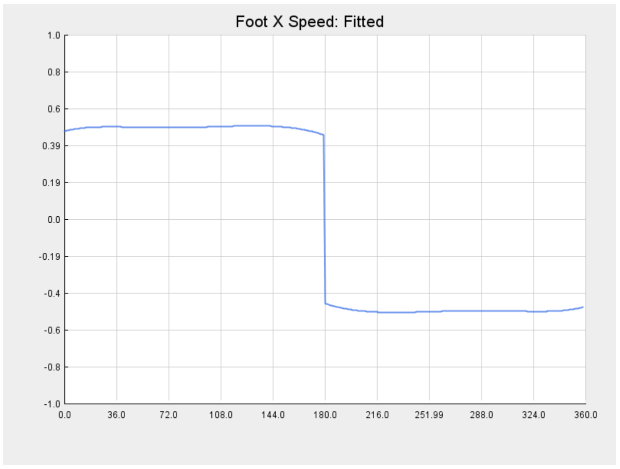
I have written three numerical genetic optimizers now, and this approach ended up being very effective and memory-efficient (only 16 – 128 arrays are stored). Comparing the values of each candidate array to the existing population allowed the program to keep a diverse population and avoid local minima very effectively.

Now while it was originally written for the Arduino to optimize the robot's gait, I eventually ported it to Java to do that in a virtual environment, allowing me to run a million iterations to find the best walking gait. This was done by writing a simulation of the robot's leg movement using Java's Graphics2D and Math classes and using a rating of the gait as the fitness function. An example screenshot of my simulation is seen below, where the green traces the movement pattern of each leg. The simulation allowed me to judge the walking gait and whether there would be any conflicts between the legs before applying it to the robot. Using some syntax tricks I was able to write the Java code so that it could be directly copy-pasted onto the Arduino and vice-versa.



The function defining the servo movement simply takes a value between 0 and 360 as input and returns a thigh servo position value between 0 and 180 (the knee servo position is predefined as a function of the thigh position). The genetic numerical optimization was applied to this function to fit an array of 360 points, and later turned into an approximating polynomial. In the graphs below, you can see the optimized thigh rotation as a function of the input value, and a plot of the movement speed of the foot with that function applied. As you can see, it is quite linear.





Once the final function had been defined, it was copied to the Arduino and used to make continuous-time servo operation using the Arduino's millis() function. The loop makes repeated calls to a moveServos() function that sends the corresponding new position to the servos and judge the distance that each servo had to move, waiting for an amount of time proportional to how far the servo had to move. That gave the smoothest possible walking pattern.

Finally, on every iteration of the loop the ping sensor would check the current distance, apply exponential smoothing to ensure quality data, then perform the corresponding action depending on its distance: turn if approaching an obstacle, stop if too close.

Testing

Primarily, the robot's gait was tested using the simulation software. For testing the mechanical parts of the robot, it was suspended from a pipe on the ceiling, keeping the legs off the ground so it would not fall over or walk away. I used the 'harness' attached to Doggo to affix a leash and walk him with that. It both added authenticity to the robot as a dog and allowed me to pick it up easily or prevent it from falling. It only fell over (uncaught) once during testing, luckily no damage was sustained aside from a battery falling out.

Parts and Reusability

Doggo is built of an 8” x 3.5” x 4” chassis from the container store, housing:

- Elegoo Arduino w/ Prototype Expansion Board
- Battery pack w/ two 18650 rechargeable batteries at 4V and 3600mah each
- Yeeco DC Voltage Regulator
- Cardstock Paper Smiley Face
- Scrap plastic sheet from ePlastics in Kearney Mesa
- Ultrasonic Distance “ping” Sensor
- 8 Longruner MG996R Servos

As I bought all the parts on Amazon with my own money, I was able to bring Doggo home safe and sound.

Conclusion

Doggo was an interesting and challenging project full of twists and surprises. A few hours into the initial testing, I wasn’t sure if it was ever going to be able to walk. But in the end, Doggo walked, and he walked pretty well. He has stumbled his way into my heart, and I hope to keep him around forever.

Finally, I’d like to thank all the professors and TAs of this class for their help and dedication to the students’ success. It was a great quarter!