Cornell University

Collective Embodied Intelligence Lab

Arachnabot
Spider Inspired Robot

Kirstin Petersen
William Abajian
Lawrence Chen
Daria Efimov
Christopher Fedors
Eugene Ng

Contents

List of Figures

List of Tables

Nomenclature

D_O	= Outer Diameter of Leg, m
D_I	= Inner Diameter of Leg, m
r_O	= Outer Radius of Leg, m
r_I	= Inner Radius of Leg, m
F	= Upper Joint (Femur) Length, m
T	= Lower Joint (Tibia) Length, m
f	= Distance from Base of Femur to Center of Mass of Femur, m
t	= Distance from Base of Tibia to Center of Mass of Tibia, m
m_b	= Mass of Spider Body, kg
m_F	= Mass of Spider Femur, kg
m_T	= Mass of Spider Tibia, kg
h	= Jump Height, m
θ	= Bend Angle (with Respect to the Horizontal), rad
ϕ	= Joint Angle (with Respect to the Vertical), rad
I_T	= Moment of Inertia of Tibia around its Center of Mass, kg-m^2
I_F	= Moment of Inertia of Femur around its Center of Mass, kg-m^2
p	= Pressure, Pa
$\vec{r}$	= Position Vector, m
$\dot{\vec{r}}$	= Velocity Vector, m/s
$\ddot{\vec{r}}$	= Acceleration Vector, m/s^2
$\vec{F}$	= Force Vector, N

1 Abstract

The jumping spider family is the largest among its species, making up about 13% of all spiders. Jumping as a method of locomotion is advantageous in several ways, especially in rough terrain where rolling or walking might be difficult. Jumping spiders can jump many times their body length, allowing them to pounce on prey or traverse great distances relative to their size. They do so by contracting muscles in their upper body (cephalothorax), decreasing the volume of blood there, and forcing it into the legs, causing an increase in pressure in the extremities. This pressure increase allows for rapid extension of the spiders' legs, causing it to catapult into the air. The primary focus of Arachnabot is to mimic this sequence of events, so that we may gain a better understanding of the biomechanics involved, and apply this understanding to make better and more versatile robots in the future.

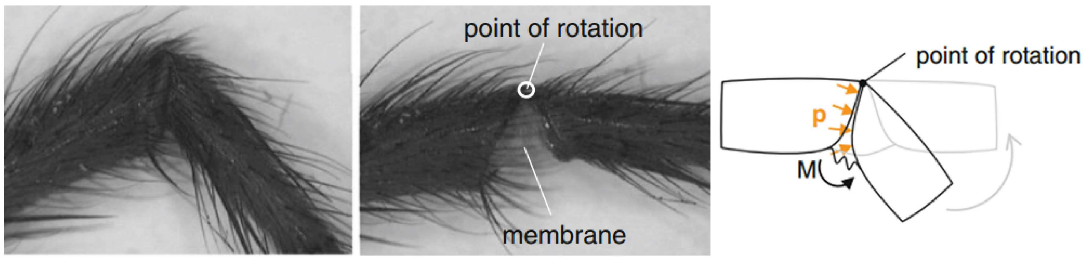


Figure 1: A free body diagram of a spider joint (right), next to a typical joint of the spider leg in the flexed (left) and extended (middle) positions. Adapted from Zentner [8]

2 Executive Summary

We present preliminary analysis, design, and testing on a single, small, robotic spider leg. Simulations of the system were first done in MATLAB, and then iterative design was done with Solidworks. We present the results of tests on several different parts of the spider leg, including the latching system and the leg contraction system. Fabrication of spider-scale parts proved difficult due to manufacturing and material constraints, but we believe a viable solution has been found. The leg is meant to imitate an actual spider's leg, which is able to help the spider jump around by atypical means that could provide insight and inspiration for a new method of locomotion in robotics.

2.1 Primary Objective

Our primary goal is to recreate the method by which a spider jumps in a reasonable and economical fashion. We wish to achieve this goal at the same scale - that of a spider - though different means and designs may be necessary to do so. The envisioned final product is, quite simply, a robotic, jumping spider.

2.2 Secondary Objectives

In addition to our primary goal, we have a number of more specific secondary goals that we would like to obtain, many of which will help us reach our primary goal.

- Design a leg no more than $5cm$ tall at full extension.
- Reach a maximum jump height of double the leg height at full extension.
- Design a body that weighs less than $30g$.
- Create a self-supporting, self-contained system that does not need an external pressure or control source.
- Use designs and methods that are as simple as possible.
- Prototype and fabricate as quickly as possible, and worry about efficiency and optimization later.
- Be able to learn and take away knowledge and new experiences from the project.

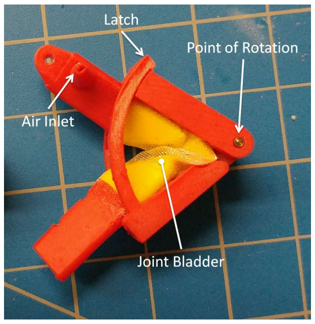


Figure 2: The prototype created using balloons and ribbon mesh with latching functionality.

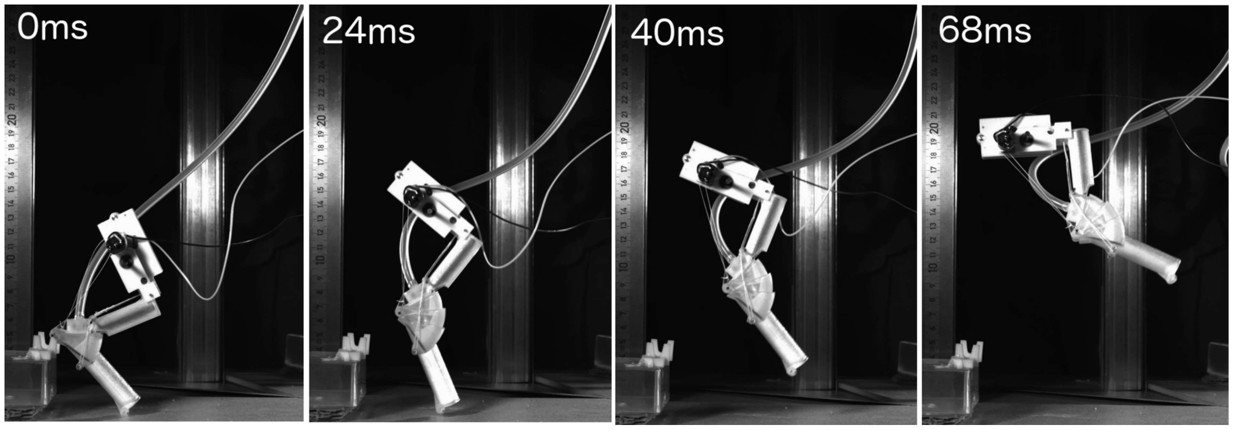


Figure 3: The work presented here is an extension of previous work (shown above) done at the Max Planck Institute for Intelligent Systems, where a larger-scale leg was demonstrated using a cable tendon, mechanical tensioning mechanisms, a rigid segmented joint, remote air supply, and remote power supply. This leg is capable of jumping its own height as configured. [7]

3 Background

A spider's leg functions very differently from other muscle-powered legs, and is even unique among other insect species (insects in general require higher limb accelerations that cannot be reached by muscular contraction). A grasshopper is one insect which has strong leg muscles for its size. However, fleas and locusts, for example, jump via an elastic instability, storing potential energy in a spring-like system which releases after a certain point is reached [2]. The spider uses similar snap and latch mechanisms, but its method of energy storage is one-of-a-kind. Spiders use a very effective hydraulic system to store energy in its hind legs and jump. Because of the insects' open blood circulatory system, a spiders *lacunae*, the space between its muscles and its exoskeleton, is filled with *hemolymph* [5], it's "blood." A jumping spider is able to squeeze muscles in its upper body (it's *cephalothorax*) to drive hemolymph to its legs, effectively increasing the pressure in its spidery veins. Once this pressure buildup is high enough, it can snap the legs, causing them to rapidly extend and launch the spider into the air.

Most spiders are able to jump farther horizontally than they are vertically. Because spiders lack mid-air control, spiders first put a strand of web down before jumping. This helps them control their orientation in the air and stabilize them so that they can land upright. It can also act as a fail-safe in case they miss their landing targets. Typically a spider will use its jump as a way to pounce on prey quickly, traverse a gap, or flee. Jumping as a method of locomotion in robots is advantageous in other ways, especially when the robot is small and everything is an obstacle: it is less energy-intensive than flying, yet more maneuverable and faster than walking or rolling.

Jumping robots are certainly not new to the scientific community. A lightweight jumping robot developed by Sandia National Labs uses a chemical combustion process to slam a piston-cylinder setup into the ground, resulting in a jump height of over 30 feet vertically [3]. The wheeled robot is capable of righting itself after the jump. A smaller robot developed by Ecole Polytechnique Federale de Lausanne (EFPL) mimics the jump of a flea and is also capable of self righting [4]. As for our spider, preliminary work was first done by the Max Planck Institute for Intelligent Systems, albeit with a larger leg and a remote air supply [7]. We hope to recreate the spider leg on a smaller scale with an on-board air supply.

4 Methods and Analysis

To better understand the system before designing, we first created a model of the spider leg to analyze. We make some preliminary assumptions that will help simplify the model.

- *2D Cartesian Coordinate System*
  Movement of the leg will be constrained to the x-y (horizontal and vertical) directions, assuming that none of the force or pressure acting on the system will be acting in the z-direction (out-of-plane).
- *Constant Pressure*
  The pressure acting on the joint remains constant throughout the motion of the joint (that is, $p = p_0$. This is a best-case scenario, as the pressure acting on the joint is likely to drop as the angle θ increases. A simple relation for a linear pressure drop, for example, is $p = p_0(1 - \frac{\theta}{\theta_0})$, where the pressure is dependent on the

angle θ. In actuality, the pressure drop may be even faster than a linear drop, as the rapidity with which the pressure force has to be applied and the overall size of a spider means we are considering a very small timespan relative to most other pneumatic systems.

In addition, we define the constant pressure to be an external force on the system, rather than an internal force like the forces acting on the hinge. We also define the mass of the body to be an external force.

- *Frictionless System*
  No energy is lost to friction or heat. All of the potential energy stored in the joint by pressurization is converted into kinetic energy for the jump. Once the jump is complete (robot is off the ground), the kinetic energy is converted back to potential energy in the form of the system's maximum height.
- *Point Loads*
  All force act as point loads. For example, the pressure force acts as a point load on the centerline of the leg, rather than a distributed load on the entire leg cross section.

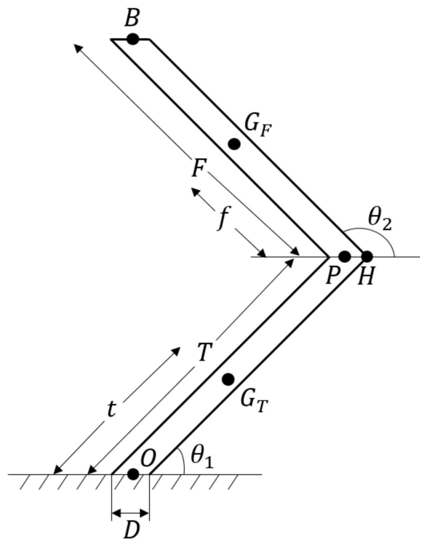


Figure 4: Problem setup, key locations and distances. (Not to scale)

4.1 Pressure Calculations

We begin with an angular momentum balance on the tibia, or lower leg (Figure 5). We approximate the pressure as a single point force acting at the center of the tibia ($r = 0$), parallel to the longitudinal axis of the leg. The tibia rotates around the point H.

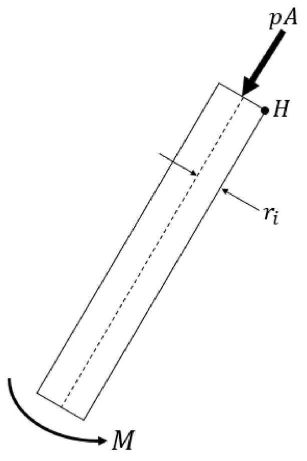


Figure 5: Moment Balance on the Tibia

The moment is given by:

$$\begin{aligned} M &= r_i \times F \\ &= pAr_i \\ &= p_0 \pi r_i^3 \end{aligned} \tag{1}$$

In order to calculate a jump height, we will use Conservation of Energy on the entire system by writing the work done on the leg in two different ways.

$$W = mgh \tag{2}$$

$$W = \int_0^{\theta_0} M(\theta) d\theta \tag{3}$$

We will equate the work done by the moment (Eq. 3) to the potential energy of the system at the height of its jump (Eq. 2).

$$\begin{aligned} W &= \int_0^{\theta_0} p_0 \pi r_i^3 = mgh \\ &= p_0 \theta_0 \pi r_i^3 = mgh \end{aligned}$$

This gives us a simple relation for the pressure required to reach the specified height.

$$p_0 = \frac{mgh}{\theta_0 \pi r_i^3} \tag{4}$$

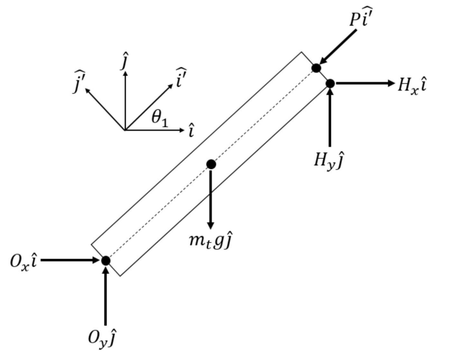
Then the force required on the joint to propel the spider into the air is

$$\begin{aligned} F &= pA \\ &= p_0 \pi r_i^2 \end{aligned} \tag{5}$$

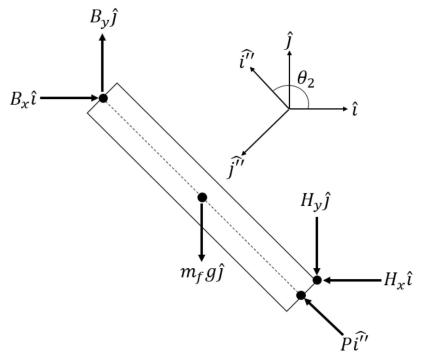
4.2 System Mechanics: Kinematics and Dynamics

The dynamic motion was simulated in MATLAB, using a Differential Algebraic Equaton (DAE) method, resulting in 10 equations which track the motion of the center of masses of the bodies, the forces applied to the base of the tibia, and the forces acting on the joint. Relevant locations and distances are shown in Figure 4.

There are two phases that we are concerned with: Phase 1 is concerned with the motion of the system when it is still in contact with the ground, and Phase 2 is concerned with the motion of the system when it has in air. Free body diagrams on each of the two legs for Phase 1 are shown in Figure 6. The free body diagrams are the same for Phase 2; however, the forces at O and the pressure forces disappear.



(a) Tibia



(b) Femur

Figure 6: Free Body Diagrams and Associated Vectors for Differential Algebraic Equations

Note that we define all angles θ_1 and θ_2 with respect to the horizontal, so that they can be written as:

$$
\begin{aligned}
\hat{\imath} &= \text{the x-direction in the non-rotating inertial frame} \\
\hat{\jmath} &= \text{the y-direction in the non-rotating inertial frame} \\
\hat{\imath}' &= cos(\theta_1)\hat{\imath} + sin(\theta_1)\hat{\jmath} \\
\hat{\jmath}' &= -sin(\theta_1)\hat{\imath} + cos(\theta_1)\hat{\jmath} \\
\hat{\imath}'' &= cos(\theta_2)\hat{\imath} + sin(\theta_2)\hat{\jmath} \\
\hat{\jmath}'' &= -sin(\theta_2)\hat{\imath} + cos(\theta_2)\hat{\jmath}
\end{aligned}
\tag{6}
$$

We are interested in 10 unknown quantities revealed by the free-body diagrams: $O_x, O_y, H_x, H_y, \ddot{x}_T, \ddot{y}_T, \ddot{x}_F, \ddot{y}_F, \ddot{\theta}_1$, and $\ddot{\theta}_2$. We can write 10 equations from linear momentum balance, angular momentum balance, and certain constraints that will allow us to simultaneously solve the system of equations.

The following distance vectors must be defined. Note that the quantity d in this case refers to the radius of the legs.

$$
\begin{aligned}
\vec{r}_{O/GT} &= -t\hat{\imath}' \\
\vec{r}_{P/GT} &= t\hat{\imath}' \\
\vec{r}_{H/GT} &= t\hat{\imath}' - d\hat{\jmath}' \\
\vec{r}_{P/GF} &= -f\hat{\imath}'' \\
\vec{r}_{H/GF} &= -f\hat{\imath}'' - d\hat{\jmath}'' \\
\vec{r}_{B/GF} &= f\hat{\imath}''
\end{aligned}
\tag{7}
$$

4.2.1 Linear Momentum Balances

A linear momentum balance is written for the tibia, resulting in two equations (one in the x-direction and one in the y-direction):

$$
\begin{gathered}
\vec{F} = m\vec{a}_{G_T} \\
O_x\hat{\imath} + O_y\hat{\jmath} - m_T g\hat{\jmath} - P\hat{\imath}' + H_x\hat{\imath} + H_y\hat{\jmath} = m_T\ddot{x}_T\hat{\imath} + m_T\ddot{y}_T\hat{\jmath}
\end{gathered}
\tag{8}
$$

A linear momentum balance is written for the femur, resulting in two equations (one in the x-direction and one in the y-direction):

$$
\begin{gathered}
\vec{F} = m\vec{a}_{G_F} \\
-m_F g\hat{\jmath} + P\hat{\imath}'' - H_x\hat{\imath} - H_y\hat{\jmath} = m_F\ddot{x}_F\hat{\imath} + m_F\ddot{y}_F\hat{\jmath}
\end{gathered}
\tag{9}
$$

4.2.2 Angular Momentum Balances

An angular momentum balance is written for the tibia around its center of mass:

$$
\begin{aligned}
\sum \vec{M}_{/GT} &= \sum \dot{\vec{H}}_{/GT} \\
[\vec{r}_{O/GT} \times (O_x\hat{\imath} + O_y\hat{\jmath})] + [\vec{r}_{P/GT} \times (-P\hat{\imath}')] + [\vec{r}_{H/GT} \times (H_x\hat{\imath} + H_y\hat{\jmath})] &= I_T\ddot{\theta}_1\hat{k}
\end{aligned}
\tag{10}
$$

An angular momentum balance is written for the femur around its center of mass:

$$
\begin{aligned}
\sum \vec{M}_{/GF} &= \sum \dot{\vec{H}}_{/GF} \\
[\vec{r}_{B/GF} \times (-m_b g\hat{\jmath})] + [\vec{r}_{P/GF} \times (-P\hat{\imath}'')] + [\vec{r}_{H/GF} \times (-H_x\hat{\imath} - H_y\hat{\jmath})] &= I_F\ddot{\theta}_2\hat{k}
\end{aligned}
\tag{11}
$$

4.2.3 Constraints

The first constraint defines the motion of point O - namely, that it is 0. This results in two equations (one in the x-direction and one in the y-direction).

$$\begin{aligned}\vec{r}_{O/\mathcal{F}} &= \vec{0}\\ x_{GT}\hat{\imath} + y_{GT}\hat{\jmath} + \vec{r}_{O/GT} &= \vec{0}\\ x_{GT}\hat{\imath} + y_{GT}\hat{\jmath} - t\hat{\imath}' &= \vec{0}\\ \dot{x}_{GT}\hat{\imath} + \dot{y}_{GT}\hat{\jmath} &= t\dot{\theta}_1\hat{\jmath}'\\ \ddot{x}_{GT}\hat{\imath} + \ddot{y}_{GT}\hat{\jmath} &= t\ddot{\theta}_1\hat{\jmath}' - t\dot{\theta}_1^2\hat{\imath}'\end{aligned} \tag{12}$$

The second constraint defines the motion of the hinge with respect to each of the tibia and femur, namely, that it is the same motion. This results in two equations (one in the x-direction and one in the y-direction).

$$\begin{aligned}\vec{r}_{H/O} &= \vec{r}_{H/O}\\ x_{GT}\hat{\imath} + y_{GT}\hat{\jmath} + \vec{r}_{H/GT} &= x_{GF}\hat{\imath} + y_{GF}\hat{\jmath} + \vec{r}_{H/GF}\\ x_{GT}\hat{\imath} + y_{GT}\hat{\jmath} + t\hat{\imath}' - d\hat{\jmath}' &= x_{GF}\hat{\imath} + y_{GF}\hat{\jmath} - f\hat{\imath}'' - d\hat{\jmath}''\\ \dot{x}_{GT}\hat{\imath} + \dot{y}_{GT}\hat{\jmath} + t\dot{\theta}_1\hat{\jmath}' + d\dot{\theta}_1\hat{\imath}' &= \dot{x}_{GF}\hat{\imath} + \dot{y}_{GF}\hat{\jmath} - f\dot{\theta}_2\hat{\jmath}'' + d\dot{\theta}_2\hat{\imath}''\\ \ddot{x}_{GT}\hat{\imath} + \ddot{y}_{GT}\hat{\jmath} + t\ddot{\theta}_1\hat{\jmath}' - t\dot{\theta}_1^2\hat{\imath}' + d\ddot{\theta}_1\hat{\imath}' + d\dot{\theta}_1^2\hat{\jmath}' &= \ddot{x}_{GF}\hat{\imath} + \ddot{y}_{GF}\hat{\jmath} - f\ddot{\theta}_2\hat{\jmath}'' + f\dot{\theta}_2^2\hat{\imath}'' + d\ddot{\theta}_2\hat{\imath}'' + d\dot{\theta}_2^2\hat{\jmath}''\end{aligned} \tag{13}$$

4.2.4 Equations of Motion

We are left with a system of linear equations, [A]x = [b], describing the motion of the spider. For Phase 1, the system is as follows:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & -m_t & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 1 & 0 & -m_t & 0 & 0 & 0 & 0\\ 0 & 0 & -1 & 0 & 0 & 0 & -m_f & 0 & 0 & 0\\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -m_f & 0 & 0\\ tsin(\theta_1) & -tcos(\theta_1) & dcos(\theta_1) - tsin(\theta_1) & dsin(\theta_1) + tcos(\theta_1) & 0 & 0 & 0 & 0 & -I_T & 0\\ 0 & 0 & -dcos(\theta_2) - fsin(\theta_2) & fcos(\theta_2) - dsin(\theta_2) & 0 & 0 & 0 & 0 & 0 & -I_F\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & tsin(\theta_1) & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -tcos(\theta_1) & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & dcos(\theta_1) - tsin(\theta_1) & -dcos(\theta_2) - fsin(\theta_2)\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & dsin(\theta_1) + tcos(\theta_1 & fcos(\theta_2) - dsin(\theta_2)\end{bmatrix} \begin{bmatrix} O_x\\ O_y\\ H_x\\ H_y\\ \ddot{x}_{GT}\\ \ddot{y}_{GT}\\ \ddot{x}_{GF}\\ \ddot{y}_{GF}\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}$$

$$= \begin{bmatrix} Pcos(\theta_1)\\ m_tg - Psin(\theta_1)\\ -Pcos(\theta_2)\\ m_Bg + m_Fg - Psin(\theta_2)\\ 0\\ m_Bgfcos(\theta_2)\\ -\dot{\theta}_1^2tcos(\theta_1)\\ -\dot{\theta}_1^2tsin(\theta_1)\\ f\dot{\theta}_2^2cos(\theta_2) + d\dot{\theta}_1^2sin(\theta_1) - d\dot{\theta}_2^2sin(\theta_2) + t\dot{\theta}_1^2cos(\theta_1)\\ d\dot{\theta}_2^2cos(\theta_2) - d\dot{\theta}_1^2cos(\theta_1) + f\dot{\theta}_2^2sin(\theta_2) + t\dot{\theta}_1^2sin(\theta_1)\end{bmatrix}$$

For Phase 2, the system is as follows:

$$\begin{bmatrix} 1 & 0 & -m_t & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & -m_t & 0 & 0 & 0 & 0\\ -1 & 0 & 0 & 0 & -m_f & 0 & 0 & 0\\ 0 & -1 & 0 & 0 & 0 & -m_f & 0 & 0\\ dcos(\theta_1) - tsin(\theta_1) & dsin(\theta_1) + tcos(\theta_1) & 0 & 0 & 0 & 0 & -I_T & 0\\ -dcos(\theta_2) - fsin(\theta_2) & fcos(\theta_2) - dsin(\theta_2) & 0 & 0 & 0 & 0 & 0 & -I_F\\ 0 & 0 & 1 & 0 & -1 & 0 & dcos(\theta_1) - tsin(\theta_1) & -dcos(\theta_2) - fsin(\theta_2)\\ 0 & 0 & 0 & 1 & 0 & -1 & dsin(\theta_1) + tcos(\theta_1 & fcos(\theta_2) - dsin(\theta_2)\end{bmatrix} \begin{bmatrix} H_x\\ H_y\\ \ddot{x}_{GT}\\ \ddot{y}_{GT}\\ \ddot{x}_{GF}\\ \ddot{y}_{GF}\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ m_T g \\ 0 \\ m_B g + m_F g \\ 0 \\ m_B g f cos(\theta_2) \\ f\dot{\theta}_2^2 cos(\theta_2) + d\dot{\theta}_1^2 sin(\theta_1) - d\dot{\theta}_2^2 sin(\theta_2) + t\dot{\theta}_1^2 cos(\theta_1) \\ d\dot{\theta}_2^2 cos(\theta_2) - d\dot{\theta}_1^2 cos(\theta_1) + f\dot{\theta}_2^2 sin(\theta_2) + t\dot{\theta}_1^2 sin(\theta_1) \end{bmatrix}$$

We can then input these equations into MATLAB.

4.2.5 MATLAB Simulation

Initial conditions are supplied and the solution is animated. The solver accounts for the two flight and ground phases by including an "Events" system. The first event, Flight, activates when the value of O_y changes from positive to negative, indicating a change in direction of the force "holding" the spider to the ground. At this point, the solver switches to Phase 2 (where O_x, O_y, and $\vec{P}$ no longer act on the system). The second event, Ground, activates when the body of the spider hits the ground, and stops the solver.

A number of checks were used on the simulation. At all times, the total energy of the system due to kinetic and potential energy was calculated. Because we took the pressure force and the mass of the body to be *external* forces, energy was being added to the system for the entirety of the simulation, as can be seen in Figure 7. We can check then, that the potential and kinetic energy add up to the energy gained from the pressure and body mass forces.

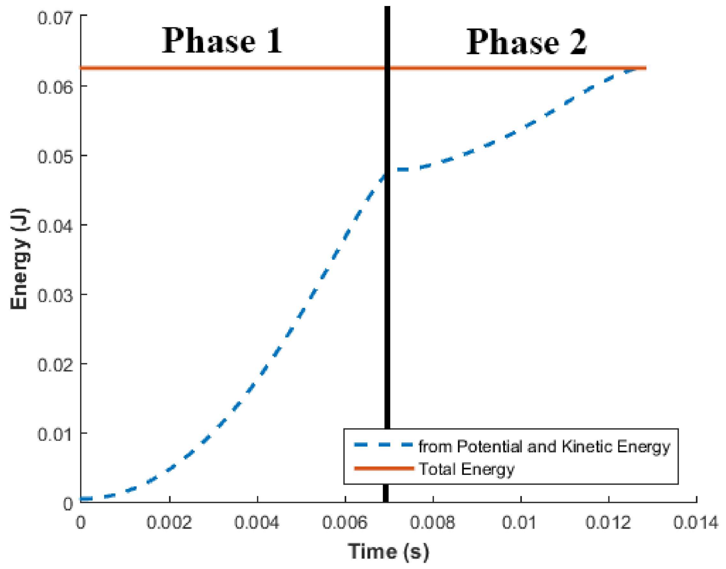


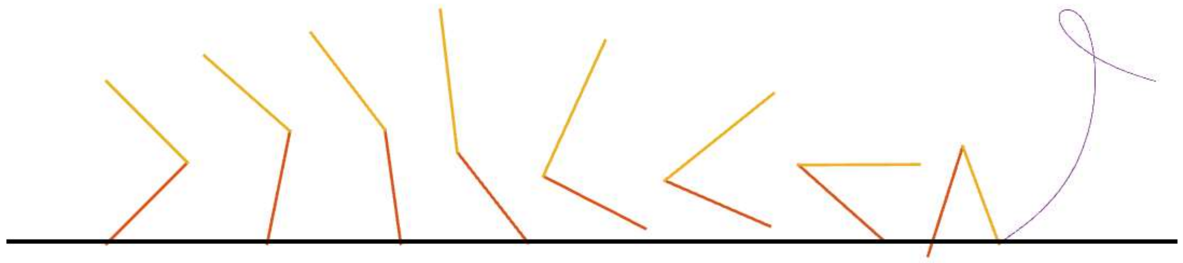
Figure 7: Conservation of Energy Check, with initial conditions $\theta_1 = \frac{\pi}{4}$, $\theta_2 = \frac{3\pi}{4}$, and $p = 60kPa$. The orange line represents the total energy that the system should have at the end of the simulation. It is calculated from the starting potential energy, the energy added by the pressure force, and the energy added by the force of the mass of the body. The dotted line represents total potential and kinetic energy. It's movement does what we expect it to - that is, gain energy until the force from pressure in Phase 1 is removed, at which point the only external force acting upon it in Phase 2 is the body mass, which causes it to approach the orange line.

In addition, a number of test cases were checked:

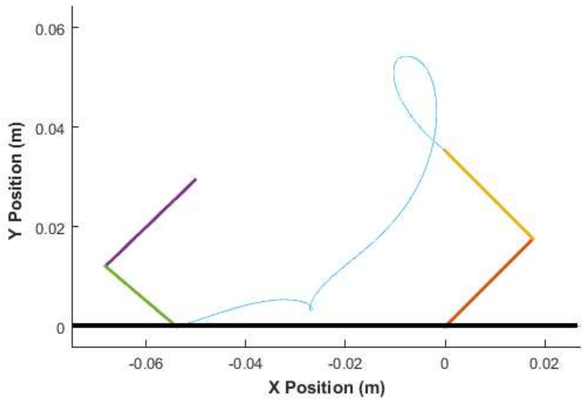
- Pressure = 0
  When the pressure is reduced to zero, the simulated spider should simply fall to the ground. The simulation performed as expected.

- Double Pendulum
  When the pressure is reduced to zero, the thickness of the leg is reduced to 0, and the body mass is reduced to 0, the simulated spider leg should act like a double pendulum system. The simulation performed as expected.

- Single Pendulum
  When the pressure is reduced to zero, the thickness of the leg is reduced to 0, the body mass is reduced to 0, and the length of the femur is reduced to 0, the simulated spider should act like a simple pendulum. The simulation performed as expected.



(a) Note the movement of the top of the femur (the body) traced by the purple line. The spider body clearly moves forward. However, the body loops around haphazardly, as we did not include a collision system in our simulation, and the mass of the body is much larger than the mass of the legs. Therefore, it is free to rotate around the joint. The total horizontal distance jumped was $3.43cm$ at a pressure of $60kPa$, and the total y distance reached was $5.09cm$. Because each leg was $2.5cm$ in length, this tells us that most of the pressure goes towards propelling the spider forwards, and *not* upwards.



(b) At a pressure of $200kPa$, the jump distance increased to $5.39cm$, and the max vertical height was $5.43cm$.

Figure 8: Simulations run in MATLAB. Both simulations have initial conditions $\theta_1 = \frac{\pi}{4}$, $\theta_2 = \frac{3\pi}{4}$, and a tibia and femur length of $2.5cm$.

The MATLAB simulation shows most of the energy propelling the spider in a horizontal direction, and not the vertical direction. This matches what we know about how spiders actually jump – most of their movement is horizontal and not vertical. We can use the motion as a good baseline for what to expect when testing the actual model. The full MATLAB code is attached in the Appendix.

5 Motivation and Design

One of the biggest challenges of making the design for the leg is achieving the scale set out in our goal. Many current micro-fabrication techniques are too small and expensive for the size and quantity we need for a prototype. As such, we focused on limiting most of our fabrication to rapid prototyping methods including Objet printers.

5.1 Joint Design

Previous research by the Max Planck Institute utilized a segmented joint design that encapsulates an internal bladder[7]. This method is proven to work, but the segments limit the minimum size of the leg due to manufacturing constraints for rapid printing methods.

Instead, we have chosen to experiment with flexible thin materials to create the pressure containing bladder. As a starting point, balloon latex was used to make sure our prototype could extend. This is similar to other research being done with pneumatic joints [5], but using hyperelastic materials usually results in a parasitic loss in the applied work and limits the maximum pressure that can be applied. To limit the elasticity, adding a lightweight ribbon around the balloon can be added to help prevent this parasitic loss of power.

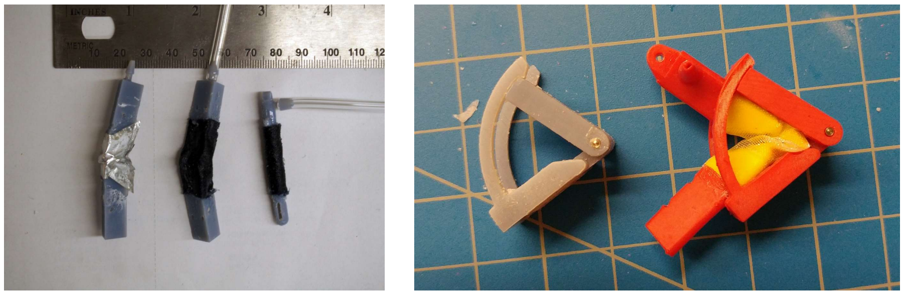


Figure 9: Examples of prototypes developed.

We also explored other alternative, non-hyperelastic materials for the joint. Mylar was an easy material to test due to it's ubiquity and ease of creating customized shapes. Mylar has a limited cycle life, but manufacturing custom shapes is fairly easy with a soldering iron and a knife. Other plastic-lined fabrics also have similar properties, but they are typically very stiff.

Silicone impregnated fabric, a common technique for strain-limited flexible robots, was another choice we explored. This material has to be made in the lab, but this allows for some variability in the base cloth. The material itself is much thicker than mylar with a much greater cycle life than mylar, but it is also much stiffer and more difficult to create an air seal.

A major benefit of a segmented design is that the bladder was independent of the hinge. The previous 3 materials discussed require consideration of pneumatic sealing in the flexed and extended positions. Depending on how the joint is designed, the bladder material has to be bonded to the leg in a systematic way. Since the inner and outer edges are different lengths when flexing or extending, extra bladder material needs to exist to allow for full joint movement.

5.2 Latch Design: Mechanical Imbalances

One of the properties of jumping is the rapid release of energy that propels the object. Pressure valves tend to be fairly large and/or expensive for our application, so instead we are planning on using a latch release mechanism to provide the instantaneous release of energy. Once the pressure builds up enough, the latch will break open and release the stored energy.

Ideally, this latch would require a mating force much lower than the de-mating force. The goal would be to have at least a 1:2 ratio of mate to de-mate force requirement. This would ease the force requirements on our flexing tendon, allowing for a smaller actuation device.

In prototyping, our material selection was limited to VeroBlue, VeroClear, or Carbon 3D's RPU. Both VeroBlue and VeroClear have similar material properties (See Table 1), and Carbon 3D's material is slightly weaker. The following equations from Bayer [1] were used as guidelines to design the latches.

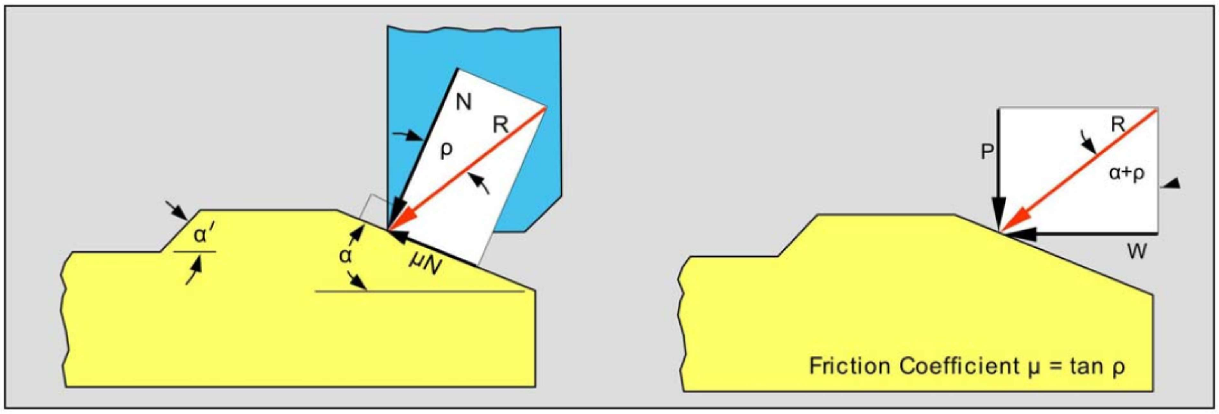


Figure 10: Relationship between deflection force, P, and mating force, W.

Latch Deflection, where ε is strain, l is the length of the latch, and h is the thickness of the latch at the root.

$$y = 1.09\frac{\varepsilon l^2}{h} \tag{14}$$

Latch Deflection Force, where b is the width of the latch at the root and E_s is the Modulus of Elasticity.

$$P = \frac{bh^2}{6}\frac{E_s\varepsilon}{l} \tag{15}$$

Latch Mating/Unmating Force, where μ is the coefficient of friction of the material, and α is the mate/unmate angle.

$$W = P\frac{\mu + tan(\alpha)}{1 - \mu tan(\alpha)} \tag{16}$$

For annular latches, the equations are a bit more complicated.

Latch Deflection, where d is the diameter at the joint.

$$y = \varepsilon d \tag{17}$$

Latch Deflection/Transverse Force, where y = undercut, and X is the geometric factor.

$$P = ydE_sX \tag{18}$$

The Latch Mating/Unmating Force remains the same for both annular and cantilevered latches.

These latch equations give us a lot of freedom to play around with different variables and try different latch designs. The same result can be reached multiple different ways. Therefore, we tested many different types of latches in the hopes of finding several that would fit within our application. Due to the rotational motion of the leg, there are also additional considerations needed to get the latch to work appropriately.

5.3 Solenoid Design

In order to retract the leg after jumping, we needed to develop an actuation method to reset the jump. We decided on linear actuation over a rotational cam design – specifically, we decided to build a solenoid actuator because the actuation distance is relatively small. This minimizes the need for a gearbox to reduce the motor speed, which we think could result in some potential weight savings.

Inertial weight is also a consideration, since the pressure will have to accelerate the solenoid magnet during the jump. On the contrary, a motor-based design will require the leg to overcome the rotational inertia of any gear that's attached to the cable. We also think that the use of moving gears for a release mechanism, similar to those done in previous studies, would be less reliable in a standalone setting [7].

The purpose of the solenoid was to return the spider leg to the bent position. The goal was to build a solenoid linear actuator that could pull a string with enough force to bend the leg and latch the joint. Initially we were trying to design a solenoid that could produce one Newton of force while using the least amount of voltage possible and not drawing more than 3A of current. We wanted the solenoid to be as small as possible so that it would not add too much weight to the PCB when mounted on top. Another factor we had to consider was amount of power dissipated by the solenoid because if the solenoid dissipated too much energy, then the power source we were using would cut us off in regards to how much voltage it supplied, and we also ran a high risk of starting a fire.

Our design needs to be small yet powerful; we consulted Schimpf [6] for guidance in designing the dimension of the solenoid including what AWG of wire we should use, the dimensions of the solenoid, and voltage for the desired force.

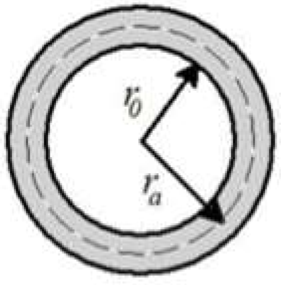


Figure 11: Diagram of the solenoid depicting the variables used. The metal core radius is r_0 and the mid-line of the wire coils is r_a

$$F = \frac{-V^2 \mu_r \mu_0}{8\pi\gamma^2 l^2} \left(\frac{r_0}{r_a}\right)^2 \alpha e^{\frac{-\alpha x}{l}} \tag{19}$$

The force can be estimated using the equation below per Schimpf [6]. However, some preliminary testing revealed this equation did not match predicted force results, thus a simplified form was used.

$$F = \frac{-V^2 \mu_0}{8\pi g^2 l^2} \left(\frac{r_0}{r_a}\right)^2 \tag{20}$$

The derivation is similar to the previous equation except it assumes more ideal conditions to simplify the math. V is voltage, l is the length of the solenoid in meters, g is the distance between the iron magnature and the solenoid in meters, μ_0 is a constant equal to $4\pi * 10^{-7}$, r_0 is the distance from the center of the solenoid to the first coil, and r_a is the radius from the center of the solenoid to the middle coil as shown in Figure 11.

The first solenoid we built was 10mm, had an AWG of 28, and r_a equal to 3.5mm where r_a was the radius from the center of the solenoid to the middle of the copper windings as shown in the image above. The next two solenoids we built were 20mm and 15mm in length, 6 layers of coils, and using the 36 AWG wire.

Once we had decided on the dimensions to fit the amount of force needed to mate the latch, we designed a frame to fit around the solenoid to ensure that it would not overextend. This frame will ideally be attached to the body of the spider.

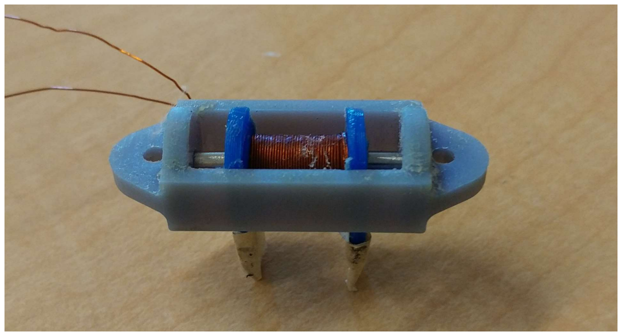


Figure 12: Solenoid Coil Holder for attachment to the spider body. The coil moves back and forth inside the coil holder to pull the string/tendon.

6 Methods and Procedure (Testing)

6.1 Fabrication Methods

To expedite the prototyping process, we decided to stick with rapid prototyping techniques to allow for quick, iterative design. Using a Stratasys Objet30 3-D printer, we were able to quickly print off a few initial designs, to ensure that certain tolerance built into the leg were appropriate. This particular printer has a layer thickness of 28 microns (0.028 mm) and an accuracy up to 0.1 mm, dependent on the material used and the part geometry. This also gave us a baseline feel for the scale of our fabrication methods. We also printed components on the Carbon 3D printer, but results varied dramatically depending on part orientation and support design.

Material	**Tensile Strength (MPa)**	**Elongation at Break (%)**	**Modulus of Elasticity (MPa)**	**Flexural Strength (MPa)**
VeroBlue RGD840	50 - 60	15 - 25	2000 - 3000	60 - 70
VeroClear RGD810	50 - 65	10 - 25	2000 - 3000	75 - 110
Carbon 3D RPU	43 - 47	20	1700 - 2100	N/A

Table 1: Polyjet Material Properties

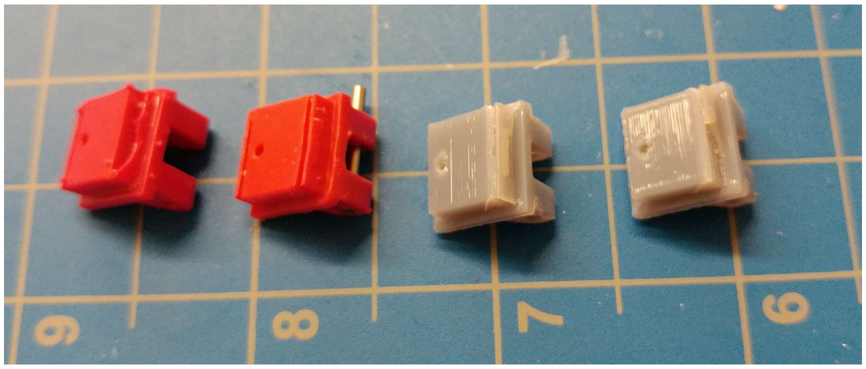


Figure 13: Comparison of the Carbon 3D and the Objet print qualities and capabilities. Red components are Carbon3D prints while the light blue components are Objet prints.

Our preferred material was Stratasys's Polyjet VeroBlue, though some parts were printed in VeroClear, properties of which can be found in Table 1. The rigidity and strength of both materials is about the same, although both came out of the printer encased in support material, which was often difficult to clean off completely.

Printing with the Carbon 3D RPU will require some tuning to get orientations and design accommodations. The printer had difficulty reliably producing the latch's ramp and other key features. Due to the curing process, some of the material also warped making the effective latch mate and de-mate forces lower than intended. However, the Carbon 3D is capable of printing very small features with some level of success, with small features like the latch on the tophat working fairly well when the feature is printed correctly.

6.2 Latch Testing

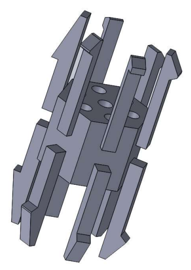
We had previously determined that a demating force of 4N and a mating force of at most half that amount (but as small as possible) would be necessary for the latch mechanism. Latch design equations show us that there are multiple ways to achieve this goal, primarily by changing the geometry of the base and angle of mating/demating. We tested multiple latches to determine which would work best within the spider leg.

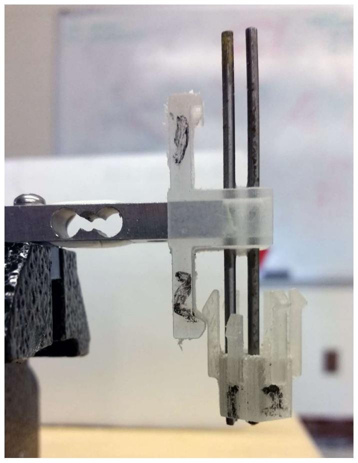
A *Phidgets* Single-Point Load Cell rated for 780 g (7.644 N) was used to test the force required to mate and demate the latches. Several test latches were attached to a hexagonal piece lined with guide holes - a separate attachment for the load cell was fabricated and two rods were used to ensure that the guide holes lined up and the force measured would be in one direction.

The load cell was first connected to a *Phidgets* PhidgetBridge input, which interpreted the voltage output and relayed the information to a computer via USB. The load cell was then calibrated with a known weight – we

(a) *Phidgets* Single-Point Micro Load Cell (0-780g) CZL616C

(b) *Phidgets* PhidgetBridge 4-Input


(c) The hexagonal test piece can test up to twelve latches.


(d) Latch Testing Setup, with load cell attachment and test piece.

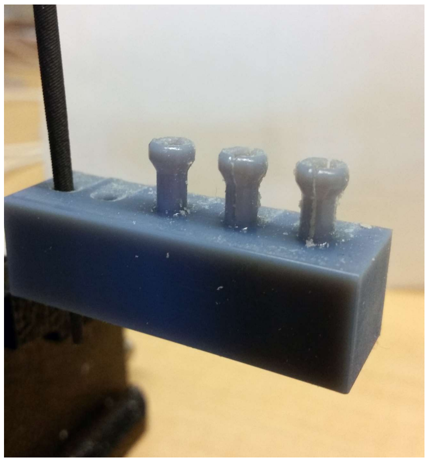

(e) A different load cell attachment for annular latches.

Figure 14: *Phidgets* kit and accessories for measuring force. The load cell connects to the bridge, which connects via USB to a computer. A Python script interprets the results.

assumed the calibration was linear and used *Phidgets* software to perform the calibration. For the linear latch tests, the calibration was $y[g] = 946.2366x[mV] + 5.1854$. Then the mass data was converted to force data by multiplying by the acceleration of gravity, $9.8m/s^2$. Once the load cell attachment was on the load cell, three trials each were done for both latchment and delatchment, for 18 different latches. The average of these trials were taken for a value of latchment/delatchment force. At the time of this writing, latch tests are still being conducted.

In addition to the traditional cantilevered latch designs, separate load cell attachments and test latches were made for testing annular snap joints. The process works much the same – the only difference is in the attachment and test latches.

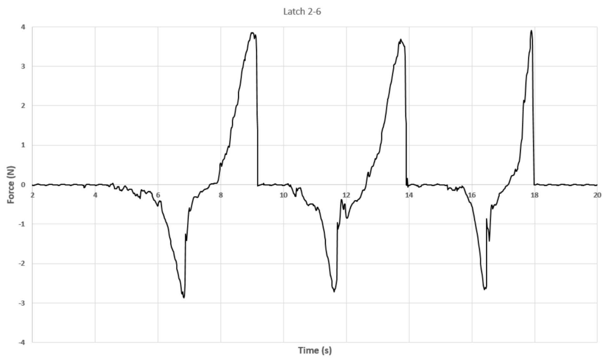


Figure 15: Results from one of the latch tests, showing an unmating force in line with our goals (the positive axis), but a mating force that is not optimal (the negative axis).

Latch	Mate Angle (deg)	Mating Force (N)	Unmate Angle (deg)	Unmate Force (N)
1	30	3.5	90	3.3
2	45	2.8	50	4.1
3	45	2.0	60	2.2
4	15	10.0	45	10.0
5	30	3.25	45	3.8
6	30	2.9	60	3.9

Table 2: Latch Test Results for Latches 1-6. Certain discrepancies are noticeable – however, some of the differences can be accounted for by the different lengths of the latches, which are not shown here.

After assessing the preliminary latch designs, we decided that linear latches were more adaptable and predictable for our application. To maintain better latch performance, the latch was arced with a the characteristic ramp at the end of this arc with a 5 degree angle with respect to the top retainer.

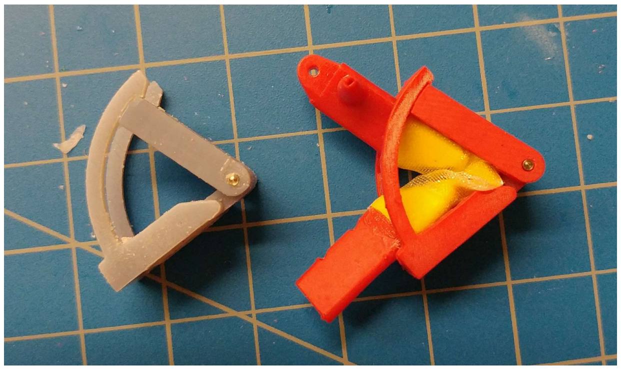


Figure 16: Example of the outer retainer in the latched state. On the left is a leg printed using the Objet Printer, and to the right is a fully assembled leg with components printed on the Carbon 3D printer.

Depending on where and how the simulated tendon is attached to the leg, the force requirements will vary. From testing real, physical latch designs, the max force before the transient averaged out to be 1.307N for the top hole, 1.934N for the 2nd hole from the top, 2.804N for the 3rd hole from the top, and 8.900 N for the bottom hole.

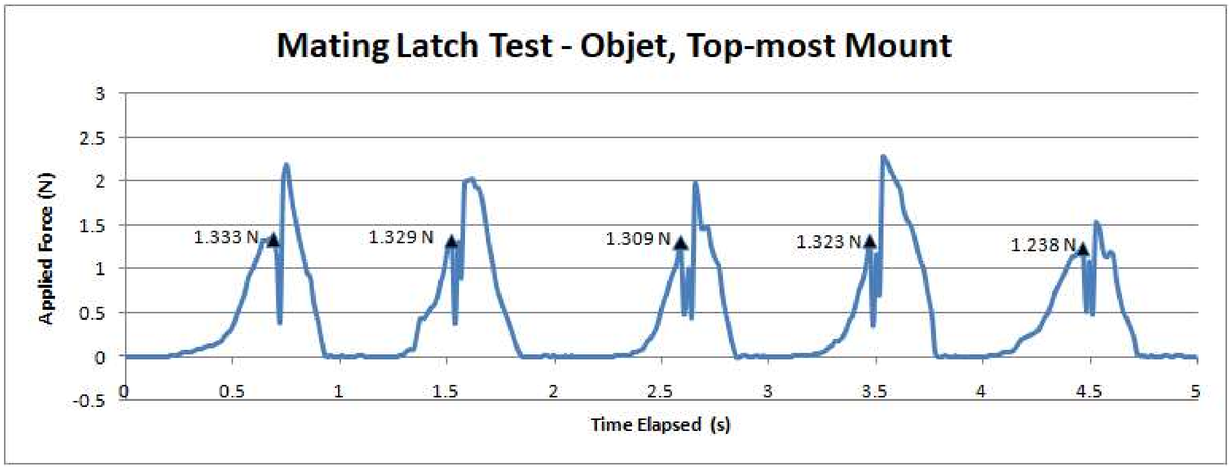


Figure 17: Results from using the top set of holes of the retainer hinge showing the mating force.

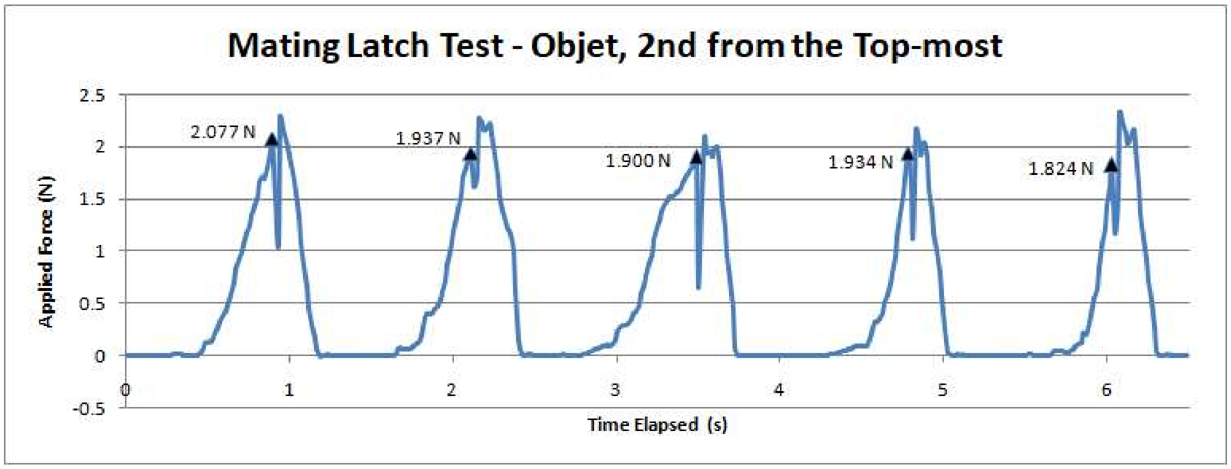


Figure 18: Results from using the 2nd set of holes of the retainer hinge showing the mating force.

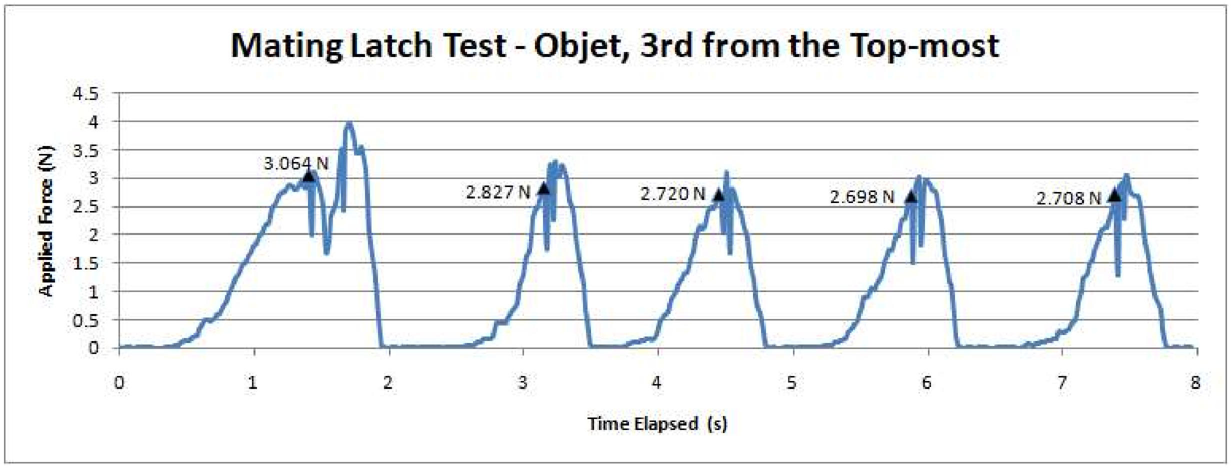


Figure 19: Results from using the 3rd set of holes of the retainer hinge showing the mating force.

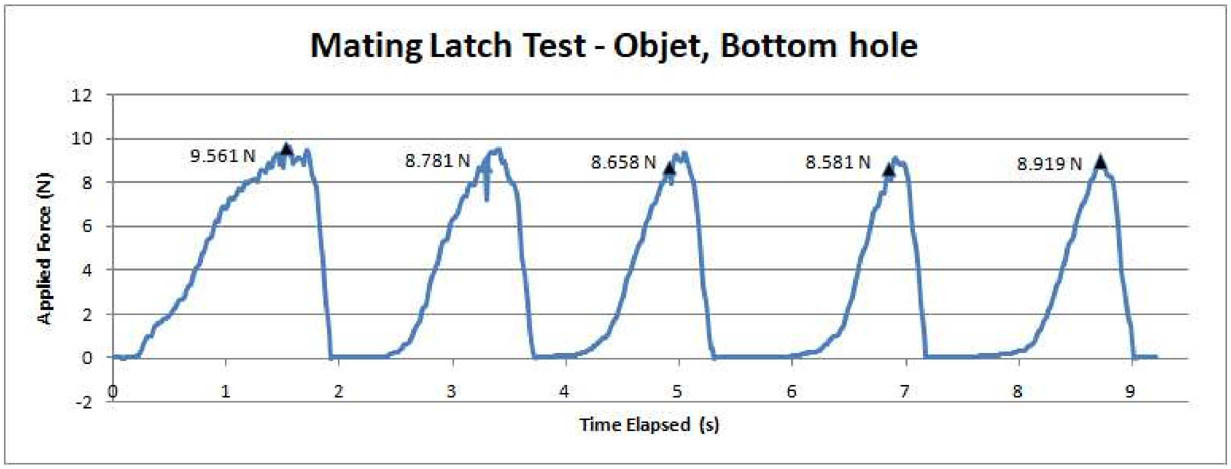


Figure 20: Results from using the bottom set of holes of the retainer hinge showing the mating force.

An interesting thing to note is that there is a transient after the first peak that happens on nearly every mate that occurs. We believe this is because the latch has fully mated at this point, causing a rapid dip in the force in the cable. The following peak is presumably due to the leg joint bottoming out soon after the latch has mated. As such, the mating force was labeled as the first peak for each test run.

The de-mating force was also meausred for the leg. The first design used a 90 degree angle that mated flat with the upper retainer, and this proved to be a very strong latch mechanism. To make it possible to de-mate with air pressure, a chamfer was added to the upper leg joint to give a small ramp. Ideally, both sides of the leg would have corresponding surfaces to avoid stress concentrations. On Carbon3D prints, we also noticed that the latch would occasionally snap off instead of releasing properly.

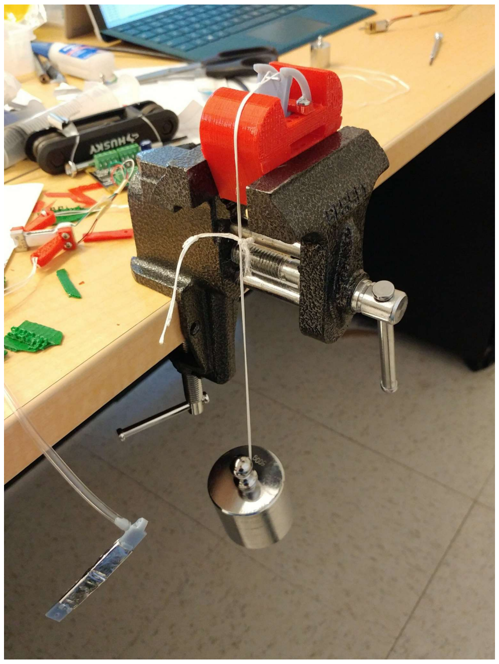


Figure 21: De-mate test with a 500g weight (4.90N).

6.3 Solenoid Strength

The solenoid used to contract the leg was tested in much the same way that the latches were tested. A load cell attachment was made and attached to the current coil holder in such a way that the load cell can measure the force of pulling on the string.

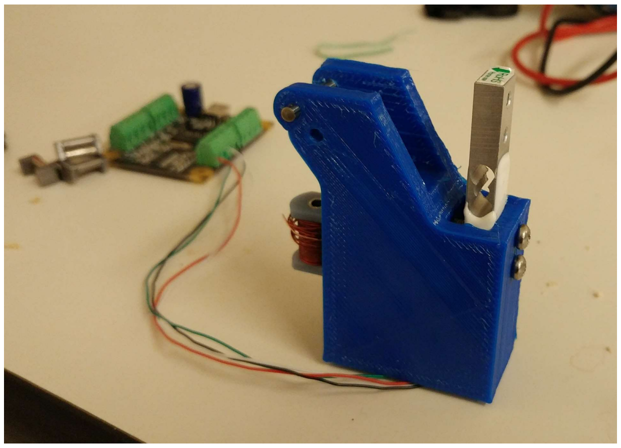


Figure 22: Test rig used to measure the output force of the solenoids

When we tested this solenoid we connected it to the power supply and set the desired voltage to 15V. The resistance of the solenoid was measured to be 1.17 . Since the resistance was so low, the solenoid dissipated a large amount of energy when 15V was applied so the power supply automatically cut us off at 3.7V. With 3.7V across the solenoid we got a force significantly lower than what we desired as shown in the results below.

Voltage Applied (Volts)	**Current Drawn (Amps)**	**Force Measured (Newtons)**
3.7	3.16	0.11
2.2	1.9	0.04

Table 3: First Solenoid Testing

For the first solenoid we were trying to produce the most amount of force possible using the smallest solenoid design. Since the resulting solenoid had such a low resistance, the power supply limited us to max 3.7V instead of the desired 15V. We decided to redesign the solenoid so that it would have a larger resistance and thus allow for a larger voltage to be applied across the solenoid.

During our testing, the first solenoid could not provide a sufficient force to pull the leg as designed. During testing, we could only apply 0.11 N, which was not enough to engage the latch. The second and third solenoids designed were longer to see if we could generate more force and meet our requirements.

Voltage Applied (Volts)	**Current Drawn (Amps)**	**Force Measured (Newtons)**
5	0.27	0.199
10	0.48	0.0.209

Table 4: 15mm Solenoid Testing Results

Voltage Applied (Volts)	**Current Drawn (Amps)**	**Force Measured (Newtons)**
5	0.191	0.215
10	0.350	0.294
15	0.450	0.364

Table 5: 20mm Solenoid Testing Results

The results show that the 20mm solenoid produced the largest amount of force at 15V. Based on the force equation mentioned earlier, we expected the shorter solenoid to produce the larger force because there is a l^2 term in the denominator. We could not discern a logical explanation for why our results did not match our predicted force results based on the force equation from Schimpf, so we decided to use the simplified force equation with fewer terms to analyze the behavior of the solenoid. However, more importantly, none of the built solenoids with the Voltage inputs tested generated enough force per the initial estimates and per the actual latch tests.

6.4 Joint Materials and Testing

As discussed earlier, the joint material is, in large part, what makes this design unique. So far, the only viable materials that are cheap, easily produced, and bond-able to Objet materials are the latex balloons, mylar, and silicone cloth sheets.

Latex balloons alone were too elastic for our purposes. To combat this hyper-elasticity, a layer of a thin mesh ribbon was added to surround the balloon after an air-tight seal was created. This restricted the balloon's expansion and dramatically improved the pressure capabilities of the joint, going up to 350 kPa absolute pressure without issue.

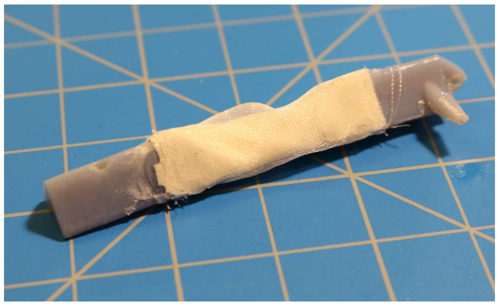


Figure 23: Image of the leg joint using balloons with a ribbon on the outer layer. A rounded triangular section was used for this design.

With the mylar joint, manufacturing required several stages. First, the backside of the mylar had to be glued while the joint was in the "flexed" state. This allows the joint to flex freely during pressure applications. Next, the joint had to be sealed to create the seam joining the ends of the Mylar. Afterwards, the remaining edges needed to be glued to the leg while the leg was in the extended position. Lastly, the leg had to be checked for air leaks which was remedied with additional glue. Each time glue was applied, we had to wait for the glue to cure before moving to the next step. This is not ideal since it's hard to guarantee a working, air tight system.

An identical procedure also had to be taken to manufacture the first leg with silicone impregnated sheets. As seen in Figure 24, the middle leg was manufactured with the same process as the Mylar leg. As an improvement to this procedure, an external hinge was developed to go around the leg depicted on the right in Figure 24, making it easier to produce, but still had challenges related to creating the air-tight seal.

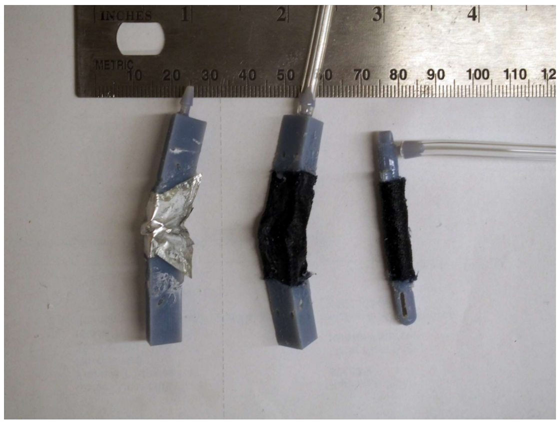


Figure 24: Images of the various legs with different joint materials. The left leg has a triangular cross section with Mylar, the middle leg has a triangular cross section with silicone impregnated cloth, and the right leg has a circular cross section with silicone impregnated cloth.

During fabrication tests, the joints with silicone impregnated cloth would build up significant thickness around the overlap, causing integration issues with the outer hinge. The solution with balloons and ribbons maintains a comparably thin layer with thickness buildup mainly due to super-glue. The mylar joint has the thinnest material layer overall, but the limited cycle life makes it impractical for this application.

With all of the aforementioned materials, we were able to get desired behaviors with restricted expansion, but ultimately the balloon with a ribbon attached showed the best performance behaviors and most reliable pressure retention. Using super glue, we were able to quickly and easily fabricate air-tight sealing joints within a matter of minutes. The ribbon helped restrict the expansion of the balloon, although the area of ribbon overlap can be tricky as the balloon can sometimes expand through this spot.

6.5 Leg Assembly Jump Test

In order to verify design requirements and capabilities of the assembly, the jump needs to be performed prior to completing the design and selection powered components. To do so, an absolute pressure sensor was attached

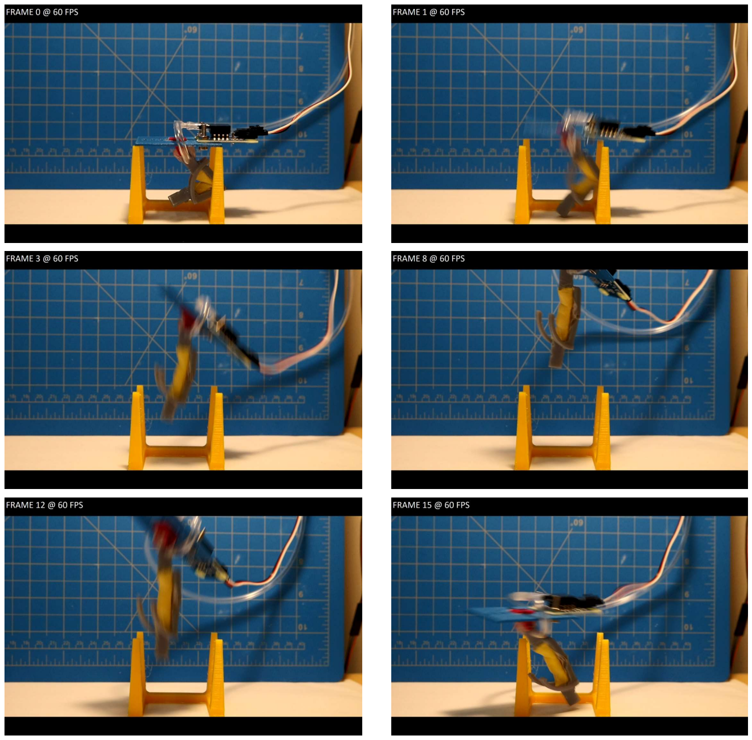


Figure 25: Jumping Leg slow-motion. Total jump spans approximately 0.25s. Filmed at 60 frames per second with images captured at 0th, 1st, 3rd, 8th, 12th, and 15th frames.

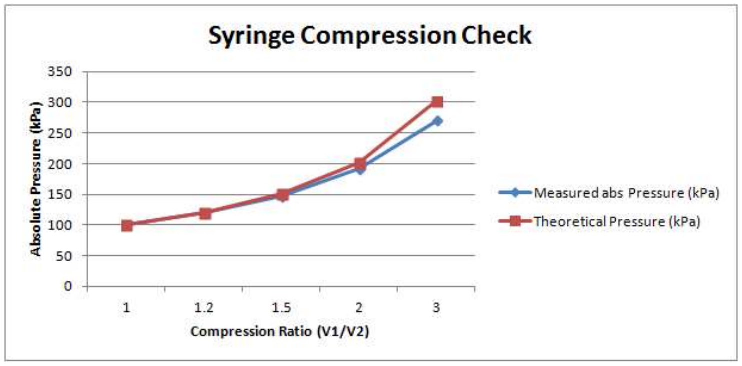


Figure 26: Syringe Compression Check

in-line with the spider leg's pneumatics. This allowed us to measure the pressures of the system through the entire jump and determine what the maximum pressure need to de-mate the latch as designed.

With the pressure sensor, we were able to get characteristic data with the syringe compression. Using the ideal gas law, p_1 and p_2, the volume relationship of V_1 and V_2 is as follows.

$$\frac{p_2}{p_1} = \left(\frac{V_1}{V_2}\right) \tag{21}$$

The following compression tests with a syringe were performed to measure the compression efficiencies. From the tests, the compression efficiency was fairly good, only dropping to 84.4% at the largest compression checked.

When we ran the tests, the pressure requirements to de-mate the 3mm thick latch ended up requiring, on average, about 304.84 kPa at close to a 7:1 compression ratio. The compression requirement for air is quite high, and would need a considerable amount of work done by the pneumatic compression mechanism to produce the desired pressure.

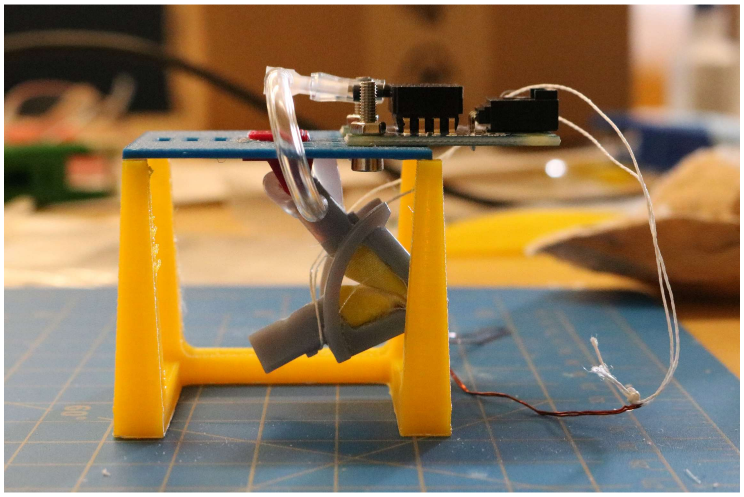


Figure 27: Arachnabot leg in its current stage of prototyping

7 Current Prototype

As of this writing, the prototype does not have any electronic components integrated into the system. The leg itself is currently an alternative design to the previous segmented joint design, but with more potential for miniaturization.

The PCB has not been designed yet for the jumping spider, thus a dummy, stand in PCB was added to mimic the end configuration. This worked well, as it allowed for quick addition of components like the pressure sensor with minimal destruction. A small stand was also created to make the jumping test easier, as it does not interfere with the movement of the leg during jumping sequences.

The leg was fixed at the top pivot pin to restrict the movement of the joint. When tested without fixing the top joint, the leg's extension was too unrestricted, and the pressure force went into pivoting rather than vertical acceleration. Ideally, this joint would also be pressurized, allowing for more work to be performed by the compressed air.

The payload capabilities for the leg itself should be characterized before determining the final electronic power requirements. This way, the desired performance characteristics can be achieved while keeping a decent safety margin on power needs. While not tested yet, it is assumed that the current prototype cannot jump nearly as high with all the required components mounted. It is also assumed, but remains to be tested, that the de-mate force requirement can be increased to achieve greater jumping capabilities.

8 Conclusion and Future Work

In order to launch a 30 g, 5 cm tall robot twice its height into the air, several additional items need to be investigated further on top of the work that has been done.

Ongoing investigations into sleeve design and the joint membrane seal have provided a potentially viable path forward. We are still looking into the design of the mechanical actuator to squeeze air or fluid into the legs. A separate actuator will be placed in the body of the spider to bend and flex the legs; currently a solenoid design is being evaluated to that end. The biggest challenge still to overcome is scaling down the size and weight of the robot. We are exploring novel fabrication methods towards this goal.

8.1 Dynamics Updates

The simulation is not perfect, and a number of clues point to problems in the model. We would expect, for example, the energy added to the system in Phases 1 and 2 to be added in a linear fashion because both the pressure and body mass forces are constant. And yet we can see in Figure 7 that that is not the case – they are actually quadratic.

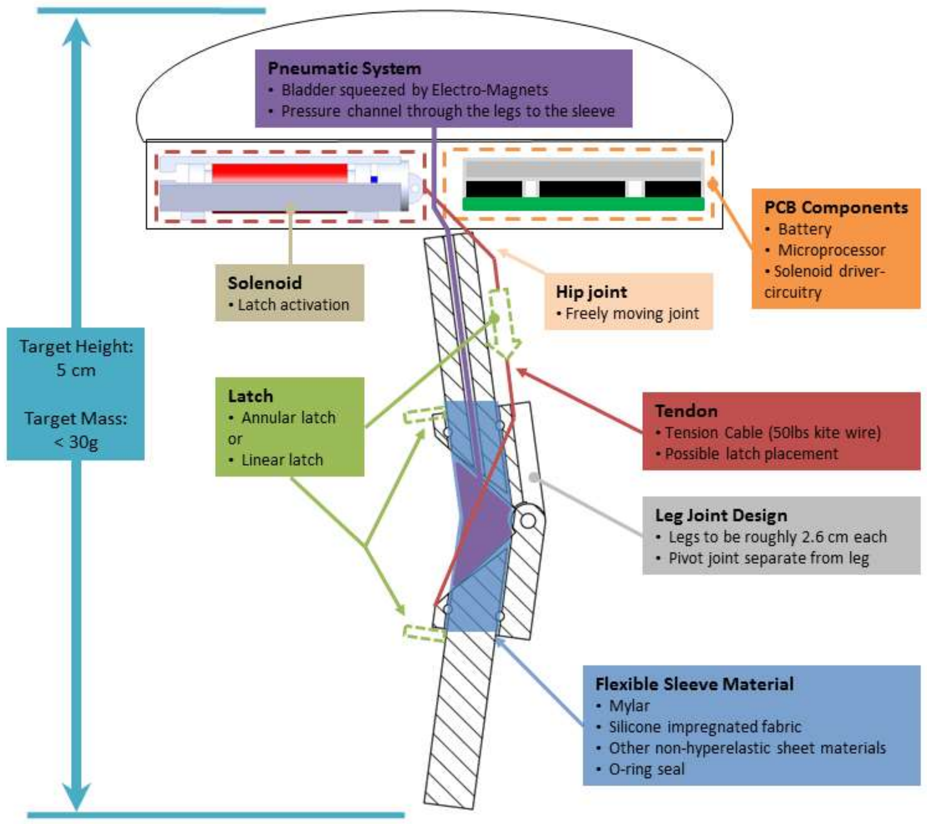

Pneumatic System
• Bladder squeezed by Electro-Magnets
• Pressure channel through the legs to the sleeve
PCB Components
• Battery
• Microprocessor
• Solenoid driver-circuitry
Solenoid
• Latch activation
Hip joint
• Freely moving joint
Target Height: 5 cm
Target Mass: < 30g
Latch
• Annular latch or
• Linear latch
Tendon
• Tension Cable (50lbs kite wire)
• Possible latch placement
Leg Joint Design
• Legs to be roughly 2.6 cm each
• Pivot joint separate from leg
Flexible Sleeve Material
• Mylar
• Silicone impregnated fabric
• Other non-hyperelastic sheet materials
• O-ring seal

Figure 28: Jumping Robot Proposed Layout

Perhaps the best way to fix that problem would be to change the pressure and body forces to be internal rather than external. That would involve expanding the model to include the body and the bladder, and accounting for internal forces between the two systems. Then the calculated energy would be expected to be constant over the timeframe of the simulation.

The simulation could also use a separate Events class to handle collisions, so that the leg does not bend backwards upon itself or rotate around its own hinge – the joint design prevents these events from happening in the non-simulated world.

8.2 Body and PCB Design

To keep the robot's overall weight down, the PCB will serve as mounts for both the electrical components and the leg(s). Limited development has been done in this area and further research is required. Based on our preliminary assessment, the robot will need a pressure sensor, an H-bridge, accelerometer, solenoid(s), a battery, and an ATMega microcontroller to control and actuate all the known components, as depicted in Figure **??**.

While some work was performed on the PCB layout, electrical component placement and PCB trace layouts still need to be determined. Based on the solenoid testing and the power requirements for a sufficiently strong solenoid, other alternatives to actuate the leg's tendon will likely need to be explored for future work.

8.3 Bladder Design

In order to design a mobile system, the robot needs to have a means of building up and lowering internal air pressure. Spiders do this by contracting muscles within their body. As a future design component, we propose using a bladder mounted to the body which will be contracted through some mechanical means of action.

One idea which could yield potentially good benefits is the use of switchable magnets. These would only require small bursts of power to actuate on or off and could provide a quick response time. Foreseeable complications with this actuation include the weight of these magnets (effectively multiple solenoids) and the limited effective distance.

There is still much testing to be done for this particular component, thus it remains to be seen if this is a viable actuation method for this application. Depending on the dynamics model, required pressures, and the final leg joint designs, the volume by which the bladder must constrict could be in excess of the 7:1 ratio, a drawback of pneumatic systems.

Due to the high compression needs, the hydraulics option may need to be explored for this to be viable, as it is much easier to generate high pressures with minimal volume change, but this will like require high torque or force actuators.

8.4 Leg Mechanics

The joint design utilizing a balloon with a ribbon has thus far been the best option for the joint bladder design. The performance of the balloon with cloth results in a very simple fabrication method that yields a strong, flexible, and thin option that works very well. The pressure limits of this material have not yet been determined, but this could be explored in future work as part of a means to improve jumping performance.

Silicone impregnated cloths require a lot of care to fabricate, and are not easy to bond to the legs per the flexibility requirement. The lack of consistency with fabrication, added thickness of the material, and insufficient reliability rule out the silicone impregnated cloth as a better alternative.

In addition to material selection, latch selection also needs to be researched further. Over the past few months, several latches have been assessed, but there needs to be some fine tuning to get the latch mate and de-mate forces right to meet the design requirements. Once again, the size of our robot is a restricting factor, as latch sizes are comparable to the size of the leg.

Material options other than those provided in the Objet Printer have not been explored, but could be investigated as an alternative to reduce the latch size. Additionally, the Objet material wears out fairly quickly, making it prone to gradual loss of latching force over multiple actuations.

8.5 Leg Actuation Alternatives

When testing the solenoids with the joint it became clear that the solenoids did not provide enough force to move the leg any significant distance. Future testing of the solenoids needs to be done; however, it's likely that a small enough solenoid that produces the desired force is not possible given the power limitations of the system. Since the solenoids have such low resistance, they dissipate too much energy which causes the power source to limit the amount of voltage applied to the solenoid. Even if we were able to obtain a power source that could constantly supply 15V, we run the risk of frying other circuit components because they do not have a high enough power rating.

In order to make the solenoid generate the required force, additional coils will likely be needed, and the benefits over using a motor may be lost. Nitinol wire could be used as an alternative as they can generate a significant force compared to its weight, but the issue would be during latch de-mating sequence as the air would need to extend the wire.

8.6 Testing

Progress has been made towards proving the latch design concept, but to make it work per our required performance specifications, some additional testing needs to be done.

Extensive solenoid testing was performed with results pointing towards not being an ideal solution. Alternative linear actuators will need to be explored, and it may even be possible to tie it in with the pneumatic system to mate the latch.

The testing rig for the actual jumping robot system also needs to be designed. Due to the predicted dynamics of the jump, there may be a need to restrict lateral movement, so the robot is restricted to vertical movements. This would allow us to get better measurements of how high the robot is capable of jumping.

Testing the upper-limit pressure capabilities of the bladder are also mandatory. Since we are considering switchable magnets, application duration, forces, manufacturability, and power requirements will all need to be assessed when testing the bladder. As an extension of the current design, additional segments could be explored to see if jumping performance is improved or hindered, as spiders have more than 1 joint per leg.

A tethered joint was tested successfully, but the limits of its capability have not been fully explored. Assessing payload limits, pressure requirements, and power requirements will all need to be done to get the untethered solution designed correctly.

In order to build a successfully jumpingm un-tethered robot, we will need to develop the aforementioned tests and any others to vet our designs and make sure we achieve our goal in the coming semesters.

References

[1] Bayer Material Science. "Snap-Fit Joints for Plastic: A Design Guide." Bayer Material Science LLC.

[2] Wulfila Gronenberg. "Fast Actions in Small Animals: Springs and Click Mechanisms." Journal of Comparative Physiology A 178.6 (1996) 178: 727-734

[3] Darrick Hurst. "Sandia hopping robots to bolster troop capabilities." Sandia Labs News Releases. Sept 11 2009. <https://share.sandia.gov/news/resources/news_releases/sandia-hopping-robots-to-bolster-troop-capabilities/>

[4] Mirko Kovac, Manuel Schlegel, Jean-Christophe Zufferey and Dario Floreano. "A Miniature Jumping Robot with Self-Recovery Capabilities." Ecole Polytechnique Federale de Lausanne. Intelligent Robots and Systems, 2009.

[5] Stefan Landkammer, Florian Winter, Daniel Schneider, and Rudiger Hornfeck. "Biomimimetic Spider Leg Joints: A Review from Biomecehanical Research to Compliant Robotic Actuators." Robotics (2016). Special Issue: Mechanics, Control, Design, Conceptualization and Fabrication of Soft Robotic Systems. 5(3), 15.

[6] Paul H. Schimpf. "A Detailed Explanation of Solenoid Force." Int. J. on Recent Trends in Engineering and Technology, Vol. 8, No. 2, Jan 2013

[7] Alexander Sprowitz, Kirstin Petersen, Chantal Gottler, Ayush Sinhaz, Corentin Caerx, Mehmet Ugur Oztekin, and Metin Sitti. "Scalable Pneumatic and Tendon Driven Robotic Joint Inspired by Jumping Spiders." Submitted to the *IEEE International Conference on Robotics and Automation (ICRA) 2017.*

[8] Lena Zentner. "Modelling and Application of the Hydraulic Spider Leg mechanism." *Spider Ecophysiology.* Springer (2013), pp. 451-462.

A Differential Algebraic Equation Derivation in MATLAB

Table of Contents

```
function spider_DAEderive

%See report for math involved

syms g mt mb mf It If t f d T F D th1 th2 om1 om2 al1 al2 ...
    xtddot ytddot xfddot yfddot Ox Oy Hx Hy P real
```

DEFINE REFERENCE FRAMES

```
i = [1 0 0];
j = [0 1 0];
k = [0 0 1];
ip = cos(th1)*i + sin(th1)*j;   %iprime
jp = -sin(th1)*i + cos(th1)*j;  %jprime
ipp = cos(th2)*i + sin(th2)*j;  %i double prime
jpp = -sin(th2)*i + cos(th2)*j; %j double prime
```

POSITION VECTORS

```
r_ogt = -t*ip;
r_pgt = t*ip;
r_hgt = t*ip - d*jp;
r_pgf = -f*ipp;
r_hgf = -f*ipp - d*jpp;
r_bgf = f*ipp;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PHASE 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
```

LINEAR MOMENTUM BALANCES

```
%Tibia
```

```
lmb1 = Ox*i + Oy*j - mt*g*j - P*ip + Hx*i + Hy*j == ...
    mt*xtddot*i + mt*ytddot*j;
eqn1 = lmb1(1);
eqn2 = lmb1(2);

%Femur
lmb2 = P*ipp - Hx*i - Hy*j - mf*g*j - mb*g*j == ...
    mf*xfddot*i + mf*yfddot*j;
eqn3 = lmb2(1);
eqn4 = lmb2(2);
```

ANGULAR MOMENTUM BALANCES

```
%Tibia about GT
amb1 = cross(r_ogt, Ox*i + Oy*j) + cross(r_pgt, -P*ip) + ...
    cross(r_hgt, Hx*i + Hy*j) == It*al1*k;
eqn5 = amb1(3);

%Femur about GF
amb2 = cross(r_pgf, P*ipp) + cross(r_hgf, -Hx*i - Hy*j) + ...
    cross(r_bgf, -mb*g*j) == If*al2*k;
eqn6 = amb2(3);
```

CONSTRAINTS

```
%Acceleration of O is 0
con1 = xtddot*i + ytddot*j == t*al1*jp - t*om1*om1*ip;
eqn7 = con1(1);
eqn8 = con1(2);

%Position of H is Position of H
con2 = xtddot*i + ytddot*j + t*al1*jp - t*om1*om1*ip + d*al1*ip + ...
    d*om1*om1*jp == xfddot*i + yfddot*j - f*al2*jpp + f*om2*om2*ipp
 + ...
    d*al2*ipp + d*om2*om2*jpp;
eqn9 = con2(1);
eqn10 = con2(2);
```

PUT ALL EQUATIONS TOGETHER

```
[A,b] = equationsToMatrix([eqn1, eqn2, eqn3, eqn4, eqn5, eqn6,
 eqn7, ...
    eqn8, eqn9, eqn10],...
    [Ox, Oy, Hx, Hy, xtddot, ytddot, xfddot, yfddot, al1, al2]);

%Creates a MATLAB function which returns the matrices A and b when
 needed
%(i.e. in the rhs_phase1.m function, which uses A and b to solve for
%variables and unknowns until 'flight'

matlabFunction(A, 'file', 'DAE_A1');
matlabFunction(b, 'file', 'DAE_b1');
```

```
fprintf('Done Deriving Phase 1 \n');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% PHASE 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
```

LINEAR MOMENTUM BALANCES

```
%Tibia
lmb1 = - mt*g*j + Hx*i + Hy*j == mt*xtddot*i + mt*ytddot*j;
eqn1 = lmb1(1);
eqn2 = lmb1(2);

%Femur
lmb2 = - Hx*i - Hy*j - mf*g*j - mb*g*j == mf*xfddot*i + mf*yfddot*j;
eqn3 = lmb2(1);
eqn4 = lmb2(2);
```

ANGULAR MOMENTUM BALANCES

```
%Tibia about GT
amb1 = cross(r_hgt, Hx*i + Hy*j) == It*al1*k;
eqn5 = amb1(3);

%Femur about GF
amb2 = cross(r_hgf, -Hx*i - Hy*j) + cross(r_bgf, -mb*g*j) == If*al2*k;
eqn6 = amb2(3);
```

CONSTRAINTS

```
%Position of H is Position of H
con1 = xtddot*i + ytddot*j + t*al1*jp - t*om1*om1*ip + d*al1*ip + ...
    d*om1*om1*jp == xfddot*i + yfddot*j - f*al2*jpp + f*om2*om2*ipp
 + ...
    d*al2*ipp + d*om2*om2*jpp;
eqn7 = con1(1);
eqn8 = con1(2);
```

PUT ALL EQUATIONS TOGETHER

```
[A,b] = equationsToMatrix([eqn1, eqn2, eqn3, eqn4, eqn5, eqn6, eqn7,
 eqn8],...
    [Hx, Hy, xtddot, ytddot, xfddot, yfddot, al1, al2]);

%Creates a MATLAB function which returns the matrices A and b when
 needed
%(i.e. in the rhs_phase2.m function, which uses A and b to solve for
%variables and unknowns until 'ground'
matlabFunction(A, 'file', 'DAE_A2');
matlabFunction(b, 'file', 'DAE_b2');
```

```
fprintf('Done Deriving Phase 2 \n');

end
```

Published with MATLAB® R2015a

B ODE Solver in MATLAB

Table of Contents

```
function spider_solve
```

VARIABLES AND CONSTANTS in MKS

```
global g mt mb mf It If t f d T F D P pP
g = 9.8;                   %Acceleration of Gravity
 [m/s^2]
T = 0.025;                 %Length of the tibia
 [m]
t = T/2;                   %Length to center of mass of the tibia
 [m]
F = 0.025;                 %Length of the femur
 [m]
f = F/2;                   %Length to center of mass of the femur
 [m]
D = 0.0064516;             %Diameter of tube
 [m]
d = D/2;                   %Radius of tube
 [m]
mb = 0.030;                %Mass of Body
 [kg]
mt = 0.0012;               %Mass of Tibia
 [kg]
mf = 0.0012;               %Mass of Femur
 [kg]
If = mf*f*f/12;            %Moment of Inertia of Femur around GF
 [kgm^2]
It = mt*t*t/12;            %Moment of Inertia of Tibia around GT
 [kgm^2]
pP = 60000;                %Pressure
 [Pa]
P = pP*pi*d^2;             %Pressure Force
 [N]
```

TIME TO SOLVE

```
tf = .025;                 %Final Time [s]
```

```
dt = .00005;                %Timesteps [s]
tspan = 0:dt:tf;            %Array of time to solve
```

INITIAL CONDITIONS

```
theta10 = pi/4;                            %Initial Angle 1
 [rad]
theta20 = 3*pi/4;                          %Initial Angle 2
 [rad]
theta1dot0 = 0;                            %Initial Angular Velocity
 [rad/s]
theta2dot0 = 0;                            %Initial Angular Velocity
 [rad/s]
xt0 = t*cos(theta10);                      %Initial x of tibia
 [m]
yt0 = t*sin(theta10);                      %Initial y of tibia
 [m]
xf0 = T*cos(theta10) + f*cos(theta20);     %Initial x of femur
 [m]
yf0 = T*sin(theta10) + f*sin(theta20);     %Initial y of femur
 [m]
xtdot0 = 0;                                %Initial x velo of tibia
 [m/s]
ytdot0 = 0;                                %Initial y velo of tibia
 [m/s]
xfdot0 = 0;                                %Initial x velo of femur
 [m/s]
yfdot0 = 0;                                %Initial y velo of femur
 [m/s]
Ox0 = 0; Oy0 = 0; Hx0 = 0; Hy0 = 0;        %Initial force values
 [N]

%Array of Initial Values
th0 = [theta10 theta20 theta1dot0 theta2dot0 xt0 yt0 xf0 yf0 ...
    xtdot0 ytdot0 xfdot0 yfdot0 Ox0 Oy0 Hx0 Hy0];
```

ODE45 PHASE 1

```
options = odeset('AbsTol', 1e-6, 'RelTol', 1e-6, 'Events', @flight);

%Solve ODE until 'flight' event (in options) --- see flight.m
[time1, th1] = ode45(@rhs_phase1, tspan, th0, options);
```

ACCUMULATE DATA PHASE 1

```
%The array th1 contains all the data solved until 'flight' happens.
 Unpack
%it and organize it.
theta1 = th1(:,1);
theta2 = th1(:,2);
omega1 = th1(:,3);
omega2 = th1(:,4);
```

```
xt = th1(:,5);
yt = th1(:,6);
xf = th1(:,7);
yf = th1(:,8);
xtdot = th1(:,9);
ytdot = th1(:,10);
xfdot = th1(:,11);
yfdot = th1(:,12);
Ox = th1(:,13);
Oy = th1(:,14);
Hx = th1(:,15);
Hy = th1(:,16);

p1_index = length(time1);
thtraveled = abs(th1(end) - theta10);
```

ODE45 PHASE 2

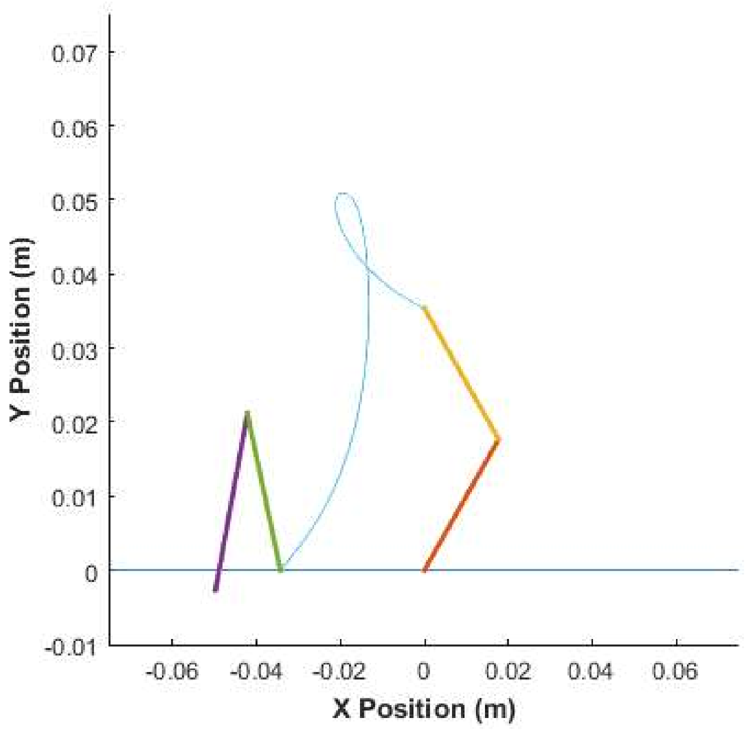
```
%If the event @flight is never reached, ignore this section
if time1(end) == tf
    time = time1;
else

%New Initial Conditions
%The ODE solver must solve from the end of the last ODE solve, so we
 take
%the output of the last solved ODE and start the new ODE with the end
 of
%the last ODE
theta10 = theta1(end);
theta20 = theta2(end);
theta1dot0 = omega1(end);
theta2dot0 = omega2(end);
xt0 = xt(end);
yt0 = yt(end);
xf0 = xf(end);
yf0 = yf(end);
xtdot0 = xtdot(end);
ytdot0 = ytdot(end);
xfdot0 = xfdot(end);
yfdot0 = yfdot(end);
Hx0 = Hx(end);
Hy0 = Hy(end);
tspan2 = time1(end):dt:tf;      %New timespan as well!

%New array of 'initial' values
th0 = [theta10 theta20 theta1dot0 theta2dot0 xt0 yt0 xf0 yf0 ...
    xtdot0 ytdot0 xfdot0 yfdot0 Hx0 Hy0];

options2 = odeset('AbsTol', 1e-6, 'RelTol', 1e-6, 'Events', @ground);

%Solve ODE until event 'ground' --- see ground.m
[time2, th2] = ode45(@rhs_phase2, tspan2, th0, options2);
```

ACCUMULATE DATA PHASE 2

```
%Combine all the data from Phase 1 and Phase 2
theta1 = [theta1; th2(:,1)];
theta2 = [theta2; th2(:,2)];
omega1 = [omega1; th2(:,3)];
omega2 = [omega2; th2(:,4)];
xt = [xt; th2(:,5)];
yt = [yt; th2(:,6)];
xf = [xf; th2(:,7)];
yf = [yf; th2(:,8)];
xtdot = [xtdot; th2(:,9)];
ytdot = [ytdot; th2(:,10)];
xfdot = [xfdot; th2(:,11)];
yfdot = [yfdot; th2(:,12)];
Hx = [Hx; th2(:,13)];
Hy = [Hy; th2(:,14)];
time = [time1; time2];

end
```

DRAW

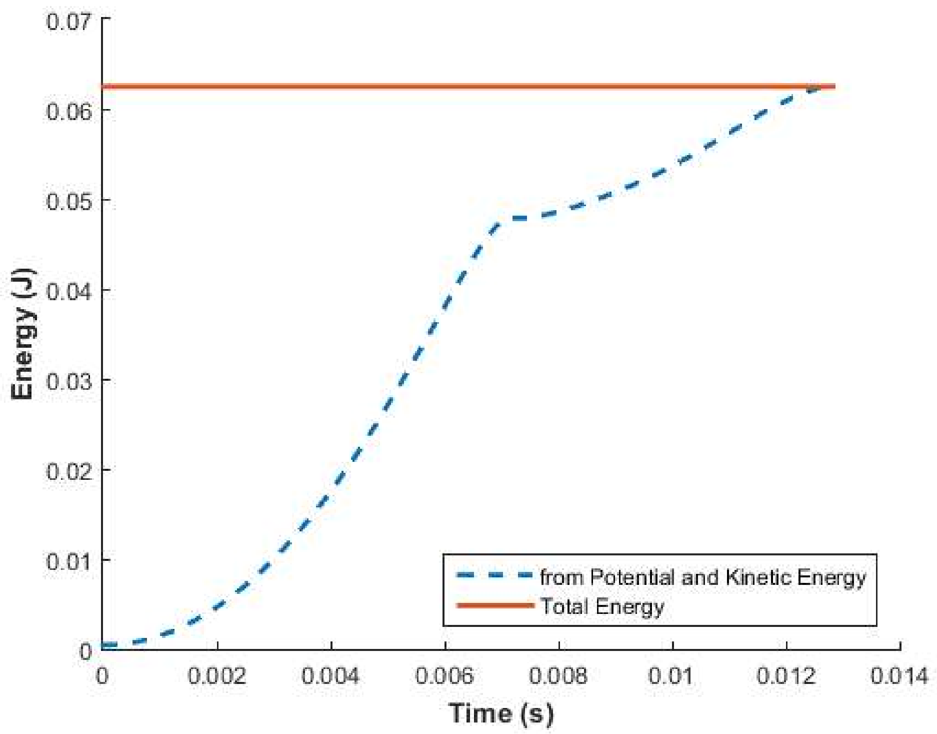
```
%x0 is the position of the end of the tibia
x0 = xt - t*cos(theta1);
y0 = yt - t*sin(theta1);
%x1 is the position of P
x1 = xt + t*cos(theta1);
y1 = yt + t*sin(theta1);
%x2 is the position of the body mass on top of the spider
x2 = xf + f*cos(theta2);
y2 = yf + f*sin(theta2);

%See draw.m
draw(time, x0, y0, x1, y1, x2, y2);
plot(x2,y2);
```



CHECK IF SOLUTION MAKES SENSE

```
%Conservation of Energy
%Note that the Pressure force and the mass of the body are EXTERNAL
 FORCES
%on our system. We must account for them as such.

%Energy at Start
PE_tibia = mt*g*yt;
PE_femur = mf*g*yf;

%Energy from EXTERNAL FORCES
%   Energy from Force of Pressure (Pressure Force * Distance Applied)
E_P = P*(max(sqrt(x1.^2 + y1.^2)) - min(sqrt(x1.^2 + y1.^2)));
%   Energy from Force of Body
E_body  = mb*g*(max(y2)-min(y2));

%Energy from Kinetic Energy
KE_tibia = 0.5*mt*(xtdot.^2 + ytdot.^2);
KE_tibia_rot = 0.5*It*omega1.^2;
KE_femur = 0.5*mf*(xfdot.^2 + yfdot.^2);
KE_femur_rot = 0.5*If*omega2.^2;

%Energy at Start + Total Energy from Pressure + Total Energy from Body
 == Potential Energy + Kinetic Energy
```

```
Etot_KEPE = KE_tibia + KE_tibia_rot + KE_femur + KE_femur_rot + ...
    + PE_tibia + PE_femur;

Etot_P = PE_tibia(1) + PE_femur(1) + E_P + E_body;

figure();
hold on;
plot(time, Etot_KEPE, '--', 'LineWidth', 2);
plot(time, Etot_P*ones(length(time)), 'LineWidth', 2);
xlabel('Time (s)', 'FontWeight', 'bold');
ylabel('Energy (J)', 'FontWeight', 'bold');
legend('from Potential and Kinetic Energy', 'Total Energy', ...
    'Location', 'southeast');

fprintf('The spider jumped a total x-distance of %f ', abs(x2(end)));
fprintf('\nThe spider jumped a total y-distance of %f ', max(y2));
fprintf('\nThe pressure was %f Pa', pP);
```



```
end
```

Published with MATLAB® R2015a

B.1 Phase 1 Equations

```
function thdot = rhs_phase1(tspan, th0)

global g mt mb mf It If t f d T F D P pP

%Unpack data
th1 = th0(1);
th2 = th0(2);
om1 = th0(3);
om2 = th0(4);
xt = th0(5);
yt = th0(6);
xf = th0(7);
yf = th0(8);
xtdot = th0(9);
ytdot = th0(10);
xfdot = th0(11);
yfdot = th0(12);

%Use A and b matrices derived from spider_DAEderive.m to solve for X
A = DAE_A1(If,It,d,f,mf,mt,t,th1,th2);
b = DAE_b1(P,d,f,g,mb,mf,mt,om1,om2,t,th1,th2);

%X contains all the data we need to solve for
X = A\b;

Ox = X(1);
Oy = X(2);
Hx = X(3);
Hy = X(4);
xtddot = X(5);
ytddot = X(6);
xfddot = X(7);
yfddot = X(8);
al1 = X(9);
al2 = X(10);

%Return data
thdot = [om1 om2 al1 al2 xtdot ytdot xfdot yfdot xtddot ytddot ...
    xfddot yfddot Ox Oy Hx Hy]';
end
```

Published with MATLAB® R2015a

B.2 Phase 2 Equations

```
function thdot = rhs_phase2(tspan, th0)

global g mt mb mf It If t f d T F D P pP

th1 = th0(1);
th2 = th0(2);
om1 = th0(3);
om2 = th0(4);
xt = th0(5);
yt = th0(6);
xf = th0(7);
yf = th0(8);
xtdot = th0(9);
ytdot = th0(10);
xfdot = th0(11);
yfdot = th0(12);

%Use A and b matrices derived from spider_DAEderive.m to solve for X
A = DAE_A2(If,It,d,f,mf,mt,t,th1,th2);
b = DAE_b2(d,f,g,mb,mf,mt,om1,om2,t,th1,th2);

%X contains all the data we need to solve for
X = A\b;

Hx = X(1);
Hy = X(2);
xtddot = X(3);
ytddot = X(4);
xfddot = X(5);
yfddot = X(6);
al1 = X(7);
al2 = X(8);

%Return data
thdot = [om1 om2 al1 al2 xtdot ytdot xfdot yfdot xtddot ytddot ...
    xfddot yfddot Hx Hy]';
end
```

Published with MATLAB® R2015a

B.3 Flight Event

```
function [value,isterminal,direction] = flight(t,th)

value = th(14);      %Find when this value (Oy) reaches 0
isterminal = 1;      %Stop solving when Oy reaches 0
direction = 0;       %Oy can reach 0 from + or - direction

end
```

Published with MATLAB® R2015a

B.4 Ground Event

```
function [value,isterminal,direction] = ground(t,th)

global f
value = th(8) + f*sin(th(2));   %Find when this value (y of body)
 reaches 0
isterminal = 1;                 %Stop solving when value reaches 0
direction = 0;                  %Can reach 0 from + or - direction

end
```

Published with MATLAB® R2015a

C Simulation

```
function draw(t, x0, y0, x1, y1, x2, y2)

hold on;
axis ([-0.075 0.075 -0.01 0.075]);
axis square;

ground = plot([-0.075 0.075], [0 0]);
init = 1;
start1 = plot([x0(init) x1(init)], [y0(init) y1(init)], 'LineWidth',
 2);
start2 = plot([x1(init) x2(init)], [y1(init) y2(init)], 'LineWidth',
 2);
line1 = plot([x0(init) x1(init)], [y0(init) y1(init)], 'LineWidth',
 2);
line2 = plot([x1(init) x2(init)], [y1(init) y2(init)], 'LineWidth',
 2);

for i = 1:length(t)

    line1.XData = [x0(i) x1(i)];
    line1.YData = [y0(i) y1(i)];
    line2.XData = [x1(i) x2(i)];
    line2.YData = [y1(i) y2(i)];
    drawnow;

end

xlabel('X Position (m)', 'FontWeight', 'bold');
ylabel('Y Position (m)', 'FontWeight', 'bold');
end
```

Published with MATLAB® R2015a

D Latch Design in MATLAB

```
%Calculation of Latch Force
clear all; clc;

%Material : VeroBlue RGD840
sig_ts = 55e6;                      %Tensile Strength (Pa)
max_strain = 0.2;                   %Elongation at Break (%)
E = 2500e6;                         %Modulus of Elasticity (Pa)
mu = 0.575;                         %Coefficient of Friction for ABS on
 ABS
                                    %Veroblue is NOT ABS but no data is
 given

%Parameters
y = 0.001;                          %Deflection (m), or Overhang
l = 0.0115;                         %Length (m)
h = 0.002;                          %Thickness at Root (m)
b = 0.002;                          %Width at Root (m)
alpha = 15*pi/180;                  %Mate Angle
alphap = 55*pi/180;                 %Unmate Angle

%Calculations
eps = y*h/l/l/1.09;                 %Strain value we want
fprintf('Strain is = %6.2f%% \n', eps*100);

P = (b*h*h/6)*(E*eps/l);            %Deflection Force
fprintf('The deflection force is = %6.2f N \n', P);

W = P * (mu+tan(alpha)) / (1-(mu*tan(alpha)));       %Mating Force
fprintf('The mating force is = %6.2f N \n', W);

Wprime = P * (mu+tan(alphap)) / (1-(mu*tan(alphap)));       %Demating
 Force
fprintf('The demating force is = %6.2f N \n', Wprime);
```

Published with MATLAB® R2015a

E Latch Testing with Python

```
#! /usr/bin/python

"""Copyright 2011 Phidgets Inc.
This work is licensed under the Creative Commons Attribution 2.5 Canada License.
To view a copy of this license, visit http://creativecommons.org/licenses/by/2.5/ca/
"""

__author__="Adam Stelmack"
__version__="2.1.8"
__date__ ="14-Jan-2011 2:29:14 PM"

#Basic imports
import sys
from time import sleep
import time
#Phidget specific imports
from Phidgets.PhidgetException import PhidgetException
from Phidgets.Devices.Bridge import Bridge, BridgeGain
from Phidgets.Phidget import PhidgetLogLevel

#Start a timer to create timestamps
initial_time = time.time()
data_file = open("data_%s.csv" % (time.strftime('%Y%m%d-%H%M%S')), 'w')

#Create an accelerometer object
try:
    bridge = Bridge()
except RuntimeError as e:
    print("Runtime Exception: %s" % e.details)
    print("Exiting....")
    exit(1)

#Information Display Function
def displayDeviceInfo():
    print("|------------|----------------------------------|--------------|------------|")
    print("|- Attached -|-              Type              -|- Serial No. -|-  Version -|")
    print("|------------|----------------------------------|--------------|------------|")
    print("|- %8s -|- %30s -|- %10d -|- %8d -|" % (bridge.isAttached(), bridge.getDeviceNam
    print("|------------|----------------------------------|--------------|------------|")
    print("Number of bridge inputs: %i" % (bridge.getInputCount()))
    print("Data Rate Max: %d" % (bridge.getDataRateMax()))
    print("Data Rate Min: %d" % (bridge.getDataRateMin()))
    print("Input Value Max: %d" % (bridge.getBridgeMax(0)))
    print("Input Value Min: %d" % (bridge.getBridgeMin(0)))

#Event Handler Callback Functions
def BridgeAttached(e):
    attached = e.device
    print("Bridge %i Attached!" % (attached.getSerialNum()))

def BridgeDetached(e):
    detached = e.device
    print("Bridge %i Detached!" % (detached.getSerialNum()))

def BridgeError(e):
    try:
        source = e.device
```

F Pressure Testing with Arduino (added Spring 2017, lc848)

```
#! /usr/bin/python

"""Copyright 2011 Phidgets Inc.
This work is licensed under the Creative Commons Attribution 2.5 Canada License.
To view a copy of this license, visit http://creativecommons.org/licenses/by/2.5/ca/
"""

__author__="Adam Stelmack"
__version__="2.1.8"
__date__ ="14-Jan-2011 2:29:14 PM"

#Basic imports
import sys
from time import sleep
import time
#Phidget specific imports
from Phidgets.PhidgetException import PhidgetException
from Phidgets.Devices.Bridge import Bridge, BridgeGain
from Phidgets.Phidget import PhidgetLogLevel

#Start a timer to create timestamps
initial_time = time.time()
data_file = open("data_%s.csv" % (time.strftime('%Y%m%d-%H%M%S')), 'w')

#Create an accelerometer object
try:
    bridge = Bridge()
except RuntimeError as e:
    print("Runtime Exception: %s" % e.details)
    print("Exiting....")
    exit(1)

#Information Display Function
def displayDeviceInfo():
    print("|------------|----------------------------------|--------------|------------|")
    print("|- Attached -|-              Type              -|- Serial No. -|-  Version -|")
    print("|------------|----------------------------------|--------------|------------|")
    print("|- %8s -|- %30s -|- %10d -|- %8d -|" % (bridge.isAttached(), bridge.getDeviceNam
    print("|------------|----------------------------------|--------------|------------|")
    print("Number of bridge inputs: %i" % (bridge.getInputCount()))
    print("Data Rate Max: %d" % (bridge.getDataRateMax()))
    print("Data Rate Min: %d" % (bridge.getDataRateMin()))
    print("Input Value Max: %d" % (bridge.getBridgeMax(0)))
    print("Input Value Min: %d" % (bridge.getBridgeMin(0)))

#Event Handler Callback Functions
def BridgeAttached(e):
    attached = e.device
    print("Bridge %i Attached!" % (attached.getSerialNum()))

def BridgeDetached(e):
    detached = e.device
    print("Bridge %i Detached!" % (detached.getSerialNum()))

def BridgeError(e):
    try:
        source = e.device
        print("Bridge %i: Phidget Error %i: %s" % (source.getSerialNum(), e.eCode, e.descri
    except PhidgetException as e:
        print("Phidget Exception %i: %s" % (e.code, e.details))

def BridgeData(e):
    source = e.device
    #print("Bridge %i: Input %i: %f" % (source.getSerialNum(), e.index, e.value))
    #print("%f, %f" % (time.time()-initial_time, e.value))
    data_file.write("%f, %f \n" % (time.time()-initial_time - 4.05, e.value))
#Main Program Code
try:
     #logging example, uncomment to generate a log file
    #bridge.enableLogging(PhidgetLogLevel.PHIDGET_LOG_VERBOSE, "phidgetlog.log")

    bridge.setOnAttachHandler(BridgeAttached)
    bridge.setOnDetachHandler(BridgeDetached)
    bridge.setOnErrorhandler(BridgeError)
    bridge.setOnBridgeDataHandler(BridgeData)
except PhidgetException as e:
    print("Phidget Exception %i: %s" % (e.code, e.details))
    print("Exiting....")
    exit(1)

print("Opening phidget object....")

try:
    bridge.openPhidget()
except PhidgetException as e:
    print("Phidget Exception %i: %s" % (e.code, e.details))
    print("Exiting....")
    exit(1)

print("Waiting for attach....")

try:
    bridge.waitForAttach(10000)
except PhidgetException as e:
    print("Phidget Exception %i: %s" % (e.code, e.details))
    try:
        bridge.closePhidget()
    except PhidgetException as e:
        print("Phidget Exception %i: %s" % (e.code, e.details))
        print("Exiting....")
        exit(1)
    print("Exiting....")
    exit(1)
else:
    displayDeviceInfo()

try:
    print("Set data rate to 8ms ...")
    bridge.setDataRate(16)
    sleep(2)

    print("Set Gain to 8...")
    bridge.setGain(0, BridgeGain.PHIDGET_BRIDGE_GAIN_8)
    sleep(2)
    print("Enable the Bridge input for reading data...")
    bridge.setEnabled(0, True)
    sleep(2)

except PhidgetException as e:
    print("Phidget Exception %i: %s" % (e.code, e.details))
    try:
        bridge.closePhidget()
    except PhidgetException as e:
        print("Phidget Exception %i: %s" % (e.code, e.details))
        print("Exiting....")
        exit(1)
    print("Exiting....")
    exit(1)

print("Press Enter to quit....")

chr = sys.stdin.read(1)

print("Closing...")

try:
    print("Disable the Bridge input for reading data...")
    bridge.setEnabled(0, False)
    sleep(2)
except PhidgetException as e:
    print("Phidget Exception %i: %s" % (e.code, e.details))
    try:
        bridge.closePhidget()
    except PhidgetException as e:
        print("Phidget Exception %i: %s" % (e.code, e.details))
        print("Exiting....")
        exit(1)
    print("Exiting....")
    exit(1)

try:
    bridge.closePhidget()
except PhidgetException as e:
    print("Phidget Exception %i: %s" % (e.code, e.details))
    print("Exiting....")
    exit(1)
data_file.close()
print("Done.")
exit(0)
```

G Balloon Joint Assembly Procedure (added Spring 2017, lc848)

Compiled 5/22/17 by Lawrence Chen

Arachnabot Fabrication Methodology

The following is a summary of the steps and processes used to fabricate the Arachnabot leg as pictured in the report.

List of necessary materials

1. 3D Printed Components
    a. x1 TCJ Lower Leg.stl
    b. x1 TCJ Lower Retainer.stl
    c. x1 TCJ Upper Leg.stl
    d. x1 TCJ Upper Retainer.stl
    e. x1 TopHat
2. 260 or 160 type balloon, cut to ~40mm in length
3. Ribbon mesh cut to ~40mm x ~35mm
4. x2 ∅1.5mm x 14mm pins
5. Super glue (Loctite 431 was used for these assemblies)

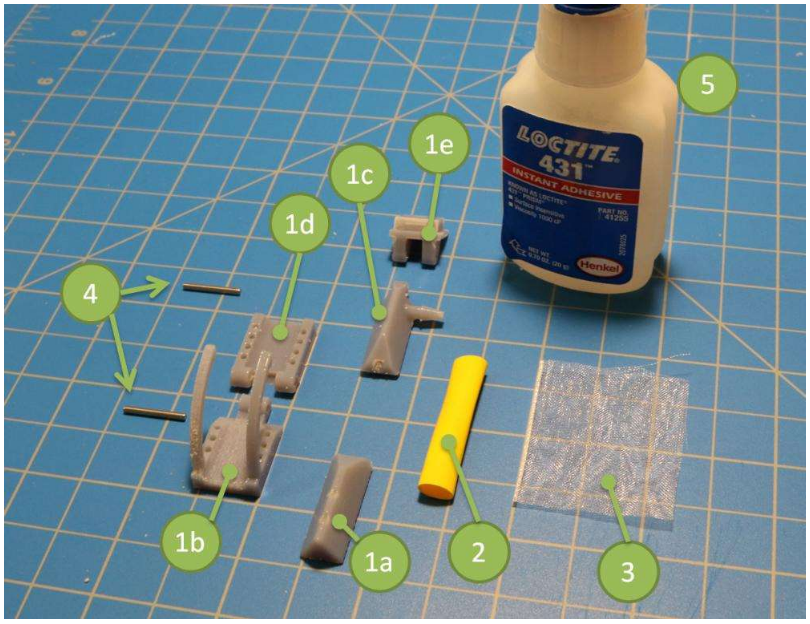


Figure 1: Materials needed to fabricate the leg assembly.

List of tools

- Sharp knife or scissors

Procedure

1. Insert the lower leg into the balloon until the balloon covers up the entirety of the sloped face

2. Insert the upper leg into the balloon until the balloon covers up the entirety of the sloped face. Prior to insertion, verify that the air channel is completely vacated and that air can flow freely from the nipple through the orifice. The final length of the assembly should be roughly 64mm long. Verify that there is no damage to the balloon's surface to avoid the possibility of air leaks in this region.

3. Apply superglue around the circumference of the balloon border, making sure to cover all of the edges of the balloon as this will be the primary air-sealing surface.

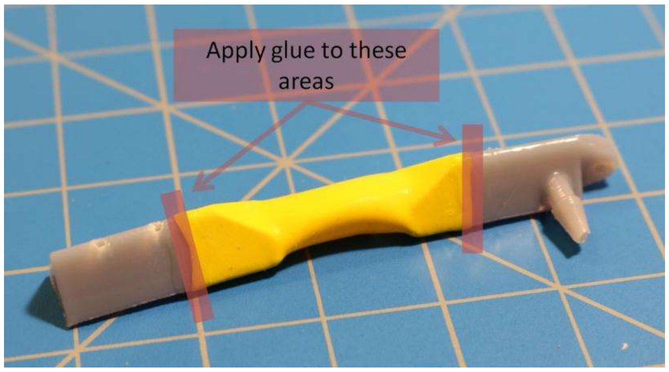


Figure 2: Areas to apply glue to bond the balloon to the leg.

4. After waiting for the super glue to cure, test the air sealing capabilities of the joint by pumping air into the leg. A syringe with some tubing works best. For best verification of air leaks, submerge the assembly into water and see if air bubbles form around any surfaces when pressure is applied.

5. After verification of the air-tight seal for the balloon joint, glue the edge of the ribbon to the flat surface on the back of the joint. It is not recommended to apply glue where there is only flexible material, as this will affect the joint's ability to expand and contract freely. It is important to keep the fabric weave along and perpendicular to the leg. Diagonal weave patterns have a harder time preventing the hyper-elastic behavior and will cause the leg to shorten when the joint expands.

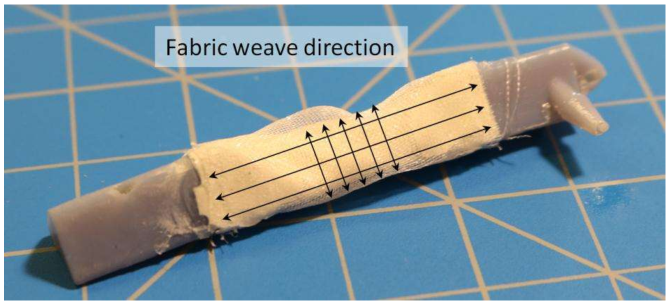


Figure 3: Depiction of fabric weave direction for best performance.

6. Wrap the ribbon around the joint, applying super glue where the upper and lower leg parts are under the ribbon/balloon. Trim excess ribbon material, being careful not to cut the underlying balloon.

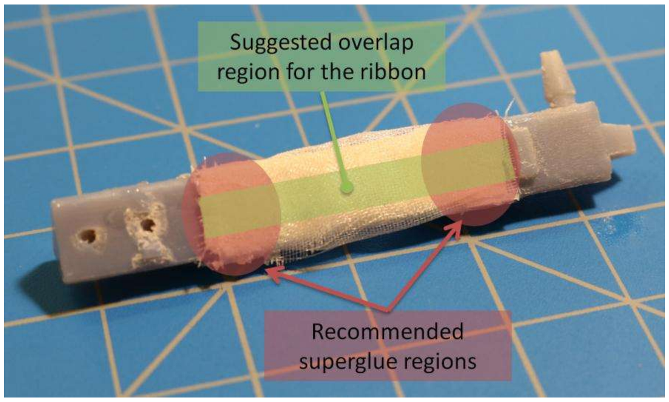

Suggested overlap region for the ribbon
Recommended superglue regions

Figure 4: Guide to adhering the fabric weave to the assembly.

7. Align the tophat hinge hole to the upper leg hinge hole, and insert a pin through these components. The pin is oversized and will have some excess length, but this does not impact the performance of the leg.

8. Align the upper retainer and the lower retainer components then slowly insert a pin through the hinge. The hinge hole may require some drilling, as the interference fit with the lower joint is very snug. The pin is installed when it bottoms out and reaches the other side.

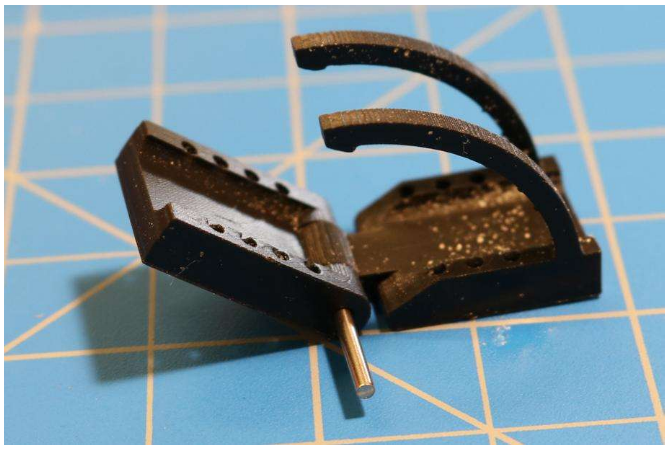


Figure 5: Pin insertion for the hinge.

9. Glue the upper leg to the upper retainer such that the middle of the joint and the middle of the retainer-hinge assembly are aligned. Bend the joint and verify that the leg can latch, observing where the lower leg must be fixed to allow for proper leg actuation. If the lower leg is too close

to the hinge, full leg extension is not possible, or the excess balloon material will prevent the leg from latching completely.

10. Glue the lower leg piece to the retainer-hinge assembly per the test fit in the previous step.

Testing the Assembly

1. Attach a small hose to the barbed fitting on the leg. Attach the other end of the hose to a syringe.

2. Latch the joint together.

3. Pressurize the assembly and see if the leg jumps/releases.

Troubleshooting the Leg Assembly

- If the leg does not release check the following…
    - Verify that air is not leaking when pressure is applied. If a leak is found near a static component (i.e. ontop of the upper leg piece) then super glue can be applied to attempt to remedy. If a leak is found in the latex material, then the joint must be re-assembled
    - Verify that the balloon inflates when pressure is applied. If it does not, then either the leg joint is clogged with support material, or there is a leak/crack in the leg.
- If the leg is not latching completely…
    - Check that the bladder material is not being squeezed when the joint is compressed. If this is the case, the leg must be reassembled, but with a longer piece of balloon.
    - Check the condition of the latch ramp and the upper retainer to make sure that the features have been resolved as some printers may not reliably do this.
- If the leg is not unlatching when pressure is applied…
    - If the design has a 90° contact between the latch and the top retainer, then a small angle needs to be applied or else the latch ramp will likely break or snap off.
    - Measure the pressures being applied to see if they are high enough