Picking the right encoder interface is often forced by the controller it hooks up to and the simple digital quadrature interface is by far the most common for relative encoders. This interface simply pulses at the rising and falling edge of the encoder lines, leaving counting to the user. An absolute encoder needs to send a unique signal depending on where the code wheel is, so they often use parallel or serial connections that are much more complex. They are also often available with analog voltage out, as though it were a potentiometer, or with a variable duty cycle pulse width modulated signal. These two options, while easier to interface with, have large issues in most cases. The analog connection is susceptible to noise, an encoder with 1000 counts that makes a 5V max signal will have signal all the way down to 5mV, and that's a very low count encoder. Pulse width modulated signals are limited to the balance of update rate and clock accuracy. The faster the encoder sends updates the less time it has for the period of its signaling square wave. A 1000 count encoder updating at 1000hz would require 1 microsecond clock resolution to read. On the plus side, both of these interfaces only require a single signal wire and a ground, this may be very appealing if trying to work through a slip ring.

The encoder we ended up using is the AEDA-3300 from Avago. This encoder offers a unique combination of a large variation of available tick counts, 2400 to 80000 counts per revolution in a very small lightweight, and inexpensive package with its own integrated bearings. This means that the sensor can be used in almost any application on a robot which makes it a great part to design around. The only thing it doesn't offer is its own housing, but that is easy to deal with after the fact.

2.1.4 Terrain Sensing

Figuring out the angles of the robot's links is only half the battle in establishing the state of the whole system. It's vitally important to know which leg is the stance leg and what the world around the robot looks like. Depending on which leg is the stance leg the direction that torque needs to be applied in changes sign because of of how the chain is connected to the ground. One very important facts we learned early in experimenting with walking controllers on the robot is that thinking you're on the wrong stance leg puts many controllers into positive feedback, meaning that the actuator applies the limiting torque almost instantly, completely ruining any control that had been going on before things went wrong. We'll see later on that this has implications for time indexed controllers.

The stance leg detector went through multiple iterations depending on the hardware configuration of the robot. The initial plan was to have the series elastic actuator toes read the amount of force on them, expecting the robot to always have more weight on the leg it's standing on and so should provide a reliable indication of the stance leg. This turned out to be a bad assumption, not because that statement is false, but because the weight of the robot under static conditions doesn't actually move the springs in the feet. The stiffness of springs and amount of preload required to make the toe action not too spongy under dynamic conditions turns out to make the static readings from the sensors useless. This was overcome by looking at impacts instead of static conditions. Any impact, even very small, produces distinct readings from the load cells which provide the indication that the stance leg has changed. Knowing which leg the robot was previously on makes this enough information to determine the stance leg throughout time, but this strategy isn't tolerant of errors, an errant impact detection can make the stance leg decision wrong for a long time. There's a little bit more information available though, that's an occasional indication of which leg the robot is on under dynamic conditions in mid step. Even though static conditions aren't enough to exercise the springs the robot often undergoes accelerations without impacts that are enough to trigger a positive stand leg identification. If this information is used to affirm the stance leg choice from the impact detector then the wrong leg is chosen extremely rarely, in fact once this strategy was implemented the only errors ever made were from hardware failures.

A Hokuyo UTM-30LX scanning laser rangefinder is also used to sense terrain. It's mounted in plane with the robot's walking plane, making the single scan line the sensor produces capable of describing all relevant terrain to the robot. While originally intended for the purpose of picking up rough terrain the sensor ended up also being extremely useful on flat ground. The small amount of drift the IMU picks up is simple to eliminate using the range estimates from the laser scanner if most of the ground is known to be flat which is the case in many of our experiments. The sensor provides millimeter resolution and repeatability up to 10 meters away and scans in 0.25 degree steps at 40Hz providing a deluge of scan points which a line can be fit to to produce a ground estimate. Knowing the laser scanner provides absolute truth but at a slow rate compared to the IMU, this data combined with the IMU attitude estimate in the robot's state observer using a slow zeroing filter. The zeroing filter maintains a state which is the difference between the two raw sensor readings after being passed through a first order low pass filter with a time constant on the order of several seconds.

2.2 Mechanical Design

2.2.1 Feet

The robot's feet serve a few purposes. The simple compass gait model assumes that the swing leg has some way of avoiding hitting the ground as it swings through, so some way of making the legs shorter during swing is necessary. In addition to that, it could be helpful if the same mechanism could be used to push off from the ground to add energy or if the actuators could do some crude ground speed matching in order to slow down the dynamics of the impact.

Not much of an actuator is required if the only job to accomplish is shortening the leg during the swing phase. Previous iterations of compass gait robots at the lab have used linkages with weak but fast motors which allow the link to extend and retract quickly and lock into place in the extended position, but this strategy doesn't work when the actuator is used to actually apply a force to the rest of the robot. A good alternative to the locking linkage is a lead screw which also has a resistance to backdriving, but doesn't have the same nonlinearities of the four bar linkage, it's able to operate in the same way at point in its range.

Another design consideration which lends itself to the lead screw is the integration

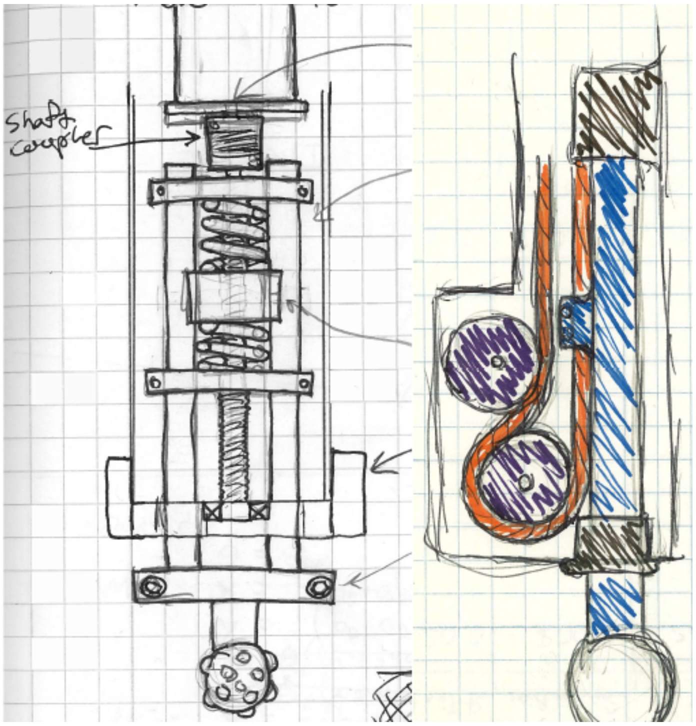


Figure 2-7: Concept drawings for the robot's toe actuators. Lead screw design on the left and cable drive on the right.

of a series elastic element. The series elastic actuator has been a popular robot design element in recent years because it allows force output from actuator that work primarily in position like lead screws. Of more importance here is ability to bring the contact dynamics of interaction with the world outside the robot into the robot's actuator [16]. By making the designed elastic element in the robot joint much more compliant than the ground contact itself the dynamics of the ground contact are brought into the actuator itself. This is a huge advantage because it means the known compliance of the actuator dominates the impact and the hard to model dynamics of the contact can be neglected. It also means that the collision is much more inelastic, much like a car's suspension system is designed to have a small mass on the end of the spring to maintain contact with the road, the toe has a small mass on the end of the spring which means the weight of the robot forces it to stay in contact with the ground.

Because the ground contact force can only ever be applied in one direction (the ground will never pull the foot) only a one-sided series elastic element is required

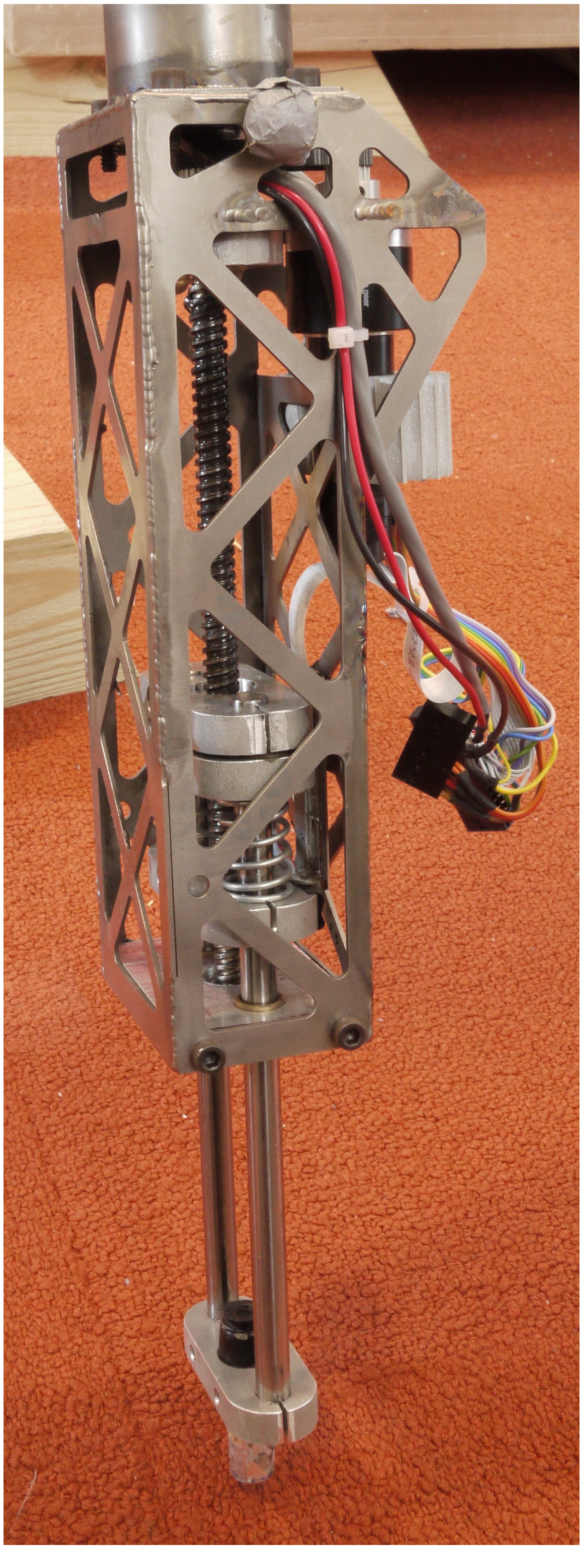


Figure 2-8: The robot's series elastic foot actuator.

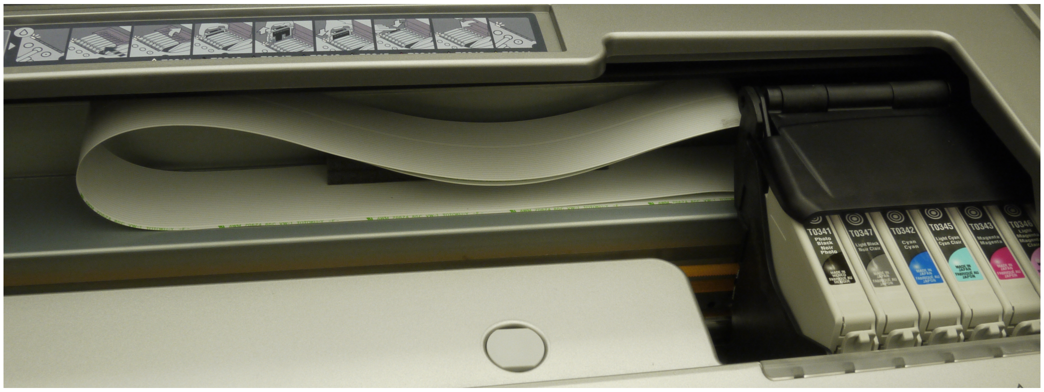


Figure 2-9: The cable which connects to the moving carriage in an inkjet printer was the design inspiration for the similar mechanism in the robot's feet.

which saves a significant amount of room in the mechanism. While the mechanism worked well it turned out to be a bad decision in hindsight because the hard contact on one side means that an inelastic collision is experienced every time the foot force transitions over the spring preload making the dynamics model unnecessarily complex.

The picture of one of the feet in Figure 2-8 shows a few key design elements worth discussion. The linear potentiometer which measures the spring compression can be seen parallel to the spring and the ribbon flex cable which connects it to the controller board is the flat white cable connected to it. Getting that cable right was one of the major design challenges of the actuator because the carriage which holds the potentiometer translates a long distance. Either a full cable carriage is required or the cable must somehow constrain itself be planar and fold over itself reliably. The design is borrowed from what is commonly used in inkjet printers which need to solve exactly the same problem with their moving carriage. A flat, stiff ribbon is used which maintains itself in plane but is flexible out of that plane, allowing it to fold over itself as the carriage moves.

The offset motor configuration is important for a reason besides packaging of the actuator, it allows a timing belt to be inserted between the motor output and the lead screw. This is critical because the specific loading and velocity requirements of the foot weren't known at the time of construction, the timing belt allowed the gear ratio be modified very easily to put the actuator's operating range in the right place once testing established where that was. The same design decision is behind the way the aluminum platens are clamped onto the guide shafts instead of using a more positive locking mechanism, it allows different length springs to be added after the fact and the amount of preload to be easily changed.

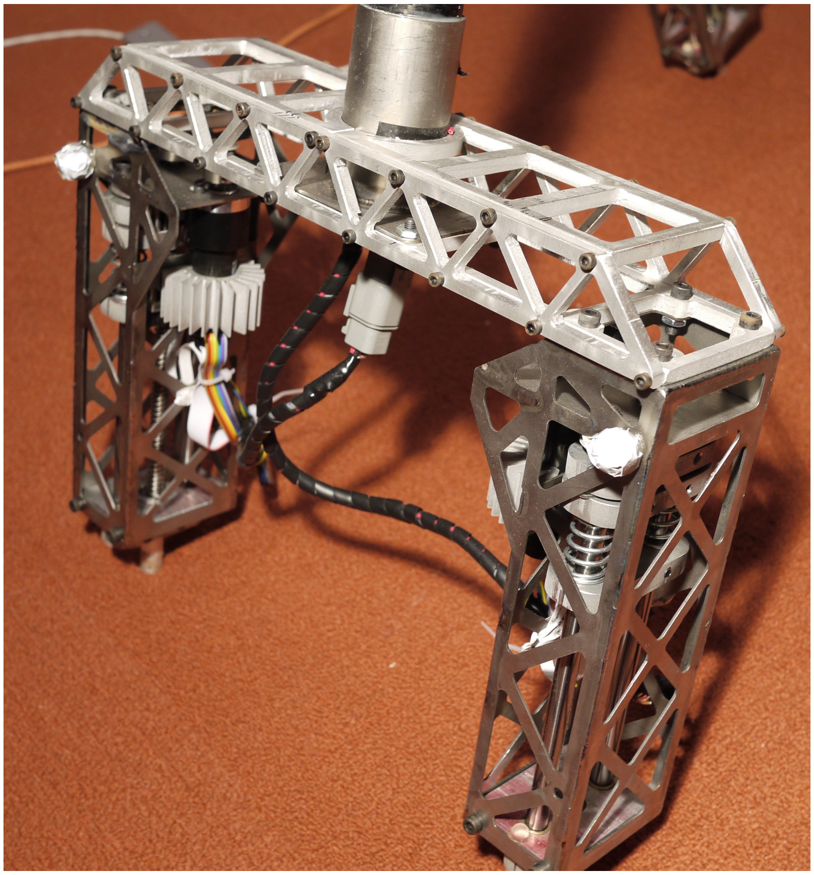


Figure 2-10: The two middle feet branching off from the single middle leg.

The final main design point on the feet is why there are four of them. The original reason here was that it's simpler to build four of the same actuators than two of the same and one different because it carries twice the loading of the other two. Later on a more important reason emerged, stabilization of side to side motion. Originally the two middle feet were joined back to back but early in testing it became apparent that a larger stance distance was required to keep the robot solidly stable in that direction. This is why the spreading truss pictured is aluminum rather than the titanium sheet construction of the rest of the robot, it was produced after all of the expensive titanium sheet had been used up. With the feet further spaced apart as

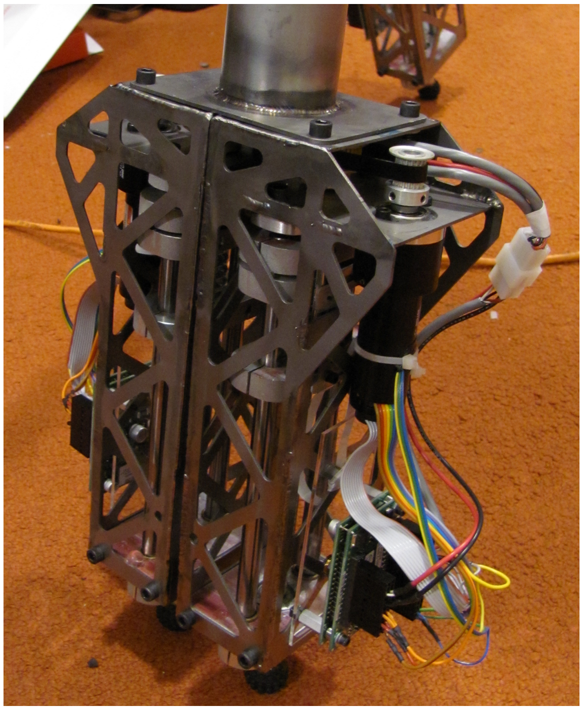


Figure 2-11: The original foot configuration where the middle two were bolted back to back.

pictured that motion is no longer a problem.

2.2.2 Body and Bisection Mechanism

The body itself is probably the simplest part of the robot. Its only job is to hold all of the robot's support systems together in an organized fashion. The main interesting aspect to it is the angle bisecting mechanism whose job it is to keep the angle of the body parallel to a bisector of the inter-leg angle. While it sounds a little complicated, it's surprisingly easy once it's noted that the body is mounted to the frame of the outer leg and the driveshaft for the inner leg is easily accessible. These two can be driven against each other with some drive system and gear ratio to produce any kind of prescribed angle with relation to the two legs. A 2 : 1 drive ratio between the links produces the desired angle bisecting behavior.

As shown in 2-12 this is accomplished with a tensioned cable drive. While this configuation was more difficult to design and assemble than alternatives such as a chain drive the big advantage is that it's possible to completely eliminate backlash and produce a very smooth drive system.

While the prescribed angle mechanism is straightforward and reliable there are serious advantages to being able to actuate the body against the rest of the robot. The large mass of the robot's support equipment (computer, batteries, etc) makes it a great body to actuate against to assist in regulating the motion of the legs. In addition to this, changing the forward or backward bias of the body while largely keeping it bisecting the inter-leg angle can change the passive dynamics of walking gaits, producing faster and slower walks without the cost of spending energy on another full actuator.

2.2.3 Hip Actuation

The robot's hip actuator, which applies torque between the legs, is easily the most important actuator on the robot as it provides almost all of the regulation during a walking gait. Ideally this actuator would have great bandwidth, zero backlash, unlimited speed, be able to provide more torque than we would ever be able to safely use, and be lightweight. Stepping away from the ideals, we can limit the requirements on speed to what would be reasonable for the leg swinging during a running gait. As for the torque requirements, they were put together with the idea that that robot should be able to easily lift one leg to a right angle, based on the somewhat arbitrary initial weight budget.

Initial concepts were made around using a direct drive actuator at the hip. Direct drive actuators provide the ultimate in bandwidth, torque accuracy, zero backlash, and friction characteristics because they completely forgo gearboxes, using the elements of an electric motor to move the robot's links directly. This means that the passive dynamics of the robot are easy to maintain and manipulate with little energy input. It's possible to mimic the desired passive dynamics with a motor and high ratio gearbox up to some limiting frequency range, but requires a lot of energy to keep the motor following what the passive dynamics want to do. Most robotics work up to this point is focused on completely ignoring the passive dynamics of the system and imposing desired dynamics with some energy efficiency which is what the large

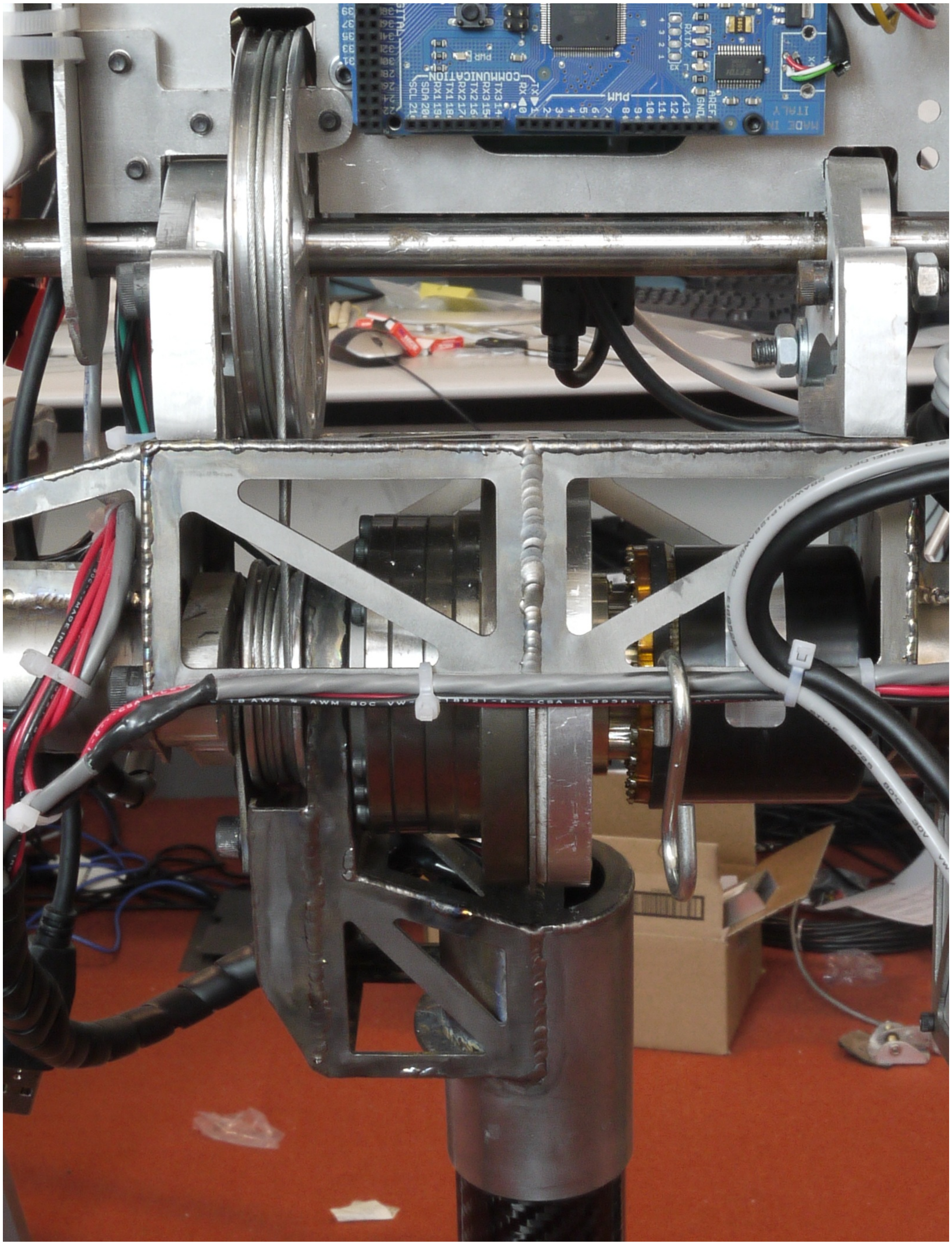


Figure 2-12: The body's angle bisection mechanism.

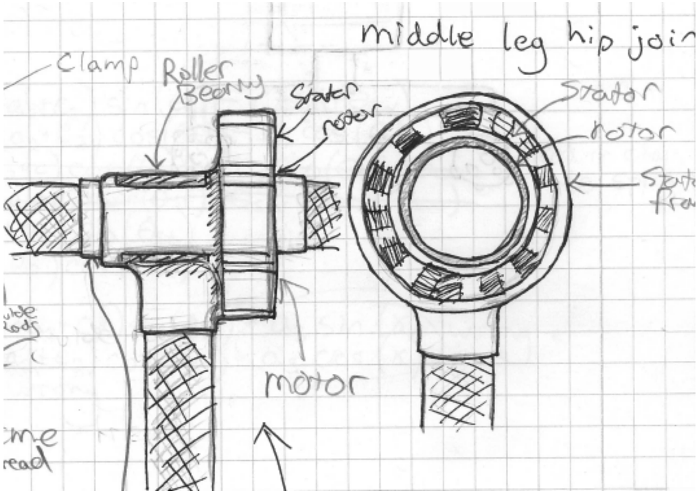


Figure 2-13: Initial concept for a direct drive hip actuator using a frameless motor and our own aluminum frame. This allows the motor to be built into the robot in the most efficient way possible.

gearboxes traditional to robotics excel at. A full discussion of the topic is available in [1].

The big problem with direct drive actuators is that to get torques in the range that we require, around $30N-m$ the actuators get to be unfeasibly large and heavy because they need a large radius to apply the small electromagnetic forces generated on. Meeting somewhere in the middle on this design issue is difficult because almost all gearboxes have some backlash inherent to them, something that we really wanted to avoid based on past experience with the acrobot. One gearbox that doesn't suffer from backlash is the Harmonic Drive. Typically Harmonic Drive gearboxes are the domain of very high gear ratios and the complete antithesis of a passive dynamic actuator, but with some creativity a very acceptable compromise between direct drive and weight concerns was found.

The operating principles behind the harmonic drive are very different from traditional gearboxes, involving flexible metal gears that deform elastically. The fact that the gears deform elastically into each other means that the input and output are always tightly meshed together. A full explanation of the operating principles, along with a very helpful animation is available from the producer [12]. Normally the Harmonic Drive isn't considered to be backdrivable, the large torques at the output

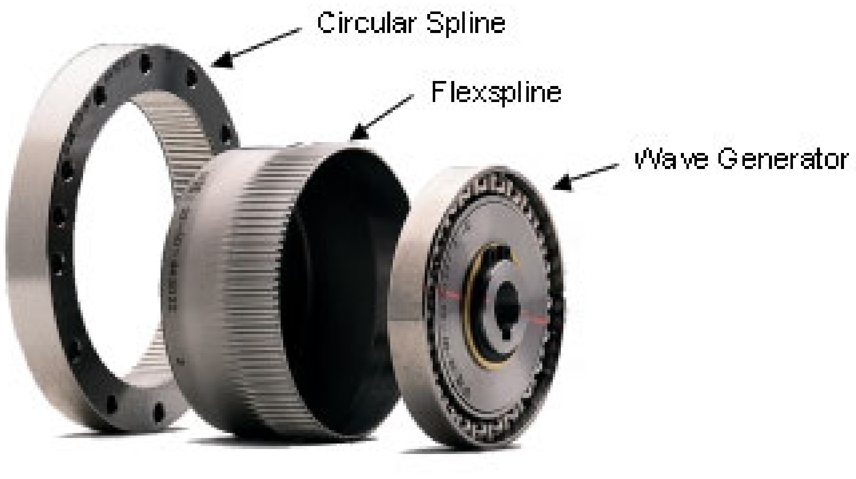


Figure 2-14: Explosion of a Harmonic Drive gearbox from the Harmonic Drive operating principles literature [12].

required to break free even the tiny amount of friction at the input would cause the very small teeth on the flex spline to skip and permanently damage the drive. In the case of the lowest gear ratios available however, two things work in favor of backdrivability. First is that with a small ratio the friction at the input is multiplied by a much smaller amount (as it would be with any gearbox), but more importantly the teeth inside a low ratio Harmonic Drive are very large. This means that the drive is much more robust to large torques at the output and suffers from less friction internally. The lowest ratio drive of 30 : 1 was selected for this application, opening up a large selection of small, high specific torque motors to use.

Another note of interest about the gearbox used is that it's a fully contained unit including an output bearing. The output bearing is of the cross roller type and is tightly loaded in order to minimize play in the output. This is because the drive is designed for having cantilevered loads applied to it, such as fully supporting a robot arm, but has the unfortunate effect of vastly increasing the amount of friction at the drive output. Initial tests with the gearbox were very disappointing because very large and unpredictable torques were required to break the leg free and very little of the desired passive dynamics were exhibited. Because the robot's leg joint is double support thanks to the bearing opposite the gearbox this bearing doesn't need to be nearly as tight, so the gearbox output bearing was modified by shimming

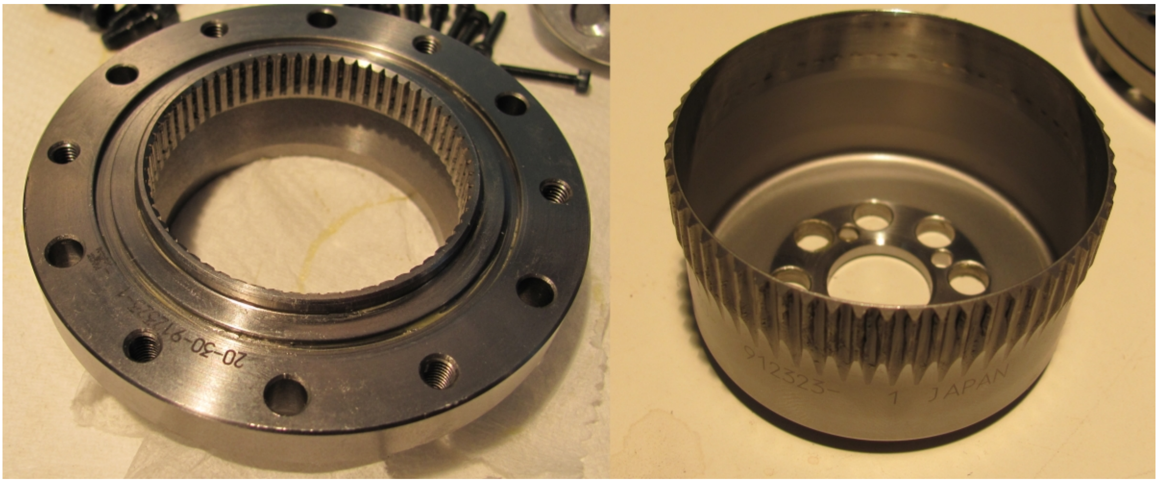


Figure 2-15: The 30 : 1 ratio Harmonic Drive gearbox used in this robot, showing the very large driving teeth. Overall diameter of the flex spline (right) is about 2.5 inches.

out the bearing races about 0.001 inches. This small change turned the gearbox from disappointing to better than we ever expected, exhibiting very small and very predictable static and viscous friction characteristics.

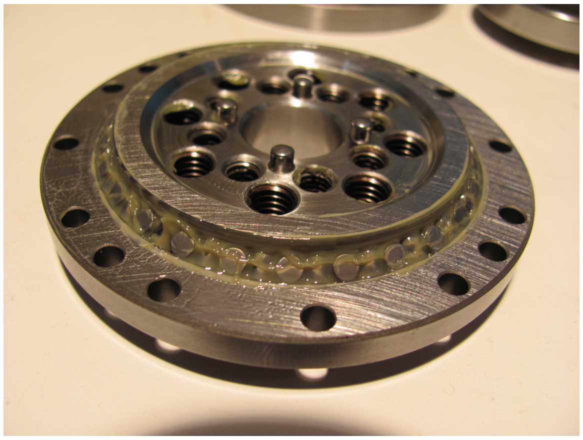


Figure 2-16: The cross roller bearing inside the robot's Harmonic Drive gearbox.

The motor paired with the gearbox is a ThinGap TG2310 brushless, ironless DC motor. This choice is just as notable as the gearbox because the ThinGap motor exhibits the best specific torque characteristics available [21] at moderate speeds and the ironless core means that the motor doesn't exhibit any of the cogging that normal brushless DC motors have. The moderate speed point is important because with the gearbox the motor is no longer moving extremely slowly. These characteristics are due to a new method of producing the windings from copper sheet rather than wire. This motor paired with the aforementioned gearbox produces a hip actuator capable of exerting over 30N-m of force at high speeds very accurately and without any backlash.

The big disadvantage to the gearbox is that the Haromic Drive introduces a series compliance with the motor. This is due to the thin structure of the flex spline, the part of the gearbox that deforms to make the magic of the device. While this isn't noticeable most of the time, many of control experiments excited the lightly damped (the motor side of the drive system has very little friction) high frequency dynamics that this introduced. This can be handled in software and is discussed in Chapter 3.

2.2.4 Frame

The robot's frame, while the most visible part of the robot, is one of the less important parts of the whole system. It serves mostly to locate all of the important actuators, sensors, and mechanisms in space in a reliable, lightweight manner. Once it's known what goes where and how much force will be applied between these pieces the required structural properties can be determined and fulfilled. Rather than spend a lot of time going over this process, I'd like to mention just a couple important design choices and lessons learned.

The frame is a fabricated sheet metal structure. This decision has more to do with efficiency of resources than performance of the robot. It's much cheaper than cutting the large three dimensional structures from solid pieces of material and we have better equipment for working with sheet metal parts in-house. The waterjet available at the lab allows arbitrarily complex sheet metal parts to be cut with ease and with little material waste. This means that the robot's frame can be prototyped quickly from an expensive high performance material very quickly and at little cost. Besides the waterjet the main enabling resource for this path is access to and skill to use a tungsten inert gas (TIG) process welder. The TIG process allows very high quality welds to be made with almost all structural metals, the main barrier to use being the high manual skill required to perform it.

Following the construction method, the second piece is the material choice. All of the frame pieces besides the legs are made from 6AL-4V titanium for a couple unusual reasons. There are three common materials for structures such as this: steel, aluminum, and titanium. All three materials have similar specific strength (yield strength divided by density) and specific modulus (modulus of elasticity divided by density).

Much of the robot's frame construction is governed by the minimum thickness of material that can be used. Even though a part could be made exceedingly thin according to the predicted loading it's often unwise to do so because the bumps and scratches of everyday use could damage it and sheet thinner than about 0.035 inches thick is difficult to weld by hand reliably. This means a low density material is desired, leaving aluminum and titanium.

The main problem with using aluminum is that it's difficult to produce good quality welds with it in a prototyping situation. The high thermal conductivity of the metal makes it necessary to use very large electric currents to weld it because it draws heat away from the weld site so effectively. When welding the material the portion of the workpiece just on the edge of melting is much larger than with steel or titanium, requiring more manual dexterity to manage the heat input and torch movement.

Titanium on the other hand has thermal characteristics much closer to steel and is very simple to weld except for one very big caveat. The material pulls in atmospheric contaminants very easily at the temperatures involved with welding causing serious embrittlement problems. Special care must be taken to fully shield much more of the workpiece in a pure argon atmosphere than with steel or aluminum which still experience contamination issues, but to a much smaller extent.

The specific titanium alloy used is 6AL-4V, otherwise known as Grade 5. It has a good balance of stiffness and weldability, but more importantly, it's the most commonly used alloy. This means that it's widely available on the surplus market at reasonable prices.

The reason why the legs are carbon fiber as opposed to titanium is because they're

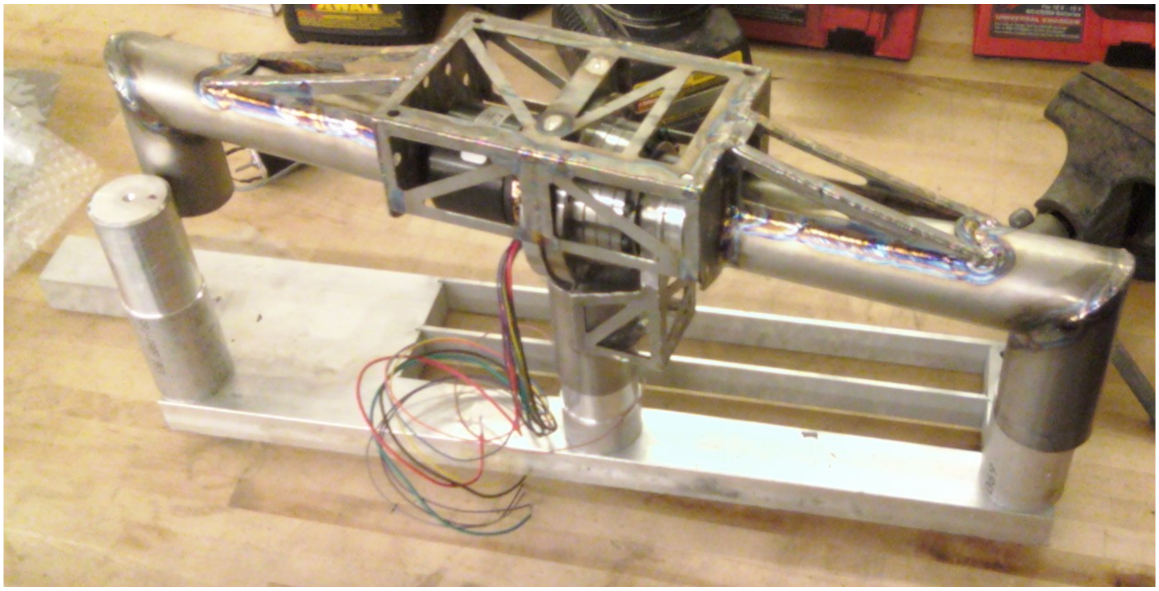


Figure 2-17: The robot's titanium sheet metal hip shortly after being welded. The aluminum fixture was used to hold the three leg connections parallel.

extremely simple geometry-wise. The only thing the legs do is provide a point to point structural connection between the feet and the hip meaning that a mass produced tube can be used in that place without modification. Carbon fiber construction could have been used for the rest of the robot with performance benefits, but the molding and layup process is much, much more difficult and expensive than the sheet metal welding alternative.

Hip Box Analysis

As part of a side project to learn about finite element analysis, the robot hip was subjected to an in-depth analysis after being built with interesting findings worth noting here. The cutouts in the hip box structure that contains the hip actuator were made based on design intuition. The box can be thought of as a simple beam and the triangular cutouts attempt to remove material from the neutral axis where it isn't being used effectively. This should produce a structure that has a better stiffness to weight ratio. In order to find out if this actually happened a simplified model of the hip box was compared to a a model of the same outside dimensions without the cutouts. The box without the cutouts has a weight of 383 grams versus 275 grams

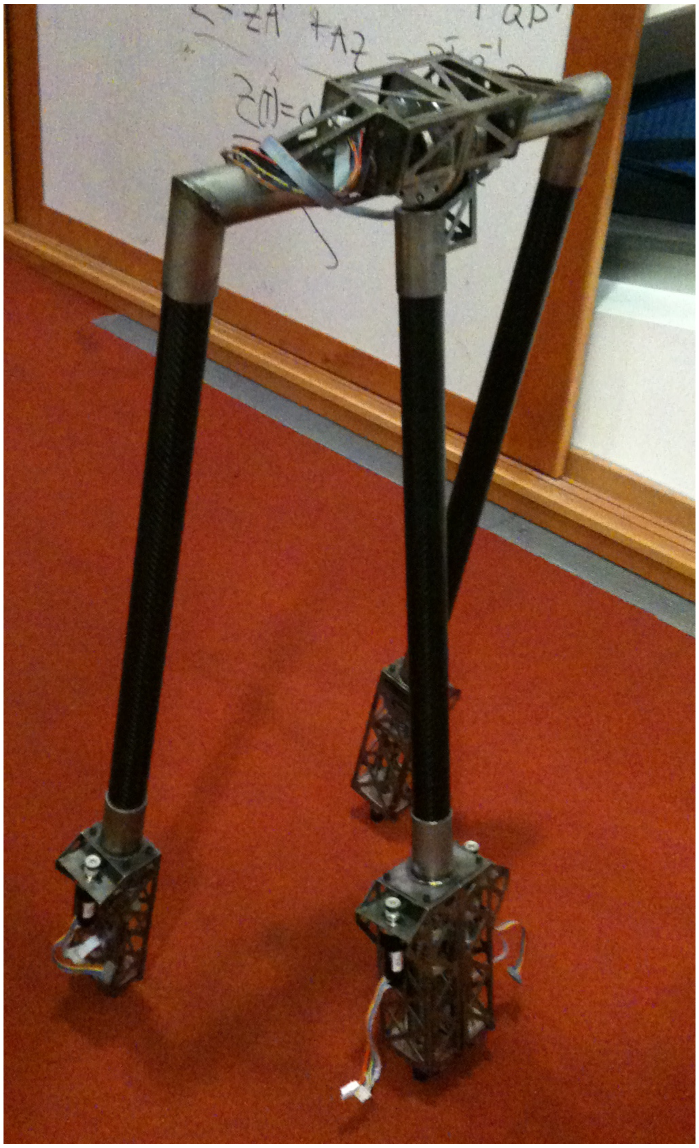


Figure 2-18: Initial mockup of the robot without the upper stiffening hoop.

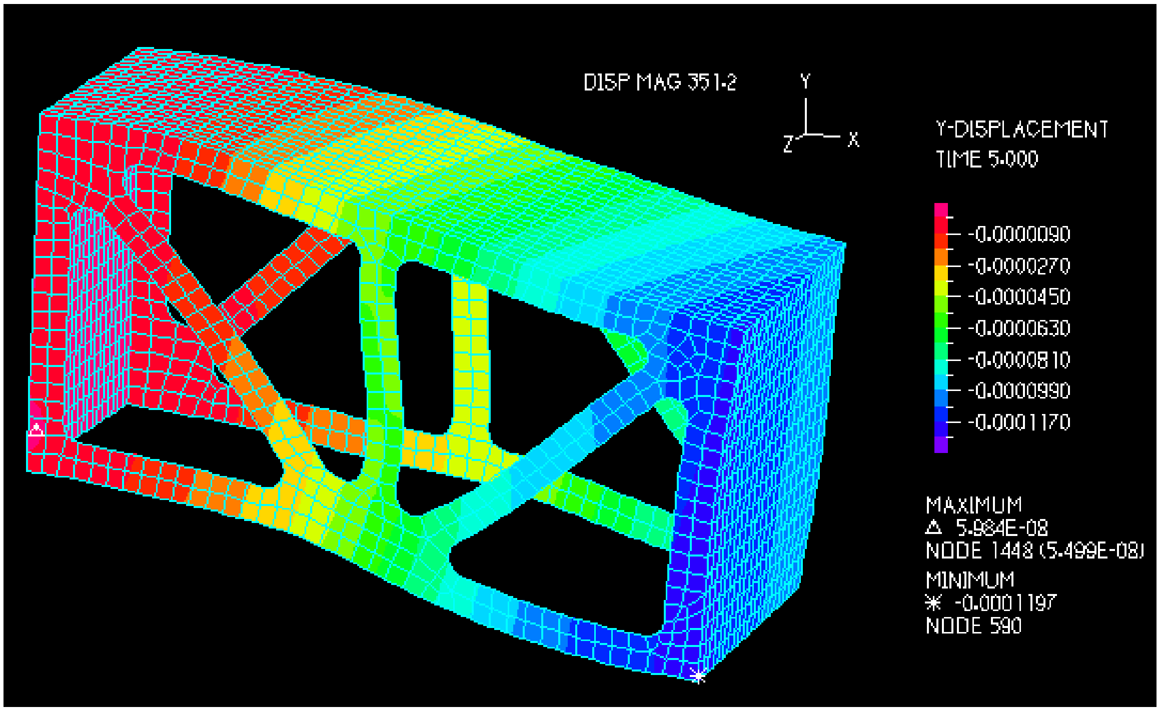


Figure 2-19: The simplified version of the robot's hip box subjected to an end load.

with the cutouts.

The analysis was performed using ADINA, a commercial FEA package, with shell elements. The analysis assumed small strains but large displacements. The box is fully fixed at one end and an end load was applied along the top edge of the opposite end. The loading isn't what is actually seen with the robot, but is representative of some of the real loadings when looking at how the overall structure behaves. The primary reason for the simplified loading is to make the simple box model analytically tractable so that the initial FEA results could be checked against a known answer.

Figure 2.2.4 shows the relevant numerical results of the analysis. The stiffness of the simple box remains high and linear up to extremely high loads but the modified box shows much less rigidity initially and the analysis fails at a relatively small load. In both cases the analysis failed because of catastrophic buckling. The initial takeaway message is that the material removal was a bad idea, it actually ruined both the stiffness and strength of the structure for little weight savings.

The exact structural mechanism by which this happened can be seen in the comparison between Figures 2-20 and 2-19. The simple beam shows a pronounced bulging

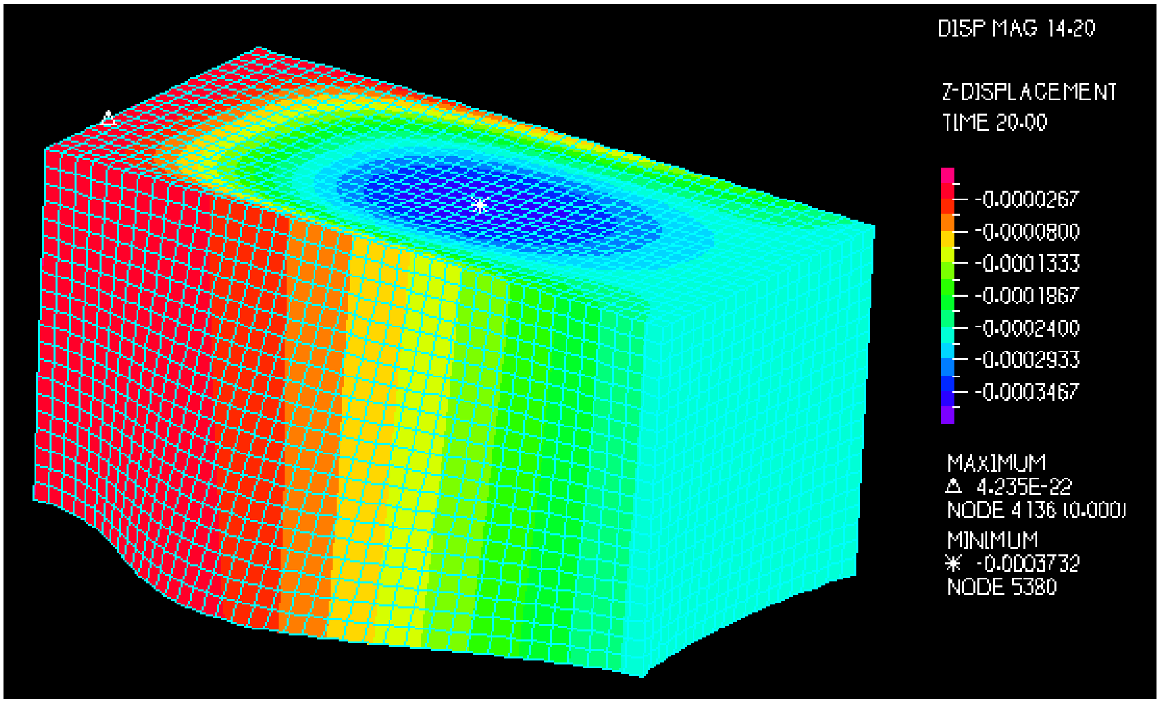
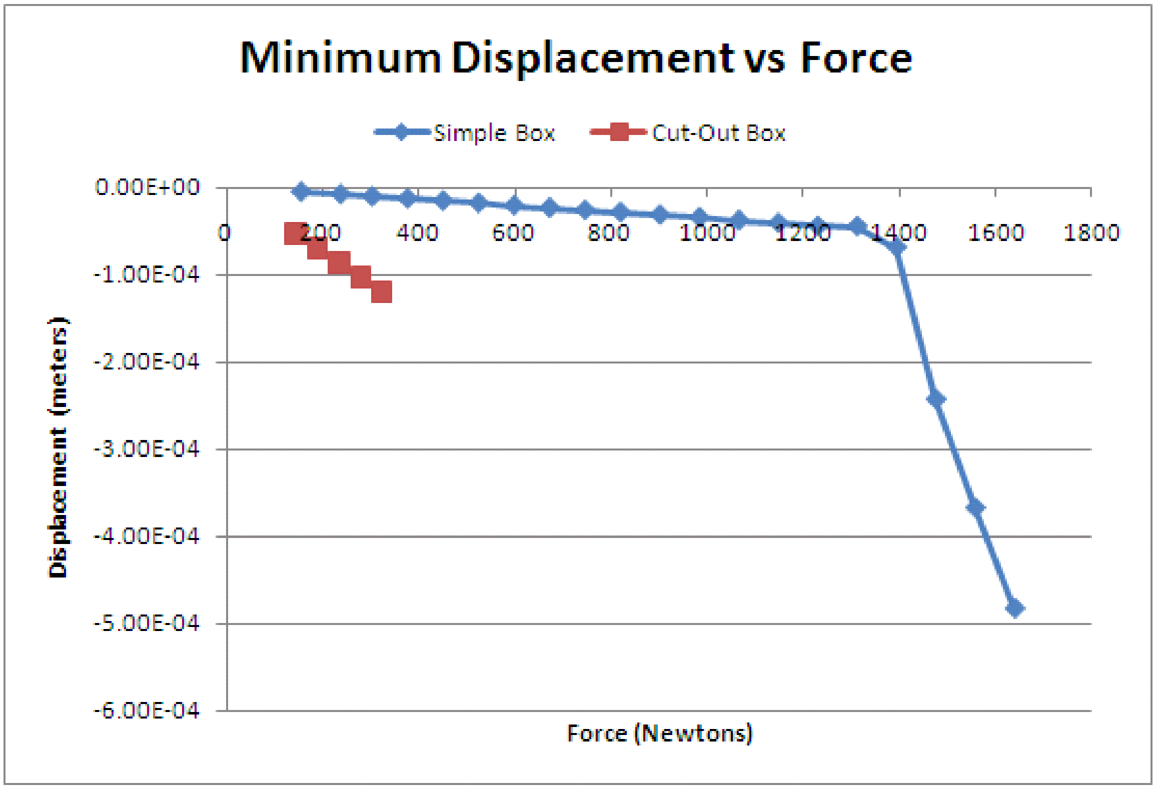


Figure 2-20: The hip box without the triangular cutouts under the same loading.



downward of the top plate which doesn't happen with the complex beam, what's happening here is the material removal removes the coupling between the box sides and the top. In the case of the simple box the highly loaded bottom part of the sides is held in plane by the section that first order beam intuition suggests isn't doing anything useful. When this material is removed the thin sections that should be carrying the load go into buckling almost immediately.

Chapter 3

System Architecture

Besides the base mechanical and electrical design of the robot there is another world of software infrastructure that ties all the individual sensors and actuators together. Previous experimental platforms we had built at the lab used relatively simple software systems, a master program that communicated will all the onboard sensors and actuators, typically connected via a single data acquisition board.

This kind of setup works halfway decently for robots that have a few similar sensors, for example a couple encoders, an IMU, and some motors. All the libraries to work with the different devices can be linked in and threaded together without too much complication. This kind of system often uses a software backend like DSpace, Labview, or Simulink XPC. As a robot grows in complexity the software's responsibilities start to bloat and the system becomes more fragile unless failure cases are expected and handled. Along with that bloat, it can be expected that most of the software in a research lab's arsenal will be somewhat buggy and have little in the way of fault tolerance because it was produced by graduate students. Ideally pieces of the robot's system should be able to fail without bringing down any other parts, be easily monitored, and be trivially reusable without knowledge of the underlying code.

It's also often the case that a robot often requires software pieces compatible only with different programming languages and communication between multiple computers. For example, a motion capture arena, the robot's onboard computer, and a controller computer with a user interface. As it's a buggy research platform we'd also like to log every little thing that happens so that we pin down fault conditions and replay sensor streams for offline algorithm testing. This list of desires in more complex robotic systems quickly gets away from the capabilities of monolithic design.

Seeing the need to integrate many different subsystems on our robot, a CAN network with five motor controllers, motion capture arena, inertial measurement unit, a LIDAR scanner, and possibly multiple other computer systems running control systems offboard we looked toward the Lightweight Communications and Marshalling (LCM) system developed at MIT around the DARPA Urban Challenge vehicle. The premise of the system, in contrast to many other heavyweight robotics software packages is to form the simplest, most decentralized system possible with little in the way of predefined structure in the system or in the data. It's a simple set of libraries that let programs written in many different languages seamlessly communicate with each other either on a local computer or over a network [9].

This is a big deal to roboticists interested in control of dynamic systems because we care about different things than normal roboticists. We typically care about sending small packages of data, sets of gains, encoder readings, etc. around very fast and closing loops with them without too much software infrastructure. Small, fast programs are desired that keep errors isolated and allow deep logging without interference with operation. To that end we've found great use in LCM. The standardized, but lightweight and distributed architecture has allowed us to develop a system that fits our needs exactly is usable with all of our robots.

The architecture we have designed closely resembles the canonical control loop with a few key differences which make it compatible with the real world. At the hardware interfacing level we have a handful of sensors that all connect to the computer via different interfaces. The four toe actuators connect via a single CAN bus, the hip motor via a separate CAN bus, the IMU directly over USB, and a wide range of other possible sensors connect over TCP/IP, like the Vicon system. Each of these sensors, actuators, or sensor networks has its own independent process running at the operating system level. Programs like the IMU interface simply send status updates as fast as they get them while others send their data out on a clock.

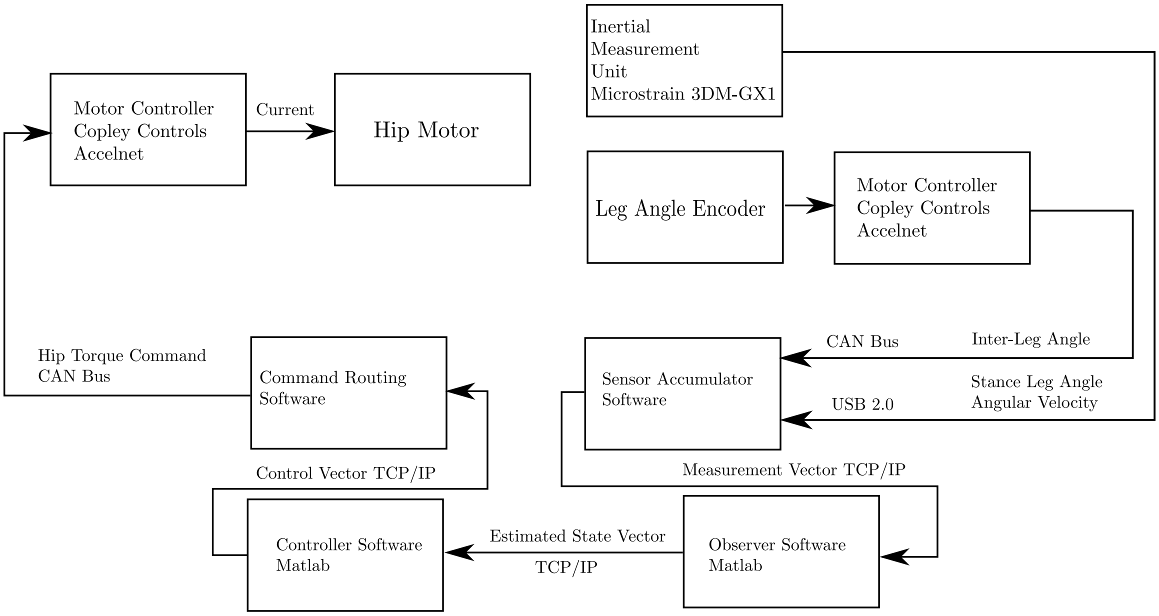


Figure 3-1: The system control and sensing architecture.

A good example of a clocked interface is the dead man switch, it sends an 'enable' message at 100hz which all of the actuator programs look for. Because it is connected to a button it could send messages as fast as it likes and flood the network. Alternatively it could send messages only on state switches, but this is dangerous because messages can be dropped, we would like a signal that's constantly available. The switch is actually located on the robot's joystick along with several other buttons and control sticks which do only send their state when they change. The single joystick interface program provides for all of this functionality.

3.0.5 The Sensor Accumulator

The next piece in the control loop past the sensors would normally be a state estimator, but, this being the real world, things can't be that easy. All of the sensors send their data at different rates, sometimes at rates higher than we want to run the control loop at. This problem is especially acute because ideally the state estimator would be written in Matlab, providing the best prototyping environment and access to the full model of the robot, but Matlab can't handle the IMU blasting out data at 1kHz. The solution to this is to put a synchronization program in between them, which can easily be written in C because it doesn't have to do anything complicated. This program, called the Sensor Accumulator, subscribes to the LCM channels of every sensor on the robot, takes in every message they send, and sends out a consolidated "Robot Measurement" message at a fixed rate. This not only fixes the performance issue already stated, but eliminates another more subtle issue. The state estimator can be expected to have a lot of computation to do with the data it gets, enough that it would need a separate thread to manage all the incoming sensor messages while it does its real job. A consolidated measurement message on a fixed clock means that the state estimator can not only operate with a single thread, but can simply block until it gets the measurement message, making the sensor accumulator a very effective clock for the whole control loop. The measurement message triggers a state update and the consequential state estimate message, which then triggers a control update.

3.0.6 The Command Dispatch

The control update bring into light the Sensor Accumulator's brother, the Command Dispatch. The command dispatch receives a consolidated actuator command message and has the job of splitting that up into all the different messages that actually go out to the hardware interfaces. This functionality is less critical than the Sensor Accumulator's asynchronous sensor management, but is still important to the ability to abstract away the hardware for the purpose of simulation, the dual purpose of the architecture.

3.0.7 Plug and simulate functionality

With the Sensor Accumulator and Command Dispatch both serving as a firewall to the detailed hardware interfaces a few very cool things are easily accomplished. The robot is reduced to a system that is commanded and sensed via two well defined messages. These two messages have can be read by matlab and turned into the canonical measurement and command vectors, $\mathbf{y}$ and $\mathbf{u}$. The state vector that is traditionally produced from the measurements is also defined as an LCM message and message to vector mapping to be sent from an independently operating state estimator process to a controller.

This means that it's possible to replace the hardware robot with either a simulator that produces perfect state messages or with a simulator that produces measurement messages according to a sensor model with zero changes to the code of any existing piece. It also means that it's easy to log the inputs and outputs of the entire system using preexisting LCM utilities and play back real sensor messages from the robot to debug state estimator issues, or play back state messages to investigate controller changes.

The base hardware communication software has the very straightforward job of forming the bridge between the software that contains all of the intelligence of interest and the hardware system which means that all of the information really available about the robot is accessible to the logging utilities. This vastly simplifies debugging because it means the actual hardware isn't required to perform diagnostics on the experimental software and the record can be replayed slowed down. For example, it's simple to take several data records of the robot switching stance legs and then to design and test an impact detector offline.

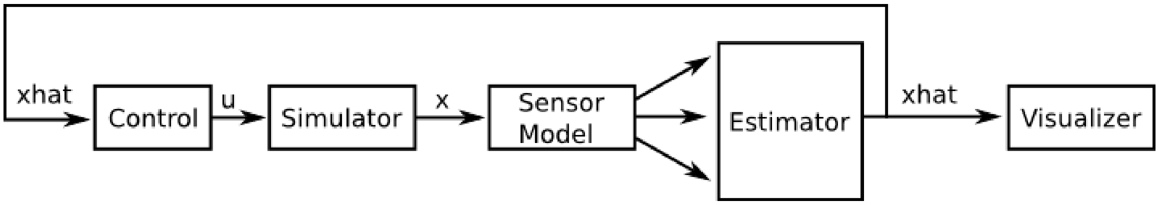


Figure 3-2: Diagram showing the typical use case for the architecture used in the context of simulation. Graphic courtesy of Andrew Barry.

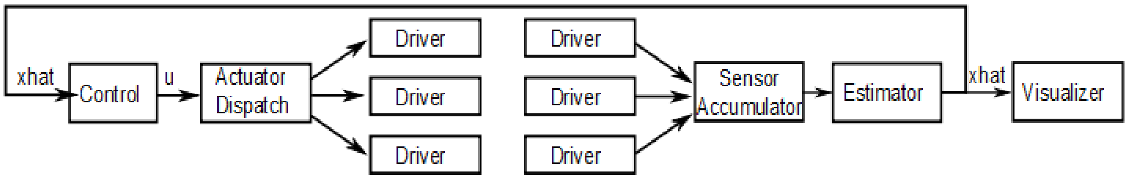


Figure 3-3: Graphic courtesy of Andrew Barry.

3.0.8 System Decentralization

In the LCM architecture the computer's networking infrastructure forms something akin to a big 'pool' of messages, in which any program is able to add or read messages without any other program knowing about it. This means that in addition to the ability for a logging program to dip into the pool without disturbing anything, it's also possible to bridge these pools of messages between different computers completely transparently. For example, if the robot has a scanning laser rangefinder it's possible to run a computationally intensive data processing program on a different computer, avoiding any possible problems with bogging down the main hardware interfacing system that needs to operate on a strict clock. It's also possible to run experimental control loops on a separate computer with a monitor and full Matlab GUI, or have a remote control that's plugged into a separate computer and transmits LCM messages back to the robot.

We found there are also places where this can turn against system performance. The robot commonly generates megabytes per second of data, which is trivially handled by a wired network connection, but when the robot is run untethered on a wireless connection the link can get overwhelmed and will slow down the entire message passing system. An expanded package of LCM utilities called bot-lcm-tools is available by contacting the authors of LCM [9] which includes a selective tunneling program. This allows the message pool to be explicitly segregated between computers and a TCP tunnel established between them to transmit the messages that actually need to travel across the link. On this robot those two signals were the remote control and dead man switch messages from a command computer.

Chapter 4

Control Experiments

While the mechanical design and system architecture are the biggest physical artifacts of the project, the controls work that actually turns those pieces into a walking machine is both the heart and the real target of the project. The design and fabrication took the first nine months of the project while the investigation of what it takes to control the system well has taken the second nine months.

Actually going through the process of making what are straightforward tasks in simulation work on real hardware has taught us a tremendous amount about the strengths and weaknesses of our methods and has given us a wealth of problems to focus new theory on. This chapter walks through the control experiments attempted chronologically.

4.1 Virtual Constraints Experiment

When we finished the actual robot, got all the actuators and sensors working well, the first thing we wanted to do was get it walking via a well investigated method with which good results were nearly guaranteed. Walking control based on the theory of virtual constraints has been applied successfully to several robots, notably RABBIT [3] and MABEL [5].

The basic idea of the controller is that, having a desired walking trajectory defined, all of the motions of the system such as foot positions or the angle of the swing leg are treated as prescribed motions clocked off a key state of the robot which progresses monotonically through a single period of the gait. In the case of this robot that state is the angle of the stance leg, the most difficult state to effect change on because of the large inertia it's tied to. Leaving this state free to evolve on its own frees the control system from trying to regulate the trajectory in time. While it's possible that the system can be stopped by a large enough disturbance as with any controller (no level of cleverness is going to pick it up off the floor) the controller has some guarantees of convergence to the nominal trajectory if all the prescribed motions are followed well.

Without a full system model it's still possible to put together a walking controller using this method for the compass gait because we know something important about the geometry of the robot. As long as the swing leg is in a position to catch the robot as it's falling forward or backward it's impossible to fall over. There are obviously limits to this, the robot can speed up until the leg position can't be kept in front of it, but it provides a good base for keeping the robot standing up. A second controller, possibly modifying the trajectory the toes follow can regulate the robot's speed, providing more or less push-off or causing the swing leg to impact the ground with a shorter or longer step.

A controller based on this architecture was built using mostly trial and error. While that isn't what I would have done if repeating the process it worked quite well to quickly produce a working, if fragile, walking controller in a short time frame. Having the wide variety of tools we've developed in the process of additional control experiments like a reliable system model and trajectory optimization tools I think we could produce a more reliable controller of the same architecture much more quickly. In the iteration of the controller that ended up producing the best trajectories on the real robot the position of the toes was constrained to be the absolute value of a linear function of the stance leg angle, producing triangular trajectories where the swing toe is at minimum extension at zero stance leg angle and the stance toe is at maximum to facilitate the swing through. The swing leg position is regulated to a bezier curve, again dependent on the stance leg angle. The curve was initialized as line with a slope of 2, simply putting the leg where the robot is falling and was then modified experimentally. As an additional way to add energy half of a period of a sine wave is added to the start of the swing toe trajectory. This allows a short lived 'nudge' to be exerted to speed up the robot without significantly changing the rest of the motion.

A couple of perhaps obvious, but still important observations were made during this experiment. First is that we found it very difficult to regulate the position of the swing leg with a conventional linear controller, a result also noticed in the development of MABEL [18]. The swing leg is in essence a pendulum and if we try to maintain the inter-leg angle as twice the angle of the stance leg it means that the the leg will be brought to angles far from vertical, very much invalidating any linearization of the dynamics. As the leg gets further from vertical more torque is required to hold position which must get taken up by a proportional or integral gain term, gains that work well at one angle don't work at others. While gain scheduling is a possible solution here, a much better solution is available: feedback linearization. The simple addition of a torque counteracting gravity, proportional to the sine of the swing leg angle eliminated the largest nonlinearity without needing a full model of the system and vastly improved the tracking of the linear controller.

The second observation applies to the design of the robot for controllability. In this underactuated walking with a point foot there are two control objectives in competition: regulation of the stance leg versus regulation of the swing leg. Trying to balance in place requires more attention to be paid to the stance leg making it advantageous to have a swing with with a large inertia (there's more to 'push' against when making corrections to the stance leg). In walking the opposite is often desired; the stance leg dynamics are left to evolve on their own while the swing leg is quickly moved to keep the robot upright and regulate energy via changing the step length. In this kind of walking it's advantageous to have very lightweight toes and legs in comparison to the rest of the robot. While we were eventually able to accomplish both control tasks with the robot as designed, the aforementioned changes for each case would have made the control task significantly easier.

4.2 System Model and Identification

The system model used for control design is simple in concept, but rather complex mathematically. Diagrammed in Figure 4-1, the robot is treated as three links, each with a mass and inertia, connected at the hip joint. The body angle θ_3 is constrained to bisect the angle between the two legs so it isn't a system state but it is kept to simplify defining the system Lagrangian. The motor rotor inertia is rolled into the body link inertia because they both act together and will be indistinguishable via system identification. The chain is connected to the ground with a second pin joint representing the ground contact of the point toe, free to rotate but fixed translationally by the weight of the robot on top of it. In practice it is possible for the toe to slip when the hip applies large torques. This is left out of the model to keep it at the minimum complexity necessary to accomplish the planning and control objectives.

The Lagrangian is constructed from the positions, velocities, and angular velocities of the three bodies in the conventional fashion. The derivation, while largely mechanical, is quite long and performed using Mathematica. The Mathematica notebook used is included in Appendix A. The final equations are formatted in the manipulator equation form shown in Equation 4.1, where $\mathbf{q}$ and $\mathbf{u}$ are defined as in Equations 4.2 and 4.3. The constants C_i are the physical quantities related to combinations of masses, lengths and inertias.

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{B}\mathbf{u} \tag{4.1}$$

$$\mathbf{q} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \tag{4.2}$$

$$\mathbf{u} = \begin{bmatrix} \tau \\ -\tau \end{bmatrix} \tag{4.3}$$