Design, Construction, and Experiments with a Compass Gait Walking Robot

by

Zachary J Jackowski

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011

© Massachusetts Institute of Technology 2011. All rights reserved.

Author . . . .
Department of Mechanical Engineering
May 6, 2011

Certified by . . . .
Russell L Tedrake
Associate Professor
Thesis Supervisor

Certified by . . . .
Sangbae Kim
Assistant Professor
ME Faculty Reader

Accepted by . . . .
David E. Hardt
Chairman, Department Committee on Graduate Theses

Design, Construction, and Experiments with a Compass Gait Walking Robot

by

Zachary J Jackowski

Submitted to the Department of Mechanical Engineering
on May 6, 2011, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

In recent years a number of new computational techniques for the control of nonlinear and underactuated systems have been developed and tested largely in theory and simulation. In order to better understand how these new tools are applied to real systems and to expose areas where the theory is lacking testing on a physical model system is necessary. In this thesis a human scale, free walking, planar bipedal walking robot is designed and several of these new control techniques are tested. These include system identification via simulation error optimization, simulation based LQR-Trees, and transverse stabilization of trajectories. Emphasis is put on the topics of designing highly dynamic robots, practical considerations in implementation of these advanced control strategies, and exploring where these techniques need additional development.

Thesis Supervisor: Russell L Tedrake
Title: Associate Professor

ME Faculty Reader: Sangbae Kim
Title: Assistant Professor

Acknowledgments

I'd like to thank Russ Tedrake for support, both in resources and direct help with this project in the form of countless discussions and late night code contributions. I'd also like to thank Anirudha Majumdar for a summer of great help getting the robot take its first steps and Amy Qian for her help early on with design work and fabrication of the robot's foot actuators. A great deal of thanks is also deserved by everyone else at the Robot Locomotion Group, Michael Levashov for lots of help with dynamics and modeling, Ian Manchester for the many consultations on control theory and direct help with making it work, John Roberts for many useful discussion both on robotics and everything else in life and everyone else for making my years at the lab so enjoyable.

Contents

1 Introduction 15

2 Physical System Design 19

2.1 Robot Design 19

2.1.1 Overall Concept 24

2.1.2 Inertial Measurement Unit 27

2.1.3 Hip Sensing 29

2.1.4 Terrain Sensing 31

2.2 Mechanical Design 33

2.2.1 Feet 33

2.2.2 Body and Bisection Mechanism 38

2.2.3 Hip Actuation 39

2.2.4 Frame 44

3 System Architecture 51

3.0.5 The Sensor Accumulator 53

3.0.6 The Command Dispatch 54

3.0.7 Plug and simulate functionality 54

3.0.8 System Decentralization 56

4 Control Experiments 57

4.1 Virtual Constraints Experiment 57

4.2 System Model and Identification 60

4.2.1 Impact Model 62
4.2.2 Actuator Friction 64
4.2.3 System Identification 64
4.3 Observer Design 70
4.4 Harmonic Drive Compliance 73
4.5 TVLQR Stabilized Trajectories 77
4.6 Simulation LQR-Trees 81
4.7 Transverse Stabilized Walking 84

5 Conclusion **89**

A System Model Derivation **95**

List of Figures

1-1 The compass gait with a body is a very simple model of walking which still reflects all the reasons why walking is a difficult problem. The model can be fully described by the continuous dynamics of these three links along with an instantaneous ground contact which switches the stance leg. 17

2-1 The full robot, a free walking realization of the simple compass gait model with a body. The robot is a planar walker but has three legs and four feet, with good reason. 20

2-2 Full image of the acrobot 21

2-3 Close-up of the acrobot elbow. 23

2-4 The lab's old compass gait walker on its boom. 24

2-5 'Max', an example of a dynamic walking robot with a bisecting body. [22] 25

2-6 Encoder flexure bracket. 30

2-7 Concept drawings for the robot's toe actuators. Lead screw design on the left and cable drive on the right. 34

2-8 The robot's series elastic foot actuator. 35

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. 36

2-10 The two middle feet branching off from the single middle leg. 37

2-11 The original foot configuration where the middle two were bolted back to back. 38

2-12 The body's angle bisection mechanism. 40
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. 41
2-14 Explosion of a Harmonic Drive gearbox from the Harmonic Drive operating principles literature [12]. 42
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. 43
2-16 The cross roller bearing inside the robot's Harmonic Drive gearbox. . 43
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. 46
2-18 Initial mockup of the robot without the upper stiffening hoop. 47
2-19 The simplified version of the robot's hip box subjected to an end load. 48
2-20 The hip box without the triangular cutouts under the same loading. . 49

3-1 The system control and sensing architecture. 53
3-2 LCM robot in simulation. 55
3-3 LCM system controlling the real robot hardware. 55

4-1 The three-link system model describing the compass gait plant. . . . 61
4-2 Plot of leg velocities resulting from an open loop command played on the robot versus the simulator. 63
4-3 The robot setup during system identification. Note the springs stabilizing the system at the equilibrium point. 67
4-4 Robot with one leg fixed for system identification. 68
4-5 The final system identification fits, compared against the training data. 71
4-6 Validation plots for the system identification 72
4-7 Time response of the balancing controller without steps taken to attenuate high frequency dynamics. 73

4-8 Discrete Fourier transform of the torque signal from the good balancing response. 74
4-9 Discrete Fourier transform of the torque signal from the balancing response with unwanted high frequency dynamics. 74
4-10 The bracket the IMU is attached to, possibly flexing as the hip motor applies large torques to the structure. 76
4-11 Frequency response of the torque to IMU flex system. 77
4-12 Frequency response of the torque to inter-leg angle encoder system. . 78
4-13 TVLQR stabilized 'one step rebalance' trajectory. The small ripple along the trajectory is due to the very high controller gains exciting unmodeled high frequency dynamics in the system. Nominal trajectory is in cyan while the recorded data is in red. 79
4-14 Plots of the hybrid LQR Tree controller designed to bring the robot to equilibrium on either of the stance legs. 85
4-15 The planed periodic trajectory with the step-in trajectory from equilibrium. 86
4-16 The TVLQR stabilized step-in and first step of the periodic trajectory as run on the real robot. 87

List of Tables

4.1 Physical Constants to Identify 66

Chapter 1

Introduction

As a robot designer and fabricator I come to the area of controls mostly as a consumer but working closely with real theorists. The Robot Locomotion Group has been producing a tremendous amount of promising new ideas in the field of applied nonlinear control, specifically LQR Trees [19] and transverse stabilization [13]. While mathematically well grounded results are important one must look to the actual purpose of the research in order to see that it's only half the story. The other half is how well those techniques work to solve real problems with real hardware and whether they can be implemented successfully and efficiently. This is the end that the compass gait walker project has been working toward. We as a research lab believe that all of the tools required to effectively solve the control of the compass gait currently exist, especially with the development of our LQR Trees method and wanted to develop a hardware platform that can show that fact without a doubt.

The work involved in implementing a full demonstration of our methods on real hardware is also important in and of itself. Previously a great amount of effort has gone into making our methods theoretically sound and working in simulation, but the real goal is almost always making real physical systems work better. Making this happen brings to light a wide variety of new considerations such as how the control theory fits in software system architecture, the constraints of running in real time, and the amenability of common physical systems to the precise modeling our methods require. It is by design that difficulties in this project drive the development of our work in the future, if not explicitly finding ways to deal with them, knowing of their importance and severity while investigating new methods to steer us away from fragile theories.

While the goals stated above apply for the project as a whole, my goal in this thesis is to record as many of the little bits that fall through the cracks of documentation as possible along with the big ideas. I've been surprised and saddened to find a general absence of design information for those getting started in building highly dynamic robots and lot of effort is lost learning the right order of priorities and rules of thumb. While in design there's no replacement for personal experience, knowing pitfalls ahead of time can mean the success of an entire project.

I think it's safe to say that any robot designer knows, as a few trivial examples, to make structures lightweight, to validate what can be before committing to build, and to keep control loop delays small. How these individual factors should play in an ecosystem of many competing priorities is usually completely unknown. The designer's art is to figure out how to spread the resources available around to produce something that works. Figuring out the right places to break the bank can mean the difference between a successful ten thousand dollar robot and a failed fifty thousand dollar robot. The fact is that when you're trying to push the boundaries of control something as simple as a bargain bin inertial measurement unit can keep you from solving the problems that actually matter. The same goes for time of course, a week spent simulating the most critical parts of a robot could save months effort later, not only from the first order effects of having to fix what's wrong, but the second order effects of wasted time leading up to deciding band-aid fixes won't cut it and that something needs to be fixed in the first place. I hope to provide an artifact of my design process and how I balanced these competing factors.

Designers work in a different currency from most scientists and engineers because their products are different. The working machine that's produced may not be a great intellectual work, but it's often the real world outlet and test of those works. I hope that the reader will come away from this not so much with an intimate understanding of all the wonderful things I've discovered, but with a feel for the design process of this kind of machine and the kinds of control methods that can bring it to life. Hopefully the theorists whose work I've drawn on will also be able to see the spots that are still sore and find inspiration for new work in them.

Why we chose to make a simple planar walking robot is an important topic when it seems like similar things have been done before [10] [22] [5] [11] and even more impressive walking robots have been demonstrated [8] [4] [2] [7]. We feel strongly that in order to understand the real theoretical problems at the heart of these complicated control problems concisely the target system needs to be exactly as complicated as necessary and no more so. The free walking compass gait concept takes a scalpel to complexity and reduces the physical system to the minimum necessary to bring forth all the problems we think matter. It represents highly nonlinear hybrid dynamics with realistic but simple ground contact. It also brings the issues of managing a full robotic system such as carrying its own computation resources into the picture without overshadowing what is going on at the lowest levels.

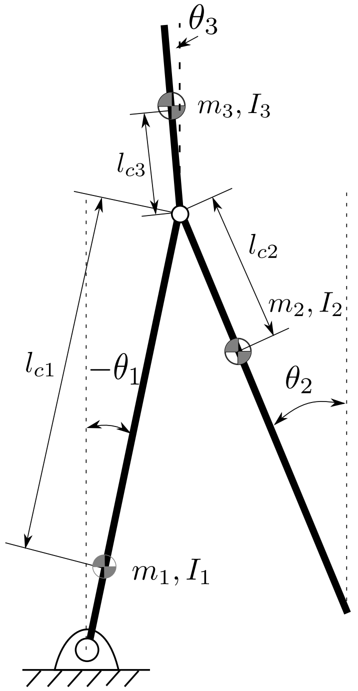


Figure 1-1: The compass gait with a body is a very simple model of walking which still reflects all the reasons why walking is a difficult problem. The model can be fully described by the continuous dynamics of these three links along with an instantaneous ground contact which switches the stance leg.

A system as complicated as ASIMO or BigDog with a very large state adds little of value as far as understanding goes while the system designed here with its small state and number of actuators still representing all the important problems that need to be solved on the control front. Building the robot in-house puts the expertise with the mechanical and software system in direct contact (often in the same person) with the people developing the control ideas that make it work which is important in highly dynamic systems such as this where things like loop delays and modelability can make or break experiments. The robot developed here is also in a performance regime that hasn't been entered by a similar robot before. Careful attention to maximizing actuator performance and ideality has enabled the boundaries of what the system is capable of to be pushed as far out as possible while working within the constraints of an academic lab.

Chapter 2

Physical System Design

With a relatively simple concept and purpose for the robot we thought the actual design of the robot would be very straightforward. The lab already had built a much smaller compass gait walker and I had personally built an acrobot, so we knew more than a few key design points for the robot. There's always more to learn, especially after making a jump in complexity as large as this one and we certainly learned a lot in the process. Hopefully the most important points will all be remembered and recorded here.

2.1 Robot Design

I mean to make a point with the layout of this chapter, that robot design is the most important design issue in a robot. The defining characteristic of a robot is the integration of sensing, actuation, and decision making into a singular fluid system. The mechanical parts of the robot are inseparable in design from the electrical system that brings them to life and the software systems that breathes intelligence into it all.

As a simple example of this, when we designed the acrobot the position encoder for the second link was originally on the back end of the motor, a very clean design decision that integrates the motor assembly and protects the encoder well. Previous experience had shown that kit encoders which rely on user provided bearings had been big sources of problems because of misalignment causing missed counts. Encoders that

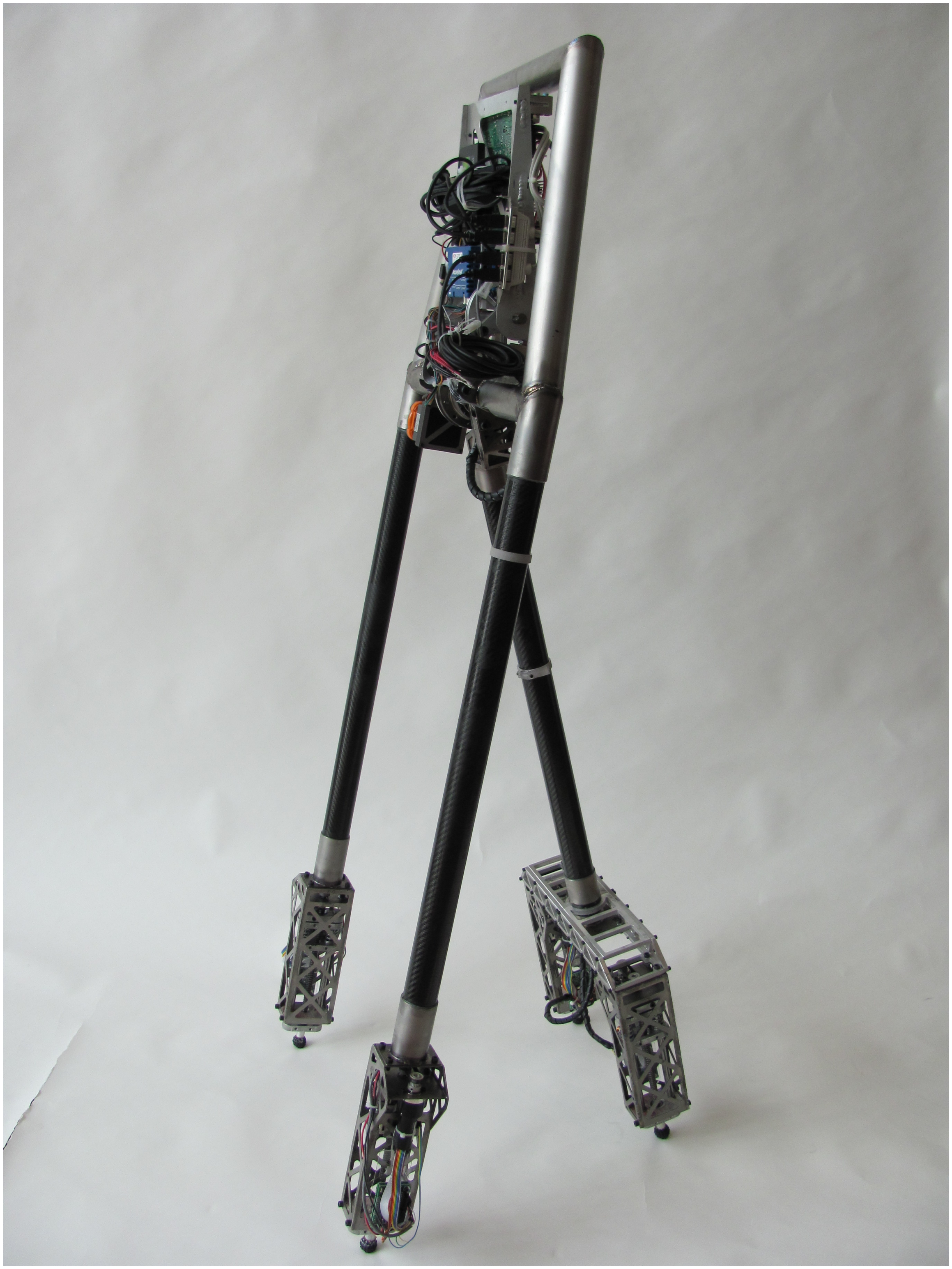


Figure 2-1: The full robot, a free walking realization of the simple compass gait model with a body. The robot is a planar walker but has three legs and four feet, with good reason.

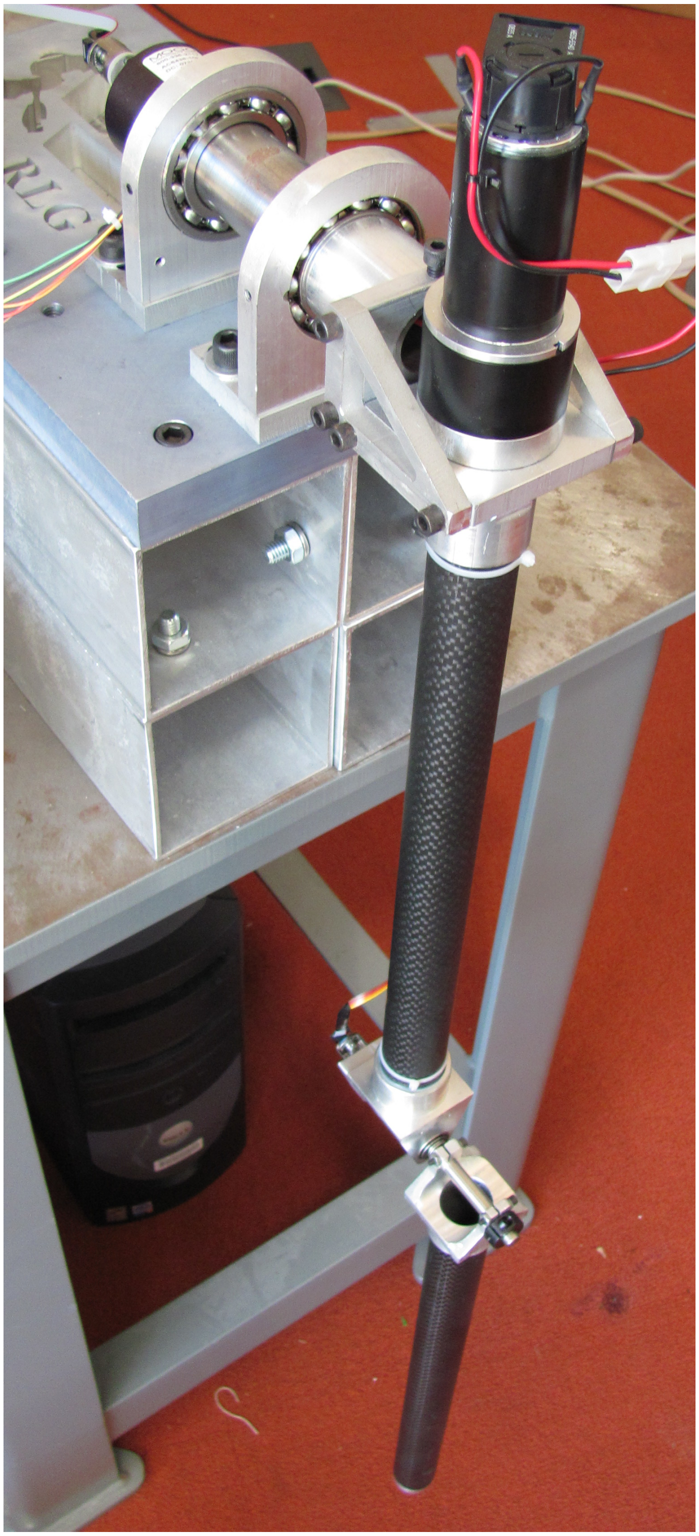


Figure 2-2: The acrobot built previous to the design of the compass gait walker. The motor on the first link is connected to the elbow by a long driveshaft. The encoders in question are located on the back of the motor at the top and on the elbow near the bottom of the picture.

integrate their own bearings and housing are much heavier and expensive, so using the encoder on the back of the motor is a big advantage, or at least seemed to be so until we realized the error we had made. The two foot long carbon fiber driveshaft that connected the motor at the shoulder of the robot to the elbow introduced a small amount of compliance which caused extreme issues for the high-gain control required to balance the arm around the upright position. The compliance of the shaft, combined with the small amount of backlash in the right angle gearbox at the elbow introduced extra dynamics that made control of the system near impossible. In order to make headway in control of the robot we had to sacrifice the integration of the encoder and move it down to the elbow on the other side of the gearbox. In addition to the compromise made in the beauty of the design, we also had to sacrifice some money in buying an encoder with its own integrated bearing set.

One of the most important takeaway lessons from that sensor change is that modeling, control, and advanced sensing strategies are rarely the right solution for a problem that can be designed away from the start. This is a surprisingly difficult lesson to internalize, especially surrounded by people very good at these methods, and it took most of the compass gait project to get all the way there. Elements of this can be seen in almost every major design change on the robot if you look closely, for example modifications to the hip gearbox and the location of the middle toes which be looked at closely in their respective sections.

Structural components make up most of the 'robot' by weight and volume, but it's important to remember that they're the facilitators of the robot's function. In the case of a walking robot this is a little bit strange because the main function of the machine is to push the limits of what has been demonstrated by walking robots before, a nebulous goal that doesn't outwardly say anything about what the functional requirements are. As with many design problems the target specifications aren't known and may never be, the best we can do is make successive approximations and prototypes.

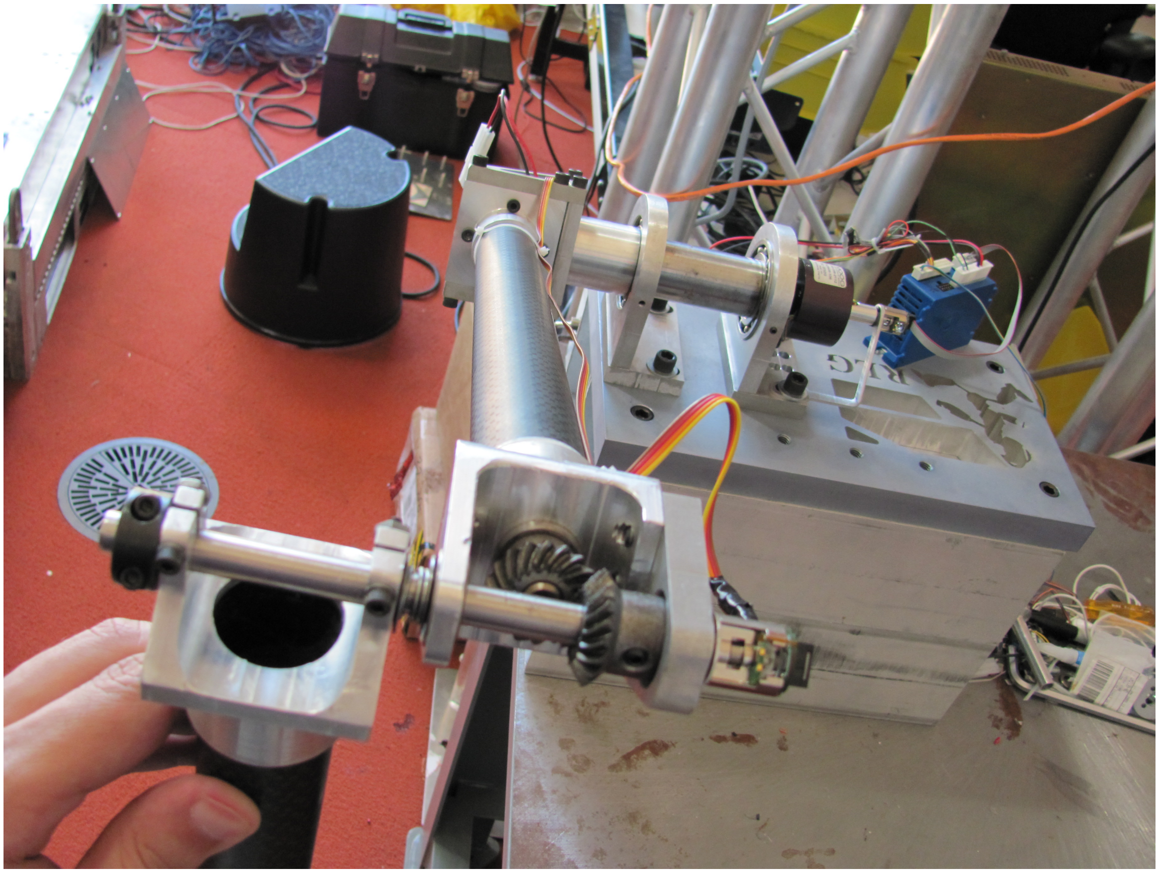


Figure 2-3: One of the several acrobot elbow iterations. In this case helical miter gears are used to achieve smoother operation along with a spring between the two links pushing them apart. This light preload keeps the gears tightly meshed which minimizes backlash while avoiding binding. The integrated encoder can be seen hanging off the backside.

2.1.1 Overall Concept

The initial germs of the idea to build this robot came from a few sources and past projects. The acrobot I had designed for the lab previously was the precursor for many of the design details and lessons, but our work in theory and on our small scale compass gait robot are where the desire for it actually came from. We would have kept working with this small robot, but it had a couple flaws.

Most important was the boom. Originally the robot was put on a boom in order to keep it from falling over sideways, but the boom turned out to have another bad effect: it allowed the robot's key inertias and masses to be changed at will. When the robot was originally built it wasn't actually able to walk effectively until weight was added to the end of the boom on the other side of the fulcrum from the robot. This can be seen in the system parameters provided with one of the papers on experiments with the robot [10], the counterweight provided almost enough force to cancel out gravity, providing a moon-like bounce and slowness in the robot's steps. The time constants associated with the robot falling over without the boom were too fast to be worked with with the available actuators and sensing. This produced experiments which were overly optimistic and appeared nonphysical.

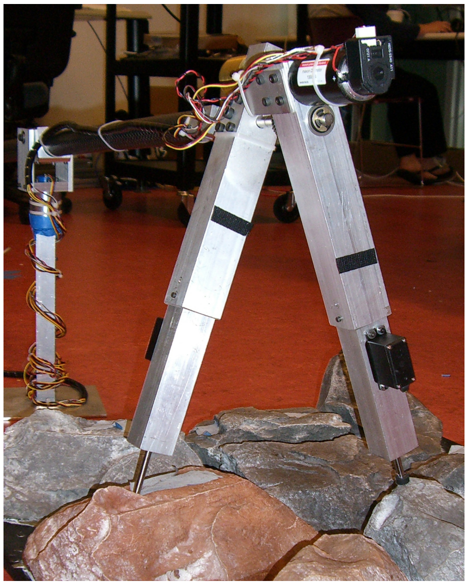


Figure 2-4: The lab's old compass gait walker on its boom.

This robot design works to deal with all the issues that the boom was used to compensate for so that it could be eliminated, producing a much more honest and believable demonstration, over more impressive terrain.

One way to change the inertial characteristics of the system without making the legs heavier and therefore more difficult to move themselves is to introduce a bisecting body to the robot. The example of this that the most inspiration was drawn from was the robot 'Max' from Martijn Wisse et al at TU Delft [22]. Because the body only travels half the angle that a leg does when its moving it only appears half the size dynamically to the leg, but because the falling over action involved both legs moving together the body's mass is fully represented in that portion of the dynamics. Adding a bisecting body was also important because we planned for the new robot to be fully autonomous, requiring it to carry its own power and computation equipment. It's advantageous to locate all this mass in the body for the same inertial reasons, helping keep the legs as easy to move as possible.

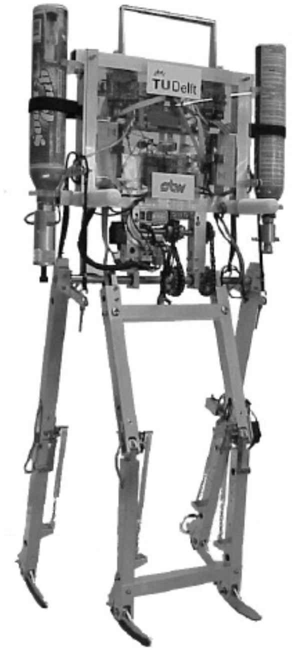


Figure 2-5: 'Max', an example of a dynamic walking robot with a bisecting body. [22]

In deciding to make a planar dynamic walker one of the other main design questions is whether the robot should have knees. To us this was as much a question of research philosophy as mechanical design. In order to avoid hitting the ground as the swing leg moves it must get shorter somehow. In most animals and on many walking robot this is accomplished with active or passive knees, making the leg to break during swing allows the distance between the hip and toe to get shorter. This action, however, greatly increases the complexity of the dynamic model (remember that point about the integration of design and control?). The canonical compass gait model only deals with one collision, impact of the swing toe with the ground, because the robot is assumed to be symmetric. Knees introduce a new hybrid transition into the model: knee locking (knee unlocking usually happens in conjunction with impact). Because this robot will be asymmetric this means that the introduction of knees would change the system from two modes with two transitions, to at least four modes with four transitions, many more if one wishes to account for breaking the knee of the stance leg or impacting with a broken knee. In many cases this wouldn't be much of a concern because the necessary simulation would be performed by an automatic dynamics solver, but we prefer to keep the number of plants as small as possible in order to keep analysis clean. The simpler the analysis the system is the more we likely we are to understand it at a fundamental level.

The alternative to this is prismatic feet, where the feet are actuated in and out parallel to the leg, something that doesn't happen often in biological systems. As long as the actuators don't hit their limits then the number of plant modes and transitions doesn't increase and the complexity of the continuous plant modes doesn't increase too badly. Prismatic feet also have the advantage of being able to push off using the same actuator which would be impossible with a passive or clutch based knee.

The overall size of the robot is one of the really free variables in the design. There are a couple considerations that push it to be large: computation and system time constants. Ideally the robot should be able to carry all the computer power it needs to function, which we decided up front to be a mini-ITX based computer running a modern x86 processor which means the robot's body will at least need to accommodate the volume and weight of the motherboard and batteries to feed a 65W processor. The mechanical time constants determine how fast and precise the control action needs to be, getting easier with the robot getting larger. This should intuitively make sense thinking of the simplest case, the point mass pendulum, whose natural frequency is $\omega = \sqrt{\frac{g}{l}}$. On the other hand, the smaller the robot, the smaller the motors required will be, along with being more safe and less expensive. We decided the scale of an adult human is a nice compromise between these competing objectives, with one more advantage: it's easy to take measurements of adult humans in order to help nail down further design aspects. This scale helps set reasonable expectations for walking and running speed and is also visually impressive when demonstrated in person. While humans don't normally perform the compass gait, it isn't too hard to imitate it for the purpose of collecting some rough data.

Much of the initial rollout of the specifications that individual components were built to was based on an initial guess for the weights of all the different robot parts. This is one area where some design intuition is extremely important. The process of doing things like picking how powerful the hip actuator should be is iterative, it's impossible to know exactly what will be expected of the actuator until the entire robot is designed, and even worse, functioning. Much of the intuition in designing a system like this comes down to having a feel for how much assemblies should weigh before detailed design has been done, in essence seeding the design optimization problem. There are also a lot of constraints that can be exploited if you know what to look for. For example, we know going into the design that the robot will need to carry a mini-ITX form factor computer, various standard sensors, and a battery pack to power at least the computer. These weights can help a lot to set the scale of the design problem.

2.1.2 Inertial Measurement Unit

Because the robot isn't fixed to the ground it's surprisingly difficult to figure out the current position of all its links. The angle between the legs is simple to measure, but figuring out the angle of the legs relative to gravity is more difficult and extremely important because control of the robot depends on knowing where the robot is and how the dynamics are moving. If the robot were on a boom this would not be a concern because the boom would be able to provide a reference to the ground. If the robot was always in contact with the ground it would be possible to attach a plate to the toes that could measure the ground angle, but that option falls apart if the robot has an aerial phase where neither foot is in contact with the ground.

An inertial measurement unit or IMU uses a collection of accelerometers, gyroscopes, and often magnetometers to produce orientation estimates without a ground reference besides gravity and an initial zeroing. A tremendous amount of development has gone into these systems due to interest from military and commercial aerospace applications and even a protracted discussion is well beyond the scope of this paper. This also means that there's a diverse field of off the shelf hardware available for the job.

There's a wide divide between the top of line MEMS IMUs available and the the bottom end of the next step up, laser ring based units which offer much better accuracy and resolution. The smallest laser ring based IMUs are designed for large aircraft or wheeled ground based vehicles and still weigh several kilograms and cost tens of thousands of dollars as they're largely targeted at military and aerospace applications so the choice here is easy, the top of the line MEMS device.

The best MEMS based IMU available at the time is the Microstrain 3DM-GX3 [14], which costs about \$1.9k and weighs 18 grams. According to the sensor specifications it has an accuracy of 2 degrees in dynamic conditions and 0.5 degrees in static conditions which is strikingly close to what we experienced in actual use. Most of the inaccuracy is in slow drift which can be pulled out using extra sensing with a little effort which we ended up needing to do for the balancing experiment. The other major nonideality we experienced is that the orientation estimate lags quite badly in fast motions, for example, taking about a second to settle after a 90 degree sensor rotation at 300 degrees per second. This turned into a nagging issue as we worked on control for walking.

2.1.3 Hip Sensing

Measuring the angle between the two legs is plainly a place for a shaft encoder, but there are thousands of encoders available. Absolute, relative, transmissive, reflective, magnetic? We need to look at the functional requirements: Reliability, resolution, interface, minimum volume and weight, availability, and cost.

Reliability first, because without being able to trust the most basic of sensors there really is nothing that can be accomplished. Every other requirement really can be pushed, we can always carve out a little bit of weight elsewhere, but lost ticks means unpredictable behavior. That's worse than no behavior. Previous experience indicates that encoders that rely on user-implemented bearings to constrain the code wheel are susceptible to lots of problems. Misalignment between the code wheel and shaft axes will produce wobbling over the reading head resulting in sporadic lost ticks, axial alignment with distance misalignment will produce lost ticks more consistently. The worst case is that these misalignments result in the code wheel crashing into the read head which will permanently damage it. The solution to all of these issues is to use an encoder with its own integrated shaft and bearing set.

Second, resolution, is relatively easy to deal with. Encoders are usually available in a wide range of resolutions, but the higher it goes the more fragile and expensive the encoder gets because the lines that must be sensed are so much smaller. Once they get small enough even a piece of dust can wreak havoc on reliability. For that reason it's smart to pick a resolution that isn't excessive compared to its purpose. So how does the resolution of an encoder relate to the robot it's attached to? It's hard to tell how much resolution is needed in position to accomplish the control objectives, but it's pretty easy to figure out what level won't be necessary in a feasible system. The best IMU available before moving into the laser ring range, coincidently the one selected for this robot already, has an orientation repeatability of 0.2 degrees[14] which translates to 1800 counts per revolution. Because the position estimate for the whole robot is driven off the IMU the encoder will effectively inherit its flaws, so anything much more than that is wasted, at least in terms of position.

It would be nice if there were enough ticks to go around to provide a fine estimate of velocity too. The way this is usually done is by looking at the difference in position over some very small amount of time, either the control loop rate, or faster than that if it's done on lower level hardware, and then applying a low pass filter to smooth out the signal. Figuring out this specification requires knowing a little bit about the velocities the robot will be seeing and how the actual encoder velocity estimate is produced. Because of the uncertainty of all of these pieces (and the fact that in the end we won't even actually use the encoder's velocity estimate thanks to better methods) we'll leave this one at just the mention of it.

When the encoder is maintained as a closed structure it maintains the accuracy that it passed quality assurance with at the factory. As the encoder shaft is fully constrained with respect to the reader structure in all degrees of freedom except a single rotation it's important not to overconstrain it when mounting the encoder to the robot. This is almost always accomplished with a flexure bracket. When the encoder is attached to the shaft to measure, usually by a combination of a loose slip fit and a set screw the whole assembly loses all of its translational degrees of freedom. The job of the bracket is fix the angular position of the encoder body while allowing it a small amount of motion in the two other rotational directions. An appropriate design for the bracket is shown in Figure 2-6.

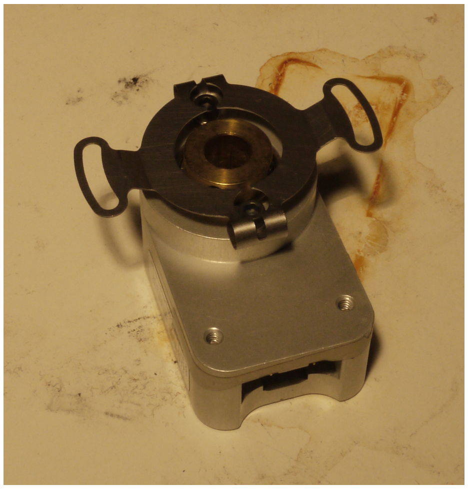


Figure 2-6: Encoder flexure bracket.

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

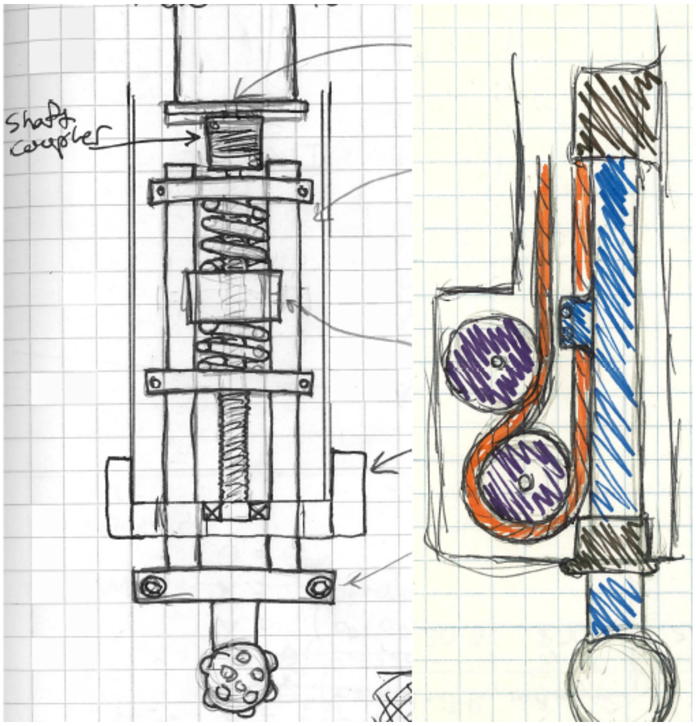


Figure 2-7: Concept drawings for the robot's toe actuators. Lead screw design on the left and cable drive on the right.

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

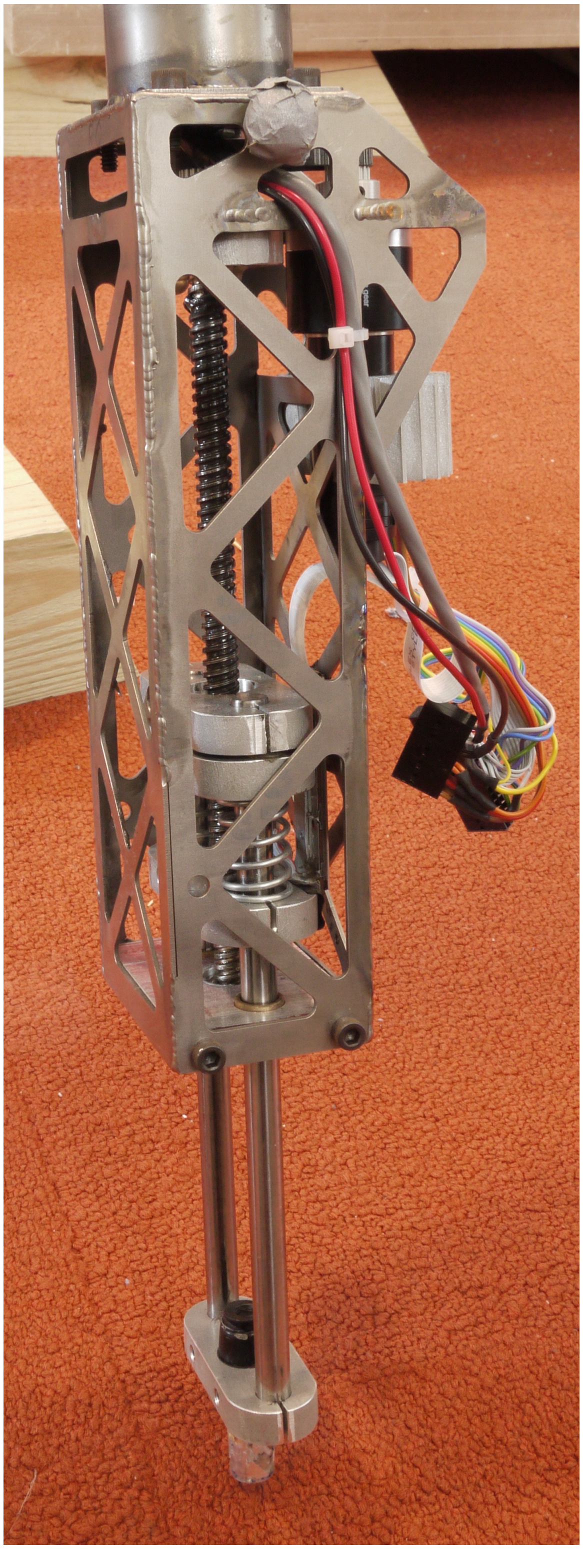


Figure 2-8: The robot's series elastic foot actuator.

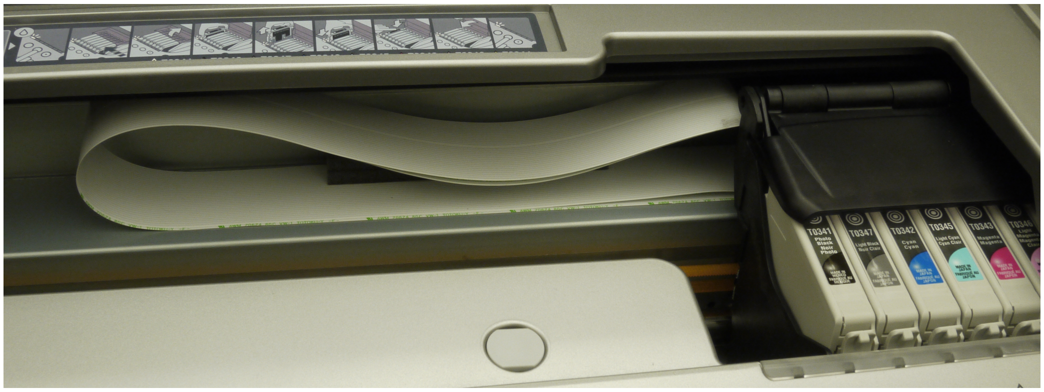


Figure 2-9: The cable which connects to the moving carriage in an inkjet printer was the design inspiration for the similar mechanism in the robot's feet.

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.

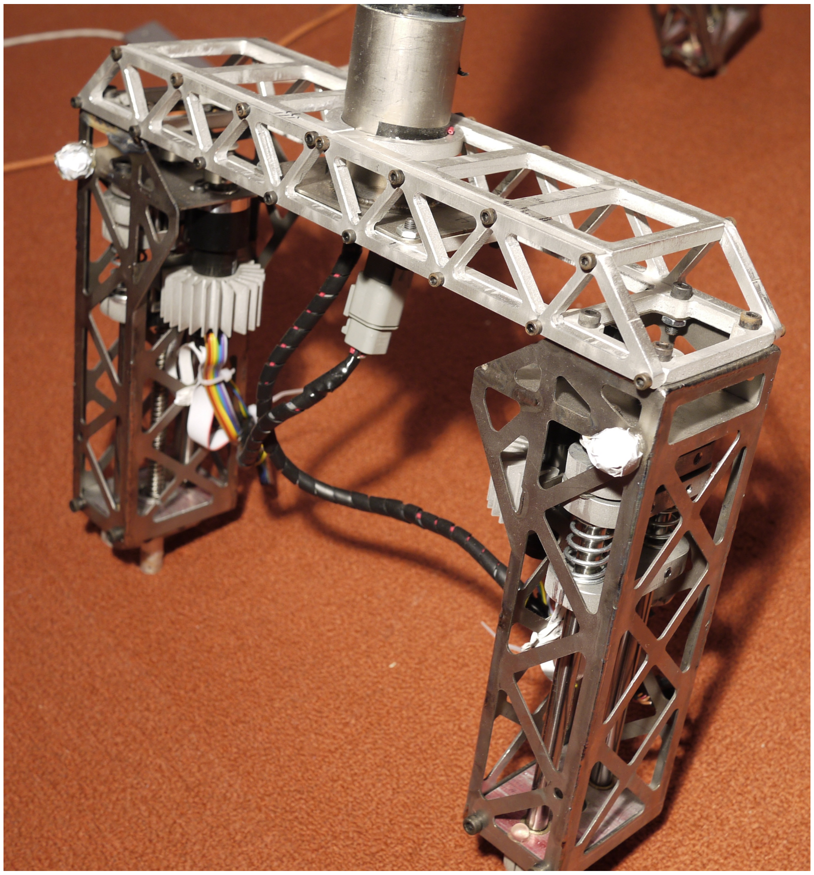


Figure 2-10: The two middle feet branching off from the single middle leg.

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

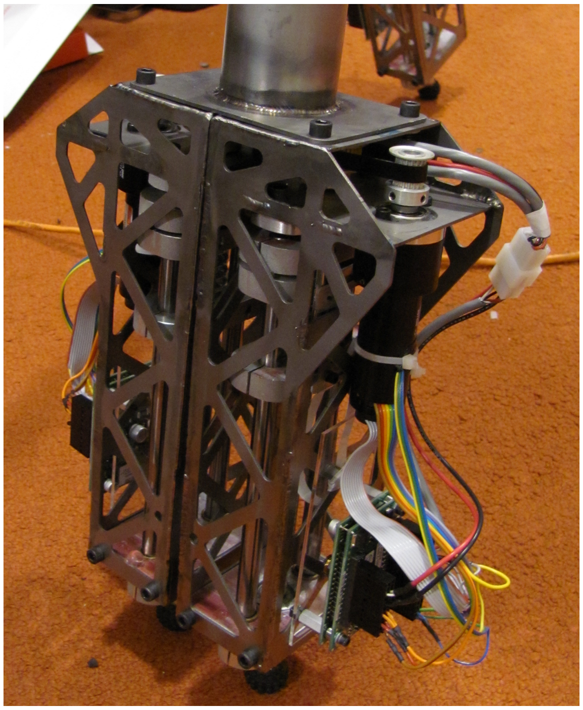


Figure 2-11: The original foot configuration where the middle two were bolted back to back.

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

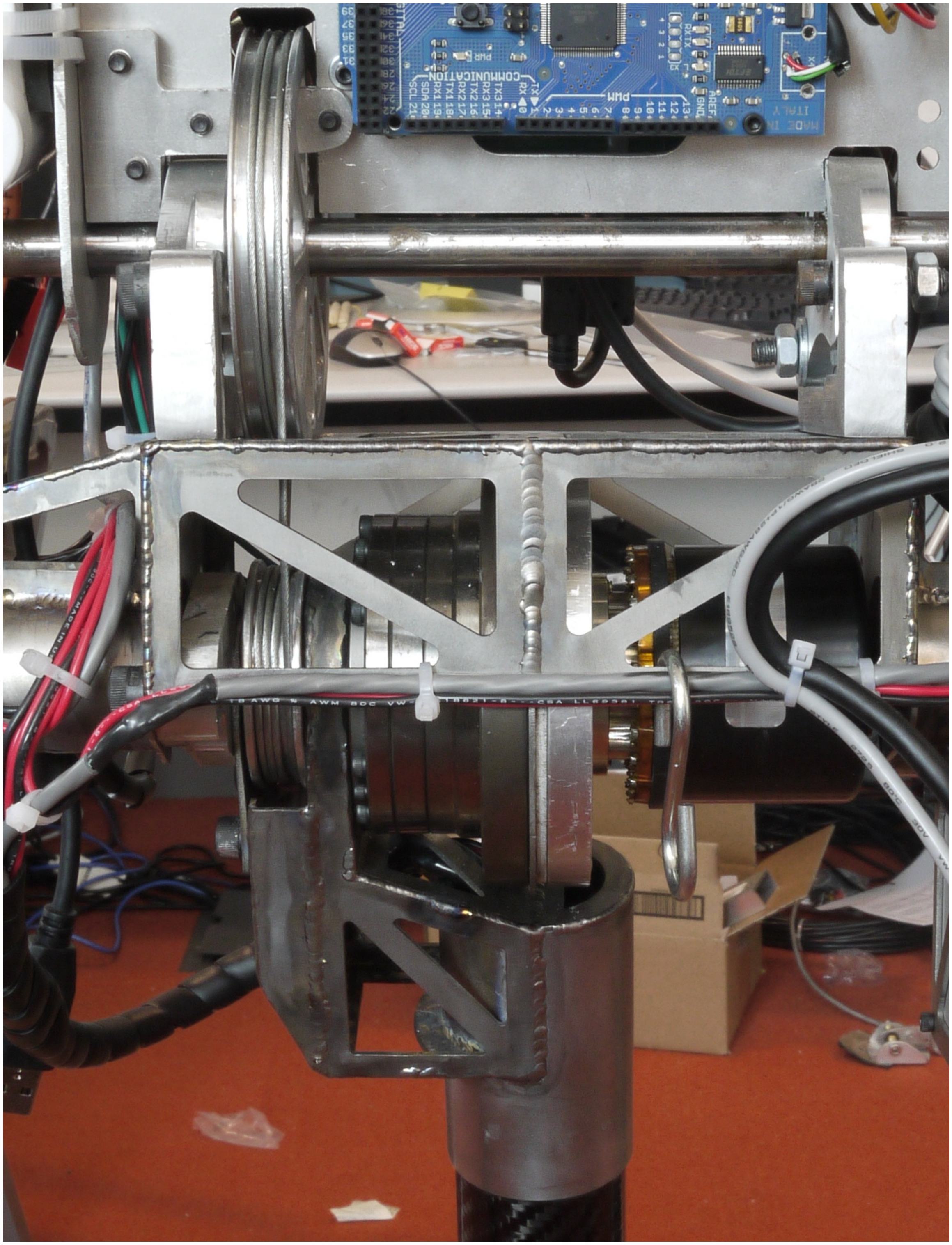


Figure 2-12: The body's angle bisection mechanism.

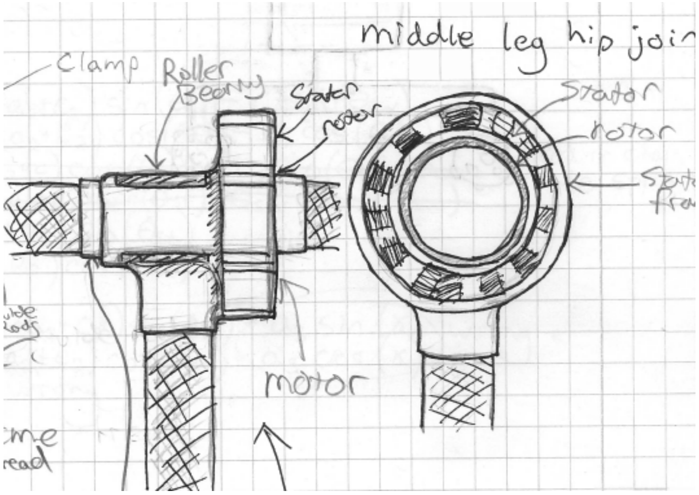


Figure 2-13: Initial concept for a direct drive hip actuator using a frameless motor and our own aluminum frame. This allows the motor to be built into the robot in the most efficient way possible.

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

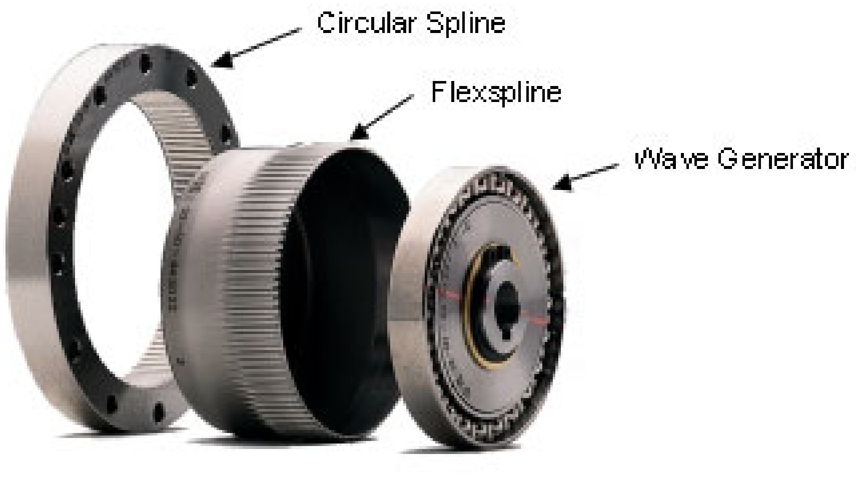


Figure 2-14: Explosion of a Harmonic Drive gearbox from the Harmonic Drive operating principles literature [12].

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

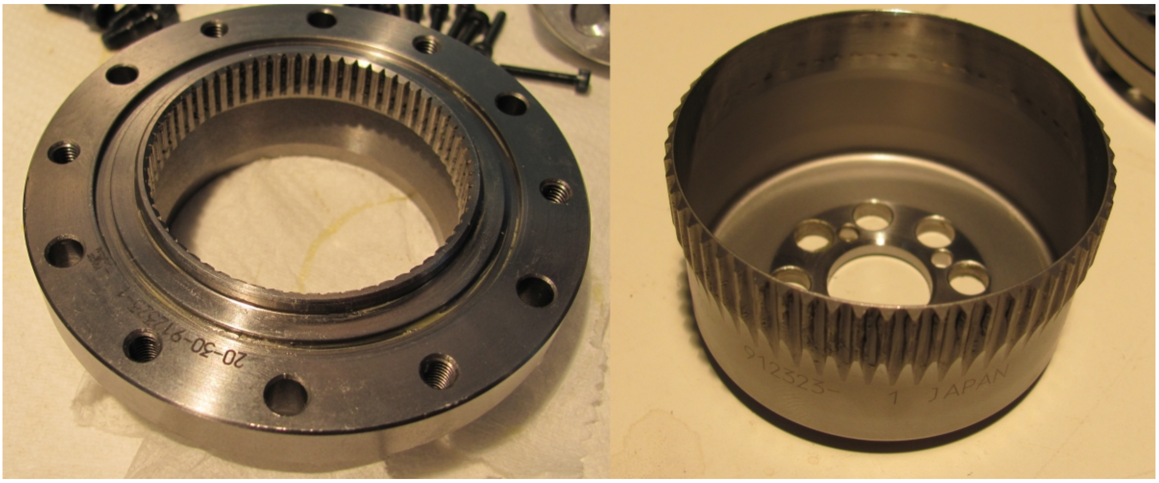


Figure 2-15: The 30 : 1 ratio Harmonic Drive gearbox used in this robot, showing the very large driving teeth. Overall diameter of the flex spline (right) is about 2.5 inches.

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.

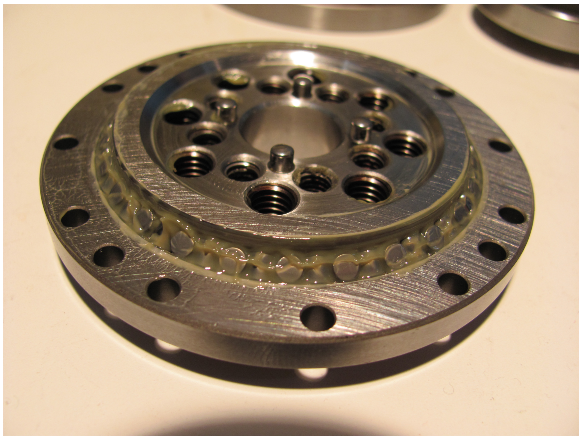


Figure 2-16: The cross roller bearing inside the robot's Harmonic Drive gearbox.

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

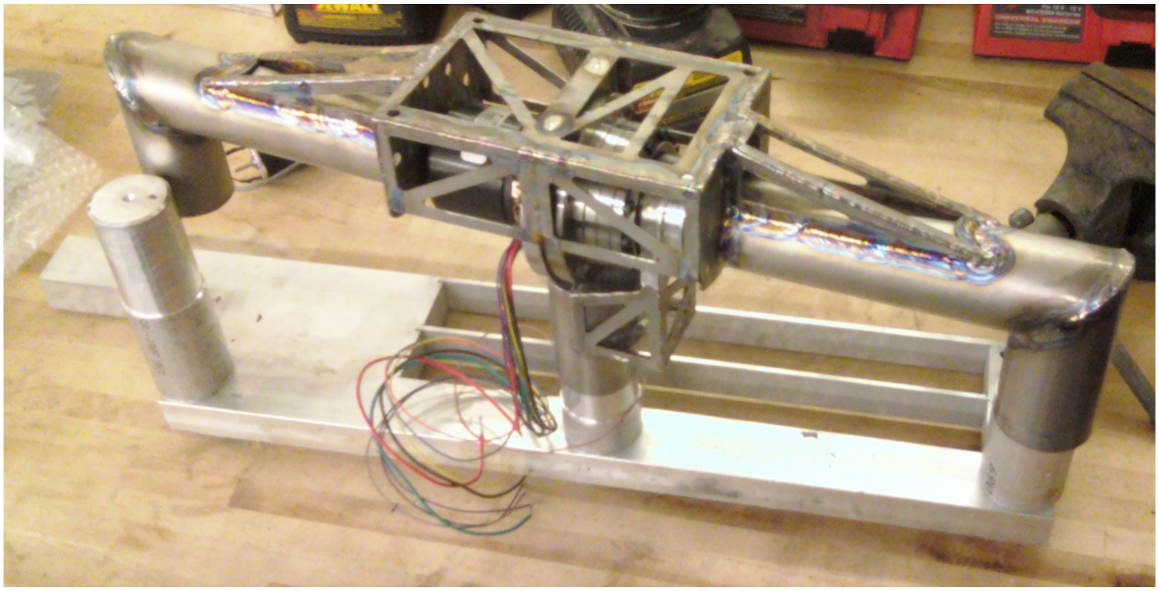


Figure 2-17: The robot's titanium sheet metal hip shortly after being welded. The aluminum fixture was used to hold the three leg connections parallel.

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

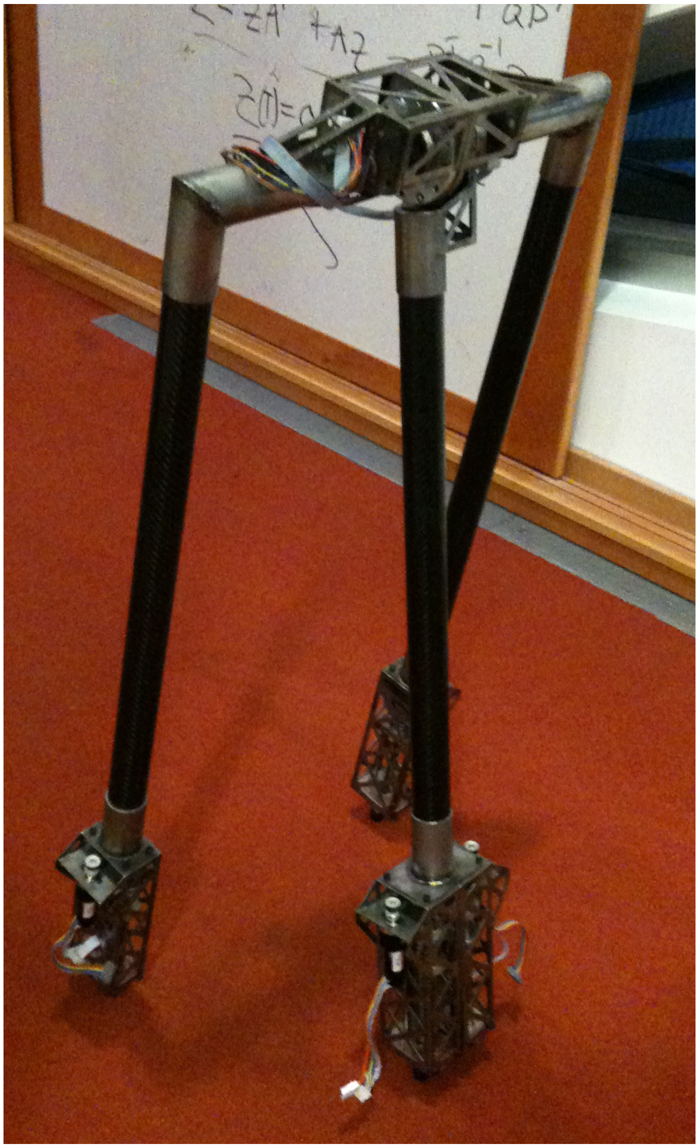


Figure 2-18: Initial mockup of the robot without the upper stiffening hoop.

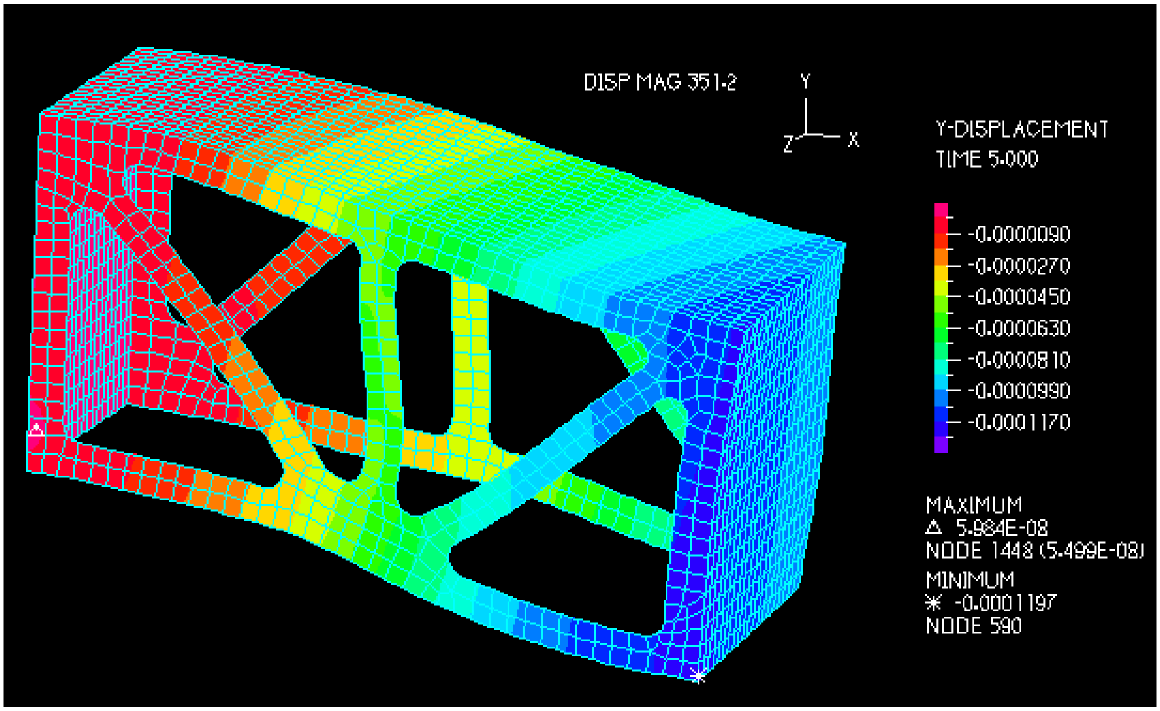

DISP MAG 351.2
Y
Z
X
Y-DISPLACEMENT
TIME 5.000
-0.0000090
-0.0000270
-0.0000450
-0.0000630
-0.0000810
-0.0000990
-0.0001170
MAXIMUM
△ 5.984E-08
NODE 1448 (5.499E-08)
MINIMUM
✳ -0.0001197
NODE 590

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

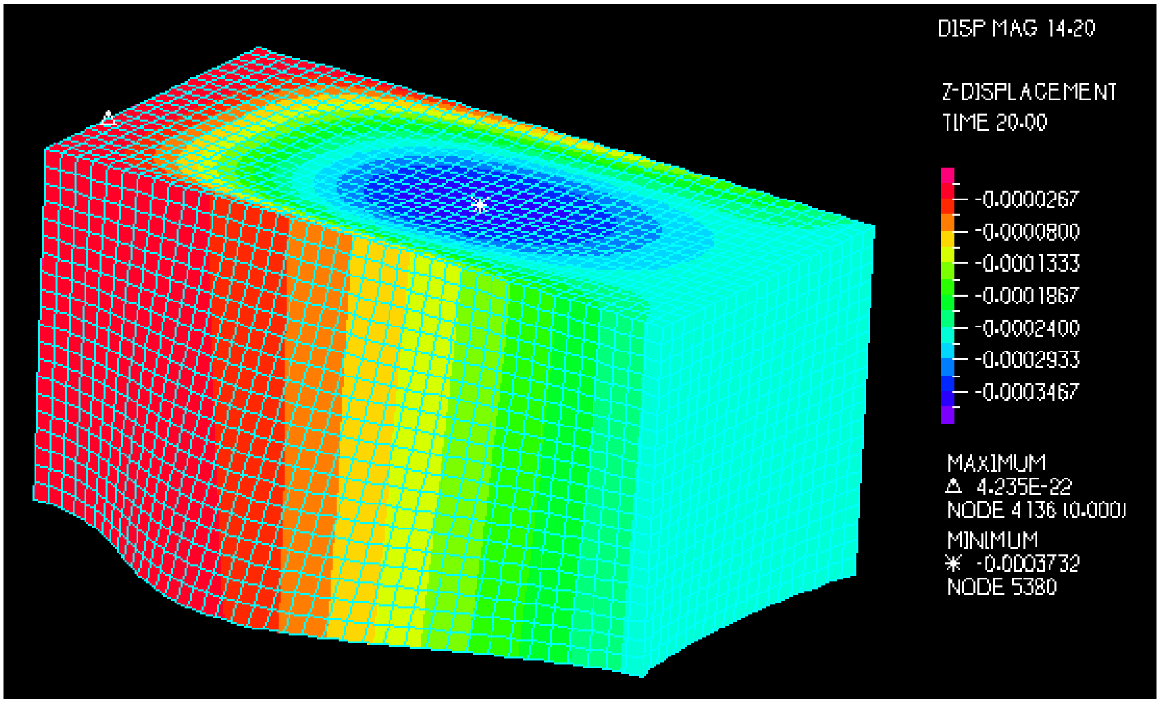
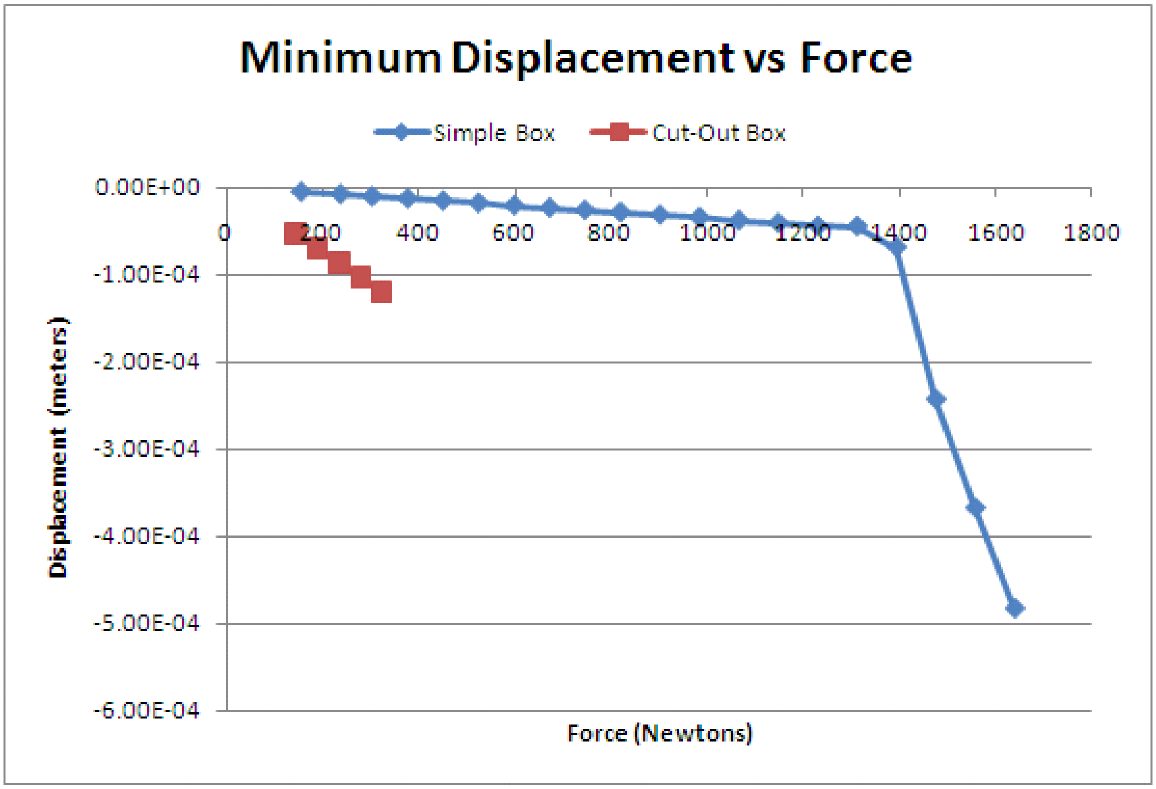

DISP MAG 14.20
Z-DISPLACEMENT
TIME 20.00
-0.0000267
-0.0000800
-0.0001333
-0.0001867
-0.0002400
-0.0002933
-0.0003467
MAXIMUM
△ 4.235E-22
NODE 4136 (0.000)
MINIMUM
✳ -0.0003732
NODE 5380

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.

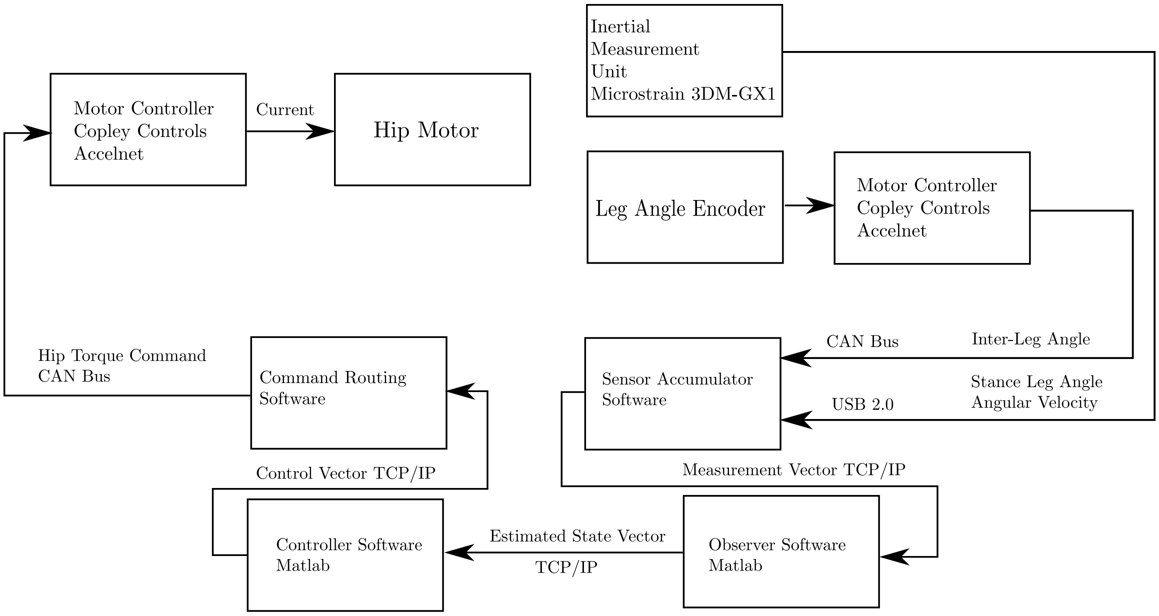

Inertial
Measurement
Unit
Microstrain 3DM-GX1
Motor Controller
Copley Controls
Accelnet
Current
Hip Motor
Leg Angle Encoder
Motor Controller
Copley Controls
Accelnet
CAN Bus
Inter-Leg Angle
Hip Torque Command
CAN Bus
Command Routing
Software
Sensor Accumulator
Software
USB 2.0
Stance Leg Angle
Angular Velocity
Control Vector TCP/IP
Measurement Vector TCP/IP
Controller Software
Matlab
Estimated State Vector
TCP/IP
Observer Software
Matlab

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.

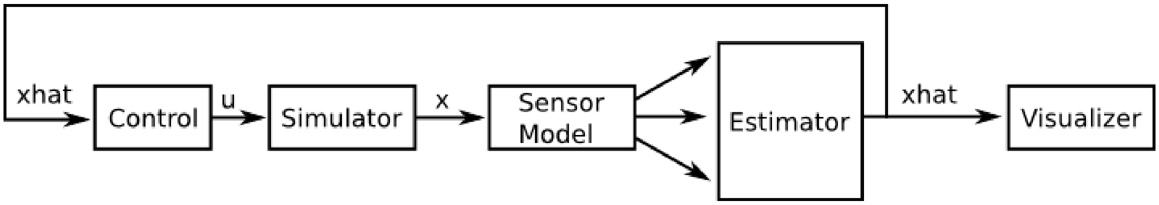

xhat
Control
u
Simulator
x
Sensor Model
Estimator
xhat
Visualizer

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

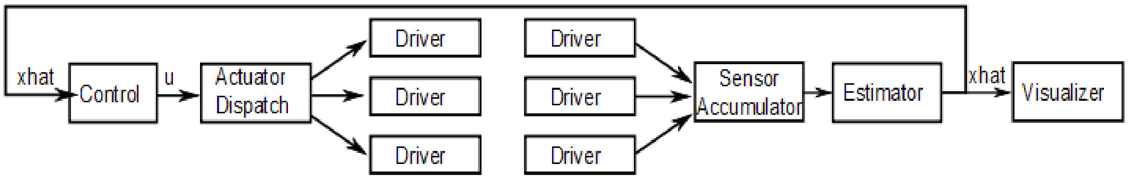

xhat
Control
u
Actuator Dispatch
Driver
Driver
Driver
Driver
Driver
Driver
Sensor Accumulator
Estimator
xhat
Visualizer

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.

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

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

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

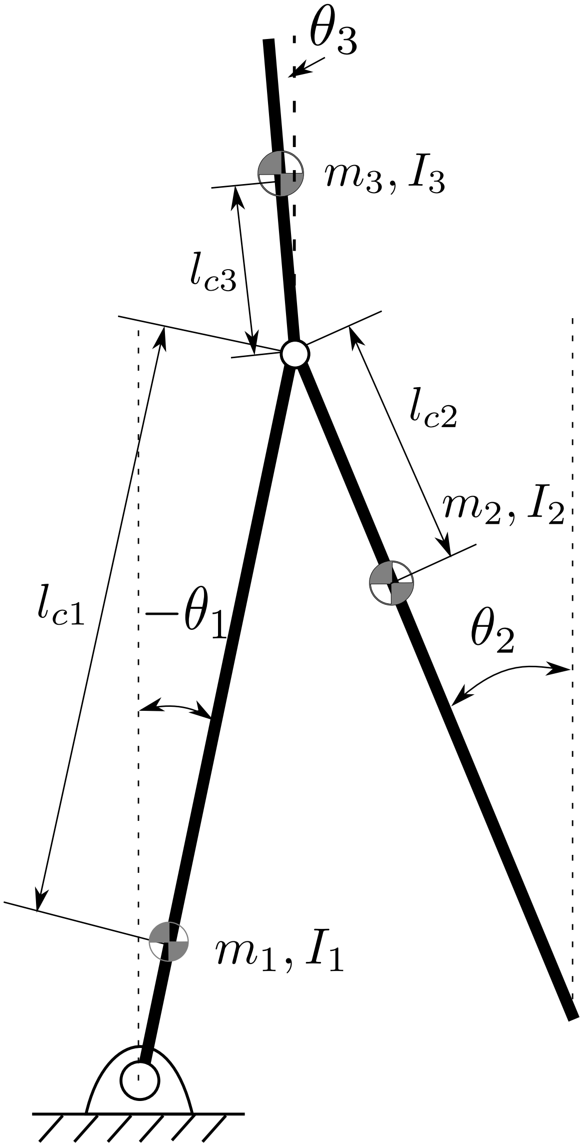

θ_3
m_3, I_3
l_{c3}
l_{c2}
m_2, I_2
l_{c1}
$-\theta_1$
θ_2
m_1, I_1

Figure 4-1: The three-link system model describing the compass gait plant.

$$\mathbf{H} = \begin{bmatrix} c_2 & c_3 - c_5 cos(\theta_-) + 2c_6 cos(\frac{\theta_-}{2}) \\ c_3 - c_5 cos(\theta_-) + 2c_6 cos(\frac{\theta_-}{2}) & 1 + c_1 - 2c_4 + 4c_6 cos(\frac{\theta_-}{2}) \end{bmatrix} \tag{4.4}$$

$$\mathbf{C} = \begin{bmatrix} 0 & (c_5 sin(\theta_-) - 2c_6 sin(\frac{\theta_-}{2}))\dot{\theta}_2 \\ (c_6 sin(\frac{\theta_-}{2}) - c_5 sin(\theta_-))\dot{\theta}_1 & c_6(2\dot{\theta}_1 - \dot{\theta}_2) sin(\frac{\theta_-}{2}) \end{bmatrix} \tag{4.5}$$

$$\mathbf{G} = \begin{bmatrix} w^2(2c_6 sin(-\theta_+) + c_5 sin(\theta_1)) \\ w^2(2c_6 sin(-\theta_+) + (c_4 - 1) sin(\theta_2)) \end{bmatrix} \tag{4.6}$$

The manipulator equations can be linearized around a fixed point in order to produce linearized dynamics in the familiar state space form [20], the end product shown in Equations 4.8 and 4.9.

$$\mathbf{x} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{4.7}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{H}^{-1}\frac{\partial \mathbf{G}}{\partial \mathbf{q}} & -\mathbf{H}^{-1}C \end{bmatrix}\Bigg|_{\mathbf{x}=\mathbf{x_0},\mathbf{u}=\mathbf{u_0}} \tag{4.8}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0} \\ \mathbf{H}^{-1}B \end{bmatrix}\Bigg|_{\mathbf{x}=\mathbf{x_0},\mathbf{u}=\mathbf{u_0}} \tag{4.9}$$

4.2.1 Impact Model

We chose to use an inelastic impact model for these experiments for a few reasons. First, this is what we would like to be happening in reality. A partially elastic collision would mean that the leg bounces off the ground at impact and that the dynamics of the event can't be approximated as instantaneous. As mentioned earlier, we took care to make sure the design of the toes helped to enforce this condition by adding compliance into the feet. This little bit of of a 'suspension' system between the toe and the rest of the robot helps accomplish that goal by making the toe very lightweight compared to the rest of the robot and the spring that holds it against the ground, any oscillation coming from the impact is forced to happen between the robot and the toe rather than between the toe and the ground. This is desirable because the robot-toe system is designed by us, easily modeled, and stays inside a continuous plant mode (in the ideal case).

Analysis of the real impact data shows that the impacts are indeed almost perfectly inelastic even with the toes fully retracted which removes the series elastic effect from the feet. In the fully retracted case with the toes impacting on wood blocks we found that the impact dynamics happened practically instantaneously, within a single time step of the control loop. This is shown in Figure 4-2. Important to note here is that the velocities here are produced using a state observer based on the hybrid model which makes the impact appear more clean that it would from the true velocity.

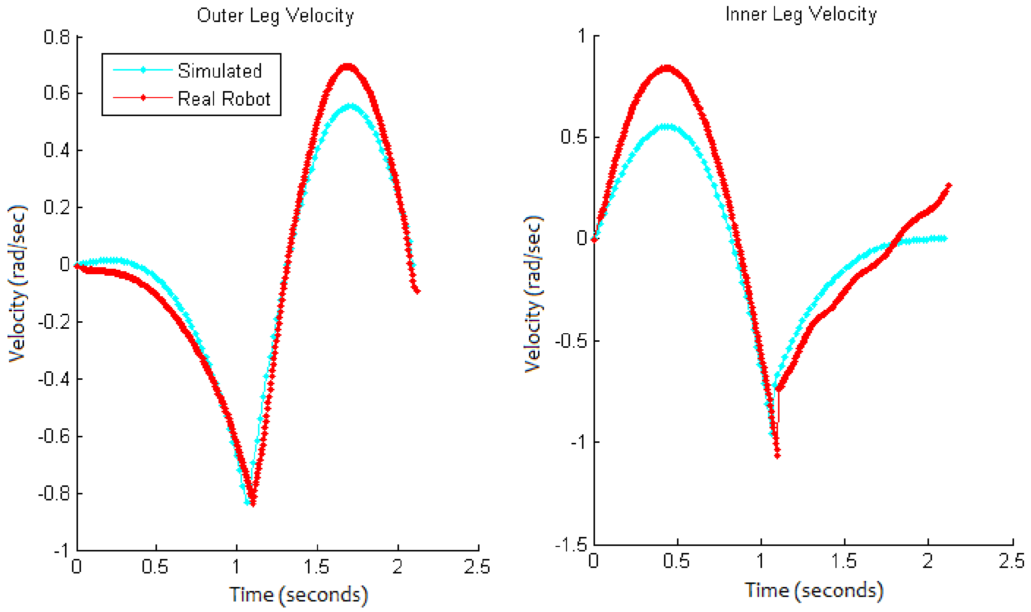


Figure 4-2: Plot of leg velocities resulting from an open loop command played on the robot versus the simulator.

The actual solution for the impact dynamics of a multi link chain like this robot is a little bit complicated but thankfully the process is very mechanical. The derivation of this for an open kinematic chain like this robot and most others of interest in robotics is available from several sources, one of which is Yanzhen Xie's master's thesis on a similar bipedal robot but with knees [23]. With the gradients of the collision function for this robot defined as in equation 4.10 the impact update is as in equation 4.11.

One common confusion when performing the impact update is which model gets used for mass matrix in the update equation. The answer here is *neither* of the stance models. The plant dynamics derivation must be performed in the unpinned configuration. This is why the collision function gradients has entries for two extra states, the x and y position of the stance foot. The derivation for the unpinned plant was performed by Michael Levashov as is included in Appendix A.

$$\mathbf{A}^T = \frac{\partial \mathbf{g}}{\partial \mathbf{q}} = \begin{bmatrix} -cos(\theta_{in}) & -sin(\theta_{in}) \\ cos(\theta_{out}) & sin(\theta_{out}) \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.10}$$

$$\dot{\mathbf{q}}^+ = \dot{\mathbf{q}}^- - \mathbf{M}^{-1}\mathbf{A}^T(\mathbf{A}\mathbf{M}^{-1}\mathbf{A}^T)^{-1}\mathbf{A}\dot{\mathbf{q}}^- \tag{4.11}$$

4.2.2 Actuator Friction

Previous investigation with the Harmonic Drive gearbox has shown a pair of basis functions including Coulomb and viscous friction fit the friction rather well and those basis functions are used here. Rather than modeling the Coulomb friction as a true discontinuity a somewhat arbitrarily sloped but steep sigmoid is substituted to make the dynamics differentiable.

In practice we ended up being able to identify the friction model quite accurately which allowed us to feedback linearize the friction away so the nonlinearities don't need to be accounted for in the rest of the control design performed.

$$\tau_{\mathbf{friction}} = \begin{bmatrix} -b_{visc}(\dot{\theta_2} - \dot{\theta_1}) - b_{coul}(\frac{2}{1+e^{(}-200(\dot{\theta_2}-\dot{\theta_1}))} - 1) \\ b_{visc}(\dot{\theta_2} - \dot{\theta_1}) + b_{coul}(\frac{2}{1+e^{(}-200(\dot{\theta_2}-\dot{\theta_1}))} - 1) \end{bmatrix} \tag{4.12}$$

4.2.3 System Identification

While finding a suitable model form is important, it's only half the job required to produce a suitable system model for control design. For a large mechanical system such as this it's often adequate to measure the actual system parts or compute their properties from the software used to design them (Solidworks in this case), but these individual static measurements are usually only so accurate when it comes to a system even as complex as the robot under discussion. We would like to be able to simulate the whole system accurately with a time horizon of several seconds, the expected length of a step or recovery maneuver. To this end the technique of system identification via simulation error optimization is used. Unlike the conventional least squares system identification which optimizes based on the one step prediction error of the system accelerations, simulation error based system identification uses the metric shown in Equation 4.13 where $\mathbf{y}_{ex}$ are the system outputs from experiment and $\mathbf{y}_{sim}$ are the system outputs from the simulator.

$$J = \sum_{k=0}^{k=N} \|\mathbf{y}_{ex}[k] - \mathbf{y}_{sim}[k]\|^2 \tag{4.13}$$

A set of data is collected on the real robot with some input signal used to excite it, this same input with the same initial conditions is applied to the robot simulator and the measurement vectors produced by each at each time step are compared. The squared difference between simulation and reality is the cost to be minimized and is a nonlinear optimization problem. The minimization is performed by varying the inputs to the robot model according to one of the many nonlinear optimization methods available, in this case Matlab's `fminsearch` function was used.

When fully expanded it's easily seen that the system's equations of motion take the form of a set of scaled nonlinear basis functions. Specific combinations of masses, lengths and inertias are what is really important to the motion of the robot rather than the individual measured parameters we commonly work with. Terms such as sines and cosines of state variables form a set of basis functions in the equations that are multiplied by the system parameters. When doing the more conventional least squares identification this is made very obvious, but it still needs to be remembered when working with the simulation error based methods because working with a set of optimization variables that are overparameterized will cause incorrect output, but without any warnings as the computer chugs along. Only these combinations that multiply distinct basis functions can be identified by the dynamic motions of the robot. These parameters are listed in Table 4.1.

Table 4.1: Physical Constants to Identify

Parameter	Definition	Calculated	SysID
l	Leg length - measured	1.045	1.045
m_t	Total robot mass - measured	15.2	15.2
c_1	$I_1 m_1$	0.0207	0.1007
c_2	$I_2 m_2$	0.1101	0.1434
c_3	$I_3 m_3$	0.0017	0.000596
c_4	$\frac{l_{c1} l m_1}{m_t}$	0.1382	0.1151
c_5	$\frac{l_{c2} l m_2}{m_t}$	0.0987	0.1264
c_6	$\frac{l_{c3} l m_3}{m_t}$	0.0329	0.0026
b_v	Hip viscous friction	none	0.0418
b_c	Hip coulomb friction	none	0.1109

The actual experimental records used to optimize the simulator took a little bit of creativity to come up with themselves. The ideal place to take data is where the robot will be during normal walking and balancing tasks, but without a working controller those areas of state space is unstable. In order to make those areas stable for the purpose of identification two springs were added as pictured in Figure 4-3 which make the system passively stable around the balancing fixed point. The two spring constants were identified to high accuracy in a separate experiment and were added explicitly to the equations of motion for the simulator. This setup still identifies the system parameters without the springs because those parts of the equations of motion are unchanged.

Several other different system identification setups were also used to make sure that all parts of the dynamics were adequately excited and to check against the results. In addition to the spring-stabilized tests conducted with each leg as the stance leg two additional tests were conducted with each leg fixed in the upright position. These two fixed leg tests each provide data on a different subset of the parameters, while the spring tests provide data on all the parameters but with them represented in the experiment with different prominence.

In addition to these tests we also performed a test in which both legs are parallel and the robot moves in a full pendulum mode with the stabilizing springs. The

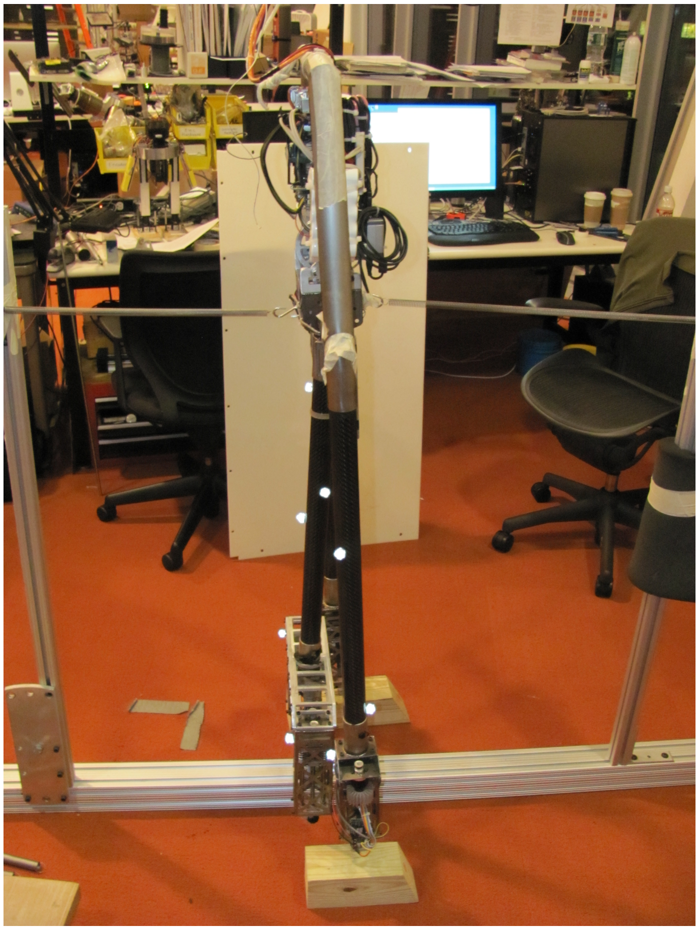


Figure 4-3: The robot setup during system identification. Note the springs stabilizing the system at the equilibrium point.

dynamics in this case are a very simple degenerate form of the full equations of motion and doesn't even include actuation, the robot is only excited by its initial conditions. This test represents all of the physical parameters, but in a way in which they can't be individually identified, and makes a good independent check on all of the other tests.

In performing the system identification a lot of expertise in the process of producing good data sets was amassed. Someone performing the process again will likely re-learn a lot of these points in their own experimentation, but hopefully the right track will be found a little more quickly.

One of the things we noticed is that initial conditions and zeroing can play an undesirably large role if the experiment design allows them to. This is especially bad

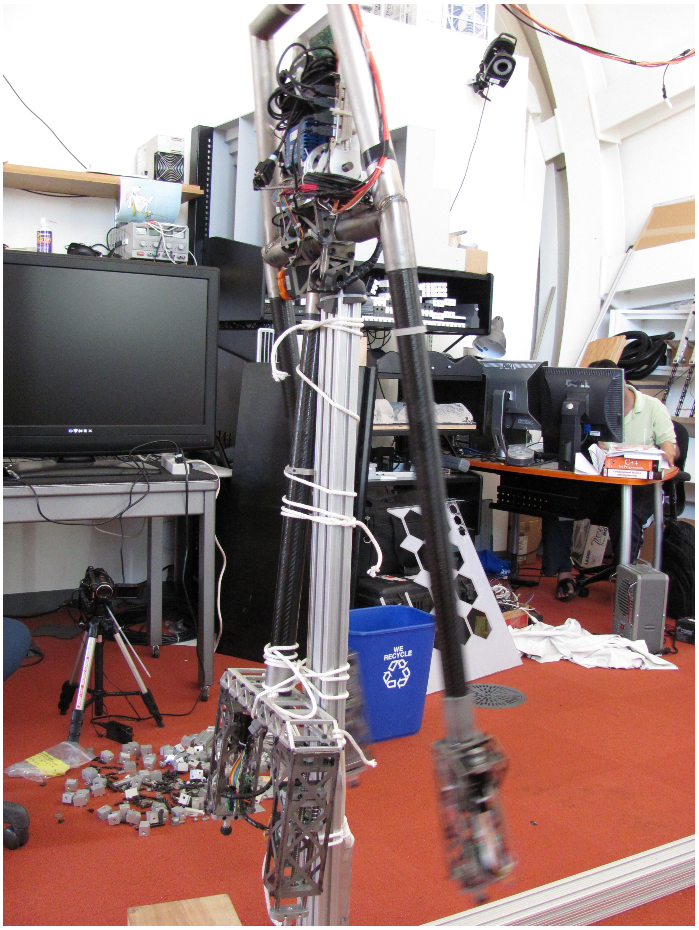


Figure 4-4: The robot with one of the legs fixed, producing a stable configuration and focusing on a subset of the system's dynamics.

because there is necessarily some variation in how the system is set up each time. For example, if the system is initially standing at the upright equilibrium then it can fall in either direction and the dynamics evolve slowly from that point. It would be much better to start from some known position in which the speed and direction with which the motion evolves is less dependent on the initial conditions like zero velocity at some point far from the origin.

A similar point applies for parts of state space reached during the test. Unstable equilibria, where the motion can evolve in two different directions depending on some very small change in state, like being near zero velocity at the upright, can make it very difficult for the simulation error optimization to converge. Say the optimizer wants to add a tiny bit of inertia to the stance leg, in this situation at some point it will mean the difference between making it over the upright position and not. This produces cliffs in the cost landscape which, while they reflect the problem that has been posed to the optimizer, aren't really a part of the identification problem you want to solve. If only the experiment had been designed to carry more velocity through the upright position, or avoid it entirely, then the optimizer would be able to smoothly vary that inertia, allowing the whole set of parameters to slide into place more easily.

We also spent a fair bit of time trying to identify the system when it's closed loop stable, after we had produced a working balancing controller. While technically this kind of setup can work, in systems like this robot and many of the robots we work with it doesn't have a chance. This is because the control action required to keep the robot upright is extremely strong and the nonidealities of the control loop corrupt the response a great deal. These nonidealities are things such as delays, the frequency response of the motor and controller, unmodeled high frequency dynamics, and the frequency responses and drift characteristics of the sensors used. All of these items have their own parameters which a more thorough system identification would be interested in, but really need to be handled in smaller, more focused experiments.

A point that is easy to overlook, but may be more important than any other point here is that utmost care needs to be taken in how the system identification data is handled. Because there is a lot of data coming from many different tests and setups every detail needs to be recorded and confirmed. Several times during the experiments and various re-fits of the data and re-runs of experiments we lost confidence in previous data sets because of very simple things, like not being sure that the sensor zeroing was performed properly, or what the sampling rate was because it had been changed at some point, or if the sampling rate was maintained properly throughout a test run. This means complete record keeping whether in the log files themsevles or in another notebook. Confidence in the working set of data is extremely important because it means less work will need to be repeated, and there are fewer incorrect rationalizations to be made when the analysis turns out bad. Being able to look back at the data set and know for sure that the analysis is wrong instead of the data being wrong is invaluable.

One last note on the system identification process is that while ideally the basis functions are capable of representing the physical system perfectly, this is almost never true thanks to difficult to model pieces like joint friction. As a specific example, when identifying the friction drive elbow joint of the acrobot we used viscous, coulomb, and quadratic friction terms to obtain a very good fit in the area of state space we collected data on. This included mostly medium velocity and long, smooth motions of the robot. When we wanted to start using the model we identified for balancing control it turned out to be wildly inaccurate because the near zero velocity regime for the joint friction looks very different from where we collected data and having incorrect basis functions, the the model didn't generalize to other velocities well. The point to be taken here is that data should be collected in *exactly* the regions of state space you want the model to be accurate in, close often will not cut it when the basis functions aren't true to the real dynamics.

4.3 Observer Design

While both positions are measured by the robot's sensors as previously described, only one of the velocities that make up the full state is measured. The IMU produces position and velocity of the stance leg, but the swing leg velocity must be deduced from a position encoder only. Previous experience with similar platforms has shown that the low pass filtering required on the quantized position signal from the encoder produces results in two competing nonidealities. Either a delay that causes instability in the kind of high gain controller required for balancing or the remnants of enough noise that the controller produces unacceptable vibration must be accepted.

To work around this problem a discrete, full order high-gain observer is used to produce the velocity measurements. It's possible to use the nonlinear system model inside the conventional fixed gain observer structure as described in Equations 4.14, 4.15 and 4.16 in order to produce more accurate observations with the tradeoff that the observer dynamics are harder to analyze. In this case the nonlinear observer can be much better because it accommodates the nonlinear friction model. Luckily, if the

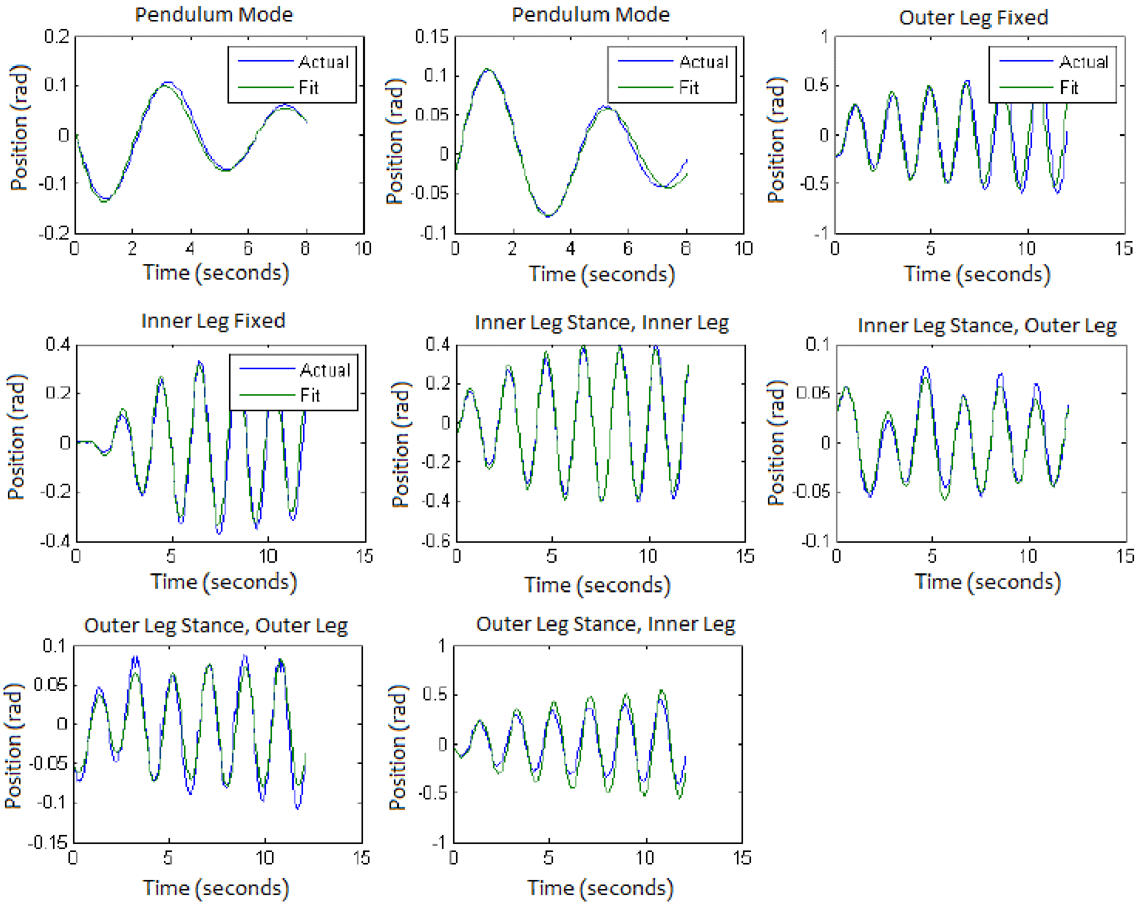


Figure 4-5: The final system identification fits, compared against the training data.

gains and system assume the form described by Khalil shown in Equation 4.17 the system has good stability properties[15]. The ϵ term in the gain matrix provides a convenient way to tune the time constant of the observer, in practicality to balance between disturbance rejection and the nice low-pass/low-delay characteristics of the observer that kill off sensor noise and unmodeled high frequency dynamics. In practice the observer estimates both velocities very well and the velocity from the IMU is actually dropped from the measurement vector which is reflected here and the value $\epsilon = 0.2$ was used.

$$\mathbf{y} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \tag{4.14}$$

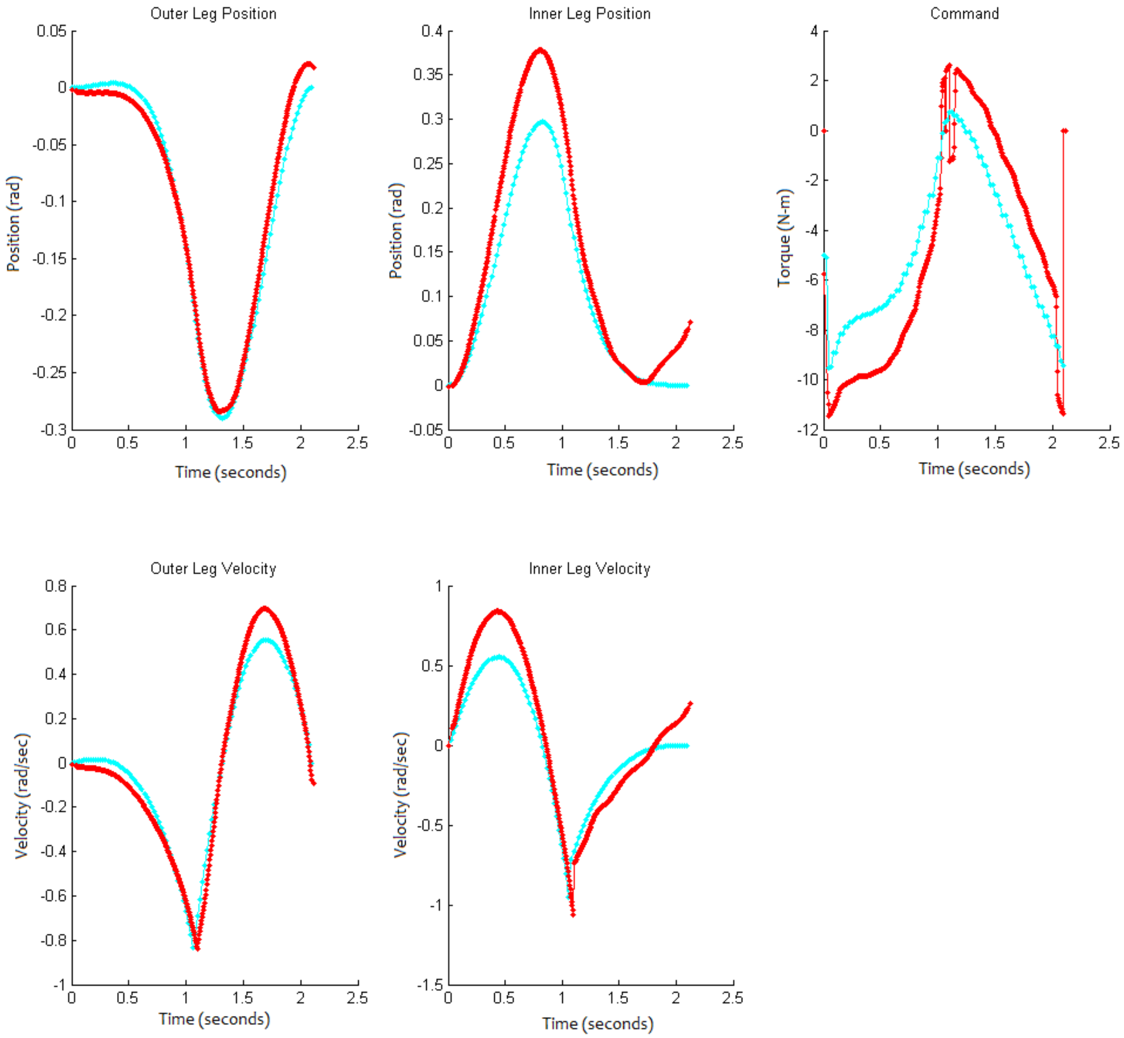


Figure 4-6: Time plots of the real robot executing a trajectory planned with the identified model for model validation. The actual command differs from the nominal because friction was not included in the model used by the planner. The friction compensator used was also identified using the same techniques.

$$\dot{\mathbf{x}}[k+1] = f(\mathbf{x}[k], \mathbf{u}[\mathbf{k}]), \mathbf{y}[k] = \mathbf{C}\mathbf{x}[k] \tag{4.15}$$

$$\hat{\mathbf{x}}[k+1] = \hat{\mathbf{x}}[k] + T(\hat{\dot{\mathbf{x}}}[k] + L(\mathbf{y}[k] - \hat{\mathbf{y}}[k])) \tag{4.16}$$

$$\mathbf{L} = \begin{bmatrix} \frac{2}{\epsilon} & 0 \\ 0 & \frac{2}{\epsilon} \\ \frac{1}{\epsilon^2} & 0 \\ 0 & \frac{2}{\epsilon^2} \end{bmatrix} \tag{4.17}$$

4.4 Harmonic Drive Compliance

An issue that showed up in every control experiment was the excitation of unmodeled high frequency dynamics when feedback gains were pushed high enough. This was most notable in the case of the balancing controller which wasn't able to stabilize the system until the gains were raised past a specific threshold, well past the point at which the high frequency dynamics were shaking the robot apart. In contrast to Figure **??**, Figure 4-7 shows the time response of the system with the balancing controller when steps to combat the unwanted excitation aren't taken. Fourier transforms for the torque signal in each case are shown in Figures 4-8 and 4-9. Note the large peak just above 10Hz in the second plot as compared to the DFT of the good response.

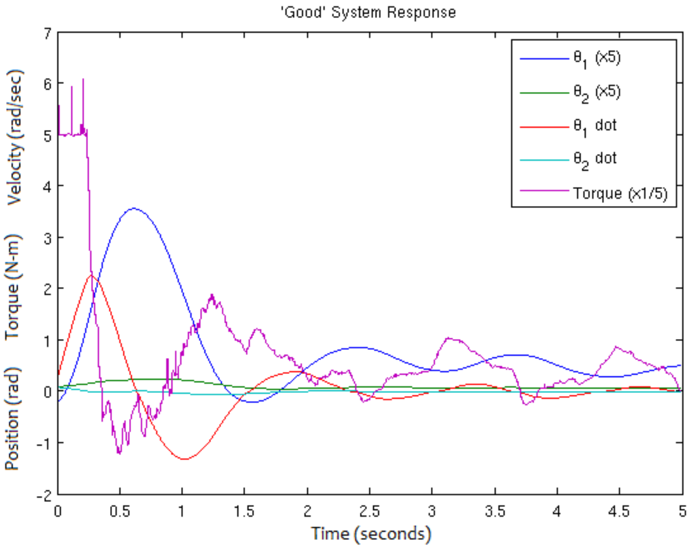


Figure 4-7: A typical time response of the balancing controller when switched on with a nonzero initial condition.

The tricky issue with this situation was that without a couple very specific mea-

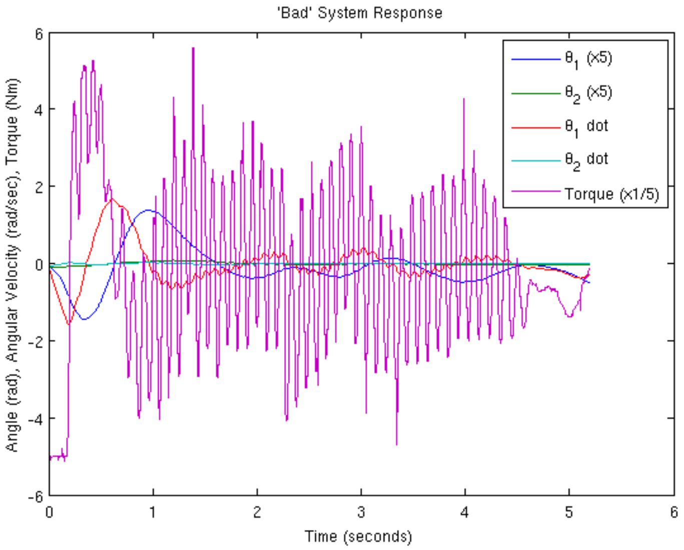


Figure 4-8: Time response of the balancing controller without steps taken to attenuate high frequency dynamics.

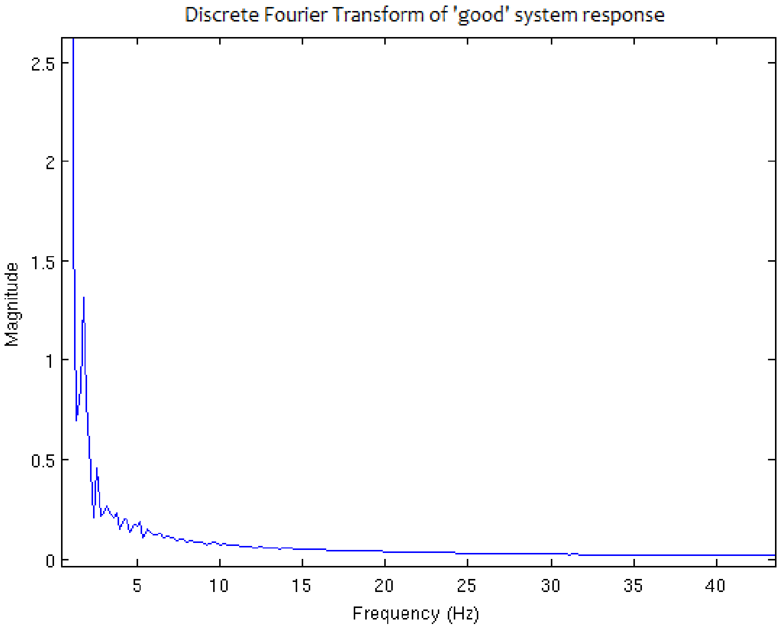


Figure 4-9: Discrete Fourier transform of the torque signal from the good balancing response.

sures it wasn't feasible to produce a working balancing controller for the robot. A first order IIR filter on either the input or the output is able to block the unwanted high frequencies, but brings with it too much delay and prevents successful balancing. A much more selective filter (a fifth order Chebychev Type I filter was one of many

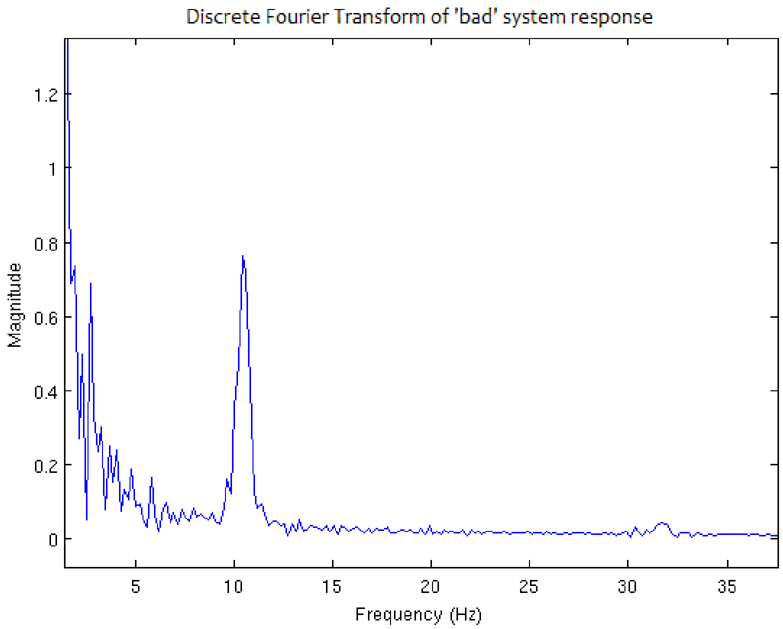


Figure 4-10: Discrete Fourier transform of the torque signal from the balancing response with unwanted high frequency dynamics.

tried) is capable of better performance, but wasn't able to solve the issue. The key piece of the rejection puzzle was the state observer.

The process of 'tuning up' the state observer initially involved selecting an ϵ which produced smooth, reasonable looking velocities from the position data in a few simple test cases of moving the robot around by hand. This led to the very small value of $\epsilon = 0.02$, placing most of the emphasis on the measurements rather than the system model. This would normally be a good thing because it means unmodeled dynamics like the robot getting pushed unexpectedly are able to come through. The unwanted high frequency dynamics are treated the same way by the observer, passing them through from the sensors to the output.

If the observer has a perfect model of the system the sensor signals would pass right through unaltered because the internal model and sensors would always be in perfect agreement. Anything that isn't part of the observer's model gets attenuated to some extent, in essence in addition to generating the velocity states the observer is a 'model-pass filter'. It lets though things not based on a specific frequency range but based on what it expects to see at the inputs. This is exactly what is needed in this case. Because the model the observer operates on is very accurate it's possible to turn ϵ all the way up to 0.2, relying heavily on the model and leaving the unwanted high frequency dynamics that exist in real life, but not in the model, at the door.

The main disadvantages to this approach are that it rejects all disturbances, even those wanted in the state like a push from an external source, and that it isn't robust to model changes. Usually the observer's internal model is treated more as a general guide than the ground truth so they tend to perform well even with inaccurate models, but that is the not case when used in this manner. Nevertheless, it solved the problem at hand surprisingly well when conventional filtering methods failed.

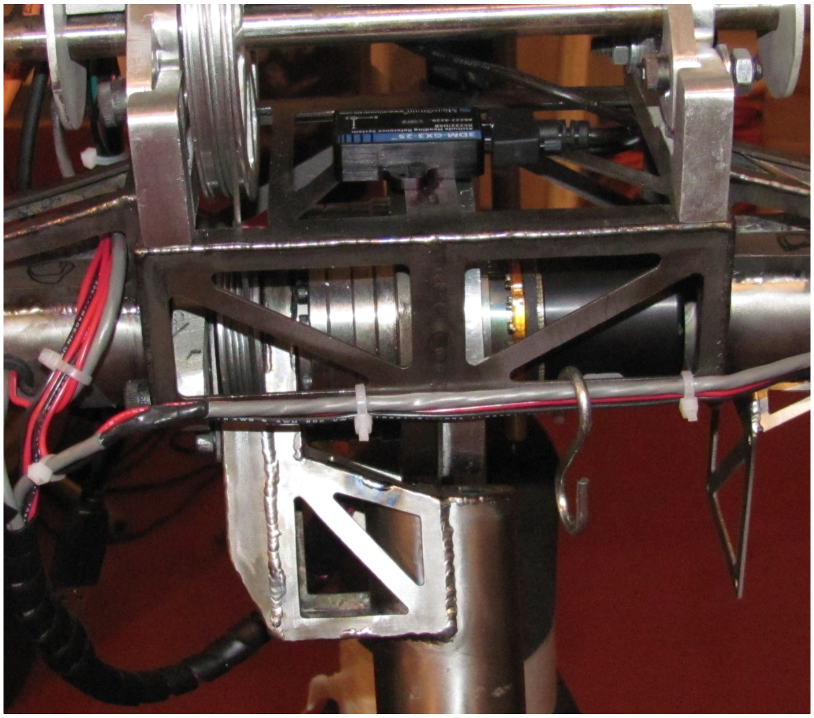


Figure 4-11: The bracket the IMU is attached to, possibly flexing as the hip motor applies large torques to the structure.

A final question to ask about the high frequency dynamics is where they are coming from. Originally they were thought to be coming from flexing of the IMU mount which is very close to the motor as shown in Figure 4-10. It's possible that when large torques are applied the IMU moves in relation to the rest of the structure. To test this hypothesis a conventional frequency sweep was performed, looking at the magnitude response between the motor input and the robot's various sensors. The robot was stabilized upright by hand without holding it tightly and a $5N - m$ chirp between 1 and 20Hz was applied over a duration of several minutes. The recorded signal was high pass filtered at 1Hz in order to eliminate slow movement of the robot as a whole, the absolute value taken, and finally low pass filtered in order to extract the shape of the magnitude response from the high frequency sinusoid used to drive the system. The result shown in Figure 4-11 was somewhat surprising as it didn't show significant peaking at 10Hz, but did show a significant drop off right around that point.

When the original hypothesis didn't seem to hold up the search for the true cause was expanded and the same treatment was performed on the system of torque to the inter-leg angle encoder. This data is shown in Figure 4-12. In contrast to the IMU mount this data shows a noticeable peak right at the target frequency. The current hypothesis as to what is happening here is that the harmonic drive flex spline is the compliance that is getting excited. This was further confirmed (somewhat haphazardly) by the addition of mechanical damping to the motor rotor which put an abrupt end to the oscillations.

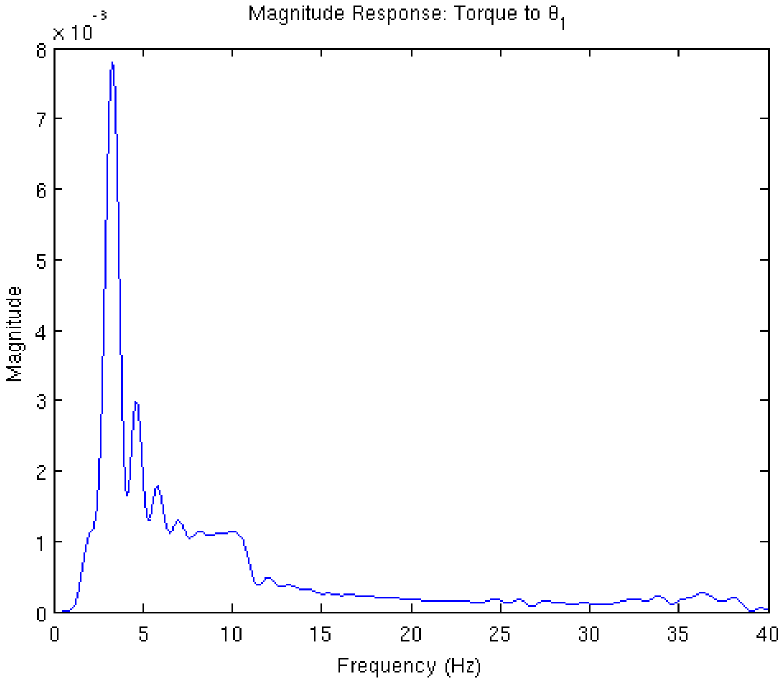


Figure 4-12: Frequency response of the torque to IMU flex system.

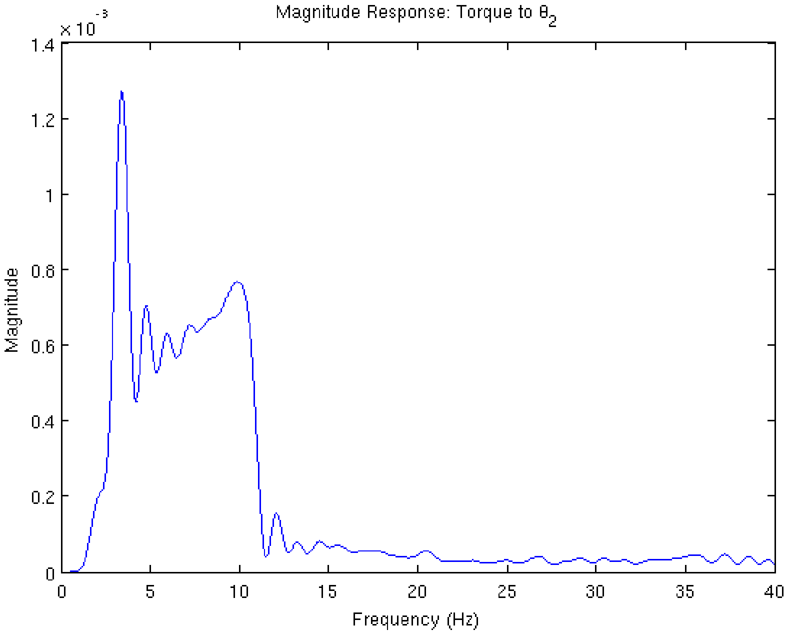


Figure 4-13: Frequency response of the torque to inter-leg angle encoder system.

4.5 TVLQR Stabilized Trajectories

Time Varying LQR control for the stabilization of trajectories has become a staple solution for nonlinear systems in the lab and was the second walking solution we attempted to get working on the real robot. TVLQR tackles the problem of control of a nonlinear system by linearizing the plant around a predefined trajectory. As a test problem to work on we chose the 'one step rebalance' task, starting from equilibrium conditions on one leg the robot takes a step and attempts to get into the balancing controller on the other leg. This was done with the robot raised up on blocks so the feet don't need to be actuated.

One of the main failings of TVLQR is that it runs on a predefined clock. What this means is that the controller not only needs to regulate the system to a specific state, but it's trying to get there in a specific amount of time which is an unnecessarily difficult control problem in most cases. This is particularly bad for hybrid systems because the linearization that TVLQR uses is also indexed with time which means that it's possible that the controller gains are completely wrong for the system's equations of motion. This is particularly bad for this system where the sign

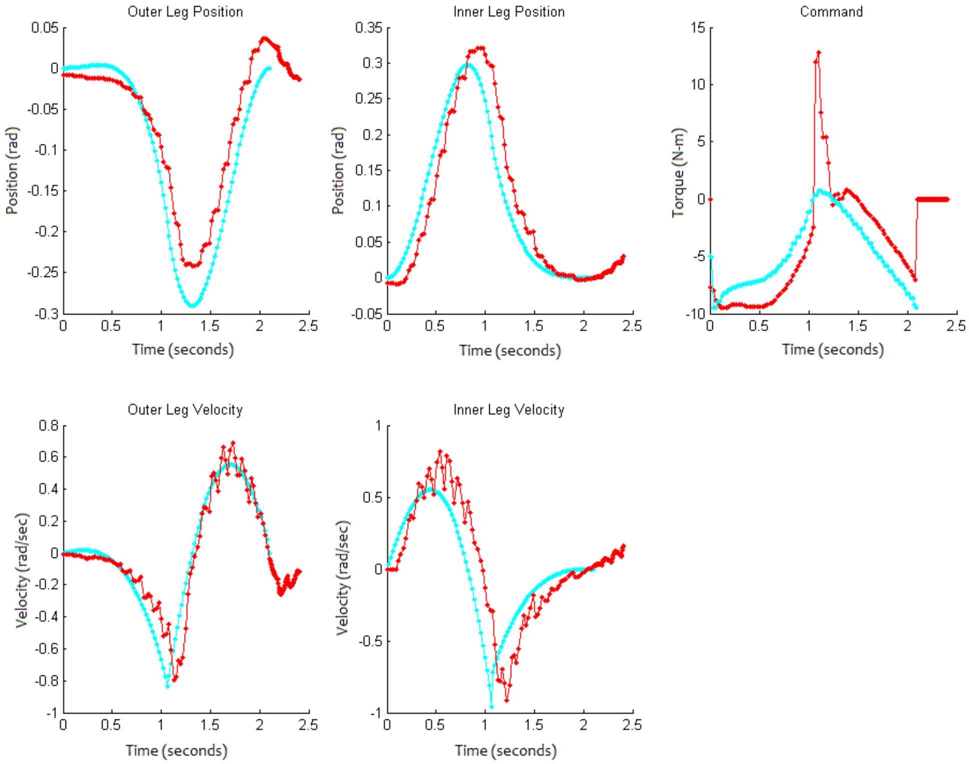


Figure 4-14: TVLQR stabilized 'one step rebalance' trajectory. The small ripple along the trajectory is due to the very high controller gains exciting unmodeled high frequency dynamics in the system. Nominal trajectory is in cyan while the recorded data is in red.

on the control output changes depending on which stance leg the robot is on, if the robot doesn't impact the ground exactly when the controller expects it to then it gets thrown into positive feedback resulting in spectacular failure. We chose to combat this problem by disabling the controller feedback near the expected impact point in time which helped to minimize the problem, figure 4-13 shows this well. The trajectories are slightly misaligned in time, enough that the time around where feedback is disabled is exceeded resulting in a short lived spike in the command signal until the real system came into line with the model.

Another important topic to discuss on TVLQR as it relates to underactuated system like this robot is the issue of cost tuning. The state cost matrices $\mathbf{Q}$ final cost $\mathbf{Q_f}$ provide a lot of knobs to turn in developing controllers for specific application allowing the designer to weight how the controller values errors in each of the systems and couplings between those states but this freedom is a double-edged sword in that it makes it possible to make bad decisions in addition to good ones. Simple strategies like weighting all of the states similarly which often works in fully actuated systems can produce extremely bad controllers for systems like the compass gait as measured by the size of the controller's basin of attraction. Finding the combination of state costs and final state costs which produce the best controllers takes a combination of design intuition and brute force.

Design intuition in the selection of controller costs is important because in the case of an underactuated system errors in state *must* be traded off against each other in order to nail the desired final state. High costs need to be put on states that are important to the system linearization and long term stability, so the stance leg position gets the highest cost, followed by the swing leg position second, and the two leg velocities as a very, very distant third and fourth. This allows the controller make the correct decision to give up lots of state error in the swing leg in order to regulate errors in the stance leg which is the ultimate decider in whether you're still standing up. In addition to this, the final costs need to be very high in relation to the costs along the trajectory in order to allow the controller to make significant excursions from the nominal in order to actually get close to the final states, presumably the state that really matters. The danger here is that the further off the nominal trajectory the plant goes in reality the more incorrect the linearization is, so some cost along the trajectory is necessary to keep it reasonably close.

4.6 Simulation LQR-Trees

The LQR-Trees algorithm, originally described by Russ Tedrake in his 2009 paper [19], provides a way to fill a robot's state space with a sparse tree of trajectories leading to a goal point or trajectory. This is useful for situations where you wish to bring a complicated nonlinear system such as the acrobot to an equilibrium, but perhaps more importantly, bringing a similar system into a limit cycle like walking.

The algorithm combines aspects of rapidly exploring randomized trees with trajectory planning and TVLQR stabilization of the trajectories. This combination of tools isn't very interesting in itself because it's impossible to know what parts of the space the existing trajectories already cover, but with tools recently developed it's possible to figure out how large a volume around a trajectory can be brought to the goal point by that trajectory. This makes it possible to figure out where new trajectories are needed and to avoid volumes that are already covered but the code for doing this is highly complex to implement and at its current stage of development, quite fragile.

An alternative to the formal verification methods was also developed along with the algorithm which uses simulations instead of mathematical proofs to produce nonconservative estimates of the basins of attraction though falsification [17]. Reist's method relies on dividing the tree of trajectories into nodes, each of which represents a discrete time step in the trajectory it belongs to. These nodes contain their location in state space, nominal control signals, control gains, a Lyapunov function, and scalar ρ which marks the level set of the Lyapunov function which the controller is known to stabilize.

Instead of finding a value for ρ when the trajectory is first added to the tree as in the original LQR-Trees algorithm, ρ is initialized to be infinite. On each major loop of the algorithm a random point in the state space is sampled and the tree is asked if any of the nodes in it claim to be able to bring that point to the goal. For each node that claims this a TVLQR controller is constructed starting with that point and running to the goal and then simulated, if the simulation ends within the basin of attraction of the time invariant controller it is considered successful and a new major iteration is started. If the simulation fails, the edge of that node's basin is cut down to where the trajectory is at the point in time where it is active. As more simulations are performed the basins will converge on a nonconservative estimate of where they are valid.

The primary addition of this work is the extension to hybrid systems. The situation as presented with this robot is generating a tree for getting the robot back to the upright equilibrium on a specific stance leg, so the problem has four continuous states and one discrete state for which leg the robot is standing on. The robot can be in either of the two stance leg states and should be able to take multiple steps if necessary to get back to the goal, meaning that even if it's on the target leg the robot should be able take some multiple of two steps to get back into the equilibrium state if it's not feasible to get directly there. There are a couple major concerns with hybrid systems as they relate to LQR-Trees, the lack of a distance metric for states between plant modes, and the need to specify a schedule of plant modes for the direct collocation method of trajectory optimization used [6].

The solution here was brute force but effective. The tree nodes were extended to also track which plant mode their state belongs to and the distance metric the tree uses was modified to report states in a different mode as infinitely far away. This means that the tree would never attempt to build trajectories between plant modes, but if trajectories already existed which crossed between modes then the tree would be able to grow to the nodes in the matching mode on either side of the switching surface. In order to populate the tree with trajectories which cross the switching surfaces the extend operation makes multiple attempts for each sample point with different mode schedules.

1. If the sample state is the same as the target state attempt to grow to the nearest node.

2. If this isn't feasible, attempt to take a step into the other plane mode and back to the goal state.

1. If the sample state not in the same mode as the target state first look for any nodes which connect back from the sample mode and try to grow to the closest.

2. If this isn't feasible, attempt to connect directly back to the goal node in the other mode.

This method allows the tree to be well populated by trajectories that cross the switching surface without having a distance metric by specifying a known good node instead. While this results in excess trajectory density around the goal, the effect should be greatly lessened with a trajectory optimizer capable of optimizing over the tree end constraint which is formally part of the algorithm but was omitted in this case for ease of implementation. The results from this extend strategy can be seen in Figure 4-14 which shows a pair of hybrid trees, one for each possible stance leg. This is because in reality we don't care which leg the robot ends up standing on once it's back at equilibrium, but each requires its own tree because the robot is asymmetric.

The major time sink in running the algorithm is checking whether a specific sample and closest node are feasible to connect. The only way to figure this out is to run the trajectory optimizer until either a time limit is exceeded or it reports that the problem is infeasible, but we found it can take up to 30 seconds to produce a trajectory of reasonable accuracy. Taking 30 seconds to decide a sample is infeasible isn't a winning strategy when tens of thousands of points need to be processed to map out the edges of the feasible space. One way around this is recognizing that assessing the feasibility of a trajectory, a yes or no answer, isn't very dependent on the trajectory produced being very accurate. To make use of this a two step trajectory optimization strategy is employed where the first optimization run uses a very small number of knot points, one about every quarter second, followed by an accurate optimization with a large number of knot points, seeded with the output of the first. This allows feasability to be tested in about one second on average and only sinking the time to develop the full accuracy when it's likely to have a useful result.

Along with this issue, it became very clear that the claims the algorithm makes about filling the feasible state space need to be heavily qualified in practice. While randomness in the trajectory optimizer initialization makes this technically true, in this case the direct collocation method used often failed to produce a trajectory using known good start and end points several times before eventually finding an initialization it likes. This means that it may take an extremely long time to produce something that looks like it covers the space. In the case of Figure 4-14 the algorithm was run for 48 hours, sampling 306422 points with 45 of them resulting in successful trajectories. This performance is expected to improve once the final tree constraint is allowed to move up and down the tree.

The reason why experimental results haven't been produced for the LQR-Trees experiment yet has to do with finding which node the robot is closest to quickly. The distance metric is based on evaluating the LQR Cost To Go between the robot's current state and the tree node which is currently done for every node on the tree and then sorted. For the tree shown here this takes long enough that the robot is in a completely different part of the tree. In the past this has been solved by running the robot's dynamics forward for as long as it's expected to find where the robot is in the tree and finding the nearest node to that state. The problem is that in this case the robot usually falls over far enough in that time to be unrecoverable. Work is currently being done to improve the speed of this operation including simple code optimization and making use of additional information available about which nodes are relevant to look at to reduce the number of calculations necessary.

4.7 Transverse Stabilized Walking

Transverse stabilization is a control strategy much like TVLQR, but completely different. Rather than indexing the trajectory in time it's indexed off a phase variable τ, freeing it from time, much like the virtual constraints controller mentioned before except without a single state of the robot being the driving signal. This control strategy has been developed in large part by Ian Manchester, who showed it working on the simple compass gait robot mentioned in Section 1 [13] and played a major part in making it work on this robot. His paper contains a full discussion of the technique. Because the controller isn't explicitly indexed off time the previously mentioned problem of the mode switch with TVLQR is no longer an issue.

We would like to demonstrate LQR Trees working with this transverse control with the real robot as had been shown only in theory. Toward that goal we constructed the most simple realization of the tree, the periodic orbit with a single recovery maneuver, a step in from the upright equilibrium. A periodic walking trajectory was planned using the previously mentioned DIRCOL code along with transversal

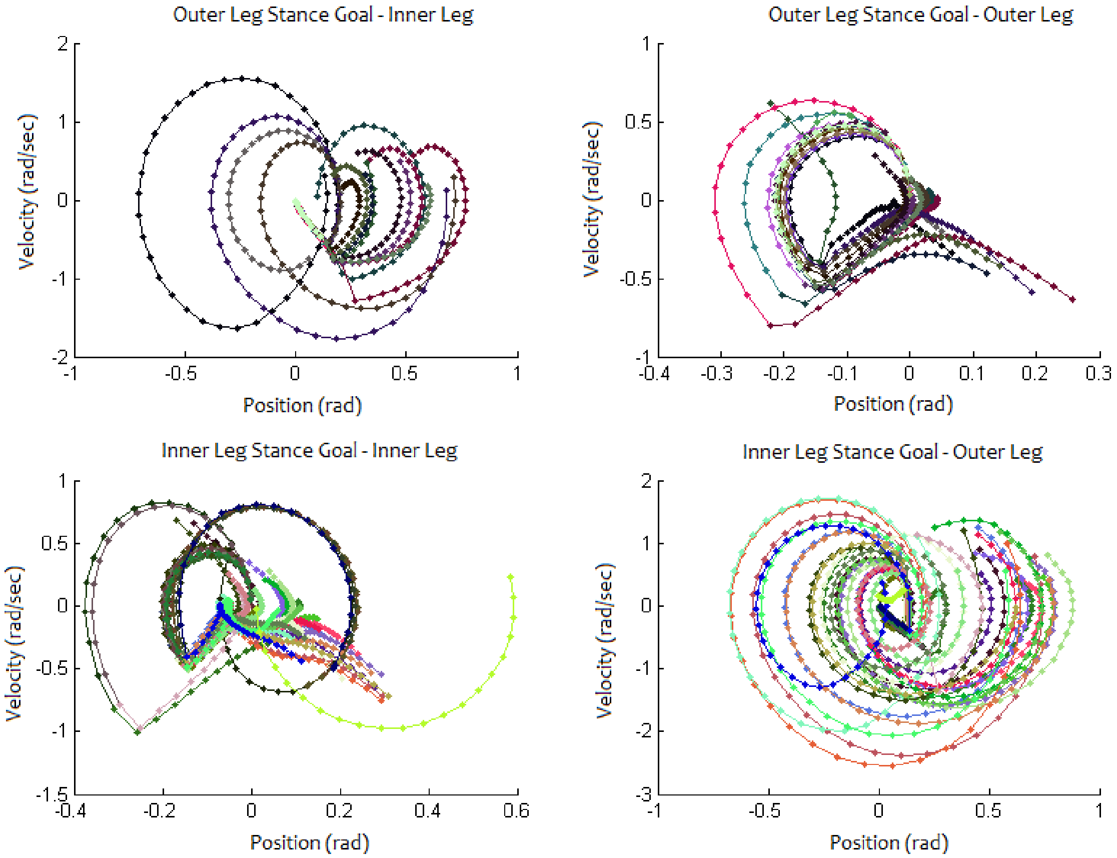


Figure 4-15: Plots of the hybrid LQR Tree controller designed to bring the robot to equilibrium on either of the stance legs.

surfaces and a transverse controller to stabilize the trajectory. A second trajectory from the upright equilibrium on the outer legs to the impact state in the periodic trajectory was planned similarly, along with matching the transversal surface at the end of the trajectory with the impact surface just as with the periodic trajectory. The experiment results are shown here in Figure 4-16 for the first two pieces of the trajectory, the step in and first periodic phase.

A couple important notes from the trajectory. First, the difference in how the trajectory follows time should be noted. The time indexed nominal state trajectories are plotted in cyan while the the τ indexed state trajectories which the controller is regulating to are shown in blue. Second, the controller is able to handle the model errors seen previously in the TVLQR response plots more naturally. Model errors that cause the trajectory to take slightly longer or shorter in time don't impact the

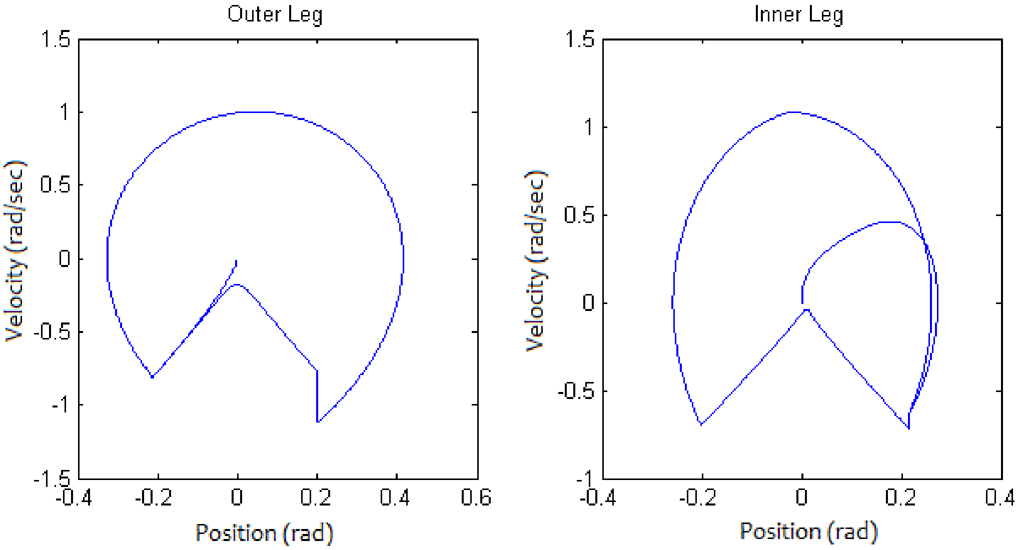


Figure 4-16: The planed periodic trajectory with the step-in trajectory from equilibrium.

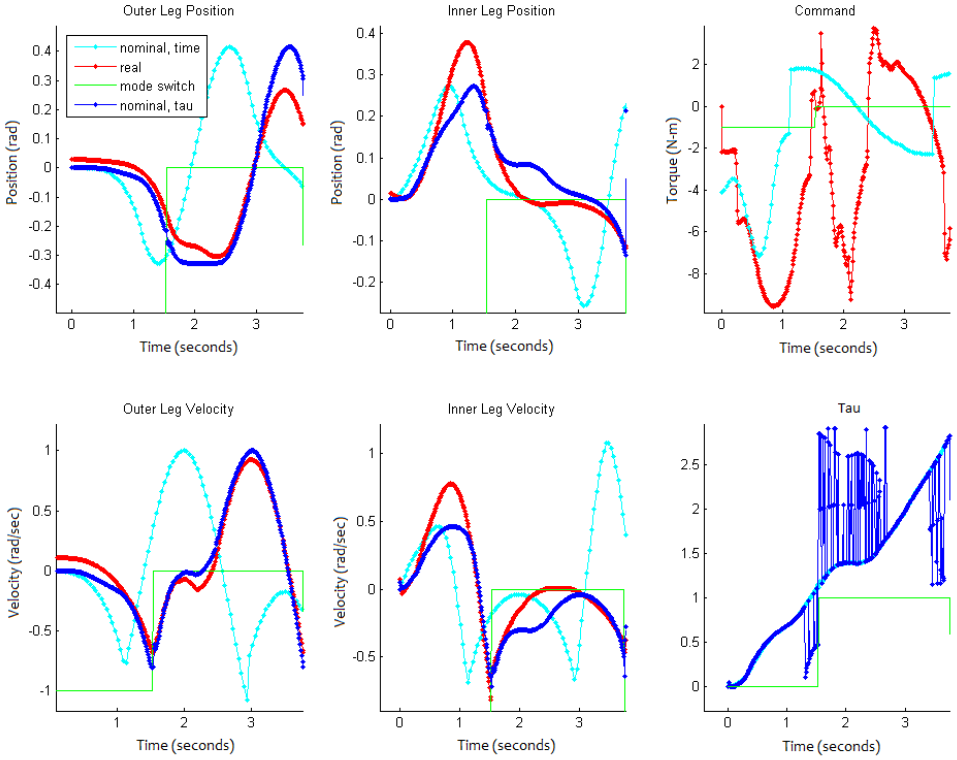


Figure 4-17: The TVLQR stabilized step-in and first step of the periodic trajectory as run on the real robot.

rest of the trajectory, instead the trajectory slides into place as it's needed by the actual robot's dynamics. In applications where the state of the system matters while the specific time that state happens at doesn't, walking instead of catching a ball for example, the transverse controller presents a significant performance and robustness advantage.

In process of working with Ian Manchester to bring the transverse controller from its largely theoretical status to working on arbitrary systems several sore spots were uncovered, specifically in the optimization of the transversal surfaces along the trajectory which define the space the controller operates in and how the progress of the robot along the nominal trajectory is tracked by the controller. The last plot in Figure 4-16 shows the progress of the τ phase variable via the previous method used to calculate it (blue) and a new method based on an observer first attempted on this robot (cyan). The actual work that went into bringing the controller into the condition of working well on this robot will be published along with more complete results of the transverse stabilized walking experiment shortly following this thesis. The limited results here are largely due to a lack of time rather than problems with the method.

Chapter 5

Conclusion

This thesis has presented the development of a compass gait walking robot from concept through experiments, focusing on the interaction between mechanical design choices and the control of the system with several aims. The first of these is the application of several new ideas in control including LQR-Trees and transverse stabilization to a full robotic system with all of the nonidealities of sensing and system modeling issues. While not explicitly the focus of the writing here, the issues experienced have been very helpful in provoking the development of these control strategies past the theory stage and into a stage of development compatible with widespread implementation. Further, the difficulties exposed have been fed back into the research process, guiding the development of theory into areas where it's weak such as the high reliance on an accurate system model and the limitations of performing complex control strategies in real time.

The second of these aims is to further the development of high performance and highly dynamic robotics as a field of design. This thesis should provide a good feel for the important objectives and pitfalls specific to designing highly dynamic mechanical systems and the way the mechanical design is inseparable from the control strategy. This is especially the case when heavily model reliant methods are used such as here, many core design decisions rested on how amenable mechanisms were to modeling and how they impacted model complexity.

Going forward, I hope that my contributions in the hardware platform and the software system design that support it continue to produce useful research results for years to come and provide direction for new designers in their own design of high performance dynamic robots.

Bibliography

[1] Haruhiko Asada and Kamal Youcef-Toumi. *Direct-Drive Robots - Theory and Practice*. The MIT Press, 1987.

[2] Humanoid Robot Research Center. Introduction of khr-3(hubo). *http://hubolab.kaist.ac.kr/KHR-3.php*.

[3] C. Chevallereau, G. Abba, Y. Aoustin, F. Plestan, E. R. Westervelt, C. Canudas-De-Wit, and J. W. Grizzle. Rabbit: a testbed for advanced control theory. *IEEE Control Systems Magazine*, 23(5):57–79, Oct. 2003.

[4] Boston Dynamics. Bigdog - the most advanced rough-terrain robot on earth. *http://www.bostondynamics.com/robotbigdog.html*.

[5] Grizzle, JW, Hurst, J., Morris, B., Park, H.W., Sreenath, and K. Mabel, a new robotic bipedal walker and runner. In *American Control Conference, 2009. ACC'09.*, pages 2030–2036. IEEE, 2009.

[6] C. R. Hargraves and S. W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *J Guidance*, 10(4):338–342, July-August 1987.

[7] Hobbelen, D., de Boer, T., Wisse, and M. System overview of bipedal robots flame and tulip: Tailor-made for limit cycle walking. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2486–2491. IEEE, 2008.

[8] Honda. The honda humanoid robot asimo. *http://world.honda.com/ASIMO/*.

[9] Albert S. Huang, Edwin Olson, and David C. Moore. Lcm: Lightweight communications and marshalling. *International Conference on Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ*, pages 4057–4062, October 2010.

[10] Fumiya Iida and Russ Tedrake. Minimalistic control of a compass gait robot in rough terrain. In *Proceedings of the IEEE/RAS International Conference on Robotics and Automation (ICRA)*. IEEE/RAS, 2009.

[11] Karssen and J.G.D. Design and construction of the Cornell Ranger, a world record distance walking robot., 2007.

[12] Harmonic Drive LLC. Operating principles. Website. `http://www.harmonicdrive.net/reference/operatingprinciples/`.

[13] Ian R. Manchester, Uwe Mettin, Fumiya Iida, and Russ Tedrake. Stable dynamic walking over rough terrain: Theory and experiment. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2009.

[14] Microstrain. 3dm-gx3 specifications. Website. `http://www.microstrain.com/productdatasheets/3DM-GX3-25datasheetversion1.06.pdf`.

[15] S. Oh and H.K. Khalil. Nonlinear Output-Feedback Tracking Using High-gain Observer and Variable Structure Control*, 1. *Automatica*, 33(10):1845–1856, 1997.

[16] Gill A. Pratt, Matthew M. Williamson, Peter Dillworth, Jerry Pratt, Karsten Ulland, and Anne Wright. Stiffness isn't everything. In *Proceedings of the 4th Iternational Symposium on Experimental Robotics (ISER)*, 1995.

[17] Philipp Reist and Russ Tedrake. Simulation-based LQR-trees with input and state constraints. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010.

[18] Sreenath, K., Park, H.W., Poulakakis, I., Grizzle, and JW. A compliant hybrid zero dynamics controller for stable, efficient and fast bipedal walking on MABEL. *International Journal of Robotics Research*, 2010.

[19] Russ Tedrake. LQR-Trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems (RSS)*, page 8, 2009.

[20] Russ Tedrake. *Underactuated Robotics: Learning, Planning, and Control for Efficient and Agile Machines: Course Notes for MIT 6.832*. Working draft edition, 2010.

[21] ThinGap. Brushless tg2310 bldc motor. Website. `http://www.thingap.com/pdf/tg2310series.pdf`.

[22] Martijn Wisse. *Essentials of dynamic walking; Analysis and design of two-legged robots*. PhD thesis, Technische Universiteit Delft, 2004.

[23] Xie and Y. Dynamic effects of an upper body on a 2d bipedal robot. *Master's thesis, University of Twente*, 2006.

Appendix A

System Model Derivation

```
In[7]:=  phip = {Sin[-θ 1] * l, Cos[-θ 1] * l} (* Position of hip joint*)
         SetAttributes[ l, Constant]

Out[7]=  {-l Sin[θ 1], l Cos[θ 1]}
```

```
In[11]:=  pm1 = phip - {Sin[-θ 1] * (lc1), Cos[-θ 1] * (lc1)} (* Position of mass 1 *)
          SetAttributes[lc1, Constant]

Out[11]=  {-l Sin[θ 1] + lc1 Sin[θ 1], l Cos[θ 1] - lc1 Cos[θ 1]}
```

```
In[13]:=  pm2 = phip + {-Sin[θ 2] * lc2, Cos[θ 2] * lc2} (* Position of mass 2 *)
          SetAttributes[lc2, Constant]

Out[13]=  {-l Sin[θ 1] - lc2 Sin[θ 2], l Cos[θ 1] + lc2 Cos[θ 2]}
```

```
               (θ 1+ θ 2)
In[15]:=  θ 3= ──────────;(* Introduce bisection: q3=(q1+q2)/2 *)
                   2
```

```
In[16]:=  pm3 = phip + {-Sin[θ 3] * lc3, Cos[θ 3] * lc3} (* Position of mass 2 *)
          SetAttributes[lc3, Constant];

                                        θ 1+ θ 2                        θ 1+ θ 2
Out[16]=  {-l Sin[θ 1] - lc3 Sin[ ──────── ], l Cos[θ 1] + lc3 Cos[ ──────── ]}
                                           2                               2
```

```
In[18]:=  U = Simplify[m1 * g * pm1[[2]] + m2 * g * pm2[[2]] + m3 * g * pm3[[2]]]
          (* Total Potential Energy of the system *)
          SetAttributes[{ m1, m2, m3, g}, Constant]

                                                                                      θ 1+ θ 2
Out[18]=  g ((-lc1 m1 + l (m1 + m2 + m3)) Cos[θ 1] + lc2 m2 Cos[θ 2] + lc3 m3 Cos[ ──────── ])
                                                                                          2
```

```
In[20]:=  q = {θ 1, θ 2};
          dq = Dt[q, t]

Out[21]=  {Dt[θ 1, t], Dt[θ 2, t]}
```

```
In[22]:=  vm1 = D[pm1, {q}].dq

Out[22]=  {(-l Cos[θ 1] + lc1 Cos[θ 1]) Dt[θ 1, t], Dt[θ 1, t] (-l Sin[θ 1] + lc1 Sin[θ 1])}
```

```
In[23]:=  vm2 = D[pm2, {q}].dq

Out[23]=  {-l Cos[θ 1] Dt[θ 1, t] - lc2 Cos[θ 2] Dt[θ 2, t], -l Dt[θ 1, t] Sin[θ 1] - lc2 Dt[θ 2, t] Sin[θ 2]}
```

```
In[24]:=  vm3 = D[pm3, {q}].dq

                             1          θ 1+ θ 2                      1          θ 1+ θ 2
Out[24]=  {(-l Cos[θ 1] -  ─ lc3 Cos[ ──────── ]) Dt[θ 1, t] - ─ lc3 Cos[ ──────── ] Dt[θ 2, t],
                             2             2                          2             2

            1                       θ 1+ θ 2                                 1          θ 1+ θ 2
          - ─ lc3 Dt[θ 2, t] Sin[ ──────── ] + Dt[θ 1, t] (-l Sin[θ 1] - ─ lc3 Sin[ ──────── ])}
            2                           2                                    2             2
```

In[25]:=

$$\mathbf{T = Simplify\left[\frac{1}{2}\, m1 * vm1.vm1 + \frac{1}{2}\, m2 * vm2.vm2 + \frac{1}{2}\, m3 * vm3.vm3 + \frac{1}{2}\, I1 * Dt[\theta 1, t]^2 + \frac{1}{2}\, I2 * Dt[\theta 2, t]^2 + \frac{1}{2}\, I3 * Dt[\theta 3, t]^2\right]}\ \texttt{(* Total kinetic energy for the system *)}$$

`SetAttributes[{I1, I2, I3}, Constant]`

Out[25]=

$$\begin{aligned}\frac{1}{8}\Bigg(&\left(4\, I1 + I3 + 4\, l^2\, m1 - 8\, l\, lc1\, m1 + 4\, lc1^2\, m1 + 4\, l^2\, m2 + 4\, l^2\, m3 + lc3^2\, m3 + 4\, l\, lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right]\right)\\
&Dt[\theta 1, t]^2 + 2\left(I3 + lc3^2\, m3 + 2\, l\, lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right] + 4\, l\, lc2\, m2\, \mathrm{Cos}[\theta 1-\theta 2]\right)\\
&Dt[\theta 1, t]\, Dt[\theta 2, t] + \left(4\, I2 + I3 + 4\, lc2^2\, m2 + lc3^2\, m3\right) Dt[\theta 2, t]^2\Bigg)\end{aligned}$$

In[27]:= **`L = Simplify[T - U]`**

Out[27]=

$$\begin{aligned}\frac{1}{8}\Bigg(&-8\, g\left(\left(-lc1\, m1 + l\,(m1 + m2 + m3)\right)\mathrm{Cos}[\theta 1] + lc2\, m2\, \mathrm{Cos}[\theta 2] + lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1+\theta 2}{2}\right]\right) +\\
&\left(4\, I1 + I3 + 4\, l^2\, m1 - 8\, l\, lc1\, m1 + 4\, lc1^2\, m1 + 4\, l^2\, m2 + 4\, l^2\, m3 + lc3^2\, m3 + 4\, l\, lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right]\right)\\
&Dt[\theta 1, t]^2 + 2\left(I3 + lc3^2\, m3 + 2\, l\, lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right] + 4\, l\, lc2\, m2\, \mathrm{Cos}[\theta 1-\theta 2]\right)\\
&Dt[\theta 1, t]\, Dt[\theta 2, t] + \left(4\, I2 + I3 + 4\, lc2^2\, m2 + lc3^2\, m3\right) Dt[\theta 2, t]^2\Bigg)\end{aligned}$$

`(* F1 and F2 are generalized non-conservative forces *)`

In[45]:=

```
eqnMotion = Simplify[Dt[D[L, {dq}], t] - D[L, {q}] - {τ, -τ} /. D[Dt[q[[1]], t], q[[1]]] → 0 /.
   D[Dt[q[[2]], t], q[[2]]] → 0]
```

Out[45]=

$$\begin{aligned}\Bigg\{\frac{1}{4}\Bigg(&-4\,\tau + \left(4\, I1 + I3 + 4\, l^2\, m1 - 8\, l\, lc1\, m1 + 4\, lc1^2\, m1 + 4\, l^2\, m2 + 4\, l^2\, m3 + lc3^2\, m3 + 4\, l\, lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right]\right)\\
&Dt[\theta 1, \{t, 2\}] + \left(I3 + lc3^2\, m3 + 2\, l\, lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right] + 4\, l\, lc2\, m2\, \mathrm{Cos}[\theta 1-\theta 2]\right) Dt[\theta 2, \{t, 2\}] -\\
&4\, g\, l\, m1\, \mathrm{Sin}[\theta 1] + 4\, g\, lc1\, m1\, \mathrm{Sin}[\theta 1] - 4\, g\, l\, m2\, \mathrm{Sin}[\theta 1] - 4\, g\, l\, m3\, \mathrm{Sin}[\theta 1] -\\
&l\, lc3\, m3\, Dt[\theta 1, t]^2\, \mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] + 2\, l\, lc3\, m3\, Dt[\theta 1, t]\, Dt[\theta 2, t]\, \mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] +\\
&l\, lc3\, m3\, Dt[\theta 2, t]^2\, \mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] + 4\, l\, lc2\, m2\, Dt[\theta 2, t]^2\, \mathrm{Sin}[\theta 1-\theta 2] - 2\, g\, lc3\, m3\, \mathrm{Sin}\left[\frac{\theta 1+\theta 2}{2}\right]\Bigg),\\
\frac{1}{4}\Bigg(&4\,\tau + \left(I3 + lc3^2\, m3 + 2\, l\, lc3\, m3\, \mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right] + 4\, l\, lc2\, m2\, \mathrm{Cos}[\theta 1-\theta 2]\right) Dt[\theta 1, \{t, 2\}] +\\
&\left(4\, I2 + I3 + 4\, lc2^2\, m2 + lc3^2\, m3\right) Dt[\theta 2, \{t, 2\}] - 2\, l\, lc3\, m3\, Dt[\theta 1, t]^2\, \mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] -\\
&4\, l\, lc2\, m2\, Dt[\theta 1, t]^2\, \mathrm{Sin}[\theta 1-\theta 2] - 4\, g\, lc2\, m2\, \mathrm{Sin}[\theta 2] - 2\, g\, lc3\, m3\, \mathrm{Sin}\left[\frac{\theta 1+\theta 2}{2}\right]\Bigg)\Bigg\}\end{aligned}$$

In[46]:=
```
temp = Normal[CoefficientArrays[eqnMotion, Dt[dq, t]]];
(* temp has now two elements: coefficients of Dt[dq,t] and remainder *)
remain1 = temp[[1]];
Mfixed = temp[[2]];
MatrixForm[Mfixed]
FullSimplify[eqnMotion - remain1 - Mfixed.Dt[dq, t]] (* Should be all 0's *)
```

Out[48]//MatrixForm=

$$\left(\begin{array}{ll} \frac{1}{4}\left(4\,\mathrm{I1} + \mathrm{I3} + 4\,l^2\,\mathrm{m1} - 8\,l\,\mathrm{lc1}\,\mathrm{m1} + 4\,\mathrm{lc1}^2\,\mathrm{m1} + 4\,l^2\,\mathrm{m2} + 4\,l^2\,\mathrm{m3} + \mathrm{lc3}^2\,\mathrm{m3} + 4\,l\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right]\right) & \frac{1}{4}\left(\mathrm{I3} + \mathrm{lc3} \right. \\ \frac{1}{4}\left(\mathrm{I3} + \mathrm{lc3}^2\,\mathrm{m3} + 2\,l\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Cos}\left[\frac{\theta 1-\theta 2}{2}\right] + 4\,l\,\mathrm{lc2}\,\mathrm{m2}\,\mathrm{Cos}[\theta 1-\theta 2]\right) & \end{array}\right.$$

Out[49]= {0, 0}

In[50]:=
```
temp = Normal[CoefficientArrays[remain1, dq, "Symmetric" → True]];
remain2 = temp[[1]];
Ctensor = temp[[3]];
Cfixed = Ctensor.dq;
MatrixForm[Cfixed]
Simplify[remain1 - remain2 - Cfixed.dq](*Should be 0's*)
```

Out[54]//MatrixForm=

$$\left(\begin{array}{ll} -\frac{1}{4}\,l\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Dt}[\theta 1, t]\,\mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] + \frac{1}{4}\,l\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Dt}[\theta 2, t]\,\mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] & \frac{1}{4}\,l\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Dt}[\theta 1, t]\,\mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] + \mathrm{Dt}[\theta \\ \mathrm{Dt}[\theta 1, t]\left(-\frac{1}{2}\,l\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Sin}\left[\frac{\theta 1-\theta 2}{2}\right] - l\,\mathrm{lc2}\,\mathrm{m2}\,\mathrm{Sin}[\theta 1-\theta 2]\right) & \end{array}\right.$$

Out[55]= {0, 0}

In[56]:=
```
Bfixed = -{Coefficient[remain2, τ]}ᵀ;
MatrixForm[Bfixed]
```

Out[57]//MatrixForm=

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

In[58]:=
```
Gfixed = remain2 + Bfixed.{τ};
MatrixForm[Gfixed]
```

Out[59]//MatrixForm=

$$\begin{pmatrix} -g\,l\,\mathrm{m1}\,\mathrm{Sin}[\theta 1] + g\,\mathrm{lc1}\,\mathrm{m1}\,\mathrm{Sin}[\theta 1] - g\,l\,\mathrm{m2}\,\mathrm{Sin}[\theta 1] - g\,l\,\mathrm{m3}\,\mathrm{Sin}[\theta 1] - \frac{1}{2}\,g\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Sin}\left[\frac{\theta 1+\theta 2}{2}\right] \\ -g\,\mathrm{lc2}\,\mathrm{m2}\,\mathrm{Sin}[\theta 2] - \frac{1}{2}\,g\,\mathrm{lc3}\,\mathrm{m3}\,\mathrm{Sin}\left[\frac{\theta 1+\theta 2}{2}\right] \end{pmatrix}$$