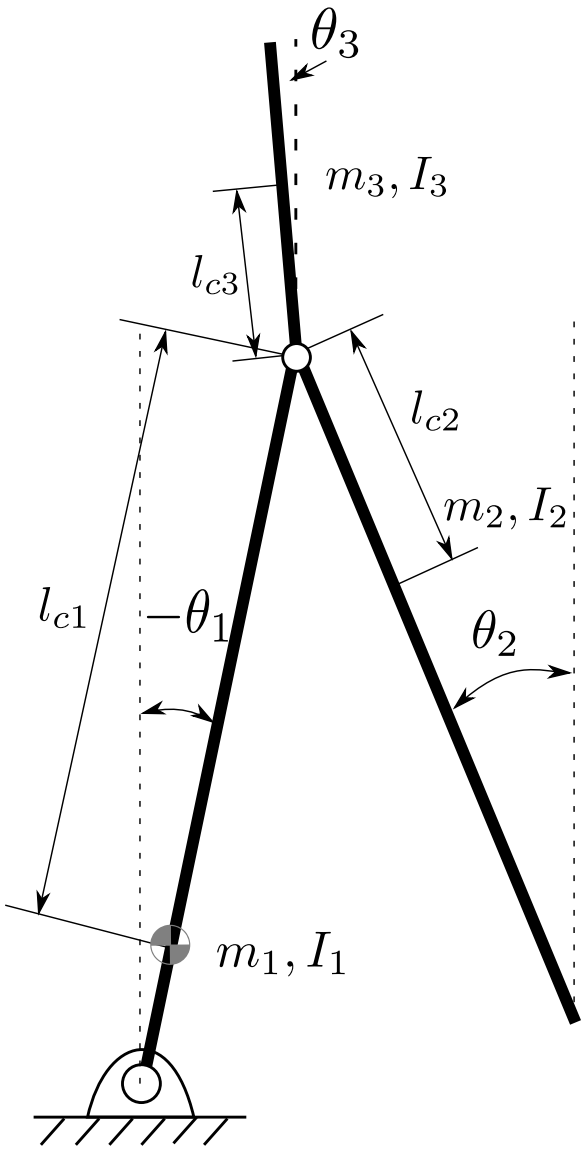

Figure 4-1: The three-link system model describing the compass gait plant.

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

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

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

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.

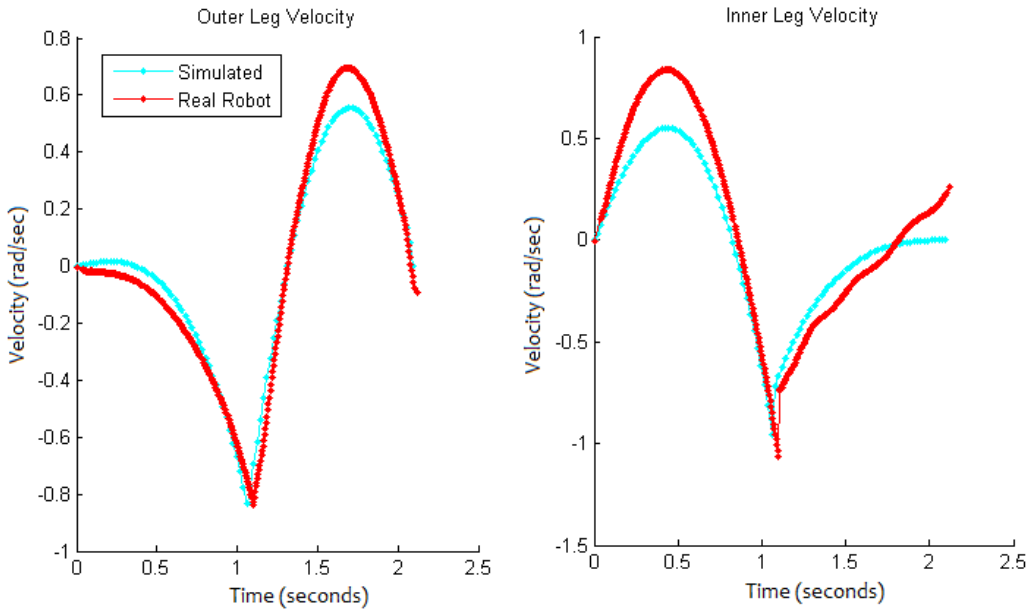

Figure 4-2: Plot of leg velocities resulting from an open loop command played on the robot versus the simulator.

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 sys-

tem 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

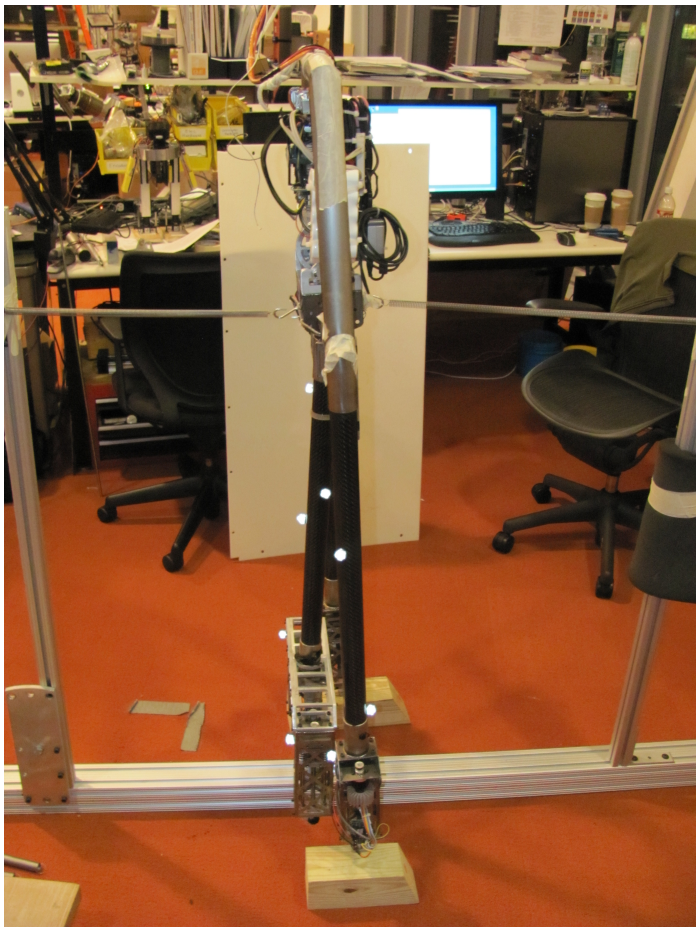

Figure 4-3: The robot setup during system identification. Note the springs stabilizing the system at the equilibrium point.

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

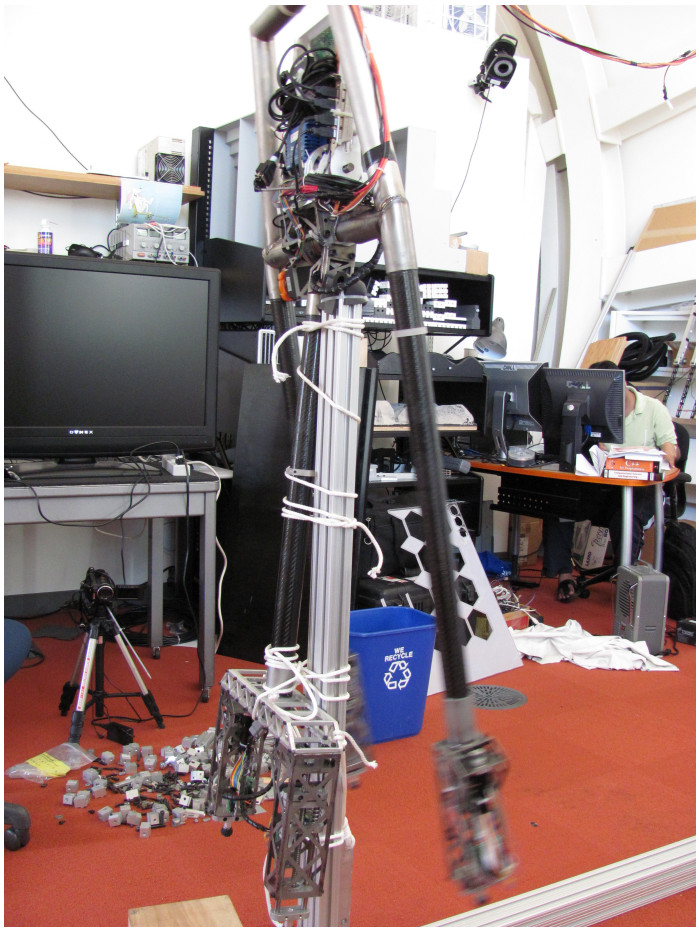

Figure 4-4: The robot with one of the legs fixed, producing a stable configuration and focusing on a subset of the system's dynamics.

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

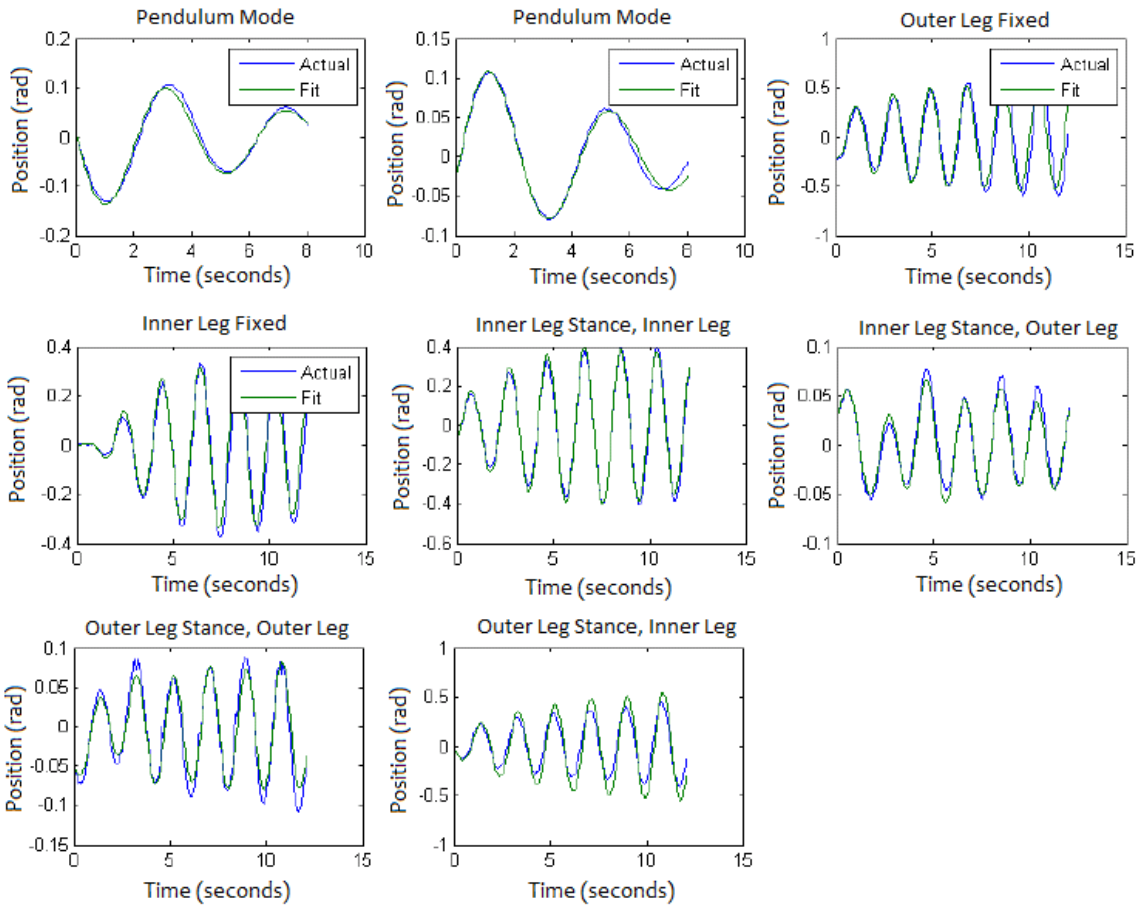

Figure 4-5: The final system identification fits, compared against the training data.

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.

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

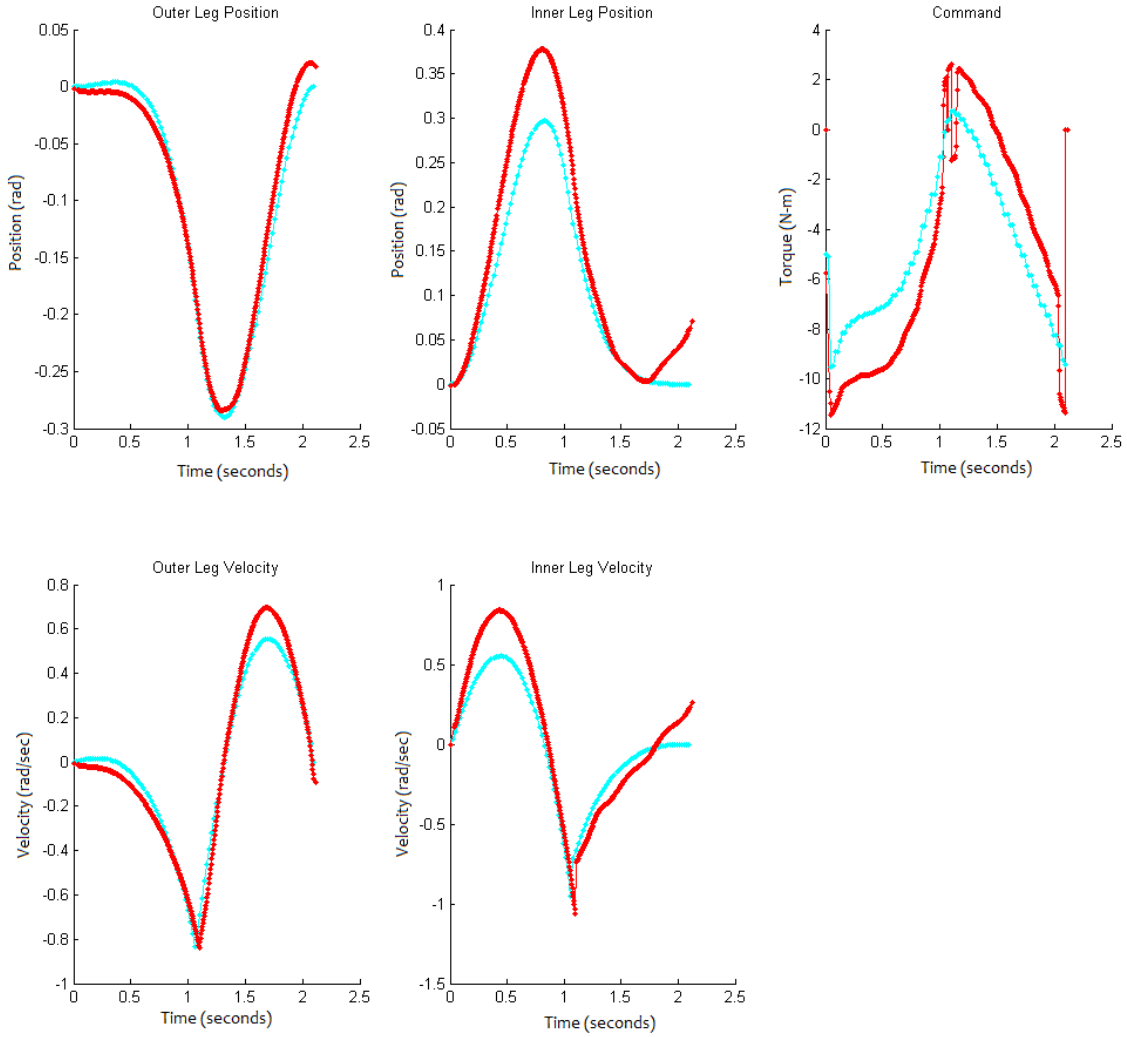

Figure 4-6: Time plots of the real robot executing a trajectory planned with the identified model for model validation. The actual command differs from the nominal because friction was not included in the model used by the planner. The friction compensator used was also identified using the same techniques.

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

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

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

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.

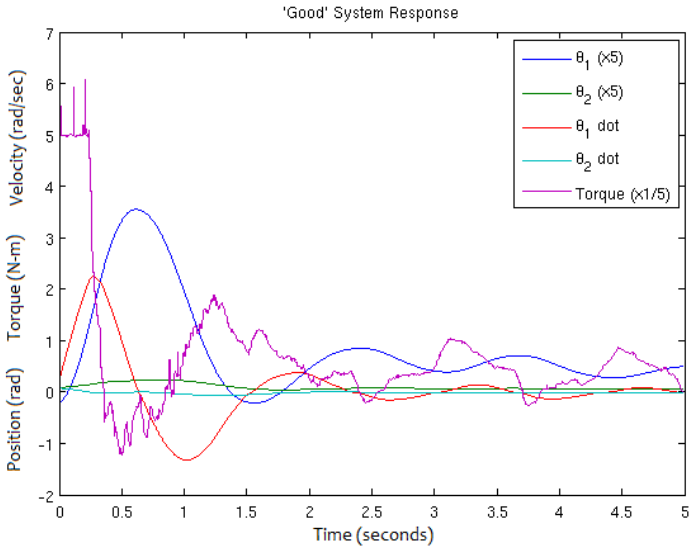

Figure 4-7: A typical time response of the balancing controller when switched on with a nonzero initial condition.

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

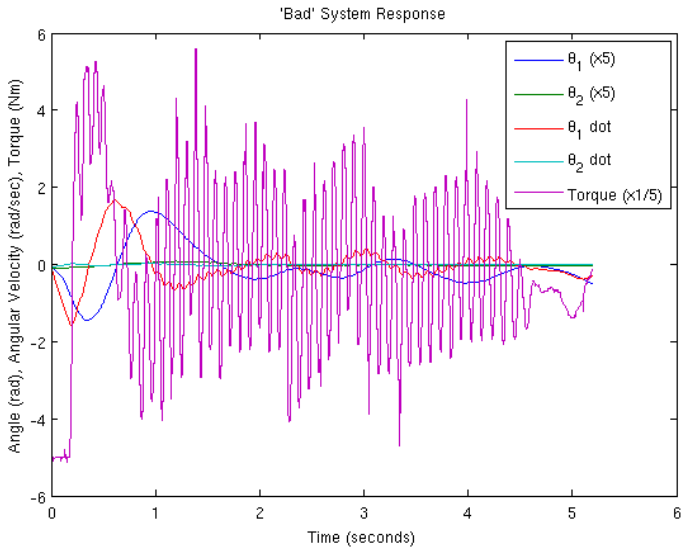

Figure 4-8: Time response of the balancing controller without steps taken to attenuate high frequency dynamics.

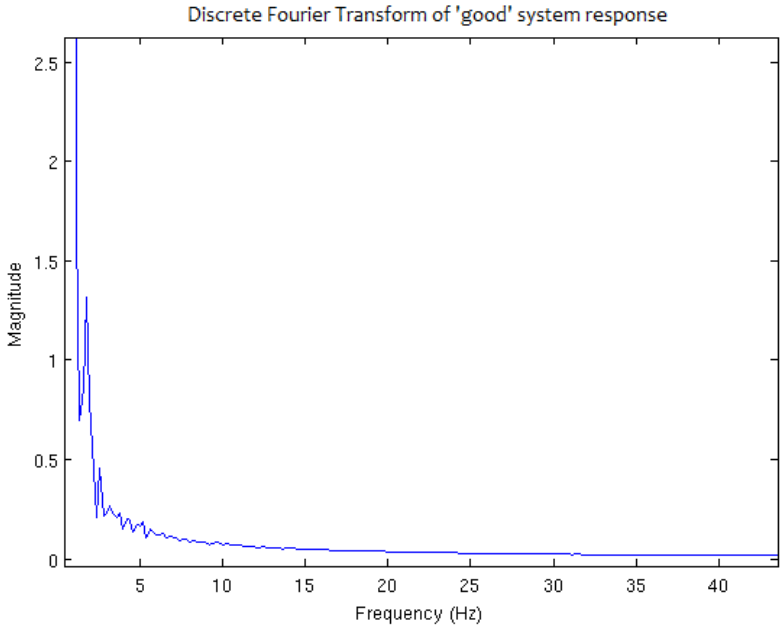

Figure 4-9: Discrete Fourier transform of the torque signal from the good balancing response.

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

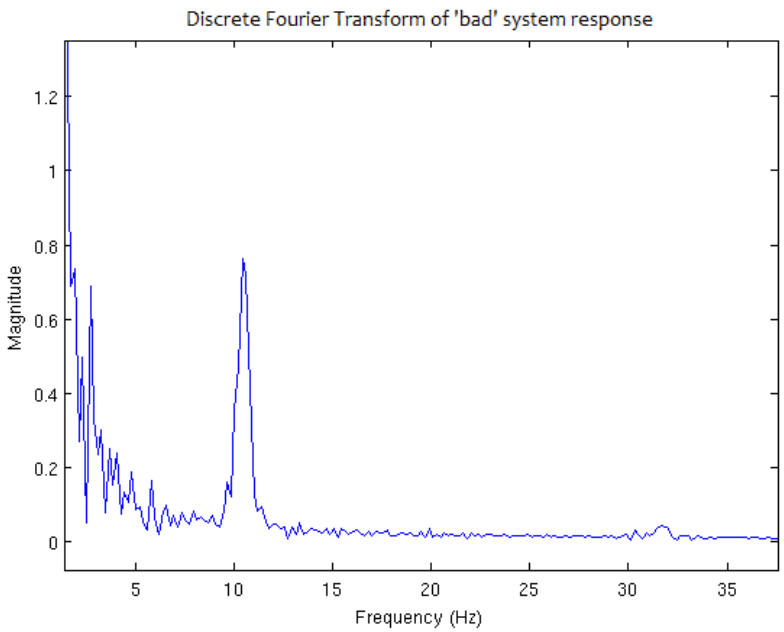

Figure 4-10: Discrete Fourier transform of the torque signal from the balancing response with unwanted high frequency dynamics.

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.

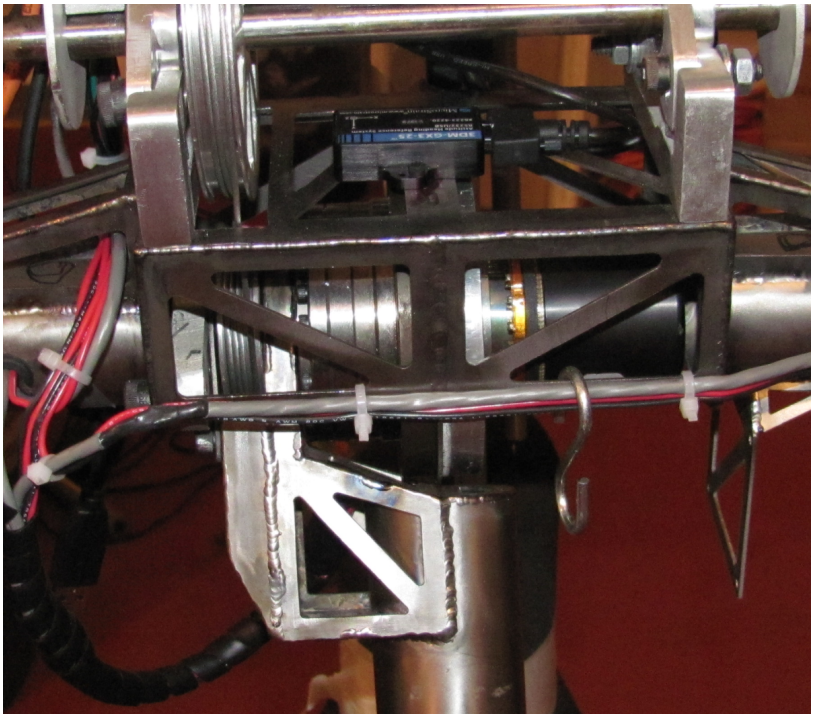

Figure 4-11: The bracket the IMU is attached to, possibly flexing as the hip motor applies large torques to the structure.

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.

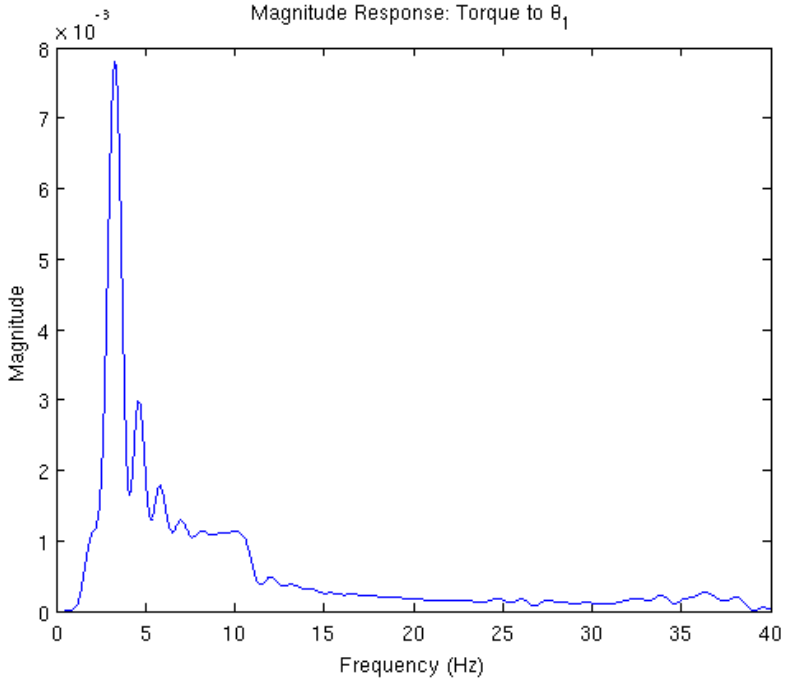

Figure 4-12: Frequency response of the torque to IMU flex system.

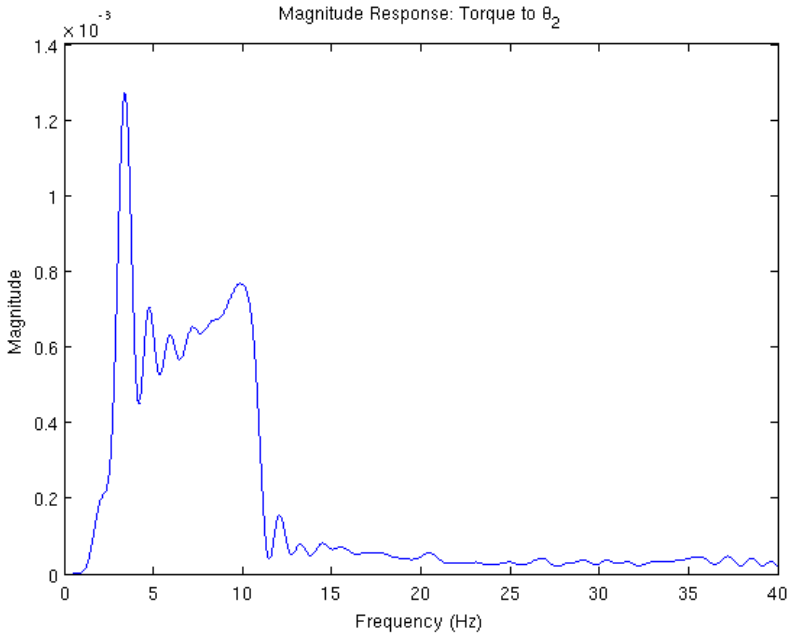

Figure 4-13: Frequency response of the torque to inter-leg angle encoder system.

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

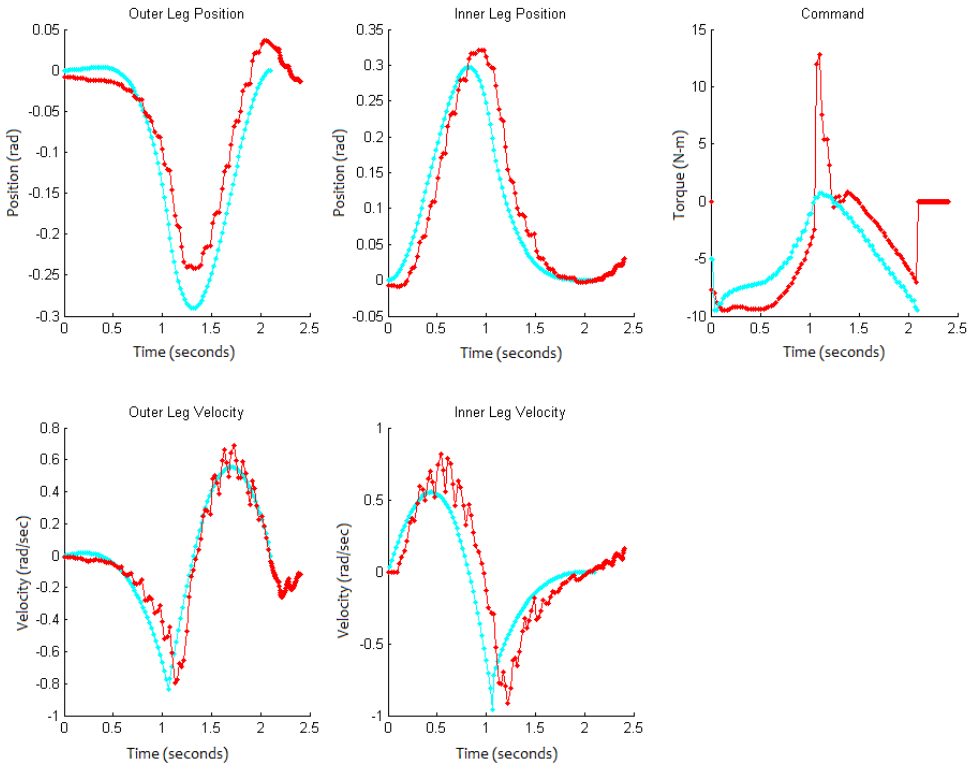

Figure 4-14: TVLQR stabilized 'one step rebalance' trajectory. The small ripple along the trajectory is due to the very high controller gains exciting unmodeled high frequency dynamics in the system. Nominal trajectory is in cyan while the recorded data is in red.

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

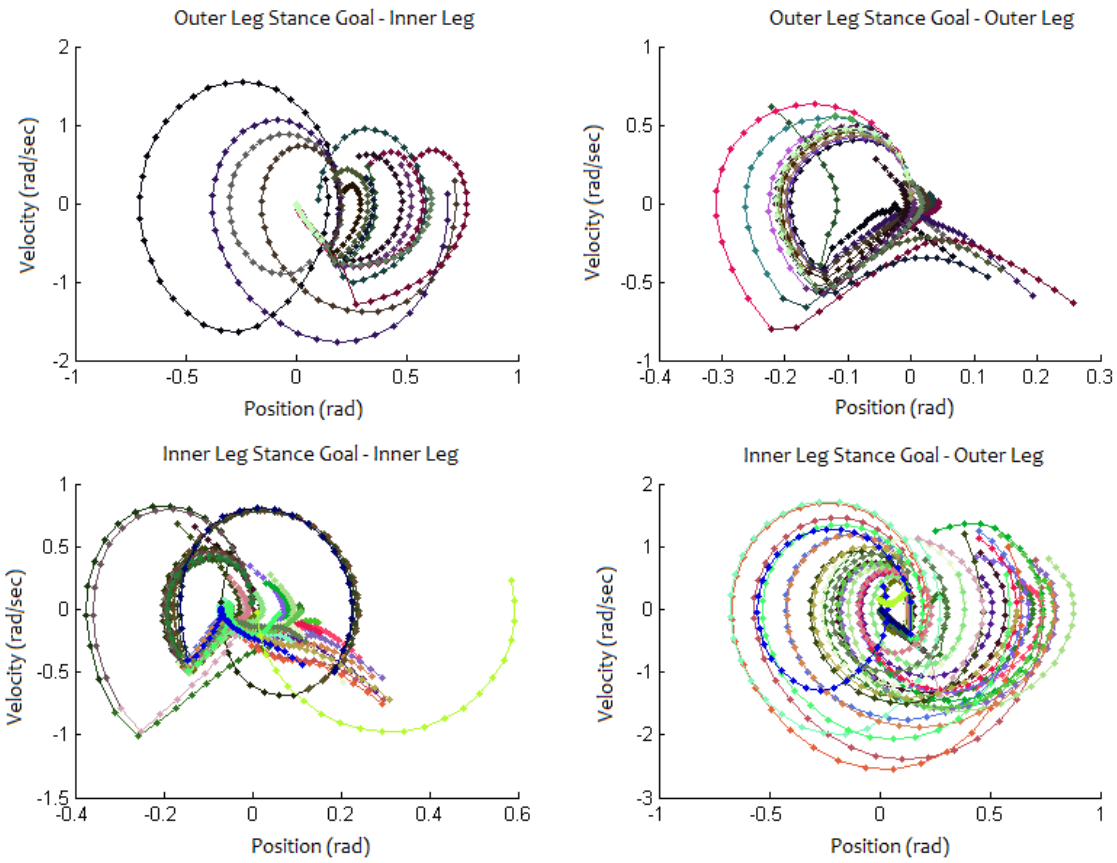

Figure 4-15: Plots of the hybrid LQR Tree controller designed to bring the robot to equilibrium on either of the stance legs.

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

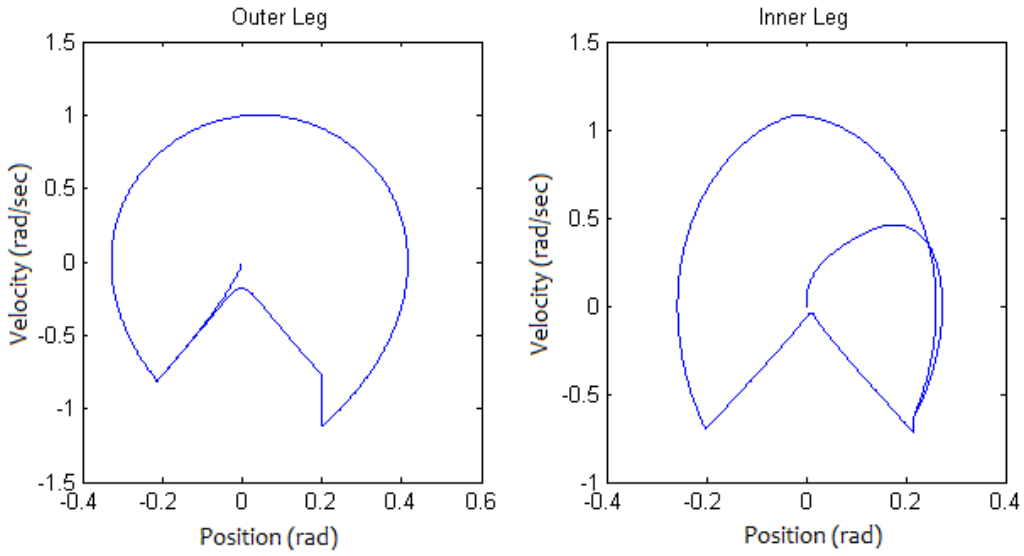

Figure 4-16: The planed periodic trajectory with the step-in trajectory from equilibrium.

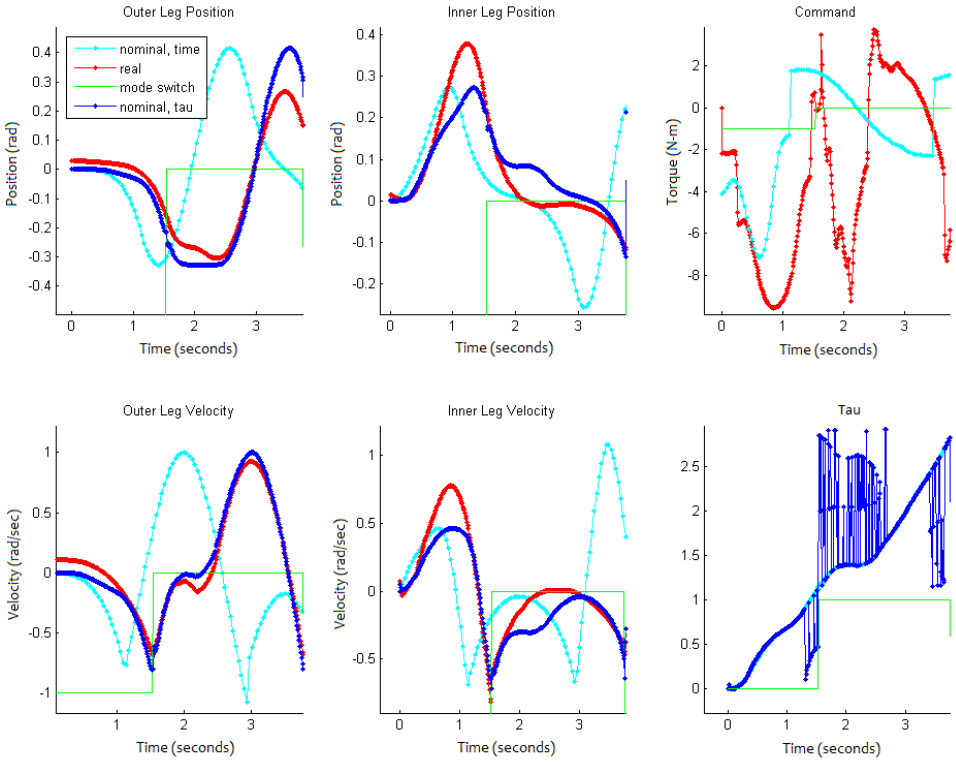

Figure 4-17: The TVLQR stabilized step-in and first step of the periodic trajectory as run on the real robot.

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]=
$$\{-l \sin[\theta_1],\ l \cos[\theta_1]\}$$

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

Out[11]=
$$\{-l \sin[\theta_1] + lc1 \sin[\theta_1],\ l \cos[\theta_1] - lc1 \cos[\theta_1]\}$$

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

Out[13]=
$$\{-l \sin[\theta_1] - lc2 \sin[\theta_2],\ l \cos[\theta_1] + lc2 \cos[\theta_2]\}$$

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

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

Out[16]=
$$\left\{-l \sin[\theta_1] - lc3 \sin\left[\frac{\theta_1+\theta_2}{2}\right],\ l \cos[\theta_1] + lc3 \cos\left[\frac{\theta_1+\theta_2}{2}\right]\right\}$$

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]
```

Out[18]=
$$g\left((-lc1\ m1 + l\ (m1 + m2 + m3))\cos[\theta_1] + lc2\ m2 \cos[\theta_2] + lc3\ m3 \cos\left[\frac{\theta_1+\theta_2}{2}\right]\right)$$

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

Out[21]=
$$\{Dt[\theta_1, t],\ Dt[\theta_2, t]\}$$

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

Out[22]=
$$\{(-l \cos[\theta_1] + lc1 \cos[\theta_1])\ Dt[\theta_1, t],\ Dt[\theta_1, t]\ (-l \sin[\theta_1] + lc1 \sin[\theta_1])\}$$

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

Out[23]=
$$\{-l \cos[\theta_1]\ Dt[\theta_1, t] - lc2 \cos[\theta_2]\ Dt[\theta_2, t],\ -l\ Dt[\theta_1, t] \sin[\theta_1] - lc2\ Dt[\theta_2, t] \sin[\theta_2]\}$$

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

Out[24]=
$$\left\{\left(-l \cos[\theta_1] - \frac{1}{2} lc3 \cos\left[\frac{\theta_1+\theta_2}{2}\right]\right) Dt[\theta_1, t] - \frac{1}{2} lc3 \cos\left[\frac{\theta_1+\theta_2}{2}\right] Dt[\theta_2, t],\right.$$
$$\left.-\frac{1}{2} lc3\ Dt[\theta_2, t] \sin\left[\frac{\theta_1+\theta_2}{2}\right] + Dt[\theta_1, t]\left(-l \sin[\theta_1] - \frac{1}{2} lc3 \sin\left[\frac{\theta_1+\theta_2}{2}\right]\right)\right\}$$

In[25]:=
```
T = Simplify[1/2 m1 * vm1.vm1 + 1/2 m2 * vm2.vm2 + 1/2 m3 * vm3.vm3 + 1/2 I1 * Dt[θ 1, t]^2 +
    1/2 I2 * Dt[θ 2, t]^2 + 1/2 I3 * Dt[θ 3, t]^2] (* Total kinetic energy for the system *)
SetAttributes[{ I1, I2, I3}, Constant]
```

Out[25]=
$$\frac{1}{8}\left(\left(4\,I1 + I3 + 4\,l^2\,m1 - 8\,l\,lc1\,m1 + 4\,lc1^2\,m1 + 4\,l^2\,m2 + 4\,l^2\,m3 + lc3^2\,m3 + 4\,l\,lc3\,m3\,\text{Cos}\left[\frac{\theta 1 - \theta 2}{2}\right]\right)\right.$$
$$\text{Dt}[\theta 1, t]^2 + 2\left(I3 + lc3^2\,m3 + 2\,l\,lc3\,m3\,\text{Cos}\left[\frac{\theta 1 - \theta 2}{2}\right] + 4\,l\,lc2\,m2\,\text{Cos}[\theta 1 - \theta 2]\right)$$
$$\left.\text{Dt}[\theta 1, t]\,\text{Dt}[\theta 2, t] + \left(4\,I2 + I3 + 4\,lc2^2\,m2 + lc3^2\,m3\right)\text{Dt}[\theta 2, t]^2\right)$$

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

Out[27]=
$$\frac{1}{8}\left(-8\,g\left(\left(-lc1\,m1 + l\,(m1 + m2 + m3)\right)\,\text{Cos}[\theta 1] + lc2\,m2\,\text{Cos}[\theta 2] + lc3\,m3\,\text{Cos}\left[\frac{\theta 1 + \theta 2}{2}\right]\right) + \right.$$
$$\left(4\,I1 + I3 + 4\,l^2\,m1 - 8\,l\,lc1\,m1 + 4\,lc1^2\,m1 + 4\,l^2\,m2 + 4\,l^2\,m3 + lc3^2\,m3 + 4\,l\,lc3\,m3\,\text{Cos}\left[\frac{\theta 1 - \theta 2}{2}\right]\right)$$
$$\text{Dt}[\theta 1, t]^2 + 2\left(I3 + lc3^2\,m3 + 2\,l\,lc3\,m3\,\text{Cos}\left[\frac{\theta 1 - \theta 2}{2}\right] + 4\,l\,lc2\,m2\,\text{Cos}[\theta 1 - \theta 2]\right)$$
$$\left.\text{Dt}[\theta 1, t]\,\text{Dt}[\theta 2, t] + \left(4\,I2 + I3 + 4\,lc2^2\,m2 + lc3^2\,m3\right)\text{Dt}[\theta 2, t]^2\right)$$

```
(* 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]=
$$\left\{\frac{1}{4}\left(-4\,\tau + \left(4\,I1 + I3 + 4\,l^2\,m1 - 8\,l\,lc1\,m1 + 4\,lc1^2\,m1 + 4\,l^2\,m2 + 4\,l^2\,m3 + lc3^2\,m3 + 4\,l\,lc3\,m3\,\text{Cos}\left[\frac{\theta 1 - \theta 2}{2}\right]\right)\right.\right.$$
$$\text{Dt}[\theta 1, \{t, 2\}] + \left(I3 + lc3^2\,m3 + 2\,l\,lc3\,m3\,\text{Cos}\left[\frac{\theta 1 - \theta 2}{2}\right] + 4\,l\,lc2\,m2\,\text{Cos}[\theta 1 - \theta 2]\right)\text{Dt}[\theta 2, \{t, 2\}] -$$
$$4\,g\,l\,m1\,\text{Sin}[\theta 1] + 4\,g\,lc1\,m1\,\text{Sin}[\theta 1] - 4\,g\,l\,m2\,\text{Sin}[\theta 1] - 4\,g\,l\,m3\,\text{Sin}[\theta 1] -$$
$$l\,lc3\,m3\,\text{Dt}[\theta 1, t]^2\,\text{Sin}\left[\frac{\theta 1 - \theta 2}{2}\right] + 2\,l\,lc3\,m3\,\text{Dt}[\theta 1, t]\,\text{Dt}[\theta 2, t]\,\text{Sin}\left[\frac{\theta 1 - \theta 2}{2}\right] +$$
$$\left.l\,lc3\,m3\,\text{Dt}[\theta 2, t]^2\,\text{Sin}\left[\frac{\theta 1 - \theta 2}{2}\right] + 4\,l\,lc2\,m2\,\text{Dt}[\theta 2, t]^2\,\text{Sin}[\theta 1 - \theta 2] - 2\,g\,lc3\,m3\,\text{Sin}\left[\frac{\theta 1 + \theta 2}{2}\right]\right),$$
$$\frac{1}{4}\left(4\,\tau + \left(I3 + lc3^2\,m3 + 2\,l\,lc3\,m3\,\text{Cos}\left[\frac{\theta 1 - \theta 2}{2}\right] + 4\,l\,lc2\,m2\,\text{Cos}[\theta 1 - \theta 2]\right)\text{Dt}[\theta 1, \{t, 2\}] +\right.$$
$$\left(4\,I2 + I3 + 4\,lc2^2\,m2 + lc3^2\,m3\right)\text{Dt}[\theta 2, \{t, 2\}] - 2\,l\,lc3\,m3\,\text{Dt}[\theta 1, t]^2\,\text{Sin}\left[\frac{\theta 1 - \theta 2}{2}\right] -$$
$$\left.\left.4\,l\,lc2\,m2\,\text{Dt}[\theta 1, t]^2\,\text{Sin}[\theta 1 - \theta 2] - 4\,g\,lc2\,m2\,\text{Sin}[\theta 2] - 2\,g\,lc3\,m3\,\text{Sin}\left[\frac{\theta 1 + \theta 2}{2}\right]\right)\right\}$$

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=

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

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=

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

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=

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