Adaptive Swing-up and Balancing Control of Acrobot Systems

by

Luke B. Johnson

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science at the Massachusetts Institute of Technology

June 2009

Signature of Author.............. [illegible]

Department of Mechanical Engineering
May 11, 2009

Certified by .... [illegible]

John J. Leonard
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by .... [illegible]

Professor J. Lienhard V
Collins Professor of Mechanical Engineering
Chairman, Undergraduate Thesis Committee

Adaptive Swing-up and Balancing Control of Acrobot Systems

by

Luke B. Johnson

Submitted to the Department of Mechanical Engineering
on May 11, 2009 in Partial Fulfillment of the
Requirements for the Degree of Bachelors of Science in
Mechanical Engineering

Abstract

The field of underactuated robotics has become the core of agile mobile robotics research. Significant past effort has been put into understanding the swing-up control of the acrobot system. This thesis implements an online, adaptive swing-up and balancing controller with no previous knowledge of the system's mass or geometric parameters. A least squares method is used to identify the 5 parameters necessary to completely characterize acrobot dynamics. Swing up is accomplished using partial feedback linearization and a pump up strategy to add energy to the system. The controller then catches the swung up system in the basin of attraction of an LQR controller computed using the estimated parameter values generated from online system identification. These results are then simulated using a MATLAB simulation environment.

Thesis Supervisor: John J. Leonard
Title: Professor of Mechanical Engineering

Introduction

Adaptive algorithms have been extensively successful in industrial applications where arm mass parameters undoubtedly change when interacting with their environments. This interaction forces control algorithms to adapt to these changes and continuously update their estimated model of the system. The basic idea behind all types of adaptive control is to create a skeleton model with some unknown parameters a_m that combine with some measureable quantities f_m and y to form constraint equations. A linear version of this constraint is shown below as Equation 1

$$y = a_1 f_1(q) + a_2 f_2(q) + \ldots + a_m f_m(q). \tag{1}$$

Any number of update laws can be used to estimate the unknown parameters a_m, and given sufficiently diverse set of f_m's we can obtain an arbitrarily precise estimate of the system's true dynamics.

There are a couple different "flavors" of adaptive control algorithms. This paper specifically uses the model known as a self tuning controller. A block diagram of this model is shown in Figure 1 below.

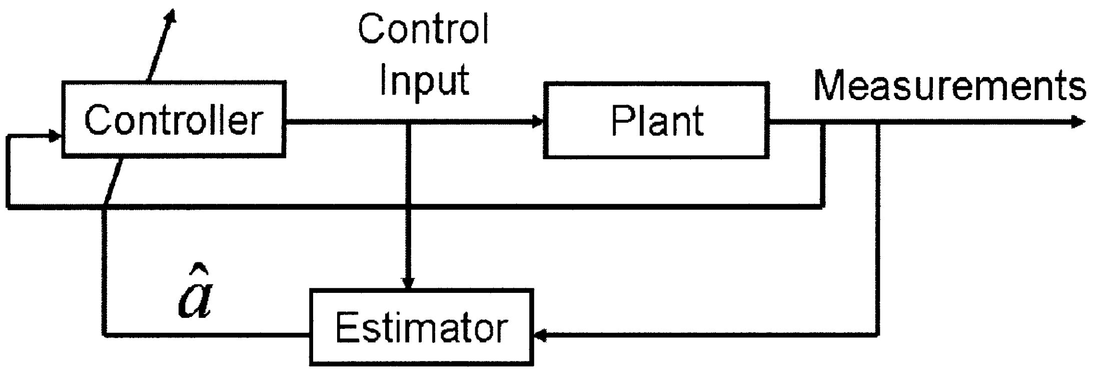

Figure 1: Block diagram of a self tuning controller, where â is defined as a vector of the current best estimate of the system parameters

In the figure above the "controller" is the law that defines the control input, the "plant" is the unchangeable system dynamics, the output of the plant is what is measurable, the control input is the output of the controller, and both the measurements and the control input are passed into an "estimator" that produces a vector of the current best guess of the parameters, $\hat{a}$.

The industrial implementation of adaptive control is slightly different because control engineers are often interested in following specified trajectories for arms with unknown parameters. This gives rise to a particularly elegant form of adaptive control call Model-Reference Adaptive Control [1]. This implementation uses linear feedback around the desired trajectory and generally leads to very fast convergence and excellent performance. Unfortunately this method is not quite adequate to handle underactuated robotics as it is defined. By the definition of underactuated systems, certain desired accelerations are simply unattainable (in the acrobot case this is because there is no actuator on one of the joints.) The controller in underactuated cases is more usually concerned with reaching goals. The case study presented in this paper is specifically attempting to swing up the acrobot and balance in the unstable vertical position. The problem with doing simple trajectory planning is that we may not know ahead of time what type of trajectories are possible, given the actuator limits. Thus we are left using the slightly less profound self tuning model of adaptive control.

Why the acrobot?

Up until this point I have been generic about talking about "underactuated systems." This thesis focuses on the case study of an acrobot system. The reasons behind choosing this system are not completely arbitrary. 1) The acrobot has been extensively studied and therefore there is extensive previous literature on the problem. 2) Part of the convenient nature of the acrobot system is that it has a constrained state space on position. An unstable acrobot system when system id is poor will not drive itself around the room while learning to swing up like a cart-pole system might have to. 3) The acrobot system is easy to reset after a failed attempt, a controller of virtual damping between links 1 and 2 will bring the system back to its stable equilibrium.

As was mentioned above, the swing up control of the acrobot has been extensively explored. In [2], Spong outlines a swing up controller using already known mass parameters. The method presented in Spong's paper for swing up is what this paper uses as a base for its swing up controller. As will be explained later in the paper, this choice was most likely not ideal for the purposes of this problem. The method as Spong proposed it assumed a strict system identification before the controller will run. In [3] calibration is addressed using an Unscented Kalman filter to obtain system parameters, but it too is run offline ahead of time, in order to guarantee strict error tolerances on the system parameters before the swing up attempt was run. One note about this paper is they use a relatively new Lyaponov based scheduling procedure for the balancing controller that may have solved some of the swing up to LQR switching instability that was incurred under this paper's implementation.

Experimental Setup

The entire system was simulated in a MATLAB environment. Figure 2 below outlines the geometry of the acrobot arm as well as the relevant system parameters.

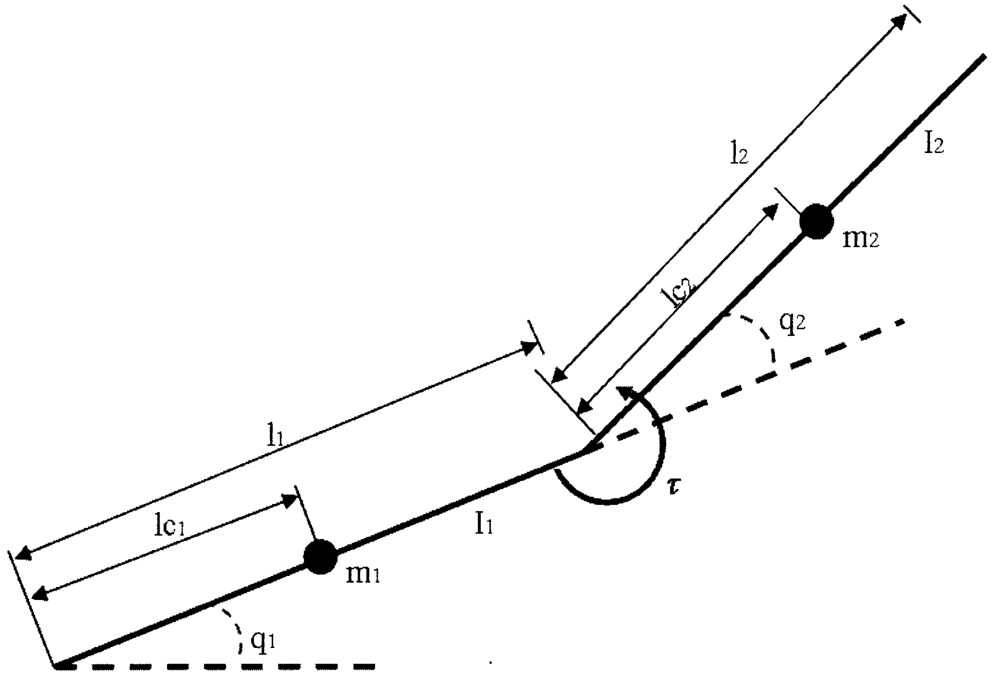

Figure 2: geometry of the acrobot arm as well as the relevant system parameters

As seen in the figure above, q_i is the angle of the ith link, m_i is the mass of the ith link, l_i is the length of the ith link, l_{ci} is the length from the base joint of link i to the center of mass of length i, I_i is the moment of inertia of the ith link about the center of mass, and

τ is the input torque applied. The equations of motion of the system are listed below as Equations 2 and 3 and are also verified in [2]:

$$d_{11}\ddot{q}_1 + d_{12}\ddot{q}_2 + h_1 + \phi_1 = 0, \tag{2}$$

$$d_{12}\ddot{q}_1 + d_{12}\ddot{q}_2 + h_2 + \phi_2 = \tau. \tag{3}$$

In these equations $\ddot{q}_i$ is the second derivative of the ith angular measurement. The other constants used in Equations 2 and 3 are defined in Equations 4-10

$$d_{11} = m_1 l_{c1}^2 + m_2(l_1^2 + l_{c2}^2 + 2l_1 l_{c2}\cos(q_2)) + I_1 + I_2 \tag{4}$$

$$d_{22} = m_2 l_{c2}^2 + I_2 \tag{5}$$

$$d_{12} = m_2(l_{c2}^2 + l_1 l_{c2}\cos(q_2)) + I_2 \tag{6}$$

$$h_1 = -m_2 l_1 l_{c2}\sin(q_2)\dot{q}_2^2 - 2m_2 l_1 l_{c2}\sin(q_2)\dot{q}_2\dot{q}_1 \tag{7}$$

$$h_2 = m_2 l_1 l_{c2}\sin(q_2)\dot{q}_1^2 \tag{8}$$

$$\phi_1 = (m_1 l_{c1} + m_2 l_1)g\cos(q_1) + m_2 l_{c2} g\cos(q_1 + q_2) \tag{9}$$

$$\phi_2 = m_2 l_{c2} g\cos(q_1 + q_2) \tag{10}$$

where g is the acceleration due to gravity, and $\dot{q}_i$ is the first derivative of the angular position of the ith link.

The system is modeled as a continuous time system integrated with MATLAB's ode45 implementation of Runga Kutta 4. The input torque is modeled as a continuous input that is held constant during each sampling interval to help simulate how a physical system would be controlled. The values for the simulation constants are shown below in Table 1.

Table 1: values of simulation constants with their associated units

m_1	1.0 (kg)
m_2	1.0 (kg)
l_1	1.0 (m)
l_2	1.0 (m)
l_{C1}	0.5 (m)
l_{C2}	0.5 (m)
I_1	0.2 (kg*m^2)
I_2	1.0 (kg*m^2)
g	9.8 (m/s)
dt	0.01 (s)
Max torque	5.0 (Nm)

In this simulation, $q_1, \dot{q}_1, q_2, \dot{q}_2$ are assumed to be measureable and are treated as a the complete set of state variables. If these quantities were not measurable we would have to create an estimator. If this were the case it might be easiest to use an Unscented Kalman Filter such as in [3], to perform both the state estimation and parameter estimation. For the purposes of this paper this was assumed to be too complicated and thus the fully observable assumption was maintained.

Swing up implementation

The design purpose of the swing up controller is to move the acrobot system into a state contained within the basin of attraction of the LQR controller for successful balancing. In order to decrease the complexity of the swing up controller we use partial feedback linearization (PFL) on link 2 in or order to decouple the influence of link 1 on link 2's dynamics. This derivation is shown in [2] but an abridged version is shown here for completeness. If we rearrange Equation 2 we obtain Equation 11 shown as

$$\ddot{q}_1 = -d_{11}^{-1}(d_{12}\ddot{q}_2 + h_1 + \phi_1), \tag{11}$$

Equation 11 can then be substituted into Equation 3 to produce Equation 12,

$$\bar{d}_{22}\ddot{q}_2 + \bar{h}_2 + \bar{\phi}_2 = \tau, \qquad (12)$$

where $\bar{d}_{22}$, $\bar{h}_2$, $\bar{\phi}_2$ are defined in Equations 12-15 as

$$\bar{d}_{22} = d_{22} - d_{21} - d_{11}^{-1}d_{12}, \qquad (13)$$

$$\bar{h}_2 = h_2 - d_{21}d_{11}^{-1}h_1, \qquad (14)$$

$$\bar{\phi}_2 = \phi_2 - d_{21}d_{11}^{-1}\phi_1. \qquad (15)$$

This is a convenient form for implementing PFL because with the assumption that we have sufficiently large torque allowances, we can define the input τ in Equation 12 as Equation 16,

$$\tau = \bar{d}_{22}v_2 + \bar{h}_2 + \bar{\phi}_2, \qquad (16)$$

where v_2 is the desired link 2 angular acceleration. This completely decouples the actions of link 1 with link 2. (Note the actions of link 2 still influence link 1 but not vice-versa) This result in illustrated in the Equations 17 and 18 below as

$$d_{11}\ddot{q}_1 + h_1 + \phi_1 = -d_{12}v_2, \qquad (17)$$

$$\ddot{q}_2 = v_2. \qquad (18)$$

It is notable that although the motions on link 1 do not affect link 2 with this control strategy (all the inertial coupling is canceled out through the PFL), the control input v_2 does and must affect the motion of link 1. It is this inertial coupling that we use to swing up the acrobot, all the while maintaining complete control authority of $\ddot{q}_2$. After this PFL we must decide what control input we do want to implement such that q_1 swings up, while keeping q_2 under control. [2] proposes the following desired trajectory of link 2, as shown in Equation 19

$$q_2^d = \frac{2\alpha}{\pi}\tan^{-1}(\dot{q}_1), \tag{19}$$

What this desired trajectory effectively does is smoothly keep q_2 between the angles of $\pm\alpha$. More importantly, the net torque required to follow this desired input forces energy into the system (this is only necessarily true if there is sufficient torque to hold the PFL constraints.) This way of pumping energy into the system turns out to be both convenient and troublesome as will be described later in the paper. The controller to obtain this desired trajectory is then a PD controller as shown in Equation 20,

$$v_2 = k_p(q_2^d - q_2) - k_d q_2, \tag{20}$$

where k_p is the proportional gain and k_d is the derivative gain. If the arm parameters where known a priori, a method such as stochastic gradient descent could be used to tune these gains for optimal performance. Since the controller needs to adapt to a wide range of arm parameters, they are kept constant. The parameters used in this simulation are shown in table 2 below

Table 2: swing up simulation parameters

k_p	400
k_d	10
α	0.75

Adaptation techniques

As has been mentioned above there are many ways to perform online system identification. The method that was used in this implementation is least squares estimation. The final implementation combines a batch least squares method when the input is non-rich, with a recursive update that takes over when the input is sufficiently rich to effectively update every time step. The initial batch data was necessary to give legitimate initial conditions to the recursive least squares (RLS) algorithm. For non-rich data, the RLS algorithm was converging to significantly wrong local minima when random initial conditions were given. In order to frame the dynamic equations to be compatible with least squares estimation, the state equations were parameterized using 5 variables as shown below in Equations 21-25 as:

$$a_1 = m_1 l_{c1}^2 + m_2 l_1^2 + m_2 l_{c2}^2 + I_1 + I_2 \tag{21}$$

$$a_2 = m_2 l_1 l_{c2} \tag{22}$$

$$a_3 = m_2 l_{c2}^2 \tag{23}$$

$$a_4 = (m_1 l_{c1} + m_2 l_1) g \tag{24}$$

$$a_5 = m_2 l_{c2} g \tag{25}$$

These 5 parameters are then plugged into Equations 2 and 3 and using the shorthand defined in Equations 26-29 below

$$c_{12} = \cos(q_1 + q_2), \tag{26}$$

$$c_1 = \cos(q_1), \tag{27}$$

$$c_2 = \cos(q_2), \tag{28}$$

$$s_2 = \sin(q_2), \tag{29}$$

the dynamic equations of the acrobot system become Equations 30 and 31,

$$a_1 \ddot{q}_1 + a_2(2c_2\ddot{q}_1 + c_2\ddot{q}_2 - s_2\dot{q}_2^2 - 2s_2\dot{q}_2\dot{q}_1) + a_3\ddot{q}_2 + a_4 c_1 + a_5 c_{12} = 0, \tag{30}$$

$$a_1(0) + a_2(c_2\ddot{q}_1 + s_2\dot{q}_1^2) + a_3(\ddot{q}_2 + \ddot{q}_1) + a_4(0) + a_5 c_{12} = \tau. \tag{31}$$

These equations are not especially convenient to perform least squares on separately because equation 31 especially gives rise to a notably insufficiently rich set of inputs values to the least squares algorithm. In an attempt to resolve this, Equations 30 and 31 above are combined to create Equation 32,

$$\begin{aligned}&a_1\ddot{q}_1 + a_2(3c_2\ddot{q}_1 + s_2\dot{q}_1^2 + c_2\ddot{q}_2 - s_2\dot{q}_2^2 - 2s_2\dot{q}_2\dot{q}_1) + \\ &a_3(2\ddot{q}_2 + \ddot{q}_1) + a_4c_1 + 2a_5c_{12} = \tau\end{aligned}, \tag{32}$$

a significantly more diverse input equation which also maintains the final linear form that will be used in the least squares parameter estimation of $a_1 \ldots a_5$. Equation 33 below

$$a_1u_1 + a_2u_2 + \ldots + a_mu_m = y\,. \tag{33}$$

is the basic linear form required for least squares estimation. Our simulation environment assumes $q_1, \dot{q}_1, q_2, \dot{q}_2$ are all known, so if we estimate $\ddot{q}_1, \ddot{q}_2$ as Equation 34 below

$$\ddot{q}_i = \frac{\dot{q}_i(t) - \dot{q}_i(t-1)}{dt}, \tag{34}$$

Then we have all of the information to needed to compute the lease squares inputs $u_1 \ldots u_5$. The formula for least squares estimation then requires that we define a vector of parameters as Equation 35 below

$$a = [a_1 \ldots a_m]^T, \tag{35}$$

as well as a vector of inputs, shown as Equation 36

$$\varphi = [u_1 \ldots u_m]^T. \tag{36}$$

then we can rewrite Equation 33 in its companion vector form as Equation 37,

$$y = \varphi^T a\,. \tag{37}$$

From this form we can refer to the formula for Least Squares approximation batch processing from [4] and this becomes Equation 38 below

$$\hat{a} = PB \tag{38}$$

where P is defined below as Equation 39

$$P = \left[\sum_{t=1}^{N} (\varphi(t)\varphi^T(t)) \right]^{-1}, \tag{39}$$

and B is defined as Equation 40

$$B = \sum_{t=1}^{N} y(t)\varphi(t) .. \tag{40}$$

This batch algorithm is very good a building up a P^{-1} matrix with sparse data (as it occurs when the system is either near its relaxed vertical or is stabilizing at the top.) During early swing up the parameters remain at their initial conditions as defined (all 1's). The algorithm decides when the determinant of the P matrix is greater than 1 to seed the RLS algorithm with this batch computation.

The RLS algorithm creates a refined parameter estimate every time step. The parameter update equation is shown below as Equation 40 is derived in [4]

$$\hat{a}(t) = \hat{a}(t-1) + \frac{P_{t-1}\varphi(t)}{(1+\varphi^T(t)P_{t-1}\varphi(t))}\left(y(t) - \varphi^T(t)\hat{a}(t-1)\right), \tag{40}$$

This update also requires an update in P matrix as shown below as Equation 41 which is also derived in [4]

$$P_t = P_{t-1} - \frac{P_{t-1}\varphi(t)\varphi^T(t)P_{t-1}}{(1-\varphi^T(t)P_{t-1}\varphi(t)}. \qquad (41)$$

The initial parameter estimates are the result of the batch estimate shown in Equation 42,

$$\hat{a}(0) = PB \qquad (42)$$

The initial condition for P_0 is just a 5x5 identity matrix.

Figure 3 below shows plots of estimated parameters (solid line) versus the true value of the parameters (dotted line) with respect to time. As we can see from Figure 3, the first fraction of a second is very chaotic with respect to the parameter estimates. This corresponds to when the acrobot has just started moving and the least squares algorithm has yet to measure a diverse enough set of inputs to warrant a precise measurement. In well under a second we can see the estimated value of the parameters jump to somewhat consistent values and slowly begin to converge to near their true values. Each of the parameters converges to their true value within 5%. This is more than enough accuracy to accurately perform PFL, and as we will see in Figure 4, this is good enough to suggest complete convergence of the LQR controller.

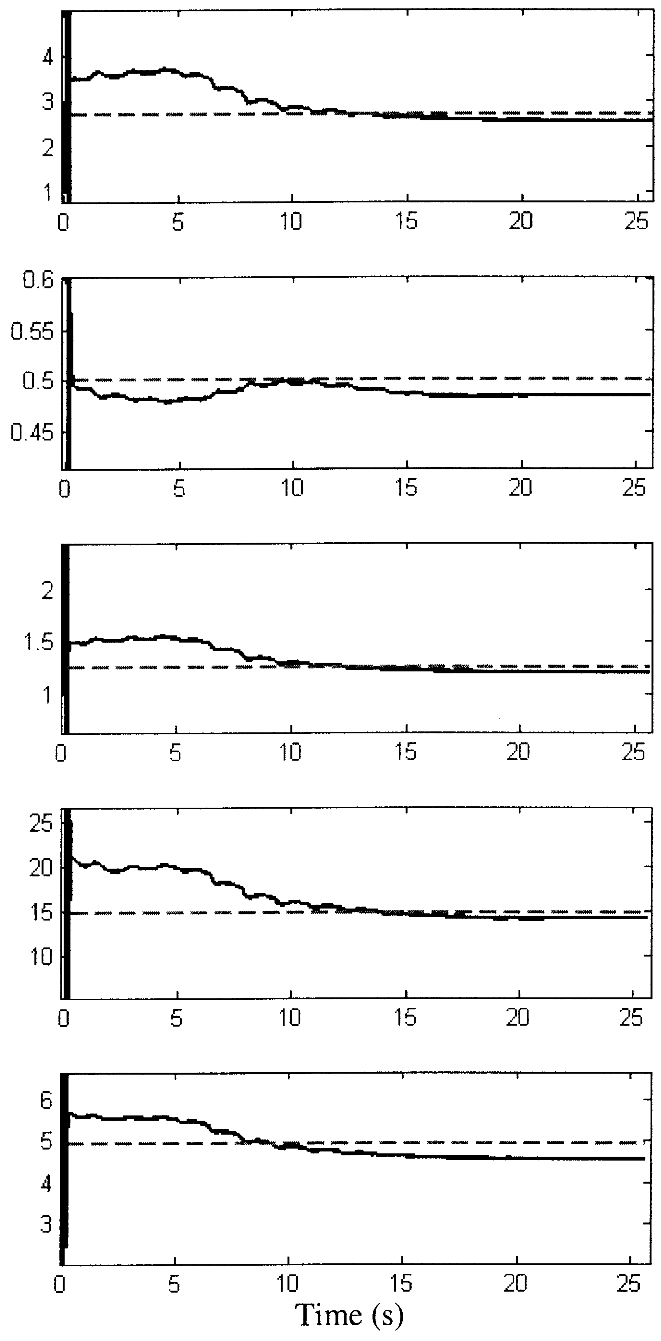

Figure 3: Plots of estimated parameter values versus true value of the parameter values with respect to time. The dotted line represents the true parameter values and the solid line represents the time history of the estimated parameter values.

A note about this method is that it will not handle time varying parameters well, but an adaptation of the Recursive least squares method using a decaying exponential of previous estimate can be used. The equations for this process are shown below as Equations 43 and 44 and are documented in [4],

$$\hat{\theta}_\beta(t) = \hat{\theta}(t-1) + \frac{P_{t-1}\varphi(t)}{(\beta + \varphi^T(t)P_{t-1}\varphi(t))}\left[y(t) - \varphi^T(t)\hat{\theta}(t-1)\right] \quad , \tag{43}$$

$$P_t = \frac{1}{\alpha}\left[P_{t-1} - \frac{P_{t-1}\varphi(t)\varphi^T(t)P_{t-1}}{(\alpha + \varphi^T(t)P_{t-1}\varphi(t)}\right], \tag{44}$$

This exponential weighting method is not implemented but could be substituted if the arm parameters are expected to vary in time continuously or in an undetectable way such that we would need to autonomously pick up the change. It is worth note that this method is slightly complicated by the nature that P_t decreases exponentially fast depending on the value of β. This requires a P_t resetting every 20 or so steps, so complications may arise with resetting P matrices during a series of insufficiently rich data collections. For these reasons this addition is not implemented in the final model.

LQR stabilization

The above swing up methods will only get the acrobot near its upright balancing position. When the acrobot is close to stabilizing near the top an LQR method is used to converge the system to balancing upright in the configuration $q_1 = \frac{\pi}{2}.$, $\dot{q}_1 = 0$, $q_2 = 0$, and $\dot{q}_2 = 0$. In order to create this LQR controller, the non-linear dynamics need to be linearized around the upright position. To do this, we first change variables defining q_1* in Equation 45 below as

$$q_1* = q_1 - \frac{\pi}{2}. \tag{45}$$

The non-linear terms there then linearized. This means that all of the second order terms are considered zero and the trigonometric functions are then converted via Equations 46 and 47 shown below

$$\cos(q_1) = -q_1 * \tag{46}$$

$$\cos(q_1 + q_2) = -[q_2 + q_1 *] \tag{47}$$

When the original non-linear equations (Equations 30 and 31) are linearized, we obtain the following differential equations shown as Equations 48 and 49

$$\ddot{q}_1 = \frac{(a_3a_4 - a_2a_5)q_1 * - a_2a_5q_2 - (a_3 + a_2)\tau}{a_1a_3 - a_3^2 - a_2^2} \tag{48}$$

$$\ddot{q}_2 = \frac{(a_1a_5 + a_2a_5 - a_3a_4 - a_2a_4 - a_3a_5)q_1 * + (a_1a_5 + a_2a_5 - a_3a_5)q_2 + (a_1 + 2a_2)\tau}{a_1a_3 - a_3^2 - a_2^2} \tag{49}$$

We can then define our state space as shown in Equation 50 as

$$q = (q_1 *, \dot{q}_1, q_2, \dot{q}_2)^T \tag{50}$$

Then transforming the above differential equations into a state space representation we get an equation shown below, Equation 51

$$\dot{q} = Aq + B\tau \tag{51}$$

Where the A matrix is defined in Equation 52,

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{(a_2a_3 - a_2a_5)}{a_1a_3 - a_3^2 - a_2^2} & 0 & \dfrac{-a_2a_5}{a_1a_3 - a_3^2 - a_2^2} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{(a_1a_5 + a_2a_5 - a_3a_4 - a_2a_4 - a_3a_5)}{a_1a_3 - a_3^2 - a_2^2} & 0 & \dfrac{(a_1a_5 + a_2a_5 - a_3a_5)}{a_1a_3 - a_3^2 - a_2^2} & 0 \end{bmatrix} \tag{52}$$

and the B matrix is defined in Equation 53 as

$$B = \begin{bmatrix} 0 \\ \dfrac{-a_3 a_2}{a_1 a_3 - a_3^2 - a_2^2} \\ 0 \\ \dfrac{(a_1 + 2a_2)}{a_1 a_3 - a_3^2 - a_2^2} \end{bmatrix}. \tag{53}$$

The weighting matrices used for the optimization were based on those from [2], where the Q weighting matrix is defined as Qw in Equation 54

$$Qw = \begin{bmatrix} 1000 & 0 & -500 & 0 \\ 0 & 1000 & 0 & -500 \\ -500 & 0 & 1000 & 0 \\ 0 & -500 & 0 & 1000 \end{bmatrix} \tag{54}$$

and the R matrix is defined in Equation 55 as

$$Rw = 0.5\,. \tag{55}$$

The Qw matrix penalizes deviations from the stabilization point as well as penalizing the cross coupling terms of $q_1 * q_2$ and $\dot{q}_1\dot{q}_2$ for being negative.

The actual implementation in code uses the MATLAB function *lqrd* which takes the continuous time Equation 51 and transforms it into a discrete time function with the simulation time step dt. This helps the optimal gains to be more successful because of the discrete nature of the input update.

Figure 4 below shows a plot containing the error's of estimated parameters in percent along the x versus the probability of convergence of the LQR controller on the y axis; for statistical sample sizes of 30. What this chart effectively shows is that the LQR gains work as if they were ideal until the estimated parameter value errors become about 6%. The data shown in figure 3 showed an estimated error of around 5% maintaining a theoretical 100% probability of convergence on the LQR controller within its basin of attraction even with the non-ideal parameter updates.

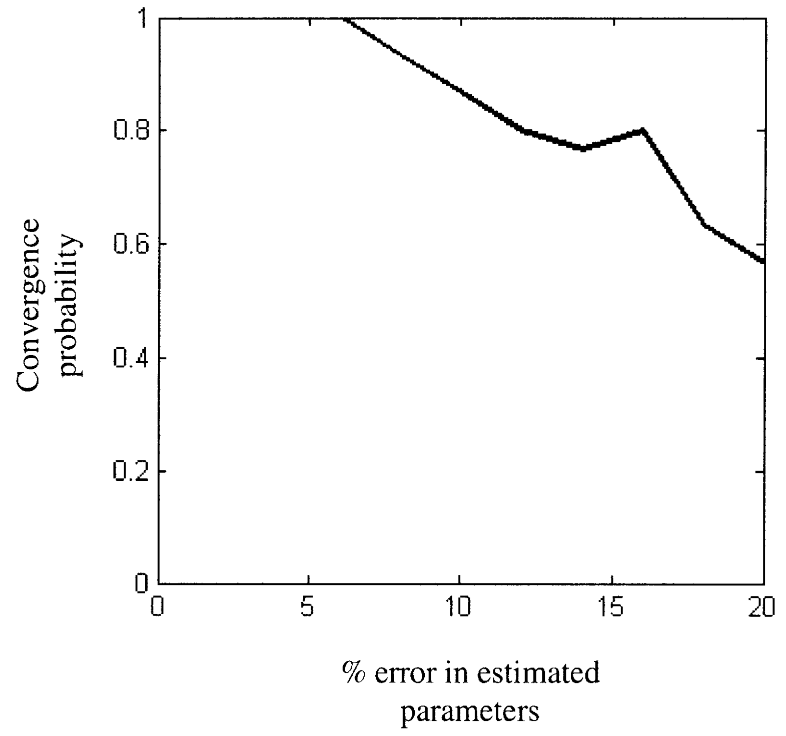

Figure 4: Plot showing the errors of estimated parameters in percent along the x versus the probability of convergence of the LQR controller; for a statistical sample size of 30.

Results

The overall results of the effort were positive but there are definitely optimizations that can be made with this approach and during the course of the project some possible improved methods were revealed. As can be seen in the state trajectory of Figures 5-8, the system is able to converge marginally well. It is pretty clear in the plots that the LQR controller kicks in around 22 (s) and only requires some small corrections to get the system onto a stabilization trajectory.

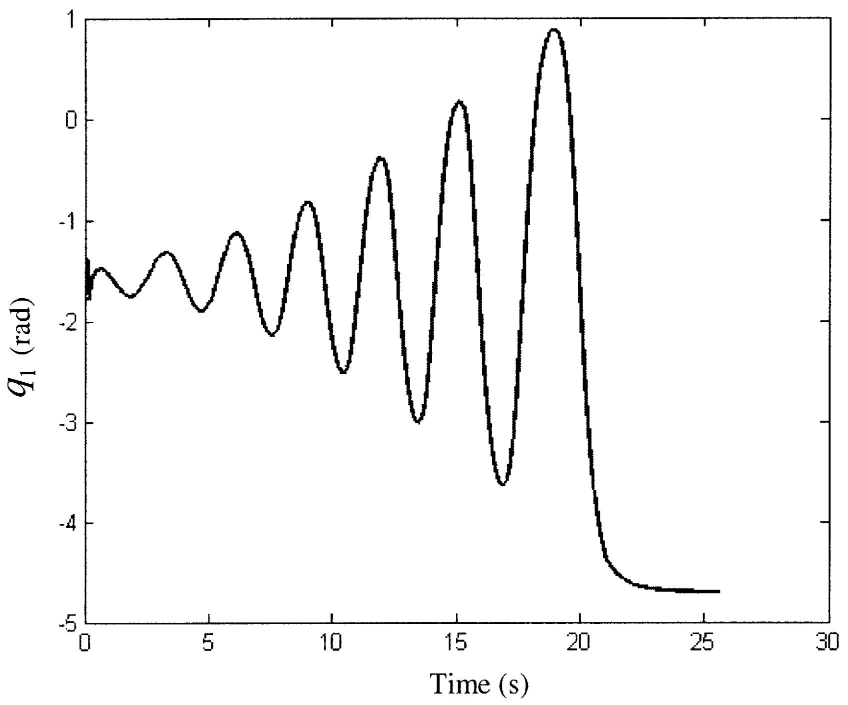

Figure 5: Trajectory of q_1 for a successful capture (time versus angular displacement)

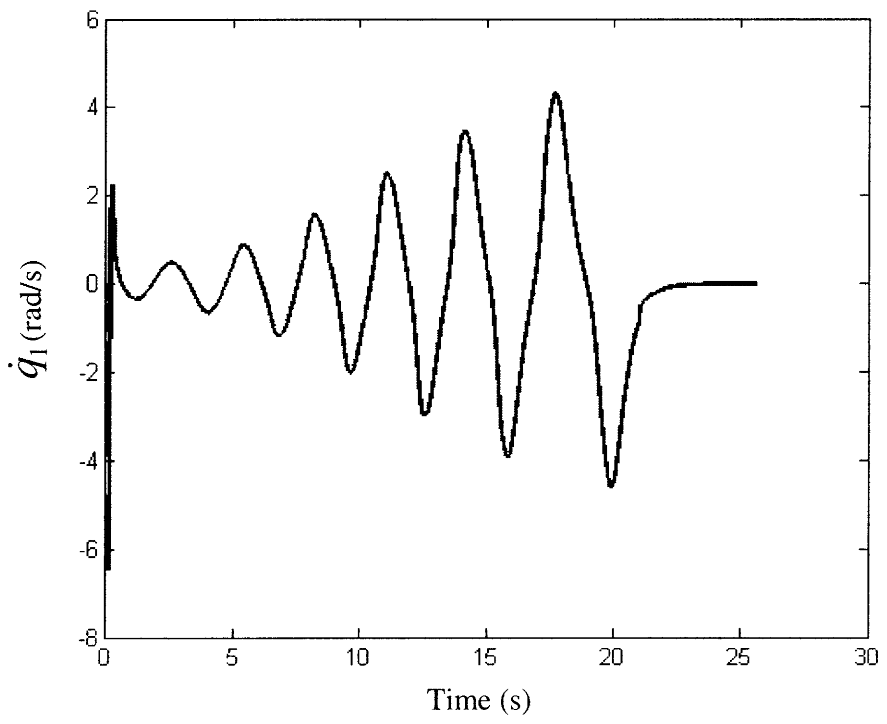

Figure 6: Time trajectory of the first derivative of q_1 for a successful capture (time versus angular rate).

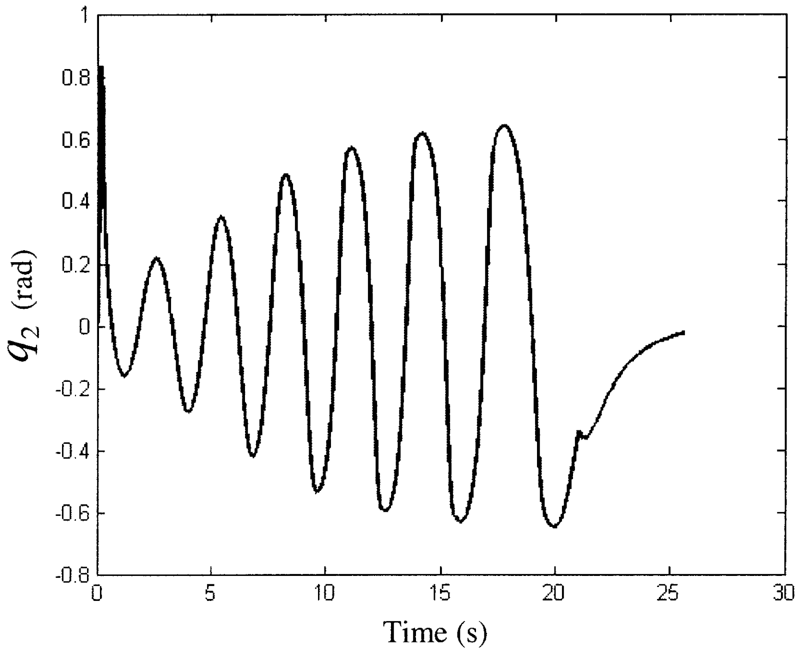

Figure 7: Time trajectory of q_2 for a successful capture (time versus angular displacement.)

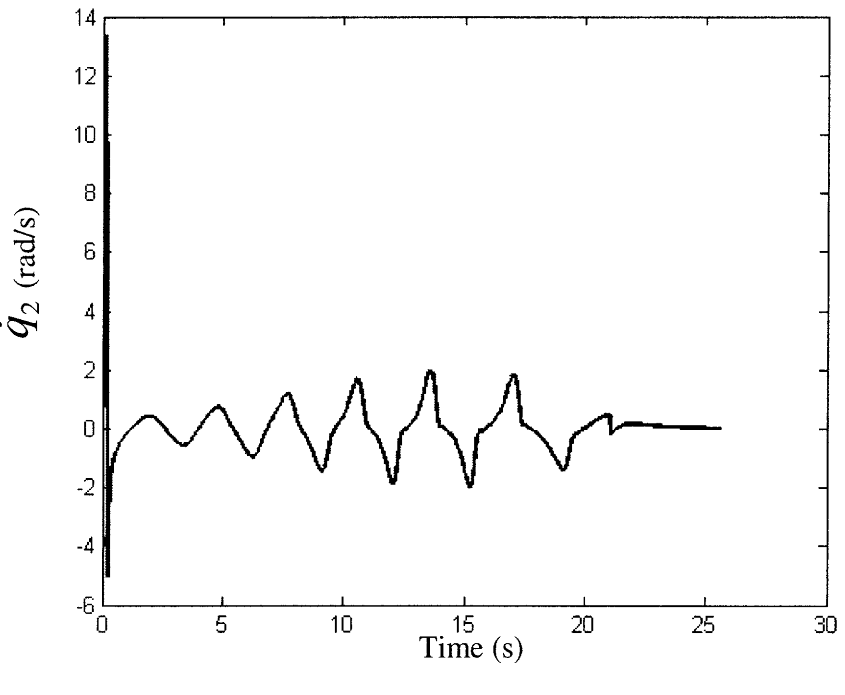

Figure 8: Time trajectory of the first derivative of q_2 for a successful capture (time versus angular rate.)

In 32 of 180 trials from random initial conditions the entire system managed to swing up. This is a 22% success rate which seems acceptable with the implemented modest torque limits and now seemingly naïve swing up controller.

Looking back over the project the parameter estimation worked very well, the swing up controller worked well by itself and the LQR controller necessarily worked given the initial conditions were within the basin of attraction of the discrete time, torque limited system. The biggest hole in the approach was the transition between the swing up controller and the LQR stabilizer. All that the swing up controller was trying to accomplish was to continuously pump energy into the system. This only works well if the controller pumps the correct amount of energy into the system such that it enters the basin of attraction of the LQR system. The majority of the random initial condition trials were plagued by a swing just under the energy of the LQR basin, followed by a swing just over the energy of the LQR basin. Giving the swing up controller more knowledge of its goal once it was close would have been a huge improvement here.

It still seems prudent to use this swing up controller when the acrobot is beginning to swing up because this is when parameter estimates are poor and energy based swing up methods would be prone to getting stuck in in-escapable loops near the stable equilibrium state.

Another thing that would have made the problem easier but less interesting would be to allow much higher torques in the LQR controller. With sufficient torque an LQR controller can even power its way through a non-linear region to balance. An attempt that I used to find a basin of attraction of the LQR controller was to store the entrance state into the LQR stabilizer for both the cases when the system failed and when it managed to converge well. Using a least squares parameter estimation technique similar to the one used to estimate the dynamic parameters I was able to obtain the following Equations 56 and 57 for if the system was in the basin of attraction of the LQR controller

$$all((q_1{*}, \dot{q}_1, q_2, \dot{q}_2) < 0.4) \tag{56}$$

$$\text{isBasin} = round(sign(q_1{*})(q_1{*}, \dot{q}_1, q_2, \dot{q}_2)(.32, -.55, -.35, -.19)^T) \tag{57}$$

Given a large number of trials this method would never give any false positives, and would only give false negatives around 30% of the time. Unfortunately, the swing up controller would almost never get the system into the regime required for these convergence criteria.

Conclusion

This thesis implements an online, adaptive swing-up and balancing controller with no previous knowledge of the system's mass or geometric parameters. A least squares method then successfully identifies the 5 parameters necessary to completely characterize acrobot dynamics. Swing up is accomplished using partial feedback linearization and a pump up strategy to add energy to the system. The controller then 22% of the time catches the swung up system in the basin of attraction of an LQR controller computed using the estimated parameter values generated from the online system identification.

References

[1] Slotine, J., Li, W., *Applied Nonlinear Control*, Prentice-Hall, Inc., New Jersey, 1991
[2] Spong, M.W., "Swing Up Control of the Acrobot," *1994 IEEE Int. Conf. on Robotics and Automation,* pp. 2356-2361, San Diego, CA, May, 1994.
[3] Araki, N.; Okada, M.; Konishi, Y.; Ishigaki, and H.;. "Parameter identification and swing-up control of an acrobot system." International Conference on International Technology, 2005.
[4] Asada, H. "2.160 Identification, Estimation, and Learning: Lecture Notes No. 2: February 9, 2009". Department of Mechanical Engineering, MIT 2009