Balancing Controller for a Four-Legged, Bipedal Walking Robot

Matt Haberland

1291 Shaker Woods Rd.
Herndon, VA 20170
`mdh52@cornell.edu`

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Engineering

Mechanical Engineering

Cornell University

Spring 2007

Abstract

A controller is developed to balance a two-dimensional walking robot. The equations of motion for the robot, modelled as a double pendulum with pivots at the hip and contact foot, are derived, linearized, and arranged in state-space form. A Linear Quadratic Regulator is developed for the linearized model; its performance simulated using both the linearized and the complete, nonlinear dynamics. The regulator fails to stabilize a physical four-legged biped due to measurement difficulties, and recommendations for measurement and controller improvement are proposed in response.

1 Introduction

Among the popular approaches toward creating a walking machine is to model the geometry of the machine after that of the human body and precisely control joint angles in time to mimic human walking motions. While machines designed according to this approach are versatile and robust, they are extremely inefficient, and as a result support only short runtimes given current battery technology. The 'passive dynamic' approach, on the other hand, is to model dynamic properties of the machine after those of the human body but offer no control or power other than that of gravity. These machines are extremely efficient, but not at all versatile or robust, eliminating practical use. An approach in between these extremes is to apply passive dynamic principles for efficiency, but provide powered control as necessary to achieve greater versatility and robustness.

The 'Cornell Ranger', a 'four-legged bipedal' walking robot [1], was designed and fabricated according to this hybrid approach. It has acheived a record combination of efficiency and robustness, with the proven ability to walk over 1km without falling. This paper discusses the development of a controller to balance this robot upright on its outer legs by swinging the inner legs, without taking a step. In Section 2, the equations of motion describing the robot's behavior are derived, linearized, and arranged in state-space form. Section 3 discusses the development of a Linear Quadratic Regulator (LQR) for the state-space model. Theoretical regulator performance is characterized and improved using a fully nonlinear simulation in Section 4 and implemented on the robot in Section 5. Actual regulator performance is presented in Section 6, suggestions for improvement are made in Section 7, and conclusions are drawn in Section 8.

Parameter	Value	Units	Parameter	Value	Units
m_a	3.135	kg	m_b	2.4015	kg
I_a	.2857	kgm^2	I_b	.2607	kgm^2
a	.835	m	b	.217	m
l	1	m	g	9.81	$\frac{\text{m}}{\text{s}^2}$

Table 1: Cornell Ranger parameters, as measured during assembly

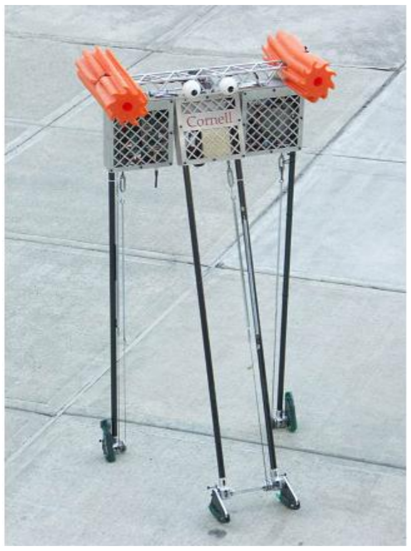
2 Modelling

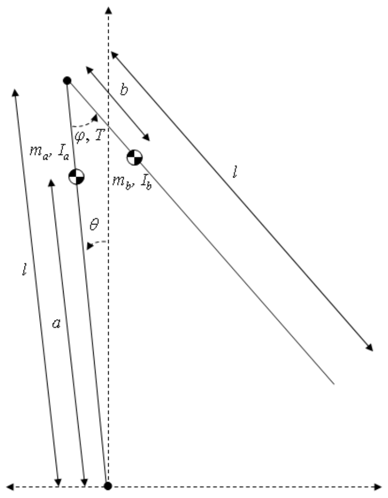
The Cornell Ranger shown in Figure 1(a) can be modelled as the double pendulum shown in Figure 1(b). The outer pair of legs is represented by a rigid bar of mass m_a, length l, and moment of inertia I_a measured about the Center of Mass (CoM), which is located a from the foot contact with the floor, modelled as a simple pivot. The inner pair of legs is represented by a rigid bar of mass m_b, length l, and moment of inertia I_b about the CoM, which is located b from the hip connection with the outer legs, modelled as a simple pivot. The numerical values of these parameters for the Cornell Ranger are listed in Table 1. The angle θ is measured positive counterclockwise from the vertical (nominally normal to the floor, but defined as parallel with the gravity vector) to the outer legs, the angle ϕ is measured positive counterclockwise from the outer legs to the inner legs, and internal torque T is positive counterclockwise.

Applying the angular momentum equation [2] to the entire system about point O, as represented in Figure 2(a), yields

$$\begin{aligned}(am_a + lm_b)g\sin(\theta) - bm_b g\sin(\theta+\phi) = \\ (I_a + I_b)\ddot{\theta} + I_b\ddot{\phi} + \\ (m_a a^2 + m_b l^2 + m_b b^2)\ddot{\theta} + m_b b^2\ddot{\phi} - \\ m_b lb(2\ddot{\theta} + \ddot{\phi})\cos\phi + m_b lb(2\dot{\theta}\dot{\phi} + \dot{\phi}^2)\sin\phi. \quad (1)\end{aligned}$$

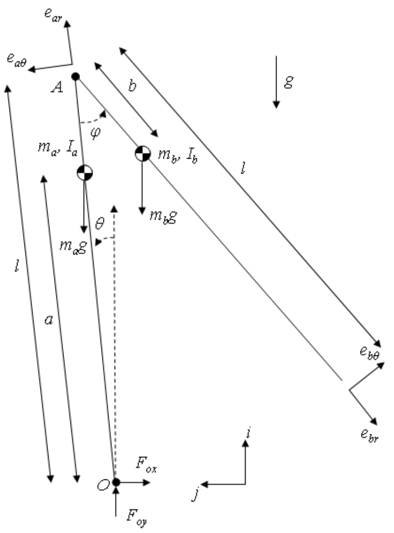
Considering the inner legs only, angular momentum


(a) Cornell Ranger

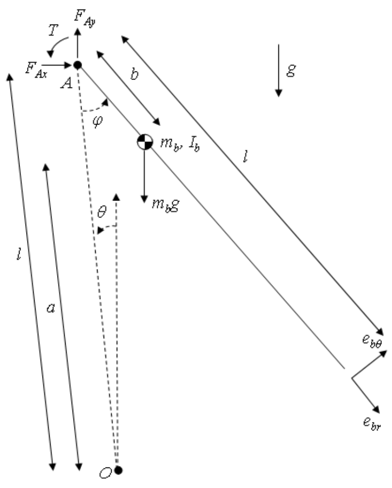


(b) Double Pendulum

Figure 1: The Cornell Ranger, modelled as a double pendulum



(a) Entire System



(b) Inner Legs Only

Figure 2: Free body diagrams of the double pendulum model

balance about point A, as illustrated in Figure 2(b), yields

$$T - bm_b g\sin(\theta+\phi) = \\ I_b(\ddot{\theta}+\ddot{\phi}) - m_b l b\ddot{\theta}\cos\phi - \\ m_b l b\dot{\theta}^2\sin\phi + m_b b^2(\ddot{\theta}+\ddot{\phi}). \quad (2)$$

Defining

$$\begin{aligned} I_{11} &= I_a + I_b + m_a a^2 + m_b(l-b)^2, \\ I_{12} &= I_b + m_b(b^2 - lb), \\ I_{22} &= I_b + m_b b^2, \\ M_{g1} &= -g[am_a + m_b(l-b)], \text{ and} \\ M_{g2} &= bm_b \end{aligned}$$

and linearizing about $\theta = 0$ and $\phi = 0$, the two scalar Equations 1 and 2 can be represented as a single matrix equation,

$$\begin{bmatrix} I_{11} & I_{12} \\ I_{12} & I_{22} \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} + \begin{bmatrix} M_{g1} & M_{g2} \\ M_{g2} & M_{g2} \end{bmatrix} \begin{bmatrix} \theta \\ \phi \end{bmatrix} = \begin{bmatrix} 0 \\ T \end{bmatrix} \quad (3)$$

Adding states to convert the second-order equation to an equivalent first-order form yields

$$\overbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & I_{11} & 0 & I_{12} \\ 0 & 0 & -1 & 0 \\ 0 & I_{12} & 0 & I_{22} \end{bmatrix}}^{I} \overbrace{\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix}}^{\dot{x}} + \quad (4)$$

$$\overbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ M_{g1} & 0 & M_{g2} & 0 \\ 0 & 0 & 0 & 1 \\ M_{g2} & 0 & M_{g2} & 0 \end{bmatrix}}^{-IA} \overbrace{\begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix}}^{x} = \overbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}^{IB} \overbrace{T}^{u}.$$

Finally, the equation can be cast in standard state-space form by adding IAx to both sides, then premultiplying both sides by I^{-1}, as

$$\dot{x} = Ax + Bu. \quad (5)$$

3 Control

We seek a control law that will prescribe a torque at the hip as a function of the robot's state, with a goal of keeping the robot 'balanced'. At a system level, we say that the 'input' is the hip torque, the 'output' is a measure of the robot's deviation from a perfectly 'balanced' state, and the controller must prescribe an input that will drive to zero, or regulate, the output based on state feedback. The choice of the measure of 'balance' is not trivial, however; possible elements of the definition include:

- The outer (stance) legs are vertical,
- The inner (swing) legs are parallel with the stance legs,
- The legs are not moving,
- The center of mass of the robot lies directly above the contact point,
- The 'zero-moment point' of the robot lies within the contact patch of the foot [3].

Depending on the definition of balance, the system can be seen as 'single-input, single-output' (SISO) as motivated in Appendix A, for which controller design can be handled by classical, single variable control theory. Alternatively, there could be multiple requirements for balance, and thus the system would be seen as 'single-input, multiple-output' (SIMO), and controller design calls for multivariable control methods. For simplicity, we will tentatively define our output as the angle of the stance leg with respect to the vertical, or

$$y = Cx = \theta, C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}. \quad (6)$$

and choose our definition of balance to be $\theta = 0$, thus our output and performance variable y is a measure of deviation from the balanced state. Since it is difficult to generate ahead of time the definition of balance that will yield the 'best' system performance, we may desire to change this definition later, and thus it is preferable to choose a general procedure that yields a controller for a wide variety of balance definitions.

The Linear Quadratic Regulator algorithm (LQR), provides such a framework [4] [5]. LQR is used to find a linear feedback control law that minimizes a quadratic cost function of performance variables and control effort (our system input, which is simply the motor torque). Mathematically speaking, we can use LQR to determine a gain matrix K such that for a system with the dynamics described by Equation 5, input $u = -Kx$ minimizes the value of a cost function of the form

$$J = \int_0^\infty (x^T Qx + u^T Ru)dt. \tag{7}$$

As it is unclear, at this stage, how 'balance' should be defined, so it is uncertain what cost function should be minimized to yield the 'best' controller. The form of this cost function seems *reasonable* for the task of balancing the robot, as the term $x^T Qx$ can provide a measure of the deviation from the balanced state, $u^T Ru$ can provide a measure of control effort, and minimization of the time integral of these terms guarantees, in some sense, an efficient, balancing controller. We proceed by generating a controller according to this algorithm, reserving judgement until we can analyze the strengths and weaknesses of the resulting controller, at which point another cycle of the iterative design process will begin.

For the purposes of this application, $Q \triangleq C^T C$, and thus for our prelimary choice of C, $x^T Qx = \theta$. We choose a scalar R to weight the importance of minimizing control effort relative to minimizing the integral of the deviation from the balanced state.[1]

The MATLAB function `lqr(A,B,Q,R)` yields the optimal gain matrix $K = R^{-1}B^T P$ by solving the reduced-matrix Ricatti equation,

$$A^T P + PA - PBR^{-1}B^T P + Q = 0, \tag{8}$$

for a positive-definite matrix P.

Finally, it can be verified that the controller $T = f(x)$ is

$$T = [\ 0 \quad 0 \quad 0 \quad 1\ ]IBKx. \tag{9}$$

4 Simulation

4.1 Linearized Dynamics

Once the gain matrix K is generated, the response of the linearized system is obtained using the familiar MATLAB command `[y x] = lsim(A-B*K,B,C,D, u, t, x0)`, where `x`, `A`, `B`, and `K` are as previously defined, `y`, `C`, and `D` are not used, `t` is a vector of time values, `u` is zero (as the control input is contained in the term `-B*K`), and `x0` is the initial state vector. A plot of the stance leg angle θ = `x(:,1)`, swing leg angle ϕ = `x(:,3)`, and control torque T = `-I*B*K*x'` as functions of time vector `t` is generated using the `plot()` function.

4.2 Complete, Nonlinear Dynamics

Nonlinear simulation is performed using Professor Andy Ruina's 'Double Pendulum Simulation' m-code, which, given parameters and initial conditions, derives and integrates the equations of motion for a double pendulum and animates the results. The code has been modified to derive the Linear Quadratic Regulator for the given parameters and implement the resulting control law for torque at the hip. For consistency with the Cornell Ranger, a torque saturation limit of approximately 5 Nm is enforced in the dynamics.

4.3 Initial Simulation Results

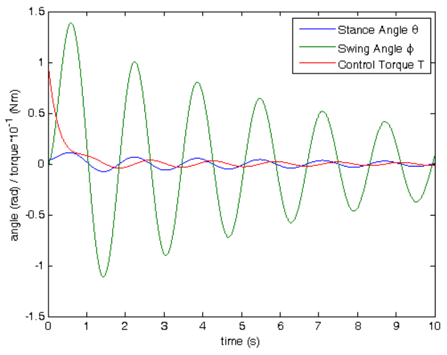
For $R = .03$ and with very small, single state initial condition $\theta(t = 0) = .02$, the controller effectively stabilizes the double pendulum as reflected by both linear and nonlinear simulation results in Figure B-1. For slightly larger initial condition $\theta(t = 0) = .03$ rad, both the linear and nonlinear model predict stable behavior in Figure B-2, but the nonlinear simulation shows that the hip motor begins to saturate. For even larger (but still very small) initial condition $\theta(t = 0) = .04$ rad, the linear model predicts similar, stable response to the initial conditions, as in Figure 3(a). The nonlinear model, however, reveals that the controlled double pendulum is unstable, as in Figure 3(b).

The simulation results show that this particular controller fails to regulate the simulated system even for small initial values of θ, all other initial state variables being zero. It is clear that the controller needs to be tuned before it will be ready for use on the Cornell Ranger.

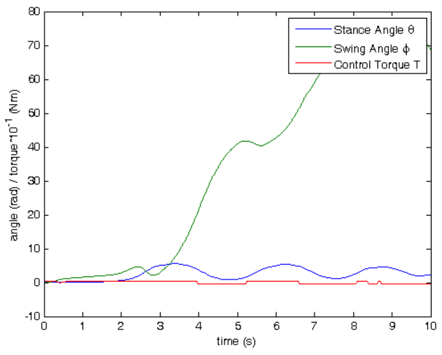
4.4 Controller Tuning

Even though the controller is 'optimal' in the sense that it minimizes the value of a particular cost function, mini-

[1] In general, the choice of R can be as debatable as that of Q. However, for this system the input vector u is simply the scalar T. Moreover, $T \propto i$, the current into the motor,$\int i^2 dt$, as it appears in the cost function, is proportional to the total energy used to regulate the system, and energy is a particularly meaningful measure of control effort. Unless otherwise specified, R is chosen to be .03, i.e. energy efficiency is judged to by only 3% as important as stabilization.



(a) Linear Simulation



(b) Nonlinear Simulation

Figure 3: Simulated response of the controlled double pendulum to initial condition $\theta(t = 0) = .04$

mization of this particular cost function does not necessarily yield the 'best' performance. Thus, we vary R, without changing the definition of our performance variable through Q, to change the cost function and tune the system for better performance. Classical measures of performance include response time and relative stability, but since the reference input is constant and the system is nonminimum phase, we are more interested in the system's damping. The system is actually nonlinear, so the size of the basin of attraction is of great importance, and since the system is physical and measurements are imperfect, robustness is also critical to the controller's success.

4.4.1 Damping

The term 'damping', for a linear system, is related to real parts of the system's eigenvalues. The unregulated system has four eigenvalues: one negative real eigenvalue, one positive (unstable) real eigenvalue, and two complex conjugate eigenvalues with real part zero. The controlled, linearized system also has four eigenvalues: two complex conjugate pairs, both with negative real parts. Plots of the real part of each of these conjugate pairs are shown in Figure B-4. Most notably, the magnitude of the (negative) real parts of both of these eigenvalues increase asymptotically with decreasing R. This is consistent with intuition, because decreased R means greater relative cost of the error integral in the cost function, and minimizing this 'balancing error' requires greater damping.

4.4.2 Basin of Attraction

The term 'basin of attraction' refers to the set of initial states for which the controller can successfully stabilize the system. For this system, the basin of attraction is a four-dimensional volume and is a function of the $n \times 4$ matrix Q and scalar R. The challenge of finding the controller that will absolutely maximize the volume of this basin of attraction, even restricting the controller to a Linear Quadratic Regulator, is an intractable problem because of the number of degrees of freedom provided by Q and R and the high dimension of the basin of attraction. To simplify the problem, we will consider each dimension separately. For example, will measure, for a controller generated using a particular value of R, the maximum value of θ^* for which the initial state

$$x(t=0) = \left|\begin{bmatrix} \theta & \dot{\theta} & \phi & \dot{\phi} \end{bmatrix}^T\right|_{t=0} = \begin{bmatrix} \theta^* & 0 & 0 & 0 \end{bmatrix}^T$$

lies in the basin of attraction. Generated by an iterative script, a plot of $\theta^*(R)$ is shown in Figure 4; plots of similarly defined $\phi^*(R)$, $\dot{\theta}^*(R)$, and $\dot{\phi}^*(R)$ are available in Figure B-5. Most notably, all of these simplified measures of basin of attraction increase asymptotically with increasing R. Also, relative to the expected magnitudes of initial

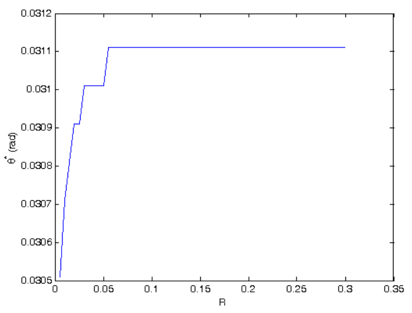


Figure 4: The effect of control cost weight R on maximum initial stance angle θ^* lying in the basin of attraction of the controlled double pendulum system

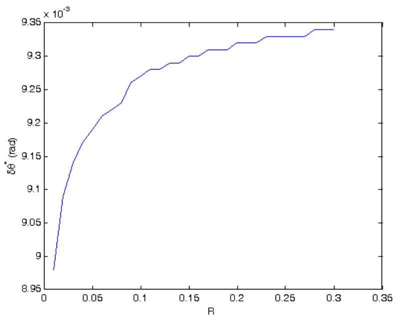


Figure 5: The effect of control cost weight R on maximum constant stance angle measurement error $\delta\theta^*$ for which the controlled double pendulum system is stable

states conditions, the system is most sensitive to initial displacements of the θ state.

4.4.3 Robustness

The term 'robustness' refers to the ability of the controller to perform in the presence of uncertainty. For the Cornell Ranger, the greatest single type of uncertainty is an offset from the true value. For ϕ, this is caused by gearbox backlash between the angle measuring encoder and the leg itself; for $\dot{\phi}$ and $\dot{\theta}$, offset is inherent in the gyro signals and can change after being removed during calibration. Offset error in θ, known as drift, is the result of integrating gyro offset. Again, it is an intractable problem to determine the effect of all errors in all measurements simultaneously. To further simplify the problem, we seek, for a controller generated using a particular value of R, the maximum value of $\delta\theta^*$ for which the initial state $x(0) = [0]$ lies in the basin of attraction given input

$$T = [\begin{array}{cccc} 0 & 0 & 0 & 1 \end{array}] IBK(R) [\begin{array}{cccc} \theta + \delta\theta^* & 0 & 0 & 0 \end{array}]^T.$$

Generated by an iterative script, a plot of $\delta\theta^*(R)$ is shown in Figure 5; plots of similarly defined $\delta\phi^*(R)$, $\delta\dot{\theta}^*(R)$, and $\delta\dot{\phi}^*(R)$ are available in Figure B-6. Most notably, all of these simplified measures of robustness increase asymptotically with increasing R. Also, relative to expected errors in measurement, the system is most sensitive to uncertainties in θ.

4.5 Performance Trends

Several trends are clear from the results:

- All simplified measures of robustness and basin of attraction increase asymptotically with increasing R,
- Both measures of damping *decrease* asymptotically with increasing R.

Thus, there is a conflict among these characterizations of performance; increased robustness and basin of attraction apparently come at the cost of damping. However, the primary goal is to design a controller that *works*, and to work the system needs to be robust and have a sufficiently large basin of attraction; high damping is a secondary goal. Therefore, higher values of R are preferable.

Another trend is that the system is, in some sense, most sensitive to initial displacements and measurement offset of the θ state. The fact that the system is sensitive to displacements θ seems to confirm the choice of performance variable and output $y = \theta$ as the variable to regulate for a greater basin of attraction. The fact that the system is also most sensitive to measurement offset in θ, however, suggests that choice of a different performance variable would be beneficial for increased robustness. Investigation of the use of alternative performance variables by changing Q reveals only marginally better performance. For example, regulating the horizontal location of the center of mass yields less than 1% increase in each of the four measures of robustness discussed in 4.4.3. Thus, while changing the performance variable yields some improvement and the final controller will be designed to regulate the position of the center of mass, we do not perform an exaustive search for the best Q.

Finally, even with high R, the system is not very robust relative to expected error in measurement, and the basin of attraction is not very large compared to possible initial conditions. While initial conditions can be controlled to some extent, measurement error remains a concern. We have exausted variation of the cost function with respect to R and shown that changing performance variables through Q does not have a significant effect. The controller has been tuned, and still the system performance is unacceptable. As a last resort, we tune the system itself:

Addition of mass δm_2 at the bottom of the swing leg increases θ^*, as shown in Figure 7(c), and, up to a point, all four measures of robustness, as in Figure B-8. While the values of ϕ^*, $\delta\phi^*$, and $\delta\dot{\phi}^*$ all peak and begin to fall within the domain of additional masses simulated, the increases in θ^* and $\delta\theta^*$ outweigh the disadvantages of using the highest reasonable value of δm_2. Therefore, $\delta m_2 = .2$kg will be added to the bottom of the swing leg, and the controller will regulate the horizontal position of the center of mass with a control effort weight $R = .3$.

5 Implementation

The control law is implemented on top of existing robot code by replacing the walk controller with a balancing controller. Sources of state feedback are summarized in Table 2. After calibration, pressing the 'Walk' button brings the robot into a 'balance standby'; the inner feet retract and the outer feet are held in an orientation that permits only the point of the heel to contact the ground. The operator attempts to balance the passive robot manually and then presses the 'CalEncoders' button to enter 'balance mode'. The reference values for the relative, integrated angles used to track θ and ϕ are set, and the torque control law is enabled. After the test, the operator may return to 'balance standby' by pressing the 'CalGyros' button to prepare for another test.

State	Variable Name	Measurement
θ	AngleGyroOuter	Integral of AngleRateGyroOuter
$\dot{\theta}$	AngleRateGyroOuter	Output of Outer Gyro
ϕ	AngleGyroHip	Difference of AngleGyroOuter and AngleGyroInner
$\dot{\phi}$	AngleRateGyroOuter	Difference of AngleRateGyroOuter and AngleRateGyroInner

Table 2: State variables, their corresponding global variable names in the Cornell Ranger code, and their method of measurement

6 Results

In 'balance standby', the robot responds as shown in Figure B-9. When released but retained firmly by its leash, the stance angle falls slowly, with the swing angle close behind, and the angular rates dip and return to near zero, all within a few seconds. When fully released, the stance angle falls rapidly and the swing angle falls even faster until the robot is saved from crashing to the ground.

In 'balance mode', the robot responds as shown in Figure B-10. When the robot is released but retained firmly by its leash, the control torque oscillates rapidly and shakes the robot violently. When fully released, the robot shakes and falls without hesitation.

Based on prior experience with the robot, the shaking is recognized to be characteristic of excessive damping, so the gain on the $\dot{\theta}$ term is relaxed. Now, after entering 'balance mode' but retained by its leash, the robot oscillates unstably about the balanced state. After being fully released from the balanced state, the robot again falls immediately. These responses are shown in Figure B-11.

7 Discussion

When implemented on the actual robot, the high damping gain on $\dot{\theta}$ is the source of an instability. Sensor noise is amplified by the damping gain, exciting small, rapid oscillations in the hip torque. The robot shakes in response to the varying torque, the sensors shake with the robot, and thus the sensor noise is apparently amplified by a combination of positive feedback, lag due to signal filtering, the discrete-time nature of the system, and time delay caused by backlash in the hip gearbox. Oscillations in the angular rates overwhelm the 'true' feedback terms, rendering the controller completely ineffective. When the damping gain

	$\delta\theta$	$\delta\phi$
Average	.02 rad	.035 rad
Max	.075 rad	.04 rad

Table 3: Approximate magnitude of angle measurement error after 10s due to integration drift

	$\delta\dot{\theta}$	$\delta\dot{\phi}$
60%	.004 rad/s	.012 rad/s
90%	.007 rad/s	.0175 rad/s

Table 4: Estimated magnitude bounds within which a given percentage of angle rate gyro noise samples lie

is relaxed, however, the controller is unable to compensate for the natural instability of the inverted pendulum. The inability to provide damping suggests that no amount of gain tuning will yield a function controller - the feedback is somehow of the wrong form given the unmodelled imperfections of the system.

Sensor noise aside, the controller's failure is still not surprising considering the sensor offset errors summarized in Table 3. The average drift of the variable θ over the course of 10 seconds is about .02 rad, and has been observed to be as high as .075 rad, while the maximum constant stance angle error under which the controller could theoretically stabilize the system is only $\delta\theta^* \approx .013$ rad. This suggests that even if the drift in θ measurement was the *only* uncertainty and the initial state was zero, then the controlled robot would become unstable within seconds.

In retrospect, the Linear Quadratic Regulator algorithm was not the correct approach toward designing the controller. This 'optimal' algorithm minimizes the integral of error over time when the closed-loop system is stable, but is not particularly suited to making the closed loop system stable in the presence of uncertainties like sensor noise and offset error. This suggests that the techniques of robust control might be better suited for the task of designing a controller for this robot.

The most significant improvement to be made is in the robot state estimation. The accelerometers must be used to provide 'absolute' angle reference. The gyro signals must be filtered further, and the offsets must be removed more effectively. Documentation of the current state estimator and more detailed suggestions for improvement are available in Appendix C.

8 Conclusion

Although the Linear Quadratic Regulator successfully stabilizes a simulated double pendulum under a reasonable range of initial conditions and measurement errors, it is unable to stabilize the Cornell Ranger. Through a combination of high damping gain, measurement noise, and time delay, the controller introduces a new instability to the system. When the damping gain is relaxed to quell this instability, the controller is unable to compensate for the natural instability of the inverted pendulum. State-estimation improvement, and possibly mass re-distribution will yield a more controllable system, and application of robust control techniques are likely to generate a successful controller.

References

[1] Karssen, Daniël. *Design and construction of the Cornell Ranger, a world record distance walking robot*. Internship Final Report. Jan 2007.

[2] Ruina, Andy and Pratap, Rudra. *Introduction to Statics and Dynamics*. Pre-release text. Oxford University Press. 2002.

[3] Vukobratović, Miomir and Borovac, Bransilav. *Zero Moment Point - Thirty Five Years of its Life*. International Journal of Humanoid Robotics, Vol. 1, No. 1 (2004) 157-173.

[4] Sherback, Mike and Mohuiddin, Shan. *Notes on the LQR Algorithm*. 2007.

[5] Ogata, Katsuhiko. *Modern Control Engineering*. Third Edition. Prentice Hall, 1997.

APPENDIX A - SISO Control of a 2 DOF System

As an illustration of the claim that a controller can be developed for the system using classical control theory, consider the system

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} \ddot{x_1} \\ \ddot{x_2} \end{bmatrix} + \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} f. \tag{A-1}$$

Decomposing the single matrix equation into two scalar equations and transforming into the Laplace domain yields

$$(M_{11}s^2 + K_{11})X_1 + (M_{12}s^2 + K_{12})X_2 = 0, \text{ and} \tag{A-2}$$
$$(M_{21}s^2 + K_{21})X_1 + (M_{22}s^2 + K_{22})X_2 = F. \tag{A-3}$$

Solving the first equation for X_2 in terms of X_1, substituting into the second equation, and rearranging yields

$$\frac{X_1}{F} = \frac{M_{12}s^2 + K_{12}}{(M_{12}s^2 + K_{12})(M_{21}s^2 + K_{21}) - (M_{11}s^2 + K_{11})(M_{22}s^2 + K_{22})} = G_{X_1} \tag{A-4}$$

Likewise,

$$\frac{X_2}{F} = \frac{M_{11}s^2 + K_{11}}{(M_{11}s^2 + K_{11})(M_{22}s^2 + K_{22}) - (M_{12}s^2 + K_{12})(M_{21}s^2 + K_{21})} = G_{X_2} \tag{A-5}$$

Using these transfer functions, we can develop a controller for either X_1 or X_2 if we are interested in only one. More generally, however, we can define a new variable Y, a linear combination of X_1 and X_2, as $Y = c_{X_1}X_1 + c_{X_2}X_2$, and thus

$$\frac{Y}{F} = c_{X_1}G_{X_1} + c_{X_2}G_{X_2} = G_Y, \tag{A-6}$$

for which the closed loop transfer function is

$$TF_{CL} = \frac{G_Y}{1 + CG_Y} \tag{A-7}$$

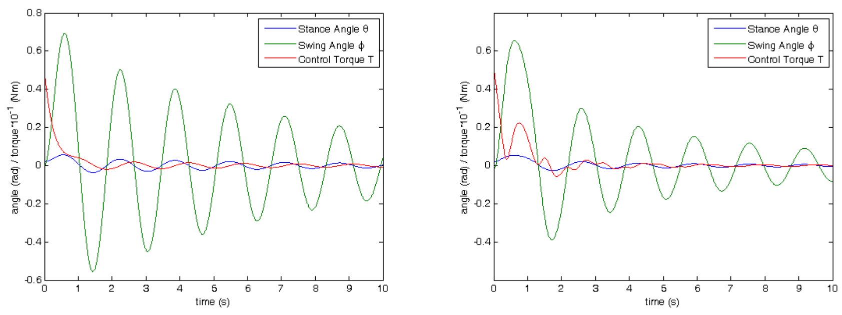
with controller transfer function C. Equation 3 is of the form of Equation A-1 with

$$\begin{array}{llll} X_1 = \theta, & M_{11} = I_{11}, & M_{22} = I_{22}, & M_{12} = M_{21} = I_{12}, \\ X_2 = \phi, & K_{11} = M_{g1}, & K_{22} = M_{g2}, & K_{12} = K_{21} = M_{g2}, \end{array}$$

and thus a controller can be designed to regulate the horizontal position of the system center of mass by defining

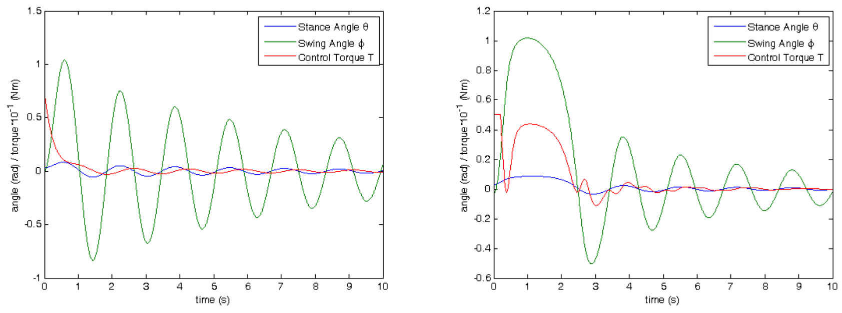
$$Y = \overbrace{\left(\frac{-m_a a - m_b l + m_b b}{m_a + m_b}\right)}^{c_{X_1}} \theta + \overbrace{\left(\frac{m_b b}{m_a + m_b}\right)}^{c_{X_2}} \phi. \tag{A-8}$$

APPENDIX B - Figures



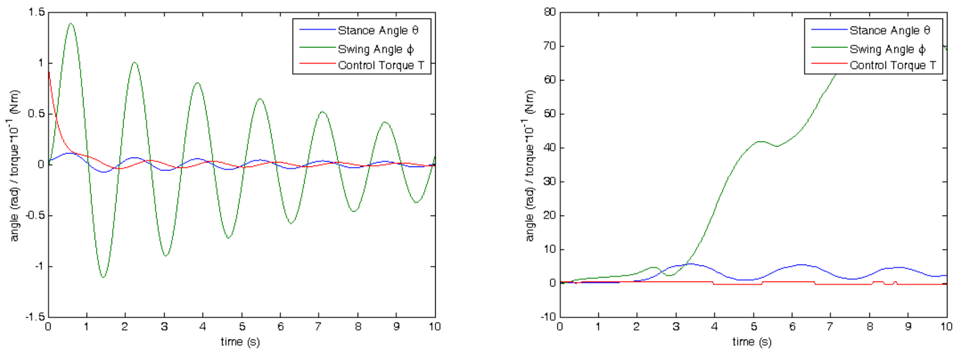
(a) Linear Simulation (b) Nonlinear Simulation

Figure B-1: Simulated response of the controlled double pendulum to initial condition $\theta(t = 0) = .02$



(a) Linear Simulation (b) Nonlinear Simulation

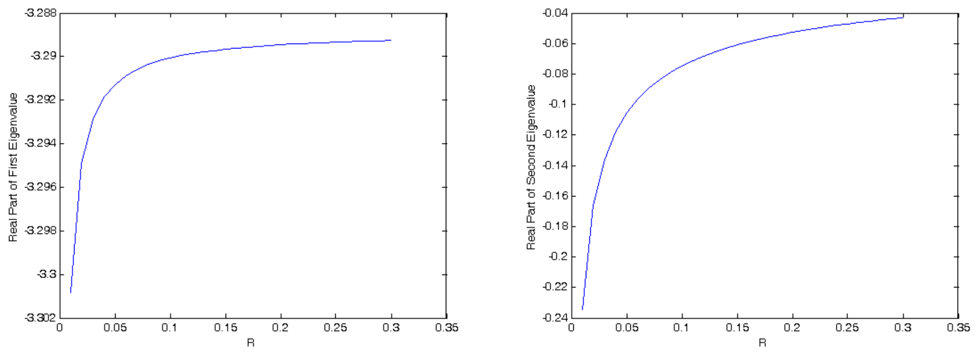
Figure B-2: Simulated response of the controlled double pendulum to initial condition $\theta(t = 0) = .03$



(a) Linear Simulation

(b) Nonlinear Simulation

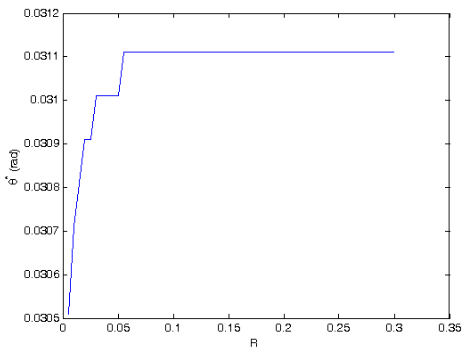
Figure B-3: Simulated response of the controlled double pendulum to initial condition $\theta(t = 0) = .03$



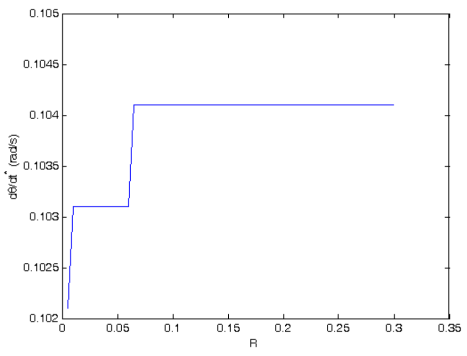
(a) The effect of R on the real part of first complex conjugate pair

(b) The effect of R on the real part of second complex conjugate pair

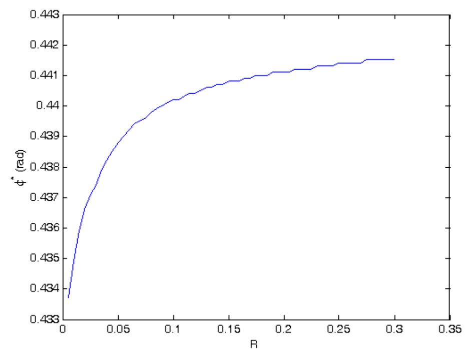
Figure B-4: The effect of control effort weight R on the eigenvalues of the linearized double pendulum equations of motion



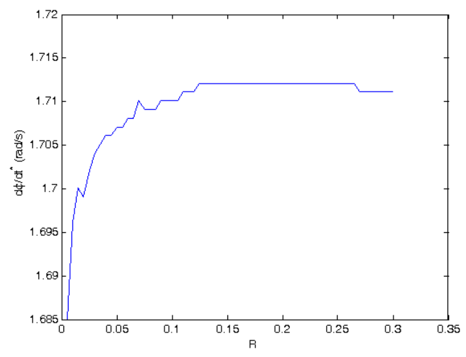
(a) The effect of R on θ^*



(b) The effect of R on $\dot{\theta}^*$

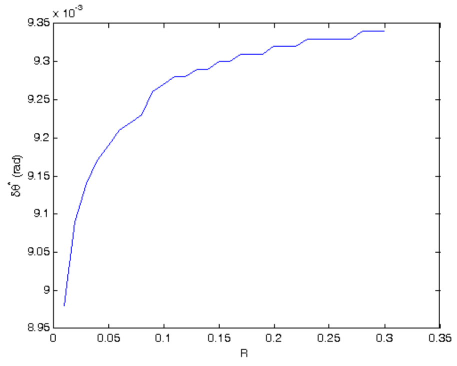


(c) The effect of R on ϕ^*

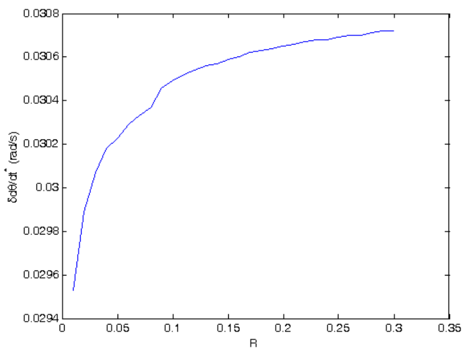


(d) The effect of R on $\dot{\phi}^*$

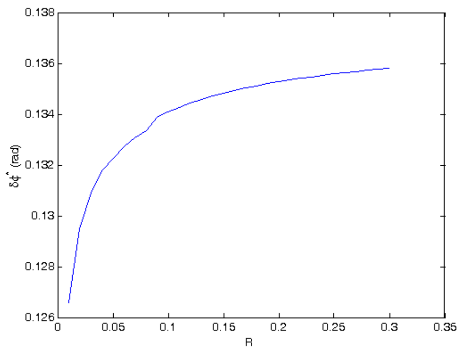
Figure B-5: The effect of control effort weight R on maximum initial states lying within the basin of attraction of the controlled double pendulum system



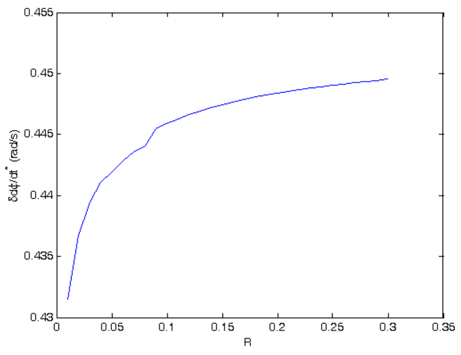
(a) The effect of R on $\delta\theta^*$



(b) The effect of R on $\delta\dot{\theta}^*$

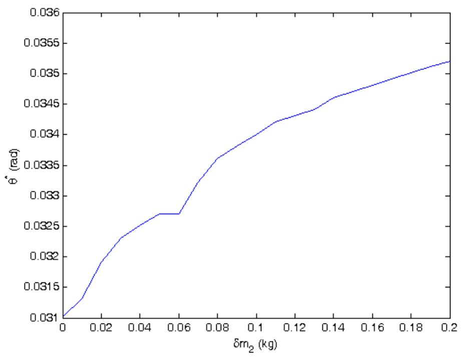


(c) The effect of R on $\delta\phi^*$

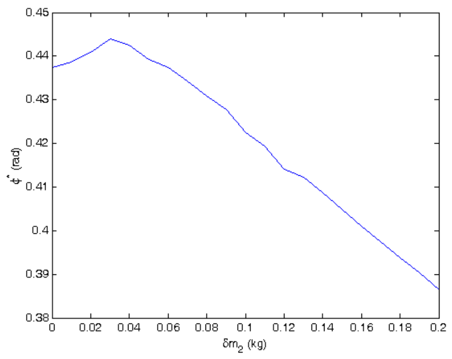


(d) The effect of R on $\delta\dot{\phi}^*$

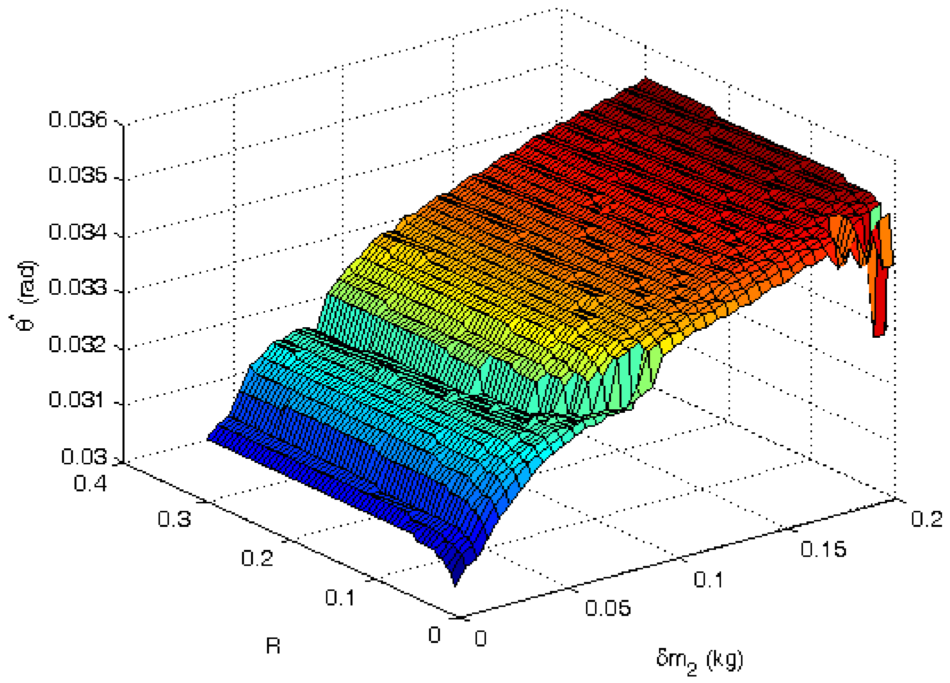
Figure B-6: The effect of control effort weight R on the maximum constant measurement offset error for which the controlled double pendulum system is stable



(a) The effect of δm_2 on θ^*, $R = .03$

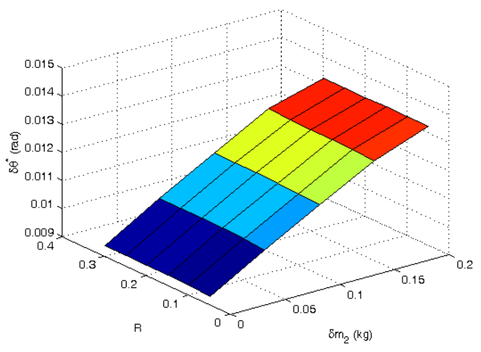


(b) The effect of δm_2 on ϕ^*, $R = .03$

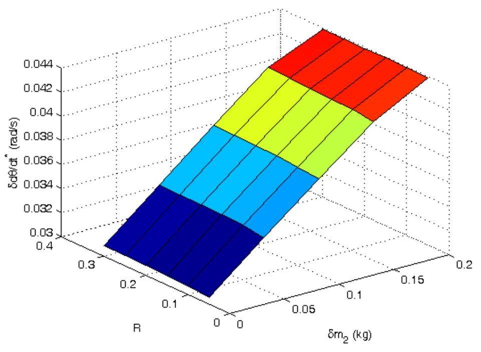


(c) The effect of δm_2 and R on θ^*

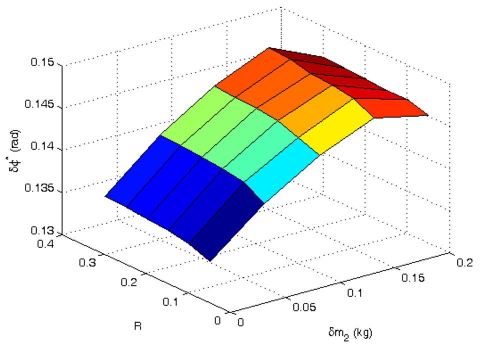
Figure B-7: The effect of control effort weight R and additional mass δm_2 at the bottom of the swing leg on maximum initial states lying within the basin of attraction of the controlled double pendulum system



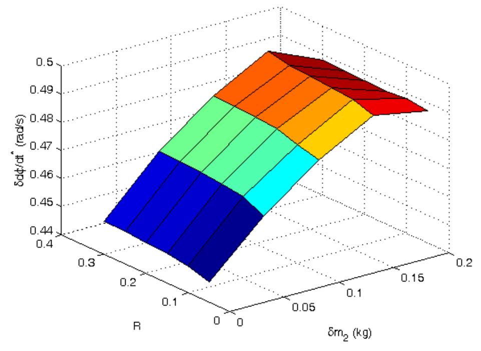
(a) The effect of δm_2 on $\delta\theta^*$



(b) The effect of δm_2 on $\delta\dot{\theta}^*$

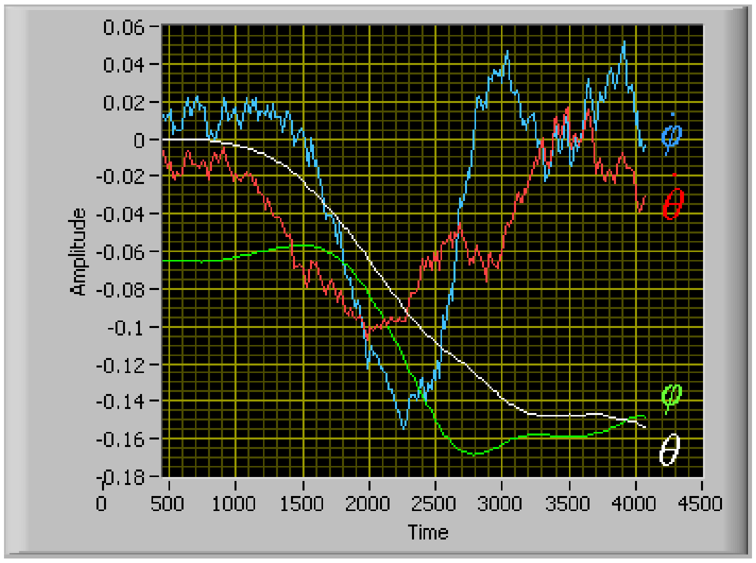


(c) The effect of δm_2 on $\delta\phi^*$

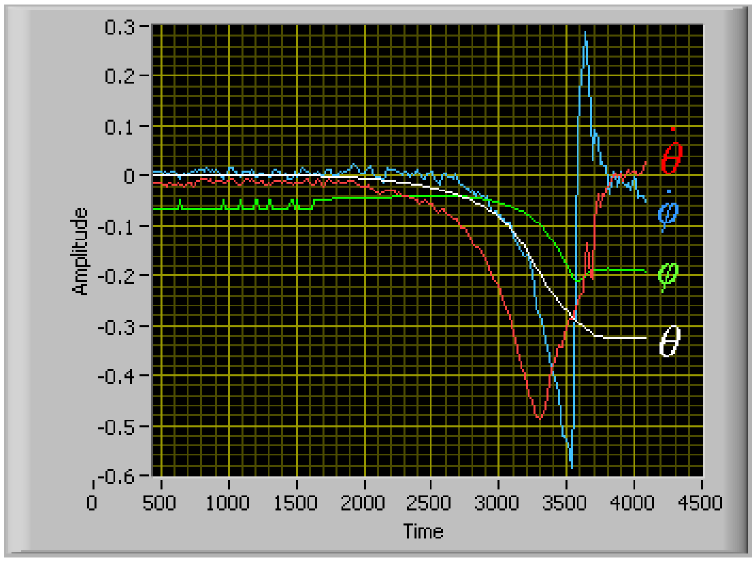


(d) The effect of δm_2 on $\delta\dot{\phi}^*$

Figure B-8: The effect of additional mass δm_2 at the bottom of the swing leg on the maximum constant measurement offset error for which the controlled double pendulum system is stable

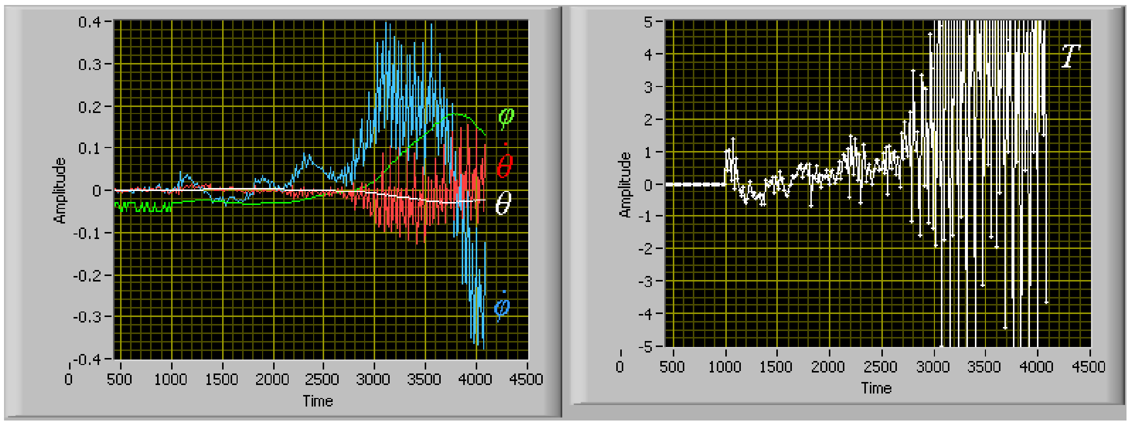


(a) ...while supported by leash.

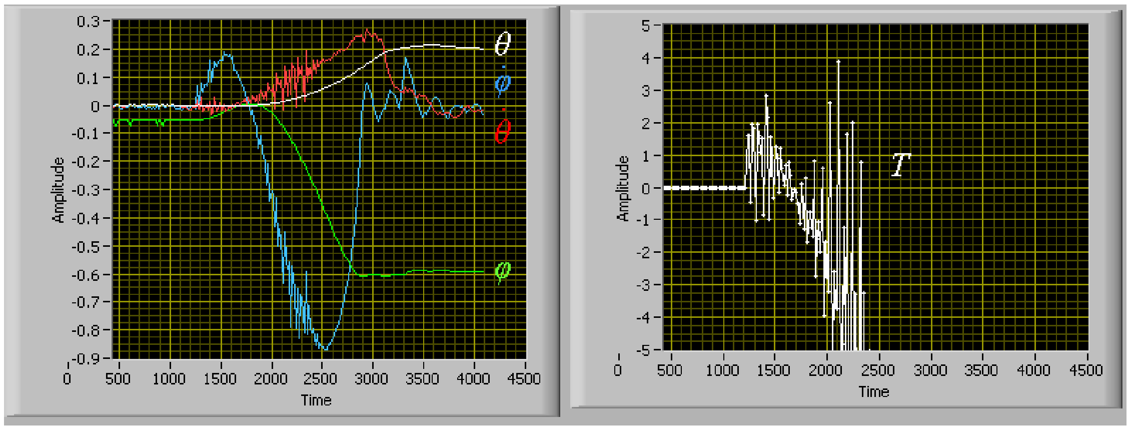


(b) ...upon complete release.

Figure B-9: Response of the uncontrolled robot...

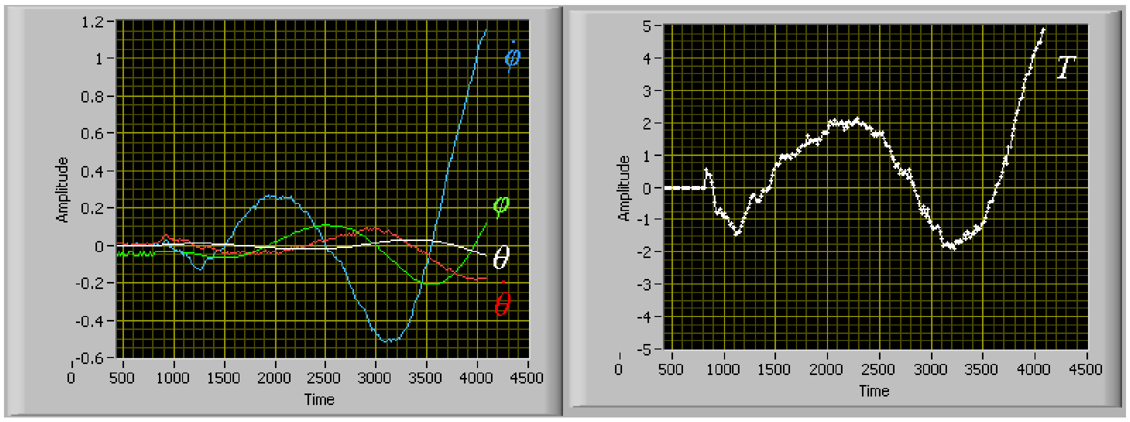


(a) ...while supported by leash.

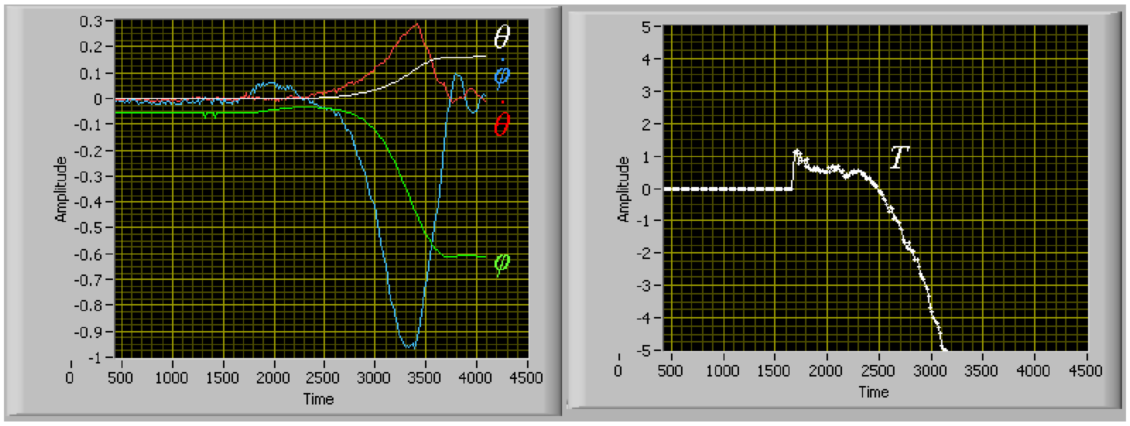


(b) ...upon complete release.

Figure B-10: Response of the ‘LQR’ controlled robot...

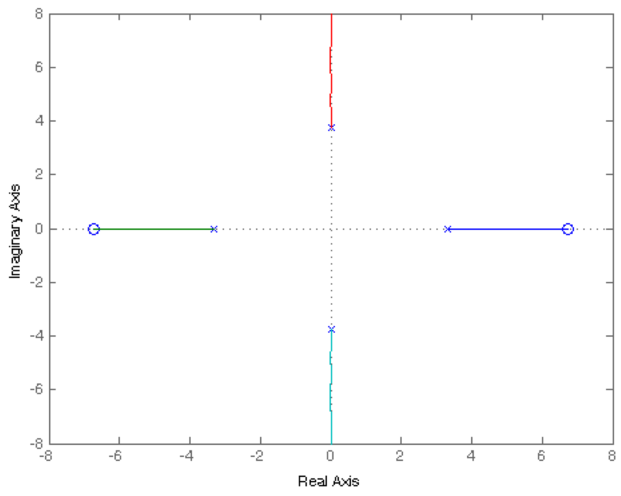


(a) ...while supported by leash.

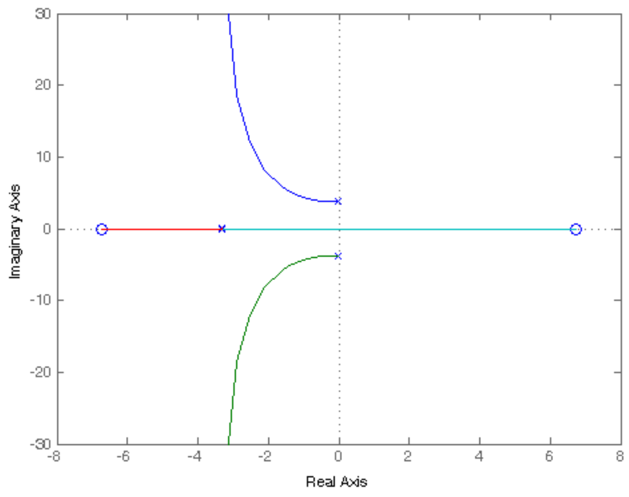


(b) ...upon complete release.

Figure B-11: Response of the controlled robot with manually lowered $\dot{\theta}$ damping...

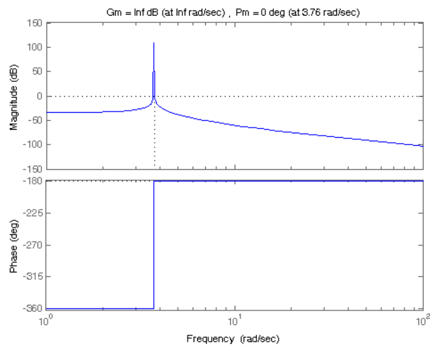


(a) Uncontrolled System

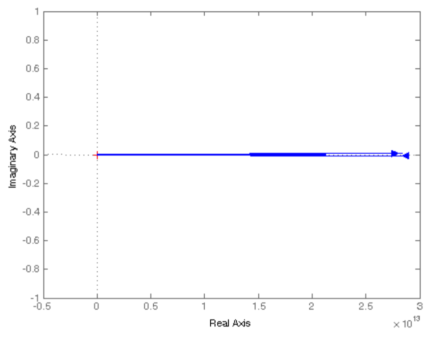


(b) Controlled System

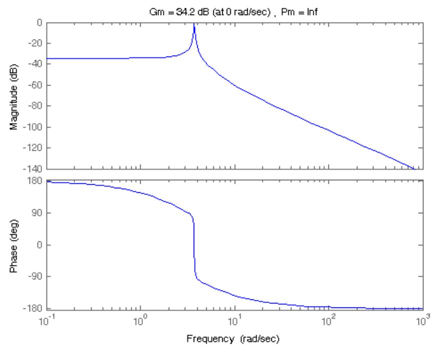
Figure B-12: Root locus plots of the uncontrolled and controlled system (unreferenced)



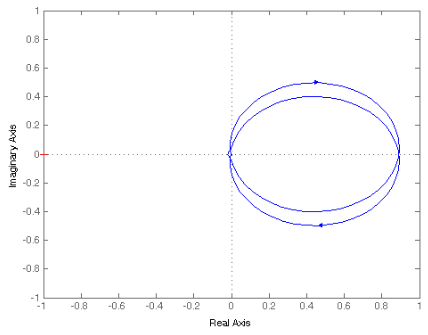
(a) Bode plot of the uncontrolled system



(b) Nyquist plot of the uncontrolled system



(c) Bode plot of the controlled system



(d) Nyquist plot of the controlled system

Figure B-13: Classical relative stability analysis of the uncontrolled and controlled system (unreferenced)

APPENDIX C - Measurement on the Cornell Ranger

Controllers for the Cornell Ranger require feedback from onboard sensors. Sensors, unfortunately, are not perfect:

- Sensors do not always measure the particular information that is ultimately desired
- Sensor output is not ideal, rather it is affected by noise, offset, nonlinearities, etc...

The challenge of state estimation is to obtain a good approximation of desired physical information from available sensors. This appendix documents my efforts toward state estimation of the Cornell Ranger.

Scope This document will briefly describe each of the sensors used by the robot, although it is not meant to substitute for the wealth of information available online and in each product's user manual. It will define each physical variable of interest to the robot controller, describe the measurement variables associated with each of these physical variables; it will not necessarily identify the shortcomings of each measurement variable explicity. It will explain how each measurement variable is determined in the robot code at an intermediate level; it will not discuss how each is used, nor will it delve into low level details such as quadrature signal processing, accelerometer filtering, or gyro avaraging. Finally, it will document selected measurement routines from the robot code, but will not cover measurement routines written by others.

C-1 Sensors on the Cornell Ranger

C-1.1 Sensor Types

Four types of sensors are used for state estimation on the Cornell Ranger.

C-1.1.1 Accelerometer

Each accelerometer used on the Cornell Ranger senses the deflection of a micro-scale cantilevered beam. The output is a voltage corresponding to translational acceleration of the sensor in a particular 'body-fixed' direction. The output signal is affected by noise and offset, but is assumed to be linearly proportional to the acceleration. The analog voltage is converted to digital form by an analog to digital converter on the robot controller. The signal is digitally filtered to reduce noise, introducing a phase lag. During a calibration routine, the average output voltage is measured for period of time during which the accelerometer is stationary in a known orientation relative to the gravity vector. To compensate for offset, a constant is added to the signal such that the average corresponds with the known acceleration due to gravity. The signal is multiplied by a nominal gain to produce a floating point number representing translational acceleration in m/s^2.

C-1.1.2 Gyro

Each 'gyro' used on the Cornell Ranger senses deflection of a vibrating, micro-scale beam due to Coriolis forces. The output is a voltage corresponding to angular velocity of the gyro about a particular 'body-fixed' axis. The output signal is affected by noise and offset, but is assumed to be linearly proportional to the angular rate. The analog voltage is converted to digital form by an analog to digital converter on the robot controller. The signal is digitally filtered to reduce noise, introducing a phase lag. During a calibration routine, the average signal is measured for period of time during which the gyro is not rotating in an inertial frame. To compensate for offset, a constant is added to the signal such that the average corresponds with the known angular rate, zero. The signal is multiplied by a gain to produce a floating point number representing angular velocity in rad/s.

C-1.1.3 Encoder

Each encoder used on on the Cornell Ranger senses passage of light beams through a slotted wheel. The output is a quadrature signal; ‘quadrature signal processing’ will not be explained in this document. After processing, a timer in the microcontroller counts incremental changes of motor shaft angular orientation in either a clockwise or counter-clockwise direction. The signed, net number of ‘ticks’ is counted and multiplied by the known angle per tick to determine a total angle relative to the reference at which the counter was initialized. Angle measurement made by the encoder are not affected by noise, but their resolution is limited to the angular increment of each tick. Time between ticks can be measured to determine the angular velocity, but these angular velocity measurement *are* affected by noise.

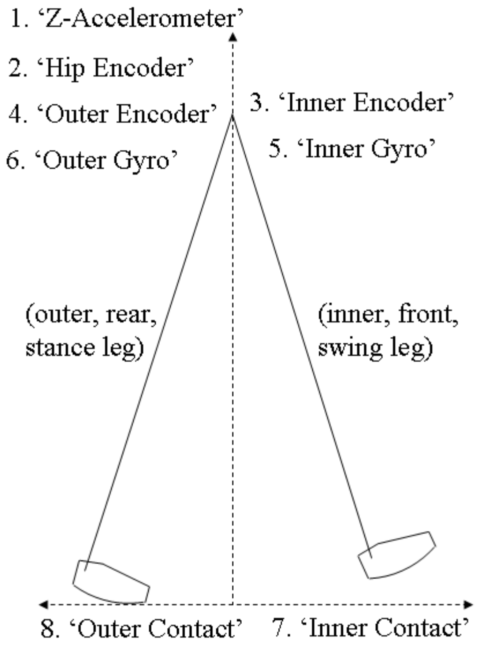
C-1.1.4 Contact Sensor

Each contact sensor used on the Cornell Ranger is actually a pair of infrared emitter-detectors (one emitter-detector on each foot per pair of legs) that sense passage of light through a gap, which changes size as the bottom of the foot deflects in response to applied pressure. The details of the signal processing will not be explained here. Each contact sensor reliably determines whether significant pressure is applied to the foot.

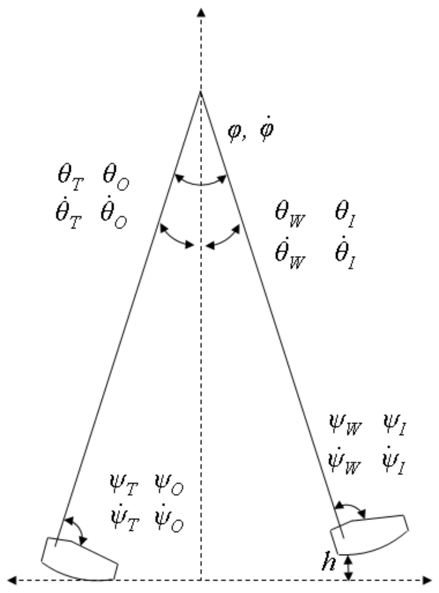
C-1.2 Available Sensors

The sensors currently available on the Cornell Ranger are presented in Figure 1(a). Note that in order to reinforce the important distinction between the quantity measured by each sensor and the physical variable the sensor is used to estimate, the text corresponds with the approximate location of the sensor and *not* necessarily the relevant location of the physical variable it is used to estimate. Also note that the approximation that the robot is two-dimensional is usually very accurate. Unless otherwise noted, it is deemed acceptable to consider the sensor to ‘live’ in a two-dimensional plane. In general, then, this document will refer to the robot as a two dimensional object, referring to pairs of legs as a ‘leg’, for example, unless greater detail is necessary.

1. The ‘Z-Accelerometer’ is mounted on the robot controller and measures translational acceleration of the outer legs in a direction approximately parallel to the axis of the outer leg tubes.
2. The ‘Hip Encoder’ is mounted on the hip motor and detects incremental changes in the angle of the hip motor output shaft about its axis.
3. The ‘Inner Encoder’ is mounted on the inner motor and detects incremental changes in the angle of the inner motor output shaft about its axis.
4. The ‘Outer Encoder’ is mounted on the outer motor and detects incremental changes in the angle of the outer motor output shaft about its axis.
5. The ‘Inner Gyro’ is mounted near the battery on the inner legs and outputs a voltage corresponding to the angular velocity of the inner leg about the axis of the hip.
6. The ‘Outer Gyro’ is mounted on the robot controller and outputs a voltage corresponding to the angular velocity of the outer leg about the axis of the hip.
7. The ‘Inner Contact’ sensor is mounted on the inner foot and detects pressure on the heel of the inner foot
8. The ‘Outer Contact’ sensor is mounted on the outer foot and detects pressure on the heel of the outer foot



(a) Available sensors listed in their approximate location


$\varphi,\ \dot{\varphi}$
θ_T θ_O
$\dot{\theta}_T$ $\dot{\theta}_O$
θ_W θ_I
$\dot{\theta}_W$ $\dot{\theta}_I$
ψ_W ψ_I
$\dot{\psi}_W$ $\dot{\psi}_I$
ψ_T ψ_O
$\dot{\psi}_T$ $\dot{\psi}_O$
h

(b) Desired physical variables near their relevant location

Figure C-1: Available Sensors and Desired Variables of the Cornell Ranger

C-2 Desired Variables

Desired variables must be subdivided into physical variables, which are the true values sought, and measurement variables, which are imperfect measurements of the physical variables.

C-2.1 Physical Variables

The physical variables currently desired for use by the Cornell Ranger are represented in Figure 1(b).[C-1] Each angle is represented by a greek letter: ϕ for the relative angle between the legs, θ for the absolute angle of a leg with respect to vertical, and ψ for the angle between the top of the foot and the leg. Each greek letter is given a capital letter subscript to make further distinctions:

- T, for 'sTance', refers to the leg corresponding with the foot that most recently experienced heelstrike. When only one foot is in contact with the ground, the leg corresponding with this foot is the stance leg. When both feet are in contact with the ground, the front leg is the stance leg.
- W, for 'sWing', refers to the leg that is not the stance leg.
- O, for 'Outer', refers to the outer pair of legs.
- I, for 'Inner', refers to the inner pair of legs.

Each physical variable is associated with one or more measurement variables in the Cornell Ranger code, listed below. Note that front and back are relative to the robot's usual (forward) direction of travel. The word 'absolute' means 'with respect to <something> in an inertial frame' and 'relative' means 'with respect to <something> in the relevant body-fixed frame'. Absolute angles must be initialized when the relevant robot part is in a known *orientation with respect to vertical* in order to be meaningul; and relative angles must be initialized when the relevant robot parts are in a known *configuration* in order to be meaningul. When the distinction between absolute and relative is not made, as for angular velocities, it is irrelevant.

C-2.1.1 Physical Variables Associated with the Hip

ϕ is assocated with

- `GBL_Data[AngleHip]`, the relative angle of the hip, measured positive from back, swing leg to front, stance leg.
- `GBL_Data[AngleEncoderHip]`, the relative angle of the hip, measured positive from back, outer leg to front, inner leg.
- `GBL_Data[AngleGyroHip]`, the relative angle of the hip, measured positive from back, outer leg to front, inner leg.
- `GBL_Data[CorrectedAngleEncoderHip]`, the relative angle of the hip, measured positive from back, outer leg to front, inner leg. A correction is made for backlash in the hip motor gearbox.

[C-1]Note that no sign conventions are indicated for the physical variables; rather, sign conventions are defined for each measurement variable.

$\dot{\phi}$ is assocated with

- `GBL_Data[AngleRateHip]`, the angular velocity of the hip, measured positive from back, swing leg to front, stance leg.
- `GBL_Data[AngleRateEncoderHip]`, the angular velocity of the hip, measured positive from back, outer leg to front, inner leg.
- `GBL_Data[AngleRateGyroHip]`, the angular velocity of the hip, measured positive from back, outer leg to front, inner leg.

C-2.1.2 Physical Variables Associated with the Legs

θ_T is assocated with

- `GBL_Data[AngleStanceLeg]`, the absolute angle of the stance leg, measured positive from vertical to back, stance leg.

$\dot{\theta}_T$ is assocated with

- `GBL_Data[AngleRateStanceLeg]`, the angular rate of the stance leg, measured positive as stance leg moves back.
- `GBL_Data[AngleRateStanceLegFiltered]`, a heavily filtered measurement of the angular rate of the stance leg. Note that the signal exhibits less noise, but greater phase lag.

θ_W is assocated with

- `GBL_Data[AngleSwingLeg]`, the absolute angle of the swing leg, measured positive from vertical to back, swing leg.

$\dot{\theta}_W$ is assocated with

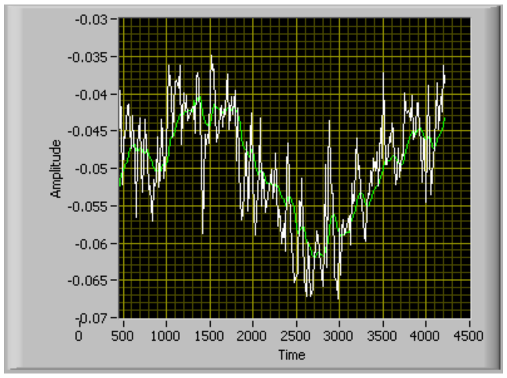
- `GBL_Data[AngleRateSwingLeg]`, the angular velocity of the swing leg, measured positive as swing leg moves back.

θ_I is assocated with

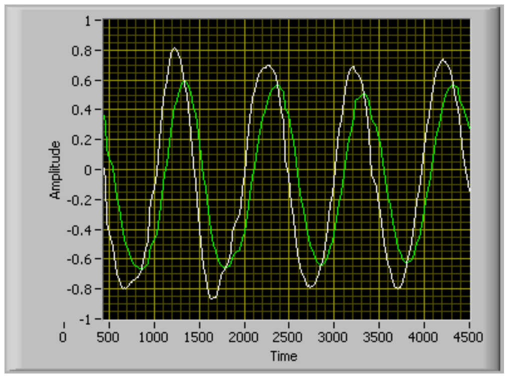
- `GBL_Data[AngleGyroInner]`, the absolute angle of the inner leg, measured positive from vertical to back, inner leg.

$\dot{\theta}_I$ is assocated with

- `GBL_Data[AngleRateGyroInner]`, the angular rate of the inner leg, measured positive as inner leg moves back.
- `GBL_Data[AngleRateGyroInnerFiltered]`, a heavily filtered measurement of the angular rate of the inner leg. Note that the filtered signal is less noisy, but has significant phase lag, as illustrated in Figure 2(a)



(a) ...at very low angular rate, to emphasize the high noise level of the regular signal



(b) ...at angular rates experienced during the walking cycle, to emphasize the attenuation and phase lag of the filtered signal

Figure C-2: Comparison of regular (white) and heavily filtered (green) inner gyro signals...

θ_O is assocated with

- `GBL_Data[AngleGyroOuter]`, the absolute angle of the outer leg, measured positive from vertical to back, outer leg.

$\dot{\theta}_O$ is assocated with

- `GBL_Data[AngleRateGyroOuter]`, the angular rate of the outer leg, measured positive as outer leg moves back.
- `GBL_Data[AngleRateGyroOuterFiltered]`, a heavily filtered measurement of the angular rate of the outer leg. Note that the signal will have less noise, but greater phase lag.

C-2.1.3 Physical Variables Associated with the Ankles

ψ_I is assocated with

- `GBL_Data[AngleEncoderInner]`, the relative angle of the inner ankle, measured positive from the inner leg to the top of the inner foot.
- `GBL_Data[AngleEncoderInnerCorrected]`, the relative angle of the inner ankle, corrected for stretching of the cable connecting the actuator to the ankle pulley.

$\dot{\psi}_I$ is assocated with

- `GBL_Data[AngleRateEncoderInner]`, the angular rate of the inner ankle, measured negative as the foot retracts.

ψ_O is assocated with

- `GBL_Data[AngleEncoderOuter]`, the relative angle of the outer ankle, measured positive from the outer leg to the top of the outer foot.
- `GBL_Data[AngleEncoderOuterCorrected]`, the angle of the outer ankle, corrected for stretching of the cable connecting the actuator to the ankle.

$\dot{\psi}_O$ is assocated with

- `GBL_Data[AngleRateEncoderOuter]`, the angular rate of the outer ankle, measured negative as the outer foot retracts.

ψ_T is assocated with

- `GBL_Data[AngleStanceFoot]`, the relative angle of the stance ankle, measured positive from the stance leg to the top of the stance foot.

$\dot{\psi}_T$ is assocated with

- `GBL_Data[AngleRateStanceFoot]`, the angular rate of the stance ankle, measured negative as the stance foot retracts.

ψ_W is assocated with

- `GBL_Data[AngleSwingFoot]`, the relative angle of the swing ankle, measured positive from the swing leg to the top of the stance foot.

$\dot{\psi}_W$ is assocated with

- `GBL_Data[AngleRateSwingFoot]`, the angular rate of the swing ankle, measured negative as the swing foot retracts.

C-2.1.4 Physical Variables Associated with the Feet

h is assocated with

- `GBL_Data[DistanceToHeelStrike]`, the minimum absolute distance from the bottom of the front, swing foot to the ground. Undefined if the swing foot is not in front.

Some measurement variables are not directly associated with any physical variables of interest, rather they are only used in the calculation of other measurement variables.

- `GBL_Hip_Backlash`, half the backlash in the gearbox (measured in units of encoder ticks), is a measurement variable associated with the hip.
- `GBL_Data[OffsetGyroInner]`, the offset voltage of the inner gyro, is a measurement variable associated with the legs.

- GBL_Data[OffsetGyroOuter], the offset voltage of the outer gyro, is a measurement variable associated with the legs.
- GBL_Data[GyroGainInner], the gain, or transuction constant, of the inner gyro, is a measurement variable associated with the legs.
- GBL_Data[GyroGainOuter], the gain, or transuction constant, of the outer gyro, is a measurement variable associated with the legs.
- GBL_Data[AccelZ], the measurement made by the Z Accelerometer, is a measurement variable associated with the legs.
- GBL_Data[AccelZAverage], a heavily filtered version of the measurement made by the Z Accelerometer, is a measurement variable associated with the legs.
- GBL_Data[AngleEncoderStiffnessInner], the stiffness of the inner leg cables, is a measurement variable associated with the ankles.
- GBL_Data[AngleEncoderStiffnessOuter], the stiffness of the outer leg cables, is a measurement variable associated with the ankles.
- GBL_Data[FootContactInner] indicates whether pressure is applied to the heel of the inner foot. It is a measurement variable associated with the feet.
- GBL_Data[FootContactOuter] indicates whether pressure is applied to the heel of the outer foot. It is a measurement variable associated with the feet.
- GBL_Data[StanceLeg] indicates whether the inner or outer leg is the stance leg. It is a measurement variable associated with the feet.

C-2.2 Measurement Variables

Most measurement variables follow a standard naming convention. The appearance of the following terms within the measurement variable name indicates the type or the origin of the measurement.

- Angle signifies that the measurement variable represents an angular measurement in radians
- AngleRate signifies that the measurement variable represents an angular velocity measurement in radians per second
- Contact signifies that the measurement variable is a binary value; a value of 1 indicates contact
- Gyro signifies that the measurement variable depends primarily on gyro data
- Encoder signifies that the measurement variable depends primarily on encoder data
- Accel signifies that the measurement variable depends primarily on accelerometer date
- Corrected signifies that the measurement variable attempts to correct an inaccuracy in the way the variable was originally measured

- `Filtered` and `Average` signify that the measurement variable has been digitally filtered more than usual to reduce noise; these measurement variables will not be given particular attention below
- `Stance`, `Swing`, `Inner`, and `Outer` carry the meanings defined in C-2.1
- `Hip` signifies that the measurement variable contains information related to a ϕ physical variable associated with the hip
- `Leg`, when it appears along with `Angle` or `AngleRate` signifies that the measurement variable contains information related to a θ physical variable associated with the leg
- `Foot`, when it appears along with `Angle`, signifies that the measurement variable contains information related to a ψ physical variable associated with the ankle. The names of these measurement variables should be renamed with `Ankle` in place of `Foot`
- `Foot`, when it appears along with `Contact`, signifies that the measurement variable contains information related to a physical variable associated with the feet

A few measurement variable names are exceptions to these rules, most notably `StanceLeg`, `DistanceTo-HeelStrike`, and those names that contain `OffsetGyro`, `GyroGain`, and `AngleEncoderStiffness`.

C-2.2.1 Measurement Variables Associated with the Hip

`GBL_Data[AngleHip]` is initialized in the `IntRoutine_TMRC0()` function within the `MODECALGYROS` case as `FFsub(GBL_Data[AngleSwingLeg], GBL_Data[AngleStanceLeg])`. Note that the calibration routine assumes that the inner, stance leg is in front during calibration. `GBL_Data[AngleHip]` is recalculated in the `StateEstimator()` function . According to its sign convention, if the inner leg is the stance leg, then it is equal to `GBL_Data[AngleEncoderHip]`; if the outer leg is the stance leg, then it is equal to the negative of `GBL_Data[AngleEncoderHip]`. `GBL_Data[AngleHip]` can be set from `GBL_Data[AngleGyroHip]` instead of `GBL_Data[AngleEncoderHip]` according to the same rules, if desired.

`GBL_Data[AngleEncoderHip]` is set in the `ReadAngleEncoderHip()` function as `FFmult(0xFFF-440FD,S16int2FFloat(*TMRA0_CNTR))`, or the product of gain `0xFFF440FD`, which is the conversion from encoder tick to rad/s in `FFloat` format, and `*TMRA0_CNTR`, which is the signed, net number of encoder ticks since `*TMRA0_CNTR` was initialized. Initialization of `*TMRA0_CNTR` is discussed in C-3.1.

`GBL_Data[AngleGyroHip]` is set in the `StateEstimator()` function as `FFsub(GBL_Data[Angle-GyroOuter], GBL_Data[AngleGyroInner])`. It agrees closely with `GBL_Data[AngleEncoderHip]` when the gyro angles are properly initialized, as in Figure 4(a).

`GBL_Data[CorrectedAngleEncoderHip]` is is set in the `ReadAngleEncoderHip()` function. If the PWM value at the hip is zero, then the motor output shaft can be at either side of the gearbox backlash. Thus it is impossible to correct for backlash, and `GBL_Data[CorrectedAngleEncoderHip]` is equal to `GBL_Data[AngleEncoderHip]`. If the PWM value at the hip is nonzero, then the motor shaft is known to be at one side of the backlash or the other, and thus `GBL_Hip_Backlash` can be added or subtracted from `*TMRA0_CNTR` before converting from encoder ticks to radians. Figure C-3 illustrates the difference between `GBL_Data[AngleEncoderHip]` and `GBL_Data[CorrectedAngleEncoderHip]`.

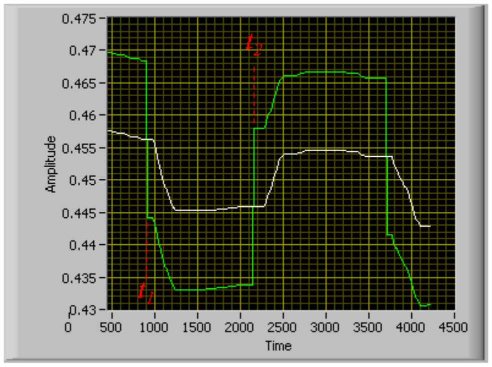


Figure C-3: Comparison of `GBL_Data[AngleEncoderHip]` (white) and `GBL_Data[CorrectedAngle-EncoderHip]` (green) while the controller attempts to maintain a prescribed hip angle against a disturbance. When the disturbance causes the true hip angle to decrease at time t_1, the controller recognizes that backlash has been eliminated in a particular direction, and `GBL_Data[CorrectedAngleEncoderHip]` responds appropriately. When the disturbance causes the true hip angle to increase at time t_2, the controller recognizes that the backlash has been eliminated in the other direction, and `GBL_Data[CorrectedAngleEncoderHip]` responds appropriately. The discontinuity is essentially unavoidable, as true position of the leg is uncertain when there is backlash in either direction. When backlash has been eliminated on one side, `GBL_Data[AngleEncoderHip]` is always incorrect by half the backlash angle of the gearbox.

`GBL_Data[AngleRateHip]` is set in the `StateEstimator()` function as `FFsub(GBL_Data[Angle-RateSwingLeg], GBL_Data[AngleRateStanceLeg])`.

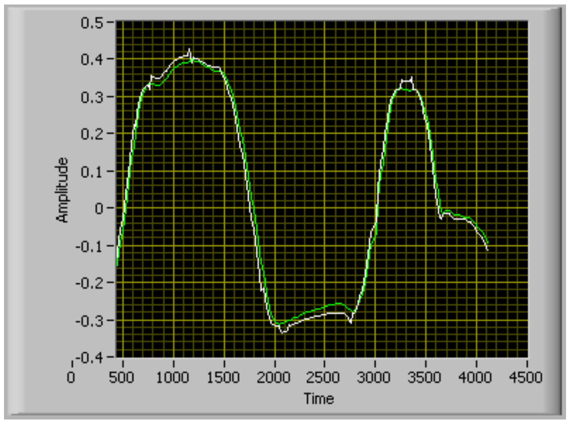
`GBL_Data[AngleRateEncoderHip]` is set in the `ReadAngleRateEncoderHip()` function. It is essentially obsolete, replaced by the difference in the gyro angular velocity signals, because it was very noisy, as illustrated in Figure 4(b).

`GBL_Data[AngleRateGyroHip]` is set in the `ReadGyroRate()` function as `FFsub(GBL_Data[Angle-RateGyroOuter], GBL_Data[AngleRateGyroInner])`. It is a cleaner, drop-in replacement of `GBL_Da-ta[AngleRateEncoderHip]`, as illustrated in Figure 4(b).

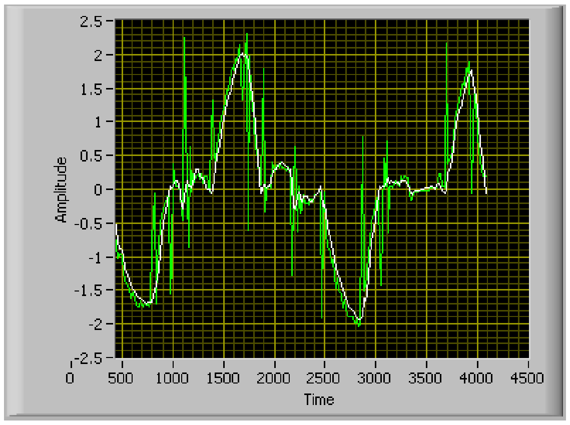
`GBL_Hip_Backlash` is discussed in C-3.1.

C-2.2.2 Measurement Variables Associated with the Legs

`GBL_Data[AngleStanceLeg]` is initialized in the `IntRoutine_TMRC0()` function within the `MODE-CALGYROS` case as the negative of one half of `GBL_Data[CorrectedAngleEncoderHip]`. (Assuming that the inner, stance leg is in front during gyro calibration and the feet are fully retracted, the robot legs form an isosceles triangle with the ground, so this is a valid approximation.) It is re-initialized upon heelstrike in the `StateEstimator()` function as the difference between `GBL_Data[AngleSwingLeg]` and the absolute value of `GBL_Data[CorrectedAngleEncoderHip]`. Once initialized, changes are integrated from `GBL_Data[AngleRateStanceLeg]` in the `StateEstimator()` function.



(a) Actual data from the robot showing good agreement between GBL_Data[AngleEncoderHip] (white) and GBL_Data[AngleRateGyroHip] (green). The agreement between the integrated gyro measurement and the relative encoder measurement is typical, provided that the reference gyro angles are properly set at heelstrike.



(b) Actual data from the robot showing good agreement between GBL_Data[AngleRateEncoderHip] (green) and GBL_Data[AngleRateGyroHip] (white), but that the gyro measurement does not suffer from the wild noise problems of the encoder measurement.

Figure C-4: Graphical comparison of encoder and gyro measurements

GBL_Data[AngleRateStanceLeg] is set in the StateEstimator() function as GBL_Data[AngleRateGyroOuter] if the outer leg is the stance leg, or as GBL_Data[AngleRateGyroInner] if the inner leg is the stance leg.

GBL_Data[AngleSwingLeg] is initialized in the IntRoutine_TMRC0() function within the MODECALGYROS case as one half of GBL_Data[CorrectedAngleEncoderHip]. (Assuming that the inner, stance leg is in front during gyro calibration and the feet are fully retracted, the robot legs form an isosceles triangle with the ground, so this is a valid approximation.) It is re-initialized upon heelstrike in the StateEstimator() function according to a relatively long trigonometric formula, which is explained in C-3.2. Once initialized, changes are integrated from GBL_Data[AngleRateSwingLeg] in the StateEstimator() function.

GBL_Data[AngleRateSwingLeg] is set in the StateEstimator() function as GBL_Data[AngleRateGyroOuter] if the inner leg is the stance leg, or as GBL_Data[AngleRateGyroInner] if the outer leg is the stance leg.

GBL_Data[AngleGyroInner] is initialized in the IntRoutine_TMRC0() function within the MODECALGYROS case equal to GBL_Data[AngleStanceLeg]. It is re-initialized in the StateEstimator() function equal to GBL_Data[AngleStanceLeg] when the inner foot experiences heelstrike. Otherwise, including the case when the outer foot experiences heelstrike, changes are integrated from GBL_Data[AngleRateGyroInner] in the StateEstimator(). Therefore *its value may differ from* GBL_Data[AngleSwingLeg] *even when the inner* *leg is the swing leg*; this is illustrated in Figure C-12 and important in automatic gyro calibration described in C-3.3. Integration drift is apparent in Figure 5(b).

GBL_Data[AngleRateGyroInner] is set in the `ReadGyroRate()` function. It is calculated by subtracting `GBL_Data[OffsetGyroInner]` from the filtered signal from inner gyro, then multiplying the difference by the inner gyro transduction constant `GBL_Data[GyroGainInner]`. Measurement noise is visible in Figure 5(a).

GBL_Data[OffsetGyroInner] and GBL_Data[GyroGainInner] are documented in C-3.3.

GBL_Data[AngleGyroOuter] is initialized in the `IntRoutine_TMRC0()` function within the `MODECAL-GYROS` case equal to `GBL_Data[AngleSwingLeg]`. It is re-initialized in the `StateEstimator()` function equal to `GBL_Data[AngleStanceLeg]` when the outer foot experiences heelstrike. Otherwise, including the case when the inner foot experiences heelstrike, changes are integrated from `GBL_Data[AngleRateGyroOuter]` in the `StateEstimator()` function. Therefore *its value may differ from* `GBL_Data[AngleSwingLeg]` *even when the outer leg is the swing leg*; this is illustrated in Figure C-12 and important in automatic gyro calibration described in C-3.3. Integration drift is apparent in Figure 6(b).

GBL_Data[AngleRateGyroOuter] is set in the `ReadGyroRate()` function. It is calculated by subtracting `GBL_Data[OffsetGyroOuter]` from the filtered signal from outer gyro, then multiplying the difference by the outer gyro transduction constant `GBL_Data[GyroGainOuter]`. Measurement noise is visible in Figure 6(a).

GBL_Data[OffsetGyroOuter] and GBL_Data[GyroGainOuter] are documented in C-3.3.

GBL_Data[AccelZ] is calculated in the `ReadAccelZ()` function by multiplying the average of the gyro signal provided by `*ADCB_ADRSLT7` by the transduction constant. `GBL_Data[AccelZAverage]` is not yet defined. Further detail is not provided in this ducument.

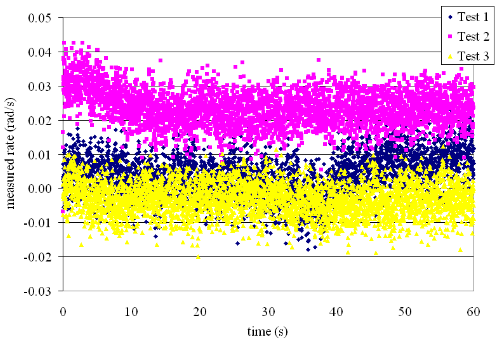
C-2.2.3 Measurement Variables Associated with the Ankles

GBL_Data[AngleEncoderInner] is calculated in the `ReadAngleEncoderInner()` function as the product of `*TMRA2_CNTR + 25000` and the conversion from tick to radians in `FFloat` format. `*TMRA2_CNTR` is the timer that counts the signed, net number of inner encoder tick since it was initialized to a value of `-25000`. Further detail regarding `*TMRA2_CNTR` initialization is provided in C-3.1.

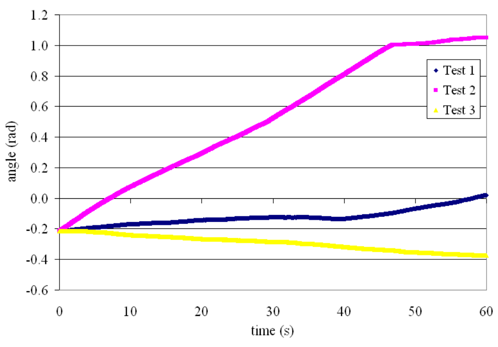
GBL_Data[CorrectedAngleEncoderInner] is calculated in the `ReadAngleEncoderInner()` function by subtracting the product of `GBL_Data[AngleEncoderStiffnessInner]` and the inner motor torque (Nm) from `GBL_Data[AngleEncoderInner]`. The effect of this correction is illustrated in Figure C-7.

GBL_Data[AngleEncoderStiffnessInner] is documented in C-3.1.

GBL_Data[AngleRateEncoderInner] is calculated in the `ReadAngleRateEncoderInner()` function by dividing the number of radians per inner motor encoder tick by the average time between ticks. No further detail is provided in this document.

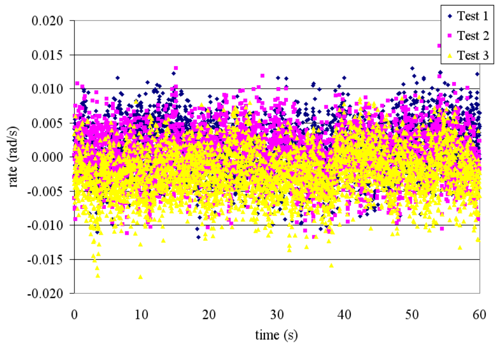


(a) GBL_Data[AngleRateGyroInner] data suffers from noise and offset. Offset differs from trial to trial.

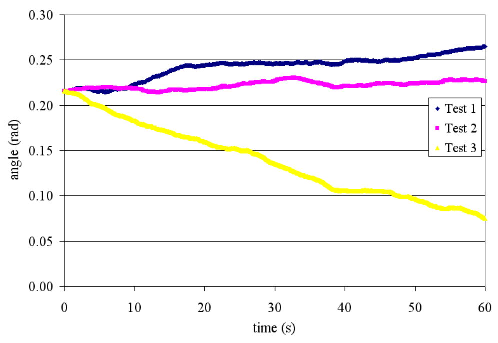


(b) GBL_Data[AngleGyroInner] data suffers from integration drift. Note extreme drift in Trial 2 corresponds with apparent offset in angular rate data.

Figure C-5: Sample inner leg angle data from three trials to illustrate drift in absolute angle measurement and noise in angular velocity measurement immediately after calibration.



(a) GBL_Data[AngleRateGyroOuter] data suffers from noise and offset. Offset differs from trial to trial.



(b) GBL_Data[AngleGyroOuter] data suffers from integration drift. Note extreme drift in Trial 3 corresponds with apparent offset in angular rate data.

Figure C-6: Sample outer leg angle data from three trials to illustrate drift in absolute angle measurement and noise in angular velocity measurement immediately after calibration.

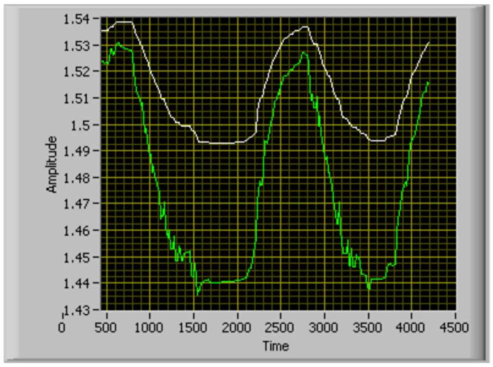


Figure C-7: Comparison of `GBL_Data[AngleEncoderInner]`(white) and `GBL_Data[CorrectedAngleEncoderInner]`(green) while the controller attempts to maintain a prescribed inner ankle angle against a disturbance. `GBL_Data[CorrectedAngleEncoderInner]` is a closer approximation to the actual rotation of the foot because it accounts for the fact that the cable linking the motor to the ankle joint stretches as torque is applied. `GBL_Data[AngleEncoderInner]` does not account for the stretching, and thus underestimates the actual rotation of the foot.

`GBL_Data[AngleEncoderOuter]` is calculated in the `ReadAngleEncoderOuter()` function as the product of `*TMRB0_CNTR + 25000` and the conversion from tick to radians in `FFloat` format. `*TMRB0_CNTR` is the timer that counts the signed, net number of outer encoder tick since it was initialized to a value of `-25000`. Further detail regarding `*TMRB0_CNTR` initialization is provided in C-3.1.

`GBL_Data[CorrectedAngleEncoderOuter]` is calculated in the `ReadAngleEncoderOuter()` function by subtracting the product of the outer cable stiffness (rad/Nm) and the outer motor torque (Nm) from `GBL_Data[AngleEncoderInner]`.

`GBL_Data[AngleEncoderStiffnessOuter]` is documented in C-3.1.

`GBL_Data[AngleRateEncoderOuter]` is calculated in the `ReadAngleRateEncoderInner()` function by dividing the number of radians per outer motor encoder tick by the average time between ticks. No further detail is provided in this document.

`GBL_Data[AngleStanceFoot]` is set in the `StateEstimator()` function equal to `GBL_Data[AngleEncoderOuter]` if the outer leg is the stance leg, or equal to `GBL_Data[AngleEncoderInner]` if the inner leg is the stance leg.

`GBL_Data[AngleRateStanceFoot]` is set in the `StateEstimator()` function equal to `GBL_Data[AngleRateEncoderOuter]` if the outer leg is the stance leg , or equal to `GBL_Data[AngleRateEncoderInner]` if the inner leg is the stance leg.

GBL_Data[AngleSwingFoot] is set in the `StateEstimator()` function equal to `GBL_Data[AngleEncoderInner]` if the outer leg is the stance leg, or equal to `GBL_Data[AngleEncoderOuter]` if the inner leg is the stance leg.

GBL_Data[AngleRateSwingFoot] is set in the `StateEstimator()` function equal to `GBL_Data[AngleRateEncoderInner]` if the outer leg is the stance leg, or equal to `GBL_Data[AngleRateEncoderOuter]` if the inner leg is the stance leg.

C-2.2.4 Measurement Variables Associated with the Feet

GBL_Data[DistanceToHeelStrike] is set in the `StateEstimator()` function equal to the difference between the cosine of `GBL_Data[AngleStanceLeg]` and the cosine of `GBL_Data[AngleSwingLeg]`. This is a valid approximation of the minimum distance (m) between the foot and the ground just before heelstrike because both legs are of unit length, hence no scaling is necessary, and both feet are in approximately the same orientation, so the ankle angles do not need to be accounted for. There is signficant offset from the true value, but the measurement is *consistent* to within $\sim$ 5mm at heelstrike, as illustrated in C-8.

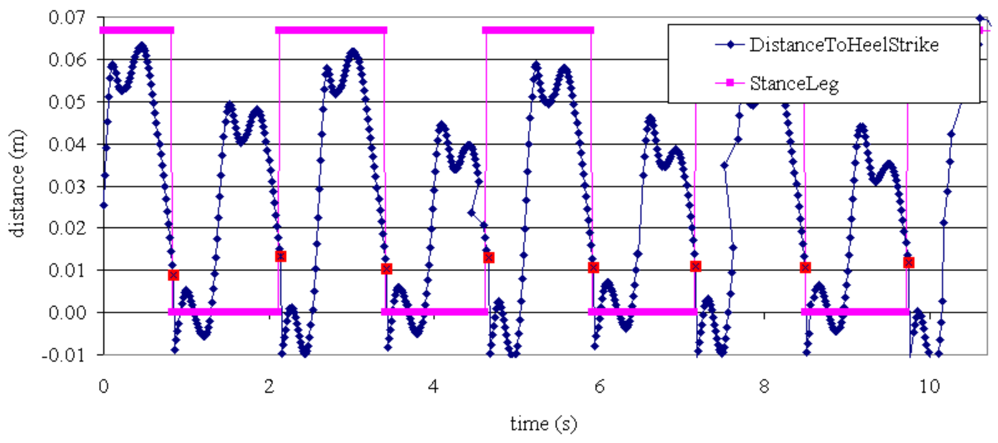


Figure C-8: Illustration of the consistency of `GBL_Data[DistanceToHeelStrike]` measurement variable; `GBL_Data[StanceLeg]` is shown to indicate heelstrike. The points at which the lines intersect (red markers) indicate the value of `GBL_Data[DistanceToHeelStrike]` at heelstrike. Although the measurement is offset from the true value of zero, it is consistent to within $\sim$ 5 5mm and thus can still be used to predict heelstrike.

GBL_Data[FootContactInner] and GBL_Data[FootContactOuter] are set in `CheckFootStatus()` according to time tested, reliable logic. No further detail is provided in this document.

GBL_Data[StanceLeg] is set in the `StateEstimator()` function according to the following logic: If the front foot is in contact with the ground and the front leg is not currently the stance leg, then the front leg is the new stance leg.

C-3 Measurement Routines

The following subsections document selected measurement routines in the Cornell Ranger code.

C-3.1 Encoder Calibration

The encoders counters can only track angles relative to a reference value - they must be initialized to a value corresponding with a known robot configuration before they can provide meaningful data. Furthermore, each encoder measures angular rotation of a *motor output shaft*. The hip motor encoder does *not* directly measure the hip angle due to backlash in the hip motor gearhead, and the angle motor encoders do not directly measure measure the ankle anges due to both backlash in the gearheads and flexibility in the cables. The Encoder Calibration routine, contained in the `IntRoutine_TMRC0()` function within the `MODECALGYROS` case, initializes reference values for encoders, measures hip gearbox backlash, and determines cable stiffness so that the final measurement variables can meaningfully describe the desired physical variables. While this code runs, the robot must be configured with all relative angles in the 'zero' position: the inner and outer legs must be parallel and the feet must be retracted to their limits. The legs and feet must be secured in these positions, as torques are be applied during calibration.

Here, the heavily commented code is broken into blocks; further explanation is included below each block as deemed necessary.

```
case MODECALENCODERS:
  {
  ReadSensors();
  switch(BacklashMeasured)
```

The variable `BacklashMeasured`, initialized to 0, is used to determine whether the calibration routine is just beginning (0), or if it is in progress (1). The name `BacklashMeasured` is a misleading artifact, as it does not at all indicate whether the backlash has been measured.

```
    {
    case 0: \\ the calibration routine is just beginning
      {
      EncoderZero(); //Initialize all encoder counters, that is:
                     //*TMRA0_CNTR = 0; // Hip encoder
                     //*TMRA2_CNTR = -25000; //Inner encoder
                     //*TMRB0_CNTR = -25000; //Outer encoder
                     //*TMRB2_CNTR = 0; //Spare encoder
```

The `EncoderZero()` routine initializes all the encoder counters to tentative values. Note that `*TMRA0_CNTR` is re-initialized later in the routine.

```
      read_encoders_time_1 = GBL_Elapsed_mS+500;
      read_encoders_time_2 = read_encoders_time_1+500;
      TestHipEncoderTime1 = read_encoders_time_2+500;
      TestHipEncoderTime2 = TestHipEncoderTime1+500;
```

`GBL_Elapsed_mS` serves as the robot's clock; it is an in integer indicating the number of milliseconds that have elapsed since the robot was turned on. There are four phases to the calibration mode, two reading phases and two test phases. Each phase is 500 milliseconds long. The time when each phase will end, relative to the current clock time, is calculated in this block.

```
    BacklashMeasured = 1; \\ now the routine is in progress
    break;
    }
  case 1:
    {
    // reading phase 1
    if (GBL_Elapsed_mS < read_encoders_time_1)
      {
      // send a small PWM signal to each motor
      GBL_Data[PWMDesiredHip] = S16int2FFloat(75);
      GBL_Data[TorqueInner] = IEEE2FFloat(.05);
      GBL_Data[TorqueOuter] = IEEE2FFloat(.05);
      }
```

During reading phase 1, a small PWM signal is sent to the hip motor to eliminate gearbox backlash in one direction. Likewise, a small torque is applied by the inner and outer motors. By the end of the 500 millisecond phase, the motors have stalled and all gears are meshed in one direction.

```
    // reading phase 1 almost over
    else if (GBL_Elapsed_mS == read_encoders_time_1)
      {
      // continue to send a small PWM signal to each motor
      GBL_Data[PWMDesiredHip] = S16int2FFloat(75);
      GBL_Data[TorqueInner] = IEEE2FFloat(.05);
      GBL_Data[TorqueOuter] = IEEE2FFloat(.05);
```

"The first measurement is about to be taken, but keep applying the small torque in the same direction to keep the gears meshed…"

```
      // measure the encoder values
      GBL_Hip_Backlash =  *TMRA0_CNTR;
      Angle_Encoder_Inner_Low = GBL_Data[AngleEncoderInner];
      Angle_Encoder_Outer_Low = GBL_Data[AngleEncoderOuter];
      }
```

"Now that all the gears are lightly meshed in one direction, remember the encoder readings on this 'side' of the backlash."

```
    // reading phase 2
    else if (GBL_Elapsed_mS < read_encoders_time_2)
      {
      // send a small PWM signal to the hip in the other direction
      GBL_Data[PWMDesiredHip] = S16int2FFloat(-75);
      // apply a considerable torque at the ankles
      GBL_Data[TorqueInner] = S16int2FFloat(1);
      GBL_Data[TorqueOuter] = S16int2FFloat(1);
      }
```

"Send a small PWM signal to the hip motor to eliminate backlash in the *other* direction. Apply a significant torque at the ankles *to stretch the cables*."

```
    // reading phase 2 almost over
    else if (GBL_Elapsed_mS == read_encoders_time_2)
      {
      // continue to send the small PWM signal
      GBL_Data[PWMDesiredHip] = S16int2FFloat(-75);
      // continue to apply the considerable torque
      GBL_Data[TorqueInner] = S16int2FFloat(1);
      GBL_Data[TorqueOuter] = S16int2FFloat(1);
```

"The second measurement is about to be taken, but keep applying the torques…"

```
      // now make the second set of measurements
      GBL_Hip_Backlash =  GBL_Hip_Backlash - *TMRA0_CNTR; // calculate the
                                                          // hip backlash
      GBL_Hip_Backlash = GBL_Hip_Backlash>>1; // divide by two
      *TMRA0_CNTR = -GBL_Hip_Backlash; // re-initialize hip counter
```

"Calculate the backlash in the hip as the difference between `GBL_Hip_Backlash`, the counter reading on the other side of the backlash, and `*TMRA0_CNTR`, the counter reading on this side of the backlash. Redefine `GBL_Hip_Backlash` as half of its previous value. We'd like the center of the backlash to correspond with `*TMRA0_CNTR` equal to zero, but the motor is at one side of the backlash now, re-initialize `TMRA0_CNTR` equal to the negative of half the number of encoder counts of backlash."

```
      Angle_Encoder_Inner_High = GBL_Data[AngleEncoderInner];
      Angle_Encoder_Outer_High = GBL_Data[AngleEncoderOuter];
      GBL_Data[AngleEncoderStiffnessInner] = FFsub(Angle_Encoder
       _Inner_High,Angle_Encoder_Inner_Low);
      GBL_Data[AngleEncoderStiffnessOuter] = FFsub(Angle_Encoder
       _Outer_High,Angle_Encoder_Outer_Low);
      }
```

Since the difference in applied torques between the first and second reading is approximately the unit value of 1 Nm (actually .95 Nm, but the difference is negligible), the stiffness is simply the difference between the encoder reading taken at high applied torque and the encoder reading taken at negligible torque.

```
    // test phase 1
    else if (GBL_Elapsed_mS < TestHipEncoderTime1)
      {
      GBL_Data[PWMDesiredHip] = S16int2FFloat(+75);
      }
    // test phase 2
    else if (GBL_Elapsed_mS < TestHipEncoderTime2)
      {
      GBL_Data[PWMDesiredHip] = S16int2FFloat(-75);
      }
```

"Apply small torques to the hip again so that the operator can confirm that `CorrectedAngleEncoderHip` responds as expected."

```
      else // almost done!
        {
        BacklashMeasured = 0; // Next time MODECALENCODERS runs,
                              // start the routine at the beginning
        Calibrated = 1; // Encoders are calibrated
        SpringCalibrated = 0; // Springs needed to be re-calibrated
        GBL_Data[RobotMode] = MODESTANDBY; // Return to standby mode
        LCDWriteWord("CalEncDn"); // Indicate this on the LCD Displace
        }
      // Calculate the PWM values that need to be sent to the motors to
      // produce the desired torques
      TorqueControllerInner();
      TorqueControllerOuter();
      SendPWM(); // Send those PWM signals to the motors
      break;
      }
    }
    break;
  }
```

"Finally, reset `BacklashMeasured` and end the `MODECALENCODERS` case."

The readings of `AngleEncoderHip` and `CorrectedAngleEncoderHip` during the calibration routine are compared in Figure C-9. The readings of `AngleEncoderInner` and `CorrectedAngleEncoderInner` during the calibration routine are compared in Figure C-10.

C-3.2 Absolute Angle Reset

The gyros on the Cornell Ranger sense angular rate. Relative angles can be determined by integrating the angular rate, but absolute angle determination is possible only if a known, initial reference angle is established. This reference angle can be determined during gyro calibration, when the robot is stationary in a known configuration. Small offsets and random noise in the gyro output coupled with the limited precision of analog to digital converion and the discrete, numerical integration of the signal, however, generally cause the integrated angle to become less accurate, or 'drift' over time. Thus it is necessary to periodically reset the reference angle to properly measure absolute angle with respect to the inertial frame. Ideally, the accelerometer can be used to determine the orientation of the gravity vector, but offset, noise, and accelerations other than that of gravity complicate this measurement. Due to the two dimensional nature of this the machine, though, the absolute angle of the swing leg can be determined according to relative angle measurements made by the encoders and the geometry of the double stance condition, when both feet are in contact with the ground. A diagram of this condition is presented in Figure 11(a). Fixing origin O and rotating our perspective θ_W yields Figure 11(b). We will refer the distance between the ankle pivot and the center of the foot profile arc as $l_1 = \overline{OA} = \overline{CD}$, the length of the leg as $l_2 = \overline{AB} = \overline{BC}$, and the ratio of the two lengths as $r = \frac{l_2}{l_1}$. Refering to Figure 1(b), we recognize that $\angle OAB = \psi_A = |\psi_W|$ where $|\psi_W|$ is the absolute magnitude of ψ_W. Also, $\angle ABC = |\phi|$, and $\angle BCD = \psi_C = |\psi_T|$, all of which are measured by the encoders. Making use of rotation matrices and MATLAB symbolic manipulation, we can determine the vector from O to D as

$$\boldsymbol{r_{O\to D}} = [-l_1 \sin\psi_A + l_2 \sin\phi - l_1 \sin(\phi - \psi_C)]\hat{\boldsymbol{\imath}} + [-l_1 \cos\psi_A + l_2(1-\cos\phi) + l_1 \cos(\phi - \psi_C)]\hat{\boldsymbol{\jmath}}. \quad \text{(C-1)}$$

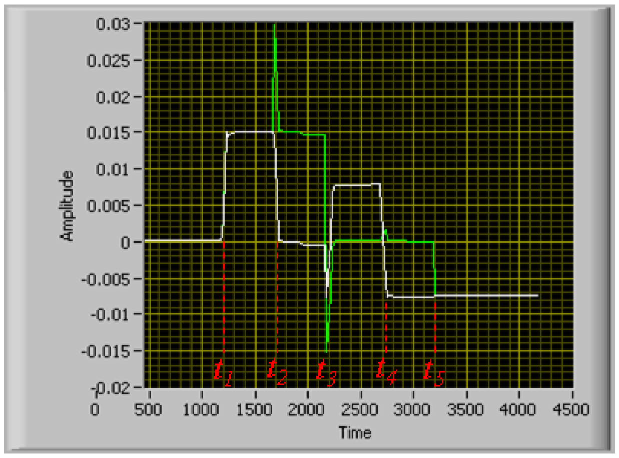


Figure C-9: A comparison of `AngleEncoderHip`(white) and `CorrectedAngleEncoderHip`(green) during the calibration routine, during which the true hip angle is fixed at zero. At time t_1, the backlash has not yet been measured, so the signals are identical. During reading phase 1, a small PWM value is applied to the hip motor, causing the backlash to be eliminated in one direction. At time t_2 corresponding to the end of reading phase 1, the backlash variable is `GBL_Backlash_Hip` is initialized (although the value is not final), causing the corrected value to jump. The motor eliminates backlash in the other direction during reading phase 2. At time t_3, the backlash has been properly measured, and the first test phase begins. The motor applies a torque in the first direction, eliminating backlash, and while the uncorrected signal jumps, the corrected signal immediately returns to the zero value - as the hip itself has not moved. At time t_4, the second test phase begins, and the motor eliminates backlash in the other direction. The corrected value once again returns to zero, while the uncorrected signal would suggest that the hip has rotated. At time t_5 the motor stops applying torque, so the true position of the hip is uncertain, and thus the corrected measurement assumes the uncorrected value.

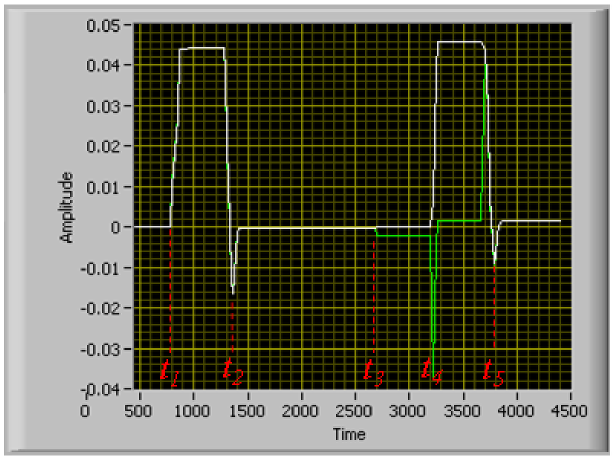
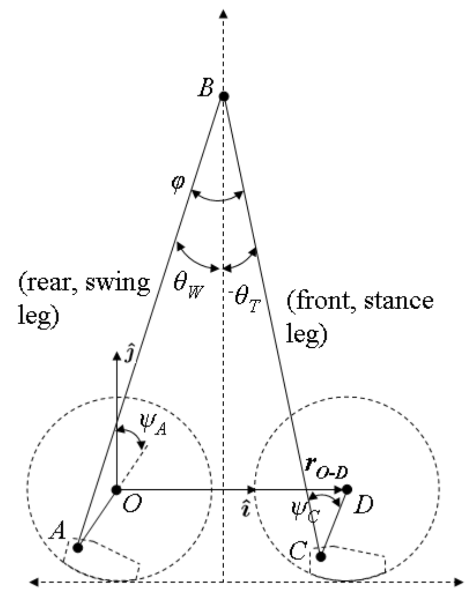
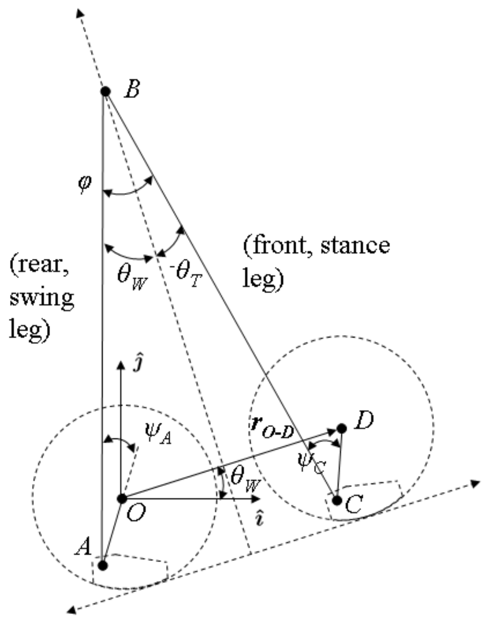


Figure C-10: A comparison of `AngleEncoderInner`(white) and `CorrectedAngleEncoderInner`(green) during the calibration routine, during which the true inner ankle angle is fixed at zero. During reading phase 1 (prior to t_1), a small torque is applied by the inner ankle motor, causing the encoder to turn and the backlash in one direction to be eliminated. At time t_1, the cable stiffness has not yet been measured, so the signals are identical. A considerable torque is applied during reading phase 2, causing the cables to stretch and the signals to jump as the encoders turn. At time t_2, the torque turns off and the encoders return to the zero position. At time t_3, the calibration routine begins again with reading phase 1. The small applied torque causes the corrected measurement to drop slightly, but does not cause the cable to stretch, and thus the uncorrected measurement does not change. However, at time t_4 reading phase 2 begins again, and a larger torque is applied, causing the cables to stretch, and the uncorrected measurement to jump. The corrected measurement returns to near the zero position, as it should. At time t_5, the torque is turned off, and thus the two measurements become identical.



(a) ...from the usual perspective



(b) ...from a perspective rotated by θ_1

Figure C-11: Diagrams of the double stance condition...

Then,

$$\theta_W = \tan^{-1}\left[\frac{r - \cos\psi_A - r\cos\phi + \cos(\phi - \psi_C)}{-\sin\psi_A + r\sin\phi - \sin(\phi - \psi_C)}\right]. \tag{C-2}$$

Subsituting measurement variables for physical variables, this formula is evaluated `GBL_Param[ResetGyroDelay]` milliseconds after heelstrike in the `StateEstimator()` function to determine `GBL_Data[AngleSwingLeg]`, which is a measurement of θ_W. All other absolute leg angles can be calculated from θ_W and known relative angles. Actual results are shown in Figure C-12.

C-3.3 Automatic Gyro Calibration

Each gyro on the Cornell Ranger outputs a voltage that is approximately linearly proportional to its angular rate, but the slope (known as gain or transduction constant) and offset can vary among particular units, and even then can change noticeably over the course of a few steps. Differences between the actual gyro gain and offset and those assumed by the robot controller lead to error in measurement. Fortunately, it is possible to automatically measure the gyro gain and offset periodically to minimize these errors.

For example, the offset of the inner gyro can be measured as follows:

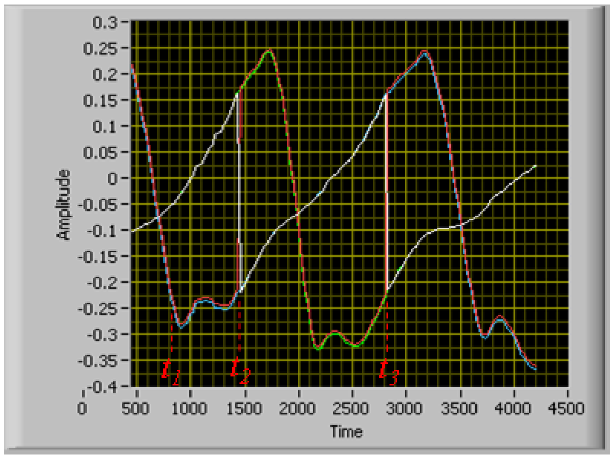


Figure C-12: Leg angle measurements `GBL_Data[AngleStanceLeg]` (white), `GBL_Data[AngleSwingLeg]` (red), `GBL_Data[AngleGyroInner]` (green), and `GBL_Data[AngleGyroOuter]` (blue) over a complete cycle. At time t_1, the inner leg is the stance leg, the outer leg is the swing leg, and all angles are being integrated from angular rate data. At time t_2 heelstrike occurs, the outer leg becomes the stance leg, and the inner leg becomes the swing leg, and all the angles are reset acording to kinematics. Continuity between curves at heelstrike indicates agreement between integrated values and values calculated according to Equation C-2. Note that discontinuity of individual stance and swing angle curves at heelstrike are expected as the stance leg becomes the swing leg and vice versa. Also note the slight difference between `GBL_Data[AngleGyroInner]` and `GBL_Data[AngleSwingLeg]` when the inner leg is the swing leg, and between `GBL_Data[AngleGyroOuter]` and `GBL_Data[AngleSwingLeg]` when the outer leg is the swing leg, which is acceptable due to the way each is defined.

- Upon heelstrike of the inner foot, determine the angle of the inner leg according to the kinematic relationship developed in Equation C-2
- Until next heelstrike of the inner foot, integrate inner gyro rate data to track the inner leg angle; do *not* reset this inner leg angle at heelstrike of the outer foot[C-2]
- Upon the next heelstrike of the inner foot, determine the new angle of the inner leg according to the kinematic relationship developed in Equation C-2
- Calculate the difference between the new, kinematically determined angle and the value integrated from the previous kinematically determined angle
- Theoretically these should be identical; any error must be due to error integration of angular rate
- Most of this error will be due to gyro offset, because the numerical integration error is small, and the gyro gain is not significant because the net rotation of the leg is approximately zero
- Add a fraction of this error to `GBL_Data[OffsetGyroInner]`

[C-2]This is the reason why `GBL_Data[AngleGyroInner]` is not reset at outer leg heelstrike and `GBL_Data[AngleGyroOuter]` is not reset at inner leg heelstrike.

Consider measurement of the gain of the inner gyro:

- Upon heelstrike of the inner foot, determine the angle of the inner leg according to the kinematic relationship developed in Equation C-2
- Until next heelstrike, integrate inner gyro rate data to track the inner leg angle
- Upon heelstrike of the outer foot, determine the angle of the inner leg according to the kinematic relationship developed in Equation C-2
- Calculate the change in inner leg angle according to the two kinematically determined values
- Calculate the change in inner leg angle according to the integrated inner gyro rate
- Calculate the ratio between these two measurements
- Theoretically this ratio should be unity; any error must be due to error in the integration of angular rate
- Most of this error will be due to inaccuracy in the gyro gain, because the numerical integration error is small, and the drift due to gyro offset is assumed to have little effect over half a step.
- Factor a fraction of this error into `GBL_Data[GyroGainInner]`

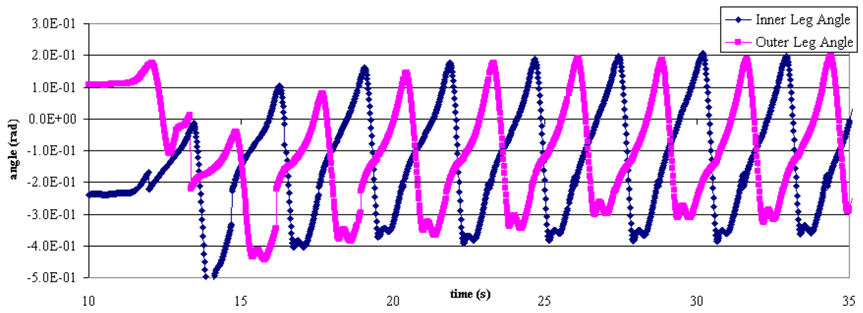
These calculations are implemented in the `StateEstimator` function. Results of automatic gyro calibration are shown in Figure C-13

C-4 Recommended Improvements

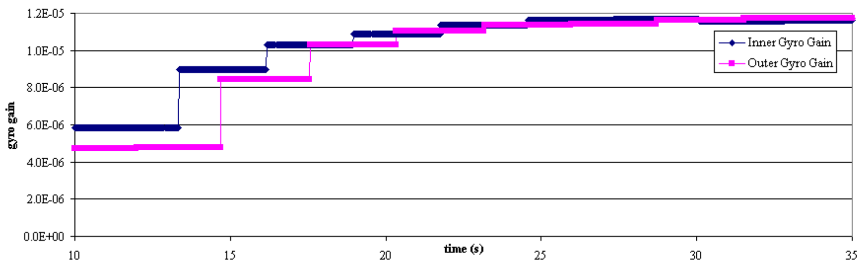
In order to be fully functional, several improvements must be made:

- The formula for θ_W developed in C-3.2 is not valid when either foot contacts the ground at the point of the heel instead of the arc of the foot. The state estimator must be modified to detect this condition and formulae for the alternate cases must be developed and implemented.
- Figure 5(b) shows unnaceptable drift immediately following gyro calibration. The algorithm used by the `GyroZero()` function to determine the offset must be improved.
- The gyro offsets should be measured in rad/s instead of the unintuitive units currently used, and all relevant routines must be converted for use with the new convention.
- Automatic gyro calibration is not enabled in the most recent code for the following reason: when the robot is being tested and the operator handles the robot, it is possible for the gyros to measure rotations that are outside of the usual plane. When the change in stance angle as measured by integration of angular rate is compared to the change in stance angle according to geometry, very large discrepancies causing large, undesired changes in gyro gain and offset. Provisions must be made to minimize this effect; for instance, perhaps the angular rate should only be integrated when at least one foot is in contact with the ground.
- The Z accelerometer must be used to determine the orientation of the gravity vector. This information should be intelligently combined with the integrated gyro rates and the kinematically determined angles to improve the accuracy of all absolute angles.

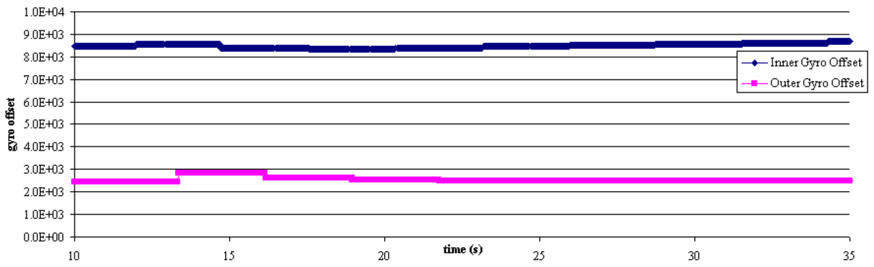
Once these changes are implemented and verified, robot state estimation will be sufficient for use by most controllers.



(a) The inner/outer leg angle is reset according to the kinematic formula of C-3.2 each time the inner/outer foot experiences heelstrike; otherwise the angle is integrated from gyro data. Initially, there is poor agreement between the integrated angle and new kinematically determined angle at heelstrike. After several steps, however, the gyro gains have been automatically calibrated to yield better agreement.



(b) The gyro gains are automatically calibrated and settle upon final values.



(c) Measured gyro offsets are adjusted only slightly over time.

Figure C-13: Demonstration of automatic gyro gain calibration. Initially, the gyro gain has intentionally been set to about half its nominal value, so the automatic gyro calibration routine corrects it.