# Modeling Dynamics and Exploring Control of a Single-Wheeled Dynamically Stable Mobile Robot with Arms

Eric M. Schearer

CMU-RI-TR-06-37

August 31, 2006

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

*Submitted in partial fulfillment of the requirements for the degree of Master of Science*

## Abstract

This paper focuses on simulations of a dynamically stable mobile robot (Ballbot) with arms. The simulations are of Ballbot lifting its arms in various directions. A PD arm controller works independently of an LQR-designed balancing/station keeping controller. The PD controller drives the arms to follow desired trajectories. When the arms are raised, Ballbot assumes a leaning equilibrium (the physical equilibrium) as opposed to the standing equilibrium (body stands totally upright - a predefined desired equilibrium) that the LQR drives toward. The conflict between these two equilibria causes the robot to lose its balance when lifting heavy (10 kg) loads. A unified arm and station keeping/balancing controller is also described. The unified controller outperforms the independent controllers in some cases. Balancing only using arms and driving body movement with arms are briefly explored.

## Acknowledgments

Thanks to my advisor, Dr. Ralph Hollis for his vision in creating Ballbot and guiding my work. Thanks to Dr. George Kantor for constant technical advice and brainstorming. Thanks to Anish Mampetta for his companionship and ideas through all of our experimenting with Ballbot. Anish was also the first to derive the Ballbot motion equations using the current coordinates. Thanks to Dr. Matt Mason and Jonathan Hurst for reviewing my work as part of my committee. Thanks to my family, friends, and classmates for every day support. This research is supported in part by NSF grant IIS-030867. The United States Air Force funds my tuition. Thanks to Col(ret.) Chuck Wolfe, my supervisor at the Air Force Research Lab Munitions Directorate and to the Engineering Mechanics Department at the U.S. Air Force Academy for making the Air Force support possible.

# Contents

## 1 Introduction

Ballbot is a mobile robot with human-like height, width, and weight. It actively balances and moves on a single wheel using closed loop feedback, making it dynamically stable. Dynamic stability affords it advantages in maneuverability over statically stable robots and makes it a good candidate for operating in human environments. Balancing on a ball allows Ballbot to be omni-directional, allowing it to move in any direction without turning.

The current version of Ballbot has no arms and is described in detail in [4] and [5]. Ballbot's structure is three aluminum channels held together by circular decks that rest on top of a single ball as seen in Figure 1. Figure 1a shows Ballbot standing with its three legs (used when not dynamically balancing) deployed. On the circular decks are a battery, a computer, a battery charger, and an inertial measurement unit (IMU) for measuring the tilt angle of the body as seen in Figure 1b. Figure 1c. shows Ballbot balancing.

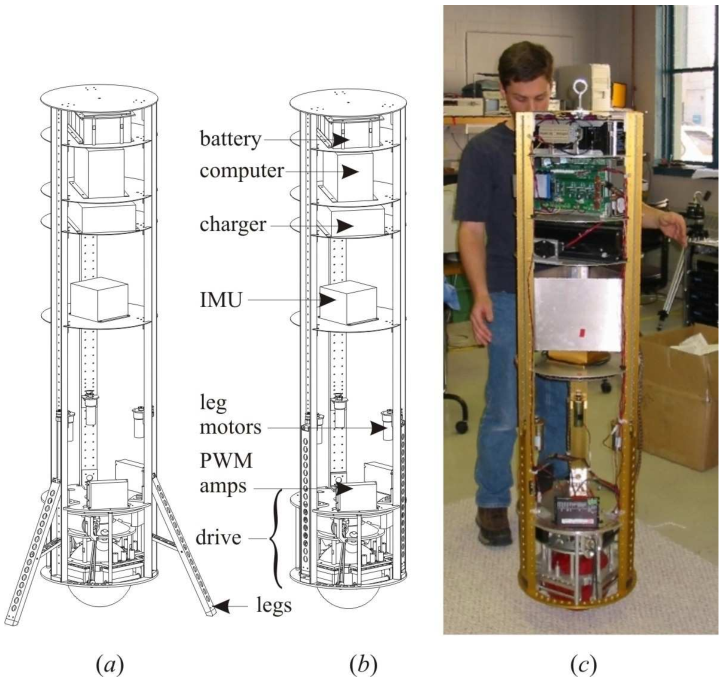


Figure 1:

The drive mechanism in Figure 2 includes four motors that drive the rollers which drive the ball. Encoders on the motors measure the position of the rollers and thus the position of the ball. A feedback controller allows Ballbot to move and balance.

Future versions of Ballbot will include arms with two degrees of freedom and will look something like Figure 3. Arms will give Ballbot the capability to manipulate objects. Thibodeau et. al. [9] have shown that dynamically stable mobile robots with arms can apply larger contact forces than similar statically stable mobile manipulators. Arms can also increase the stability and mobility of Ballbot.

The objectives of this work are:

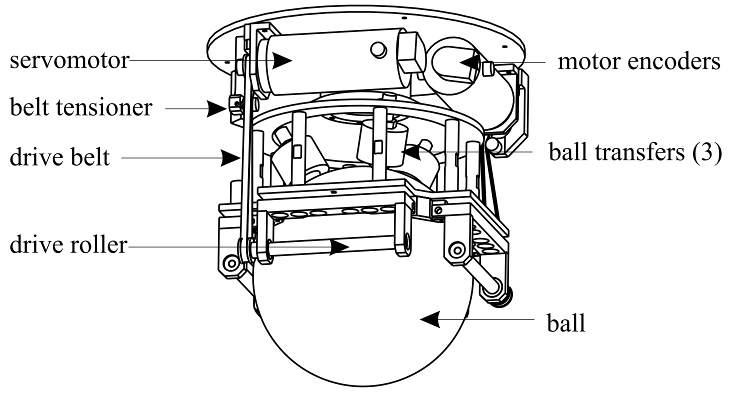


Figure 2:

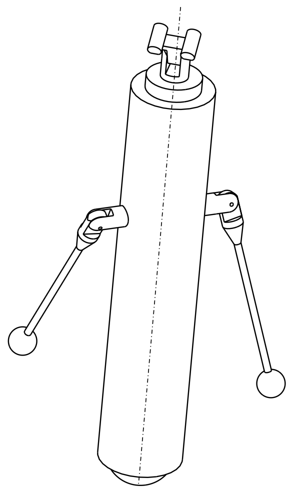


Figure 3:

- Create a 3D mechanical simulation of Ballbot with arms to study the dynamics of the armed configuration. Section 2 discusses this model.
- Design a simple arm controller to operate simultaneously with and independently of a body station keeping/balancing controller. This is seen in Section 3.
- Design a unified station keeping/balancing and arm controller that uses feedback from the arms, ball, and body to both balance and move the arms. Section 4 describes this controller.
- Section 5 evaluates and compares the performances of these two controllers.
- Section 6 suggests future work.

# 2 Modeling and Simulation

This section describes a 3D Ballbot simulation in SimMechanics and a planar Ballbot model used for deriving controllers.

## 2.1 Simulation in SimMechanics

A 3D simulation of Ballbot with arms was created using SimMechanics. SimMechanics is a mechanical modeling package used with Simulink®. SimMechanics uses the Newton-Euler equations to describe the motion of rigid bodies and allows users to create models in the Simulink® block environment. Rigid bodies are modeled by a center of mass position and body orientation relative to some coordinate system (a fixed coordinate system or a coordinate system of a connected body), a mass, and an inertia tensor. The user can create degrees of freedom by connecting bodies with joints and take away degrees of freedom with constraints. Virtual sensors connected to bodies and joints allow the user to measure outputs and feed them back to controllers which send signals to actuators that can apply forces, torques, or prescribed motions to joints and bodies.

### 2.1.1 SimMechanics Block Diagram

Figure 4 shows a block diagram modeling the Ballbot with arms in SimMechanics. The model consists of nine rigid bodies (the ground, the ball, the body, two arms, and four drive rollers), the joints connecting them, the constraints limiting their motion, and two controllers that sense movement and apply torques to the bodies. Orange blocks represent bodies, green blocks represent joints, red blocks represent constraints, blue blocks represent controllers, and the cyan block represents the fixed ground. The "Ball" is connected to "Ground1" by a six degree of freedom joint. The 6-DoF joint is three perpendicular prismatic axes and a quaternion representing rotation. The "Rolling Constraint" block constrains the relative motion of the ball with respect to the ground. The rolling constraint equation is:

$$\begin{bmatrix} v_{bx}^g \\ v_{by}^g \\ v_{bz}^g \end{bmatrix} = \begin{bmatrix} r_b\omega_{by}^g \\ r_b\omega_{bx}^g \\ 0 \end{bmatrix}$$

where $v_b$ and $\omega_b$ are the linear and angular ball velocities, $r_b$ is the ball radius, the $z$ axis is vertical, and the $x$ and $y$ axes are parallel to the ground. These constraints prescribe that the ball rolls on the ground without slipping and does not move in the vertical direction. Superscripts indicate the coordinate system where $g$ stands for ground. Subscripts refer to bodies and vector components.

The "Body" which represents the Ballbot's tower of motors, sensors, computers, etc. is connected to the "Ball" by a spherical joint represented by a quaternion. Revolute joints, defined by an axis and rotation angle, connect the four rollers ("y1 roller", "x1 roller", "y2 roller", and "x2 roller") to the "Body". The red block, "Roller Constraints", constrains the relative motion of the rollers and ball. These constraint equations are:

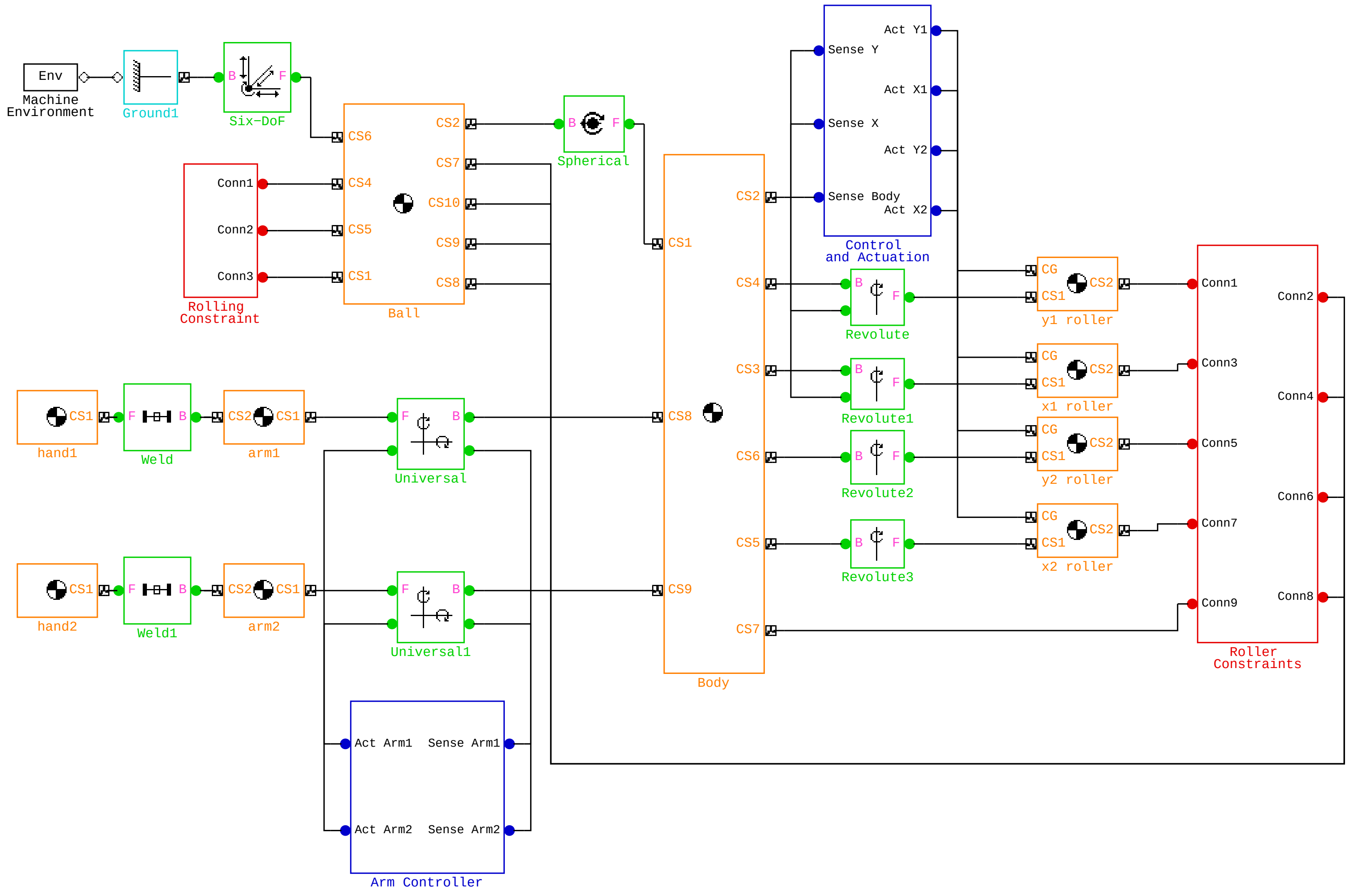


Figure 4: SimMechanics Block Diagram

$$r_{x1r}\omega_{x1rz}^{x1r} = r_b\omega_{by}^{g} = r_{x2r}\omega_{x2rz}^{x2r}$$

$$r_{y1r}\omega_{y1rz}^{y1r} = r_b\omega_{bx}^{g} = r_{y2r}\omega_{y2rz}^{y2r}$$

where $x1r$, $x2r$, $y1r$, and $y2r$, refer to the rollers and $b$ refers to the ball. The $z$ axis of each roller is its rotation axis. These constraints mean that the rollers roll without slipping on the ball. The rollers can slip on the ball in the non-rolling directions. For example, in Figure 2, if the ball is rolling on the roller labeled "drive roller", it must be slipping on the roller perpendicular to the "drive roller" (the other visible roller). The second roller still will roll on the ball without slipping when it is rotating. Note that the ball angular velocity is expressed in the ground frame when it should be expressed in the corresponding roller frame. A major flaw of SimMechanics is that it only allows the user to express constraint equations in either the body frame or the ground frame and not in the coordinate frame of another body. Here we want to express the ball's angular velocity in the roller frame. With these limitations, if Ballbot yaws, the constraints are no longer correct (making the simulation somewhat invalid) because the ground frame no longer lines up with the roller frames.

The arms, "arm1", and "arm2" are connected to the "Body" by universal joints, which allow two perpendicular degrees of rotational freedom. Based on the Ballbot's height (about 1.5 m), Ballbot's arms are 0.58 m long - proportional to a human. The arms are cylinders with the density of aluminum. The hands "hand1" and "hand2" are spheres welded to the ends of the arms.

The blue "Control and Actuation" block senses the orientation of the "Body" and positions of the rollers (and the arms in one controller) and applies torques to the rollers. The "Arm Controller" block senses the position of the arms (and the body orientation and roller positions in one controller) and applies torques to the arms. The appendix discusses these two blocks in more detail. Section 3 describes the controllers in detail.

### 2.1.2 Friction Model

The friction model is found in the "Control and Actuation/Actuation and Friction" block of Figure 4. No friction is modeled in the arms, and no friction is explicitly modeled between the ball and ground and between the ball and body. The ball/ground, ball/body, and ball/roller friction torques are lumped together and implemented by subtracting friction torques from the torque applied to the rollers as follows:

$$\tau = \tau_r - \tau_s - \tau_v - \tau_c \tag{1}$$

where $\tau$, $\tau_r$, $\tau_s$, $\tau_v$, and $\tau_c$, are the total torque applied to the roller, the torque that the controller prescribes, the torque on the roller due to static friction, the torque on the roller due to viscous friction, and the torque on the roller due to coulomb friction, respectively. The friction torques are further defined below.

$$\tau_s = \begin{cases} \tau_r & \text{if } \tau_r \leq \tau_{sm} \\ 0 & \text{if } \tau_r > \tau_{sm} \end{cases} \tag{2}$$

where $\tau_{sm}$ is the maximum static friction torque.

$$\tau_v = \gamma_v \omega_{rz} \tag{3}$$

where $\omega_{rz}$ is the angular velocity of the roller about its rotation axis.

$$\tau_c = \gamma_c \text{sign}(\omega_{rz}) \tag{4}$$

To determine values for $\tau_{sm}$, $\gamma_v$, and $\gamma_c$, a series of Ballbot tests were conducted. The tests were conducted in the first floor hallway of Smith Hall on the black ribbed carpet. Ballbot rolled on a hollow aluminum ball with urethane coating. No controller was used. For each test, predetermined torque commands were given to the motors. Without a controller to keep Ballbot standing, people held Ballbot vertical as it moved. The first test series used a slow ramping (increment torque input by 0.0785 Nm or 5 DAC counts per second) of the input torque. For much of each of these runs the torque was not enough to cause any ball movement. Eventually enough torque was applied to move the ball. This first series determined the static friction torque. The next set of tests used faster increases to the input torque (1.57 Nm or 100 counts per second) to provide data to estimate the viscous and coulomb friction torques. We ran tests in positive and negative $x$ and $y$ directions and in the diagonal directions as well.

The maximum static friction torque for each test run is the value of $\tau_r$ when the ball velocity becomes non-zero. By looking at the data (ball velocity vs. time plots), we can determine the maximum static friction. For example, in Figure 5a the ball appears to move shortly after 3 seconds. We know that the rollers applied 4.71 Nm to the ball at that time. This torque is the maximum static friction for the particular test run. The ball velocity threshold used was 0.1 rad/s. $\tau_v$ and $\tau_c$ were determined by assuming a quasi-steady state for the region of the data where the velocity is non-zero, namely:

$$\tau_r - \tau_v - \tau_c = 0 \tag{5}$$

substituting equations (3 - 4) into equation (5) and rearranging terms:

$$\gamma_v \omega_{rz} + \gamma_c = \tau_r \tag{6}$$

The process for finding $\gamma_v$ and $\gamma_c$ is outlined below:

- Fit a line to the ball velocity vs. time plot in the non-zero velocity region.
- Compute the value of the ball velocity at the end of the test run using the best fit line.
- Plug this final ball velocity (converted to roller velocity) into equation (6) for each test.
- Now we have a system of n (number of test runs for each direction) equations in two unknowns, $\gamma_v$ and $\gamma_c$. Solve these equations using singular value decomposition (SVD).
- An alternate solution method was using two ball velocity values from the best fit line and solving a system of 2 equations and 2 unknowns for each test run. Then average the $\gamma_v$ and $\gamma_c$ values for each test run over the set of similar (same ball velocity direction) test runs.

Table 1 shows the values of each of the friction terms for a number of different directions (e.g. $+x$-$y$ is the direction 45 degrees from the positive $x$ and 45 degrees from the negative $y$) and for all directions. The fact that the values vary across the different directions suggests that friction is not isotropic.

| **Direction** | $\tau_{sm}$ **(AVG)** | $\gamma_v$ **(AVG)** | $\gamma_c$ **(AVG)** | $\gamma_v$ **(SVD)** | $\gamma_c$ **(SVD)** |
|---|---|---|---|---|---|
| $+x$ | 4.91 | 2.05 | 5.45 | 2.11 | 5.42 |
| -$x$ | 5.06 | 2.18 | 5.60 | 0.91 | 6.38 |
| $+y$ | 4.3 | 1.56 | 5.30 | 0.61 | 6.72 |
| -$y$ | 5.38 | 2.01 | 6.41 | 1.12 | 8.58 |
| $+x+y$ | 4.35 | 0.96 | 5.82 | 0.96 | 5.82 |
| $+x$-$y$ | 7.04 | 1.84 | 7.06 | 1.75 | 7.33 |
| -$x+y$ | 4.35 | 0.61 | 6.08 | 0.58 | 6.46 |
| -$x$-$y$ | 5.55 | 2.74 | 6.02 | 1.48 | 6.25 |
| All | 5.01 | 2.08 | 5.61 | 0.57 | 8.88 |

Table 1: Model Coefficients

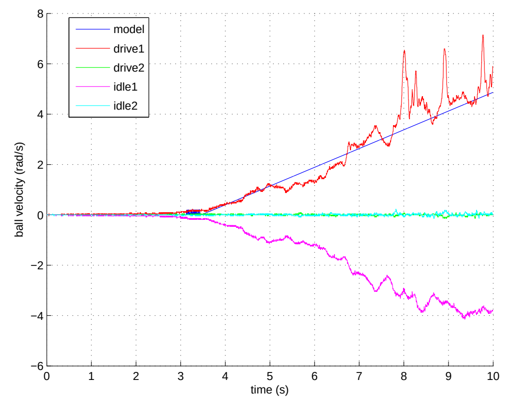
Note that the data in Table 1 are for ball torques which were used for a previous simulator. The current simulator uses roller torques so the appropriate conversion must be made. Also note that the test data was taken when Ballbot had only two driven rollers (now all four are driven). To account for more driven rollers, the current simulation uses the Table 1 values divided by 2. This may be a poor assumption, and the tests should be run again to obtain more accurate friction terms.

To check the goodness of the friction terms in Table 1, a 1D simulation of the ball motion ignoring the effects of the body was created. The simulation solves the model equation (7) using the new friction values in Table 1. The moment of inertia of the ball is $I_b = 0.0463\,\mathrm{kg\,m^2}$, and the angular acceleration of the ball is $\ddot{\theta}_b$. Plots of the simulation output with the corresponding plots of hallway test runs are displayed in Figure 5. In the legend of Figure 5, "drive1" and "drive2" refer to the $x$ and $y$ ball positions taken from the motor encoders, and "idle1" and "idle2" refer to the $x$ and $y$ ball positions taken from the passive roller encoders. Note that the motor encoders and passive roller encoders have opposite polarity, but are both meant to measure ball position.

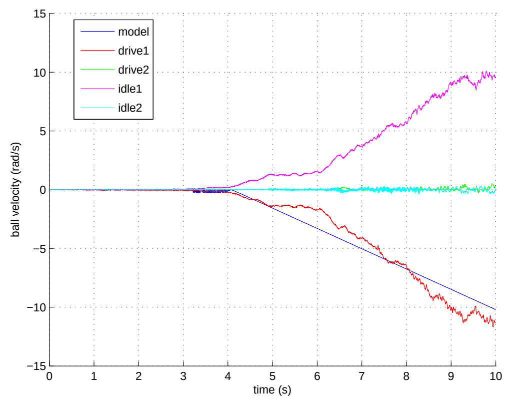
$$I_b\ddot{\theta}_b = \tau_r - \tau_s - \tau_v - \tau_c \tag{7}$$

Figure 6a shows ball velocity vs. time for the simulation using a model with no friction and using one with only static friction. Figure 6b shows ball velocity vs. time for the simulation using a model with all of the friction terms and one using static and viscous friction. Notice the large disparity of velocity magnitudes between these two figures. This disparity indicates that viscous friction is a major factor. Also notice in Figure 6b that coulomb friction has a significant effect on the ball velocity.

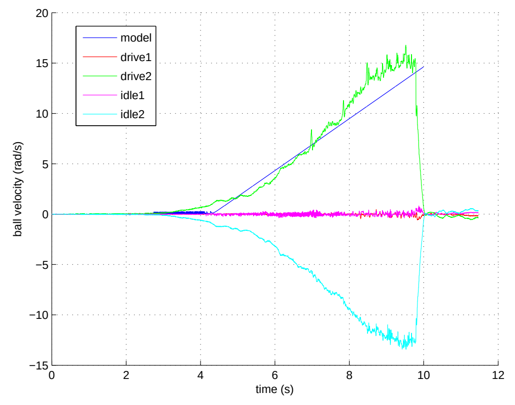
The quasi-static assumption means that by ramping the torque slowly, the dynamic effects are negligible. Judging by the accelerations seen in Figure 5a-b (1-2 rad/s) and $I_b$, the left side of equation (7) is small ($< 0.1\,\mathrm{Nm}$) compared to the right side ($> 5\,\mathrm{Nm}$) making the quasi-static assumption reasonable.



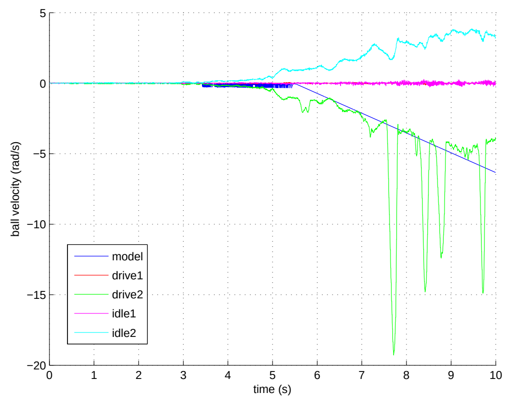
(a) Positive $x$ Direction, Ramp = 100



(b) Negative $x$ Direction, Ramp = 100

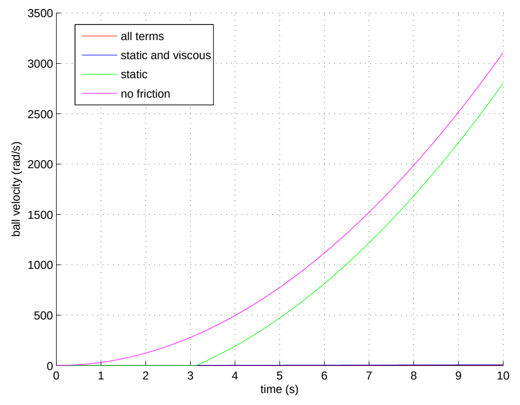


(c) Positive $y$ Direction, Ramp = 100

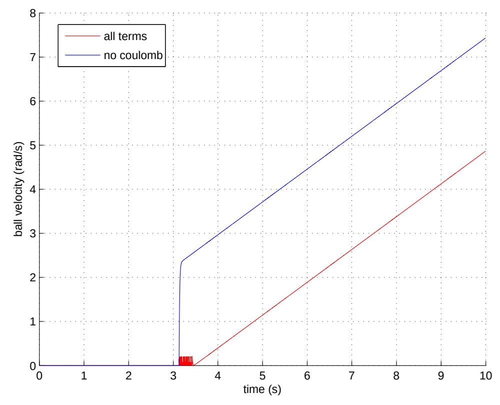


(d) Negative $y$ Direction, Ramp = 100

Figure 5: Friction Test and Simulation Results



(a) Output of Limited Friction Models



(b) Output of Better Friction Models

Figure 6: Comparison of Friction Models

### 2.1.3 Actuation Model

The actuation model is found in the "Control and Actuation/Actuation and Friction" block of Figure 4. Figure 7 is a block diagram of the Ballbot actuation system. An input is sent to the digital to analog converter (DAC) which outputs a voltage to the amplifier. The amplifier sends a current to the motor which outputs a torque to the roller. The roller then transmits a torque to the ball. In each block the input is multiplied by the number in the block to produce the output.

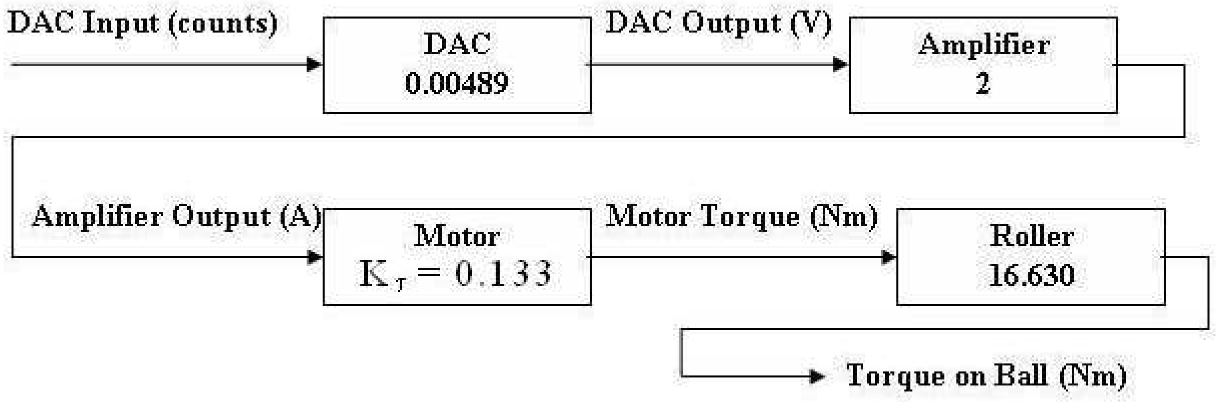


Figure 7: Actuation System Block Diagram

Below are the DAC output voltage, $V_{DAC}$, amplifier output current, $I_{out}$, and motor torque, $\tau_r$, expressed in terms of the DAC input, $C_{DAC}$. The DAC input is an integer in the range -2048 to 2047 and has units counts.

$$V_{DAC} = \frac{10 * C_{DAC}}{C_{max}} = \frac{10 * C_{DAC}}{2047} \approx 0.00489 * C_{DAC} \text{ V} \tag{8}$$

where the maximum magnitude DAC input, $C_{max}$, is 2047. We have assumed that the amplifier operates in torque mode. According to page 23 of the amplifier spec sheet, the amplifier outputs a current, $I_{out}$, according to:

$$I_{out} = \frac{V_{DAC} I_{peak}}{10} \text{ A} = \frac{10 * C_{DAC} * 20}{2047 * 10} \text{ A} \approx 0.00977 * C_{DAC} \text{ A} \tag{9}$$

where $I_{peak}$=20A is the peak current limit. If the magnitude of $I_{out}$ is greater than $I_{peak}$, then $I_{out}$=$I_{peak}$sign($I_{out}$). In the real amplifier (Copley Controls Model 412), if the continuous current limit is exceeded, a capacitor with a 1 second time constant at peak current begins to charge. When it is fully charged, the output current decays exponentially to the continuous current. The simulation models this as follows: If the magnitude of $I_{out}$ is greater than $I_{cont}$=10 A, the continuous current limit, an integral begins accumulating. If the integral is less than 14.427, then $I_{out}$=$I_{out}$, otherwise $I_{out}$=$I_{cont}$sign($I_{out}$). The 14.427 was chosen so that the integral acts like the real circuit. Note that the peak and continuous current limits can be changed in the amplifier.

The torque constant, $K_\tau$, for the motor is 0.133 Nm/A. To convert DAC input to motor torque,

$$\tau_r = I_{out}K_\tau \approx 0.00977 * C_{DAC}\ \text{A} * 0.133\frac{\text{Nm}}{\text{A}} \approx 0.00130 * C_{DAC}\ \text{Nm} \tag{10}$$

which assuming no losses, is the torque applied to the roller $\tau_r$.

## 2.2 2D Ballbot with Arms Model

In order to design controllers we derive equations of motion for Ballbot. The dynamics of Ballbot with arms are derived using the Lagrangian formulation. Figure 8 shows a planar model of Ballbot. The planar equations of motion (and the corresponding planar controllers) are much simpler both to derive, understand, and implement than the 3D equations. Two perpendicular planar models can represent a 3D Ballbot accurately if Ballbot operates near the upright standing position where coupling effects between the two planes is small. The planar model is likely not as accurate in the case of arms that will operate out of the plane. Note that:

- $\phi$ is measured with respect to the fixed world vertical, just as the IMU on the physical robot measures pitch and roll.
- $\theta$ is measured with respect to moving $\phi$, just as the encoders on the physical robot measure the position of the ball.
- $\psi$ is measured with respect to the moving $\phi$, like the arm encoders would measure the arm angles on the physical robot.

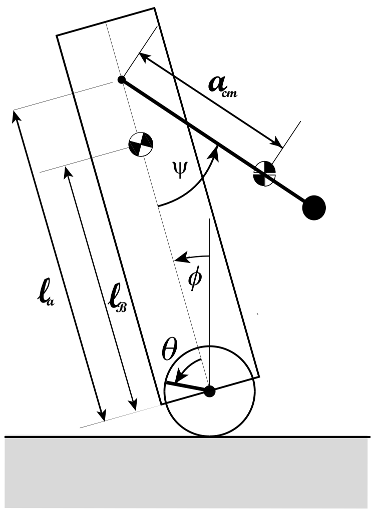


Figure 8: Planar Ballbot Model

The position of the ball, $p_b$, the position of the body, $p_B$, and the position of the arm, $p_a$ are:

$$p_b = \begin{bmatrix} r_b\left(\theta+\phi\right) \\ 0 \\ 0 \end{bmatrix}$$

$$p_B = \begin{bmatrix} r_b\left(\theta+\phi\right) + l_B\sin\phi \\ 0 \\ l_B\cos\phi \end{bmatrix}$$

$$p_a = \begin{bmatrix} r_b\left(\theta+\phi\right) + l_a\sin\phi - a_{cm}\left(\cos\phi\sin\psi + \sin\phi\cos\psi\right) \\ 0 \\ l_a\cos\phi + a_{cm}\left(\sin\phi\sin\psi + \cos\phi\cos\psi\right) \end{bmatrix}$$

The velocity of the ball, $v_b$, the velocity of the body, $v_B$, and the velocity of the arm, $v_a$ are:

$$v_b = \begin{bmatrix} r_b\left(\dot\theta+\dot\phi\right) \\ 0 \\ 0 \end{bmatrix}$$

$$v_B = \begin{bmatrix} r_b\left(\dot\theta+\dot\phi\right) + l_B\dot\phi\cos\phi \\ 0 \\ -l_B\dot\phi\sin\phi \end{bmatrix}$$

$$v_a = \begin{bmatrix} r_b\left(\dot\theta+\dot\phi\right) + l_a\dot\phi\cos\phi + a_{cm}\left(\dot\phi\sin\phi\sin\psi - \dot\psi\cos\phi\cos\psi - \dot\phi\cos\phi\cos\psi + \dot\psi\sin\phi\sin\psi\right) \\ 0 \\ -l_a\dot\phi\sin\phi + a_{cm}\left(\dot\phi\cos\phi\sin\psi + \dot\psi\sin\phi\cos\psi + \dot\phi\sin\phi\cos\psi + \dot\psi\cos\phi\sin\psi\right) \end{bmatrix}$$

The kinetic energy of the ball, $T_b$, the kinetic energy of the body, $T_B$, and the kinetic energy of the arm, $T_a$, are:

$$\begin{aligned} T_b &= \frac{1}{2}m_b{v_b}^\mathsf{T}v_b + \frac{1}{2}I_b\left(\dot\theta+\dot\phi\right)^2 \\ T_B &= \frac{1}{2}m_B{v_B}^\mathsf{T}v_B + \frac{1}{2}I_B\dot\phi^2 \\ T_a &= \frac{1}{2}m_a{v_a}^\mathsf{T}v_a + \frac{1}{2}I_a\left(\dot\psi+\dot\phi\right)^2 \end{aligned}$$

The potential energy of the ball, $V_b$, the potential energy of the body, $V_B$, and the potential energy of the arm, $V_a$, are:

$$\begin{aligned} V_b &= 0 \\ V_B &= -m_B g l_B\cos\phi \\ V_a &= -m_B g\left(l_a\cos\phi + a_{cm}\left(\sin\phi\sin\psi - \cos\phi\cos\psi\right)\right) \end{aligned}$$

The Lagrangian, $L$, is the total kinetic energy minus the total potential energy.

$$L = T - V$$

The Lagrangian formulation dictates:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = \tau_1 \tag{11}$$
$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} - \frac{\partial L}{\partial \phi} = 0 \tag{12}$$
$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\psi}} - \frac{\partial L}{\partial \psi} = \tau_3 \tag{13}$$

where $\tau_1$ is the torque applied to the ball by the rollers (including friction), and $\tau_3$ is the arm torque. Evaluating equations (11-13) using the Matlab® symbolic package and rearranging yields the following equation of motion.

$$M \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \\ \ddot{\psi} \end{bmatrix} + C \begin{bmatrix} \dot{\phi}^2 \\ \dot{\psi}^2 \\ \dot{\phi}\dot{\psi} \end{bmatrix} + G = T \tag{14}$$

where $M$ is the 3x3 mass matrix with columns, $M(:,1)$, $M(:,2)$, and $M(:,3)$,

$$M(:,1) = \begin{bmatrix} mr_b^2 + I_b \\ mr_b^2 + r_b\left(\cos\phi\left(m_B l_B + m_a l_a\right) - m_a a_{cm}\cos\left(\phi+\psi\right)\right) + I_b \\ -m_a r_b c_{ma}\cos\left(\phi+\psi\right) \end{bmatrix}$$

$$M(:,2) = \begin{bmatrix} mr_b^2 + r_b\left(m_B l_B\cos\phi + m_a l_a\cos\phi - m_a a_{cm}\cos\left(\phi+\psi\right)\right) + I_b \\ mr_b^2 + 2r_b\left(\cos\phi\left(m_B l_B + m_a l_a\right) - m_a a_{cm}\cos\left(\phi+\psi\right)\right) + m_B l_B^2 + m_a\left(l_a^2 + a_{cm}^2\right) + I \\ -m_a r_b c_{ma}\cos\left(\phi+\psi\right) - m_a l_a a_{cm}\cos\psi + m_a a_{cm}^2 + I_a \end{bmatrix}$$

$$M(:,3) = \begin{bmatrix} m_a r_b a_{cm}\cos\left(\phi+\psi\right) \\ -m_a r_b a_{cm}\cos\left(\phi+\psi\right) - m_a l_a a_{cm}\cos\psi + m_a a_{cm}^2 + I_a \\ m_a c_{ma}^2 + I_a \end{bmatrix}$$

$m$ is the sum of the three masses,

$$m = m_b + m_b + m_a$$

$I$ is the sum of the three inertias,

$$I = I_b + I_B + I_a$$

$C$ is the 3x3 matrix of coriolis and centrifugal terms with columns $C(:,1)$, $C(:,2)$, and $C(:,3)$,

$$C(:,1) = \begin{bmatrix} -r_b\left(\sin\phi\left(m_B l_B + m_a l_a\right) - m_a a_{cm}\sin\left(\phi+\psi\right)\right) \\ -r_b\left(\sin\phi\left(m_B l_B + m_a l_a\right) - m_a a_{cm}\sin\left(\phi+\psi\right)\right) \\ -m_a l_a c_{ma}\sin\psi \end{bmatrix}$$

$$C(:,2) = \begin{bmatrix} m_a r_b a_{cm} \sin(\phi+\psi) \\ m_a r_b a_{cm} \sin(\phi+\psi) + m_a l_a a_{cm} \sin\psi \\ 0 \end{bmatrix}$$

$$C(:,3) = \begin{bmatrix} 2m_a r_b a_{cm} \sin(\phi+\psi) \\ 2m_a r_b a_{cm} \sin(\phi+\psi) + 2m_a l_a a_{cm} \sin\psi \\ 0 \end{bmatrix}$$

$G$ is the 3x1 gravity matrix,

$$G = \begin{bmatrix} 0 \\ m_B l_B g \sin\phi + m_a l_a g \sin\phi - m_a a_{cm} g \sin(\phi+\psi) \\ -m_a a_{cm} g \sin(\phi+\psi) \end{bmatrix}$$

and $U$ is the 3x1 matrix of applied torques.

$$U = \begin{bmatrix} \tau_r - \tau_s - \tau_c - \gamma_v \dot{\theta} \\ 0 \\ \tau_a - \gamma_{va} \dot{\psi} \end{bmatrix}$$

where $\gamma_{va}$ is the coefficient of viscous friction for the arms.

To apply linear control strategies we must linearize this system using the form of equation (15).

$$\dot{x} = \frac{d\dot{x}}{dx} x + \frac{d\dot{x}}{du} u \tag{15}$$

where the state vector is $x = \begin{bmatrix} \theta & \phi & \psi & \dot{\theta} & \dot{\phi} & \dot{\psi} \end{bmatrix}^{\mathsf{T}}$, and the input vector is $u = [\tau_r \; \tau_a]^{\mathsf{T}}$. We linearize the system of equation (14) about the equilibrium point $\theta = \phi = \psi = \dot{\theta} = \dot{\phi} = \dot{\psi} = 0$ to fit the system of equation (15). This corresponds to Ballbot standing upright up with its arms at its sides.

## 3 Independent Balancing and Arm Controllers

The logical first step for implementing arms with Ballbot is to design an arm controller that works independently of an existing balancing/station keeping controller that uses only the rollers. To control the rollers we design a planar linear quadratic regulator (LQR)[1] excluding the arm states. In this case the state vector is $x = \begin{bmatrix} \theta & \phi & \dot{\theta} & \dot{\phi} \end{bmatrix}^{\mathsf{T}}$, and the input vector is $u = \tau_r$. LQR finds a state feedback law, $u = -Kx$ that minimizes the following cost function near the linearization point.

$$J = \int \left( x^{\mathsf{T}} Q x + u^{\mathsf{T}} R u \right) dt$$

where $Q$ is a 4x4 matrix of weights for the states, and $R$ is the weight for the applied torque. The diagonal terms in $Q$ correspond to the ball position, $\theta$, the body angle, $\phi$, the ball velocity, $\dot{\theta}$, and the body angular velocity, $\dot{\phi}$. The $Q$ and $R$ used to obtain the results that follow are:

$$Q = \begin{bmatrix} 3000 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 \\ 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 30 \end{bmatrix}$$

and $R = 1$. These values were chosen only to achieve reasonable balancing (qualitatively), and not to achieve any specific quantitative performance goal. They are intended to give high importance to the ball position (first diagonal term) and low importance to the torque used. The Matlab® command **dlqr** finds $K$ for the given linear system of equation (15). In order to make Ballbot move we define the augmented control law,

$$u = -K\left(x - x_p\right) \tag{16}$$

where $x_p$ is a vector of desired states that can change in time. In other words, if Ballbot is doing something other than station keeping (non-zero desired state), we must specify $x_p$ as a function of time. We choose the body tilt angle path, $\phi_p$, and angular velocity path, $\dot{\phi}_p$ to be zero at all times. We choose the ball position path, $\theta_p$, to be a quintic polynomial in time as in equation (17) and ball velocity path, $\dot{\theta}_p$, as the derivative of the polynomial as in equation (18).

$$\theta_p = At^5 + Bt^4 + Ct^3 + Dt^2 + Et + F \tag{17}$$

and

$$\dot{\theta}_p = 5At^4 + 4Bt^3 + 3Ct^2 + 2Dt + E \tag{18}$$

If one defines initial and final (or even a knot point in between) positions, velocities, and accelerations, one can solve for the coefficients, $A$, $B$, $C$, $D$, $E$, and $F$. The choice of a quintic polynomial is to ensure smooth accelerations and thus smooth torques. Sciavicco and Siciliano [8] suggest using a polynomial joint trajectory when no specific trajectory is desired between points. Eventually we will determine trajectories which respect Ballbot's dynamic constraints (e.g. the body must lean forward and the ball must move backward before Ballbot moves forward while the quintic polynomial moves forward immediately).

Equation (16) defines a planar controller. The real Ballbot and the SimMechanics simulation are both in 3D, so in practice we use two independent planar controllers. The first is for $x$ ball movement, $\theta_x$ and $\dot{\theta}_x$, and $x$ body movement, namely pitch and pitch rate, $\phi_y$ and $\dot{\phi}_y$. The second is for $y$ ball movement, $\theta_y$ and $\dot{\theta}_y$, and $y$ body movement, $\phi_x$ and $\dot{\phi}_x$. Two independent planar controllers will be most effective when the body is near its equilibrium point and coupling between $x$ and $y$ dynamics is small.

Each arm degree of freedom (2 DoF for each arm) is controlled independently with a PD controller defined in equation (19)

$$\tau_a = P_\psi\left(\psi - \psi_p\right) + D_\psi\left(\dot{\psi} - \dot{\psi}_p\right) \tag{19}$$

where $P_\psi$ and $D_\psi$ are the gains, and $\psi_p$ and $\dot{\psi}_p$ are desired arm paths defined by quintic polynomials similar to equations (17) and (18). This controller requires solving the inverse kinematics problem for $\psi_p$ and $\dot{\psi}_p$ given a goal point in space. The inverse kinematics problem is not solved here because this work deals only with simple movements (e.g. lifting the arms in the $x$ direction).

## 4 Unified Balancing and Arm Controller

There might be some cases where coordinating roller and arm movement is desirable. One example is when lifting a heavy object causes the Ballbot to begin to lose balance. With the independent controllers of the previous sections, the arms might continue to lift and cause Ballbot to fall. With a unified controller, the arms might lower the heavy object to regain balance.

Again we turn to LQR using the full state vector $x = \left[\theta\ \phi\ \psi\ \dot{\theta}\ \dot{\phi}\ \dot{\psi}\right]^\mathsf{T}$, and the input vector $u = [\tau_r\ \tau_a]^\mathsf{T}$ and linearizing equation (14) about the zero state. This time $Q$ is a 6x6 matrix of weights for the states and $R$ is a 2x2 matrix of the weights for the applied torques. The $Q$ and $R$ used to obtain the results that follow are:

$$Q = \begin{bmatrix} 3000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{bmatrix}$$

and

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

This means that the relative importance of the ball position, $\theta$, and the arm angle, $\psi$, is high and again the importance of the torque is low.

When merely balancing, an LQR controller optimized to operate around the standing position with arms down is effective. In situations where Ballbot must move its arms (reach for something on a shelf, or carry a drink, for instance) this controller is not sufficient. First, LQR is a linear controller, and the sines and cosines in the gravity terms when the arms move away from the sides of Ballbot are nonlinear. Second, the gains for moving the arms are optimized for movement around the zero position where gravity has little effect. When lifting the arms, gravity works against the motion, and the LQR derived controller does not apply enough torque to move the arms very far. To make up for this, we introduce gravity compensation [8] for the arms. The arm torque, $\tau_a$ is a combination of the torque, $\tau_{LQR}$, prescribed by the LQR gains (a feedback torque) and gravity (a feed forward torque).

$$\tau_a = \tau_{LQR} + m_a a_{cm} g \sin\psi \tag{20}$$

Gravity compensation fixes the two problems mentioned above. It essentially takes away the nonlinear gravity terms and allows control about non-zero arm positions.

This unified controller is also a planar controller, and in practice we use two planar controllers. In the previous section we assumed that the $x$ and $y$ dynamics are uncoupled when Ballbot is standing. Here the arms certainly have coupled dynamics in any out of plane movement. This controller is most effective when operating in the plane. Planar arm control is not a final solution. The unified controller of this section is intended to demonstrate the possible benefits of the unified approach vs. the independent roller and arm controller approach.

## 5 Performance

The purpose of these simulations was to:

- Tune an arm controller independent of the balancing controller. For a number of simple movements (lifting the arms in various directions) can the controller follow the desired path? How much torque is required?
- Observe the behavior of the balancing controller when the various arm motions are executed. Determine what problems exist with the balancing controller as a result of arm movement.
- Compare the behaviors of the independent controllers and the unified controller.
- Explore a few new control ideas.

One SimMechanics output is a visualization window, shown in Figure 9. The red parts of the figure are what the visualization window actually shows. The black outline of the Ballbot's body and the black dots representing the connection points of the arms are added for clarity. Red ellipsoids represent the bodies. The ellipsoids come from the inertia tensors of the bodies. The actual top of the Ballbot body is about 1.5 m whereas its ellipsoid does not extend that far. The arm ellipsoids extend further than the physical arms.

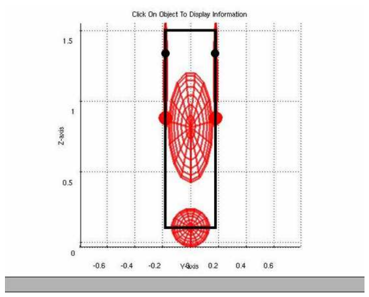
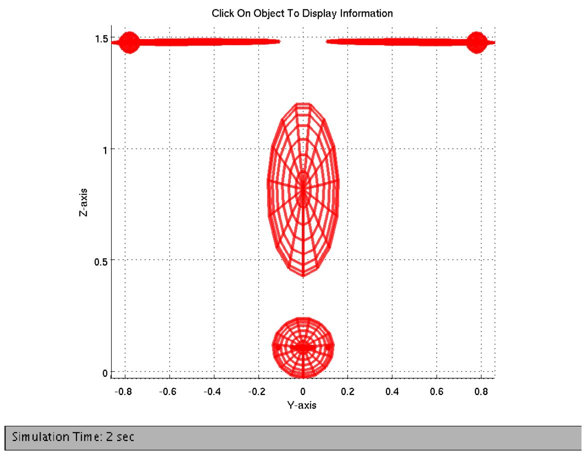


Figure 9: Example of SimMechanics Visualization Window

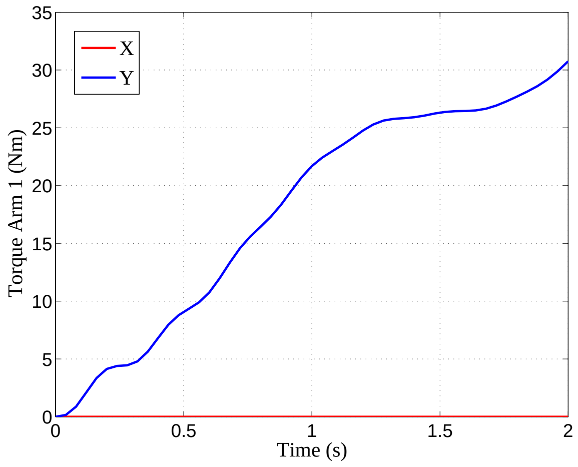
## 5.1 Arm Controller Performance

The basic design of the arms is an aluminum cylinder with a 1 kg hand. One test of the arm controller is to lift both arms up along Ballbot's sides (the $y$ direction) as seen in Figure 10a. This test was conducted using various hand masses to simulate what happens when Ballbot lifts objects. Note that lifting both arms in opposite directions will not affect Ballbot's balance, allowing us to test the arm controller while Ballbot has perfect balance. The results in Figure 10 are for 5 kg hands. Figure 10b shows that the peak torque required for this action is 30.76 Nm. The peak torque required for 1 kg hands is 7.37 Nm. Figure 10c-d shows that this controller follows the desired path very well.

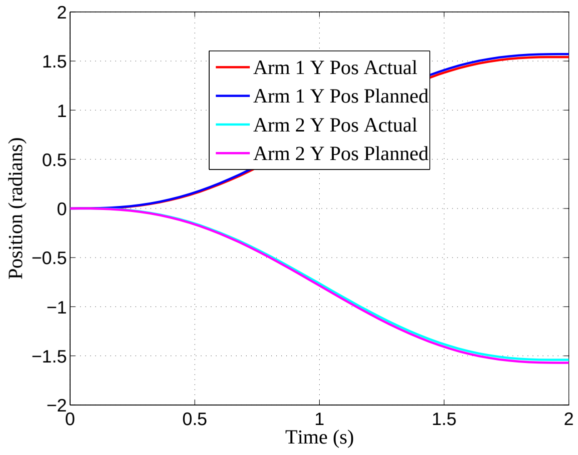
Obviously the arms cannot supply an infinite amount of torque. Figure 11 shows results from a simulation where a 15 Nm torque limit was put on the arm motors. Figure 11a-b shows that when the motor saturates, the arms oscillate undamped (no friction is modeled) about a position less than the desired position.



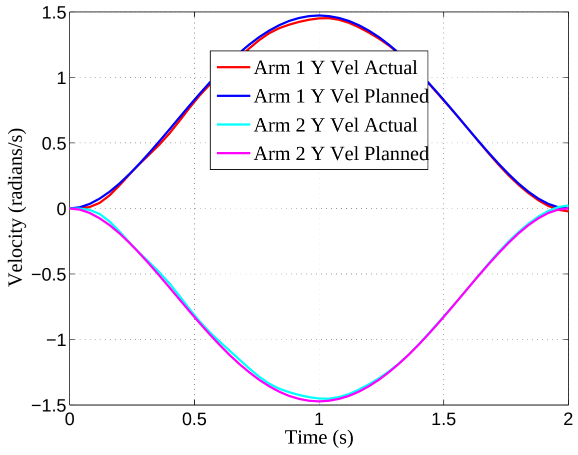
(a) Final Pose



(b) Torque vs. Time

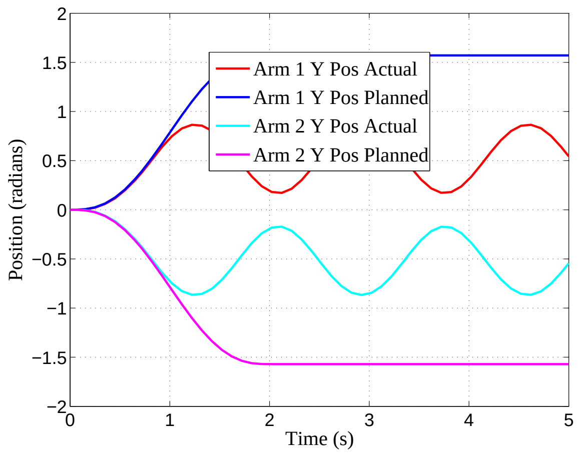


(c) Arm $y$ Angle vs. Time

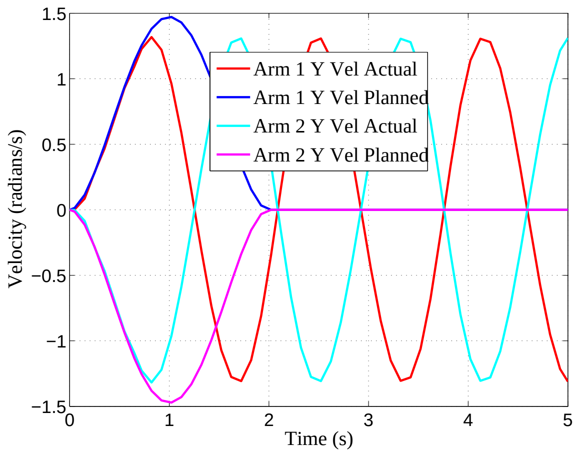


(d) Arm $y$ Angular Velocity vs. Time

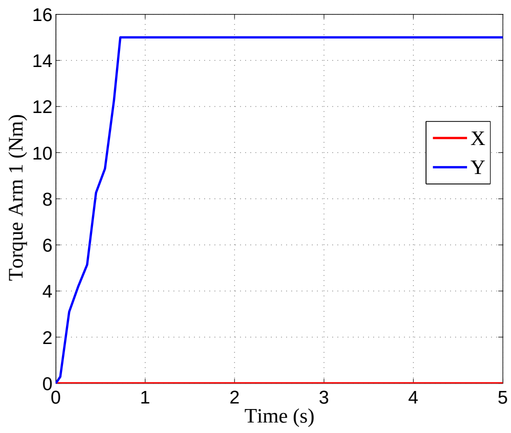
Figure 10: Both 5 kg Arms Raised in $y$ Direction



(a) Arm $y$ Angle vs. Time



(b) Arm $y$ Angular Velocity vs. Time



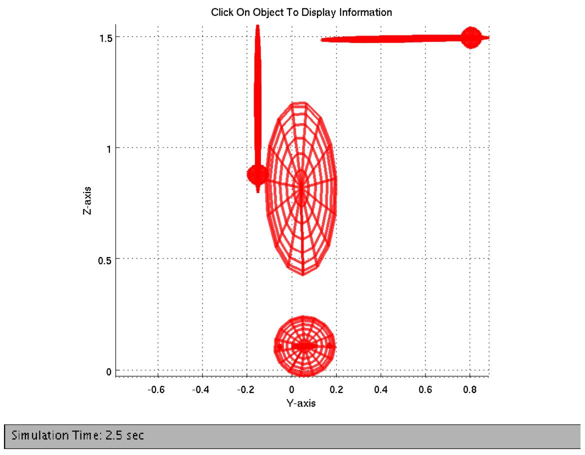
(c) Torque vs. Time

Figure 11: Both 5 kg Arms Raised in $y$ Direction with Saturated Torque

## 5.2 Balancing When Arms Move

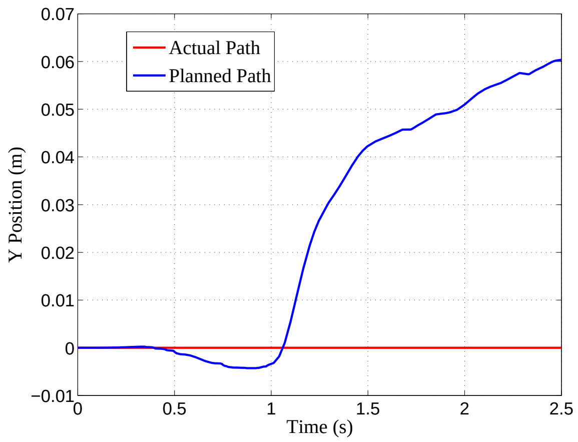
Figure 12a shows the final pose of Ballbot raising one arm in the $y$ direction. Note in Figure 12b-c that the ball moves to the right by about 6 cm and from Figure 12d-e that the body leans to the left by about 1.3 degrees. When Ballbot raises an arm, it settles into a leaning equilibrium that is different from the desired standing upright equilibrium where the body angle is zero. The arm movement causes an overall change in the center of mass which changes the equilibrium position. Figure 12f-g shows that the arm controller follows the planned path well.

Figure 13a-f shows results of the same arm lifting simulation except with heavier 5 kg arms. This models what might happen when Ballbot picks up a load. In this case the station keeping/balancing controller fails. Figure 13a-b shows increasing ball oscillation and Figure 13c-d shows increasing body oscillation. As Ballbot loses its balance, the arm controller is not as effective as seen near the end of Figure 13e-f. The increased arm mass causes a larger change is the leaning equilibrium. Eventually the ball and body are far enough away from the desired standing upright equilibrium that the station keeping/balancing controller fights back. This causes the ball and body to oscillate between the leaning and standing upright equilibria and finally fall.

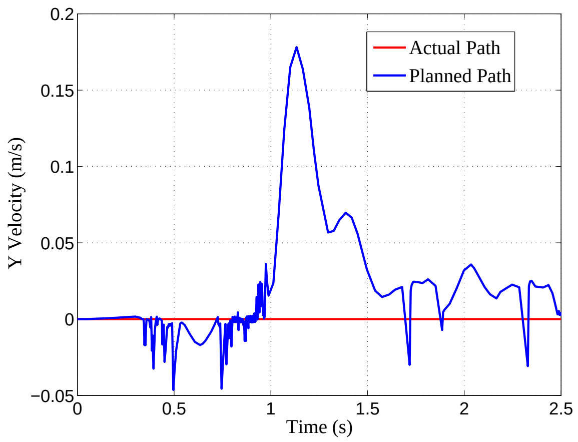


(a) Final Pose

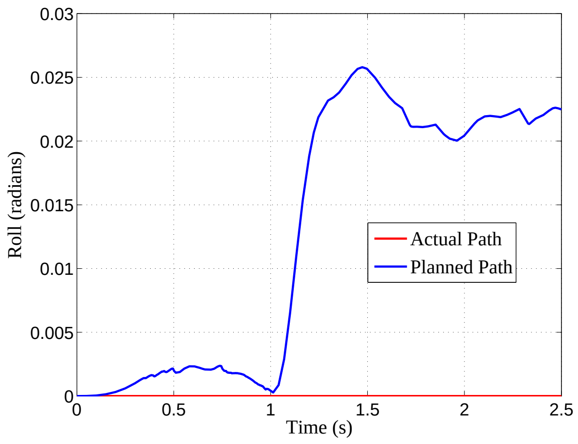
Figure 12: One 1 kg Arm Raised in $y$ Direction



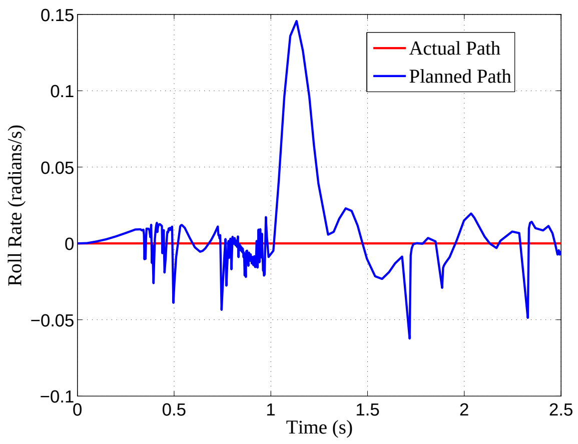
(b) $y$ Position vs. Time



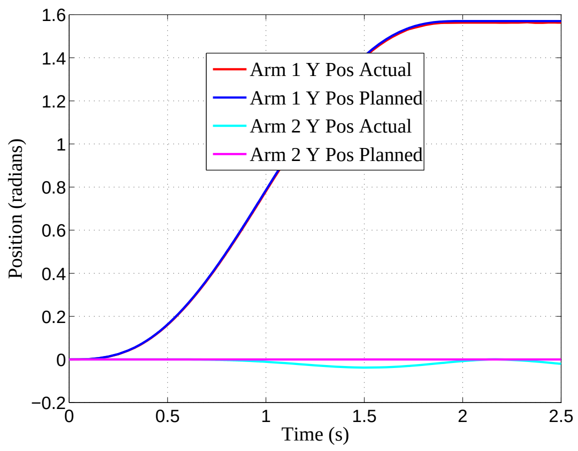
(c) $y$ Velocity vs. Time



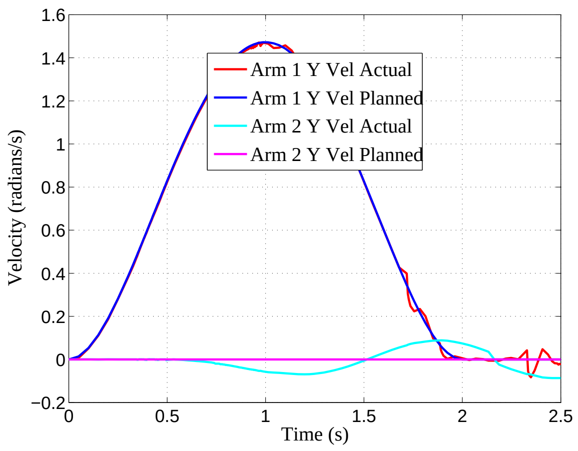
(d) Roll vs. Time



(e) Roll Rate vs. Time

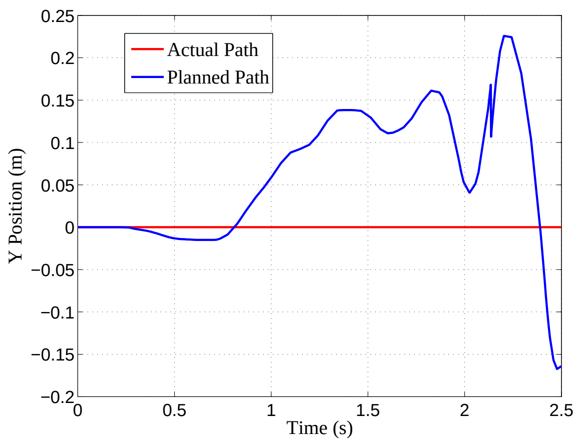


(f) Arm $y$ Angle vs. Time

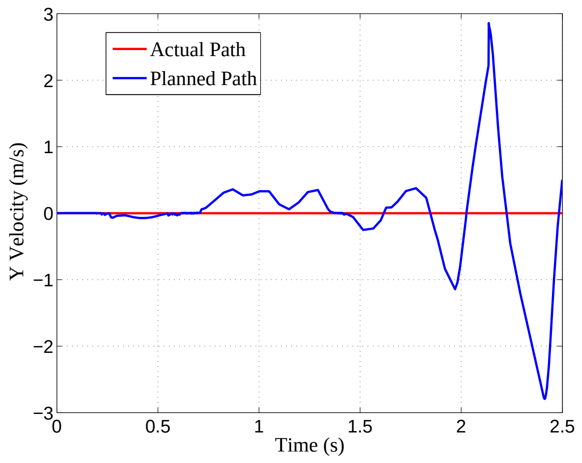


(g) Arm $y$ Angular Velocity vs. Time

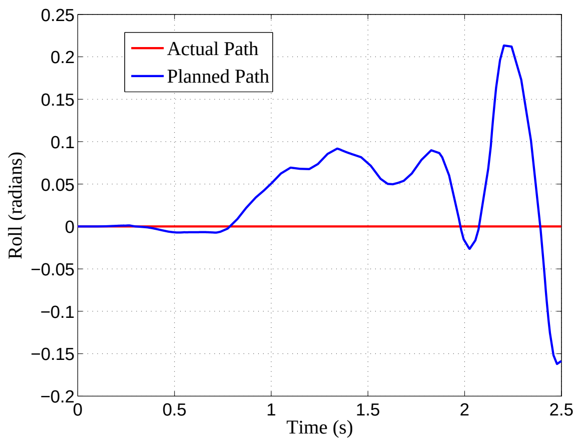
Figure 12: One 1 kg Arm Raised in $y$ Direction



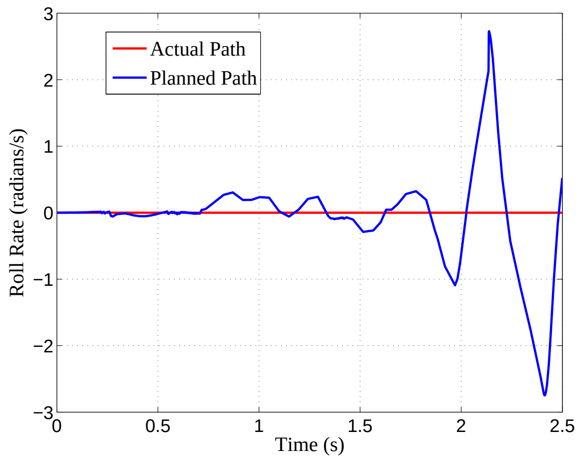
(a) $y$ Position vs. Time



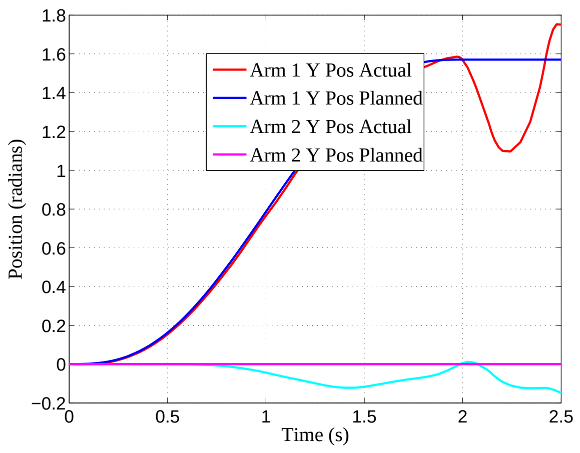
(b) $y$ Velocity vs. Time



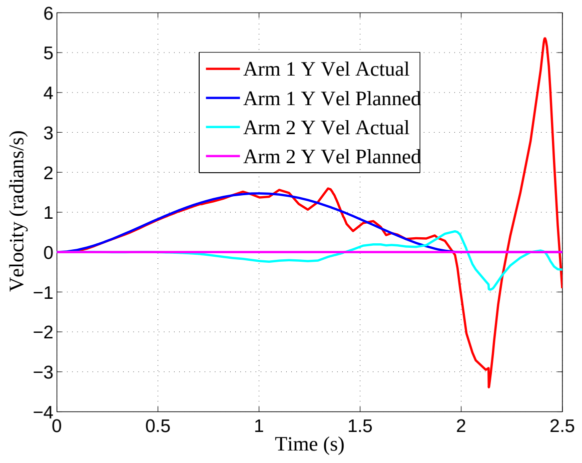
(c) Roll vs. Time



(d) Roll Rate vs. Time



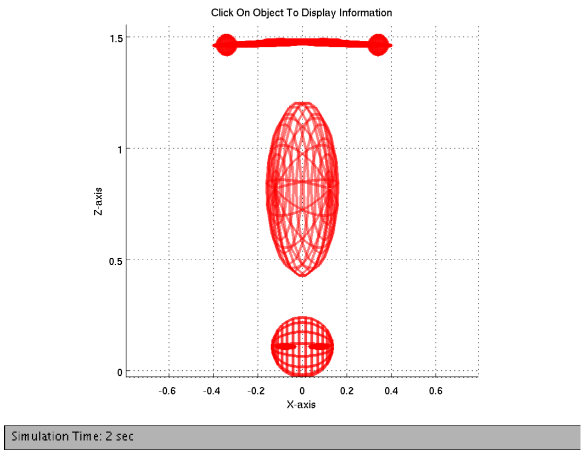
(e) Arm $y$ Angle vs. Time



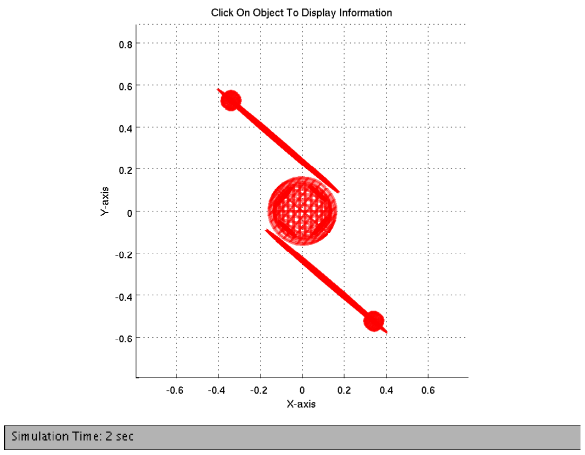
(f) Arm $y$ Angular Velocity vs. Time

Figure 13: One 5 kg Arm Raised in $y$ Direction

Raising the arms in opposite $x$ directions causes Ballbot to rotate about its vertical axis (a yaw). Figure 14a shows this pose in the $xz$ plane. The $xy$ plane view (from above) of Figure 14 makes the yawing apparent, as Ballbot began the simulation facing in the $x$ direction. Keep in mind that no friction is modeled between the ground and ball or between the body and ball. In the real case where there is friction, there might not be any yawing, or at least there will be less yawing.



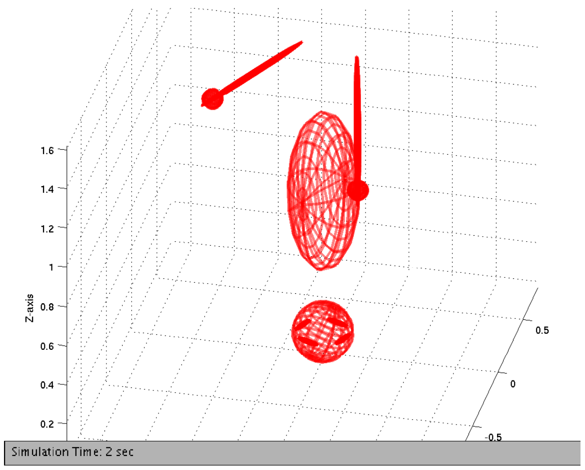
(a) Final Pose: $xz$ View



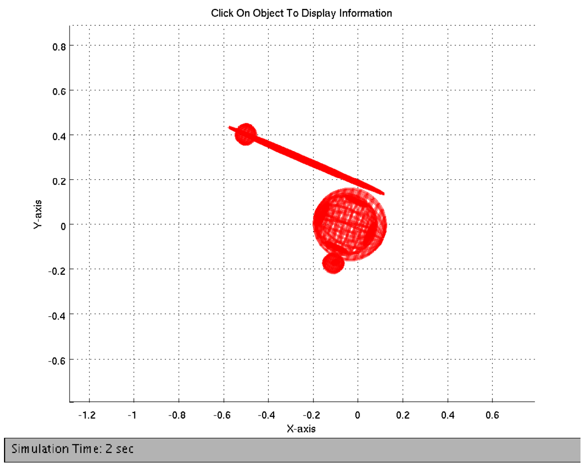
(b) Final Pose: $xy$ View

Figure 14: 5 kg Arms Raised in Opposite $x$-Directions

Figure 15a-b is the final pose of Ballbot raising one arm in the $x$ direction and out in the $y$ direction. This causes both a yaw (Figure 15b) and the leaning equilibrium (Figure 15b-f and i-l) seen earlier. Note that the leaning is in both the $x$ and $y$ directions. The arm controller performs well (Figure 15g-h and m-n).

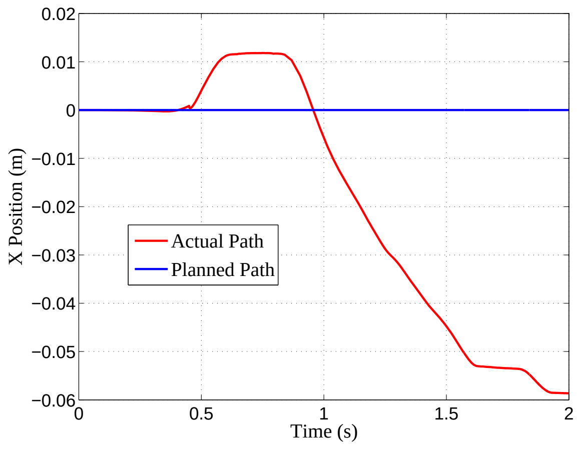


(a) Final Pose: 3D View

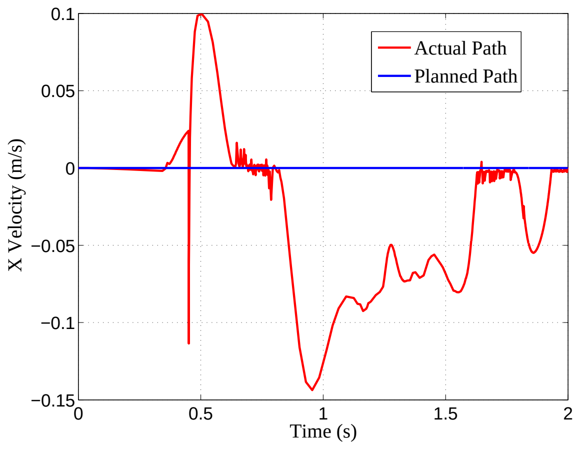


(b) Final Pose: $xy$ View

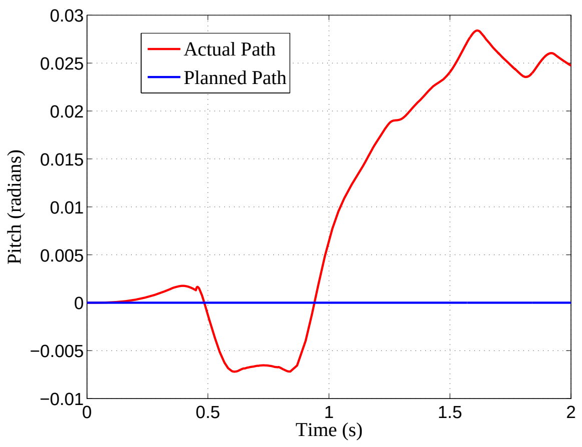
Figure 15: 1 kg Arm Raised in $x$ and $y$ Directions



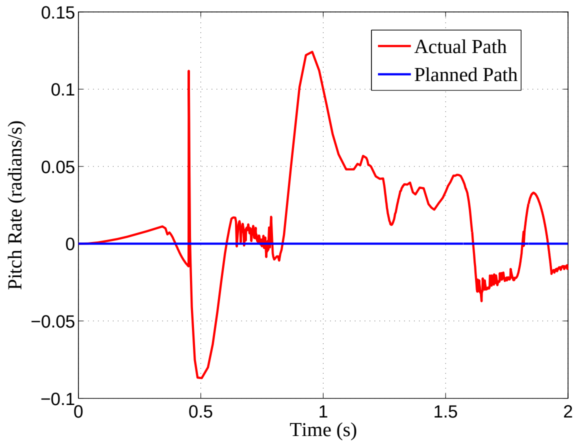
(c) $x$ Position vs. Time



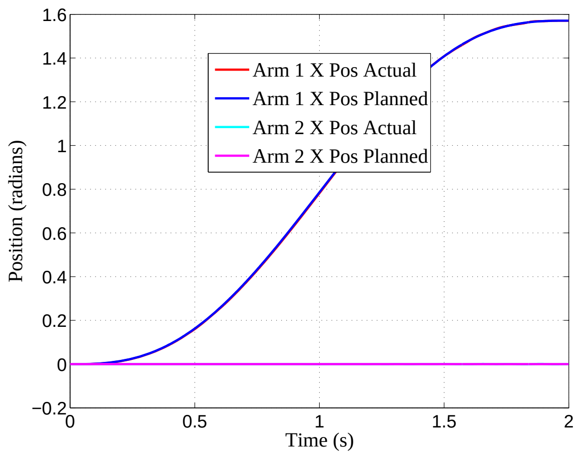
(d) $x$ Velocity vs. Time



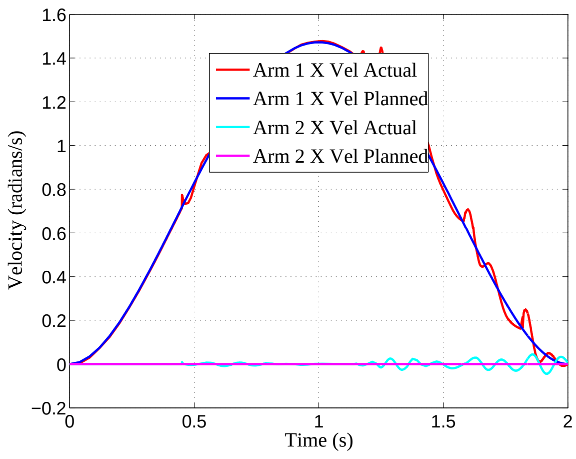
(e) Pitch vs. Time



(f) Pitch Rate vs. Time

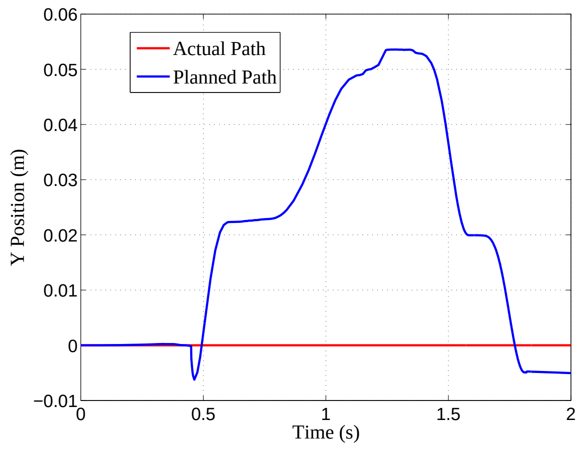


(g) $x$ Arm Angle vs. Time

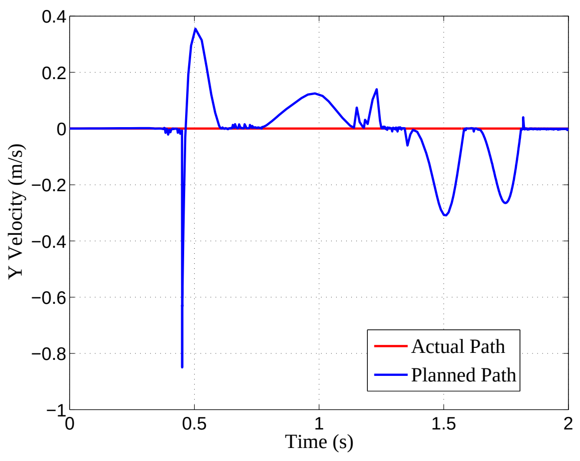


(h) $x$ Arm Angular Velocity vs. Time

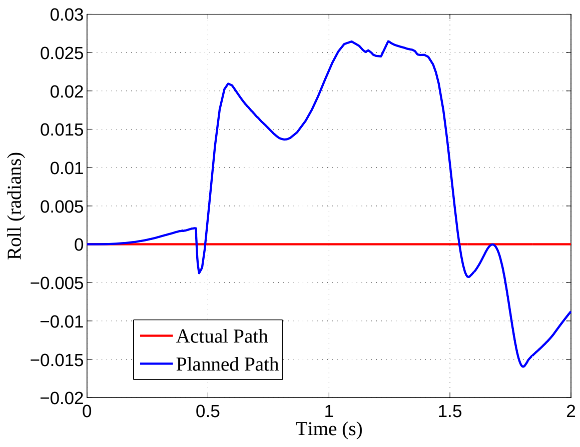
Figure 15: 1 kg Arm Raised in $x$ and $y$ Directions



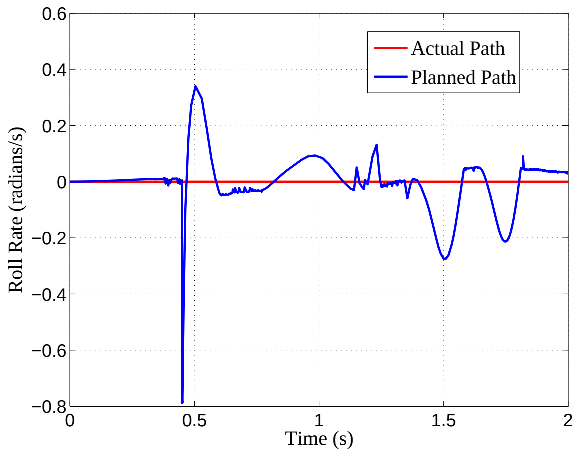
(i) $y$ Position vs. Time



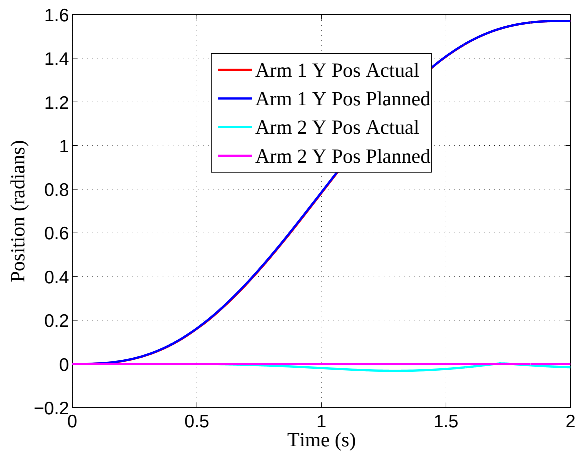
(j) $y$ Velocity vs. Time



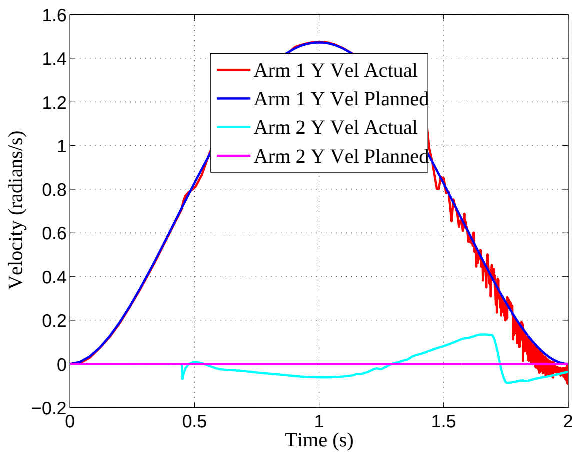
(k) Roll vs. Time



(l) Roll Rate vs. Time



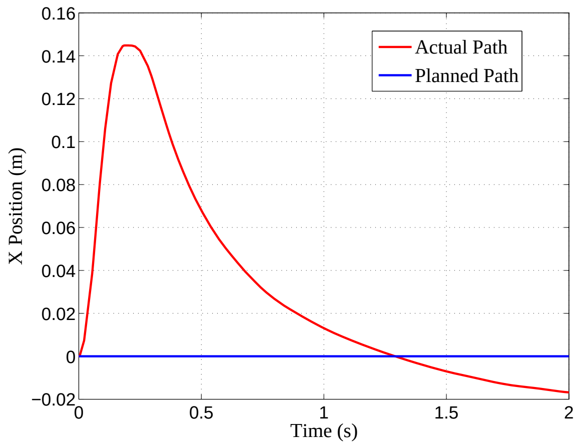
(m) $y$ Arm Angle vs. Time



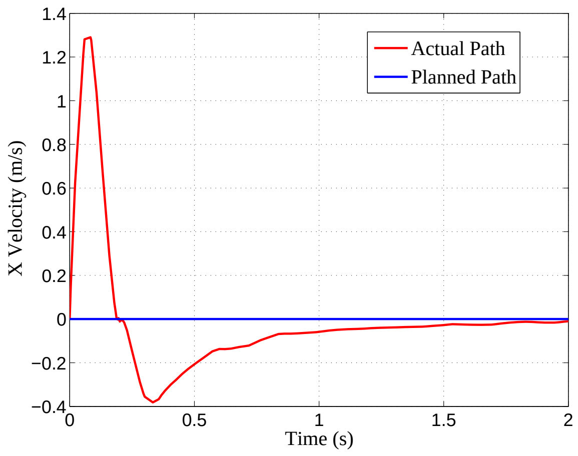
(n) $y$ Arm Angular Velocity vs. Time

Figure 15: 1 kg Arm Raised in $x$ and $y$ Directions

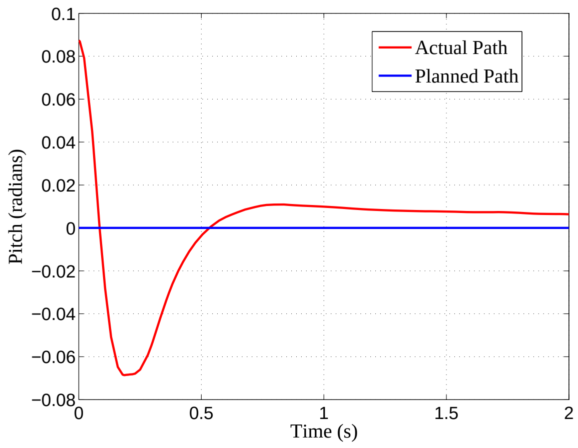
Figure 16a-j shows results of Ballbot raising one arm in the $y$ direction (as in Figure 12a) but starting off balance by 5 degrees of roll and pitch (rotations about $x$ and $y$). Note the initial roll (Figure 16g) and pitch (Figure 16c). Ballbot recovers its balance nicely and establishes its leaning equilibrium. It finishes with non-zero $x$ and $y$ position and pitch and roll and zero $x$ and $y$ velocity and pitch rate and roll rate. The arm controller is initially a bit noisy (Figure 16j), but eventually follows the planned path closely.



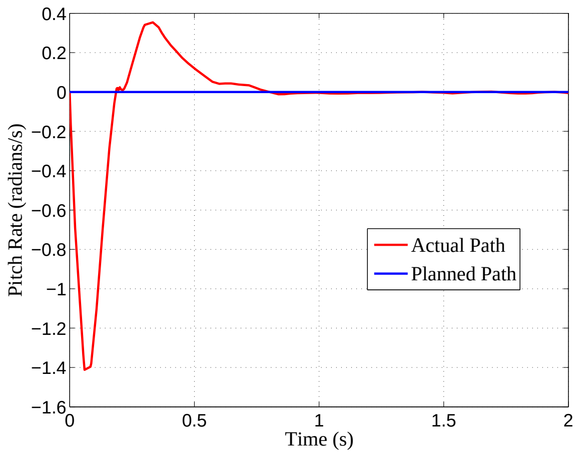
(a) $x$ Position vs. Time



(b) $x$ Velocity vs. Time

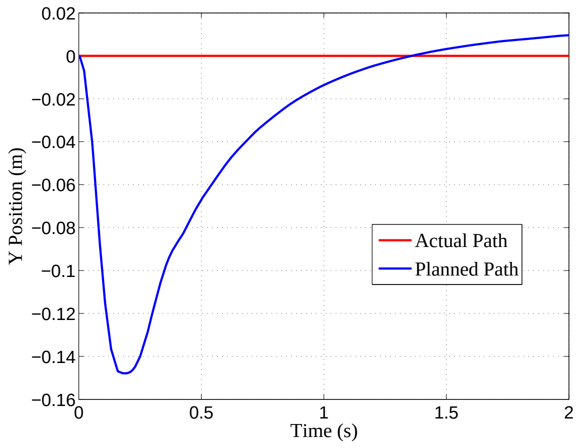


(c) Pitch vs. Time

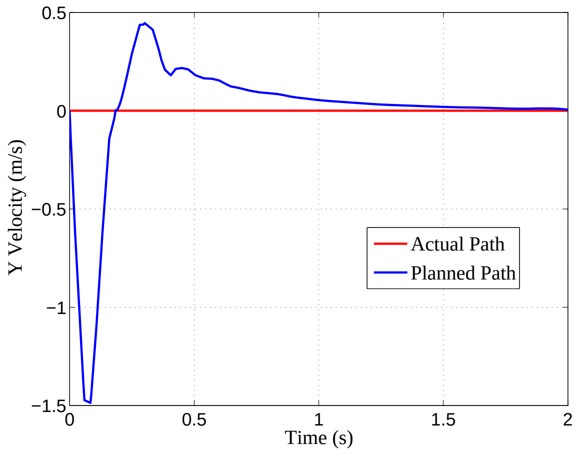


(d) Pitch Rate vs. Time

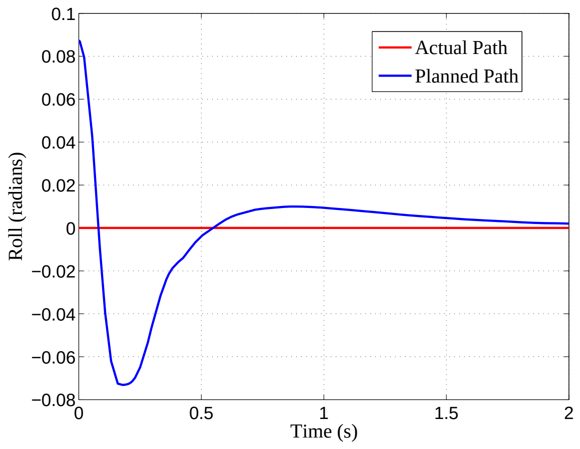
Figure 16: 1 kg Arms Raised in $y$ Direction From Off Balance



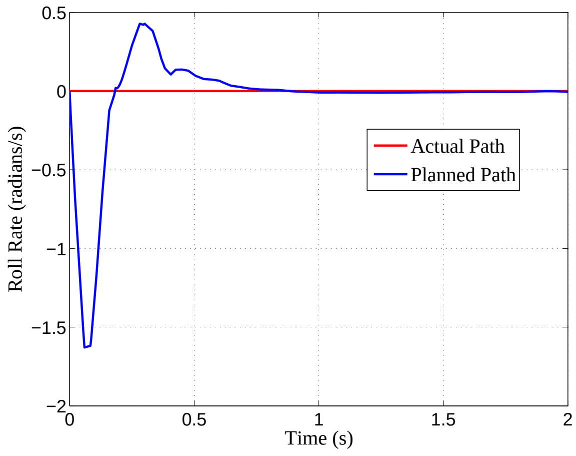
(e) $y$ Position vs. Time



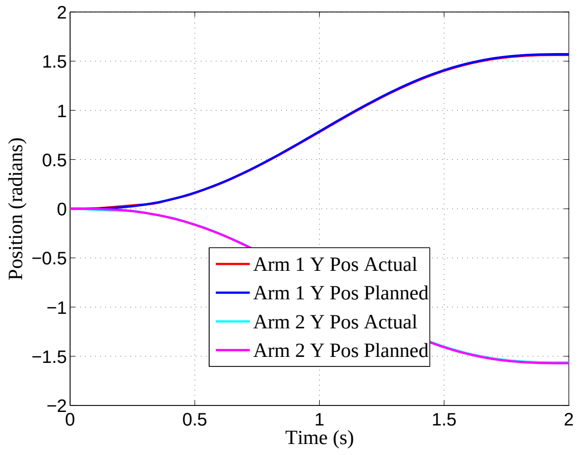
(f) $y$ Velocity vs. Time



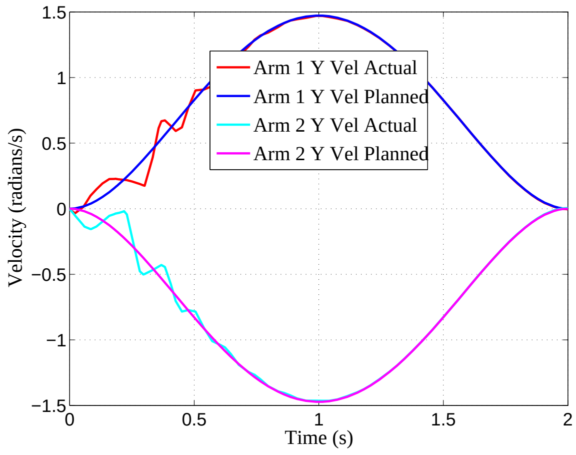
(g) Roll vs. Time



(h) Roll Rate vs. Time



(i) $y$ Arm Angle vs. Time

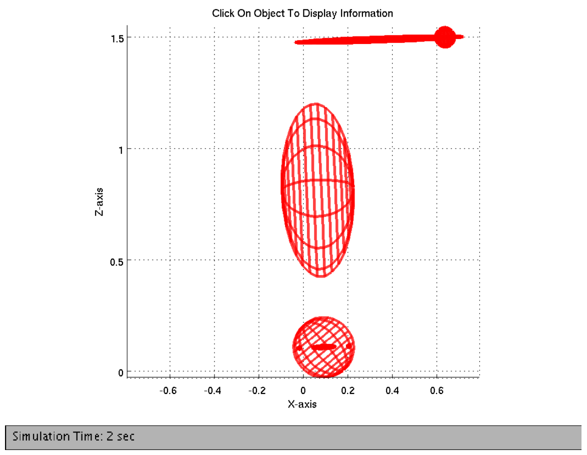


(j) $y$ Arm Angular Velocity vs. Time

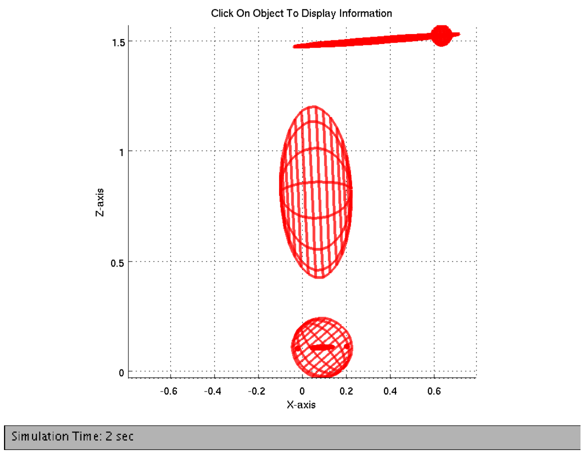
Figure 16: 1 kg Arms Raised in $y$ Direction From Off Balance

## 5.3 Compare Independent to Unified Control Approach

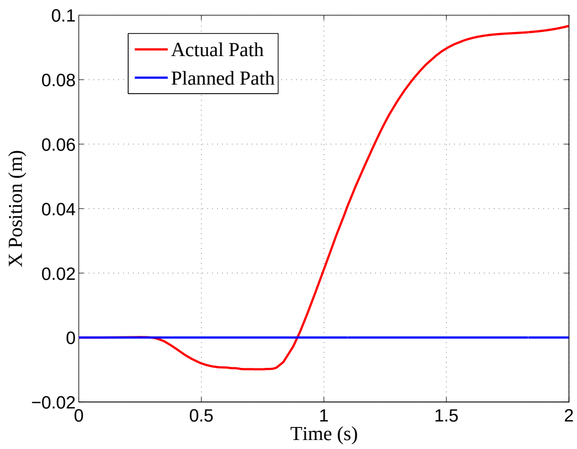
Figure 17a-b shows the final pose of Ballbot lifting its arms in the $x$ direction using the independent controllers and the unified controller. The controllers show very similar performance (Figure 17c-j) in that Ballbot assumes the leaning equilibrium in both cases. With both control strategies the ball moves about 10 cm to the right and leans about 2.3 degrees to the left. We can see from Figure 17k-n that the independent arm controller follows the planned path slightly better than the unified controller does, while the unified controller requires slightly less torque (Figure 17o-p).



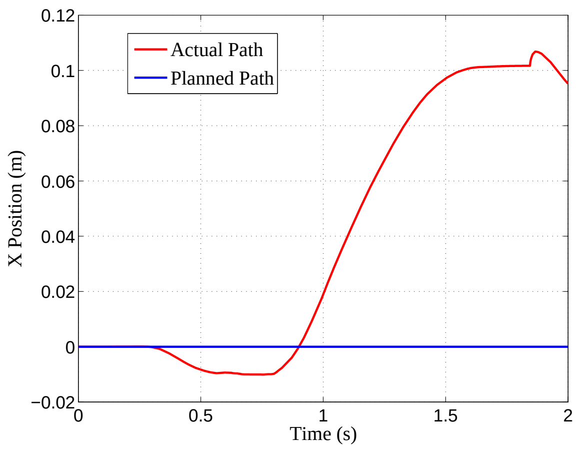
(a) Final Pose: Independent Controllers



(b) Final Pose: Unified Controller

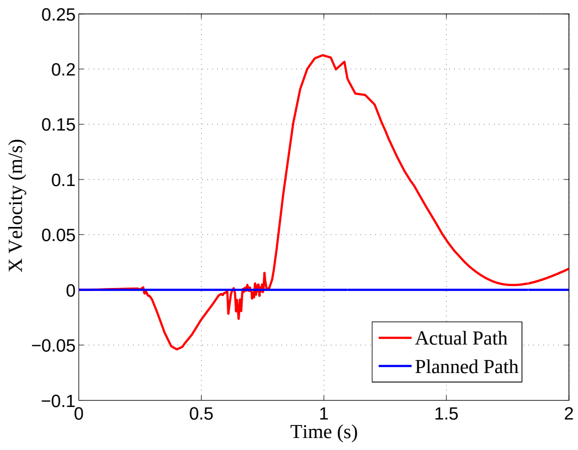


(c) $x$ Position vs. Time: Independent

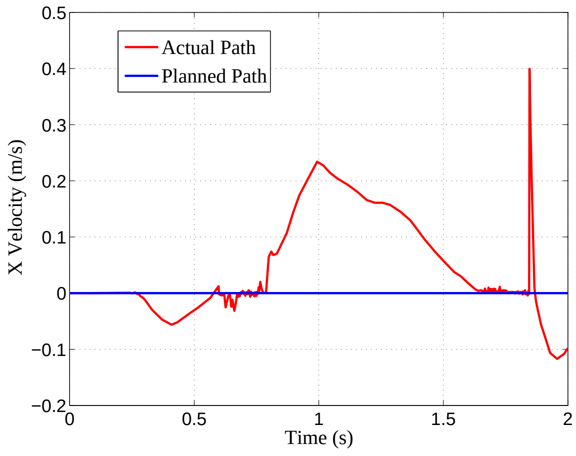


(d) $x$ Position vs. Time: Unified

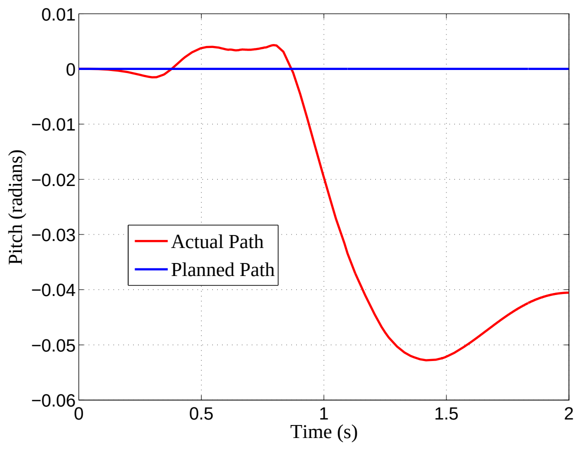
Figure 17: Both 1kg Arms Raised in $x$-Direction: Compare Independent and Unified Control



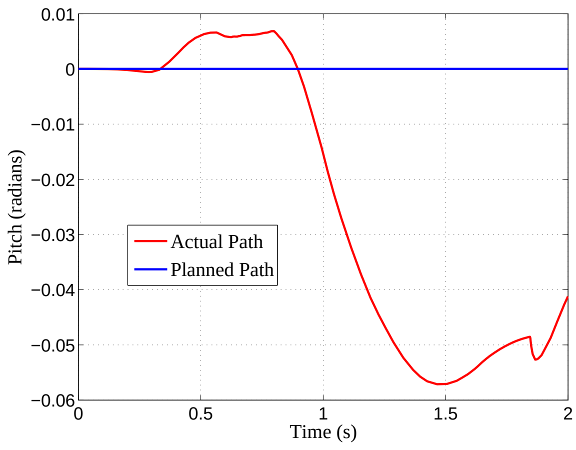
(e) $x$ Velocity vs. Time: Independent



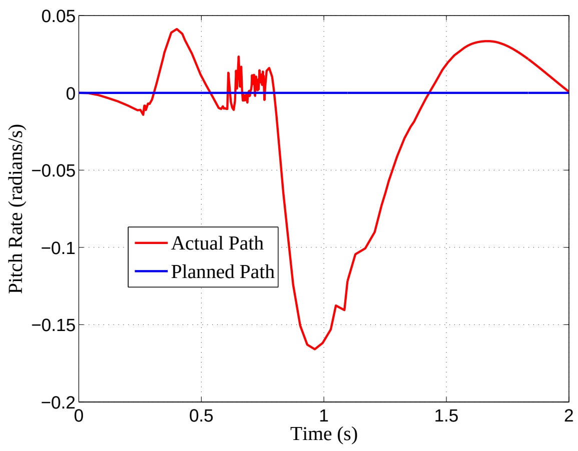
(f) $x$ Velocity vs. Time: Unified



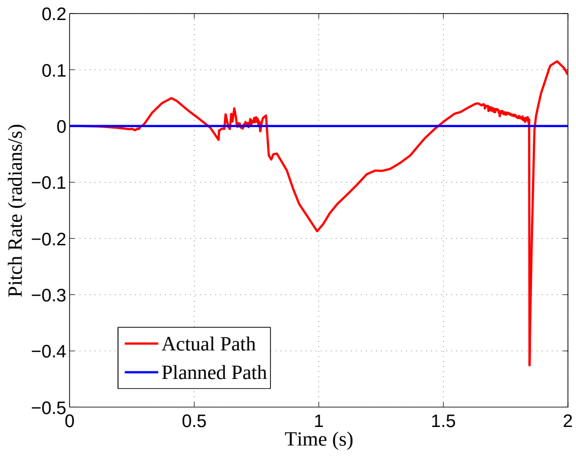
(g) Pitch vs. Time: Independent



(h) Pitch vs. Time: Unified

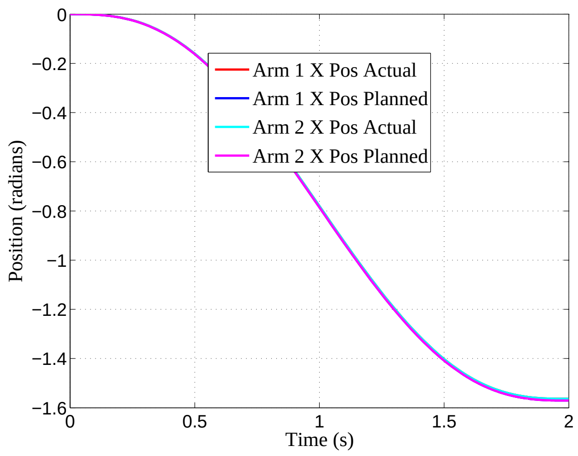


(i) Pitch Rate vs. Time: Independent

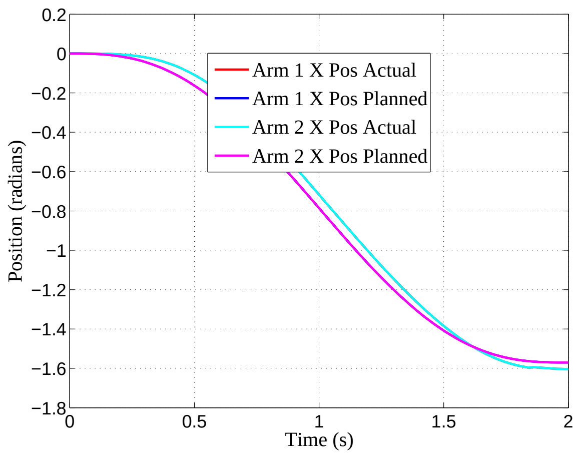


(j) Pitch Rate vs. Time: Unified

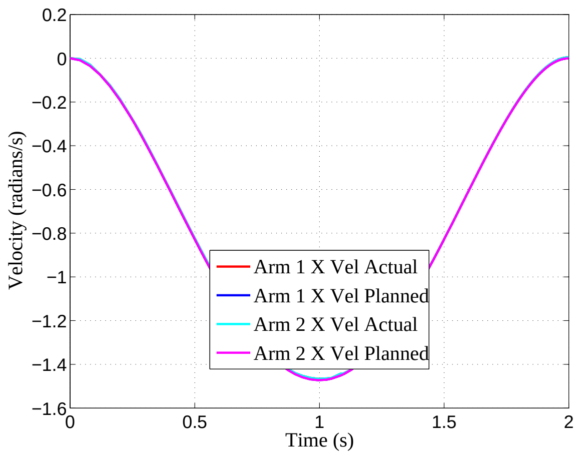
Figure 17: Both 1kg Arms Raised in $x$-Direction: Compare Independent and Unified Control



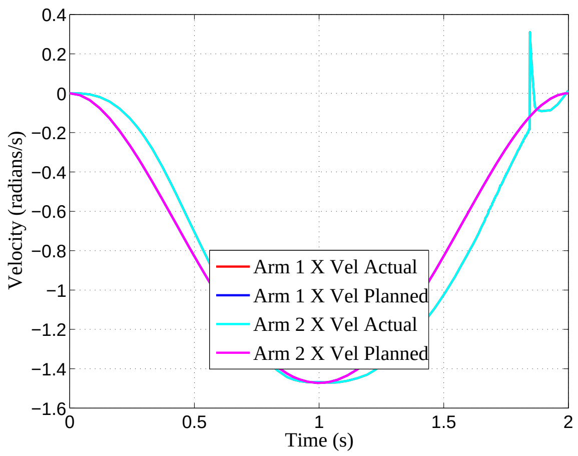
(k) $x$ Arm Angle vs Time: Independent



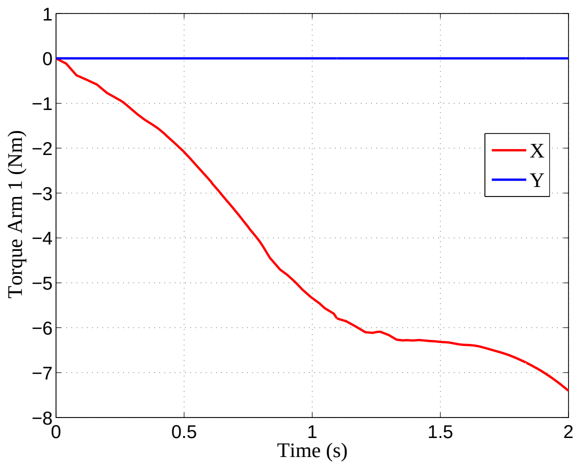
(l) $x$ Arm Angle vs. Time: Unified



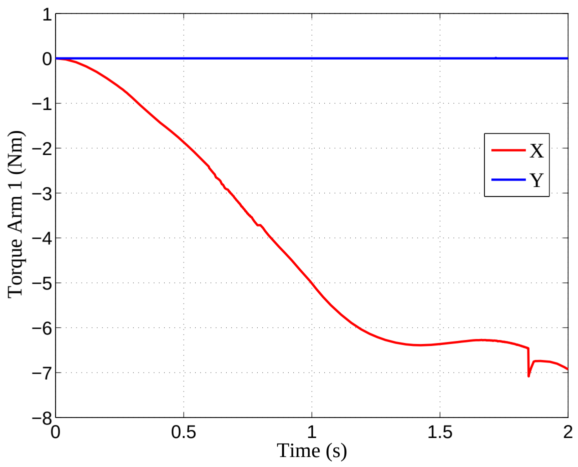
(m) $x$ Arm Ang. Vel. vs. Time: Independent



(n) $x$ Arm Ang. Vel. vs. Time: Unified



(o) Arm Torque vs. Time: Independent

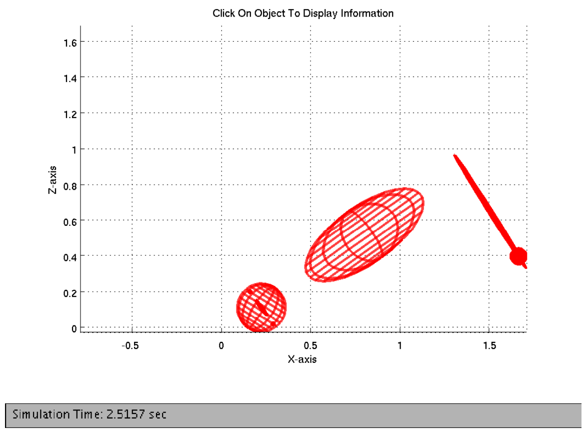


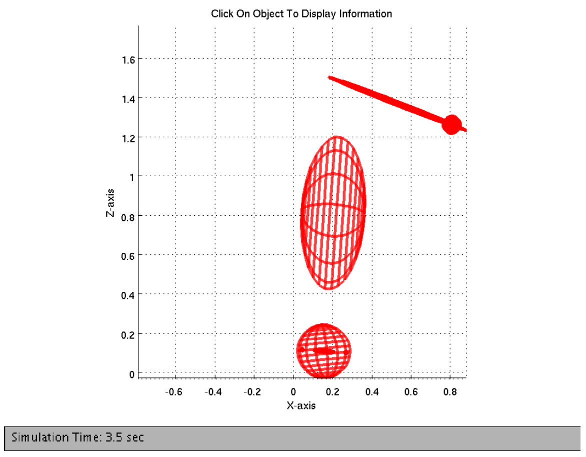
(p) Arm Torque vs. Time: Unified

Figure 17: Both 1kg Arms Raised in $x$-Direction: Compare Independent and Unified Control

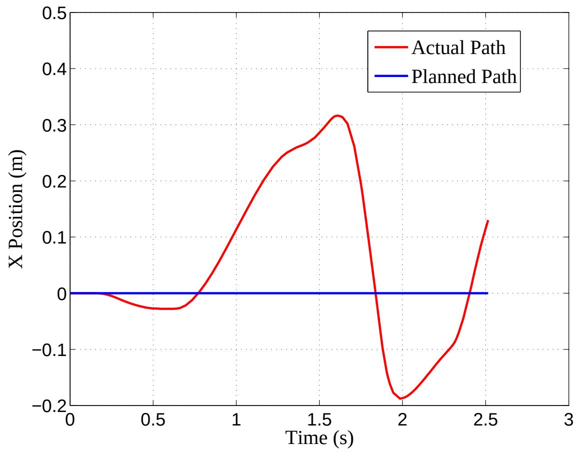
When executing the same x lifting motion with heavier 5 kg arms, the results are different. This time the independent controller (Figure 18a) fails almost immediately, and the unified controller (Figure 18b) balances for longer. The independent arm controller follows the planned path closely and uses more arm torque (Figure 18k, m, and o) while the unified controller deviates from the planned arm path and uses less arm torque (Figure 18l, n, and p) in order to stay balancing for longer. The unified controller has "decided" to abandon the goal of moving the arms to a desired position in favor of moving the arms to help balance.

The next comparison is recovering from a 16 degree initial body pitch (Figure 19). It is obvious from Figure 19c-l that the unified controller is able to recover and the independent controllers are not. The independent arm controller attempts to keep the arms at Ballbot's sides and uses small amounts of torque (Figure 19m, o, and q), while the unified controller swings the arms to help balance using more torque (Figure 19n, p, and r).

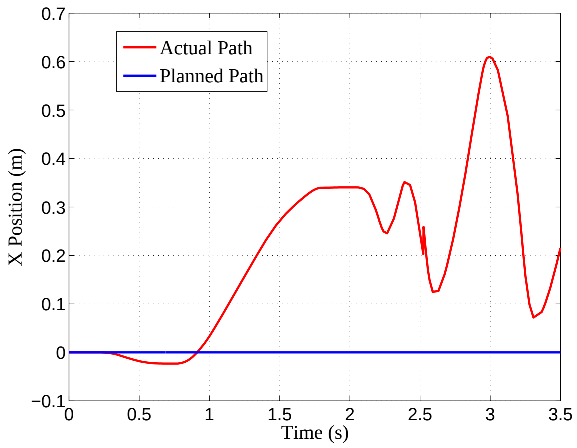

Click On Object To Display Information
Z-axis
X-axis
Simulation Time: 2.5157 sec
(a) Final Pose: Independent Controllers



(b) Final Pose: Unified Controller

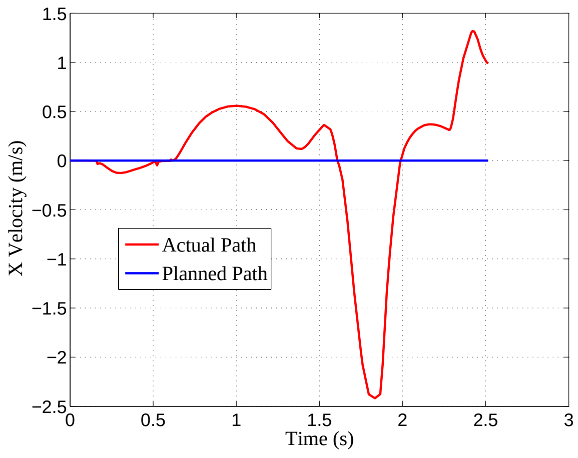


(c) $x$ Position vs. Time: Independent

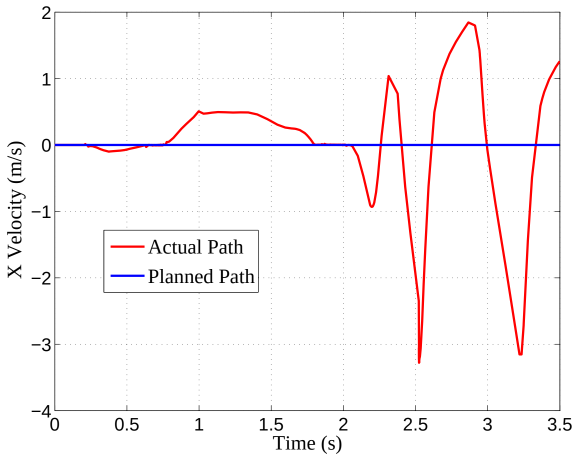


(d) $x$ Position vs. Time: Unified

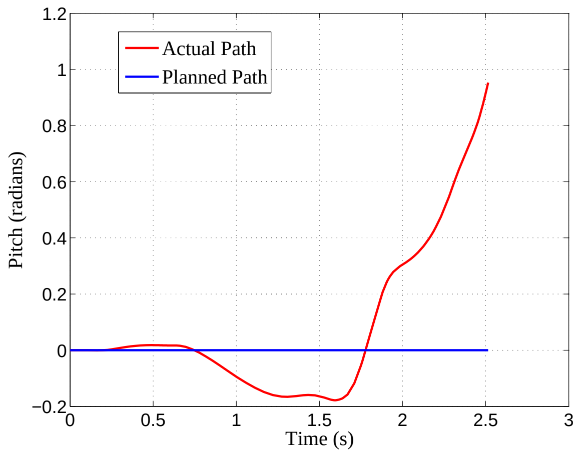
Figure 18: Both 5kg Arms Raised in $x$ Direction: Compare Independent and Unified Control



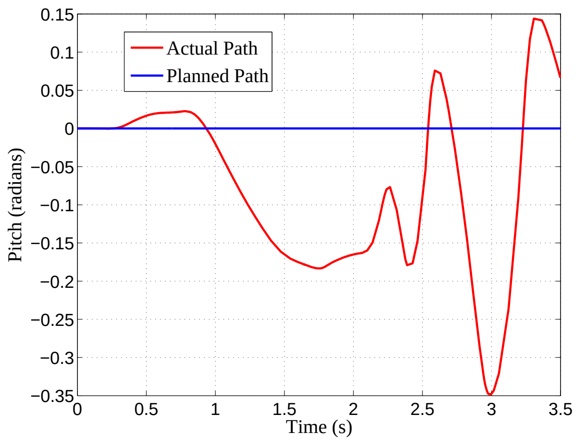
(e) $x$ Velocity vs. Time: Independent



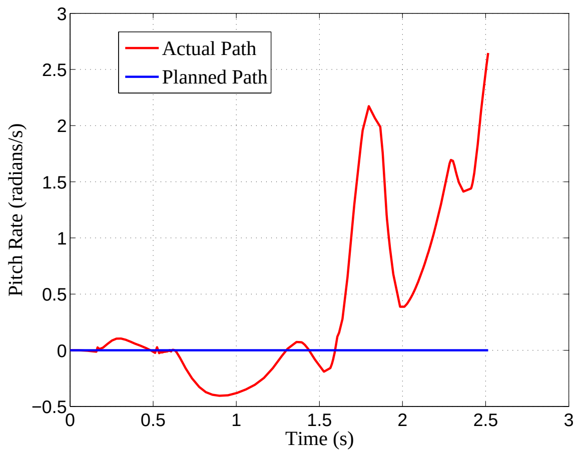
(f) $x$ Velocity vs. Time: Unified



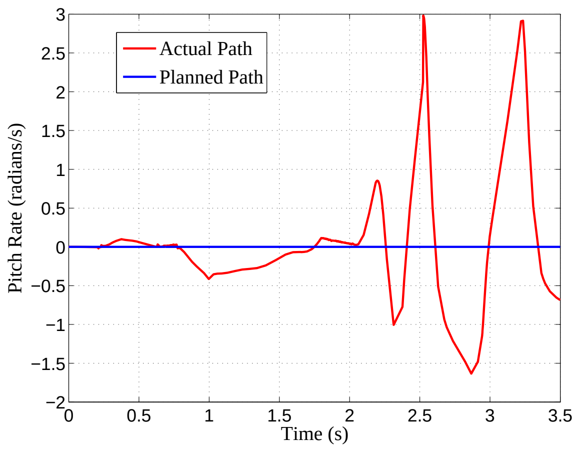
(g) Pitch vs. Time: Independent



(h) Pitch vs. Time: Unified

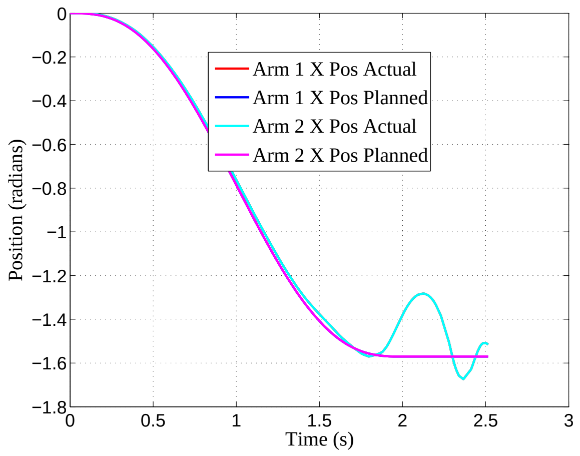


(i) Pitch Rate vs. Time: Independent

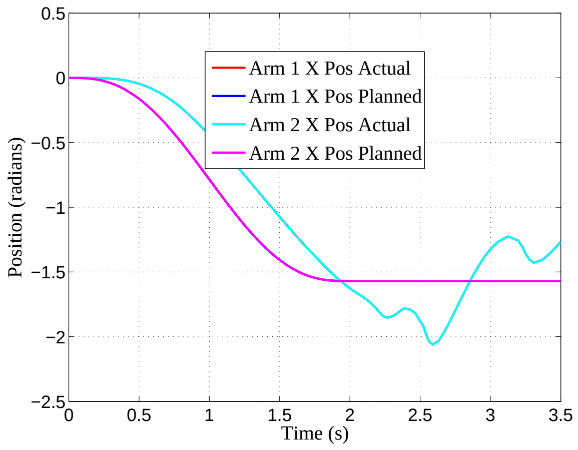


(j) Pitch Rate vs. Time: Unified

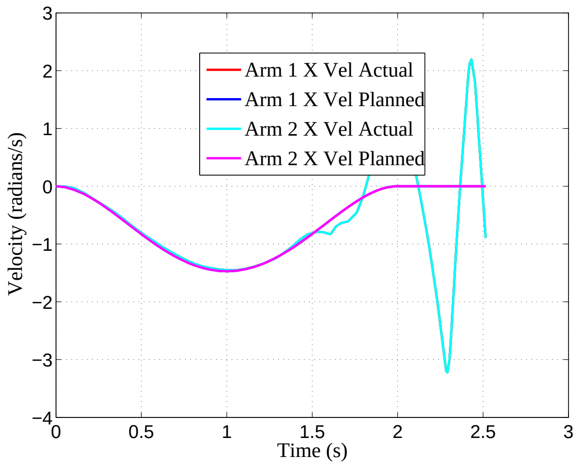
Figure 18: Both 5kg Arms Raised in $x$ Direction: Compare Independent and Unified Control



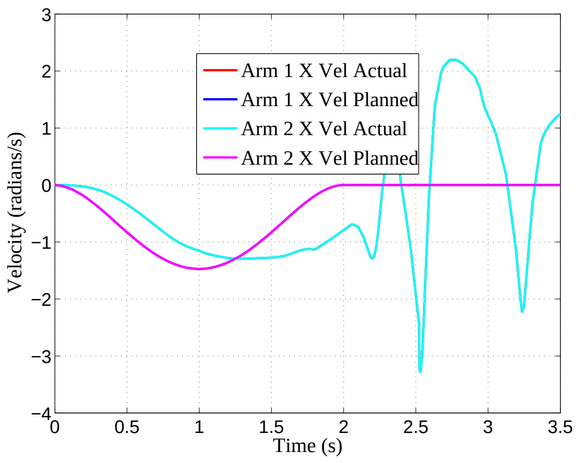
(k) $x$ Arm Angle vs. Time: Independent



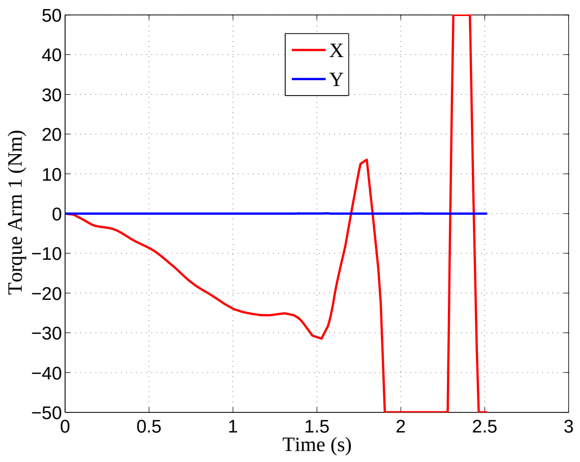
(l) $x$ Arm Angle vs. Time: Unified



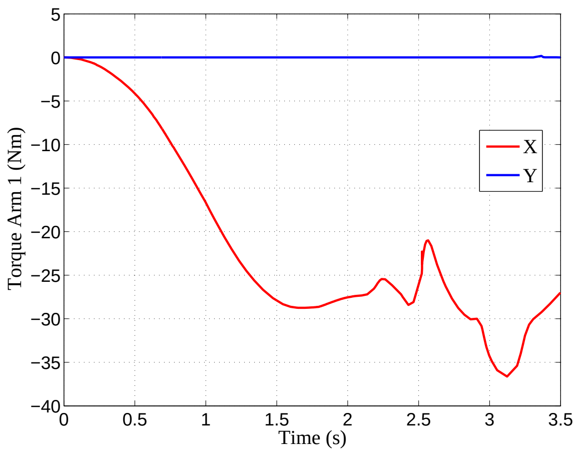
(m) $x$ Arm Ang. Vel. vs. Time: Independent



(n) $x$ Arm Ang. Vel. vs. Time: Unified

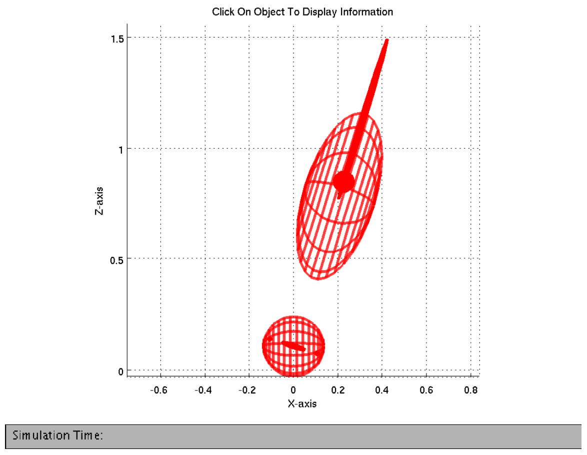


(o) Arm Torque vs. Time: Independent

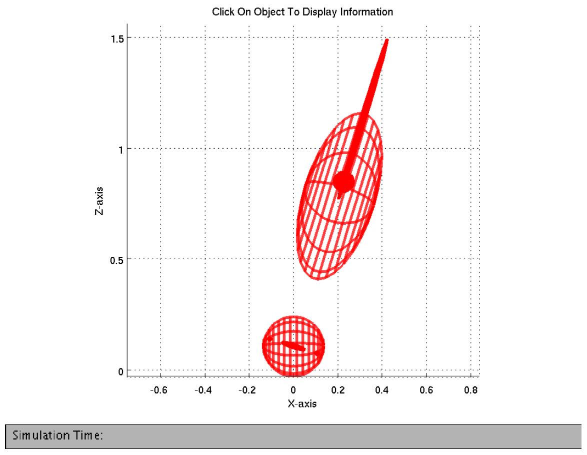


(p) Arm Torque vs. Time: Unified

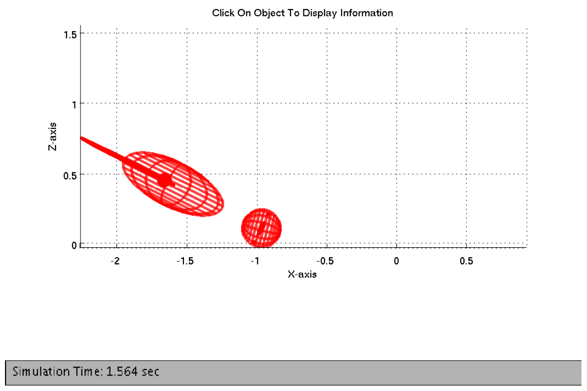
Figure 18: Both 5kg Arms Raised in $x$ Direction: Compare Independent and Unified Control



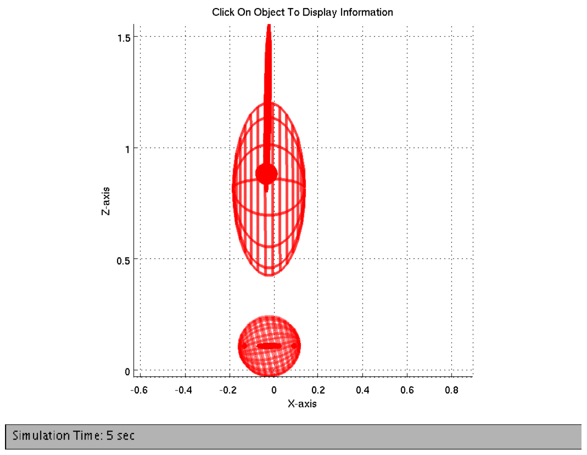
(a) Initial Pose: Independent Controllers



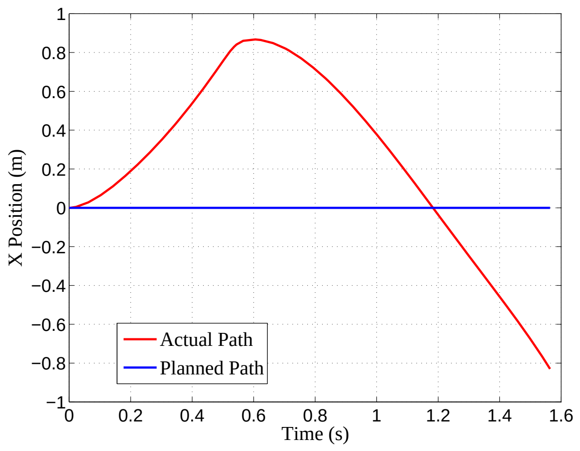
(b) Initial Pose: Unified Controller


Click On Object To Display Information
Z-axis
X-axis
Simulation Time: 1.564 sec

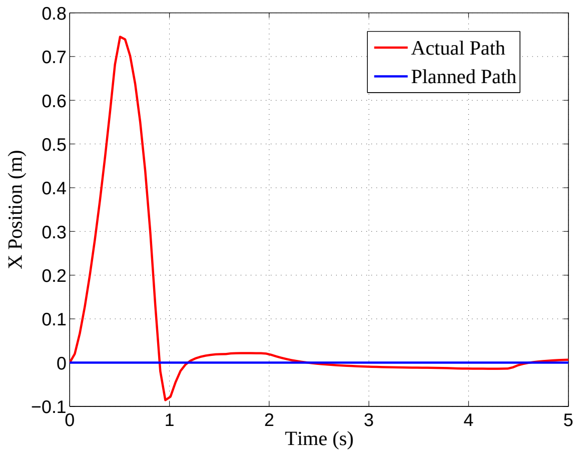
(c) Final Pose: Independent Controllers



(d) Final Pose: Unified Controller

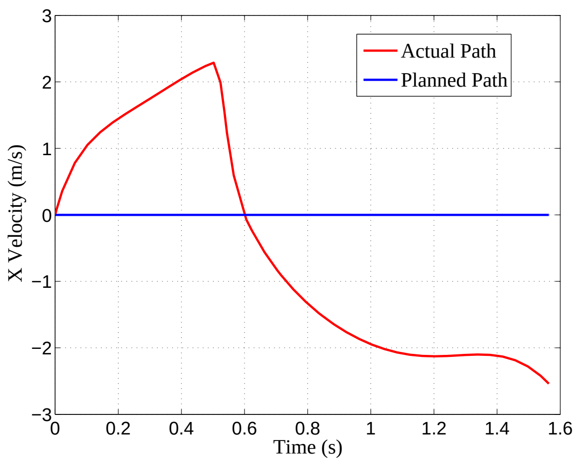


(e) $x$ Position vs. Time: Independent

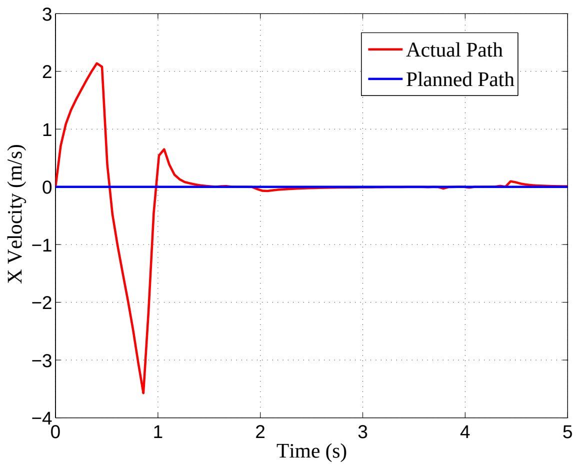


(f) $x$ Position vs. Time: Unified

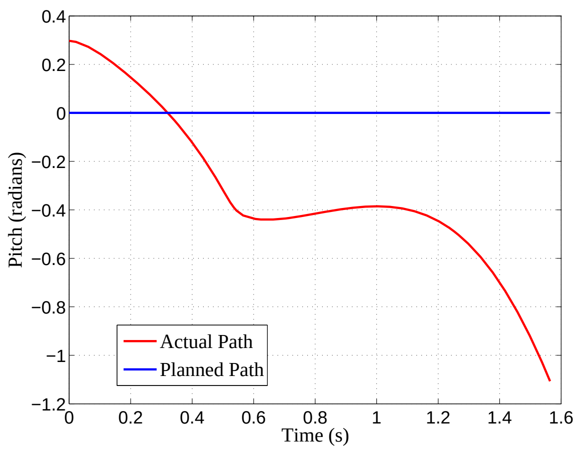
Figure 19: Extreme Balancing: Compare Independent and Unified Control



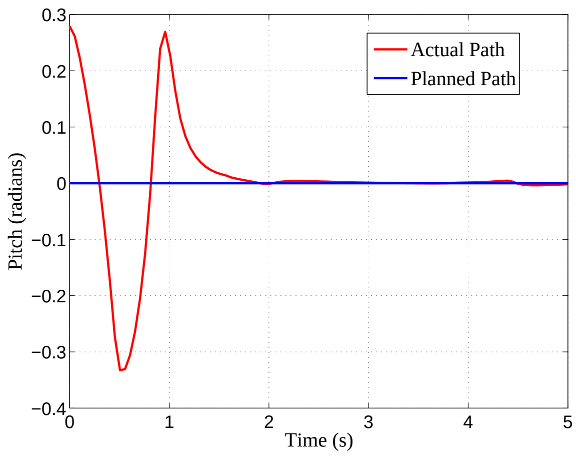
(g) $x$ Velocity vs. Time: Independent



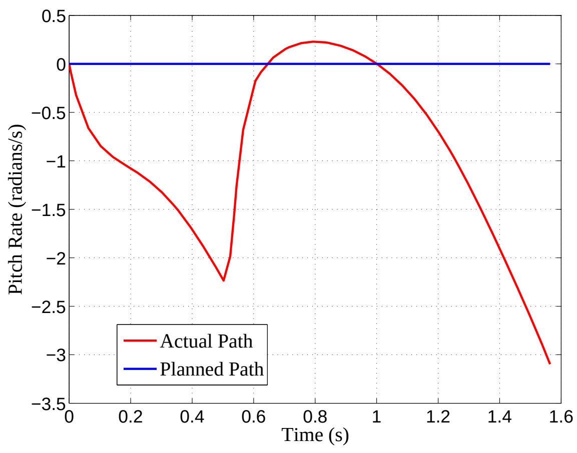
(h) $x$ Velocity vs. Time: Unified



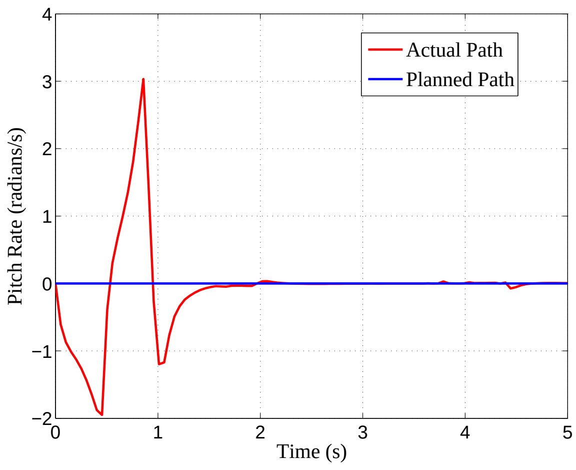
(i) Pitch vs. Time: Independent



(j) Pitch vs. Time: Unified

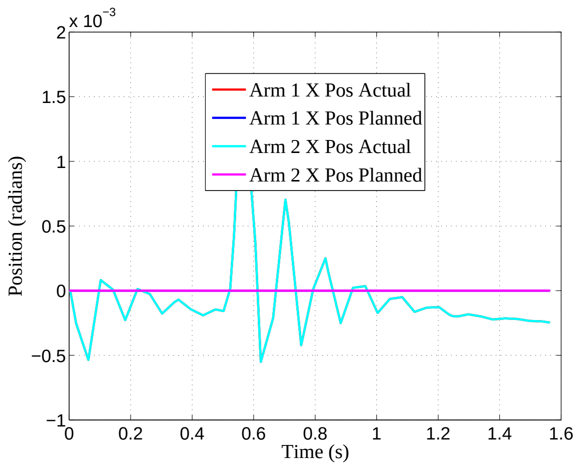


(k) Pitch Rate vs. Time: Independent

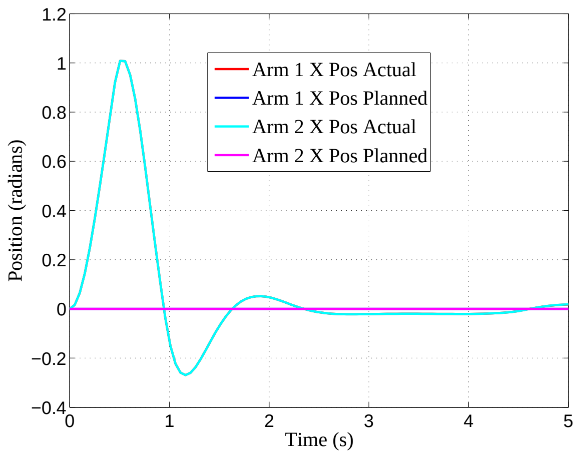


(l) Pitch Rate vs. Time: Unified

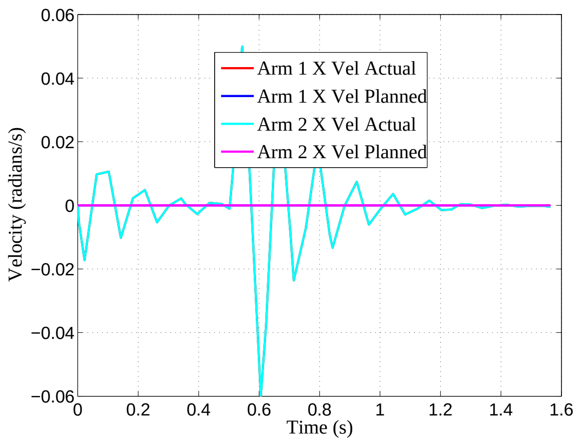
Figure 19: Extreme Balancing: Compare Independent and Unified Control



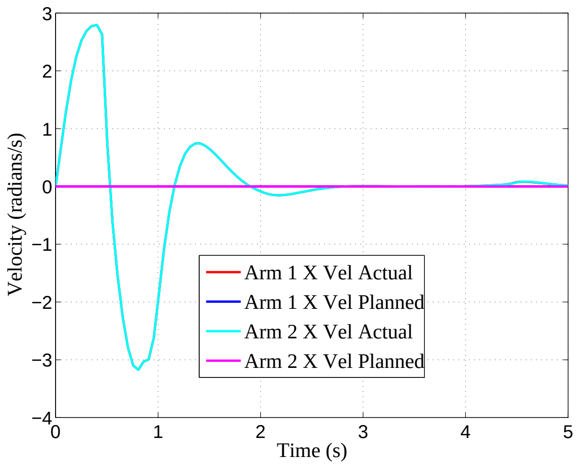
(m) $x$ Arm Angle vs. Time: Independent



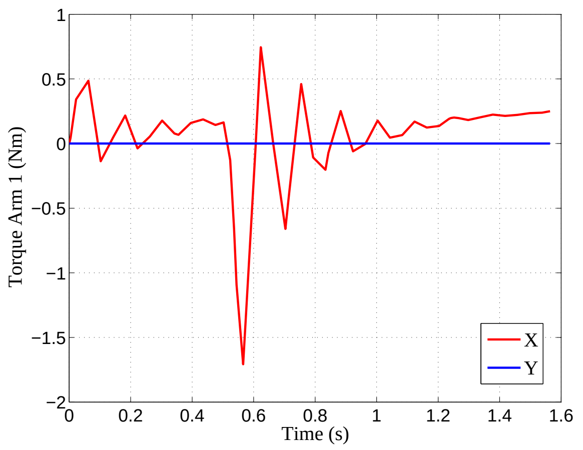
(n) $x$ Arm Angle vs. Time: Unified



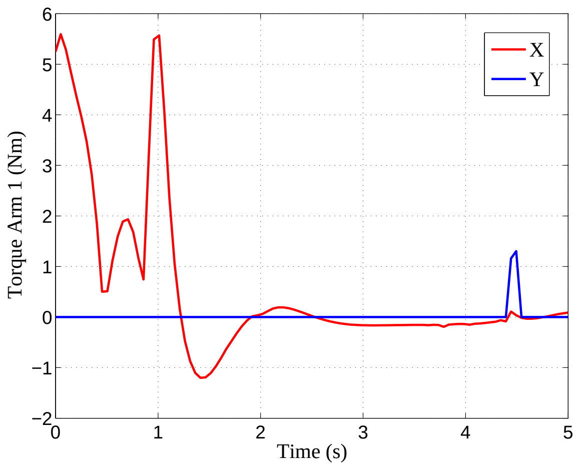
(o) $x$ Arm Ang. Vel. vs. Time: Independent



(p) $x$ Arm Ang. Vel. vs. Time: Unified



(q) Arm Torque vs. Time: Independent

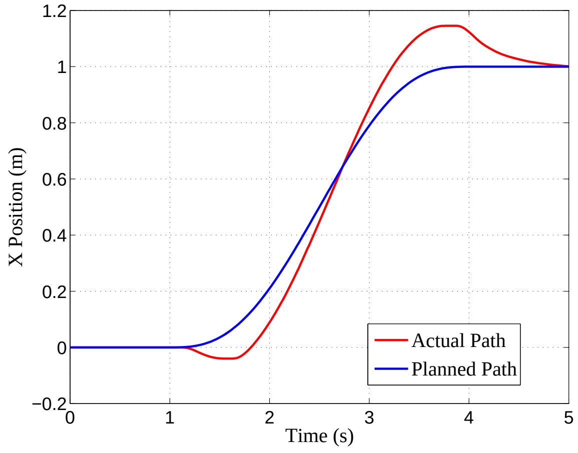


(r) Arm Torque vs. Time: Unified

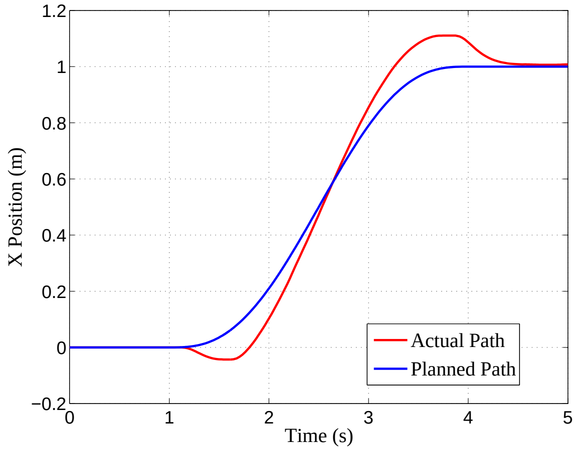
Figure 19: Extreme Balancing: Compare Independent and Unified Control

The next simulation is Ballbot waiting one second and then attempting to move 1 m in the $x$ direction in the next 3 seconds (Figure 20a-n). The unified controller, which uses its arms to aid movement, gets to the goal slightly faster as seen in (Figure 20a-b). The independent arm controller, which is trying to keep its arms at Ballbot's sides, has higher frequency (albeit very small) arm oscillations (Figure 20i, k, and m) while the unified controller operates more smoothly (Figure 20j, l, and n).

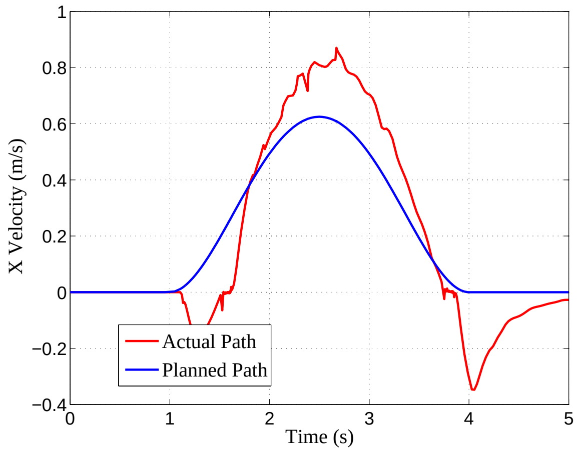
As stated in section 4, the unified controller is two planar controllers, and its weakness is out of plane arm motion. Figure 21a-b shows that with an initial out of plane body angle (5 degrees of roll and pitch), the unified controller can still balance Ballbot. This balancing scenario does not require much arm motion. Excessive out of plane arm motion might cause Ballbot to fall.



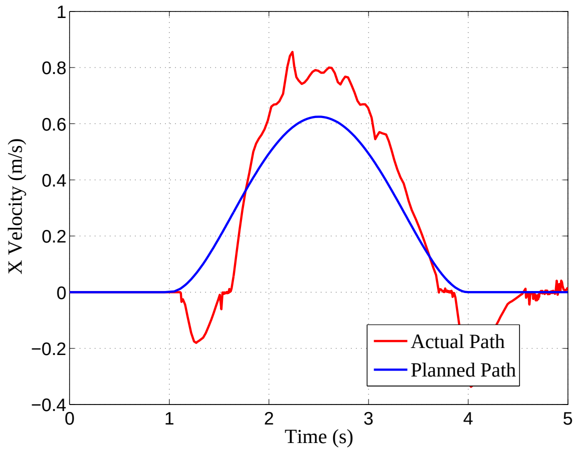
(a) $x$ Position vs. Time: Independent



(b) $x$ Position vs. Time: Unified

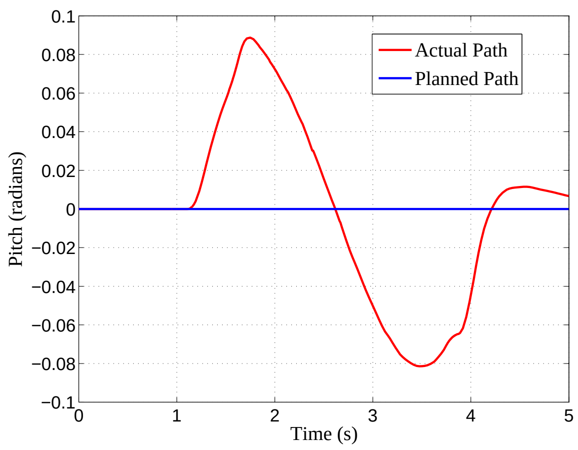


(c) $x$ Velocity vs. Time: Independent

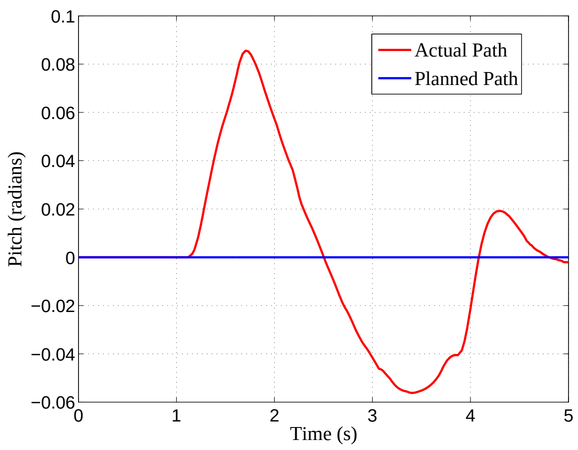


(d) $x$ Velocity vs. Time: Unified

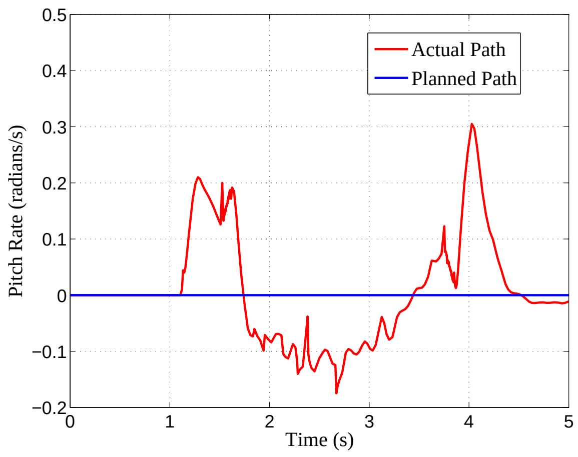
Figure 20: Move 1m in 3 seconds: Compare Independent and Unified Control



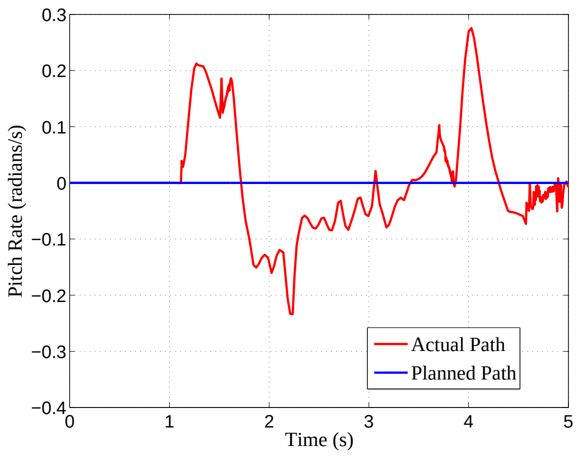
(e) Pitch vs. Time: Independent



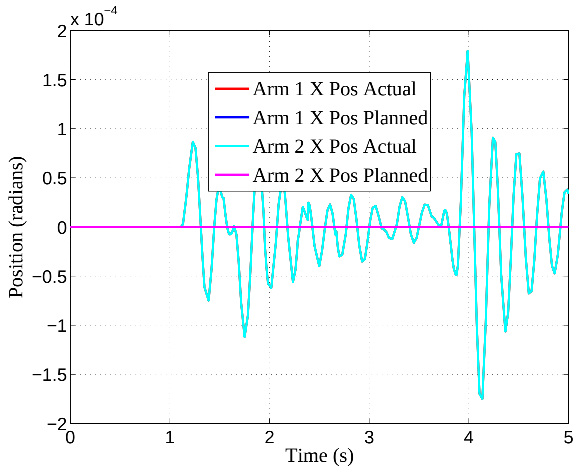
(f) Pitch vs. Time: Unified



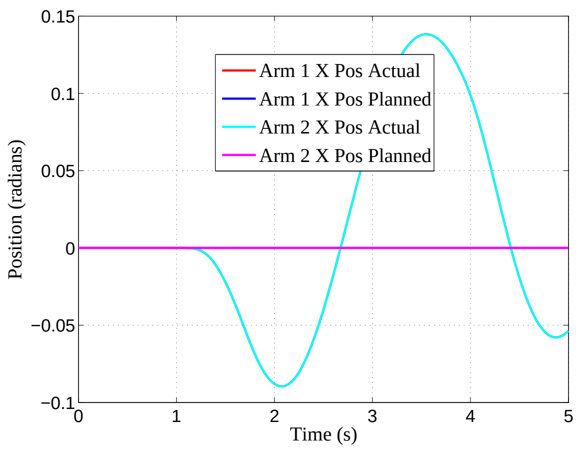
(g) Pitch Rate vs. Time: Independent



(h) Pitch Rate vs. Time: Unified

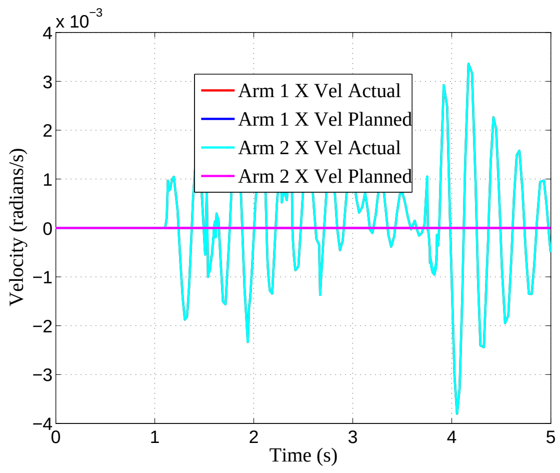


(i) $x$ Arm Angle vs. Time: Independent

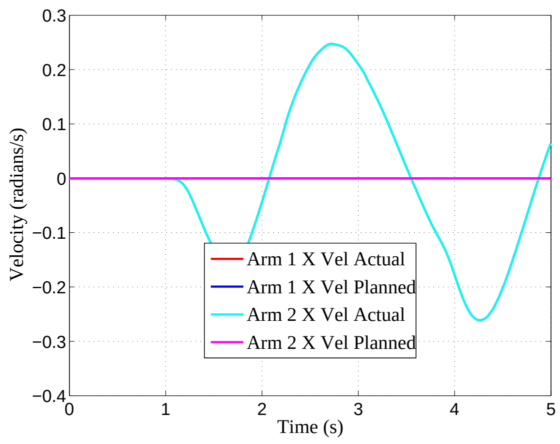


(j) $x$ Arm Angle vs. Time: Unified

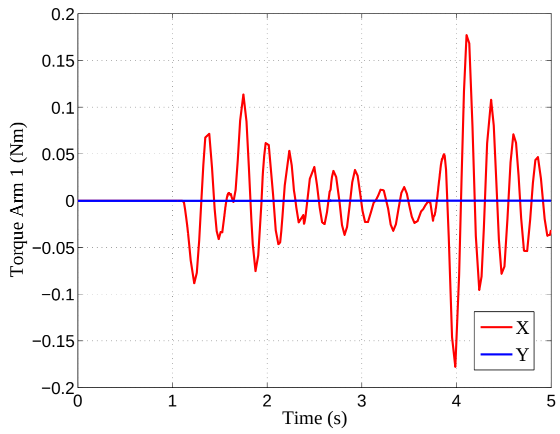
Figure 20: Move 1m in 3 seconds: Compare Independent and Unified Control



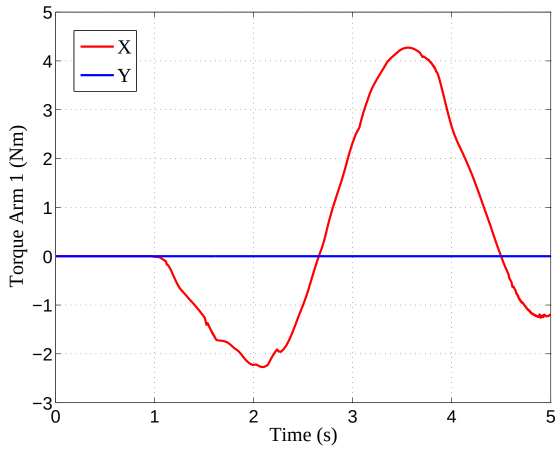
(k) $x$ Arm Ang. Vel. vs. Time: Independent



(l) $x$ Arm Ang. Vel. vs. Time: Unified

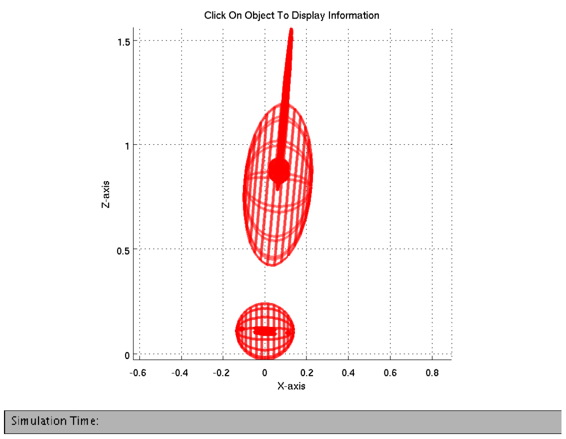


(m) Arm Torque vs. Time: Independent

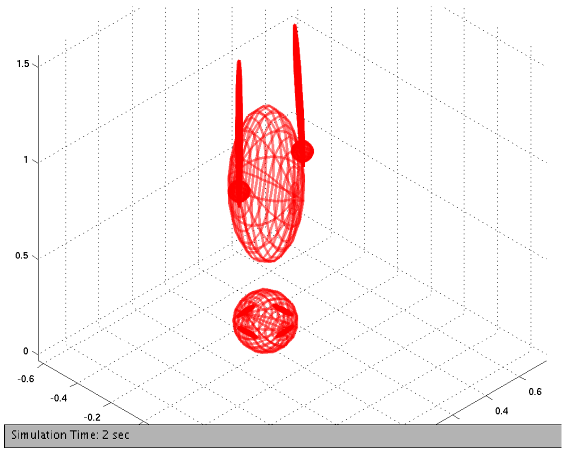


(n) Arm Torque vs. Time: Unified

Figure 20: Move 1m in 3 seconds: Compare Independent and Unified Control



(a) Initial Pose



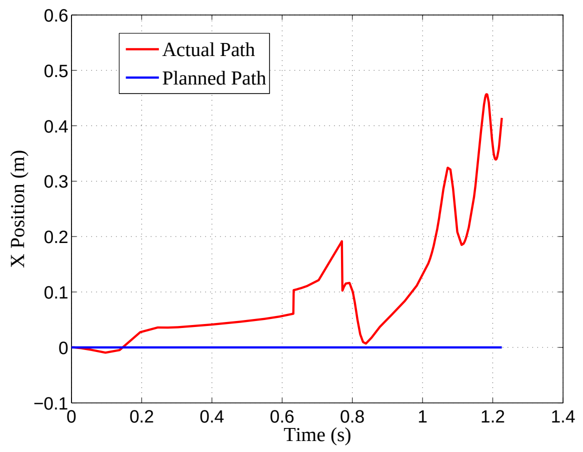
(b) Final Pose

Figure 21: Unified Controller Balancing Out of Plane

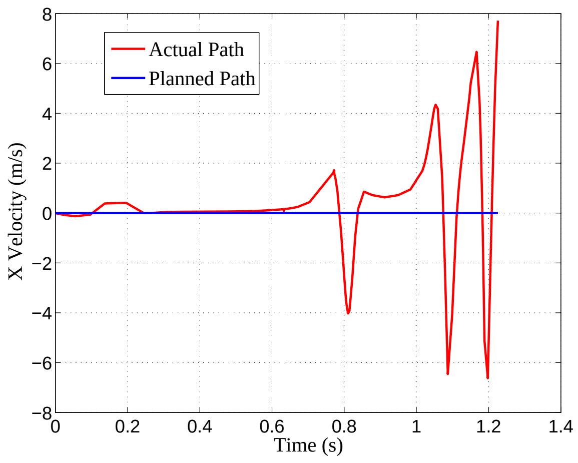
## 5.4 New Ideas

Two other means of controlling Ballbot were briefly explored. The first is using only the arms to balance starting from a 1 degree body pitch. Figure 22a-g shows that Ballbot eventually falls down with this specific arm-only controller. Note in Figure 22c-d that Ballbot does reverse its pitch before going unstable. This requires a large initial torque (Figure 22g) of over 100 Nm. This large torque makes balancing using only arms unpractical.

The second new idea is inspired by the Segway personal mobility device (www.segway.com) and the Robotic Mobility Platform [7]. These devices balance in 2D and move by leaning forward. The lean causes the controller to accelerate to regain the robot's balance. It is possible that Ballbot can move by inducing the same body lean by moving its arms. Here we control Ballbot's forward motion by defining a ball path and applying an arm torque proportional to the planned ball velocity while damping the arm motion so that the arms do not swing out of control. Figure 23a-d show that this is a feasible approach, but at least in this initial trial, this strategy does not follow the planned path as closely as the controllers described earlier (see Figure 20).

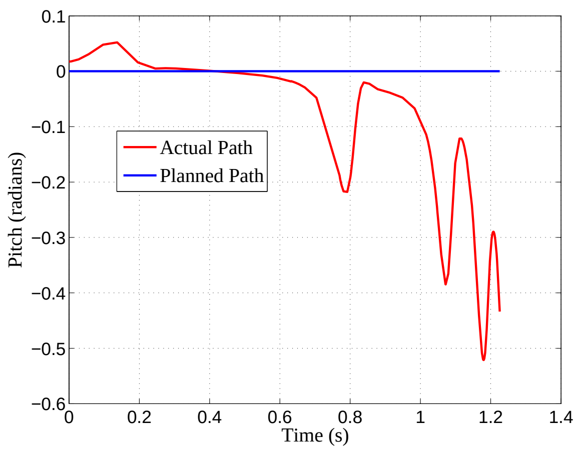


(a) $x$ Position vs. Time

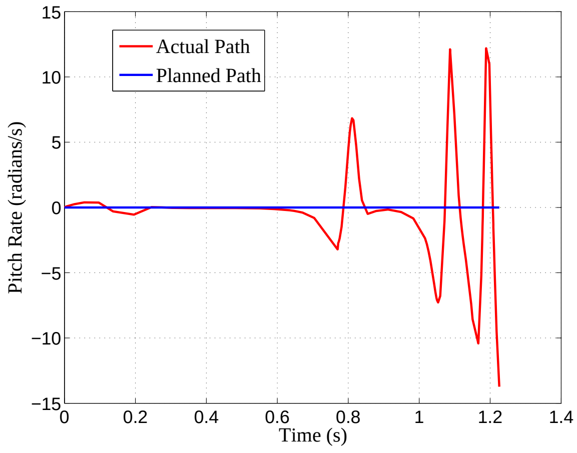


(b) $x$ Velocity vs. Time

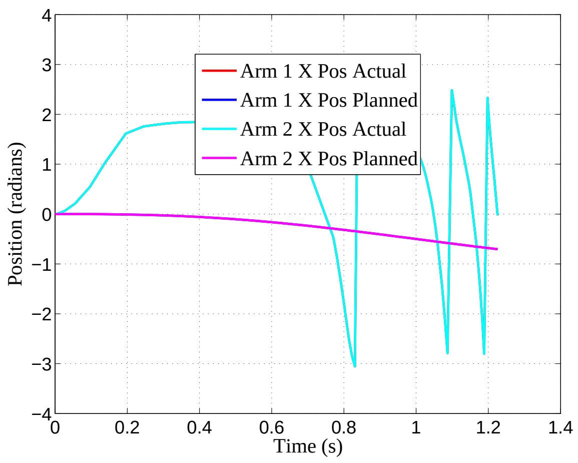
Figure 22: Balancing With Only Arms



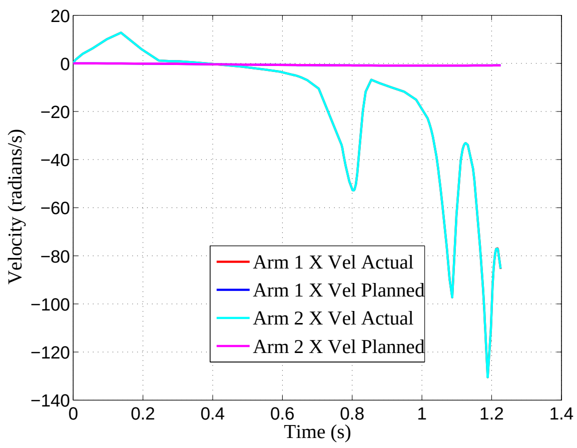
(c) Pitch vs. Time



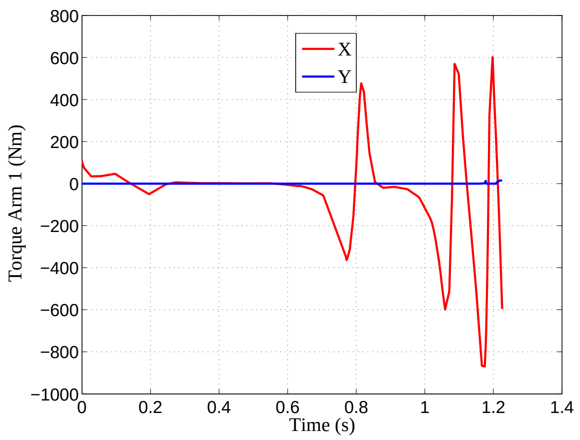
(d) Pitch Rate vs. Time



(e) Arm $x$ Angle vs. Time

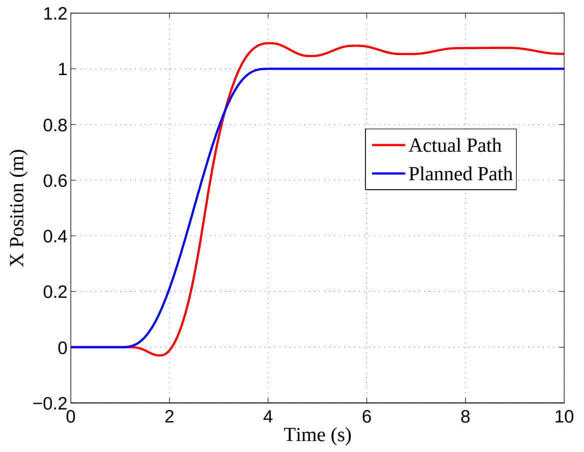


(f) Arm $x$ Ang. Vel. vs. Time

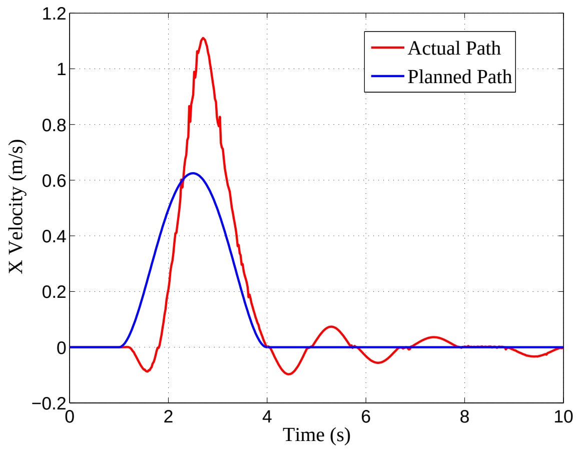


(g) Arm Torque vs. Time

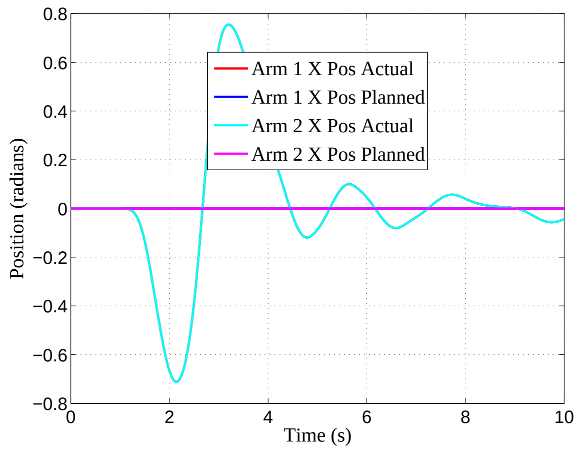
Figure 22: Balancing With Only Arms



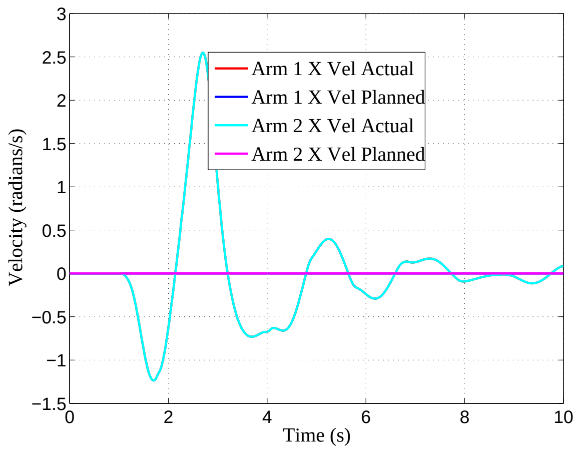
(a) $x$ Position vs. Time



(b) $x$ Velocity vs. Time



(c) Arm $x$ Angle vs. Time



(d) Arm $x$ Angular Velocity. vs. Time

Figure 23: Driving Body Movement With Arms

## 6 Conclusions and Future Work

This work examined the dynamics of a dynamically stable mobile robot with arms and explored two control strategies. The first control strategy uses independent controllers for balancing/station keeping (ball and body feedback to drive the rollers) and for the arms (arm feedback to drive the arms). The second controller is a unified controller with full state feedback (ball, body, and arms) to drive the rollers and arms. From the results of simulations, we make these conclusions:

- When working independently of the balancing/station keeping controller, the PD arm controller followed simple planned arm trajectories very well. The exception is in extreme cases when Ballbot is falling.

- For both the independent controllers and the unified controller, Ballbot assumes a leaning equilibrium when the arms move. This moves Ballbot's center of mass.

- Both control strategies try to drive Ballbot to a standing equilibrium. When Ballbot carries a heavy load, the leaning equilibrium is significantly different than the standing equilibrium. In this case, the independent controllers cannot stabilize Ballbot because the controller tries to keep the arms up. The unified controller also fails in this circumstance, but unified balancing/arm control in general might prove to be successful in handling heavy loads.
- In cases where there is no planned arm motion (simply balancing or simply moving) the unified controller outperforms the independent controllers. This is because the arm controller uses it arms to help balance or move Ballbot's body.
- The unified arm controller uses gravity compensation. This strategy only works when the gravity forces are known. If Ballbot carries an object of unknown mass, Ballbot does not know how to compensate for its weight. Another problem of unknown objects is possible saturation of the arm motors which causes arm oscillations.
- The SimMechanics simulation does not handle yaw well because of a software quirk in specifying constraints.

To overcome some of the problems mentioned the following work is suggested:

- Because of the conflict between a leaning equilibrium and the standing equilibrium new control strategies should be explored. Calculating the natural equilibrium point online and setting that as the control goal is an obvious idea to try. Research in humanoid balancing provides a wealth of resources including momentum methods [10] and zero moment point methods [3].
- Instead of using LQR to balance Ballbot while arms act independently as disturbances, treat the arms as disturbances and try a disturbance rejection strategy such as $H_\infty$ [2].
- Explore other unified control strategies or a hybrid strategy that uses independent controllers in some cases and a unified controller in other cases.
- Explore adaptive controllers or online load estimation to compensate for heavy or unknown loads.

These tasks need to be done to advance Ballbot in general:

- First and foremost, develop an interface/operating environment like RHexLib to make Ballbot testing easy.
- Solve the yaw problem in SimMechanics or use other software that has a more flexible way of defining constraints.
- The controllers of this paper are planar. This is because of the complexity of Ballbot's 3D dynamics. 3D controllers must be developed for the arms especially. The ball and body control problem also becomes a 3D problem when the body is away from the standing equilibrium as it is when carrying heavy loads.

- Develop better trajectory plans for both the ball/body and the arms. The quintic polynomial is smooth, but it does not always plan things that are physically possible. For example, Ballbot cannot begin moving forward immediately. It must develop a forward lean first which causes it to move backward.

- Develop better contact models. The friction model here lumps all friction into friction between the ball and rollers. There is also contact between the ground and ball and between the body and ball. A good starting reference is Montana [6].

- Determine the parameters of the system experimentally. Some of the parameters were measured directly (body and ball mass and center of mass) and others are taken from data sheets (the entire actuation system like the motor torque constant). Run tests to tune or even optimize the physical parameters so that the simulation fits the real Ballbot test data.

## 7 References

[1] Åström, K.J. and Wittenmark, B. *Computer-Controlled Systems: Theory and Design.* 3rd Ed., Prentice Hall, Upper Saddle River, NJ, 1997.
[2] Chen, B. *$H_\infty$ Control and Its Applications*, Springer, London, 1998.
[3] Erbatur, K., Obazaki, A., Obiya, K., Takahashi, T., and Kawamura, A. "A Study on the Zero Moment Point Measurement for Biped Walking Robots", *Proc. of the 7th International Workshop on Advanced Motion Control*, 3-5 July, 2002, Pages 431-436.
[4] Lauwers, T.B., Kantor, G.A., and Hollis, R.L. "One is Enough!" *Proc. of 2005 International Symposium of Robotics Research*, October 12-15, 2005.
[5] Lauwers, T.B., Kantor, G.A., and Hollis, R.L. "A Dynamically Stable Single-Wheeled Mobile Robot With Inverse Mouse-Ball Drive". *Proc. of 2006 IEEE International Conference on Robotics and Automation*, May 15-19, 2006, Pages:2884 - 2889.
[6] Montana, D.J., "The Kinematics of Contact and Grasp" *International Journal of Robotics Research*, Vol. 7, No. 3, June, 1988, pages 17-32.
[7] Nguyen, J., Morrell, J., Mullens, A., Burnmeister, S., Farrington, K., Thomas, K., and Gage, D. "Segway Robotic Mobility Platform". *Proc. of SPIE - Volume 5609: Mobile Robots XVII*, October, 2004.
[8] Sciavicco, L. and Siciliano, B. *Modelling and Control of Robot Manipulators.* 2nd Ed., Springer, London, 2000.
[9] Thibodeau, B.J., Deegan, P., and Grupen, R. "Static Analysis of Contact Forces With a Mobile Manipulator". *Proc. of 2006 IEEE International Conference on Robotics and Automation*, May 15-19, 2006, Pages:4007 - 4012.
[10] Yoshida, E., Guan, Y., Sian, N.E., Hugel, V., Blazevic, P., Kheddar, A., and Yokoi, K. "Motion Planning for Whole Body Tasks by Humanoid Robots", *Proc. of 2005 IEEE International Conference on Mechatronics and Automation*, July, 2005, Pages 1784-1789.

# A SimMechanics Block Diagrams

## A.1 Independent Controllers

Figure 24 is a lower level SimMechanics block diagram of the “Control and Actuation” block in Figure 4. Starting from the left side of the figure, one sees the “X Sensor” block which senses the angular position and velocity of the revolute joint connecting the “Body” and the “x1 roller” in Figure 4. The range of the angular position measurement is 0 to $2\pi$, so it is converted to a continuous angle in the next block. Below the “X Sensor” is the “Body Sensor” block which outputs the orientation of the body in the form of a quaternion and quaternion derivative. The next block converts the quaternion to pitch, pitch rate, roll, and roll rate. Below the “Body Sensor” is the “Path” block. This block takes a distance, the desired number of seconds in which to move that distance, an amount of time to wait before moving, and a direction and creates a path according to equations (17) and (18). Below the “Path” block is the “Y Sensor” which measures the angular position and velocity of the joint connecting the “Body” and the “y1 roller” in Figure 4. Moving to the right, the augmented (continuous angles and quaternion changed to pitch and roll) outputs of these three sensor blocks and the “Path” block output are input to the “Controller”. The “Controller” outputs a signal to the “Actuation and Friction” block, which in turn sends torques (after being divided by two by the two gain blocks to split the overall torque between the two motors) to the four roller actuation blocks, “x1 Act”, “x2 Act”, “y1 Act”, and “y2 Act”.

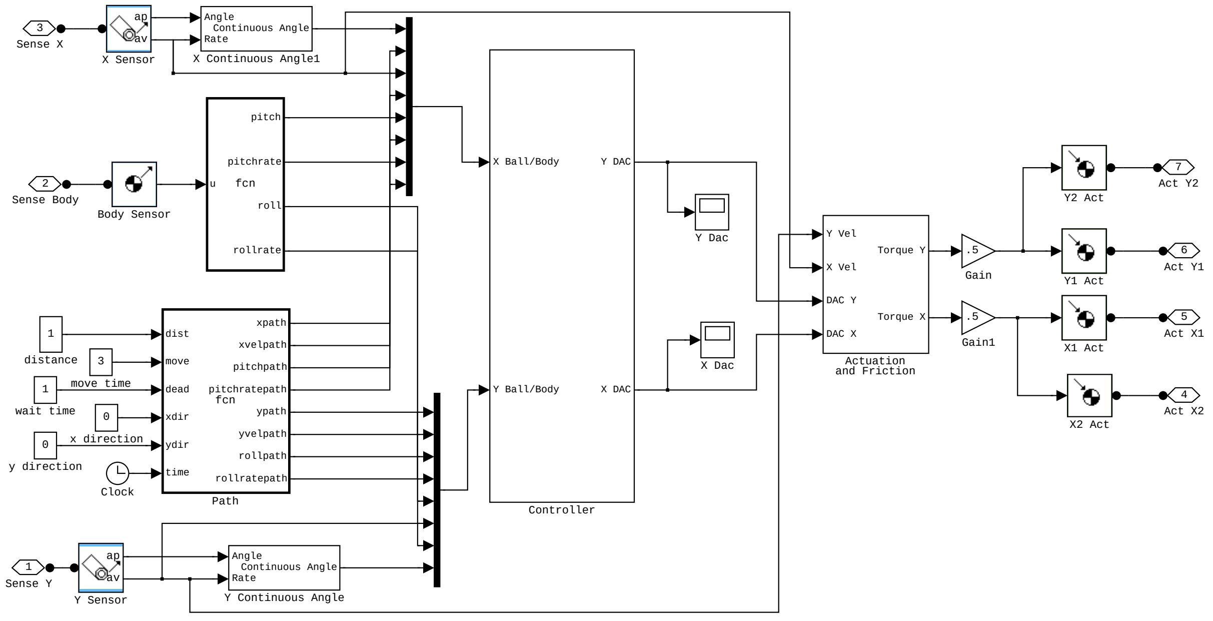


Figure 24: Ball Control and Actuation Block for Independent Controllers Model

Figure 25 is a lower level SimMechanics block diagram of the “Controller Block” in Figure 24. It takes all of the sensor data described above and inputs them into two identical controllers - one in the $x$ direction and one in the $y$ direction Moving from left to right, each controller takes the differences between roller position, roller velocity, body angle (either roll or pitch), and body velocity and their respective paths (all measured in radians). These differences are multiplied by gains and then added together and multiplied by negative one. This results in a torque (in Nm) to be applied to the ball. The ball torque is first converted to a roller torque and then to a DAC input.

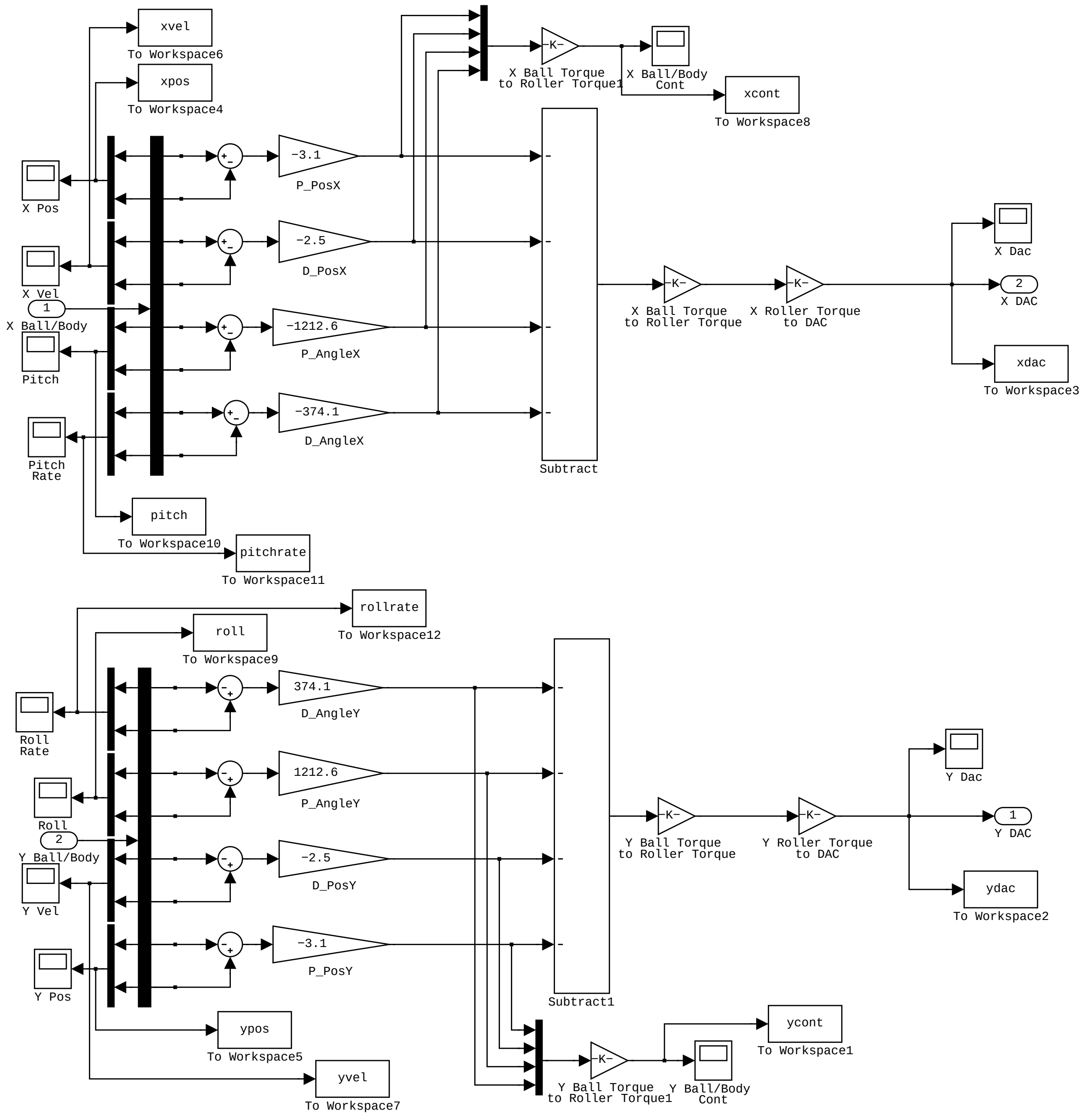


Figure 25: Ball Controller Block for Independent Controllers Model

Figure 26 is a lower level SimMechanics block diagram of the “Actuation and Friction” block in Figure 24. This block contains the actuation model summarized by Figure 7. The blocks “Handle X Friction” and “Handle Y Friction” contain Matlab code that implements the friction model described in Section 2.1.2.

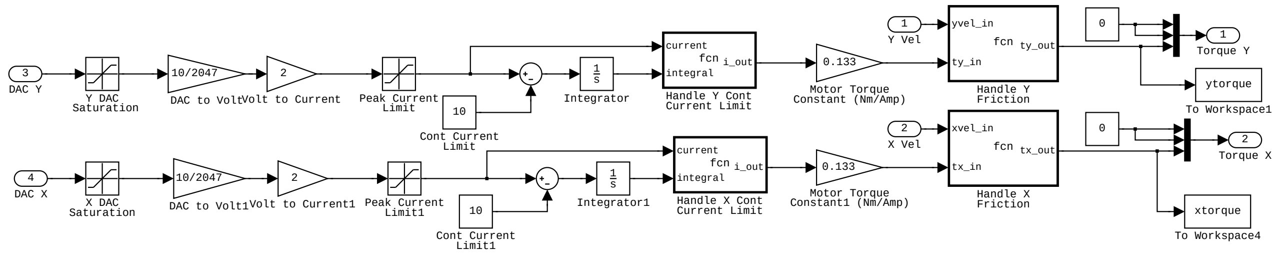


Figure 26: Actuation and Friction Block for Both Models

Figure 27 is a lower level SimMechanics block diagram of the “Arm Controller” block in Figure 4. At the bottom are four blocks with light blue outlines. These are the arm sensors. They sense the angular position and velocity of the $x$ and $y$ rotations of each arm. Also at the bottom is the “Path” block which is similar to the “Path” block in Figure 24. The outputs of the arm sensors and the “Path” block are sent to the “Controller” block. The “Controller” outputs torques (in Nm) which are compared to torque limits in the “Saturation” blocks. The outputs of the “Saturation” blocks are sent to the four arm actuation blocks, “Arm1x Torque”, “Arm1y Torque”, “Arm2x Torque”, and “Arm2y Torque”. These four blocks apply torques to the arms.

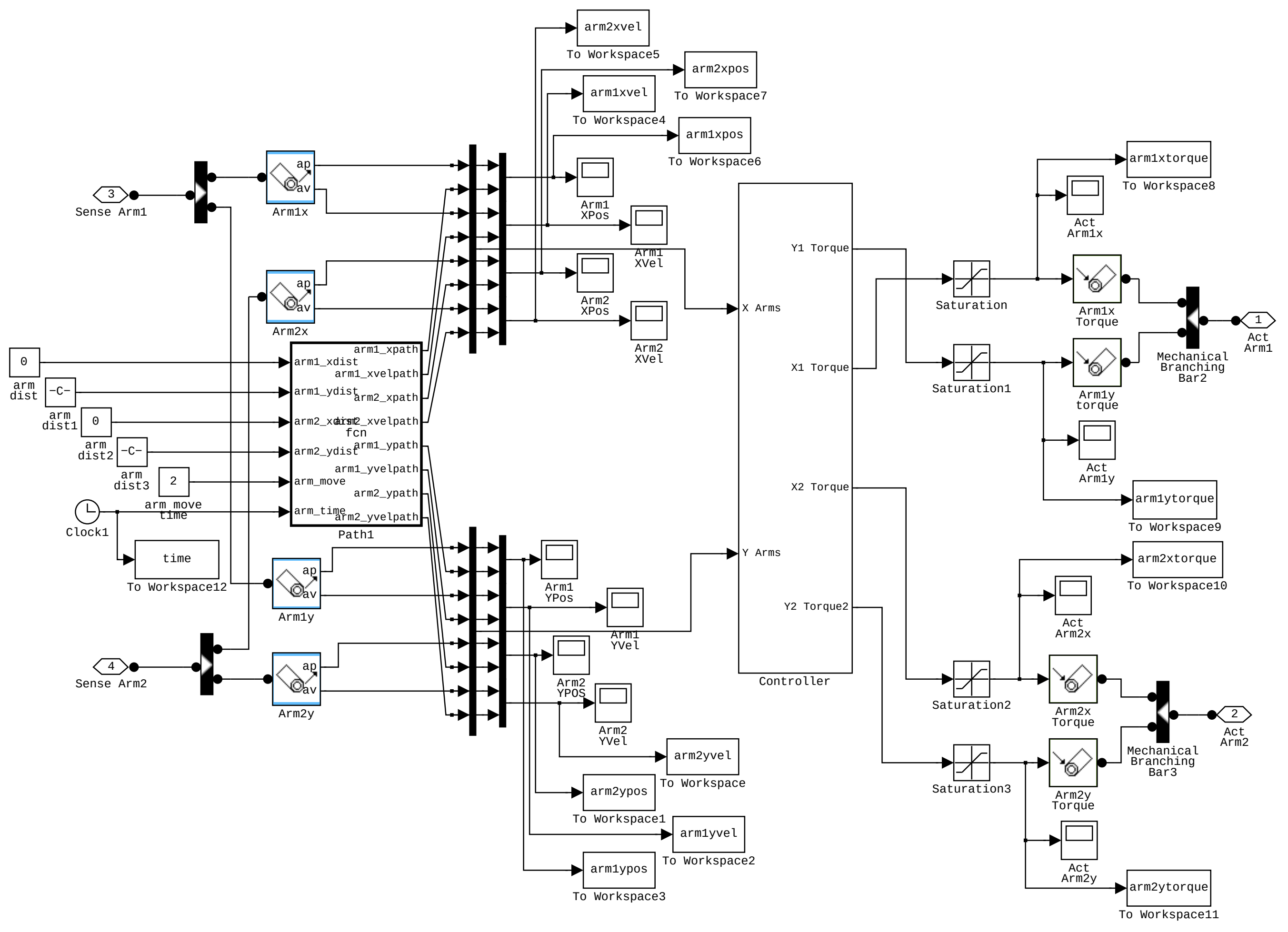


Figure 27: Arm Control and Actuation Block for Independent Controllers Model

Figure 28 is a lower level SimMechanics block diagram of the “Controller Block” in Figure 27. It shows four identical PD controllers (2 DoFs for each arm). The differences between the arm angular position and velocity and their respective paths are multiplied by gains and then multiplied by negative one. The controllers output torques in Nm.

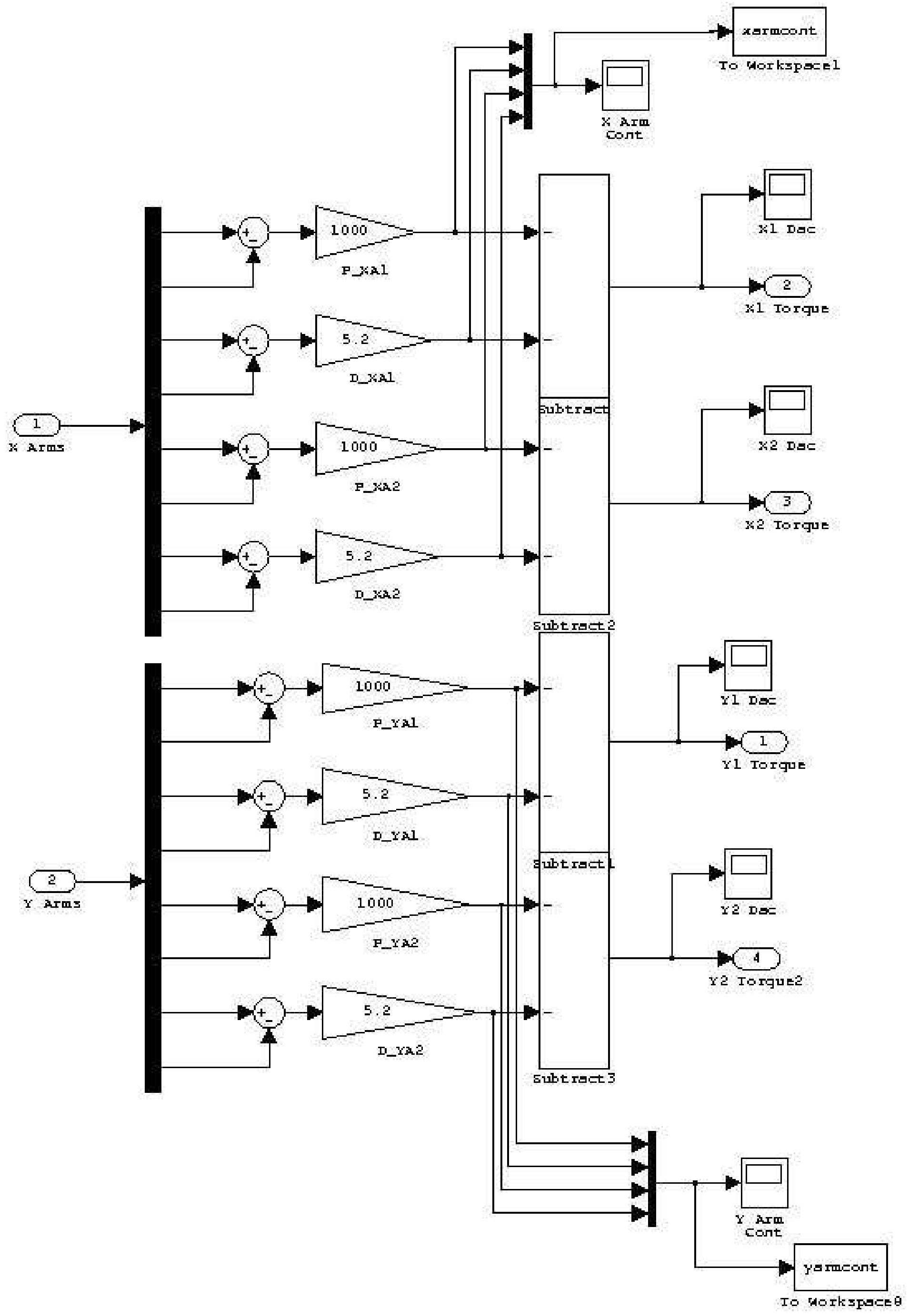


Figure 28: Arm Controller Block for Independent Controllers Model

## A.2 Unified Controller

Figure 29 is very similar to Figure 24 except that it is the “Control and Actuation” block for the model that uses the unified balancing/station keeping and arm controller. The difference is the addition of arm sensing blocks and an arm path block in the lower left corner. The “Actuation and Friction” block is identical to the block for the independent controllers shown in Figure 26

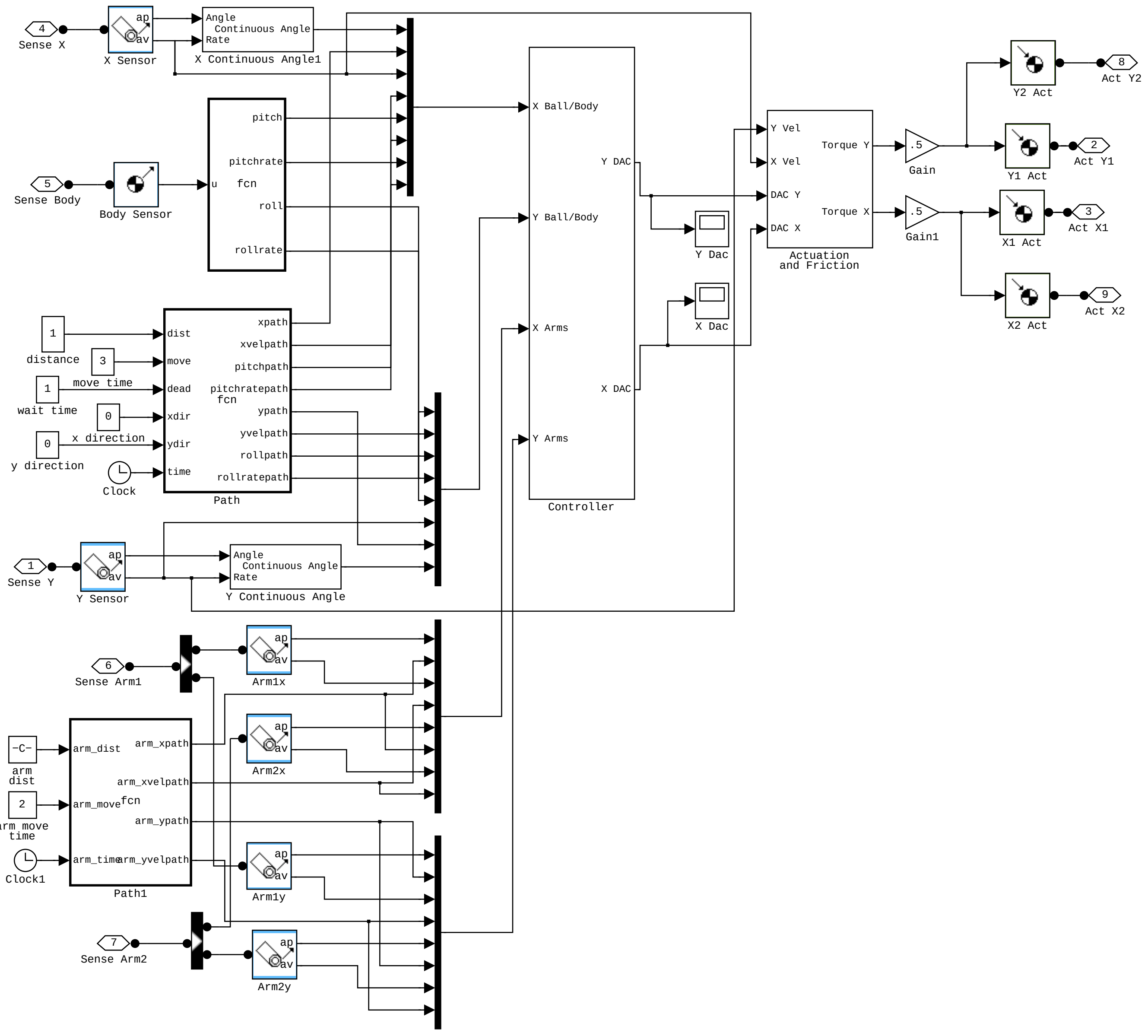


Figure 29: Ball Control and Actuation Block for Unified Controller Model

Figure 30 is a lower level SimMechanics block diagram of the “Controller Block” in Figure 29. It takes all of the sensor data from the rollers, body, and arms and inputs them into two identical controllers - one in the $x$ direction and one in the $y$ direction. Moving from left to right, each controller takes the differences between roller position, roller velocity, body angle (either roll or pitch), body velocity, arm 1 angular position and velocity, and arm 2 angular position and velocity and their respective paths (all measured in radians). These differences are multiplied by gains and then added together and multiplied by negative one. There are eight inputs to each controller. The controllers output torques (in Nm) to be applied to the ball. The ball torque is first converted to a roller torque and then to a DAC input.

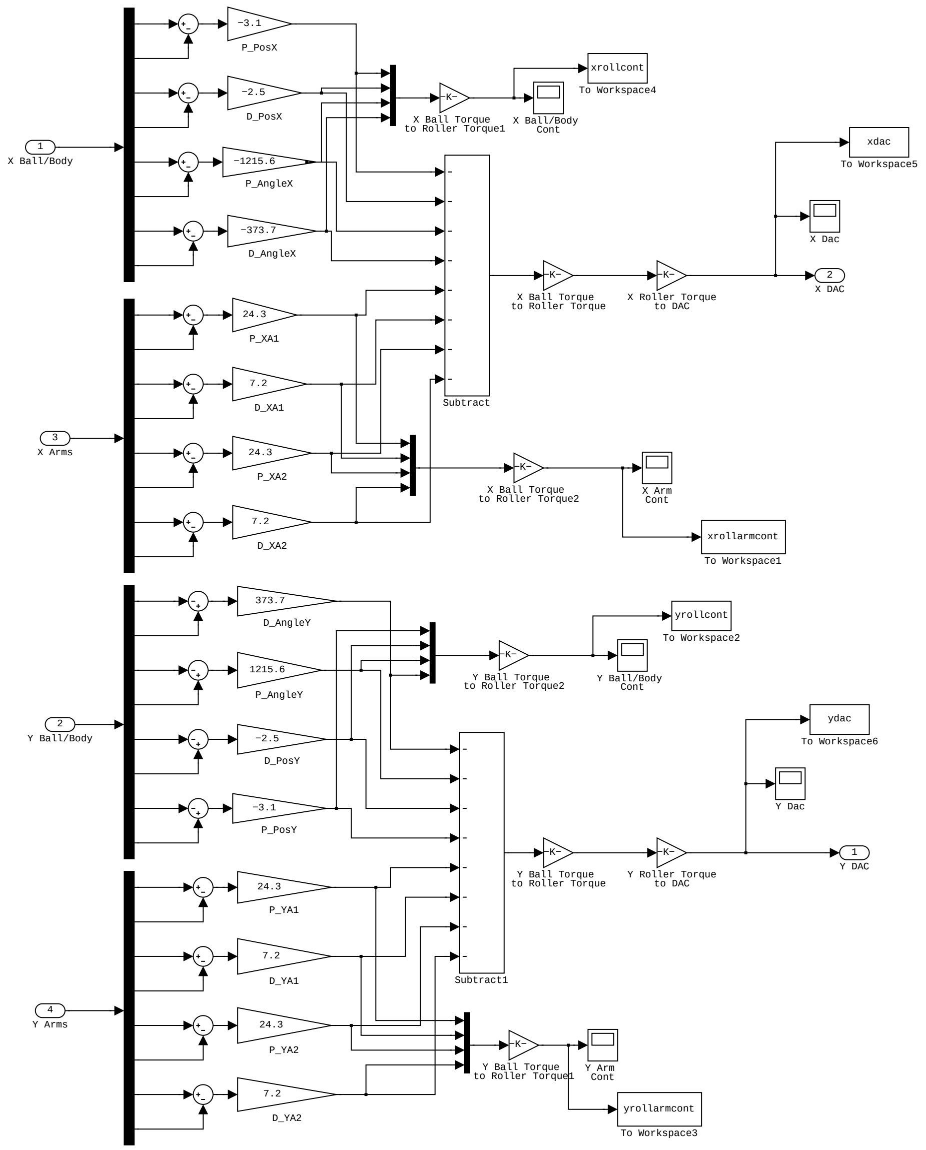


Figure 30: Ball Controller Block for Unified Controller Model

Figure 31 is very similar to Figure 27 except that it is the "Arm Controller" block for the model that uses the unified balancing/station keeping and arm controller. One difference is the addition of arm sensing blocks and an arm path block in the lower left corner. The other difference is the addition of four "gravity compensation" blocks (Matlab code inside).

These blocks use equation (20).

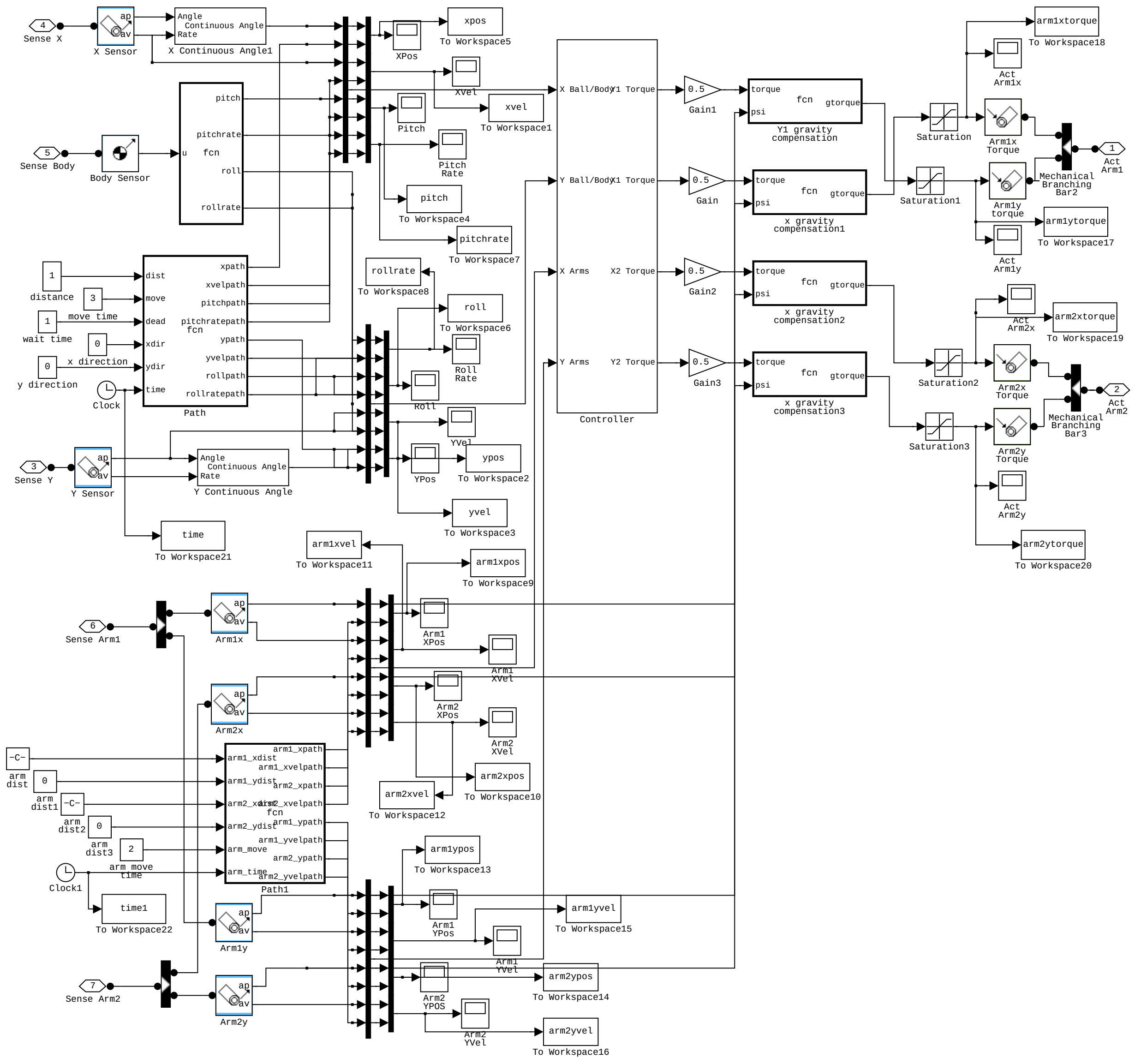


Figure 31: Arm Control and Actuation Block for Unified Controller Model

Figure 32 is a lower level SimMechanics block diagram of the "Controller Block" in Figure 31. There are four identical controllers - one for each DoF of each arm. Moving from left to right, each controller takes the differences between roller position, roller velocity, body angle (either roll or pitch), body velocity, the arm angular position of the DoF being controlled and the arm angular velocity of the DoF being controlled, and their respective paths (all measured in radians). These differences are multiplied by gains and then added together and multiplied by negative one. There are six inputs to each controller. The controllers output torques (in Nm) to be applied to the arms.

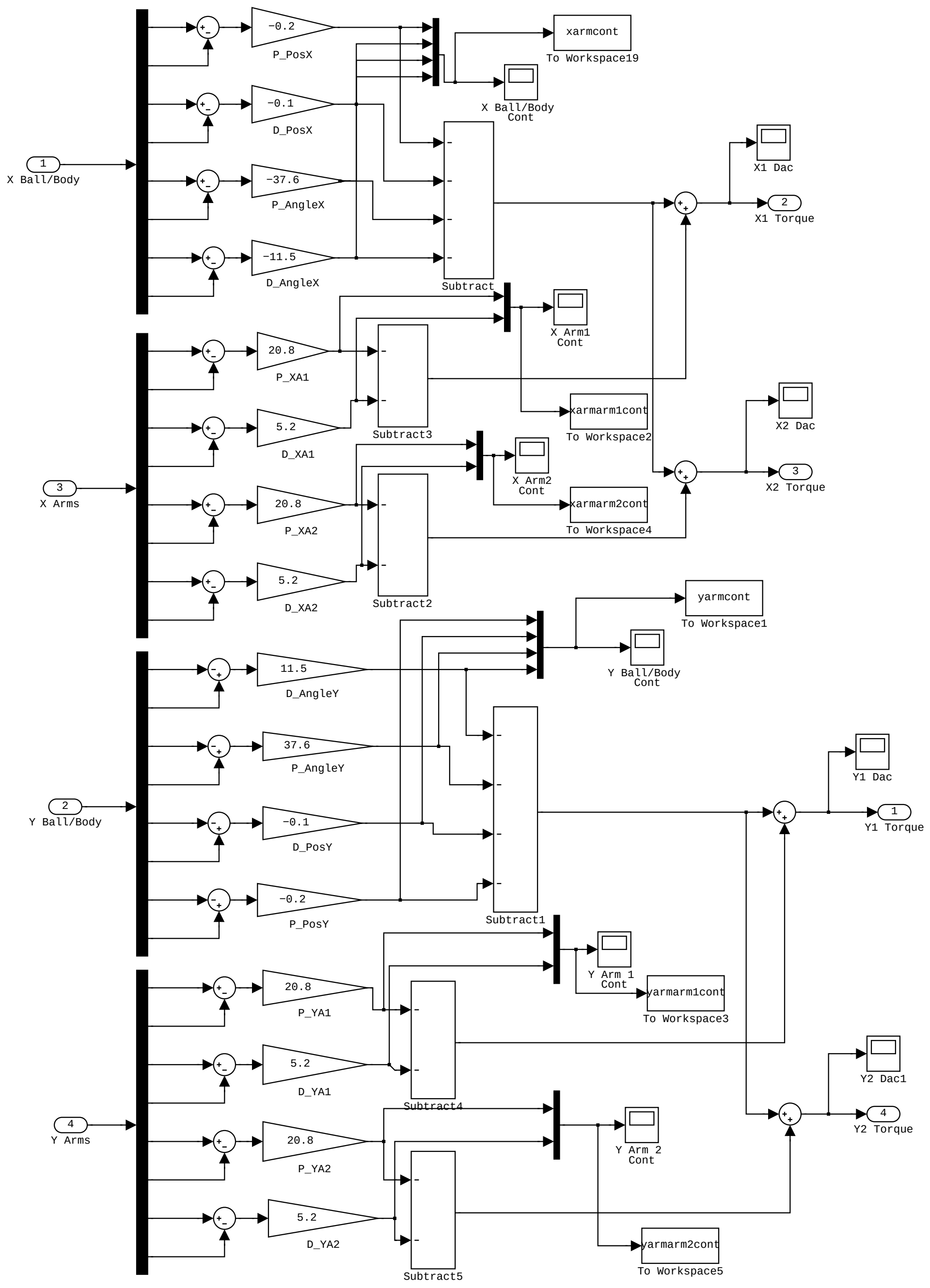


Figure 32: Arm Controller Block for Unified Controller Model