MAE 3260:

System Dynamics Final Project:
CU-D2 Self Balancing Robot

By:
Alan Munschy (apm238), Ryan Brynjolfson (rrb224), Charles Wall-Davis (caw362), and Bryan Kim (bjk228)

Project Started On:
11/21/2025

Project Completed On:
12/09/2025

Project Description:

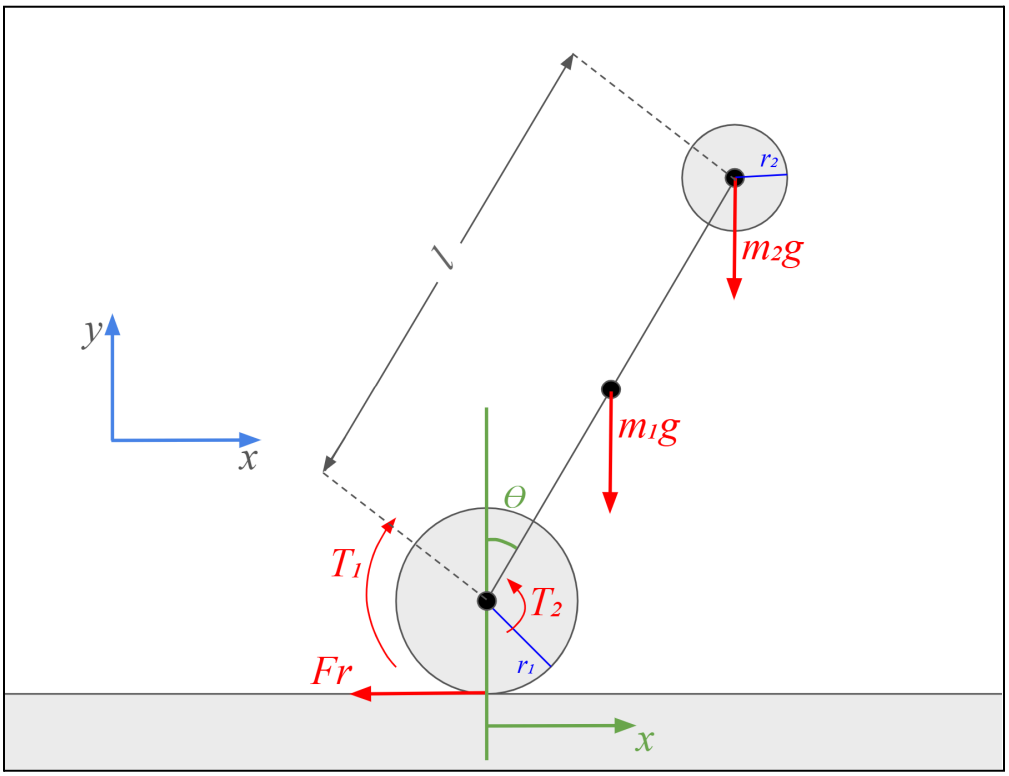
We came up with this project, CU-D2 after brainstorming interesting systems that involved controllers. We arrived at modelling a self-balancing robot that could be modelled closely to an inverted pendulum with two links; we modelled this in the computer simulation. Unlike an inverted pendulum system, though, we also have a wheel that provides a torque. Also, for the dynamic and state space representation, we simplified the model for one linkage.

We planned to study multiple parts in this system, including a dynamic representation for the system and developing a state-space model for the system. Furthermore, we wanted to make a computer simulation using a PD control law and see if we could achieve a true self-balancing robot in a 2D environment.

Project Outline:

1. **Dynamic Model**
2. **State-Space Representation of Model**
3. **Control Law, Reference Equations**
4. **Computer Simulation for Model**

Brief Overview of System:



Model of Static Self-Balancing Robot CU-D2

Some notes about the parameters of this model:
*Similar to inverted pendulum, but with another point mass on top, and wheel for base instead of cart.

Inputs:
T1 : Motor provided torque on wheel
T2 : Motor provided torque on linkage

States:
Position x
Theta θ
Velocity x'
Angular Velocity θ'

Outputs:
Position x
Angular tilt θ

1. Dynamic Modelling:

Ryan Brynjolfson (rrb244) & Charles Walls-Davis (caw362) Contribution:

To find our first governing 2nd order ODE, we analyze the positional states.

#1

1. Determine coordinates. We have:

$$x = x_{wheel}$$
$$x_1 = x + \frac{l}{2}sin\theta$$
$$x_2 = x_1 + lsin\theta$$

Taking derivatives with respect to time:

$$x_1' = x' + \theta'\frac{l}{2}cos\theta$$
$$x_2' = x_1' + \theta' lcos\theta$$
$$x_1'' = x'' - \theta'^2\frac{l}{2}sin\theta + \frac{l}{2}cos\theta\theta''$$
$$x_2'' = x_1'' - \theta'^2 lsin\theta + lcos\theta\theta''$$

2. Force Balance. We have:

$$\sum F_x = m_{wheel}x'' - \frac{T_1}{r_1} + m_1x_1'' + m_2x_2'' = 0$$

3. Plug in to find first governing ODE:

After plugging in, we find:

ODE 1:

$$(m_{wheel} + m_1 + m_2)x'' + \theta''(\frac{l}{2}cos\theta m_1 + lcos\theta m_2) + \theta'^2(-m_2 lsin\theta - m_1\frac{l}{2}sin\theta) = \frac{T_1}{r_1}$$

To find our second governing 2nd order ODE, we analyze the angular states.

#2

1. Determine coordinates.

   Because we only have one angular θ rotating, we only need to find moments of inertia to account for rigid bodies being rotated by lone θ; θ is our only relevant coordinate.

   Because the wheel is connected via a free revolutionary joint to the linkage arm's torque actuator, theta is not acting directly on the wheel, so wheel torque is negligible.

2. Moment Balance. We have:

   $$\sum I_{tot}\theta'' = \circlearrowleft^{+} \sum T_{tot}.$$

   For a dynamic system, this turns into:

   $$\sum(I_{link1} + I_{link2})\theta'' = \sum T_{gravity} + \sum T_{dynamic\ acceleration} + T_{motor},$$

   And more specifically:

   $$\sum(I_{link1} + I_{link2})\theta'' = T_{grav\ mass1} + T_{gravmass\ 2} + T_2 + (r_{arm} \times m_1 x_1'') + (r_{arm} \times m_2 x_2'').$$

3. After plugging in using previous global coordinates and solving, we rearrange to get θ'', θ'^2, x'', extra terms.

   ODE 2:

   $$\theta''(\tfrac{1}{2}m_2 r_2^{\ 2} + m_2 l^2 + \tfrac{1}{3}m_1 l^2 + \tfrac{l^2}{4}cos^2\theta m_1 + l^2 cos^2\theta m_2) + x''(\tfrac{l}{2}cos\theta m_1 + lcos\theta m_2) + \theta'^2(-\tfrac{l^2}{4}sin\,\theta cos\theta m_1 - l^2 sin\theta cos\theta m_2) + (\tfrac{l}{2}sin\theta m_1 g + lsin\theta m_2 g) = T_2$$

2. State - Space Modelling:

Because we have two coupled 2nd order ODE's, we need to solve for x" and θ'' using matrices.

First, we should linearize our model ODE's.

For a small angle deviation ϕ from vertical:

$cos(\phi) \approx -1,\ \theta \approx 0,\ sin(\phi) \approx \phi,\ cos^2\phi \approx 1,\ \theta'^2 = \phi'^2 \approx 0.$

So,

ODE 1 LINEARIZED:

$$(m_{wheel} + m_1 + m_2)x'' + \theta''(-\tfrac{l}{2}m_1 - lm_2) = \frac{T_1}{r_1}$$

ODE 2 LINEARIZED:

$$\theta''(\tfrac{1}{2}m_2 r_2^{\,2} + m_2 l^2 + \tfrac{1}{3}m_1 l^2 + \tfrac{l^2}{4}m_1 + l^2 m_2) + x''(-\tfrac{l}{2}m_1 - lm_2) + (\tfrac{l}{2}\phi m_1 g + l\phi m_2 g) = T_2.$$

*We can already see how much more simple our model is.

We can now turn to figuring out how to make our state-space model by rearranging and plugin in the ODE's to the matrix equation:

$M(x)x'' = F(x', x).$

To solve for x" vector: $x'' = M^{-1}F(x', x)$.

x" = $$M^{-1}(q) = \frac{1}{M_{11}M_{22} - M_{12}M_{21}} \begin{bmatrix} M_{22} & -M_{12} \\ -M_{21} & M_{11} \end{bmatrix} * F(x', x).$$

For reference, $M_{11} = m_w + m_1 + m_2,\ M_{12} = -(\tfrac{l}{2}m_1 + lm_2),$

$M_{21} = (\tfrac{l}{2}m_1 + lm_2),\ M_{22} = \tfrac{1}{2}m_2 r_2^{\,2} + m_2 l^2 + \tfrac{1}{3}m_1 l^2 + \tfrac{l^2}{4}m_1 + l^2 m_2.$

$$F(x', x) = \begin{bmatrix} \dfrac{T_1}{r_1} \\ T_2 - \left(\dfrac{l}{2}m_1 g + lm_2 g\right)\phi \end{bmatrix}.$$

We now can solve for our state space model.

$$x' = Ax + Bu$$
$$y = Cx + Du\,.$$

We get:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{M_{12}D}{\Delta} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{M_{11}D}{\Delta} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{M_{22}}{\Delta r_1} & -\frac{M_{12}}{\Delta} \\ 0 & 0 \\ -\frac{M_{21}}{\Delta r_1} & \frac{M_{11}}{\Delta} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$$

$$y = \begin{bmatrix} x \\ \phi \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

With $\Delta = M_{11}M_{22} - M_{12}M_{21}$,

D = $(\frac{l}{2}m_1 g + lm_2 g)$,

$M_{11} = m_w + m_1 + m_2$, $M_{12} = -(\frac{l}{2}m_1 + lm_2)$,

$M_{21} = (\frac{l}{2}m_1 + lm_2)$, $M_{22} = \frac{1}{2}m_2 r_2^{\,2} + m_2 l^2 + \frac{1}{3}m_1 l^2 + \frac{l^2}{4}m_1 + l^2 m_2$.

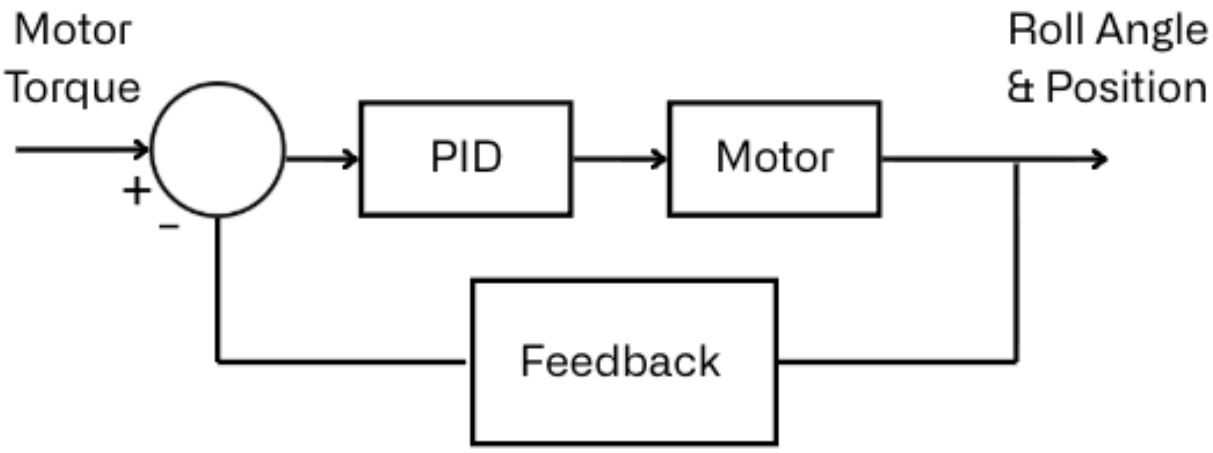
This is our completed linearized State-Space Model that Ryan and Charles derived. We note that in the controller simulation, we used an Ansatz for a PD control law using the roll angle and roll angle derivative, which was not directly modelled after this particular state space representation.

3. Dynamic Modeling:

Bryan Kim (bjk228) contribution:

The key consideration of CU-D2 is to not let angle θ between the horizontal and the linkage exceed a critical value, otherwise it will lose balance and become unstable. This happens because the body has some inertia, causing a lag between when the wheel starts rolling and when the body moves along with it. This critical angle can be calculated at the point where the center of mass of the body's vertical component falls under a point where it becomes unsupported by the wheel (i.e. the point is to the left or right of the wheel). So $\theta_{critical} = \arcsin(r_1/l)$, since the radius of the wheel and the length of the linkage are known constants.

We can also use the inputs, states, and outputs in the state space model to create a block diagram that governs CU-D2's behavior.



- Input: Motor torques T1 and T2 → The PID controller decides how much

- States: Position x, roll angle θ, linear velocity x', angular velocity θ' → Monitored by the controller to calculate inputs
- Outputs: Position x, roll angle θ → New results due to controller adjustments, can be determined through the equations of motion as part of the dynamics calculation

The aims of CU-D2 are to minimize overshoot and rise time, as either taking too long to reach the reference value or going too far past it causes it to exceed the critical angle and lose its balance. While it makes sense to add a PID controller to mitigate the effects, the motors would reach saturation at lower angular velocities as they are often the cheap, low performance variety used in these types of hobby projects, and not the commercial, high performing ones. To prevent saturation and windup due to integral error, PD control would be the optimal.

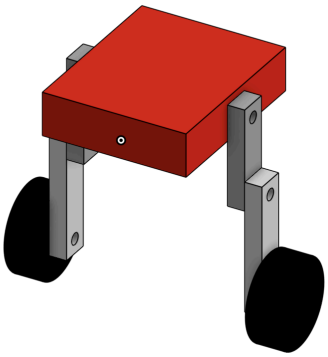
4. Mujoco Simulation

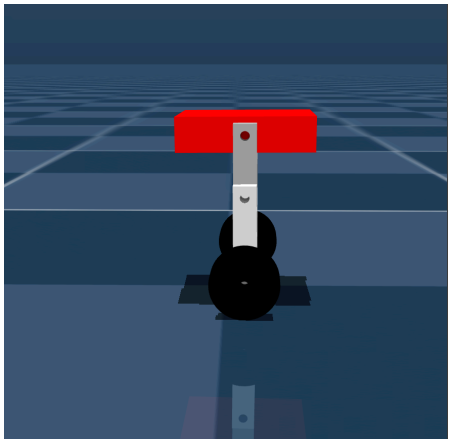
Alan Munschy (apm238) contribution

Mujoco is a physics simulator where a robot file can be added to a physically realistic environment.

The robot was modeled with a rectangular prism body, 2 double linkage legs, and wheels attached at the end of each leg. Originally, the model had a constant density, but it was too difficult to balance the robot because the mass of the prism caused the robot to fall too quickly. To fix this, the mass of each wheel was set to ~ 50 grams, the mass of the legs were ~ 10 grams, and the body had a mass of ~ 1 gram. This is unrealistic, but made the balance simulation much easier. The wheels were modeled as motor joints (so an actuation torque could be applied).

Images of the robot CAD model (1) and the robot file inside of a Mujoco environment (2)





The first goal was having the robot balancing around the origin. This was done by applying a torque to the robot wheels based on the control law:

$$u_{torque} = K_p \theta_{roll} + K_d \frac{d\theta}{dt}$$

The PD values were tuned by testing different combinations until balance was achieved. The end result was $K_p = 10,\ K_d = 5$. Since the masses were so bottom weighted, many

PD controls would work, however, having a larger K_p would cause many roll oscillations in the robot, so the chosen values were able to minimize the oscillations while maintaining balance.

A demo of the simulation is below, with the first part of the video showing the control for balancing in place, and the second half showcasing position control, where the robot will move to a desired position.

https://youtu.be/kZ9yXSptH3A

To implement position control, the general idea is that the roll angle should be pitched in the direction that the robot wants to move in. The roll angle control law used the position error $\Delta y = y_{target} - y_{current}$ to compute the desired roll angle with the law relation:

$$\theta_{desired} = -(K_{p-pos}\Delta y) + K_{d-pos}\frac{dy}{dt}$$

This $\theta_{desired}$ can be fed back into the torque control law to find the torque control input that the robot uses to maintain the $\theta_{desired}$.

$$u_{torque} = K_p(\theta_{roll} - \theta_{desired}) + K_d\frac{d\theta}{dt}$$

The gains $K_{p-pos} = 2$, $K_{d-pos} = 1.5$ did a good job for this new position control law. A higher gain could cause instability or failure as if the roll angle is too big, the robot might fall.

The scripts can be found on https://github.com/MagnetMan103/biped_control

And the README file describes how to run the simulations.

References

The main inspiration for the project:
https://www.youtube.com/watch?v=WIU8gnqQJJM

Inverted Pendulum Dynamics Modeling References
https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling
https://www.mathworks.com/help/simulink/slref/inverted-pendulum-with-animation.html

The ideal controller for this case:
https://medium.com/@svm161265/when-and-why-to-use-p-pi-pd-and-pid-controller-73729a708bb5