Optimal Control for Lunar Tumbling Robot

by

Chris Breaux

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2021 by the
Graduate Supervisory Committee:

Sze Zheng Yong, Chair
Hamid Marvi
Zhe Xu

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

JOOEE is a cube-shaped lunar robot with a simple yet robust design. JOOEE is hermetically sealed from its environment with no external actuators. Instead, JOOEE spins three internal orthogonal flywheels to accumulate angular momentum and uses a solenoid brake at each wheel to transfer the angular momentum to the body. This procedure allows JOOEE to jump and hop along the lunar surface. The sudden transfer in angular momentum during braking causes discontinuities in JOOEE's dynamics that are best described using a hybrid control framework. Due to the irregular methods of locomotion, the limited resources on the lunar surface, and the unique mission objectives, optimal control profiles are desired to minimize performance metrics such as time, energy, and impact velocity during different maneuvers. This paper details the development of an optimization tool that can handle JOOEE's dynamics including the design of a hybrid control framework, dynamics modeling and discretization, optimization cost functions and constraints, model validation, and code acceleration techniques.

TABLE OF CONTENTS

Page

LIST OF TABLES iv

LIST OF FIGURES v

INTRODUCTION 1

LITERATURE REVIEW 3

PROBLEM STATEMENT 5

METHOD 6

Hybrid Control Framework 6

Maneuvers 7

Modeling 8

Discretization 11

Cost Functions 12

Constraints 13

Model Validation 14

Code Acceleration 14

RESULTS 15

Code Acceleration 15

Swing-Up Optimization 17

Slow-Fall Optimization 22

Page

Slow-Step Optimization 28

CONCLUSIONS 31

REFERENCES 32

APPENDIX

USER MANUAL 33

Installation Procedure 34

Code Structure 34

LIST OF TABLES

Table Page

1. Default Parameters 15

2. Parallel Computing 16

3. Wind-Up Torque Multiplier 16

4. Swing-Up Optimizations 17

5. Slow-Fall Optimizations 22

6. Slow-Step Optimizations 28

7. Code Structure 34

LIST OF FIGURES

Figure Page

1. Simulation of JOOEE Balancing on a Corner 1

2. JOOEE Renders 3

3. JOOEE Simulations 4

4. Swing-Up Procedure: Wind-Up, Brake, Balance 7

5. Slow-Fall Procedure: Lean, Fall 7

6. Slow-Step Procedure: Wind-Up, Brake, Fall 8

7. JOOEE's Single-Axis FIP Model 9

8. JOOEE's Single-Axis Wind-Up Model 10

9. JOOEE's Single-Axis Brake Model 11

10. Swing-Up Time Optimization (20Hz) 18

11. Swing-Up Time Optimization (10Hz) 19

12. Swing-Up Energy Optimization (10Hz) 20

13. Slow-Fall Time Optimization (20Hz) 23

14. Slow-Fall Energy Optimization (20Hz) 24

15. Slow-Fall Time Optimization (10Hz) 25

16. Slow-Fall Energy Optimization (10Hz) 26

17. Slow-Step Time Optimization (10Hz) 29

INTRODUCTION

The Jumping Observation Orientable Environment Explorer (JOOEE) is a cube-shaped lunar robot developed by a team of engineers at NASA's Goddard Space Flight Center. JOOEE's mission is to deploy scientific payloads from a lunar lander and explore the lunar surface. JOOEE protects the payload of sensitive instruments from the abrasive lunar regolith with a simple yet robust design. JOOEE is hermetically sealed from its environment with no external actuators. Instead, JOOEE spins three internal orthogonal flywheels to accumulate angular momentum and uses a solenoid brake at each wheel to transfer the angular momentum to the body. This procedure allows JOOEE to jump and hop along the lunar surface. More precise control of the wheels allows JOOEE to swing up and balance on an edge or corner to reorient itself. My personal contributions to JOOEE's development include a numerical simulator and 3D visualization tool that I worked on during a Summer internship. Figure 1 shows an example of JOOEE balancing on a corner in the simulation environment.

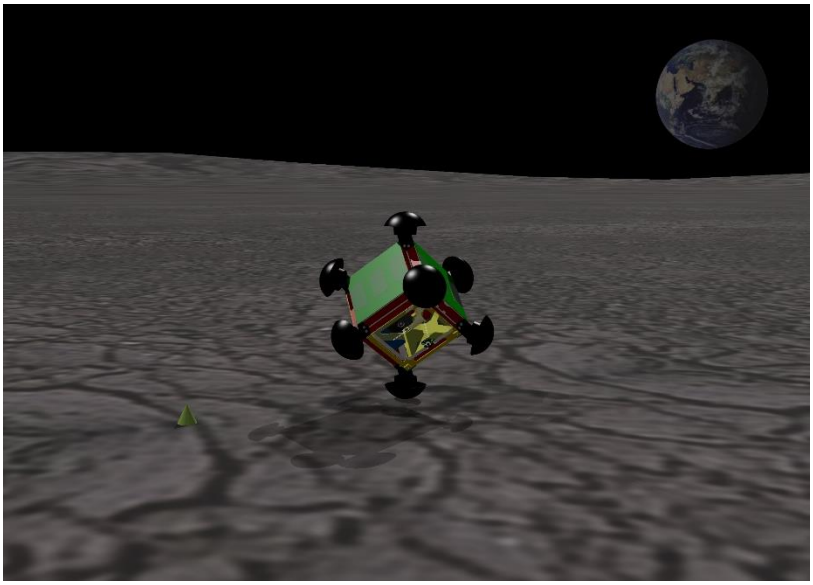

Figure 1. Simulation of JOOEE Balancing on a Corner

In this simulator, JOOEE employs a series of simple proportional–integral–derivative (PID) controllers that can be iteratively tuned to meet certain performance requirements such as overshoot, steady-state error, and settling time. However, to conserve resources and to place sensors without disturbing the lunar regolith, JOOEE may be subject to additional performance requirements to minimize time elapsed, energy, and impact velocity. Optimal control theory is better suited for handling these performance requirements than an iterative PID approach. Optimal control theory can solve for a control profile that minimizes some cost function while subject to a set of constraints. Unlike the tunable PID controller, optimal control theory requires a model of the system to solve for the optimal control profile. The sudden transfer in angular momentum during braking causes discontinuities in the JOOEE's dynamics that are best described using a hybrid control framework. A hybrid system is a discontinuous combination of continuous and discrete dynamic behavior. A bouncing ball is an example of a hybrid system because it exhibits continuous projectile motion with discrete jumps when it impacts the ground.

LITERATURE REVIEW

JOOEE was inspired by Cubli, another cube-shaped balancing robot that was developed by a team from ETH Zurich (Gajamohan, Muehlebach, Widmer, & D'Andrea, 2013). The team first developed a single-axis flywheel inverted pendulum (FIP) prototype as a proof of concept before building the full robot. The team implemented a linear controller and a tunable nonlinear controller to investigate jumping and balancing maneuvers (Muehlebach & D'Andrea, 2017).

The team at NASA followed a similar approach for JOOEE's development beginning with a single-axis FIP prototype before developing the full, untethered three-axis model. CAD renders for each prototype are shown in Figure 2.

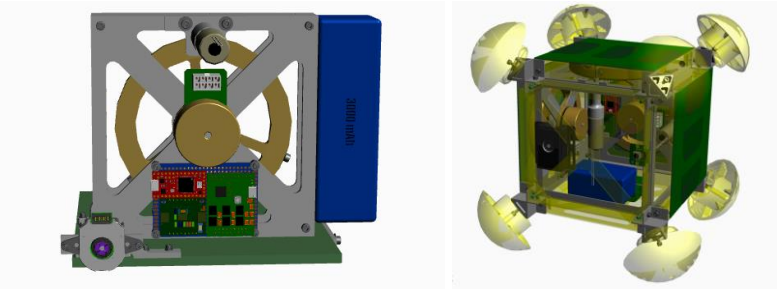

Figure 2. JOOEE Renders

The objective of my internship was to develop a numerical simulator and 3D visualization tool as a platform to develop estimation and control algorithms for the JOOEE prototypes. After building the simulator, I implemented a control algorithm consisting of simple PID controllers to enable stepping, hopping, swing-up, balancing, and slow-fall maneuvers. Simulation renders for each prototype are shown in Figure 3.

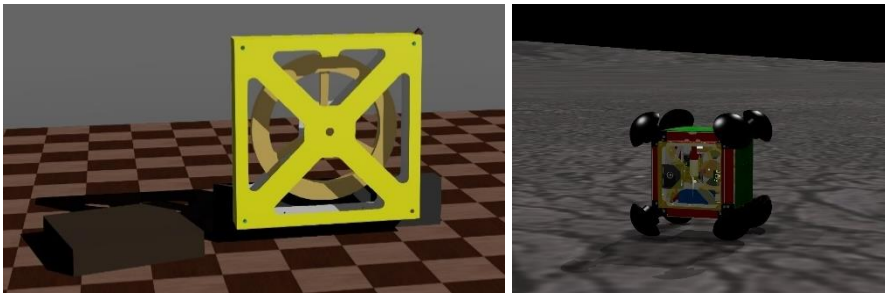

Figure 3. JOOEE Simulations

JOOEE's simulator relied on a physics engine, so no dynamic modeling had been performed yet. However, inverted pendulums are a common system, and many variations of inverted pendulum models have been developed including a one-dimension flywheel inverted pendulum (Olivares & Albertos, 2014) and Cubli's Reaction-Wheel-Based 3-D Inverted Pendulum (Muehlebach & D'Andrea, 2017).

A general framework for hybrid systems was defined in UC Santa Cruz's Model Predictive Control Framework for Hybrid Dynamical Systems (Altin, Ojaghi, & Sanfelice, 2018), and the hybrid system examples were simulated using MATLAB's Hybrid Equations Toolbox (Sanfelice, 2021). However, this toolbox is only a simulator and cannot be used to solve optimization problems. The YALMIP optimizer (Löfberg, 2004) and the Hybrid Toolbox for MATLAB (Bemporad, 2004) can use the Gurobi optimizer (Gurobi Optimization, LLC, 2021) to solve optimization problems for discretized hybrid systems. The YALMIP optimizer was selected because JOOEE's performance metrics and mode-switching conditions were easier to handle using YALMIP's versatile implementation of constraints and binary variables.

PROBLEM STATEMENT

The primary objective of this research is to develop an optimization tool that can handle JOOEE's hybrid dynamics and to use the tool to perform optimizations on three of JOOEE's single-axis maneuvers:

1. Swinging up and balancing at the unstable equilibrium
2. Falling from the unstable equilibrium and gently settling on the ground
3. Swinging up past the unstable equilibrium and gently settling on the ground

My contributions in pursuit of these objectives include:

1. Designing a hybrid control framework that handles multiple control modes and instantaneous jump maps
2. Defining three of JOOEE's maneuvers within that framework
3. Modeling the maneuvers' dynamics
4. Discretizing the continuous-time equations of motion
5. Defining performance parameters as cost functions and constraints
6. Defining constraints to handle the dynamics and the physical limits of the system
7. Validating the model with a high-resolution simulation
8. Accelerating the code to reduce computation time

METHOD

Hybrid Control Framework

Hybrid control frameworks generally define the state x and input u as elements of either a flow set C, where x is governed by a continuous-time flow map f, or as elements of a jump set D, where x is governed by a discrete-time jump map g. This general form is shown in equations (1)-(2).

$$\dot{x} = f(x,u), \quad (x,u) \in C \tag{1}$$

$$x^+ = g(x,u), \quad (x,u) \in D \tag{2}$$

A subscript q is introduced to indicate different control modes where Q is the total number of control modes. This multi-mode form is shown in equations (3)-(5).

$$\dot{x} = f_q(x,u), \quad x \in C_q \tag{3}$$

$$x^+ = g_q(x,u), \quad x \in D_q \tag{4}$$

$$q \in \{1 \quad \dots \quad Q\} \tag{5}$$

The YALMIP optimizer handles discrete-time equations, so the continuous-time equations are discretized using the forward Euler method shown in equation (6).

$$x_{k+1} = x_k + \dot{x}_k dt \tag{6}$$

Lastly, a signed notation is introduced to handle the instantaneous jumps. The flow map only affects the state during a time step. The jump map only affects the state between time steps. The multi-mode, discretized, signed format is shown in equations (7)-(9).

$$x_{k+1}^- = f_{q_k}(x_k^+, u_x), \quad x_k^+ = x_k^- \tag{7}$$

$$x_k^+ = g_{q_k}(x_k^-, u_x) \tag{8}$$

$$x_k^- = x_k^+ \xrightarrow{f_{1k}} x_{k+1}^- \xrightarrow{g_{1k+1}} x_{k+1}^+ \xrightarrow{f_{2k+1}} x_{k+2}^- = x_{k+2}^+ \tag{9}$$

Maneuvers

The swing-up procedure shown in Figure 4 begins in a resting position on the ground. The motor winds up the flywheel, accumulating angular velocity, until the solenoid brake is engaged. The flywheel instantly stops, transferring its angular momentum to the body. The solenoid brake retracts, and the motor spins the wheel to guide the body to the unstable equilibrium position. The wind-up, brake, balance sequence can be represented in the hybrid control framework as two flows separated by an instantaneous jump.

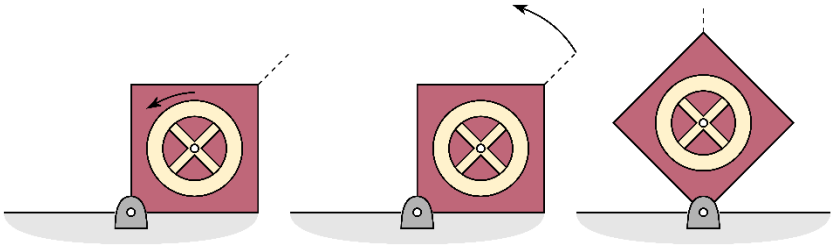

Figure 4. Swing-Up Procedure: Wind-Up, Brake, Balance

The slow-fall procedure shown in Figure 5 begins in the unstable equilibrium position. The motor spins the flywheel to lean the body to one side then applies torque against the falling direction to slow the body's fall until it settles on the ground. The lean and fall are both depend on the same FIP dynamics, so the sequence can be represented in the hybrid control framework as one flow.

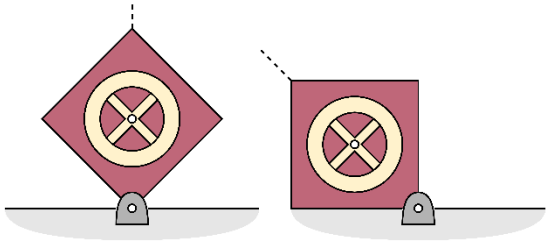

Figure 5. Slow-Fall Procedure: Lean, Fall

The slow-step procedure shown in Figure 6 combines the swing-up and slow-fall procedures. The slow-step procedure begins in a resting position on the ground. The motor winds up the flywheel, accumulating angular velocity, until the solenoid brake is engaged. The flywheel instantly stops, transferring its angular momentum to the body. The solenoid brake retracts, and the motor spins the wheel to guide the body past the unstable equilibrium position then applies torque against the falling direction to slow the body's fall until it settles on the ground. The wind-up, brake, fall sequence can be represented in the hybrid control framework as two flows separated by an instantaneous jump.

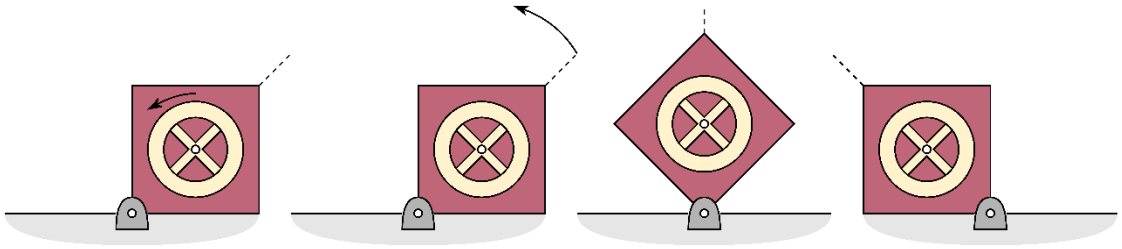

Figure 6. Slow-Step Procedure: Wind-Up, Brake, Fall

Modeling

JOOEE's single-axis maneuvers are best represented by a flywheel inverted pendulum (FIP) model with four states: the body's angle and angular velocity with respect to the Newtonian frame and the wheel's angle and angular velocity with respect to the body frame. A motor mounted to the robot body applies an input torque to spin the flywheel. The states and input are shown in equations (10)-(11) and Figure 7.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \theta_b \\ \dot{\theta}_b \\ \theta_w \\ \dot{\theta}_w \end{bmatrix} = \begin{bmatrix} \theta_b \\ \omega_b \\ \theta_w \\ \omega_w \end{bmatrix} \tag{10}$$

$$u = T \tag{11}$$

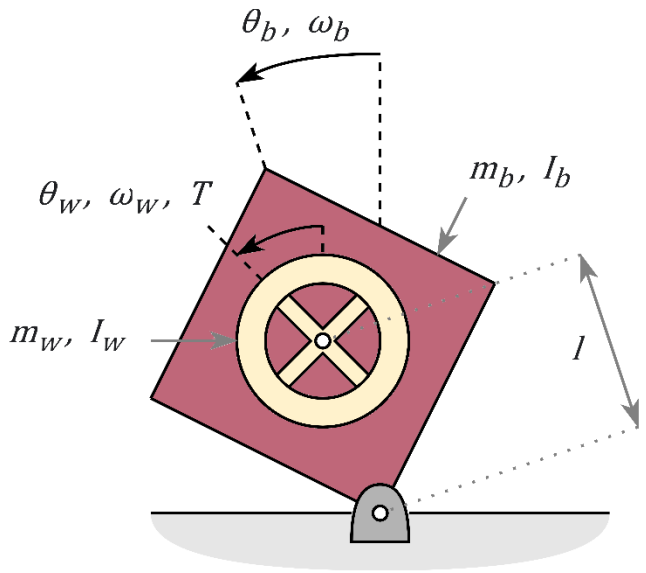

Figure 7. JOOEE's Single-Axis FIP Model

Generally, flywheel inverted pendulums are pinned to a fixed base, and the flywheel is mounted at the end of the pendulum. JOOEE is untethered, but the foot is still modeled as a pinned joint assuming that there is enough static friction to prevent slip. Unlike a typical FIP, JOOEE's motor axis is aligned with the body's center of mass such that the body's center of mass and the flywheel's center of mass are the same distance from the fulcrum, the generalized nonlinear FIP dynamics equations can be reduced to the form shown in equation (12).

$$\dot{x} = \begin{bmatrix} \dot{\theta}_b \\ \ddot{\theta}_b \\ \dot{\theta}_w \\ \ddot{\theta}_w \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{a_2 \sin(x_1) - u}{a_1 - I_w} \\ x_4 \\ \frac{a_2 \sin(x_1) - a_1 u / I_w}{I_w - a_1} \end{bmatrix} \tag{12}$$

where

$$I_b' = I_b + m_b l^2 \tag{13}$$

$$a_1 = m_w l^2 + I_b' \tag{14}$$

$$a_2 = (m_b l + m_f l) g \tag{15}$$

The wind-up model shown in Figure 8 uses the same states and input defined in equations (10)-(11) and Figure 7.

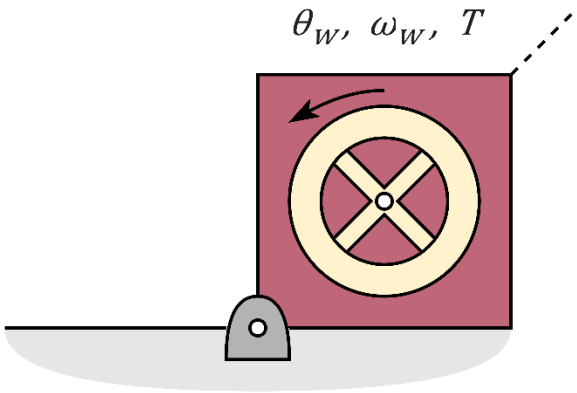

Figure 8. JOOEE's Single-Axis Wind-Up Model

When JOOEE is at rest on the ground, the motor can spin the flywheel but does not have enough torque to break contact with the ground, so the body's angle and angular velocity are constant. As the motor spins the flywheel, the wheel's angle and angular velocity change according to the motor wind-up equation (16).

$$\dot{x} = \begin{bmatrix} \dot{\theta}_b \\ \ddot{\theta}_b \\ \dot{\theta}_w \\ \ddot{\theta}_w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ x_4 \\ \frac{u}{I_w} \end{bmatrix} \tag{16}$$

The brake model shown in Figure 9 uses the same states and input defined in equations (10)-(11) and Figure 7.

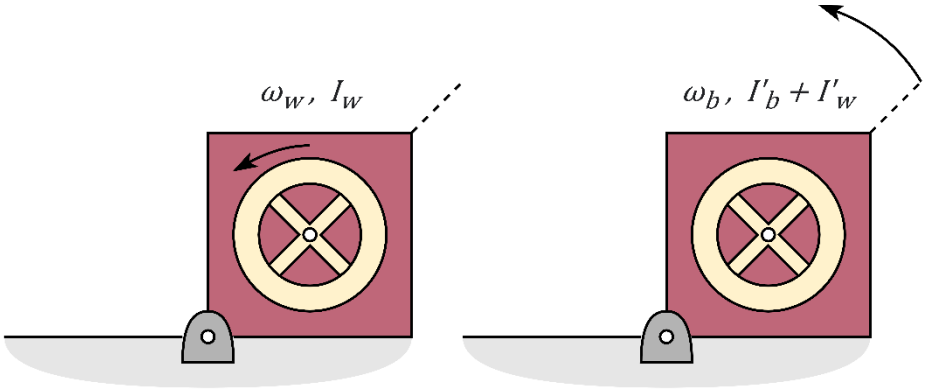

Figure 9. JOOEE's Single-Axis Brake Model

When the solenoid brake is engaged, the wheel instantly stops, transferring its angular momentum through the solenoid to the body. The body's angular velocity immediately following the transfer of angular momentum is described by the jump equation (17).

$$x^+ = \begin{bmatrix} x_2 \\ \frac{I_w x_4}{a_1 + I_w} \\ x_3 \\ 0 \end{bmatrix} \tag{17}$$

Discretization

The YALMIP optimizer handles discrete-time systems easier than continuous-time systems, so the continuous-time equations above were converted to discrete-time equations using the forward Euler approximation with step size dt over N time steps. The nonlinear FIP model becomes equation (18).

$$x_{k+1} = \begin{bmatrix} x_{1_k} + x_{2_k} dt \\ x_{2_k} + \frac{a_2 \sin(x_{1_k}) - u_k}{a_1 - I_w} dt \\ x_{3_k} + x_{4_k} dt \\ x_{4_k} + \frac{a_2 \sin(x_{1_k}) - a_1 u_k / I_w}{I_w - a_1} \end{bmatrix}, \quad k = 0, \ldots N-1 \tag{18}$$

The wind-up model shown in equation (19) differs slightly from the forward Euler approximation definition because the body angular velocity is locked to zero since it is settled on the ground during the wind-up phase.

$$x_{k+1} = \begin{bmatrix} x_{1_k} \\ 0 \\ x_{3_k} + x_{4_k} dt \\ x_{4_k} + \frac{u_k}{I_w} dt \end{bmatrix}, \quad k = 0, \dots N-1 \tag{19}$$

The brake model shown in equation (20) is already in a usable discrete-time form, so the forward Euler approximation is not necessary. However, the brake model is not the same form as the forward Euler approximation. Instead, the instantaneous jump that occurs during the transfer of angular momentum is defined explicitly. This time step is removed after the optimization as if the jump happened in an instant between time steps.

$$x^+ = \begin{bmatrix} x_2^- \\ \frac{I_w x_4^-}{a_1 + I_w} \\ x_3^- \\ 0 \end{bmatrix} \tag{20}$$

Cost Functions

To evaluate the minimum time solution of a simulation, the cost function is constant as shown in equation (21). Instead, the optimization is iterated with a decreasing time interval to find the last valid solution.

$$cost_{time} = 1 \tag{21}$$

To evaluate the minimum energy solution of a simulation, the cost function is equal to the sum of the magnitude of the input at each time step as shown in equation (22).

$$cost_{energy} = \sum_{k=1}^{N} |u_k| \tag{22}$$

Constraints

The optimization tool loops through the set of time steps applying constraints to the state and input variables that describe the dynamics and the physical limits of the system. Each dynamic model is defined as a set of constraints on the next state based on a function of the current state and input. Each dynamic model must be activated by a binary variable evaluated at each time step. A constraint is added such that the sum of the binary variables must be 1 at each time step so that only one model is applied at a time. This binary variable is also used to activate physical limits for each model such as maximum input motor torque. The wind-up model has a special constraint such that torque can only be applied in the desired swing-up direction. The FIP model has no constraint on the direction of the torque. Outside of the loop, constraints are added that force the initial and final states to their desired values.

The swing-up procedure has additional constraints. After the loop, the sum of the binary variable that activates the brake is constrained to 1 so that the brake is only applied once. Inside of the loop, constraints are added that prohibit the wind-up or balance models from being activated depending on if the jump has already occurred. This ensures that the models are activated in the order: wind-up, brake, balance.

The slow-fall procedure does not switch models, but it does constrain the final body angular velocity within some threshold. This threshold can be iteratively reduced like the minimum time solution to find the minimum impact velocity.

The slow-step procedure uses all of the swing-up procedure's model-switching constraints and the slow-fall procedure's impact velocity constraint.

Model Validation

Optimal control theory is very computationally expensive, so large time steps may be used to reduce the number of variables. To validate the optimal control solutions, the low-frequency control profile is passed as a step function through a high-frequency ordinary differential equation (ODE) simulator. The states are evaluated over the time interval and plotted with the optimizer's solution for comparison. The simulator's solution serves as the true value. Large discrepancies between the optimizer's solution and the simulator's true value invalidate the optimized model. These discrepancies are caused by the use of the forward Euler discretized dynamics in the optimizer and can be reduced by reducing the step size. If the optimizer does not find a valid solution, then the model validation is skipped. Increasing the number of time steps or easing constraints can help the optimizer find a valid solution.

Code Acceleration

Again, optimal control theory is very computationally expensive, so two code acceleration methods are used to improve the speed of the process. To accelerate the iterative process for minimum time and minimum impact velocity, batches of optimizations can be run in parallel, given a set of inputs such as control mode, cost function, steps, step size, initial and final state, and torque constraints. To accelerate each individual optimization, a wind-up torque multiplier is enabled as an optional input. This eases the torque constraint on the wind-up model, reducing the number of variables and time spent in the wind-up phase. The wind-up phase is relatively uninteresting because the motor applies maximum torque up to some critical angular velocity before braking. This does not affect the more interesting, nonlinear FIP dynamics.

RESULTS

All optimizations and simulations were performed using the default parameters in Table 1. These parameters are approximate values taken from the models used in JOOEE's numerical simulator.

Table 1

Default Parameters

Parameter	Symbol	Value	Unit
Body Mass	m_b	5	kg
Body Inertia	I_b	0.05	kg·m^2
Wheel Mass	m_w	0.75	kg
Wheel Inertia	I_w	0.002	kg·m^2
Center of Mass Distance	l	0.18	m
Max Motor Torque	T_{max}	0.134	N·m
Lunar Gravity	g	1.62	m/s^2

Maximum input motor torque and the wind-up torque multiplier are supplied as inputs to the optimizer. Due to time constraints, a maximum motor torque of 1Nm and a wind-up torque multiplier of 5 were used for all optimizations to reduce computation time. Low frequency solutions were solved for each control mode and cost function, but the high frequency solutions for the swing-up and slow-step ran too long and were cancelled.

Code Acceleration

Arbitrary optimizations were performed to test the efficacy of the code acceleration methods. Table 2 shows the results of the parallel computing technique accelerating a batch of simulations. Table 3 shows the results of the wind-up torque multiplier technique accelerating a single optimization.

Table 2

Parallel Computing

Method	Simulations	Threads	CPU Time
For Loop	4	1	109.9
Parallel Loop	4	4	35.3

The computer used for this test has six cores, so a batch of four optimizations was performed to maximize parallelization while leaving enough cores for the computer to still function normally while the optimizations run in the background. This parallel batch was compared to the same batch of optimizations performed in series in a normal for loop. Using four times as many cores, the parallel computing method performed a batch of optimizations three times faster than a regular for loop.

Table 3

Wind-Up Torque Multiplier

Method	Brake Velocity	Time Steps	CPU Time
1x Wind-up	250rad/s	17	314s
5x Wind-up	250rad/s	10	22s

Two optimizations were performed with the same parameters except for the five times wind-up torque multiplier applied to one optimization. This reduced the number of time steps by less than half, but the process time for the accelerated optimization was fourteen times faster. Despite the torque multiplier, both optimizations wound the wheel up to the same angular velocity before braking, preserving the dynamics after wind-up.

Swing-Up Optimization

Table 4 shows the batch of swing-up optimizations and their inputs. Low-frequency optimizations were performed to minimize time and energy. One high-frequency optimization was performed to minimize time. The high-frequency swing-up energy optimization ran too long and was cancelled. JOOEE's previous control algorithm calculated the minimum wheel angular velocity that would transfer just enough energy to the body to swing the body up to the equilibrium position. This swing-up velocity is indicated on the wheel angular velocity plot for comparison.

Table 4

Swing-Up Optimizations

Figure	Cost	Time Steps	Step Size	CPU Time
Figure 10	Time	20	0.1s	24s
Figure 11	Energy	20	0.1s	2m 42s
Figure 12	Time	40	0.05s	3m 9s

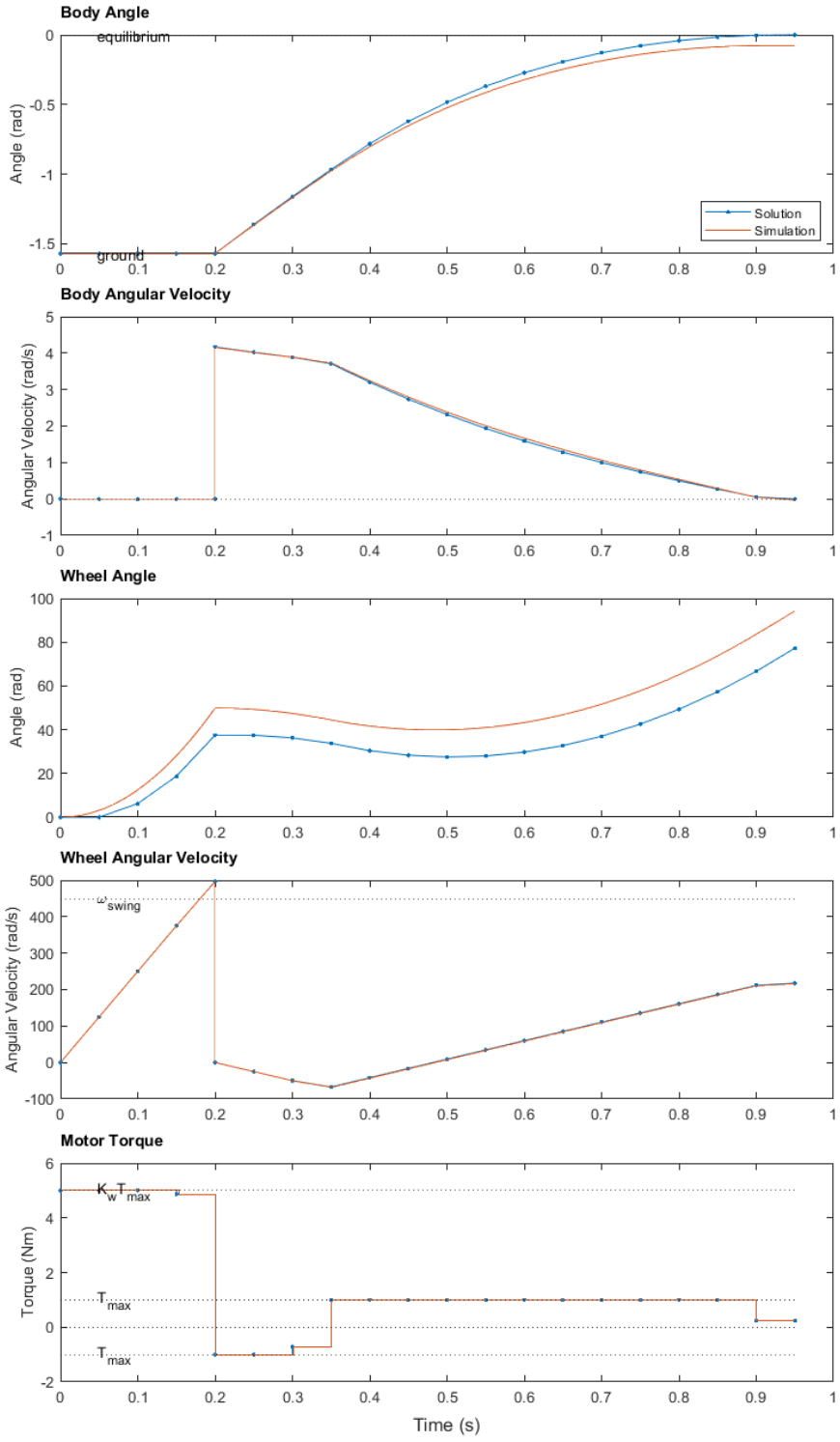

Figure 10. Swing-Up Time Optimization (20Hz)

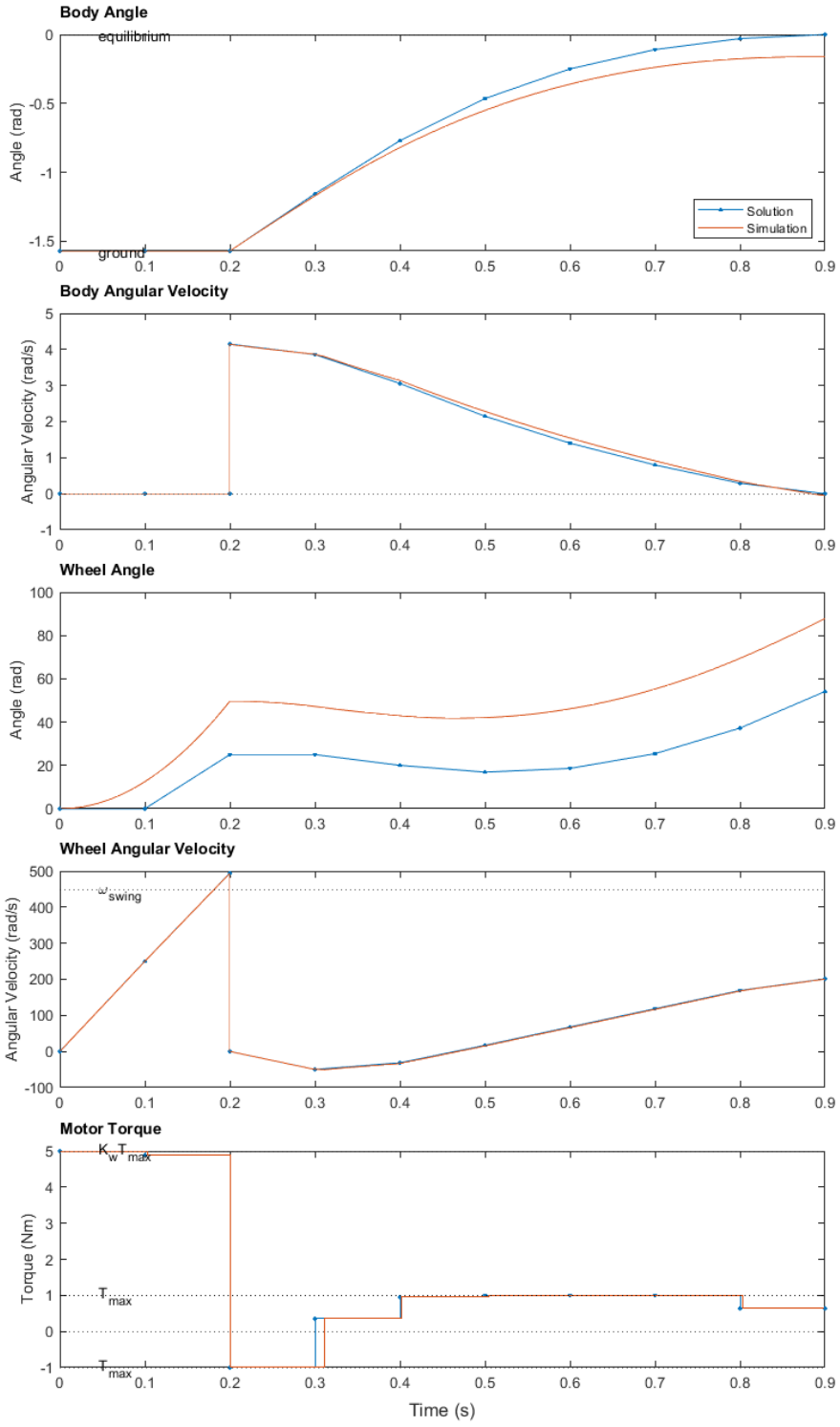

Figure 11. Swing-Up Time Optimization (10Hz)

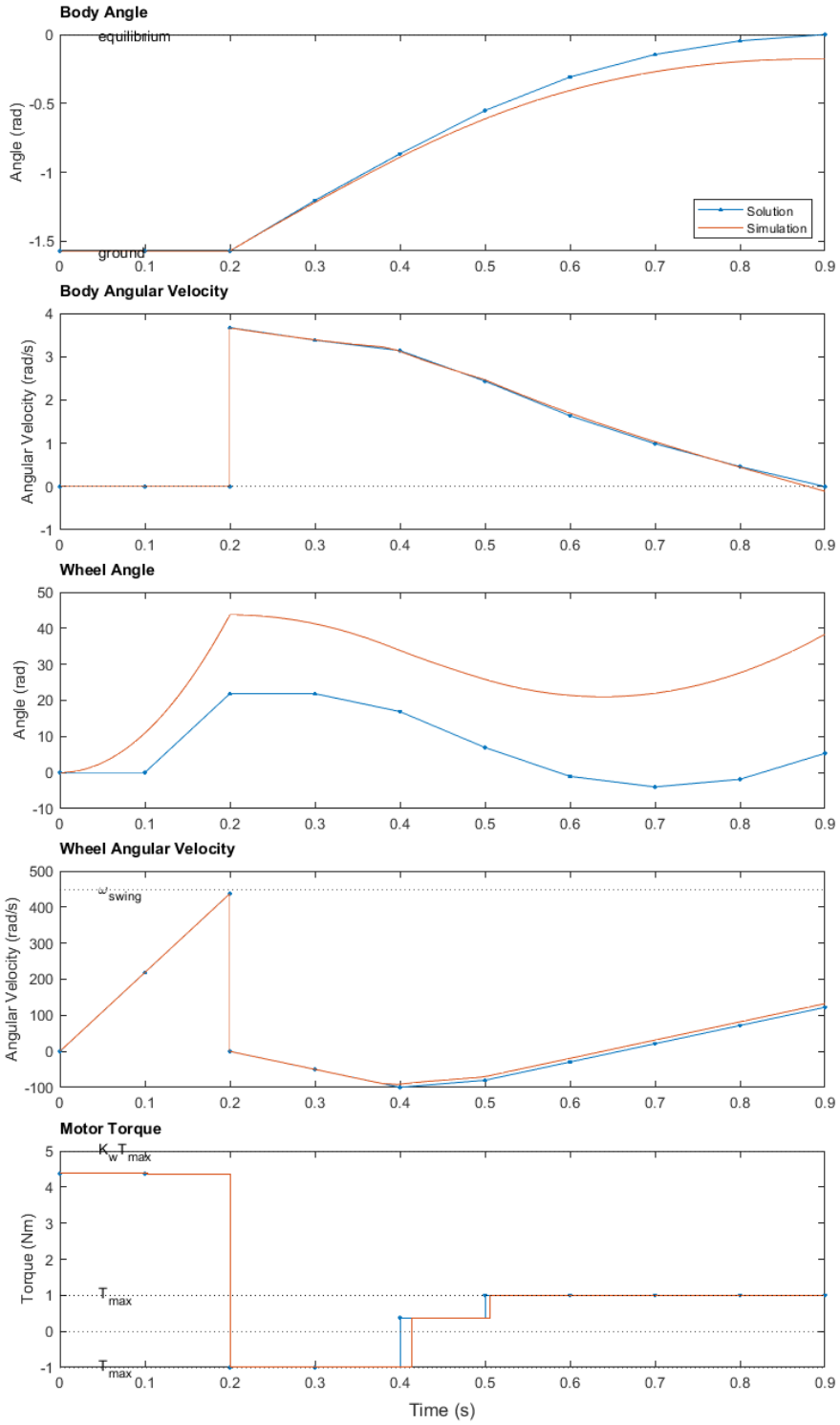

Figure 12. Swing-Up Energy Optimization (10Hz)

In all three optimizations, the body's angle and angular velocity successfully settle at the equilibrium point. The low-frequency simulations have drift in the body angle which causes a steady state error. The high-frequency solution has significantly less drift. In all three solutions, the motor winds up with maximum torque to the minimum swing-up velocity, brakes, and applies more torque in the opposite direction to accelerate the body towards the equilibrium position as it swings up, and then reverses the torque again to slow itself down to a stop at the equilibrium position. The wheel reached the minimum swing-up velocity, so the body had enough energy to reach the equilibrium point with no further input. However, to minimize time, the additional torque was applied to accelerate and slow the body to the equilibrium position.

The minimum energy solution used the same number of time steps as the minimum time solution, so the behavior is very similar. The minimum energy solution applied less than the maximum torque during the wind-up and had a more consistent torque as it guided the body to the equilibrium position. If the minimum energy solution were allowed more time steps, then it would likely coast more after braking instead of accelerating and slowing back down.

Slow-Fall Optimization

Table 5 shows the batch of slow-fall optimizations and their inputs. Four optimizations were performed to minimize time and energy over different time intervals. The regular-fall impact velocity, where the body is allowed to fall with no motor input to slow it down, was calculated. This freefall angular velocity and the maximum impact velocity threshold are indicated on the body angular velocity plot for comparison.

Table 5

Slow-Fall Optimizations

Figure	Cost	Time Steps	Step Size	CPU Time
Figure 13	Time	40	0.05s	8s
Figure 14	Energy	40	0.05s	2h 50m 55s
Figure 15	Time	20	0.05s	1s
Figure 16	Energy	20	0.05s	9m 4s

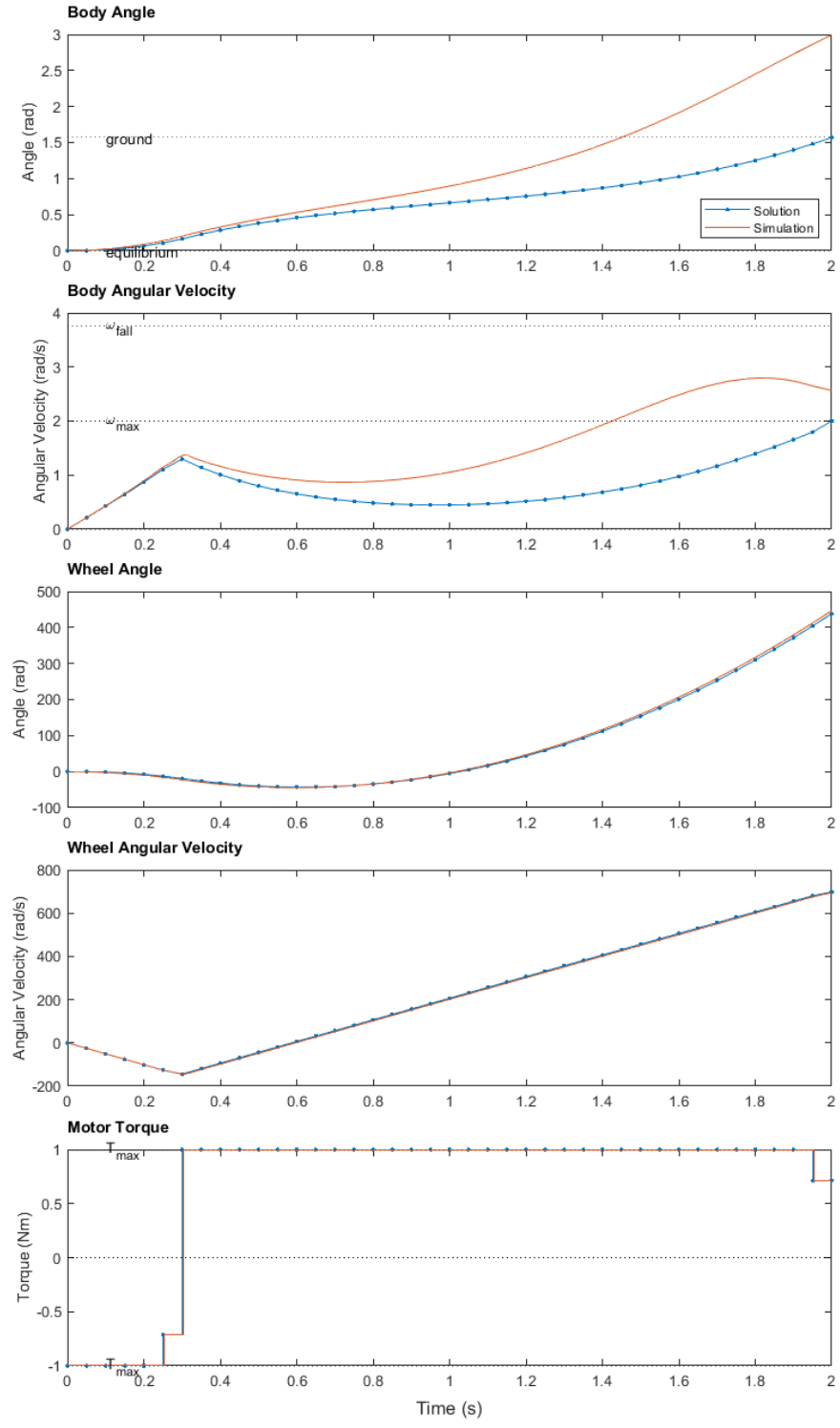

Figure 13. Slow-Fall Time Optimization (20Hz)

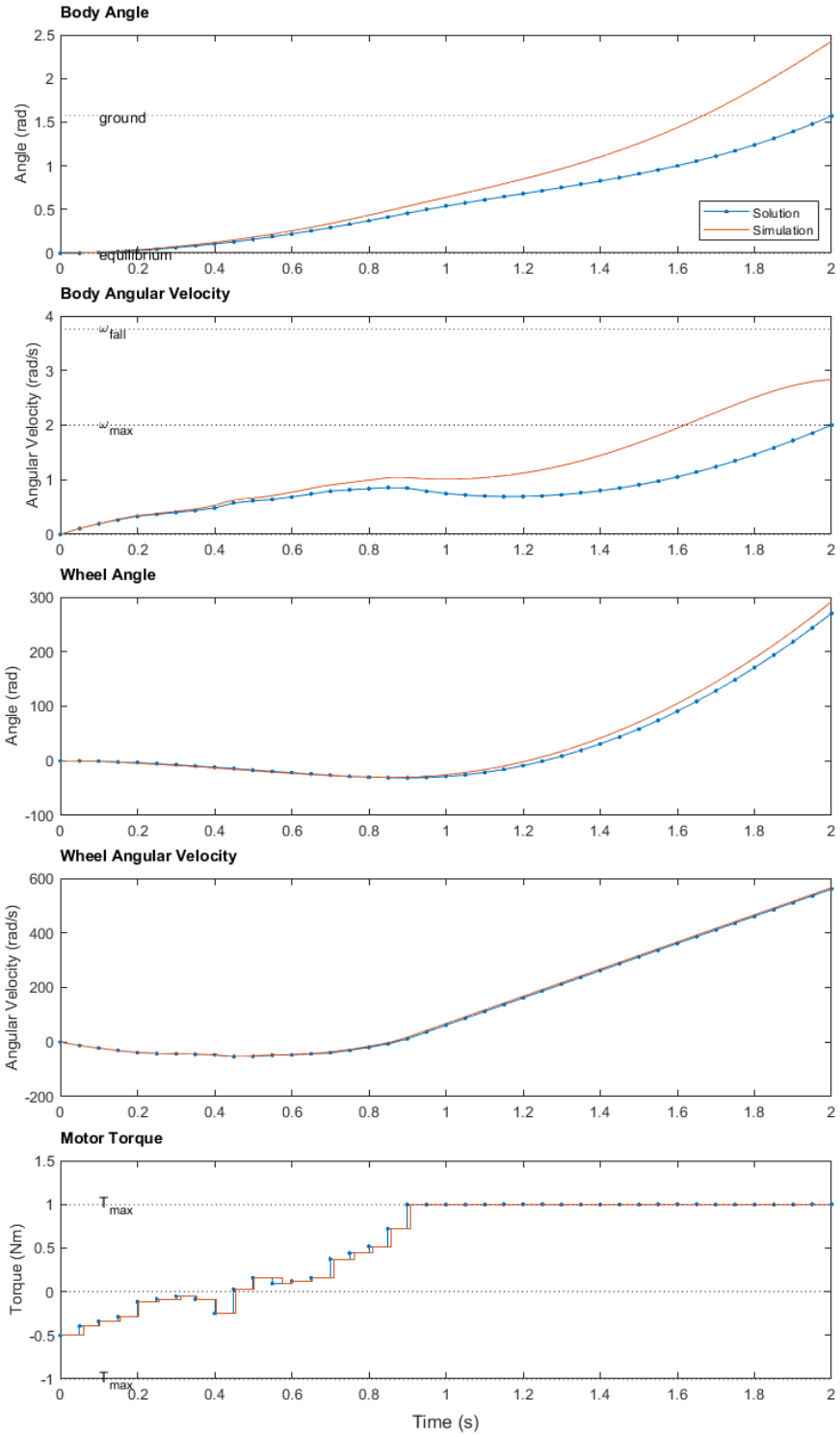

Figure 14. Slow-Fall Energy Optimization (20Hz)

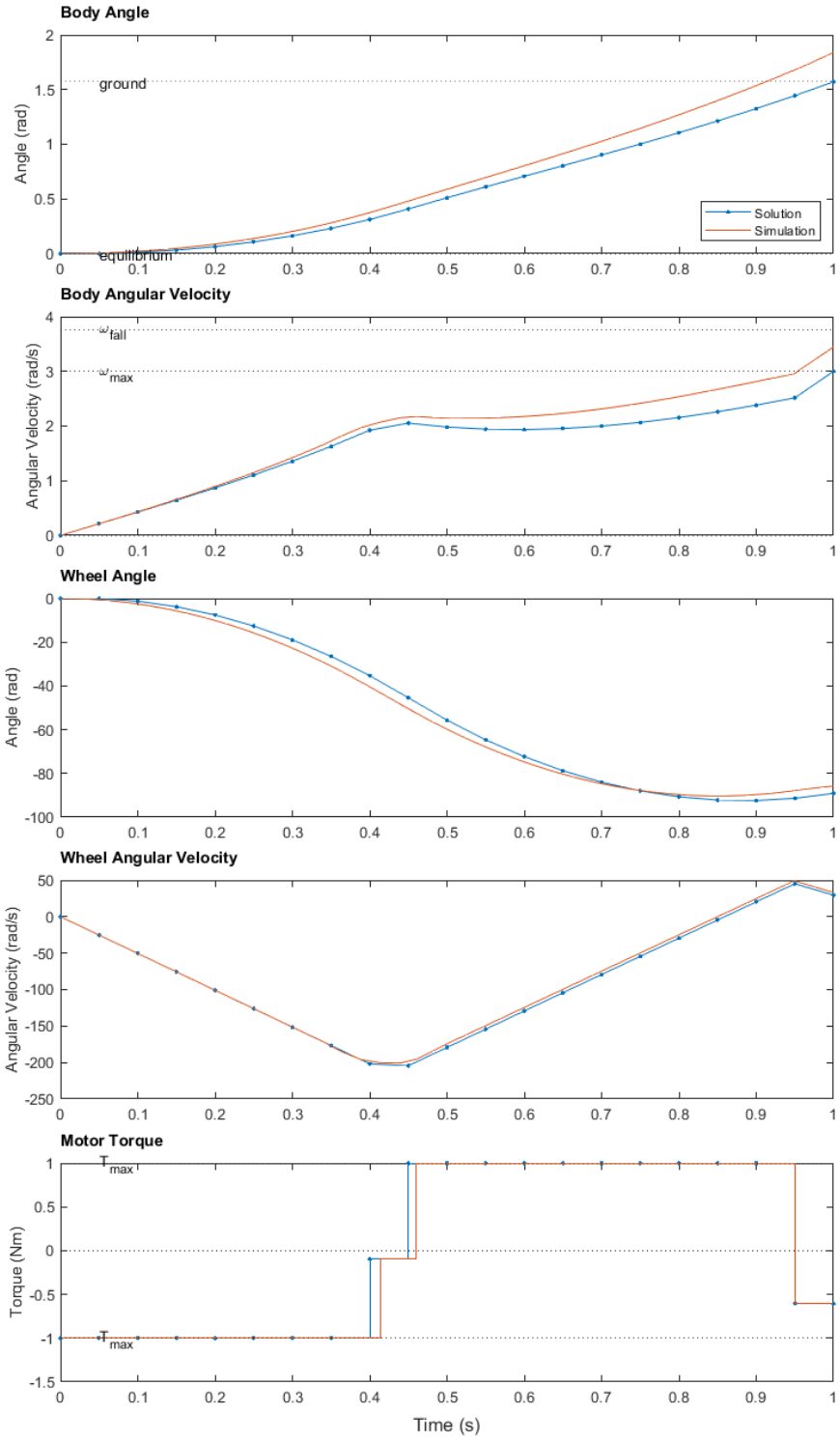

Figure 15. Slow-Fall Time Optimization (10Hz)

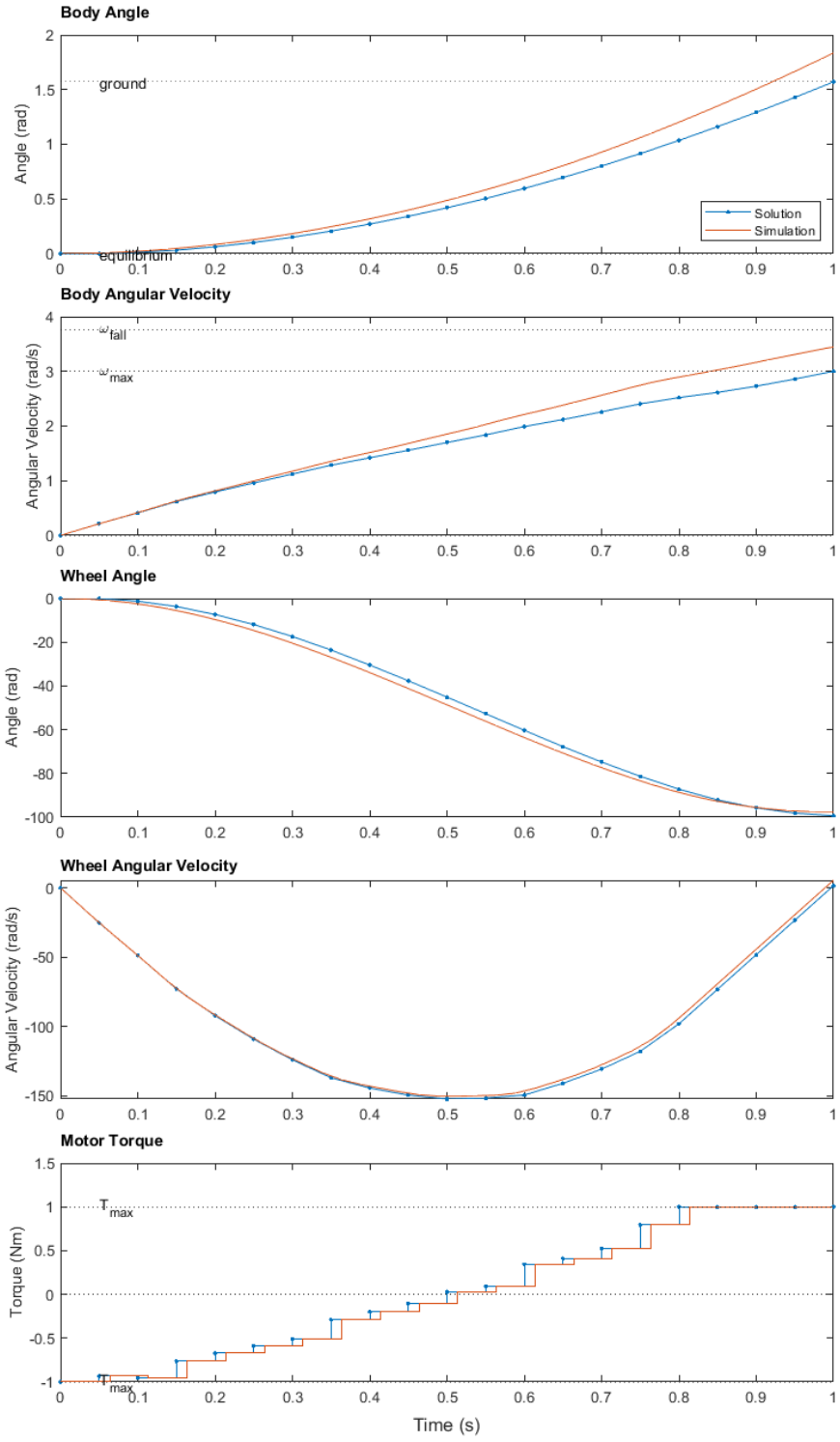

Figure 16. Slow-Fall Energy Optimization (10Hz)

In all four optimizations, the body's angle and angular velocity successfully reach the ground within the maximum impact velocity threshold. There is significant drift in the longer optimizations but less drift in the shorter time intervals. While the drift decreased with increasing frequency in the previous example, here, the drift increases with a greater time interval at the same frequency. The simulation solution reaches the ground before the optimization solution, but the impact velocity is still near the threshold even though the model is inaccurate. The minimum time solution is, again, very aggressive, switching between maximum torque values. The minimum energy solution has a very different behavior, steadily increasing the to the maximum torque as it falls.

Slow-Step Optimization

Table 6 shows the batch of slow-step optimizations and their inputs. A low-frequency optimization was performed to minimize time. The low-frequency slow-step energy optimization, and the high-frequency slow-step time and energy optimizations ran too long and were cancelled. JOOEE's previous control algorithm calculated the minimum wheel angular velocity that would transfer just enough energy to the body to swing the body up to the equilibrium position. This swing-up velocity is indicated on the wheel angular velocity plot for comparison. The regular-fall impact velocity, where the body is allowed to fall with no motor input to slow it down, was calculated. This regular-fall angular velocity and the maximum impact velocity threshold are indicated on the body angular velocity plot for comparison.

Table 6

Slow-Step Optimizations

Figure	Cost	Time Steps	Step Size	CPU Time
Figure 17	Time	20	0.1s	37s

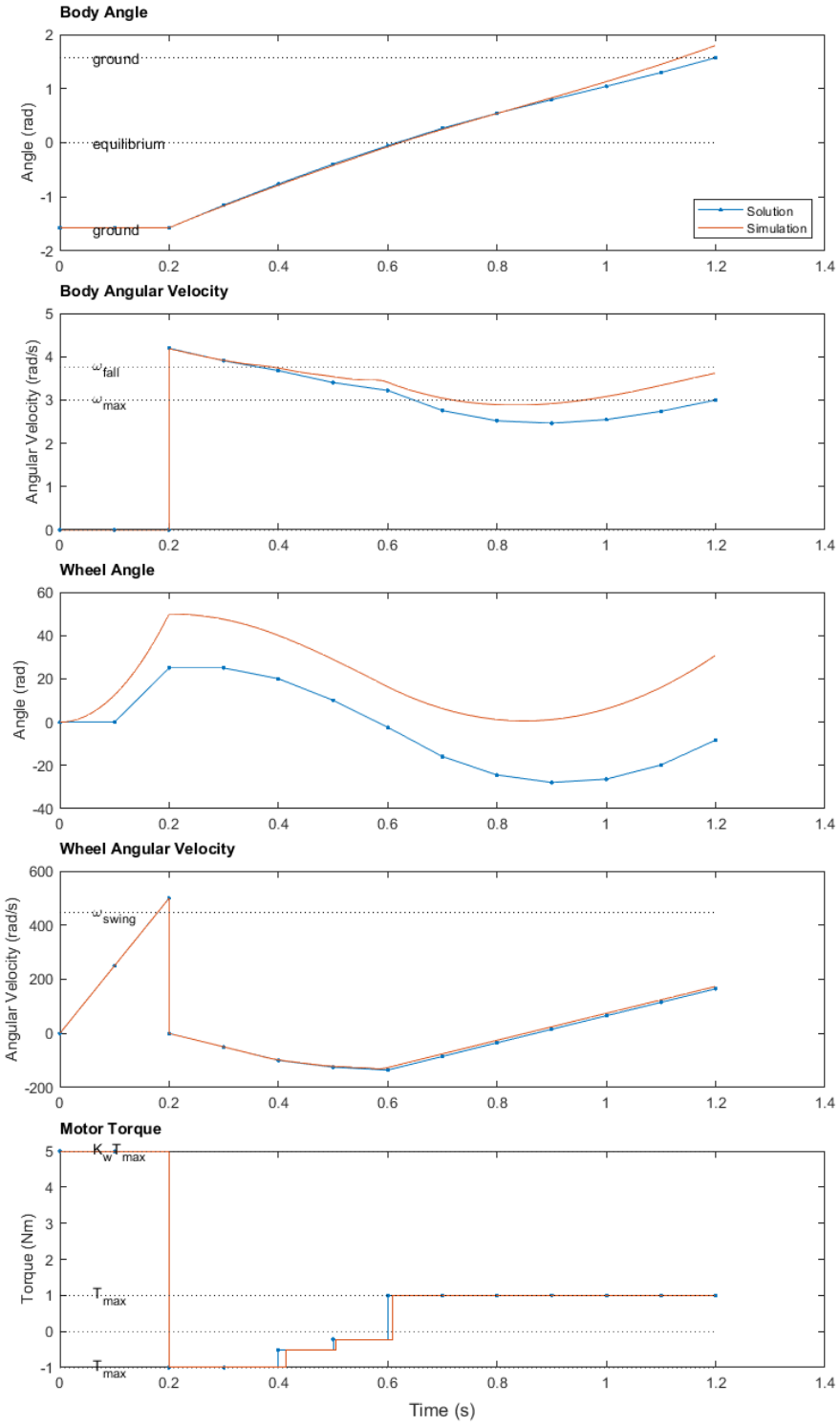

Figure 17. Slow-Step Time Optimization (10Hz)

In all three optimizations, the body's angle and angular velocity successfully reach the ground within the maximum impact velocity threshold. The simulation solution shows very little drift in the body angle, but there is significant drift in the body angular velocity. The impact velocity is greater than the threshold, near the regular-fall impact velocity. The torque profile is very aggressive, applying maximum torque before and after braking to swing past the equilibrium point and then reversing to slow itself down as it falls.

CONCLUSIONS

The hybrid control framework handles multiple control modes and instantaneous jump maps when used with the YALMIP optimizer. Three of JOOEE's single-axis maneuvers are defined within that framework. The dynamics comprising those maneuvers were modelled and discretized for usage within the framework and the optimizer. A set of performance parameters were defined as cost functions and constraints. Additional constraints were added to handle the dynamics and the physical limits of the system. The code was accelerated to reduce process time, and after optimization, models were validated with a high-resolution simulation. The optimizations ultimately produced valid solutions with improved performance and revealed distinct behavior for different cost functions.

In its current state, the tool can be used to solve optimizations offline for sets of initial conditions. These control profiles can be saved onboard, selected, and applied depending on the initial conditions. Alternatively, the optimal control profiles and their unique behaviors can be used to inspire new control algorithms and educate the tuning of simpler control methods to produce a similar response. The code can be accelerated further by parallelizing on a larger cluster or restructuring the mode-switching conditions to use fewer binary variables. Faster code might reduce the computation costs to enable model predictive control, which is a real-time optimization performed at each time step to find the best input for the immediate next time step. Model predictive control would reduce the drift in the simulated response since the optimization is reevaluated at each time step. Alternatively, switching from an optimal control profile to a feedback controller near the equilibrium position would reduce computation time and error. In the future, I would like to extend this work to the 3D JOOEE and to include more maneuvers.

REFERENCES

Altin, B., Ojaghi, P., & Sanfelice, R. G. (2018). A Model Predictive Control Framework for Hybrid Dynamical Systems. *IFAC PAPERSONLINE, 51*(20), 128-133. doi:10.1016/j.ifacol.2018.11.004

Bemporad, A. (2004). Hybrid Toolbox - User's Guide. Retrieved from http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox

Gajamohan, M., Muehlebach, M., Widmer, T., & D'Andrea, R. (2013). The Cubli: A reaction wheel based 3D inverted pendulum. *2013 European Control Conference (ECC)*, 268-274. doi:10.23919/ECC.2013.6669562

Gurobi Optimization, LLC. (2021). Retrieved from Gurobi Optimizer Reference Manual: http://www.gurobi.com

Löfberg, J. (2004). YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *In Proceedings of the CACSD Conference.* Taipei, Taiwan.

Muehlebach, M., & D'Andrea, R. (2017, Jan). Nonlinear Analysis and Control of a Reaction-Wheel-Based 3-D Inverted Pendulum. *IEEE Transactions on Control Systems Technology, 25*(1), 235-246. doi:10.1109/TCST.2016.2549266

Olivares, M., & Albertos, P. (2014). Linear Control of the flywheel inverted pendulum. *ISA Transactions, 53*(5), 1396-1403. doi:10.1016/j/isatra.2013.12.030

Sanfelice, R. (2021). *Hybrid Equations Toolbox v2.04*. Retrieved from MATLAB Central File Exchange: https://www.mathworks.com/matlabcentral/fileexchange/41372-hybrid-equations-toolbox-v2-04

APPENDIX A

USER MANUAL

Installation Procedure

1. Install MATLAB: https://www.mathworks.com/products/matlab.html
2. Install Parallel Computing Toolbox: https://www.mathworks.com/products/parallel-computing.html
3. Install YALMIP: https://yalmip.github.io/tutorial/installation/
4. Install Gurobi: https://www.gurobi.com/downloads/gurobi-optimizer-eula/
5. Download code: https://github.com/ChristopherDBreaux/Optimal-Control-for-Lunar-Tumbling-Robot
6. Add YALMIP and Gurobi to path in MAIN.m

Code Structure

The MAIN file handles batches of optimizations and data processing. A YALMIP-based optimization function is defined for each maneuver. An ODE function is defined for each dynamic model. The hybrid simulator handles multiple ODE functions. The continuous simulator handles one ODE function.

Table 7

Code Structure

File	Type	Description
MAIN.m	Script	This is the main script. Edit the list of inputs. Run "Optimizer" section to run a batch of inputs in parallel. Run "Plotter" section to select a previous batch for plotting.
swingup_opt.m	Optimizer	YALMIP function for swingup
slowfall_opt.m	Optimizer	YALMIP function for slowfall
slowstep_opt.m	Optimizer	YALMIP function for slowstep
hybrid_sim.m	Simulator	ode45 simulator for swingup and slowstep
continuous_sim.m	Simulator	ode45 simulator for slowfall
balance_odefun.m	ODE	Nonlinear FIP ode function
windup_odefun.m	ODE	Wind-up ode function
get_properties.m	function	Contains default system parameters