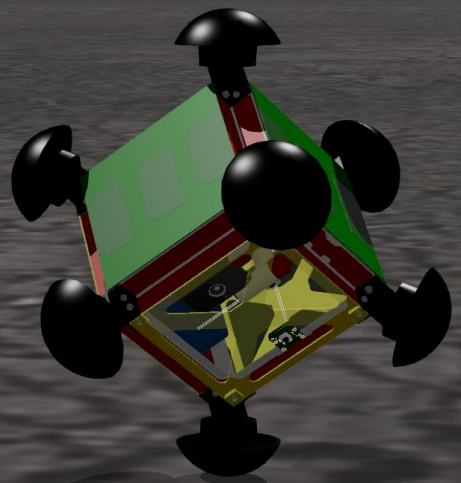
Optimal Control for Lunar Tumbling Robot



Master's Thesis by Chris Breaux

Supervised by Sze Zheng Yong, Chair, Hamid Marvi, Zhe Xu

Ira A. Fulton Schools of Engineering
Arizona State University

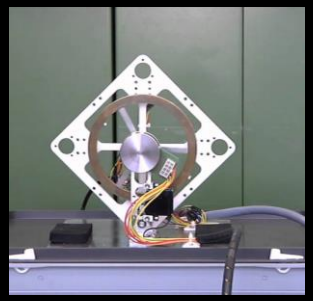
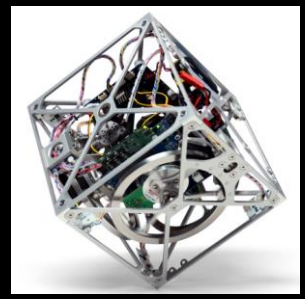
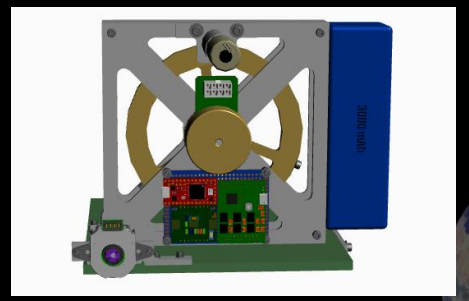
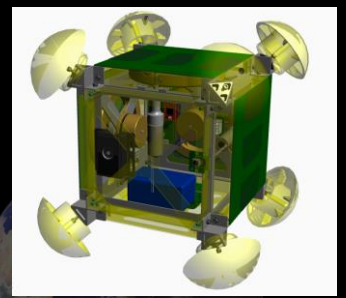
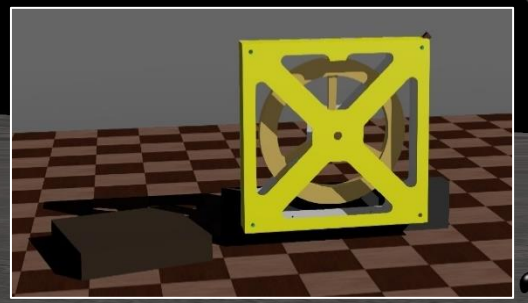
Contents

Introduction: Motivation

- Jumping Observation Orientable Environment Explorer (JOOEE)
  - Cube-shaped lunar robot
  - Developed by a team at NASA GSFC
- Hermetically sealed, no external actuators
  - Three internal orthogonal flywheels
  - Solenoid brake at each wheel
- Accumulate angular momentum, transfer to the body
  - Stepping, hopping, swing-up, balancing, and slow-fall
- Sudden transfer in angular momentum during braking causes discontinuities
  - Best described using a hybrid control framework
- Optimal control profiles are desired to minimize performance metric
  - Time
  - Energy
  - Impact velocity

Introduction: Previous Work

- Cubli Robot
  - 1D prototype
  - 3D prototype
- NASA's JOOEE Prototypes
  - Similar approach
  - Legged design
- My Numerical Simulator
  - Simple PID algorithms
- Inverted Pendulum Modeling
  - Flywheel inverted pendulum
  - Reaction-wheel-based 3D inverted pendulum
- Optimization
- Hybrid Systems
  - Model Predictive Control Framework for Hybrid Dynamical Systems


Introduction: Problem Statement

- Develop an optimization tool that can handle JOOEE's hybrid dynamics
- 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

Method: Hybrid Control Framework

- General Form
  - Flow set C, flow map f
  - Jump Set D, flow map g

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

$$x^+ = g(x,u), \quad (x,u) \in D$$

- Multi-Mode Form
  - Mode q

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

$$x^+ = g_q(x,u), \quad x \in D_q$$

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

- Forward Euler Approximation
  - Easier to solve

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

- Multi-Mode, Discretized, Signed Format
  - Signs indicate jumps

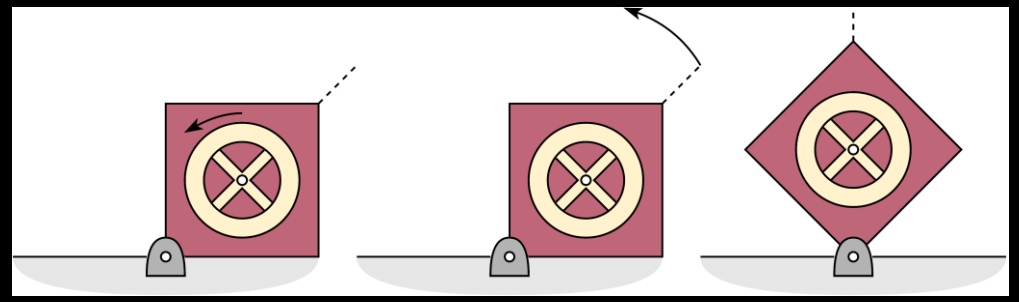
$$x_{k+1}^- = f_{q_k}(x_k^+, u_x),\ x_k^+ = x_k^-$$

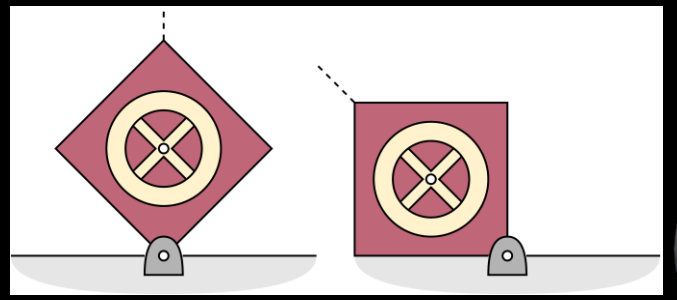
$$x_k^+ = g_{q_k}(x_k^-, u_x)$$

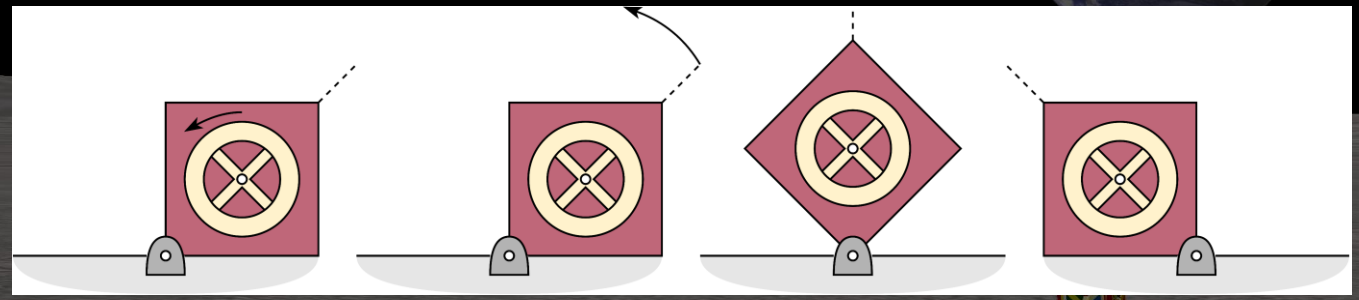
$$x_k^- = x_k^+ \xrightarrow{f_{1k}} x_{k+1}^- \xrightarrow{g_{1k+1}} x_{k+1}^+ \xrightarrow{f_{2k+1}} x_{k+2}^- = x_{k+2}^+$$

Method: Maneuvers

- Swing-Up
  - Wind-up
  - Brake
  - Balance
- Slow-Fall
  - Lean
  - Fall
- Slow-Step
  - Wind-up
  - Brake
  - Fall



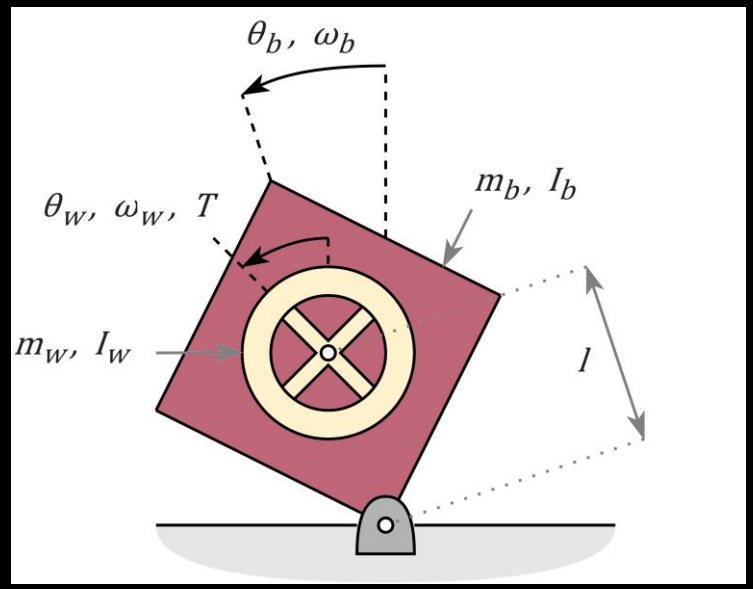




Method: Modeling

- States
  - Body angle
  - Body angular velocity
  - Wheel angle
  - Wheel angular velocity

- Input Torque

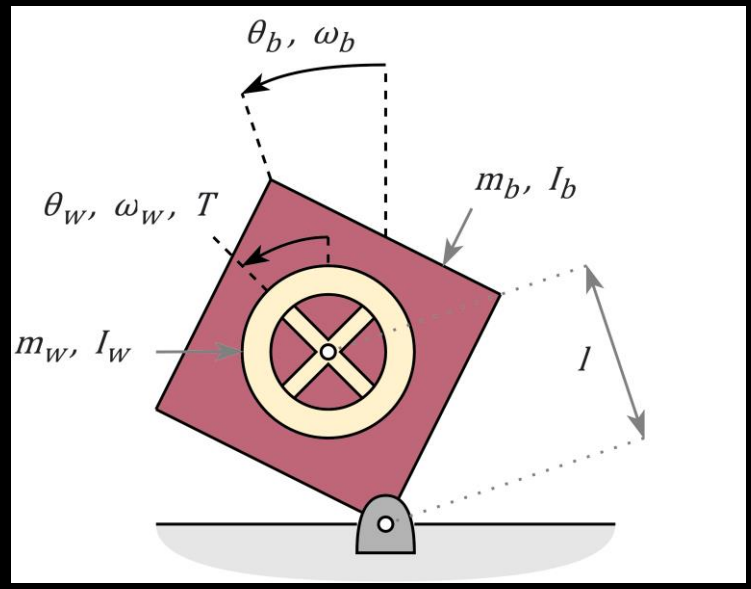


$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \theta_b \\ \dot{\theta}_b \\ \theta_w \\ \dot{\theta}_w \end{bmatrix} = \begin{bmatrix} \theta_b \\ \omega_b \\ \theta_w \\ \omega_w \end{bmatrix}$$

$$u = T$$

Method: Modeling

- Single-Axis FIP
  - Nonlinear



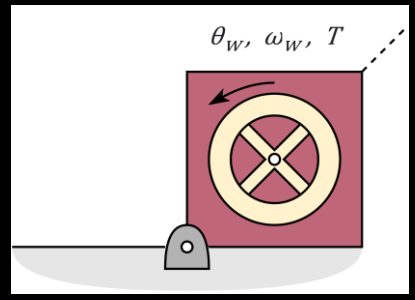
$$\dot{x} = \begin{bmatrix} \dot{\theta}_b \\ \ddot{\theta}_b \\ \dot{\theta}_w \\ \ddot{\theta}_w \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{a_2 \sin(x_1) - u}{a_1 - I_w} \\ x_4 \\ \frac{a_2 \sin(x_1) - a_1 u / I_w}{I_w - a_1} \end{bmatrix}$$

$$I_b' = I_b + m_b l^2$$

$$a_1 = m_w l^2 + I_b'$$

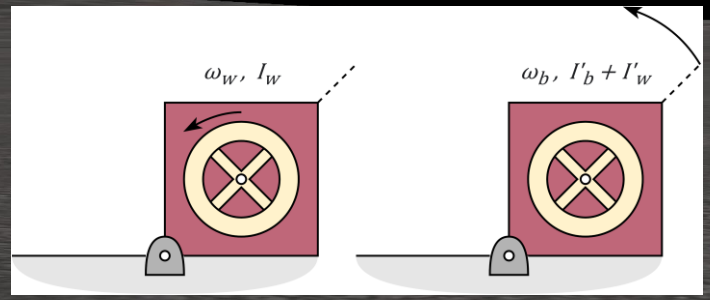
$$a_2 = (m_b l + m_f l) g$$

- Wind-Up
  - Body is fixed



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

- Brake
  - Angular Momentum Transfer



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

Method: Discretization

- Single-Axis FIP

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

$$x_{k+1} = \begin{bmatrix} x_{1_k} + x_{2_k} dt \\ x_{2_k} + \frac{a_2 \sin(x_{1_k}) - u_k}{a_1 - I_w} dt \\ x_{3_k} + x_{4_k} dt \\ x_{4_k} + \frac{a_2 \sin(x_{1_k}) - a_1 u_k / I_w}{I_w - a_1} \end{bmatrix}$$

- Wind-Up

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

$$x_{k+1} = \begin{bmatrix} x_{1_k} \\ 0 \\ x_{3_k} + x_{4_k} dt \\ x_{4_k} + \frac{u_k}{I_w} dt \end{bmatrix}$$

- Brake

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

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

Method: Cost Functions

- Minimum Time
  - Cost function is constant

$$cost_{time} = 1$$

  - Iteratively decrease time interval to find last valid solution

- Minimum Energy
  - Cost function is sum of norm at each time step

$$cost_{energy} = \sum \|u\|_1$$

  - Optimizer minimizes energy directly

Method: Constraints

- In Loop – Performed at each time step
  - Dynamic model constrains next time step
    - Activated by binary variable
  - Sum(binary variables) = 1
    - One model active at a time
  - Physical limits
    - Activated by binary variable
  - Windup is directional
    - 0 <= u <= maxT
  - FIP is not directional
    - -maxT <= u <= maxT
  - Swing-up in order
    - Wind-up, brake, balance
    - Dependent on if jump has occurred
- After Loop – Performed once
  - Initial State
  - Final State
  - Swing-up
    - Brake activated once
  - Slow-fall
    - Impact velocity threshold
    - Iteratively reduced like minimum time
  - Slow-step
    - Uses swing-up and slow-fall constraints

Method: Model Validation

- Large time steps may be used to reduce the number of variables
- Low-frequency control profile is passed through a high-frequency ODE simulator
  - Step function
- Simulation solution compared to optimization solution and old algorithms

- Discrepancies caused by optimizer’s zero-order-hold model
  - Reduce with step size
- If no valid optimization, then the model validation is skipped
  - Increase time steps
  - Ease constraints

Method: Code Acceleration

- Parallel Computing
  - Accelerates batches of optimizations for iterative process
  - Does not accelerate individual optimizations

- Wind-Up Torque Multiplier
  - Accelerates individual optimizations
  - Eases torque constraint on wind-up model
  - Reduces time, number of variables

Results: Code Acceleration

- Parallel Computing
  - Parallel Loop v For Loop
  - 4 threads → 3 times faster
  - Scales easily

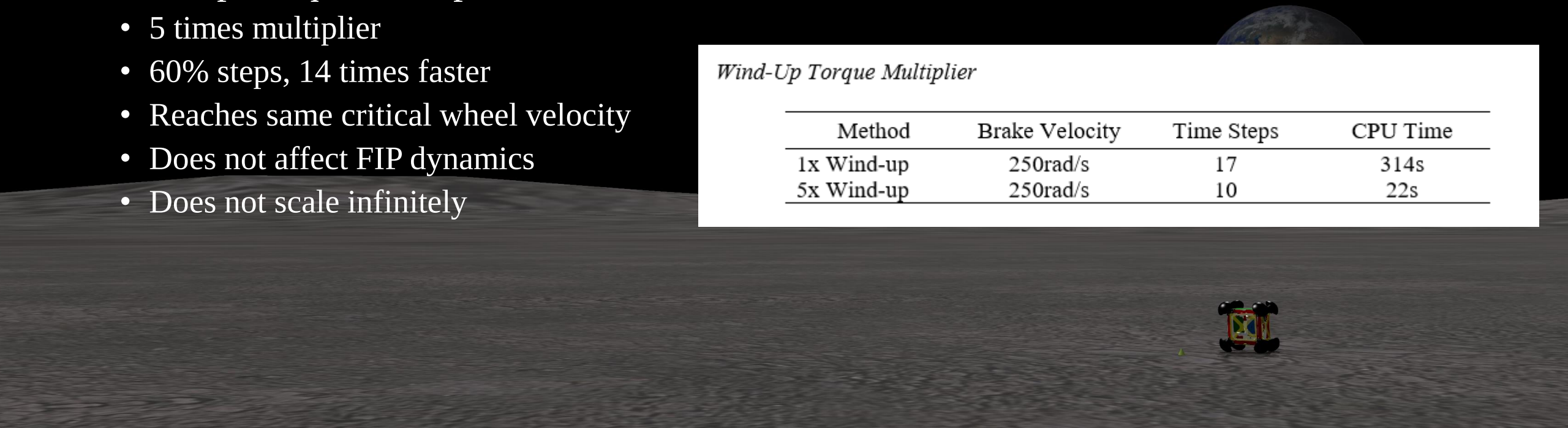
- Wind-Up Torque Multiplier
  - 5 times multiplier
  - 60% steps, 14 times faster
  - Reaches same critical wheel velocity
  - Does not affect FIP dynamics
  - Does not scale infinitely

Parallel Computing

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

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



Results: Swing-Up Optimization

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

- Max Torque indicated on plot
- Swing-up velocity from old algorithm indicated on plot

Results: Swing-Up

10Hz, minimum time, minimum energy

- Optimization reaches equilibrium
- Drift causes steady state error

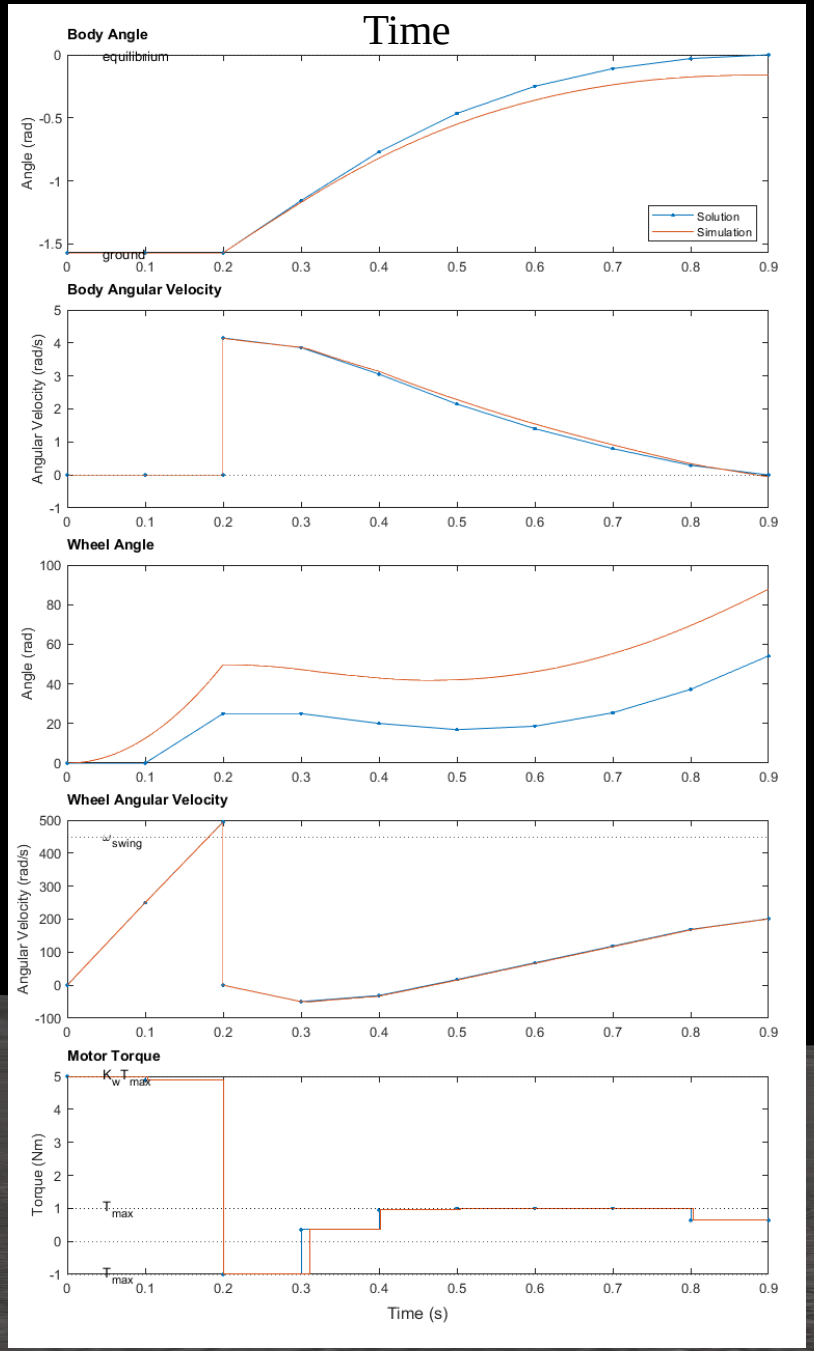
Time

- Applies maximum torque during windup
- Reaches swing-up velocity
- Applies additional torque to swing up faster

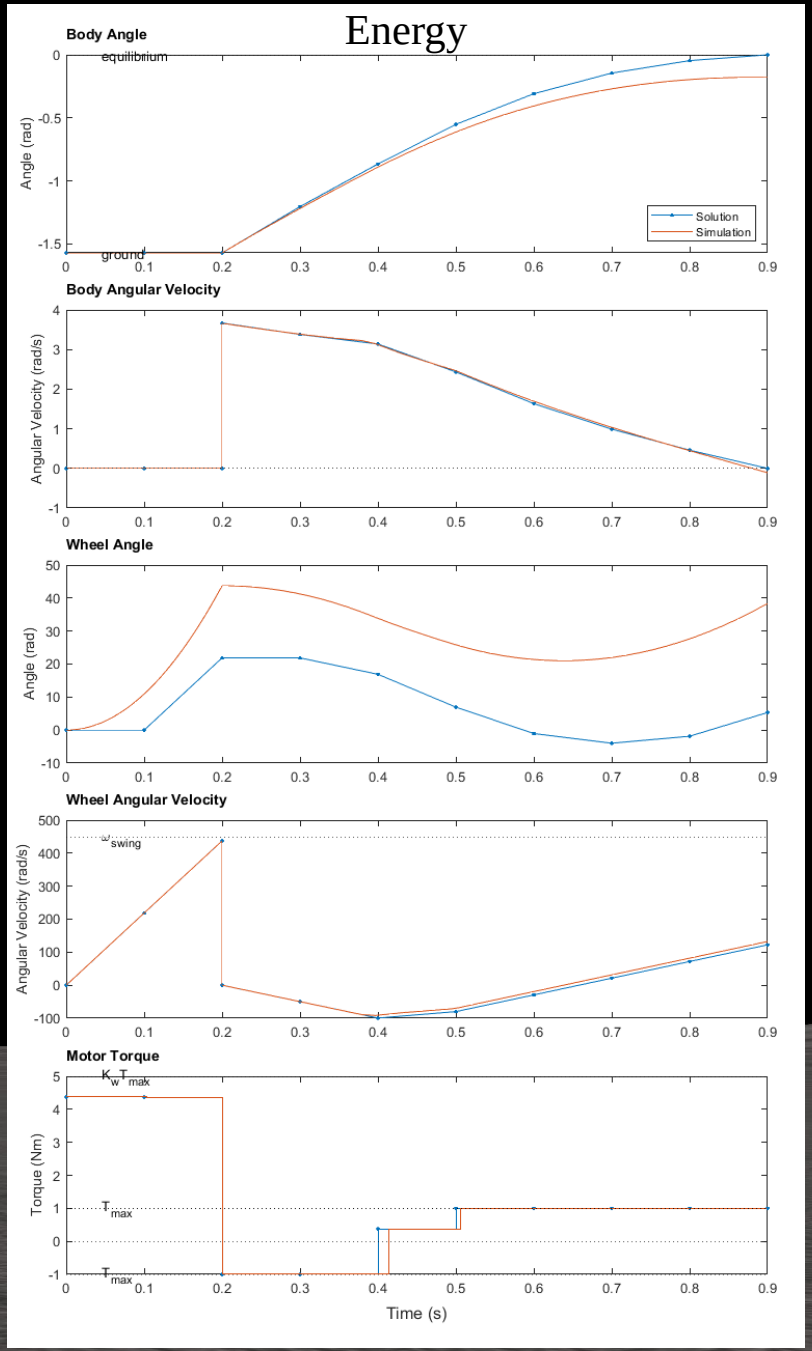
Energy

- Same behavior
- Less than maximum torque

Time

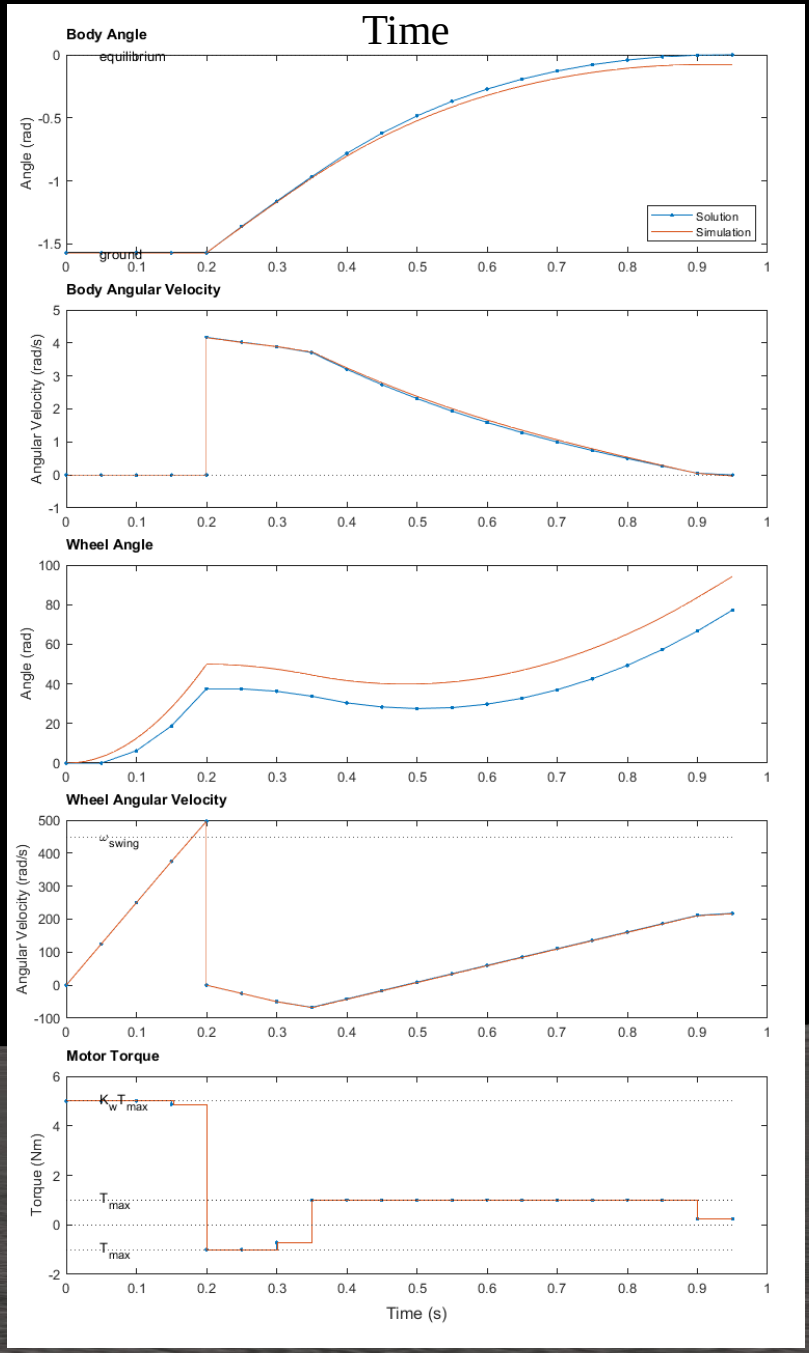


Energy



Results: Swing-Up Optimization

- 20Hz, minimum time
- Same behavior
- Drift is better at higher frequency, same period



Results: Slow-Fall Optimization

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	0.1s	1s
Figure 16	Energy	20	0.1s	9m 4s

- Max Torque indicated on plot
- Impact velocity threshold indicated on plot
- Freefall impact velocity indicated on plot

Results: Slow-Fall

20Hz, minimum time, minimum energy

- Optimization reaches ground
- Within impact velocity threshold
- Simulation reaches ground sooner
- Impact velocity still near threshold

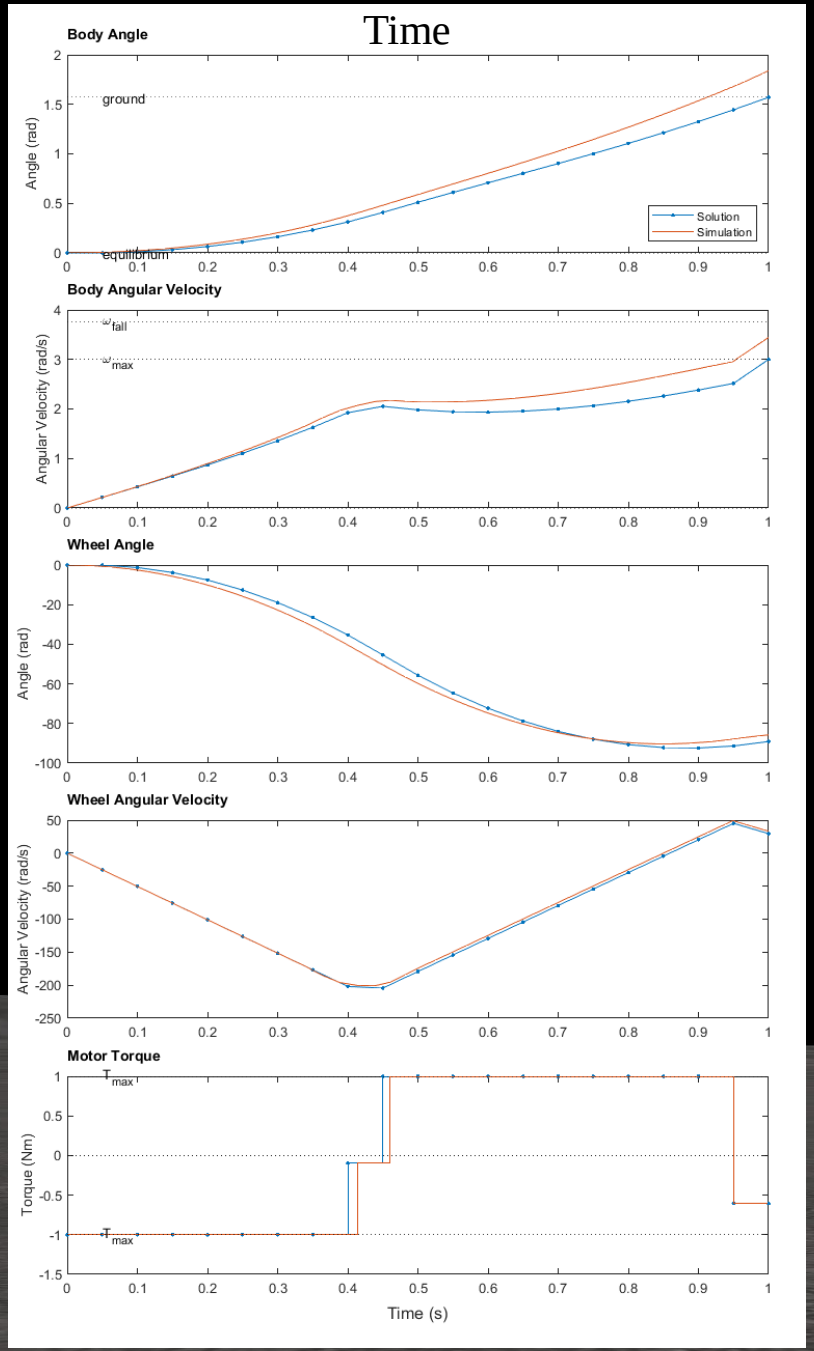
Time

- Torque switches between maximums

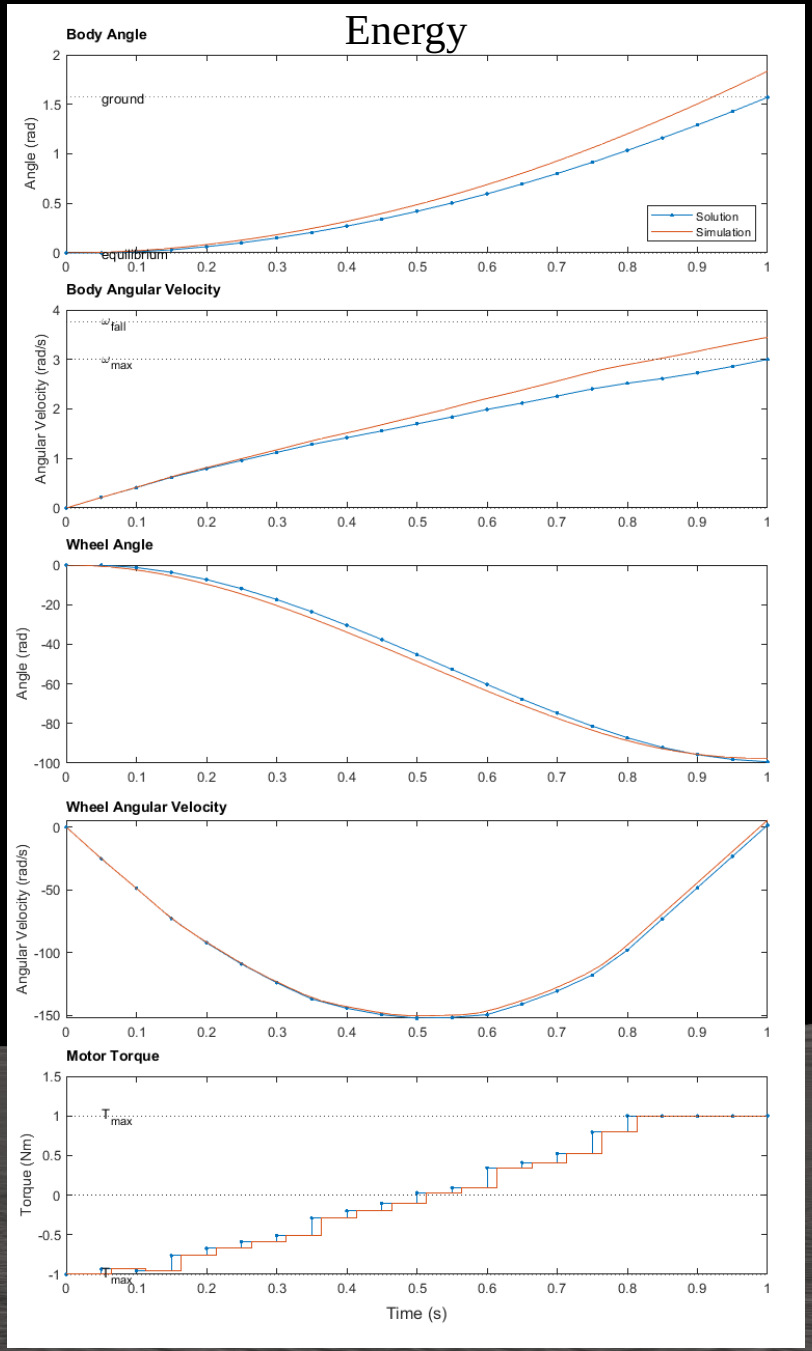
Energy

- Torque steadily increases

Time



Energy

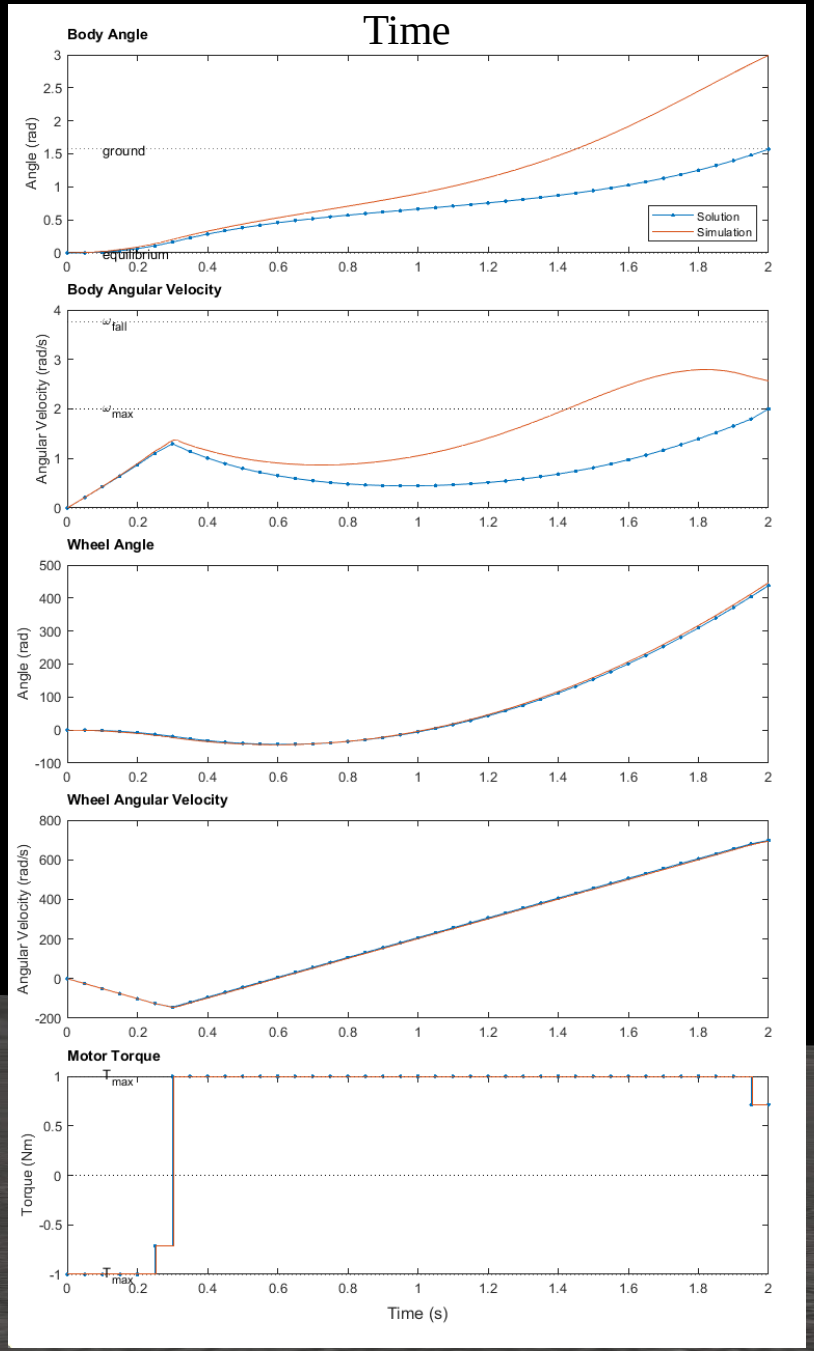


Results: Slow-Fall

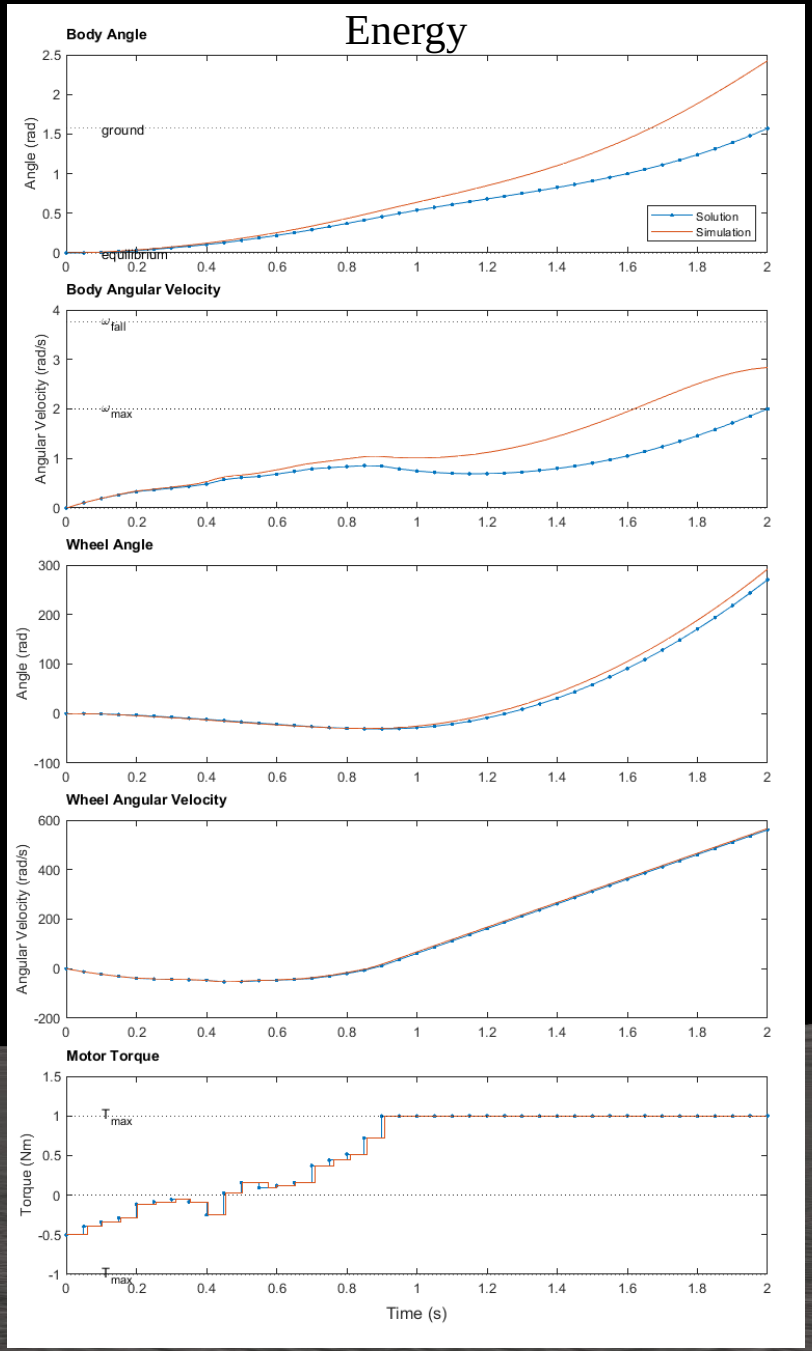
20Hz, double duration, minimum energy

- Similar behavior
- Drift is worse over longer period, same frequency

Time



Energy



Results: Slow-Step Optimization

Slow-Step Optimizations

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

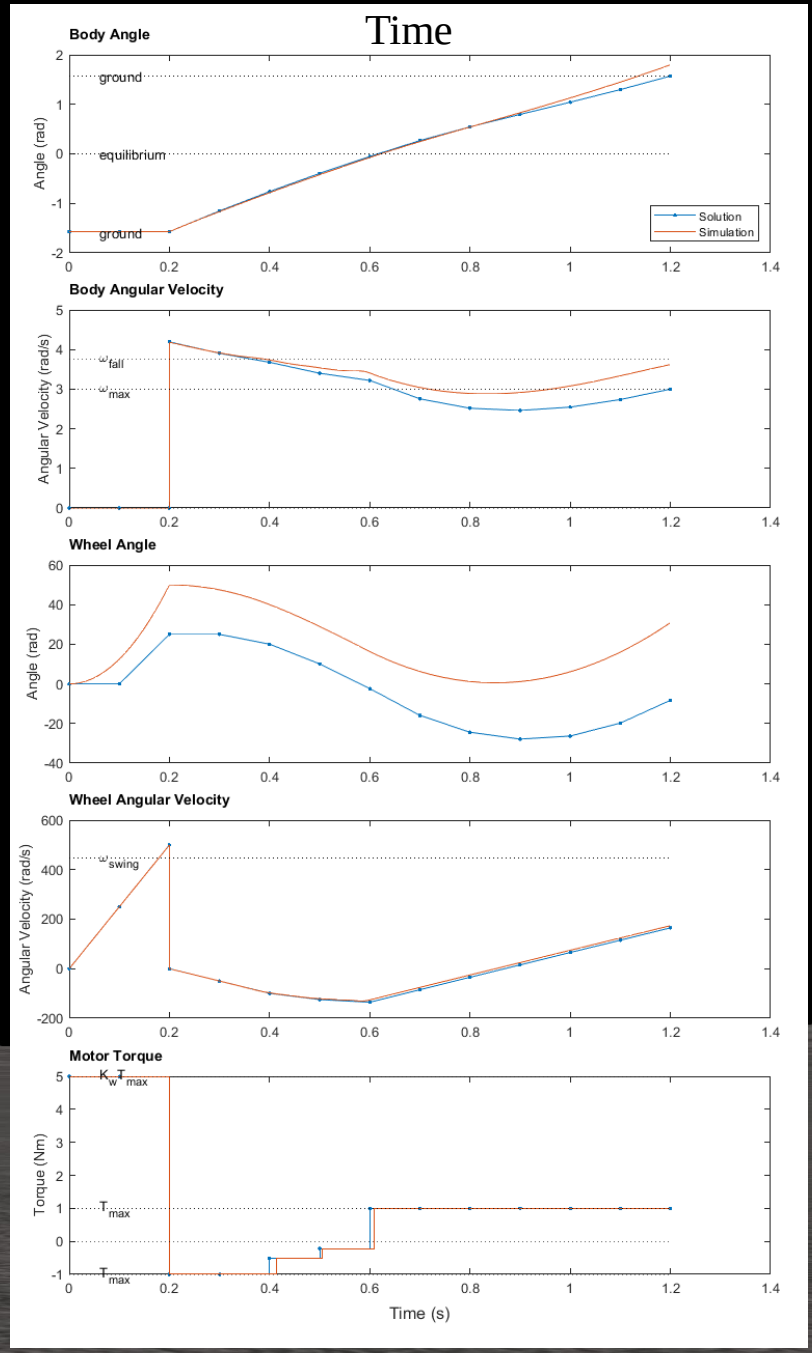
- Max Torque indicated on plot
- Swing-up velocity from old algorithm indicated on plot
- Impact velocity threshold indicated on plot
- Freefall impact velocity indicated on plot

Results: Slow-Step Optimization

10Hz, minimum time

- Optimization reaches equilibrium
- Body angle drift is low
- Body angular velocity drift is high
- Simulation no better than freefall

- Applies maximum torque during windup
- Reaches swing-up velocity
- Applies additional torque to swing up faster
- Reverses at maximum torque to slow fall



Conclusions: Summary

- Hybrid control framework handles multiple control modes and instantaneous jump maps
- Three single-axis maneuvers defined within framework
- The maneuver dynamics modelled and
- Models discretized for optimizer
- Performance metrics defined as cost functions and constraints
- Constraints handle dynamics and physical limits
- Code accelerated to reduce process time
- Models validated with a high-res simulation.

- Specifies and improves performance metrics
- Reveals unexpected control behavior

Conclusions: Future Work

- Reduce Error
  - Model predictive control
  - Feedback, LQR
- Model predictive control not currently feasible
  - Even one-shot optimization too computationally expensive
  - MPC recalculates the optimization at each time step
- Accelerate code further
  - Restructure problem to reduce variables
  - Parallelize on larger cluster
- Solve optimizations offline for sets of initial conditions
  - Save profiles onboard, select and apply profile depending on initial conditions
- Use optimizations to educate tuning of simpler control methods
  - Optimization reveals unique behavior
  - Tune simple controller to produce similar response
- Extend to 3D JOOEE

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

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

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

Appendix: Code Structure

- 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.

Code Structure

File	Type	Description
MAIN.m	Main 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