NASA Fall 2020 Internship, Final Report: Modeling and Control for a Flexible Inverted Pendulum Robot

Nashir A. Janmohamed[1]
Santa Monica College, Santa Monica, CA 90404, USA

Michael A. DuPuis[2]
NASA, Kennedy Space Center, M/S: NE-L6, FL 32899, USA

I. Abstract

This report describes the tasks accomplished during Fall 2020 at the Kennedy Space Center under the scope of the Robotic Control System Design internship project. These tasks primarily supported development of an augmented adaptive control system for an inverted pendulum (IP) robotic system on a 4-wheel mobile base *(Penny)*. This system serves as an analogue to the control problems in the flight of rockets in the initial and latter stages of launch, and methods developed as part of this research can later be applied to more complex IP systems, such as launch vehicles. To more accurately model launch vehicles, a flexible aluminum pendulum is used both on the hardware and in the simulated models. In order to capture the flexible dynamics of the system, hardware modifications were made to *Penny* (including installation of a rate gyro at the tip of the pendulum). Simulink models were created to control and model the hardware system, and Simscape models were created/updated to model the system in simulation. MATLAB programs were created throughout the internship to analyze data generated from hardware and simulation runs. Linear fixed-gain controllers have been applied to the simulated and hardware system, and work continues with augmenting these controllers using sensor blending and direct output adaptive control methods to improve stabilization of system states and cancel flex dynamics. In addition to describing the work done to support these modeling efforts, an Independent Research and Technology Development (IR&TD) proposal for a lunar simulation with soil deformation modeling developed during the internship is briefly described.

Nomenclature

IP	=	Inverted Pendulum
RIP	=	Rigid Inverted Pendulum robot
FIP	=	Flexible Inverted Pendulum robot
x	=	cart position
$\dot{x}$	=	cart velocity
θ	=	rigid pendulum angle w.r.t. vertical
$\dot{\theta}$	=	rigid pendulum angular velocity w.r.t. vertical
θ_b	=	flexible pendulum angle w.r.t. vertical (at base of pendulum)
$\dot{\theta_b}$	=	flexible pendulum angular velocity w.r.t. vertical (at base of pendulum)
θ_t	=	flexible pendulum angle w.r.t. vertical (at tip of pendulum)

[1] NASA KSC, OSTEM Intern, NE-L6.
[2] NASA KSC, PI, Robotics and Autonomous Systems, NE-L6.

$\dot{\theta}_t$	=	flexible pendulum angular velocity w.r.t. vertical (at tip of pendulum)
DOAC	=	Direct Output Adaptive Control
IR&TD	=	Independent Research and Technology Development
SEELO	=	Simulated Excavation Environment for Lunar Operations
RTMB	=	Research and Technology Management Board

II. Introduction

The rigid inverted pendulum (RIP) system is a canonical example of an unstable mechanical system that is used as a pedagogical tool in physics, dynamics and control that has been studied since at least the 1950s[1]. It is also used to model the control problems in the flight of rockets in the initial and latter stages of launch, when aerodynamic forces are too small for aerodynamic stability[2]. Another prevalent example of a stabilized RIP is a human being. When standing, the feet act as the pivot point; without constant small muscular adjustments, the person will fall over[1]. Many control architectures have been used to solve this control problem, ranging from simple Bang-Bang control to more complex nonlinear control architectures such as Sliding Mode, Model Predictive, and Fuzzy Logic controllers[1]. Various reinforcement learning (RL) approaches, including those that use neural network architectures, have also been applied to this problem[3]. Less study has been given to flexible IP systems (FIP), a variant of the RIP system with a much higher degree of nonlinearity that more accurately models the dynamics of rockets[4], though there are examples in the literature of controllers developed for this problem[5].

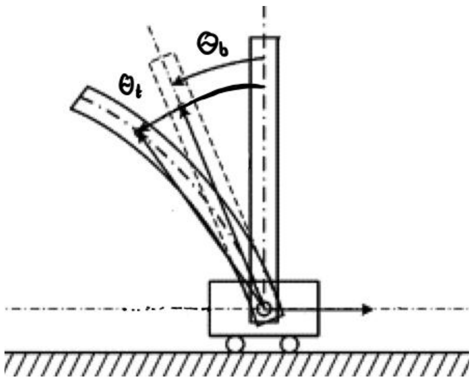

Figure 1. Depiction of a FIP system. *Pendulum flex causes different angle to be measured at the pivot point ($\boldsymbol{\theta_b}$) and at the free rotating end of the pendulum ($\boldsymbol{\theta_t}$).*

This research builds upon the work done by a previous intern to develop a MATLAB-based Simscape model to control a FIP system on a wheeled mobile base (*Penny*) and theoretical modeling by Dupuis and Okasha[6]. These models describe the dynamics of the system, but it is desired to enhance model fidelity and efficacy through other approaches; the existing theoretical and empirical models of the system are not yet accurate enough to transfer simulated controllers onto the hardware and have them provide sufficient stabilization.

Before implementing new control laws and architectures, with a focus on direct output adaptive control (DOAC) methods, it is necessary to effect hardware upgrades to *Penny*. Currently, measurements are taken of the cart position (x) and pendulum angular velocity with respect to the vertical ($\dot{\theta}$). Given the time history of all prior measurements, cart velocity ($\dot{x}$) and pendulum angle (θ) can be computed. These four states are sufficient to predict the future behavior of a RIP system, but they do not capture the overall state of a FIP system and thus cannot predict future behavior. Since the angle of the pendulum is measured only at the base, if there is flex in the pendulum, a state measurement (comprised of $[x, \dot{x}, \theta, \dot{\theta}]$) can correspond to different pendulum states (i.e. many with flex and one without). Even if two system states may have the same four state measurements, they will evolve differently through time, which motivates measuring additional information to estimate pendulum flex. The flex can be implicitly estimated by measuring the angular velocity of the pendulum tip with respect to the vertical and comparing this to the angle at the base of the pendulum. To this end, a rate gyro assembly was installed on the pendulum tip. The Simulink models were updated concurrently to incorporate a pseudo sensor that reports angle and angular velocity of the pendulum tip with respect to the vertical (yielding a state measurement with six states $[x, \dot{x}, \theta_b, \dot{\theta_b}, \theta_t, \dot{\theta_t}]$).

For the purposes of exclusively investigating cancellation of flex modes in the inverted pendulum system, a modification to the system was made that employed an extremely flexible aluminum pendulum with a fixed base (i.e. $\theta_b = \dot{\theta_b} = 0$). With a tip gyro on this flexible pendulum, all measured θ_t states correspond directly to flex. This variant on the inverted pendulum system was used to research control methods that act directly on the flex.

To capture the flexible dynamics of the system, hardware modifications were made to *Penny* (including installation of a rate gyro at the tip of the pendulum). Simulink models were created to control and model the hardware system, and Simscape models were created/updated to model the system in simulation. MATLAB programs were created throughout the internship to analyze data generated from hardware and simulation runs. Linear fixed-gain controllers have been applied to the simulated and hardware system, and work continues with augmenting these controllers using sensor blending and direct output adaptive control methods to improve stabilization of system states and cancel flex dynamics. In addition to describing the work done to support these modeling efforts, an Independent Research and Technology Development (IR&TD) proposal for a lunar simulation with soil deformation modeling developed during the internship is briefly described.

III. Hardware Modifications

Before implementing the sensor upgrades and installing a flat fixed base pendulum, a wiring spreadsheet of *Penny* (which consists of two Arduino Mega 2560's, electronic speed controls, a serial conversion circuit, motor encoders, a MicroStrain IMU, 5V regulators, servos, and XBees for wireless communication between the master Arduino and the tip rate gyro assembly) was made to facilitate ease of modification to the electrical system at a later date. Voltage monitors were also installed on the chassis and on the tip gyro assembly. Since the battery voltages were unknown without removing them and checking their voltage with a multimeter, occasional performance anomalies due to exiting the nominal voltage range (as shown in Figure 2) were harder to diagnose while the robot was being operated.

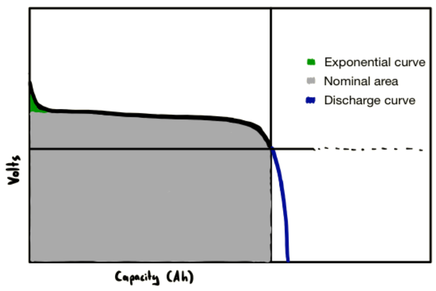

Figure 2. Ni-MH Discharge Curve[7]. *Battery voltage drops precipitously after leaving range of nominal capacity.*

The new flat pendulum used to study flex modes exclusively required a different mount from the one previously on *Penny* for two reasons: 1) the different pendulum geometry (original rectangular: 0.25in.^{2} cross section; new flat: 0.0945in×1.995in) and 2) the desire to fix one end of the pendulum. After a design was created by fellow intern Alex Lacerna, the part was printed, fastening holes were drilled, and it was affixed to the chassis assembly (as shown in Figure 3). The flat pendulum also had holes drilled on both ends to allow fixing to the base mount and to the rate gyro assembly to the tip of the pendulum.

Summer 2020 intern, Juan Halleran, began work on creating a tip rate gyro assembly (Figure 4) built around a Pro Micro 3.3V microcontroller, an XBee radio communication module, and an MPU6050. The assembly was completed in the early stages of the internship. A wiring diagram of the tip rate gyro assembly is shown in Figure 5. The various boards were initially soldered directly together, but the wiring was redone to improve cable management by installing Molex connectors. In addition to improving the mechanical stability of the assembly, these connectors make the system more modular, which will reduce time and effort spent fixing the assembly if any constituent element fails.

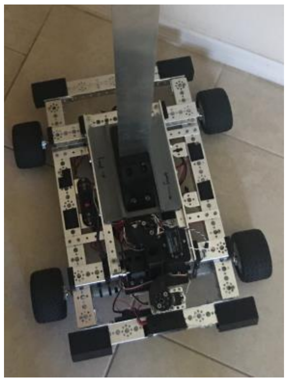

Figure 3. *Penny* chassis. *The flat fixed pendulum is seen installed; both the original (black) pendulum holder and the new (grey) one are visible.*

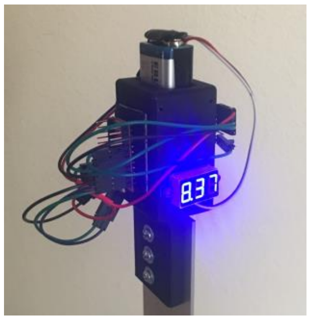

Figure 4. *Penny* tip rate gyro assembly. *The voltage monitor, 9V battery, and Pro Micro are shown; the XBee and MPU6050 are on the other side of the assembly.*

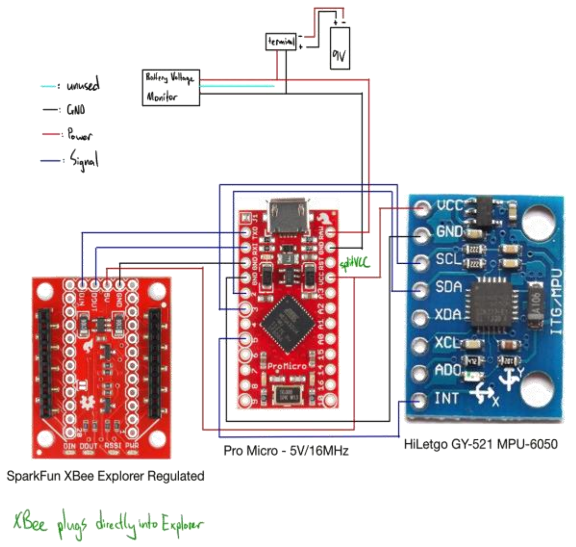

Figure 5. *Penny* wiring diagram.

IV. Robot Code

A. Arduino/Simulink wireless communication

Penny has master control code loaded onto one of the Arduino Mega 2560s, and this Arduino receives serial inputs from another Arduino Mega 2560, and software serial (over radio) inputs from the Pro Micro (which reports angular velocity of the tip of the pendulum, i.e. θ_t). Juan Halleran previously used XCTU to set up wireless communication between the Pro Micro and the master Arduino, but there was a limitation in the data that could be sent, with the largest discrete unit of information that could be sent being a single byte. After reading the gyro data from the MPU6050 using code from the i2cdevlib/Arduino/MPU6050 library[8], the measured floating-point value on the Arduino side was converted to an array of bytes as specified by the IEEE Standard for Floating-Point Arithmetic (IEEE 754)[9] each of which was sent in succession. With a continuous sequence of bytes and no demarcation between packets, it is not possible to reconstruct the data without ambiguity. To eliminate this ambiguity on the receiver side, a header byte (0xFF) was prepended to the sequence, and a checksum byte (the XOR of the four bytes that comprised the original floating-point measurement) was appended to the sequence. On the Simulink side, the bytes were received and after aligning with the header and verifying the checksum, the floating-point value was reconstructed. See Appendix A for the code on both the Arduino and Simulink side, as well as the Simulink model to reconstruct the measured float data.

The precision of the Arduino measurement can be specified in the setup code on the Arduino side; the four options for dps (the precision/range of the measurement) are $\pm$250°/s, $\pm$500°/s, $\pm$1000°/s, and $\pm$2000°/s. Regardless of the value selected for dps, the measurements reported by the MPU6050 code were mapped to the range of [-1431,1430]. Since it was clear that increasing the limit of measurable values decreased the precision of the measurement, the smallest range that fully measured the spectrum of pendulum angular velocities (in non-extreme operations) would be the most optimal. Through empirical testing, the smallest range that measured the full spectrum of pendulum angular velocity was determined to be 1000°/s. On the Simulink side, the reported value was converted to a measurement in degrees using the relationship $res = inp \cdot \frac{2dps}{nbins}$, where inp is the value reported by the sensor, dps is as defined above, and $nbins = 2862$ is the fixed range of values reported by the sensor.

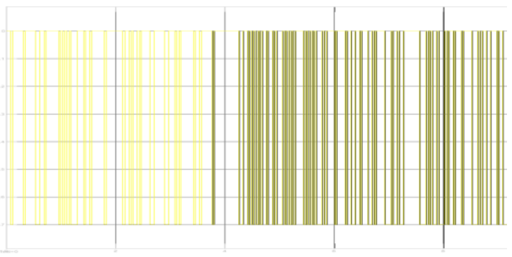
Figure 6. Gyro bias. *Measurement alternates randomly between* -0.7 *and* 0.

One last issue with the reported angular velocity that was addressed was the sensor bias in the gyro measurement (i.e. it alternates between 0 and a fixed negative value, which depends on the sensitivity used for dps, randomly as shown in Figure 6). Only velocity is measured, so angle is obtained by integrating the angular velocity measurement with respect to time. When the sensor is at rest, this bias is most noticeable, and shows up in the angle measurement as a relatively linear walk away from zero. A naïve method of cancelling this drift was employed to reduce the error in the position measurement: the slope of this drift ($\frac{d\theta}{dt}$) was measured and subtracted from the rate measurement. This reduced the error to more of a random walk than a linear drift away from zero (see Figure 7). To verify that the angle reported was not drifting from zero, the sensor was flipped between 90°, and the performance was much better (see Figure 8).

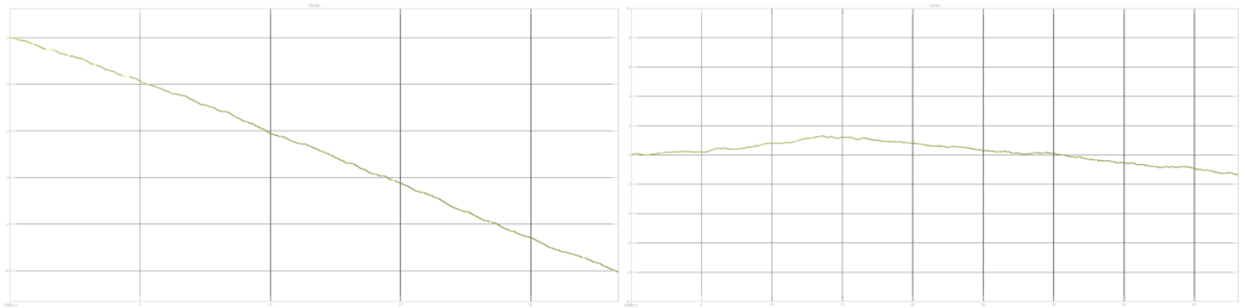
Figure 7. Angle at rest a) before and b) after correcting for rate bias. *With the sensor at rest, the error in the measurement goes from linear w.r.t. time, to constant small error. In a), after 25 seconds, the* $\boldsymbol{\theta}$ *measurement is* -10°*, while in b), after 40s, the error is only* $\pm\mathbf{1}$°.

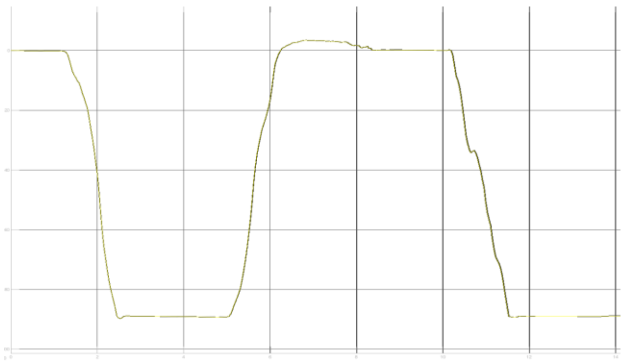

Figure 8. Angle obtained from integrating angular velocity, with sensor positioned at 0° and 90°. *After correcting for drift, the reported position is highly accurate.*

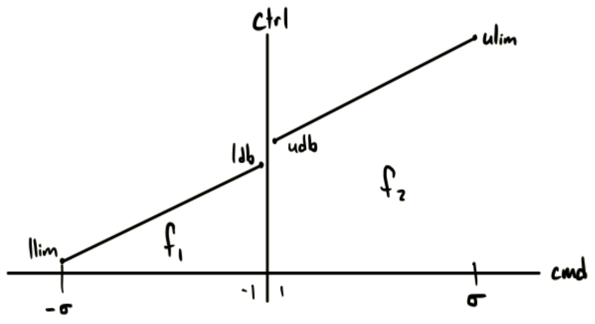

Figure 9. Piecewise functions that make up motor signal builder.

B. Motor signal builder

Penny's motors are controlled on the Simulink side by using the "Standard Servo Write" block. According to the Arduino documentation[10], this block writes a voltage value to the continuous rotation servo, which in effect specifies a speed: "0 being full-speed in one direction, 180 being full speed in the other, and a value near 90 being no movement". However, deadband (a band of input values where the output is zero) exists in the range around 90 (~[86,95]). Also, past certain values on the lower and upper end of the range (e.g. ctrl < $llim$ and ctrl > $ulim$), the signal drops out. For these reasons, in addition to the desire to use a command signal of the form $cmd \in [-\sigma, \sigma]$, a motor signal builder was created.

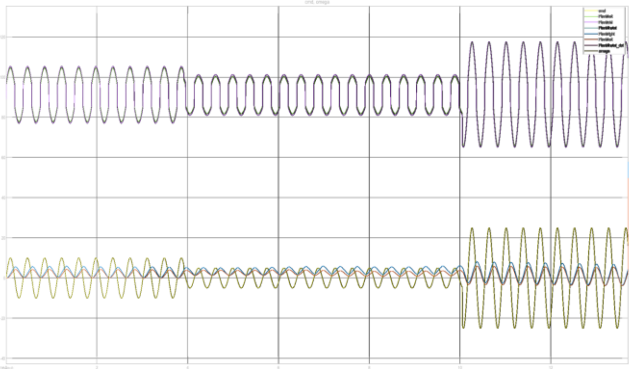

Figure 10. Tuned motor signal builder. *Yellow: cmd, purple & green: ctrl, red & blue: motor encoder readings.*

The motor signal builder takes cmd values in the range of $[-\sigma, \sigma]$ (for *Penny*, $\sigma = 79$) and outputs a $ctrl$ value in the range of $[llim, ulim]$ (for *Penny*, $llim = 6$ and $ulim = 179$). If $cmd \geq \sigma$, then $ctrl = ulim$; if $cmd \leq -\sigma$, then $ctrl = llim$. If $cmd = 0$, then $ctrl = 90$ (i.e. in the deadband which produces no movement). Otherwise, the $ctrl$ signal is constructed using two piecewise functions, one for each side of the deadband. If $-\sigma \leq cmd < 0$, then $ctrl = f_1(cmd)$, else if $0 < cmd \leq \sigma$, then $ctrl = f_2(cmd)$. Two terms $fbias$ and $bbias$ are used to change the slope of the linear functions f_1 and f_2 that map $cmd \rightarrow ctrl$ as shown below.

$$f_1(cmd) = (ldb + cmd) * (1 - bbias)$$
$$f_2(cmd) = (udb + cmd) * (1 + fbias)$$

After creating the $ctrl$ signal from the input cmd, it was clear that there were still inconsistencies between the right and left sides of the robot (e.g. one side had a different deadband). Since the left and right sides of the robot are not mechanically linked, each might need slightly different parameters, and so two motor signal builder functions are used, one for each side of the robot.

C. Deadband identification

Since the deadband on *Penny* will occasionally shift, it was desired to automate the process of identifying the deadband. This was accomplished by creating a Simulink model to, given a known starting deadband value db (such as 90), send a $ctrl$ value of $db \pm 1$ (+ if identifying upper deadband, − if identifying lower deadband) and increment/decrement the value of db until the motor encoders start reporting motion. The first value of db to produce motion on each side of the robot is then reported to the user using a "Display" block. One consideration made was that small differences in encoder values would falsely trigger the condition indicating that the deadband had been identified; a min_delta value was then created, and if $abs(encoder\_value - prev\_encoder\_value) > min\_delta$, then the simulation would report the identified deadband.

D. Master control model w/ data collection and script for saving data to workspace

After developing the models to construct the motor signal and read the data from the tip rate gyro assembly, a master control model was created that combined the two. To efficiently save the data generated during hardware tests, “To Workspace” blocks were added to the model to save the generated data $[enc_l, enc_r, \theta_t, \dot{\theta}_t, cmd, t]$, and a script was created to save the generated data from the workspace to a .mat file (e.g. u_sin_w_3_a_5_10_2_2020.mat, with parameters u -> input function specifying cmd, w -> frequency, a -> amplitude). An example plot of data generated during a hardware test is shown in Figure 11.

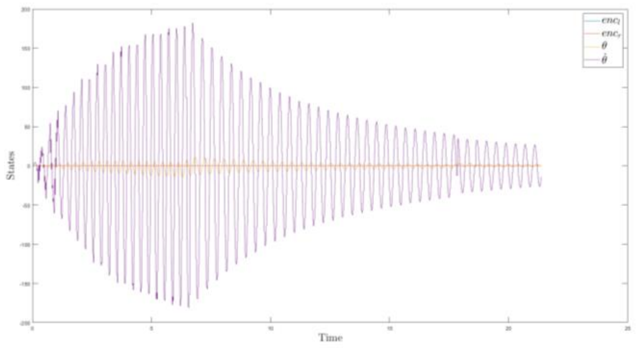

Figure 11. Raw states recorded and plotted from within MATLAB. *States generated from* $\boldsymbol{u(t) = 5\,sin(3 * 2\pi t)}$.

V. Modeling & analysis

A. Simscape models

A Simscape model of the original *Penny* system with a swiveling pendulum was previously created (Summer 2019) by intern Juan Halleran. This Simscape model was itself a modification of an inverted pendulum model created by Mathworks, which can be loaded from within MATLAB by typing the command $load('rct\_pendulum')$;. Halleran’s modifications included replacing the pendulum block with a general flexible beam and changing the expected input to the plant from force to voltage. Instead of applying force directly to the cart, the input voltage is applied to a “DC Motor” block which produces rotation of a friction parameterized tire; the force generated from the tire interaction with the ground is then applied to the base of the cart. Using this model as a baseline, a) a new Simscape model was created that is more representative of the swiveling *Penny* system, and b) a new Simscape model was created to represent the fixed base flat pendulum system.

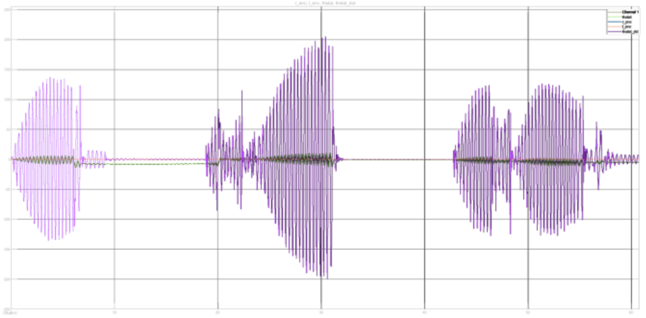

Figure 12. Test to determine natural frequency on hardware. *Left peak:* $\boldsymbol{\omega = 2.95}$ Hz, *Middle peak:* $\boldsymbol{\omega = 3.00}$ Hz, *Right peak:* $\boldsymbol{\omega = 3.05}$ Hz. *The value* $\boldsymbol{\omega_n}$ *was determined through this process to be* $\boldsymbol{3.00}$ Hz. *A similar process was used to tune the natural frequency in*

To represent the fixed pendulum base system, the pendulum geometry was updated to reflect the new length and cross section, limits were set on the revolute joint such that the pendulum was fixed at the base, and a pseudo tip sensor (using the “Transform Sensor” block) and small mass were added to the pendulum tip to reflect the addition of the tip rate gyro assembly to the hardware system. The mass of the tip weight was tuned such that the natural frequency of the pendulum (the frequency at which a system tends to oscillate in the absence of any driving or damping force) matched that of the physical pendulum on *Penny*. The natural frequency for both the hardware and simulated systems were identified by sending a sinusoidal input at some frequency ω and noting the value ω_n (i.e. the natural frequency) that created the largest resonance.

In both systems, a motor signal builder was made to reflect the nonlinearity of the hardware and to allow input *cmd* signals in the same form as those that would be used on the real *Penny* system. This motor signal builder takes input *cmd* signals and outputs *ctrl* signals in the range of $[-\sigma, \sigma]$ (where σ is the value used in the hardware, in this case, 79). After converting the cmd to a $ctrl$ signal, the $ctrl$ signal is converted to a voltage using a relationship between cmd and $voltage$ on the hardware ($voltage = 0.1431ctrl$) that was empirically determined by Dupuis. This motor signal builder allows for developing controllers in simulation that will be more easily transferrable to the hardware system.

B. Identifying transfer functions from data

In addition to developing a representative Simscape model that reflects the nonlinearities present in the hardware system, it was desired to develop linear models of the *Penny* system that could then be used in developing linear quadratic regulator (LQR) control methods, as well as DOAC augmentations to these linear controllers. To this end, a system for taking recorded input-output relationship data (e.g. cmd, x, θ, t) from both the hardware system and Simscape models and producing transfer functions (a mathematical function which theoretically models the device's output for each possible input) was developed. Using the $tfest$ method provided by MATLAB, transfer functions were produced that described the input cmd relationship to each of the four output states for the fixed flexible IP system ($[x, \dot{x}, \theta, \dot{\theta}]$). These transfer functions were then used to construct a state space model of the system. At the time of this publication, work is just beginning with control implementation. However, these linear state space models describe the system well, and LQR controllers developed from these models perform relatively well tracking reference signals (i.e. demanded $\theta = 1°$) on the Simscape system.

VI. Miscellaneous Work

A. IR&TD development: Simulated Excavation Environment for Lunar Operations (SEELO)

As an enrichment activity, a proposal for a research project was created as a submission to the Independent Research & Technology Development (IR&TD) proposal call. IR&TD is an internal Kennedy Space Center (KSC) program that funds research efforts proposed by NASA PIs (principal investigators). The proposed project entails developing an interactive simulation environment that efficiently models lunar In-Situ Resource Utilization (ISRU) excavation operations, including soil deformation, while running in faster-than-real-time. There is a need for a simulation environment with accurate regolith interaction mechanics that is less computationally expensive than commercially available multiphysics tools. This simulation will allow the development of robust machine learned models and policies to improve and enable certain ISRU operations. It would also enable various projects and capabilities such as validation of excavation mechanisms and testing/validation of mission architectures or robotic systems for ISRU. For ISRU missions currently in the planning stages, leveraging the data generated by this simulation in a wide variety of mission architectures will significantly improve the fidelity of planning efforts. The skills required to develop this simulation, both for ISRU concept testing and machine learning to enable autonomy, are valuable and transferable to other dynamical systems and environments at KSC.

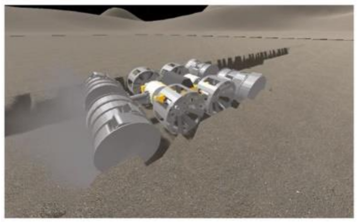
Figure 13. Lunar sim concept developed for presentation to the RTMB.

SEELO was selected as one of eight projects for presentation to the NASA KSC Research and Technology Management Board (RTMB) and was approved by the board. It is in the final stages of approval for funding in FY21, pending authorization by Burt Summerfield, the Associate Director of Management at KSC.

VII. Conclusion

The hardware modifications and software developed that were described in this report provide a baseline for further work in application of controllers to both the hardware and simulated *Penny* system. Given the parallels between *Penny* and launch vehicles, control methods developed for this system can have significant impact on the state of the art in control methods for launch vehicles. Future work (which is ongoing at the time of this publication) includes implementation of sensor blending and DOAC methods, use of a zero dynamics estimator[11], and use of disturbance accommodating DOAC[12], which is the main focus of Michael Dupuis' research.

Appendix

A. Arduino/Simulink wireless communication

Arduino Code to construct array of bytes from float

```
typedef union
{
  float number;
  uint8_t bytes[4];
} FLOATUNION_t;
FLOATUNION_t myFloat;

void sendFloatAsBytes(float val) {
  myFloat.number = val;
  XBee.write(0xff);
  byte b = 0;
  delay(5);
  for (int i = 0; i < 4; i++)
  {
    XBee.write(myFloat.bytes[i]);
    b ^= myFloat.bytes[i];
    delay(5);
  }
  XBee.write(b);
  delay(5);
}
```

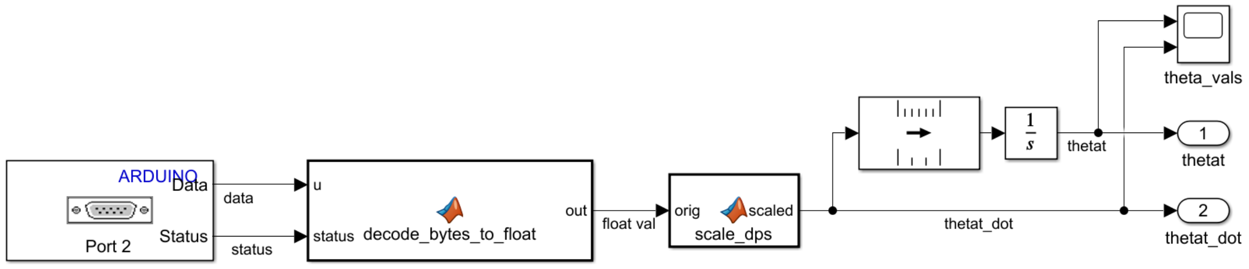

Figure 6. Simulink Model for reading θ_t states from tip sensor. *Array of bytes is sent over software serial to Port 2, deconstructed to float in* "decode_bytes_to_float" *block, rate bias is corrected in* "scale_dps" *block, and then angle is obtained from integrating the measured angular velocity,* $\dot{\theta}_t$.

MATLAB Code to decode array of bytes to float

```
function out = decodeBytesToFloat(u, status)
% u should be length 6, 1 startbit, 4 data, 1 checksum
% if check fails or bitxor fails, return previous value

persistent receiveBuffer;
persistent counter;
persistent prevOut;

if isempty(counter)
    counter = 1; %Initialize counter
end

if isempty(receiveBuffer)
    %Initialize receive buffer
    receiveBuffer = uint8([0 0 0 0 0 0]);
end

if isempty(prevOut)
    % Initialize temp variable to hold previous output
    prevOut = single(0);
end

if status == 0
    out = prevOut;
    return
end

%Store received byte to buffer
receiveBuffer(counter) = u;

if counter == 6 % Data received.
    b = uint8(0);
    for idx = 2:5
        b = bitxor(b, receiveBuffer(idx));
    end

    if b == receiveBuffer(6)
        A = uint8(receiveBuffer(2:5));
        out = typecast(A, 'single');
        prevOut = out;
    else
        out = prevOut;
    end

    counter = 1;
else %Still receiving data
    counter = counter+1;
    out = prevOut;
end

if receiveBuffer(1) ~= 255 %Header mismatch
    %Synchronize header incase of mismatch
    counter = 1;
end
```

Acknowledgments

Nashir thanks the NASA KSC education office, the Minority University Research and Education Project (MUREP), and Universities Space Research Association (USRA) for the opportunity to be a part of the exciting work being done at KSC. In addition, he thanks both the NE-L6 branch and the Granular Mechanics and Regolith Operations (GMRO) laboratory for hosting him as an intern, as well as Michael DuPuis for his mentorship.

References

[1]O. Boubaker, "The inverted pendulum: A fundamental benchmark in control theory and robotics," *International Conference on Education and e-Learning Innovations*, Sousse, 2012, pp. 1-6, doi: 10.1109/ICEELI.2012.6360606.

[2]Lundberg, Kent. (2009). History of Inverted-Pendulum Systems. IFAC Proceedings Volumes. 42. 131-135. 10.3182/20091021-3-JP-2009.00025.

[3]A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, pp. 834–846, Sept./Oct. 1983.

[4]Dupuis, M. NASA, KSC Dynamics and Controls SME.

[5]J. Tang and G. Ren, "Modeling and simulation of a flexible inverted pendulum system," in *Tsinghua Science and Technology*, vol. 14, no. S2, pp. 22-26, Dec. 2009, doi: 10.1016/S1007-0214(10)70025-0.

[6]Dupuis, M., Okasha, H., "Flexible Dynamics Modeling for *Penny* Robot," *NASA Scientific and Technical Information Program* (unpublished).

[7]Mebarki, Brahim & Belkacem, Draoui & Rahmani, Lakhdar & Allaoua, Boumediene. (2013). Electric Automobile Ni-MH Battery Investigation in Diverse Situations. Energy Procedia. 36. 130-141. 10.1016/j.egypro.2013.07.016.

[8]Rowberg, J., "i2cdevlib: I2C device library collection for AVR/Arduino or other C++-based MCUs" https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050

[9]"IEEE Standard for Floating-Point Arithmetic," in *IEEE Std 754-2019 (Revision of IEEE 754-2008)* , vol., no., pp.1-84, 22 July 2019, doi: 10.1109/IEEESTD.2019.8766229.

[10]Various, n.d. Arduino - Servowrite. [online] Arduino.cc. Available at: <https://www.arduino.cc/en/Reference/ServoWrite>.

[11]M. J. Balas and S. A. Frost, "Direct Adaptive Control of Non-Minimum Phase Linear Infinite-Dimensional Systems in Hilbert Space Using a Zero Dynamics Estimator," *2019 IEEE 58th Conference on Decision and Control (CDC)*, Nice, France, 2019, pp. 3072-3079, doi: 10.1109/CDC40024.2019.9029835.

[12]R. J. Fuentes and M. J. Balas, "Direct adaptive disturbance accommodation," *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)*, Sydney, NSW, 2000, pp. 4921-4925 vol.5, doi: 10.1109/CDC.2001.914711.