Nonlinear Model Predictive Controller Implementation on a Microprocessor for 2D Lunar Powered Descent

Saumya Sharma[1]
Georgia Institute of Technology, Atlanta, GA, 30332, USA

Designing the optimal lunar landing trajectory for a single thrust output, SmallSat form-factor vehicle involves analyzing the trajectory from a deorbit burn (braking) and powered descent phase. Due to the need to reduce altitude and velocity while optimizing for maximum landing mass, the system is modeled with non-linear dynamics based on various assumptions which are described in the paper. Open-loop continuous propagation is used with optimal control theory to establish a trajectory that will be used as the source of truth and compared against the hardware implementation of this simulation. Because the designed control algorithm needs to be capable of flying on an on-board computer, a model predictive controller (MPC) was implemented to show how discrete real-time updates impact the optimization of the trajectory. MPC reduces the computational load through an online optimization algorithm instead of using a true optimization problem to produce a more flyable control scheme. To show the effects of running an MPC for a "flight-like" algorithm on a processor that would fit in a SmallSat form-factor, a Raspberry Pi 3B was used to demonstrate how varying the time horizon length and time-step frequency impact computing performance and fuel consumption.

Nomenclature

c	=	Control Horizon
Δ	=	Constant Ratio
f	=	Generic Function
g	=	Earth Acceleration due to Gravity [m/s^2]
g_m	=	Lunar (Moon) Acceleration due to Gravity [m/s^2]
H	=	Hamiltonian Function
H_0	=	Initial Altitude of Powered Descent
I_{sp}	=	Specific Impulse [sec]
J	=	Cost Function
K_i	=	Open-Loop Optimal Control Constant (i = 1, 2, …)
l	=	Length of the Pole
λ	=	Lagrange Multiplier associated with f
m_{total}	=	Total Spacecraft Mass
m_{dry}	=	Dry Mass of Spacecraft
m_{prop}	=	Propellant Mass of Spacecraft
m_c	=	Mass of the Cart [kg]
m_p	=	Mass of the Pole [kg]
μ	=	Lagrange Multiplier associated with ψ
p	=	Prediction Horizon
ψ	=	Terminal Constraints
Ψ	=	Scalar Function associated with Hamiltonian Function

[1] Graduate Research Assistant, Daniel Guggenheim School of Aerospace, Georgia Institute of Technology

Q	=	Weight Matrix for the State in MPC Performance Index
R	=	Weight Matrix for the Control in MPC Performance Index
t	=	Time Step
T	=	Terminal Descent Touchdown Time
T_s	=	Sampling Time [sec]
θ	=	Angle between Initial Downward Pole Location (+y)
u	=	Input Vector
u_1	=	Horizontal Normalized Thrust over Spacecraft Mass [m/s^2]
u_2	=	Vertical Normalized Thrust over Spacecraft Mass [m/s^2]
V	=	Value Function
Δv	=	Delta-V Maneuvers
W	=	Arbitrary Weightage Function
x	=	State Vector
x_{10}	=	Initial Horizontal Velocity Condition [m/s]
x_{20}	=	Initial Vertical Velocity Condition [m/s]
x_1	=	Spacecraft Horizontal Velocity [m/s]
x_2	=	Spacecraft Vertical Velocity [m/s]
x_3	=	Spacecraft Vertical Position (Altitude) [m]
x_4	=	Spacecraft Horizontal Position (Downrange) [m]
CLPS	=	Commercial Lunar Payload Services
DDP	=	Dynamic Differential Programming
GTRI	=	Georgia Tech Research Institute
ISRU	=	*in situ* Resource Utilization
LLO	=	Low Lunar Orbit
LQR	=	Linear Quadratic Regulator
MPC	=	Model Predictive Control
PIL	=	Processor in the Loop
QP	=	Quadratic Programming

I. Introduction

With the increased commercial funding and human activity around the lunar sphere, the ability for small companies and university programs to conduct experiments will become more prevalent. This rapid growth and investment are motivated by global interest to return humans to the Moon and establish a permanent working presence there. While there are significant nationally driven activities such as the Artemis program, ESA's Lunar Pathfinder, and China's Chang'e mission, there has been an equal diversification in enabling the finances for private companies to produce orbiters/landers that will operate in the cislunar domain. These larger orbiters & landers designed by companies such as Intuitive Machines and Astrobiotic Technology have received award funding through the Commercial Lunar Payload Services (CLPS) contract. These companies will open the space for smaller payloads and SmallSats to rideshare to the Moon to save on fuel expenditure and financial costs. The ability to harness Low Lunar Orbit (LLO) delivery services means low-cost scientific demonstrations for *in situ* resource utilization (IRSU) can occur more frequently.

To conduct such demonstrations but in a more cost-effective manner involves accepting a deployment release in LLO and then performing relevant orbital maneuvers that will allow SmallSat-sized landers to touchdown on the lunar surface. Modern spacecraft maneuvers are automated allowing for braking phase and powered descent phase burns without human input. Furthermore, as these orbital maneuvers are done autonomously, they are based on closed-loop control. The difference between open-loop and closed-loop control is the ability to use the current estimated state to help inform the control sequence while factoring in potential external disturbances. Allowing a closed-loop control environment to dictate the ability for a spacecraft to reach the desired landing spot on the Moon will enable more reliable landing capabilities.

II. LISA RUE Mission Concept

A. Purpose of Concept Study

Through a GTRI conceptual study, the LISA RUE Mission Concept was developed as a potential mission profile to land on the surface of the Moon near the South Pole region. LISA RUE stands for the Lunar *in situ* Atomic Resource Utilization Experiment. The goal of LISA RUE is to land a SmallSat-sized spacecraft on the lunar surface to conduct *in situ* experiments. Some of the main driving requirements of this mission involve the following metrics:

a) Using a single thrust output engine land, a SmallSat-sized spacecraft with sufficient propellant to conduct Δv maneuvers through necessary orbit changes.

b) Upon landing extract from the lunar regolith Si-, Fe-, Ni-, K-, Ca- (important metal elements) that can be used for optical and glass applications.

c) Once the precious metals have been processed, the spacecraft will attempt a "ground-based" solar cell fabrication for the inclusion on the SSTEF-1.

From a big perspective, the LISA RUE mission once arriving on the lunar surface will show a demonstration of how to mine lunar resources for space-based manufacturing. Proving such a capability will open the door for other precious mineral-based manufacturing. However, prior to any lunar surface processing, LISA RUE needs to reach the surface of the lunar surface safely.

B. Mission Assumptions

Before diving into the specifics of the closed-loop dynamics, some initial conditions and constraints of the modeling need to be presented. The following analysis was done assuming that the Intuitive Machine Nova-C lander used for "orbital delivery services" will drop off LISA RUE at a 100 km LLO [11]. By choosing not to land on the lunar surface with Nova-C, the monetary budget saving will be ~$100K for LLO insertion versus $1M for lunar surface descent. However, descending from 100 km LLO to the lunar surface involves propellant expenditure and associated mass changes. Thus, the following assumptions were made regarding the spacecraft mass:

a) Knowing that the lander needs to accommodate the processing space for a mining and production payload, a SmallSat ESPA Class Satellite was selected in terms of maximum mass. Hence, the total mass (m_{total}) will be constrained at 180 kg.

b) To provide enough bandwidth for potential propellant mass required the dry mass of the spacecraft (m_{dry}) will be capped at 80 kg.

c) Lastly, in the spirit of reducing the complexity of the dynamics of the powered descent, the mass of the spacecraft was kept constant and modeled as a point mass. The reduction of the propellant mass consumption was not included even though the propellant was approximately half of the spacecraft's total mass. In the future, this will be remediated for more accurate results, but holding mass fixed allowed the analysis to baseline some useful metrics.

Furthermore, in terms of the engine characteristics used for maneuvering, the specific impulse (I_{sp}) selected was based on what was used for the JPL Lunar Flashlight monopropellant chemical propulsion system developed by Georgia Tech. I_{sp} was set to 202 seconds based on the Lunar Flashlight characteristics [5].

C. Mission Concept of Operations (CONOPs)

The LISA RUE mission will be inserted at 100 km LLO altitude from the Intuitive Machines lander before undergoing "Phase II" and "Phase III" that will allow it to successfully descend to the lunar surface. After arriving in LLO, LISA RUE will perform some initial identification of landing spots and then calculate the ideal descent trajectory based on the ideal landing spot and current position in LLO. As shown in Figure 1 and Figure 2, LISA RUE will go from 100 km to 5 km through the braking phase. The braking phase is critical because it will allow the spacecraft to substantially slow down to a velocity that sets up the spacecraft to do a controlled powered descent. The focus of this analysis is to understand the closed-loop simulation of LISA RUE in the powered descent phase, which begins at 5 km.

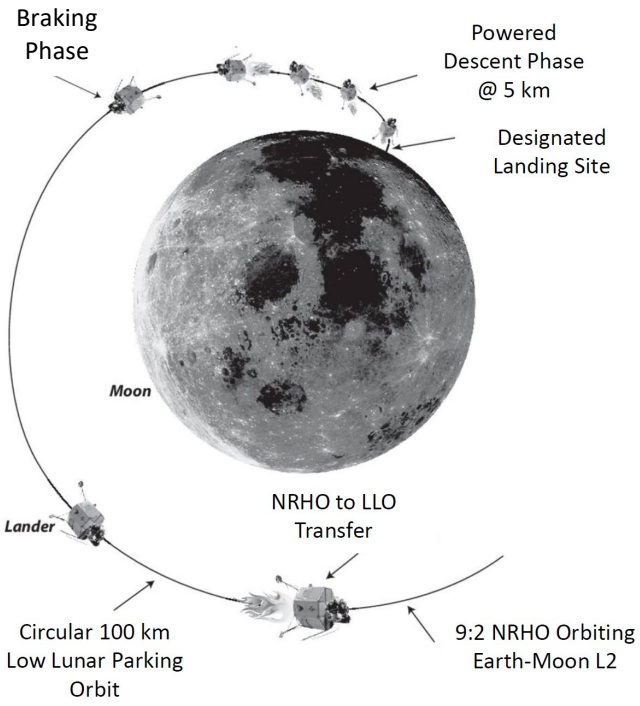

Figure 1. Lunar Descent CONOPs post Intuitive Machines Deployment into LLO [3].

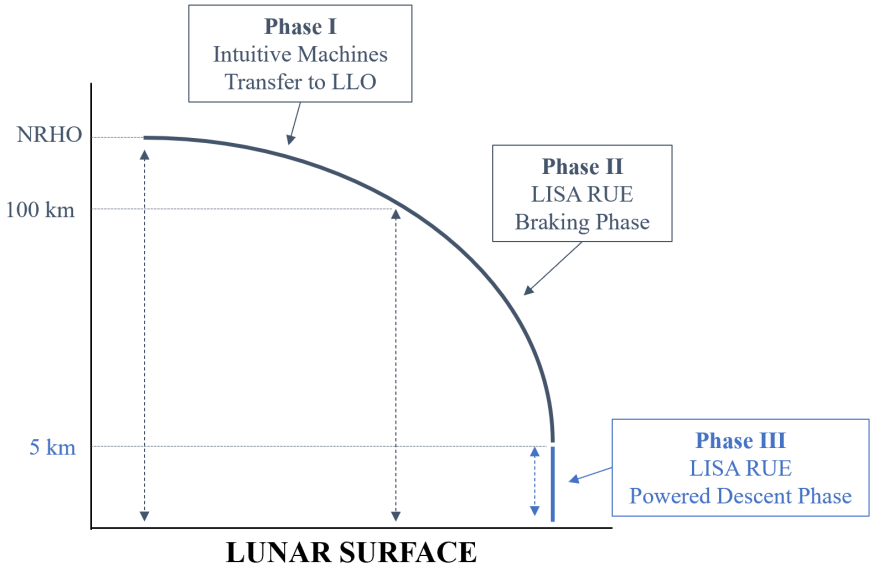

Figure 2. 2D View of the Lunar Surface Approach Phases.

Upon completion of the braking phase, the powered descent will allow a soft landing of the spacecraft. To determine how the powered descent will be controlled, initial conditions needed to be selected. The initial conditions for the powered descent were determined from the braking phase simulation which was completed in a previous analysis done by another student. The two-dimensional final values of braking phase became the initial conditions for the powered descent. To learn more about the braking phase one can reference this paper by Ramanathan and Lightsey [14]. While it would be ideal to achieve the horizontal and vertical velocity of 0 m/s, the final desired values were bounded for soft constraints under the assumption that the LISA RUE spacecraft will have landing gear than can brace the impact force that comes from a non-zero terminal velocity.

III. Spacecraft Model

A. Open-Loop Dynamical Model

The descent and landing of the spacecraft are depicted in Figure 3. The two-dimensional planar motion of the spacecraft, which is modeled as a point-mass, is governed by the following equations of motion:

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 - g \\ \dot{x}_3 &= x_2 \\ \dot{x}_4 &= x_1 \end{aligned} \tag{1}$$

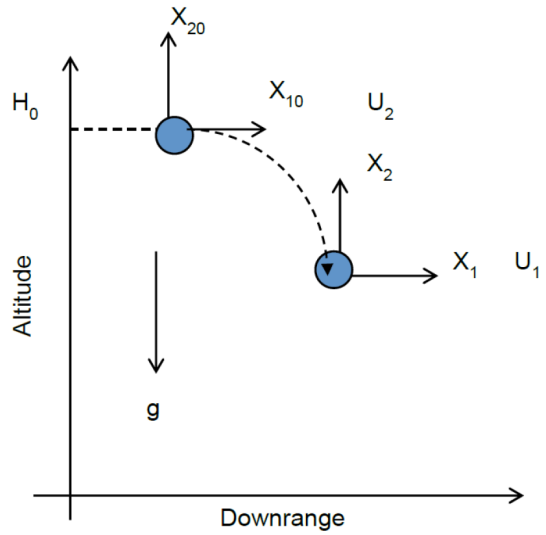

Figure 3. 2D Lunar Descent Free Body Diagram.

Here, $x_1$is the spacecraft's horizontal velocity [m/s], x_2 is vertical horizontal velocity [m/s], x_3 is the spacecraft's landing altitude also known as vertical position [m], and x_4 is the spacecraft's downrange also known as horizontal position [m]. The horizontal and vertical accelerations, also the controlled inputs, are the gimbaled engine thrust normalized over the constant mass of the spacecraft. In this case the horizontal normalized input thrust is u_1[m/s^2] and the vertical normalized input thrust is u_2[m/s^2]. In this optimization problem, the terminal value for the downrange is unconstrained. All the dynamics were derived from the 2011 AIAA Powered Descent paper written by Allan Y. Lee [10].

Furthermore, the initial conditions of x_1, x_2, and x_3 are associated with x_{10}, x_{20}, and H_0 respectively. The following are the initial and final conditions that are used for both the open-loop and closed-loop simulation. As a reminder, the initial horizontal and vertical velocity came from the braking phase simulation done in prior work.

$$x_0 = \begin{bmatrix} 16 \\ 4.7 \\ 5000 \\ 0 \end{bmatrix}$$

$$x_f = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

Now to solve these dynamics, there needs to be a well-posed optimization problem, where a cost function will be minimized. The purpose of the cost function is to minimize the least sum of squares of the landing/terminal velocity at T and fuel consumed which is the summation of the normalized thrust.

$$J = \frac{1}{2}[x_1^2(T) + x_2^2(T)] + \frac{W}{2}\int_0^T [u_1^2(t) + u_2^2(t)]\, dt \tag{3}$$

The first component of the cost function is related to the spacecraft's terminal velocity at the final time. The vertical velocity of the spacecraft at the time of touchdown needs to be minimized because that drives the amount of energy that the landing gear of the spacecraft needs to absorb. It is important to minimize this because more energy can be converted into forward velocity that can cause the spacecraft to tip over upon touchdown. The second component of the cost function is attributed to the consumed fuel by the time touchdown occurs. The weighting parameter, W, in Eq. 3 holds units of seconds and the purpose is to allow the user to determine how much emphasis to place on the consumed fuel in the open-loop analysis. The weighting factor cannot be zero because that would lead to an ill-posed problem since the cost function is typically minimized around the variable control input.

The formulation of the optimal control problem comes from solving the classical calculus of variations technique [4]. The following is the general optimal control problem:

$$\begin{gathered} \min_u J = \phi(x_f) + \int_0^{t_f} L(x(t), u(t), t)\, dt \\ \dot{x}(t) = f(x(t), u(t), t) \\ \psi(x_f) = 0 \end{gathered} \tag{4}$$

In this case, x is the state vector and u is the controlled input vector. The following are the conditions required for the optimization problem where the use of scalar functions H (Hamiltonian function) and Ψ are required. In this derivation, λ is the Lagrange multipler variable that associated with f and μ is the multiplier that coincides with ψ.

$$\begin{gathered} \lambda^T = -H_x \\ H_u = 0 \\ \lambda_f^T = (\Psi_x)_f \\ H(x, u, \lambda, t) = L(x, u, t) + \lambda^T f(x, u, t) \\ \Psi(x_f, \mu) = \phi(x_f) + \mu^T \psi(x_f) \end{gathered} \tag{5}$$

Solving this optimal control problem gives the following input control laws:

$$\begin{aligned} u_1(t) &= -K_1 = \text{constant} \\ u_2(t) &= K_2 t - K_3 = \text{linearly proportional to time} \end{aligned} \tag{6}$$

Optimal control laws are dictated by the constants solved through the methodology explained above. The horizontal component of the normalized thrust is kept constant while the vertical component varies linearly with time. The goal of the vertical component is to reduce the touchdown velocity to zero while minimizing fuel consumption. The following is how the constants were solved:

$$
\begin{aligned}
r &= \frac{T}{W} \\
\Delta &= \frac{T^3}{3}\left(1 + \frac{1}{4}r\right) \\
K_1 &= \frac{x_{10}/W}{1+r} \\
K_2 &= \frac{1}{\Delta}\left(T\left(1+\frac{r}{2}\right)x_{20} + (1+r)H_0 - \frac{g_m}{2}T^2\right) \\
K_3 &= \frac{T}{\Delta}\left(T\left(1+\frac{r}{3}\right)x_{20} + \left(1+\frac{r}{2}\right)H_0 - \frac{g_m}{2}T^2\left(1+\frac{r}{6}\right)\right) \\
K_4 &= K_3 + g
\end{aligned} \tag{7}
$$

Using the solved constants, the velocity and position equations can be constructed. The following expressions are what enabled the open-loop simulation results and discussions by varying weights and starting altitudes.

$$
\begin{aligned}
x_1(t) &= -K_1 t + x_{10} \\
x_2(t) &= \frac{1}{2}K_2 t^2 - K_4 t + x_{20} \\
x_3(t) &= \frac{1}{6}K_2 t^3 - \frac{1}{2}K_4 t^2 + x_{20} t + H_0
\end{aligned} \tag{8}
$$

The summated control inputs, which are the normalized thrust values, over the course of the entire trajectory one can help calculate the expended fuel cost for this open-loop optimization problem:

$$
\Delta v = \int_0^T \sqrt{u_1^2 + u_2^2(t)dt} \tag{9}
$$

B. Closed-Loop Dynamical Model

In the previous section, the process to formulate an open-loop control system for two-dimensional lunar descent dynamics was shown. Optimal control typically refers to open-loop control, while closed-loop control (which will be further expanded into a framework called Model Predictive Control) allows one to design a more robust control algorithm. In optimal control a sequence of input signals is computed that remain predetermined throughout the course of the entire system execution. However, when factoring in the real world, the input sequence calculated through optimal control is not sufficient because there is typically a deviation between the predicted and the actual system behavior. This deviation can be attributed due to actual system dynamics and model mismatches as well as external disturbances such as drag, mechanical constraints, etc. When just using an open-loop model, one can accept the deviation; however, a closed-loop model allows the input to be updated through the error correction. As will be discussed in the MPC section, Model Predictive Control, a form of closed-loop control, allows the system to recompute the optimal input sequence at a finite time step by reevaluating the initial conditions by looking the state deviation.

To set up the lunar dynamics, the equations of motion need to be converted into a state-space model. Because lunar gravity is an external constant not tied to state variable, it needs to be modeled at an external disturbance vector. The general state space model is structured as: $\dot{x} = Ax + Bu + w$ and $y = Cx + Du$.

$$
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -g \\ 0 \\ 0 \end{bmatrix}
$$

$$y = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + [0] \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{10}$$

This state space model, to fit into the MPC framework, had to be augmented such that the disturbance vector was incorporated into the standard state space matrices. The following is the line of code in MATLAB that allowed the augmentation process:

```
>> LunarMPCModel = ss(A,[B F],C,[D zeros(size(C,1),size(F,2))]);
```

C. Open-Loop Simulation Results

Referencing Section III.A for Open-Loop Dynamics, the following are the output results to understand how varying different unconstrained variables impacts the trajectory and fuel consumption.

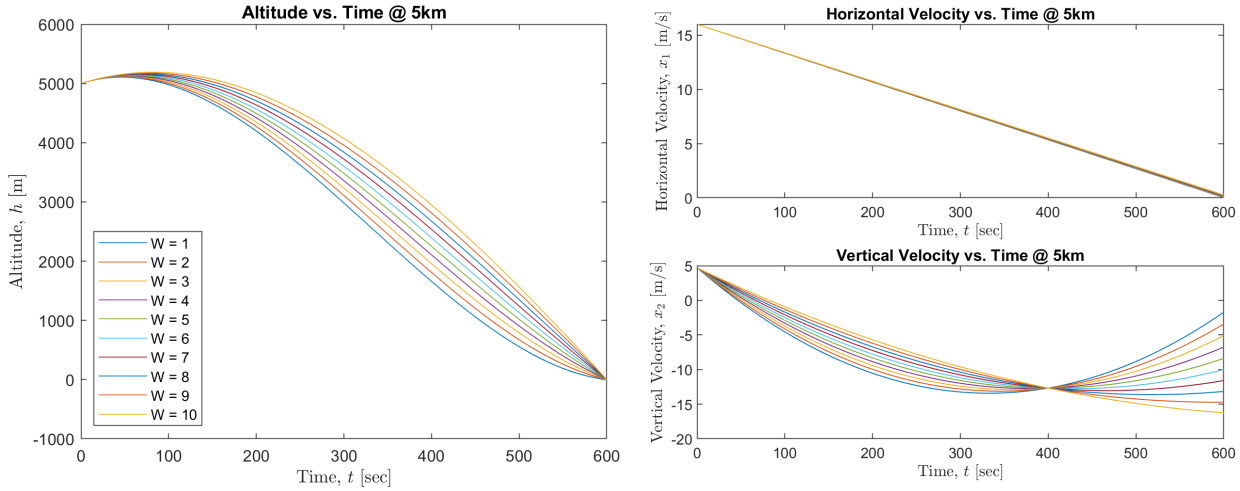

Figure 4. Altitude vs. Time (left) and Velocity vs. Time (right) for Open-Loop Analysis.

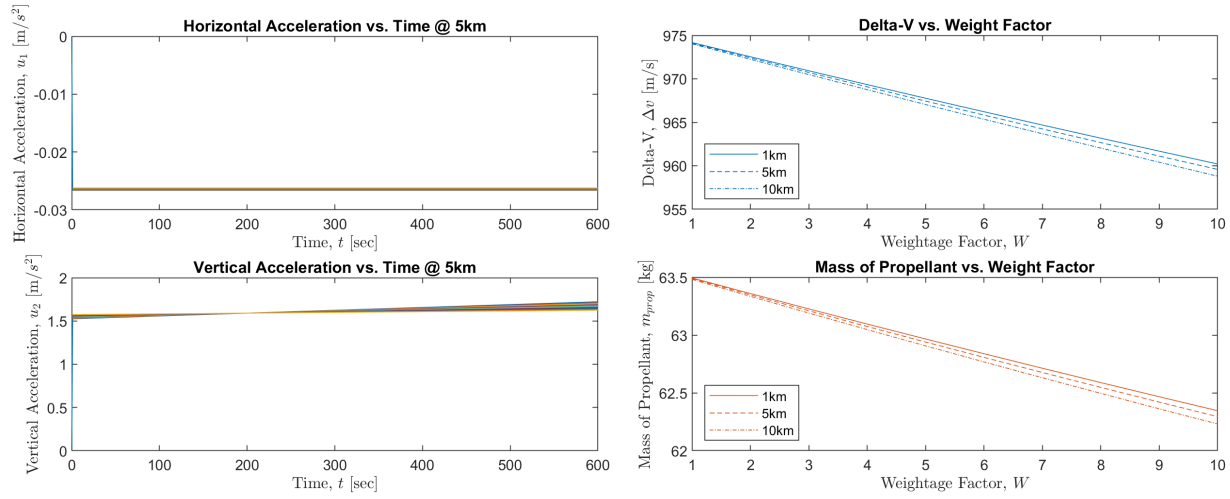

Figure 5. Acceleration vs. Time (left) and Mass of Propellant vs. Weight Factor for Open-Loop Analysis.

Given the final values seen from Figure 4 – the touchdown performance in terms of final altitude and velocity achieved are all within the performance metrics dictated for the final state. The variation of magnitudes for the touchdown velocities are due to the weighting factor, W. The large weight of W will contribute to a large touchdown velocity but less fuel consumption. The reason for this is because the larger weightage of W will ensure less normalized thrust usage, however, less aggressive thrust usage will cause the spacecraft to have a large touchdown vertical velocity and be unable to slow down as much. Essentially, only a $W = 1$ weightage allows the touchdown to reach a near-zero velocity, otherwise a negative velocity arises meaning that the spacecraft goes back up in altitude which is unfeasible with the landing requirement.

Figure 5 shows that the weightage does not impact the horizontal and vertical acceleration (also known as normalized thrust), which then translates into less than 1.5 kg difference in mass of propellant calculated. Given that the only state being impacted by weight significantly is vertical velocity this helps drive that a weightage of one is the best selection. Furthermore, the mass of propellant was between 62 to 63 kg when added to the 80 kg dry mass ensures that the spacecraft sizing does not violate the 180 kg ESPA-class total mass cap.

IV. Using Model Predictive Control to Simulate Closed-Loop Dynamics

A. Model Predictive Control Algorithm

Model Predictive Control refers to the control problem that takes a linear time invariant (LTI) system whose dynamics have been discretized at certain a specific time sampling value. The model designed is subjected to some uncertainty in the form of unknown external disturbances or imprecise modeling of the system mechanics. The system dynamics are described in an LTI state-space model described in Section II.B.

A classic regulating controls problem such as Linear Quadratic Regulator (LQR) is simply concerned with allowing a system state to achieve a desired setpoint/final value using an acceptable amount of control effort. In the case of LQR and MPC, both are centered around calculating the control input by minimizing a performance index also known as a quadratic cost index which takes the form in the equation below. Q and R are weighted matrixes that place emphasis on specific states and inputs in the cost [9].

$$J(x_0, \{u_0, u_1, u_2 \dots\}) = \sum_{k=0}^{\infty} \left(\|x_k\|_Q^2 + \|u_k\|_R^2 \right) \tag{11}$$

With the increase in computing power on flight computers, implementing MPC has become significantly more feasible as a close-loop control system. In Figure 6, the Model Predictive Controller workflow is depicted. MPC essentially takes in a setpoint or reference value, typically the final state, and attempts to iterate until a minimized input helps reach that state. The optimizer minimizes the cost function and then within the controller the predicted output is compared against the setpoint. The control input is computed at the sampling interval based off the parameters which minimize the performance index. Factoring in an optimizer allows MPC to minimize the input and then adjust the input based on external disturbances.

The main advantage of MPC involves its ability to handle constraints and heed them in during the optimization process. This is especially important with lunar descent dynamics where going past lunar surface due to normal thrust inputs is not a feasible option to achieve zero touchdown velocity. The ability to change the optimal manipulated input that satisfies the active constraints at each time step makes MPC a robust controller method [7] [8].

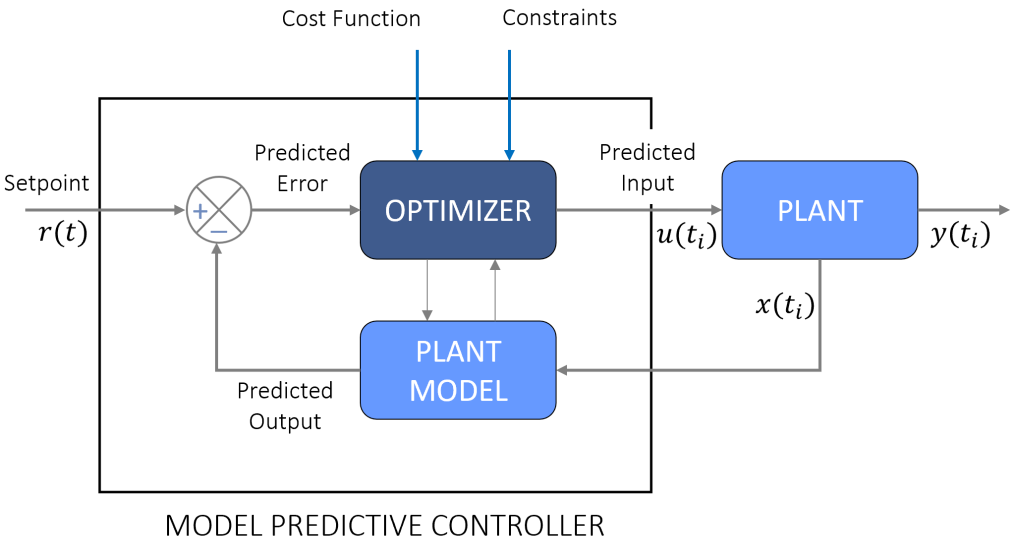

Figure 6. Model Predictive Controller Workflow.

Another fundamental aspect of MPC are the parameters that can be tuned to help the controller reach the desired setpoint within the constraints. Prediction horizon (p), control horizon (c), and sampling time (T_s) all impact computational performance and the ability for the control to converge about the desired setpoint. Specifically for this project there was not much analysis done on optimal selection of parameters and more of a trial-and-error method was employed. However, here are some metrics and explanations as to how p, c, and T_s can be selected [19]:

a) Sample time (T_s) is typically a starting guess that allows about 10 to 20 samples covering the rise time of the open loop plant.

b) Prediction horizon (p) is the number of samples over which the controller calculates minimizing the cost. It is typically set so that it captures the complete response till steady of the system dynamics. Typically, a larger prediction computational performance.

c) Control horizon (c) is a small subset of control samples used to minimize the cost over the prediction horizon. The controller optimizes over a receding horizon and can typically be as long as the prediction horizon but is usually kept much shorter since one is just trying to capture the portion of the state that is most turbulent. The length of the control horizon also impacts computational performance.

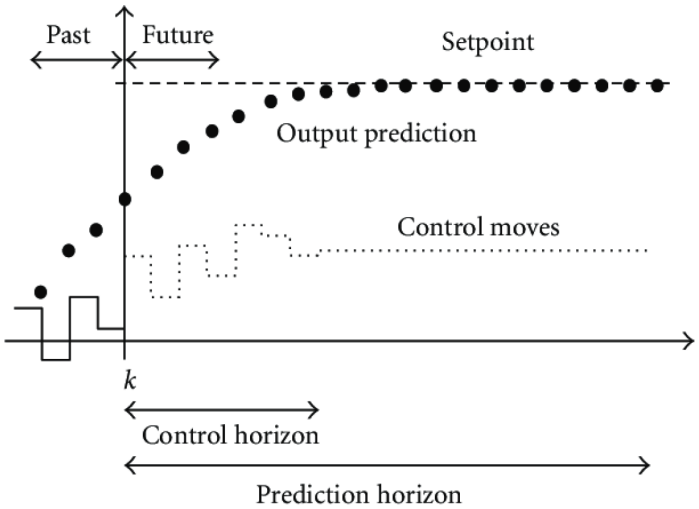

Figure 7. Variable Parameters of Model Predictive Control [16].

Given the progression of the project, for the lunar dynamics MPC modeling, the MATLAB toolbox was utilized. The MATLAB MPC toolbox allowed the abstraction of the controller mechanics through in-built functionality. In Engel 2022, the toolbox was used to model the nonlinear dynamics of a blunt body Mars entry vehicle. However, given the need to implement the controller on an embedded system an MPC algorithm was designed from the ground-up to simulate cart-pole dynamics – this will be elaborated upon in future sections [6] [12].

B. Dynamic Differential Programming (DDP) Optimizer

Given that the foundation of DDP is based on LQR, it was decided to use DDP as the optimizer in the custom MPC algorithm that was tested and validated using cart-pole dynamics. DDP is extremely useful because it scales linearly with time and is used specifically for trajectory optimization, while QP is just a general solver because it finds controls and states simultaneously. Unlike LQR, DDP solves optimal control problems through solving localized cost function and dynamics and then iteratively solving the sequence recursively through the time steps.

The algorithm for DDP is significantly expanded upon in Tassa, et al. 2014, however the following are important equations that allow the iterative optimizer of the input to occur. The value function is the cost function evaluated at a fixed state. Thus, at each time step in MPC has an associated state at which the theoretical minimum input is calculated for a state [17].

$$V(x,i) = min_u \, J_i(x,u_i) \tag{12}$$

Dynamic

The DDP algorithm essentially solves an optimal control problem by "recursively solving the value-function for a single optimal control problem, proceeding backwards in time, hence the name dynamic programming". By minimizing control at each time step and then calculating recursively backwards, DDP can build the full sequence of controls for a specific prediction horizon.

$$V(x,i) = min_u \, [l(x,u) + V(f(x,u), i+1)] \rightarrow \text{Backwards Recursive Stepping}$$
$$V(x,n) = l_f(x) \rightarrow \text{Base Case} \tag{13}$$

Differential

The differential aspect of DDP essentially solves an equivalent optimal control off a nominal trajectory and then compare the two value functions. The difference in the perturbations is the differential or change. The value function listed below shows the general formulation accounting for the perturbation:

$$V(x,i) = min_{\delta u} \, [l(\hat{x}+\delta x, \hat{u}+\delta u) + V(f(\hat{x}+\delta x, \hat{u}+\delta u), i+1)] \tag{14}$$

C. Quadratic Programming (QP) Optimizer

In the MATLAB MPC toolbox, a QP optimizer is utilized. Because the toolbox QP solver converts a linear MPC optimization problem to a general QP problem, it can minimize the performance index it creates. QP is significantly smarter at dealing with constraints than DDP, but not as open source in terms of mathematical formulation as DDP.

$$min_x \left(\frac{1}{2}x^T H x + f^T x\right)$$
$$Ax \leq b \tag{15}$$

In the problem stated above, x is the solution vector, H is the Hessian matrix that represents the dynamics and weighting matrices at a fixed time stamp, A is the matrix for constraints, and b and f are other vectors [13].

D. Prediction Horizon and Sampling Time Impacts on Lunar Dynamics

After designing an MPC controller using the toolbox, the prediction horizon, control horizon, and sampling time were tuned to achieve a trajectory that did not violate the hard altitude and velocity constraints. As can be seen between Figures 8 - 10, not having properly tuned prediction horizon and sampling times can lead to overshooting and constraint violations.

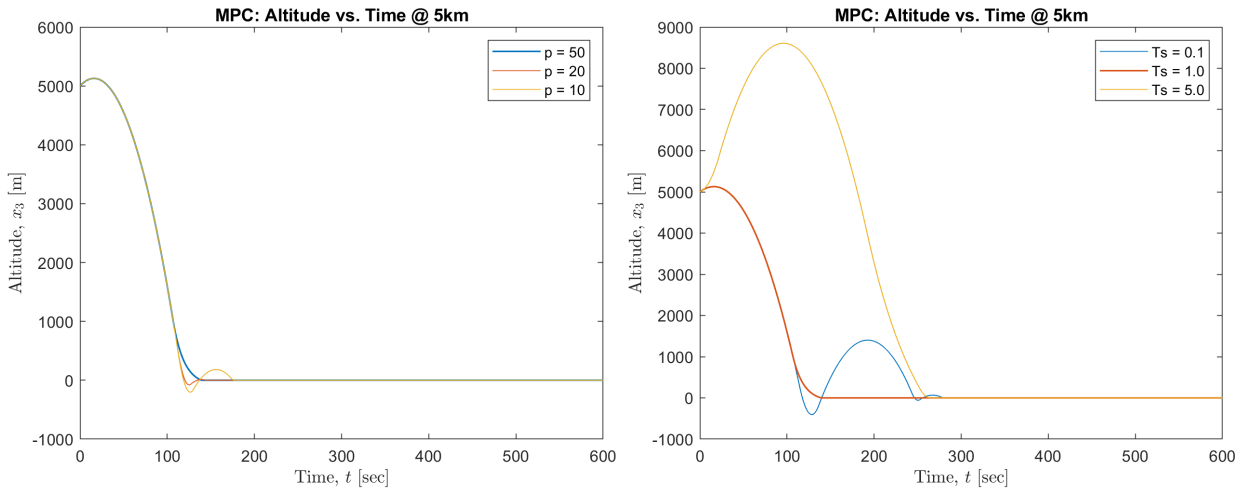

Figure 8. Model Predictive Control for Altitude vs. Time while varying Prediction Horizon (left) and Sampling Time (right)

The control parameters set for the lunar dynamics simulation were $p = 50$, $c = 10$, and $T_s = 1.0$. These parameters lead to the quickest descent while not violating the 0 km hard constraint for altitude final value. Even though all the prediction horizon options: 10, 20, and 50 reached 0 km around a similar touchdown time, it was evident that not looking far enough ahead till more transient response appears caused the system to violate the altitude constraint.

Regarding the sampling time, having a significantly smaller sampling time impacts the total prediction horizon length and total control horizon length. Hence, a similar issue arises with a smaller sampling time the controller loses the ability to look ahead enough with its control input optimization. However, having a large sampling time at 5.0 seconds did not violate the altitude constraint but took significantly longer to descend and furthermore caused the spacecraft to raise in altitude prior to beginning in descent which would lead to more fuel consumption.

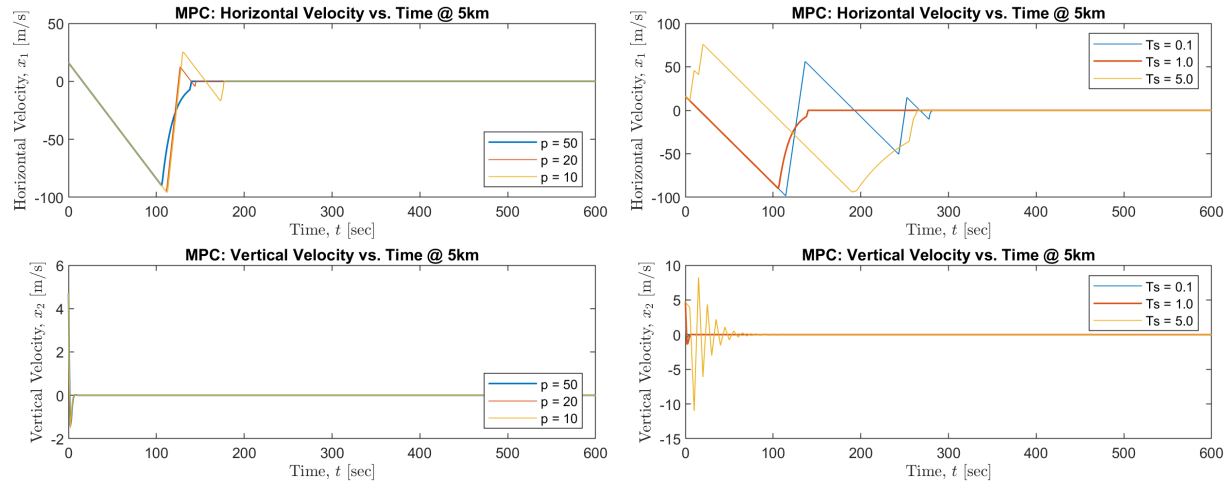

Figure 9. Model Predictive Control for Velocity vs. Time while varying Prediction Horizon (left) and Sampling Time (right)

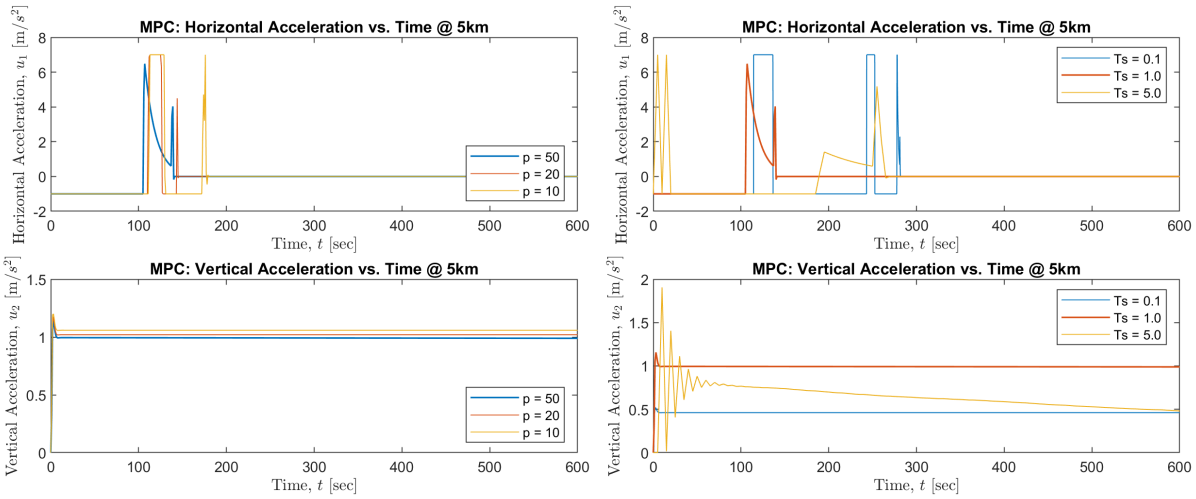

Figure 10. MPC for Acceleration vs. Time while varying Prediction Horizon (left) and Sampling Time (right)

For both velocity and input acceleration (normalized thrust) the larger sampling time led to a significantly unstable input normalized thrust and vertical velocity. Such an erratic thrust output is not possible with actual flight hardware. The prediction horizon variations did not have drastic differences in output for velocity and acceleration but when viewed from the lens of the altitude profile it illuminated what the proper tuned value was required.

V. Implementation of MPC on Embedded Hardware

A. Executable Development in MATLAB

Most controllers, when only implemented on a host machine, do not give context to how it could potentially be implemented on an actual spacecraft. Thus, seeing how the model predictive controller would operate on an actual embedded system gives insights into parameters impacting computational performance in a flight-like setting. High-level, running a standalone executable, also known as processor-in-the-loop (PIL) testing, gives the engineering team confidence that the flight software runs correctly by verifying the code's execution on the target processor. PIL mode allows satisfies verification objectives without manual code review. Without having the knowledge of programming in C, one can utilize the MATLAB Autocoder feature to generate code that could turn into an executable and the PIL confirms that the code generation works correctly. By running executables on the target, one can discover things like unexpected behavior due to cross-compiler settings tack overflows due to smaller memory on-target, etc.

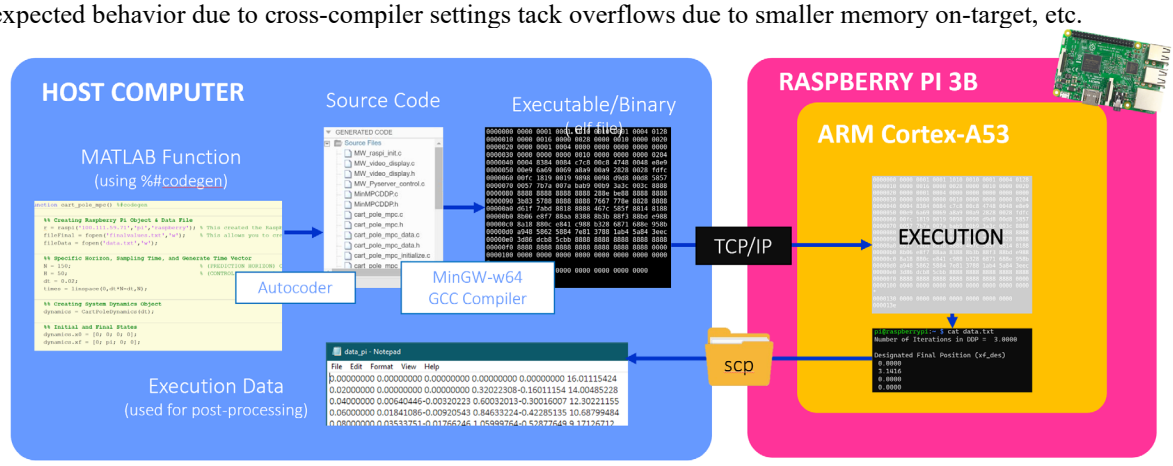

Figure 11. MATLAB Execution Generation Process.

The workflow shown in Figure 11 involves using the MATLAB Autocoder to generate compatible C files. Then the MinGW-w64 GCC compiler generates an executable or binary that will then be subsequently executed on the Raspberry Pi 3B. The specific compiler listed is important to generate an executable that can successfully be run on an ARM Cortex-A53 processor. The MATLAB code was outfitted with opening a text file where the data from the algorithm is stored for post-processing. Then, through a secure copy, the data can be brought back onto the host computer to post-process the data.

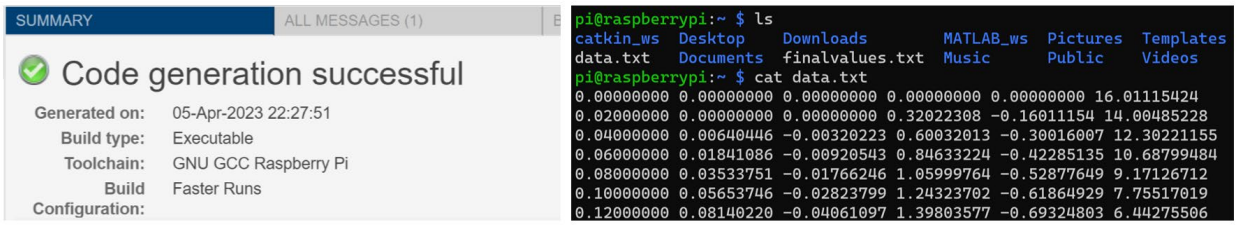

Figure 12. Successful Executable (.elf) Build (left) and Data Dump into Text File on Pi (right).

The figure above the successful executable build through MATLAB's PIL infrastructure and furthermore the state and input output in a data file that could be viewed on the Raspberry Pi.

B. Space Radiation Considerations

While the Raspberry Pi does provide the platform to see how an embedded system reacts to executing a flight-like controller, one would be remiss to not account for radiation impacts on electronics aboard a lunar mission. Given previous studies done on the Raspberry Pi's ability to handle radiation, it is very clear that the Pi can only operate in Low Earth Orbit (LEO) rather than Low Lunar Orbit. The cislunar space experiences approximately 150 times more radiation than Low Earth Orbit. NASA typically requires LEO SmallSat avionics components to withstand 20-50 krad total dose of radiation with an 10^{-7} to 10^{-8} errors/bit-day. Furthermore, Figure 13 shows that the Raspberry Pi has the ability to boot up and run normally up through 40 krad and levels beyond that lead to peripheral connections experiencing issues [18].

TID (krad(Si))	Condition DUT 2	Condition DUT 3	Condition DUT 4	Condition DUT 5	Condition (DUT 1 – Control)
10	Nominal	Nominal	Nominal	Nominal	Nominal
20	Nominal	Nominal	No Logon	Nominal	Nominal
30	Nominal	Nominal	Nominal	Nominal	Nominal
40	Nominal	Nominal	Nominal	Nominal	Nominal
50	No Logon	No Logon	No Logon	Nominal	Nominal
60	No Test	No Test	No Logon	No Logon	Nominal
70	No Test	No Test	No Logon	No Logon	Nominal
80	No Test	No Test	No Logon	No Logon	Nominal
100	No Test	No Test	No Logon	No Logon	Nominal
120	No Test	No Test	No Logon	No Logon	Nominal
150	No Test	No Test	No Logon	No Logon	Nominal

Figure 13. Testing Data for Raspberry Pi Boot-Up based on Radiation Doses.

Knowing that the Raspberry Pi has limitations in terms of operating with robust reliability to higher levels of radiation, it begs the question what could potentially be a better alternative. Given the command and data handling success of the Lunar Flashlight mission, the use of the JPL Sphinx Board would be a good alternative to further the testing of the MPC algorithm on a board that can handle lunar levels of radiation. The Sphinx Board has several space-grade parts that performance flawlessly at minimum 30 krads of radiation. Furthermore, the GR72 processor itself has a tolerance up to 300 krads. In terms of dealing with bit flipping due to radiation, the Sphinx Board carries up to four copies of the flight software executable image. Hence if a reset occurs, the Sphinx can reflash an uncorrupted version of the flight software [15].

C. Cart-Pole Dynamics

Because of the decision to use the MPC toolbox through MATLAB and the added complexity of requiring custom MEX functions to explain the toolbox functions, a simpler problem was implemented with verified results to show how MPC would run using DDP for cart-pole dynamics on a custom, "written from scratch" algorithm. The goal of the controller was to stabilize the pole vertically upright but with no change to position and angular or linearly rates. The final angular position vertically would be equivalent to 180 degrees.

The initial state is $x_0 = [0 \quad 0 \quad 0 \quad 0]^T$ and the final state is $x_f = [0 \quad \pi \quad 0 \quad 0]^T$. The following are the cart-pole dynamics used for the system propagation with the additional Jacobian (also known as the gradient) of the system dynamics and control input. The Jacobian is required to linearize the cart-pole nonlinear dynamics so that the MPC system is capable of iterating.

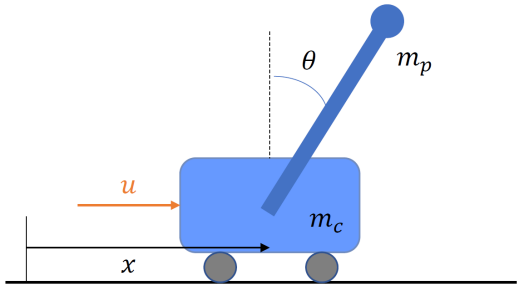

Figure 14. Cart-Pole System.

$$\begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{bmatrix} = Ax = f = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \dfrac{m_p \sin(\theta)\left(l\dot{\theta}^2 + g\cos(\theta)\right) + u}{m_c + m_p \sin(\theta)^2} \\ \dfrac{-m_p l\dot{\theta}^2 \cos(\theta)\sin(\theta) - (m_c + m_p) g \sin(\theta) - \cos(\theta)\, u}{l(m_c + m_p \sin(\theta)^2)} \end{bmatrix} \tag{16}$$

$$\frac{\delta f}{\delta x} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots \end{bmatrix} \rightarrow \text{Complex Dynamics (Jacobian)} \tag{17}$$

$$\frac{\delta f}{\delta u} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \dfrac{}{m_p \sin(\theta)^2 + m_c} \\ \dfrac{-\cos(\theta)}{l(m_p \sin(\theta)^2 + m_c)} \end{bmatrix} \tag{18}$$

The decision to use DDP was addressed above, but the following figure shows how the MPC algorithm flowed.

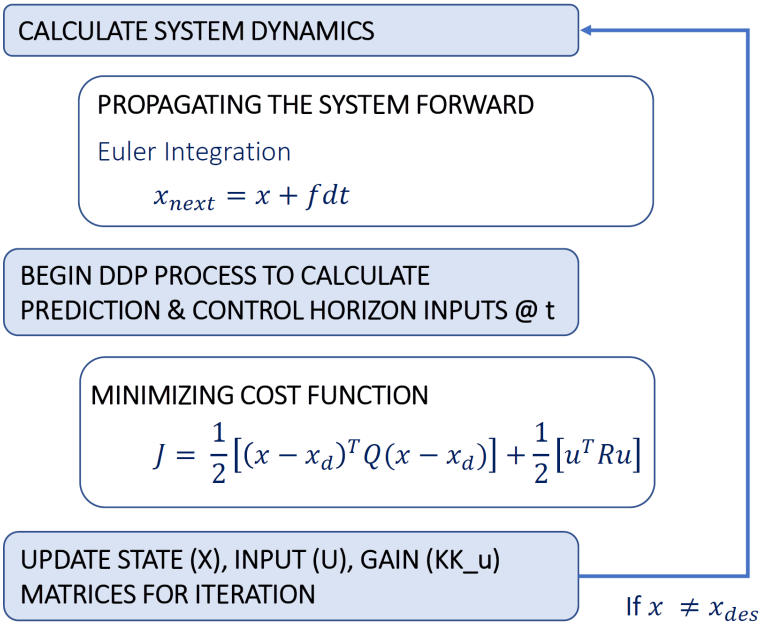

Figure 15. MPC-DDP Code Workflow.

The MATLAB code was structured in an object-oriented programming manner. The main function initializes the cart pole dynamics as an object and passes the initial and final conditions. The cost function object was created and passed the state running cost matrix, input running cost matrix, and final state difference cost matrix. Then, variable allocation occurred for all the inputs and states to be stored at each DDP iteration. The DDP process, which was described early, begins iterating to optimize the state closest to the final desired value and then outputs the correcting gain. Then the MPC retains the control horizon sized input to feed back into the simulation until the state is equal to the desired final state based on the convergence threshold [2].

VI. Comparison of MPC on MATLAB versus Embedded Hardware

A. Validation & Verification Metrics

To validate that the MPC algorithm in MATLAB autocoded to a functional executable, the algorithm was run both on MATLAB as well as the Raspberry Pi. Because the operating system of the Raspberry Pi, through the MATLAB add-on package, is a 32-bit system, the assumption is that the bit structure follows the IEEE 754 Float 32-bit Single Precision. Given the size of the mantissa in the IEEE 754 structure, the approximate absolute normalized range is from 10^{-38} to 10^{38} in terms of the value of the number. However, the most important aspect to validate the results is ensuring that the precision between the MATLAB and the Pi results match to a minimum of seven decimal digits. That order of precision allows the difference between the Pi and MATLAB to be within an acceptable range.

B. Prediction Horizon Impacts

The goal of this section and the following is to show the results of varying one of the MPC parameters and the results from a MATLAB run and a Raspberry Pi execution. When taking the difference between the MATLAB and Raspberry Pi data there was no difference – the error was exactly zero up to the required seven digit 32-bit precision.

For the cart-pole dynamics the system almost achieved the setpoint in the fastest time span with the following parameters values: $p = 150$, $c = 50$, and $T_s = 0.02$. When the prediction horizon was equal to 250, it did reach the setpoint much faster, but because of the significantly larger prediction horizon, the computational time which can be seen in Section VI.D was much longer. On the other hand, a prediction horizon of 60 did not give the system enough "prediction runway" to see the state achieve transient response. As a result, the cart-pole system was not able to reach the desired setpoint. Trying to strike a balance between the parameters and the computational load is critical when trying to implement an MPC controller on actual flight hardware.

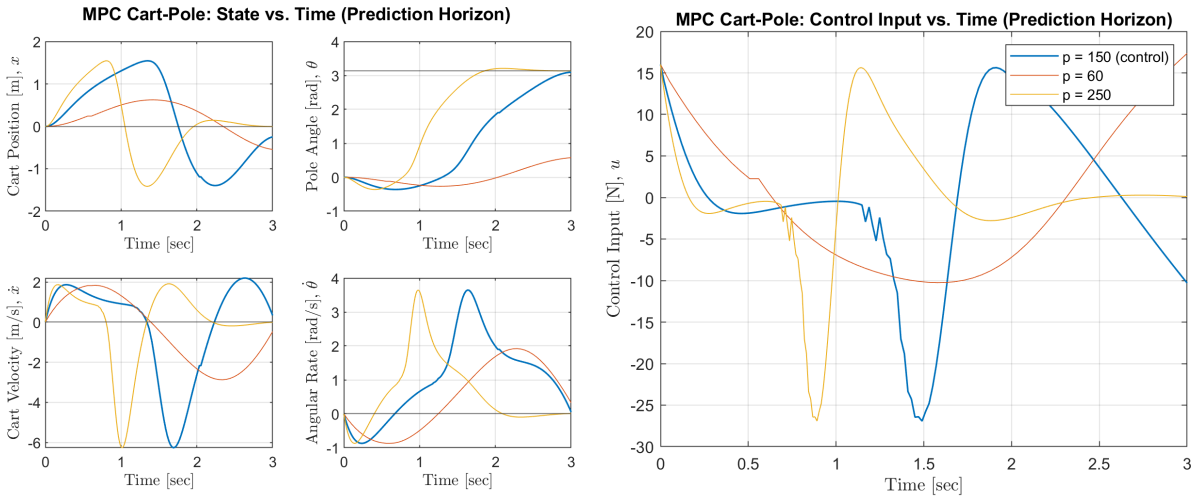

Figure 16. Varying the Prediction Horizon for State and Control Input in MATLAB Execution.

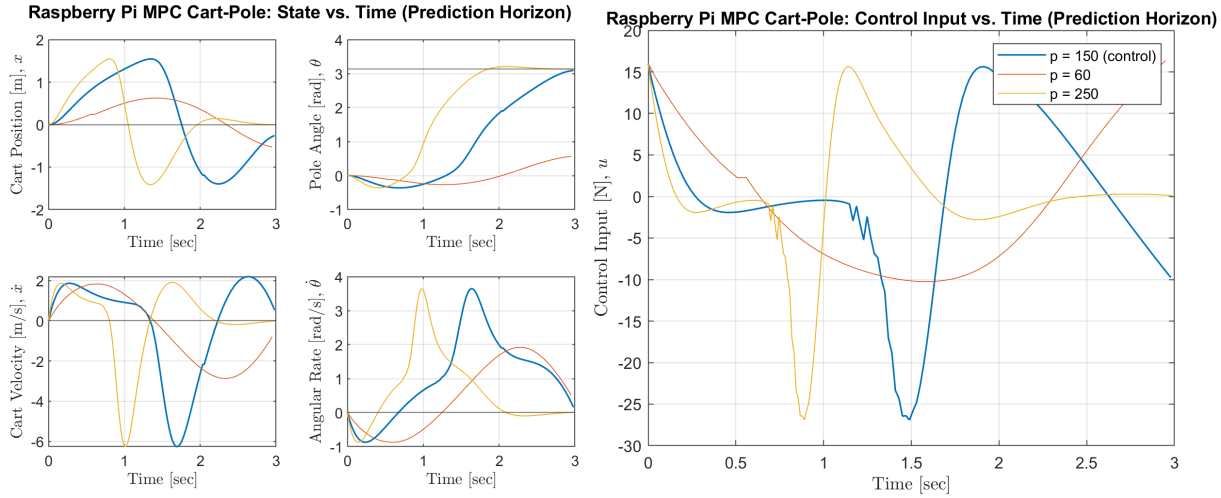

Figure 17. Varying the Prediction Horizon for the State and Control Input in Raspberry Pi Execution.

C. Control Horizon Impacts

Like the process conducted for the prediction horizon verification, there was virtually no difference between the Raspberry Pi and MATLAB executions. The level of precision between both styles of running the MPC-DDP algorithm shows the ability for the process to successfully run on a flight-like processor.

As mentioned earlier, the control horizon is essentially what is fed into the final state and control input at a specific time step. When the control horizon is too small (like 10 steps) for a prediction horizon of 150, the system has an unstable response and does not achieve the desired setpoint. The outcomes of not properly tuned prediction and control horizons are similar in nature. Conversely, a very large control horizon does allow the system to reach a significantly closer final value, in terms of deviation from the setpoint, however the computation intensity of calculating a much large control horizon impacts computational performance. In comparison to a control horizon of 50 steps, the 140 version does minimize the error more which leads to a potential trade-off when extrapolated to the lunar descent dynamics. The state vector does not have much of a difference between angular position and rate and the 50 steps control horizon is better for the lateral position and rate. As a result, the need for a control horizon that is almost the same length as the prediction horizon is unnecessary.

In a real-life flight setting, picking a prediction and control horizon that allows the spacecraft to land safely within the landing margin of error and not violating the processor's memory capabilities is a critical decision.

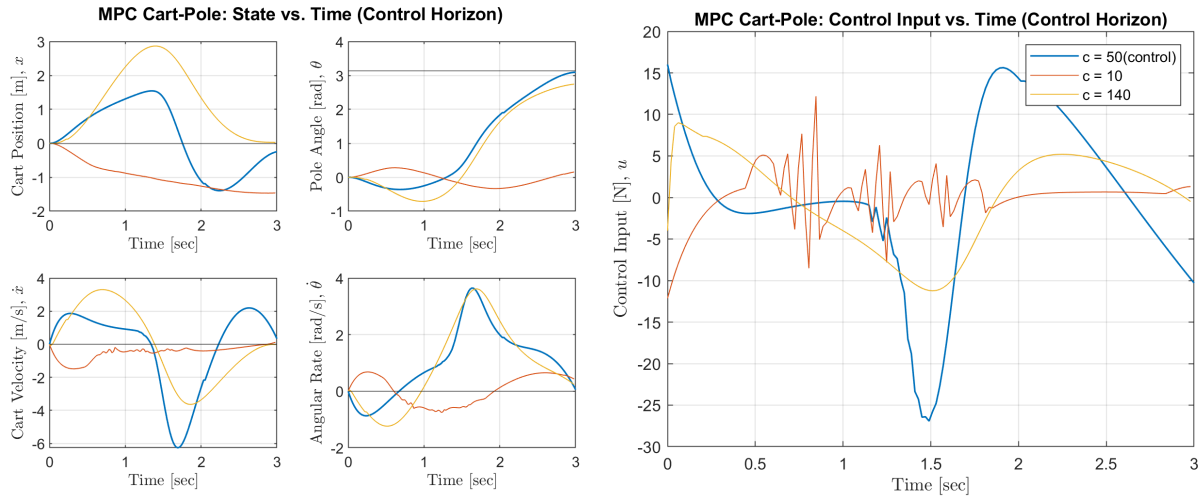

Figure 18. Varying the Control Horizon for the State and Control Input in MATLAB Execution.

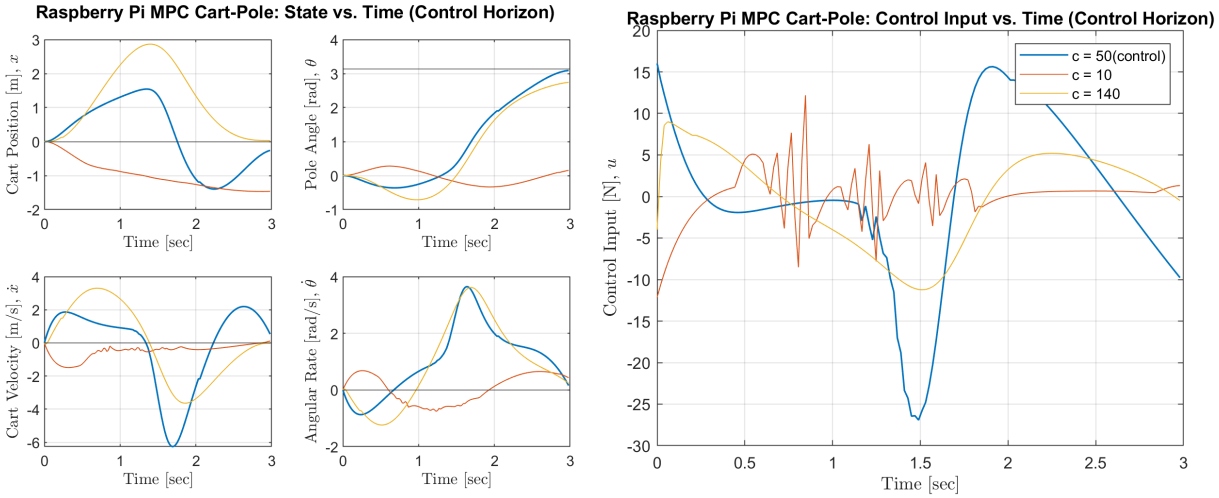

Figure 19. Varying the Control Horizon for the State and Control Input in Raspberry Pi Execution.

D. Runtime Differences

Another important aspect of understanding controller executions on a SmallSat tailored processor involves looking at runtime speeds. In a situation like powered descent, the ability to make quick, real-time adjustment to the reaction-control system to prevent a sideways landing or impact upon landing is for a processor to be able to compute at a relatively fast speed. To see how quickly the Raspberry Pi could run a 3-minute cart-pole dynamic simulation, the "time" function was used to register the speed and a similar mechanism with MATLAB through the timeit() function. The timeit() function in MATLAB runs the algorithm several times and returns a median time.

Figure 20. Timing Execution on Pi (left) and MATLAB (right)

The following tables show the runtime differences between the Pi and MATLAB when varying the prediction horizon and control horizon as discussed in the previous sections.

Table 1. Comparison between MATLAB and Raspberry Pi Run Times for Prediction Horizon Changes

Prediction Horizon	MATLAB Runtime [sec]	Raspberry Pi Runtime [sec]
60	0.1153	0.174
150	0.8428	0.209
250	1.5538	0.258

Table 2. Comparison between MATLAB and Raspberry Pi Run Times for Control Horizon Changes

Control Horizon	MATLAB Runtime [sec]	Raspberry Pi Runtime [sec]
10	0.2636	0.170
50	0.8428	0.209
140	0.2752	0.166

The comparison shows that the MATLAB execution was slower than the Raspberry Pi. This is interesting to document because in practice, a Windows 64-bit host computer with an Intel Core i7 processor should have several orders of magnitude faster processing time than a 32-bit Raspberry Pi 3B. There are several potential causes that can explain this unexpected difference:

a) MATLAB is an application running on the host machine. Thus, the time it takes to call the classes and create the objects within its application framework takes significantly longer than the Pi directly running a compiled and linked executable ready to run in binary format on its processor. A more accurate representation of this test would be to cross-compile the MATLAB autogenerated C code into an executable that could run directly on the host computer.

b) Another important distinction is that the host computer is running several other processes while the majority of the Raspberry Pi's memory was being allocated to running the executable.

E. Lessons Learned

Throughout the course of this project there were several unsuccessful attempts at designing a model predictive controller that was capable of running on an embedded system. The following is a summary of some of the lessons learned from this project:

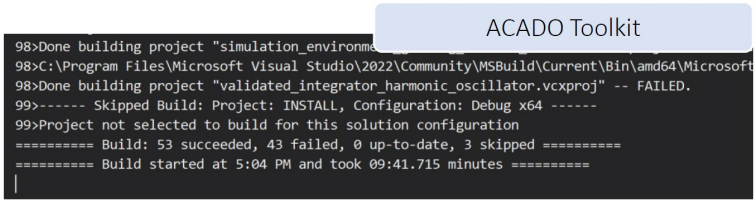

Figure 21. Failed ACADO Toolkit CMake Build.

Using the ACADO Toolkit, which was designed to be a user-friendly package to design an MPC controller and then generate an executable to run on several select embedded processors, had limited documentation post 2013. The complexity behind using Visual Studio and CMake in conjunction with MATLAB unfortunately did not produce any reasonable results. Figure 21 shows that attempt at trying to build the ACADO Toolkit however about half of the projects failed to build [1].

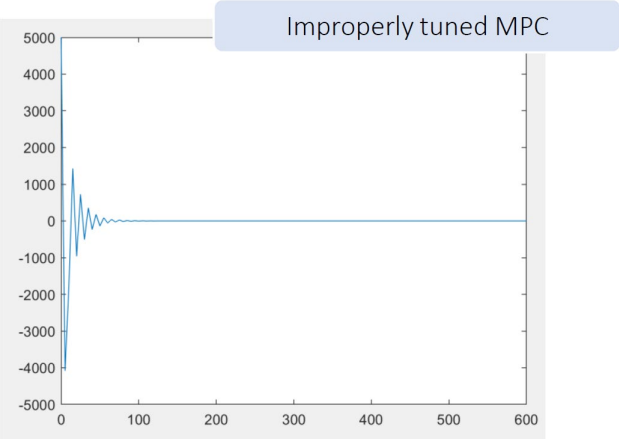

Figure 22. Improperly tuned Lunar Descent MPC using toolbox.

The largest takeaway from this project involved the usage of the MATLAB MPC Toolbox. While the toolbox allowed to simplify the process of writing all the optimizer algorithm, the "black-box effect" of the toolbox lead to several issues:

a) It is harder to properly tune the MPC and due to the nature of the controller wanting to quickly achieve steady state and the desired setpoint, the controller is very likely to violate hard and soft constraints. In the case of the lunar lander situation that caused it to simulate going 4000 meters beneath the lunar surface on initial tuning attempts as shown in Figure 22.
b) Using custom MATLAB functions made it difficult to use the Autocoder to convert the code into functional C code without implementing custom MEX functions. Hence it is better to understand the mechanics by implementing one's own algorithm when attempting to translate code that can operate on an embedded system.
c) When just operating in the MATLAB framework, the toolbox is not capable of linearizing the dynamics. Specifically, if one wanted to linearize the dynamics that would require using Simulink which again abstracts the controller design making it difficult to understand its implementation on the Raspberry Pi.

Future Work

There are several areas of potential future work that can build off the foundation laid above. Specifically in the realm of the spacecraft model one can expand the model into nonlinear dynamics by not assuming constant mass and possibly expanding into three dimensions. Furthermore, folding in the braking phase such that the initial state for powered descent varies more accurately with altitude changes.

Specifically with the model predictive controller algorithm, one should phase out the MPC toolbox and only use the code that can run on the Raspberry Pi and replace the cart-pole dynamics with the lunar descent dynamics. Redesigning the code such that it is flexible for different time sampling values is an important future improvement. Furthermore, right now Gaussian white noise is used to simulate the difference between the plant and plant model but replacing that would realistic sensor sense would produce more error deviation values. Lastly, regarding the embedded systems implementation, one should replace the Raspberry Pi with a flight computer that is radiation compliant for lunar standards.

Acknowledgments

I would like to thank the people who have helped me reach this point in my education. Thank you to Dr. Lightsey for giving me the opportunity to work on this project and enabling me to gain relevant skills useful to the GNC field within the aerospace industry. This project allowed me to gain invaluable experience that will be immensely helpful for the next portion of this project and future jobs. Thank you for your continued support throughout my graduate career.

I would also like to acknowledge the technical help of Keshav Ramanathan who contributed to this project's premise and provided me a foundation for braking phase and powered descent analysis that I used as a starting point. Furthermore, I would like to extend gratitude to Joshua Kuperman and Andrew Fear who helped me understand the premise of model predictive control and how to approach designing the algorithm. Lastly, thank you to my former internship boss at Blue Origin Dr. Matt Jardin for allowing me to learn the skills needed to implement a processor-in-the-loop framework during the Summer of 2021. Also, thank you to my roommate Allison Schwoboda who has endlessly inspired me to keep chipping at research when roadblock presented themselves. Finally, thank you to my parents and younger brother for always being my unwavering support system.

References

[1] Adhau, S., Patil, S., Ingole, D., and Sonawane, D., "Implementation and analysis of Nonlinear Model Predictive Controller on embedded systems for real-time applications," *2019 18th European Control Conference (ECC)*, 2019.

[2] Almubarak, H., Stachowicz, K., Sadegh, N., and Theodorou, E. A., "Safety embedded differential dynamic programming using discrete barrier states," *IEEE Robotics and Automation Letters*, vol. 7, 2022, pp. 2755–2762.

[3] Azimov, D. M., "Enhanced Apollo-Class Real-Time Targeting and Guidance for Powered Descent and Precision Landing," *ResearchGate* Available: https://www.researchgate.net/publication/337705198_Enhanced_Apollo-Class_Real-Time_Targeting_and_Guidance_for_Powered_Descent_and_Precision_Landing, 15 April, 2023.

[4] Bryson, A.E. and Ho, Y.C., "Applied Optimal Control," Hemisphere Publishing Corporation, Bristol, Pennsylvania, 1975

[5] Cheek, N., Daniel, N., Lightsey, E. G., Peet, S., Smith, C., and Cavender, D., "Development of a COTS-Based Propulsion System Controller for NASA's Lunar Flashlight CubeSat Mission," *35th Annual Small Satellite Conference*, 2021.

[6] Engel, D., and Putnam, Z. R., "Control of a blunt body Mars entry vehicle with flaps using model predictive control," *AIAA SciTech 2022 Forum*, 2022.

[7] Fear, A., and Lightsey, E. G., "Implementation of small satellite autonomous rendezvous using model predictive control," *AIAA SciTech 2022 Forum*, 2022.

[8] Khalid, A., Jaffery, M. H., Javed, M. Y., Yousaf, A., Arshad, J., Ur Rehman, A., Haider, A., Althobaiti, M. M., Shafiq, M., and Hamam, H., "Performance analysis of Mars-powered descent-based landing in a constrained optimization control framework," *Energies*, vol. 14, 2021, p. 8493.

[9] Kouvaritakis, B., and Cannon, M., *Model Predictive Control - Classical, Robust and Stochastic*, Cham: Springer, 2016.

[10] Lee, A., "Fuel-efficient descent and landing guidance logic for a safe lunar touchdown," *AIAA Guidance, Navigation, and Control Conference*, 2011.

[11] "Lunar Access Services User's Guide V1.4," *Intuitive Machines* Available: https://www.intuitivemachines.com/lunar-access-services

[12] "Model Predictive Control Toolbox User's Guide ," *PDF Room* Available: https://pdfroom.com/books/model-predictive-control-toolbox-users-guide/Jr2EL8LJgyv

[13] "QP Solver - MATLAB & Simulink" Available: https://www.mathworks.com/help/mpc/ug/qp-solver.html

[14] Ramanathan, K., and Lightsey, E. G., "Investigation and Analysis into Establishing a Cislunar PNT System and

Performing a Soft Landing on the Lunar Surface," 2022.

[15] Rizvi, A., Ortega, K. F., and He, Y., "Developing Lunar Flashlight and Near-Earth Asteroid Scout Flight Software Concurrently using Open-Source F Prime Flight Software Framework," 2022.

[16] Silva, C. H., Henrique, H. M., and Oliveira-Lopes, L. C., "Experimental application of Predictive Controllers," *Journal of Control Science and Engineering*, vol. 2012, 2012, pp. 1–18.

[17] Tassa, Y., Mansard, N., and Todorov, E., "Control-limited differential dynamic programming," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[18] Violette, D. P., "Arduino/Raspberry Pi: Hobbyist Hardware and Radiation Total Dose Degradation," *NASA Electronics and Parts Packaging (NEPP)* Available: https://nepp.nasa.gov/workshops/eeesmallmissions/talks/11%20-%20THUR/1030%20-%20EEE%20Parts%20Selection%20for%20Small%20Spacecraft%20Missions.pdf

[19] "What is Model predictive control? - MATLAB & Simulink" Available: https://www.mathworks.com/help/mpc/gs/what-is-mpc.html