Lunar Lander Model Predictive Control Performance

Shrivatsan K Chari
Smead Aerospace Engineering Sciences
University of Colorado
Boulder, USA

Patrick Miga
Smead Aerospace Engineering Sciences
University of Colorado
Boulder, USA

***Abstract*—A simulation is created in Julia to apply Model Predictive Control to a lunar lander. The dynamics included in the simulation are vertical and horizontal positions and velocities, as well as a rotation of the lander. Real world tuning parameters were found and scaled down to values that would allow for faster convergence. This work successfully randomizes an initial state within the vicinity of a landing pad and accomplishes the task of landing the lunar lander safely on the pad, which works for all tested cases.**

I. Introduction

The rise in autonomous technology opens up many applications in a wide variety of industries. With much of space exploration taking place robotically, the use of autonomous technology is extremely prevalent. One area of application for this technology is in landing spacecraft, whether it is rockets or landers on another body. This paper details the use of this technology applied to a lunar lander.

II. Background and Related Work

Much work has been done in the area of using Model Predictive Control (MPC) and specifically for landing spacecraft. The most high-profile case of landing rockets using MPC is performed by SpaceX, but there are many cases of MPC being used in space applications.

The foundation of much of this work comes from Kochenderfer, Wheeler, and Wray[1]. The decision making textbook provides a section on MPC and a simple example of running MPC in Julia, which is the language used in this endeavor. This example was the baseline model for this work. One example of using MPC in powered descent is from Pascucci, Bennani, and Bemporad[2]. They first highlight that the powered descent phase for a pin-point landing is a thrust vectoring problem. Additionally, in their approach, they formulate a state space approach with dynamics, constraints, and an objective function for the MPC problem. They perform a case study on a Mars landing and show that using an MPC is effective for path following as well as landing at a desired position without a preconfigured path. Their work was helpful in providing insight on the dynamics and objective function. Jung and Bang[3] approach the problem of a Mars landing using MPC, where they use linearized dynamics for convex optimization and an objective function minimizing landing error and fuel consumption; a new cost design is used to avoid infeasible solutions based on competing objectives and it is shown that performance is near-optimal. Findeisen and Allgower[4] provide a great introduction to using nonlinear MPC, with insights into the principles and mathematical formation. Lastly, further insights were given by a paper by Liao-McPherson, Dunham, and Kolmanovsky[5], which gave a few different options of a cost function for landing on a low gravity body.

III. Problem Formulation

This problem is formulated as a lunar lander subject to some relatively simple dynamics, although some components are nonlinear. The lunar lander is composed of 6 states: lateral position, vertical position, orientation angle, lateral velocity, vertical velocity, and orientation angular velocity. These are represented by the state space $S = \begin{bmatrix} x & z & \theta & v_x & v_z & \omega \end{bmatrix}^T$.

A basic sketch of the lander can be seen in Figure 1. The lunar lander has two input external forces, a lateral force which can be positive or negative and is perpendicular to the lander from top to bottom, and a thrust which is positive that is in the direction of the lander from top to bottom. There is a slight offset from the center of gravity, which is represented by δ, that the lateral force acts on the lander. This causes rotation, which is represented in a clockwise manner by θ.

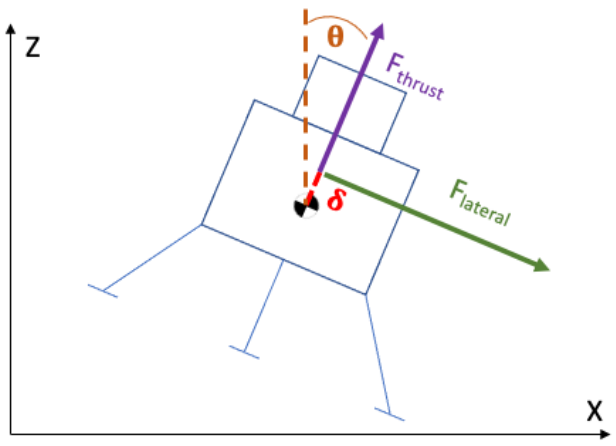

Fig. 1. An illustration of the lunar lander orientation and forces.

The lander is subject to the following dynamics, with F_{thr} and F_{lat} inputs:

$$
\begin{aligned}
F_x &= cos\theta\ F_{lat} - sin\theta\ F_{thr} \\
F_z &= cos\theta\ F_{thr} + sin\theta\ F_{lat} \\
\tau &= \delta\ F_{lat} \\
a_x &= \frac{F_x}{M} \\
a_z &= \frac{F_z}{M} \\
\dot{\omega} &= \frac{\tau}{I} \\
F_x &= cos\theta\ F_{lat} - sin\theta\ F_{thr} \\
F_z &= cos\theta\ F_{thr} + sin\theta\ F_{lat}
\end{aligned}
$$

With the following objective function and constraints:

$$
\begin{aligned}
\min_{s} \quad & S^T Q S + F^T R F \\
\text{s.t.} \quad & \dot{S} = f(s,u) \\
& z \geq 0 \\
& -\frac{\pi}{3} \leq \theta \leq \frac{\pi}{3}
\end{aligned}
$$

Where Q and R are the weights given to minimizing the states and actions, S is the state vector, and F is the vector of input forces.

IV. Solution Approach

The overall solution process is to start with an initial state, which is either random or selected, then it is run through an MPC function which optimizes over some number of steps. Once the optimization is complete and a solution is found, the MPC function outputs the next action. This action is then taken and applied to the current state by running it through the Update State Function. The Update State Function takes the state and action and calculates the next state from the dynamics and noise. This continues until the terminal state is reached. To approach the terminal state, the desired goal state is used; in this case, it is $S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$.

A. MPC Function and Constraints

The MPC works by creating variables for the states and actions over a time horizon of steps d, creating constraints which are a combination of dynamics for each state as well as general state space constraints, creating an objective function, and optimizing over the number of steps. In the case of this solution, the objective function is defined as shown in section III, and the main constraints were that the vertical position must be above 0 so the lander isn't going through the ground, and the lander could only rotate between (-60,60) degrees. The lateral forces were constrained between (-5,5) N, and the vertical force was constrained between (0,-10) N. Gravity was a constant 1.6 $\frac{m}{s^2}$.

B. Update State Function

Updating the state is done with simple Euler integration, which is $S_i = S_{i-1} + f(s,u)\Delta t$, where $f(s,u)$ represents the dynamics of the state update. For position updates, $f(s,u)$ is equivalent to the velocity, and for velocity updates, $f(s,u)$ is the forces. In the Euler integration of velocity updates, there is an additional random perturbation ϵ to introduce Gaussian noise. The following equations dictate these updates:

$$
\begin{aligned}
v_x' &= v_x + a_x \Delta t + \epsilon \\
v_z' &= v_z + (a_z - g)\Delta t + \epsilon \\
\omega' &= \omega + \dot{\omega}\Delta t + \epsilon \\
x' &= x + v_x \Delta t \\
z' &= z + v_z \Delta t \\
\theta' &= \theta + \omega \Delta t
\end{aligned}
$$

Lastly, the angle wrapping is done in the update state, where if θ reaches -2π or 2π it becomes 0. A timestep $\Delta t = 0.25$ seconds was chosen for the simulations.

C. Terminal State

The overall program loops on the condition that the terminal state has not yet been reached. On each iteration, the MPC is called, which propagates the trajectory and selects the best action. This action is passed to update the state one step. This repeats until the terminal condition. In this case, the terminal condition is a horizontal displacement of 15 meters of the center of the landing pad, a vertical position between 0 and 3 meters, and a rotation less than 0.25 radians. If this terminal condition is reached, the program concludes. It is observed that if the positions are near their terminal conditions, the velocities are near zero, therefore there is no need for velocity terminal conditions.

D. Tuning Parameters

Many parameters in this problem had to be tuned. Without values that are pretty accurate to real world applications, the solution is infeasible. Real world working values were found and then scaled down to small numbers for simplification and faster run times. The following table shows these parameters and their working values after iterating through different tuning levels.

Table I. Tuning values for a correct simulation.

Parameter	Value	Unit
Q	500*$[I_{6x6}]$	-
R	$[I_{2x2}]$	-
M	1	kg
I	1	kg m^2
δ	1	m
F_{lat}	$\pm$5	N
F_{thr}	10	N
g	1.6	$\frac{m}{s^2}$
σ	1	-
Δt	0.25	sec

E. Optimization

To perform this simulation, the JuMP package was used in Julia. JuMP is a modeling language that supports mathematical optimization. There are 4 main components of using JuMP: defininining the model (which is where the optimizer is selected), defining variables (states and actions), defining constraints (force limits, position limits, attitude limits as well as state updates and the initial condition), and the objective function. Once these are all defined and the number of steps are selected, the optimize! function can be run. Constraints and the objective function can be defined as nonlinear. The return from this function is the actions f_{lat} and f_{thr} to take at the current step. The optimizer used in this simulation is Ipopt, which is a nonlinear solver typically used with JuMP optimization problems.

V. Results

To show that the code works for many initial conditions, the state was randomized within the following range of values: $S = \begin{bmatrix}\pm 50 & (0, 100) & \pm 1.5 & \pm 20 & \pm 20 & \pm 0.5\end{bmatrix}^T$. Many cases were run and some are shown below; case 3 does a good job of showing behavior, and will be discussed first. Therefore, initial state is $S_3 = \begin{bmatrix}38.77 & 76.74 & 0.695 & -9.8 & 5.92 & -0.392\end{bmatrix}^T$. The following plots show the trajectory over the course of the simulation for case 3.

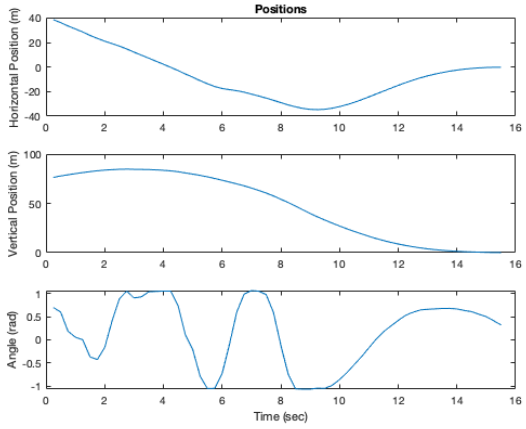

Fig. 2. For case 3, the position states are shown over time.

As can be seen in the figure, the lander starts at a positive horizontal position around 40 meters, then moves in a negative horizontal motion over the pad, and then comes back to the pad. Meanwhile, it starts at around 75 meters in the vertical direction, rises up a little bit, and then comes back down to the pad. This behavior is understood by looking at the angular plot, which shows some oscillations between the angle limits while the horizontal and vertical positions approach the goal.

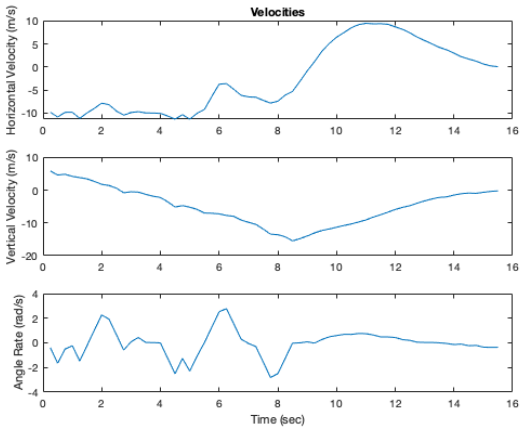

Fig. 3. For case 3, the velocity states are shown over time.

This is clearly exhibited in the velocity plots, where there is a negative horizontal velocity which begins to switch to positive around the same spot as the lander begins to decelerate once it gets closer to the ground. The angular rates have smaller oscillations but there are a couple points where there are large spikes, which look like they may correspond with horizontal velocity changes while the vertical velocity is negative.

Additionally, an animation was created to display the behavior of the lander. To show the progression of the states over the entire time horizon, a multi-exposure photograph was created, as shown in figure 4 below. The pointed tip of the shape in the figure is the top of the lander.

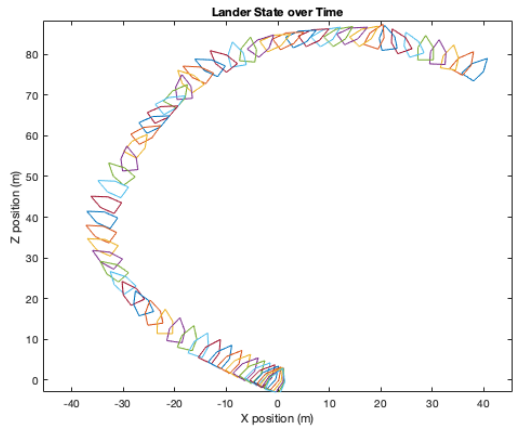

Fig. 4. A multi-exposure photograph of the lander over time in position space for case 3.

As can be seen, the multi-exposure photograph shows a better understanding of the lander behavior. There is clearly some wobbling in the first half of the trajectory as the lander attempts to steady, but then there isn't as much wobbling once the vertical velocity starts increasing, which corresponds to the point where there is seemingly more stability.

Another random case was run, this time to additionally display forces and some of the more interesting behavior

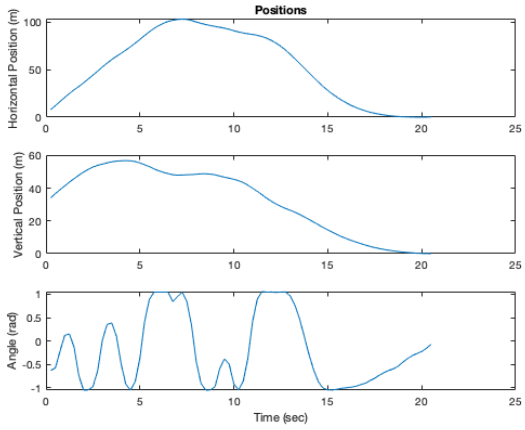

Fig. 5. For case 7, the position states are shown over time.

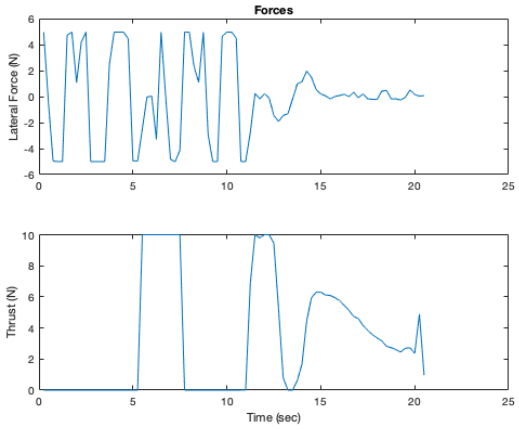

Fig. 7. For case 7, the forces are shown over time.

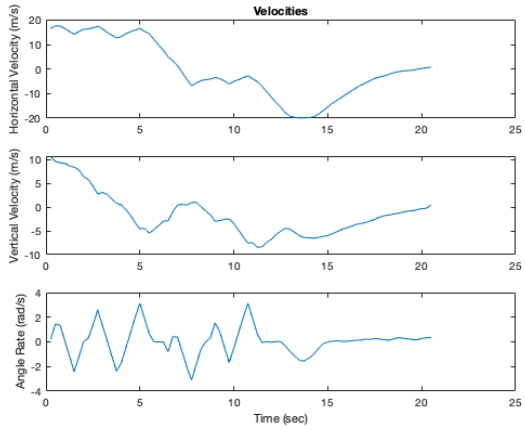

Fig. 6. For case 7, the velocity states are shown over time.

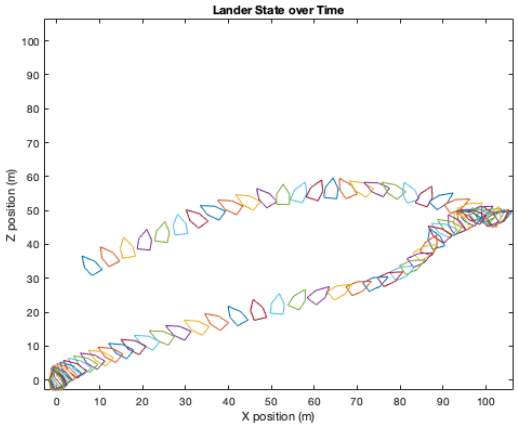

Fig. 8. A multi-exposure photograph of the lander over time in position space for case 7.

that happens in some cases. The initial conditions are $S_7 = \begin{bmatrix} 7.63 & 34.12 & -0.627 & 16.56 & 10.57 & 0.206 \end{bmatrix}^T$. The same figures are displayed consecutively in Figures 5-8, including a figure displaying the forces.

In this case, the lander starts just above the pad, but is quickly moving away in the horizontal and vertical directions. Once it begins to descend, it quickly rights itself with a dramatic turn and then it thrusts hard to stop moving horizontally away from the pad, and then once it rotates again, it thrusts hard to move back towards the pad. The MPC utilizes the lateral force quite a bit to constantly rotate and right itself, but the thrust force is used sparsely.

VI. Conclusion

MPC was used to successfully bring a moving and rotating lunar lander to rest on a landing pad. In section II, background and other work that motivated and inspired this project is introduced. The problem is defined in section III. The method for solving the problem is described in section IV. Finally, results are shown in section V. The lander displays interesting behavior in some of its cases, but always successfully lands on the pad while staying within its constraints.

For future work, one major contribution would be to run the simulation using a GPU and see if there is a computational advantage for optimization problems. Additionally, another expansion on the current simulation would be to add a changing mass and minimize the amount of fuel used, which would need the inclusion of the rocket equation and many other more considerations. However, if this was added, then the problem could be much more applicable to real situations. Finally, expanding the state space to 3 dimensions would be the ultimate culmination of this project.

References

[1] Mykel Kochenderfer, Tim Wheeler, and Kyle Wray. "Algorithms for Decision Making." MIT Press, 2022.

[2] Carlo Pascucci, Samir Bennani, and Alberto Bemporad. "Model Predictive Control for Powered Descent Guidance and Control." European Control Conference (ECC), Linz, Austria July 15-17 2015.

[3] Jung, Youeyun, and Hyochoong Bang. "Mars Precision Landing Guidance Based on Model Predictive Control Approach." Proceedings of the Institution of Mechanical Engineers, Part G: Journal of

Aerospace Engineering, vol. 230, no. 11, Sept. 2016, pp. 2048–2062, doi:10.1177/0954410015607893.

[4] Rolf Findeisen, Frank Allower. ”An Introduction to Nonlinear Model Predictive Control.” 21st Benelux Meeting on Systems and Control, Technische Universiteit Eindhove, Eindhoven, 2002.

[5] Dominic Liao-McPherson, William Dunham, and Ilya Kolmanovsky. ”Model Predictive Control Strategies for Constrained Soft Landing on an Asteroid.” AIAA SPACE Forum, Long Beach, California, September 13-16 2016.

VII. Code Appendix

A. MPC Julia Code

```
using JuMP
using Ipopt
using POMDPs
using POMDPModelTools
using Distributions
using Random
using LinearAlgebra
using Plots
using XLSX

struct LunarLander <: POMDP{Vector{Float64}, Vector{Float64}, Vector{Float64}}
    dt::Float64
    m::Float64 # 1's kg
    I::Float64 # 1's kg*m^2
    Q::Matrix{Float64}
    R::Matrix{Float64}
    g::Float64
end

function LunarLander(;dt::Float64=0.25, m::Float64=1.0, I::Float64=1.0)
    Q = diagm([0.0, 0.0, 0.1, 0.1])
    R = diagm([1.0, 0.01])
    g = 1.6
    return LunarLander(dt, m, I, Q, R, g)
end

struct LanderActionSpace
    min_lateral::Float64
    max_lateral::Float64
    max_thrust::Float64
    max_offset::Float64
    function LanderActionSpace()
        new(-5.0, 5.0, 15.0, 1.0)
    end
end


function update_state(lander::LunarLander, s::Vector{Float64}, f::Vector{Float64};
    rng::AbstractRNG=Random.GLOBAL_RNG, sigma::Float64=1.0)

    x = s[1] # lander's postion
    z = s[2] # lander's altitude
    theta = s[3] #positive clockwise, measured from vertical 12 o clock position
    vx = s[4] # lateral velocity
    vz = s[5] # vertical veolcity
    omega = s[6] # angular rate about cg

    f_lateral = f[1] # lateral thrusters
    thrust = f[2] # vertical thrusters
    delta = f[3]

    fx = cos(theta)*f_lateral - sin(theta)*thrust  # components of force in x and z direction
    fz = cos(theta)*thrust + sin(theta)*f_lateral
    torque = delta*f_lateral

    ax = fx/lander.m
    az = fz/lander.m
    omegadot = torque/lander.I

    epsilon = randn(rng, 3)*sigma
    vxp = vx + ax*lander.dt + epsilon[1]*0.1
    vzp = vz + (az - lander.g)*lander.dt + epsilon[2]*0.1
    omegap = omega + omegadot*lander.dt + epsilon[3]*0.01
```

```
    xp = x + vx*lander.dt # Euler integration
    zp = z + vz*lander.dt
    thetap = theta + omega*lander.dt

    sp = [xp, zp, thetap, vxp, vzp, omegap]

    if thetap > 2*3.14159265 # wrap the angle
        thetap = thetap - 2*3.14159265
    elseif thetap < -2*3.14159265
        thetap = thetap + 2*3.14159265
    end

    return sp
end

function isterm(::LunarLander, s::Vector{Float64})
    x = s[1]
    z = s[2]
    delta = abs(x)
    theta = abs(s[3])

    if z <= 1.0 && z >= 0 && delta <= 5.0 && theta <= 0.15
        return true
    else
        return false
    end
end

function MPC(cs,lander, act_limits)

    goal = [0,0,0,0,0,0] # Goal state

    model = Model(Ipopt.Optimizer)
    d = 20 # horizon length

    @variables model begin
        s[1:6, 1:d] # 6 for states
        f[1:2,1:d] # we command lateral and vertical thrusters
    end
    @constraint(model, [i=1:d], -5 <= f[1,i] <= 5) # Limit for lateral thrusters
    @constraint(model, [i=1:d], 0 <= f[2,i] <= 10) # Limit for vertical thrusters

    @constraint(model, [i=2:d], s[2,i] >= 0) # Altitude constraint (don't go digging!)
    @constraint(model, [i=2:d], 3.14159/3 >= s[3,i] >= -3.14159/3) # Angle constraint

    # velocity update
    # vx
    @NLconstraint(model, [i=2:d,j=3], s[1+j,i] == s[1+j, i-1] +
    (lander.dt/lander.m)*(cos(s[3, i-1])*f[1,i-1] - sin(s[3, i-1])*f[2,i-1]))
    # vz
    @NLconstraint(model, [i=2:d,j=4], s[1+j,i] == s[1+j, i-1] +
    (lander.dt/lander.m)*(cos(s[3, i-1])*f[2,i-1] + sin(s[3, i-1])*f[1,i-1]) - lander.g*lander.dt)
    #
    @constraint(model, [i=2:d,j=5], s[1+j,i] == s[1+j, i-1] +
    ( (act_limits.max_offset*f[1,i-1])/lander.I )*lander.dt)

    # position update
    # x
    @constraint(model, [i=2:d,j=0], s[1+j,i] == s[1+j, i-1] + lander.dt*s[4+j,i-1])
    # z
    @constraint(model, [i=2:d,j=1], s[1+j,i] == s[1+j, i-1] + lander.dt*s[4+j,i-1])
    #
    @constraint(model, [i=2:d,j=2], s[1+j,i] == s[1+j, i-1] + lander.dt*s[4+j,i-1])

    # initial condition
    @constraint(model, s[1:6,1] .== cs) # cs: current state
```

```
    # Cost function
    @objective(model, Min, 500*sum((s[1:6,d] - goal).^2) + sum(f.^2))

    optimize!(model)

    return JuMP.value.(f[1:2,1])
end


function mpc_simulation(start, lander, act_limits)
    s = start
    global hist = []
    global hist2 = []
    global hist3 = []
    global hist4 = []
    global hist5 = []
    global hist6 = []
    while(!isterm(lander,s))
       append!(hist,s[1])
       append!(hist2,s[2])
       append!(hist3,s[3])
       append!(hist4,s[4])
       append!(hist5,s[5])
       append!(hist6,s[6])

       f = MPC(s,lander, act_limits) # Call MPC for optimal action
       f = vcat(f,act_limits.max_offset) # concat offset to make f the right size

       sp = update_state(lander,s,f) # Get next state
       s = sp

    end
    #=
    XLSX.openxlsx("histories.xlsx", mode="w") do xf
        sheet = xf[1]
        sheet["A1"] = hist
        sheet["A2"] = hist2
        sheet["A3"] = hist3
        sheet["A4"] = hist4
        sheet["A5"] = hist5
        sheet["A6"] = hist6
    end
    =#
end

lander = LunarLander()
act_limits = LanderActionSpace()
#start = 20*rand(6)
start = [100*(0.5 - rand(1)[1]), 100*rand(1)[1], 3*(0.5 - rand(1)[1]),
40*(0.5 - rand(1)[1]), 40*(0.5 - rand(1)[1]), 0.5 - rand(1)[1]]

#start = [-20, 20, 0, -15, 15, 0.0]
mpc_simulation(start, lander, act_limits)
```

B. MATLAB Plotting Code

```
function animateSC

[num,~,~] = xlsread('histories.xlsx');
timevec = 1:length(num);
timevec = timevec/4;
figure
subplot(3,1,1)
plot(timevec,num(1,:))
title('Positions')
ylabel('Horizontal Position (m)')
```

```
subplot(3,1,2)
plot(timevec,num(2,:))
ylabel('Vertical Position (m)')
subplot(3,1,3)
plot(timevec,num(3,:))
xlabel('Time (sec)')
ylabel('Angle (rad)')
figure
subplot(3,1,1)
plot(timevec,num(4,:))
title('Velocities')
ylabel('Horizontal Velocity')
subplot(3,1,2)
plot(timevec,num(5,:))
ylabel('Vertical Velocity')
subplot(3,1,3)
plot(timevec,num(6,:))
xlabel('Time (sec)')
ylabel('Angle Rate (rad/s)')
figure
subplot(2,1,1)
plot(timevec,num(7,:))
title('Forces')
ylabel('Lateral Force (N)')
subplot(2,1,2)
plot(timevec,num(8,:))
xlabel('Time (sec)')
ylabel('Thrust (N)')
 xlimmax = max(num(1,:));
 xlimmin = min(num(1,:));
 zlimmax = max(num(2,:));
 bound = max([abs(xlimmax) abs(xlimmin) abs(zlimmax)]);
% zlimmin = min(num(2,:));
animation = false;
if animation
    h = figure;
    filename = 'Lander.gif';
else
    g = figure;
    filename = 'Lander_lapse.png';
end
for i = 1:length(num)
    mid = num(1,i); mid2 = num(2,i); rot = num(3,i);
    rotM = [cos(rot) sin(rot); -sin(rot) cos(rot)];
    increase = 3;
    shape1 = [-0.5 0.5 0.5 0 -0.5 -0.5]*increase;
    shape2 = [-1 -1 0 1 0 -1]*increase;

    outvec = [];
    for j = 1:length(shape1)
        tempvec = [shape1(j); shape2(j)];
        tempvec = rotM*tempvec;
        outvec = [outvec tempvec];
    end
    shape1 = outvec(1,:); shape2 = outvec(2,:);
    plot(shape1 + mid, shape2 + mid2)
```

```
    if bound == zlimmax
        xlim([-bound/2-increase bound/2+increase])
        ylim([0-increase bound+increase])
    elseif bound == abs(xlimmin) && xlimmax < 1
        xlim([-bound-increase increase])
        ylim([0-increase bound+increase])
    elseif bound == abs(xlimmax) && xlimmin > -1
        xlim([-increase bound+increase])
        ylim([0-increase bound+increase])
    else
        xlim([-bound-increase bound+increase])
        ylim([0-increase bound*2+increase])
    end

    if animation
        pause(0.25)
        frame = getframe(h);
        im = frame2im(frame);
        [imind,cm] = rgb2ind(im,256);
        if i == 1
            imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
        else
            imwrite(imind,cm,filename,'gif','WriteMode','append');
        end
    else
        hold on
        title('Lander State over Time')
        xlabel('X position (m)')
        ylabel('Z position (m)')
    end
end

end
```