2015

Fuel-Optimal Trajectory Planning of a VTOL Spacecraft Using SQP and RRT

Ian Miller
Lehigh University

Fuel-Optimal Trajectory Planning of a VTOL Spacecraft Using SQP and RRT

by

Ian Miller

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of Master of Science

in

Electrical Engineering

Lehigh University

December 2015

Certificate of Approval

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

_____

Date

_____

Terry Hart, Thesis Advisor

_____

William Best, Co-Advisor

_____

Filbert J. Bartoli, Chairperson of Department

Acknowledgements

I would like to thank Professor Terry Hart for providing me with the opportunity to pursue this project for a Master's thesis. His expertise and guidance allowed me to even take the first steps in this project. His aid in the necessary background topics of this topic was critical to my understanding of the material and the field at large.

I would also like to thank Professor William Best for his assistance and guidance for the last four years. His unwavering support for all that I did through my undergraduate and graduate career has truly made a difference in my education and development into the engineer I am today.

I would like to thank my friends and family who pushed me to strive for greatness and to achieve. Without your encouragement I would never have been able to accomplish such great things.

Table of Contents

List of Tables

List of Figures

Abstract

While rovers have traditionally been used to explore extraterrestrial bodies, they reduce the total area explored on the ground as they are limited by traversing the surface. For this reason vertical takeoff and landing crafts are explored. The major downfall of this type of craft for exploration is the extra fuel costs which must be carried into orbit. Reducing the fuel burn for a given maneuver allows the mission to either bring less propellant or to explore further. In either case, it is highly advantageous to reach destination points with the least amount of fuel. This paper looks at fuel-optimal trajectory planning for these reasons. A combination of optimal control theory with sequential quadratic programming and rapidly exploring random trees is proposed to achieve a robust, real time optimal trajectory.

Chapter 1: Background

1.1 Introduction

Trajectory or path planning has applications in a wide variety of fields and is the topic of much work to date. If it is desired to not only produce a viable trajectory, but also one that optimizes some cost function (such as time, distance traveled or energy used) optimal control theory gives an excellent foundation. Unfortunately, most problems of practical use are sufficiently complex such that solving analytically for an optimal solution is not possible. Instead a numerical approach such as nonlinear optimization must be utilized [1]. This process can be computationally intensive and therefore difficult to implement in real time for higher order systems.

Sampling based planning algorithms such as the probabilistic road map (PRM) [2] and rapidly exploring random trees (RRT) [3] produce feasible trajectories for potentially high order systems in short time. These methods trade optimality for time to run. Fortunately, there are many methods to increase the performance of such algorithms while maintaining their speed.

While rovers have traditionally been used to explore extraterrestrial bodies, they reduce the total area explored on the ground as they are limited by traversing the surface. For this reason vertical takeoff and landing crafts are explored. For this reason vertical takeoff and landing crafts are explored. The major downfall of this type of craft for exploration is the extra fuel costs which must be carried into orbit. Reducing the fuel burn

for a given maneuver allows the mission to either bring less propellant or to explore further.

1.2 Assumptions, State Equations and Basic Derivations

A generalized VTOL spacecraft may be viewed as a having three degrees of freedom (DOF) in translation and three in rotation. Additionally, as the craft will have mass, rate for each DOF would need to be included. Fortunately, we may look at the system in two dimensions without loss of generality. The x axis may be oriented such that it produces a line between starting location and the goal position and may be seen in figure one. This removes the need to concern ourselves with yaw and motion in the z axis. Additionally, rate of change in roll of the spacecraft is assumed to be zero and therefore is neglected. These simplifications leave three dimensions for control and optimization.

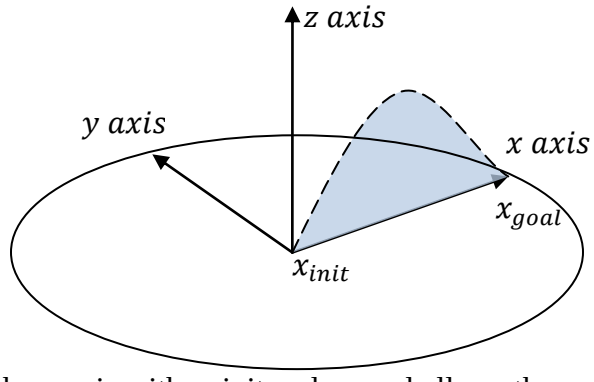

Figure 1: Aligning the x axis with x_init and x_goal allows the y axis to be neglected. The circle represents all of the possible landing locations given the same trajectory and a reorientation of the coordinate system

For the control of the lander craft, it is assumed that acceleration will be applied in line with the center of gravity of the craft. This thrust can be applied at any arbitrary level between an upper and lower bound. The final control input is the angle at which the thrust is vectored. This input is also bounded between some maximum and minimum angle.

Initially, the trajectories considered are close to the surface of a celestial body absent of an atmosphere or one with a sufficiently thin atmosphere to be neglected such as the moon. The system has a five dimensional state space x of horizontal position ($x_1(t)$), vertical position ($x_2(t)$), horizontal velocity ($x_3(t)$), vertical velocity ($x_4(t)$) and mass ($x_5(t)$). This vector may be seen formally in Eq 1.

$$\boldsymbol{x} = \begin{matrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \end{matrix}$$

Equation 1.

Figure 2 illustrates the forces being applied to the vehicle close to the terrestrial body's surface.

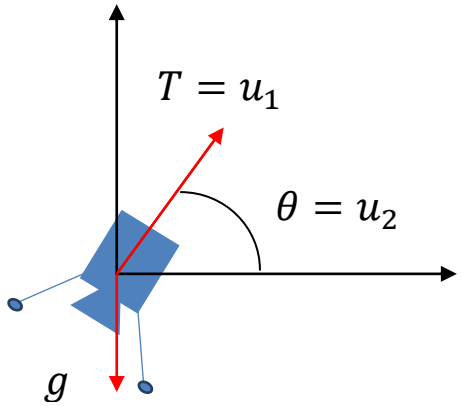

Figure 2: Forces applied to the craft

1.3.1 Ideal Trajectory

Regarding fuel burn, a slow decent is relatively inefficient [7]. An easy way to think of this is to consider the lander as it descends. Taken to an extreme, the lander would eventually be hovering in place burning fuel to increase the time of flight. Taken to the other extreme, the craft could free fall and just before impact with the surface apply an impulse to negate its velocity. This impulsive maneuver would result in the fuel optimal trajectory [7].

Unfortunately, real system constraints do not allow for the infinite thrust such a maneuver would require [7]. Fortunately other methods of solving while considering the system constraints exist and will be explored later in this paper. Even though this solution is realistically infeasible, the output can serve as a lower bound to compare solutions against. Additionally, it can provide useful initial guesses for several of the proposed algorithms.

1.3.2 Impulsive Burn-Coast-Burn

The ideal impulsive maneuver is comprised of three phases. The first stage is an impulsive maneuver where the change in velocity may be called ΔV_{init}. The final velocity will be called v_{init}. The second stage is simply coasting to the destination. The final stage is an impulsive maneuver fired in the opposite direction of the craft to negate v_{init}. This change in velocity will be called ΔV_f. Figure 3 illustrates these three stages.

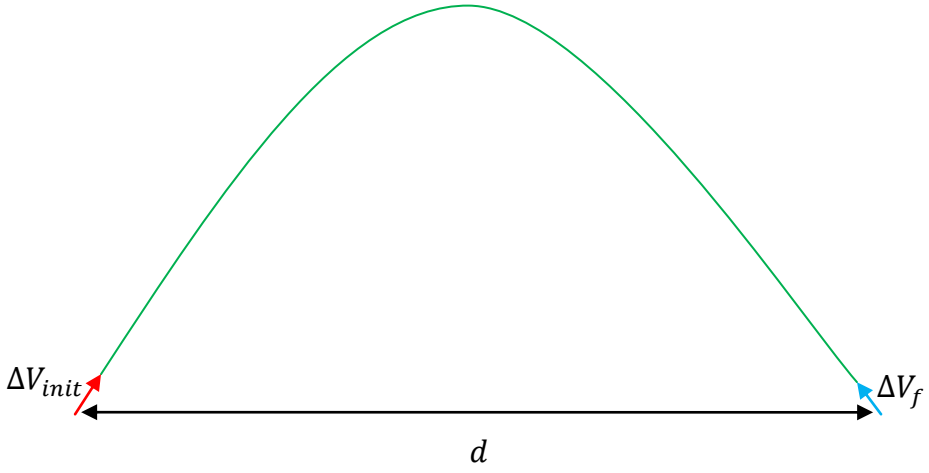

Figure 3: Initial burn is shown in red, coast in green and final burn in blue

When the system constraints are ignored and the rocket is able to produce impulses, the system may be analyzed with basic equations of ballistic motion [7]. If the altitude of x_{init} and x_{goal} are equal, then the distance traveled by the craft, d, is defined as [10]:

$$d = \frac{v^2 \sin(2\theta)}{2}$$

Equation 2.

From equation 2 it is easily seen that $\theta = \frac{\pi}{4}$ would yield the greatest distance traveled. Under these assumptions, the time of flight is defined as [10]:

$$T_{of} = \frac{\sqrt{2}}{g} v$$

Equation 3.

Solving equation 2 for v and subbing the solution into equation 3 yields the following equation for time of flight:

$$T_{of} = \frac{\sqrt{2dg}}{g}$$

Equation 4.

Where g is the acceleration due to gravity near the surface of the terrestrial body.

1.4 Thrust and Mass Flow

For the craft to develop any change in velocity, it will require some thrust. Thrust may be described as:

$$T = -I_{isp} g_e \frac{dm}{dt}$$

Equation 5.

where I_{sp}[1] is the specific impulse of the chosen rocket, g_E is the gravity on the surface of the earth and m is the mass of the vessel [8]. As the total amount of thrust used is reliant upon the mass flow, it creates an excellent metric to compare different trajectories. Mass flow may be described by the following:

$$\Delta m = -\frac{T}{I_{sp} g_E} \Delta t$$

Equation 6.

Where Δm is the mass flow and Δt is the change in time.

[1] I_{sp} is defined as the thrust per sea level weight rate per second of propellant consumption. Otherwise written as $I_{sp} = \frac{T}{\dot{m} g_E}$

1.5 Dynamic Equations

Based on the definition of the states put forth in section 1.2 and in equation one may define the system dynamics in the following way:

$$\dot{\boldsymbol{x}} = \begin{matrix} x_3 \\ x_4 \\ \dfrac{\cos(\theta) * \mathrm{T}}{x_5} \\ \dfrac{\sin(\theta) * \mathrm{T}}{x_5} - g \\ \Delta \mathrm{m} \end{matrix}$$

Equation 7.

where T as defined in section 1.2 and is described by equation 5. θ is as defined in section 1.2. The input vector u is comprised of thrust, u_1, and the angle at which the rocket is directed, u_2. This means equation 7 may be rewritten as

$$\dot{\boldsymbol{x}} = \begin{matrix} x_3 \\ x_4 \\ \dfrac{\cos(\mathrm{u}_2) * \mathrm{u}_1}{x_5} \\ \dfrac{\sin(\mathrm{u}_2) * \mathrm{u}_1}{x_5} - g \\ \dfrac{\mathrm{u}_1}{I_{sp} g_E} \end{matrix}$$

Equation 8.

When the craft is farther from the surface relative to the radius of the body, a uniform distribution of gravity can no longer be assumed. In this case the model must be changed to accommodate. However, as none of the trajectories take the vessel beyond 1% of the radius of the terrestrial body, the uniform assumption remains valid.

Chapter 2: Optimal Control Theory

2.1 Trajectory Planning with Optimal Controls

Finding the set of control inputs which brings the system from an initial state to a desired goal state is equivalent to solving for the trajectory itself [1]. This statement can be made as the state equations may be integrated forward to produce the trajectory. For this reason, optimal controls are explored.

Given a system, a control input is desired that takes the system from an initial state, $x_{initial}$, at an initial time of t_0 to a goal state, x_{goal}, at some teminal time t_f where $t_f > t_0$. Additionally, it is desired that the input is selected such that it minimizes some cost function J. In the example provided, we let the cost function be a function of the mass flow, or put differently the fuel burnt, in our maneuver. The cost function may be seen in equation 3.

$$J = \int_0^{t_f} -x_5 = \int_0^{t_f} -m$$

Equation 3.

Only for simple systems is it possible to solve analytically for the optimal control strategy. Because of this, some numerical method is required to solve the system. One such method is nonlinear optimization.

2.2 Time as a Free Variable

Termination time can be fixed ahead of optimization. However, in many optimal controls problems time is either the variable to be minimized or does not matter as long as it is less than ∞. In these cases, a transformation of time from $[t_0\ t_f]$ to [0 1] is desirable. After the transformation, termination time may be left as a free variable to be optimized.

2.2 Nonlinear Optimization

Under this method the control inputs are parameterized and the problem is discretized with n points [3]. Thus any continuous constraints will be replaced with $n + 1$ discrete constraints spaced out by $\Delta t = \frac{t_f}{k}$. After this the problem may be written as:

Find	X
Minimizing	$J(x)$
Subject to	$u_{min}(X, k\Delta t) \leq u(X, k\Delta t) \leq u_{max}(X, k\Delta t)$
	$x(X, 0) = x_{inital}$
	$x(X, t_f) = x_{final}$

Once the problem is written in this form, a gradient based nonlinear optimization problem (NLP) solving algorithm, such as sequential quadratic programming (SQP), may be used. SQP requires an initial guess for each discretized point. This guess can vastly effect the time to convergence and even the final solution. As these methods are gradient methods by nature, they are subject to finding locally optimal solutions. Fortunately in the

particular problem proposed, producing a good first guess is easy enough. For a more complex system, a different global solver may be used first to seed the SQP[1].

The SQP solves a NLP problem iteratively by recursively solving a quadratic programming (QP) problem. At the termination of each iteration, the estimate for the solution is improved by taking a step in the direction of the solution of the QP problem. Many implementations for this technique exist but this paper used MATLAB Optimization Toolbox's FMINCON.

Chapter 3: Rapidly Exploring Random Trees

3.1 RRT for Trajectory Planning

An alternative to the classic optimal control solution can be found in sample based planning such as rapidly-exploring random trees (RRT). These methods sacrifice optimality for time of calculation. In fact, it has been shown that the RRT cannot reach an optimal solution [4]. However, variations of the algorithm such as RRT* asymptotically approach an optimal solution [4].

The goal of the algorithm is to find the control input u, which creates a feasible path from the initial state to the goal state while obeying the dynamics of the system. Additionally it is desired that the resulting trajectory minimize the cost function. To do so, RRT and RRT* develop a tree, T, comprised of a set of states V and a set of edges linking the states E.

3.2 RRT Algorithm

The main components of the algorithm and its pseudo-code are shown below:

Sample: the function *Sample* returns random state samples from the free configuration space, C_{free}.

Steer: provided two input states x and x_{new}, the function *Steer* creates a path which connects the two states. In case of a holonomic system this is typically a straight line. Otherwise, this function must obey the system dynamics.

Nearest Neighbor: provided a point, x and a tree, $T = (V, E)$, the function *Nearest Neighbor* returns a vertex that is closest to the point given some distance function. For the purposes of this paper, we may assume the distance to be calculated with the Euclidian distance.

Algorithm 1: $Build\ RRT$

1. $V \leftarrow x_{init};\ E \leftarrow \emptyset;$
2. **for** i=1 to n **do**
3. $\quad T \leftarrow (V, E);$
4. $\quad x_{rand} \leftarrow Sample(i);\ i \leftarrow i + 1;$
5. $\quad Extend(T, x_{rand})$
6. **end for**
7. **return *T***

where $Extend(T, x)$ is defined as seen in algorithm 2:

Algorithm 2: $Extend(T, x)$

1. $x_{near} \leftarrow Nearest\ Neighbor(T, x)$
2. $x_{new} \leftarrow\ Steer(x_{near}, x)$
3. **if** $ObstacleFree(x_{near}, x_{new})$ **then**
4. $\quad V \leftarrow V \cup x_{new}$
5. $\quad E \leftarrow E \cup (x_{near}, x_{new})$
6. **end if**
7. **return T**

The RRT algorithm begins by initializing its tree at the initial state, x_{init}. The algorithm then samples the free configuration space, C_{free}. C_{free} may be defined as the

set of all configruations the craft may move through without colliding with obstacles. The RRT then finds the closest vertex in the current tree and attempts to connect it via the *Steer* function. Typically a collision check is done to verify that no obstacles are present along the path between x_{near} and x_{new}. If no collision is detected, the point becomes part of the tree by adding its vertex and edge to T. In the instance of a collision, the point will not be added to the tree. One step of this process is shown in figure four.

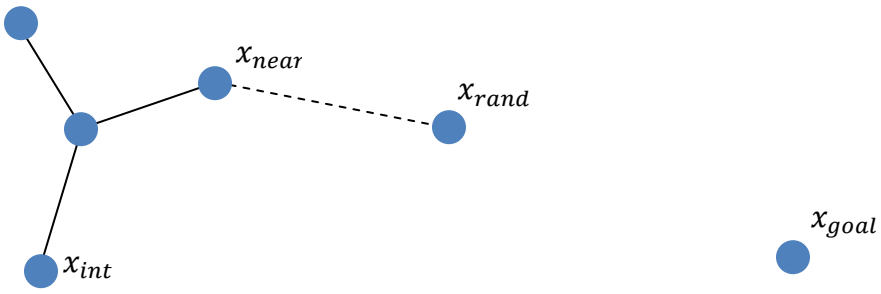

Figure 4: One step of the RRT visualized

This process is continued iteratively until the goal state is found or the maximum number of iterations, n, have been completed. The samples quickly search the space, providing a feasible trajectory very quickly, albeit a sub-optimal one [4]. An example of an RRT in action can be seen in figure 5. This figure shows just the first 100 samples of a simple, holonomic search.

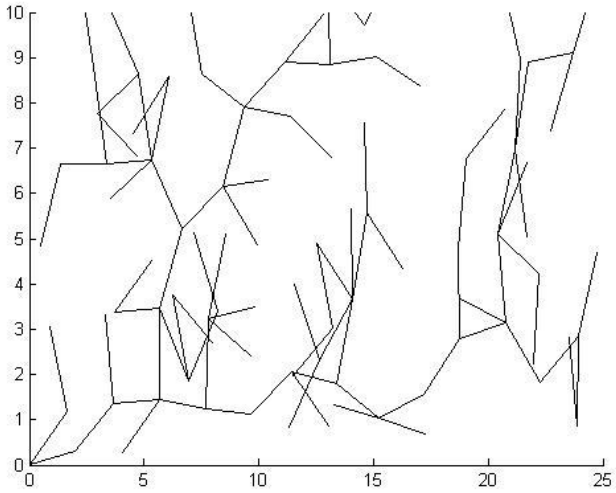

Figure 5: First 100 samples of a simple RRT implementation. Only the x and y axes are shown. x_{int} is [0,0,0,0].

The RRT is probabilistically complete, meaning as the number of samples approaches ∞ the probability of finding a fiesable path if one exists approaches one and will come arbitrarily close to the goal state [5]. In practice, one would like for the planner to return a feasible path quickly. To speed convergence to the goal state, one may augment the function *Sample* by assigning some probability, p, to which the function returns the goal state.

This greedy sampling can allow the RRT to converge to a solution much faster. However, if p is increased too high, the RRT will run the risk of getting stuck in local minima [1-2][5].

3.3 RRT* Algorithm

While the RRT is probabilistically complete, the algorithm will return a sub-optimal trajectory even as the number of samples approaches ∞. Because of this characteristic of the basic RRT, RRT* (pronounced RRT star) was developed and introduced by Karaman and Frazzoli in [4]. It was shown to be asymptotically optimal [4].

RRT* differs only slightly from the original algorithm. The RRT* will rewire the tree to reduce cost on each iteration. After sampling, it reviews all vertices in a ball defined by some distance and discerns if each of vertices would have a lower cost being routed through the newly sampled point. If a vertex does cost less being routed through the new point, the tree is reconfigured to reflect the change.

The rewiring of the RRT* effectively removes the odd zigzags and loops that can be present due to the random sampling in the original RRT.

Chapter 4: Simulation Results

4.1 Proposed Implementation

Much work with planetary landers has been done involving NLP techniques. These methods have the benefits of many years of research and plenty of actually flown trajectories to give the methods validity. Unfortunately, when it comes to space trajectories, RRT does not have near the same background. However, the RRT does an excellent job of searching a high dimensional configuration space in a short amount of time.

The proposed implementation solves with a SQP while the vessel is on the ground. While on the ground, the vessel is not burning any fuel or having to worry about collisions so calculation time is no longer an extremely important design consideration. Once in flight, the navigation loop runs the RRT* algorithm with a three second update cycle.

If no new obstacles or changes in the goal state have been presented, the navigation loop continues along the previously calculated trajectory. If something does arise, the RRT* computes a new trajectory in flight.

4.2 Input Parameters

The following chart serves as a guide to what input values were used for the presented results unless otherwise noted:

Parameter	Abbreviation	Value
Translation Length	$d = x_{goal} - x_{init}$	500m
Initial Mass	m_{init}	100kg
Maximum Thrust	T_{max}	$10 * m_{init}$
Minimum Thrust	T_{min}	0 N
Maximum Theta	θ_{max}	$5 * \frac{\pi}{6}$
Minimum Theta	θ_{min}	$\frac{\pi}{6}$
Gravity	g	$1.662 \frac{m}{s^2}$

Table 1.

The following table contains the input "option" values for the MATLAB's NLP SQP implementation, *fmincon*:

Parameter	Value
`'Algorithm'`	`'sqp'`
`'TolX'`	`1e-5`
`'TolFun'`	`1e-9`
`'TolCon'`	`1e-9`
`'MaxFunEvals'`	`2e6`
`'MaxIter'`	`10000`
`'DiffMinChange'`	`1e-9`

Table 2.

4.3 Simulation Results

Using the input parameters as seen in figures 1 and 2, the following optimal trajectory shown in figure 6 was found. This trajectory can be seen to have a similar three stage burn-coast-burn maneuver topology as discussed in section 1.3.2.

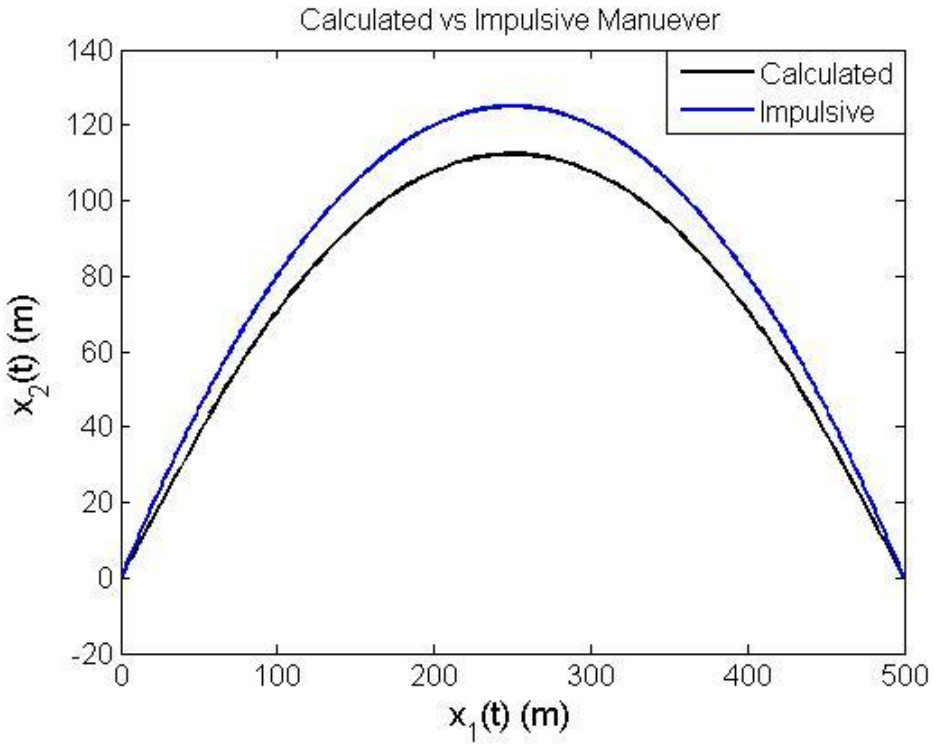

Figure 6: Vertical and Horizontal position of optimal trajectory compared against the optimal, impulsive maneuver.

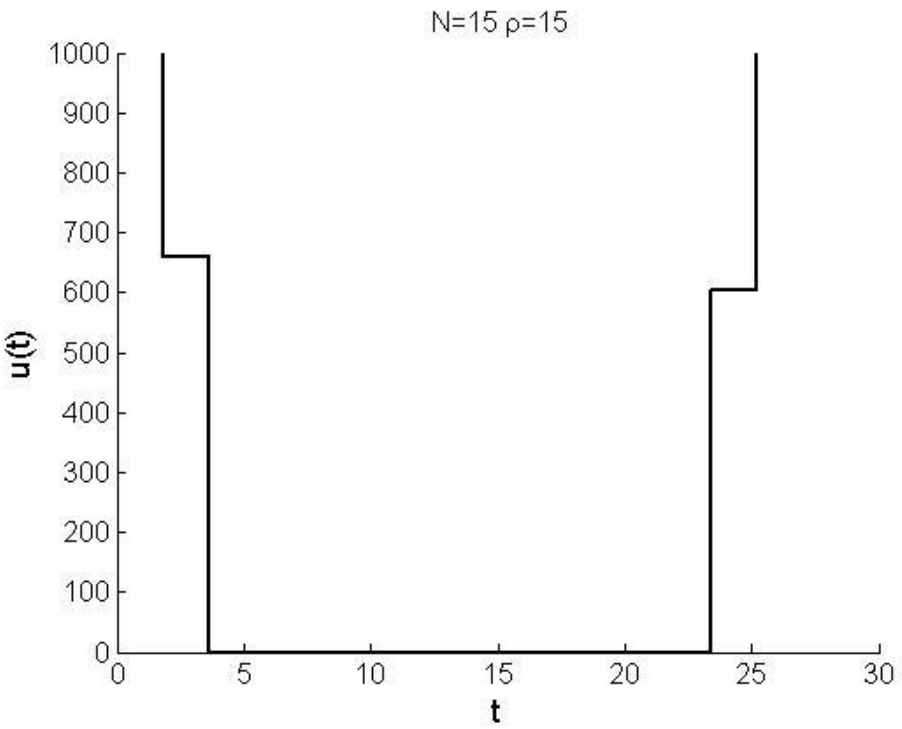

Figure 7: Thrust Profile

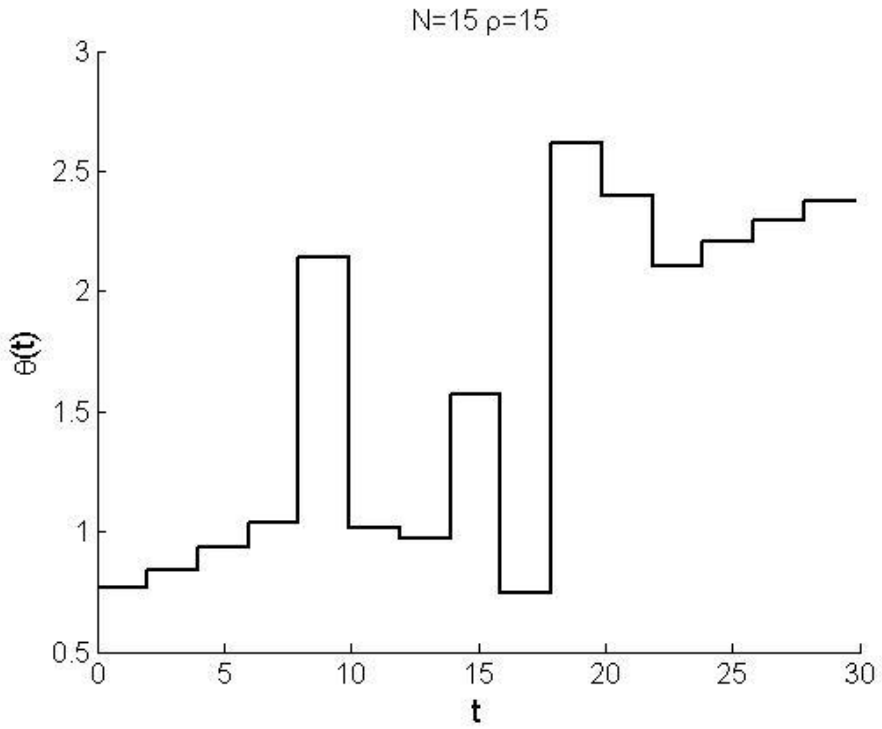

Figure 8: Angle Profile

Figure 6 compares the calculated optimal trajectory for the given system parameters versus the impulsive, ballistic trajectory. As the thrust to weight ratio is relatively high, the calculated trajectory closely resembles the impulsive trajectory. As this value is lowered the calculated trajectory continues to differ greater and greater, as seen in Figure 9 where the maximum thrust is reduced by twofold and again twofold.

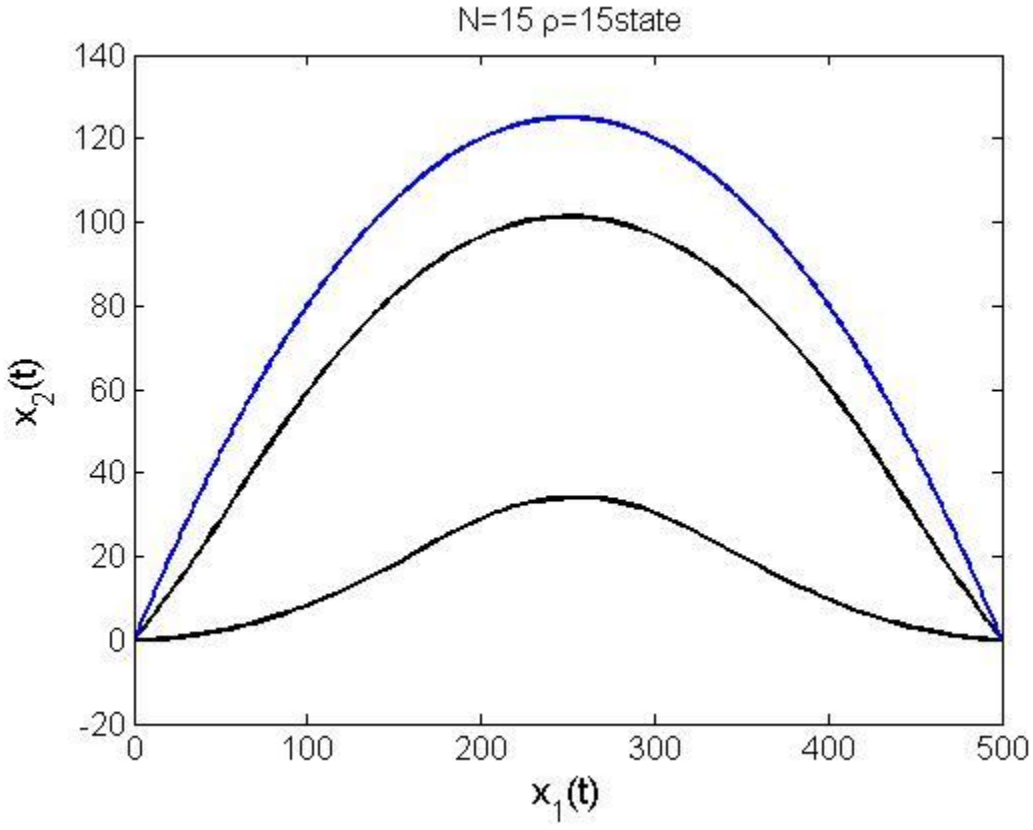

Figure 9: A comparison of a thrust to weight ratio of 5 and 2.5.A higher ratio produces a closer approximation to the impulsive trajectory shown in blue

Of interest in Figure 7 is the asymmetry in the initial burn versus the landing burn. A slight reduction in total thrust or Δv can be seen. The reduction in the landing portion of the maneuver can be attributed to the reduction of mass from the burn on launch. As

mass was expelled to make the launch possible, the landing now takes place with a lighter vessel.

Figure 8 shows the angle of the craft varying immensely during the period between about four seconds and 24 seconds. The angle seems to vary without much reason which raises questions regarding the validity of the result. When considered in conjunction with the thrust profile it begins to be much more reasonable. As the vessel has entered the coast phase the thrust is reduced to zero and therefore the angle of θ has no bearing on the state of the lander.

In reality the angle would need to be ignored during this phase of the maneuver. Alternatively, the lower bound for thrust could be increased very slightly. This increase could be low enough to not affect the overall trajectory but still serve to give the NLP solver something to keep the angle reasonable.

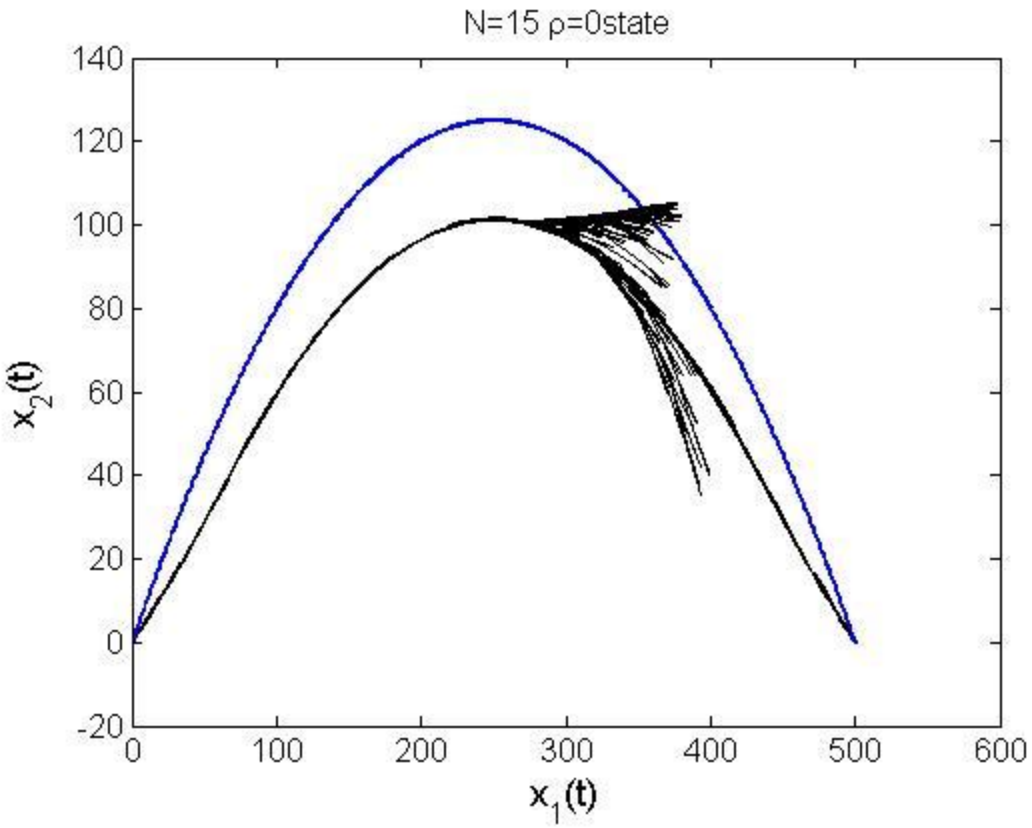

Figure 10: Example of RRT* update

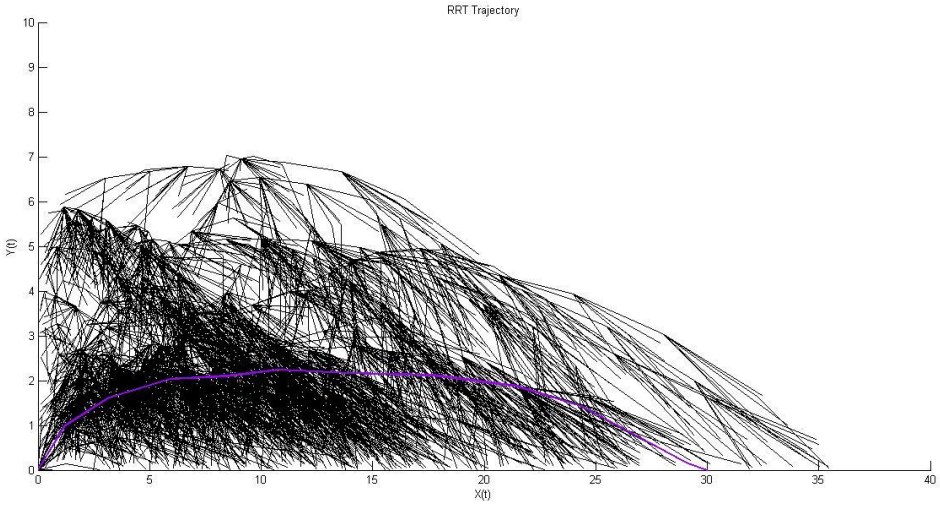

Figure 11: Example of RRT* Trajectory

Figure 10 shows the RRT* running in the loop for one update cycle. If no new obstacles are found or the goal state did not change the loop will return with the same trajectory with which it was seeded. Figure 11 shows an example of a completely RRT formed trajectory. In both figures, the RRT looks to have very densely searched the area. However, only position is visualized by these plots. In reality the algorithm is searching across velocity in both dimensions as well.

4.4 Closing Remarks

Solving with a SQP seems to produce high quality trajectories. Unfortunately, its complexity makes it poor candidate for a real time planner for a lander. The RRT* shows that it can be utilized to produce a real time solution.

Bibliography

[1] Choset, Howie M. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, Mass.: MIT Press, 2005.

[2] D. J. Webb and J. V. D. Berg, "Kinodynamic RRT-: Optimal motion planning for systems with linear differential constraints," arXiv:1205.5088v1, submitted.

[3] Rao, A. V., "A Survey of Numerical Methods for Optimal Control," AAS/AIAA Astrodynamics Specialist Conference, AAS Paper 09-334, Pittsburgh, PA, August 2009.

[4] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in Proc. Robotics: Science and Systems (RSS), 2010.

[5] S. M. LaValle and J. J. Kuffner. *Randomized kinodynamic planning.* International Journal of Robotics Research, 20(5):378--400, May 2001.

[6] Curtis, Howard D. *Orbital Mechanics for Engineering Students*. Amsterdam: Elsevier Butterworth Heinemann, 2010. Print.

[7] ebrary, Inc., and Andrew Ball. *Planetary Landers and Entry Probes*. Cambridge: Cambridge University Press, 2007.

[9] D. G. Hull, "Conversion of Optimal Control Problems into Parameter Optimization Problems," Journal of Guidance, Control, and Dynamics, Vol. 20, No. 1, 1997, pp. 57–60

Appendix I: Matlab Implementation

Introduction

A large portion of the work for this thesis went into the development of effective implentetations of the SQP and RRT for the specific application of the lunar lander. The code provided in this appendix gives the main functions for each written for Matlab. The entirety of the code is provided in supplemental files.

SQP Main Function

```
function dvar0 = runOptimal( )
transLength=500; %(m)

T = 25; % terminal time
N =15; % number of control stages

rho = 15; % weight on missing the final target
beta = 4.9038e12; % parameters of the external force
x0 = zeros(5,1);% initial state
x0(5) = 100; %starting mass
thrustMax=2.5*x0(5);

xf=[transLength,0,0,0,0]; %our goal state

tic
% Options for ODE & NLP Solvers
optODE = odeset( 'RelTol', 1e-2, 'AbsTol', 1e-2);
optNLP = optimset('LargeScale', 'off','GradObj','off',...
      'GradConstr','off','DerivativeCheck', 'off', 'TolX', 1e-5,...
      'TolFun', 1e-9, 'TolCon', 1e-9, 'MaxFunEvals',2e6,...
      'MaxIter', 10000, 'DiffMinChange',1e9,'Algorithm','sqp',...
      'Display','iter');


% Remark: Due to the nonconvex nature of this problem, you may end up
with
% an optimal point that is local and not global.
```

```
%for i=1:5 % use for creating multiple repetitions
ts = 0:(1/N):  1;

dvar = [repmat(thrustMax/2,1,N),pi()/4:pi()/(2*(N-
1)):3*pi()/4,xf(1:4),T*1.1]; % design variable contains N components of
% u, N components of theta and the final position
lb = -Inf(1,2*N+5);
lb(1:N) = 0.00000000001; % enforce lower bound on  Thrust
lb(N+1:2*N)=pi()/6;
lb(end)=T+1;

ub = Inf(1,2*N+5);
ub(end)=T*2;
ub(1:N) = thrustMax; % enforce upper bound on Thrust
ub(N+1:2*N)=5*pi()/6;


% Sequential Approach of Dynamic Optimization
 [dvar0,JO] = fmincon(@(dvar) costFunction(x0, ts,tf,dvar, rho, N,
beta, optODE),...
     dvar,[],[],[],[],lb,ub, ...
     @(dvar) constraintFunction(x0, dvar, ts,tf, N, beta,
optODE),optNLP);

toc %reads out the total time the calculation took

ts=dvar0(end)*ts;
hold on
[topt,xopt,uopt,thetaopt] = plotTrajectory(
x0,N,ts,dvar0,beta,rho,optODE ); % Produces Trajectory plot, thrust
% profile and angle profile
```

RRT Main Function

```
function [path ] = BUILD_RRT_STAR(start,goal )
% RRT implementation with system contraints
% start is our initial state and goal is our goal state
%
clc
close all
time=3; % time allowed in the navigation loop
%% Define the searchable configuration space
xMin=start(1);
xMax=goal(1);
yMin=goal(2);

dotMag=[0,2];
g=1.622; % gravity
yMin=0;
if start(4)>=0
yMax=start(2);
else
    yMax=start(2)*2;
end

segmentLength =1.75;
world = [ xMin,xMax,yMin,yMax ];
%%

%create random ground
ground = createGround( start, goal);

tree=start; %initalize tree with goal state
figure % for visualiztion
hold on

tic

if (norm(start(1:2)-goal(1:2))<segmentLength) && ...
        (collision(start,goal,world)==0) % obstacle detection
    path=[start,goal];
else
    numPaths=0;
    i=0;
    while toc <time
        [tree,flag] =
extendTree(tree,goal,segmentLength,world,ground,dotMag,g,numPaths);
        numPaths = numPaths +flag;
        i=i+1;
    end
    toc
    iterations=i
end
```

```
path = findMinimumPath(tree,goal); % outputs our Trajectory
toc;
if path==0
else

    plot(path(:,1),path(:,2),'LineWidth',1) %plots our output
end
end
```

Vita

Ian Miller was born in Phoenixville, Pa on August 3rd, 1991 to Wolfe and Claire Miller. After graduating from Holy Name High School in West Reading, Pa he attended Lehigh University in Bethlehem Pennsylvania. In May of 2014 he completed a Bachelors of Science in Electrical Engineering. He then continued his studies at Lehigh with a Master of Science in Electrical Engineering. He will be graduating with honors in December of 2015.