EML6934 Optimal Control
Final Project

Elias Reyes

April 18, 2022

Contents

List of Figures

List of Tables

Listings

1 Problem Statement

2 Differential Equations of Motion

The differential equations of motion were derived using both Newton's second law. The schematic for the problem can be seen Figure 1. The spacecraft is modeled as point P of mass m. The spacecraft moves relative to an inertial reference frame l. The reference frame fixed in l is expressed as $\{e_x, e_y, e_z\}$. The position of the spacecraft is denoted as $r_{P/O}$, where O is modeled as the sun, fixed in l. The spacecraft is parameterized in the basis $\{u_r, u_\theta, u_z\}$, where the rotation is about $u_z = e_z$. The rotation creates an angle θ between e_x and u_r, which can be seen in Figure 2. Two forces are said to act on the spacecraft. The first is the gravitational force which is given as

$$G = -m\mu \frac{r_{P/O}}{||r_{P/O}||^3}, \tag{1}$$

while the second is the thrust force given as

$$T = Tw, \tag{2}$$

where w is the unit vector that lies an angle β from the direction u_θ as seen in Figure 3.

2.1 Position, Velocity and Acceleration of the Spacecraft

As seen in Figure 1, the position of the spacecraft represented in reference frame A, is given as

$$^{A}\vec{r}_{P/O} = ru_r. \tag{3}$$

The velocity can then be represented in the inertial frame l by using equation 4, where $^{l}\vec{\omega}^{A}$ is the angular velocity between reference frame l and A.

$$\frac{^{l}d\vec{r}_{P/O}}{dt} = \frac{^{A}d\vec{r}_{P/O}}{dt} + \ {^{l}\vec{\omega}^{A}} \ \times \ {^{A}\vec{r}_{P/O}} \tag{4}$$

Using Equation 4, the velocity of the spacecraft in the inertial frame is then formulated as

$$\begin{aligned} {^{l}\vec{v_p}} &= \frac{^{l}d\vec{r}_{P/O}}{dt} \\ {^{l}\vec{v_p}} &= \dot{r}u_r + \ \dot{\theta}u_z \ \times \ ru_r \\ {^{l}\vec{v_p}} &= \dot{r}u_r + \ \dot{\theta}ru_\theta \end{aligned} \tag{5}$$

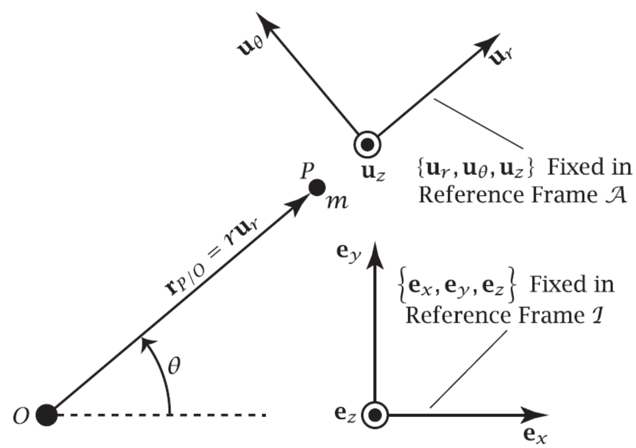

Figure 1: Schematic of particle moving in an inertially fixed plane

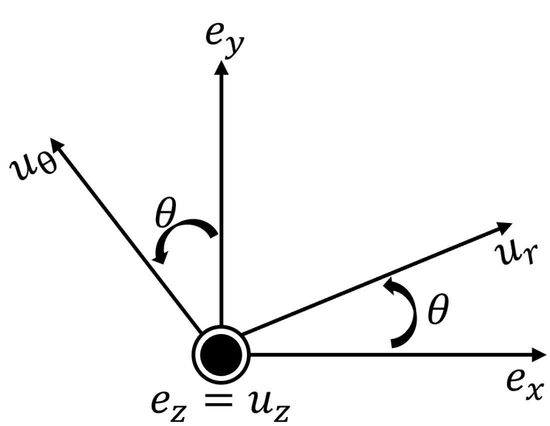

Figure 2: Reference frame rotation

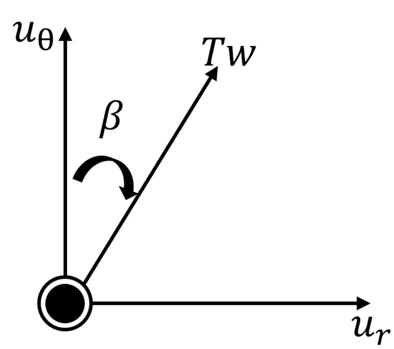

Figure 3: Thrust Force at Angle β

The acceleration of the particle can then be formulated as

$$
\begin{aligned}
{}^{l}\vec{a_p} &= \frac{{}^{l}d\vec{v_P}}{dt} \\
{}^{l}\vec{a_p} &= \frac{{}^{A}d\vec{v_P}}{dt} + \ {}^{l}\vec{\omega}^{A} \ \times \ {}^{A}\vec{v_P} \\
{}^{l}\vec{a_p} &= \ddot{r}u_r + \ (\ddot{\theta}r + \dot{\theta}\dot{r})u_\theta + \ \dot{\theta}\dot{r}u_\theta - \ \dot{\theta}^2 r u_r \\
{}^{l}\vec{a_p} &= (\ddot{r} - \ \dot{\theta}^2 r)u_r + \ (\ddot{\theta}r + 2\dot{\theta}\dot{r})u_\theta \qquad (6)
\end{aligned}
$$

2.2 Newton's Second Law for A Particle

Newton's second law for a particle is represented by

$$\sum F_P = m_P * a_P. \qquad (7)$$

Figure 4 represents the free body diagram of the particle system. F_G, represented by equation 1, is the gravitational force and acts along the u_r direction. F_T, represented by equation 2, is the thrust force and acts in the direction w. It can be seen in Figure 3 that the thrust force can be re-written as

$$F_T = Tsin(\beta)u_r + T\cos(\beta)u_\theta, \qquad (8)$$

while the gravitational force can be written as

$$F_G = -m\mu\frac{1}{r^2}u_r. \qquad (9)$$

We can now substitute equations 6, 8, and 9 into equation 7 to obtain

$$-m\mu\frac{1}{r^2}u_r + Tsin(\beta)u_r + T\cos(\beta)u_\theta = m[(\ddot{r} - \ \dot{\theta}^2 r)u_r + \ (\ddot{\theta}r + 2\dot{\theta}\dot{r})u_\theta].$$

After equating terms, the two equations of motion using Newtons Seconds become:

$$(u_r) \qquad \ddot{r} = \dot{\theta}^2 r - \frac{\mu}{r^2} + \frac{Tsin(\beta)}{m} \qquad (10)$$

$$(u_\theta) \qquad \ddot{\theta} = -\frac{2\dot{r}\dot{\theta}}{r} + \frac{T\cos(\beta)}{mr} \qquad (11)$$

2.3 Conversion to First-Order Equations

To re-write the two second-order equations into first four first-order equations, the following substitutions can be made:

$$
\begin{aligned}
\dot{r} &= v_r, \qquad (12) \\
r\dot{\theta} &= v_\theta, \\
\dot{\theta} &= \frac{v_\theta}{r}. \qquad (13)
\end{aligned}
$$

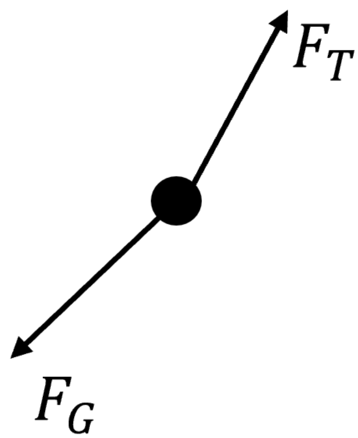

Figure 4: Free Body Diagram

Then, taking the derivatives of r and θ of equations 12 and 13, respectively, we obtain:

$$\ddot{r} = \dot{v}_r, \tag{14}$$

$$\ddot{\theta} = \frac{\dot{v}_\theta r - v_\theta \dot{r}}{r^2}. \tag{15}$$

After substituting equations 12, 13, 14, and 15 into equations 10 and 11, two first-order equations are derived as:

$$\begin{aligned}\dot{v}_r &= \frac{v_\theta^2 r}{r^2} - \frac{\mu}{r^2} + \frac{T\sin(\beta)}{m},\\ \dot{v}_r &= \frac{v_\theta^2}{r} - \frac{\mu}{r^2} + \frac{T\sin(\beta)}{m},\end{aligned} \tag{16}$$

and,

$$\begin{aligned}\frac{\dot{v}_\theta r - v_\theta v_r}{r^2} &= -\frac{2v_\theta v_r}{r^2} + \frac{T\cos(\beta)}{mr},\\ \dot{v}_\theta r - v_\theta v_r &= -\frac{2v_\theta v_r r^2}{r^2} + \frac{T\cos(\beta) r^2}{mr},\\ \dot{v}_\theta &= -\frac{2v_\theta v_r}{r} + \frac{T\cos(\beta)}{m} + \frac{v_\theta v_r}{r},\\ \dot{v}_\theta &= -\frac{v_\theta v_r}{r} + \frac{T\cos(\beta)}{m}.\end{aligned} \tag{17}$$

Next, a fifth first-order equation is given as

$$\dot{m} = -\frac{T}{v_e}. \tag{18}$$

The five first-order differential equations are listed below:

$$
\begin{aligned}
\dot{r} &= v_r, \\
\dot{\theta} &= \frac{v_\theta}{r}, \\
\dot{v}_r &= \frac{v_\theta^2}{r} - \frac{\mu}{r^2} + \frac{T\sin(\beta)}{m}, \\
\dot{v}_\theta &= -\frac{v_\theta v_r}{r} + \frac{T\cos(\beta)}{m}, \\
\dot{m} &= -\frac{T}{v_e}.
\end{aligned}
$$

3 Formulation of Optimal Control Problem

There are two objectives for the optimal control problem. The first objective is to minimize the time to transfer from an initial circular orbit to a final circular orbit. The second objective is to maximize the terminal mass when transferring from an initial circular orbit to a final circular orbit. Below is an overview of the problem:

$$
\begin{aligned}
\text{Objective 1}: &\quad \min\ t_f \\
\text{Objective 2}: &\quad \max\ m(t_f) \\
\text{State}: &\quad r(t), \theta(t), v_r(t), v_\theta(t), m(t) \\
\text{Control}: &\quad \beta(t), T(t)
\end{aligned}
$$

4 Numerical Solution of Optimal Control Problem

The optimal control problem was solved using multiple interval Legendre-Gauss-Radau collocation. For simplicity, Adigator was used for automatic differentiation and IPOPT was used for the nonlinear optimization. For this study, four different cases were investigated. The four different discretizations of Legendre-Gauss-Radau collocation are:

1. Minimize t_f with with unconstrained control parameterized by polynomial degrees N = (3,4) with intervals of K = (2,4,8,16,32)

2. Minimize $m(t_f)$ with with unconstrained control parameterized by polynomial degrees N = (3,4) with intervals of K = (2,4,8,16,32)

3. Minimize t_f with with constrained control parameterized by polynomial degrees N = (3,4) with intervals of K = (2,4,8,16,32)

4. Minimize $m(t_f)$ with with constrained control parameterized by polynomial degrees N = (3,4) with intervals of K = (2,4,8,16,32)

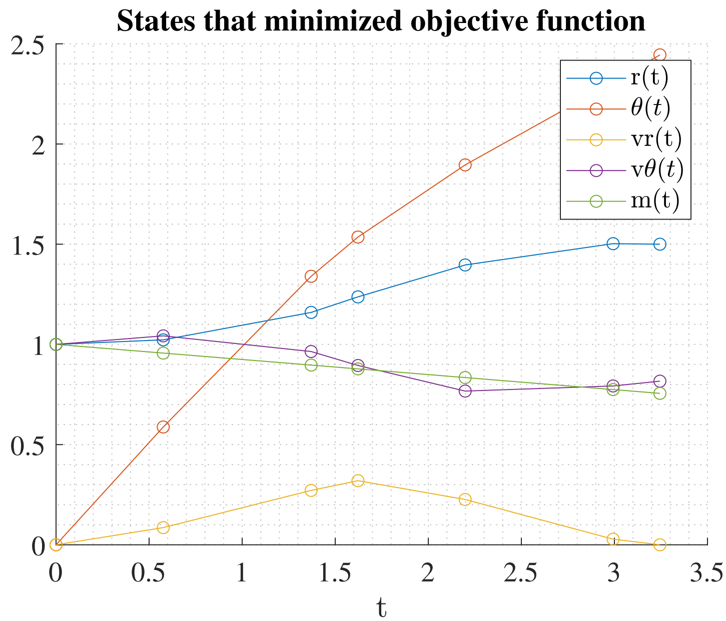

Figure 5: States for trajectory that minimized terminal time ($N : 3$, $K : 2$)

4.1 Minimize Terminal Time with Unconstrained Control

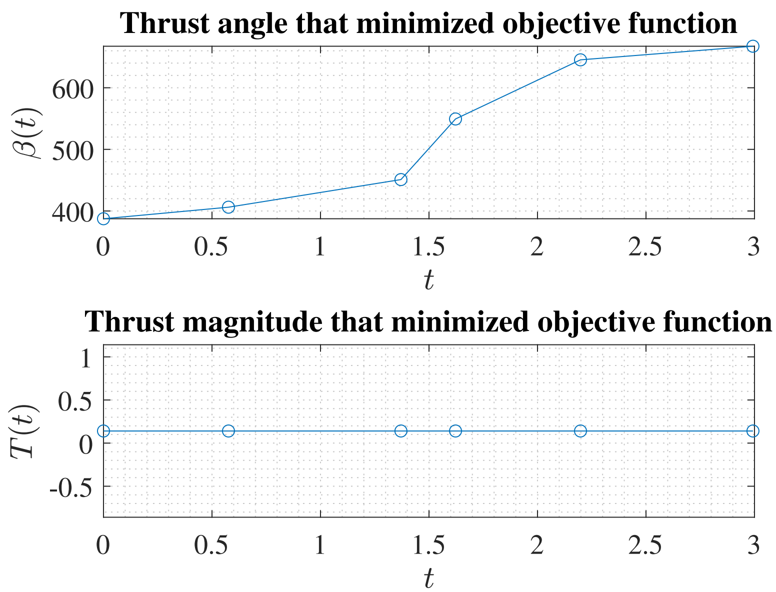

Figure 6: Control that minimized terminal time ($N : 3$, $K : 2$)

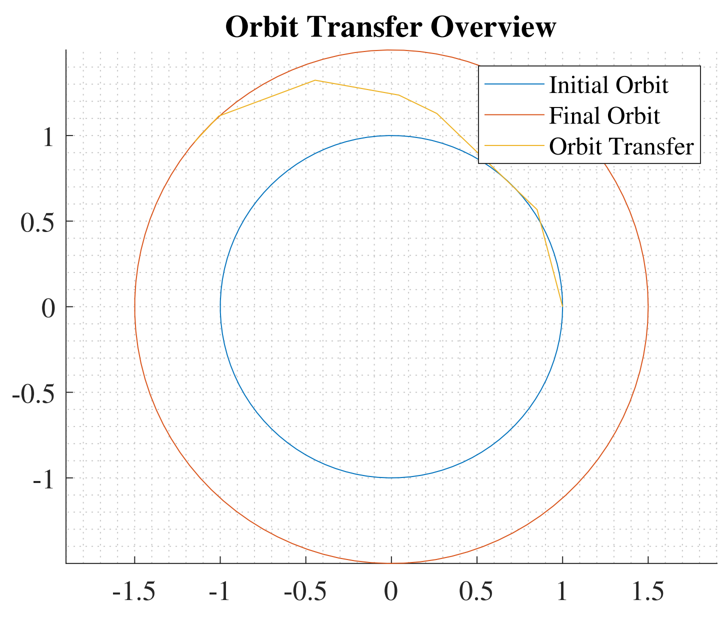

Figure 7: Trajectory from initial to final orbit ($N : 3$, $K : 2$)

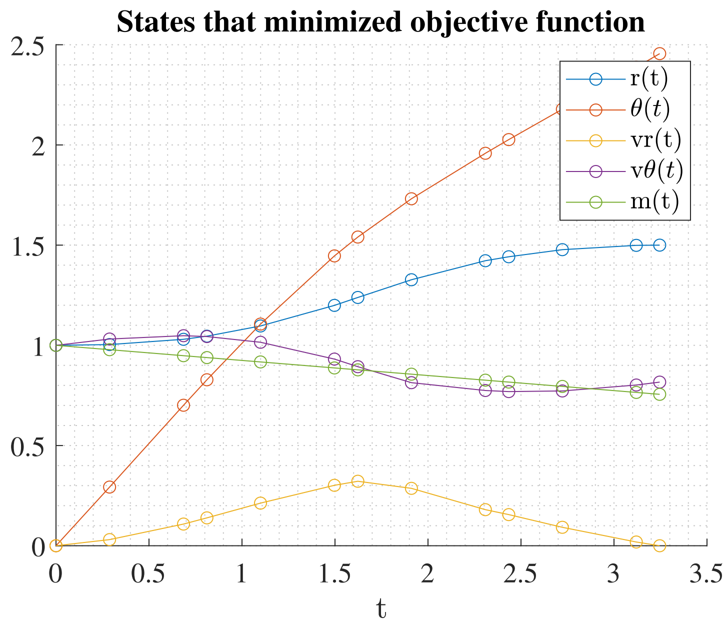

Figure 8: States for trajectory that minimized terminal time ($N:3$, $K:4$)

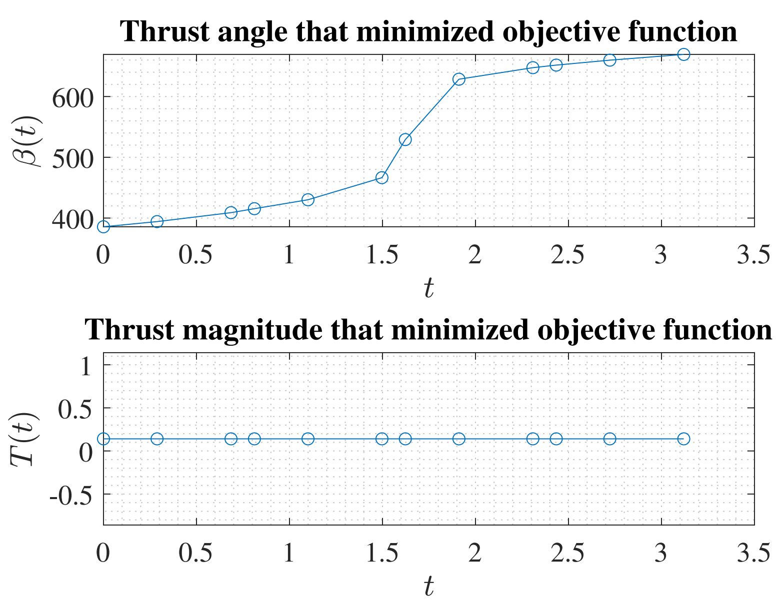

Figure 9: Control that minimized terminal time ($N:3$, $K:4$)

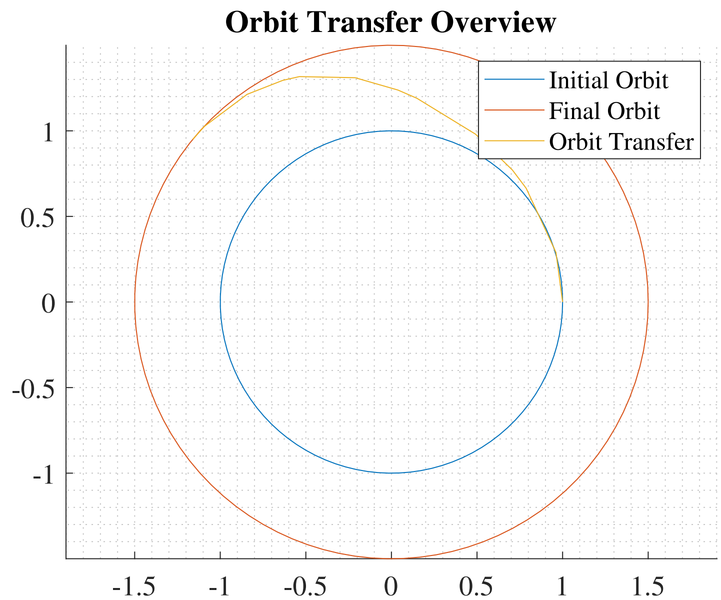

Figure 10: Trajectory from initial to final orbit ($N : 3$, $K : 4$)

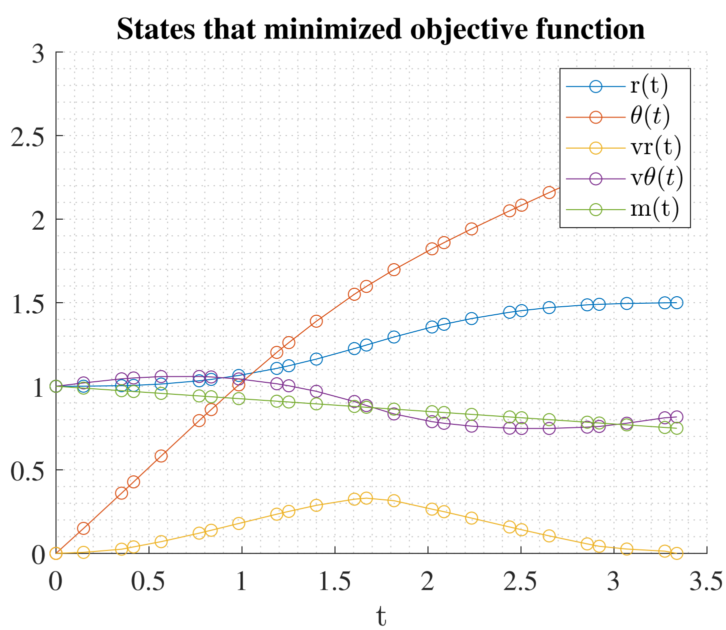

Figure 11: States for trajectory that minimized terminal time ($N : 3$, $K : 8$)

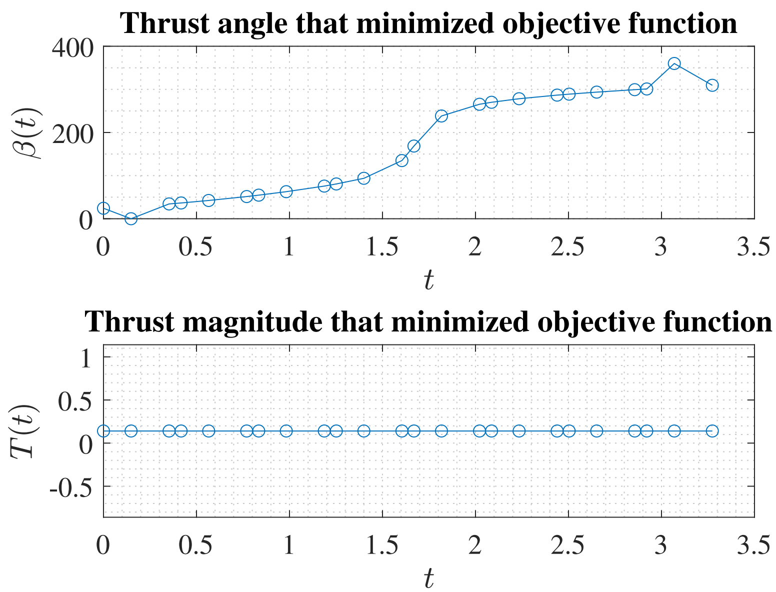

Figure 12: Control that minimized terminal time ($N : 3$, $K : 8$)

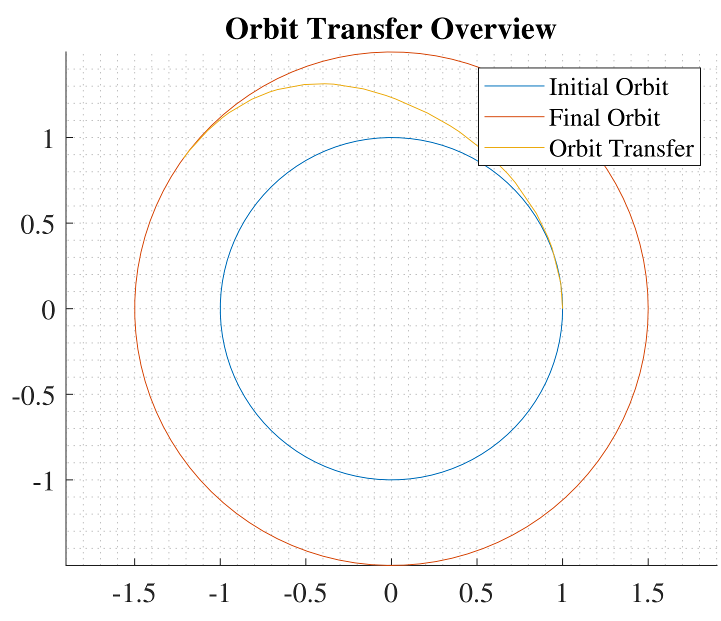

Figure 13: Trajectory from initial to final orbit ($N : 3$, $K : 8$)

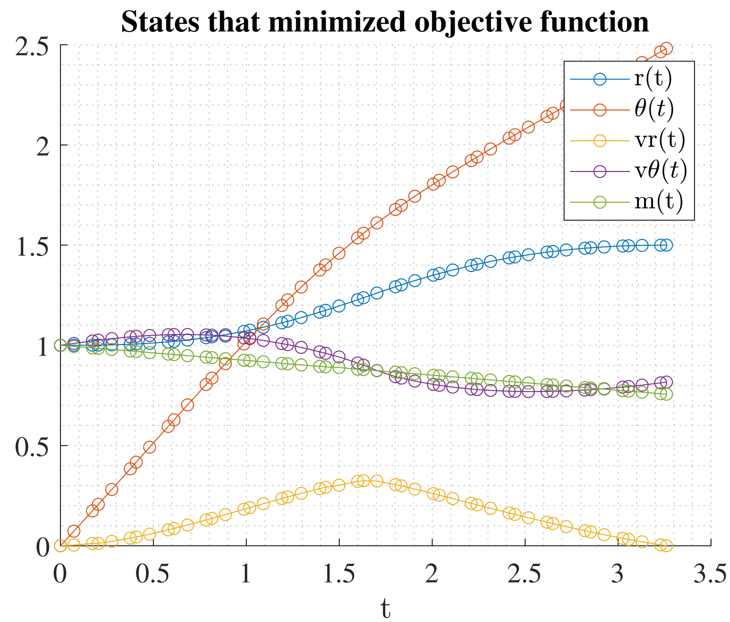

Figure 14: States for trajectory that minimized terminal time ($N : 3$, $K : 16$)

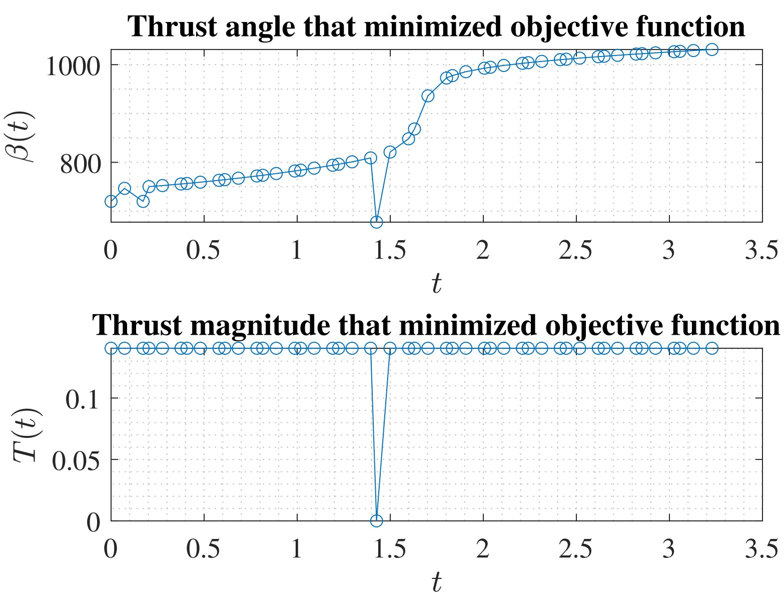

Figure 15: Control that minimized terminal time ($N : 3$, $K : 16$)

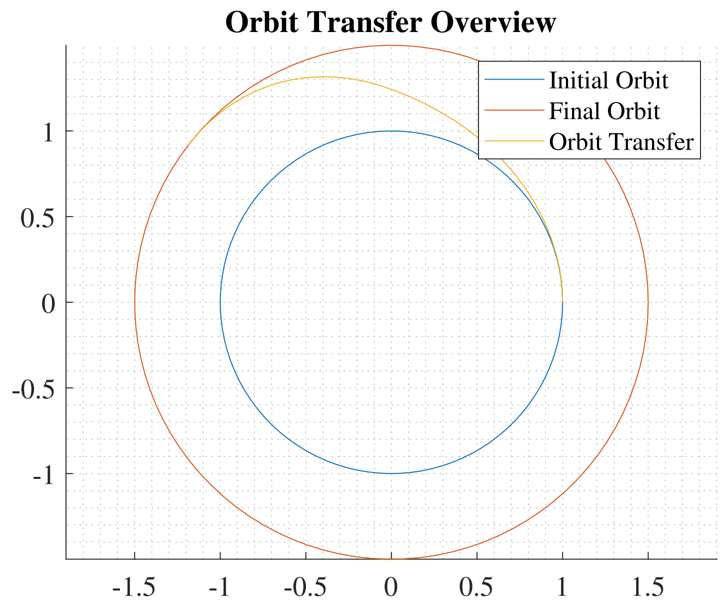

Figure 16: Trajectory from initial to final orbit ($N : 3$, $K : 16$)

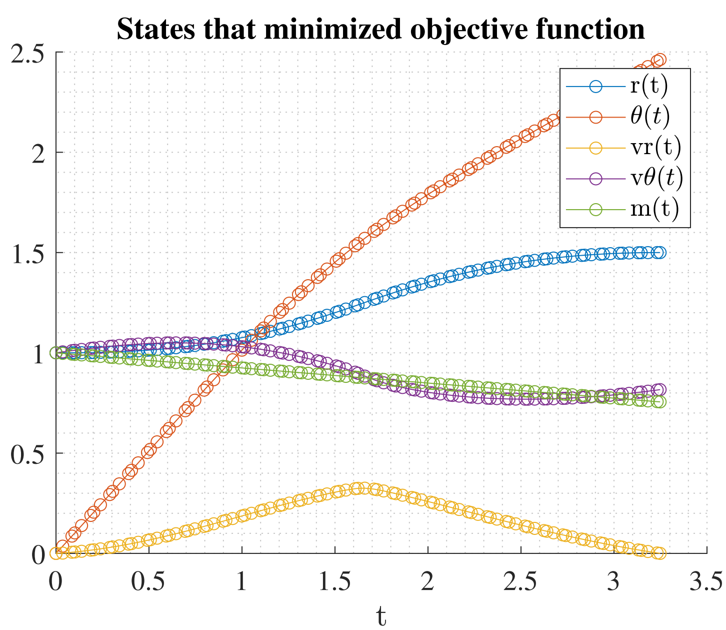

Figure 17: States for trajectory that minimized terminal time ($N : 3$, $K : 32$)

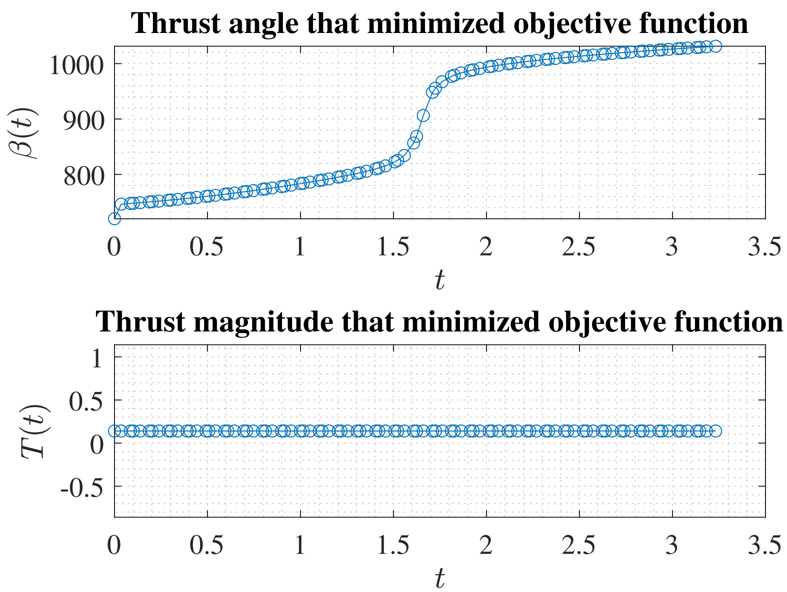

Figure 18: Control that minimized terminal time ($N : 3$, $K : 32$)

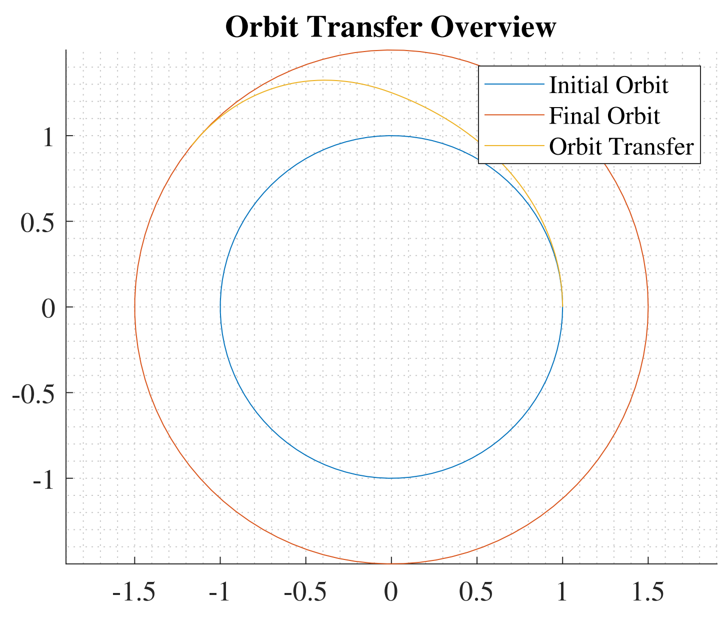

Figure 19: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

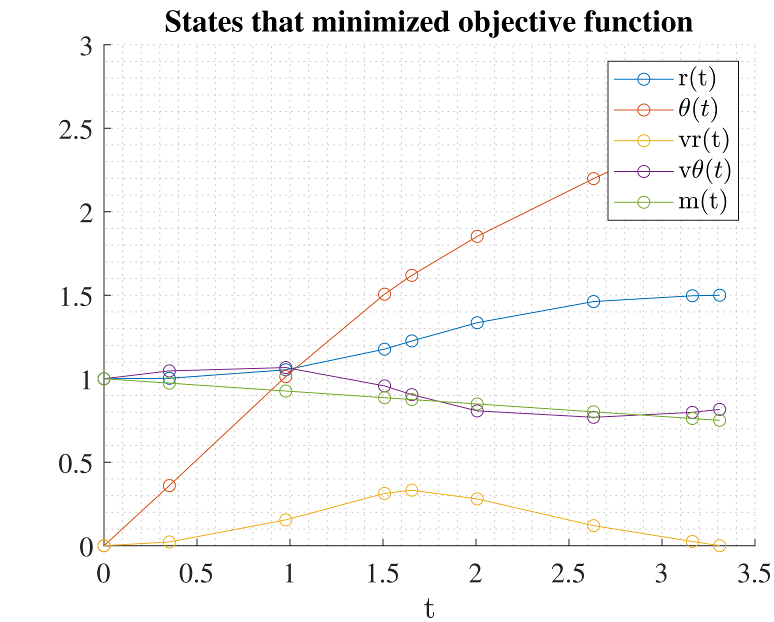

Figure 20: States for trajectory that minimized terminal time ($N:4$, $K:2$)

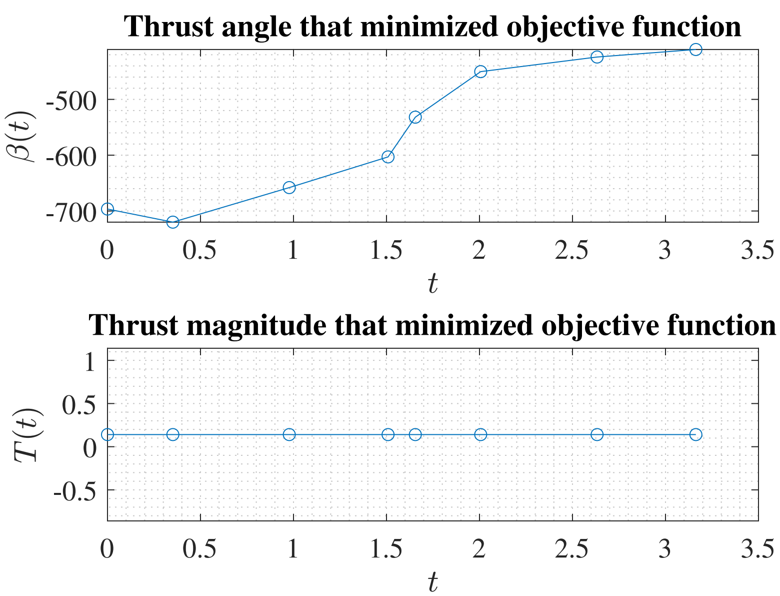

Figure 21: Control that minimized terminal time ($N:4$, $K:2$)

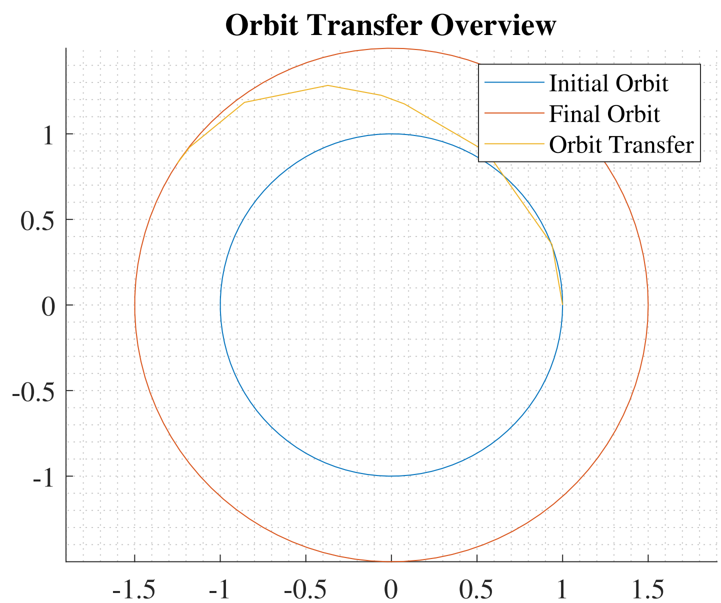

Figure 22: Trajectory from initial to final orbit ($N : 4$, $K : 2$)

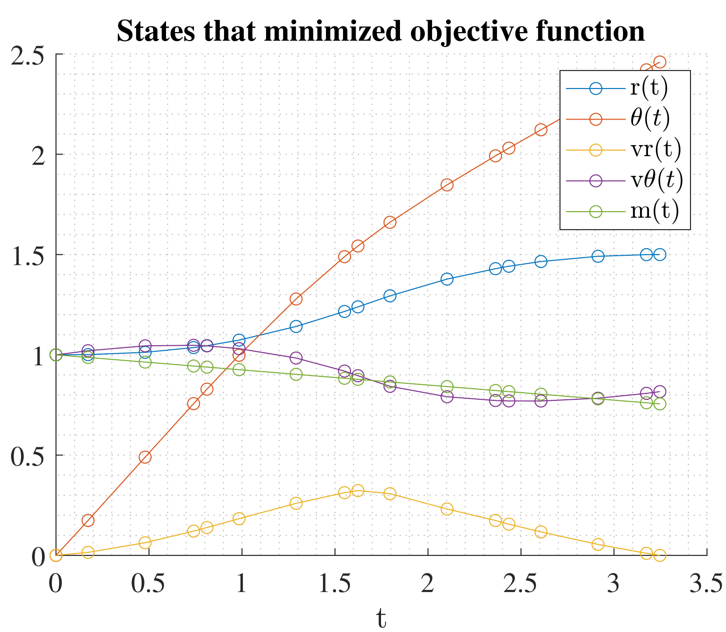

Figure 23: States for trajectory that minimized terminal time ($N : 4$, $K : 4$)

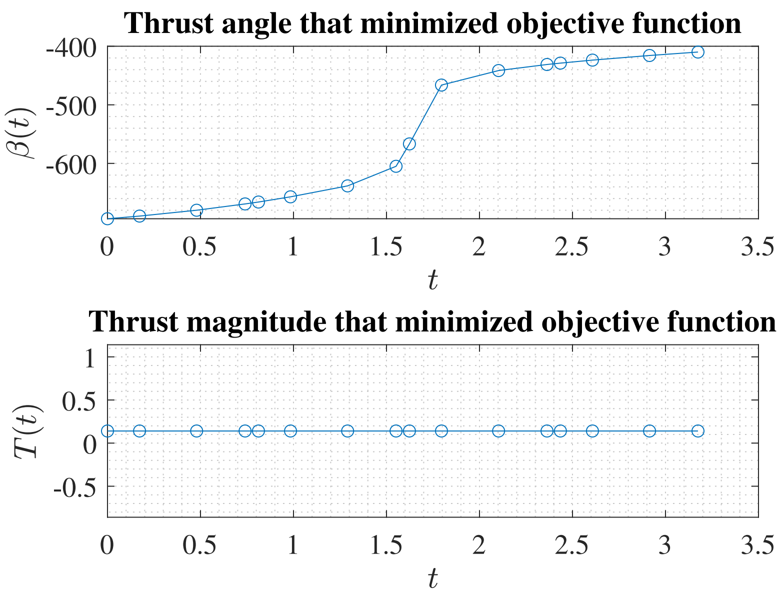

Figure 24: Control that minimized terminal time ($N : 4$, $K : 4$)

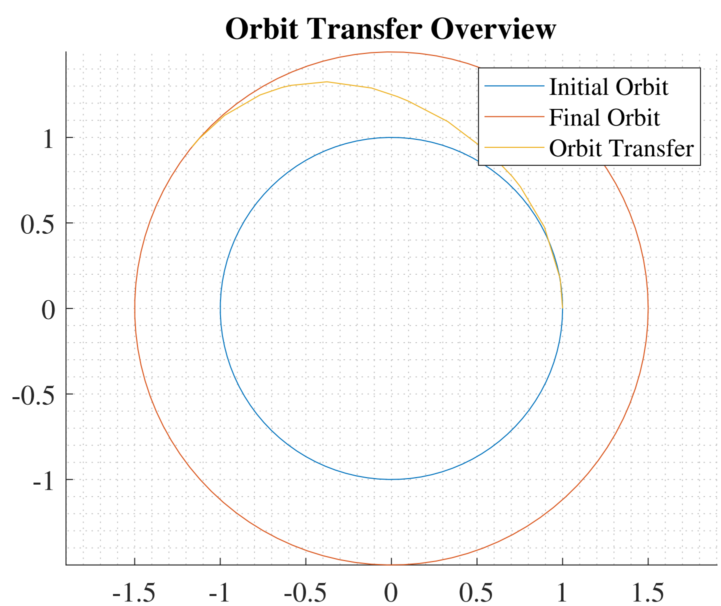

Figure 25: Trajectory from initial to final orbit ($N : 4$, $K : 4$)

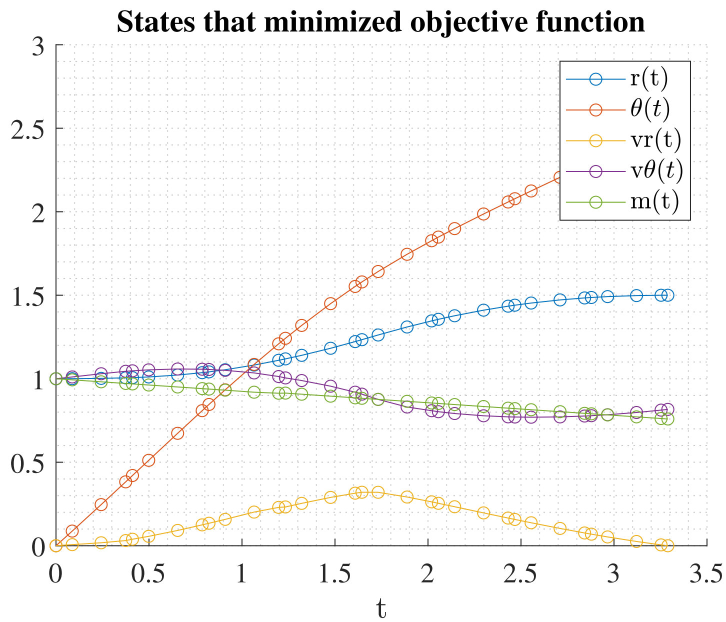

Figure 26: States for trajectory that minimized terminal time ($N:4$, $K:8$)

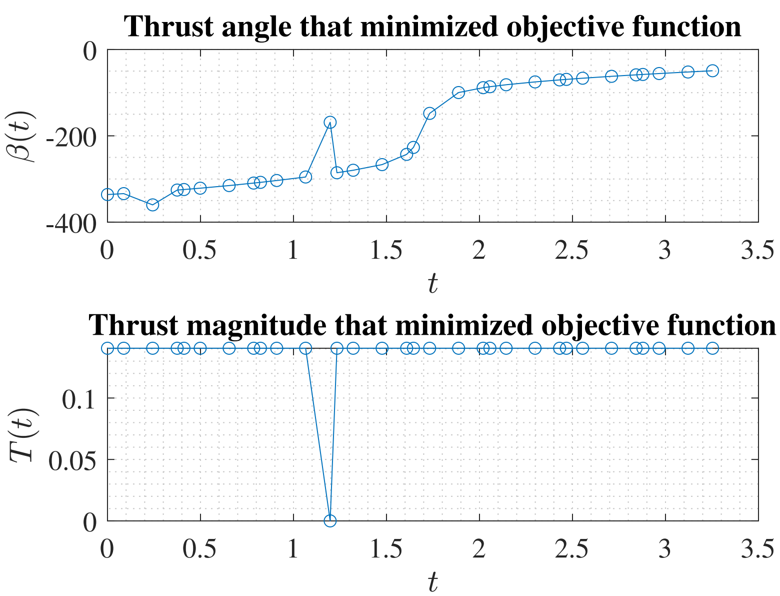

Figure 27: Control that minimized terminal time ($N:4$, $K:8$)

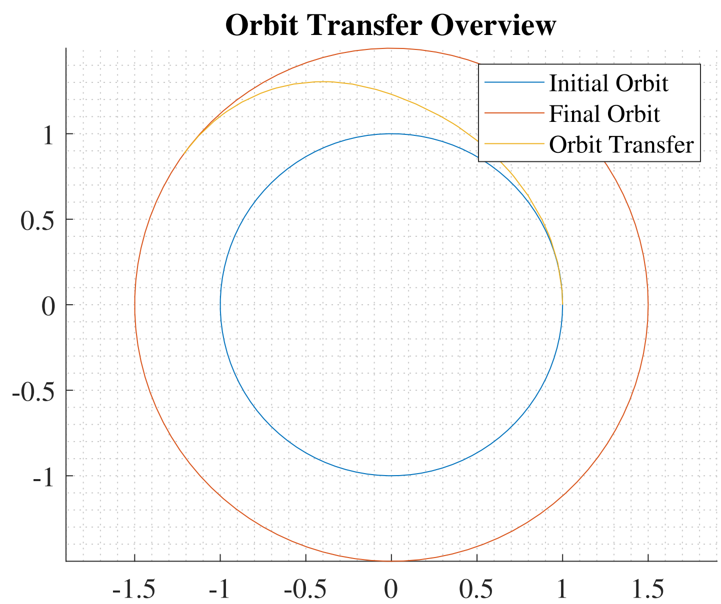

Figure 28: Trajectory from initial to final orbit ($N:4$, $K:8$)

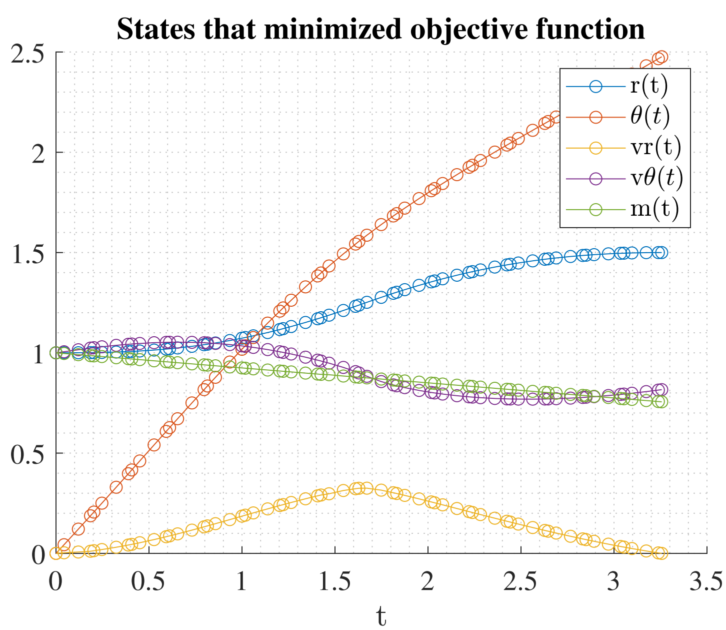

Figure 29: States for trajectory that minimized terminal time ($N:4$, $K:16$)

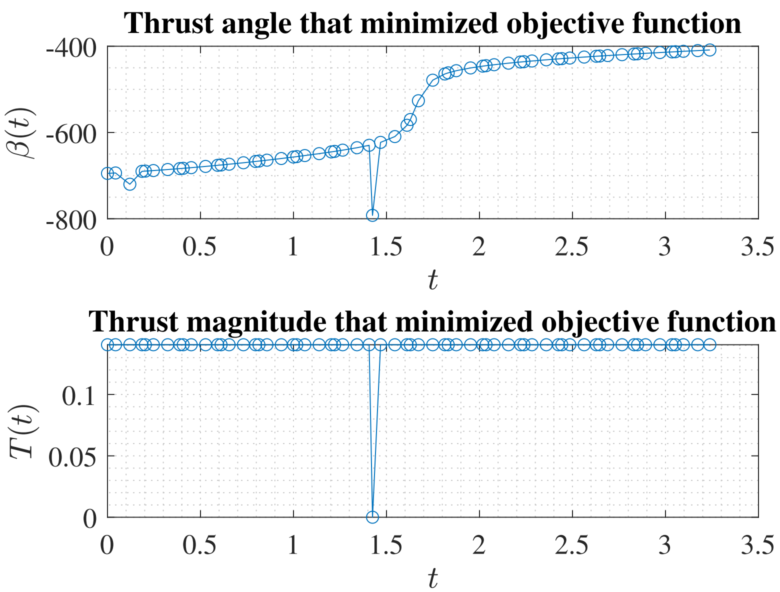

Figure 30: Control that minimized terminal time ($N : 4$, $K : 16$)

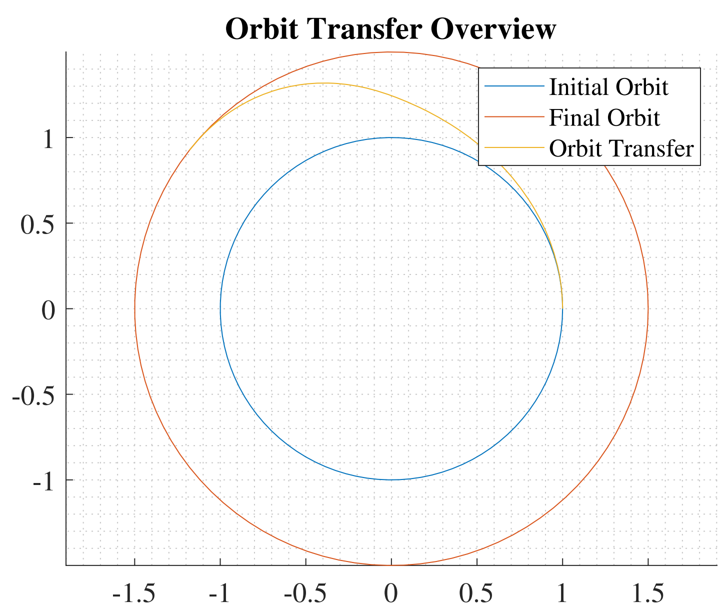

Figure 31: Trajectory from initial to final orbit ($N : 4$, $K : 16$)

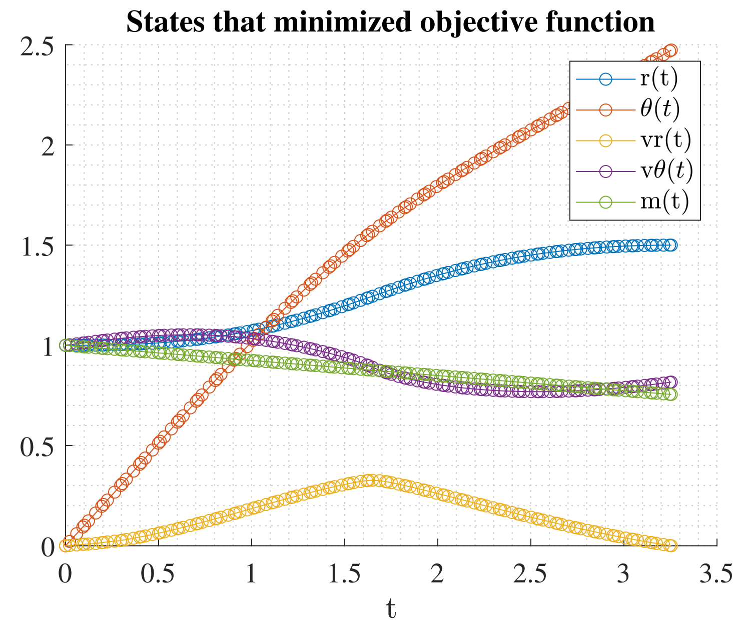

Figure 32: States for trajectory that minimized terminal time ($N : 4$, $K : 32$)

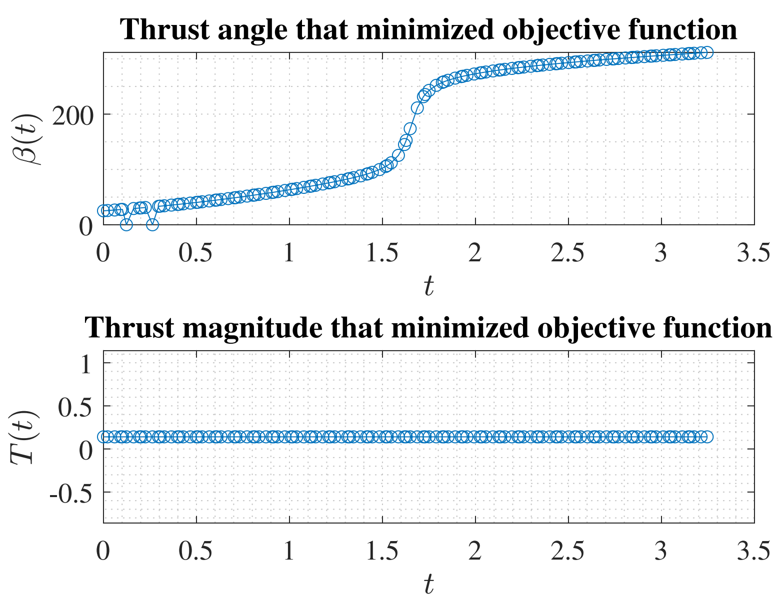

Figure 33: Control that minimized terminal time ($N : 4$, $K : 32$)

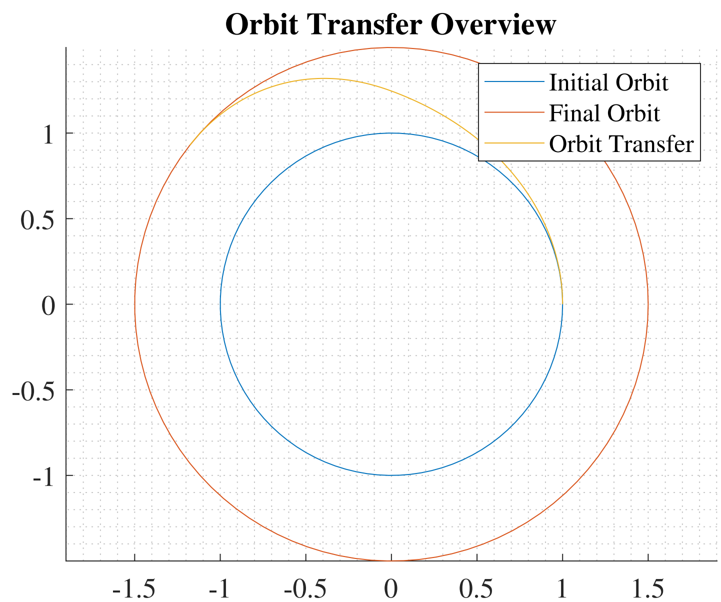

Figure 34: Trajectory from initial to final orbit ($N : 4$, $K : 32$)

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	35	0.185	3.2444	0.75569	0
3	4	66	0.427	3.2457	0.75559	0
3	8	155	0.623	3.3381	0.74864	0
3	16	110	0.499	3.2605	0.75619	0
3	32	187	0.893	3.2479	0.75543	0
4	2	80	0.337	3.31	0.75075	0
4	4	153	0.588	3.2466	0.75552	0
4	8	126	0.54	3.2896	0.75912	0
4	16	144	0.659	3.2567	0.75572	0
4	32	174	0.952	3.2546	0.75492	1

Table 1: Results for minimizing t_f with unconstrained control

4.2 Maximize Terminal Mass with Unconstrained Control

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	621	1.96	10.5501	0.90956	0
3	4	4271	13.906	21.0208	0.90633	-2
3	8	416	1.477	8.8194	0.90719	0
3	16	451	1.759	7.3181	0.9072	0
3	32	204	0.999	11.8567	0.9072	0
4	2	214	0.758	11.5211	0.90659	-2
4	4	1720	5.772	7.6934	0.90717	0
4	8	231	0.903	9.4231	0.90718	0
4	16	138	0.657	10.3297	0.90721	0
4	32	131	0.749	8.2286	0.90719	1

Table 2: Results for maximizing m_f with unconstrained control

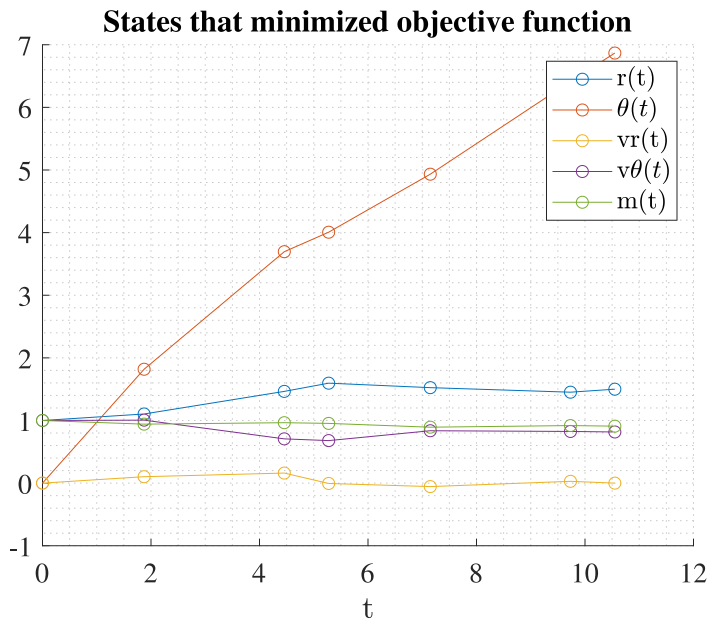

Figure 35: States for trajectory that maximized terminal mass ($N : 3$, $K : 2$)

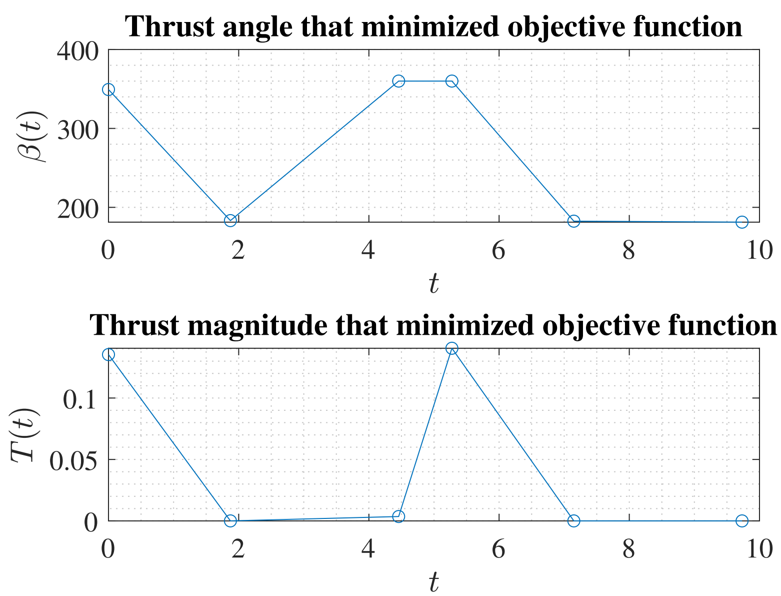

Figure 36: Control that maximized terminal mass ($N : 3$, $K : 2$)

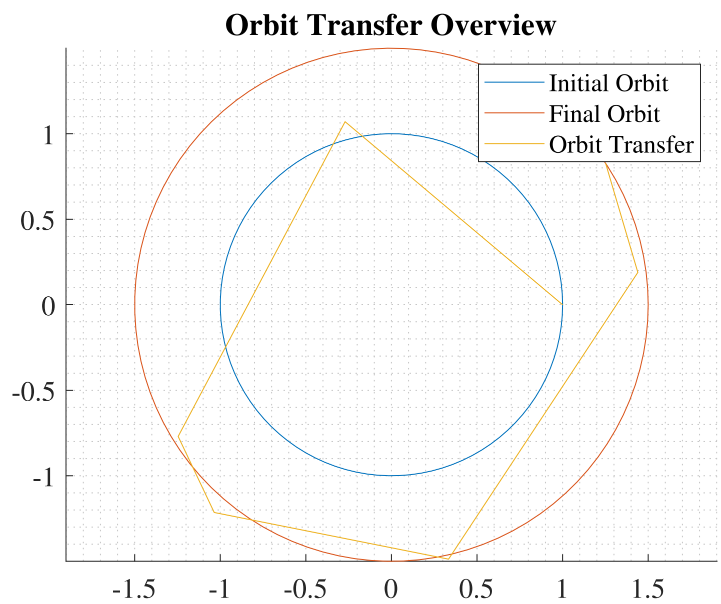

Figure 37: Trajectory from initial to final orbit ($N:3$, $K:2$)

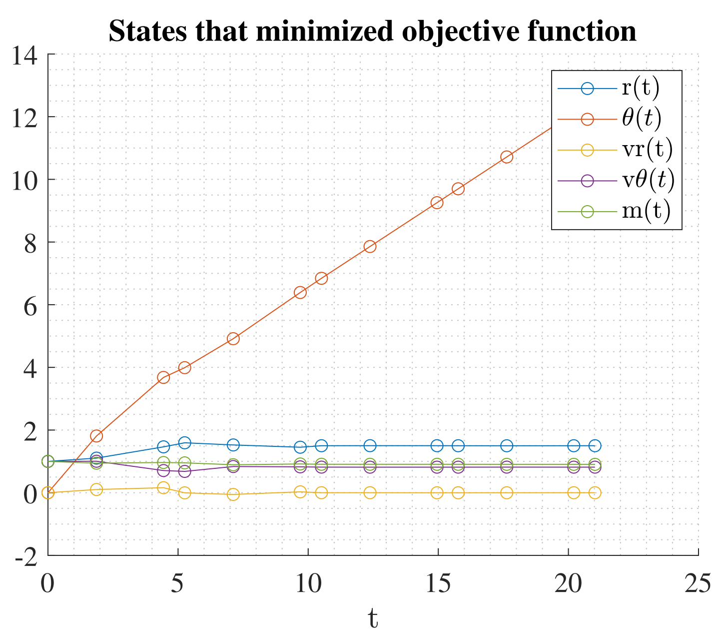

Figure 38: States for trajectory that maximized terminal mass ($N:3$, $K:4$)

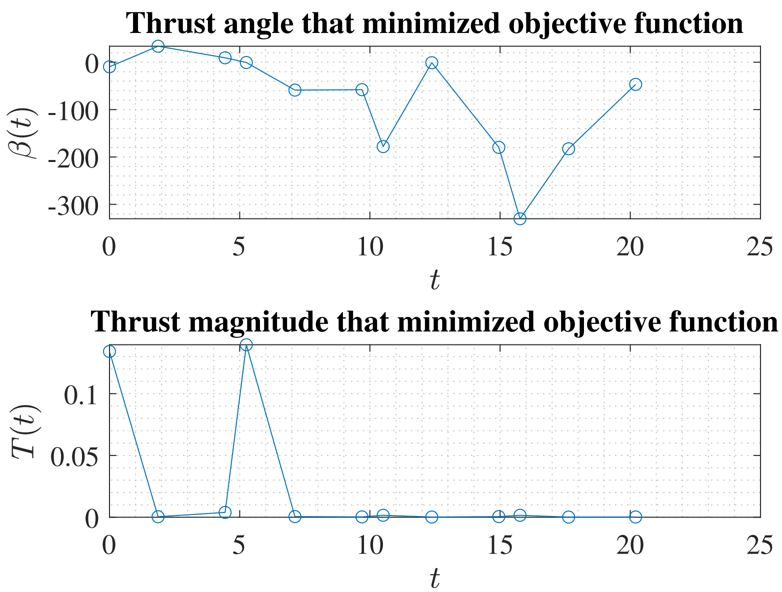

Figure 39: Control that maximized terminal mass ($N:3$, $K:4$)

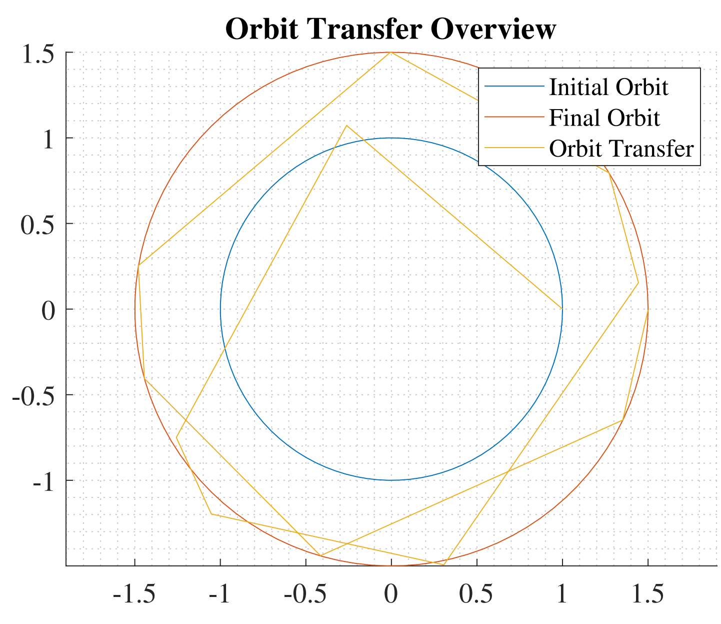

Figure 40: Trajectory from initial to final orbit ($N:3$, $K:4$)

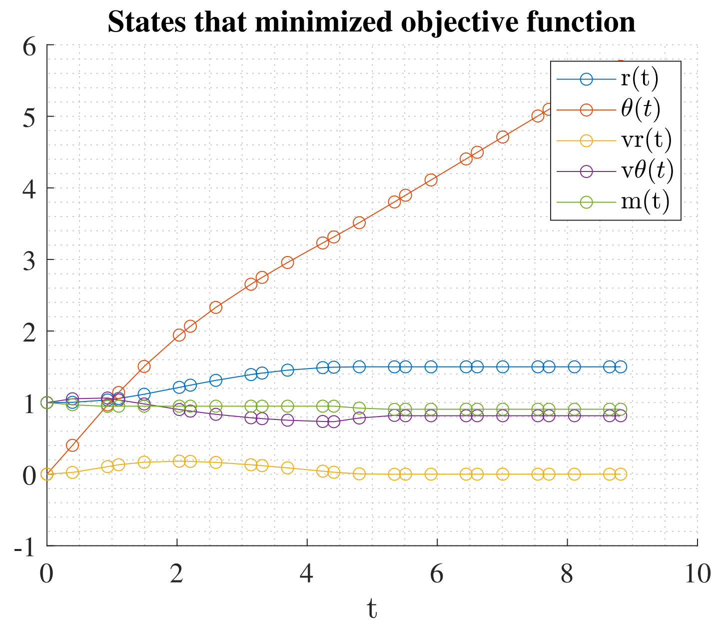

Figure 41: States for trajectory that maximized terminal mass ($N : 3$, $K : 8$)

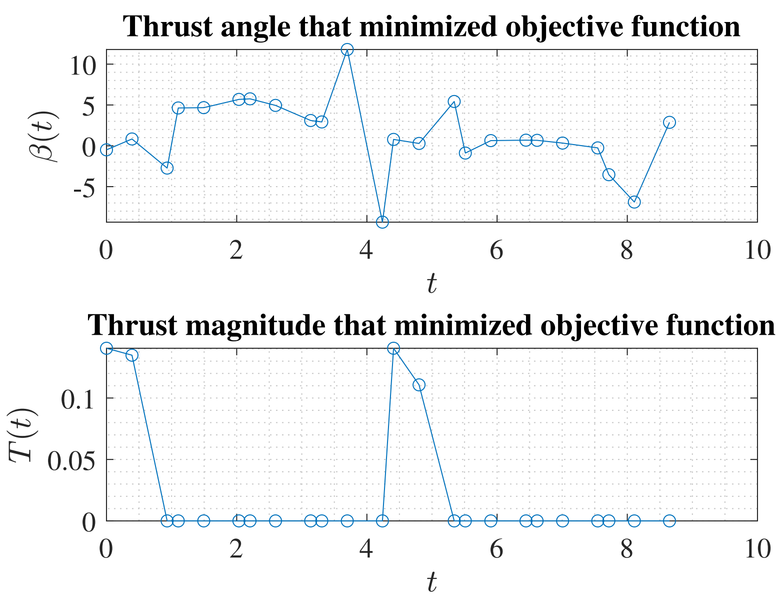

Figure 42: Control that maximized terminal mass ($N : 3$, $K : 8$)

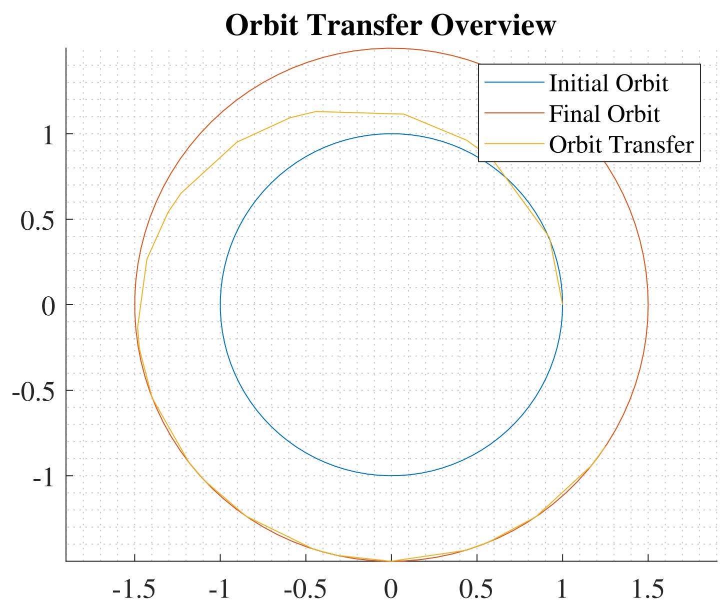

Figure 43: Trajectory from initial to final orbit ($N:3$, $K:8$)

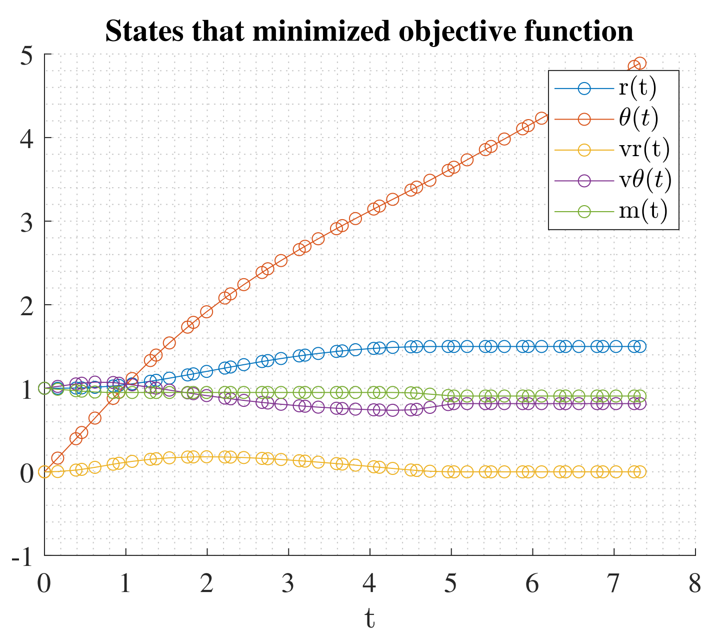

Figure 44: States for trajectory that maximized terminal mass ($N:3$, $K:16$)

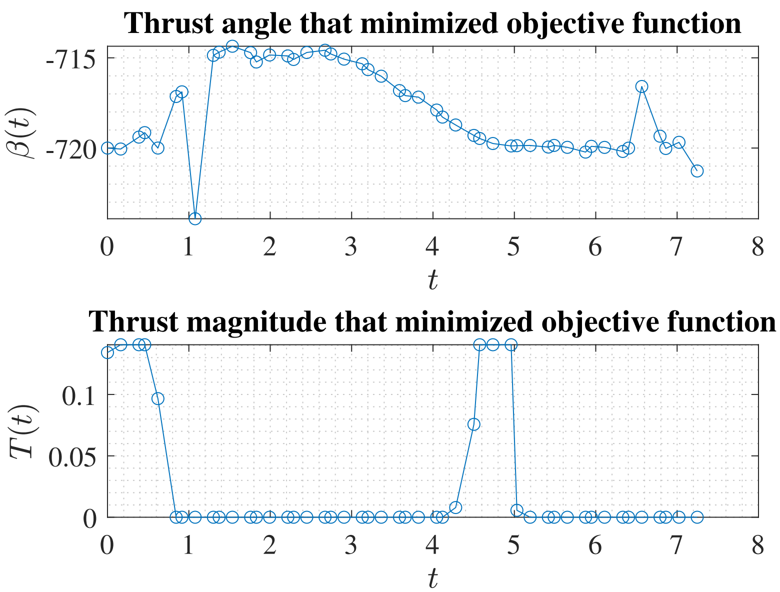

Figure 45: Control that maximized terminal mass ($N:3$, $K:16$)

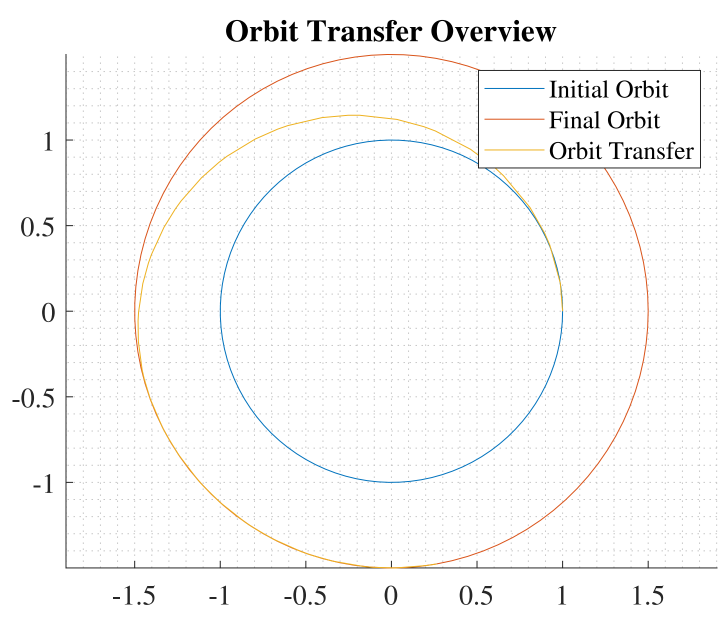

Figure 46: Trajectory from initial to final orbit ($N:3$, $K:16$)

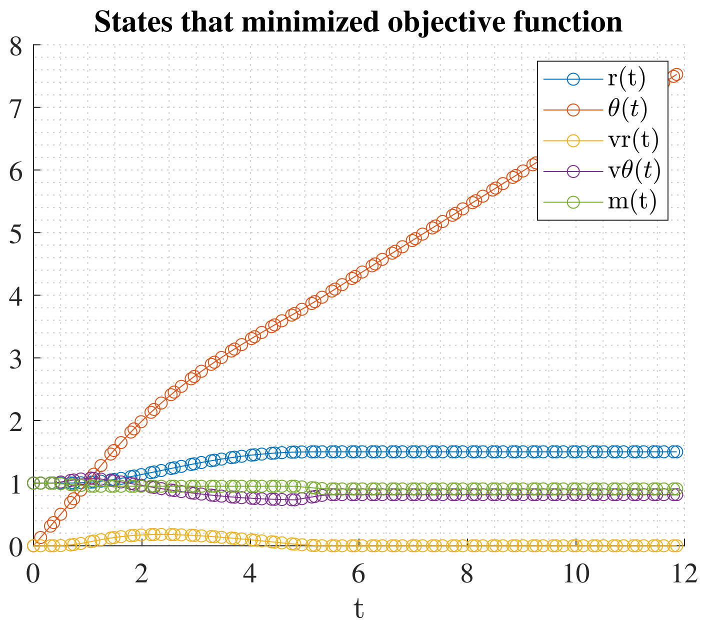

Figure 47: States for trajectory that maximized terminal mass ($N : 3$, $K : 32$)

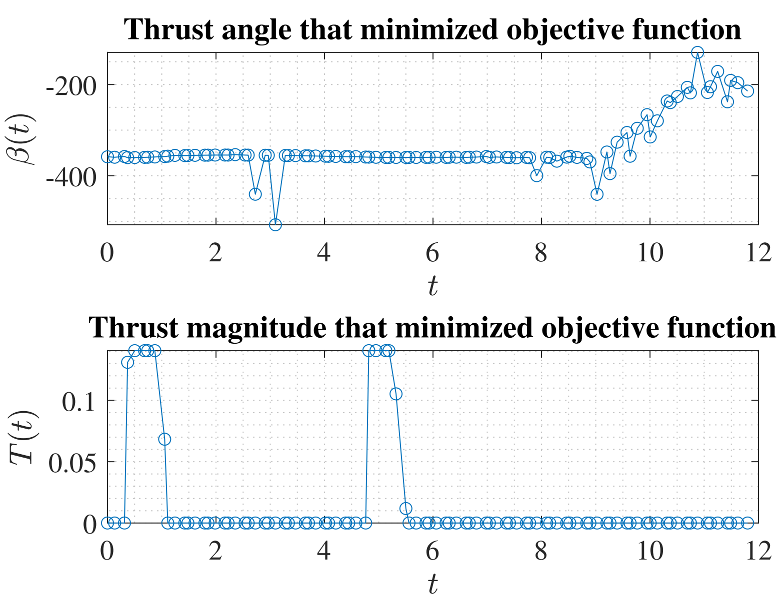

Figure 48: Control that maximized terminal mass ($N : 3$, $K : 32$)

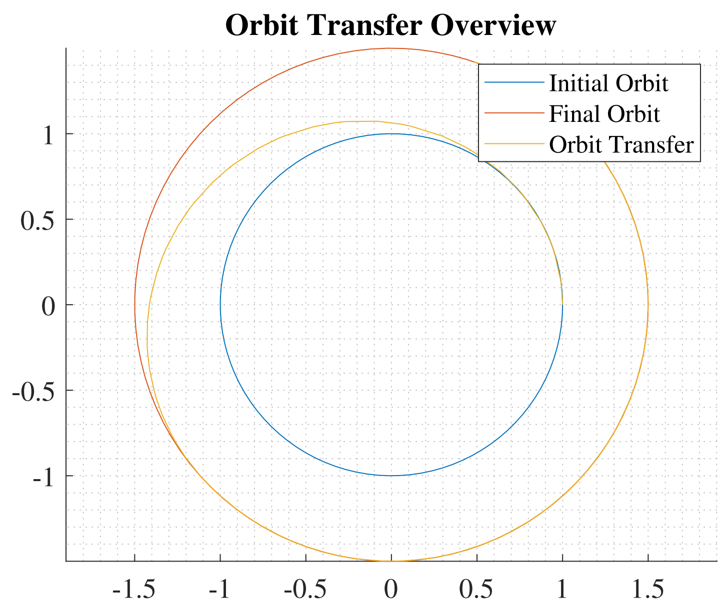

Figure 49: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

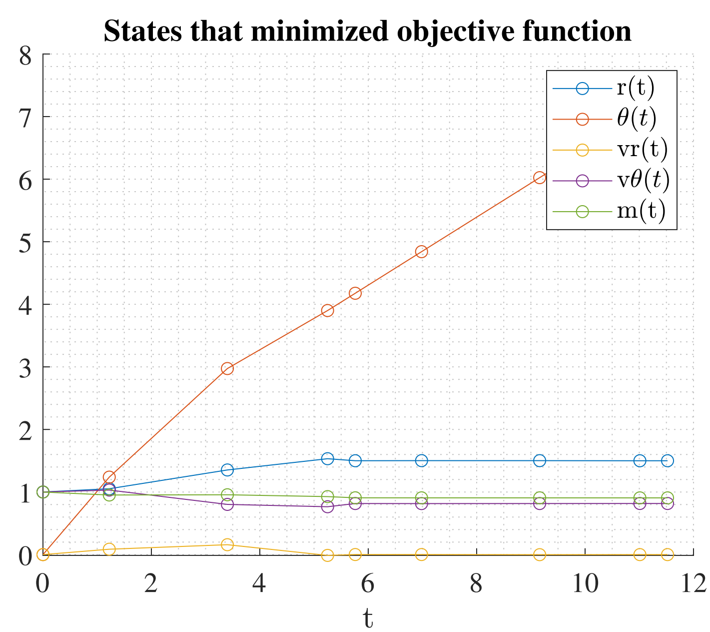

Figure 50: States for trajectory that maximized terminal mass ($N : 4$, $K : 2$)

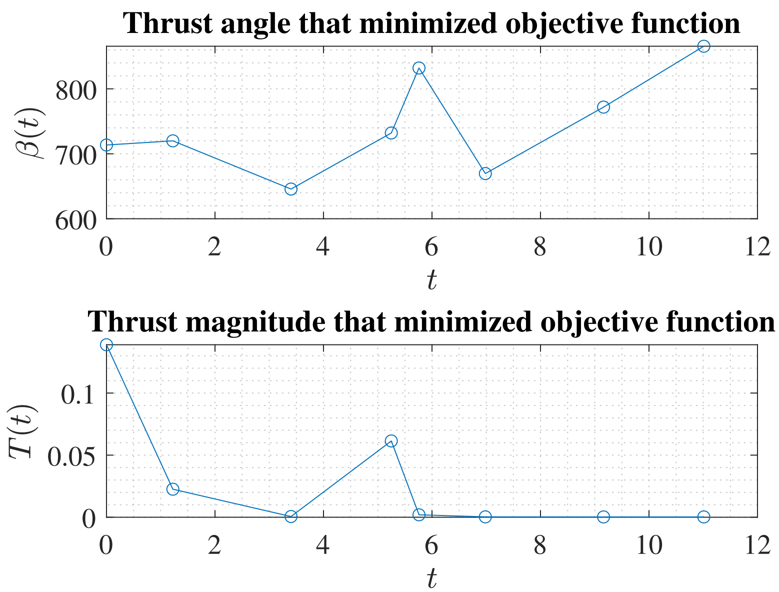

Figure 51: Control that maximized terminal mass ($N:4$, $K:2$)

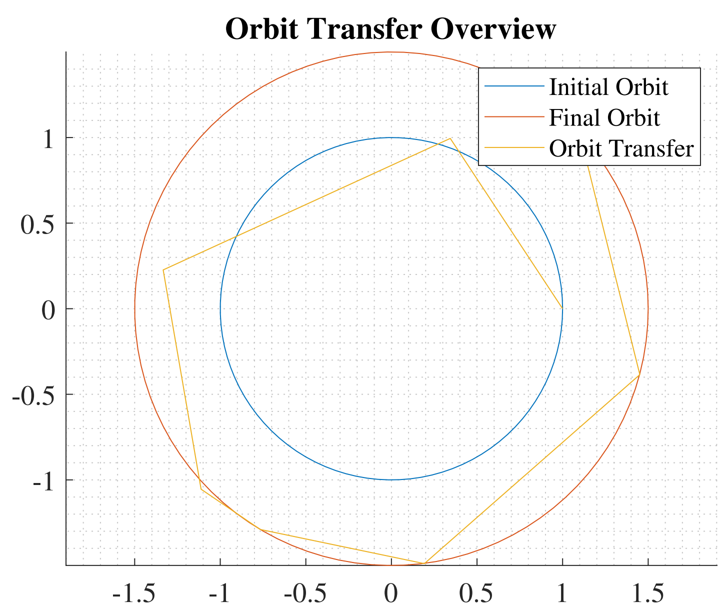

Figure 52: Trajectory from initial to final orbit ($N:4$, $K:2$)

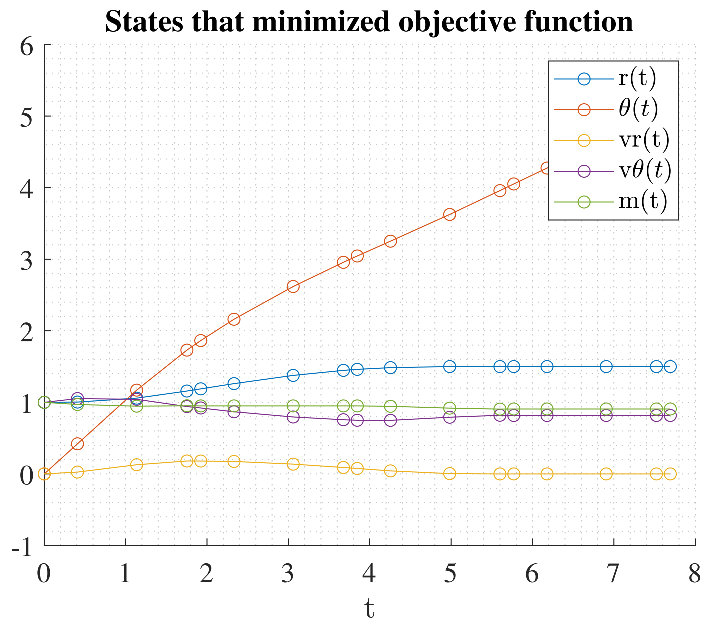

Figure 53: States for trajectory that maximized terminal mass ($N:4$, $K:4$)

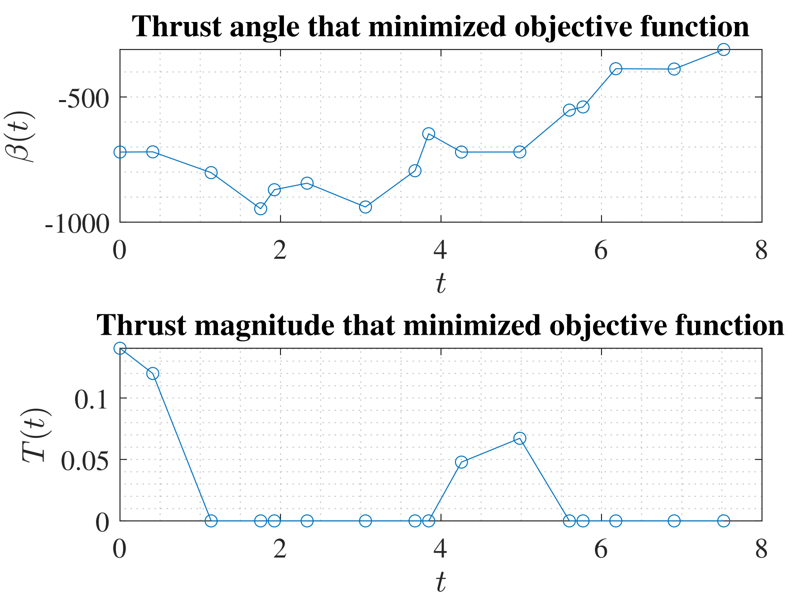

Figure 54: Control that maximized terminal mass ($N:4$, $K:4$)

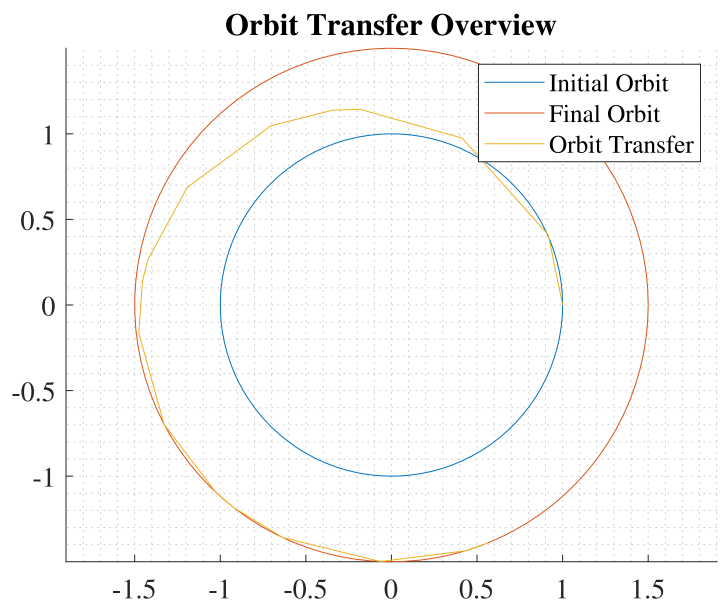

Figure 55: Trajectory from initial to final orbit ($N:4$, $K:4$)

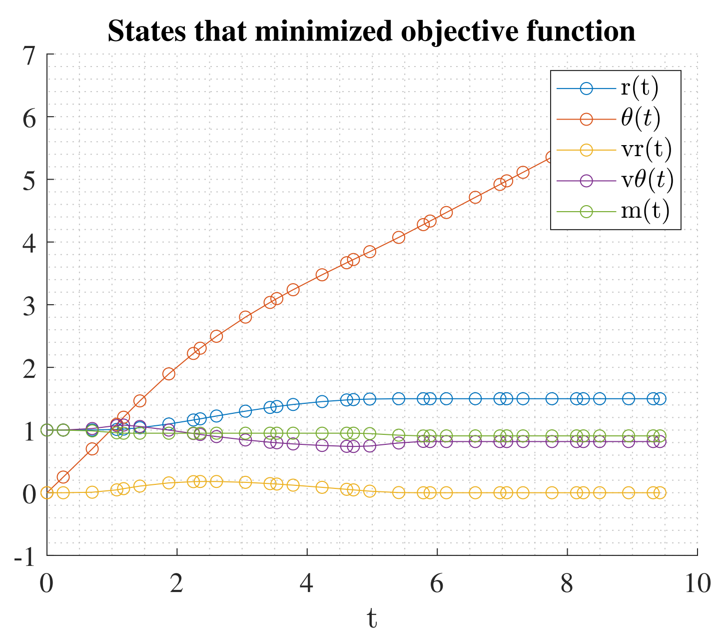

Figure 56: States for trajectory that maximized terminal mass ($N:4$, $K:8$)

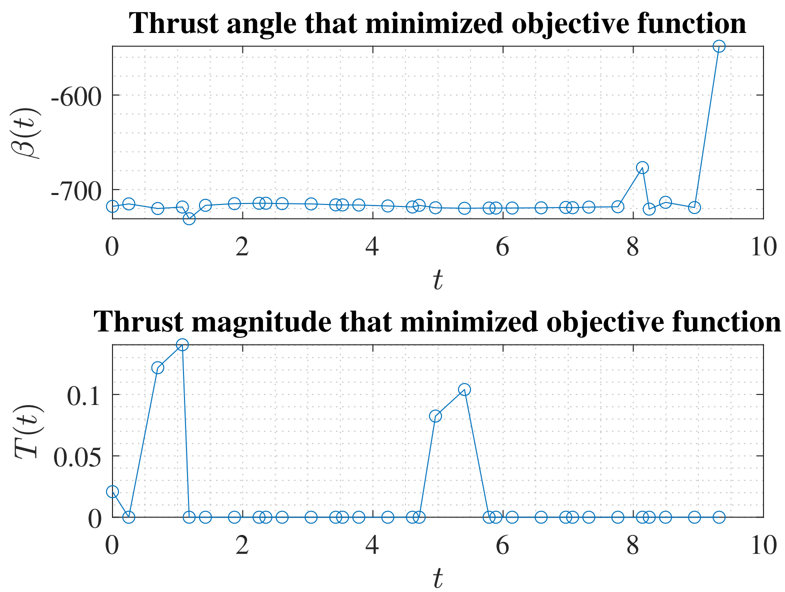

Figure 57: Control that maximized terminal mass ($N:4$, $K:8$)

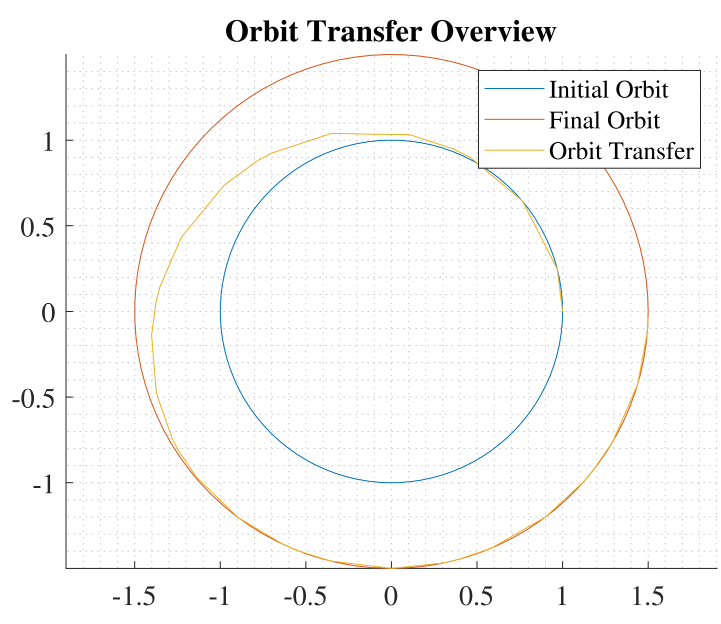

Figure 58: Trajectory from initial to final orbit ($N:4$, $K:8$)

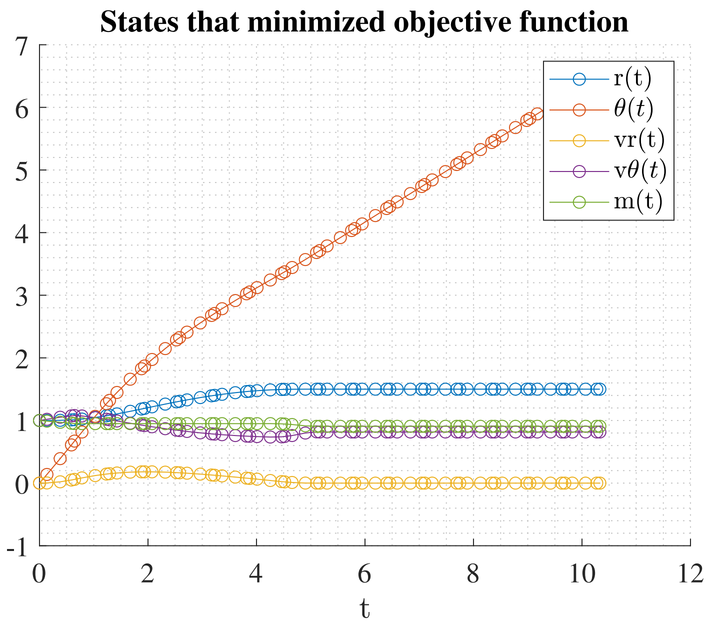

Figure 59: States for trajectory that maximized terminal mass ($N : 4$, $K : 16$)

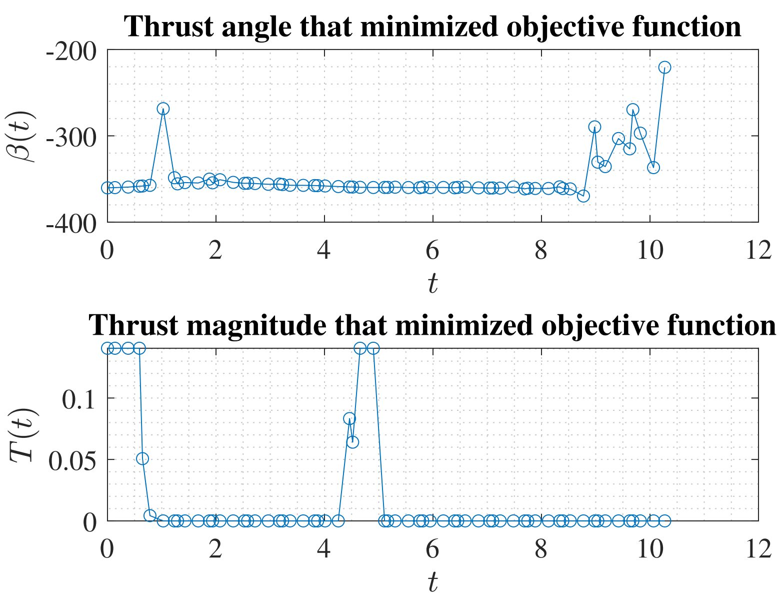

Figure 60: Control that maximized terminal mass ($N : 4$, $K : 16$)

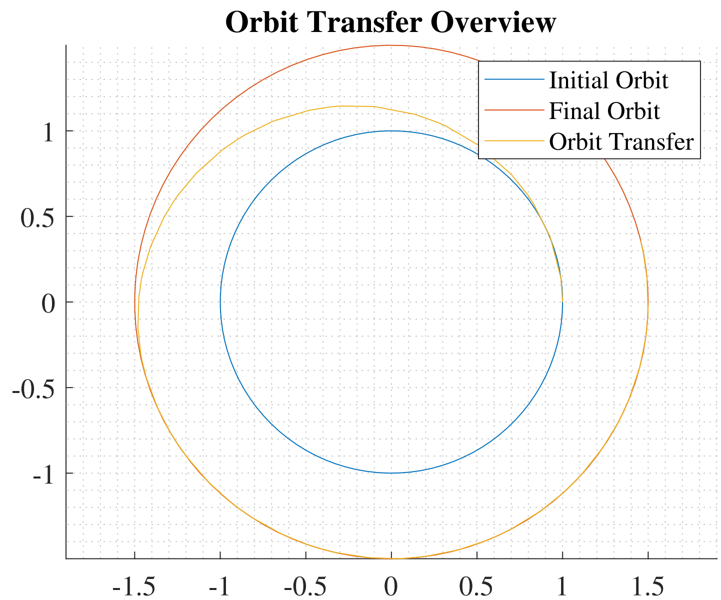

Figure 61: Trajectory from initial to final orbit ($N : 4$, $K : 16$)

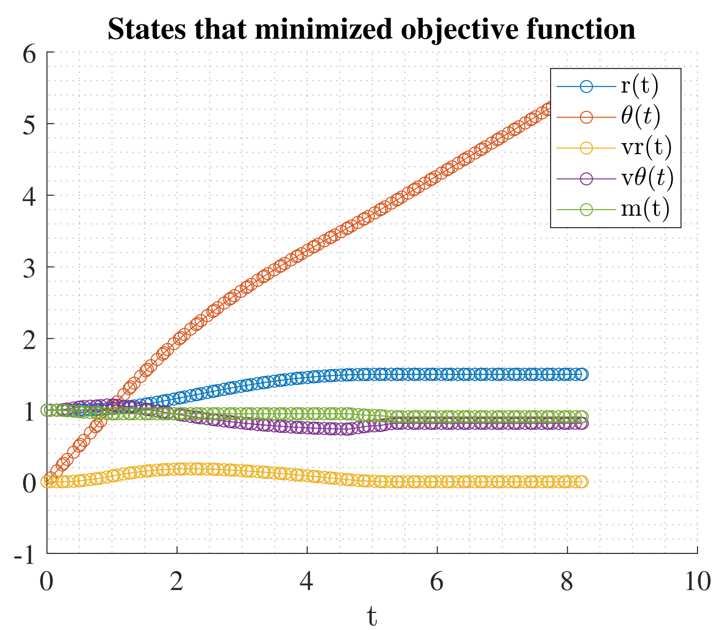

Figure 62: States for trajectory that maximized terminal mass ($N : 4$, $K : 32$)

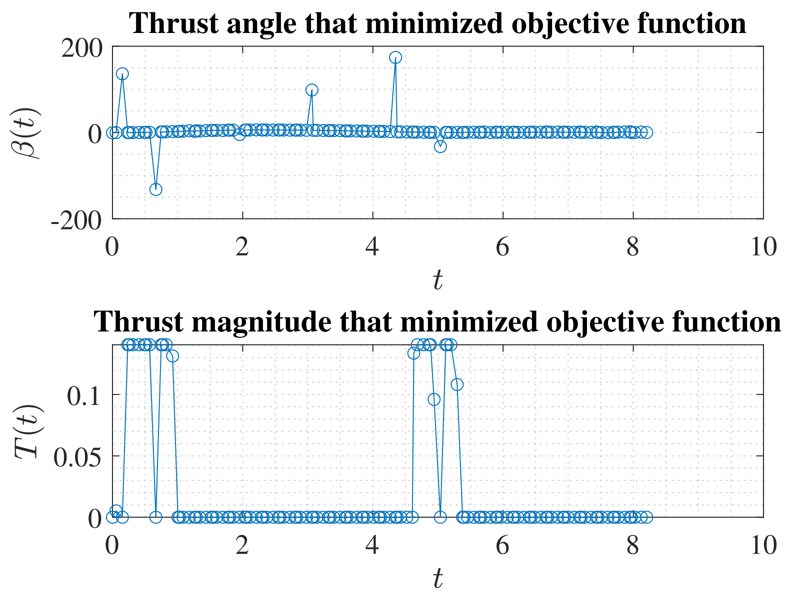

Figure 63: Control that maximized terminal mass (N : 4 , K : 32)

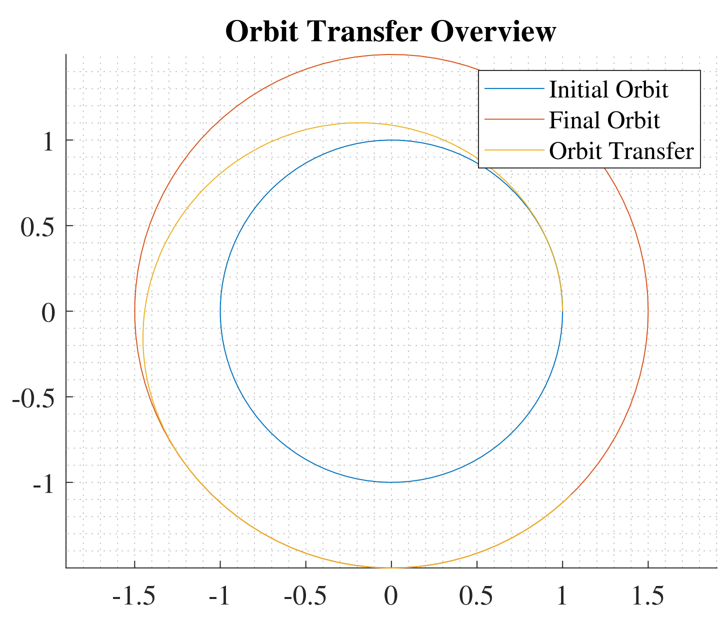

Figure 64: Trajectory from initial to final orbit (N : 4 , K : 32)

4.3 Minimize Terminal Time with Constrained Control

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	47	0.225	3.2444	0.75569	0
3	4	69	0.446	3.2457	0.75559	0
3	8	77	0.369	3.2577	0.7581	0
3	16	119	0.537	3.247	0.7555	0
3	32	143	0.738	3.2623	0.75775	0
4	2	69	0.314	3.2456	0.7556	0
4	4	81	0.363	3.27	0.75761	0
4	8	102	0.464	3.2584	0.75655	0
4	16	143	0.677	3.247	0.7555	0
4	32	151	0.889	3.2482	0.75589	1

Table 3: Results for minimizing t_f with constrained control

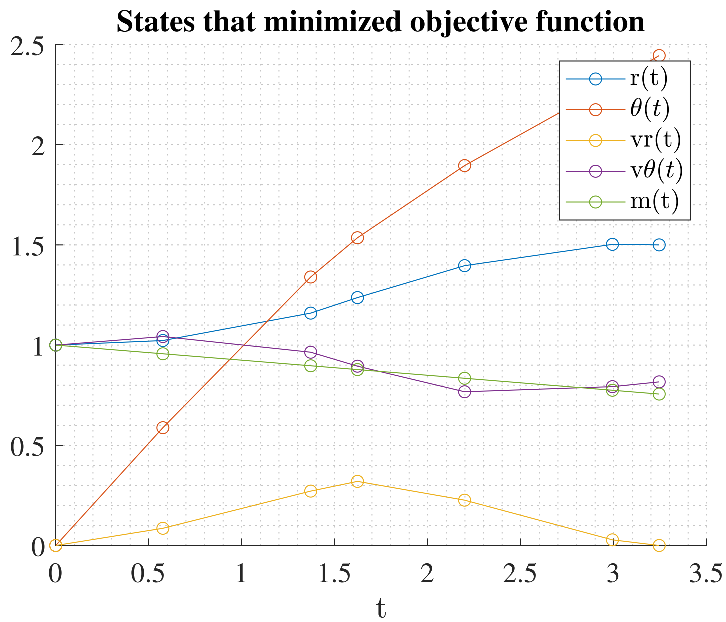

Figure 65: States for trajectory that minimized terminal time ($N:3$, $K:2$)

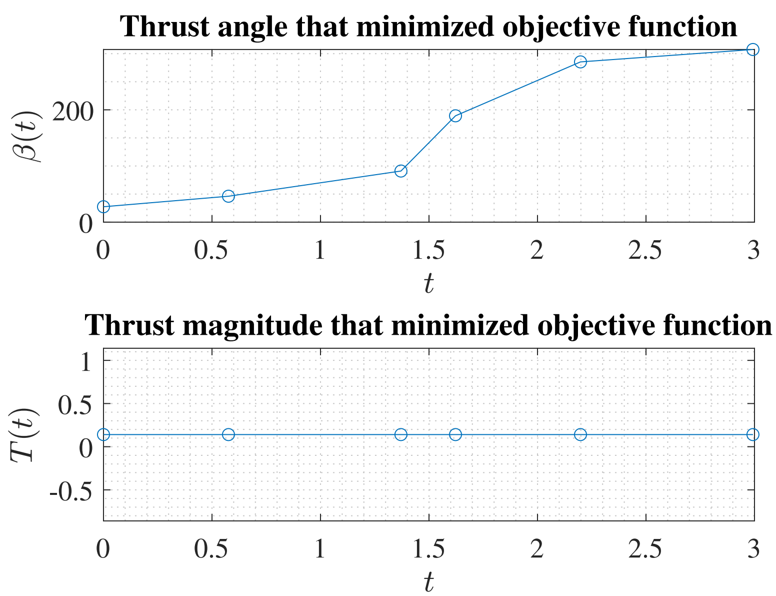

Figure 66: Control that minimized terminal time ($N:3$, $K:2$)

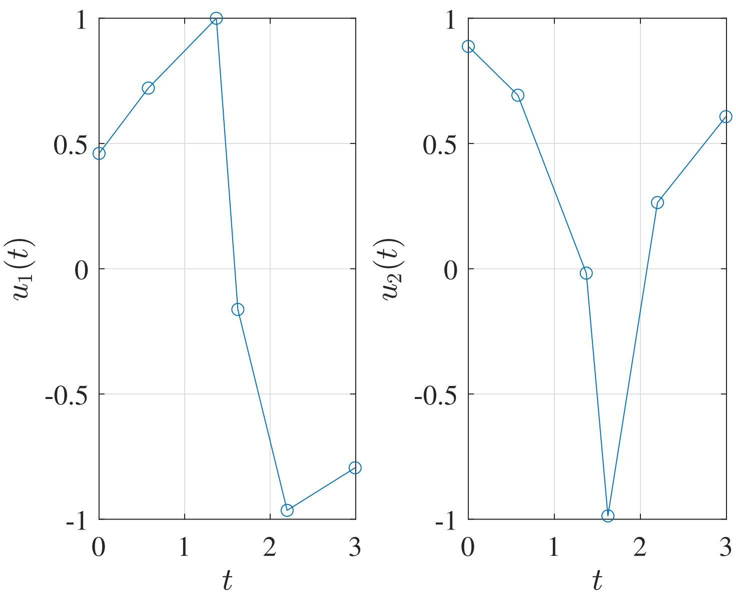

Figure 67: Path constrained control that minimized terminal time (N : 3 , K : 2)

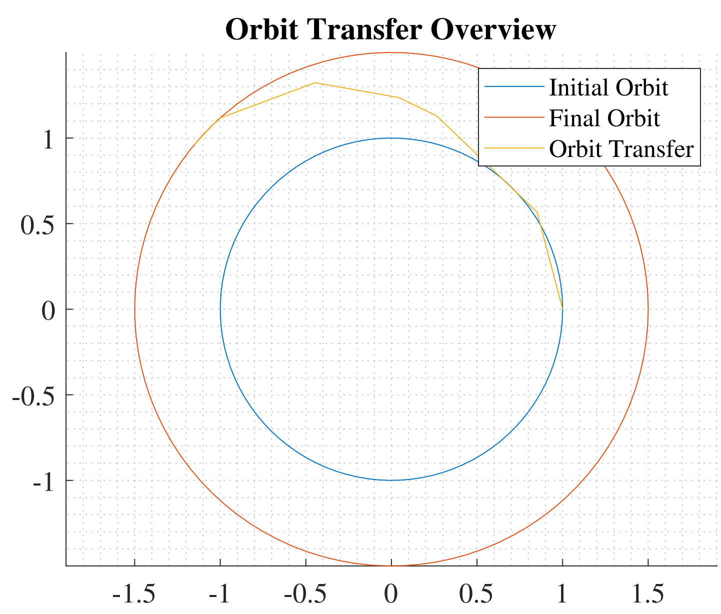

Figure 68: Trajectory from initial to final orbit (N : 3 , K : 2)

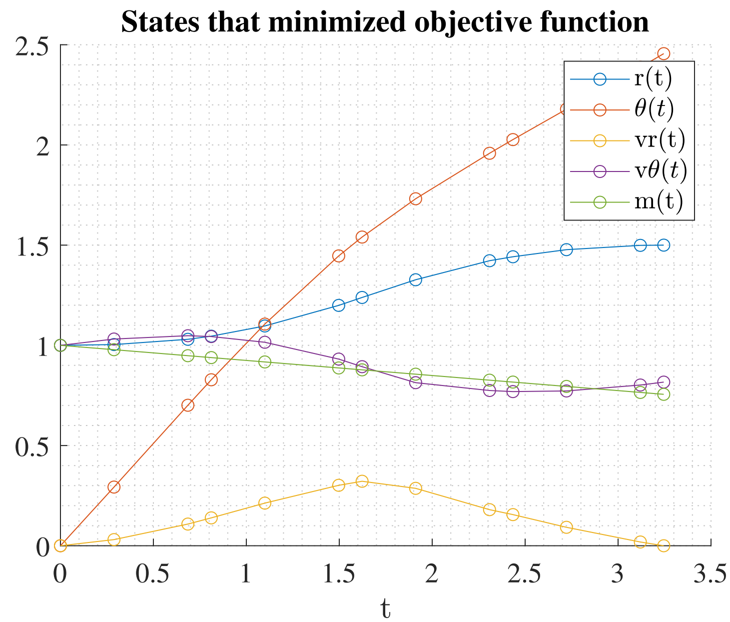

Figure 69: States for trajectory that minimized terminal time ($N:3$, $K:4$)

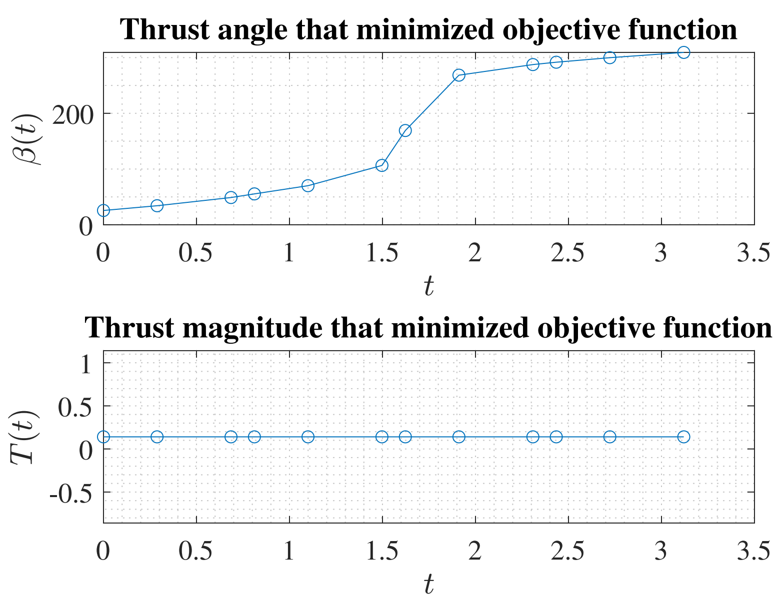

Figure 70: Control that minimized terminal time ($N:3$, $K:4$)

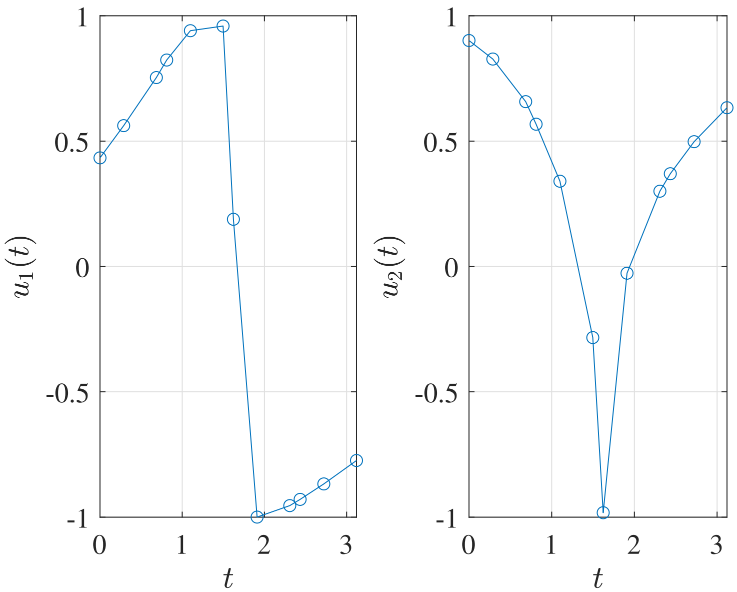

Figure 71: Path constrained control that minimized terminal time ($N : 3$, $K : 4$)

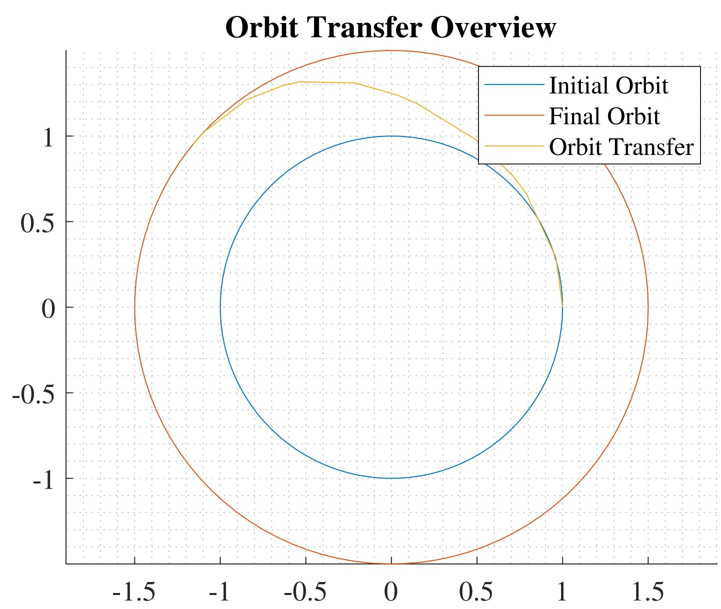

Figure 72: Trajectory from initial to final orbit ($N : 3$, $K : 4$)

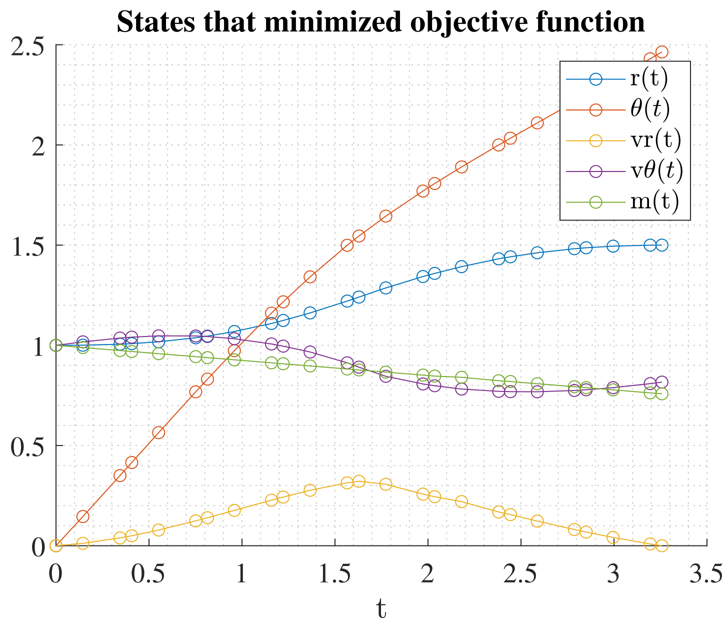

Figure 73: States for trajectory that minimized terminal time ($N:3$, $K:8$)

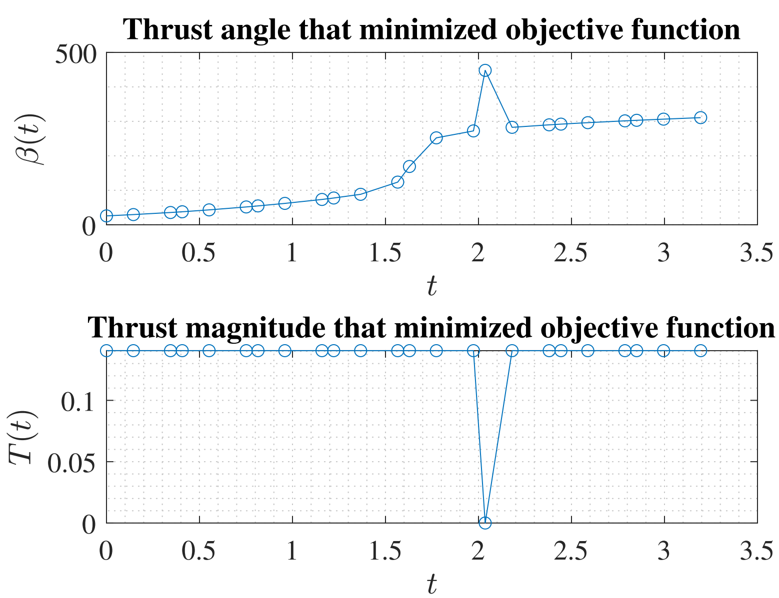

Figure 74: Control that minimized terminal time ($N:3$, $K:8$)

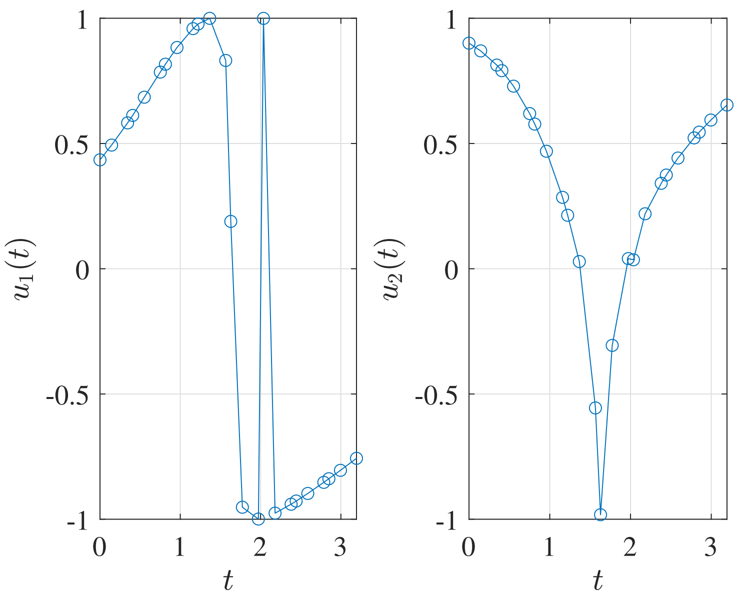

Figure 75: Path constrained control that minimized terminal time ($N : 3$, $K : 8$)

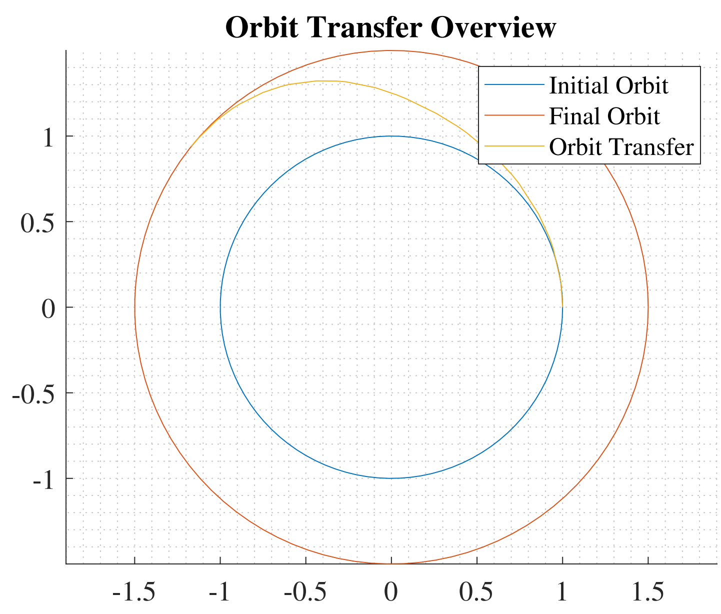

Figure 76: Trajectory from initial to final orbit ($N : 3$, $K : 8$)

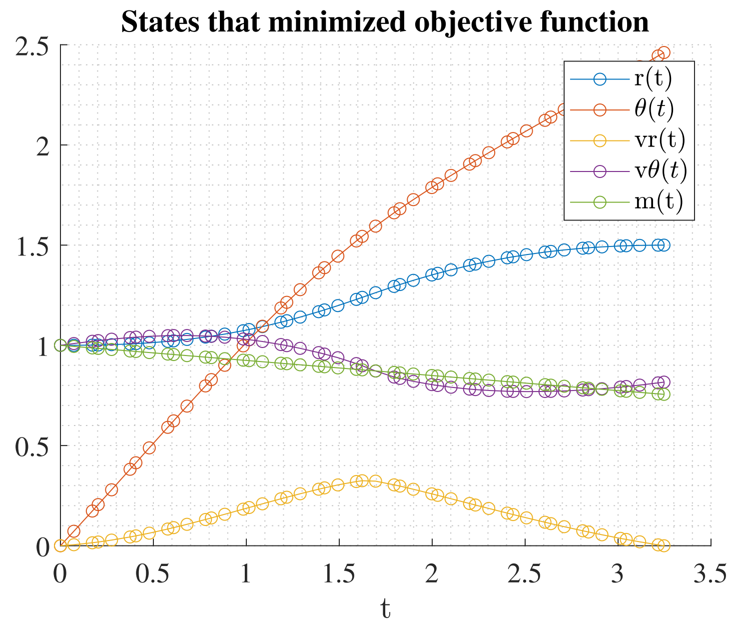

Figure 77: States for trajectory that minimized terminal time $(N : 3 , K : 16)$

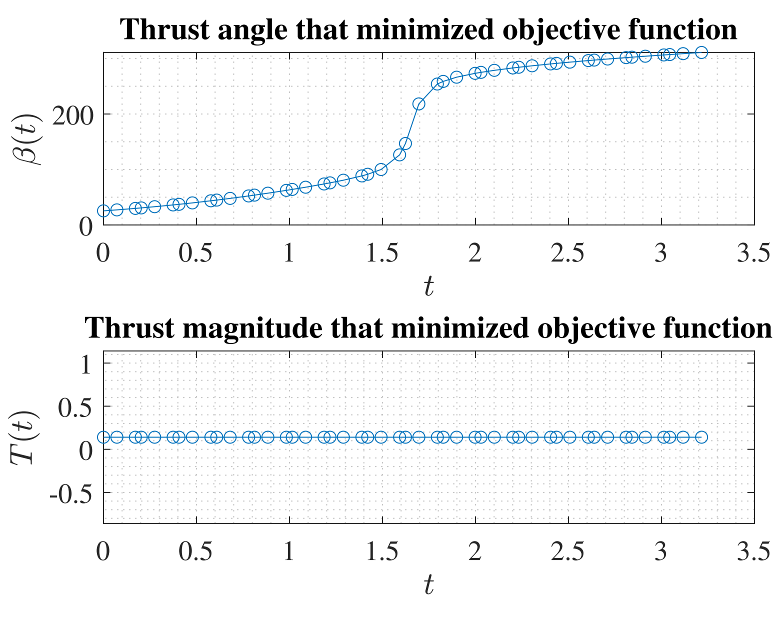

Figure 78: Control that minimized terminal time $(N : 3 , K : 16)$

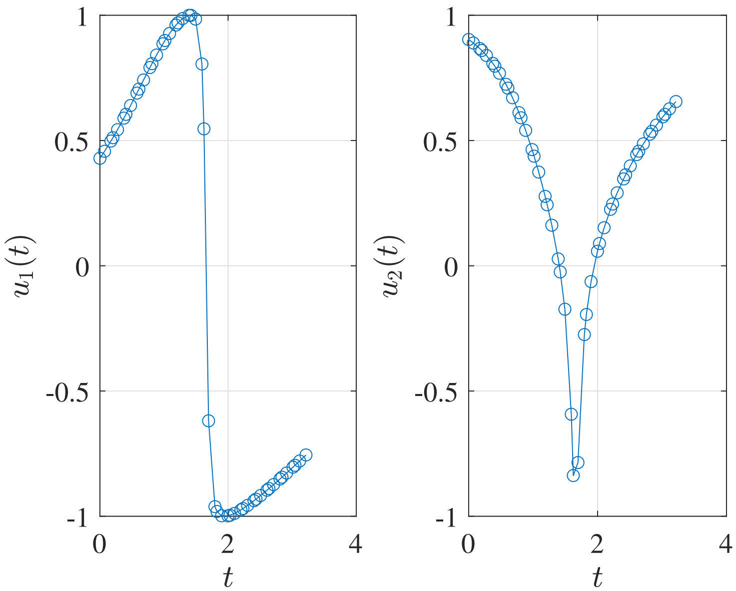

Figure 79: Path constrained control that minimized terminal time ($N : 3$, K : 16)

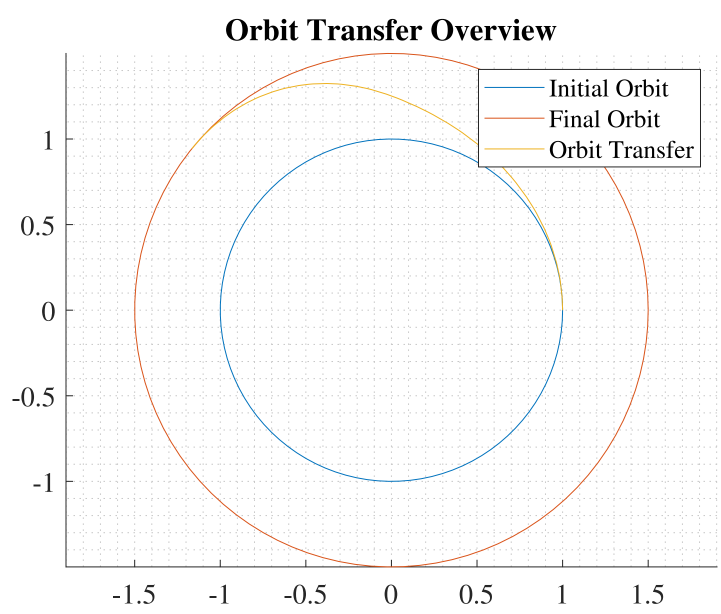

Figure 80: Trajectory from initial to final orbit ($N : 3$, $K : 16$)

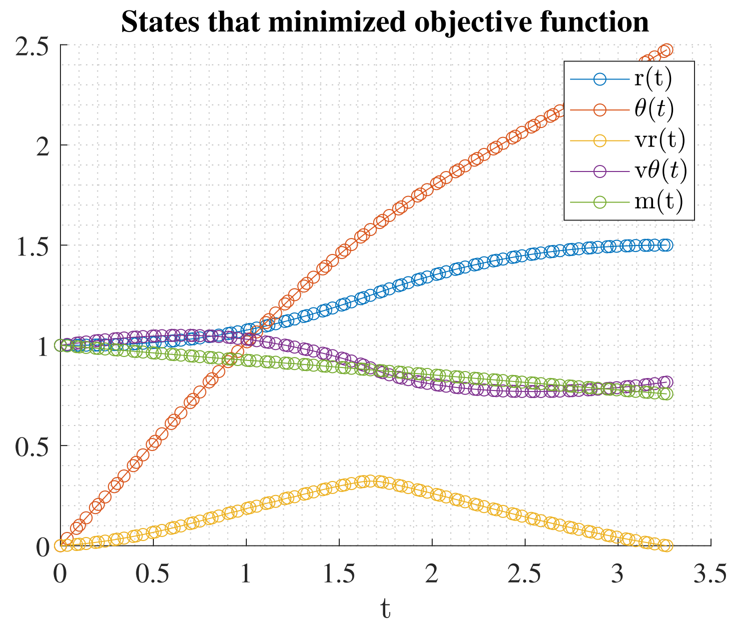

Figure 81: States for trajectory that minimized terminal time ($N:3$, $K:32$)

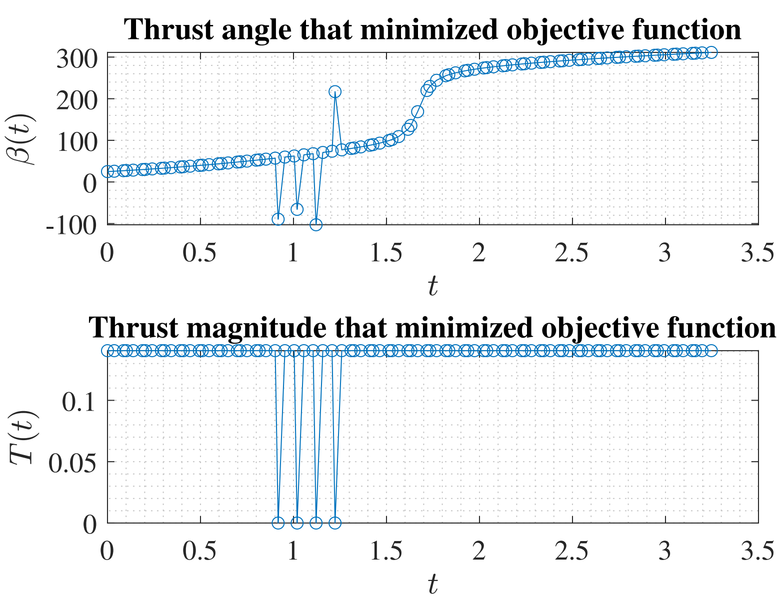

Figure 82: Control that minimized terminal time ($N:3$, $K:32$)

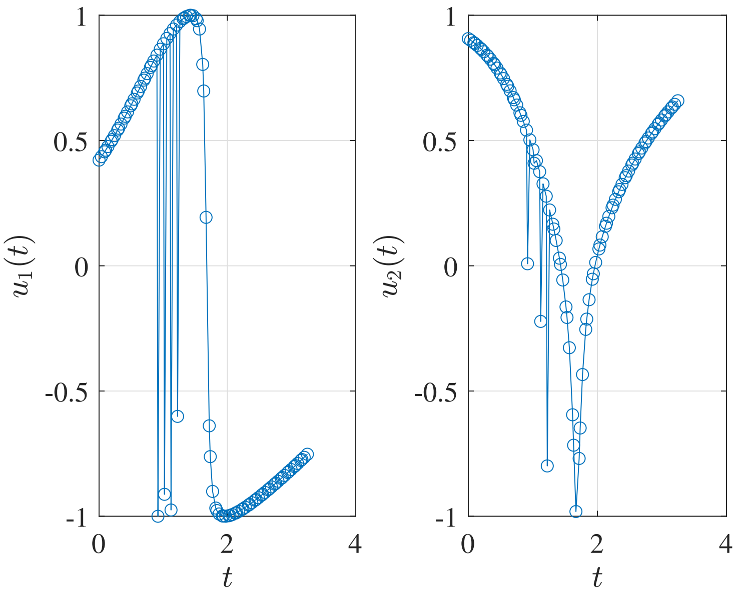

Figure 83: Path constrained control that minimized terminal time ($N : 3$, K : 32)

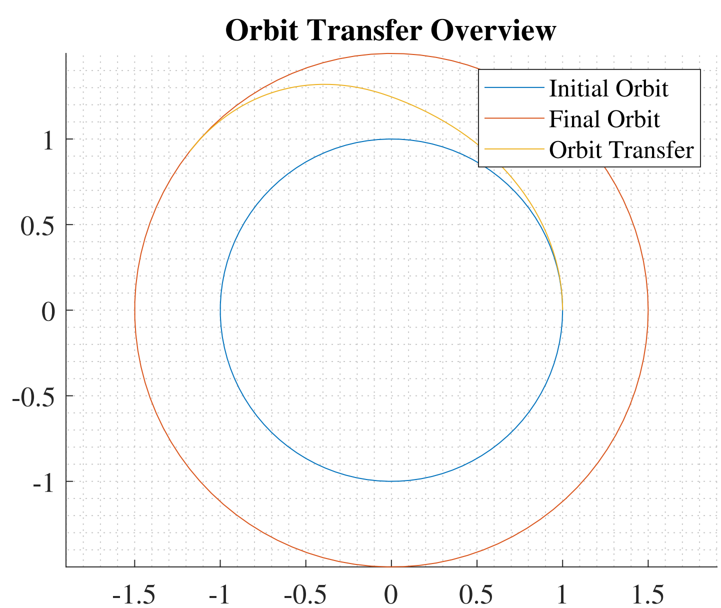

Figure 84: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

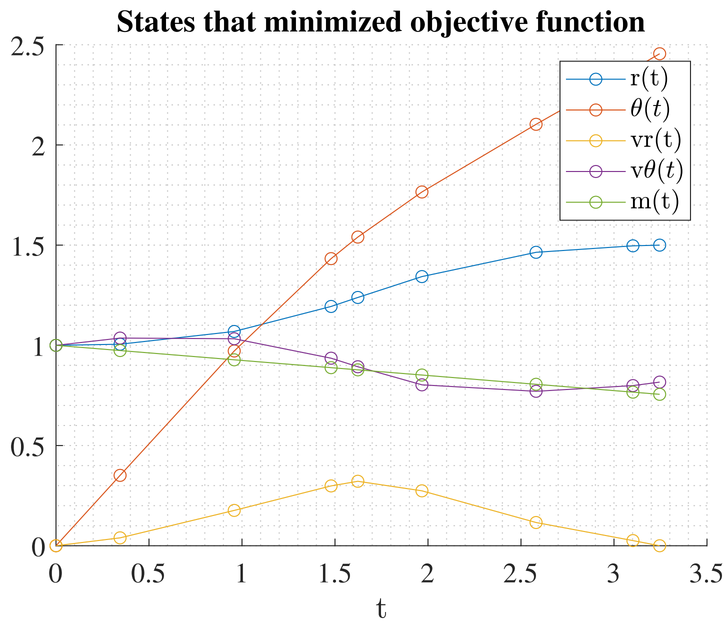

Figure 85: States for trajectory that minimized terminal time ($N : 4$, $K : 2$)

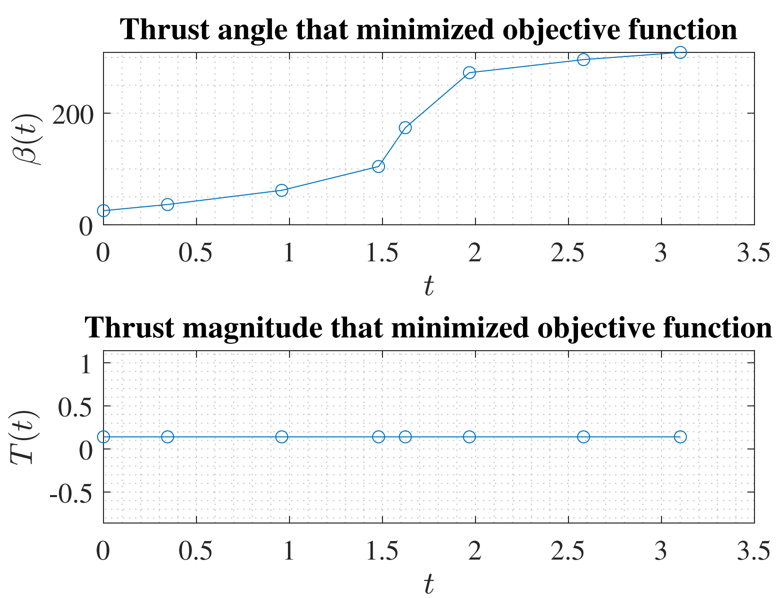

Figure 86: Control that minimized terminal time ($N : 4$, $K : 2$)

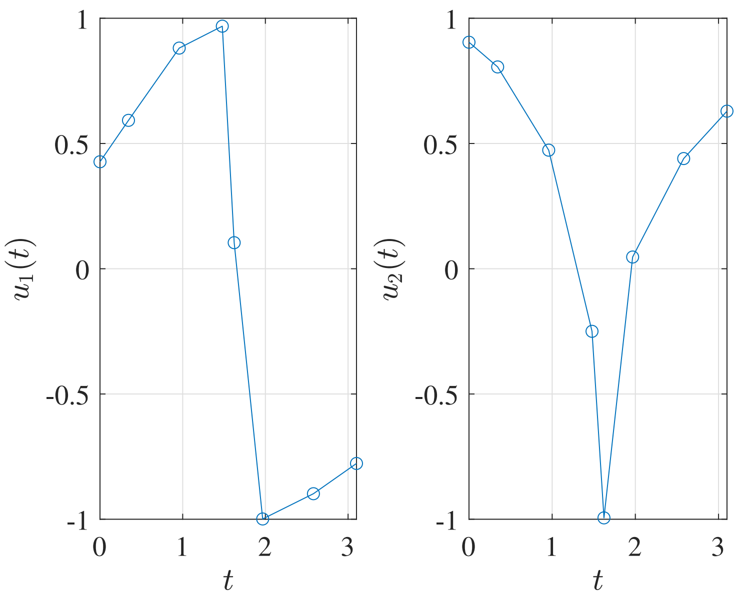

Figure 87: Path constrained control that minimized terminal time ($N : 4$, $K : 2$)

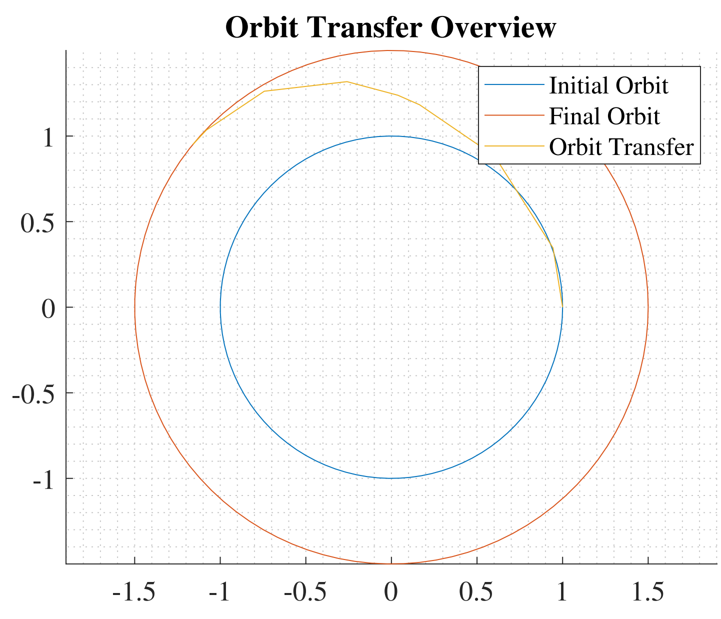

Figure 88: Trajectory from initial to final orbit ($N : 4$, $K : 2$)

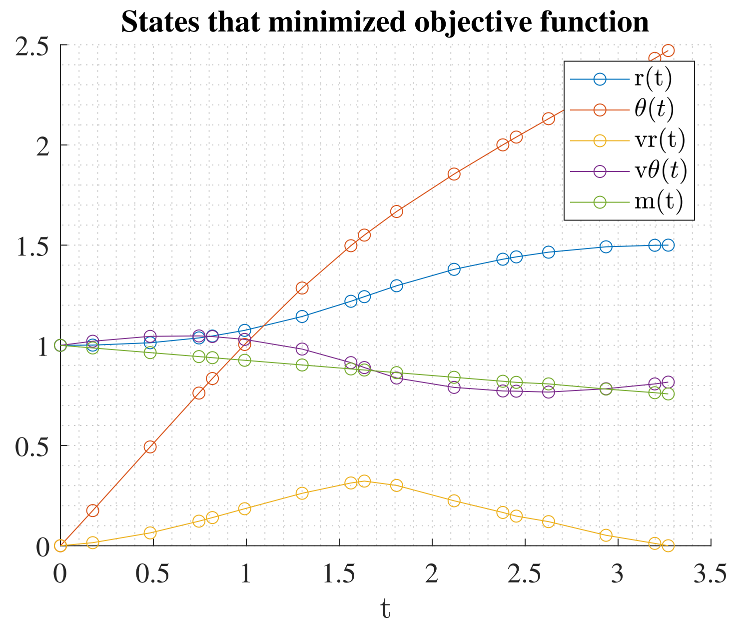

Figure 89: States for trajectory that minimized terminal time ($N:4$, $K:4$)

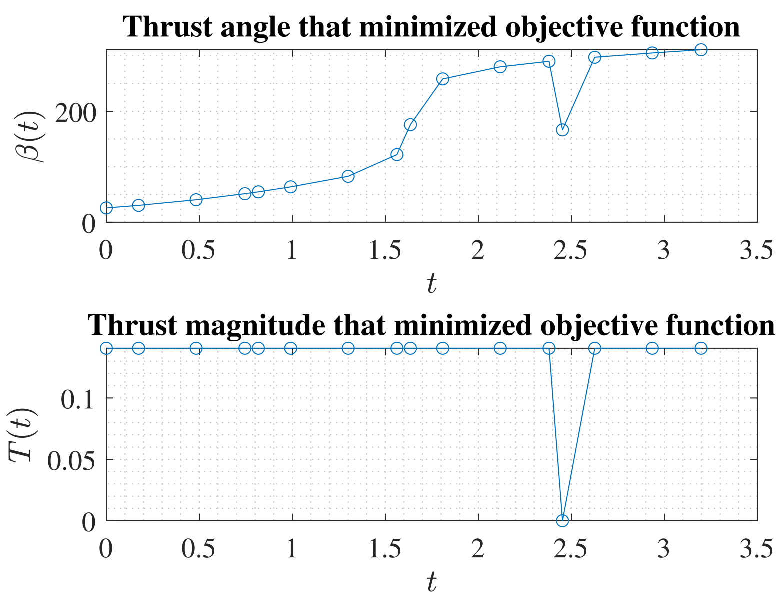

Figure 90: Control that minimized terminal time ($N:4$, $K:4$)

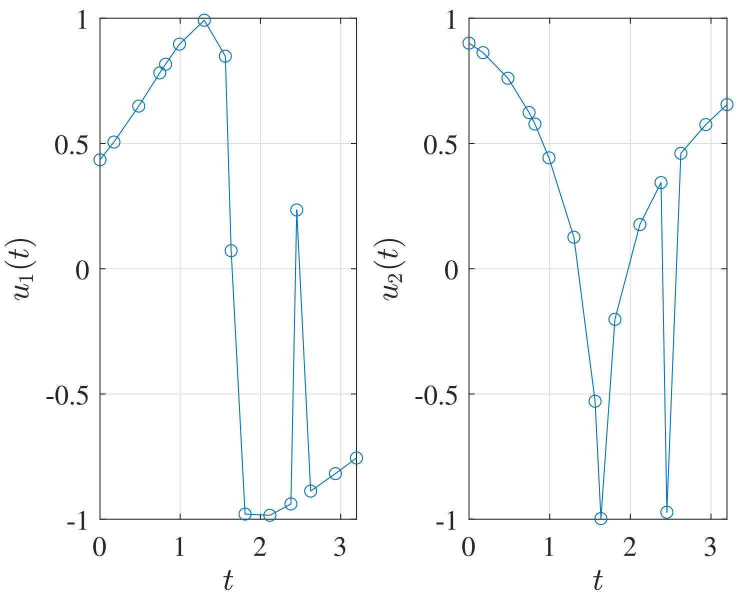

Figure 91: Path constrained control that minimized terminal time ($N : 4$, $K : 4$)

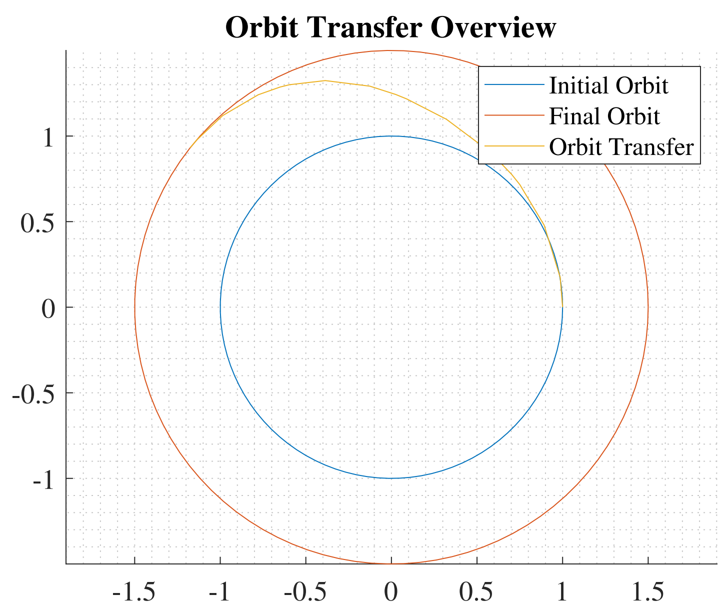

Figure 92: Trajectory from initial to final orbit ($N : 4$, $K : 4$)

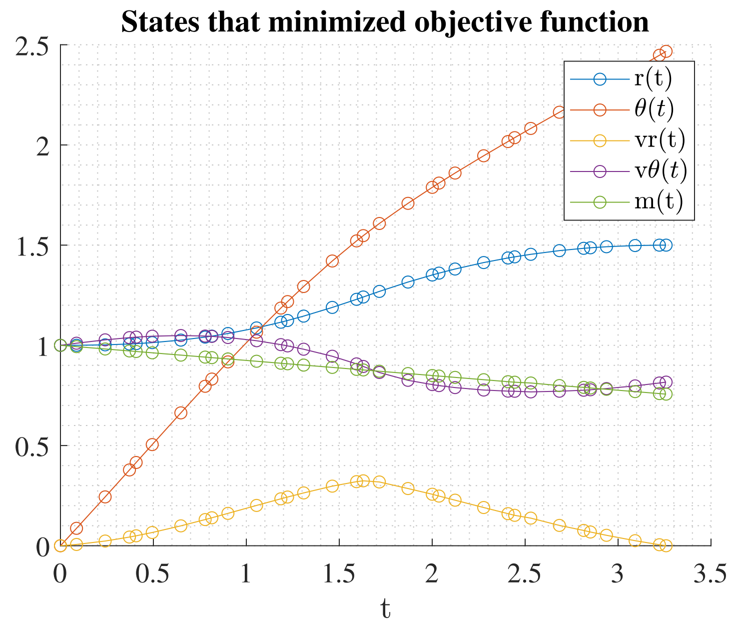

Figure 93: States for trajectory that minimized terminal time ($N:4$, $K:8$)

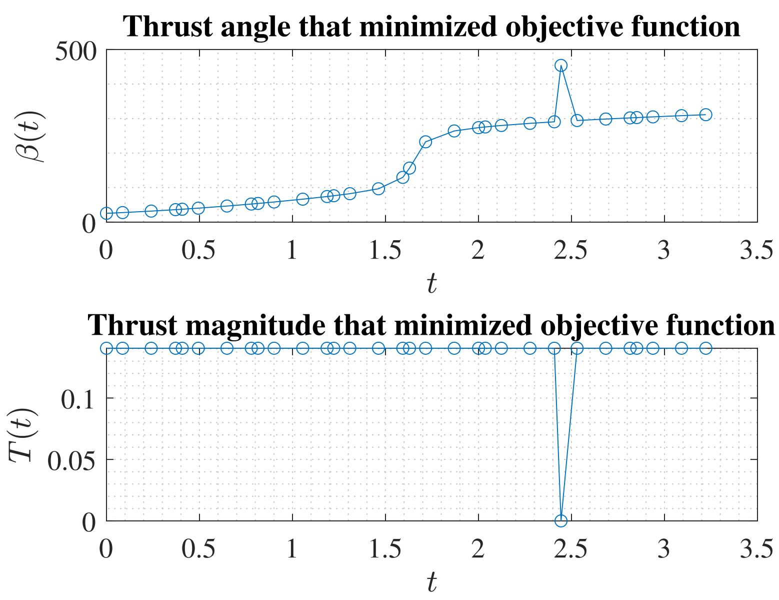

Figure 94: Control that minimized terminal time ($N:4$, $K:8$)

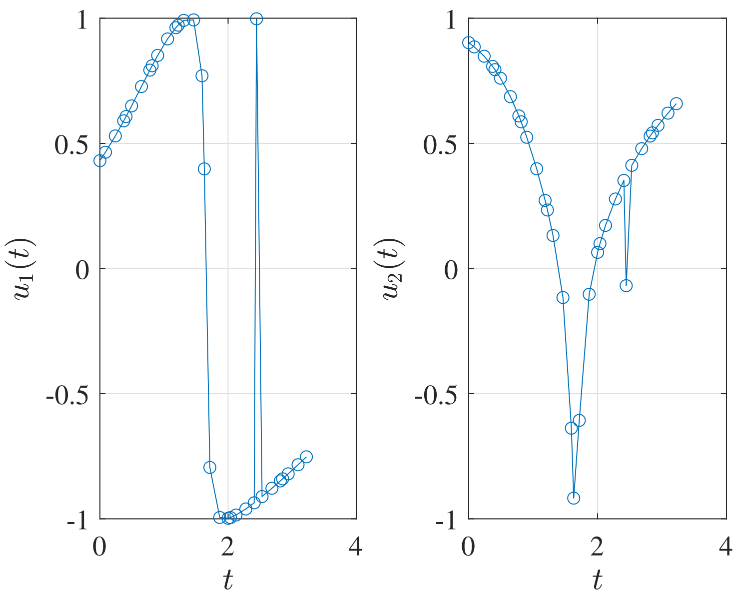

Figure 95: Path constrained control that minimized terminal time $(N : 4\ , K : 8)$

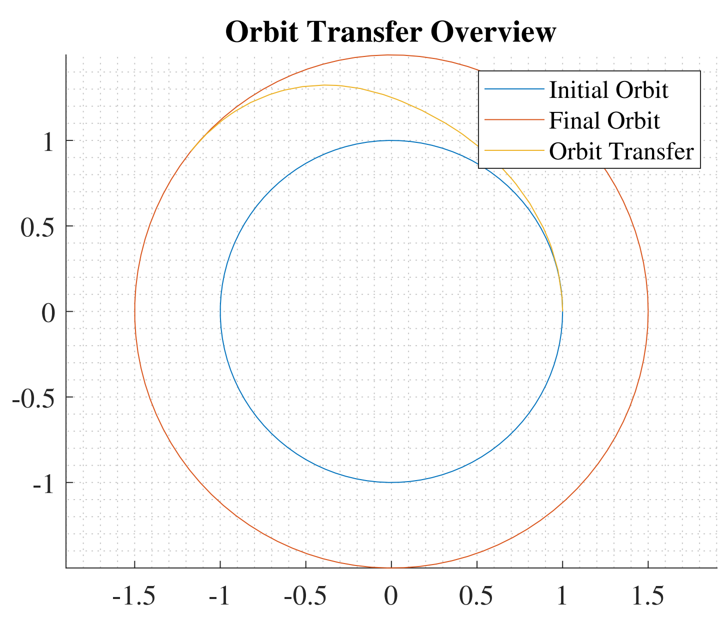

Figure 96: Trajectory from initial to final orbit $(N : 4\ , K : 8)$

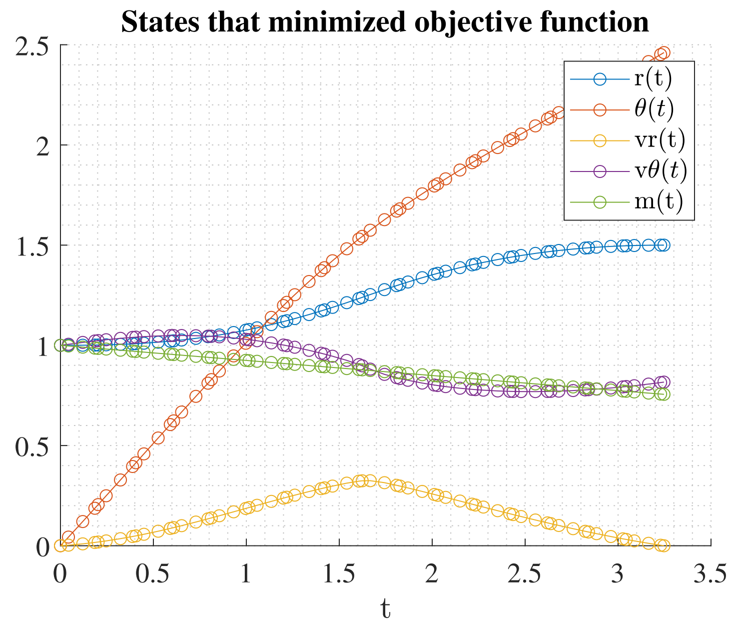

Figure 97: States for trajectory that minimized terminal time ($N : 4$, $K : 16$)

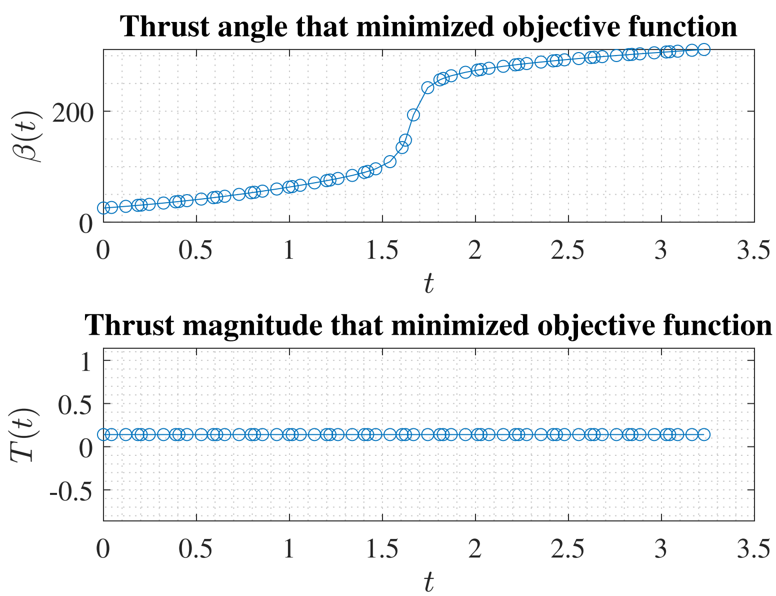

Figure 98: Control that minimized terminal time ($N : 4$, $K : 16$)

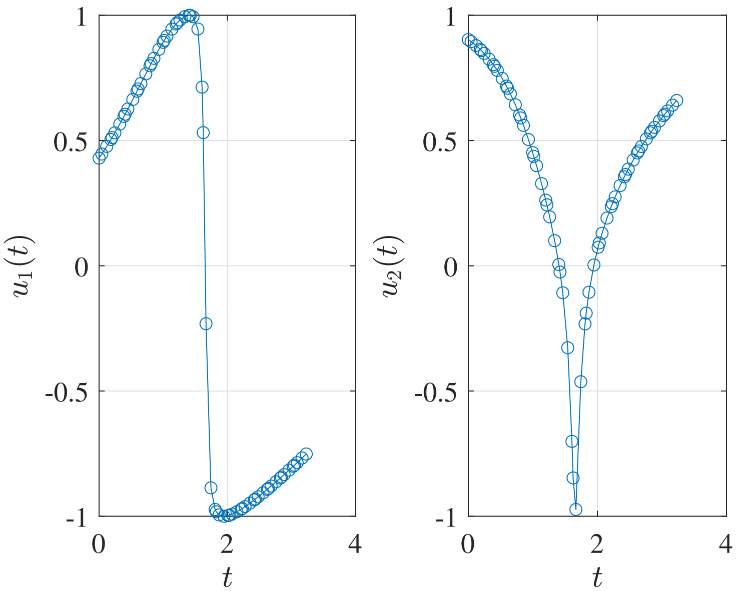

Figure 99: Path constrained control that minimized terminal time ($N : 4$, K : 16)

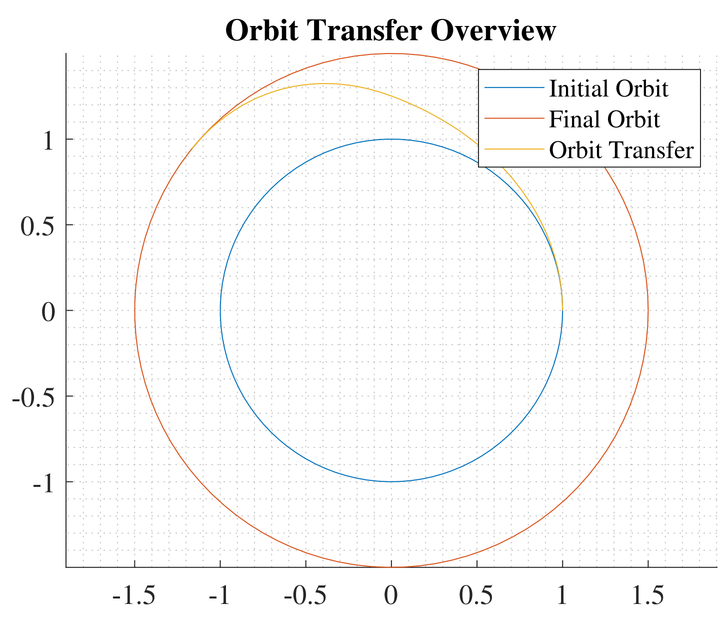

Figure 100: Trajectory from initial to final orbit ($N : 4$, $K : 16$)

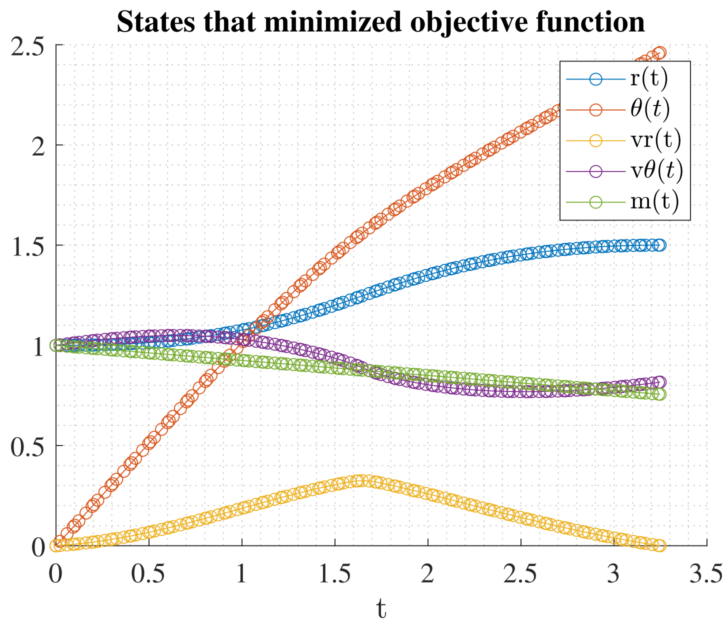

Figure 101: States for trajectory that minimized terminal time ($N : 4$, $K : 32$)

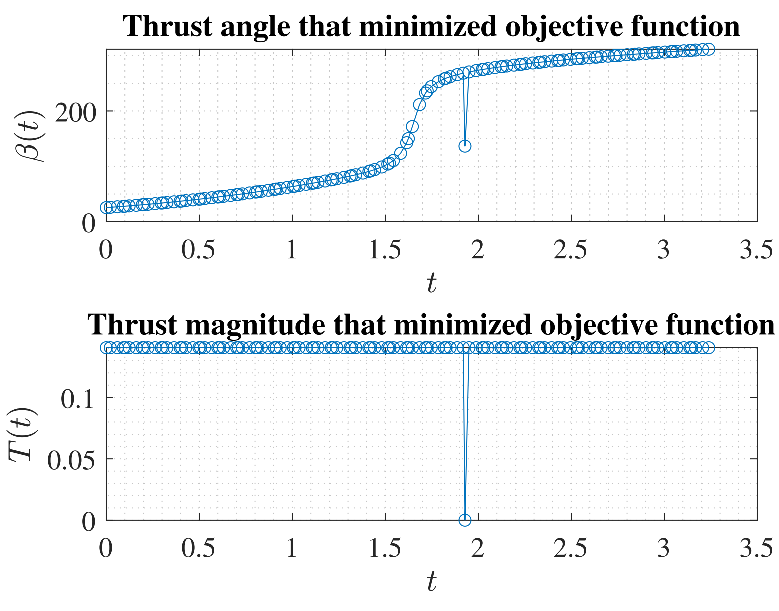

Figure 102: Control that minimized terminal time ($N : 4$, $K : 32$)

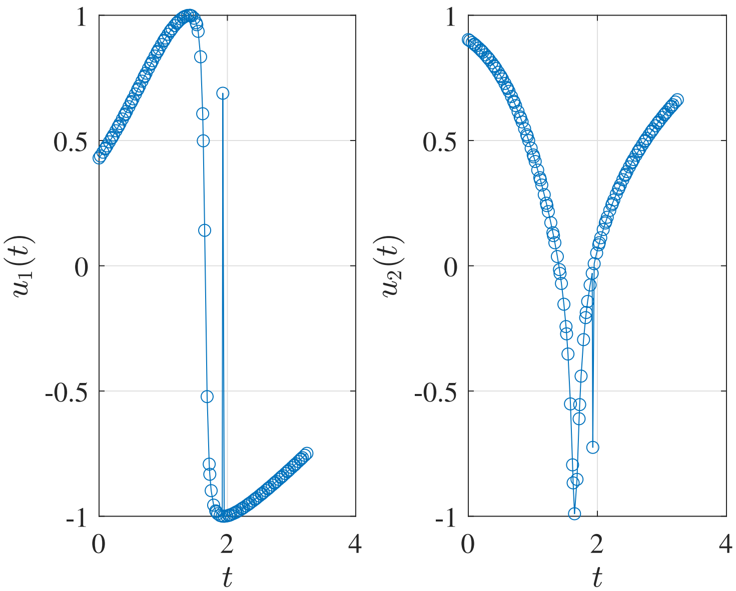

Figure 103: Path constrained control that minimized terminal time ($N : 4$, K : 32)

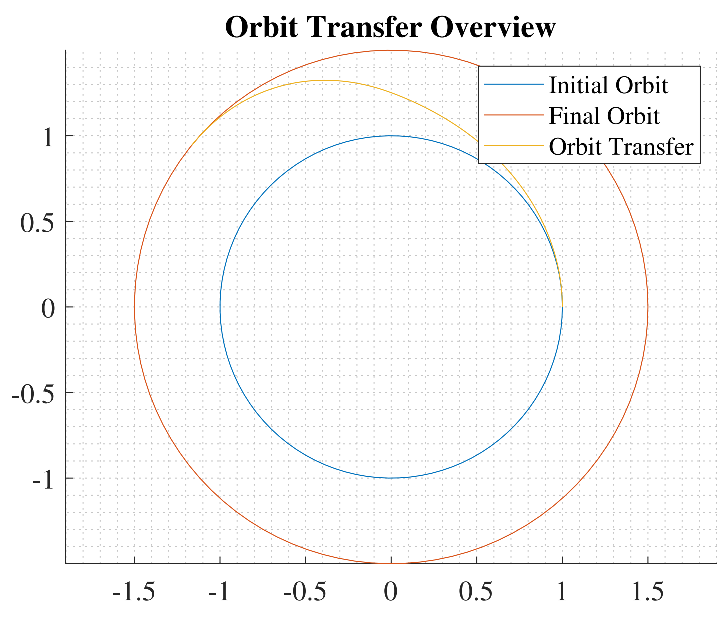

Figure 104: Trajectory from initial to final orbit ($N : 4$, $K : 32$)

4.4 Maximize Terminal Mass with Constrained Control

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	110	0.417	15.0512	0.90547	-2
3	4	2185	7.269	14.4119	0.9095	0
3	8	728	2.557	6.9851	0.90719	0
3	16	96	0.465	7.9129	0.9072	0
3	32	102	0.559	6.689	0.90721	1
4	2	531	1.765	12.3575	0.91207	0
4	4	415	1.465	6.1534	0.90718	0
4	8	397	1.511	7.0005	0.90721	0
4	16	141	0.688	6.3843	0.90721	1
4	32	1016	5.404	7.0926	0.90721	1

Table 4: Results for maximizing m_f with constrained control

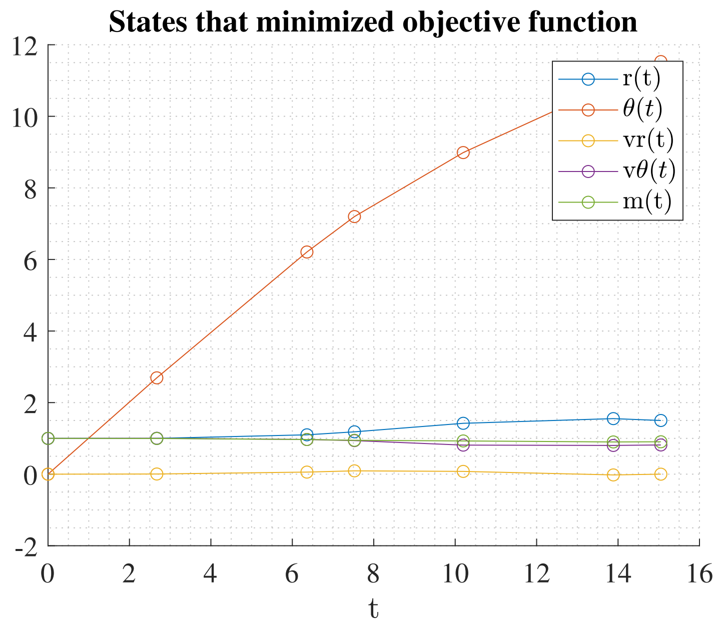

Figure 105: States for trajectory that maximized terminal mass $(N : 3\ , K : 2)$

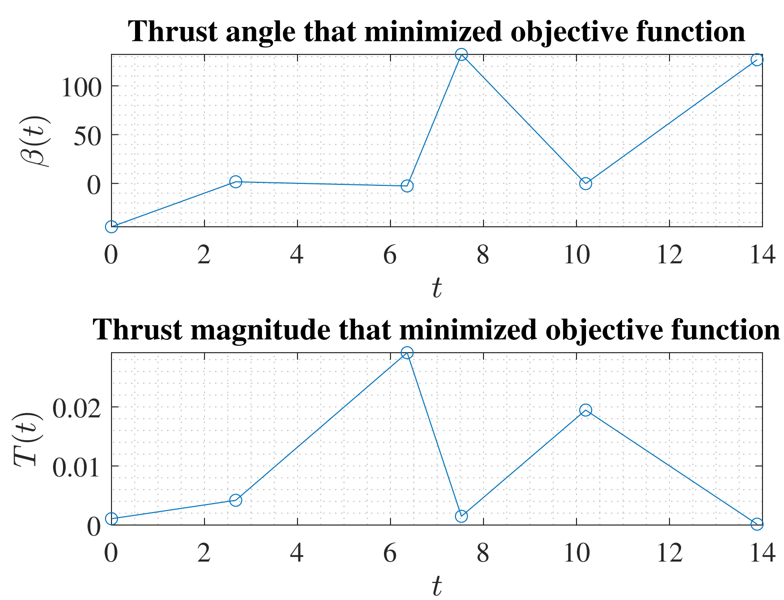

Figure 106: Control that maximized terminal mass $(N : 3\ , K : 2)$

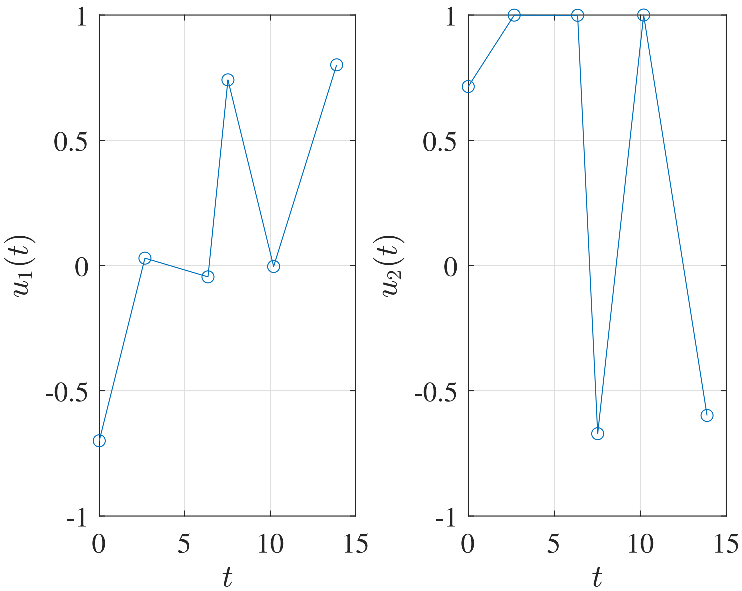

Figure 107: Path constrained control that maximized terminal mass ($N:3$, $K:2$)

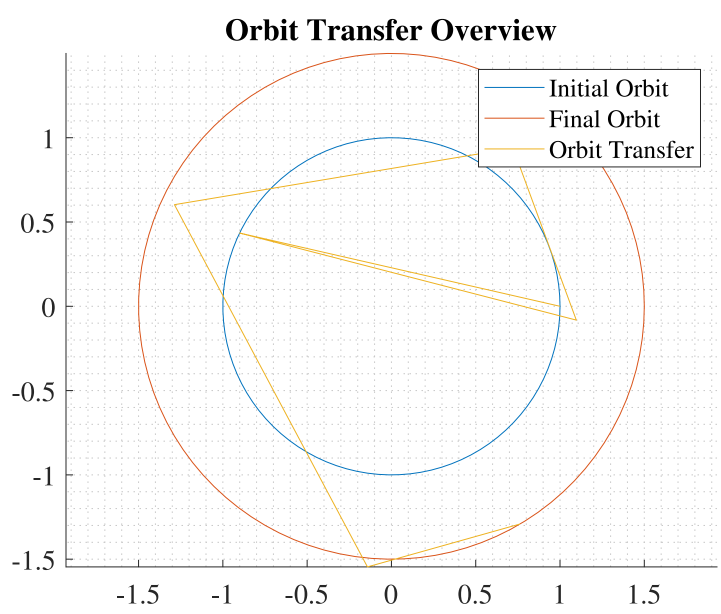

Figure 108: Trajectory from initial to final orbit ($N:3$, $K:2$)

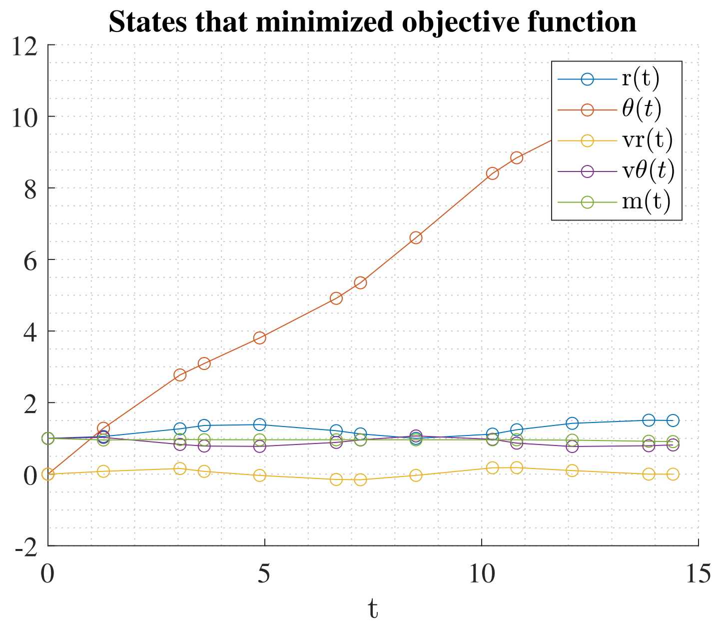

Figure 109: States for trajectory that maximized terminal mass ($N : 3$, $K : 4$)

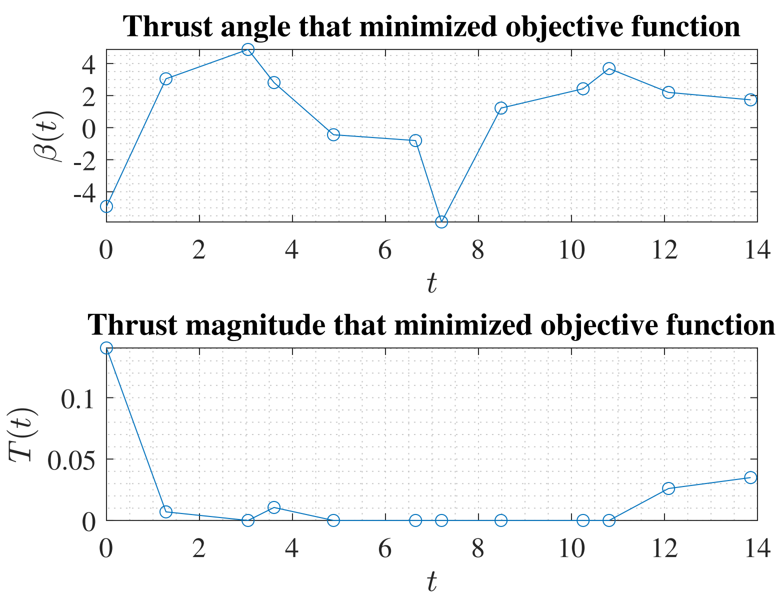

Figure 110: Control that maximized terminal mass ($N : 3$, $K : 4$)

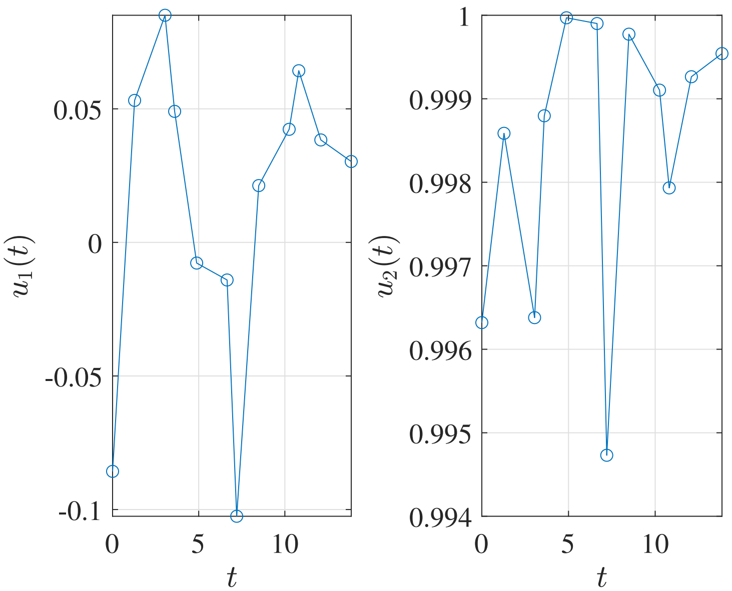

Figure 111: Path constrained control that maximized terminal mass ($N : 3$, $K : 4$)

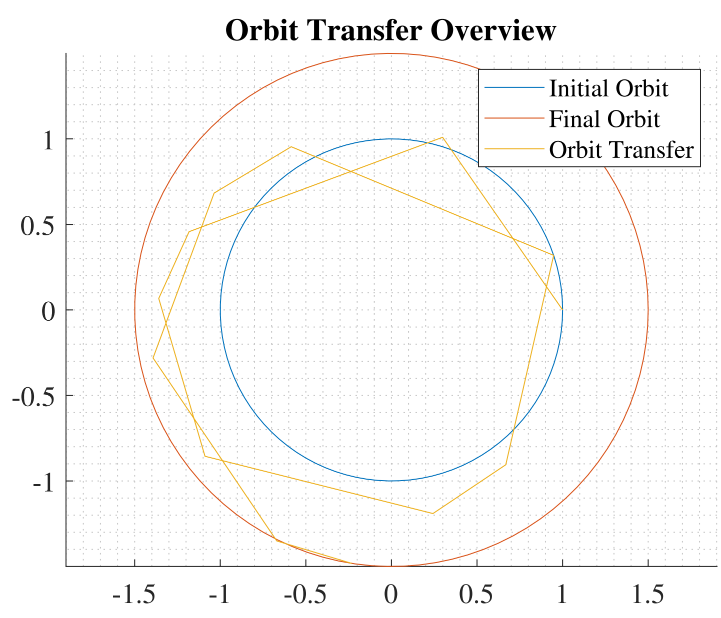

Figure 112: Trajectory from initial to final orbit ($N : 3$, $K : 4$)

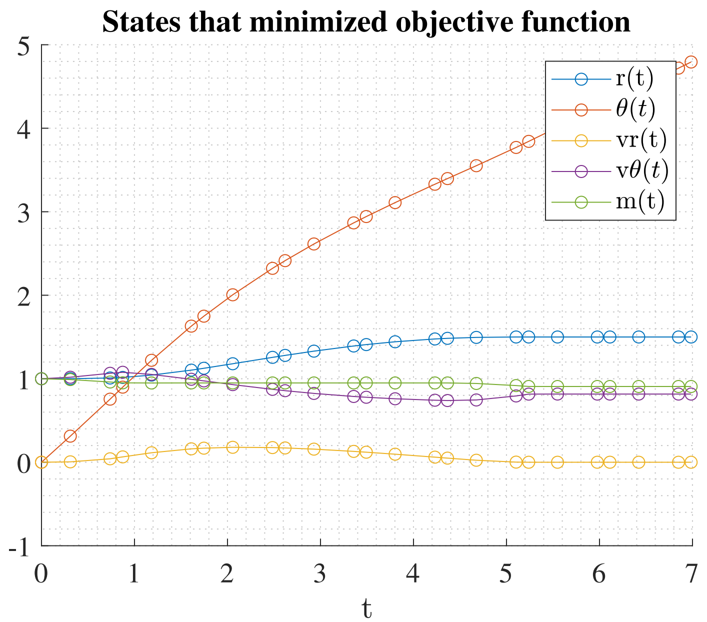

Figure 113: States for trajectory that maximized terminal mass ($N : 3$, $K : 8$)

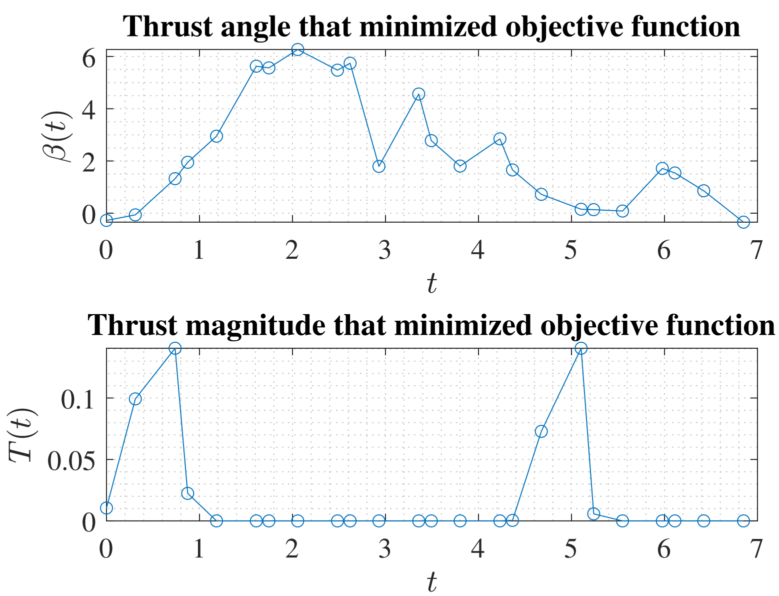

Figure 114: Control that maximized terminal mass ($N : 3$, $K : 8$)

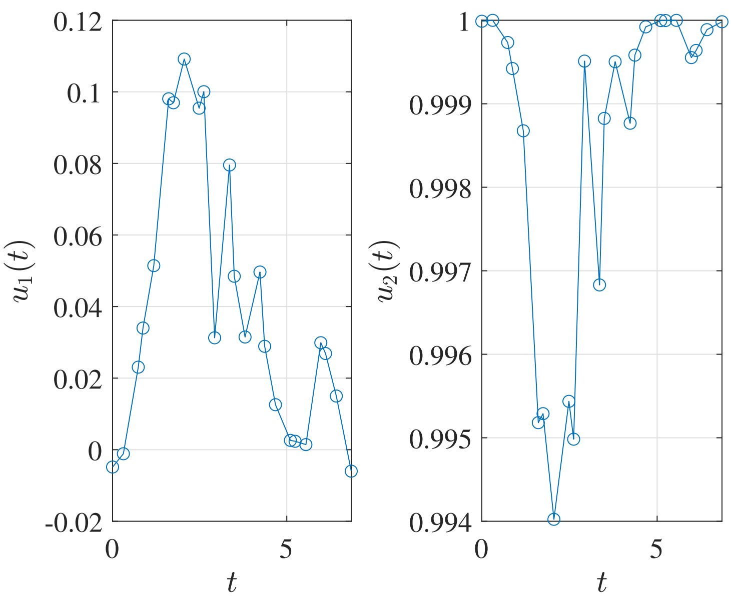

Figure 115: Path constrained control that maximized terminal mass ($N : 3$, $K : 8$)

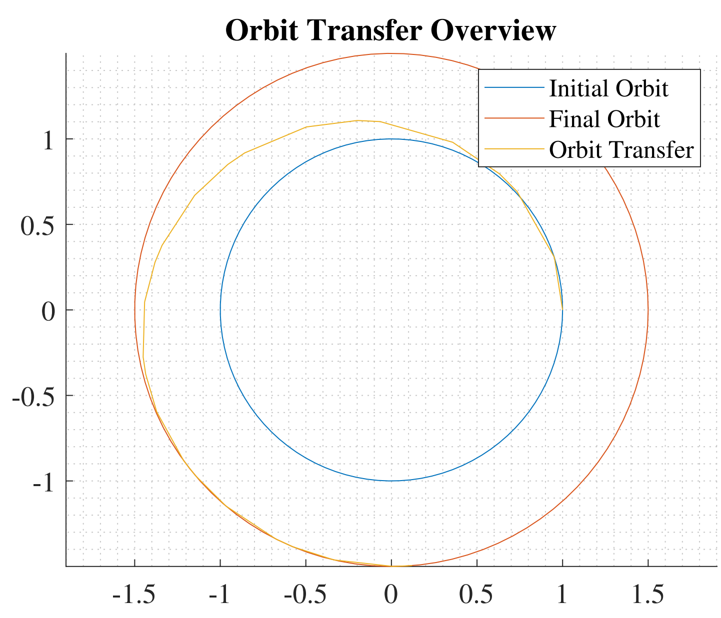

Figure 116: Trajectory from initial to final orbit ($N : 3$, $K : 8$)

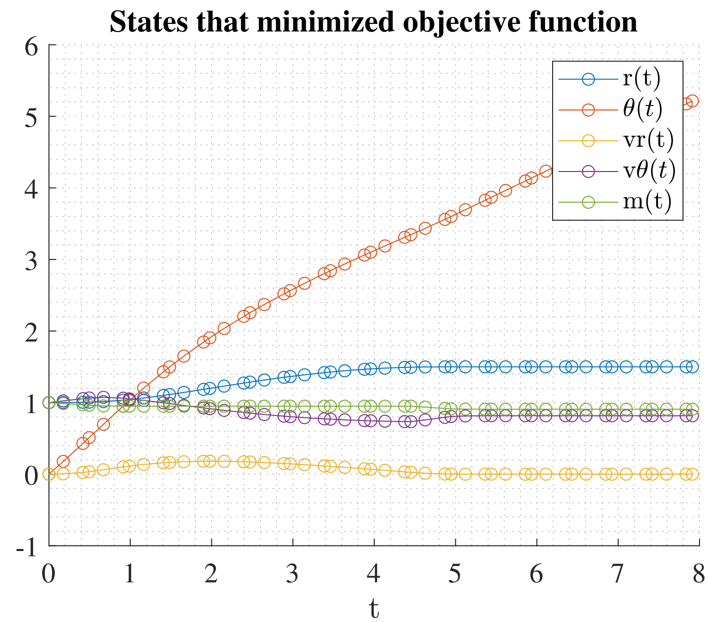

Figure 117: States for trajectory that maximized terminal mass ($N:3$, $K:16$)

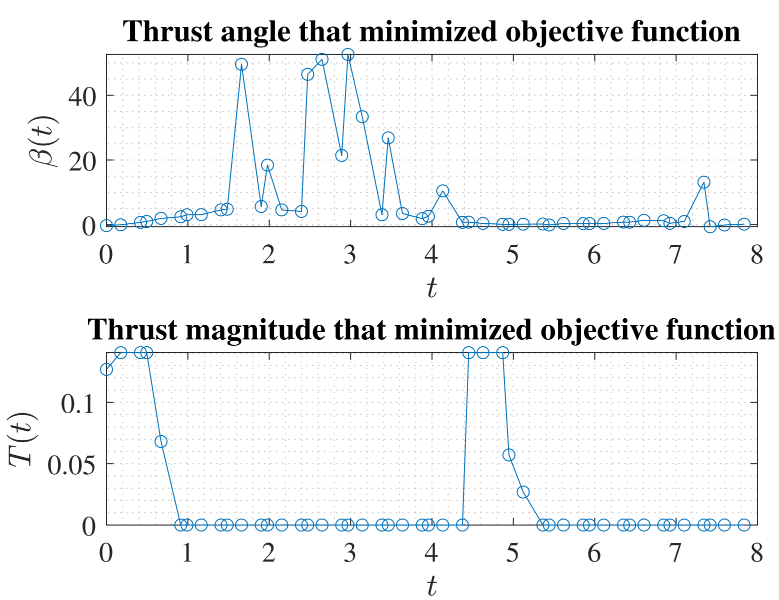

Figure 118: Control that maximized terminal mass ($N:3$, $K:16$)

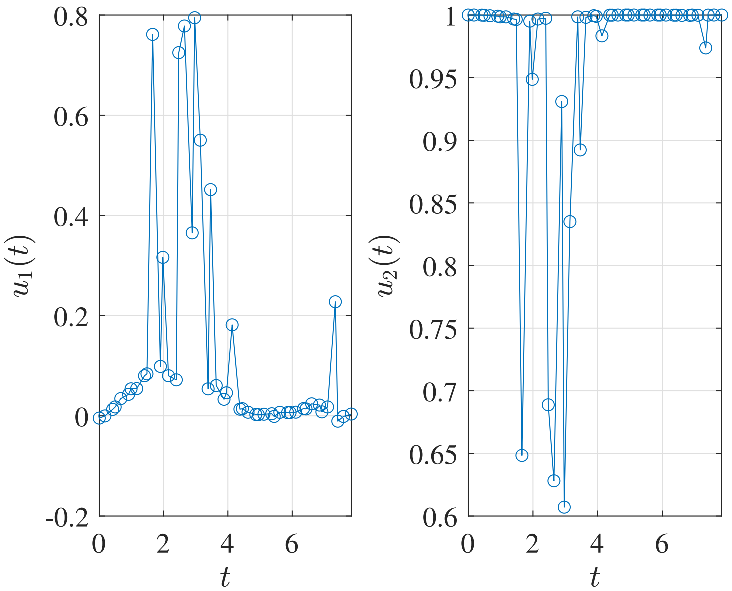

Figure 119: Path constrained control that maximized terminal mass ($N : 3$, $K : 16$)

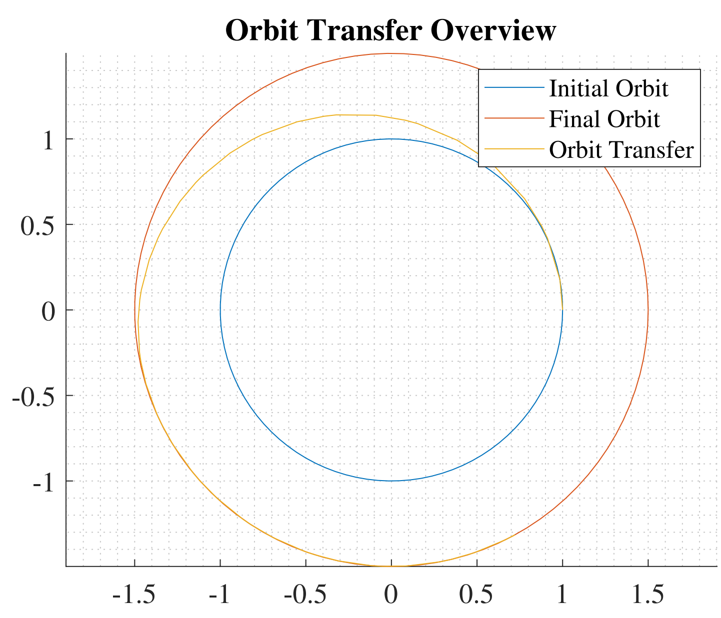

Figure 120: Trajectory from initial to final orbit ($N : 3$, $K : 16$)

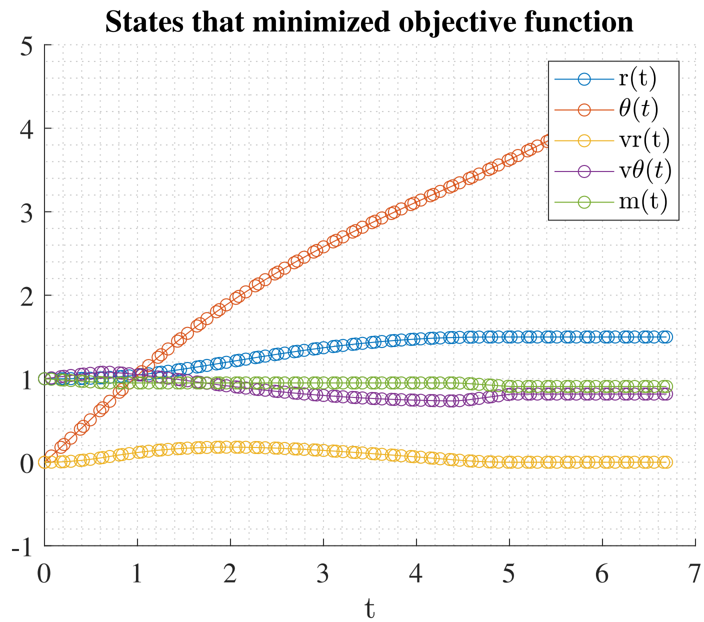

Figure 121: States for trajectory that maximized terminal mass ($N : 3$, $K : 32$)

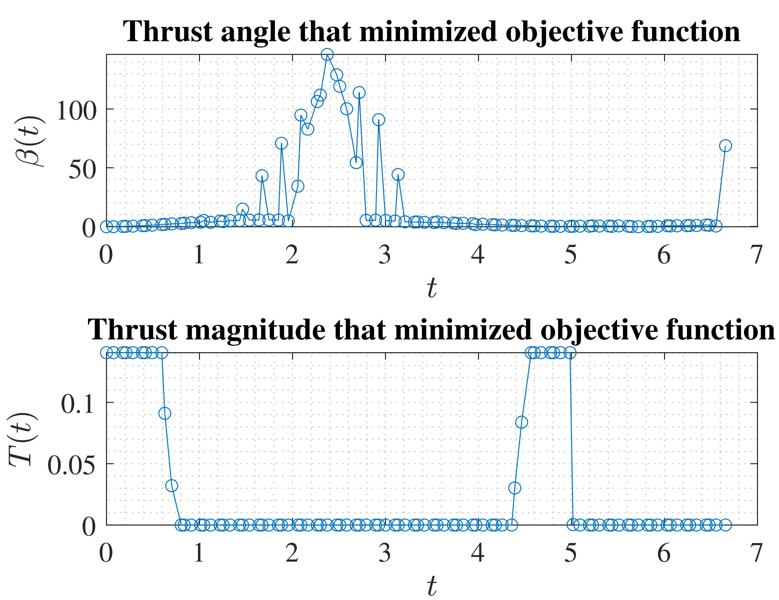

Figure 122: Control that maximized terminal mass ($N : 3$, $K : 32$)

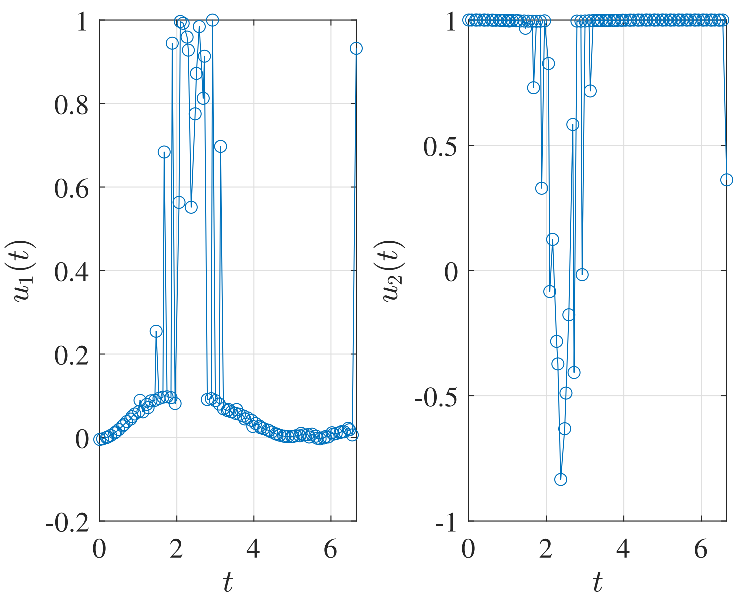

Figure 123: Path constrained control that maximized terminal mass ($N : 3$, K : 32)

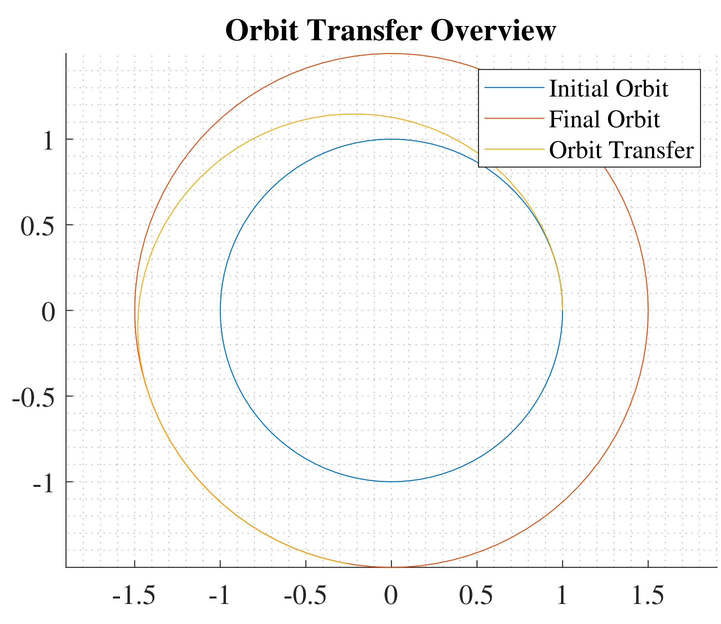

Figure 124: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

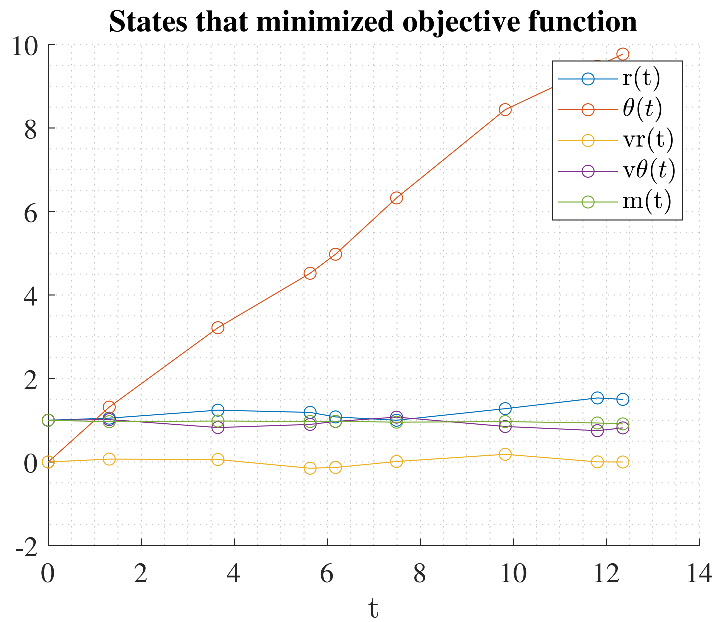

Figure 125: States for trajectory that maximized terminal mass ($N : 4$, $K : 2$)

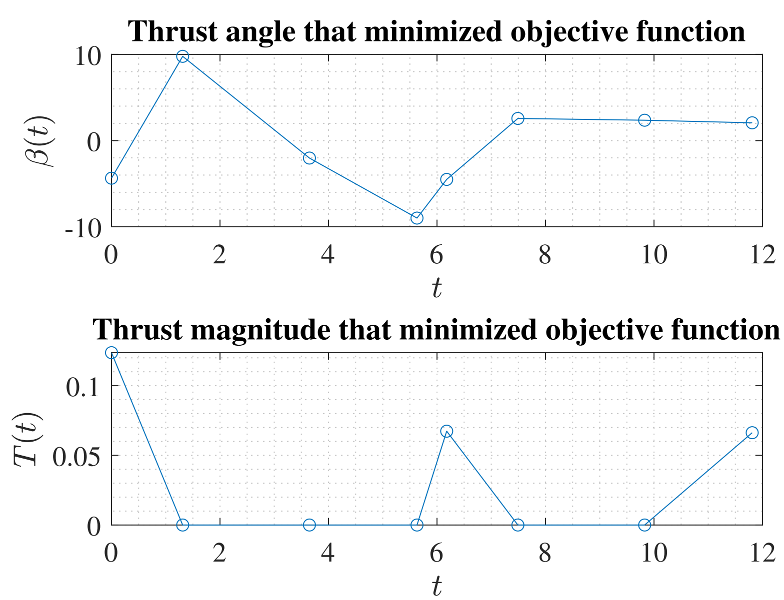

Figure 126: Control that maximized terminal mass ($N : 4$, $K : 2$)

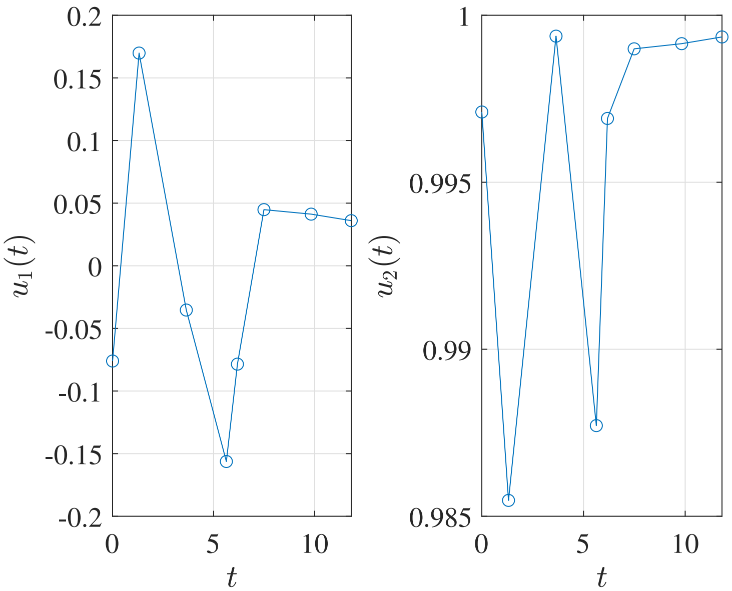

Figure 127: Path constrained control that maximized terminal mass ($N : 4$, $K : 2$)

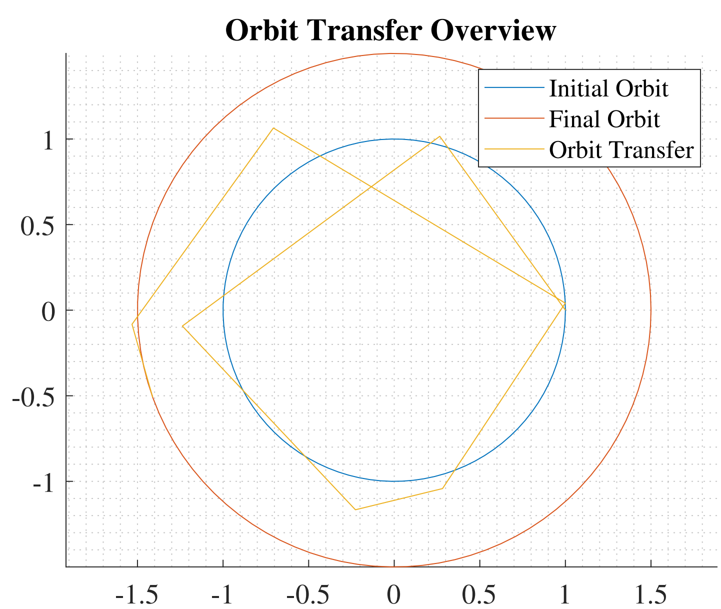

Figure 128: Trajectory from initial to final orbit ($N : 4$, $K : 2$)

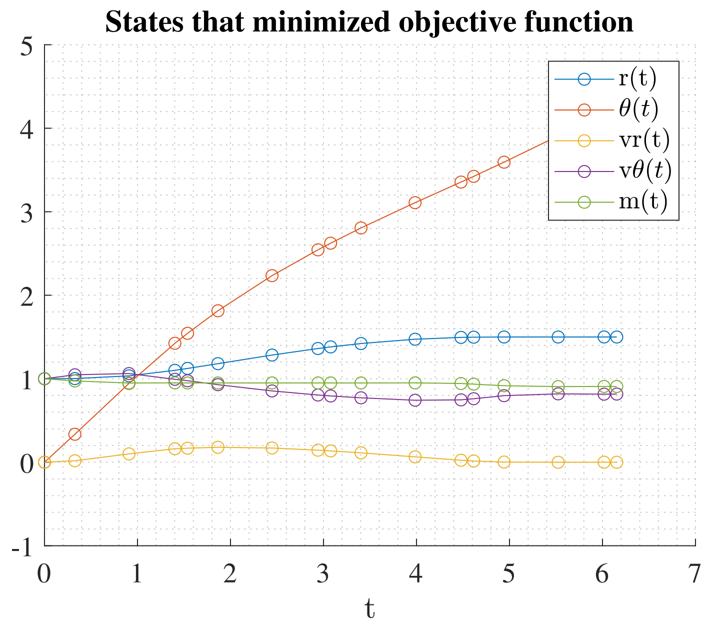

Figure 129: States for trajectory that maximized terminal mass ($N : 4$, $K : 4$)

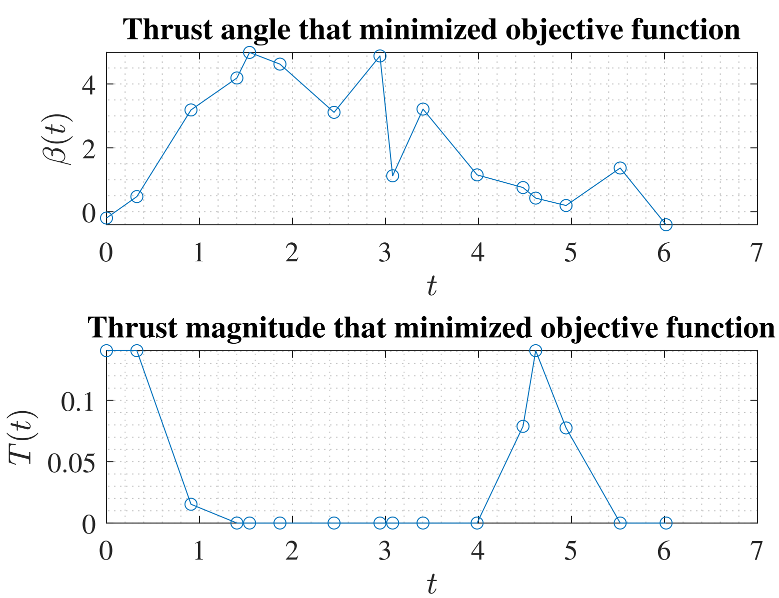

Figure 130: Control that maximized terminal mass ($N : 4$, $K : 4$)

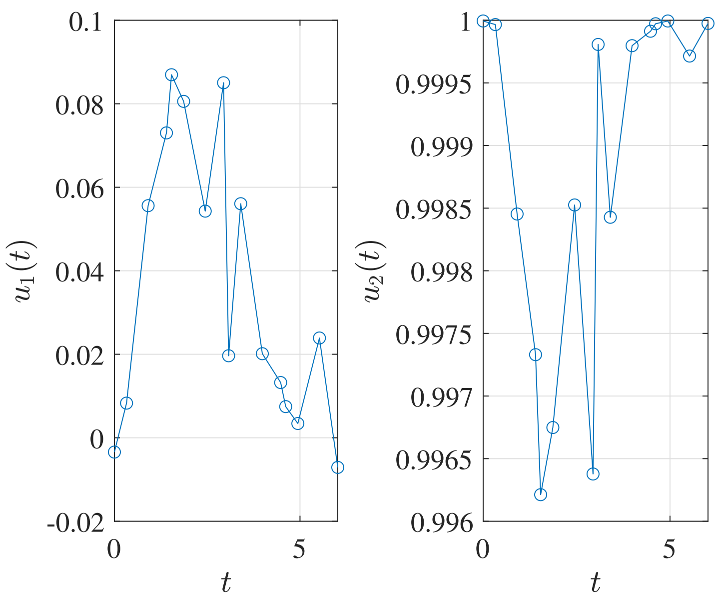

Figure 131: Path constrained control that maximized terminal mass ($N:4$, $K:4$)

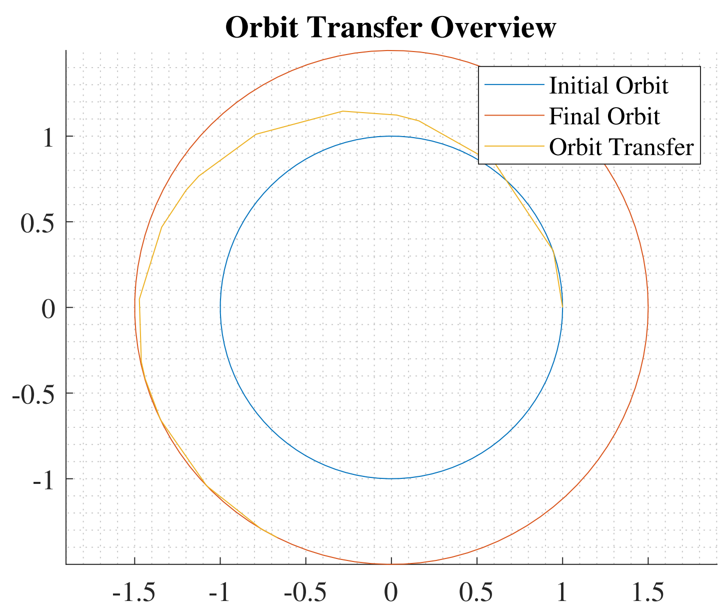

Figure 132: Trajectory from initial to final orbit ($N:4$, $K:4$)

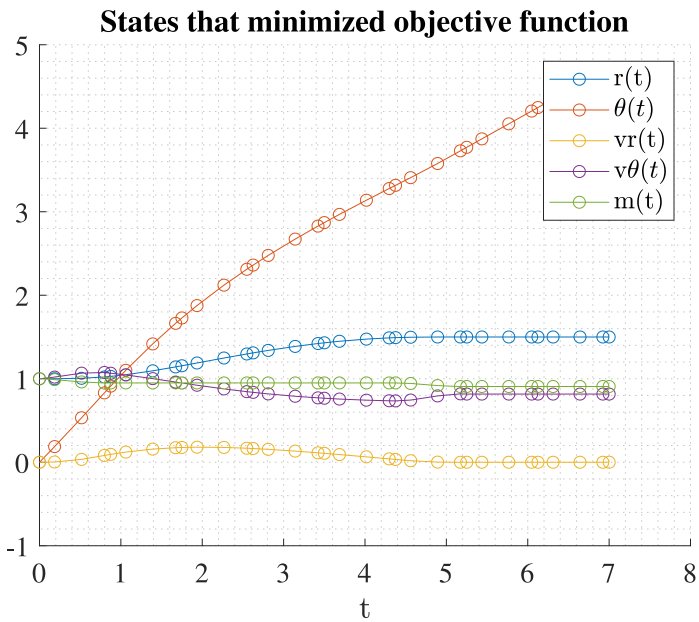

Figure 133: States for trajectory that maximized terminal mass ($N : 4$, $K : 8$)

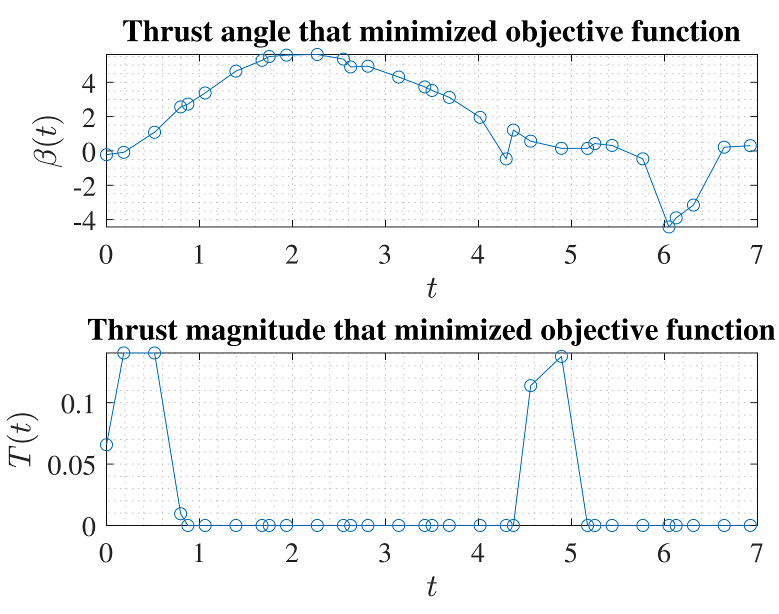

Figure 134: Control that maximized terminal mass ($N : 4$, $K : 8$)

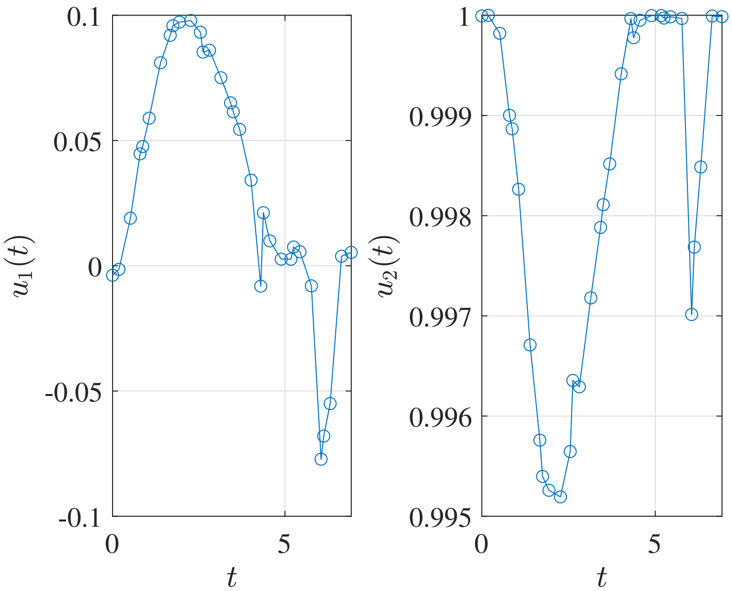

Figure 135: Path constrained control that maximized terminal mass ($N : 4$, $K : 8$)

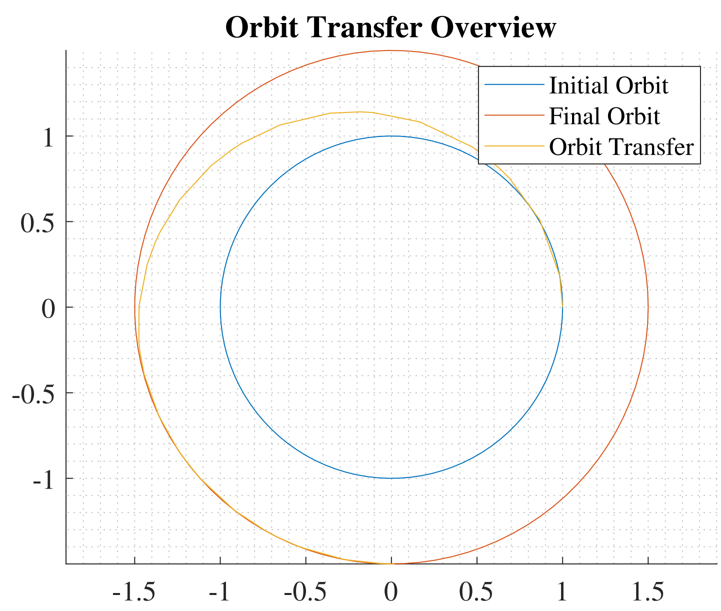

Figure 136: Trajectory from initial to final orbit ($N : 4$, $K : 8$)

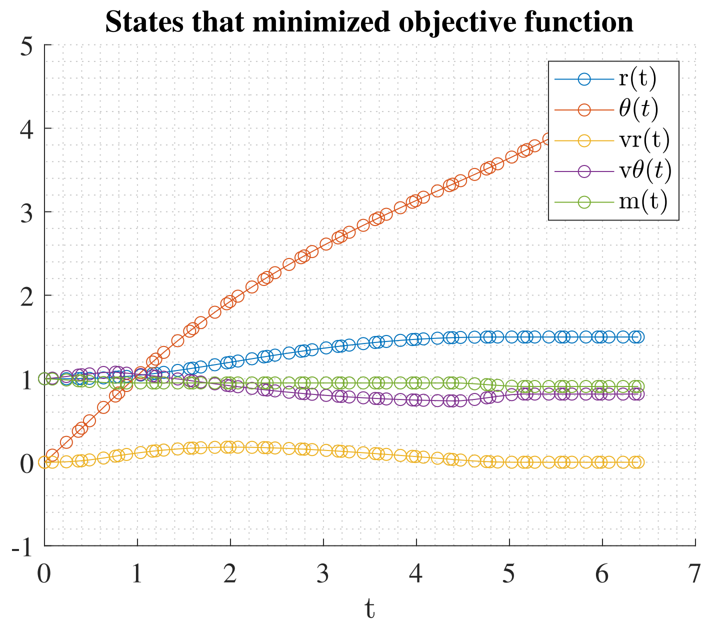

Figure 137: States for trajectory that maximized terminal mass ($N : 4$, $K : 16$)

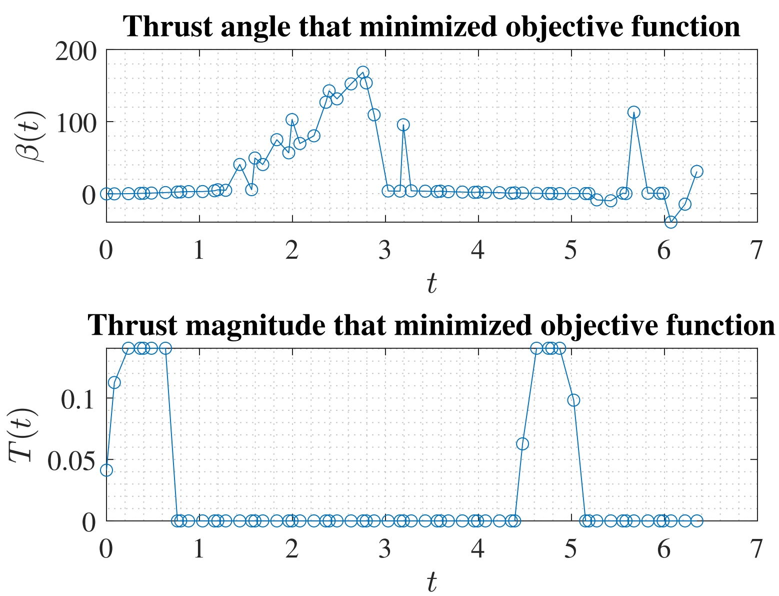

Figure 138: Control that maximized terminal mass ($N : 4$, $K : 16$)

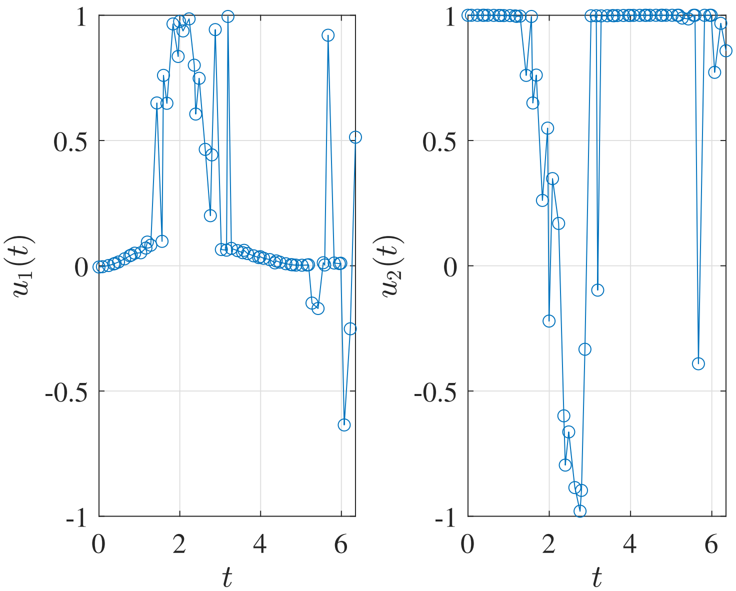

Figure 139: Path constrained control that maximized terminal mass ($N: 4$, K : 16)

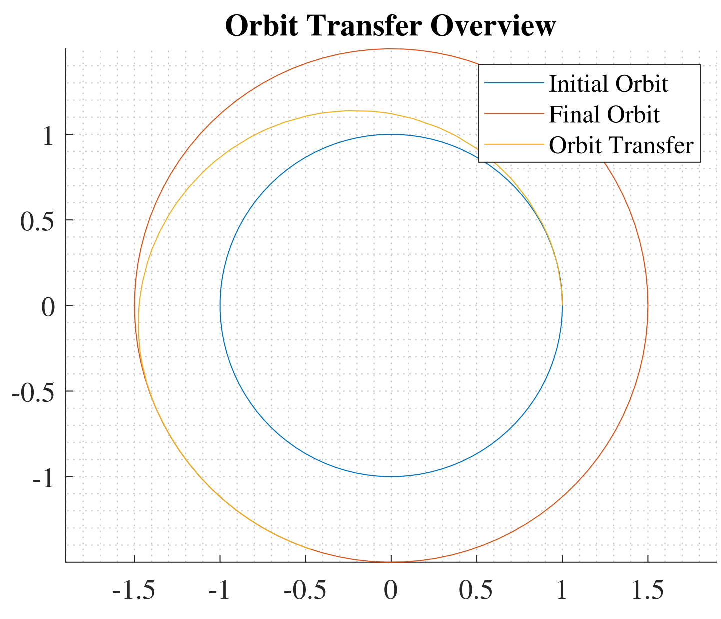

Figure 140: Trajectory from initial to final orbit ($N: 4$, $K: 16$)

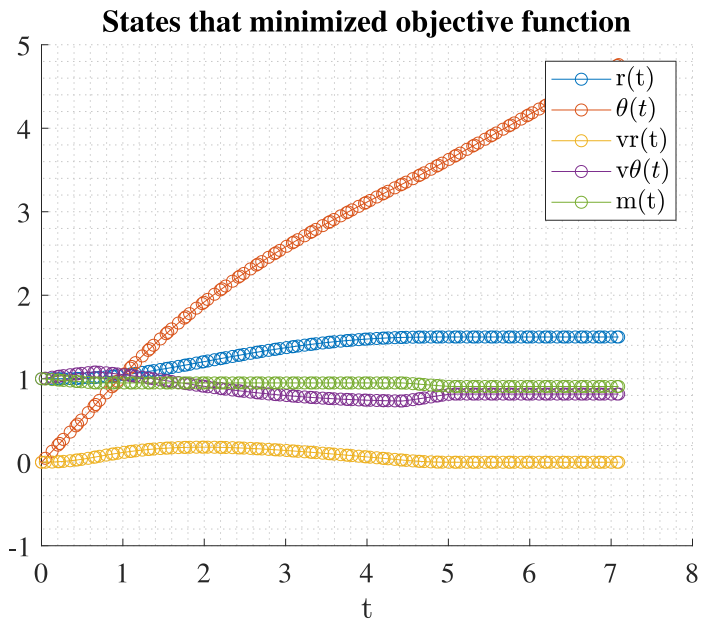

Figure 141: States for trajectory that maximized terminal mass ($N : 4$, $K : 32$)

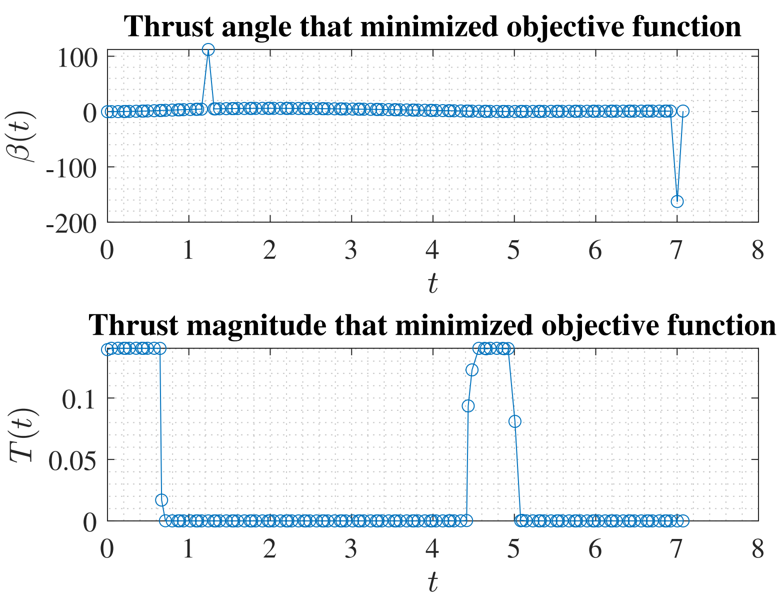

Figure 142: Control that maximized terminal mass ($N : 4$, $K : 32$)

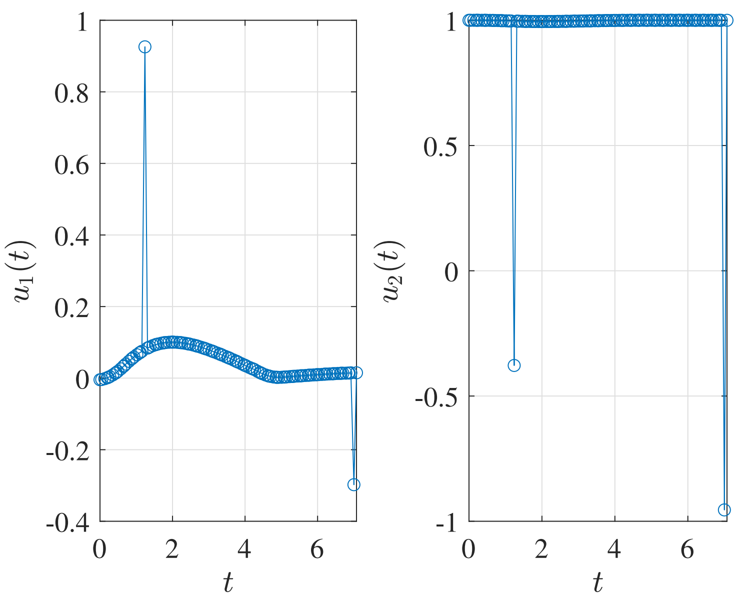

Figure 143: Path constrained control that maximized terminal mass ($N:4$, $K:32$)

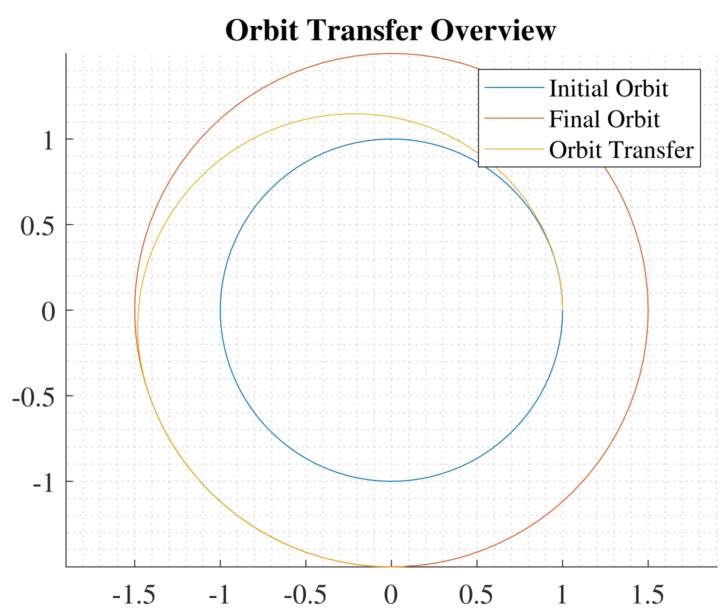

Figure 144: Trajectory from initial to final orbit ($N:4$, $K:32$)

4.5 Overall Analysis

5 Future Work

6 Appendix

Listing 1: orbitTransferMain.m

```
% ---------------------------------------------------------%
% Orbit-Trnasfer Problem %
% ---------------------------------------------------------%
% Solve the following optimal control problem: %
% Maximize t_f %
% subject to the differential equation constraints %
% dr/dt = v_r %
% dtheta/dt = v_theta/r %
% dv_r/dt = v_theta^2/r - mu/r^2 + T*u_1/m %
% dv_theta/dt = -v_r*v_theta/r + T*u_2/m %
% the equality path constraint %
% u_1^2 + u_2^2 = 1 %
% and the boundary conditions %
% r(0) = 1 %
% theta(0) = 0 %
% v_r(0) = 0 %
% v_theta(0) = sqrt(mu/r(0)) %
% m(0) = 1 %
% r(t_f) = 1.5 %
% v_r(t_f) = 0 %
% v_theta(t_f) = sqrt(mu/r(t_f)) %

 close all; clear all;
% -------------------------------------------------------- %
% BEGIN: DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% -------------------------------------------------------- %
global igrid CONSTANTS psStuff nstates ncontrols npaths path_constraint maximize_mass
% -------------------------------------------------------- %
% END: DO NOT ALTER THE FOLLOWING LINES OF CODE!!! %
% -------------------------------------------------------- %
path = 'C:\Users\elias\Documents\UF_Classes\EML6934\Final_Project\EML6934_Final_Project';
addpath(genpath(path))

% save figures or create latex table?
save_figs = 1;
create_latex_table = 1;

% Set path constrant and objective function descision
path_constraint= 1; % is there a path contraint?
maximize_mass = 1; % maximize m(tf), else min tf

% Set polynomial degrees and intervals to loop through
n_list = [3 4];
k_list = [2 4 8 16 32];

% counter for table
count = 1;

for nIdx = 1:numel(n_list)

    for kIdx = 1:numel(k_list)

    % Set polynomial degree and number of intervals
    N = n_list(nIdx); % number of polynomial degree
    numIntervals = k_list(kIdx); % number of intervals

```

```
% set gloabl constants
CONSTANTS.MU = 1;
CONSTANTS.m0 = 1;
CONSTANTS.ve = 1.8658344;
% set number of states
nstates = 5;

% set number of controls and paths
if path_constraint
    ncontrols = 3;
    npaths = 1;
else
    ncontrols = 2;
    npaths = 0;
end

% Bounds on State and Control
% thetaf and mf are free
r0 = 1; theta0 = 0; vr0 = 0; vtheta0 = 1; m0 = 1;
rf = 1.5; vrf = 0; vthetaf = sqrt(1/rf);

rmin = 1; rmax = 3;
thetamin = 0; thetamax = 4*pi;
vrmin = -10; vrmax = 10;
vthetamin = -10; vthetamax = 10;
mmin = 0.1; mmax = m0;
t0min = 0; t0max = 0;
tfmin = 0; tfmax = 100;
if path_constraint
    u1min = -10; u1max = 10; % sin(beta)
    u2min = -10; u2max = 10; % cos(beta)
    u3min = 0; u3max = 0.1405; % thrust
else
    u1min = -4*pi; u1max = 4*pi; % beta
    u2min = 0; u2max = 0; % empty
    u3min = 0; u3max = 0.1405;% thrust
end

% Create Leguandre Gauss Points
meshPoints = linspace(-1,1,numIntervals+1).';
polyDegrees = N*ones(numIntervals,1);
[tau,w,D] = lgrPS(meshPoints,polyDegrees);
psStuff.tau = tau; psStuff.w = w; psStuff.D = D; NLGR = length(w);

% Set the bounds on the NLP variables.
zrmin = rmin*ones(length(tau),1);
zrmax = rmax*ones(length(tau),1);
zrmin(1) = r0; zrmax(1) = r0;
zrmin(end) = rf; zrmax(end) = rf;

zthetamin = thetamin*ones(length(tau),1);
zthetamax = thetamax*ones(length(tau),1);
zthetamin(1) = theta0; zthetamax(1) = theta0;

zvrmin = vrmin*ones(length(tau),1);
zvrmax = vrmax*ones(length(tau),1);
zvrmin(1) = vr0; zvrmax(1) = vr0;
zvrmin(end) = vrf; zvrmax(end) = vrf;

zvthetamin = vthetamin*ones(length(tau),1);
zvthetamax = vthetamax*ones(length(tau),1);
zvthetamin(1) = vtheta0; zvthetamax(1) = vtheta0;
zvthetamin(end) = vthetaf; zvthetamax(end) = vthetaf;

zmmin = mmin*ones(length(tau),1);
zmmax = mmax*ones(length(tau),1);
zmmin(1) = m0; zmmax(1) = m0;
```

```
zu1min = u1min*ones(length(tau)-1,1);
zu1max = u1max*ones(length(tau)-1,1);

zu2min = u2min*ones(length(tau)-1,1);
zu2max = u2max*ones(length(tau)-1,1);

zu3min = u3min*ones(length(tau)-1,1);
zu3max = u3max*ones(length(tau)-1,1);

if path_constraint
    zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zu2min; zu3min; t0min; tfmin];
    zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zu2max; zu3max; t0max; tfmax];
else
    zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zu3min; t0min; tfmin];
    zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zu3max; t0max; tfmax];
end
% Set the bounds on the NLP constraints
% There are NSTATES sets of defect constraints.
defectMin = zeros(nstates*(length(tau)-1),1);
defectMax = zeros(nstates*(length(tau)-1),1);
if path_constraint
    % There is a path constraint
    pathMin = ones(length(tau)-1,1); pathMax = ones(length(tau)-1,1);
else
    % No path constraint
    pathMin = []; pathMax = [];
end
% I dont believe there is nonlinear event constraint
eventMin = []; eventMax = [];
objMin = -inf; objMax = inf;
Fmin = [objMin; defectMin; pathMin; eventMin];
Fmax = [objMax; defectMax; pathMax; eventMax];

% Supply an initial guess
rguess = linspace(r0,rf,NLGR+1).';
thetaguess = linspace(theta0,theta0,NLGR+1).';
vrguess = linspace(vr0,vrf,NLGR+1).';
vthetaguess = linspace(vtheta0,vtheta0,NLGR+1).';
mguess = linspace(m0,m0,NLGR+1).';
u3guess = linspace(0,0,NLGR).';
t0guess = 0;
tfguess = 3.5;

if path_constraint
    u1guess = linspace(1,1,NLGR).';
    u2guess = linspace(0,0,NLGR).';
    z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;u2guess;u3guess;t0guess;tfguess];
else
    u1guess = linspace(0,0,NLGR).';
    z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;u3guess;t0guess;tfguess];
end

%-------------------------------------------------------------------%
% Generate derivatives and sparsity pattern using Adigator %
%-------------------------------------------------------------------%
% - Constraint Function Derivatives
xsize = size(z0);
x = adigatorCreateDerivInput(xsize,'z0');
output = adigatorGenJacFile('orbitTransferFun',{x});
S_jac = output.JacobianStructure;
[iGfun,jGvar] = find(S_jac);

% - Objective Function Derivatives
xsize = size(z0);
x = adigatorCreateDerivInput(xsize,'z0');
output = adigatorGenJacFile('orbitTransferObj',{x});
grd_structure = output.JacobianStructure;
```

```
%-----------------------------------------------------------------%
% set IPOPT callback functions
%-----------------------------------------------------------------%
funcs.objective = @(Z)orbitTransferObj(Z);
funcs.gradient = @(Z)orbitTransferGrd(Z);
funcs.constraints = @(Z)orbitTransferCon(Z);
funcs.jacobian = @(Z)orbitTransferJac(Z);
funcs.jacobianstructure = @()orbitTransferJacPat(S_jac);
options.ipopt.hessian_approximation = 'limited-memory';

%-----------------------------------------------------------------%
% Set IPOPT Options %
%-----------------------------------------------------------------%
options.ipopt.tol = 1e-8;
options.ipopt.linear_solver = 'ma57'; %'mumps';
options.ipopt.max_iter = 8000;
options.ipopt.mu_strategy = 'adaptive';
options.ipopt.ma57_automatic_scaling = 'yes';
options.ipopt.print_user_options = 'yes';
options.ipopt.output_file = ['orbitTransfer','IPOPTinfo.txt']; % print output file
options.ipopt.print_level = 5; % set print level default

options.lb = zmin; % Lower bound on the variables.
options.ub = zmax; % Upper bound on the variables.
options.cl = Fmin; % Lower bounds on the constraint functions.
options.cu = Fmax; % Upper bounds on the constraint functions.

%-----------------------------------------------------------------%
% Call IPOPT
%-----------------------------------------------------------------%
[z, info] = ipopt(z0,funcs,options);

%-----------------------------------------------------------------%
% extract lagrange multipliers from ipopt output, info
%-----------------------------------------------------------------%
Fmul = info.lambda;

% Extract the state and control from the decision vector z.
% Remember that the state is approximated at the LGR points
% plus the final point, while the control is only approximated
% at only the LGR points.
r = z(1:NLGR+1);
theta = z(NLGR+2:2*(NLGR+1));
vr = z(2*(NLGR+1)+1:3*(NLGR+1));
vtheta = z(3*(NLGR+1)+1:4*(NLGR+1));
m = z(4*(NLGR+1)+1:5*(NLGR+1));
u1 = z(5*(NLGR+1)+1:5*(NLGR+1)+NLGR);
if path_constraint
    u2 = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
    u3 = z(5*(NLGR+1)+2*NLGR+1:5*(NLGR+1)+3*NLGR);
    beta = atan2(u1,u2);
    beta = unwrap(beta)*180/pi;
else
    u3 = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
    beta = unwrap(u1)*180/pi;
end
t0 = z(end-1);
tf = z(end);
t = (tf-t0)*(tau+1)/2+t0;
tLGR = t(1:end-1);
%-----------------------------------------------------------------%
% Extract the Lagrange multipliers corresponding %
% the defect constraints. %
%-----------------------------------------------------------------%
multipliersDefects = Fmul(2:nstates*NLGR+1);
multipliersDefects = reshape(multipliersDefects,NLGR,nstates);
%-----------------------------------------------------------------%
% Compute the costates at the LGR points via transformation %
```

```
%-------------------------------------------------------------------------%
costateLGR = inv(diag(w))*multipliersDefects;
%-------------------------------------------------------------------------%
% Compute the costate at the tau=+1 via transformation %
%-------------------------------------------------------------------------%
costateF = D(:,end).'*multipliersDefects;
%-------------------------------------------------------------------------%
% Now assemble the costates into a single matrix %
%-------------------------------------------------------------------------%
costate = [costateLGR; costateF];
lamr = costate(:,1); lamtheta = costate(:,2);
lamvr = costate(:,3); lamvtheta = costate(:,4);

%-------------------------------------------------------------------------%
% Get planer coordinates
%-------------------------------------------------------------------------%
x_transfer = r.*cos(theta);
y_transfer = r.*sin(theta);
theta_orbit = linspace(0,2*pi,100);
x_orbit_1 = r0*cos(theta_orbit);
y_orbit_1 = r0*sin(theta_orbit);
x_orbit_2 = rf*cos(theta_orbit);
y_orbit_2 = rf*sin(theta_orbit);

%--------------%
% Plot Results
%--------------%
close all

fig_cntrl = figure;
h1 = subplot(2,1,1);
plot(tLGR,beta,'-o');
ylabel('$\beta(t)$','Interpreter','LaTeX');
xlabel('$t$','Interpreter','LaTeX')
set(gca,'FontName','Times','FontSize',14);
set(gcf,'color','white')
title('Thrust angle that minimized objective function')
grid minor;

h2 = subplot(2,1,2);
plot(tLGR,u3,'-o');
set(gca,'FontName','Times','FontSize',14);
set(gcf,'color','white')
ylabel('$T(t)$','Interpreter','LaTeX');
title('Thrust magnitude that minimized objective function')
xlabel('$t$','Interpreter','LaTeX')
grid minor
linkaxes([h1 h2],'x')

fig_states = figure; hold on; grid minor
plot(t,r,'-o'); plot(t,theta,'-o'); plot(t,vr,'-o');
plot(t,vtheta,'-o'); plot(t,m,'-o');
xlabel('t','Interpreter','LaTeX')
legend('r(t)','$\theta(t)$','vr(t)','v$\theta(t)$','m(t)','Interpreter','LaTeX')
set(gcf,'color','white')
set(gca,'FontName','Times','fontsize',14)
title('States that minimized objective function')

fig_orbit = figure; hold on; grid minor
plot(x_orbit_1,y_orbit_1)
plot(x_orbit_2,y_orbit_2)
plot(x_transfer,y_transfer)
set(gca,'FontName','Times','fontsize',14)
set(gcf,'color','white')
legend('Initial Orbit','Final Orbit','Orbit Transfer')
axis equal
title('Orbit Transfer Overview')
```

```
    if save_figs
        if maximize_mass
            str_obj = 'mf';
        else
            str_obj = 'tf';
        end
        plot_str = sprintf('_N%d_K%d_C%d_',N,numIntervals,ncontrols);
        str_list = {'control','states','orbit'};
        for idx = 1:numel(str_list)
            name = [str_list{idx} plot_str str_obj];
            print(figure(idx),name,'-depsc')
        end
    end
    if path_constraint
        fig_path = figure;
        subplot(1,2,1);
        plot(tLGR,u1,'-o');
        xl = xlabel('$t$','Interpreter','LaTeX');
        yl = ylabel('$u_1(t)$','Interpreter','LaTeX');
        set(xl,'FontSize',14);
        set(yl,'FontSize',14);
        set(gca,'FontName','Times','FontSize',14);
        grid on;

        subplot(1,2,2);
        plot(tLGR,u2,'-o');
        xl = xlabel('$t$','Interpreter','LaTeX');
        yl = ylabel('$u_2(t)$','Interpreter','LaTeX');
        set(xl,'FontSize',14);
        set(yl,'FontSize',14);
        set(gca,'FontName','Times','FontSize',14);
        set(gcf,'color','white')
        grid on;

        if save_figs
            str_path = ['path' plot_str str_obj];
            print(fig_path,str_path,'-depsc')
        end
    end

% figure(1);
% subplot(2,2,1);
% plot(t,r,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$r(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
% set(gcf,'color','white')
% grid on;
%
% subplot(2,2,2);
% plot(t,theta,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$\theta(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
% set(gcf,'color','white')
% grid on;
%
% subplot(2,2,3);
% plot(t,vr,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$v_r(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
```

```
% set(gcf,'color','white')
% grid on;
%
% subplot(2,2,4);
% plot(t,vtheta,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$v_\theta(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
% set(gcf,'color','white')
% grid on;

% figure;
% subplot(2,2,1);
% plot(t,lamr,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$\lambda_r(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
% set(gcf,'color','white')
% grid on;
%
% subplot(2,2,2);
% plot(t,lamtheta,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$\lambda_\theta(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
% set(gcf,'color','white')
% grid on;
%
% subplot(2,2,3);
% plot(t,lamvr,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$\lambda_{v_r}(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
% set(gcf,'color','white')
% grid on;
%
% subplot(2,2,4);
% plot(t,lamvtheta,'-o');
% xl = xlabel('$t$','Interpreter','LaTeX');
% yl = ylabel('$\lambda_{v_\theta}(t)$','Interpreter','LaTeX');
% set(xl,'FontSize',14);
% set(yl,'FontSize',14);
% set(gca,'FontName','Times','FontSize',14);
% set(gcf,'color','white')
% grid on;

    %---------------%
    % save results %
    %---------------%
    degree(count,1) = N;
    intervals(count,1) = numIntervals;
    iterations(count,1) = info.iter;
    cpu_time(count,1) = info.cpu;
    final_time(count,1) = tf;
    final_mass(count,1) = m(end);
    solved_info(count,1) = info.status;
    count = count + 1;
    end
end
table_mat = horzcat(degree,intervals,iterations,cpu_time,final_time,final_mass,solved_info);
```

```
T = array2table(table_mat);
VarNames = {'Degree','Intervals','Iterations','CPU Time','tf','mf','Solved_Status'};
T.Properties.VariableNames = VarNames;
if create_latex_table
    str_table = sprintf('table_C%d_',ncontrols);
    table2latex(T,[str_table str_obj])
end
```

Listing 2: orbitTransferFun.m

```
function C = orbitTransferFun(z)

%---------------------------------------------------------------------%
% Objective and constraint functions for the orbit-raising %
% problem. This function is designed to be used with the NLP %
% solver SNOPT. %
%---------------------------------------------------------------------%
% DO NOT FOR ANY REASON ALTER THE LINE OF CODE BELOW! %
global psStuff nstates ncontrols npaths CONSTANTS path_constraint maximize_mass%
% DO NOT FOR ANY REASON ALTER THE LINE OF CODE ABOVE! %
%---------------------------------------------------------------------%

%---------------------------------------------------------------------%
% Extract the constants used in the problem. %
%---------------------------------------------------------------------%
mu = CONSTANTS.MU;
ve = CONSTANTS.ve;
%---------------------------------------------------------------------%
% Radau pseudospectral method quantities required: %
% - Differentiation matrix (psStuff.D) %
% - Legendre-Gauss-Radau weights (psStuff.w) %
% - Legendre-Gauss-Radau points (psStuff.tau) %
%---------------------------------------------------------------------%
D = psStuff.D; tau = psStuff.tau; w = psStuff.w;

%---------------------------------------------------------------------%
% Decompose the NLP decision vector into pieces containing %
% - the state %
% - the control %
% - the initial time %
% - the final time %
%---------------------------------------------------------------------%
N = length(tau)-1;
stateIndices = 1:nstates*(N+1);
controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
t0Index = controlIndices(end)+1;
tfIndex = t0Index+1;
stateVector = z(stateIndices);
controlVector = z(controlIndices);
t0 = z(t0Index);
tf = z(tfIndex);

%---------------------------------------------------------------------%
% Reshape the state and control parts of the NLP decision vector %
% to matrices of sizes (N+1) by nstates and (N+1) by ncontrols, %
% respectively. The state is approximated at the N LGR points %
% plus the final point. Thus, each column of the state vector is %
% length N+1. The LEFT-HAND SIDE of the defect constraints, D*X, %
% uses the state at all of the points (N LGR points plus final %
% point). The RIGHT-HAND SIDE of the defect constraints, %
% (tf-t0)F/2, uses the state and control at only the LGR points. %
% Thus, it is necessary to extract the state approximations at %
% only the N LGR points. Finally, in the Radau pseudospectral %
% method, the control is approximated at only the N LGR points. %
%---------------------------------------------------------------------%
statePlusEnd = reshape(stateVector,N+1,nstates);
stateLGR = statePlusEnd(1:end-1,:);
```

```
control = reshape(controlVector,N,ncontrols);

%-------------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR. %
%-------------------------------------------------------------------------%
r = stateLGR(:,1);
theta = stateLGR(:,1);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
if path_constraint
    u1 = control(:,1);
    u2 = control(:,2);
    u3 = control(:,3);
else
    u1 = control(:,1);
    u2 = 0;
    u3 = control(:,2);
end
%-------------------------------------------------------------------------%
% The quantity STATEF is the value of the state at the final %
% time, tf, which corresponds to the state at $\tau=1$. %
%-------------------------------------------------------------------------%
% stateF = statePlusEnd(end,:);
%-------------------------------------------------------------------------%
% The orbit-raising problem contains one nonlinear boundary %
% condition $\sqrt{mu/r(t_f)-v_\theta(t_f) = 0$. Because $r(t)$ %
% and $v_\theta(t)$ are the first and fourth components of the %
% state, it is necessary to extract stateF(1) and stateF(4) in %
% order to compute this boundary condition function. %
%-------------------------------------------------------------------------%
% rF = stateF(1);
% vthetaF = stateF(4);
%
% a = T./m;

%-------------------------------------------------------------------------%
% Compute the right-hand side of the differential equations at %
% the N LGR points. Each component of the right-hand side is %
% stored as a column vector of length N, that is each column has %
% the form %
% [ f_i(x_1,u_1,t_1) ] %
% [ f_i(x_2,u_2,t_2) ] %
% . %
% . %
% . %
% [ f_i(x_N,u_N,t_N) ] %
% where "i" is the right-hand side of the ith component of the %
% vector field f. It is noted that in MATLABB the calculation of %
% the right-hand side is vectorized. %
%-------------------------------------------------------------------------%
rdot = vr;
thetadot = vtheta./r;
mdot = -u3./ve;
if path_constraint
    vrdot = vtheta.^2./r - mu./r.^2 + u3.*u1./m;
    vthetadot = -vtheta.*vr./r + u3.*u2./m;
else
    vrdot = vtheta.^2./r - mu./r.^2 + u3.*sin(u1)./m;
    vthetadot = -vtheta.*vr./r + u3.*cos(u1)./m;
end

diffeqRHS = [rdot, thetadot, vrdot, vthetadot, mdot];

%-------------------------------------------------------------------------%
% Compute the left-hand side of the defect constraints, recalling %
% that the left-hand side is computed using the state at the LGR %
% points PLUS the final point. %
```

```
%--------------------------------------------------------------------%
diffeqLHS = D*statePlusEnd;

%--------------------------------------------------------------------%
% Construct the defect constraints at the N LGR points. %
% Remember that the right-hand side needs to be scaled by the %
% factor (tf-t0)/2 because the rate of change of the state is %
% being taken with respect to $\tau\in[-1,+1]$. Thus, we have %
% $dt/t\dau=(tf-t0)/2$. %
%--------------------------------------------------------------------%
defects = diffeqLHS-(tf-t0)*diffeqRHS/2;

%--------------------------------------------------------------------%
% Construct the path constraints at the N LGR points. %
% Reshape the path contraints into a column vector. %
%--------------------------------------------------------------------%
if path_constraint
    paths = u1.^2+u2.^2;
    paths = reshape(paths,N*npaths,1);
else
    paths = [];
end
% paths = u1.^2+u2.^2;

%--------------------------------------------------------------------%
% Reshape the defect contraints into a column vector. %
%--------------------------------------------------------------------%
defects = reshape(defects,N*nstates,1);

%--------------------------------------------------------------------%
% Construct the objective function plus constraint vector. %
%--------------------------------------------------------------------%
if maximize_mass
    m = statePlusEnd(:,5);
    J = -m(end);
else
    J = tf;
end
C = [J;defects;paths];
```

Listing 3: orbitTransferObj.m

```
function obj = orbitTransferObj(z)
% Computes the objective function of the problem

global psStuff nstates ncontrols maximize_mass

%--------------------------------------------------------------------%
% Extract the constants used in the problem. %
%--------------------------------------------------------------------%
% MU = CONSTANTS.MU; mdot = CONSTANTS.mdot; T = CONSTANTS.T;

%--------------------------------------------------------------------%
% Radau pseudospectral method quantities required: %
% - Differentiation matrix (psStuff.D) %
% - Legendre-Gauss-Radau weights (psStuff.w) %
% - Legendre-Gauss-Radau points (psStuff.tau) %
%--------------------------------------------------------------------%
D = psStuff.D; tau = psStuff.tau; w = psStuff.w;

%--------------------------------------------------------------------%
% Decompose the NLP decision vector into pieces containing %
% - the state %
% - the control %
% - the initial time %
% - the final time %
%--------------------------------------------------------------------%
```

```
N = length(tau)-1;
stateIndices = 1:nstates*(N+1);
controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
t0Index = controlIndices(end)+1;
tfIndex = t0Index+1;
stateVector = z(stateIndices);
% controlVector = z(controlIndices);
% t0 = z(t0Index);
tf = z(tfIndex);

%-------------------------------------------------------------------------%
% Reshape the state and control parts of the NLP decision vector %
% to matrices of sizes (N+1) by nstates and (N+1) by ncontrols, %
% respectively. The state is approximated at the N LGR points %
% plus the final point. Thus, each column of the state vector is %
% length N+1. The LEFT-HAND SIDE of the defect constraints, D*X, %
% uses the state at all of the points (N LGR points plus final %
% point). The RIGHT-HAND SIDE of the defect constraints, %
% (tf-t0)F/2, uses the state and control at only the LGR points. %
% Thus, it is necessary to extract the state approximations at %
% only the N LGR points. Finally, in the Radau pseudospectral %
% method, the control is approximated at only the N LGR points. %
%-------------------------------------------------------------------------%
statePlusEnd = reshape(stateVector,N+1,nstates);
stateLGR = statePlusEnd(1:end-1,:);
% control = reshape(controlVector,N,ncontrols);

%-------------------------------------------------------------------------%
% Identify the components of the state column-wise from stateLGR. %
%-------------------------------------------------------------------------%
% r = stateLGR(:,1);
% theta = stateLGR(:,1);
% vr = stateLGR(:,3);
% vtheta = stateLGR(:,4);


% Cost Function
% minizing time or maximizing mass
if maximize_mass
    m = statePlusEnd(:,5);
    J = -m(end);
else
    J = tf;
end

obj = J;

end
```

Listing 4: orbitTransferCon.m

```
function constraints = orbitTransferCon(Z)
% computes the constraints

output = orbitTransferFun(Z);
constraints = output;

end
```

Listing 5: orbitTransferGrd.m

```
function grd = orbitTransferGrd(Z)
% computes the gradient

output = orbitTransferObj_Jac(Z);
grd = output;
```

```
end
```

Listing 6: orbitTransferJac.m

```
function jac = orbitTransferJac(Z)
% computes the jacobian

[jac,~] = orbitTransferFun_Jac(Z);

end
```

Listing 7: orbitTransferJacPat.m

```
function jacpat = orbitTransferJacPat(S_jac)
% computes the jacobian structure

jacpat = S_jac;

end
```