EML6934 Optimal Control
Final Project

Elias Reyes

April 25, 2022

Contents

List of Figures

List of Tables

Listings

1 Introduction

There are many different numerical methods used to solve optimal control problems. In EML6934, a handful of numerical methods were introduced, some of which included Indirect Single Shooting, Indirect Multiple Shooting, Direct Single Shooting, Direct Multiple Shooting, Indirect Collocation, Direct Collocation and Multiple-Interval Legendre-Gauess-Radau Collocation. Every method introduced has it's pros and cons, not making a single method the best option for any application. The first four of these methods were investigated in the midterm by solving the orbit transfer problem. The last method, LGR Collocation is the focus of this project. In this project, LGR Collocation is used to solve a modified version of the orbit transfer problem. In doing so, the phenomena of using equality path constraints for optimal control is investigated. The results using LGR Collocation, with and without path constraints are compared to the results using the methods from the midterm.

2 Main Body

2.1 Differential Equations of Motion

The differential equations of motion for the orbit transfer problem were derived using 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.

2.1.1 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

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

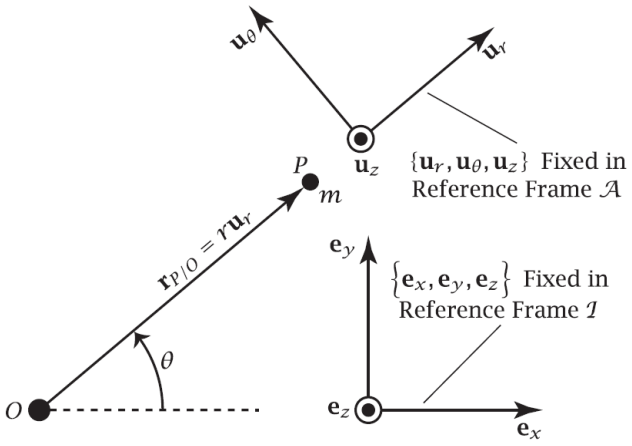

Figure 1: Schematic of particle moving in an inertially fixed plane

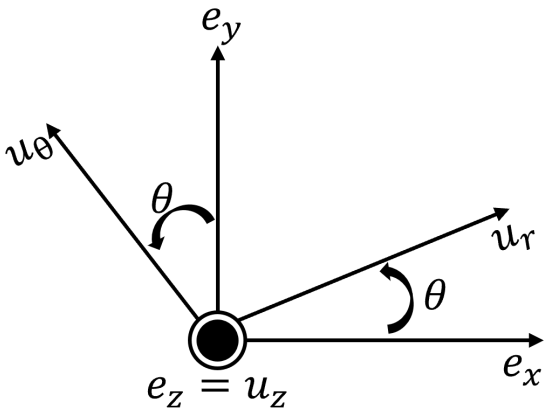

Figure 2: Reference frame rotation

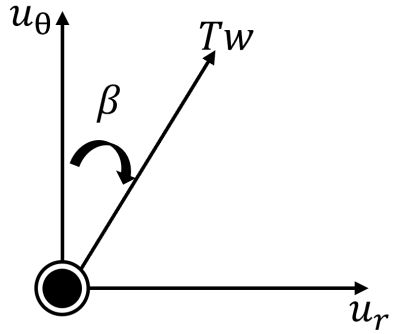

Figure 3: Thrust Force at Angle β

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

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

The acceleration of the particle can then be formulated as

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

2.1.2 Newton's Second Law for A Particle

Newton's second law for a particle is represented by

$$\sum F_P = m_P * a_P. \tag{7}$$

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

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

while the gravitational force can be written as

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

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

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

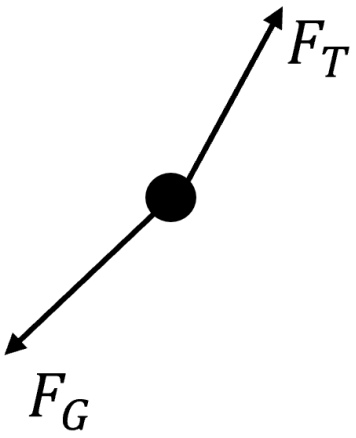

Figure 4: Free Body Diagram

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

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

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

2.1.3 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:

$$\dot{r} = v_r, \tag{12}$$

$$r\dot{\theta} = v_\theta,$$

$$\dot{\theta} = \frac{v_\theta}{r}. \tag{13}$$

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:

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

2.2 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:

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

2.2.1 Boundary Conditions

The boundary conditions for this problem are as followed:

$$\begin{aligned}
r(t_0) = r_0 &= 1, \\
\theta(t_0) = \theta_0 &= 0, \\
v_r(t_0) = v_{r_0} &= 0, \\
v_\theta(t_0) = v_{\theta_0} &= \sqrt{\frac{\mu}{r_0}}, \\
m(t_0) = m_0 &= 1, \\
r(t_f) = r_f &= 1, \\
\theta(t_f) &= free, \\
v_r(t_f) = v_{r_f} &= 0, \\
v_\theta(t_f) = v_{\theta_f} &= \sqrt{\frac{\mu}{r_f}}, \\
m(t_f) &= free.
\end{aligned}$$

2.2.2 Control Parameterization

There are two different sets of controls for the optimal control problem. The objectives will first be minimized with the controls being thrust angle β, and thrust magnitude T. The objectives will then be minimized by making the following substitutions into the differential equations:

$$u_1 = \sin(\beta), \tag{19}$$

$$u_2 = \cos(\beta). \tag{20}$$

Now the controls are u_1, u_2, and the thrust magnitude T. When applying this set of controls, the following path constraint must be made:

$$u_1^2 + u_2^2 = 1. \tag{21}$$

This path constraint is necessary due to the trigonometric identity $\sin^2\beta + \cos^2\beta = 1$. Also, the thrust magnitude is constrained as follows:

$$0 \leq T \leq T_{max}.$$

3 Results/Individual Case Analysis

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

For all cases the following parameters are used:

$$\begin{aligned} \mu &= 1, \\ v_e &= 1.8758344, \\ T_{max} &= 0.1405. \end{aligned}$$

3.1.1 Minimize Terminal Time with Unconstrained Control

The optimal control problem was first solved with the objection function being to minimize t_f with unconstrained control parameterized by polynomials of degree N = (3,4) for intervals of K = (2,4,8,16,32). The controls were thrust angle β and thrust magnitude T. The statistics for each combination are shown in Table 1. A comparison of the objective function for each combination is shown in Figure 5. Interestingly, the objective function was best optimized for the case of polynomial degree 4 with 2 intervals, which resulted in a terminal time of 3.2456. Taking a closer look at the orbit transfer for this case, Figure 24, it is noticeable that the first two segments of the transfer slightly cross the initial orbit radius. While the lower bound of the radius, r, was set to 1, the path still violated that criteria. This is due to the constraints only being evaluated at the LGR points and not in-between them. For this case the optimizer obeyed its constraints but produced a trajectory that is both undesirable and unrealistic. If an the additional constraint was added such that a segment conjoining two LGR points can not violate a specific radius, then the objective function may not have been as optimized. Of these cases, polynomial degree 3 with 2 intervals seems to be an outlier. The objective function for this case, t_f, is approximately 4.6% greater than any of the other combinations. Looking at Figure 8 it is noticeable that the thrust drops from the upper limit down to zero at the third LGR point. For the objective function to be fully optimized, the thrust should have been maximized the entire time. This means that the IPOPT had trouble approximating the solution at the 3^{rd} LGR point. Also, N = 3 and K = 2 is the only case where the control β does not resemble that of the other combinations. All the other combinations show similar results to those obtained using Indirect Shooting in the midterm. Lastly, the comparison of the execution time between all of the combinations is shown in Figure 6. Interestingly, the execution time does not increase very much as the number of intervals increases. While there is

a sight increase in execution time as polynomial degree and number of intervals increases, it is not nearly as drastic as Multiple Shooting. This is most likely due to collocation being much more efficient since it solves a mass system of equations.

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	18	0.209	3.4087	0.79163	0
3	4	62	0.514	3.2457	0.75559	0
3	8	99	0.677	3.2467	0.75552	0
3	16	151	0.994	3.247	0.7555	1
3	32	127	1.2	3.2469	0.7555	1
4	2	39	0.345	3.2456	0.7556	0
4	4	87	0.595	3.2466	0.75552	0
4	8	140	0.915	3.247	0.7555	0
4	16	143	0.867	3.247	0.7555	0
4	32	160	0.982	3.2469	0.7555	1

Table 1: Results for minimizing t_f with unconstrained control

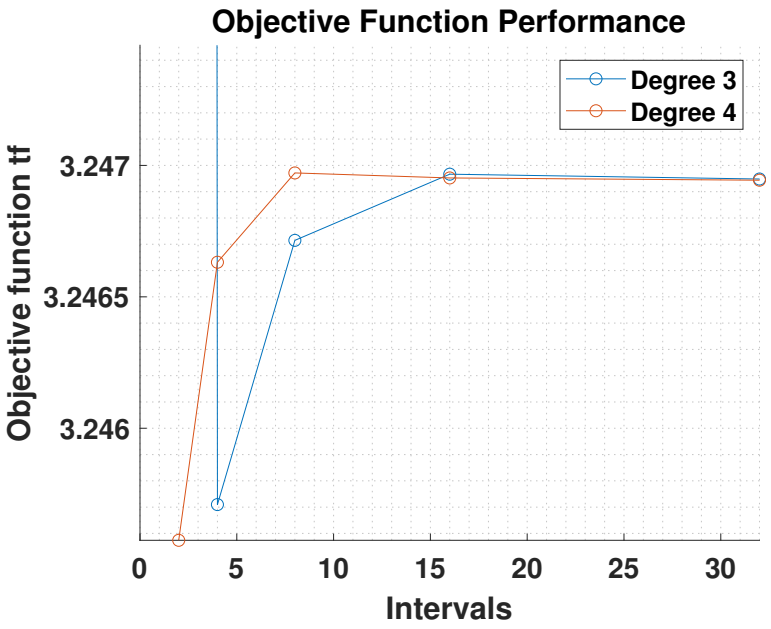

Figure 5: Objective function performance with unconstrained controls

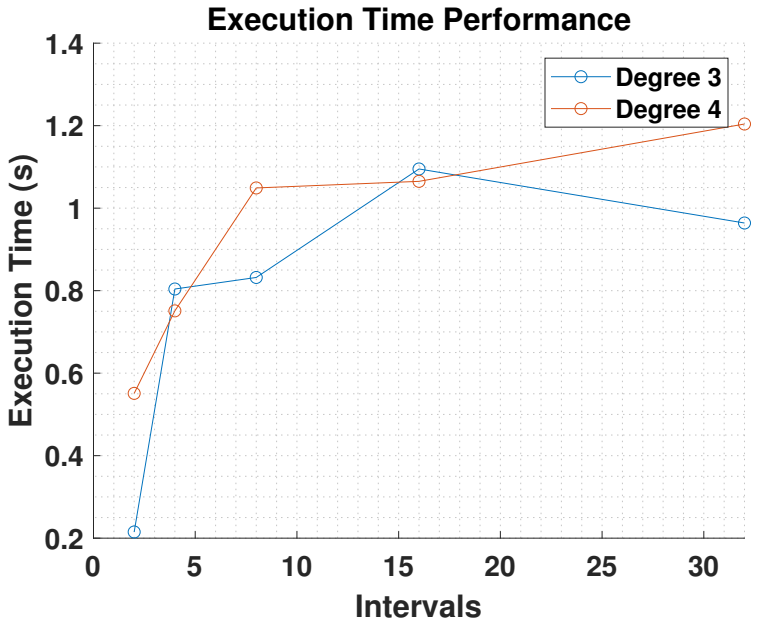

Figure 6: Execution performance of minimizing terminal time with unconstrained controls

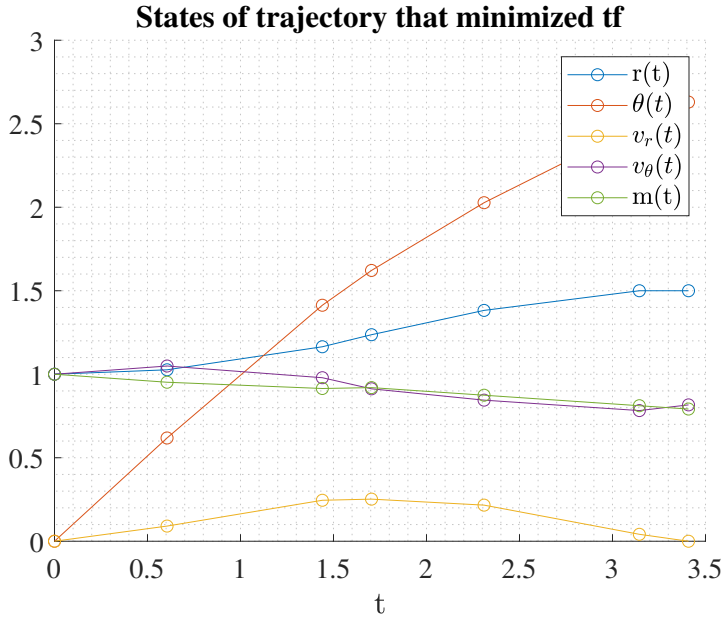

Figure 7: States for trajectory that minimized terminal time ($N : 3$, $K : 2$)

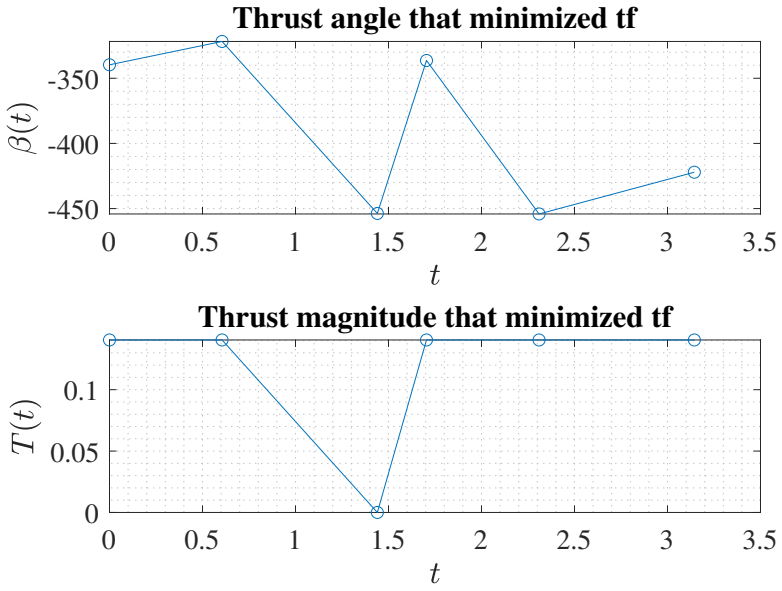

Figure 8: Control that minimized terminal time ($N : 3$, $K : 2$)

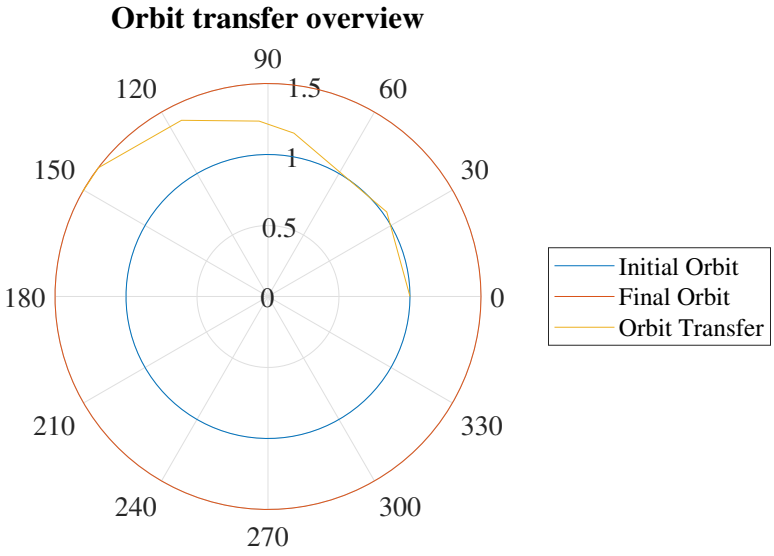

Figure 9: Trajectory from initial to final orbit ($N : 3$, $K : 2$)

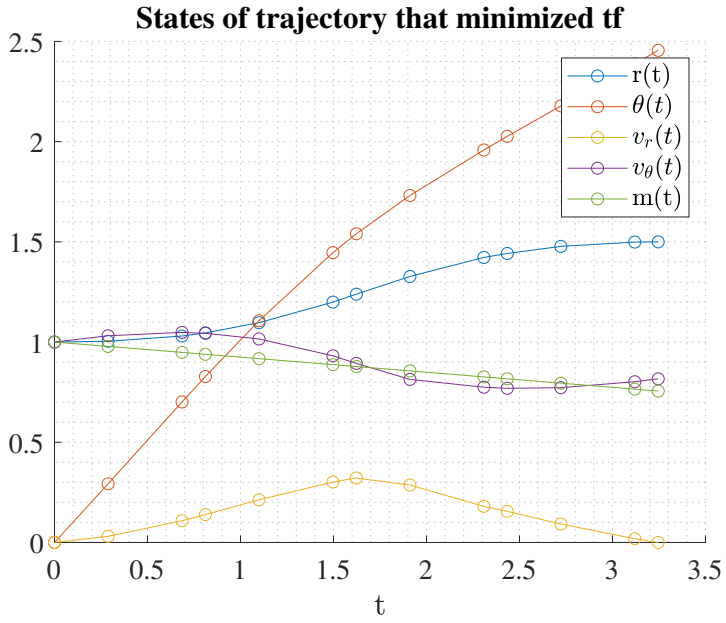

Figure 10: States for trajectory that minimized terminal time ($N : 3$, $K : 4$)

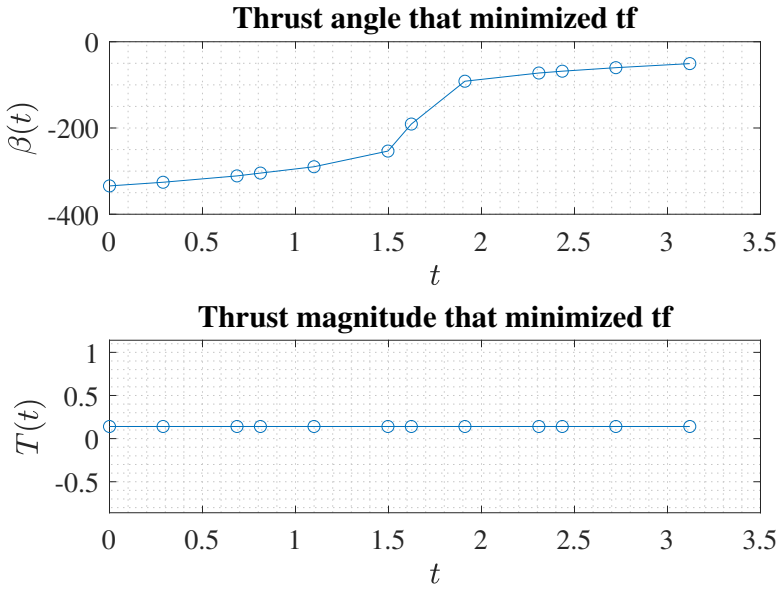

Figure 11: Control that minimized terminal time ($N : 3$, $K : 4$)

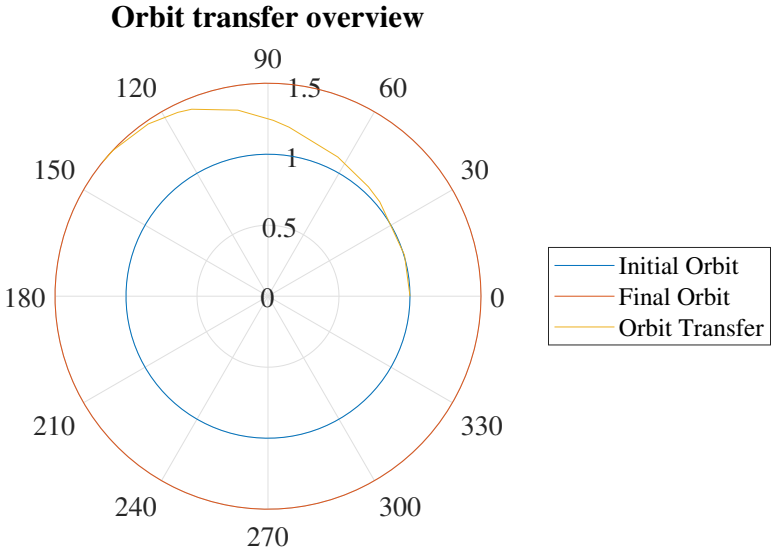

Figure 12: Trajectory from initial to final orbit ($N : 3$, $K : 4$)

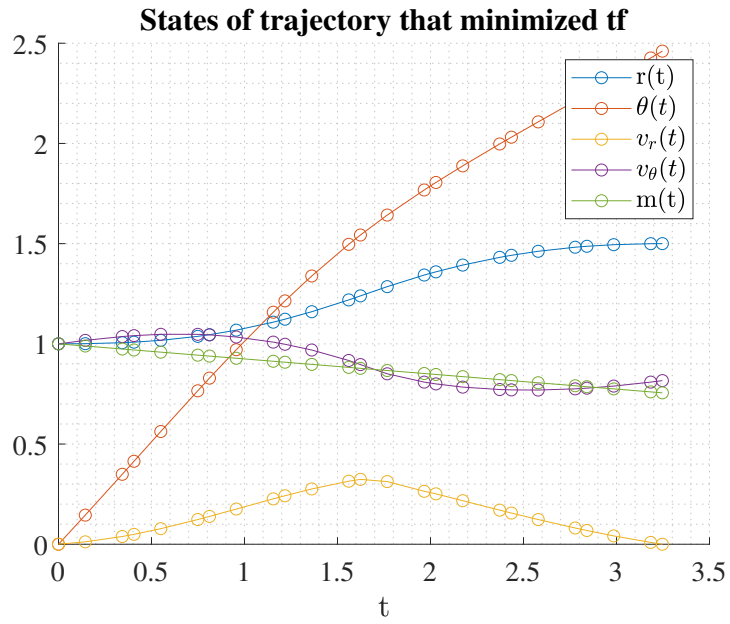

Figure 13: States for trajectory that minimized terminal time ($N : 3$, $K : 8$)

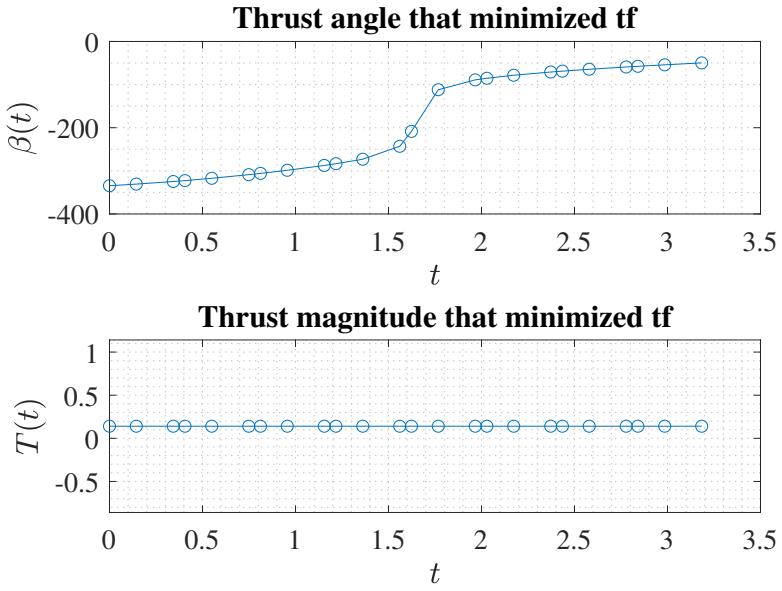

Figure 14: Control that minimized terminal time ($N : 3$, $K : 8$)

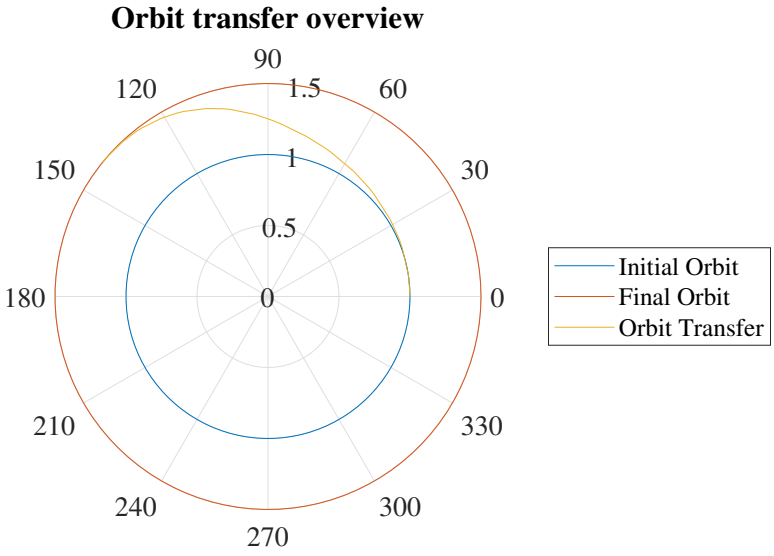

Figure 15: Trajectory from initial to final orbit ($N : 3$, $K : 8$)

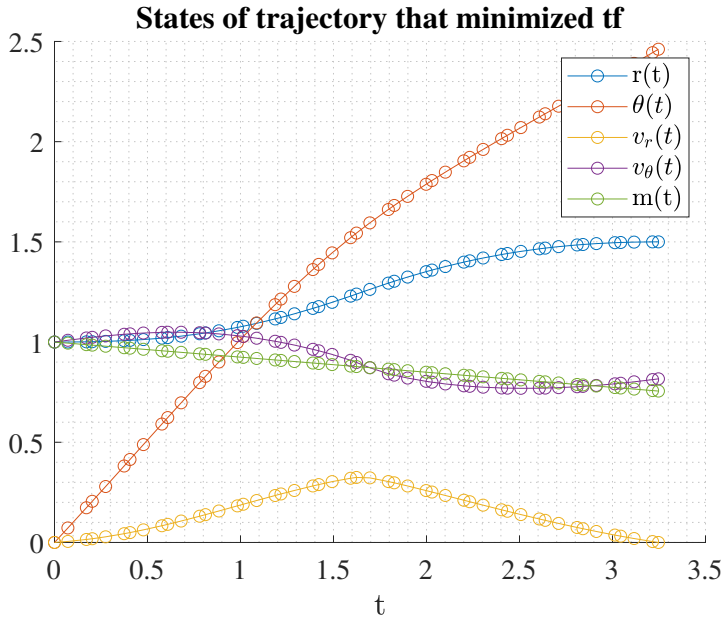

Figure 16: States for trajectory that minimized terminal time ($N : 3$, $K : 16$)

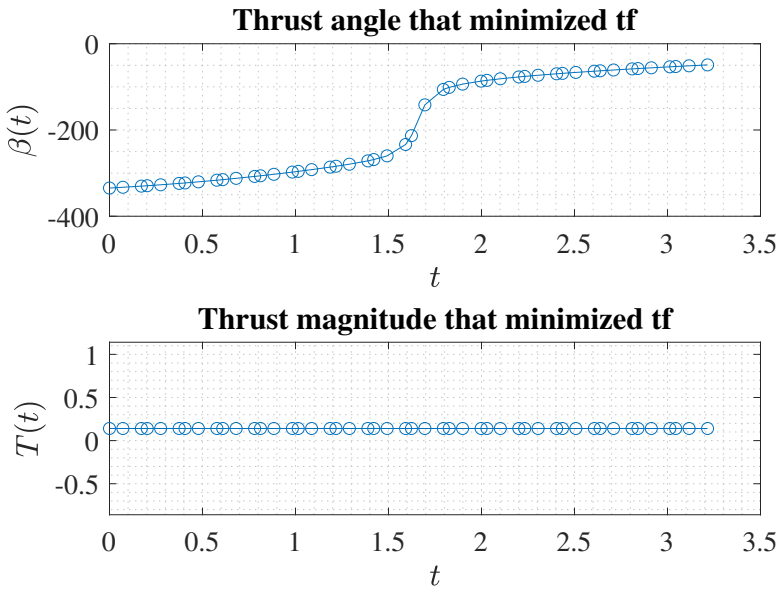

Figure 17: Control that minimized terminal time ($N : 3$, $K : 16$)

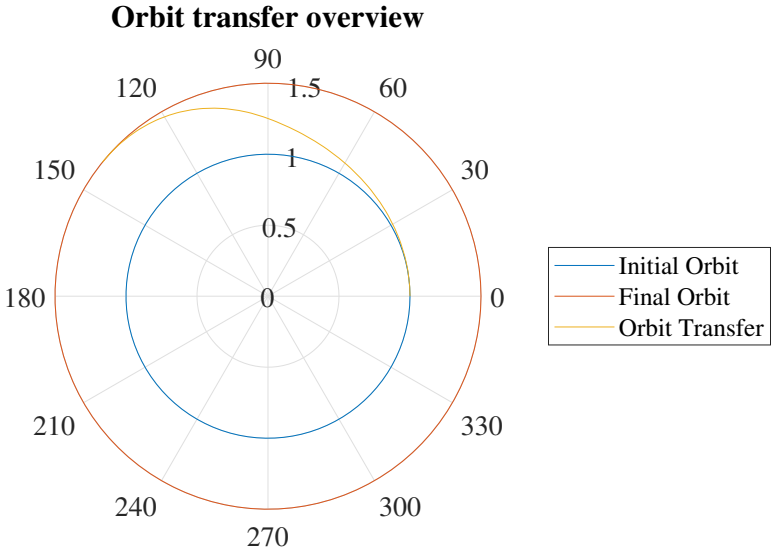

Figure 18: Trajectory from initial to final orbit ($N : 3$, $K : 16$)

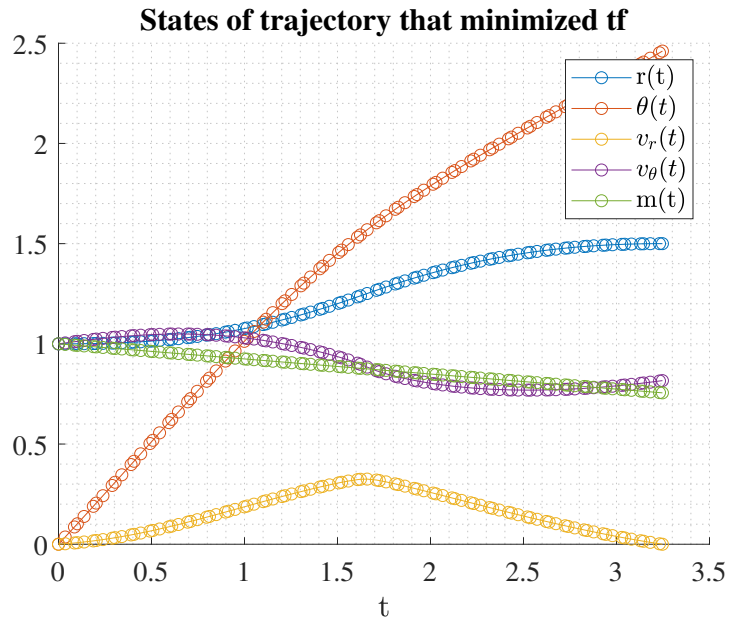

Figure 19: States for trajectory that minimized terminal time ($N : 3$, $K : 32$)

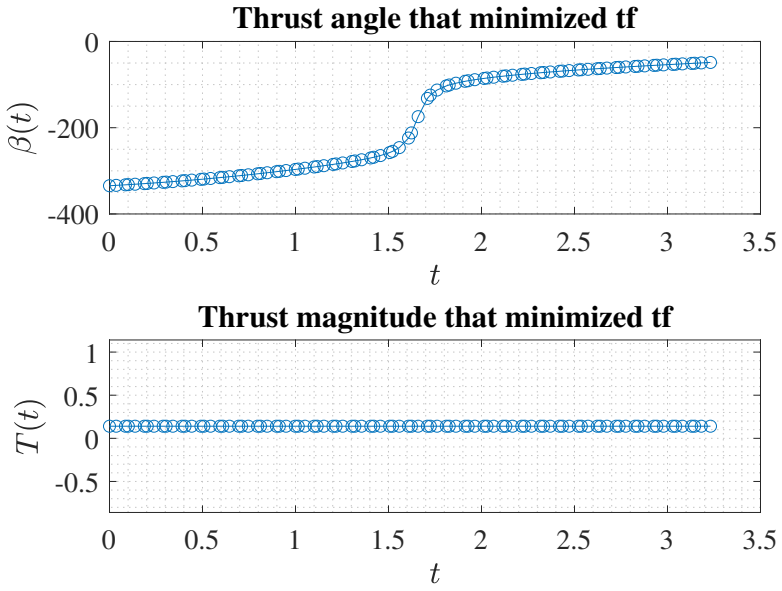

Figure 20: Control that minimized terminal time ($N : 3$, $K : 32$)

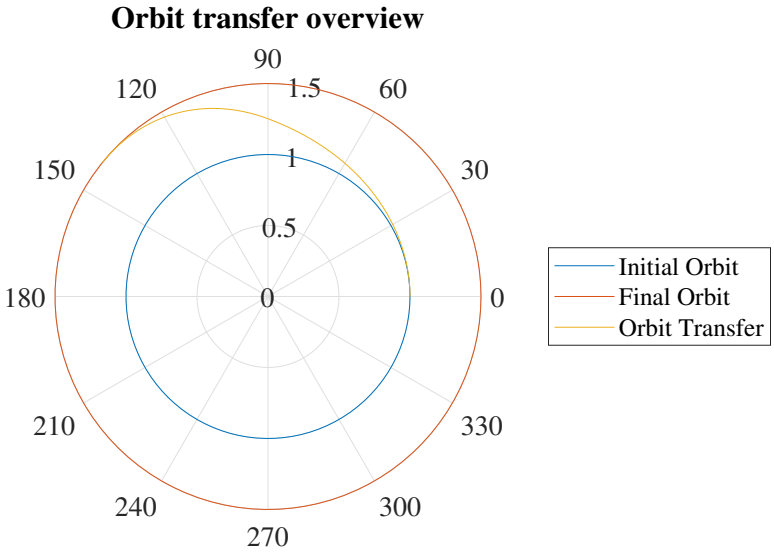

Figure 21: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

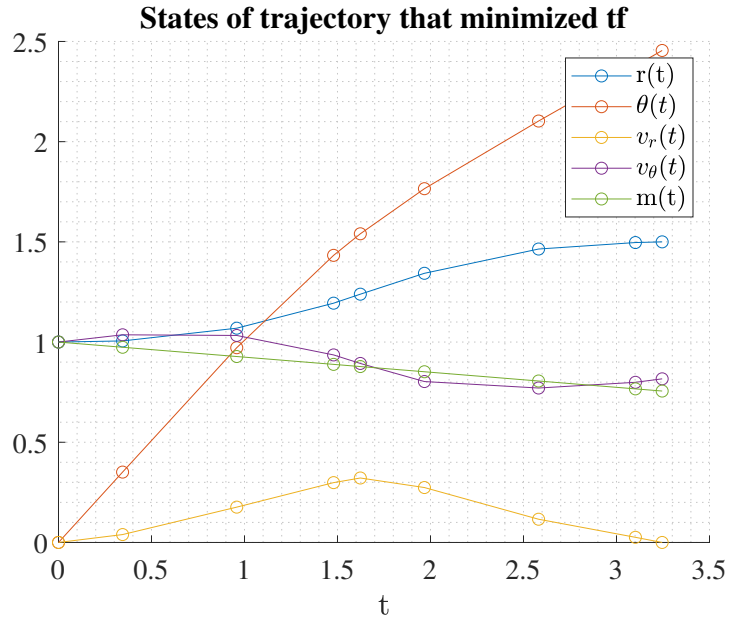

Figure 22: States for trajectory that minimized terminal time ($N : 4$, $K : 2$)

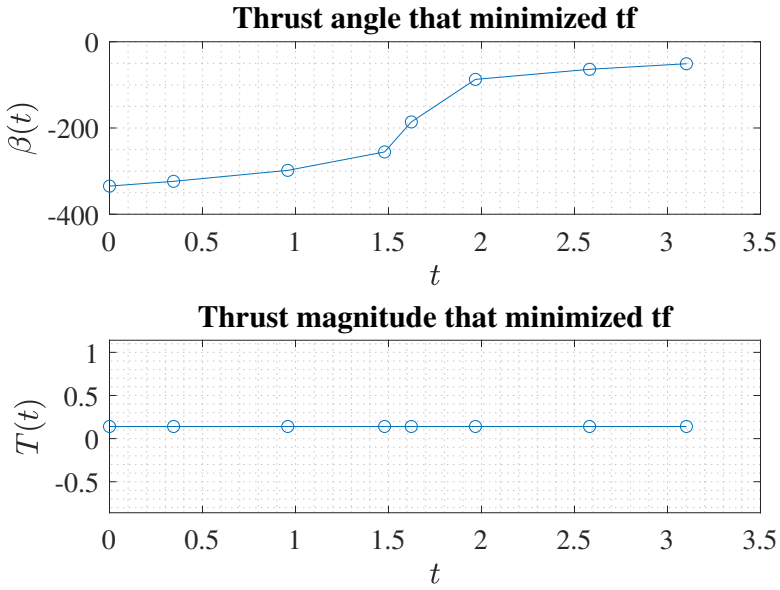

Figure 23: Control that minimized terminal time ($N : 4$, $K : 2$)

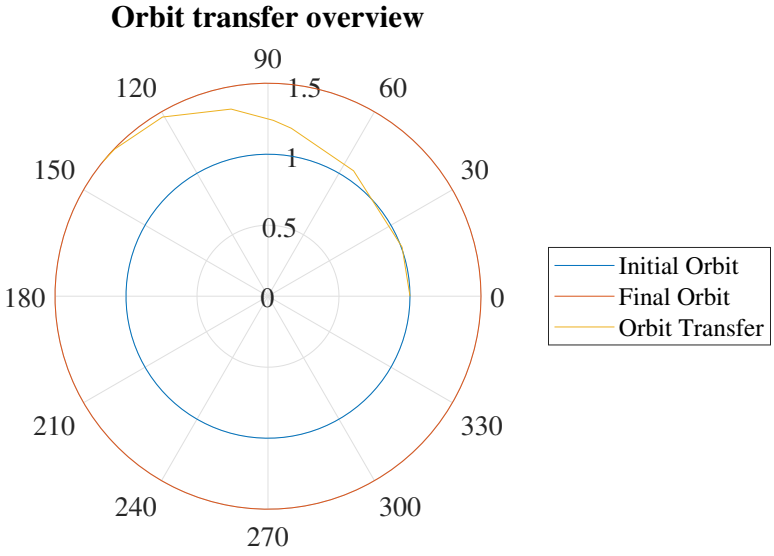

Figure 24: Trajectory from initial to final orbit ($N : 4$, $K : 2$)

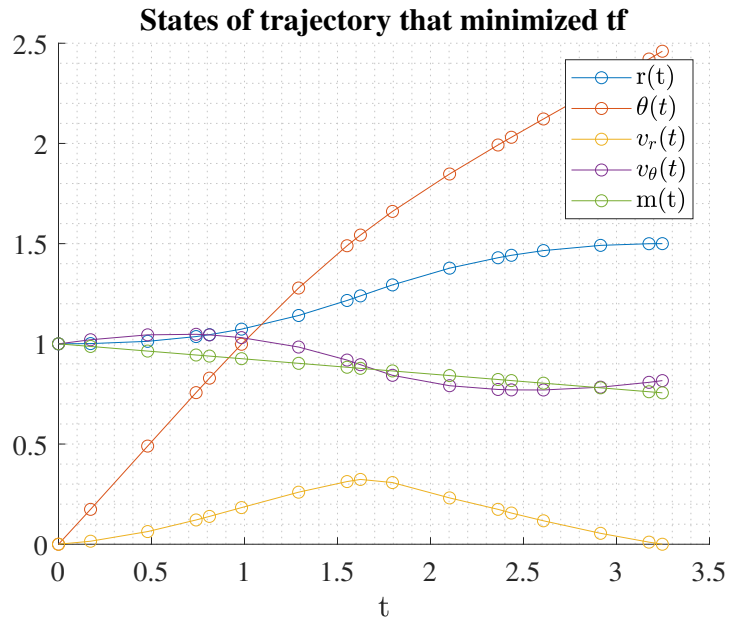

Figure 25: States for trajectory that minimized terminal time ($N : 4$, $K : 4$)

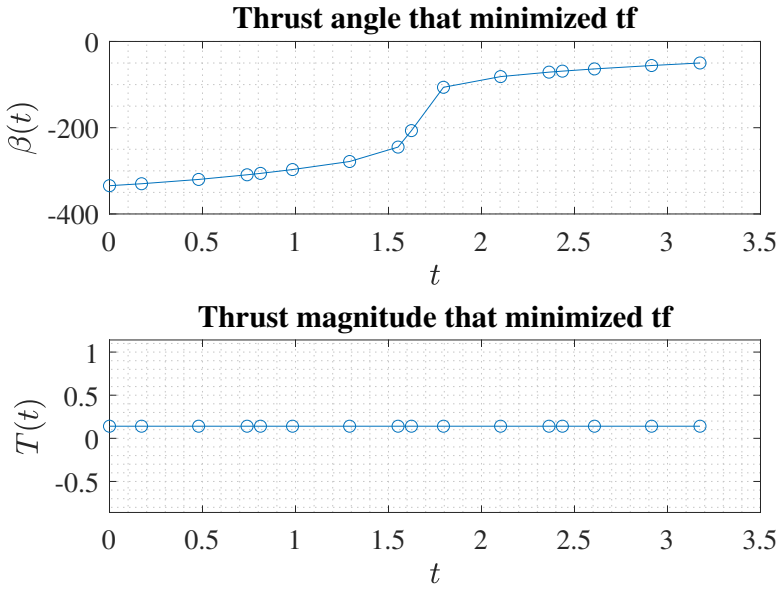

Figure 26: Control that minimized terminal time ($N : 4$, $K : 4$)

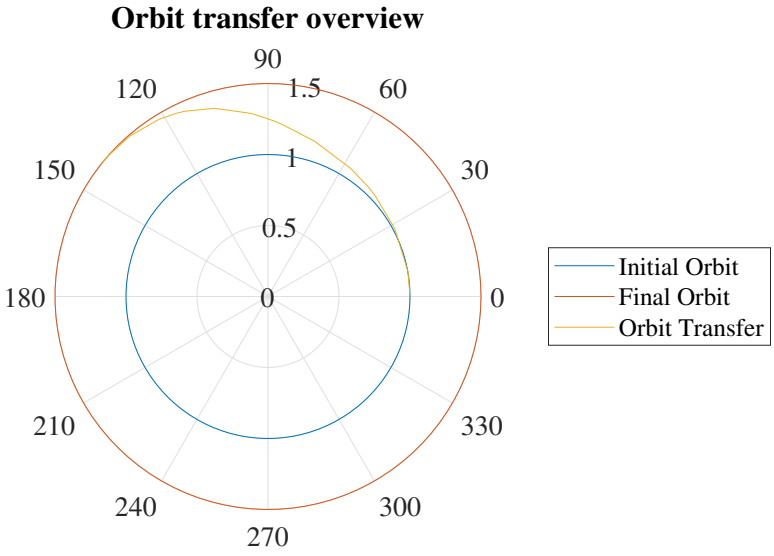

Figure 27: Trajectory from initial to final orbit ($N : 4$, $K : 4$)

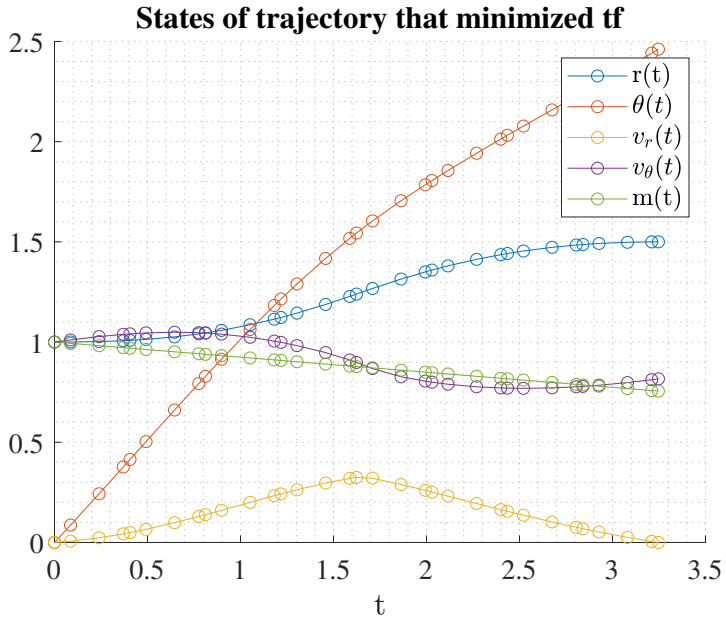

Figure 28: States for trajectory that minimized terminal time ($N : 4$, $K : 8$)

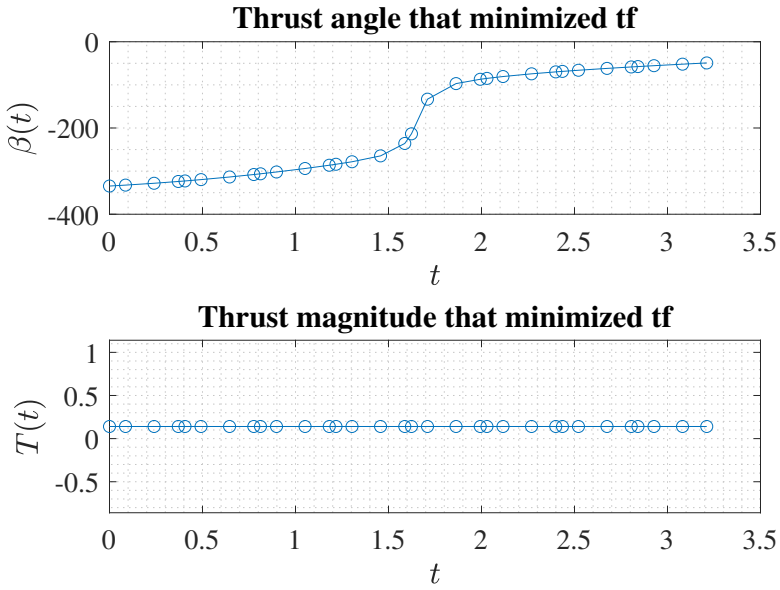

Figure 29: Control that minimized terminal time ($N : 4$, $K : 8$)

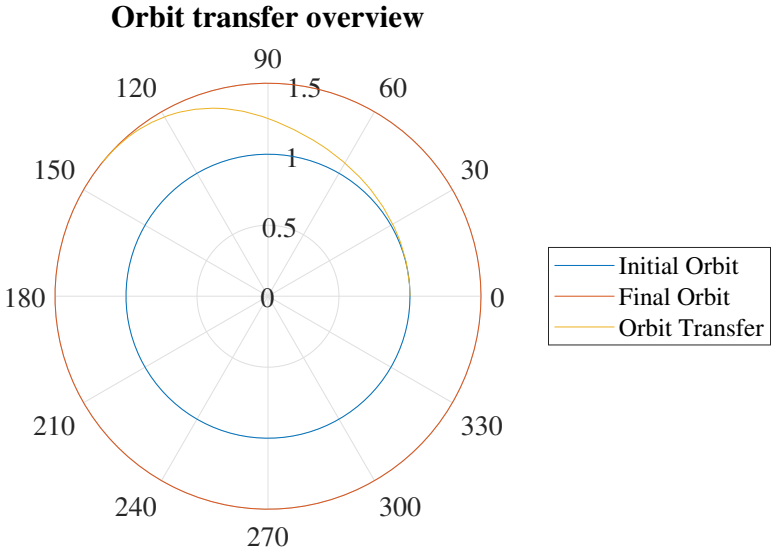

Figure 30: Trajectory from initial to final orbit ($N : 4$, $K : 8$)

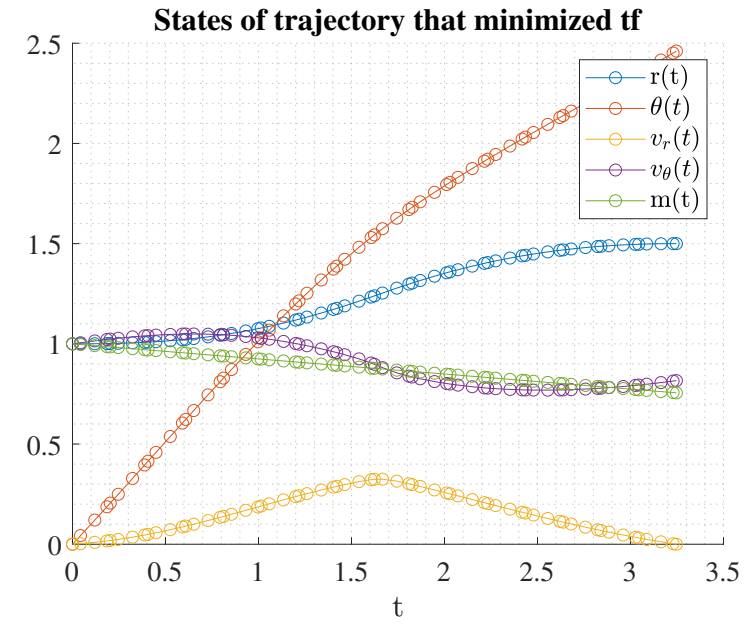

Figure 31: States for trajectory that minimized terminal time ($N : 4$, $K : 16$)

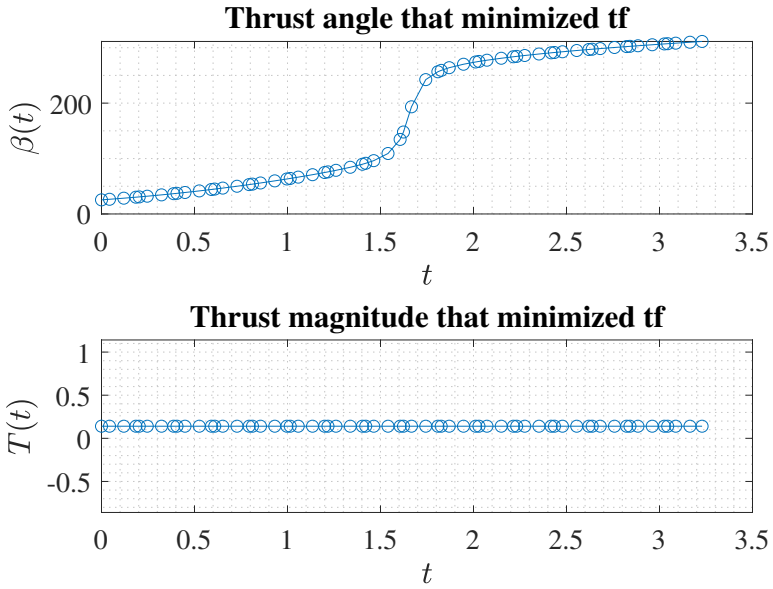

Figure 32: Control that minimized terminal time ($N : 4$, $K : 16$)

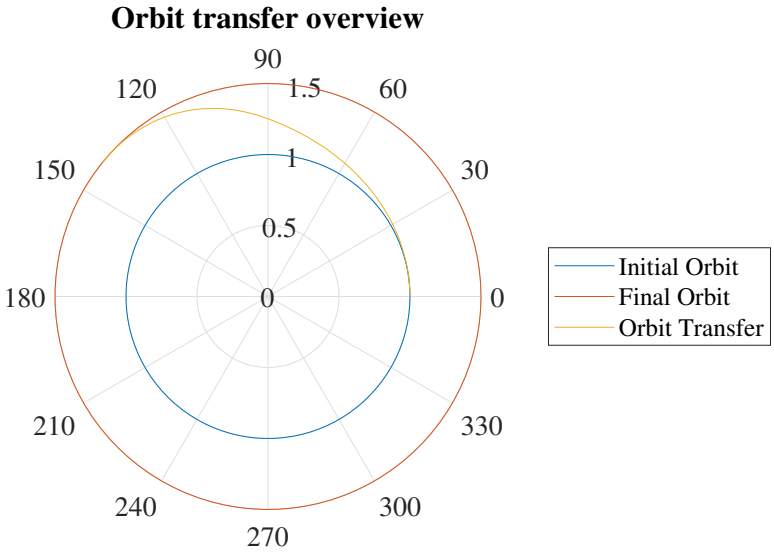

Figure 33: Trajectory from initial to final orbit ($N : 4$, $K : 16$)

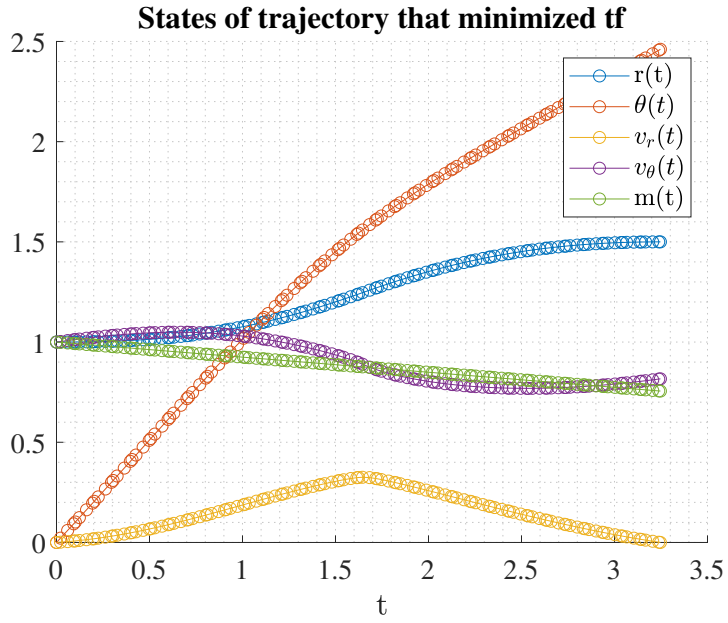

Figure 34: States for trajectory that minimized terminal time ($N : 4$, $K : 32$)

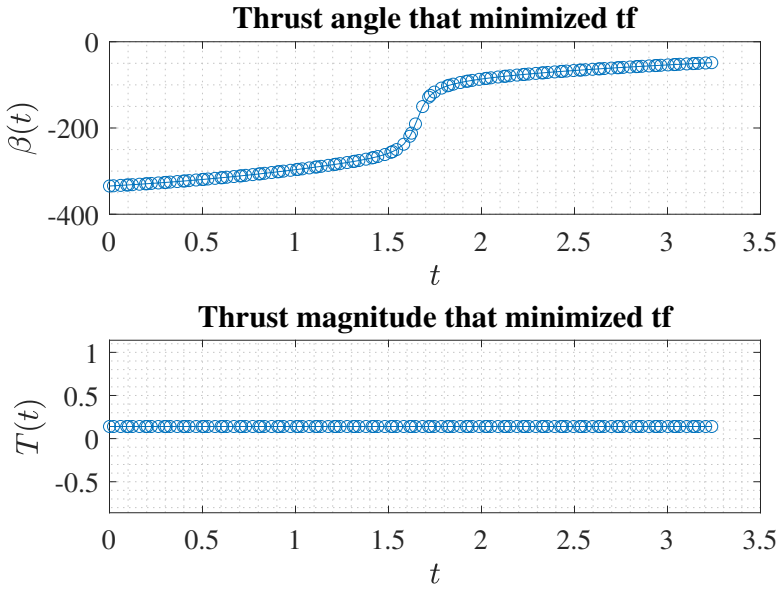

Figure 35: Control that minimized terminal time ($N : 4$, $K : 32$)

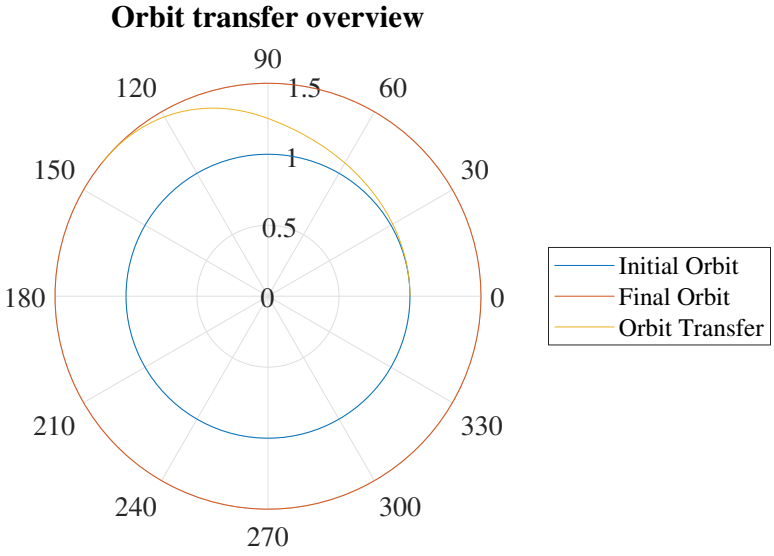

Figure 36: Trajectory from initial to final orbit ($N : 4$, $K : 32$)

3.1.2 Maximize Terminal Mass with Unconstrained Control

The optimal control problem was then solved with the objection function being to maximize m_f with unconstrained control parameterized by polynomials of degree N = (3,4) for intervals of K = (2,4,8,16,32). For these cases the controls were thrust angle β and thrust magnitude T. The statistics for each combination are shown in Table 2. Figure 37 shows that the objective function, mf, was best optimized by a control parameratized by a 3^{rd} degree polynomial with 2 intervals. Just like the previous case, a 2-interval combination optimized the objective function the best. Taking a look at the planer view of the orbit transfer in Figure 41, it is obvious that the transfer does not look realistic. In other words, the first segment cuts directly through the unit circle although the two LGR points are satisfying the radius constraint. Just like in the previous case, the solution could be made more realistic by either adding an additional constraint so that no point on a segment can cross the radius bounds or by simply increasing the number of intervals. While the terminal mass for polynomial degree 3 with 2 intervals was maximized at 0.909, the rest of the combinations converged to a terminal mass of approximately 0.9072. It is also interesting that the objective function converges to a specific value as you increase the number of intervals, regardless of the polynomial degree. Figure 37 shows this phenomena. It's also noticeable that the thrust control for each of the combinations consists of an impulse at the begging and end of the transfer. This is expected since the objective function is to maximize the terminal mass. It is known that the most fuel-efficient orbit maneuver is a Holman Transfer that consists of two impulse maneuvers which is what the thrust magnitude is resembling for this problem. Another thing to point out about this problem is that the control angle β is not well contained for any of the combinations. For instance, Figure 46 (N:3 K:8), shows that β sweeps from 0°down to approximately -900°. This is an undesirable control that could cause problems with not only the optimizer but also the hardware. Lastly, it is noticeable that a significant amount of the combinations have "chattering" in their β control. This phenomena is not completely understood but it is most likely due to numerical integration errors. The chattering seems to be extremely sensitive to the bounds and initial guesses of the variables. It may be possible to add another path constraint on beta such that the angle can not bounce around.

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	111	0.593	10.5213	0.90901	0
3	4	201	1.001	5.1606	0.9072	0
3	8	99	0.585	7.2433	0.90714	0
3	16	92	0.585	6.6139	0.9072	0
3	32	128	0.737	7.7283	0.90721	0
4	2	65	0.468	5.0469	0.90717	0
4	4	130	0.73	5.2922	0.90719	0
4	8	230	1.171	8.2586	0.9072	0
4	16	97	0.593	7.8959	0.9072	0
4	32	291	1.637	11.9018	0.9072	0

Table 2: Results for maximizing m_f with unconstrained control

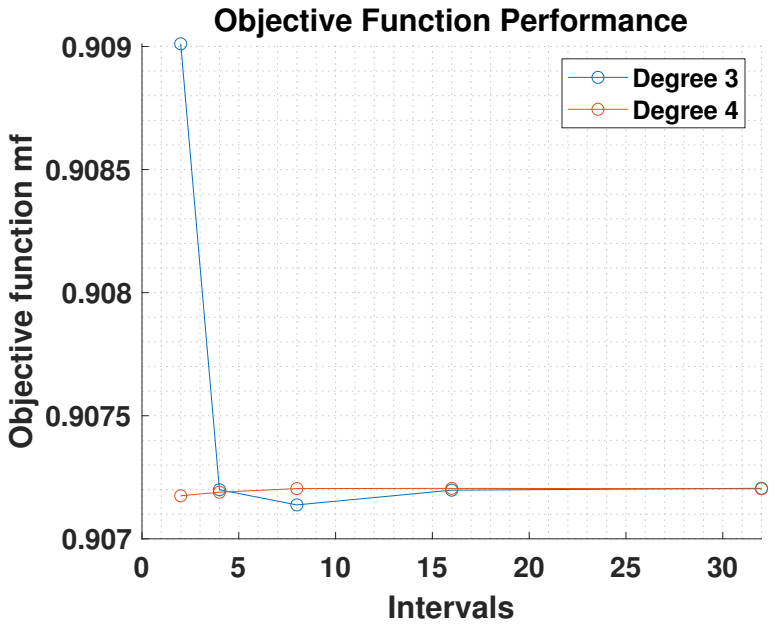

Figure 37: Objective function performance with unconstrained controls

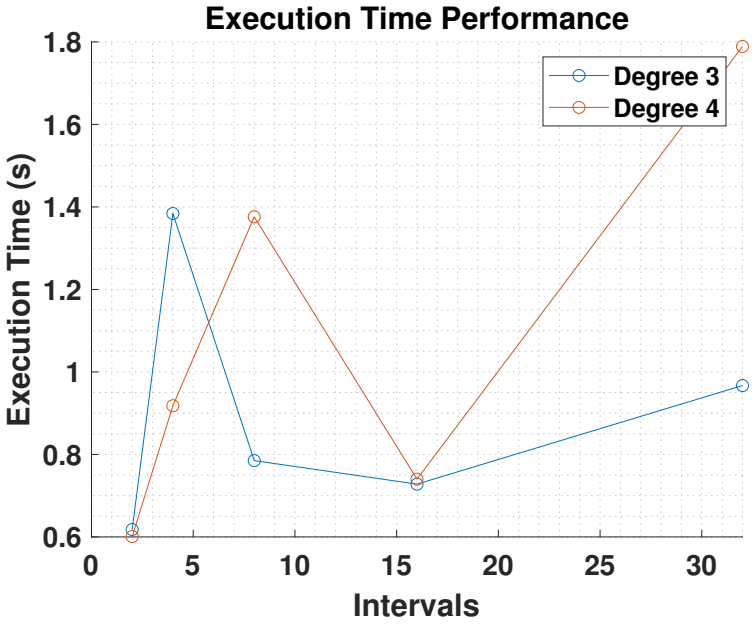

Figure 38: Execution performance of maximizing terminal mass with unconstrained controls

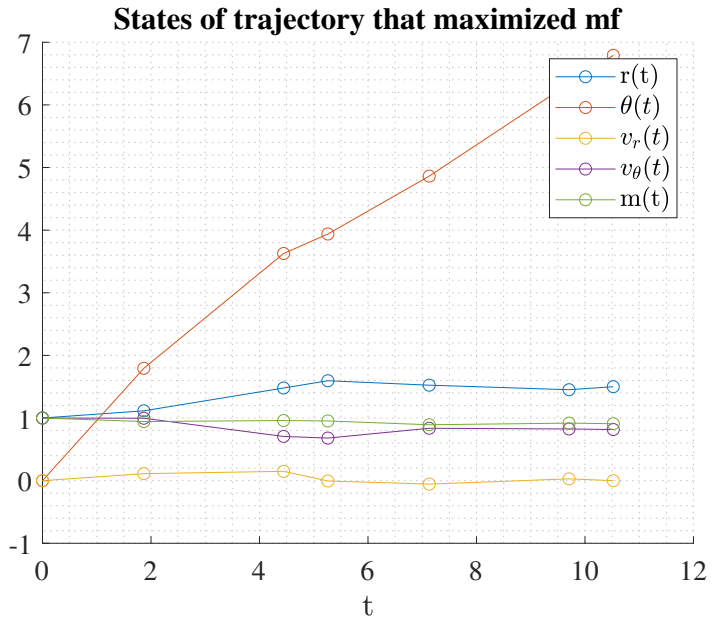

Figure 39: States for trajectory that maximized terminal mass ($N : 3$, $K : 2$)

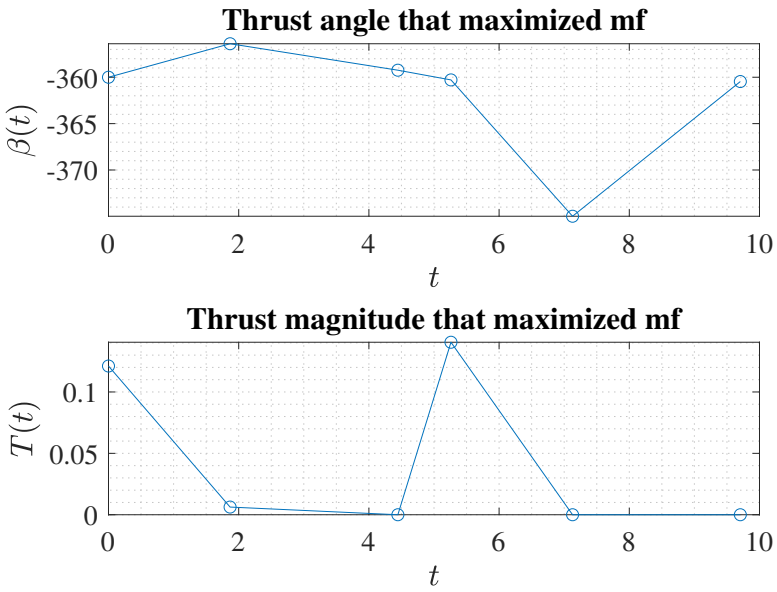

Figure 40: Control that maximized terminal mass ($N : 3$, $K : 2$)

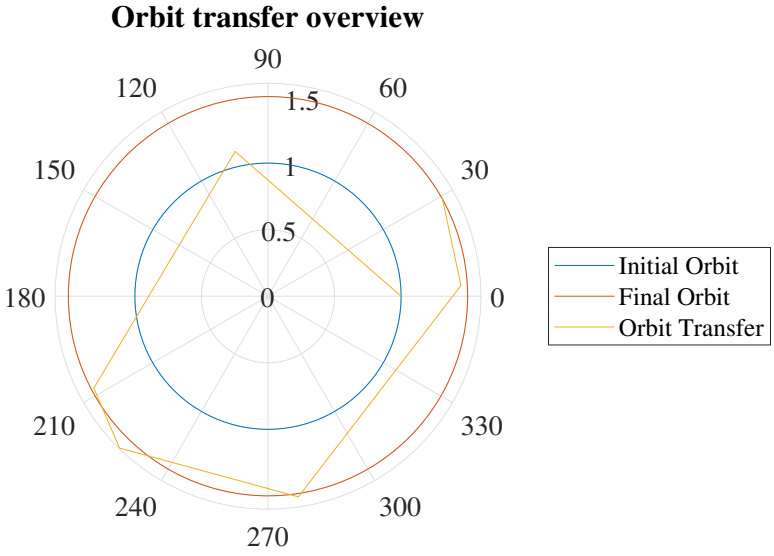

Figure 41: Trajectory from initial to final orbit ($N : 3$, $K : 2$)

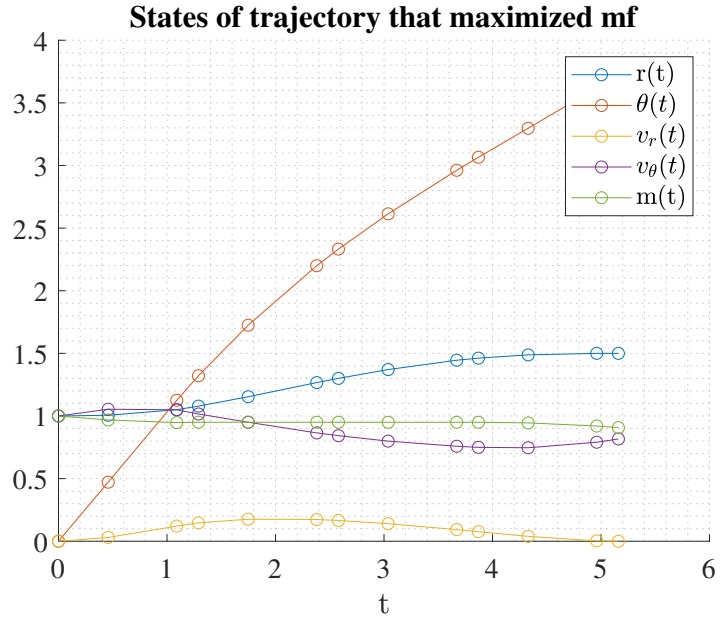

Figure 42: States for trajectory that maximized terminal mass ($N : 3$, $K : 4$)

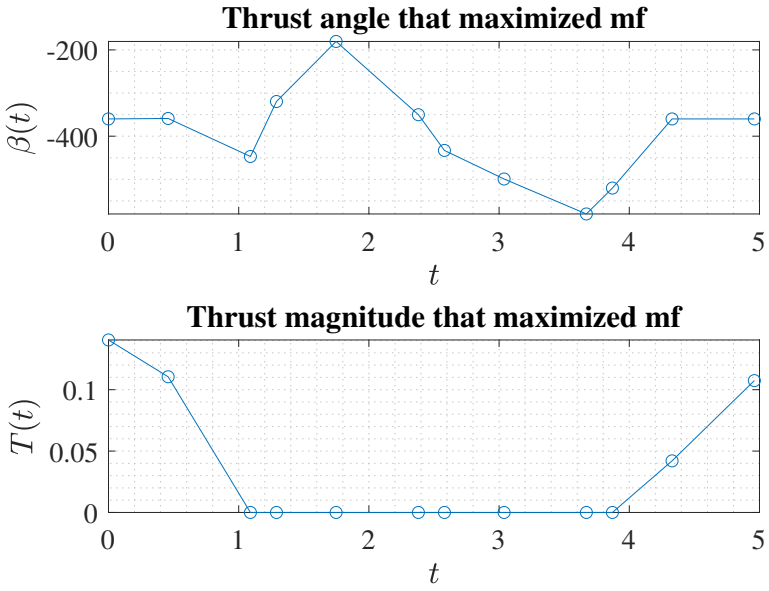

Figure 43: Control that maximized terminal mass ($N : 3$, $K : 4$)

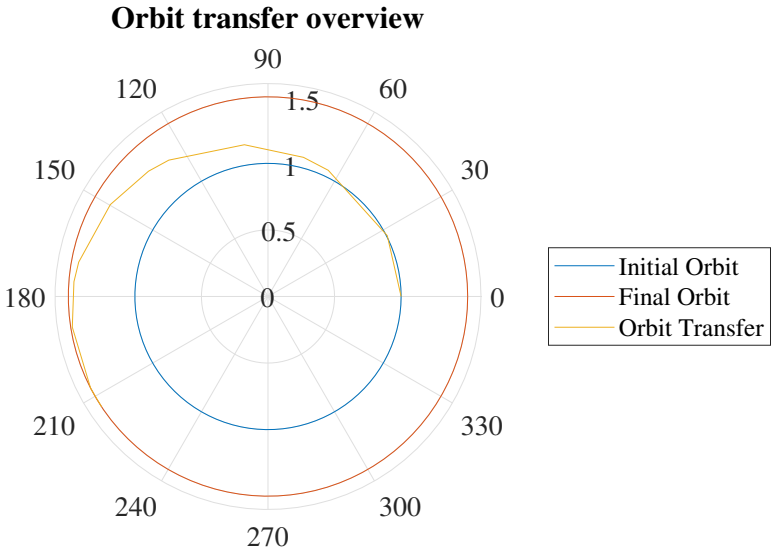

Figure 44: Trajectory from initial to final orbit ($N : 3$, $K : 4$)

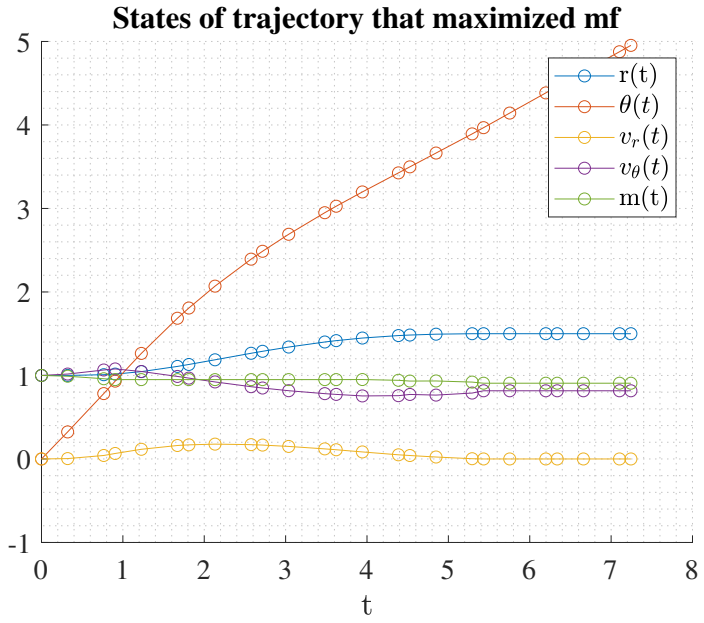

Figure 45: States for trajectory that maximized terminal mass ($N : 3$, $K : 8$)

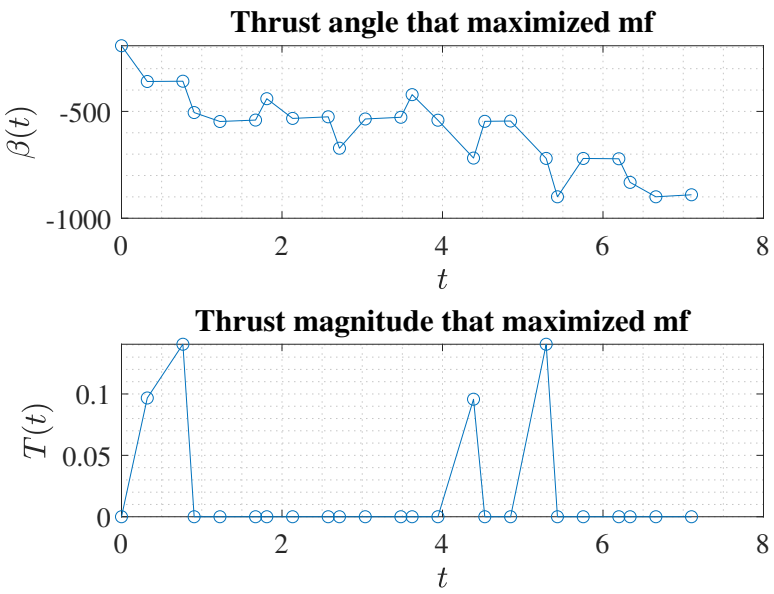

Figure 46: Control that maximized terminal mass ($N : 3$, $K : 8$)

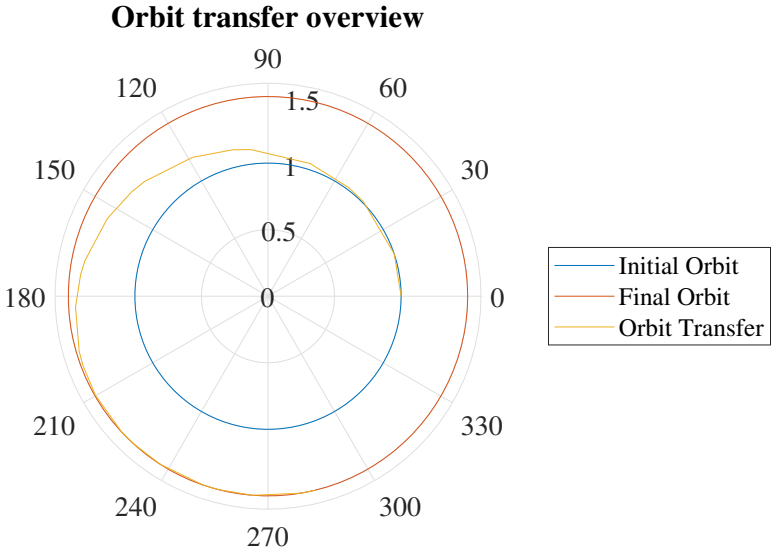

Figure 47: Trajectory from initial to final orbit ($N : 3$, $K : 8$)

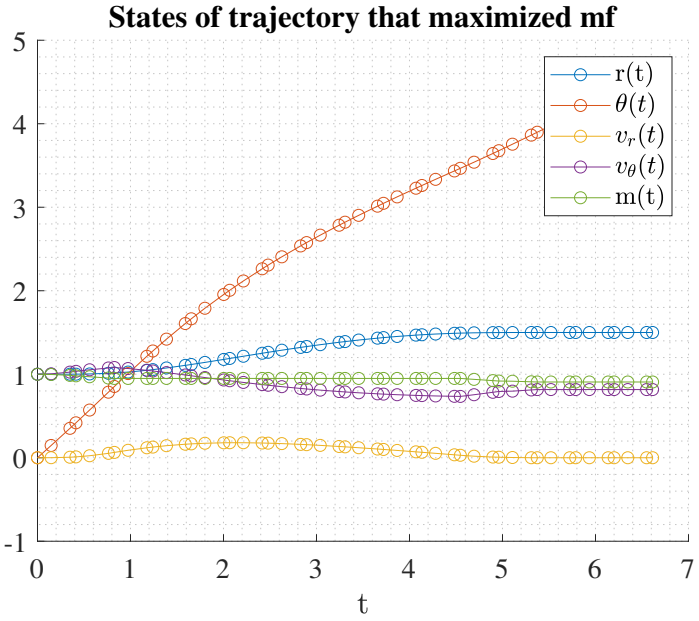

Figure 48: States for trajectory that maximized terminal mass ($N : 3$, $K : 16$)

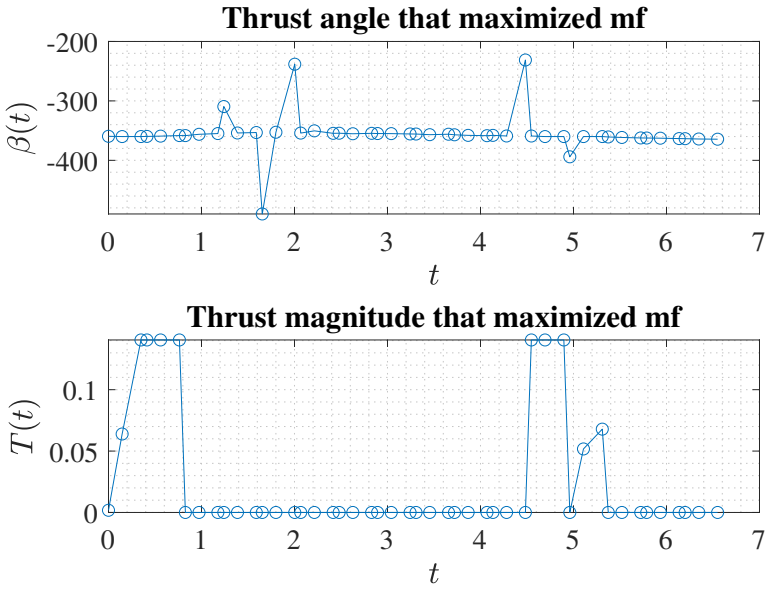

Figure 49: Control that maximized terminal mass ($N : 3$, $K : 16$)

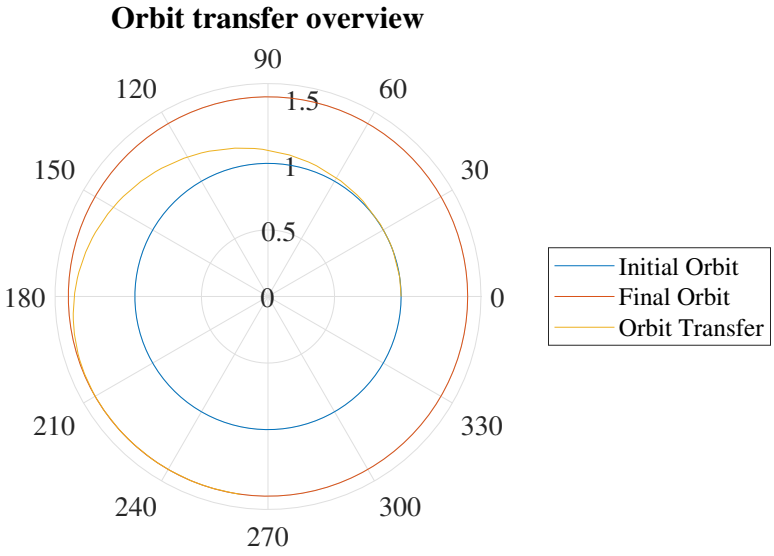

Figure 50: Trajectory from initial to final orbit ($N : 3$, $K : 16$)

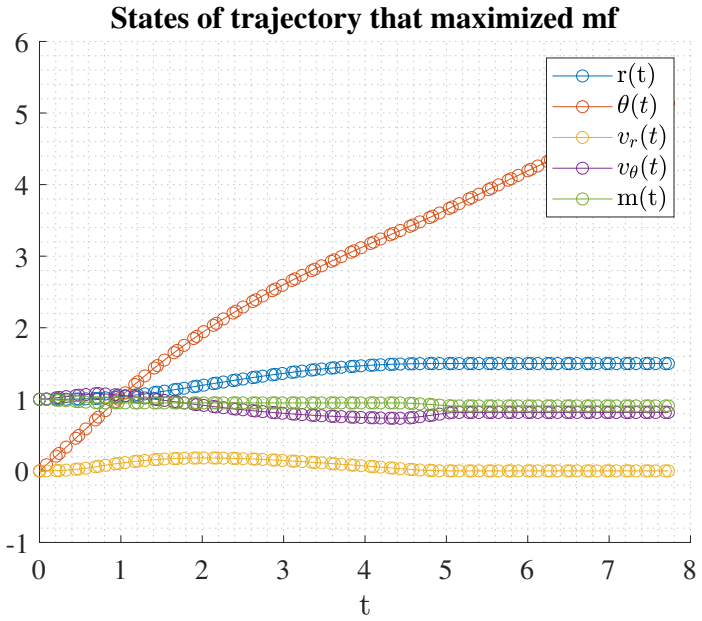

Figure 51: States for trajectory that maximized terminal mass ($N : 3$, $K : 32$)

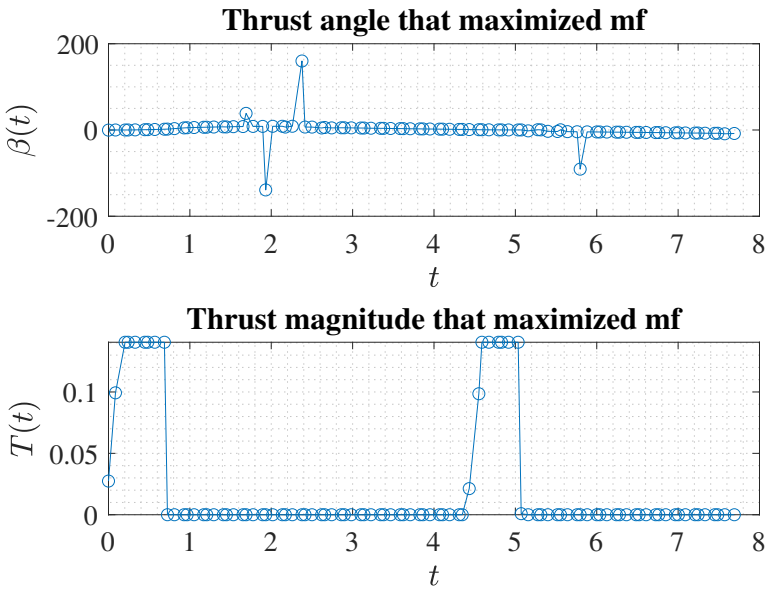

Figure 52: Control that maximized terminal mass ($N : 3$, $K : 32$)

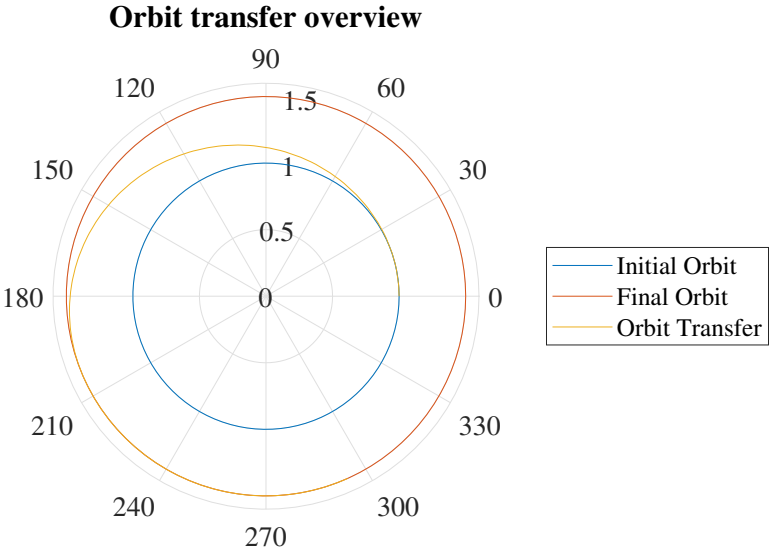

Figure 53: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

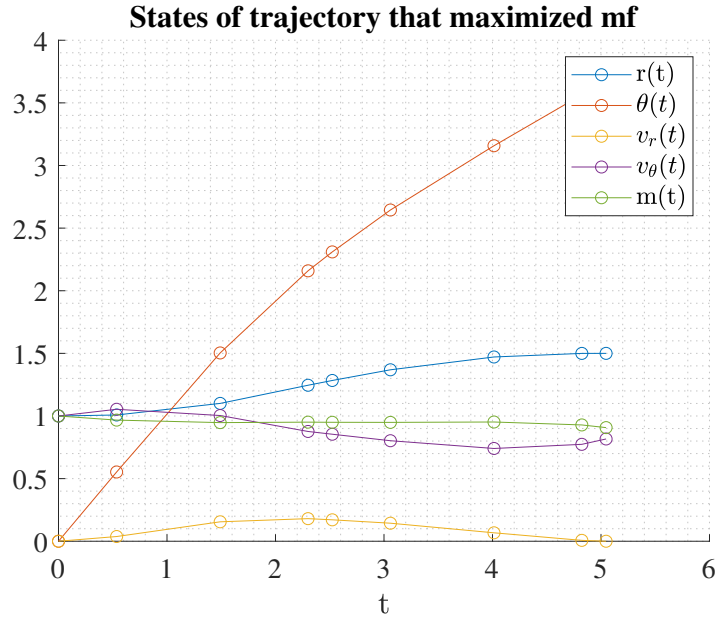

Figure 54: States for trajectory that maximized terminal mass ($N : 4$, $K : 2$)

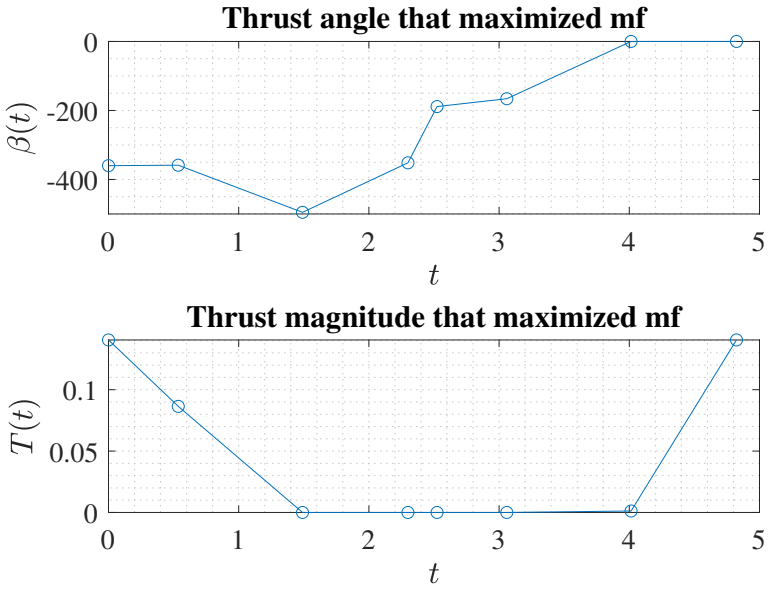

Figure 55: Control that maximized terminal mass ($N : 4$, $K : 2$)

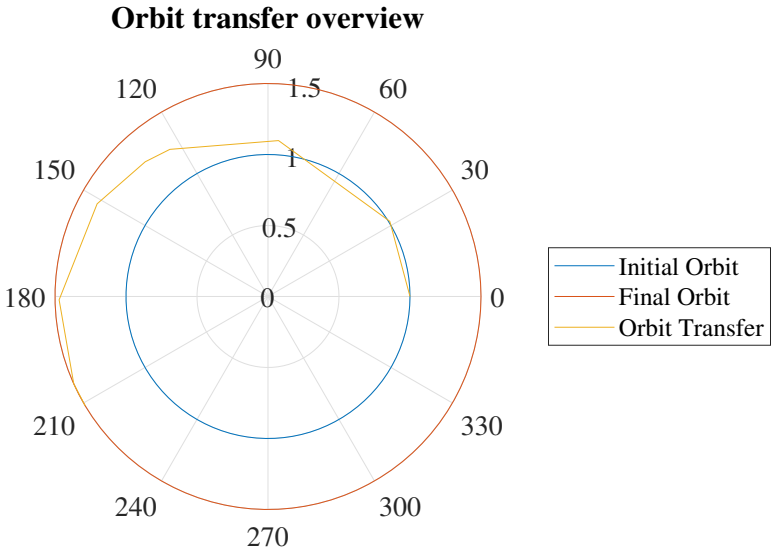

Figure 56: Trajectory from initial to final orbit ($N : 4$, $K : 2$)

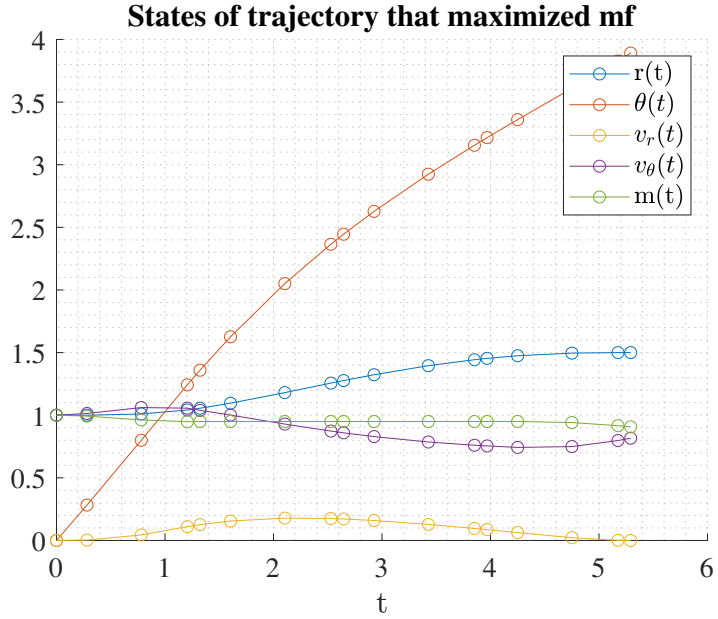

Figure 57: States for trajectory that maximized terminal mass ($N : 4$, $K : 4$)

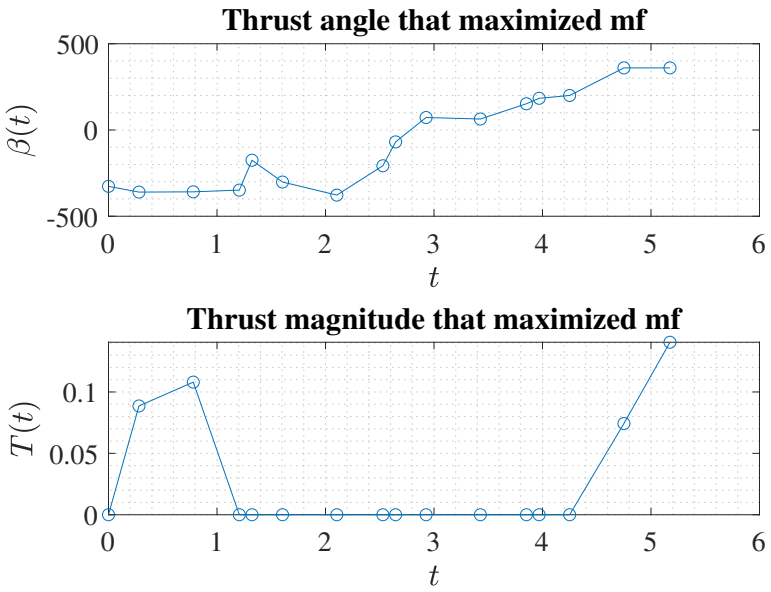

Figure 58: Control that maximized terminal mass ($N : 4$, $K : 4$)

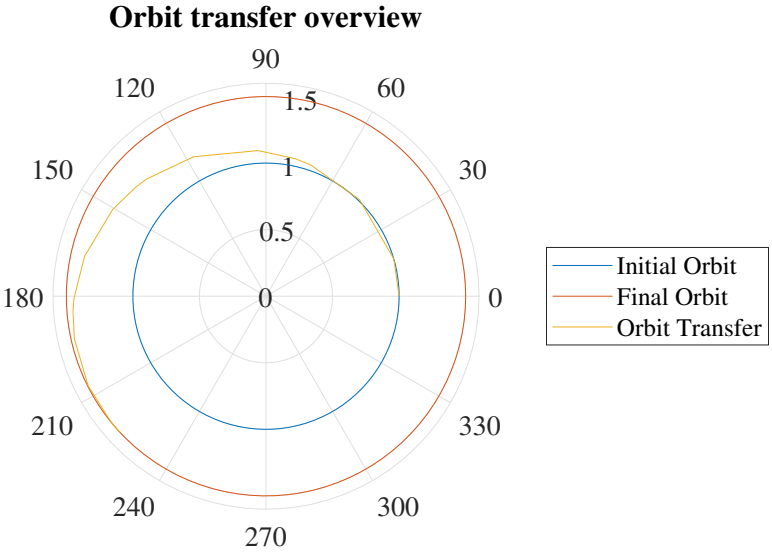

Figure 59: Trajectory from initial to final orbit ($N : 4$, $K : 4$)

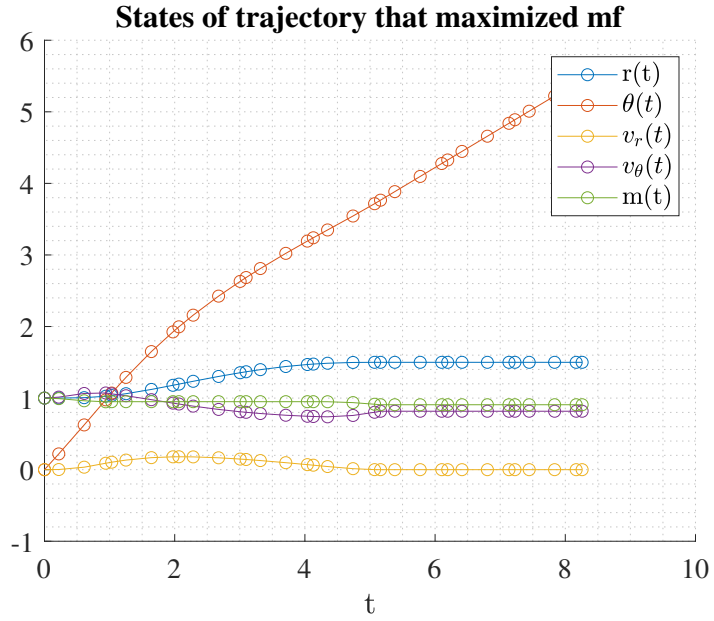

Figure 60: States for trajectory that maximized terminal mass ($N : 4$, $K : 8$)

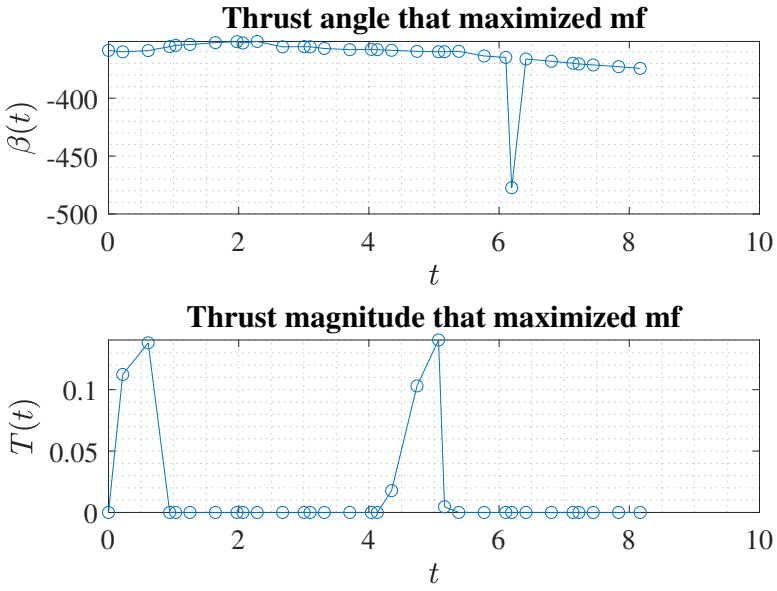

Figure 61: Control that maximized terminal mass ($N : 4$, $K : 8$)

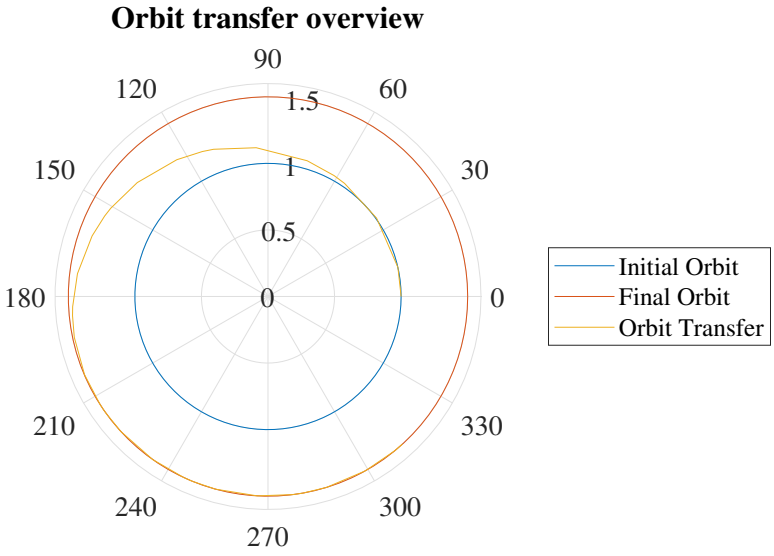

Figure 62: Trajectory from initial to final orbit ($N : 4$, $K : 8$)

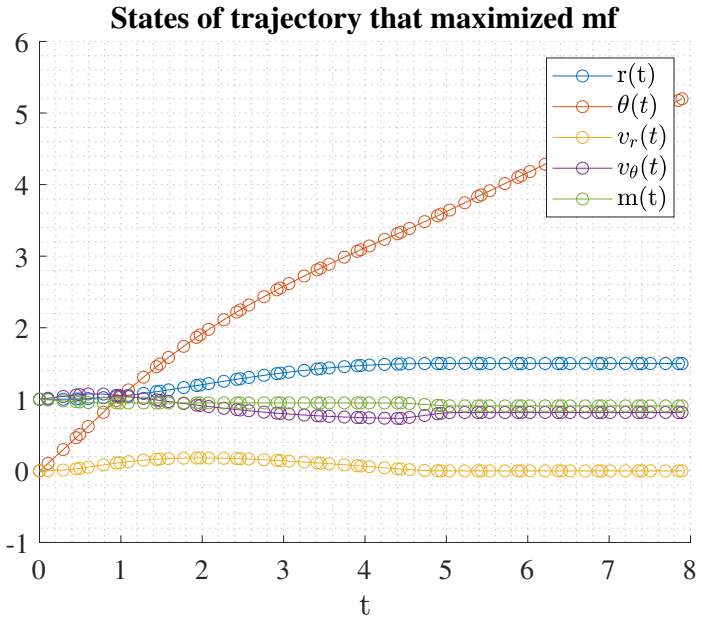

Figure 63: States for trajectory that maximized terminal mass ($N : 4$, $K : 16$)

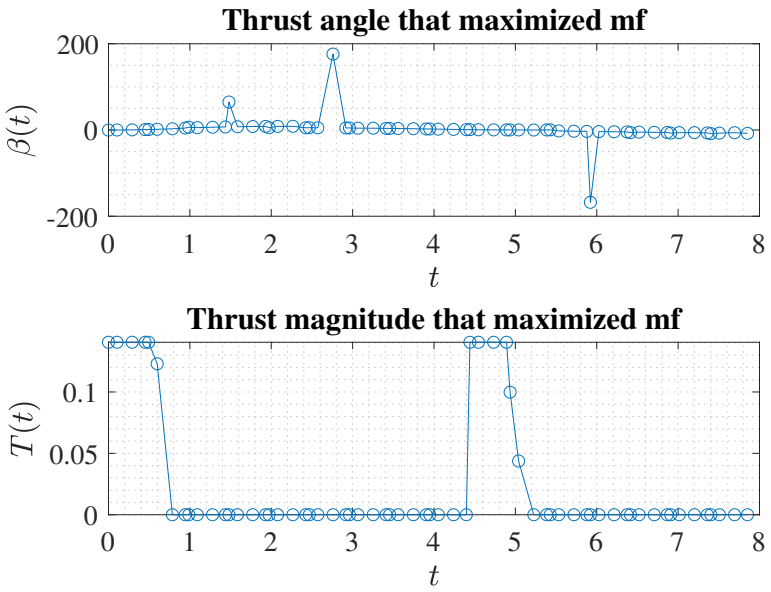

Figure 64: Control that maximized terminal mass ($N : 4$, $K : 16$)

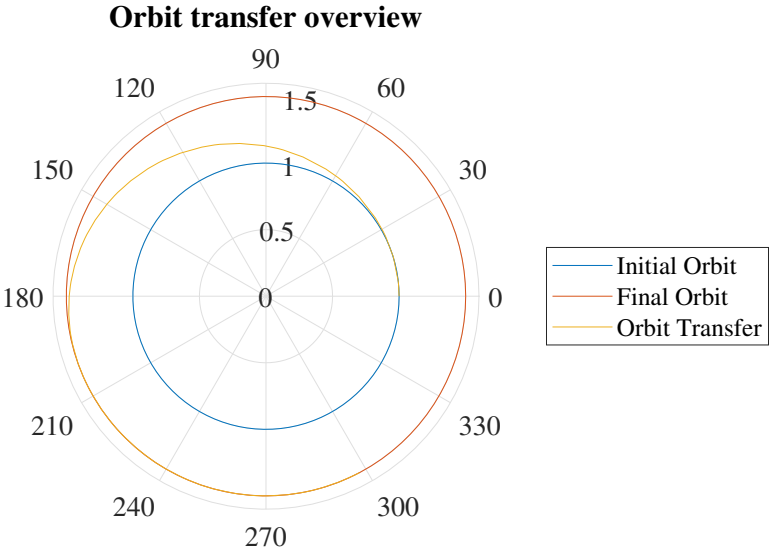

Figure 65: Trajectory from initial to final orbit ($N : 4$, $K : 16$)

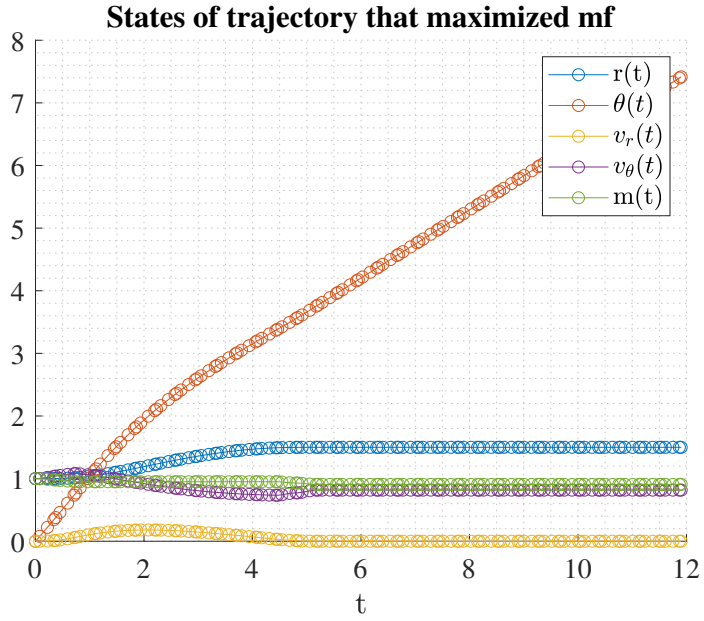

Figure 66: States for trajectory that maximized terminal mass ($N : 4$, $K : 32$)

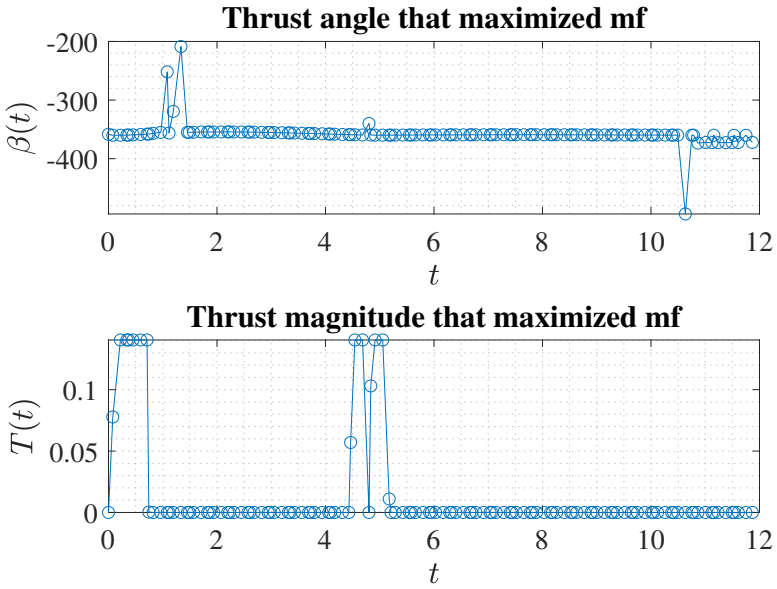

Figure 67: Control that maximized terminal mass ($N : 4$, $K : 32$)

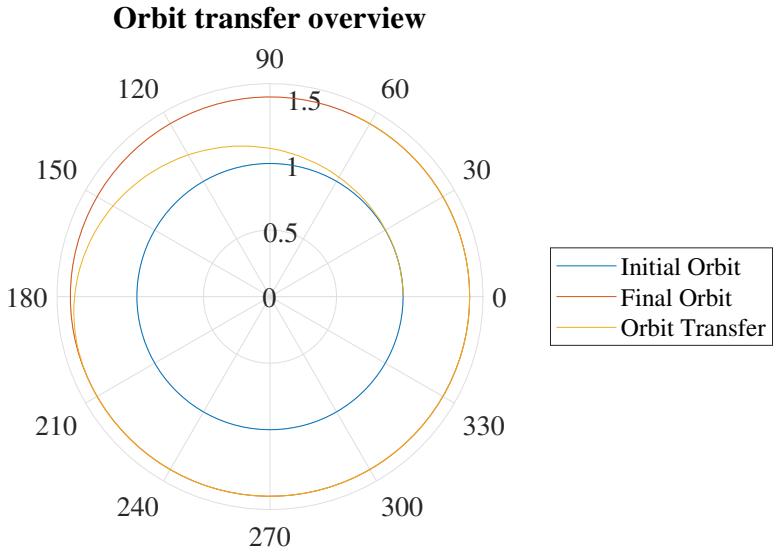

Figure 68: Trajectory from initial to final orbit ($N : 4$, $K : 32$)

3.1.3 Minimize Terminal Time with Constrained Control

The optimal control problem was again solved with the objective function being to minimize the final time, t_f. However, instead of one of the controls being the thrust angle β, (u_1, u_2) were used in conjunction with the path constraint represented by equation 21. Thrust magnitude remains a third control for this problem. Just like the other two cases, the control is parameterized by polynomials of degree N = (3,4) for intervals of K = (2,4,8,16,32). The results for all the combinations can be seen in Table 3. Similarly to the case of minimizing t_f with unconstrained control, the objective function was best optimized by a 4^{th} order polynomial with 2 intervals, which resulted in a terminal time of 3.2456. The comparison of the objective function for all combinations can be seen in Figure 69. Similar to the previous two cases, the objective function seems to converge to a specific value as the number of intervals increase. This value is 3.247 which is consistent with the solution obtained with the indirect shooting methods used for the midterm. The orbit transfer for N = 4 and K = 2 is shown in Figure 93. The orbit transfer is much like that of optimal one for the case of minimizing t_f with unconstrained control. The first two segments create paths that result in a radius less than the lower bound due to the constraints only being applied at the LGR points. This is likely why increasing the number of intervals results in the optimal objective function converging to a specific value. When β is reconstructed with u_1 and u_2, it is evident that it is much more contained than the β control for the unconstrained solution. The path constrained controls, u_1 and u_2, show the thrust angle β is contained. For example, Figure 109 shows the path constrained control for the case of N = 4 and K = 32. At each LGR point, the path constraint $u_1^2 + u_2^2 = 1$ is satisfied. This allows the reconstruction of β, as seen in Figure 108, to be well contained. This also allows the nonlinear optimization to run more smoothly.

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	45	0.329	3.2472	0.75548	0
3	4	64	0.521	3.2457	0.75559	0
3	8	86	0.553	3.2467	0.75552	0
3	16	131	0.702	3.247	0.7555	1
3	32	137	0.757	3.2469	0.7555	1
4	2	72	0.574	3.2456	0.7556	0
4	4	71	0.441	3.2466	0.75552	0
4	8	107	1.143	3.247	0.7555	1
4	16	145	1.125	3.247	0.7555	1
4	32	160	0.964	3.2469	0.7555	0

Table 3: Results for minimizing t_f with constrained control

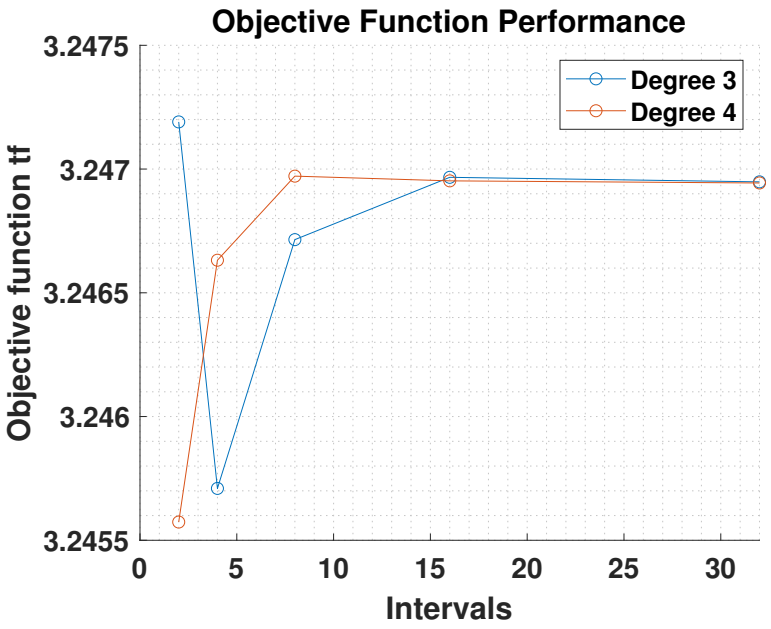

Figure 69: Objective function performance with constrained controls

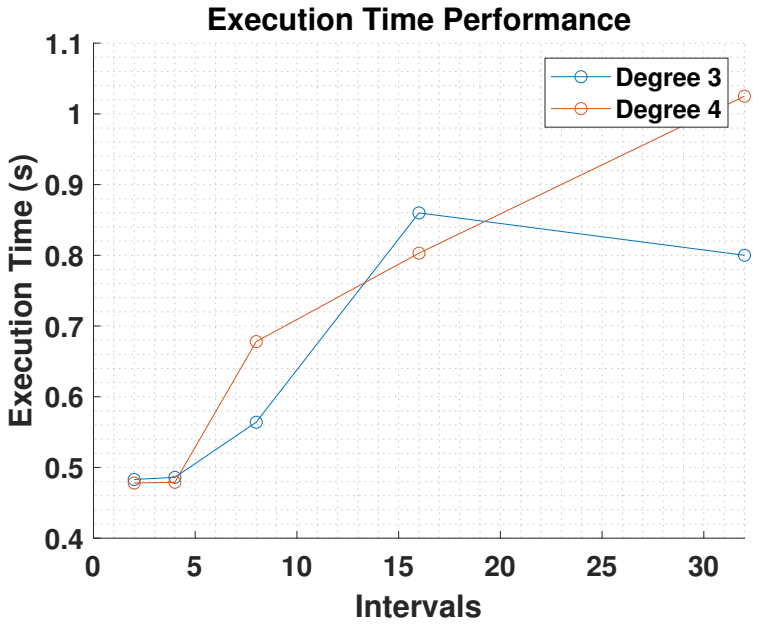

Figure 70: Execution performance of minimizing terminal time with constrained controls

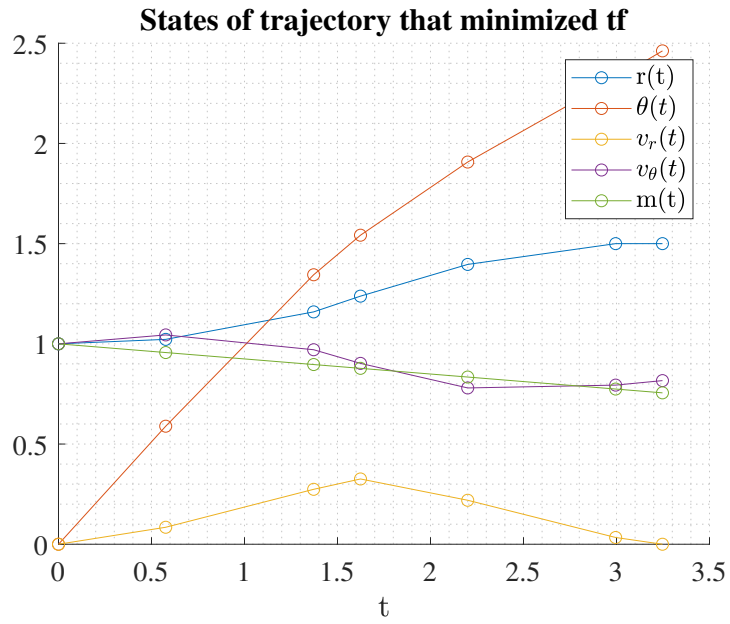

Figure 71: States for trajectory that minimized terminal time ($N : 3$, $K : 2$)

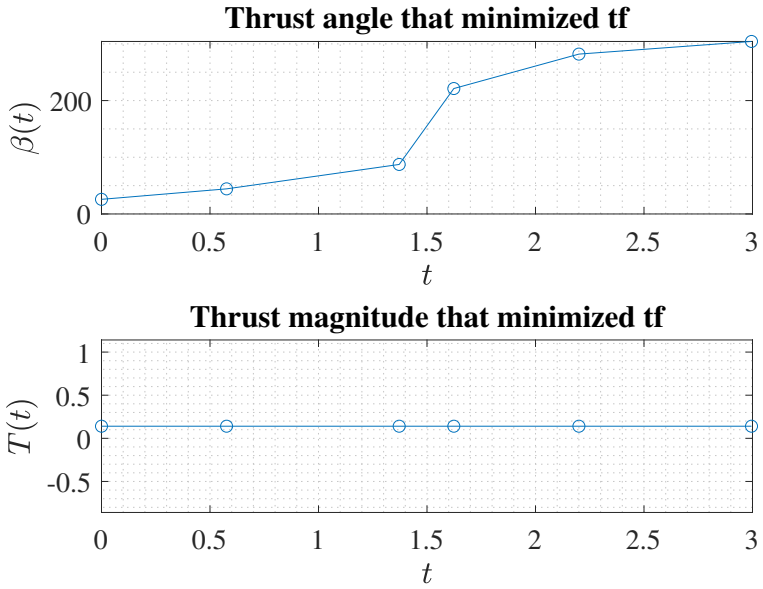

Figure 72: Control that minimized terminal time ($N : 3$, $K : 2$)

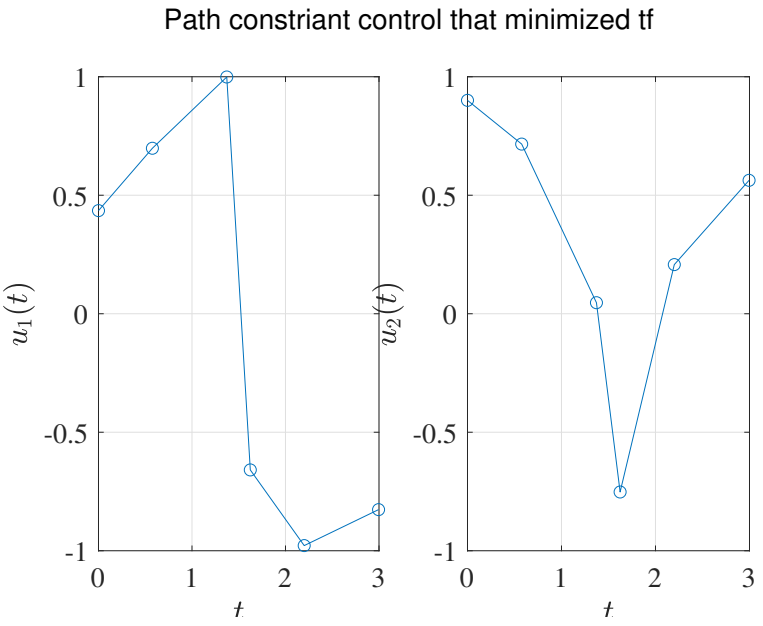

Figure 73: Path constrained control that minimized terminal time ($N : 3$, $K : 2$)

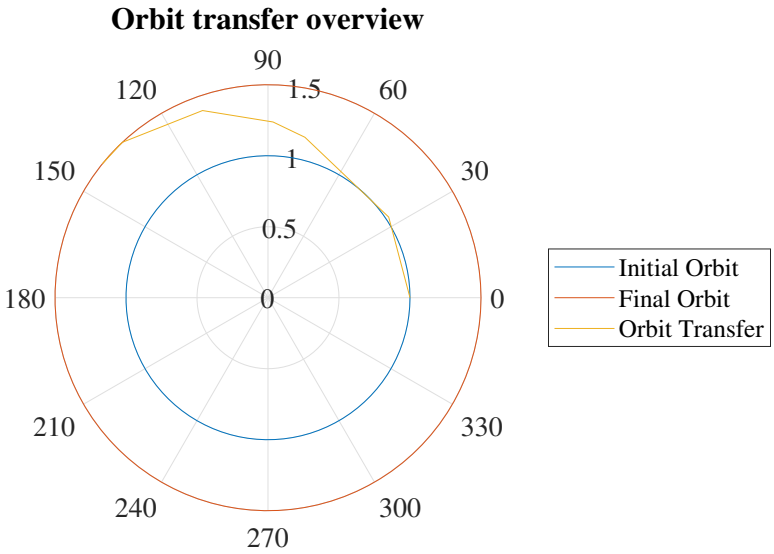

Figure 74: Trajectory from initial to final orbit ($N : 3$, $K : 2$)

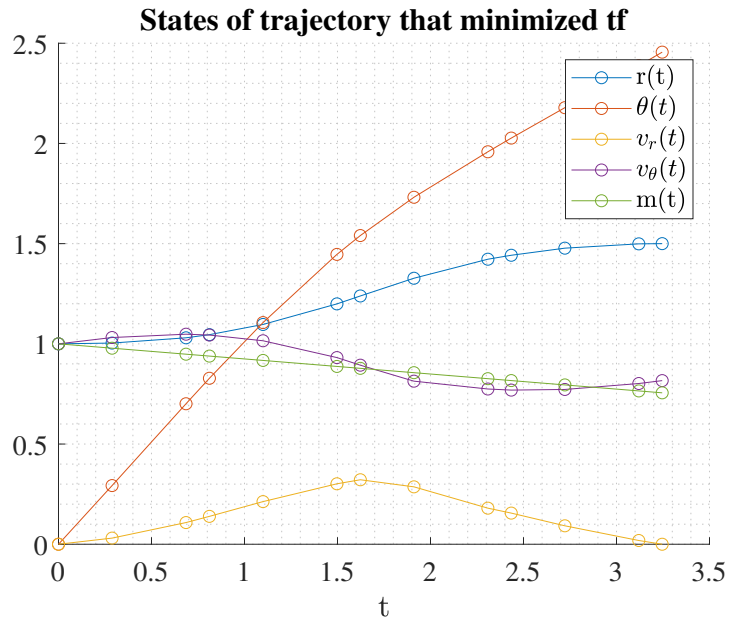

Figure 75: States for trajectory that minimized terminal time ($N : 3$, $K : 4$)

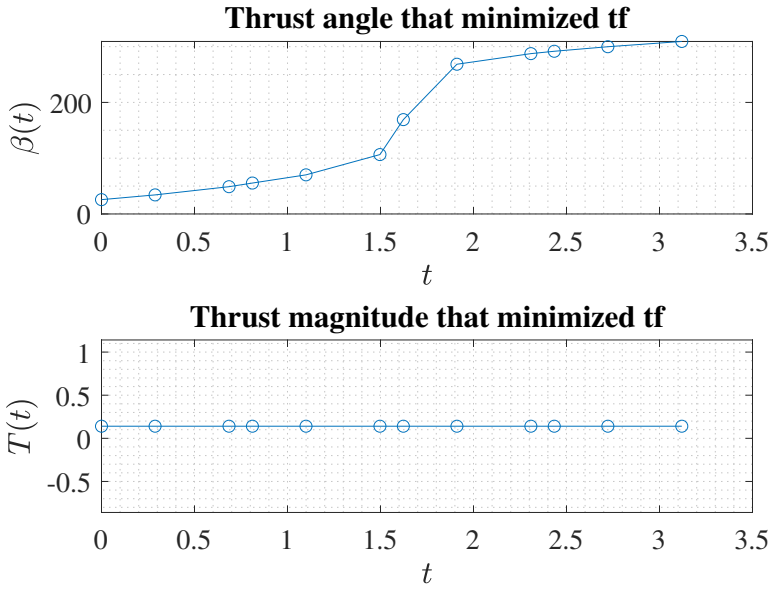

Figure 76: Control that minimized terminal time ($N : 3$, $K : 4$)

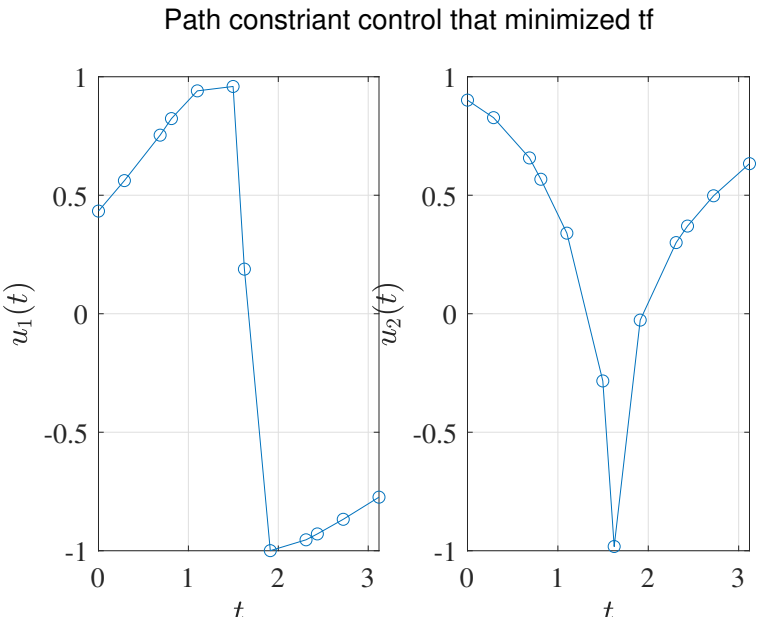

Figure 77: Path constrained control that minimized terminal time ($N : 3$, $K : 4$)

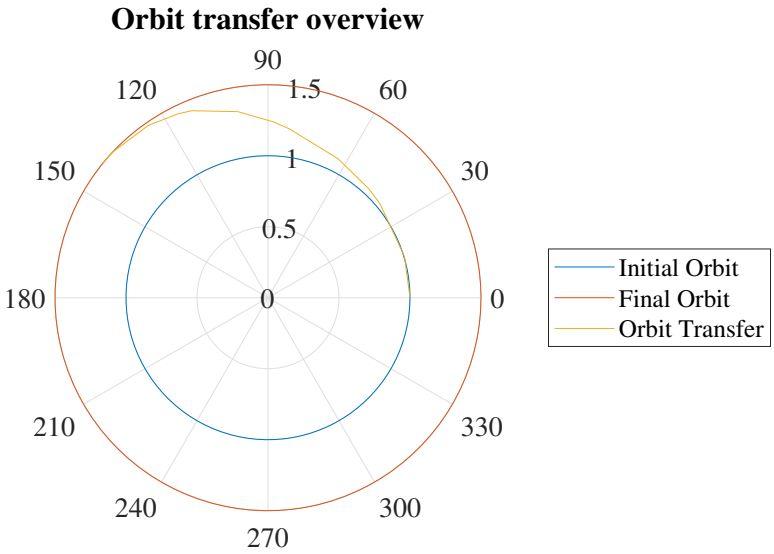

Figure 78: Trajectory from initial to final orbit ($N : 3$, $K : 4$)

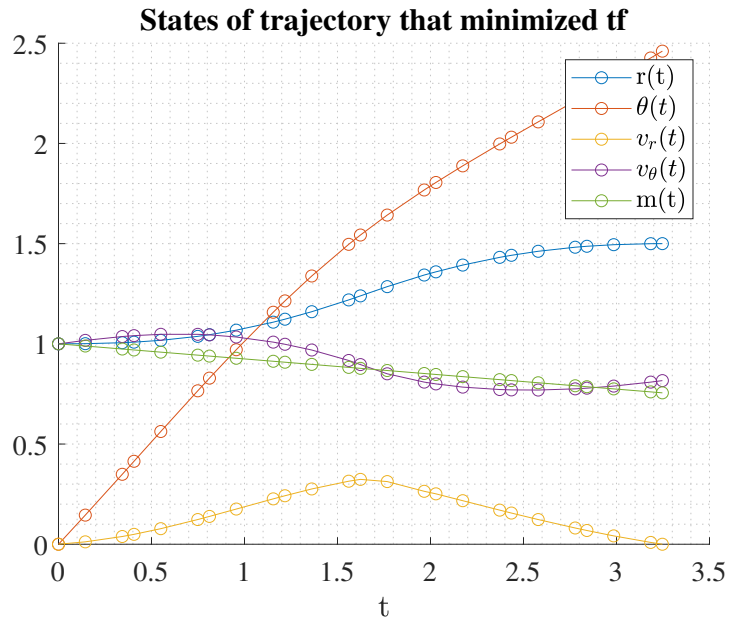

Figure 79: States for trajectory that minimized terminal time ($N : 3$, $K : 8$)

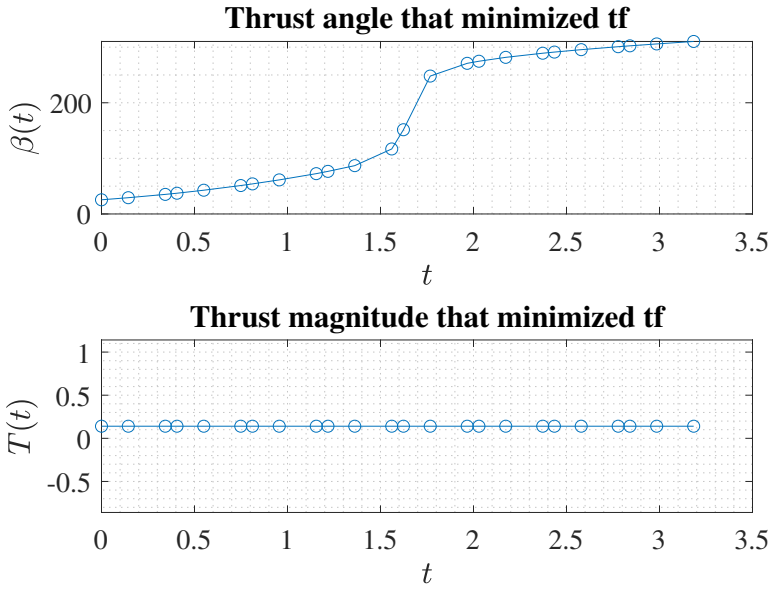

Figure 80: Control that minimized terminal time ($N : 3$, $K : 8$)

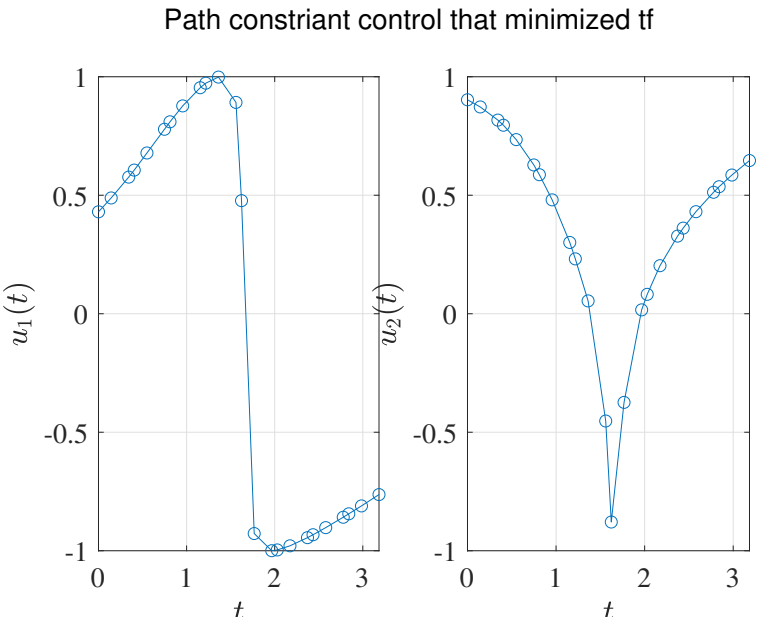

Figure 81: Path constrained control that minimized terminal time ($N : 3$, $K : 8$)

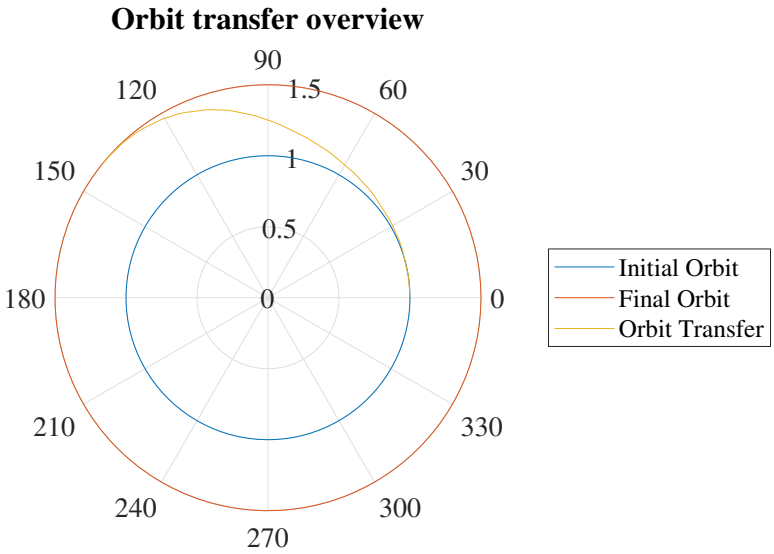

Figure 82: Trajectory from initial to final orbit ($N : 3$, $K : 8$)

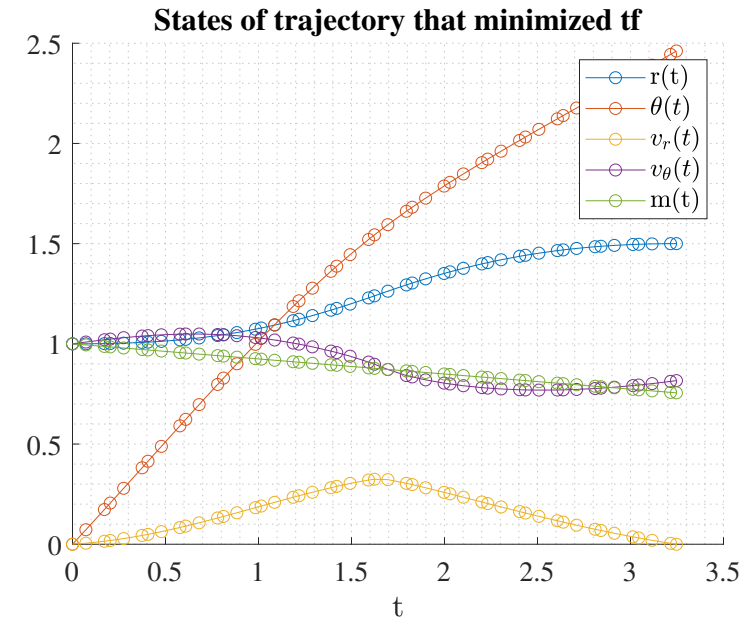

Figure 83: States for trajectory that minimized terminal time ($N : 3$, $K : 16$)

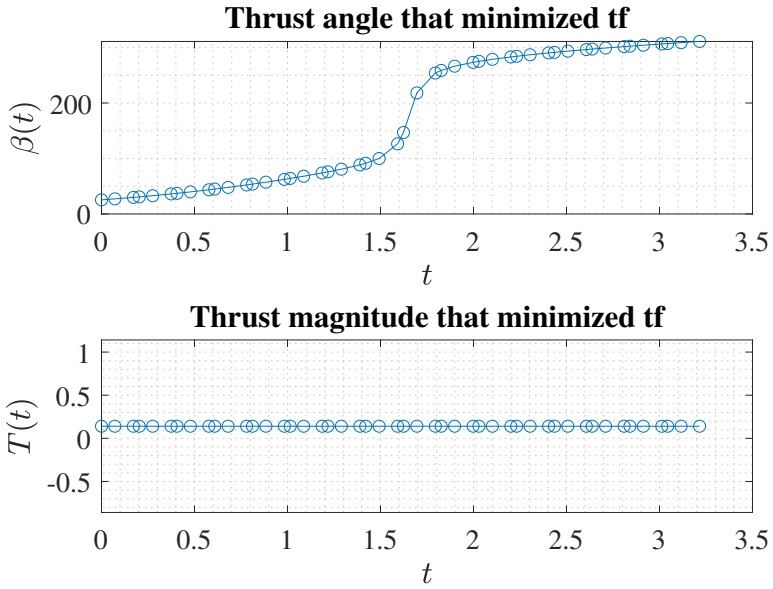

Figure 84: Control that minimized terminal time ($N : 3$, $K : 16$)

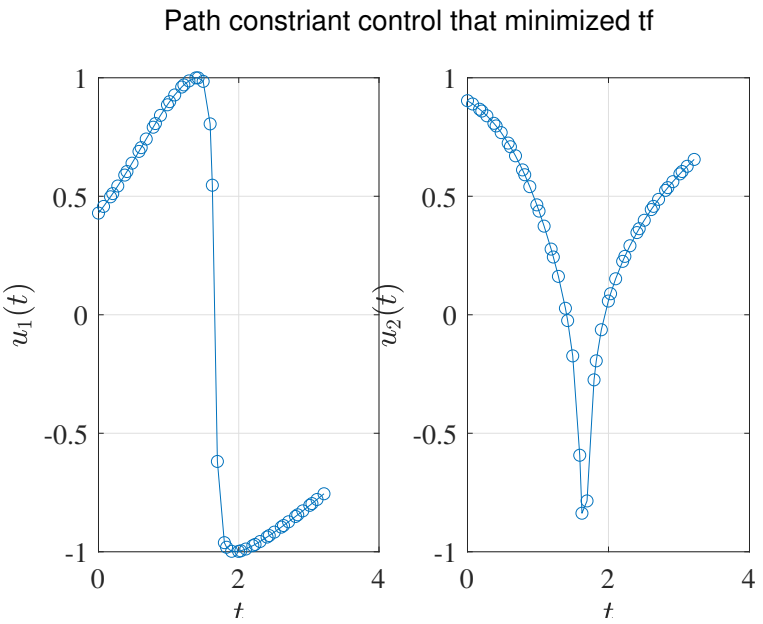

Figure 85: Path constrained control that minimized terminal time ($N : 3$, $K : 16$)

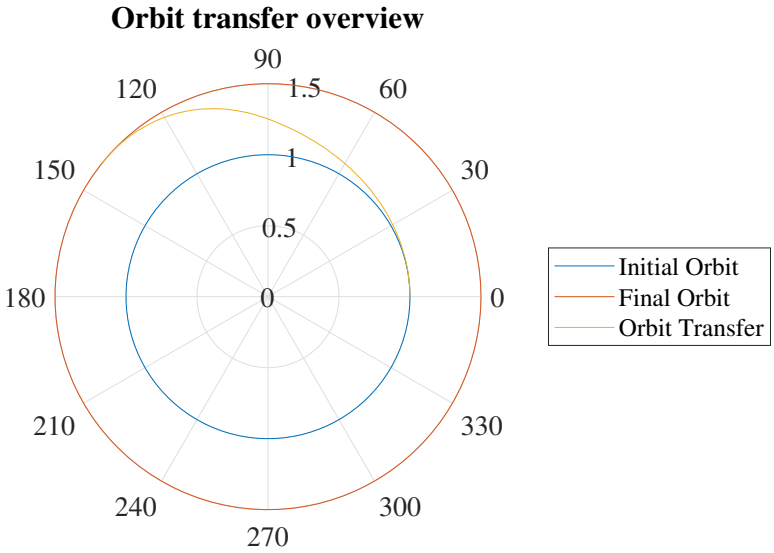

Figure 86: Trajectory from initial to final orbit ($N : 3$, $K : 16$)

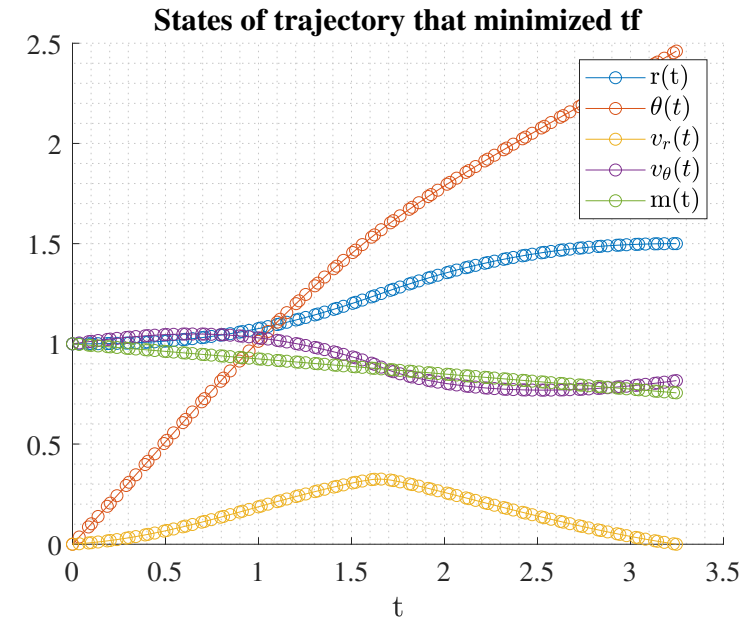

Figure 87: States for trajectory that minimized terminal time ($N : 3$, $K : 32$)

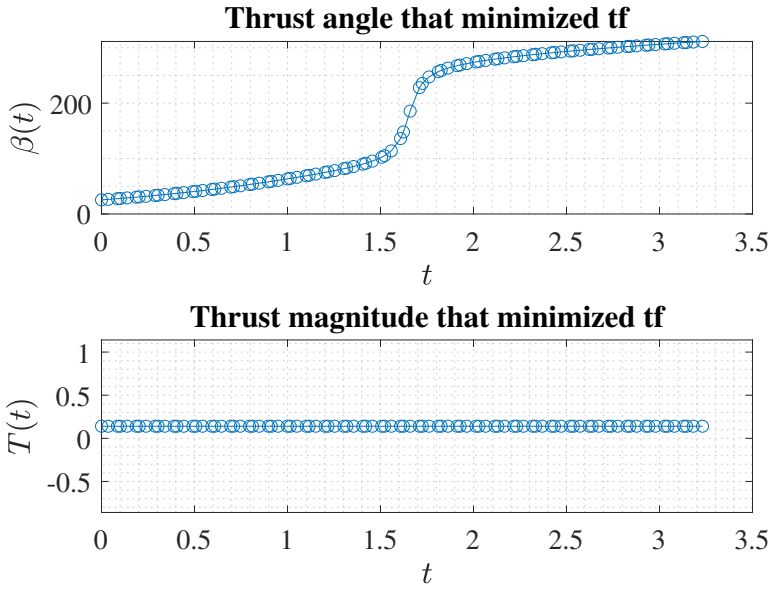

Figure 88: Control that minimized terminal time ($N : 3$, $K : 32$)

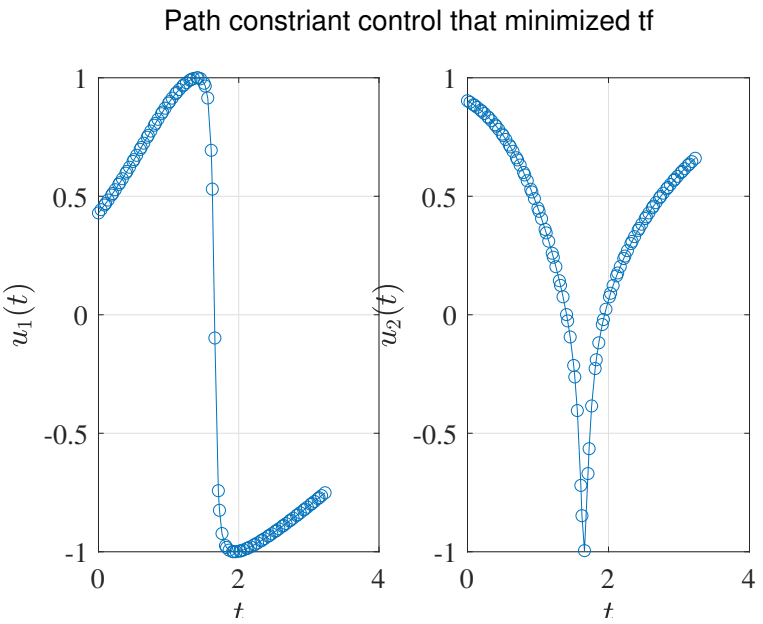

Figure 89: Path constrained control that minimized terminal time ($N : 3$, $K : 32$)

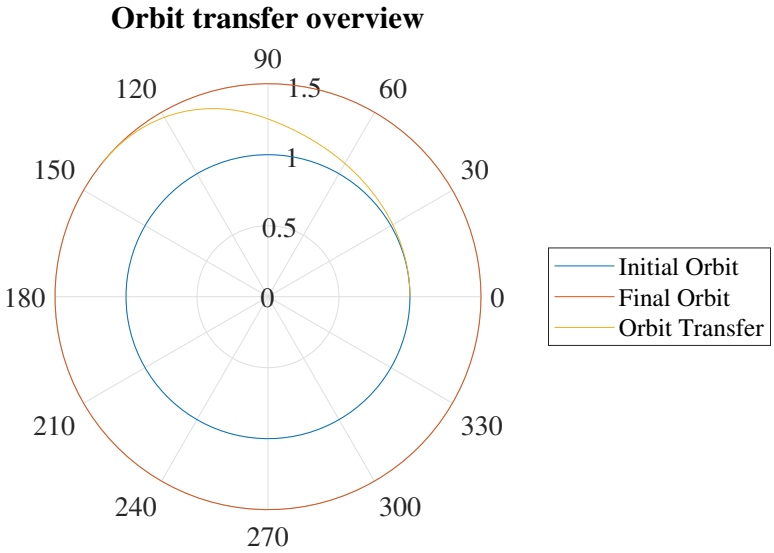

Figure 90: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

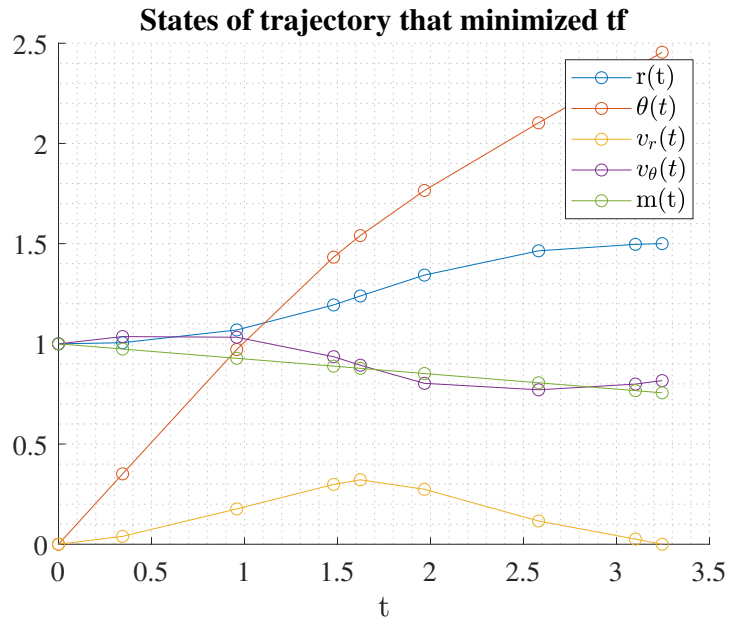

Figure 91: States for trajectory that minimized terminal time ($N : 4$, $K : 2$)

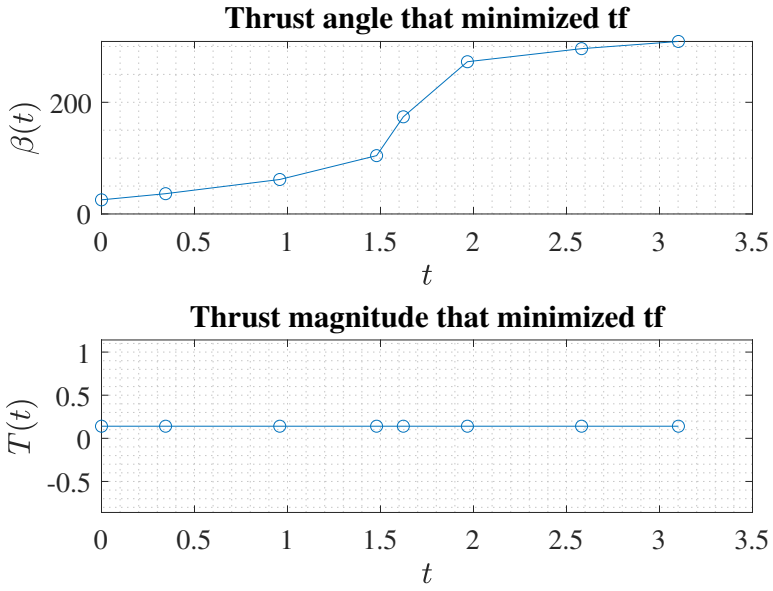

Figure 92: Control that minimized terminal time ($N : 4$, $K : 2$)

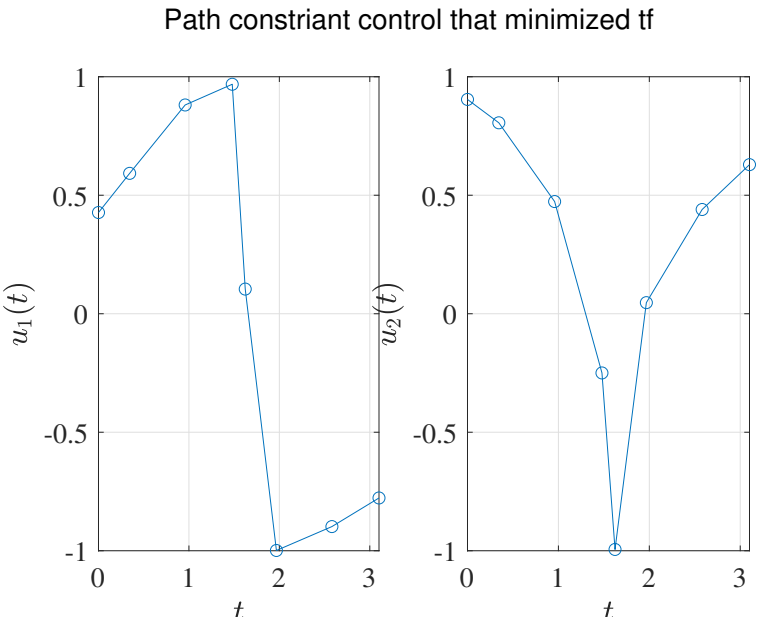

Figure 93: Path constrained control that minimized terminal time ($N : 4$, $K : 2$)

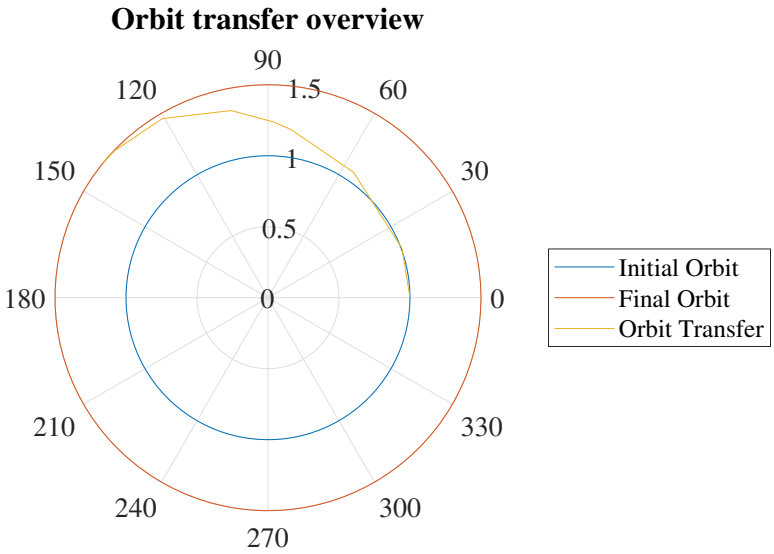

Figure 94: Trajectory from initial to final orbit ($N : 4$, $K : 2$)

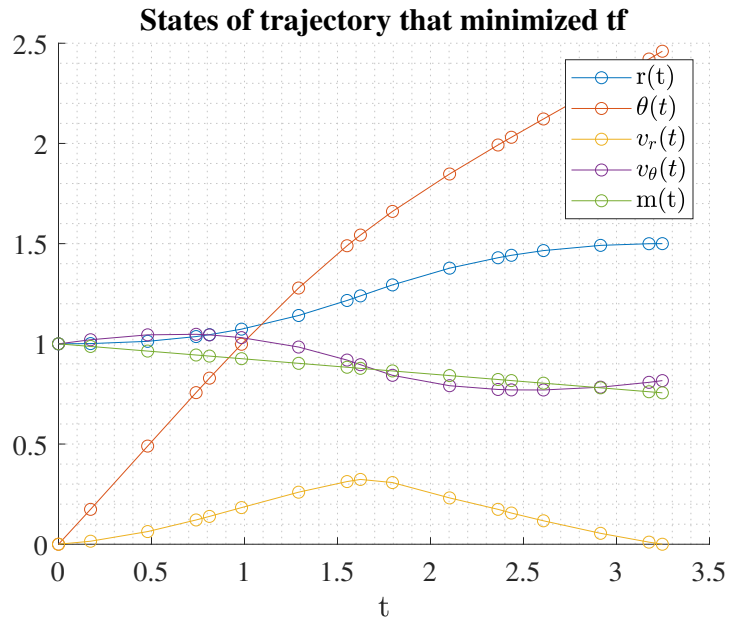

Figure 95: States for trajectory that minimized terminal time ($N : 4$, $K : 4$)

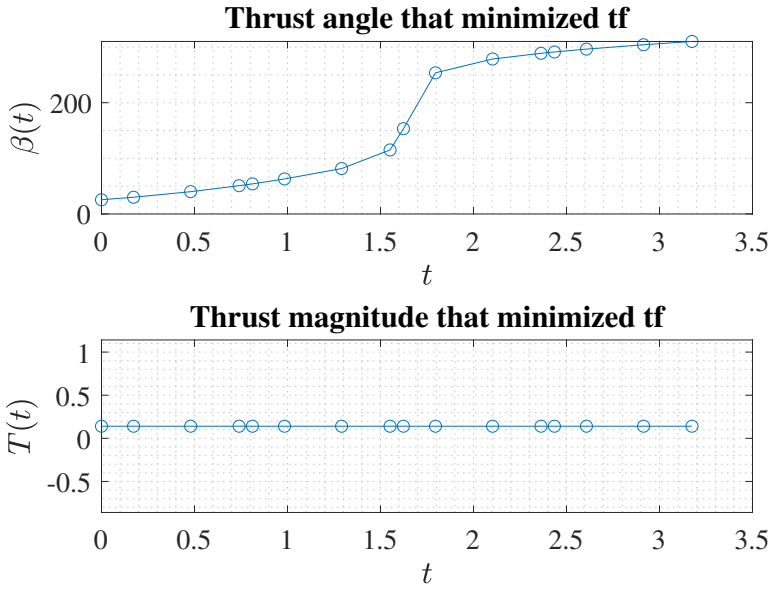

Figure 96: Control that minimized terminal time ($N : 4$, $K : 4$)

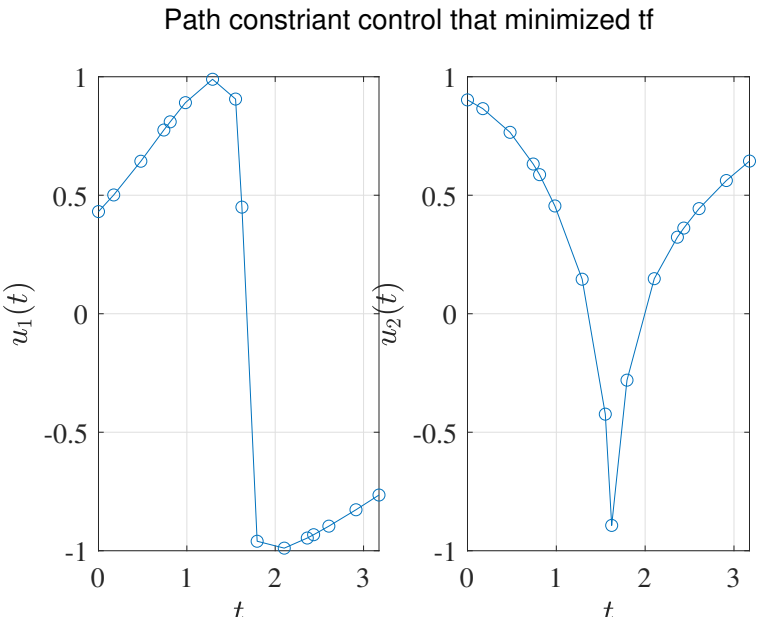

Figure 97: Path constrained control that minimized terminal time ($N : 4$, $K : 4$)

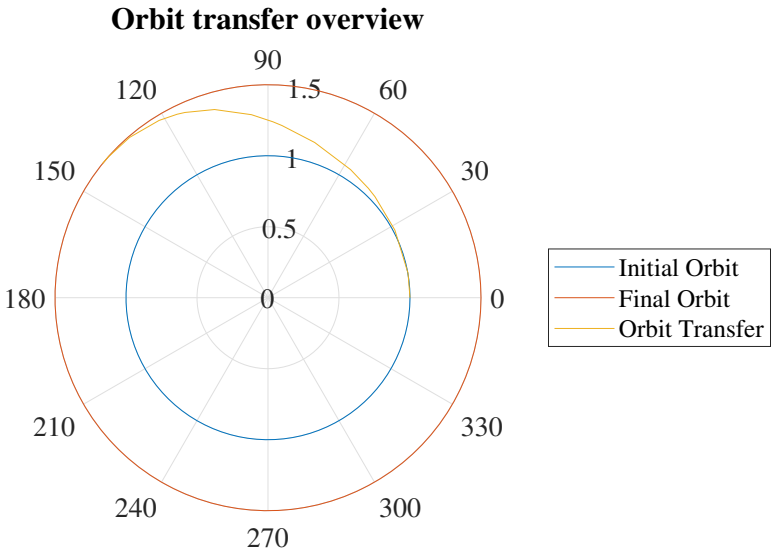

Figure 98: Trajectory from initial to final orbit ($N : 4$, $K : 4$)

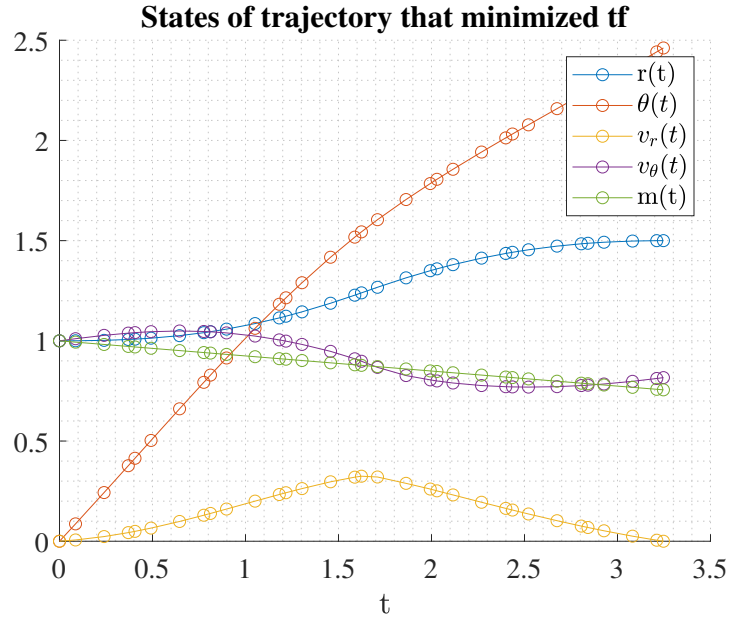

Figure 99: States for trajectory that minimized terminal time ($N : 4$, $K : 8$)

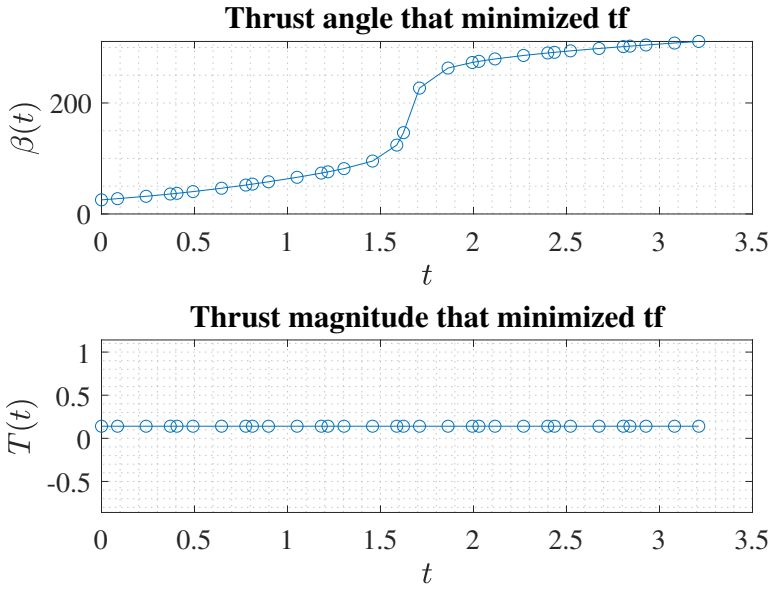

Figure 100: Control that minimized terminal time ($N : 4$, $K : 8$)

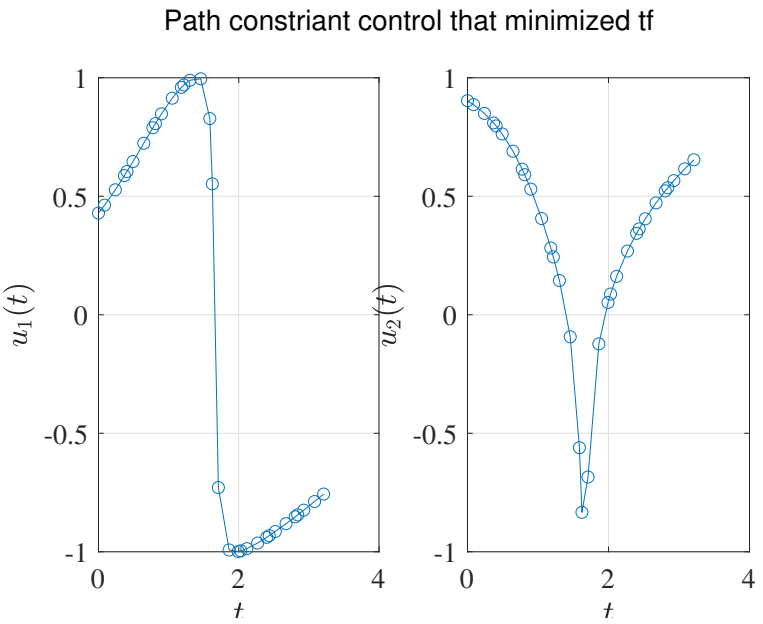

Figure 101: Path constrained control that minimized terminal time ($N : 4$, $K : 8$)

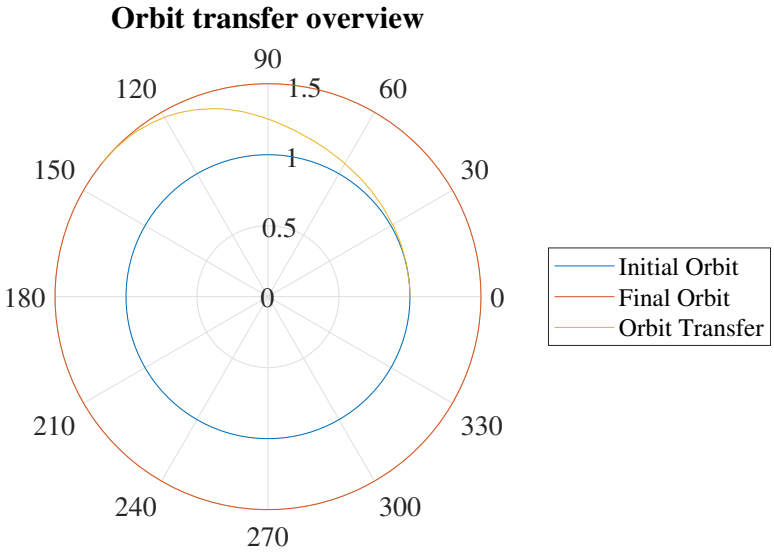

Figure 102: Trajectory from initial to final orbit ($N : 4$, $K : 8$)

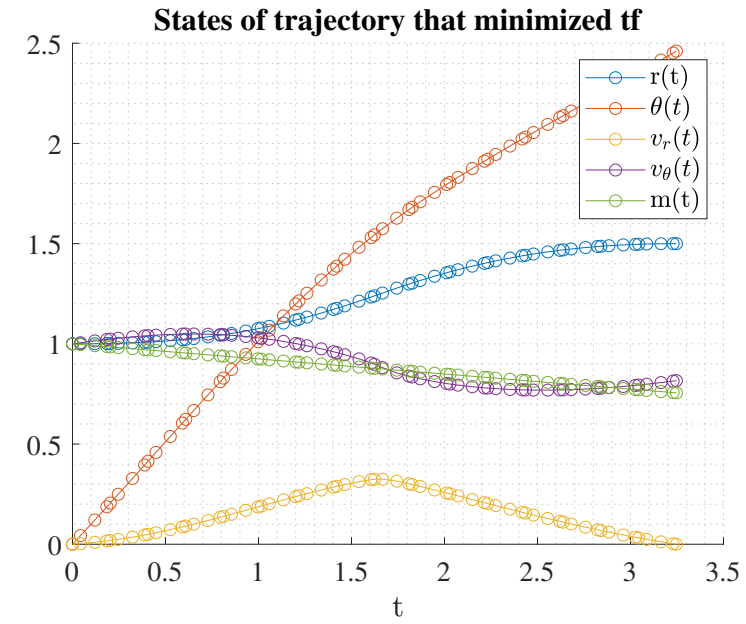

Figure 103: States for trajectory that minimized terminal time ($N : 4$, $K : 16$)

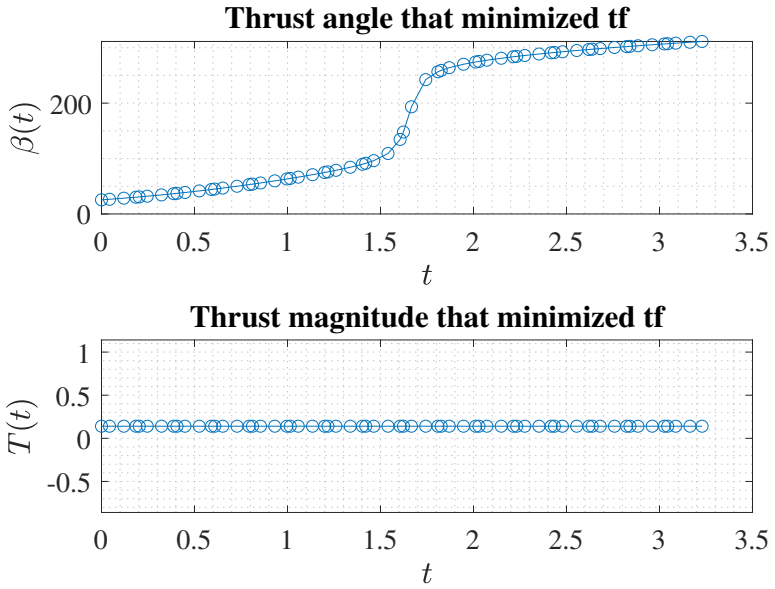

Figure 104: Control that minimized terminal time ($N : 4$, $K : 16$)

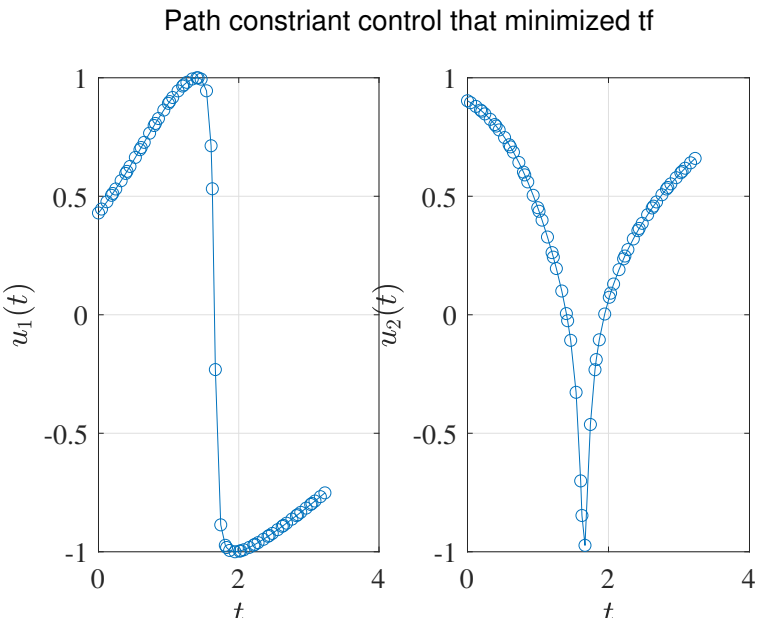

Figure 105: Path constrained control that minimized terminal time ($N : 4$, $K : 16$)

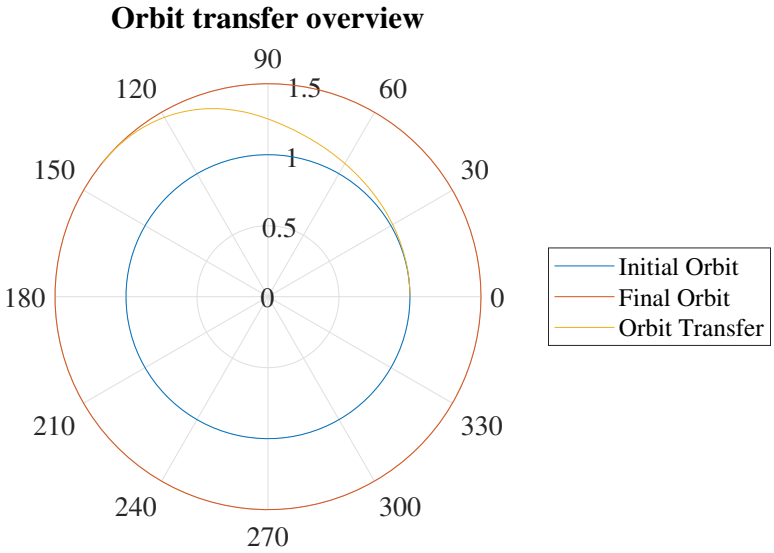

Figure 106: Trajectory from initial to final orbit ($N : 4$, $K : 16$)

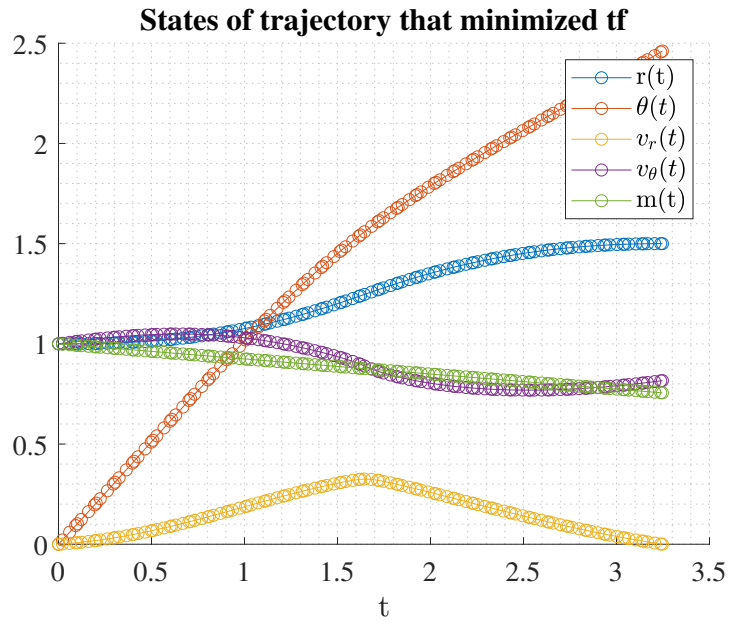

Figure 107: States for trajectory that minimized terminal time ($N : 4$, $K : 32$)

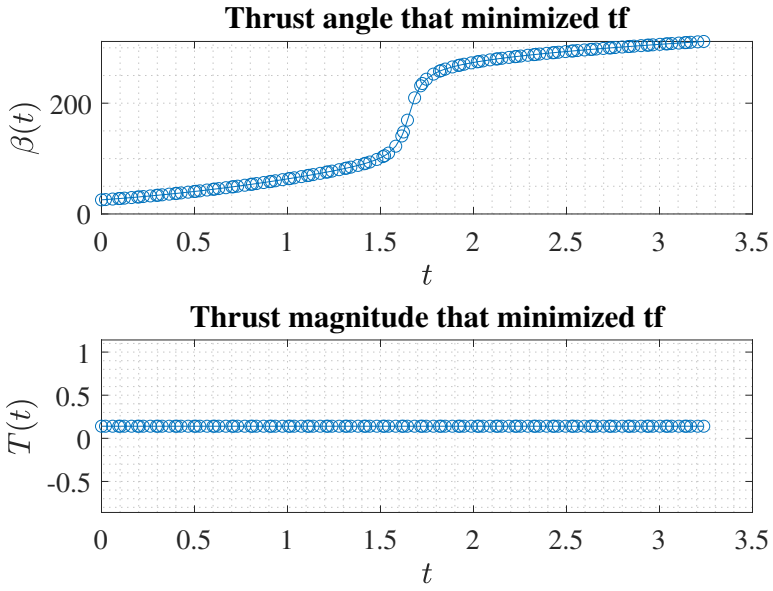

Figure 108: Control that minimized terminal time ($N : 4$, $K : 32$)

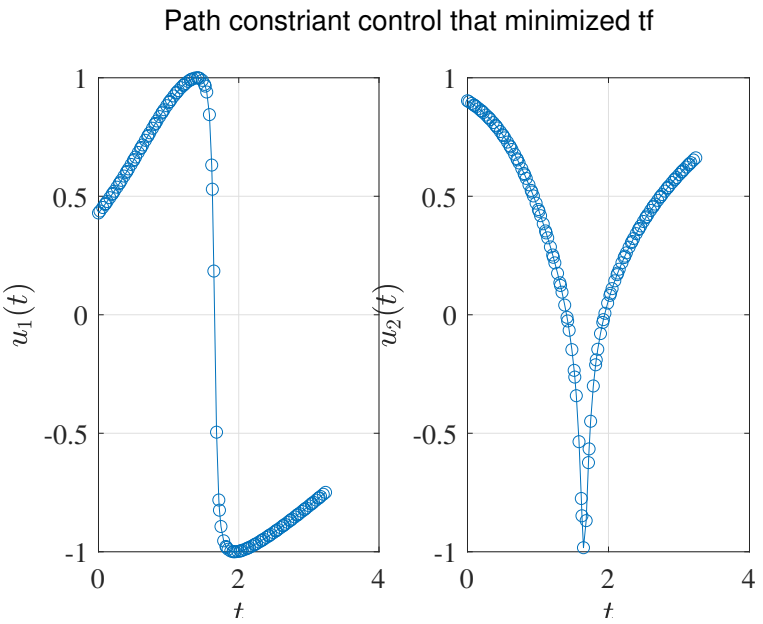

Figure 109: Path constrained control that minimized terminal time ($N : 4$, $K : 32$)

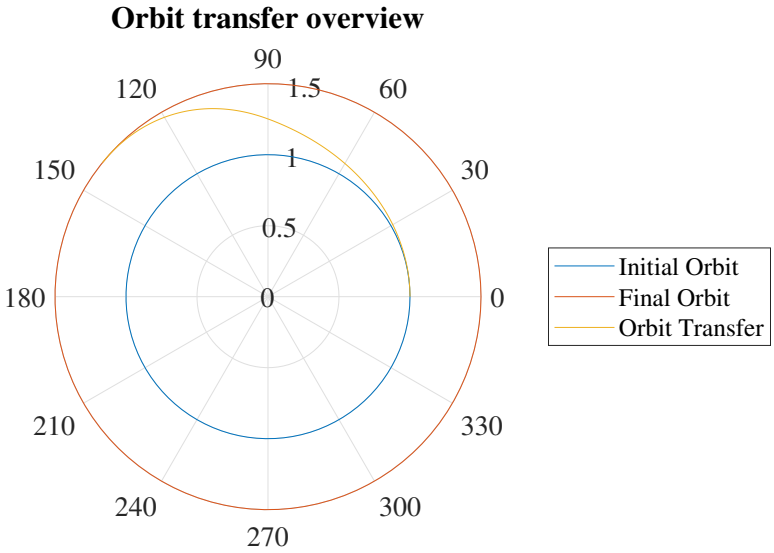

Figure 110: Trajectory from initial to final orbit ($N : 4$, $K : 32$)

3.1.4 Maximize Terminal Mass with Constrained Control

The optimal control problem was lastly solved with the objective function being to maximize the terminal mass m_f with the controls u_1, u_2, and thrust magnitude T. Additionally, the path constraint $u_1^2+u_2^2 = 1$ was enforced. The controls u_1 and u_2 are represented by equations 19 and 20, respectively. Similar to the previous cases, the control for the problem is parameterized by polynomials N = (3,4) with intervals K = (2,4,8,16,32). The results for all of the combinations are shown in Table 4. The table, along with Figure 111, show that the objective function m_f was best optimized by a 4^{th} order polynomial control with 2 intervals, which resulted in a value of 0.90894. Just like the other cases, the most optimal result was obtained with fewer intervals. Looking at Figure 111, it is noticeable that the objective function converges to a value of 0.9072 as the number of intervals increases. This is significant because it tells us that the most optimal solution of 0.90894 is most likely unrealistic. Figure 135 shows the orbit transfer for the optimal combination. The orbit looks very unrealistic since two of the segments conjoining LGR points cut directly though the initial unit circle. While results for maximizing m_f with constrained control are very similar to those produced with unconstrained control, the control output is much different. When u_1 and u_2 are used along with the path constraint, the control is much more contained and realistic. For instance, Figure 146 (N:4, K:16), shows the thrust angle β for an orbit transfer that maximized m_f to the converged value of 0.9072. β is noticeably much more contained when it is not directly used as a control. Figure 147 shows the u_1 and u_2 controls that construct β. These controls satisfy the desired path constraint which is why β is well contained. One thing to point out with the results is that the control seem to have "chattering". In other words, there are sometimes spikes that are not desired. After extensive investigation, there does not seems to be a clear reason why the "chattering" occurs. One theory is the inconstancy with the nonlinear optimizer or numerical integration errors. The "chattering" can be slightly minimized by tuning initial guesses and bounds of variables but only to a certain extent. One idea that was not successfully implemented was to create an additional constraint on the control such that it cannot change directions.

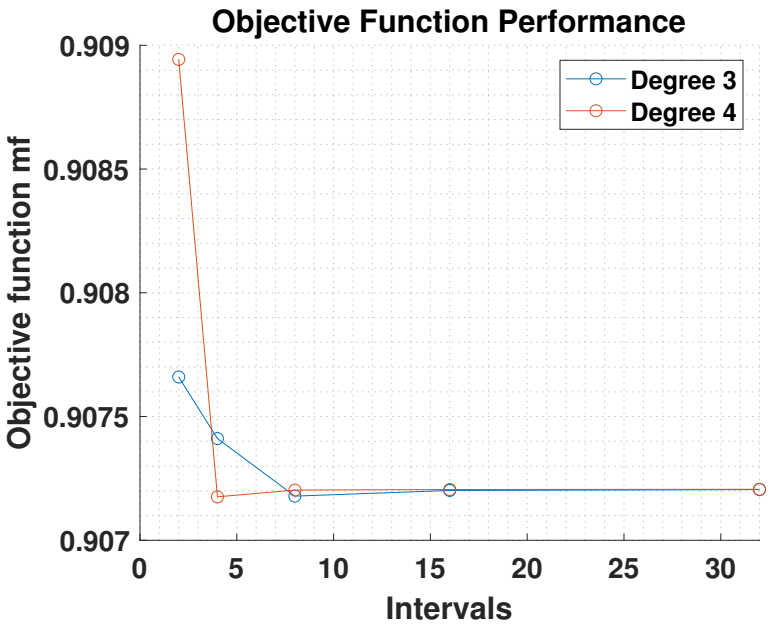

Figure 111: Objective function performance with constrained controls

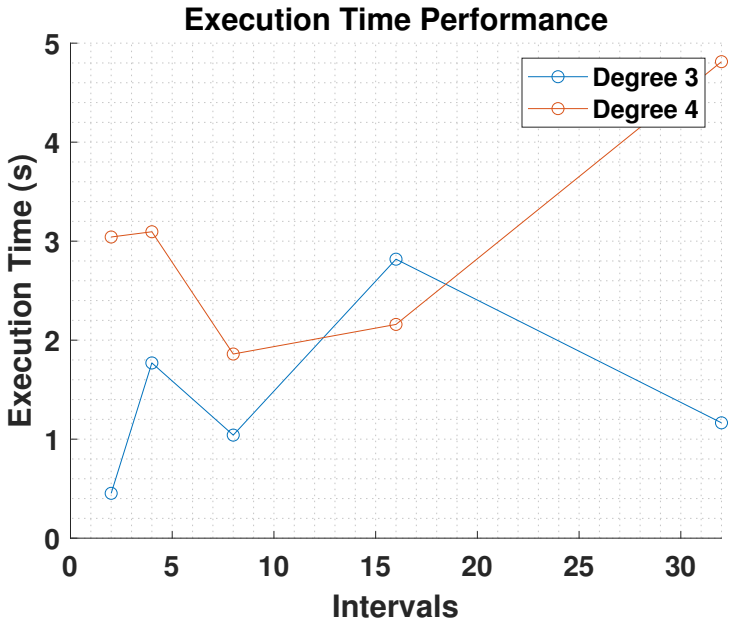

Figure 112: Execution performance of maximizing terminal mass with constrained controls

Degree	Intervals	Iterations	CPU Time	tf	mf	Solved Status
3	2	54	0.41	5.3953	0.90766	0
3	4	313	1.522	8.8202	0.90741	0
3	8	182	0.947	8.7513	0.90718	0
3	16	517	2.443	6.911	0.9072	1
3	32	189	1.026	8.5699	0.9072	0
4	2	594	2.793	11.1036	0.90894	0
4	4	406	1.96	6.1534	0.90718	0
4	8	349	1.678	6.4506	0.9072	0
4	16	397	1.938	6.6388	0.90721	0
4	32	687	3.704	8.1883	0.90721	0

Table 4: Results for maximizing m_f with constrained control

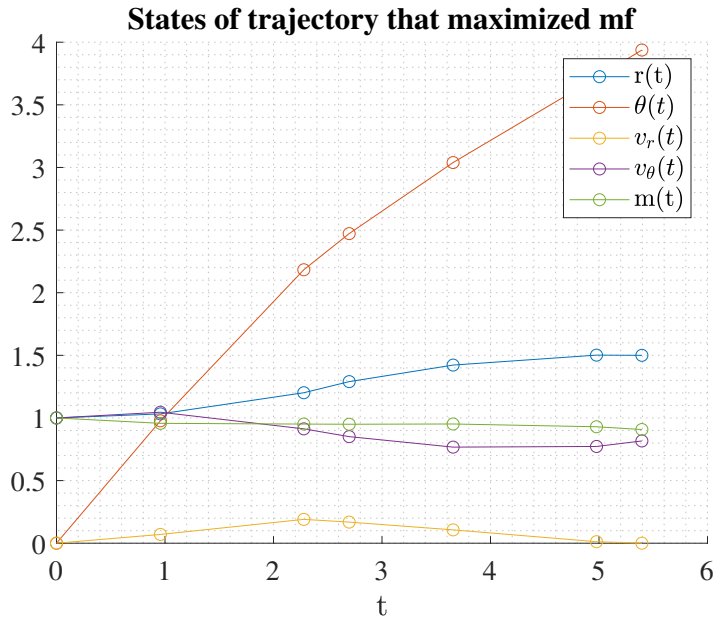

Figure 113: States for trajectory that maximized terminal mass ($N : 3$, $K : 2$)

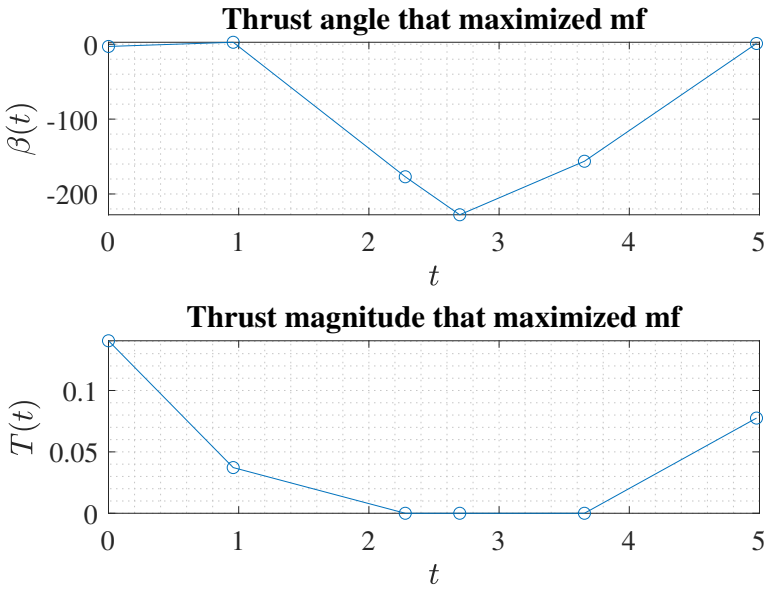

Figure 114: Control that maximized terminal mass ($N : 3$, $K : 2$)

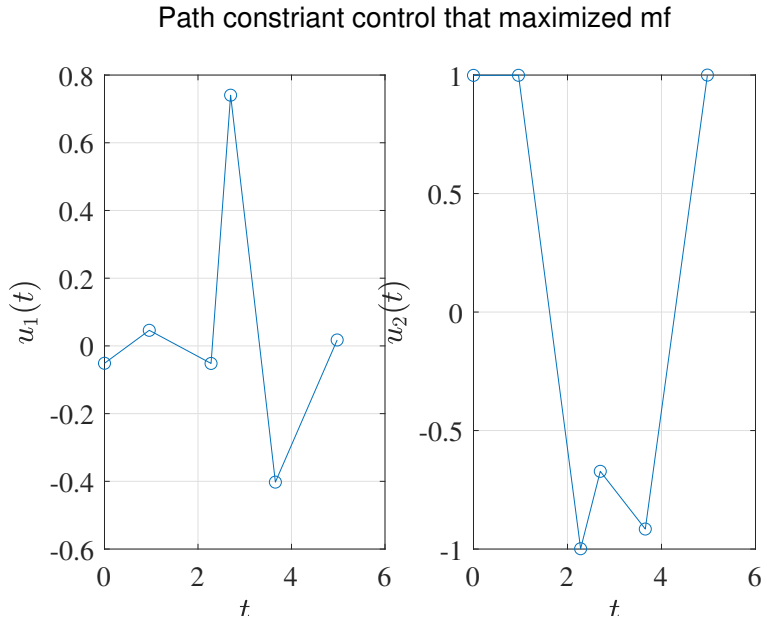

Figure 115: Path constrained control that maximized terminal mass ($N : 3$, $K : 2$)

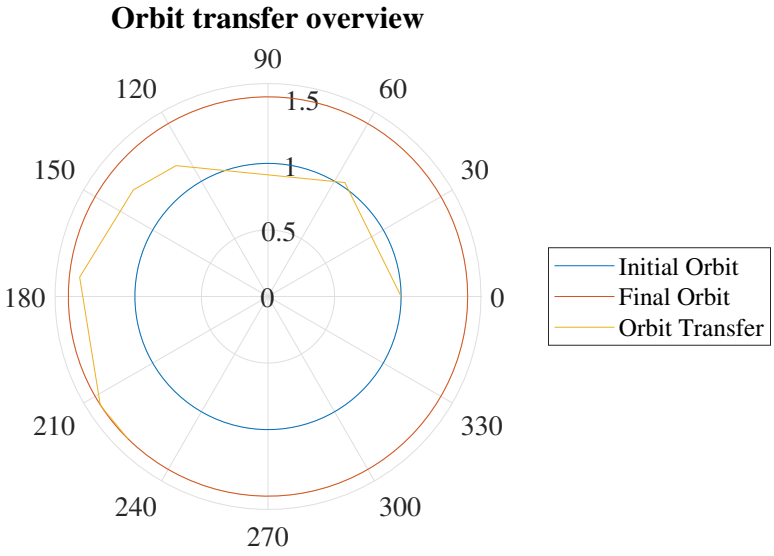

Figure 116: Trajectory from initial to final orbit ($N : 3$, $K : 2$)

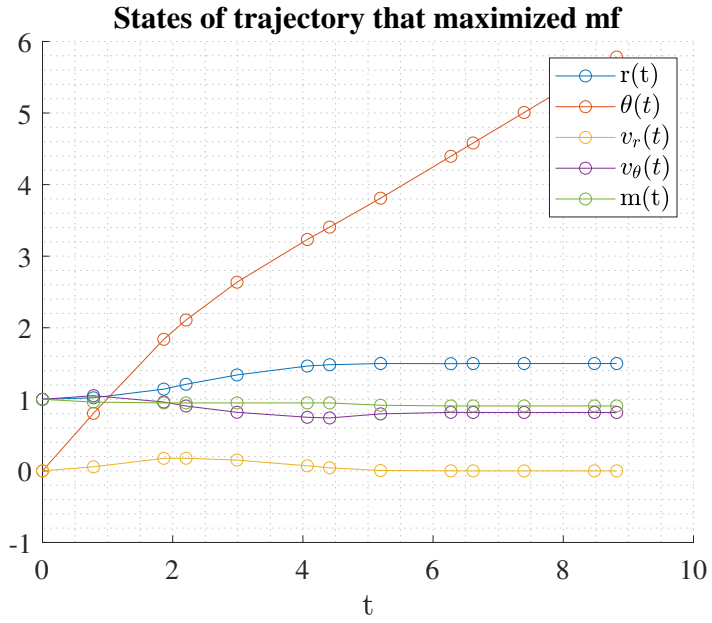

Figure 117: States for trajectory that maximized terminal mass ($N : 3$, $K : 4$)

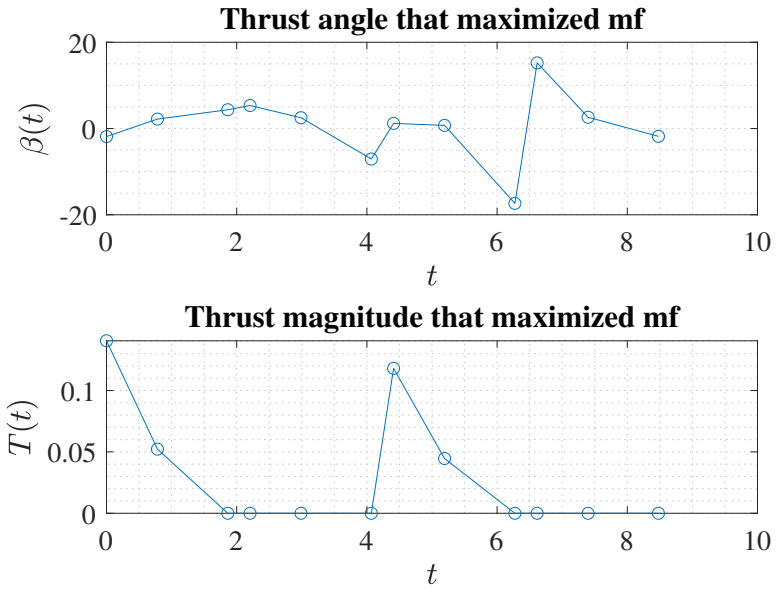

Figure 118: Control that maximized terminal mass ($N : 3$, $K : 4$)

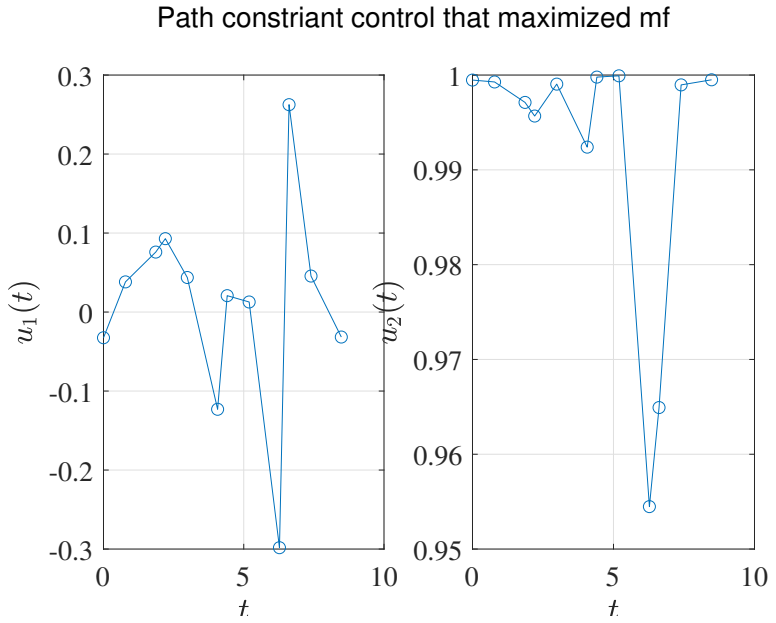

Figure 119: Path constrained control that maximized terminal mass ($N : 3$, $K : 4$)

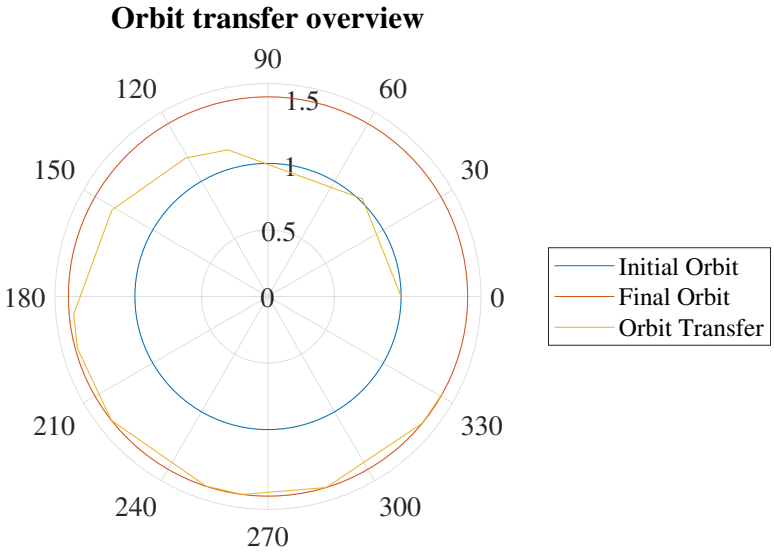

Figure 120: Trajectory from initial to final orbit ($N : 3$, $K : 4$)

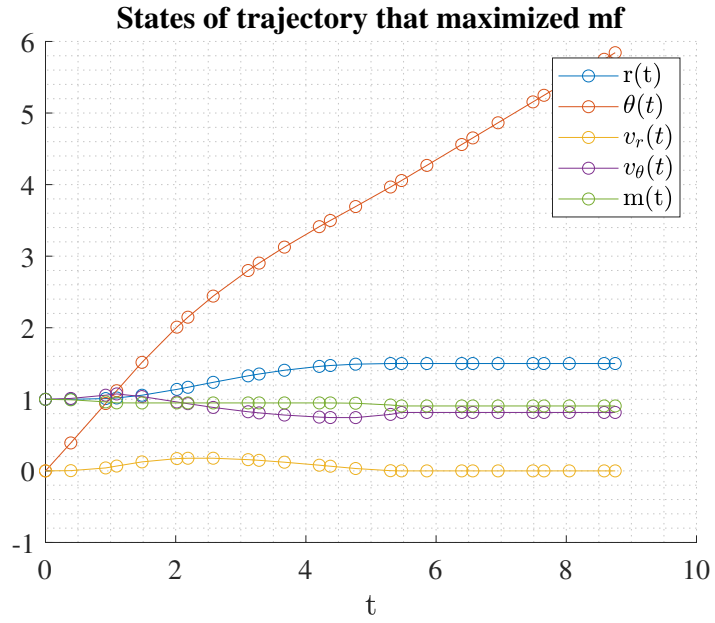

Figure 121: States for trajectory that maximized terminal mass ($N : 3$, $K : 8$)

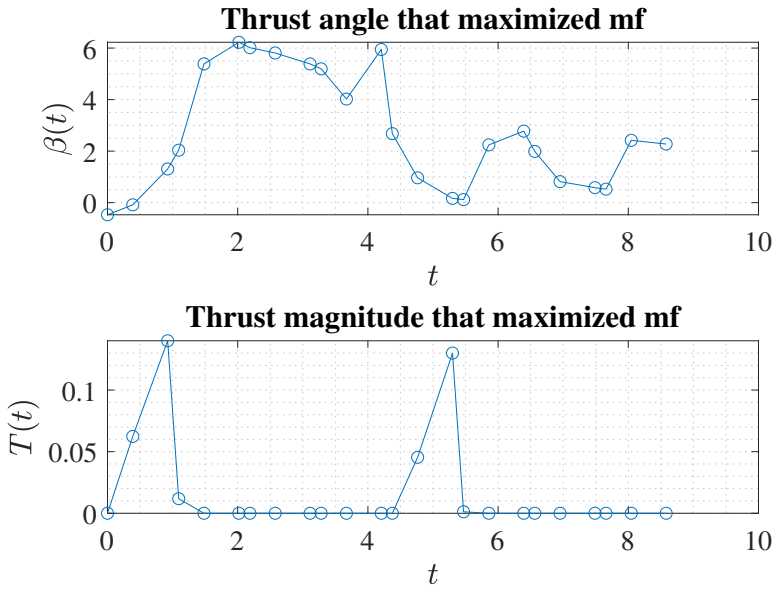

Figure 122: Control that maximized terminal mass ($N : 3$, $K : 8$)

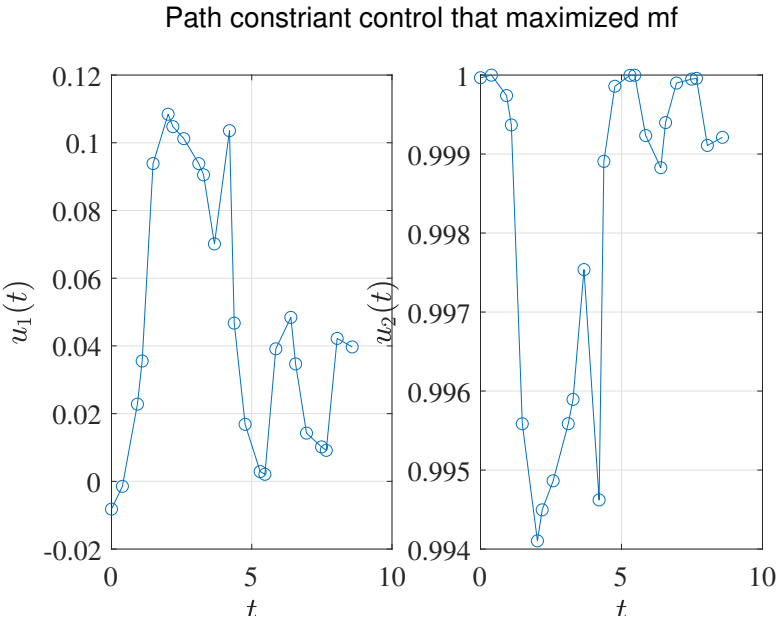

Figure 123: Path constrained control that maximized terminal mass ($N : 3$, $K : 8$)

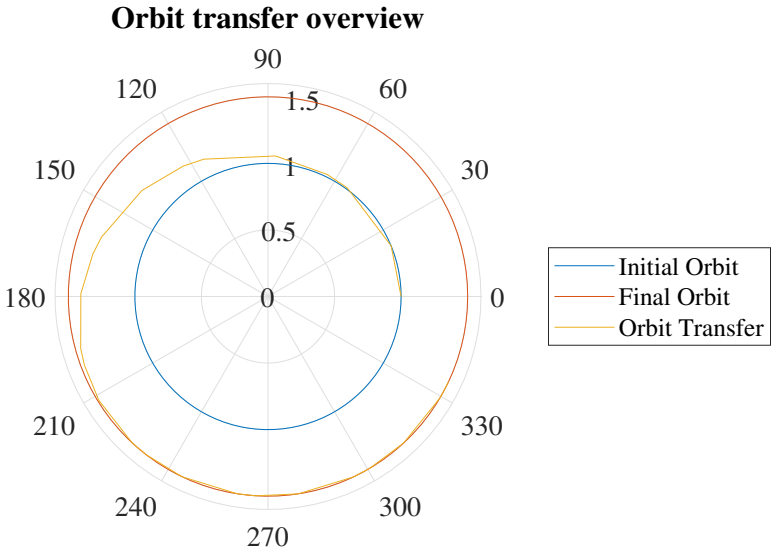

Figure 124: Trajectory from initial to final orbit ($N : 3$, $K : 8$)

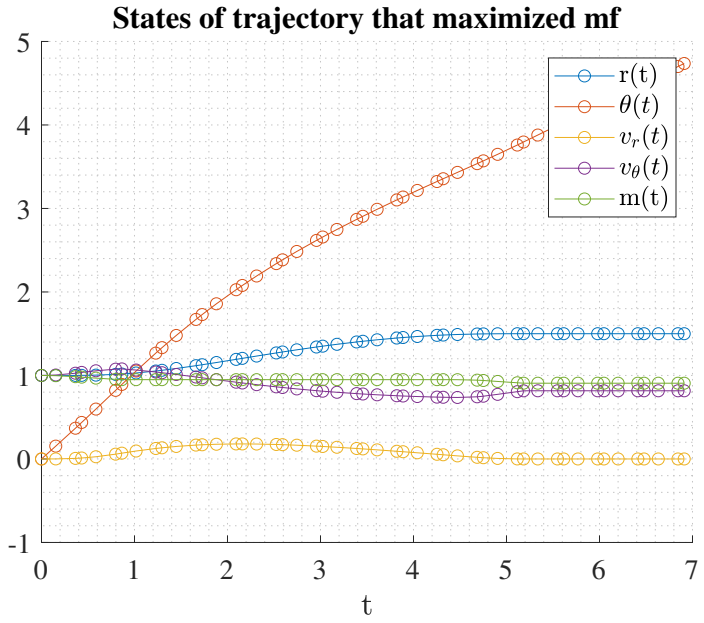

Figure 125: States for trajectory that maximized terminal mass ($N : 3$, $K : 16$)

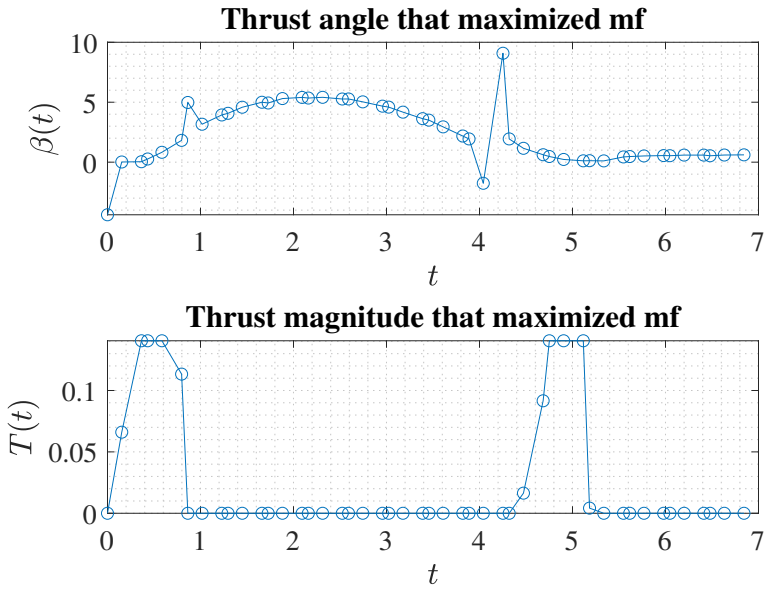

Figure 126: Control that maximized terminal mass ($N : 3$, $K : 16$)

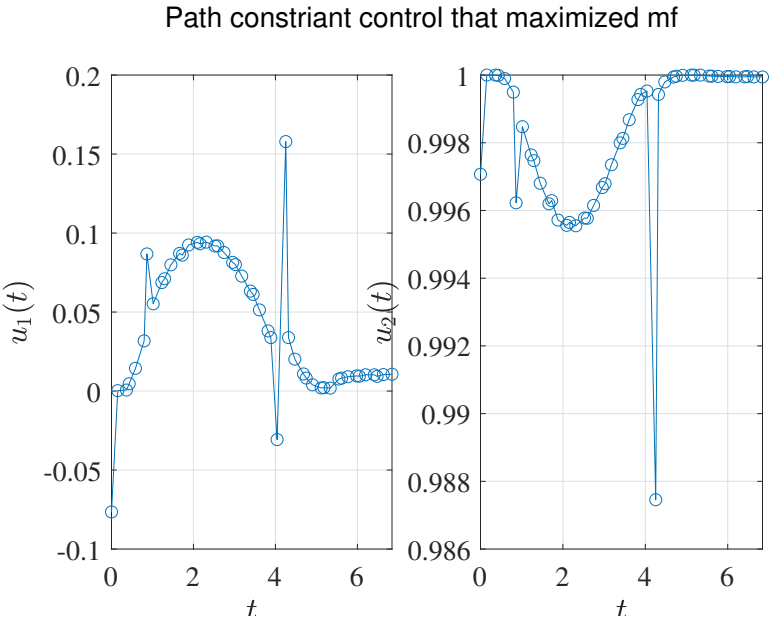

Figure 127: Path constrained control that maximized terminal mass ($N : 3$, $K : 16$)

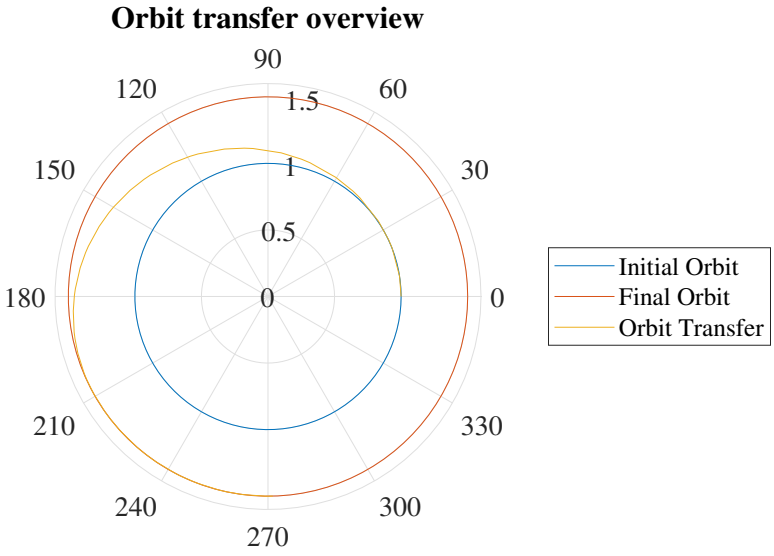

Figure 128: Trajectory from initial to final orbit ($N : 3$, $K : 16$)

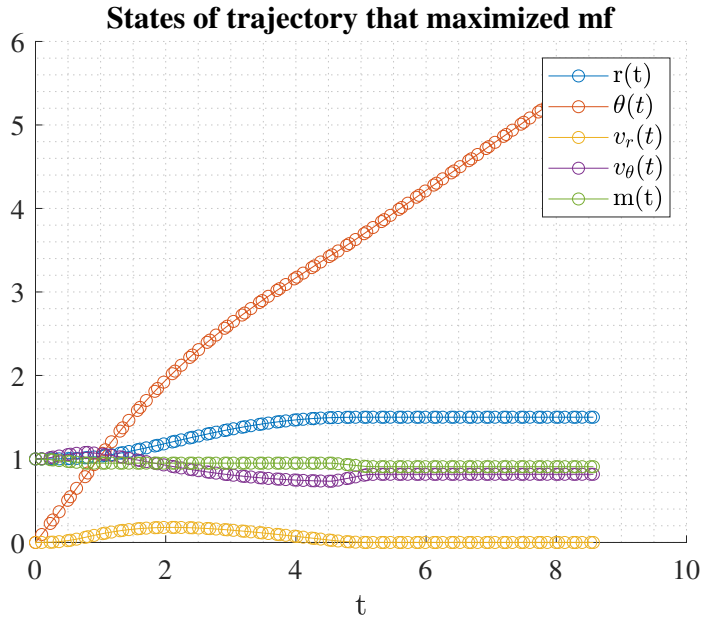

Figure 129: States for trajectory that maximized terminal mass ($N : 3$, $K : 32$)

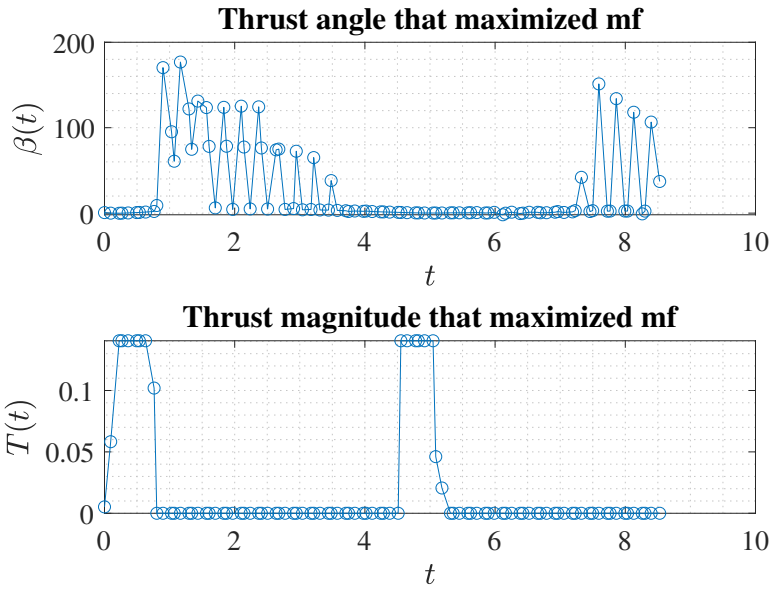

Figure 130: Control that maximized terminal mass ($N : 3$, $K : 32$)

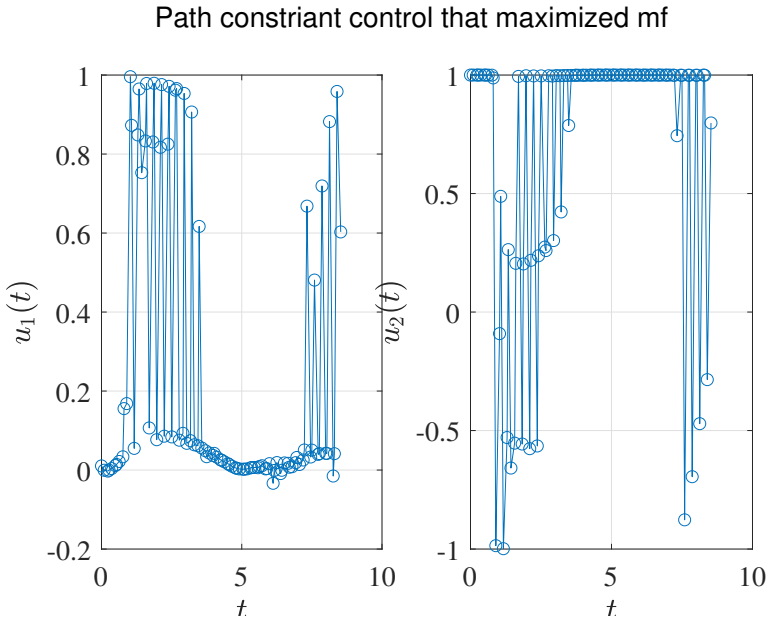

Figure 131: Path constrained control that maximized terminal mass ($N : 3$, $K : 32$)

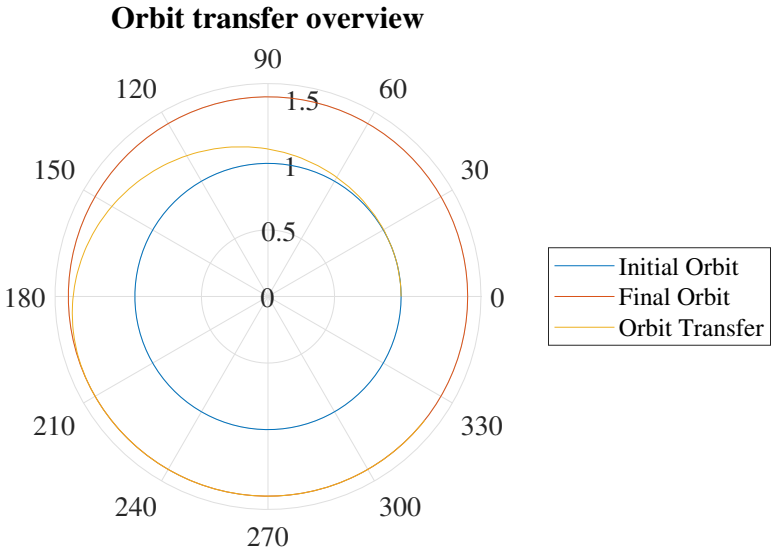

Figure 132: Trajectory from initial to final orbit ($N : 3$, $K : 32$)

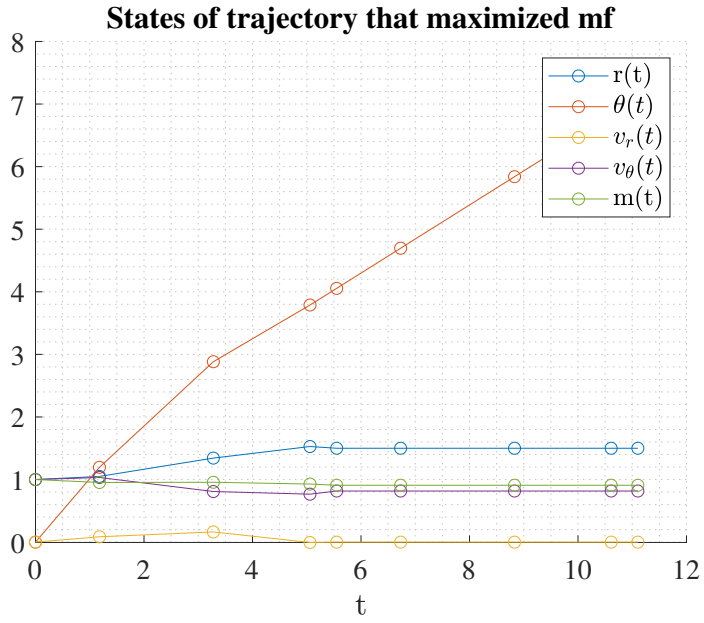

Figure 133: States for trajectory that maximized terminal mass ($N : 4$, $K : 2$)

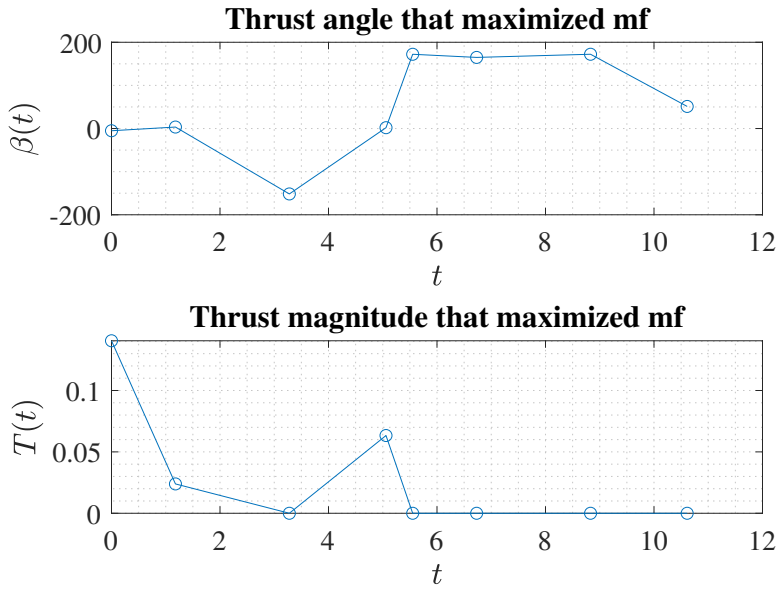

Figure 134: Control that maximized terminal mass ($N : 4$, $K : 2$)

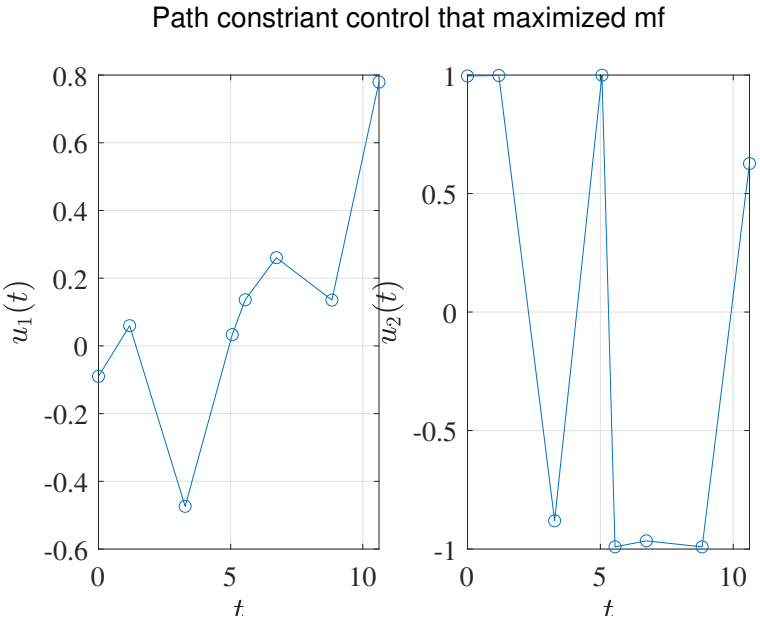

Figure 135: Path constrained control that maximized terminal mass ($N : 4$, $K : 2$)

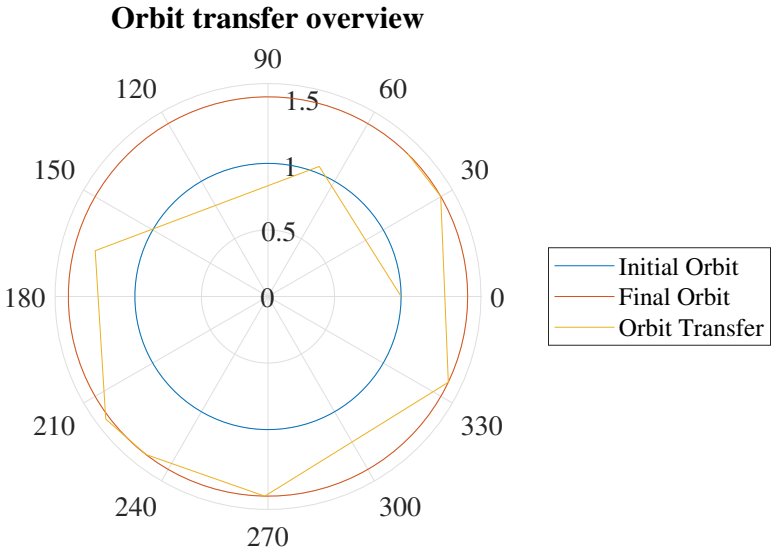

Figure 136: Trajectory from initial to final orbit ($N : 4$, $K : 2$)

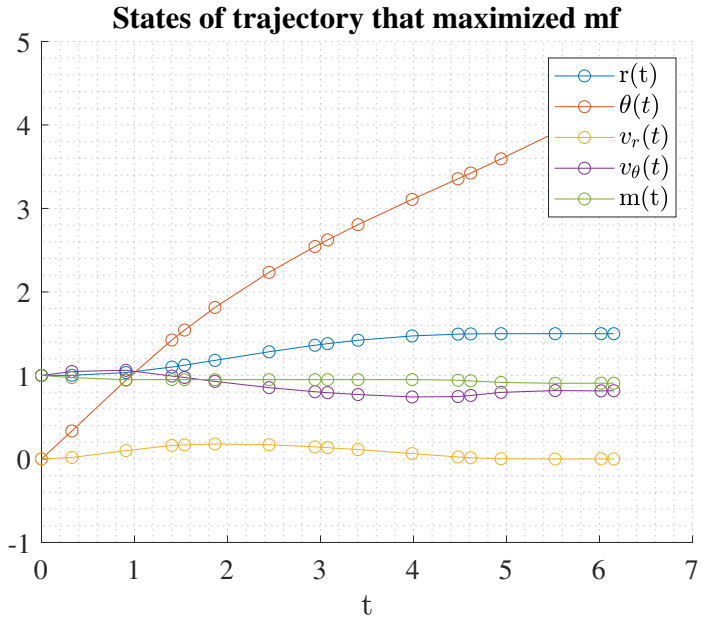

Figure 137: States for trajectory that maximized terminal mass ($N : 4$, $K : 4$)

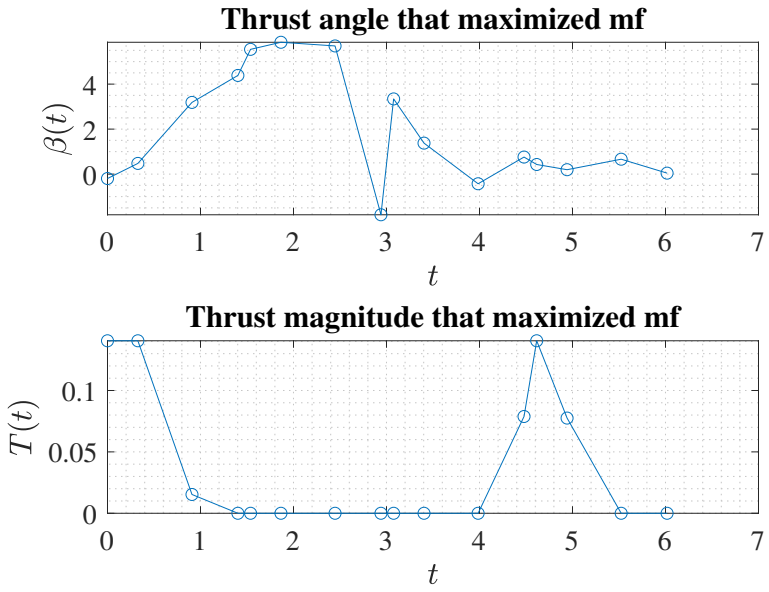

Figure 138: Control that maximized terminal mass ($N : 4$, $K : 4$)

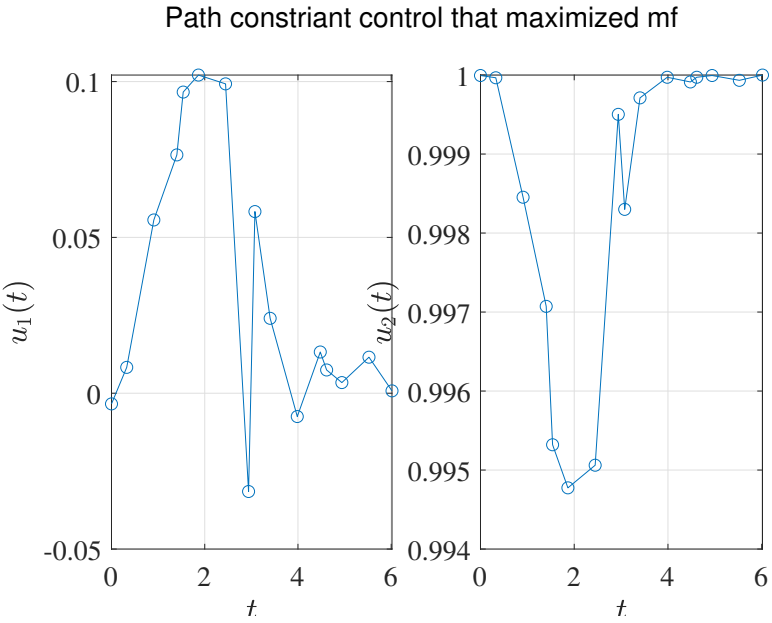

Figure 139: Path constrained control that maximized terminal mass ($N : 4$, $K : 4$)

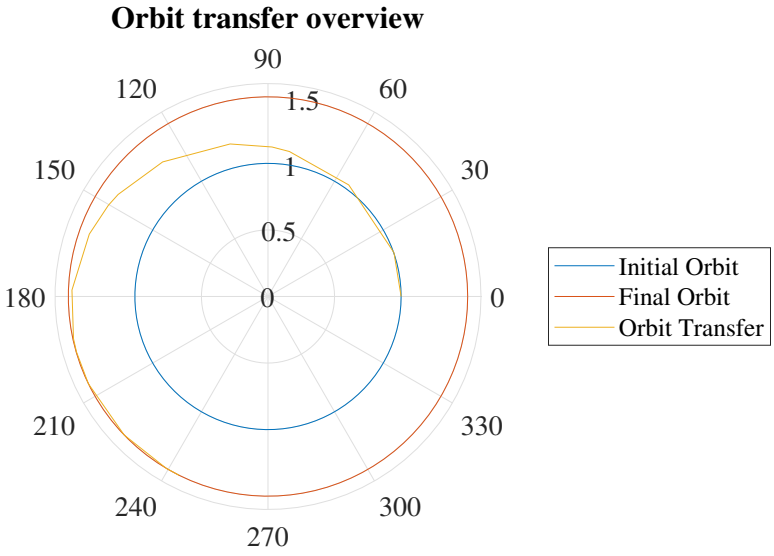

Figure 140: Trajectory from initial to final orbit ($N : 4$, $K : 4$)

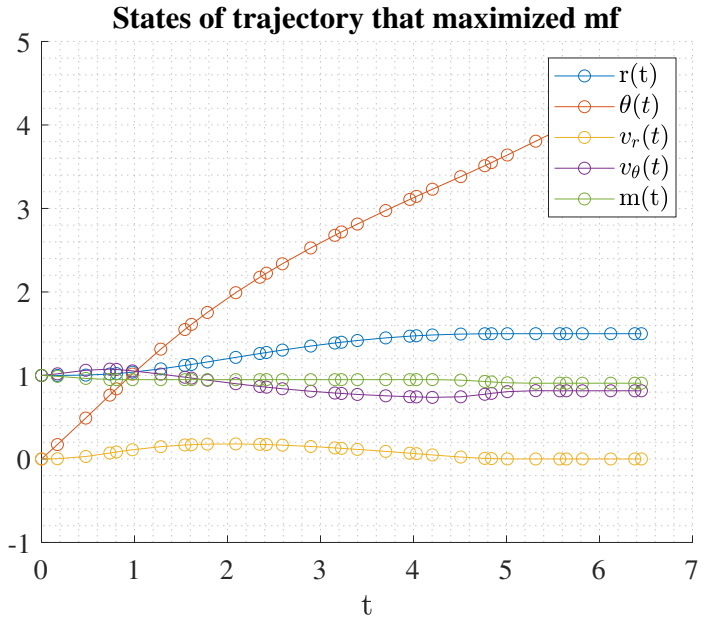

Figure 141: States for trajectory that maximized terminal mass ($N : 4$, $K : 8$)

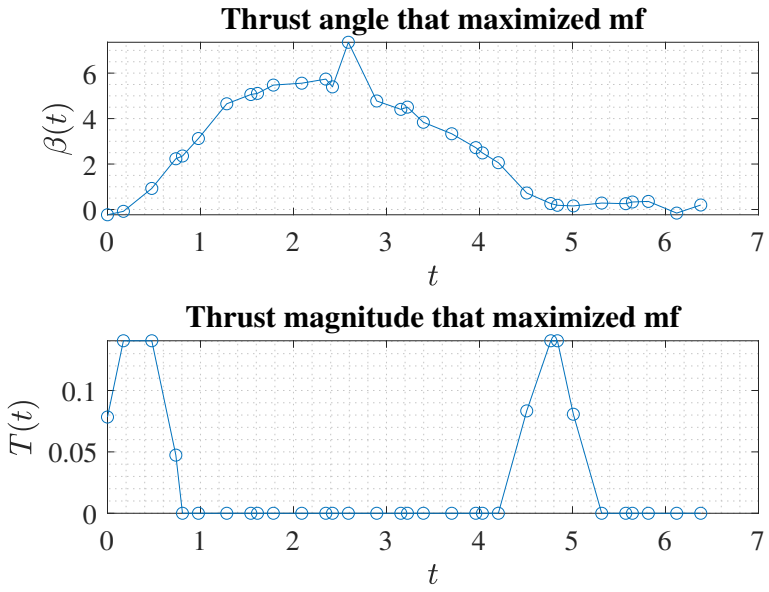

Figure 142: Control that maximized terminal mass ($N : 4$, $K : 8$)

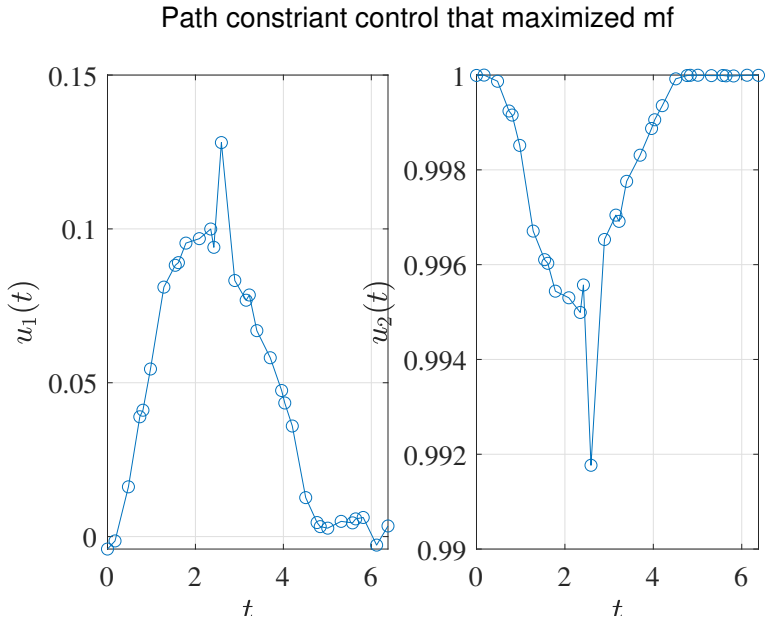

Figure 143: Path constrained control that maximized terminal mass ($N : 4$, $K : 8$)

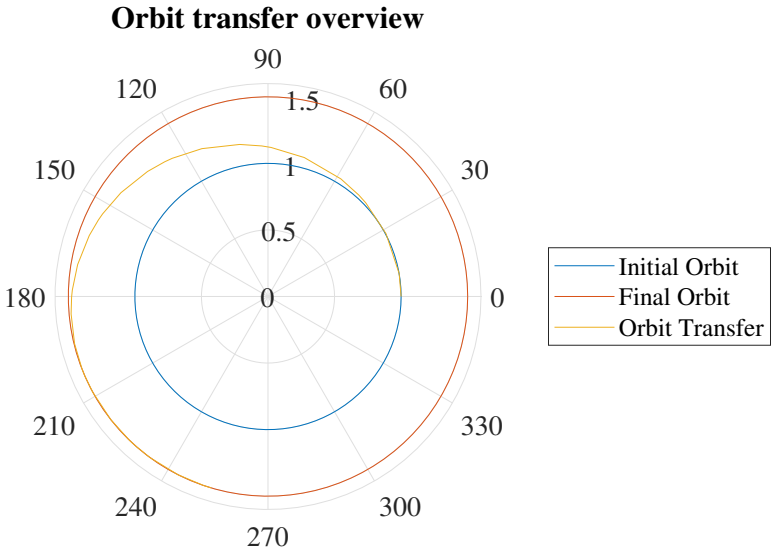

Figure 144: Trajectory from initial to final orbit ($N : 4$, $K : 8$)

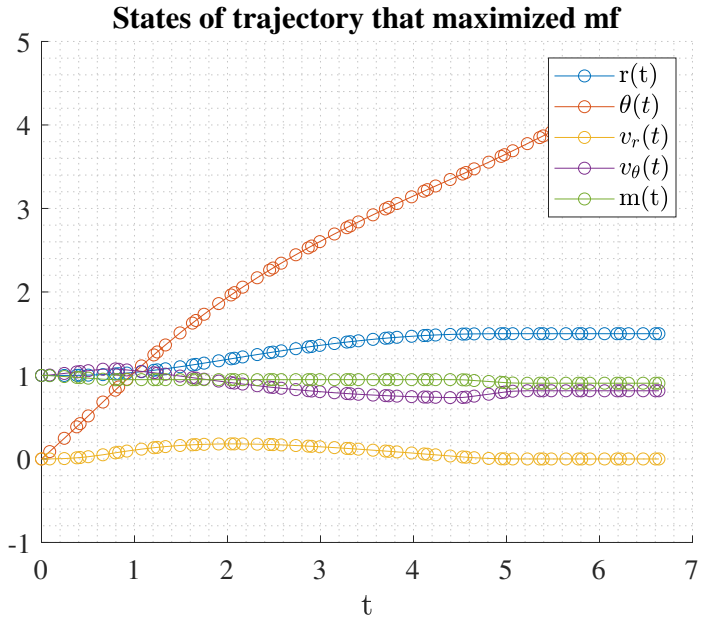

Figure 145: States for trajectory that maximized terminal mass ($N : 4$, $K : 16$)

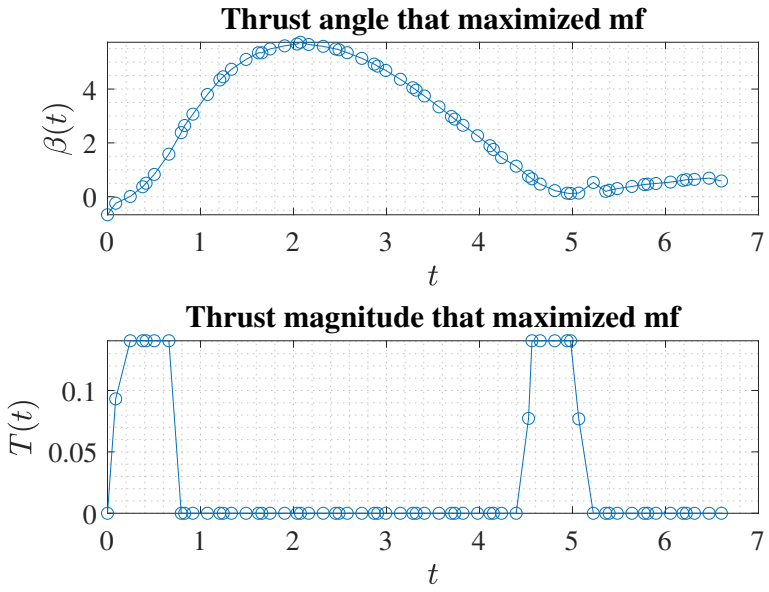

Figure 146: Control that maximized terminal mass ($N : 4$, $K : 16$)

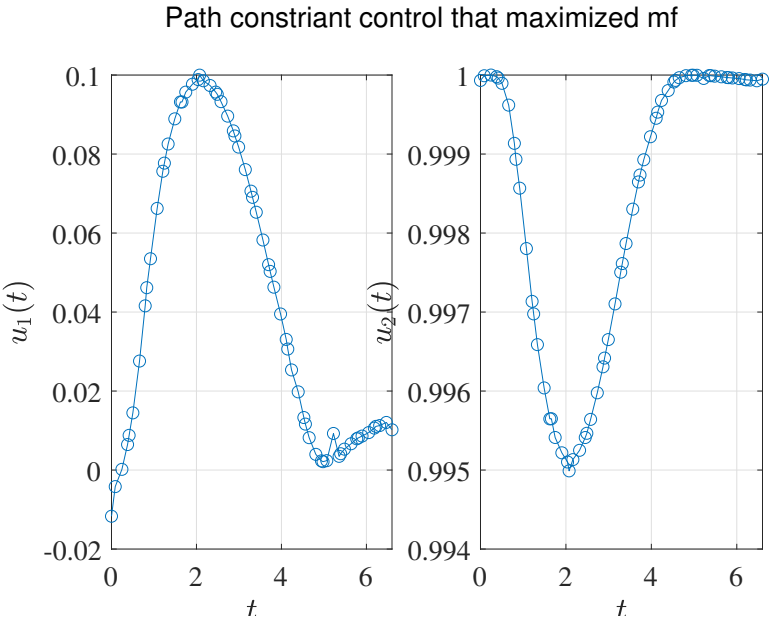

Figure 147: Path constrained control that maximized terminal mass ($N : 4$, $K : 16$)

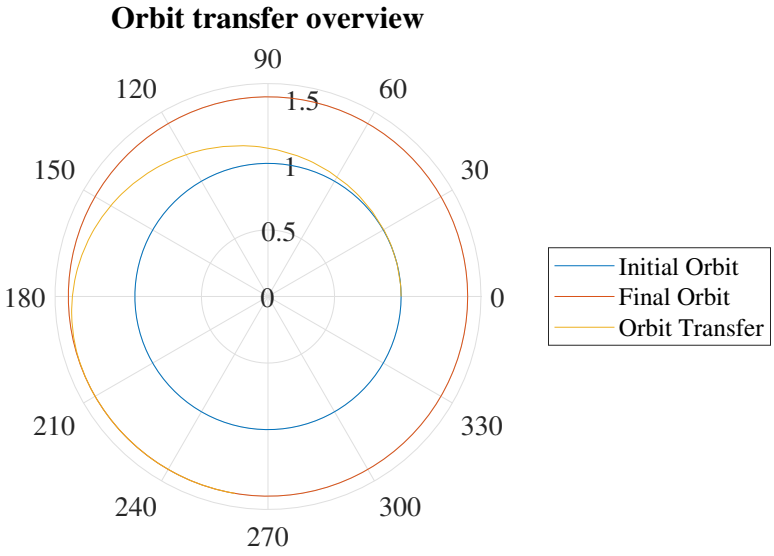

Figure 148: Trajectory from initial to final orbit ($N : 4$, $K : 16$)

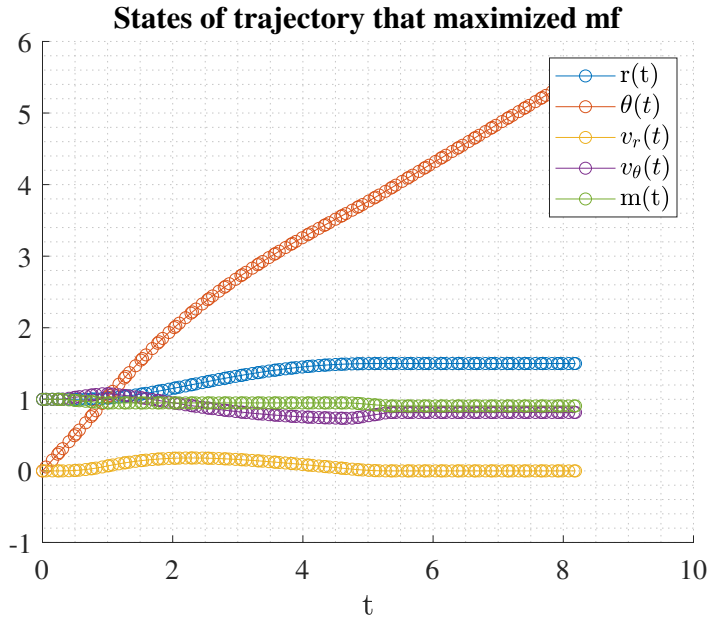

Figure 149: States for trajectory that maximized terminal mass ($N : 4$, $K : 32$)

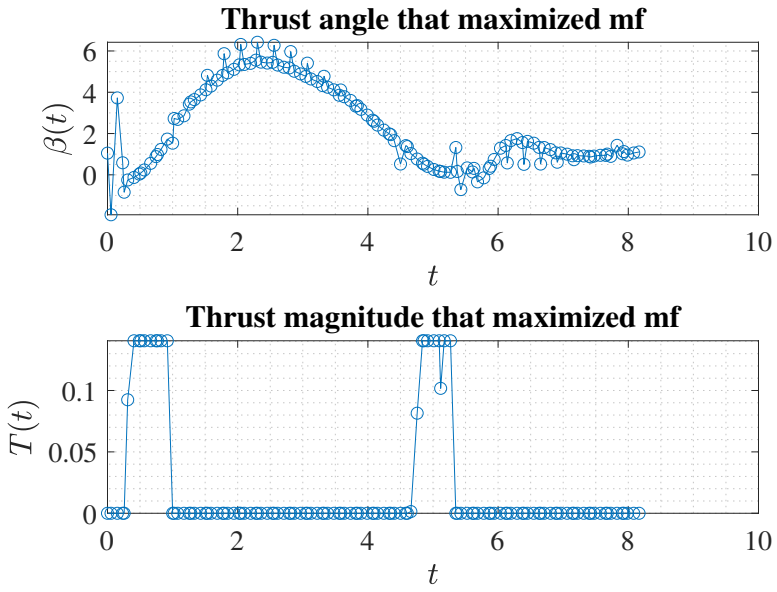

Figure 150: Control that maximized terminal mass ($N : 4$, $K : 32$)

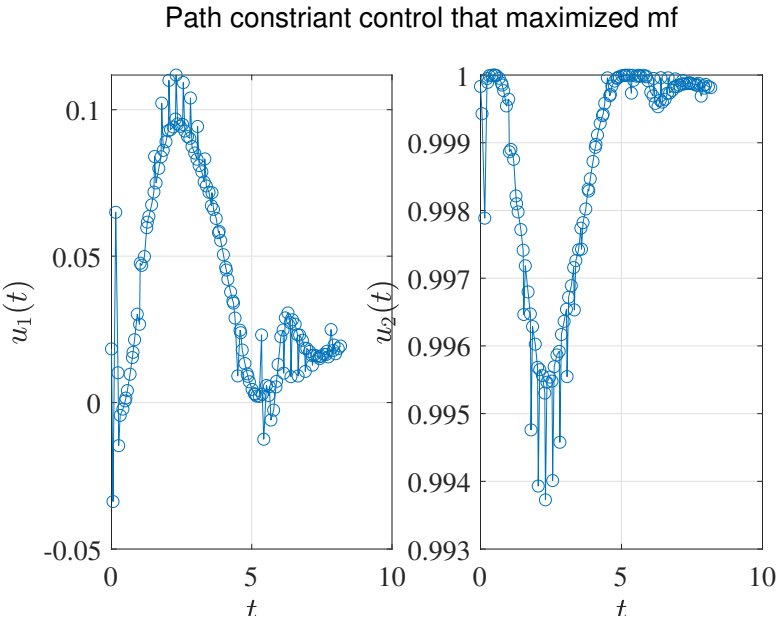

Figure 151: Path constrained control that maximized terminal mass ($N : 4$, $K : 32$)

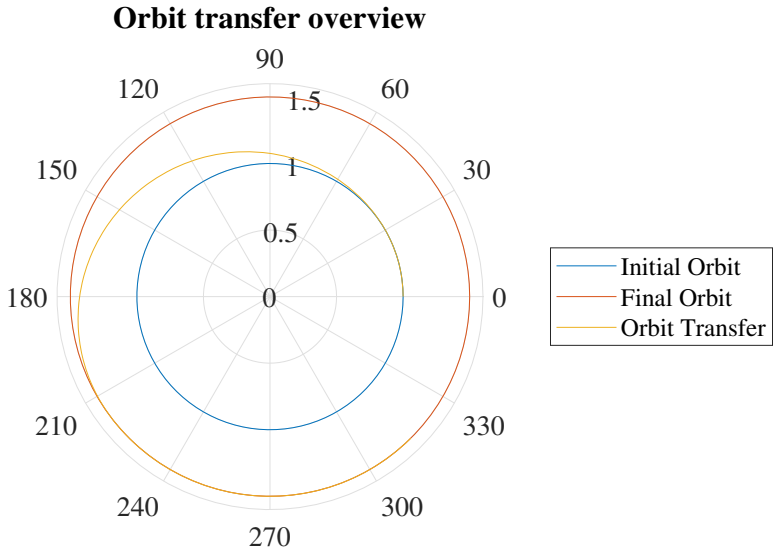

Figure 152: Trajectory from initial to final orbit ($N : 4$, $K : 32$)

4 Discussion/Comparisons

4.1 Minimize t_f Analysis/Comparison

The first objective of this problem was to minimize t_f with control parameterized by a polynomial of degrees N = (2,3) with intervals K = (2,4,8,16,32). This was done with two different sets of controls. The first set of controls were thrust angle β and thrust magnitude T. The second set of controls were u_1, u_2 and thrust magnitude T. Along with these controls, a path constraint was implemented in the form $u_1 + u_2 = 1$. Both sets of controls produced very similar results. The optimized objective function for all combinations is shown in Figure 153. Interestingly, both sets of controls produced the same objective function for all interval cases with polynomial degree 4. Polynomial degree 3 produced the same results for all interval cases except K = 4. It seems like both polynomial degrees converge on to the same result as the number of intervals increases. The converged objective function, t_f, is 3.247, which is the same value produced by the indirect shooting methods used for the midterm. Because of this, the result, 3.2456, produced by the combination N = 4 and K = 2, is most likely unrealistic. As mentioned in the individual sections, the orbit transfers for the low-interval cases are not desired due to the large segments between LGR points. For instance, the lower bound of the radius, r, was set to 1 for this problem. While the constraint is satisfied at the LGR points, the path between the points intersects the radius bound. In this case, the optimizer is satisfying all of its constraints and bounds but the solution is not realistic. While the objective function is optimized equally with both sets of controls, the constrained control may be preferred for the orbit transfer problem. Looking at all of the control plots for the cases where β is a control, it is noticeable that most of the combinations have an initial thrust angle of approximately -335°that sweeps to -49°, while a few other combinations have an initial thrust angle of approximately 25°that sweeps to 311°. Now looking at all the control plots where u_1 and u_1 were controls, it is clear that the thrust angle for all combinations starts at approximately 25°and sweep to 311°. While both angle sweeps are mathematically the same, it is much more desirable for the nonlinear optimizer to deal with consistent angles that are constrained. The thrust angle for the u_1, u_2 cases are constrained do to the trigonometric path constraint applied. When using u_1 and u_1 with the path constraint, the results are very similar to those using the indirect shooting methods. A major reason why Legendre-Gauss-Radau collocation is a better choice than the indirect methods is the cost of computation. LGR collocation is more computational efficient than the indirect methods. For instance, solving the orbit transfer problem with Indirect Multiple shooting with 16 intervals took 11 seconds to find a solution while LGR collocation took less than 1.5 seconds. This is significant because a lot of real-world applications require quick and accurate solutions. It was shown with this study that LGR collocation can be both quick and accurate for the case of minimizing t_f when performing an orbit transfer.

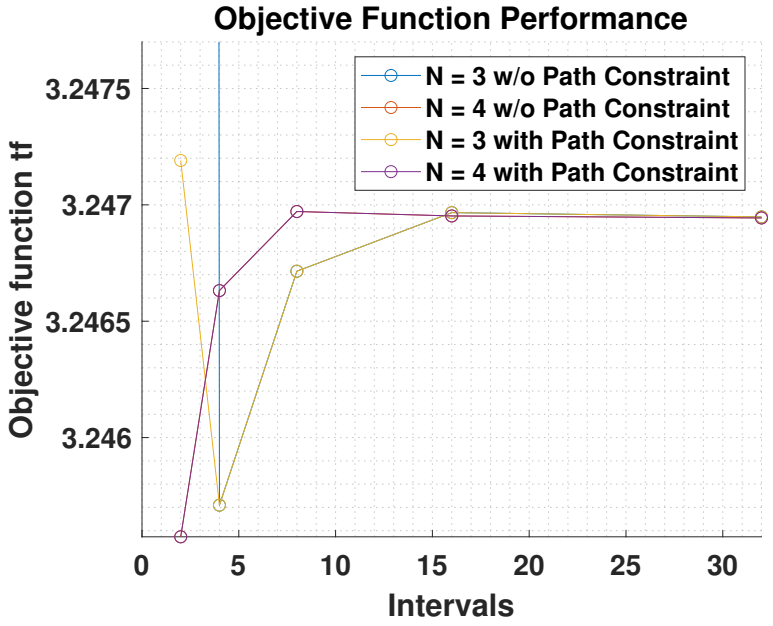

Figure 153: Objective function t_f comparison

4.2 Maximize m_f Analysis/Comparison

The second objective of this problem was to maximize m_f with control parameterized by a polynomial of degrees N = (2,3) with intervals K = (2,4,8,16,32). This was done with two different sets of controls. The first set of controls were thrust angle β and thrust magnitude T. The second set of controls were u_1, u_2 and thrust magnitude T. Along with these controls, a path constraint was implemented in the form $u_1+u_2 = 1$. Similar to the comparison of the minimize t_f cases, both maximize m_f cases had very similar results in terms of the objective function. Figure 154 shows a comparison of the optimized objective function for all of the maximize m_f combinations. The case that reached the terminal orbit with the largest mass remaining, 0.90894, was with β control, parameterized by a 3^{rd} degree polynomial with 4 intervals. Similar to the minimize t_f cases, the lower interval combinations may optimize the objective function best but the control and flight path may not be desired. As explained in previous sections, the orbit for the low-interval cases are unrealistic and not smooth. For a problem such as an orbit transfer, it is better to utilize more intervals so that more LGR points can be evaluated. This is due to the optimal control being highly nonlinear. As shown in the midterm with the direct methods, lower number of intervals do not approximate the optimal control for the orbit transfer problem very well. While both sets of controls converged to the same objective function, the u_1 and u_2 set seem to produce a much more contained control than β. For example, Figure 155 shows a comparison of the thrust angle between the

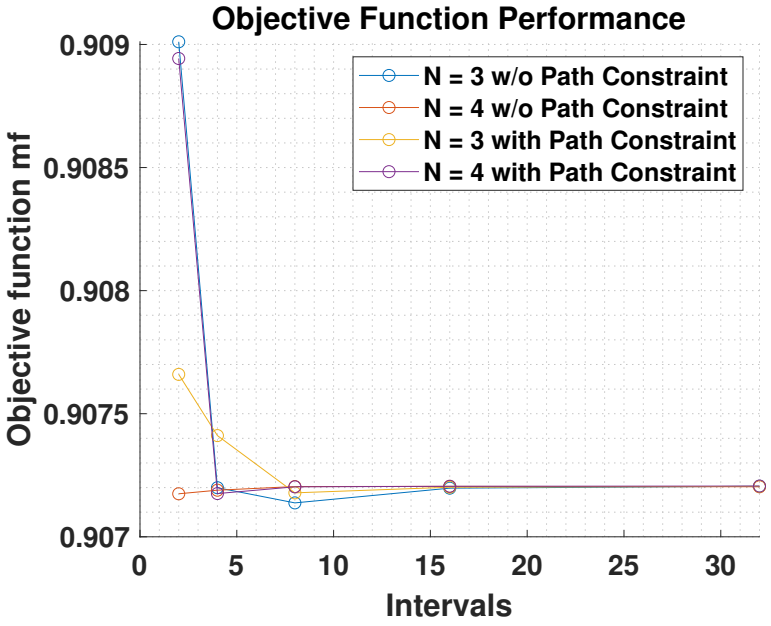

Figure 154: Objective function m_f comparison

two sets of controls for N=3 and K = 8. The top plot shows the thrust angle when the controls were β and thrust magnitude T. The bottom plot shows the thrust angle when the controls were u_1, u_2 and thrust magnitude T. While both thrust angle sweeps optimized the objective function, it is evident that the path constrained control is much more desirable. The β control starts at a thrust angle of 0°and sweeps down to about -900° while the u_1, u_2 controls keep the thrust angle contained between 0°and 6°. This path constrained control is much easier on the nonlinear optimizer, especially when bounds and initial guesses of variables aren't intuitively set. In other words, a path constrained control is more robust to bad bounds and initial guesses. For the current problem, all of the combinations were able to terminate with the status "Optimal Solution Found" or "Solved to an Acceptable Solution" due to fine tuning of the initial guesses and bounds on the variables. As explained in the individual analysis section, the control plots for these cases seem to have a significant amount of "chattering". The differential equations, defect constraints, path constraints and objective functions are implemented correctly, yet the optimal control still shows chattering. While the chattering can be slightly minimized by tuning initial guesses and bounds, there seems to be an issue with the nonlinear optimizer IPOPT or numerical integration errors.

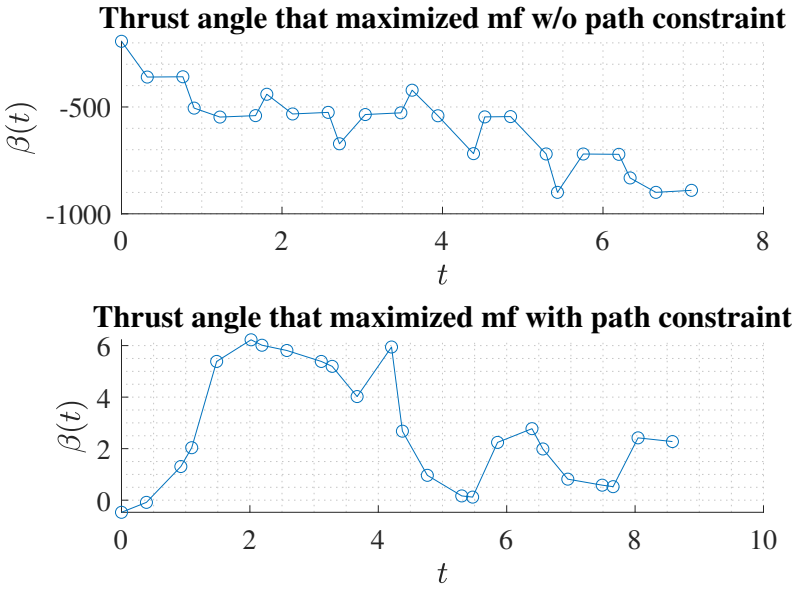

Figure 155: Thrust angle comparison between unconstrained and constrained control (N:3,K:8)

5 Conclusion/Future Work

The goal of this project was to solve the orbit transfer problem, from the midterm, using multiple-interval Legendre-Gauss-Radau collocation. For the midterm, the only control was thrust angle β, however, for this project, the problem was modified so that both the thrust angle β and magnitude T were controls. For the project, two objective functions were considered. The objective functions were to minimize terminal time t_f and maximize terminal massm_f. While these two objectives were equivalent for the midterm, they are not in this case since the thrust magnitude is no longer modeled as a constant. The problem was first solved with the control parameratized by polynomials degrees N = (3,4) with intervals K = (2,4 8,16,32). The problem was then solved again with the same parameterization except instead of β being a control, u_1 and u_2 were used along with the equality path constraint $u_1^2 + u_2^2 = 1$. A more detailed explanation can be found in section 2.2.2. For the cases where the objective function was to minimize t_f, the results were expected to be very similar to those obtained using the indirect methods in the midterm. This was indeed true with the exception of a few cases with lower intervals. Because Legendre-Gauss-Radau collocation does not require calculus of variations, more intervals are required to obtain a realistic solution. Using u_1 and u_2 as controls, the control that optimized the objective function was almost exactly the same as indirect shooting. The effect of using the control u_1, u_2 and a path constraint was much more evident in the cases where the objective was to maximize m_f.

When β was used as a control, the thrust angle, in some cases, swept over multiple cycles. When u_1 and u_2 were used, the thrust angle remained constrained to a single cycle. This makes it much easier for the nonlinear optimizer to converge on an optimal solution. It is also undesirable in real world applications to have controls that make large jumps. This project has proved a few power things. First, multiple-interval Legendre-Gauss-Radau collocation is an extremely powerful numerical method for solving optimal control problem. It is very accurate and is significantly more efficient than indirect/direct multiple shooting. This study showed that increasing the intervals with collocation has very little impact on the execution performance. This is most likely due to collocation solving a large system of equations. The last thing that the project proved is that adding path constraints to optimization problems is very beneficial and can help the optimizer find a solution more effectively.

It would be interesting to further investigate a way to eliminate the "chattering" for the maximize m_f cases. Possibly an additional constraint can be added to the control that limits the chattering. Lastly, it would be nice to solve the orbit transfer problem using both indirect and direct collocation to compare results with other methods used.

6 Appendix

Listing 1: orbitTransferMain.m

```
% ------------------------------------------------------------%
% Orbit-Trnasfer Problem %
% ------------------------------------------------------------%
clear all; close all;
format long
% ------------------------------------------------------------ %
% 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!!! %
% ------------------------------------------------------------ %

%% Save figures and latex table.
% Set to 0 when debugging
save_figs = 0;
create_latex_table = 0;
plot_comparisons = 0;

if save_figs
    set(0,'DefaultFigureWindowStyle','normal')
    close_figs = 1;
else
    set(0,'DefaultFigureWindowStyle','docked')
    close_figs = 0;
end
%% 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 ]; % polynomial degrees
```

```
33  k_list = [32 ]; % intervals
34  numn = numel(n_list);
35  numk = numel(k_list);
36  %% Set Globals
37  % set gloabl constants
38  CONSTANTS.MU = 1;
39  CONSTANTS.m0 = 1;
40  CONSTANTS.ve = 1.8658344;
41  % set number of states
42  nstates = 5;
43
44  % set number of controls and paths
45  if path_constraint
46      ncontrols = 3;
47      npaths = 1;
48  else
49      ncontrols = 2;
50      npaths = 0;
51  end
52
53  %% Solve Optimal Control Problem
54  % counter for table
55  count = 1;
56
57  for nIdx = 1:numn
58
59      for kIdx = 1:numk
60
61      % Set polynomial degree and number of intervals
62      N = n_list(nIdx); % number of polynomial degree
63      numIntervals = k_list(kIdx); % number of intervals
64
65      % Bounds on State and Control
66      % thetaf and mf are free
67      r0 = 1; theta0 = 0; vr0 = 0; vtheta0 = 1; m0 = 1;
68      rf = 1.5; vrf = 0; vthetaf = sqrt(1/rf);
69
70      rmin = 1; rmax = 1.5;
71      thetamin = 0; thetamax = 4*pi;
72      vrmin = -10; vrmax = 10;
73      vthetamin = -10; vthetamax = 10;
74      mmin = 0.1; mmax = m0;
75      t0min = 0; t0max = 0;
76      tfmin = 0; tfmax = 25;
77      u3min = 0; u3max = 0.1405; % thrust
78      if maximize_mass
79          rmax = 1.6;
80          mmin = 0.7;
81          tfmax = 100;
82      end
83      if path_constraint
84          % set bounds for control with path contraint
85          u1min = -1.5; u1max = 1.5; % sin(beta)
86          u2min = -1.5; u2max = 1.5; % cos(beta)
87          u1min = -2; u1max = 2; % sin(beta)
88          u2min = -2; u2max = 2; % cos(beta)
89      else
90          % set bound for beta
91  % u1min = -2*pi; u1max = 2*pi; % beta
92          u1min = -2*pi; u1max = 2*pi; % beta
93      end
94
95      % Create Leguandre Gauss Points
96      meshPoints = linspace(-1,1,numIntervals+1).';
97      polyDegrees = N*ones(numIntervals,1);
98      [tau,w,D] = lgrPS(meshPoints,polyDegrees);
99      psStuff.tau = tau; psStuff.w = w; psStuff.D = D; NLGR = length(w);
100
```

```
101     % Set the bounds on the NLP variables.
102     zrmin = rmin*ones(length(tau),1);
103     zrmax = rmax*ones(length(tau),1);
104     zrmin(1) = r0; zrmax(1) = r0;
105     zrmin(end) = rf; zrmax(end) = rf;
106
107     zthetamin = thetamin*ones(length(tau),1);
108     zthetamax = thetamax*ones(length(tau),1);
109     zthetamin(1) = theta0; zthetamax(1) = theta0;
110
111     zvrmin = vrmin*ones(length(tau),1);
112     zvrmax = vrmax*ones(length(tau),1);
113     zvrmin(1) = vr0; zvrmax(1) = vr0;
114     zvrmin(end) = vrf; zvrmax(end) = vrf;
115
116     zvthetamin = vthetamin*ones(length(tau),1);
117     zvthetamax = vthetamax*ones(length(tau),1);
118     zvthetamin(1) = vtheta0; zvthetamax(1) = vtheta0;
119     zvthetamin(end) = vthetaf; zvthetamax(end) = vthetaf;
120
121     zmmin = mmin*ones(length(tau),1);
122     zmmax = mmax*ones(length(tau),1);
123     zmmin(1) = m0; zmmax(1) = m0;
124
125     zu1min = u1min*ones(length(tau)-1,1);
126     zu1max = u1max*ones(length(tau)-1,1);
127     zu3min = u3min*ones(length(tau)-1,1);
128     zu3max = u3max*ones(length(tau)-1,1);
129
130     % Contruct vector of bounds.
131     if path_constraint
132         % using u1, u2, u3
133         zu2min = u2min*ones(length(tau)-1,1);
134         zu2max = u2max*ones(length(tau)-1,1);
135
136         zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zu2min; zu3min; t0min; tfmin];
137         zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zu2max; zu3max; t0max; tfmax];
138     else
139         % using u1, u3
140         zmin = [zrmin; zthetamin; zvrmin; zvthetamin; zmmin; zu1min; zu3min; t0min; tfmin];
141         zmax = [zrmax; zthetamax; zvrmax; zvthetamax; zmmax; zu1max; zu3max; t0max; tfmax];
142     end
143
144     % Set the bounds on the NLP constraints
145     % There are NSTATES sets of defect constraints.
146     defectMin = zeros(nstates*(length(tau)-1),1);
147     defectMax = zeros(nstates*(length(tau)-1),1);
148     if path_constraint
149         % There is a path constraint
150         pathMin = ones(length(tau)-1,1); pathMax = ones(length(tau)-1,1);
151
152     else
153         % No path constraint
154         pathMin = []; pathMax = [];
155     end
156
157     objMin = -inf;
158     objMax = inf;
159     Fmin = [objMin; defectMin; pathMin];
160     Fmax = [objMax; defectMax; pathMax];
161
162     % Supply an initial guess
163     thetaguessend = 2.5;
164     rguess = linspace(r0,rf,NLGR+1).';
165     thetaguess = linspace(theta0,thetaguessend,NLGR+1).';
166     vrguess = linspace(vr0,vr0,NLGR+1).';
167     vthetaguess = linspace(vtheta0,vthetaf,NLGR+1).';
168     mguess = linspace(mmax,mmin,NLGR+1).';
```

```
169       u1guess = linspace(u1min,u1max,NLGR).';
170       u3guess = linspace(u3max,u3max,NLGR).';
171       t0guess = 0;
172       tfguess = 3;
173
174       if path_constraint
175           % contruct u1guess and u2guess for path contraint
176           u1guess = linspace(1,1,NLGR).';
177           u2guess = linspace(0,0,NLGR).';
178           z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;u2guess;u3guess;t0guess;tfguess];
179       else
180           z0 = [rguess;thetaguess;vrguess;vthetaguess;mguess;u1guess;u3guess;t0guess;tfguess];
181       end
182
183       %-------------------------------------------------------------------------%
184       % Generate derivatives and sparsity pattern using Adigator %
185       %-------------------------------------------------------------------------%
186       % - Constraint Function Derivatives
187       xsize = size(z0);
188       x = adigatorCreateDerivInput(xsize,'z0');
189       output = adigatorGenJacFile('orbitTransferFun',{x});
190       S_jac = output.JacobianStructure;
191       [iGfun,jGvar] = find(S_jac);
192
193       % - Objective Function Derivatives
194       xsize = size(z0);
195       x = adigatorCreateDerivInput(xsize,'z0');
196       output = adigatorGenJacFile('orbitTransferObj',{x});
197       grd_structure = output.JacobianStructure;
198
199       %-------------------------------------------------------------------------%
200       % set IPOPT callback functions
201       %-------------------------------------------------------------------------%
202       funcs.objective = @(Z)orbitTransferObj(Z);
203       funcs.gradient = @(Z)orbitTransferGrd(Z);
204       funcs.constraints = @(Z)orbitTransferCon(Z);
205       funcs.jacobian = @(Z)orbitTransferJac(Z);
206       funcs.jacobianstructure = @()orbitTransferJacPat(S_jac);
207       options.ipopt.hessian_approximation = 'limited-memory';
208         options.ipopt.limited_memory_aug_solver = 'extended';
209   % options.ipopt.derivative_test = 'first-order';
210   % options.ipopt.constraint_violation_norm_type = 'max-norm';
211       %-------------------------------------------------------------------------%
212       % Set IPOPT Options %
213       %-------------------------------------------------------------------------%
214       options.ipopt.tol = 1e-8;
215       options.ipopt.linear_solver = 'ma57';%'ma57'; %'mumps';
216       options.ipopt.max_iter = 4000;
217       options.ipopt.mu_strategy = 'adaptive';
218       options.ipopt.ma57_automatic_scaling = 'yes';
219       options.ipopt.print_user_options = 'no';
220       options.ipopt.output_file = ['orbitTransfer','IPOPTinfo.txt']; % print output file
221       options.ipopt.print_level = 5; % set print level default
222
223       options.lb = zmin; % Lower bound on the variables.
224       options.ub = zmax; % Upper bound on the variables.
225       options.cl = Fmin; % Lower bounds on the constraint functions.
226       options.cu = Fmax; % Upper bounds on the constraint functions.
227
228       %-------------------------------------------------------------------------%
229       % Call IPOPT
230       %-------------------------------------------------------------------------%
231       [z, info] = ipopt(z0,funcs,options);
232
233       % Extract the state and control from the decision vector z.
234       % Remember that the state is approximated at the LGR points
235       % plus the final point, while the control is only approximated
236       % at only the LGR points.
```

```
237         r = z(1:NLGR+1);
238         theta = z(NLGR+2:2*(NLGR+1));
239         vr = z(2*(NLGR+1)+1:3*(NLGR+1));
240         vtheta = z(3*(NLGR+1)+1:4*(NLGR+1));
241         m = z(4*(NLGR+1)+1:5*(NLGR+1));
242         u1 = z(5*(NLGR+1)+1:5*(NLGR+1)+NLGR);
243         if path_constraint
244             u2 = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
245             u3 = z(5*(NLGR+1)+2*NLGR+1:5*(NLGR+1)+3*NLGR);
246             beta = atan2(u1,u2);
247             beta = unwrap(beta)*180/pi;
248         else
249             u3 = z(5*(NLGR+1)+NLGR+1:5*(NLGR+1)+2*NLGR);
250             beta = unwrap(u1)*180/pi;
251     % beta = u1;
252         end
253         t0 = z(end-1);
254         tf = z(end);
255         t = (tf-t0)*(tau+1)/2+t0;
256         tLGR = t(1:end-1);
257 
258         %-------------------------------------------------------------------------%
259         % Get planer coordinates
260         %-------------------------------------------------------------------------%
261         x_transfer = r.*cos(theta);
262         y_transfer = r.*sin(theta);
263         theta_orbit = linspace(0,2*pi,100);
264         x_orbit_1 = r0*cos(theta_orbit);
265         y_orbit_1 = r0*sin(theta_orbit);
266         x_orbit_2 = rf*cos(theta_orbit);
267         y_orbit_2 = rf*sin(theta_orbit);
268 
269         %---------------%
270         % Plot Results
271         %---------------%
272         if close_figs
273             close all
274         end
275         if maximize_mass
276             str_obj = 'mf';
277             str_obj1 = 'maximized ';
278         else
279             str_obj = 'tf';
280             str_obj1 = 'minimized ';
281         end
282 
283         fig_cntrl = figure;
284         h1 = subplot(2,1,1); hold on;
285         plot(tLGR,beta,'-o');
286         ylabel('$\beta(t)$','Interpreter','LaTeX');
287         xlabel('$t$','Interpreter','LaTeX')
288         set(gca,'FontName','Times','FontSize',14);
289         set(gcf,'color','white')
290         title(['Thrust angle that ' str_obj1 str_obj])
291         grid minor;
292 
293         h2 = subplot(2,1,2);
294         plot(tLGR,u3,'-o'); hold on;
295         set(gca,'FontName','Times','FontSize',14);
296         set(gcf,'color','white')
297         ylabel('$T(t)$','Interpreter','LaTeX');
298         title(['Thrust magnitude that ' str_obj1 str_obj])
299         xlabel('$t$','Interpreter','LaTeX')
300         grid minor
301         linkaxes([h1 h2],'x')
302 
303         fig_states = figure; hold on; grid minor
304         plot(t,r,'-o'); plot(t,theta,'-o'); plot(t,vr,'-o');
```

```
305     plot(t,vtheta,'-o'); plot(t,m,'-o');
306     xlabel('t','Interpreter','LaTeX')
307     legend('r(t)','$\theta(t)$','$v_r(t)$','$v_\theta(t)$','m(t)','Interpreter','LaTeX')
308     set(gcf,'color','white')
309     set(gca,'FontName','Times','fontsize',14)
310     title(['States of trajectory that ' str_obj1 str_obj])
311
312     fig_orbit = figure;
313     polarplot(theta_orbit,1*ones(1,numel(theta_orbit)))
314     hold on;
315     polarplot(theta_orbit,1.5*ones(1,numel(theta_orbit)))
316     polarplot(theta,r);
317     set(gca,'FontName','Times','fontsize',14)
318     set(gcf,'color','white')
319     legend('Initial Orbit','Final Orbit','Orbit Transfer')
320     title('Orbit transfer overview')
321
322     if save_figs
323         if maximize_mass
324             str_obj = 'mf';
325         else
326             str_obj = 'tf';
327         end
328         plot_str = sprintf('_N%d_K%d_C%d_',N,numIntervals,ncontrols);
329         str_list = {'control','states','orbit'};
330         for idx = 1:numel(str_list)
331             name = [str_list{idx} plot_str str_obj];
332             print(figure(idx),name,'-depsc')
333         end
334     end
335     if path_constraint
336         fig_path = figure;
337         subplot(1,2,1);
338         plot(tLGR,u1,'-o');
339         xl = xlabel('$t$','Interpreter','LaTeX');
340         yl = ylabel('$u_1(t)$','Interpreter','LaTeX');
341         set(xl,'FontSize',14);
342         set(yl,'FontSize',14);
343         set(gca,'FontName','Times','FontSize',14);
344         grid on;
345         subplot(1,2,2);
346         plot(tLGR,u2,'-o');
347         xl = xlabel('$t$','Interpreter','LaTeX');
348         yl = ylabel('$u_2(t)$','Interpreter','LaTeX');
349         set(xl,'FontSize',14);
350         set(yl,'FontSize',14);
351         set(gca,'FontName','Times','FontSize',14);
352         set(gcf,'color','white')
353         grid on;
354         suptitle(['Path constriant control that ' str_obj1 str_obj])
355
356         if save_figs
357             str_path = ['path' plot_str str_obj];
358             print(fig_path,str_path,'-depsc')
359         end
360     end
361     %-----------------%
362     % save results %
363     %-----------------%
364     degree(count,1) = N;
365     intervals(count,1) = numIntervals;
366     iterations(count,1) = info.iter;
367     cpu_time(count,1) = info.cpu;
368     final_time(count,1) = tf;
369     final_mass(count,1) = m(end);
370     solved_info(count,1) = info.status;
371     count = count + 1;
372     end
```

```
373 end
374 table_mat = horzcat(degree,intervals,iterations,cpu_time,final_time,final_mass,solved_info);
375 T = array2table(table_mat);
376 VarNames = {'Degree','Intervals','Iterations','CPU Time','tf','mf','Solved_Status'};
377 T.Properties.VariableNames = VarNames;
378 if create_latex_table
379     str_table = sprintf('table_C%d_',ncontrols);
380     table2latex(T,[str_table str_obj])
381 end
382 nlistvec = repmat(n_list',1,numel(k_list));
383 klistvec = repmat(k_list,numel(n_list),1);
384 runtime = reshape(cpu_time,[],numn);
385 runtime = runtime';
386 if maximize_mass
387     mfmat = reshape(final_mass,[],numn);
388     objmat = mfmat';
389
390 else
391     tfmat = reshape(final_time,[],numn);
392     objmat = tfmat';
393 end
394 if plot_comparisons
395     fig_obj = figure; hold on;
396     for nidx = 1:numn
397         plot(k_list,objmat(nidx,:),'-o')
398         str_legend{nidx} = sprintf('Degree %d',n_list(nidx));
399     end
400     xlabel('Intervals')
401     ylabel(['Objective function ' str_obj])
402     set(gcf,'color','white')
403     legend(str_legend)
404     set(gca,'fontweight','bold','fontsize',14, 'XMinorGrid','on','YMinorGrid','on')
405     title('Objective Function Performance')
406
407
408     fig_runtime = figure; hold on;
409     for nidx = 1:numn
410         plot(k_list,runtime(nidx,:),'-o')
411         str_legend{nidx} = sprintf('Degree %d',n_list(nidx));
412     end
413     legend(str_legend)
414     set(gcf,'color','white')
415     set(gca,'fontweight','bold','fontsize',14, 'XMinorGrid','on','YMinorGrid','on')
416     xlabel('Intervals')
417     ylabel('Execution Time (s)')
418     title('Execution Time Performance')
419
420     if save_figs
421         str_control = sprintf('_c%d_',ncontrols);
422         str_path = ['runtime' str_control str_obj];
423         print(fig_runtime,str_path,'-depsc')
424
425         str_path = ['obj' str_control str_obj];
426         print(fig_obj,str_path,'-depsc')
427
428     end
429 end
430
431 % set(0,'DefaultFigureWindowStyle','docked')
432 % arrayfun(@(x) set( x,'windowstyle', 'docked'),findall(groot,'type','figure')) ;
```

Listing 2: orbitTransferFun.m

```
1 function C = orbitTransferFun(z)
2
3 %-------------------------------------------------------------------------%
4 % 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(:,2);
vr = stateLGR(:,3);
vtheta = stateLGR(:,4);
m = stateLGR(:,5);
if path_constraint
    u1 = control(:,1);
    u2 = control(:,2);
    u3 = control(:,3);
else
```

```
73      u1 = control(:,1);
74      u2 = 0;
75      u3 = control(:,2);
76  end
77
78  %-------------------------------------------------------------------------%
79  % Compute the right-hand side of the differential equations at %
80  % the N LGR points. Each component of the right-hand side is %
81  % stored as a column vector of length N, that is each column has %
82  % the form %
83  % [ f_i(x_1,u_1,t_1) ] %
84  % [ f_i(x_2,u_2,t_2) ] %
85  % . %
86  % . %
87  % . %
88  % [ f_i(x_N,u_N,t_N) ] %
89  % where "i" is the right-hand side of the ith component of the %
90  % vector field f. It is noted that in MATLABB the calculation of %
91  % the right-hand side is vectorized. %
92  %-------------------------------------------------------------------------%
93  rdot = vr;
94  thetadot = vtheta./r;
95  mdot = -u3/ve;
96
97  a = u3./m;
98  if path_constraint
99  % vrdot = vtheta.^2./r - mu./r.^2 + u3.*u1./m;
100 % vthetadot = -vtheta.*vr./r + u3.*u2./m;
101     vrdot = vtheta.^2./r - mu./r.^2 + a.*u1;
102     vthetadot = -vtheta.*vr./r + a.*u2;
103 else
104     vrdot = vtheta.^2./r - mu./r.^2 + u3.*sin(u1)./m;
105     vthetadot = -vtheta.*vr./r + u3.*cos(u1)./m;
106 end
107
108 diffeqRHS = [rdot, thetadot, vrdot, vthetadot, mdot];
109
110 %-------------------------------------------------------------------------%
111 % Compute the left-hand side of the defect constraints, recalling %
112 % that the left-hand side is computed using the state at the LGR %
113 % points PLUS the final point. %
114 %-------------------------------------------------------------------------%
115 diffeqLHS = D*statePlusEnd;
116
117 %-------------------------------------------------------------------------%
118 % Construct the defect constraints at the N LGR points. %
119 % Remember that the right-hand side needs to be scaled by the %
120 % factor (tf-t0)/2 because the rate of change of the state is %
121 % being taken with respect to $\tau\in[-1,+1]$. Thus, we have %
122 % $dt/t\dau=(tf-t0)/2$. %
123 %-------------------------------------------------------------------------%
124 defects = diffeqLHS-(tf-t0)*diffeqRHS/2;
125
126 %-------------------------------------------------------------------------%
127 % Construct the path constraints at the N LGR points. %
128 % Reshape the path contraints into a column vector. %
129 %-------------------------------------------------------------------------%
130 if path_constraint
131     paths = u1.^2 + u2.^2;
132     paths = reshape(paths,N*npaths,1);
133 else
134     paths = [];
135 end
136 % paths = u1.^2+u2.^2;
137
138
139
140 %-------------------------------------------------------------------------%
```

```
141 % Reshape the defect contraints into a column vector. %
142 %-------------------------------------------------------------------------%
143 defects = reshape(defects,N*nstates,1);
144
145
146 %-------------------------------------------------------------------------%
147 % Construct the objective function plus constraint vector. %
148 %-------------------------------------------------------------------------%
149 if maximize_mass
150     m = statePlusEnd(:,5);
151     J = -m(end);
152 else
153     J = tf;
154 end
155 C = [J;defects;paths];
```

Listing 3: orbitTransferObj.m

```
1  function obj = orbitTransferObj(z)
2  % Computes the objective function of the problem
3
4  global psStuff nstates ncontrols maximize_mass
5
6  %-------------------------------------------------------------------------%
7  % Radau pseudospectral method quantities required: %
8  % - Differentiation matrix (psStuff.D) %
9  % - Legendre-Gauss-Radau weights (psStuff.w) %
10 % - Legendre-Gauss-Radau points (psStuff.tau) %
11 %-------------------------------------------------------------------------%
12 D = psStuff.D; tau = psStuff.tau; w = psStuff.w;
13
14 %-------------------------------------------------------------------------%
15 % Decompose the NLP decision vector into pieces containing %
16 % - the state %
17 % - the control %
18 % - the initial time %
19 % - the final time %
20 %-------------------------------------------------------------------------%
21 N = length(tau)-1;
22 stateIndices = 1:nstates*(N+1);
23 controlIndices = (nstates*(N+1)+1):(nstates*(N+1)+ncontrols*N);
24 t0Index = controlIndices(end)+1;
25 tfIndex = t0Index+1;
26 stateVector = z(stateIndices);
27 tf = z(tfIndex);
28
29 %-------------------------------------------------------------------------%
30 % Reshape the state and control parts of the NLP decision vector %
31 % to matrices of sizes (N+1) by nstates and (N+1) by ncontrols, %
32 % respectively. The state is approximated at the N LGR points %
33 % plus the final point. Thus, each column of the state vector is %
34 % length N+1. The LEFT-HAND SIDE of the defect constraints, D*X, %
35 % uses the state at all of the points (N LGR points plus final %
36 % point). The RIGHT-HAND SIDE of the defect constraints, %
37 % (tf-t0)F/2, uses the state and control at only the LGR points. %
38 % Thus, it is necessary to extract the state approximations at %
39 % only the N LGR points. Finally, in the Radau pseudospectral %
40 % method, the control is approximated at only the N LGR points. %
41 %-------------------------------------------------------------------------%
42 statePlusEnd = reshape(stateVector,N+1,nstates);
43
44 % Cost Function
45 % minizing time or maximizing mass
46 if maximize_mass
47     m = statePlusEnd(:,5);
48     J = -m(end);
49 else
```

```
50      J = tf;
51  end
52
53  obj = J;
54
55  end
```

Listing 4: orbitTransferCon.m

```
1   function constraints = orbitTransferCon(Z)
2   % computes the constraints
3
4   output = orbitTransferFun(Z);
5   constraints = output;
6
7   end
```

Listing 5: orbitTransferGrd.m

```
1   function grd = orbitTransferGrd(Z)
2   % computes the gradient
3
4   output = orbitTransferObj_Jac(Z);
5   grd = output;
6
7   end
```

Listing 6: orbitTransferJac.m

```
1   function jac = orbitTransferJac(Z)
2   % computes the jacobian
3
4   [jac,~] = orbitTransferFun_Jac(Z);
5
6   end
```

Listing 7: orbitTransferJacPat.m

```
1   function jacpat = orbitTransferJacPat(S_jac)
2   % computes the jacobian structure
3
4   jacpat = S_jac;
5
6   end
```