9-2016

Takagi-Sugeno Fuzzy Model Based Discrete Time Model Predictive Control for a Hypersonic Re-Entry Vehicle

Ben Margolis
Santa Clara University

Takagi-Sugeno Fuzzy Model Based Discrete Time Model Predictive Control for a Hypersonic Re-Entry Vehicle

Ben Margolis

September 2016

Santa Clara University
School of Engineering
Department of Applied Mathematics

Thesis submitted at Santa Clara University in partial fullment of the requirements for the degree of Master of Science in Applied Mathematics

Mohammad A. Ayoubi, Thesis Adviser

Aaron Melman, Department Chair

Contents

Abstract

In this thesis, we present a control algorithm for a hypersonic re-entry vehicle during a Martian aerocapture maneuver. The proposed algorithm utilizes a discrete-time model predictive control technique with a Takagi-Sugeno fuzzy model of the vehicle to control the re-entry vehicle along an arbitrary trajectory using bank angle modulation. Simulations using model parameters and initial conditions from a Martian aerocapture mission demonstrate the stability, performance, and robustness of the proposed controller.

1. Introduction

1.1. Motivation

Aerocapture is a maneuver to insert a vehicle into orbit around a planet by entering the planet's atmosphere and using aerodynamics to approach to the desired orbit plane at an appropriate velocity. By using aerodynamic drag the aerocapture maneuver requires significantly less propellant to achieve the same orbit compared to an all-propulsive method of orbit capture. This fuel savings allows for some combination of more frequent scientific space missions and missions with larger scientific payloads.

As a specific type of aeroassisted orbital maneuver, a concept first proposed by London in in 1961, aerocapture has a wealth of applicable guidance, navigation, and control (GN&C) methods[Wal85, Mie96]. Much aerocapture specific work has been motivated by a Mars surface sample return mission. The first proposal for the aerocapture on this mission used a GN&C method based on the one used in the Apollo mission [Cru79]. More recent aerocapture work has also incorporated the Apollo terminal logic, like the calculus-of-variations derived Terminal Point Controller (TPC) method [RQ98]. Another controller well-studied in the literature is the analytic predictor-corrector (APC) method, later called the Hybrid Predictor-corrector Aerocapture Scheme (HYPAS), which has demonstrated efficient code, minimal preflight effort, and adaptability to different missions[CG85, MRFP00, MQ03]. Rousseau et al. have compared these two methods as well as energy control (EC) and numeric predictor corrector (NPC) methods[RPG$^+$02]. The study found that APC demonstrated the highest accuracy, TPC demonstrated the strongest robustness to dispersion, and that the EC struck a balance between the two. The mission-focused motivation for aeroassisted orbital transfer generally, and aerocapture study specifically, required the development of thorough evaluation methods and simulation protocols[PB93, FPRS00]. Most of the aerocapture methods use bank angle modulation to steer the vehicle through the desired trajectory, although some proposals have suggested the use of angle-of-attack modulation to increase control authority[Que04, JW04].

Many other studies and iterative improvements have been presented for the APC and TPC methods, such as tracking drag force itself rather than bank angle[CLWM08]. Studies on the optimality of aerocapture guidance algorithms have also been presented, for example by Lu et al[LCTM15].

In this paper, we propose using a Takagi-Sugeno fuzzy model (TSFM) with Model Predictive Control (MPC) technique for controlling a re-entry vehicle along a given trajectory during aerocapture. MPC is a method of optimal control in which a time-history input profile is numerically generated to optimize a chosen metric for a finite future time-frame called a receding horizon. For a linear time-invariant discrete-time (LTI-DT) system, the control optimization can be formulated as a quadratic programming (QP) problem[Wan09]. However, for a general nonlinear system a QP problem formulation cannot be guaranteed. To address this, a number of methods of discrete-time TSFM predictive control methods have been developed [ZFL07, MBAV04, RMBV99, NP99, SYLK09, KEE10]. Here, we suggest adapting the QP based formulation for LTI-DT systems using a TSFM such as presented by Mollov and Oviedo[MBAV04, OVW06]. Ultimately, the proposed controller uses linearized dynamics along the given trajectory similar to Lu's receding horizon control of a time-varying linearized model[Lu99]. Since this controller uses online QP to determine the control input, it inherently satisfies constraints on the system inputs, states, and outputs. By using a TSFM, we are able to motivate the linearization along the trajectory and the transformation of continuous-time dynamics to discrete time.

This paper is organized as follows. First we describe the aerocapture trajectory and the mathematical model of vehicle entering the atmosphere and subject to control. Then we describe the TSFM construction process and show the TSFM model for the aerocapture dynamics. Next, we describe the discrete-time TSFM based MPC optimization problem. Then we present initial simulation results for the case of Martian aerocapture. Finally, we conclude by discussing implications and future work.

1.2. Aerocapture Trajectory

The aerocapture maneuver consists of several key events, as illustrated in Figure 1.1:

1. Entry targeting - Following an elliptical approach to the target planet, the vehicle performs final trajectory adjustment so the atmospheric entry angle is within the acceptable entry corridor.

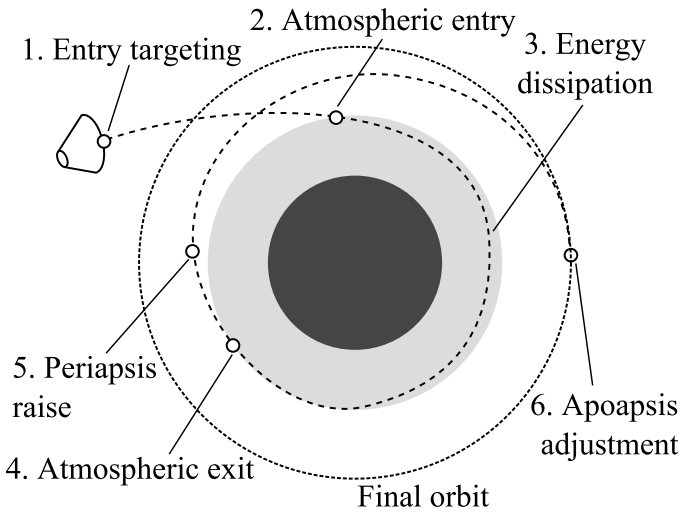

Figure 1.1.: Overview of Aerocapture Maneuver

2. Atmospheric entry interface - The vehicle enters the atmosphere and begins bank angle modulation to steer through the desired trajectory.
3. Energy dissipation - The vehicle travels through the atmosphere and without leaving the atmosphere, decelerating to achieve the desired exit velocity. Steering through a lower altitude trajectory increases drag and energy dissipation (velocity reduction), while steering through a higher altitude reduces drag and energy dissipation. Throughout this phase, the vehicle also implements a lateral control system to approach the desired orbit plane.
4. Controlled atmosphere exit - As the vehicle achieves the desired exit velocity, it begins the exit out of atmosphere. As the vehicle altitude increases, control authority through aerodynamic forces decreases until complete exit of the atmosphere.
5. Periapsis raise - At the periapsis radius, the vehicle performs a thrust maneuver to impart the appropriate change in velocity to achieve orbit and prevent return to the atmosphere.
6. Optional apoapsis adjustment maneuver (circularization) - An optional final maneuver may be performed for final adjustment of the orbit, typically a circularization thrust performed at apoapsis.

2. Mathematical Model

In this chapter, we describe the mathematical model for the hypersonic re-entry vehicle.

2.1. Base derivation

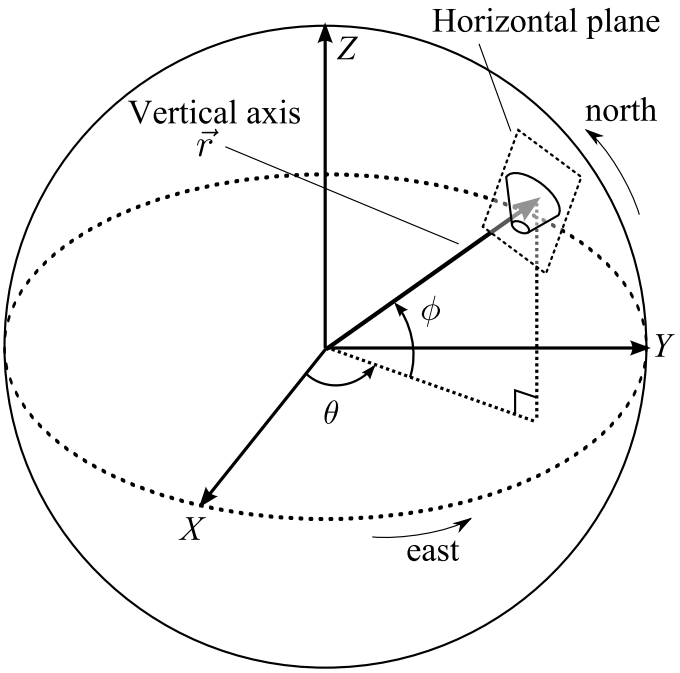

Figure 2.1.: Vehicle Position Relative to Planet

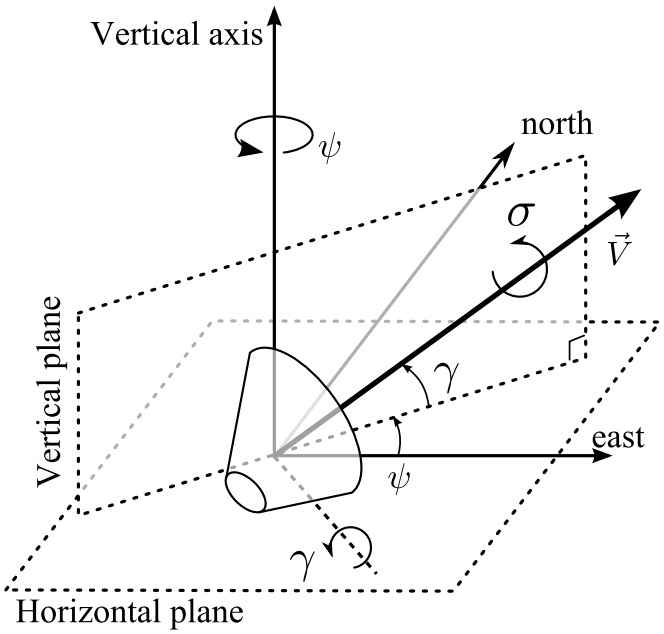

Figure 2.2.: Vehicle Orientation Relative to Planet

The three-dimensional equations of motion for re-entry vehicle are given by[VBC80]

$$\begin{aligned}
\frac{dr}{dt} &= V\ \sin\gamma \\
\frac{d\theta}{dt} &= \frac{V\ \cos\gamma\ \cos\psi}{r\ \cos\phi} \\
\frac{d\phi}{dt} &= \frac{V\ \cos\gamma\ \sin\psi}{r} \\
\frac{dV}{dt} &= -\frac{\rho\left(r\right)\ S\,C_D\left(\alpha\right)\ V^2}{2m} - g\left(r\right)\ \sin\gamma \\
\frac{d\gamma}{dt} &= \frac{1}{V}\left[\frac{\rho\left(r\right)\ S\,C_L\left(\alpha\right)\ V^2}{2m}\ \cos\sigma - \left(g\left(r\right) - \frac{V^2}{r}\right)\cos\gamma\right] \\
\frac{d\psi}{dt} &= \frac{1}{V}\left[\frac{\rho\left(r\right)\ S\,C_L\left(\alpha\right)\ V^2}{2m\ \cos\gamma}\sin\sigma - \frac{V^2}{r}\cos\gamma\ \cos\psi\ \tan\phi\right]
\end{aligned} \tag{2.1}$$

where t is time, r is the the distance from the center of the planet to the vehicle, θ and ϕ are the vehicle's longitude and latitude, V is the speed of the vehicle, γ is the pitch of the vehicle measured positive upwards from horizontal, ψ is the heading angle about the vertical axis measured from due East, ρ is the atmospheric density, S is the aerodynamic reference area, C_L and C_D are the lift and drag coefficients, g is the gravitational acceleration, m is the vehicle mass, α is the angle of attack, and σ is the bank or roll angle.

Visualizations for the quantities used in Equation 2.1 can be found in Figures 2.1 and 2.2. Note that r is measured along the vertical axis, and the heading ψ and pitch γ angles are determined by the direction of the velocity, $\vec{V}$. The aerodynamic lift points away from vertical by the bank angle σ rotated around $\vec{V}$. Aerodynamic drag always resists the velocity. The exact functional relationship between C_L and C_D are determined by the geometries of the vehicle.

2.1.1. Atmospheric density

The atmospheric density is calculated from the radially dependent differential equation

$$\begin{aligned} \frac{d\rho}{\rho} &= -\left[\frac{g\left(r\right)\ M}{R^{\star}T} + \frac{1}{T}\frac{dT}{dr}\right] dr \\ &= \beta\left(r\right)\ dr \end{aligned} \tag{2.2}$$

where g is the acceleration due to gravity, M is the mean molecular weight of the atmosphere, $R^{\star}$ is the universal gas constant, T is the absolute temperature, and $\beta\left(r\right) = \frac{gM}{R^{\star}T} + \frac{1}{T}\frac{dT}{dr}$ is a parameter called the scale height, which can itself be accurately approximated by a linear interpolation. The solution to Equation 2.2 is then approximated by exponential of the form

$$\rho\left(r\right) = \rho_0 e^{-\beta(r-r_0)} \tag{2.3}$$

2.1.2. Acceleration due to gravity

The acceleration due to gravity is governed by

$$g\left(r\right) = \frac{\mu}{r^2} \tag{2.4}$$

where μ is the gravitational parameter of the planet.

2.2. System Model

2.2.1. Vertical Plane

To simplify the control problem, we will consider the three states in the vertical plane, V, γ, and r. The governing equations, when substituting in Eqs. (2.2), (2.3), and (2.4) and simplifying, are given by

$$\begin{aligned}
\frac{dV}{dt} &= -\frac{\rho_0 e^{-\beta(r-r_0)}\, S\, C_D\left(\alpha\right)\, V^2}{2m} - \frac{\mu}{r^2}\, \sin\gamma \\
\frac{d\gamma}{dt} &= \frac{\rho_0 e^{-\beta(r-r_0)}\, S\, C_L\left(\alpha\right)\, \cos\sigma\, V}{2m} - \left(\frac{\mu}{r^2\, V} - \frac{V}{r}\right) \cos\gamma \\
\frac{dr}{dt} &= V\, \sin\gamma
\end{aligned} \tag{2.5}$$

2.2.2. Augmentation

Finally, we will augment the system to represent it in control-affine form, with the commanded bank angle rate σ' as the new input. This canonical form for control systems will also simplify the fuzzy modeling as well as describe the bank angle integrator dynamics. The state space form is given by

$$\frac{d}{dt}\begin{bmatrix} V \\ \gamma \\ r \\ \sigma \end{bmatrix} = \begin{bmatrix} -\frac{\rho_0 e^{-\beta(r-r_0)}\, S\, C_D(\alpha)\, V^2}{2m} - \frac{\mu}{r^2}\, \sin\gamma \\ \frac{\rho_0 e^{-\beta(r-r_0)}\, S\, C_L(\alpha)\, \cos\sigma\, V}{2m} - \left(\frac{\mu}{r^2\, V} - \frac{V}{r}\right) \cos\gamma \\ V\, \sin\gamma \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \sigma' \tag{2.6}$$

In the next section, these governing equations will be modeled using a Takagi-Sugeno Fuzzy Model.

3. Takagi-Sugeno Fuzzy Model

3.1. Overview

The main idea behind Takgai-Sugeno Fuzzy Model (TSFM) is to represent a nonlinear system as the convex, but non-linear, combination of linear subsystems. The TSFM is constructed by choosing p appropriate parameters in the system called premise variables, approximating the nonlinear system around the selected values of the premise variables, building membership functions in the universe of discourse about each value of each premise variable, and creating r model rules corresponding to each point. Each of the r rules is of the form

Model Rule i:

$$\text{IF } z_1(t) \text{ is about } \mu_{i1}[z_1(t)], \ldots, z_p \text{ is about } \mu_{ip}[z_p(t)]$$

$$\text{THEN } \Sigma_i: \begin{cases} \dfrac{d}{dt}\mathbf{x}(t) = \mathbf{A}_i\,\mathbf{x}(t) + \mathbf{B}_i\,\mathbf{u}(t) \\ \mathbf{y}(t) = \mathbf{C}_i\,\mathbf{x}(t) \end{cases} \qquad (i = 1, \ldots, r) \tag{3.1}$$

The firing strength of each rule h_i can be determined using the T-norm product of the corresponding membership functions $\mu_{ij} \in [0, 1]$ such that

$$h_i[\mathbf{z}(\mathbf{x}(t))] = \prod_{j=1}^{p} \mu_{ij}[z_j(\mathbf{x}(t))] \tag{3.2}$$

The membership functions are constructed to be complementary so that $\sum h_i \left[\mathbf{x}\left(t\right)\right] = 1$. The overall TSFM is defined as

$$\Sigma_{TS}: \begin{cases} \dfrac{d}{dt}\mathbf{x}(t) = \displaystyle\sum_{i=1}^{r} h_i \left[\mathbf{z}\left(\mathbf{x}\left(t\right)\right)\right] \left[\mathbf{A}_i \, \mathbf{x}\left(t\right) + \mathbf{B}_i \, \mathbf{u}\left(t\right)\right] \\ \mathbf{y}\left(t\right) = \displaystyle\sum_{i=1}^{r} h_i \left[\mathbf{z}\left(\mathbf{x}\left(t\right)\right)\right] \left[\mathbf{C}_i \, \mathbf{x}\left(t\right)\right] \end{cases} \tag{3.3}$$

This is the standard form of TSFM. For more examples and details, we refer the interested reader to Tanaka and Wang[TW01].

3.2. Sector Nonlinearity Construction

An exact TSFM representation of the nonlinear system can be constructed using the method of sector non-linearity. In this method, each of the p nonlinear terms of the system dynamics are chosen as the premise variables. Sector non-linearity uses the upper and lower bounds on the premise variables to construct the membership function for each term, resulting in 2^p rules of the form described above. For the model shown in Eq. (2.6), the premise variables can be chosen as

$$\begin{aligned} z_1 &\triangleq -\frac{S\, C_D\, \rho_0 e^{-\beta(r-r_0)}\, V}{2m} \\ z_2 &\triangleq -\frac{\mu}{r^2}\frac{\sin\gamma}{\gamma} \\ z_3 &\triangleq \frac{S\, C_L\, \rho_0 e^{-\beta(r-r_0)}\, V\, \cos\sigma}{2m} - \left(\frac{\mu}{r^2 V^2} - \frac{1}{r}\right)\cos\gamma \\ z_4 &\triangleq \sin\gamma \end{aligned} \tag{3.4}$$

giving 16 rules of the form

Model Rule i:

IF $z_1(t)$ is about $\mu_{i1}[z_1(t)], \ldots, z_4$ is about $\mu_{i4}[z_4(t)]$

$$\text{THEN } \Sigma_i: \begin{cases} \dfrac{d}{dt}\begin{bmatrix} V \\ \gamma \\ r \\ \sigma \end{bmatrix} = \begin{bmatrix} z_1^i & z_2^i & 0 & 0 \\ z_3^i & 0 & 0 & 0 \\ z_4^i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V \\ \gamma \\ r \\ \sigma \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \sigma' \\ \mathbf{y} = \begin{bmatrix} V \\ \gamma \\ r \\ \sigma \end{bmatrix} \end{cases} \qquad (i = 1, \ldots, 16) \tag{3.5}$$

The values of interest for each of the z_j premise variables are the upper and lower bounds $\overline{z}_j$ and $\underline{z}_j$ from the system dynamics such that

$$z_j^i = \begin{cases} \underline{z}_j & \text{if } \left\lfloor \frac{i-1}{2^j} \right\rfloor \mod 2 = 0 \\ \\ \overline{z}_j & \text{if } \left\lfloor \frac{i-1}{2^j} \right\rfloor \mod 2 = 1 \end{cases}$$

The corresponding membership functions of each variable are of the form

$$\mu_{ij}\left(z_j\left(\mathbf{x}(t)\right)\right) = \begin{cases} \frac{\overline{z}_j - z_j}{\overline{z}_j - \underline{z}_j} & \text{if } \left\lfloor \frac{i-1}{2^j} \right\rfloor \mod 2 = 0 \\ \\ \frac{z_j - \underline{z}_j}{\overline{z}_j - \underline{z}_j} & \text{if } \left\lfloor \frac{i-1}{2^j} \right\rfloor \mod 2 = 1 \end{cases}$$

Each rule is simply a permutation of each premise variable's two possible values.

3.3. Time Discretization

In the next section, a discrete-time TSFM based MPC controller is described. In order to discretize our continuous time TSFM, we discretize the linear dynamics for each fuzzy

rule using a generalized bi-linear transform. The resulting rules have the form

$$\begin{aligned}&\text{Model Rule } i\text{:}\\&\quad\text{IF } z_1(t) \text{ is about } \mu_{i1}[z_1(t)], \ldots, z_p \text{ is about } \mu_{ip}[z_p(t)]\\&\text{THEN } \Sigma_i\text{:}\ \begin{cases}\mathbf{x}(k+1) = \mathbf{A}_{i,T_s}\mathbf{x}(k) + \mathbf{B}_{i,T_s}\mathbf{u}(k) & (i=1,\ldots,r)\\ \mathbf{y}(k) = \mathbf{C}_{i,T_s}\mathbf{x}(k)\end{cases}\end{aligned} \tag{3.6}$$

giving a discrete-time TSFM of the form

$$\Sigma_{TS}\text{:}\ \begin{cases}\dfrac{d}{dt}\mathbf{x}(t) = \displaystyle\sum_{i=1}^{r} h_i[\mathbf{z}(\mathbf{x}(t))][\mathbf{A}_{i,T_s}\mathbf{x}(t) + \mathbf{B}_{i,T_s}\mathbf{u}(t)]\\ \mathbf{y}(t) = \displaystyle\sum_{i=1}^{r} h_i[\mathbf{z}(\mathbf{x}(t))][\mathbf{C}_{i,T_s}\mathbf{x}(t)]\end{cases} \tag{3.7}$$

where the membership functions μ_{ij} and firing strength h_i are the same corresponding rule of the continuous-time TSFM. The matrices $\mathbf{A}_{i,T_s}$, $\mathbf{B}_{i,T_s}$, and $\mathbf{C}_{i,T_s}$ are the time discretized state, input, and output matrices of each of the TSFM subsystems using a general bi-linear tansformation with time interval T_s

$$\begin{aligned}\mathbf{A}_{i,T_s} &= \boldsymbol{I} + T_s\mathbf{A}_i\\ \mathbf{B}_{i,T_s} &= T_s\mathbf{B}_i\\ \mathbf{C}_{i,T_s} &= \mathbf{C}_i\end{aligned} \tag{3.8}$$

4. Discrete-Time Model Predictive Control

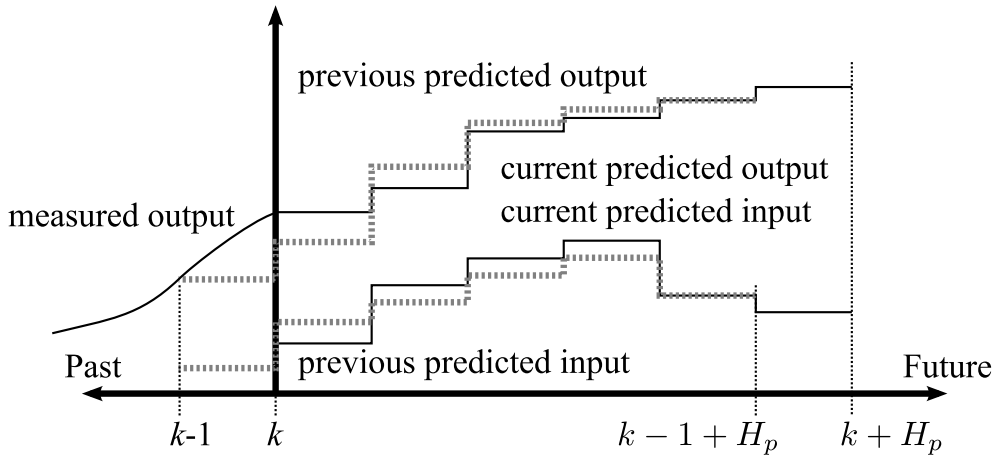

Figure 4.1.: Illustration of the Model Predictive Control Concept

Model predictive control (MPC) is the constrained optimization of a control trajectory to minimize a cost function based on predicted system dynamics from the current state over a finite time span known as the prediction horizon (denoted by H_p time intervals for a discrete time system.) Typically, only the first control interval of the optimal control trajectory is applied to the system. The control input for the next interval comes from repeating the optimization process. For each optimization process, the prediction horizon recedes to the current time and initial conditions for the predicted dynamics come from the current state of the actual system dynamics. We also note that the duration of the control trajectory subject to optimization (called the control horizon, denoted by H_c time intervals for a discrete time system) is not necessarily equal to the prediction horizon. A schematic illustrating the process is shown in Figure 4.1.

4.1. Model Prediction

For linear time-invariant, discrete-time models, the MPC optimization problem can be formulated as constrained quadratic programming problem by formulating the prediction of the system dynamics using a prediction matrix equation. To formulate the prediction, the system is augmented to take as input a control increment yielding the augmented system

$$\begin{aligned} \mathbf{x}_a(k+1) &= \overbrace{\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}}^{\mathbf{A}_a} \overbrace{\begin{bmatrix} \mathbf{x}(k) \\ \mathbf{u}(k-1) \end{bmatrix}}^{\mathbf{x}_a(k)} + \overbrace{\begin{bmatrix} \mathbf{B} \\ \mathbf{I} \end{bmatrix}}^{\mathbf{B}_a} \Delta\mathbf{u}(k) \\ \mathbf{y}_a(k) &= \underbrace{\begin{bmatrix} \mathbf{C} & \mathbf{0} \end{bmatrix}}_{\mathbf{C}_a} \mathbf{x}_a(k) \end{aligned} \tag{4.1}$$

where the control increment is defined as $\Delta\mathbf{u}(k) = \mathbf{u}(k) - \mathbf{u}(k-1)$. Then the output prediction vector over the prediction horizon of H_p time intervals

$$Y = \begin{bmatrix} \mathbf{y}^T(k+1 \mid k) & \mathbf{y}^T(k+2 \mid k) & \cdots & \mathbf{y}^T(k+H_p \mid k) \end{bmatrix}^T$$

due to the input increment vector over applied over the control horizon of H_c intervals

$$\Delta U = \begin{bmatrix} \Delta\mathbf{u}^T(k) & \Delta\mathbf{u}^T(k+1) & \cdots & \Delta\mathbf{u}^T(k+H_c-1) \end{bmatrix}^T$$

is given by

$$Y = \mathbf{R}_x \mathbf{x}_a(k) + \mathbf{R}_u \Delta U$$

where $\mathbf{R}_x$ is the free response prediction matrix given by

$$\mathbf{R}_x = \begin{bmatrix} \mathbf{C}_a\mathbf{A}_a \\ \mathbf{C}_a\mathbf{A}_a^2 \\ \vdots \\ \mathbf{C}_a\mathbf{A}_a^{H_p} \end{bmatrix}$$

and $\mathbf{R}_u$ is the forced response prediction matrix given by

$$\mathbf{R}_u = \begin{bmatrix} \mathbf{C}_a\mathbf{B}_a & 0 & 0 & \cdots & 0 \\ \mathbf{C}_a\mathbf{A}_a\mathbf{B}_a & \mathbf{C}_a\mathbf{B}_a & 0 & \cdots & 0 \\ \mathbf{C}_a\mathbf{A}_a^2\mathbf{B}_a & \mathbf{C}_a\mathbf{A}_a\mathbf{B}_a & \mathbf{C}_a\mathbf{B}_a & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_a\mathbf{A}_a^{H_p-1}\mathbf{B}_a & \mathbf{C}_a\mathbf{A}_a^{H_p-2}\mathbf{B}_a & \mathbf{C}_a\mathbf{A}_a^{H_p-3}\mathbf{B}_a & \cdots & \mathbf{C}_a\mathbf{A}_a^{H_p-H_c}\mathbf{B}_a \end{bmatrix}$$

The states of a nonlinear dynamical system can similarly be calculated using the total TSFM system from Eq. (3.3) at each time instant following the reference trajectory $\vec{r}(k)$. Then the block-index definition for the two prediction matrices are given by

$$(\mathbf{R}_x)_i = \mathbf{C}_a(k+i) \prod_{n=i-1}^{1} \mathbf{A}_a(k+n)$$
$$\text{for } i = 1, \ldots, H_p$$
$$(\mathbf{R}_u)_{ij} = \mathbf{C}_a(k+i) \left(\prod_{n=i-j}^{j-1} \mathbf{A}_a(k+n) \right) \mathbf{B}_a(k+j-1)$$
$$\text{for } j = 1, \ldots, H_c \leq i = 1, \ldots, H_p$$

Here, we will determine the fuzzy-basis of Eq. (3.2) using the value of the premise variables evaluated at the reference trajectory to determine the state and input matrices, so the state, input, and output matrices of the total TSFM at any time interval k used for constructing the prediction matrices are given by

$$\mathbf{A}(k) = \sum_{i=1}^{r} h_i[z(\mathbf{r}_y(k))]\mathbf{A}_{i,T_s}$$
$$\mathbf{B}(k) = \sum_{i=1}^{r} h_i[z(\mathbf{r}_y(k))]\mathbf{B}_{i,T_s}$$
$$\mathbf{C}(k) = \sum_{i=1}^{r} h_i[z(\mathbf{r}_y(k))]\mathbf{C}_{i,T_s}$$

where $\mathbf{A}_{i,T_s}$, $\mathbf{B}_{i,T_s}$, and $\mathbf{C}_{i,T_s}$ is the time discretized state, input, and output matrices of each of the TSFM subsystems using a general bi-linear tansformation with time interval T_s described by Eq. (3.8). We note that in our particular model, the input and output matrices for each TSFM subsystem are the same so $\mathbf{B}_a(k)$ and $\mathbf{C}_a(k)$ are constant, but

for the sake of generality keep the time index in the sequel.

4.2. Control as Optimization Problem

Then a quadratic cost function of the form

$$J = \sum_{k=0}^{H_p} \begin{array}{l} \left(\mathbf{y}_a(k) - \mathbf{r}_y(k)\right)^T \mathbf{P} \left(\mathbf{y}_a(k) - \mathbf{r}_y(k)\right) + \Delta\mathbf{y}_a^T(k)\, \Delta\mathbf{P} \Delta\mathbf{y}_a(k) \\ + \left(\mathbf{u}(k) - \mathbf{r}_u(k)\right)^T \mathbf{Q} \left(\mathbf{u}(k) - \mathbf{r}_u(k)\right) + \Delta\mathbf{u}^T(k)\, \Delta\mathbf{Q} \Delta\mathbf{u}(k) \end{array}$$

with state and control references $\mathbf{r}_y(k)$ and $\mathbf{r}_u(k)$, weighting matrices $\mathbf{P}$, $\mathbf{Q}$, $\Delta\mathbf{P}$, and $\Delta\mathbf{Q}$ can be expressed as the quadratic

$$J = \frac{1}{2} \Delta U^T \mathbf{H}\, \Delta U + \mathbf{f}\, \Delta U$$

where the hessian $\mathbf{H}$ and gradient $\mathbf{f}$ are given by

$$\begin{aligned} \mathbf{H} = 2 \Big\{ & \mathbf{R}_u^T \cdot \bar{\mathbf{P}} \cdot \mathbf{R}_u + \left(\mathbf{R}_u - \mathbf{R}_{u2}\right)^T \cdot \Delta\bar{\mathbf{P}} \cdot \left(\mathbf{R}_u - \mathbf{R}_{u2}\right) \\ & + \mathbf{I}_{\Delta u}^T \cdot \bar{\mathbf{Q}} \cdot \mathbf{I}_{\Delta u} + \Delta\bar{\mathbf{Q}} \Big\} \\ \mathbf{f} = 2 \Big\{ & \left(\mathbf{R}_x \cdot \bar{\mathbf{A}}(k) \cdot \mathbf{x}_a(k) - R_y\right)^T \cdot \bar{\mathbf{P}} \cdot \mathbf{R}_u \\ & + \left(\mathbf{I}_u \cdot \mathbf{u}(k) - R_u\right)^T \bar{\mathbf{Q}} \mathbf{I}_{\Delta u} \\ & + \left(\mathbf{R}_x \cdot \bar{\mathbf{A}} \cdot \mathbf{x}_a(k) - \mathbf{R}_x \cdot \mathbf{x}_a(k)\right)^T \cdot \Delta\bar{\mathbf{P}} \left(\mathbf{R}_u - \mathbf{R}_{u2}\right) \Big\} \end{aligned}$$

where R_y and R_u are the reference vectors

$$\begin{aligned} R_y &= \begin{bmatrix} \mathbf{r}_y^T(k+1) & \mathbf{r}_y^T(k+2) & \cdots & \mathbf{r}_y^T(k+H_p) \end{bmatrix}^T \\ R_u &= \begin{bmatrix} \mathbf{r}_u^T(k+1) & \mathbf{r}_u^T(k+2) & \cdots & \mathbf{r}_u^T(k+H_p) \end{bmatrix}^T \end{aligned}$$

and $\bar{\mathbf{P}}$, $\bar{\mathbf{Q}}$, $\Delta\bar{\mathbf{P}}$, and $\Delta\bar{\mathbf{Q}}$ is the block-diagonal cost matrix composed of repetitions of the weighting matrices $\mathbf{P}$, $\mathbf{Q}$, $\Delta\mathbf{P}$, and $\Delta\mathbf{Q}$. The matrix $\mathbf{R}_{u2}$ is a shifted forced response prediction matrix. The remaining terms will be described along with the constraints.

Constraints for the optimization problem can be formulated as follows:

1. Input amplitude constraints come from the integration of $\Delta\mathbf{u}$ and can be expressed

as

$$\begin{bmatrix} -\mathbf{I}_{\Delta u} \\ \mathbf{I}_{\Delta u} \end{bmatrix} \Delta U < \begin{bmatrix} \mathbf{I}_u \left(-\mathbf{u}_{\min} + \mathbf{u}\left(k-1\right)\right) \\ \mathbf{I}_u \left(\mathbf{u}_{\max} - \mathbf{u}\left(k-1\right)\right) \end{bmatrix}$$

where $\mathbf{I}_u$ is a block column vector composed of $m \times m$ identity matrices (generally noted by $\mathbf{I}_m$). $\mathbf{I}_{\Delta u}$ is a block lower-diagonal matrix composed of $\mathbf{I}_m$'s.

2. Input rate constraints simply require each element in ΔU to be less than or greater than the min or max constraint, expressed in matrix form as

$$\begin{bmatrix} -\mathbf{I}_{H_c m} \\ \mathbf{I}_{H_c m} \end{bmatrix} \Delta U < \begin{bmatrix} -\mathbf{I}_{H_c m} \Delta \mathbf{u}_{\min} \\ \mathbf{I}_{H_c m} \Delta \mathbf{u}_{\max} \end{bmatrix}$$

3. Output amplitude constraints (via prediction matrices) are expressed as

$$\begin{bmatrix} -\mathbf{R}_u \\ \mathbf{R}_u \end{bmatrix} \Delta U < \begin{bmatrix} -\mathbf{I}_{H_p n} \mathbf{y}_{\min} + \mathbf{R}_x \mathbf{A}_a\left(k\right) \mathbf{x}_a\left(k\right) \\ \mathbf{I}_{H_p n} \mathbf{y}_{\max} - \mathbf{R}_x \mathbf{A}_a\left(k\right) \mathbf{x}_a\left(k\right) \end{bmatrix}$$

4. State rate constraints can also be expressed via differential prediction matrices, as

$$\begin{bmatrix} -\mathbf{C}_a\left(k+1\right) \mathbf{B}_a\left(k\right) \\ \mathbf{C}_a\left(k+1\right) \mathbf{B}_a\left(k\right) \\ -\Delta \mathbf{R}_u \\ \Delta \mathbf{R}_u \end{bmatrix} \Delta U < \begin{bmatrix} -\Delta \mathbf{y}_{\min} + \mathbf{C}_a\left(k+1\right) \mathbf{A}_a\left(k\right) \mathbf{x}_a\left(k\right) - \mathbf{y}_a\left(k\right) \\ \Delta \mathbf{y}_{\max} - \mathbf{C}_a\left(k+1\right) \mathbf{A}_a\left(k\right) \mathbf{x}_a\left(k\right) + \mathbf{y}_a\left(k\right) \\ -\mathbf{I}_{(H_p-1)n} \Delta \mathbf{y}_{\min} + \Delta \mathbf{R}_x \mathbf{A}_a\left(k\right) \mathbf{x}_a\left(k\right) \\ \mathbf{I}_{(H_p-1)n} \Delta \mathbf{y}_{\max} - \Delta \mathbf{R}_x \mathbf{A}_a\left(k\right) \mathbf{x}_a\left(k\right) \end{bmatrix}$$

where the difference prediction matrices are defined as

$$\begin{aligned} \left(\Delta \mathbf{R}_x\right)_i &= \left(\mathbf{R}_x\right)_{i+1} - \left(\mathbf{R}_x\right)_i \\ \left(\Delta \mathbf{R}_u\right)_i &= \left(\mathbf{R}_u\right)_{i+1} - \left(\mathbf{R}_u\right)_i \end{aligned}$$

For more details on the derivation, we refer the reader to Oviedo or Mollov[OVW06, MBAV04].

5. Simulation Results

5.1. Overview

We simulated the system with the described controller using Python. For these simulations, constant values come from nominal Martian parameters with initial conditions and vehicle parameters from the Mars sample return orbiter[PB93, MRFP00], as shown in Table 5.1. When evaluating the MPC controller, an arbitrary reference is used mimicking a doublet maneuver. As a baseline, the cost matrix for following the reference states normalized by setting corresponding diagonal element equal to the reciprocal of the expected maximum such as the initial condition, i.e.

$$\mathbf{P} = \begin{bmatrix} \frac{1}{V(t_0)} & 0 & 0 & 0 \\ 0 & \frac{1}{\pi} & 0 & 0 \\ 0 & 0 & \frac{1}{r(t_0)} & 0 \\ 0 & 0 & 0 & \frac{1}{\pi} \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} \frac{1}{\max_t \sigma'(t)} \end{bmatrix}$$

Similarly, we set the cost for error in tracking the reference input equal to the reciprocal of the maximum input. For initial tuning, we set each element in $\Delta\mathbf{P}$ to 0. We will consider the how the sampling time, horizon length, and cost of varying the input $\Delta\mathbf{Q}$ affects the controller.

Table 5.1.: Constant Values Used for Simulation

Parameter	Symbol	Value
reference radius	r_0	$3389.5 \times 10^3\,\mathrm{m}$
scale height	β	$9.009\,\mathrm{m}^{-1}$
gravitational constant	μ	$42828 \times 10^9\,\mathrm{km}^3\mathrm{s}^{-2}$
reference area	S	$14.29\,\mathrm{m}^2$
coefficient of drag	C_D	1.53
coefficient of lift	C_L	0.45
vehicle mass	m	$1179.34\,\mathrm{kg}$

5.2. Varying parameters

In order to test this TSFM based MPC scheme, we discretized the continuous time TSFM. In Figure 5.1, we show a comparison of the model agreement between the original non-linear dynamics and the discretized TSFM for an open-loop control $\sigma' = 0$. As to be expected, the faster sampling rate increases the agreement with the original non-linear model. When applied to the discrete-time MPC, we mainly see the effects of the sampling rate in the control input as shown in Figure 5.2. The slower sampled models cause the controller to change the input so the discrete-time model states can match the reference. However, this model doesn't agree with the non-linear system, causing oscillation and overshoot in the control input. Due to the slow response of the system dynamics, the oscillations and overshoot in control input do not cause the states to deviate greatly from the reference trajectory.

The effect of varying the horizon length and change in input cost are shown in Figures 5.3, 5.4, and 5.5. In most MPC analysis, the horizon time is treated as tuning parameter that increases controller performance along with the burden of computation cost. In cases where the discrete time model has sufficient agreement with the continuous time model, this is true as shown in Figure 5.3. However, in the case where the discrete time model does not sufficiently agree with the non-linear model, such as when the sampling rate is too slow, a short prediction horizon actually improves tracking of the reference input as shown in Figure 5.4. Finally we note that by increasing the cost of varying the input $\Delta\mathbf{Q}$, it is possible to dampen the oscillations and overshoot caused by the disagreement between the discrete-time model and continuous-time nonlinear system.

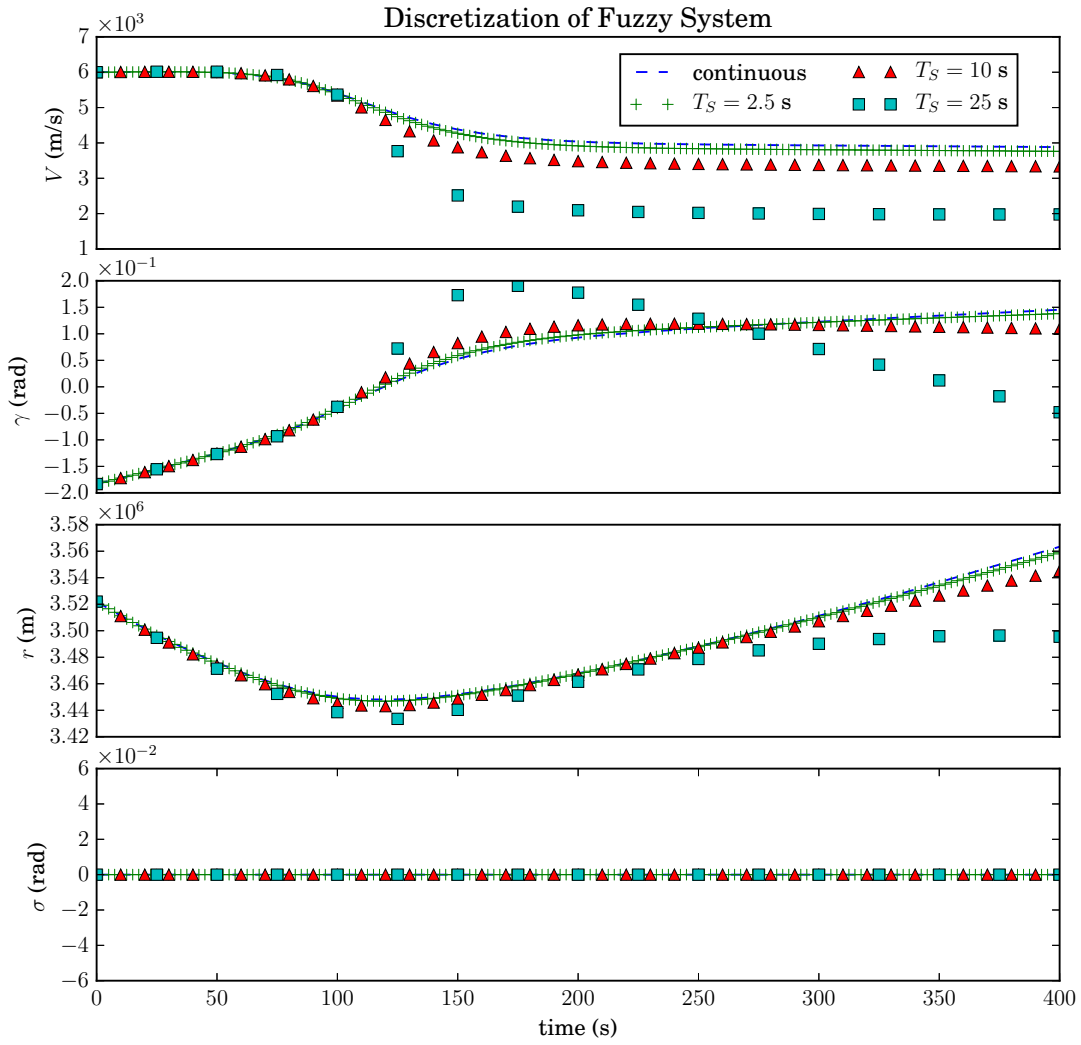

Figure 5.1.: A comparison of the open loop response with the discretized fuzzy model.

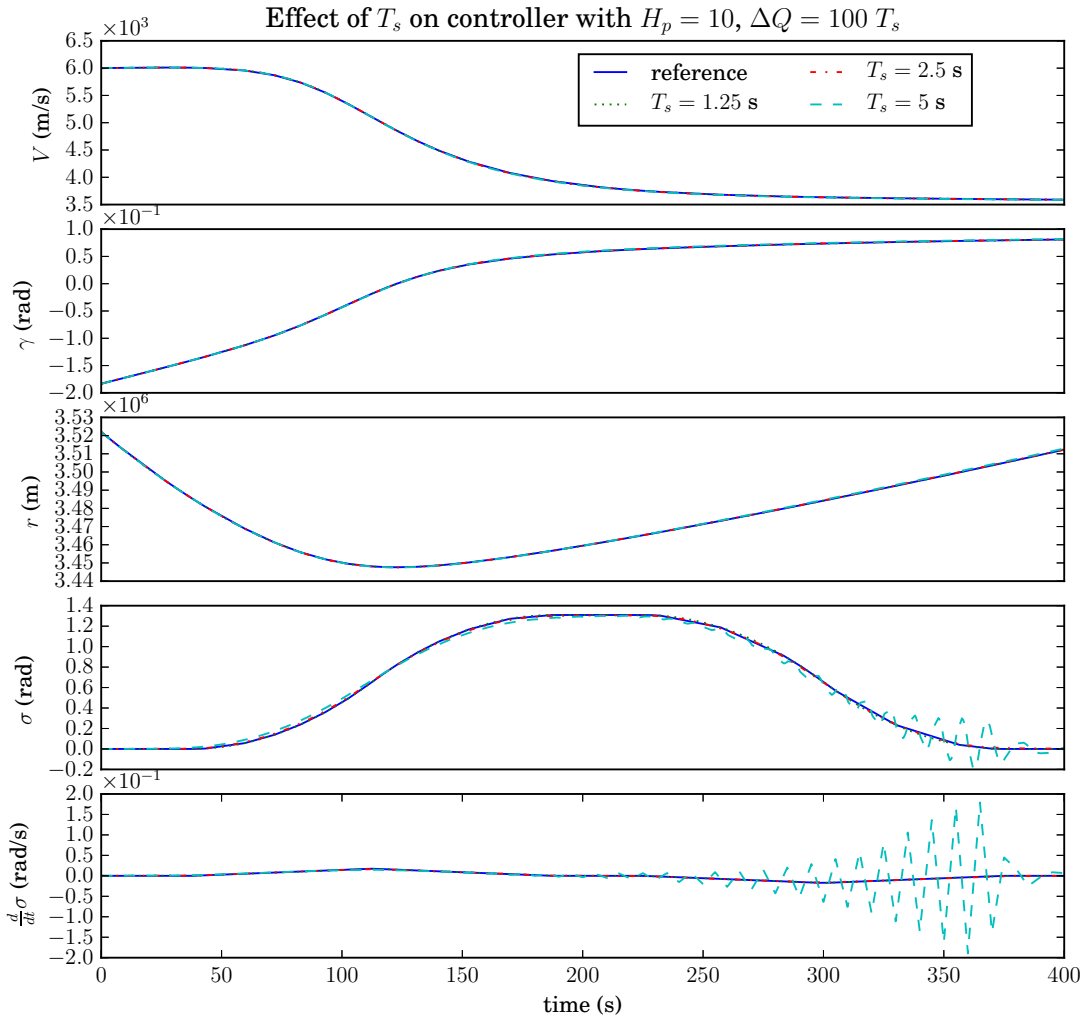

Figure 5.2.: A comparison of the MPC controller results with varying sampling rate.

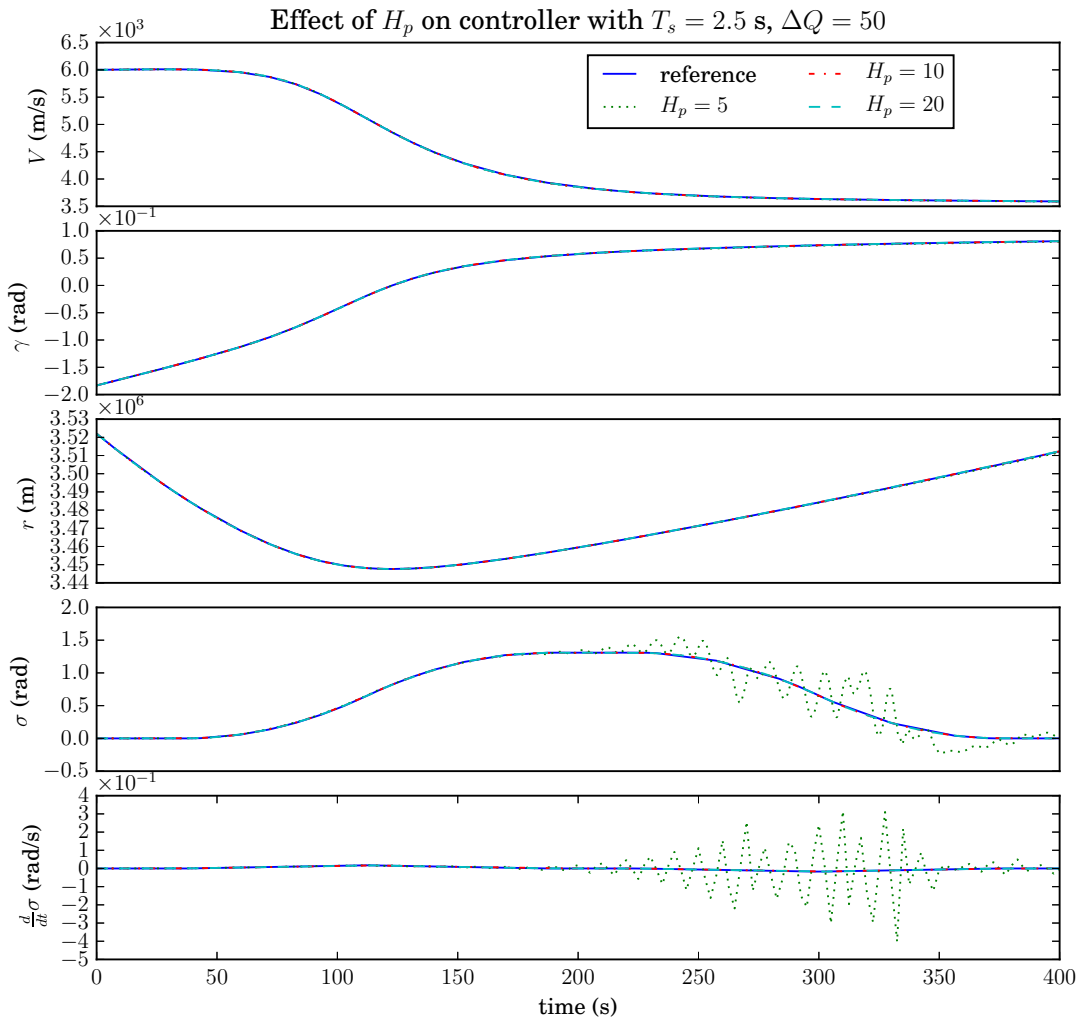

Figure 5.3.: A comparison of the MPC controller results with varying horizon length with a fast sampling rate.

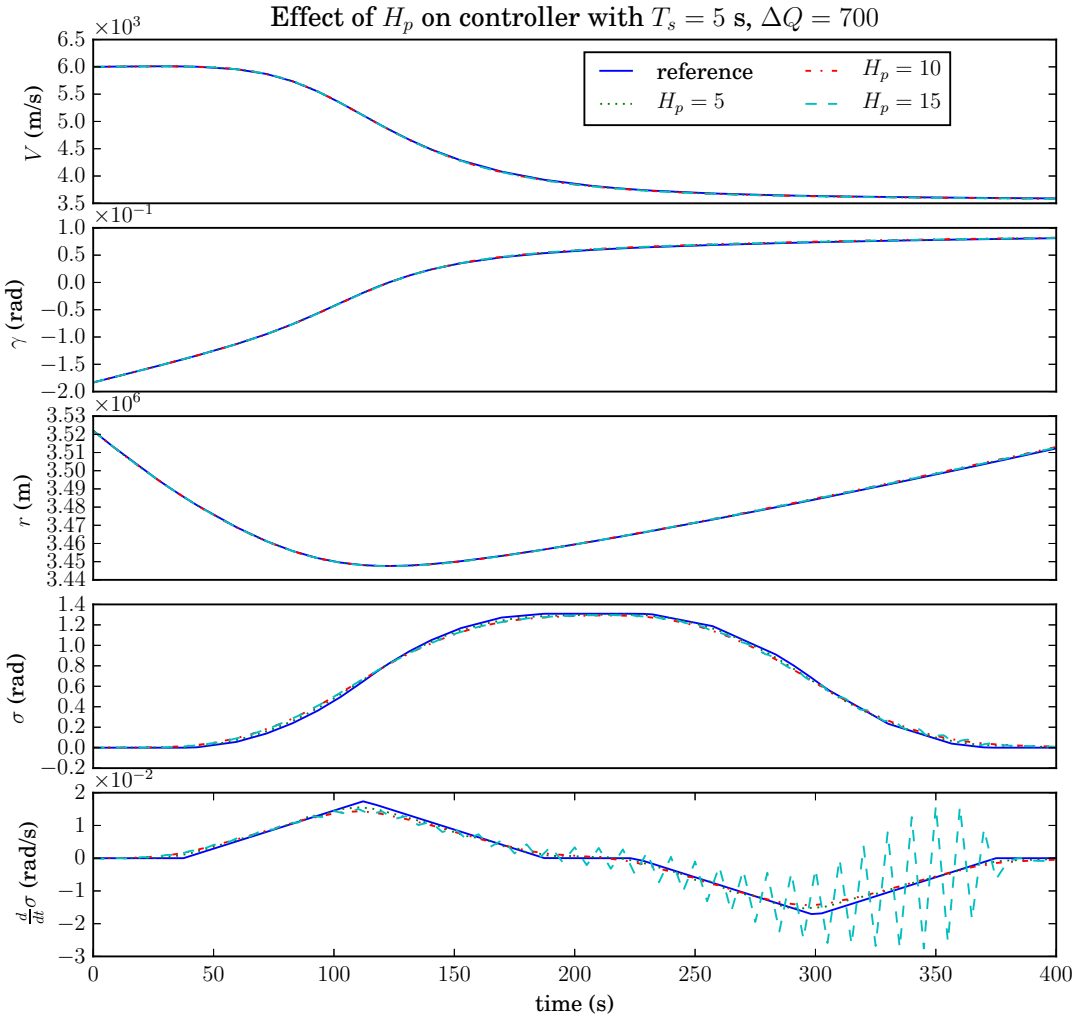

Figure 5.4.: A comparison of the MPC controller results with varying horizon length with a slow sampling rate.

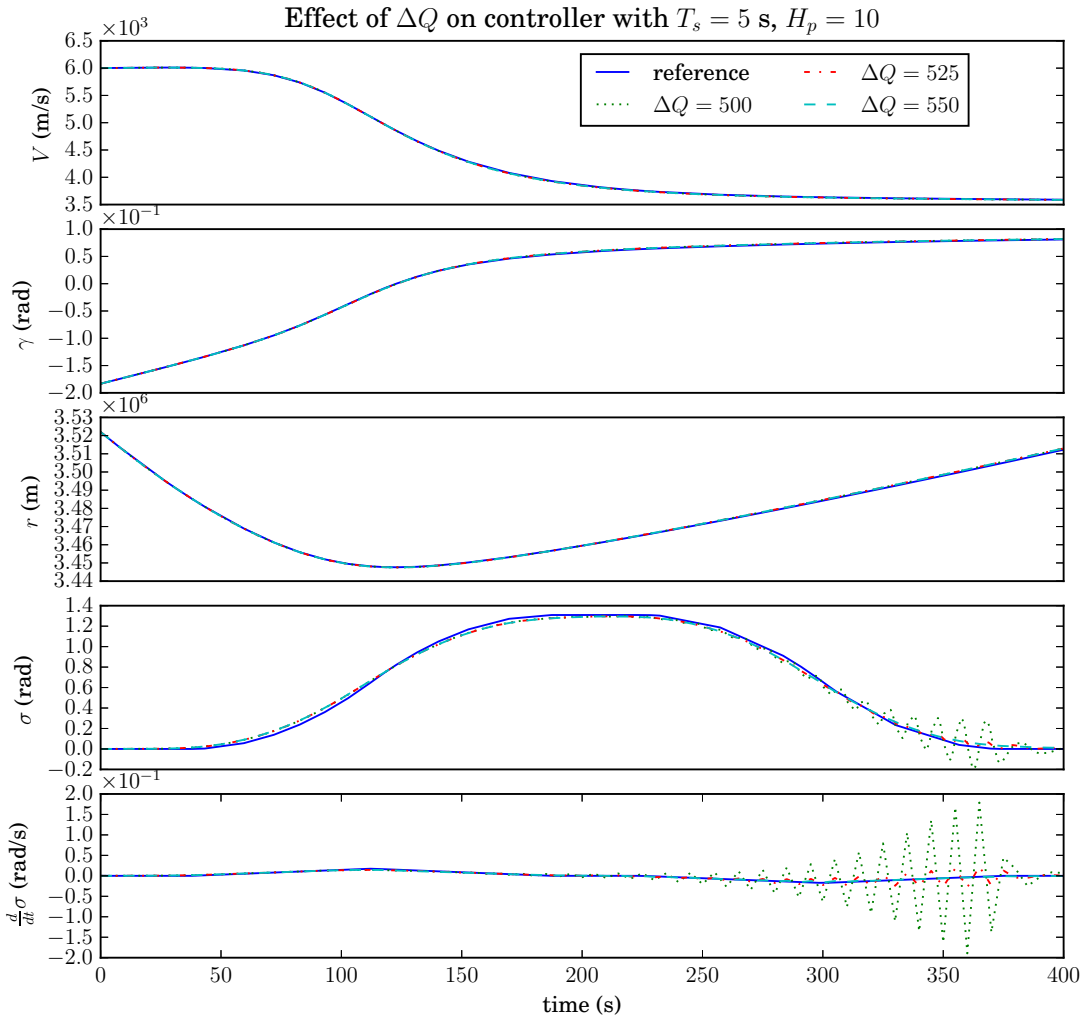

Figure 5.5.: A comparison of the MPC controller results with varying input rate cost.

6. Conclusion

In this paper we described the dynamics of a re-entry vehicle in atmosphere while performing the aerocapture maneuver, used a TSFM with a discrete time MPC scheme, and evaluated the tuning parameters of when following an arbitrary reference. By using a TSFM, we are able to formulate a linear discrete time approximation to the nonlinear model to implement a linear MPC formulation. We found that even without disturbances or model uncertainty, the tuning parameters must be designed to avoid excessive oscillation and overshoot in the control input. In the future, we hope to formulate a continuous time fuzzy-based controller using optimal feed-back gains to compare to this discrete time controller.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1135384 and the support of the Santa Clara University Department of Mechanical Engineering.

A. Python Code

The code listed below implements the Takagi-Sugeno Fuzzy Model based discrete-time Model Predictive Control and runs the simulations generating the data presented in Chapter 5.

```
## file 'dtfmpc.py'
import numpy as np
import cvxopt
from NDMS.Systems import DynamicalSystem
from NDMS.utils import process_vector_args
from FuzzyControl.FuzzyModel import TakagiSugenoFuzzyModel as TSFM, FuzzyMatrix

# a helper class for matrix_mul_list that inherits its dimensions from the
# other multiplicant
class DeferredIdentity(object):
    def __rmul__(self, other):
        return other.copy()

    def __imul__(self, other):
        return other.copy()

    def __mul__(self, other):
        return other.copy()

# a helper function for multiplying a list of matrices in place
def matrix_mul_list(matrices):
    result = DeferredIdentity() # np.eye(matrices[0].shape)
    for matrix in matrices:
        result *= matrix
    return result

def generate_dtfmpc(
    system, reference,
    P, Delta_P, Q, Delta_Q,
    u_min, u_max, Delta_u_min, Delta_u_max,
    y_min, y_max, Delta_y_min, Delta_y_max):
    """
    A function to generate a discrete-time fuzzy model predictive control
```

```
    system - a dt Takagi-Sugeno fuzzy model. Assumes direct access to states

    reference - callable that returns matrix/2D array of states (output) and
        inputs over their respective horizons.

    P, Delta_P, Q, Delta_Q - are the weighting matrices

    the remaining arguments are the constraints

    returns a controller usable in BlockDiagrams and



    """
    # system dimensions
    m = system.n_inputs
    n = system.n_states
    nbar = m+n

    # reserve space in memory for matrices
    Abar_holder = np.matrix(np.eye(n+m))
    Bbar_holder = np.matrix(np.empty((n+m,m)))
    Bbar_holder[n:n+m,0:m] = np.eye(m)
    Cbar_holder = np.matrix(
        np.concatenate((np.eye(n,n), np.zeros((n,m))), axis=1))

    # discrete time MPC controller
    def controller(*args, as_sys=False):
        # if used as a system in a block diagram, it will be passed more args
        # expand the args into the correct variables
        if as_sys:
            if len(args) == 3:
                t,x,u = args
            elif len(args) == 2:
                t,x = args
        else:
            t,x, result = args

        # extract the current control input, state, and output
        u_k = np.matrix(x[-m:]).reshape((-1,1))
        x_k = np.matrix(x[:]).reshape((-1,1))
        y_k = x_k[:-m].copy()

        # get the reference, determine the prediction and control horizons
        # based on what is returned
        ref_y, ref_u = reference(t)
        H_p = ref_y.shape[0] -1
        H_c = ref_u.shape[0] -1

        # if the prediction horizon is zero (or negative), use 0 control effort
        if H_p <= 0:
```

```
    return np.zeros((m,1))

# helper functions to evaluate the system matrices over the reference
def Abar(k):
    Abar_holder[0:n,0:n] = system.A(system.h(k,ref_y[k]))
    Abar_holder[0:n,n:n+m] = system.B(system.h(k,ref_y[k]))
    return Abar_holder.copy()

def Bbar(k):
    Bbar_holder[0:n,0:m] = system.B(system.h(k,ref_y[k]))
    return Bbar_holder.copy()

def Cbar(k=0):
    return Cbar_holder.copy()

# reserve space for free and forced prediction matrices R_x and R_u
R_x = np.asmatrix(np.zeros((n*H_p,nbar)))
R_u = np.asmatrix(np.zeros((n*H_p,m*(H_c))))

# compute R_u and R_x using block matrix construction
for i in range(H_p):
    R_x[i*n:(i+1)*n,:] = Cbar(i+1) * matrix_mul_list([
        Abar(a) for a in range(i,0,-1)
    ])

for i in range(H_p):
    for j in range(0,min(H_c,i+1)):
        R_u[i*n:(i+1)*n,j*m:(j+1)*m] = Cbar(i+1) * matrix_mul_list([
            Abar(a) for a in range(i,j,-1)
        ]) * Bbar(j)

# Compute helper matrices used in QP formulation
R_u2 = np.asmatrix(np.empty_like(R_u))
R_u2[:n,:] = np.zeros_like(R_u[:n,:])
R_u2[n:,:] = R_u[:-n,:].copy()

dR_x1 = Cbar(1)
dR_u1 = np.asmatrix(np.zeros((n,m*H_c)))
dR_u1[:,:m] = Cbar(1)*Bbar(0)

dR_x = np.asmatrix(np.empty_like(R_x[n:,:]))
dR_x[:,:] = R_x[n:,:] - R_x[:-n,:]

dR_u = np.asmatrix(np.empty_like(R_u[n:,:]))
dR_u[:,:] = R_u[n:,:] - R_u[:-n,:]

I_m = np.asmatrix(np.eye(m))

I_u = np.asmatrix(np.tile(I_m,(H_c,1)))

I_Delta_u = np.tril(np.tile(I_m, (H_c,H_c)), k=0)
```

```
I_Hc_m = np.asmatrix(np.eye(H_c*m))

# build the matrix used for inequality constraints
inequality_matrix = cvxopt.matrix(np.concatenate((
    -I_Delta_u,
    I_Delta_u,
    -I_Hc_m,
    I_Hc_m,
    -R_u,
    R_u,
    -dR_u1,
    dR_u1,
    -dR_u,
    dR_u,
), axis=0))

# build the vector used for inequality constraints
inequality_vector = cvxopt.matrix( np.concatenate((
    I_u*(-u_min + u_k),
    I_u*(u_max - u_k),
    -I_Hc_m*np.tile(Delta_u_min,(H_c,1)),
    I_Hc_m*np.tile(Delta_u_max,(H_c,1)),
    -np.tile(y_min,(H_p,1))+R_x*Abar(0)*x_k,
    np.tile(y_max,(H_p,1))-R_x*Abar(0)*x_k,
    -Delta_y_min+dR_x1*Abar(0)*x_k-y_k,
    Delta_y_max-dR_x1*Abar(0)*x_k+y_k,
    -np.tile(Delta_y_min,(H_p-1,1))+dR_x*Abar(0)*x_k,
    np.tile(Delta_y_max,(H_p-1,1))-dR_x*Abar(0)*x_k,
), axis=0) )

# build block matrices for the QP cost
Pbar = np.asmatrix(np.zeros((n*H_p,)*2))
DeltaPbar = np.asmatrix(np.zeros((n*H_p,)*2))
for i in range(H_p):
    Pbar[i*n:(i+1)*n, i*n:(i+1)*n] = P
    DeltaPbar[i*n:(i+1)*n, i*n:(i+1)*n] = Delta_P

Qbar = np.asmatrix(np.zeros((m*(H_c),)*2))
DeltaQbar = np.asmatrix(np.zeros((m*(H_c),)*2))
for j in range(H_c):
    Qbar[j*m:(j+1)*m,j*m:(j+1)*m] = Q
    DeltaQbar[j*m:(j+1)*m,j*m:(j+1)*m] = Delta_Q

# QP hessian matrix
hessian = cvxopt.matrix(
    2*(
        R_u.T*Pbar*R_u
        + I_Delta_u.T*Qbar*I_Delta_u
        + (R_u-R_u2).T*DeltaPbar*(R_u-R_u2)
        + DeltaQbar
    )
```

```
        )

        # QP gradient
        gradient = cvxopt.matrix(
            2*(
                (R_x*Abar(0)*x_k - ref_y [1:,:]. reshape ((-1,1))). T*Pbar*R_u [:,:]
                + ( I_u*u_k - ref_u [:-1,:]. reshape ((-1,1))). T * Qbar*I_Delta_u
                + (R_x* Abar(0)*x_k - R_x*x_k).T*DeltaPbar*(R_u-R_u2)
            ). T
        )


        try:
            # try to solve QP using MOSEK
            opt_result = cvxopt. solvers .qp(hessian, gradient,
                inequality_matrix, inequality_vector, solver='mosek')
        except:
            # if solver fails, return NAN
            return np.nan*np.ones((m,1))

        # if solver succeeds but does not solve, return NAN
        if opt_result ['x'] is None:
            return np.nan*np.ones((m,1))

        # if solver succeeds, extract result and return it
        Delta_u = np.matrix( opt_result ['x'])
        u_out = Delta_u [:m]
        return u_out

    # use the controller function to construct a "block" usable in simulation
    controller .dt = system.dt
    control_sys = DynamicalSystem(dt=system.dt)
    control_sys . n_states = 0
    control_sys .n_outputs = system.n_inputs
    control_sys .n_inputs = system. n_states + system.n_inputs
    control_sys . state_equation_function = lambda t, x: x
    control_sys . output_equation_function = lambda *args: controller (* args,
        as_sys=True)

    return controller , control_sys

## script to execute
import numpy as np, sympy as sp, matplotlib .pyplot as plt, numpy.matlib as ml, pandas as pd
from NDMS.Systems import DynamicalSystem, SystemFromCallable
from NDMS.BlockDiagram import BlockDiagram
from NDMS.utils import sinc, augment_inputs, process_vector_args, callable_from_trajectory
from sympy.physics.mechanics import dynamicsymbols
from scipy.optimize import minimize_scalar, root as find_zero

from FuzzyControl.FuzzyModel import TakagiSugenoFuzzyModel as TSFM, FuzzyMatrix, \
    SectorNonlinearityFromPremiseVariables, FuzzyC2D
```

```
import importlib

import cvxopt, mosek

from sympy.utilities.lambdify import lambdify, implemented_function

from sympy import symbols, sin, cos, S, exp, tan

from datetime import datetime
from scipy.io import savemat, loadmat
import os, re, itertools


# define environmental settings
sp.init_printing()
plt.ion()
plt.rc('font', family='serif')
plt.rc('text', usetex=True)
simulation_parameters={'atol': 1E-15, 'rtol': 1E-9, 'nsteps':1000}

try:
    os.chdir('.\\Projects\\HypersonicReEntry\\TO_SUBMIT')
except:
    pass

import dtfmpc

VERBOSE = False

if VERBOSE:
    cvxopt.solvers.options['MOSEK'] = {mosek.iparam.log: 1}
else:
    cvxopt.solvers.options['MOSEK'] = {mosek.iparam.log: 0}


# define symbols
tvar = dynamicsymbols._t
svar = sp.symbols('s')
V, r, gamma, sigma = dynamicsymbols('V r gamma sigma')
sigma_p_input = dynamicsymbols('sigmaprime')
C_D, C_L, m, mu, beta, r0, rho0 = symbols('C_D C_L m mu beta r0 rho0')
D,L = dynamicsymbols('D L')
rho, g = symbols('rho g', cls=sp.Function)
tF_sym = sp.symbols('t_F')

# constant values
SA = 14.29 # reference surface area in m^2
constants = {
    beta: 1/(11.1E3), # scale height in m^-1
    rho0: 0.02, # reference density in kg/m^3
    r0: 3389.5E3, # reference radius (surface) in m (3389.5 km)
```

```
    C_D: 1.53 * SA,
    C_L: 0.45 * SA,
    m: 1179.34, # vehicle mass in kg; 2600 lb
    mu: 42828E9, # reference gravity constant in km^3 * s^-2,
}
V_ic = 6E3 # initial velocity , 6 km/s
r_ic = 3522E3 # atmospheric interface radius, in m
sigma_ic = 0 # initial bank angle, in rad
gamma_ic = -10.5*np.pi/180 # initial flight path angle, in rad (-10.5 deg)
tF_value = 435 # seconds to simulate
u_rate = 1*np.pi/180 # bank angle acceleration rate

# definitions which can be used to simulate over a range of initial conditions
g_diff = 0
V_diff = 0
gs_to_test = np.linspace(0,0,1)
Vs_to_test = np.linspace(0,0,1)

Vg_ic_error_to_iter = [ test
    for test in itertools.product(Vs_to_test, gs_to_test) ]

# Define the P and Q cost matrices
DEFAULT_STATE_COST = np.diagflat(1/np.array([V_ic, r_ic, np.pi, np.pi ]))
DEFAULT_INPUT_COST = np.matrix([1/u_rate])

# define dynamics for control-affine model
rho_expr = rho0*sp.functions.exp(-beta*(r-r0))
g_expr = mu/r**2
dV_dt = -C_D*V**2*rho(r)/(2*m) - mu*sin(gamma)/r**2
dg_dt = C_L*V*rho(r)*cos(sigma)/(2*m) - mu*cos(gamma)/(V*r**2) + V*cos(gamma)/r
dr_dt = V*sin(gamma)

# define the control-affine model
physical_state = sp.Matrix([V,r,gamma,sigma])
physical_state_equation = (sp.Matrix([dV_dt, dr_dt, dg_dt, sigma_p_input])
    .subs(rho(r), rho_expr))
physical_input = sp.Matrix([sigma_p_input])
physical_output_equation = physical_state

physical_sys = DynamicalSystem(
    physical_state_equation,
    physical_state,
    physical_input,
    physical_output_equation,
    constants_values=constants
)
physical_x0 = np.matrix([V_ic, r_ic, gamma_ic, sigma_ic]).T
physical_sys.initial_condition = physical_x0

# define open loop control (no input)
def open_loop_ctr(t):
```

```
    return np.matrix([0])
open_loop_ctr_sys = SystemFromCallable(open_loop_ctr,0,1)

# define arbitrary reference control (doublet maneuver)
tint = 75
t_switch_ref_control = np.array([
    0,
    0.5,
    1.5,
    2.5,
    3,
    4,
    5, tF_value/tint
])*tint

def ref_control(t):
    if t<=t_switch_ref_control[1]:
        u=0
    elif t<=t_switch_ref_control[2]:
        u = (t-t_switch_ref_control[1])*u_rate/tint
    elif t<=t_switch_ref_control[3]:
        u = u_rate+(t_switch_ref_control[2]-t)*u_rate/tint
    elif t<=t_switch_ref_control[4]:
        u = 0
    elif t<=t_switch_ref_control[5]:
        u = (t_switch_ref_control[4]-t)*u_rate/tint
    elif t<=t_switch_ref_control[6]:
        u = (t-t_switch_ref_control[5])*u_rate/tint -u_rate
    else:
        u = 0
    return np.matrix([u])
ref_ctr_sys_dimful = SystemFromCallable(ref_control,0,1)

# define block diagram with reference control of physical system
dimful_BD = BlockDiagram(physical_sys, ref_ctr_sys_dimful)
dimful_BD.connect(ref_ctr_sys_dimful, physical_sys)

# simulate block diagram piecewise over to increase numerical accuracy
dimful_ref_reses = []
old_tswitch = 0
physical_sys.initial_condition = physical_x0
for tswitch in t_switch_ref_control[1:]:
    dimful_ref_reses.append(
        dimful_BD.simulate([old_tswitch, tswitch],
        integrator_options=simulation_parameters)
    )
    old_tswitch = tswitch
    physical_sys.initial_condition = np.matrix(dimful_ref_reses[-1].x[-1,:]).T

# combine piecewise simulation results
dimful_res__y = np.concatenate(
```

```
    [dimful_res.y for dimful_res in dimful_ref_reses], axis=0)
dimful_res__t = np.concatenate(
    [dimful_res.t for dimful_res in dimful_ref_reses])
dimful_PD = pd.DataFrame(index=dimful_res__t, data=dimful_res__y,
    columns=[st for st in physical_state]+[inp for inp in physical_input])

# create interpolation callable of simulation results
dimful_res_unique_t, dimful_res_unique_t_idx = np.unique(dimful_res__t,
    return_index=True)
dimful_ref_callable = callable_from_trajectory(dimful_res_unique_t,
    dimful_res__y[dimful_res_unique_t_idx,:])
dimful_ref_sys = SystemFromCallable(dimful_ref_callable, 0, 5)

# find min and max values of state from simulation
ref_tF_value = find_zero(lambda t: dimful_ref_callable(t)[1] - r_ic,
    300)['x'][0]
Vf_ref, gf_ref = dimful_ref_callable(ref_tF_value)[[0,2]]

V_min = Vf_ref
V_max = -minimize_scalar(lambda t: -dimful_ref_callable(t)[0],
    bounds=[0, ref_tF_value], method='Bounded').fun

r_min = minimize_scalar(lambda t: dimful_ref_callable(t)[1],
    bounds=[0, ref_tF_value], method='Bounded').fun
r_max = r_ic

g_min = gamma_ic
g_max = gf_ref

s_min = sigma_ic
s_max = -minimize_scalar(lambda t: -dimful_ref_callable(t)[3],
    [0, ref_tF_value], [0, ref_tF_value])['fun']

V_scale_factor = V_max - V_min
g_scale_factor = g_max - g_min
r_scale_factor = r_max - r_min
s_scale_factor = s_max - s_min

# create TSFM
# symbols for premise variables
z = symbols('z1:5')
z1, z2, z3, z4 = z
z_to_nl = {}
z_to_nl[z1] = rho_expr*V
z_to_nl[z2] = -mu*sinc(gamma)/(r**2)
z_to_nl[z3] = C_L*rho_expr*cos(sigma)/(2*m)-(mu/(V**2*r**2)-1/r)*cos(gamma)
z_to_nl[z4] = sin(gamma)

# fuzzy rule state and input matrix structure
sub_A = sp.Matrix([[-C_D*z1/(2*m),0,z2,0],[z4,0,0,0],[z3,0,0,0],[0,0,0,0]])
sub_B = sp.Matrix([0, 0, 0, 1])
```

```
# extreme state used for sector nonlinearity
extreme_states = {V: (S(1E3), S(8E3 ),),
    r: (300E3 + constants[r0 ], constants[r0 ],),
    gamma: (np.pi /2, -np.pi/2, 0,),
    sigma: (0, np.pi /2, np.pi ,)
}

# construct fuzzy matrices and TSFM from structure and extreme values
(fuzzy_A,fuzzy_B),h = SectorNonlinearityFromPremiseVariables (
    (sub_A, sub_B), z_to_nl , physical_state , extreme_states , constants)
fuzzy_ct = TSFM(h,fuzzy_A,fuzzy_B,None)

# augmented physical system to control
augmented_physical_sys = augment_inputs(physical_sys)
augmented_physical_sys.output_equations = augmented_physical_sys.states
augmented_physical_sys. initial_condition = np.concatenate(
    (physical_x0 , np.matrix ([0])), axis=0)
system_for_control = augmented_physical_sys

# define input , state , and increment constraints
u_constr = np.asmatrix(20* np.pi /180) # roll -rate 20 degrees /s
DU_constr = np.asmatrix (6.8* np.pi /180) # bank accell 6.8 degrees /s^2
y_min_constr = np.asmatrix ([0, constants[r0 ], -np.pi/2, -1*np.pi ]). T
y_max_constr = np.asmatrix([12E3, 1.1* r_ic , np.pi /2, 1*np.pi ]). T
DY_constr = (y_max_constr-y_min_constr)

## simulate reference trajectory with dt TSFM
Tses = (2.5, 10, 25)
 list_of_simulation_data = []
 list_of_ids = []
output_symbols = (sp. flatten (physical_sys .output_equations. tolist ())
    + [physical_sys .inputs [0]])
for ctr_sys in [open_loop_ctr_sys ]: # open_loop_ctr_sys ]: # ref_ctr_sys_dimful
    for Ts in Tses:
            dt_sys = FuzzyC2D(fuzzy_ct, Ts, method='gbt',alpha =0.0)

            dt_sys. initial_condition = physical_x0
            BD = BlockDiagram(dt_sys, ctr_sys )
            BD.connect(ctr_sys , dt_sys)

            DT_ID = str(Ts)
            sim_start_time = datetime .now()
            print("\n\nsimulating",
                DT_ID,"starting␣at", sim_start_time .__format__('%H:%M:%S')
            )
            ctr_res = BD.simulate(tF_value ,
                integrator_options =simulation_parameters
            )
            print("simulation␣completed␣at",
                datetime .now().__format__('%H:%M:%S'),"taking␣approximately",
```

```
            (datetime.now() - sim_start_time)
        )
        ctr_res_PD = pd.DataFrame(
            index=ctr_res.t,
            data=ctr_res.y[:,:],
            columns=[st for st in output_symbols]
        )
        list_of_simulation_data.append(ctr_res_PD)
        list_of_ids.append(DT_ID)



## simulate MPC with different parameters
# list of tuples of sample rate, prediction horizon, and input increment cost
compare_array = [(1,15,100)]

# matrices for state and input cost
PP = DEFAULT_STATE_COST
QQ = DEFAULT_INPUT_COST

list_of_simulation_data = []
list_of_ids = []

output_symbols = (sp.flatten(augmented_physical_sys.output_equations.tolist())
    + [augmented_physical_sys.inputs[0]]
)
for Ts, HH, DQ in compare_array:
    for dt_type in ('j'):#,'f'):
        # augmentedDFS = dtfmpc.augment_system(fuzzy_dt)
        T_ref_gen = np.arange(0, tF_value+Ts, Ts)
        ref_gen = dimful_ref_callable(T_ref_gen)[[0,1,2,3,-1],:].T

        augmented_x0 = ref_gen[0,:]
        def genned_reference(t):
            if np.isnan(t):
                return
            k = np.floor(t/Ts)
            j = min(k+HH, len(T_ref_gen))
            if VERBOSE:
                print(t,k,j)
            return ref_gen[k:j,:-1], ref_gen[k:j,-1].reshape(-1,1)

        dt_sys = FuzzyC2D(fuzzy_ct, Ts, method='gbt',alpha=0.0)

        fdtmpc_controller, ctr_sys = dtfmpc.generate_dtfmpc(dt_sys, genned_reference,
            np.asmatrix(PP), # P
            np.asmatrix(np.eye(len(physical_state))*0.0), # Delta_P
            np.asmatrix(QQ), # Q
            np.asmatrix(DQ), # Delta_Q
            -u_constr*Ts, # u_min
            u_constr*Ts, # u_max
```

```
        -DU_constr*Ts, # Delta_u_min,
        DU_constr*Ts, # Delta_u_max,
        y_min_constr, # y_min,
        y_max_constr, # y_max,
        -DY_constr*Ts, # DY_min
        DY_constr*Ts, # DY_max
        )

BD = BlockDiagram(system_for_control, ctr_sys)
BD.connect(ctr_sys, system_for_control)
BD.connect(system_for_control, ctr_sys)
BD.connect(system_for_control, ctr_sys)

for V_to_test, g_to_test in Vg_ic_error_to_iter:

    system_for_control.initial_condition[0] = V_to_test + V_ic
    system_for_control.initial_condition[2] = g_to_test + gamma_ic

    MPC_ID = "MPC_%.0f_%d_%d" % (Ts, HH, DQ)
    sim_start_time = datetime.now()
    print("\n\nsimulating",MPC_ID,"with␣IC␣error",
        V_to_test, g_to_test,
        " starting␣at",
        sim_start_time.__format__('%H:%M:%S')
    )
    ctr_res = BD.simulate(tF_value,
        integrator_options=simulation_parameters)
    print("simulation␣completed␣at",
        datetime.now().__format__('%H:%M:%S'),
        "taking␣approximately",(datetime.now() - sim_start_time)
    )
    ctr_res_PD = pd.DataFrame(index=ctr_res.t, data=ctr_res.y[:,:],
                    columns=[st for st in output_symbols])

    list_of_simulation_data.append(ctr_res_PD)
    list_of_ids.append(DT_ID)
```

Bibliography

[CG85] C. Cerimele and J. Gamble. A simplified guidance algorithm for lifting aeroassist orbital transfer vehicles. In *Aerospace Sciences Meetings*. American Institute of Aeronautics and Astronautics, January 1985, http://dx.doi.org/10.2514/6.1985-348.

[CLWM08] Jordi Casoliva, Daniel T Lyons, Aron A Wolf, and Kenneth D Mease. Robust guidance via a predictor-corrector algorithm with drag tracking for aero-gravity assist maneuvers. In *Guidance, Navigation, and Control and Co-located Conferences*. American Institute of Aeronautics and Astronautics, August 2008, http://dx.doi.org/10.2514/6.2008-6816.

[Cru79] MI Cruz. The aerocapture vehicle mission design concept. In *Conference on Advanced Technology for Future Space Systems, Hampton, VA*, pages 195–201. American Institute of Aeronautics and Astronautics, May 1979, http://dx.doi.org/10.2514/6.1979-893.

[FPRS00] H Fraysse, R Powell, Striepe Rousseau, and S Striepe. Cnes-nasa studies of the mars sample return orbiter aerocapture phase. January 2000.

[JW04] Roman Y. Jits and Gerald D. Walberg. Blended control, predictor–corrector guidance algorithm: an enabling technology for mars aerocapture. *Acta astronautica*, 54(6): 385–398, 2004.

[KEE10] Mohamed Khairy, Abdel-Latif Elshafei, and Hassan M Emara. Lmi based design of constrained fuzzy predictive control. *Fuzzy Sets and Systems*, 161(6): 893–918, 2010.

[LCTM15] Ping Lu, Christopher J. Cerimele, Michael A. Tigges, and Daniel A. Matz. Optimal aerocapture guidance. In *AIAA SciTech*. American Institute of Aeronautics and Astronautics, January 2015.

[Lu99] Ping Lu. Regulation about time-varying trajectories: Precision entry guid-

ance illustrated. *Journal of Guidance, Control, and Dynamics*, 22(6): 784–790, November 1999, http://dx.doi.org/10.2514/2.4479.

[MBAV04] Stanimir Mollov, Robert Babuška, Janos Abonyi, and Henk B Verbruggen. Effective optimization for fuzzy model predictive control. *Fuzzy Systems, IEEE Transactions on*, 12(5): 661–675, 2004.

[Mie96] A Miele. The 1st john v. breakwell memorial lecture: Recent advances in the optimization and guidance of aeroassisted orbital transfers. *Acta astronautica*, 38(10): 747–768, 1996.

[MQ03] James Masciarelli and Eric Queen. Guidance algorithms for aerocapture at titan. 4822, July 2003, http://dx.doi.org/10.2514/6.2003-4804.

[MRFP00] James Masciarelli, Stephane Rousseau, Hubert Fraysse, and Etienne Perot. An analytic aerocapture guidance algorithm for the mars sample return orbiter. In *Guidance, Navigation, and Control and Co-located Conferences*, pages 14–18. American Institute of Aeronautics and Astronautics, August 2000, http://dx.doi.org/10.2514/6.2000-4116.

[NP99] Hazem N Nounou and Kevin M Passino. Fuzzy model predictive control: techniques, stability issues, and examples. In *Intelligent Control/Intelligent Systems and Semiotics, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 423–428. IEEE, 1999.

[OVW06] Jairo Jose Espinosa Oviedo, Joos PL Vandewalle, and Vincent Wertz. *Fuzzy logic, identification and predictive control*. Springer Science & Business Media, 2006.

[PB93] Richard W. Powell and Robert D. Braun. Six-degree-of-freedom guidance and control analysis of mars aerocapture. *Journal of Guidance, Control, and Dynamics*, 16(6): 1038–1044, November 1993, http://dx.doi.org/10.2514/3.21125.

[Que04] Eric M Queen. Angle-of-attack-modulated terminal point control for neptune aerocapture. February 2004.

[RMBV99] Johannes A Roubos, Stanimir Mollov, R Babuška, and Henk B Verbruggen. Fuzzy model-based predictive control using takagi–sugeno models. *International Journal of Approximate Reasoning*, 22(1): 3–30, 1999.

[RPG+02] Stephane Rousseau, Etienne Perot, Claude Graves, James Masciarelli, and

Eric Queen. Aerocapture guidance algorithm comparison campaign. August 2002, http://dx.doi.org/10.2514/6.2002-4822.

[RQ98] Theodore Ro and Eric Queen. Study of martian aerocapture terminal point guidance. In *Guidance, Navigation, and Control and Co-located Conferences*. American Institute of Aeronautics and Astronautics, August 1998.

[SYLK09] Chonghui Song, Jinchun Ye, Derong Liu, and Qi Kang. Generalized receding horizon control of fuzzy systems based on numerical optimization algorithm. *Fuzzy Systems, IEEE Transactions on*, 17(6): 1336–1352, 2009.

[TW01] Kazuo Tanaka and Hua O. Wang. *Fuzzy Control Systems Design and Analysis*. John Wiley & Sons, Inc., 2001.

[VBC80] Nguyen X Vinh, Adolf Busemann, and Robert D Culp. *Hypersonic and planetary entry flight mechanics*. University of Michigan Press, 1980.

[Wal85] G. D. Walberg. A survey of aeroassisted orbit transfer. *Journal of Spacecraft and Rockets*, 22(1): 3–18, January 1985, http://dx.doi.org/10.2514/3.25704.

[Wan09] Liuping Wang. *Model predictive control system design and implementation using MATLAB®*. Springer Science & Business Media, 2009.

[ZFL07] Tiejun Zhang, Gang Feng, and Jianhong Lu. Fuzzy constrained min-max model predictive control based on piecewise lyapunov functions. *Fuzzy Systems, IEEE Transactions on*, 15(4): 686–698, 2007.