TRAJECTORY GENERATION BASED GUIDANCE AND CONTROL OF ROTORCRAFT UNMANNED AERIAL VEHICLES

LAI SHUPENG

(*B.Eng.(Hons.), NTU*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE SCHOOL FOR INTEGRATIVE SCIENCES AND ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2016

Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

LAI SHUPENG
21th October 2016

Acknowledgments

I would like to express my deepest and most sincere gratitude to my main supervisor, Prof. Ben M. Chen, for his guidance and support during my study as a Ph.D. student. Without his help on project identification, system work design, and various technique details, my study as a Ph.D. student would not be possible. I would also like to thank Dr. Peng Kemao from Temasek Lab at NUS, and Prof. Luo Delin from Xiamen University for showing me promising directions on the initial stage of my research. Appreciations are also given to Prof. Lee Tong Heng, Prof. Chu Delin and Dr. Chen Chang for their kind suggestions and patience during numerous meetings and discussions.

Special thanks are given to the members of the NUS Unmanned System Research Group for their help and encouragement. Especially, I would like to thank Mr. Wang Kangli and Dr. Li Kun for their help on developing the embedded system used in this thesis, Dr. Cui Jinqiang and Mr. Qin Hailong for developing the SLAM module and Dr. Lin Feng for his creative ideas on various projects. Appreciations are also given to my colleagues and classmates: Dong Xiangxu, Wang Fei, Kevin Ang, Pang Tao, Phang Swee King, Liu Peidong, Yang Zhaolin, Bai Limiao, Ke Yijie, Lin Jing, Bi Yingcai, Li Jiaxin, Qin Geng, Chen Xudong, Lao Mingjie, Lan Menglu, Li Xiang and Tian Hongyu. Moreover, I would like to thank Dr. Bai Haoyu for his hands-on instructions on software design, development and debug.

Also, I have to thank my parents for their understanding on my choice of pursuing post-graduate study.

Last but not least, I would like to appreciate the support, both mentally and economically, from the Graduate School For Integrative Sciences and Engineering of the National University of Singapore who opens the door of my research career.

Contents

Summary

This thesis studies the guidance and control problem of rotorcraft unmanned aerial vehicles in general environments targeting complex missions such as multi-vehicle stage performance and obstacle-strewn navigation.

Modern unmanned aerial vehicles are required to perform precise maneuvers utilizing their dynamic capabilities to achieve task elements which are inaccessible by other means. Pre-planned and high-quality flying trajectories are adopted to satisfy people's subjective opinions in applications like the multi-drone light show and camera path design. On the other hand, in a dynamic world, sensor-guided and sensor-guarded actions are necessary, as there is no perfectly predefined path suitable for the whole mission, and motions need to be solved instantly to deal with unforeseen changes.

The proposed guidance and control structure combines the techniques of trajectory generation and robust control to provide solutions to states-constrained nonlinear control problems. It can be used for multiple applications, such as human-in-the-loop path design, real-time autonomous navigation in obstacle-strewn environment, high-quality reference tracking flight on low-cost platforms, multi-vehicle formation flight, tracking of moving targets and disturbance rejection in windy environment. The guidance and control structure has been successfully realized and tested in real environments on actual UAV platforms.

List of Tables

List of Figures

List of Symbols

Latin variables

$\boldsymbol{a} = [a_x, a_y, a_z]^{\mathrm{T}}$	Vehicle acceleration in $\mathcal{G}$
a	Equivalent of $\boldsymbol{a}$ on a single axis
a_{co}	Acceleration during AC phase for time synchronization
a_{cruise}	Cruise acceleration
$\boldsymbol{a}_{\mathrm{I}} = [a_{\mathrm{I}x}, a_{\mathrm{I}y}, a_{\mathrm{I}z}]$	Internal acceleration state of command filter
a_{I}	Equivalent of $\boldsymbol{a}_{\mathrm{I}}$ on a single axis
a_{reach}	Reachable acceleration after AI phase
$\boldsymbol{a}_{\mathrm{ref}} = [a_{\mathrm{ref}\,x}, a_{\mathrm{ref}\,y}, a_{\mathrm{ref}\,z}]^{\mathrm{T}}$	Vehicle acceleration reference in $\mathcal{G}$
a_{ref}	Equivalent of $\boldsymbol{a}_{\mathrm{ref}}$ on a single axis
a_{rft}	Filtered acceleration command
$\boldsymbol{a}_{\mathrm{max}} = [a_{\mathrm{max}x}, a_{\mathrm{max}y}, a_{\mathrm{max}z}]^{\mathrm{T}}$	Maximum acceleration in $\mathcal{G}$
a_{max}	Equivalent of $\boldsymbol{a}_{\mathrm{max}}$ on a single axis
$\boldsymbol{a}_{\mathrm{min}} = [a_{\mathrm{min}x}, a_{\mathrm{min}y}, a_{\mathrm{min}z}]^{\mathrm{T}}$	Minimum acceleration in $\mathcal{G}$
a_{min}	Equivalent of $\boldsymbol{a}_{\mathrm{min}}$ on a single axis
$\{\mathcal{B}\} : \{\vec{x_{\mathrm{B}}}, \vec{y_{\mathrm{B}}}, \vec{z_{\mathrm{B}}}\}$	Body coordinate
B_{k}	Base function of NUBS
$\boldsymbol{c}_i$	Control point of NUBS, $i \in \mathbb{N}$
c_i	Equivalent of $\boldsymbol{c}_i$ on a single axis, $i \in \mathbb{N}$
d_{a}	Acceleration cruse direction
$\boldsymbol{d}_{\mathrm{air}}$	Air drag force
$\{\mathcal{D}\} : \{\vec{x_{\mathrm{D}}}, \vec{y_{\mathrm{D}}}, \vec{z_{\mathrm{D}}}\}$	Dynamic target coordinate
$\boldsymbol{D}_{\mathrm{s}} = [\boldsymbol{d}_0, \boldsymbol{d}_1, \boldsymbol{d}_2 ...]^{\mathrm{T}}$	3D data point vector
$D_{\mathrm{s}} = [d_0, d_1, d_2 ...]^{\mathrm{T}}$	1D data point vector

d_{lead}	Distance from p_{virtual} to carrot
d_{p}	Velocity cruse direction
$D_{\text{time}} = [t_0, t_1, t_2 ...]^{\text{T}}$	Approaching time vector
$D_{\varpi} = [\Upsilon_0, \Upsilon_1, \Upsilon_2 \dots]$	Time duration vector
$E_{\text{s}} = [\eta_0, \eta_1 \dots]^{\text{T}}$	Control vector of 2nd order NUBS on a single axis
F_{aug}	Feedback control law derived by RPT
F_{Σ}	Magnitude of rotorcraft's total thrust
$f_{\text{A}*}$	A-star cost
$\boldsymbol{g}$	Gravity force
$\{\mathcal{G}\} : \{\vec{x_{\text{G}}}, \vec{y_{\text{G}}}, \vec{z_{\text{G}}}\}$	Global inertia coordinate
G_{s}	Grids covered by a given trajectory
$g_{\text{A}*}$	Smallest cost to starting point
$h_{\text{A}*}$	Estimated cost to goal point
I_{d}	Length of D_{s} and D_{time}
$j_{\text{s}}(t)$	Jerk of cubic clamped B-spline
$\{\mathcal{L}\} : \{\vec{x_{\text{L}}}, \vec{y_{\text{L}}}, \vec{z_{\text{L}}}\}$	Intermediate heading coordinate
m_{q}	Rotorcraft's mass
N'_{c}	Tuning parameter for proportional navigation
$N_{i,p}(\varpi)$	Base function of a general B-spline $i, p \in \mathbb{N}$
$\boldsymbol{p} = [p_x, p_y, p_z]^{\text{T}}$	Vehicle position in $\mathcal{G}$
p	Equivalent of $\boldsymbol{p}$ on a specific axis
$\boldsymbol{p}_{\text{I}} = [p_{\text{I}x}, p_{\text{I}y}, p_{\text{I}z}]^{\text{T}}$	Internal position state of command filter
p_{I}	Equivalent of $\boldsymbol{p}_{\text{I}}$ on a single axis
p_{rand}	Random sampled point in RRT growing
$\boldsymbol{p}_{\text{ref}} = [p_{\text{ref}\,x}, p_{\text{ref}\,y}, p_{\text{ref}\,z}]^{\text{T}}$	Vehicle position reference in $\mathcal{G}$
p_{ref}	Equivalent of $\boldsymbol{p}_{\text{ref}}$ on a specific axis
p_{rft}	Filtered position command
p_{virtual}	Projected vehicle position on flight path
$\boldsymbol{q}$	Rotorcraft's controlled inner loop outputs
$\boldsymbol{q}_{\text{ref}}$	Reference of rotorcraft's controlled inner loop outputs
${}^{\text{G}}\mathbf{R}_{\text{B}}$	Rotation matrix from $\mathcal{B}$ to $\mathcal{G}$

${}^{\mathrm{G}}\mathbf{R}_{\mathrm{L}}$	Rotation matrix from $\mathcal{L}$ to $\mathcal{G}$
${}^{\mathrm{L}}\mathbf{R}_{\mathrm{B}}$	Rotation matrix from $\mathcal{L}$ to $\mathcal{B}$
r_{L1}	L1 nonlinear guidance law radius
r_{v}	Vehicle radius
s	Path parameter of NUBS
S_{k}	NUBS of order k
$\boldsymbol{T}_{\max} = [T_{\max x}, T_{\max y}, T_{\max z}]^{\mathrm{T}}$	Upper cuboid boundary of the virtual thrust
$\boldsymbol{T}_{\min} = [T_{\min x}, T_{\min y}, T_{\min z}]^{\mathrm{T}}$	Lower cuboid boundary of the virtual thrust
$\boldsymbol{T}_{\mathrm{s}} = [\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \boldsymbol{\tau}_2 ...]^{\mathrm{T}}$	Control vector of clamped cubic B-spline
$\boldsymbol{T}_{\Sigma}$	Total thrust vector by all propellers
$\mathbb{T}_{\mathrm{g}} = \{\boldsymbol{T} + \boldsymbol{g} \mid \boldsymbol{T} \in \mathbb{T}_{\Sigma}\}$	The set of virtual thrust considered gravity
$\mathbb{T}_{\Sigma}$	The set of achievable combined thrust
t_{cruise}	Duration of VC phase
T_{search}	RRT searching tree
t_{sp}	A specific timestamp for querying trajectory state
$T_{\mathrm{s}}^{\mathrm{F}}$	Fixed part of T_{s}
$T_{\mathrm{s}}^{\mathrm{P}}$	Programmable part of T_{s}
T_{true}	Real time duration of NUBS trajectory
$T_{\mathrm{s}} = [\tau_0, \tau_1, \tau_2 ...]^{\mathrm{T}}$	Equivalent of $\boldsymbol{T}_{\mathrm{s}}$ on a single axis
$\boldsymbol{u}_{\mathrm{j}} = [u_{\mathrm{j}x}, u_{\mathrm{j}y}, u_{\mathrm{j}z}]^{\mathrm{T}}$	Vehicle jerk in $\mathcal{G}$
u_{j}	Equivalent of $\boldsymbol{u}_{\mathrm{j}}$ on a specific axis
$\boldsymbol{u}_{\mathrm{out}}$	RPT controller's output
$\boldsymbol{u}_{\mathrm{q}}$	Rotorcraft's true inputs
$\boldsymbol{u}_{\mathrm{sat}}$	RPT controller's output after saturation
$\boldsymbol{u}_{\mathrm{v}}$	Virtual input of rotorcraft's outer loop
$\boldsymbol{u}_{\mathrm{j}\max} = [u_{\mathrm{j}\max x}, u_{\mathrm{j}\max y}, u_{\mathrm{j}\max z}]^{\mathrm{T}}$	Maximum jerk in $\mathcal{G}$
$u_{\mathrm{j}\max}$	Equivalent of $\boldsymbol{u}_{\mathrm{j}\max}$ on a single axis
$\boldsymbol{u}_{\mathrm{j}\min} = [u_{\mathrm{j}\min x}, u_{\mathrm{j}\min y}, u_{\mathrm{j}\min z}]^{\mathrm{T}}$	Minimum jerk in $\mathcal{G}$
$u_{\mathrm{j}\min}$	Equivalent of $\boldsymbol{u}_{\mathrm{j}\min}$ on a single axis
$\boldsymbol{v} = [v_x, v_y, v_z]^{\mathrm{T}}$	Vehicle velocity in $\mathcal{G}$
v	Equivalent of $\boldsymbol{v}$ on a specific axis

$\boldsymbol{v}_{\text{air}}$	Air velocity
v_{cruise}	Cruise velocity
v_{end}	End velocity
$\boldsymbol{v}_{\text{I}} = [v_{\text{I}x}, v_{\text{I}y}, v_{\text{I}z}]^{\text{T}}$	Internal velocity state of command filter
v_{I}	Equivalent of $\boldsymbol{v}_{\text{I}}$ on a single axis
$\boldsymbol{v}_{\max} = [v_{\max x}, v_{\max y}, v_{\max z}]^{\text{T}}$	Maximum velocity in $\mathcal{G}$
$v_{\max}$	Equivalent of $\boldsymbol{v}_{\max}$ on a single axis
$\boldsymbol{v}_{\min} = [v_{\min x}, v_{\min y}, v_{\min z}]^{\text{T}}$	Minimum velocity in $\mathcal{G}$
$v_{\min}$	Equivalent of $\boldsymbol{v}_{\min}$ on a single axis
$\boldsymbol{v}_{\text{ref}} = [v_{\text{ref}\,x}, v_{\text{ref}\,y}, v_{\text{ref}\,z}]^{\text{T}}$	Vehicle velocity reference in $\mathcal{G}$
v_{ref}	Equivalent of $\boldsymbol{v}_{\text{ref}}$ on a specific axis
v_{reach}	Reachable velocity after VI phase
v_{rft}	Filtered velocity command
$V_{\text{b}} = [\varpi_0, \varpi_1, \varpi_2, ...]$	Knot vector of a general B-spline
V_{c}	Closing velocity
V_{knot}	Knot vector of clamped cubic B-spline
w_{g}	Jerk weight factor
$\{\mathcal{W}\} : \{x_{\text{W}}^{\vec{}}, y_{\text{W}}^{\vec{}}, z_{\text{W}}^{\vec{}}\}$	Waypoint line coordinate
$\boldsymbol{x}_{\text{q}}$	Rotorcraft's non-translational innerloop states

Greek variables

α	Factor converting path parameter s to real time t
ϵ_{t}	Bounded error for trajectory tracking
η_{c}	Lateral acceleration reference
$\boldsymbol{\eta}_i$	Control point of clamped 2nd order NUBS, $i \in \mathbb{N}$
η_i	Equivalent of $\boldsymbol{\eta}_i$ on a single axis, $i \in \mathbb{N}$
$\boldsymbol{\gamma}_i$	Control point of clamped 1st order NUBS, $i \in \mathbb{N}$
γ_i	Equivalent of $\boldsymbol{\gamma}_i$ on a single axis, $i \in \mathbb{N}$
$\Gamma_{i,k}$	Base function for kth order clamped B-spline
λ_{LOS}	Angle between heading and LOS
$\Lambda_{\text{s}} = [\gamma_0, \gamma_1 \dots]^{\text{T}}$	Control vector of 1st order NUBS on a single axis
ω_{n}	Natural frequency

ψ_{ref}	Yaw angle reference
$\boldsymbol{\tau}_i$	Control point of clamped cubic NUBS, $i \in \mathbb{N}$
τ_i	Equivalent of $\boldsymbol{\tau}_i$ on a single axis, $i \in \mathbb{N}$
τ_i^k	Control point of kth order on a single axis, $i, k \in \mathbb{N}$
$\Theta = [\phi, \theta, \psi]^{\text{T}}$	Euler angles
Υ_i	Time duration between two adjecent members of D_{time}
ϖ	Path parameter of a general B-spline
ζ	Damping ratio

Acronyms

1D	One Dimension
2D	Two Dimensional
2.5D	Semi Three Dimensional
3D	Three Dimensional
AC	Acceleration constant
AD	Acceleration decreasing
AI	Acceleration increasing
BIT	Batch Informed Tree
CNC	Computer Numerical Control
CPU	Central Processing Unit
CT	Current Trajectory
DOF	Degree of Freedom
ET	Emergent Trajectory
GC	Ground Control
GCS	Ground Control Station
GNC	Guidance Navigation Control
GPS	Global Positioning System
GUI	Graphical User Interface
HMI	Human-Machine Interface
LOS	Line Of Sight
LOT	Line Of Track
LTI	Linear Time Invariant

MGCS	Mission Guidance Control System
MPC	Model Predictive Control
MPEPC	Model predictive Equilibrium Point Control
NDI	Nonlinear Dynamic Inversion
NUBS	Normalized Uniform B-spline
PD	Positive Definite
POI	Point Of Interest
PT	Possible Trajectory
RAM	Random-access Memory
RC	Radio Control
RHC	Receding Horizon Control
ROI	Region Of Interest
ROM	Read-only Memory
RPT	Robust Perfect Tracking
RRT	Rapid Random Tree
RUAV	Rotorcraft Unmanned Aerial Vehicle
SLAM	Simultaneous Localization and Mapping
SMC	Sliding Mode Control
SPD	Semi-positive Definite
TCP	Transmission Control Protocol
TPBVP	Two Point Boundary Value Problem
UAV	Unmanned Aerial Vehicle
UV	Unmanned Vehicle
UWB	Ultra-wideband
VC	Velocity constant
VD	Velocity decreasing
VFH	Vector Field Histogram
VI	Velocity increasing
VTOL	Vertical Take Off and Landing

Chapter 1

Introduction

Unmanned system, since its creation, serves to free human beings from hazardous, repetitive and exhaustive works such as equipment maintenance, industrial inspection, and disaster search-rescue. The rapid evolution of mechanical hardware, communication equipment, sensing technology and artificial intelligence is making impressive progress in improving our daily life through automation. The recent development of Unmanned Aerial Vehicle (UAV), especially Rotorcraft Unmanned Aerial Vehicle (RUAV) further fills up the blank in the unmanned system family. With its growing popularity, various challenges emerge due to the demands of more complicated tasks. Among the presented issues, a reliable, efficient and intelligent Mission Guidance Control System (MGCS) is assigned with high priority. In this thesis, the author proposed and implemented a trajectory generation based guidance and control system for RUAV to handle various tasks from simple path following to navigation in a complex environment. The proposed structure has been successfully implemented on multiple RUAV platforms ranging from low-cost quad-rotor to helicopters with complex dynamics.

In this chapter, an overview of the development of guidance technology and RUAV is presented. In Section 1.1, guidance of airborne systems is reviewed in detail. Section 1.2 introduces different aspects of RUAV from platforms to algorithms. Finally, the contribution of the author's work is covered in Section 1.3.

1.1 An Introduction to Guidance of Airborne Systems

Guidance remains a central problem of controlling air vehicles since their first appearance in history. Nowadays, it is usually mentioned as a part of the Guidance Navigation Control (GNC) system which frequently appears on modern autopilots, missiles, spacecraft and auto cars. Technically, guidance commonly refers to the determination of a path or trajectory that leads the vehicle from current state to the desired state [1, 2, 3]. Navigation is to determine the current location, velocity, attitude and other guidance-related state variables. And control means to manipulate the vehicle's actuator to track the guidance commands. A general GNC structure is depicted in Figure 1.1. The measurement unit consists of various sensors, which are responsible for environment perception and vehicle monitoring. The navigation module utilizes these data to estimate the vehicle's states and reconstruct the surrounding environment. The information is then processed by the guidance module to generate reference command for vehicle's controllers. And finally, the controllers respond by actuating the vehicle to track the desired path.

The burgeoning development of the guidance system from the 1950s to 1980s led to significant progress in modern control theory. Especially during the Apollo space program, many guidance and control theories have been proven great successful by numerous engineering practices. Recently, the rise of Unmanned Vehicles (UV), especially UAVs, again proposes great challenges to these established methods. For UAVs, their guidance systems require a higher level of intelligence and faster response to incoming events, due to the lack of human pilots. The new scenario that is not considered in previous work includes guidance with collision avoidance, the guidance of UAV swarm, guidance under mission management and guidance under semi-auto driving.

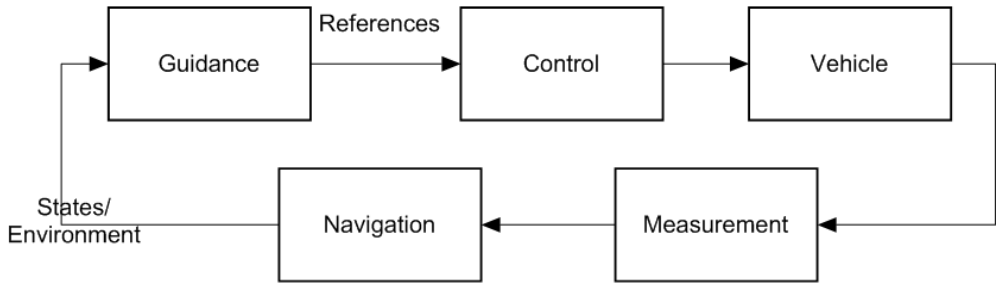


Figure 1.1: The GNC structure

The commonly applied guidance laws for airborne systems is presented in the following sub-sections. They can be categorized into guidance by target aiming, guidance by path following and guidance by trajectory tracking. Moreover, as a critical topic for UV, guidance in the obstacle-strewn environment is also included.

1.1.1 Target Aiming

Target aiming guidance laws are commonly used on missiles and are relatively mature. These guidance laws assume the vehicle travels much faster than the aimed target and pursue the target constantly by manipulating the velocity direction of the vehicle. Commonly used target aiming guidance laws include: pure pursuit guidance [4], relative velocity guidance, and three-point guidance [5]. Variations of the basic methods were reported to have better performance. In [6], fuzzy logic is utilized on top of proportional navigation law. In [7], series of neural network based guidance method are mentioned and compared to their traditional counterparts.

Pure Pursuit

Pure pursuit problem could be illustrated by Figure 1.2. The line segment connecting the vehicle and the target is called the line of sight (LOS). λ_{LOS}, which can be measured by the bearing sensors, is the angle between the vehicle's heading direction and the target. The vehicle is then constrained to march in the direction of LOS until it intercepts the target. This strategy is similar to pursuit-evasion game and often results in a tail chase. The earliest pure pursuit guidance is the proportional navigation method that aligns the vehicle's velocity along the LOS through canceling out $\dot{\lambda}_{\mathrm{LOS}}$. Once $\dot{\lambda}_{\mathrm{LOS}}$ reaches zero, the vehicle will fly to the target directly. The proportional navigation method could be written as:

$$\eta_{\mathrm{c}} = N'_{\mathrm{c}} V_{\mathrm{c}} \dot{\lambda}_{\mathrm{LOS}} \tag{1.1}$$

where η_{c} is the lateral acceleration reference, N'_{c} is a tuning parameter, V_{c} is the closing velocity. By increasing N'_{c}, the vehicle will steer more aggressively. However, this method does not work well for a fast-moving target. To tackle this problem, relative velocity method was developed. Its idea is to utilize the velocity information of the target by aligning the relative vehicle-target velocity along the LOS. Relative velocity

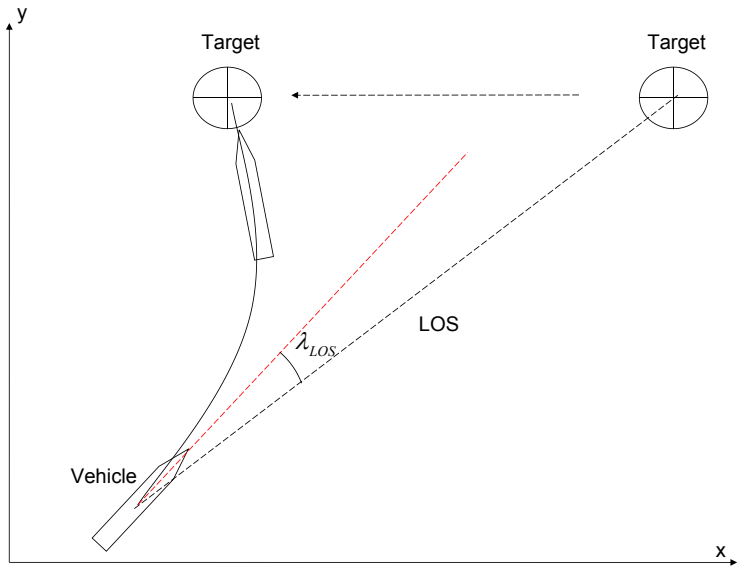


Figure 1.2: The pure pursuit guidance

method has been widely applied on air-air missiles. Also, it has been employed by mariners to avoid collisions on the sea. It has better performance when the vehicle is not significantly faster than the target. However, since it requires the information of the target velocity, the vehicle must be equipped with more avionics, which results in higher cost and increased complexity.

Three Point Guidance

On the other hand, the three-point guidance method utilizes a different strategy by introducing a ground control (GC) point and the line of track (LOT). It lies its root to the surface-air missiles, where a ground station would aim the target with a light beam and the missile would try to ride on the light beam. As illustrated in Figure 1.3, unlike the pure pursuit method, the vehicle does not utilize the LOS information at all. Therefore its onboard bearing sensors could be replaced with more sophisticated equipment at GC. The vehicle's flying path gradually converges to the LOT by changing its heading angle until it intercepts the target.

1.1.2 Path Following

Compared to the target aiming guidance, path following is more complicated. It is to design control laws that could drive the vehicle to reach and follow a geometric

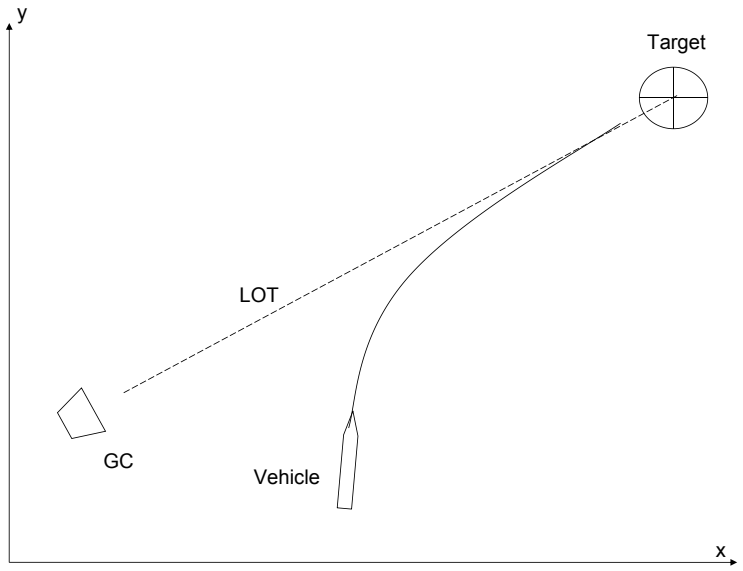


Figure 1.3: The three-point guidance

path while fulfilling the dynamic constraints of the vehicle. Path following guidance is widely adopted by the aircraft, ship, and mobile robot. The basic idea (see Figure 1.4) is to minimize the lateral tracking error from the vehicle to the path while maintaining a forwarding speed in the direction of the path. In Figure.1.4, the point O is the projection point of the vehicle's position on the path, and the formed coordinate is called a Frenet-Serret coordinate. Once the lateral error is reduced to and stabilized at zero, the vehicle will follow and stay on the path permanently. Path following is currently the most popular type of guidance implemented in real life scenario. The representative methods include carrot following algorithm, nonlinear control guidance law [8] and vector field based path following [9]. In recent years, many other more advanced path following algorithms have made noticeable improvements in vehicle performance, but their fundamental ideas are similar to the three representative cases.

Carrot Chasing Algorithm

As a basic approach of following a geometric path reference, carrot chasing has influenced the design and implementation of later algorithms. The idea is to assign a virtual target point (aka. the carrot point) on the path and let the vehicle chase it. In airborne systems, this method is commonly adopted by fixed wing aircraft. Aircraft's mission is usually specified by waypoints which are 3d positions of latitude, longitude, and

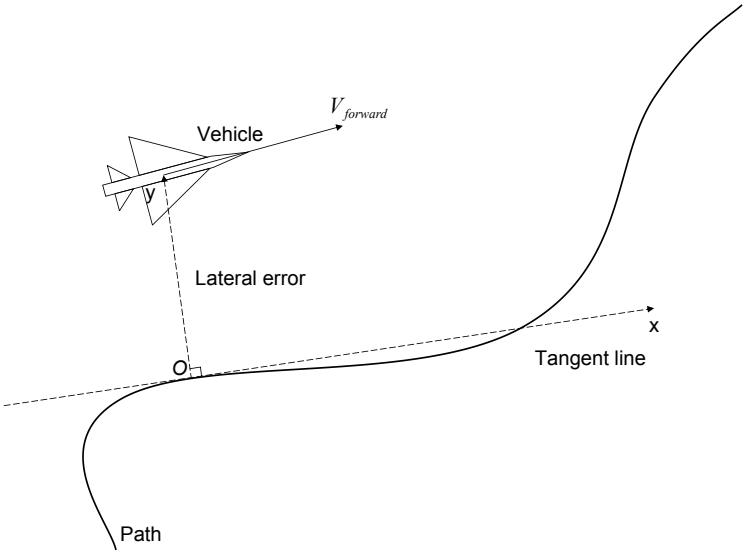


Figure 1.4: The path following guidance

altitude. The aircraft is required to follow the straight line defined by two adjacent waypoints as in Figure 1.5. The key is to localize the carrot position on the straight line path. The first step is to project the vehicle's position onto the waypoint line (point p_{virtual} in Figure 1.5) and the carrot is set to be at distance d_{lead} ahead of p_{virtual}. Then the desired heading direction of the vehicle is determined as the vector point from the vehicle's current position to the carrot. The second step is to utilize a target aiming guidance method, typically the proportional navigation method, to calculate the required lateral acceleration to turn the vehicle towards the desired heading direction. Here d_{lead} is a tuning parameter that determines the behavior of the algorithm as a damping factor. With a smaller d_{lead}, the vehicle moves faster to the waypoint line, but at the cost of longer oscillation. When it is large, the vehicle moves to the waypoint line slowly but more smoothly.

L1 Nonlinear Guidance Law

The d_{lead} in the carrot chasing algorithm depends on various factors such as vehicle speed, lateral track error, and shape of the path. And it must be tuned in realtime to achieve the best performance. In [8], a nonlinear guidance law was first developed to handle this problem, by updating the carrot's position nonlinearly. Again the waypoint line following case is used as an example. In Figure 1.6, a circle with radius r_{L1} around

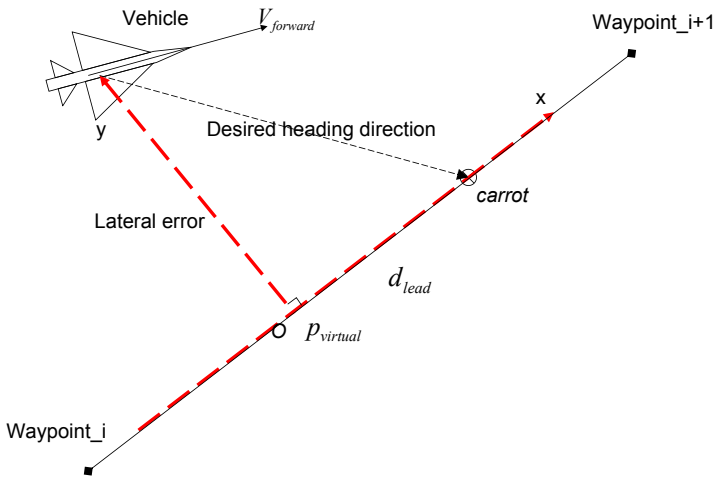


Figure 1.5: Carrot chasing method

the vehicle intercepts the waypoint line at most two times. Based on the waypoint's forwarding direction, the intersection point towards the next waypoint is chosen as the carrot. The algorithm can be proved as Lyapunov stable with a critical damped converging characteristic. Another advantage of this algorithm is that it can follow any geometric path as long as the interception point can be calculated. For special cases where the vehicle is too far away from the path to take any interception, the carrot is set to be the origin of the Frenet-Serret coordinate.

Vector Field Path Following

Different from the carrot chasing based method, the vector field algorithm generates a field of the desired velocity throughout the space, instead of giving a virtual leading target. The field acts like streams of water that push the vehicle towards the path and also in the forward direction of the path [9].

1.1.3 Trajectory Tracking

Though path following based guidance law is widely adopted by the current airborne system, the drawbacks are to be reckoned. Firstly, it is hard for the vehicle to follow a geometric path very precisely since these paths do not consider the dynamic property of the vehicle. Secondly, the timestamp for vehicle's state is unavailable. A vehicle could

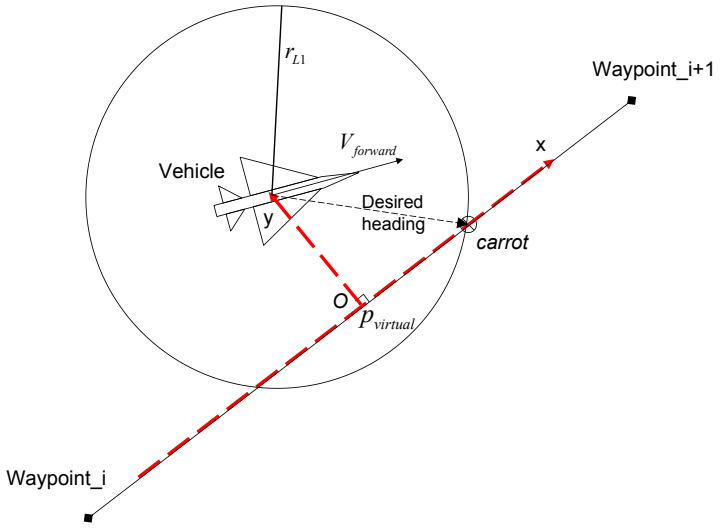


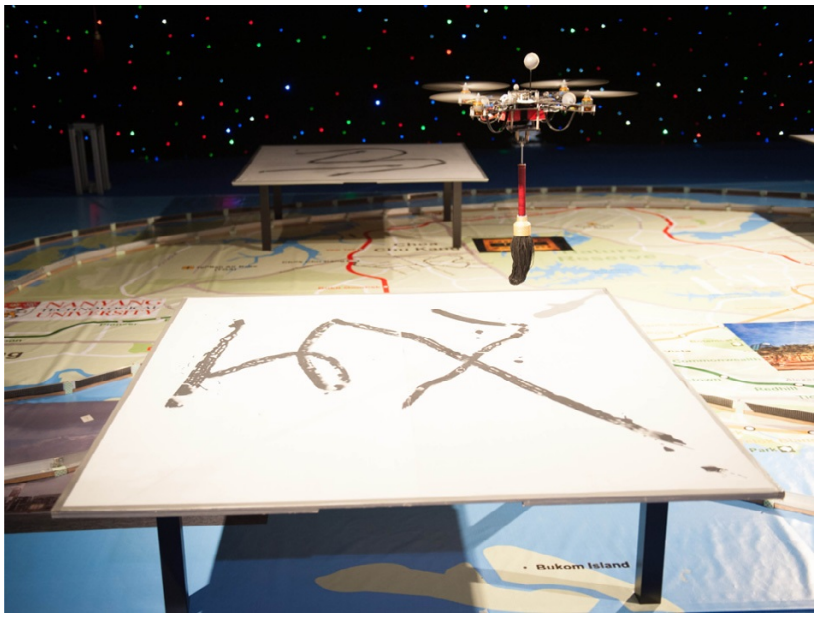
Figure 1.6: L1 nonlinear guidance law

not tell its future position, velocity and other guidance related states easily, the only way is to perform a forward simulation using current measurements. However, such simulation is time-consuming and inaccurate due to the disturbances and measurement noise. These drawbacks introduce significant difficulty for applications like indoor navigation and multi-vehicle formation reconfiguration. To tackle these issues, trajectory tracking based guidance is introduced from the robotics and computer numerical control (CNC) machining society. The philosophy of this method is straightforward:

- Generate the trajectory
- Exam the feasibility of the trajectory
- Track the trajectory as tight as possible

A trajectory is commonly a function of time t which allows its state to be examined at specific time t_{sp}. An excellent tracking performance relies on a well-designed controller in conjunction with a trajectory that resembles some key dynamics of the system. This method has been successfully applied to many RUAVs to perform delicate maneuvers that are difficult to achieve using traditional path following method. For example to fly through narrow circles in high speed [10], to intercept fast moving object [12], to cooperate and construct buildings [11], to perform multi-uav light show (see Figure 1.7(a)) and even to write calligraphy [13] (see Figure 1.7(b)).

(a) MultiUAV Lightshow



(b) UAV Calligraphy

Figure 1.7: Trajectory generation applications

Due to the relative maturity in feedback control theory, the research of this approach concentrates on the trajectory generation algorithm. Three major types of such algorithms are as follows [11]:

1. Path re-parameterization: this method is often used by CNC machines. A pure geometric path is first generated using a class of primitives, then parameterized in time to enforce the dynamic constraints of the vehicle. Frequently used primitives include line segments [14], polynomials [15] and splines [16].

2. Optimization based on differential flatness: many rotor-crafts' models or their controllable approximations are differentially flat about their position and head-

ing [17, 11]. Therefore, the trajectory's feasibility depends on the position's derivatives. A group of methods aiming to minimize these derivatives is hence invented. In [10], minimum snap trajectories are first applied to quad-rotors with excellent performance. In [18], the generated trajectory guided the quad-rotor to fly indoor at $8\,\mathrm{m/s}$, by minimizing a weighted sum of multiple derivatives. A recent progress that pushes the method to online usage which guides the vehicle to navigate in obstacle-strewn environment is reported in [19]. Most of these algorithms utilize a gradient based convex optimization method to optimize the indirect trajectory represented using polynomials. However, in [12], a different strategy was adopted to generate a large number of trajectories efficiently. Then the available trajectory that satisfies all constraints and minimizes a given optimization target was chosen. This method could generate trajectory in real time to allow a modified quad-rotor play tennis.

3. Optimal control based: the last group of methods directly deal with the non-linear dynamics of rotorcrafts. Model predictive control (MPC) strategy is suggested by numerically solving optimization problems [20] or employing the Pontryagin's minimum principle [21].

1.1.4 Guidance in Obstacle Dense Environment

For unmanned vehicles to work safely in general environment, an obstacle avoidance capability is necessary. The early developments focus on ground vehicles and mobile robots. These methods can be differentiated into two categories as the local planner and global planner. The former is responsible for realtime reaction to any environment changes and sensor updates that could lead to a collision. Typical solutions include the potential field algorithms [22], vector field histogram (VFH) method [23] and velocity obstacle strategy [24]. The later is responsible for searching the connectivity information of the environment and providing general guidance for the local planner to prevent being stuck at a local minimum point. The A-star [25] and rapid random tree (RRT) [26] algorithms can be used for this purpose.

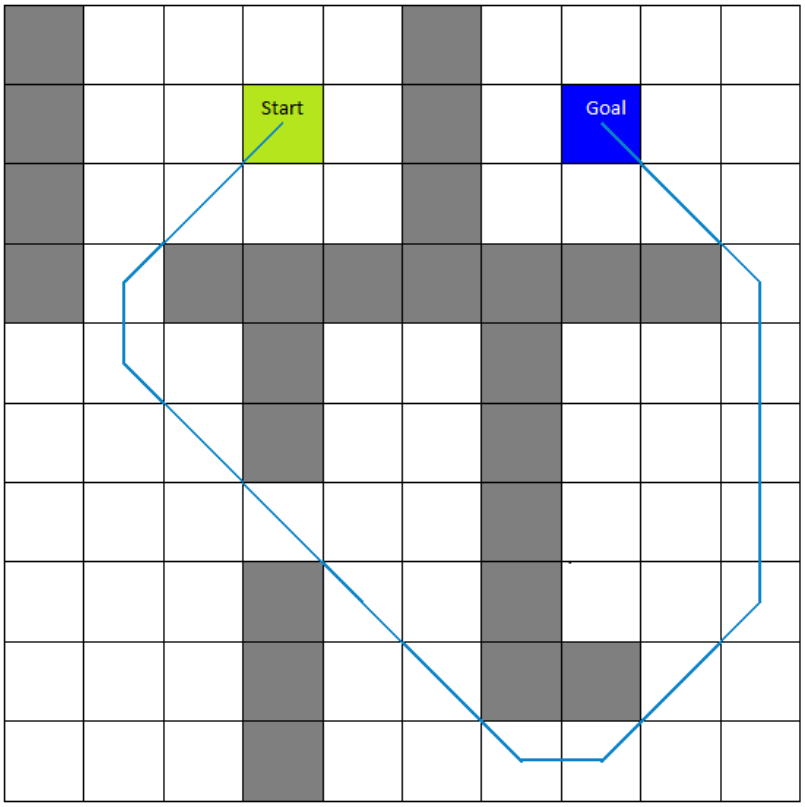


Figure 1.8: Path returned by A-star algorithm

Global Planning Algorithms

Global planner's main task is to analyze the connectivity information of the environment. Though they can also be used to handle the vehicle dynamics, it is hard to maintain algorithm's real time performance. Therefore, most global planners work in the configuration space where vehicle dynamics are largely neglected. Their results are a series of linked line segments which connect from the vehicle's current position to the target. The best example is the connectivity path returned by the A-star algorithm [27] executed in the configuration space (see Figure 1.8). These line segments give a general idea of which direction the vehicle should aim for. The A-star algorithm is developed from the dynamic programming method; it always provides the optimal path if it exists. Many improvements of this algorithm such as D-star [28], D-star lite [29] and jumping point A-star, are developed and utilized in real life scenario. The D-star algorithm which reuses the last planning's information to correct the path when environment changes has been successfully implemented on Mars rover. However, due to their nature of dynamic programming, these algorithms suffer from the curse of dimensionality and

perform poorly in the high dimensional search. On the other hand, the RRT algorithm represents another family of random sampling based technique. Its idea is to lower the dimension of searching space by examining only the randomly sampled points. Its time complexity scales much better, but it might take very long time to converge in an environment with narrow passages. Variations of RRT includes the RRT-star algorithm [30] which brings the optimal property into RRT, anytime-RRT [31] which allows efficient guidance in the unknown environment by reusing the history information, and the more recently reported batch informed tree (BIT*) algorithm [32] which tries to combine the dynamic programming idea with random sampling technique.

Local Planning Algorithms

Local planner's task is to follow the line segment path provided by the global planner while fulfilling the vehicle's dynamic and locally avoid any obstacle. Traditional successful cases of local planners include the VFH [23], potential field algorithm [22], and velocity obstacle strategy [24]. They use the environment information to choose the best control reference directly. For example, the idea of VFH algorithm is to constantly select an obstacle free direction that is close to the line segment path. Its three major steps are:

1. Creation of Cartesian grid: project the range sensors' measurement into a grid map.

2. Creation of polar histogram: construct a one-dimensional polar map based on the Cartesian map centered at the current location of the vehicle. The histogram tells the distance of surrounding obstacles to the vehicle.

3. Selection of candidate peak: the peaks in the polar histogram, that is the directions with far away obstacles, are selected based on their closeness to the desired line segment path.

The potential field algorithm simulates two forces, an attraction force centered on the target, and a repulsion force surrounding obstacles. If the force field is constructed properly, the vehicle will be captured by the target's attraction field while pushed away by obstacle's repulsion field to prevent the collision. However, such field fulfilling the

vehicle's dynamic could be difficult to design. For moving obstacles, velocity obstacle is utilized. It works by constructing a local frame centered around the moving obstacle and removing the relative velocity reference of the vehicle that could lead to a collision. Once eliminating all velocity references that are unsafe, the rest available choices are sorted based on their closeness to the target. The major drawback of these traditional methods is the lack of collision free guarantee, especially with higher order non-linear vehicle models. Recently, the trajectory generation based local planner is proven to be a better solution for fragile vehicles with complex dynamics [19].

1.2 An Overview of Rotorcraft Unmanned Aerial Vehicles (RUAVs)

1.2.1 RUAV platform

Rotorcraft is a sub-category of aircraft where the name comes from its single or multiple rotors that are capable of providing thrust to lift and actuate the vehicle. It possesses the ability of vertical takeoff and landing (VTOL), hovering, low-speed cruising and backward flying which is uncommon in other types of aircraft. These characteristics make it superb in various applications like navigation in cluttered environment [33] and short-range reconnaissance [34].

Like conventional fix-wing aircraft, rotorcrafts utilize the relative motion between air and wing surface to generate lift. Unlike the fix-wing aircraft which creates the relative motion by moving the whole body in the air, a rotorcraft rotates its wing surface around a fixed axis to achieve the same purpose.

With the difference in the number of propellers and their configuration, the four major type of rotorcrafts are classified as in Figure 1.9 (based on [35]). Based on the propellers' topology, rotorcrafts are optimized for different purposes which ultimately determine their utility.

Rotorcrafts are ideal for short-range inspection, disaster monitoring, human rescuing and cargo transportation in the crowded area. However, many of these tasks are either dangerous for human involvement or difficult to deploy human operators. It thus attracts significant interests from the unmanned system society where the focus is to

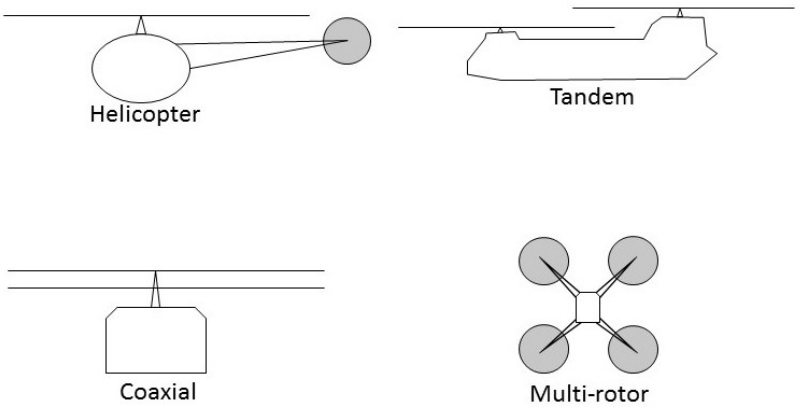


Figure 1.9: Common types of rotorcrafts

(a) PD100 BlackHornet UAV (b) Rapter 90 Helicopter

(c) An electric quad-rotor

Figure 1.10: Some models of RUAVs

produce dedicated hardware equipped with smart algorithms that could perform tasks with less human involvement. One state-of-art example is the PD100 developed by the Proxdynamics which weighs $18\,\mathrm{g}$ and is capable of flying at $5\,\mathrm{m/s}$ up to $25\,\mathrm{min}$ [36]. It is capable of cruising in the wind gust, performing autonomous missions and streaming out the realtime video (see Figure 1.10(a)). A more conventional platform Raptor-90 [37] represents a broad category of traditional RUAV; it is gasoline powered with a total length of 1.4m and a ready-to-flight weight of 4.4kg (see Figure 1.10(b)).

Despite the reduced size and weight, these rotorcrafts suffer from inconvenience in maintenance and repairing. In fact, the conventional helicopter, coaxial, tandem and synchropter configuration designs are commonly seen in larger vehicles due to the requirement of complex mechanical structures. Contrarily, multi-rotors, like the quad-rotors are usually designed for small size battery-powered UAVs thanks to their simple machinery and compatibility to electric motors (see Figure 1.10(c)). Based on the number and configuration of rotors, multi-rotor can be further classified as in Figure 1.11. A rule of thumb on the multi-rotor vehicle is: the smaller the propeller, the better the controllability but the lower the efficiency. Thus multi-rotor designers must maintain a balance between control performance and in-air endurance.

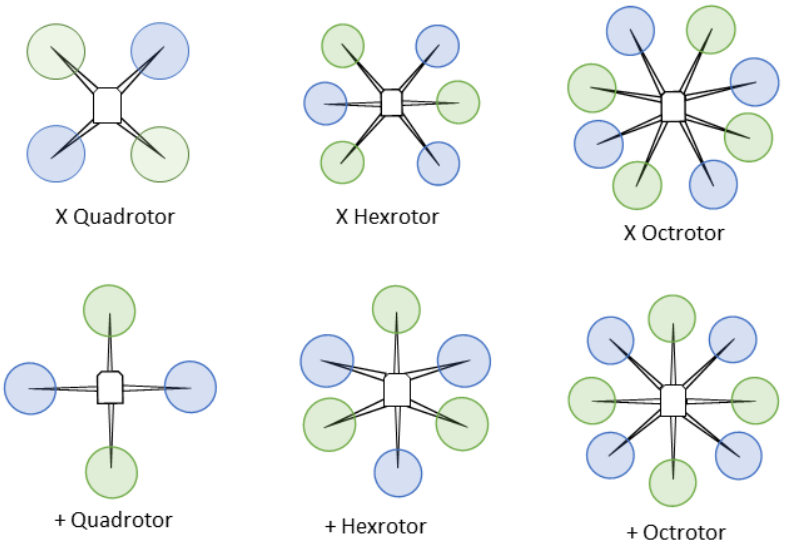


Figure 1.11: Different configurations of common multi-copters
Green circle: rotate in clockwise, blue circle: rotate in anti-clockwise

1.2.2 RUAV Navigation

Unlike ground vehicles, RUAVs lack the direct sensors like the odometer to report its velocity and position, it requires a dedicated navigation module to perform even simple missions like hovering or point-to-point flight. Here, navigation refers to the determination of the vehicle's position, velocity as well as its attitude. The navigation module is fused with other onboard sensors to provide estimation on vehicle states. Despite the variations among applications, the navigation methods of a RUAV can be classified as the following categories:

1. Global positioning system (GPS) based. The most common way of localizing an air-vehicle is to utilize a GPS module (see Figure 1.12(a)). It provides reasonably accurate position measurement, and the velocity can be estimated by filtering with other measurable states. The advantages of GPS are its availability throughout the world, and it requires no external setup. However, high-quality GPS signals can only be received in open fields, which brings problem for indoor, forest or urban applications.

2. Optical flow based. Another commonly used navigation method is to utilize an optical flow sensor to get velocity measurement (see Figure 1.12(b)). The position measurement is acquired by integrating the velocity history. Consuming

(a) A Prallax GPS module (b) A Px4flow smart camera (c) A Hokuyo laser scanner (d) A Bumblebee stereo camera

(e) A Microsoft kinect camera (f) A vicon camera (g) A UWB positioning system

Figure 1.12: Sensors used during RUAV navigation

little computational power and being effective in nearly all scenario make it an excellent companion to GPS module. The major disadvantage of this method is the drifting in position after long-time integration of noisy velocity measurement.

3. Simultaneous Localization and Mapping (SLAM) based. SLAM, being a hot researched topic on its own, is also intensively utilized as navigation method for RUAV. On top of acquiring vehicle states, SLAM also provides the environment information which can be used in planning and guidance phases. Therefore, SLAM requires more sophisticated sensors like laser scanner (see Figure 1.12(c)), stereo camera (see Figure 1.12(d)) and RGBD camera (see Figure 1.12(e)). However, the complex SLAM algorithm usually consumes a large amount of computational power.

4. External aided. Finally, for many researchers, an additional positioning system such as Vicon (see Figure 1.12(f)) or Ultra-wideband (UWB) (see Figure 1.12(g)) are used to measure the vehicles' status. However, these types of equipment are not portable and require complicated calibration procedure before each setup. Though it provides the highest level of accuracy, it is used for research and development purposes mainly.

1.2.3 RUAV Guidance

Large size rotorcrafts traditionally adopt the most mature path following based guidance method. However, due to the requirements of full automation and utilizing the agile dynamics, trajectory tracking methods are becoming popular, especially among small size RUAVs. Researchers have made a significant effort on widening the possible applications of RUAVs. To demonstrate the performance, RUAVs capable of performing aggressive maneuvers are developed. In [10, 38, 11], the multi-rotors were made to fly through narrow windows, perch on vertical walls and catch flying objects. Another hot topic of RUAV guidance is their cooperation and consensus. For missions like surveillance and area coverage, multiple vehicles significantly increase the efficiency and overall reliability. In [39, 40] both centralized and distributed solutions were applied to solve the formation control problem. Finally, to allow the vehicle to work safely in the general environment, RUAV obstacle avoidance has been intensively studied. Chen and his group [19, 41] presented a method of navigating through unknown cluttered environments by efficiently solving the convex optimization problem. The author also implemented light-weight algorithms [33] to make RUAV flying through the forest like environment by online decomposing the complex guidance problem into a series of two point boundary value problem (TPBVP).

1.3 Contributions

In this chapter, various guidance techniques utilized by the airborne system have been investigated and reviewed. Traditional methods like target based guidance and path following method have been studied by reviewing representative algorithms. Their targeting vehicle groups, advantages, and disadvantages have been discussed. Further, new approaches based on trajectory generation have been given extra emphasis due to their compatibility to the RUAVs. Rotorcraft platforms are introduced based on their working principal and classification. Multi-rotors receive special notice since they are the major RUAV type in the industry. As a part of the GNC system, RUAV navigation is studied since their performance determines the rest of the system design. Finally, developing in RUAV guidance technology is reviewed. Various aspects like aggressive maneuvering, swarming and obstacle-strewn environment navigation are covered.

Major contributions of this thesis can be categorized into three topics. Namely the implementation of a trajectory-based MGCS, the reference generation and guidance algorithms utilized by RUAVs, and an RUAV capable of flying in obstacle-strewn environment. They are covered in dedicated chapters with details. The organization of the thesis is as follows.

Chapter 2 discusses the system design and implementation of MGCS. The work covers developing a user-vehicle interface to allow mission management, introducing various global planning techniques for realtime applications and implementing a cascaded controller for better tracking performance. Such a system is highly modularized which makes it an ideal testbed for different sensing, navigation, guidance and control methods.

Chapter 3 and 4 introduce several trajectory generation and guidance algorithms. One off line method utilizes the B-spline as path primitives and optimizes the weighted derivatives of the position trajectory. There are also algorithms utilize the sliding mode control (SMC) principal to generate dynamically feasible reference in realtime. Target tracking and path following methods based on coordinate projection are also covered.

Chapter 5 discusses the developing and implementation of the trajectory based guidance system that allows an octo-rotor to fly in GPS-denied and obstacle-strewn environment. This method decomposes the complex trajectory optimization problem into a

series of TPBVPs. The case of multiple vehicles equipped with this approach forming leader follower formation while marching in obstacle-strewn environment is also studied.

Finally, Chapter 6 draws a conclusion on this thesis and the idea of trajectory generation based guidance. Its advantages, disadvantages, possible improvements and future research topics are analyzed and discussed.

Chapter 2

System Design and Implementation

In this chapter, the author designed and implemented an MGCS dedicated for trajectory-based guidance. The system is utilized in various projects and studies. The task of our RUAV's software system is to:

- Generate its trajectory based on mission and environment.
- Perform accurate trajectory tracking.
- Read command from the human operator.

To fulfill these requirements, a structure in Figure 2.1 is proposed. Though an unmanned system it is, the flow starts with a human operator and a Human-Machine Interface (HMI) which is responsible for assigning high-level missions, such as mapping a particular area, carrying a payload to rendezvous point or cooperating with other vehicles to monitor a target. Then the mission manager decomposes these high-level tasks with pre-set vehicle behaviors. For example, to reconstruct an environment model of some user-defined areas, the task will be decomposed by assigning suitable waypoints and generating corresponding flight paths which would essentially scan through the whole working area. By following the paths and taking photos along the way, image footages can be easily obtained for later vision-based reconstruction.

Once the mission is clearly defined, the guidance module handles the problem of piloting the vehicle from current location to the point of interest (POI). A two-level

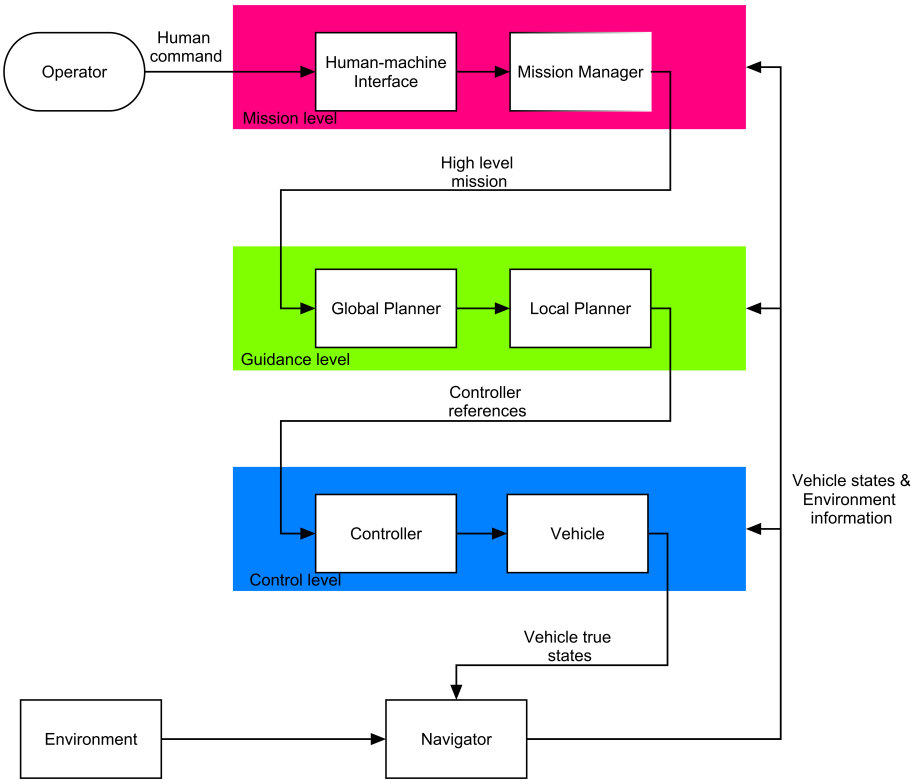


Figure 2.1: RUAV system for trajectory generation based guidance

planning process is adopted to deal with complex tasks such as exploration or navigation in the obstacle-strewn environment. The final output of the guidance module, specifically, the dynamically feasible reference, would be tracked by the lower-level vehicle controller.

Due to the nonlinear dynamics of most rotorcrafts, a two-level controller is commonly adopted. The first level is a transnational controller, which converts the reference trajectory into acceleration commands. The second level controller, which is usually referred as attitude controller, stabilizes the vehicle and tracks the generated acceleration commands.

For the controller to work properly, measurements are provided by a navigator unit which performs sensor fusion. Further, the vehicle states and environmental information also affect the planner and human operator's decision. The design and implementation of the mission, guidance, and control layer are discussed in their dedicated sections with detail. The similar system structure is reported to be utilized successfully by various type of RUAVs such as octo-rotors [33], helicopters [37] and coaxial-rotors [42].

2.1 Mission Level

The mission level consists of an HMI and the mission manager. In our implementation, they are combined into a ground control station (GCS) which handles the user interaction through a graphical user interface (GUI) and manages each vehicle in the underlying data layer.

2.1.1 Software Architecture

The GCS software consists of four layers, modules in the same layer are interchangeable to provide maximum customization capability. The architect of the software is shown in Figure 2.2. A top down tree shape design is adopted for handling connection to multiple vehicles. A centralized manager interface is maintained for better user experience. Similar design could be found in [43] which served as a base for the author's development. The four layers in Figure 2.2, namely the link layer, the protocol layer, the data layer and the GUI layer are discussed as follows.

- Link layer: The link layer provides drivers and interfaces for accessing different communication hardwares. Vehicles utilize a various type of data links, like Ethernet devices or serial devices. For each link, a dedicated thread can be assigned to prevent the process from jamming while the link is performing synchronized operation. The outputs from the link layer are raw binary arrays; no external parsing is done at this level. Time-consuming operations such as parsing are removed from the link layer since many hardware devices have limited buffer size that needs to be frequently cleaned to prevent buffer overflow. A virtual buffer is created at the protocol level to keep track of all received data.

- Protocol layer: the protocol layer holds the parser and decoder for various communication protocols which include low bandwidth heartbeat packages as well as realtime stream flows. During our practice, all low bandwidth data parsing shares the same thread while tasks like video decoding have their dedicated parsing threads. The outputs from the protocol layer are data blocks like image frames or vehicle state reports.

- Data layer: the data layer stores and manages the decoded data from vehicles.

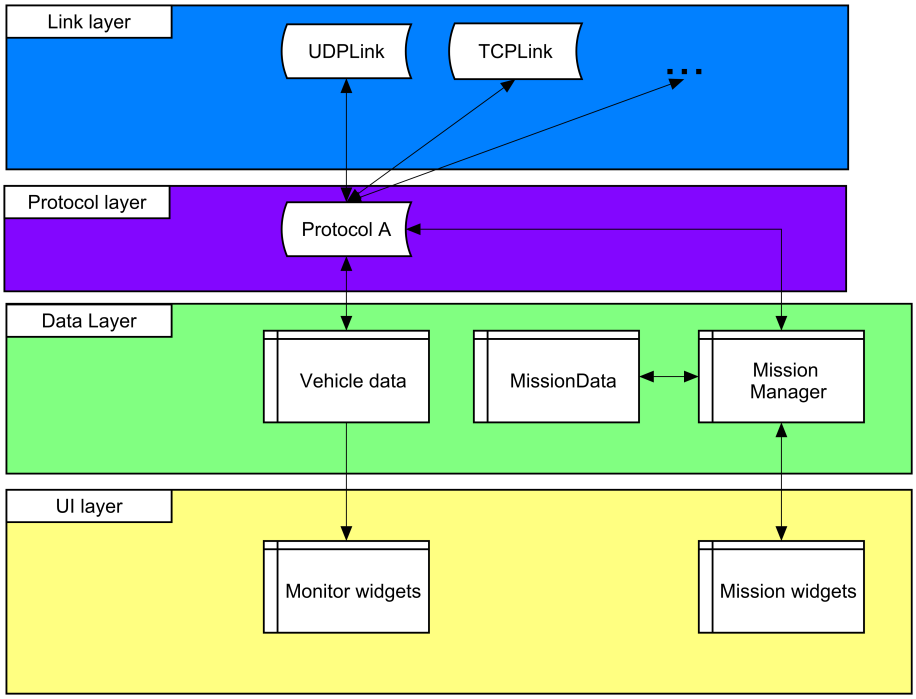


Figure 2.2: GCS software architecture

The data is categorized into state data and mission data. For state data, their main purpose is for the user to monitor the performance of the vehicle such as location, battery voltage, and temperature. Therefore, no interaction is required with the state data. The mission data, on the other hand, is different. The user might create, edit or cancel a mission through the user interface. Since the vehicle is only capable of executing a series of mission primitives, it is necessary for the mission manager to translate the human defined task into mission primitives that could be understood by the vehicle. Most human defined tasks could be decomposed into combinations of mission primitives statically. However, for operations like exploring and searching, algorithms in [44] can be utilized.

- User interface: the last layer is a GUI, which provides the terminal for human-computer interaction. Its widgets can be classified into two types. One is the monitoring type which receives information only and includes flight instrument board, video player, and warning message windows. Another is the task management type which requires communication with the mission manager. An overview of the GUI is shown in Figure 2.3.

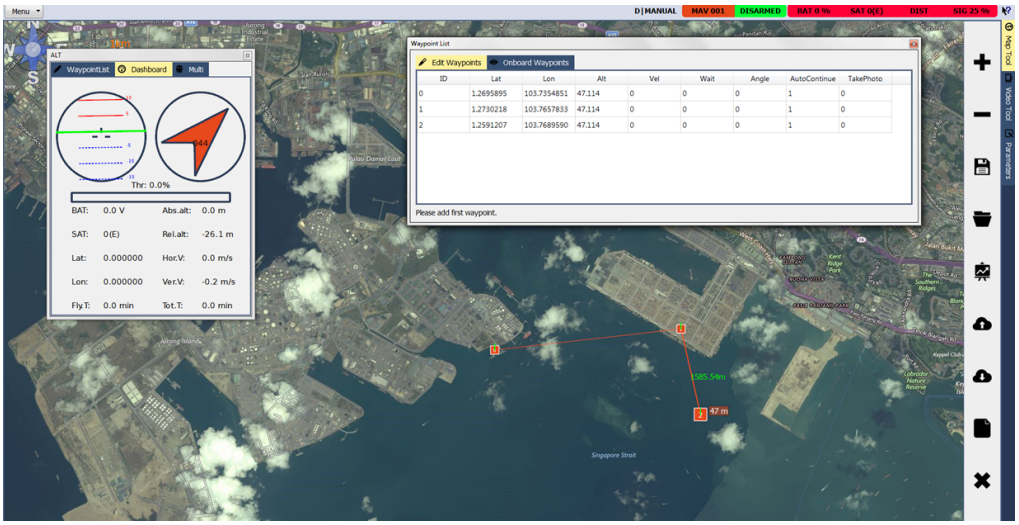


Figure 2.3: GCS user interface

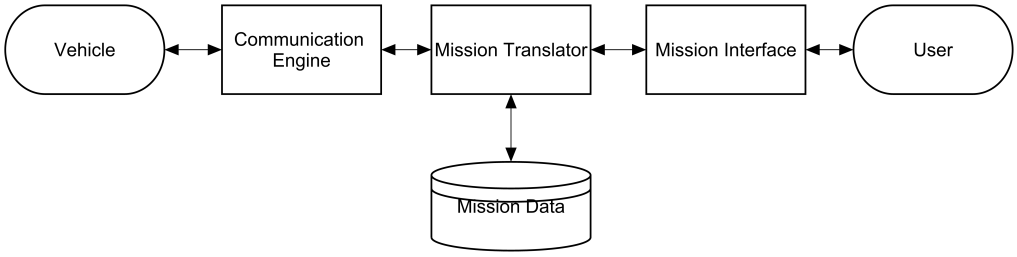


Figure 2.4: GCS mission management system

2.1.2 Mission Management System

Besides monitoring the vehicle and its sensor data, the most important function for the GCS is to provide the mission management service. In our implementation, a waypoint based mission management system is developed based on [43]. That is, all human defined tasks such as flight path, the region of interest (ROI), target to intercept and object to follow are all ultimately described by a series of waypoints. Such a mission management system consists of three main components (see Figure 2.4): the communication engine, the mission translator, and the mission interface.

- Communication Engine: mission data is different from others in a way that they need to be kept in order. Missing even one package in mission data might alter the mission and render the whole task useless. Therefore, a retransmission mechanism is needed to guarantee an ordered data transfer. Here, a technique similar to Transmission Control Protocol (TCP) is adopted from [43]. When the GCS initiates a mission transmission talk, it sends a handshaking message with the

number of mission elements to the vehicle. Once the vehicle received the message, it begins to request each mission element in order. Only when the receiving of mission element i is confirmed, it will start to request element $i + 1$. If no response is received after requesting element i, a retransmission request for that element is sent.

- Mission translator: the user defined task consists of three geometry primitives as the path, region, and target. The mission translator is responsible for decomposing them into waypoints. For example, paths can be represented by endpoints of line segments or control points of splines. Regions are defined by connecting multiple paths to form a closed shape. And the target is a single waypoint possibly on the move. Further, each waypoint also describes the functional purpose of the geometry primitive. For example, a region could be either an ROI or a forbidden zone, a flight path can be either a mission path or an emergent home return path. In this way, the whole mission is translated into a list of waypoints that could be understood by the vehicle's onboard computer.

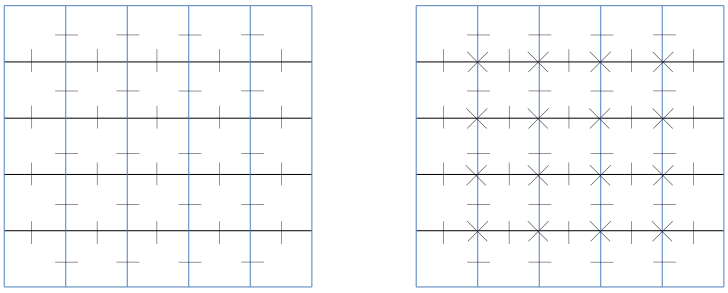
- Mission interface: the mission can be created, edited or canceled by clicking or dragging on the map. The detailed mission parameters can be set through an excel like table interface. An example of mission interaction is given in Figure 2.5. The red waypoints are the flight path point defining a desired flight path. The blue waypoints are the return-home waypoints defining the path in auto-return-home mode. The green polygon is the safe-operation area, the vehicle cannot leave this area under any mode. And finally, the orange curve line denotes the current trajectory tracked by the vehicle.

Figure 2.5: GUI for mission editing

2.2 Guidance Level

As a trajectory based guidance system, it takes inputs from the mission manager which is a list of waypoints and output reference for the vehicle's controller. To solve the more complex problem, a high-level global planner is cascaded with the trajectory generator. In fact, human uses this decoupled planning strategy all the time. When hikers are traveling in the field, they make a global plan by studying the map. For the rest of time, they focus on walking, observing the surroundings and avoiding obstacles. The global plan is not reconsidered unless the environment changes tremendously such as the breaking down of a bridge; or the high-level task shifts. This setup serves to conserve an enormous amount of computational time. It also helps to increase system robustness as most global planners are dynamic programming based and are sensitive to noises.

In our implementation, the global planner is a geometric pathfinder, and the local planner is a trajectory generator. The pathfinder provides a geometric path as connected line segments, and local planner generates trajectories to follow the geometric path while avoiding the obstacles.



(a) 4-connected tile map graph (b) 8-connected tile map graph

Figure 2.6: A graph of 2D tile map

2.2.1 Global Planner

The task of the global planner is to: find a series of line segments that connect the vehicle to its target position. Two types of path searching algorithm have been covered in this thesis; namely, the graph searching based and random sampling based. Noticeably, these searching algorithms perform search routine in the configuration space of the vehicle. Though they could be used for state space searching while considering the vehicle's dynamics, it is time-consuming and defeats the purpose of decoupled planning.

Graph Searching

Graph searching algorithm is a well-established family for pathfinding. The first step is to transfer the configuration space into a graph which is a set of vertices connected by edges. A 2 Dimensional (2D) tile map (see Figure 2.6) can be considered as a graph and it is commonly used for the path searching. The short black lines in Figure 2.6 denote the edges that connecting two tiles. In 4-connected map, a tile is connected with its four neighbors, namely, up, down, left and right. In 8-connected map, a tile is connected to four more neighbors at the corners. The most representative example of graph searching pathfinder, A-star algorithm [27], is introduced in the 8-connected graph as follows. In A-star, from the starting point, tiles will be assigned with cost $f_{\mathrm{A}*}(n) = g_{\mathrm{A}*}(n) + h_{\mathrm{A}*}(n)$, where $g_{\mathrm{A}*}(n)$ represents the shortest cost of the path from the starting point to vertex n, and $h_{\mathrm{A}*}(n)$ represents the estimated cost from vertex n to the goal. For each cycle, it examines the vertex with lowest $f_{\mathrm{A}*}(n)$. The pseudo code of A-star is given as:

Algorithm 1 A-star algorithm

```
1: Input: START, GOAL
2: Output: Shortest path from START to GOAL
3: OPEN: priority queue containing START
4: CLOSED: empty list
5: c ← lowest cost item in OPEN
6: move c from OPEN to CLOSED
7: while c is not GOAL do
8:     for each neighbor n of c: do
9:         cost = g_A*(c) + moveCost(c, n)
10:        if n already in OPEN and cost < g_A*(n) then
11:            g_A*(n) ← cost
12:            f_A*(n) ← g_A*(n) + h_A*(n)
13:            n.parent ← c
14:        if n not in OPEN and n not in CLOSED then
15:            g_A*(n) ← cost
16:            f_A*(n) ← g_A*(n) + h_A*(n)
17:            add n to OPEN
18:            n.parent ← c
19:        c ← lowest cost item in OPEN
20:        move c from OPEN to CLOSED
```

The core of A-star algorithm is two sets, OPEN and CLOSED. The OPEN set contains vertexes prepared for examining. The CLOSED set contains the vertexes need no examination (see Figure 2.7). The OPEN set first contains the $START$ vertex while the CLOSED is an empty list. For each cycle, the vertex with lowest cost $f_{\mathrm{A}*}$ (vertex c) is pulled from the OPEN set until it happens to be the $GOAL$. Then c is moved from OPEN into the CLOSED. All the neighbors connected to c is now checked. Its temporary cost value is calculated as $cost = g_{\mathrm{A}*}(c) + \mathrm{moveCost}(c, n)$. There will be two possible conditions. If the n is not in the OPEN set, it is added into it with its parent set to c. If it is already in the OPEN set, we check whether it is a better solution to travel to n through c. If not, nothing is done. Otherwise, the cost value of neighbor n is updated with its parent set to c. The path is constructed reversely by recursively finding parent starting from $GOAL$ until it reaches the $START$ (the blue dash lines in Figure 2.7). During the implementation of A-star algorithm, in order to achieve the reliable and efficient searching, the following aspects are noticeable.

- Selecting of heuristic function $h_{\mathrm{A}*}(c)$: the heuristic function is an estimation of future cost traveling from vertex c to the $GOAL$. It defines the property of the A-star algorithm. To achieve globally optimal results, it is required that the

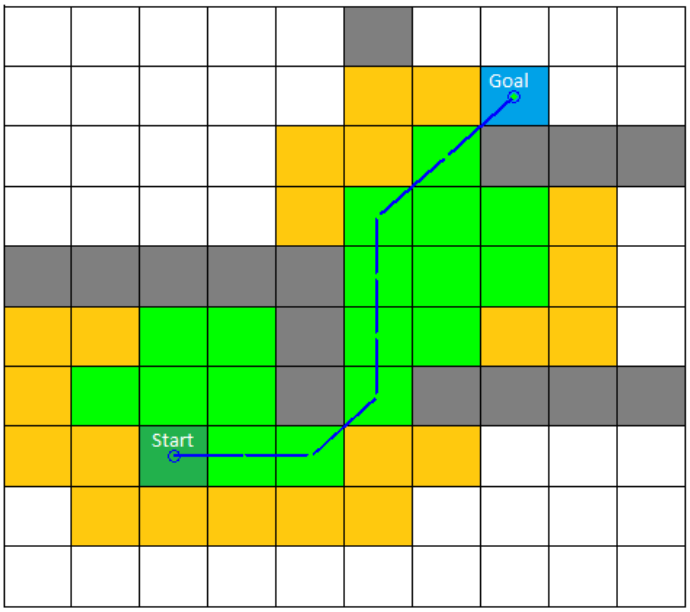


Figure 2.7: A-star path finding from one room to another
The yellow tiles are in OPEN set, and the green tiles are in CLOSED set.

estimated cost is always equal or smaller than the true value. If $h_{\mathrm{A}*}$ is always zero, the algorithm becomes the Dijkstra's algorithm. On the other hand, if $g_{\mathrm{A}*}$ is very small, then the algorithm becomes greedy search. For different tasks, the heuristic function can be varied to improve either speed or optimality.

- Implementation of priority queue: in Algorithm 1, the OPEN set is said to be a priority queue. It needs to return the vertex efficiently with the lowest $f_{\mathrm{A}*}$; a binary heap is used in our implementation with unsorted arrays.

- Introducing of hash: for each vertex, there is a need to check whether it is in the OPEN or CLOSED set. If it is in the OPEN set, the corresponding queue item also needs to be found for possible value modification. A linear search of time complexity $O(n)$ is obviously not desirable. The introducing of a pointer hash map that allows each vertex to point to its counterpart in OPEN set could reduce the time complexity to $O(1)$.

Random Sampling

The graph search method is affected by the curse of dimensionality. That is, the complexity of the algorithm grows exponentially with the dimension of the searching space.

This causes problems for traditional grid based graph search to work efficiently in higher dimensional space. In fact, the A-star algorithm works well for 2D and semi three dimensional (2.5D) path finding with RUAV onboard computers, but full three-dimensional (3D) searching is too computationally intensive. In answer to this problem, a family of randomly sampling based algorithms is developed. An RRT-based algorithm is implemented here for its simplicity. The pseudo code of the basic RRT [26] is given in Algorithm 2. The searching tree T_{search} starts with the root as $START$. For each cycle

Algorithm 2 RRT algorithm

```
Input: START, GOAL
Output: Path from START to GOAL
T_search: tree with one node START
while No node of T_search is in the GOAL region do
    sampling a random point p_rand in the configuration space
    find the node q_n in T_search that is closest to p_rand
    if q_n and p_rand can be connected with a line edge without collision then
        add p_rand into T_search with parent as q_n
reconstruct reverse path from the node in GOAL by recursively visiting parents
```

of the algorithm, a random sample is picked from the space. Then its closest neighbor node on T_{search} is retrieved. If these two points can be connected without collision, the sampled point is added to T_{search} with its parent as the closest neighbor. However, unlike the graph search algorithm, the basic RRT does not generate the optimal path. Though optimal variants of RRT exist [30], its computational cost is rather high for our purpose. Hence, a lightweight algorithm is proposed to improve the optimality of original RRT by making two modifications: the random sampling strategy is changed to a target biased sampling, and post-processing for cutting the optimal branch is introduced. According to simulation experiments, the algorithm works reasonably well in the cluttered environment with most obstacles having a convex shape. The pseudo code of the modified RRT is given in Algorithm 3. For each cycle, a target biased search is added. And for each tree growing procedure, instead of growing the branch with full length, a shortened branch is appended. Nonetheless, due to the lack of optimality, the resulting path consists of many unnecessary zig-zags (red lines in Figure 2.8). To improve the path quality, a post processing is executed to find the shortest track among the zig-zag path. Note the path is represented as a list of nodes start from $START$ and end with $GOAL$, the trimming algorithm is given as

Algorithm 3 Target biased RRT algorithm

Input: $START$, $GOAL$
Output: Path from $START$ to $GOAL$
T_{search}: tree with one node $START$
while No node of T_{search} is in the $GOAL$ region **do**
    sampling a random point p_{rand} in the configuration space
    GrowRRT(p_{rand}, T_{search})
    GrowRRT($GOAL$, T_{search})
reconstruct reverse path from the node in $GOAL$ by recursively visiting parents

---

Function GrowRRT($point, tree$)
find the node q_{n} in $tree$ that is closest to the $point$
grow $tree$ from q_{n} towards $point$ for a distance $dist < \|point - q_{\text{n}}\|$ resulting node q_{g}
if the newly grown branch resulted in a collision **then**
    delete it
else
    add q_{g} into T_{search} with parent as q_{n}

Algorithm 4 Trimming of zig-zag path

Input: $Path$ with N nodes ($N > 1$)
Output: $Path_{\text{trim}}$
$Cost \leftarrow$ a variable size list
$Path(1).g_{\text{trim}} = 0$
$Path(1).parrent \leftarrow NULL$
for n from 2 to N **do**
    $Cost$.clearMembers()
    for i from 1 to $n-1$ **do**
        calculate $distance$ from $Path(i)$ to $Path(n)$
        if connection of $Path(i)$ to $Path(n)$ results in collision **then**
            $Cost(i) = \infty$
        else
            $Cost(i) = distance + Path(i).g_{\text{trim}}$
    $k = \underset{u_k \in [1,n-1]}{\text{argmin}}\ Cost(u_k)$
    $Path(n).g_{\text{trim}} = Cost(k)$
    $Path(n).parrent \leftarrow Path(k)$
generate $Path_{\text{trim}}$ by recursively visiting parent from $Path(N)$

This is basically a dynamic programming process. For each node, it searches for the shortest collision free path that traces back to $START$. The trimmed path is shown as the green lines in Figure 2.8.

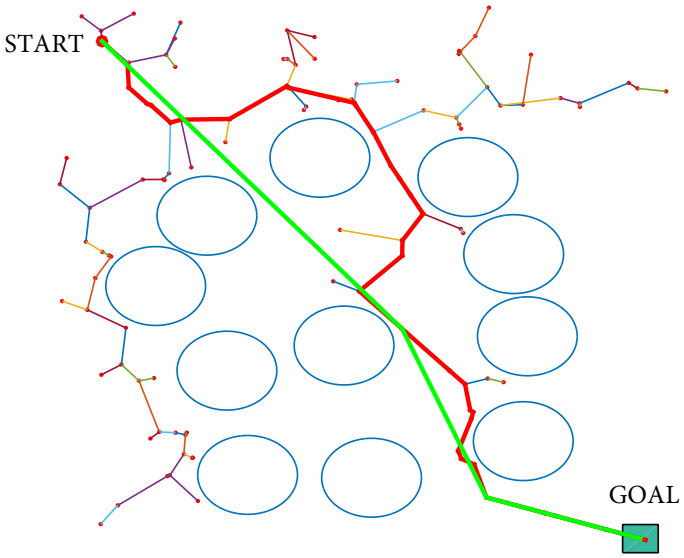


Figure 2.8: RRT path finding in cluttered environment

2.2.2 Local Planner

Following the global planner, the local planner generates open loop trajectories that is consistent with the vehicle's dynamics. Figure 2.9 shows the simulation responses of two different reference signals applied to a quad-rotor's altitude controller. It illustrates how the smoothness of reference signal would affect the final control performance. As shown in the figure, both references have a steady state value of $12\,\mathrm{m}$, i.e., the vehicle is expected to climb to $12\,\mathrm{m}$ position. However, the ramp reference is much smoother than the step one. As a result, though the same linear feedback controller is used to track both types of references, the tracking performance is much better for smooth ramp signal regarding smaller overshoots, shorter settling time, and more importantly, smaller tracking error.

As mentioned in Chapter 1, one benefit for trajectory-based guidance is the reference trajectory can be examined before being adopted. However, if the generated trajectory is not dynamically feasible and cannot be tightly tracked by the vehicle, the trajectory-based guidance then becomes less effective. Step reference is an extreme example of dynamically infeasible trajectory since no currently available vehicle is capable of teleportation. Besides dynamic feasibility, the local planner might bear other requirements such as intercepting a moving target or avoiding obstacles. Therefore, the property of the generated trajectory needs to be examined. Taking obstacle avoidance

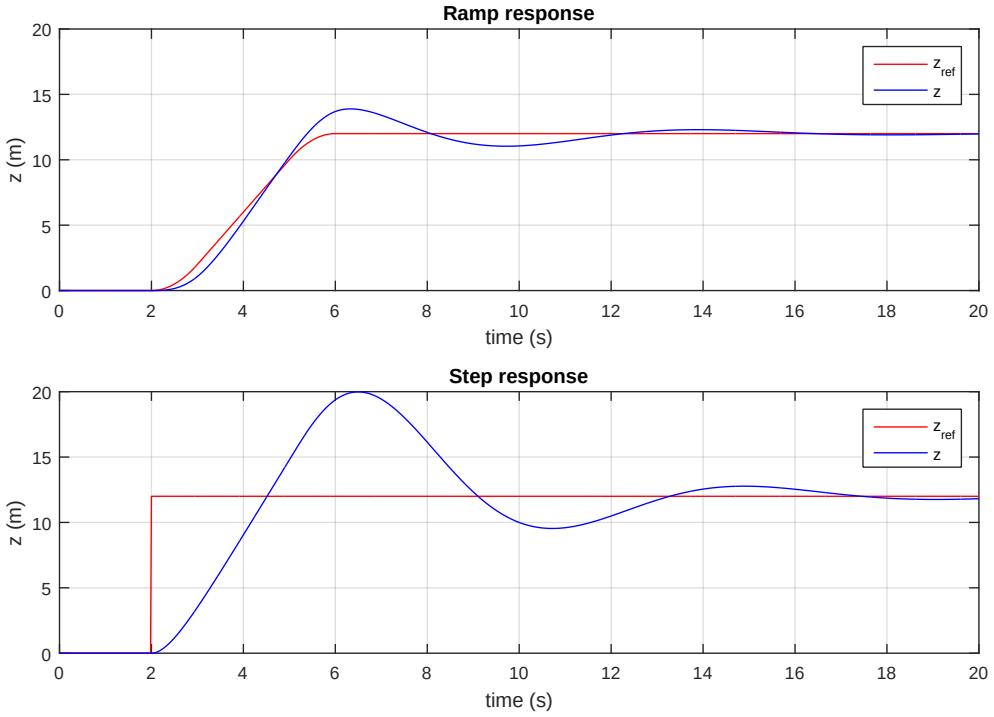


Figure 2.9: Quad-rotor altitude control by tracking different references

as an example, the clearance of the trajectory is examined by assuming an asymptotic converge with a bounded error ϵ_t. Forward simulation is less effective as it is more time consuming. The idea of the generation-examination-tracking process is depicted in Figure 2.10. Trajectory generation is repeatedly executed until one of them passes through the examination module. Finally, another important task for the local planner is to 'follow the lead' of the global planner. In our implementation, the trajectory is made to stay in the vicinity of the global path and march forward along it. Two examples of such behavior are given in Figure 2.11 and Figure 2.12. The first one guides the vehicle following a circling square path. The second one leads vehicle on a 3D global path with minimum jerk trajectory. For these trajectories, they are all constrained on the derivatives of position, which helps to resemble the dynamics of a rotorcraft. The detail of local planners are discussed in Chapter 3 and 4.

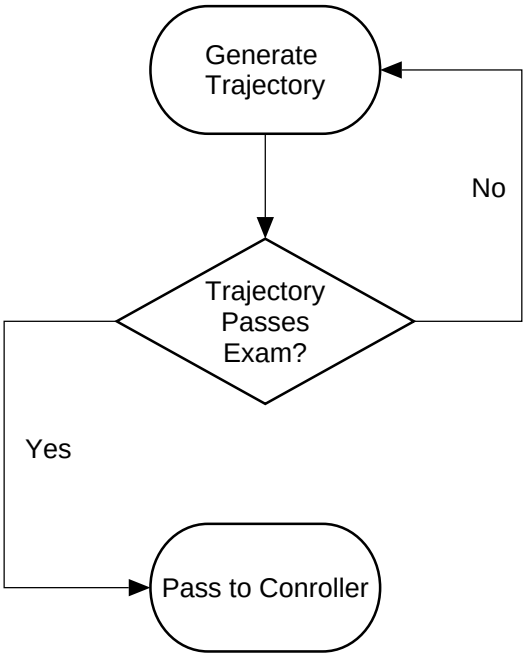
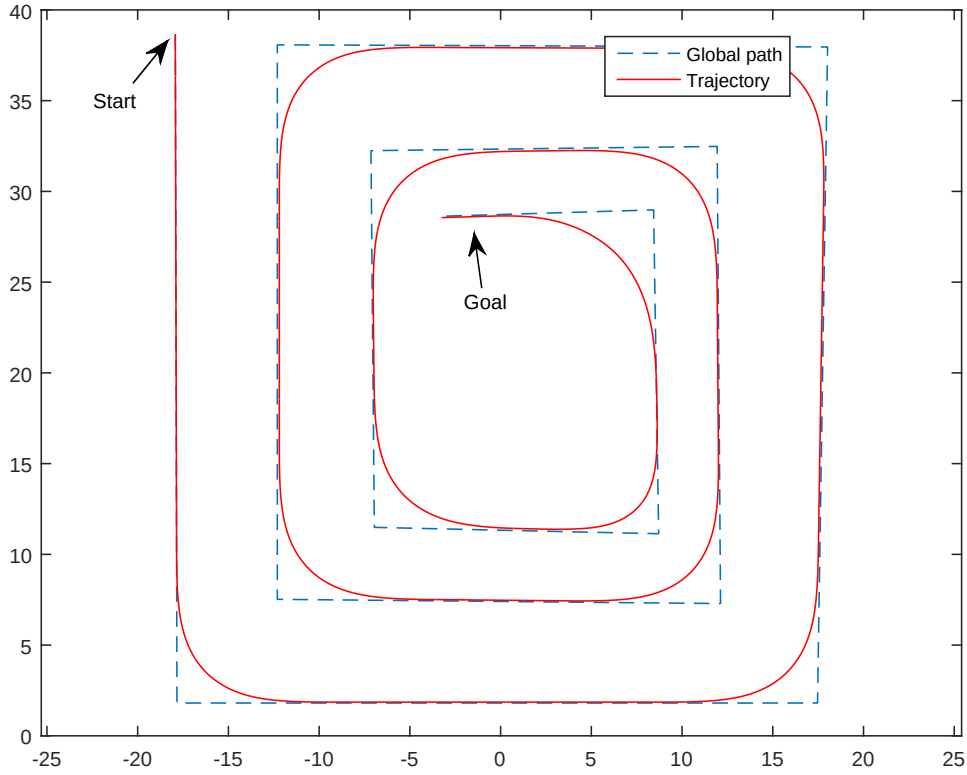
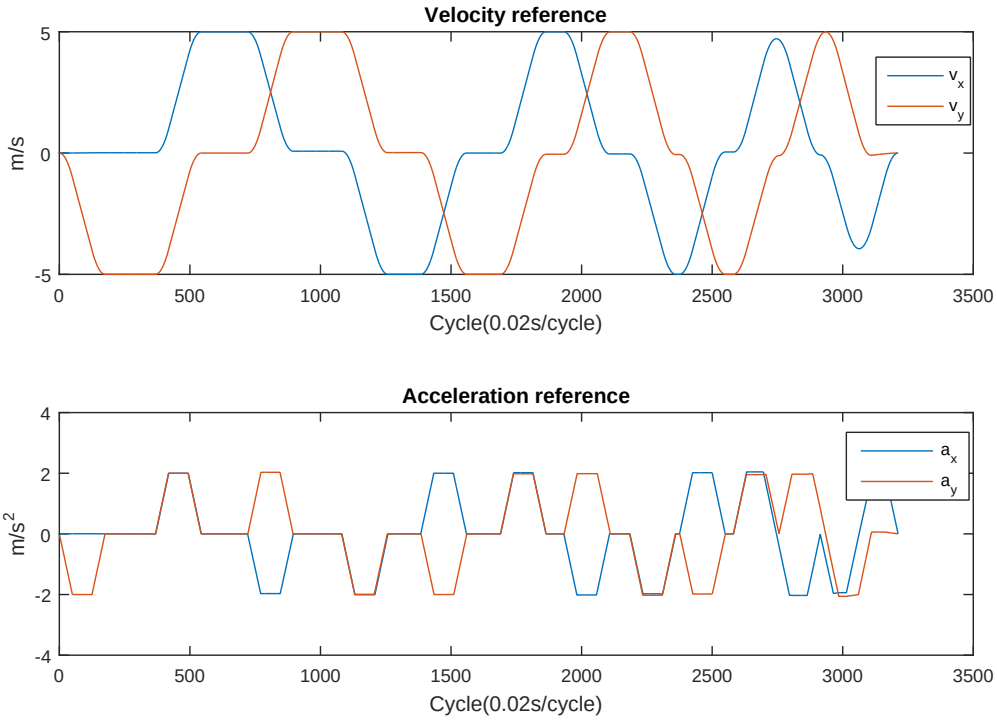


Figure 2.10: Generation-examination-tracking process for trajectory-based guidance

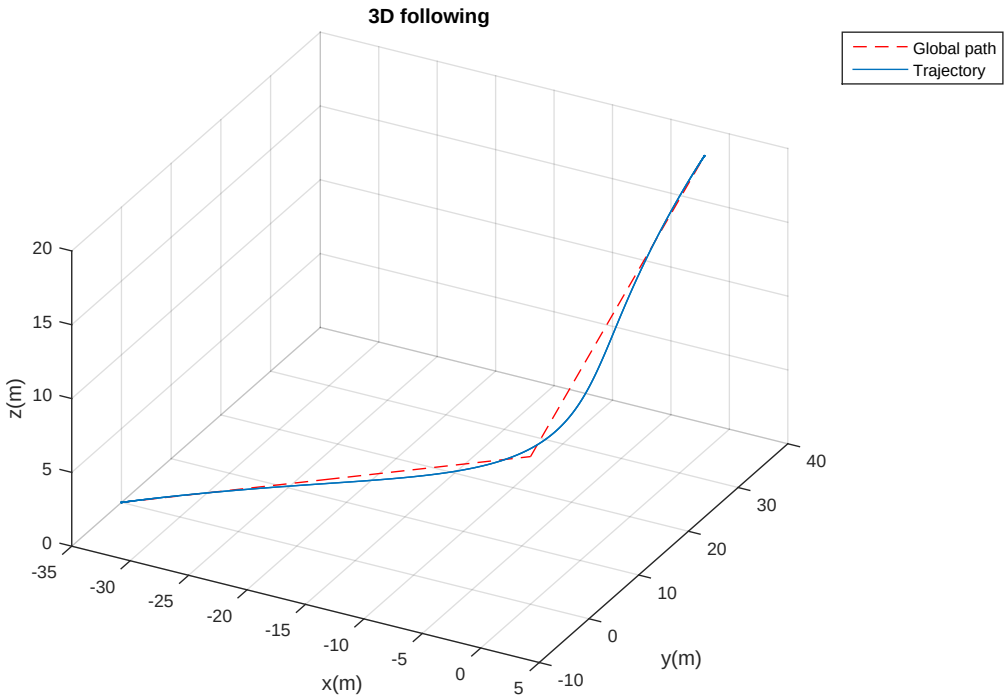


(a) 2D path

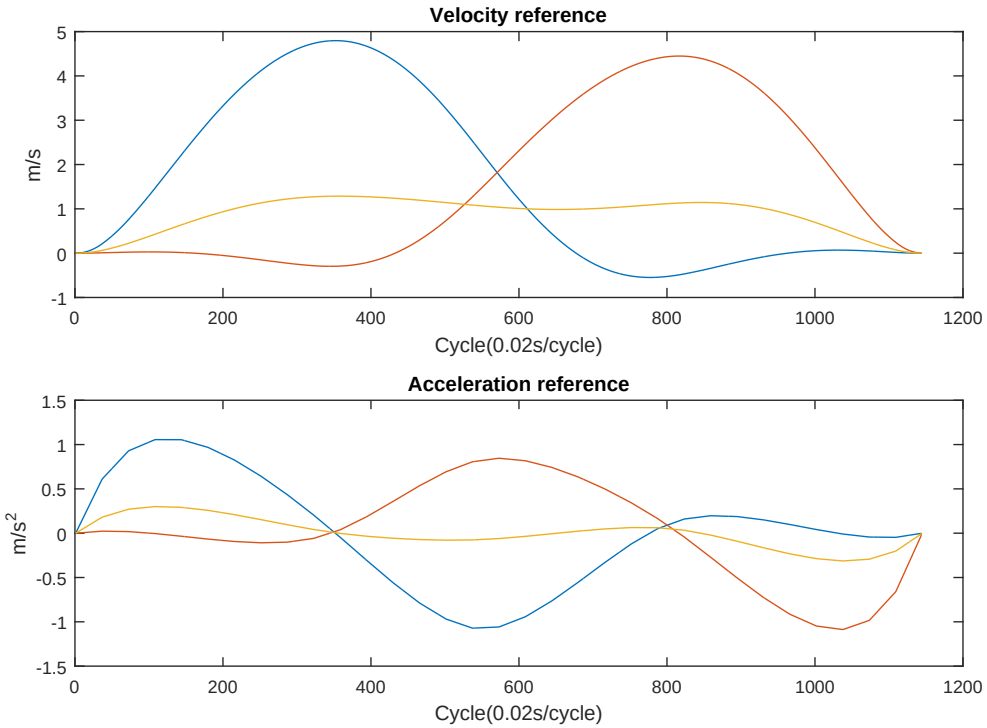


(b) Velocity and acceleration

Figure 2.11: Following of 2D circling path



(a) 3D path



(b) Velocity and acceleration

Figure 2.12: Following of 3D path

2.3 Control Level

2.3.1 Cascaded Control Structure

In this section, a generalized control structure for RUAV is discussed. The design aims to track translational reference as tight as possible. Translation movement is the foundation for a useful vehicle. First, a generalized point mass model for rotorcraft is given in Figure 2.13. The model is expressed in a global inertia coordinate $\mathcal{G}$ with the three axises as $\vec{x_{\mathrm{G}}}$, $\vec{y_{\mathrm{G}}}$ and $\vec{z_{\mathrm{G}}}$ respectively. Since rotorcrafts do not have wings, the main source of lift comes from their thrust. The three major forces acting on a rotorcraft are: combined thrust from all rotors, gravity and the air drag force. Such a model is expressed in Equation 2.1 and 2.2.

$$\begin{cases} \dot{\boldsymbol{p}} &= \boldsymbol{v} \\ \dot{\boldsymbol{v}} &= \boldsymbol{a} \\ \boldsymbol{a} &= \dfrac{1}{m_{\mathrm{q}}} \left(\boldsymbol{T}_{\Sigma}(\boldsymbol{q}) + \boldsymbol{d}_{\mathrm{air}}(\boldsymbol{v}_{\mathrm{air}}) \right) + \boldsymbol{g} \end{cases} \tag{2.1}$$

$$\begin{cases} \dot{\boldsymbol{x}_{\mathrm{q}}} &= \boldsymbol{f}_{\mathrm{q}}(\boldsymbol{x}_{\mathrm{q}}, \boldsymbol{v}) + \boldsymbol{g}_{\mathrm{q}}(\boldsymbol{x}_{\mathrm{q}}) \boldsymbol{u}_{\mathrm{q}} \\ \boldsymbol{q} &= C_{\mathrm{q}} \boldsymbol{x}_{\mathrm{q}} \end{cases} \tag{2.2}$$

where $\boldsymbol{p}$, $\boldsymbol{v}$, $\boldsymbol{a}$ are the position, velocity and acceleration vector of the vehicle, m_{q} is the vehicle's mass, $\boldsymbol{T}_{\Sigma}$, $\boldsymbol{d}_{\mathrm{air}}$, $\boldsymbol{g}$ are the three main forces of combined thrust, air drag and gravity. Then, $\boldsymbol{x}_{\mathrm{q}}$ is a collection of non-translational states such as the attitudes of the vehicle, $\boldsymbol{u}_{\mathrm{q}}$ is the true system inputs associated with physical actuators, $\boldsymbol{q}$ is a selected subsets of $\boldsymbol{x}_{\mathrm{q}}$ called controlled inner loop outputs and $\boldsymbol{f}_{\mathrm{q}}$, $\boldsymbol{g}_{\mathrm{q}}$ are functions describing the nonlinear dynamics. Here, the dynamics governed by Equation 2.1 are called the translational model. And the dynamics in Equation 2.2 are called the attitude model as $\boldsymbol{x}_{\mathrm{q}}$ usually contains parameters describing the rotorcraft's attitude. Most rotorcrafts such has helicopters, quad-rotors and coaxials are under-actuated systems, making them difficult to control in general. However, people notice that the attitude dynamics of rotorcraft are much faster than its translational dynamics. Therefore, the design of a cascaded controller utilizing the separation in time scale naturally emerged (see Figure 2.14). The idea is to have an outer loop governing the position of vehicle

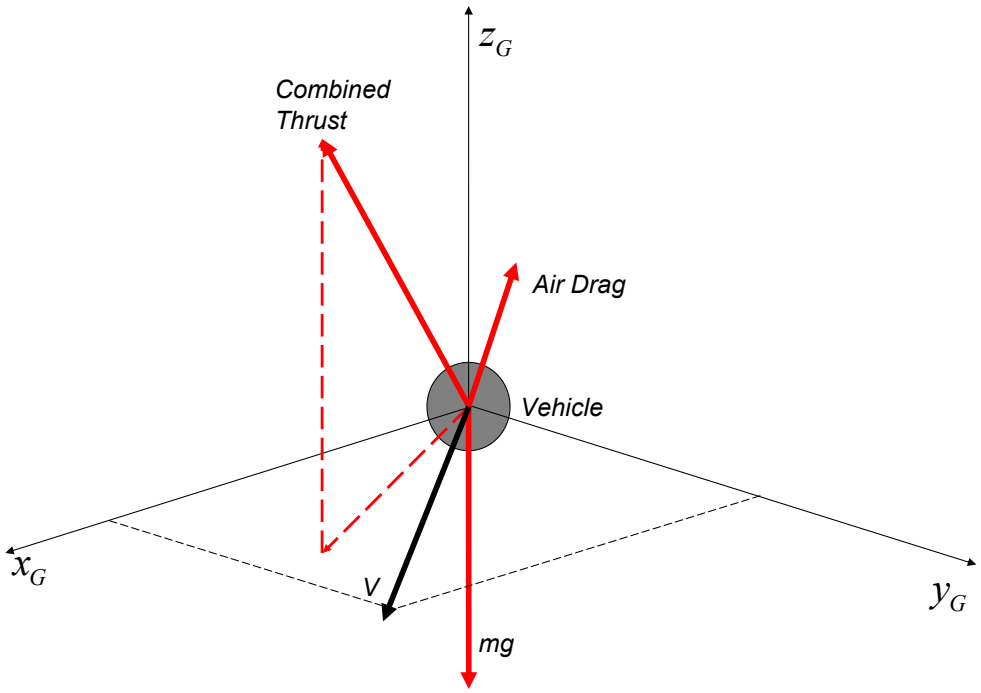


Figure 2.13: Generalized point mass model for rotorcrafts

by manipulating $\boldsymbol{q}$. The required $\boldsymbol{q}$ (denoted as $\boldsymbol{q}_{\mathrm{ref}}$) is then tracked by the attitude inner loop controller via $\boldsymbol{u}_{\mathrm{q}}$. In other words, $\boldsymbol{q}$ is the controlled output for the inner loop system. Nevertheless, $\boldsymbol{T}_{\Sigma}(\boldsymbol{q})$ is commonly a nonlinear function. To design a proper outer loop controller, the procedure of nonlinear dynamic inversion (NDI) is adopted. For translational loop, the controlling target is naturally the position $\boldsymbol{p}$. Then, one has to repeatedly differentiate the output $\boldsymbol{p}$ until $\boldsymbol{q}$ appears. This happens when

$$\ddot{\boldsymbol{p}} = \frac{1}{m_{\mathrm{q}}} \left(\boldsymbol{T}_{\Sigma}(\boldsymbol{q}) + \boldsymbol{d}_{\mathrm{air}}(\boldsymbol{v}_{\mathrm{air}}) \right) + \boldsymbol{g} \tag{2.3}$$

Considering the fact that $\boldsymbol{v}_{\mathrm{air}}$ is difficult to measure, it is treated as disturbance which is neglected during dynamic inversion and handled by the robust controller. Thus, a virtual control input $\boldsymbol{u}_{\mathrm{v}}$ is created as

$$\boldsymbol{u}_{\mathrm{v}} = \frac{1}{m_{\mathrm{q}}} \left(\boldsymbol{T}_{\Sigma}(\boldsymbol{q}) \right) + \boldsymbol{g} \tag{2.4}$$

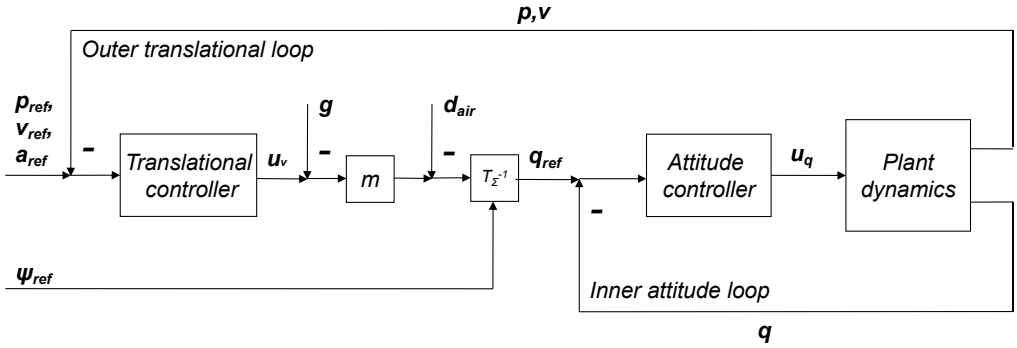


Figure 2.14: Cascaded control structure for a quad-rotor

and $\boldsymbol{q}_{\mathrm{ref}}$ can be correspondingly found as:

$$\boldsymbol{q}_{\mathrm{ref}} = \boldsymbol{T}_{\Sigma}^{-1}(m_{\mathrm{q}}(\boldsymbol{u}_{\mathrm{v}} - \boldsymbol{g})) \tag{2.5}$$

Then the control problem is reduced to design a linear controller for a double integrator with virtual input

$$\ddot{\boldsymbol{p}} = \boldsymbol{u}_{\mathrm{v}} \tag{2.6}$$

and the real desired outer loop control input can be found by Equation 2.5. Though not all controllers are structured in this way, the proposed structure has been successfully implemented on various type of rotorcrafts, such as helicopters [45], quad-rotors [46], octo-rotors [33], coaxials [47] and even unconventional vector thrust tail sitter [48]. They share a similar translational controller since it is designed based on a double integrator model. However, their inner loop controller varies largely due to the differences in the inner loop dynamics described by Equation 2.2. The cascaded control structure of a quad-rotor is shown in Figure 2.14. Note the heading angle reference of ψ_{ref} is also passed into the $\boldsymbol{T}_{\Sigma}^{-1}$ function. This is because the $\boldsymbol{q}$ of a quad-rotor's inner loop has four values $F_{\Sigma},\ \phi,\ \theta,\ \psi$ which are the total thrust magnitude, roll, pitch and yaw angle respectively. On the other hand, the virtual input for a 3D double integrator $\boldsymbol{u}_{\mathrm{v}} = [u_{\mathrm{v}x}, u_{\mathrm{v}y}, u_{\mathrm{v}z}]^{\mathrm{T}}$ has only three values. In order to get a unique solution for $\boldsymbol{q}$ through $\boldsymbol{T}_{\Sigma}^{-1}$, the value of ψ is preset with ψ_{ref}.

2.3.2 Robust Perfect Tracking (RPT) Control

The translational controller is generalizable, and a dedicated discussion is made here to improve its performance. As for the inner-loop controller, since the design relies on the nonlinear dynamics of individual vehicle type, it is difficult to outline a controller to work on all of them. Therefore, the detail discussion of inner-loop controller is omitted in this thesis.

As mentioned previously, after the dynamic inversion, the effective translational loop dynamics becomes a double integrator. From Equation 2.6, it is clear that the 3 degrees of freedom (DOF) of the point mass model are decoupled. Then it is natural to design a controller for each DOF separately. Here, the RPT controller from [49] is adopted for accuracy and robustness. Without input and states constraints, this controller can track any given reference with arbitrarily fast settling time in theory. For a linear time-invariant (LTI) system:

$$\Sigma = \begin{cases} \dot{\boldsymbol{x}_{\mathrm{L}}} &= A\boldsymbol{x}_{\mathrm{L}} + B\boldsymbol{u}_{\mathrm{L}} + E\boldsymbol{w}_{\mathrm{L}} \\ \boldsymbol{y}_{\mathrm{L}} &= C_1\boldsymbol{x}_{\mathrm{L}} + D_1\boldsymbol{w}_{\mathrm{L}} \\ \boldsymbol{h}_{\mathrm{L}} &= C_2\boldsymbol{x}_{\mathrm{L}} + D_2\boldsymbol{u}_{\mathrm{L}} + D_{22}\boldsymbol{w}_{\mathrm{L}} \end{cases} \tag{2.7}$$

with $\boldsymbol{x}_{\mathrm{L}}, \boldsymbol{u}_{\mathrm{L}}, \boldsymbol{w}_{\mathrm{L}}, \boldsymbol{y}_{\mathrm{L}}, \boldsymbol{h}_{\mathrm{L}}$ being the state, control input, disturbance, measurement and controlled output. The RPT controller provides a dynamic measurement control law of the form:

$$\begin{aligned} \dot{\boldsymbol{v}_{\mathrm{L}}} &= A_c(\varepsilon)\boldsymbol{v}_{\mathrm{L}} + B_c(\varepsilon)\boldsymbol{y}_{\mathrm{L}} + G_0(\varepsilon)\boldsymbol{r}_{\mathrm{L}} + ... + G_{\kappa-1}(\varepsilon)\boldsymbol{r}_{\mathrm{L}}^{\kappa-1}, \\ \boldsymbol{u}_{\mathrm{L}} &= C_c(\varepsilon)\boldsymbol{v}_{\mathrm{L}} + D_c(\varepsilon)\boldsymbol{y}_{\mathrm{L}} + H_0(\varepsilon)\boldsymbol{r}_{\mathrm{L}} + ... + H_{\kappa-1}(\varepsilon)\boldsymbol{r}_{\mathrm{L}}^{\kappa-1}, \end{aligned}$$

When a proper $\varepsilon > 0$ is chosen, the controller is then capable of

1. Stabilizing the closed-loop system asymptotically subjected to zero reference.

2. If $e_{\mathrm{L}}(t, \varepsilon)$ is the tracking error, then for any initial condition $\boldsymbol{x}_{\mathrm{L}0}$, there exists:

$$\|e_{\mathrm{L}}\|_p = (\int_0^\infty |e_{\mathrm{L}}(t)^p| dt)^{1/p} \to 0, \text{ as } \varepsilon \to 0. \tag{2.8}$$

The single-axis double integrator model can be written as:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_i, \quad \boldsymbol{y} = \boldsymbol{x} \tag{2.9}$$

where $\boldsymbol{x} = \begin{bmatrix} p & v \end{bmatrix}$ (p, v are the equivalents of $\boldsymbol{p}$, $\boldsymbol{v}$ on a specific axis respectively), u is the virtual input (aka. single-axis acceleration) and $\boldsymbol{y}$ stands for the single-axis measurements. An augmented system is then formulated as

$$\begin{cases} \dot{\boldsymbol{x}}_{\text{aug}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x}_{\text{aug}} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_{\text{aug}} \\ \boldsymbol{y}_{\text{aug}} = \boldsymbol{x}_{\text{aug}} \\ h_{\text{aug}} = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{x}_{\text{aug}} \end{cases} \tag{2.10}$$

where $\boldsymbol{x}_{\text{aug}} = \begin{bmatrix} p_{\text{ref}} & v_{\text{ref}} & a_{\text{ref}} & p & v \end{bmatrix}^{\text{T}}$ with $p_{\text{ref}}, v_{\text{ref}}, a_{\text{ref}}$ as the position, velocity and acceleration references in the same axis of p, v. According to [49, 50], an linear feed back law of the following form can be formulated as:

$$u_{\text{aug}} = F_{\text{aug}} \boldsymbol{x}_{\text{aug}}, \tag{2.11}$$

where

$$F_{\text{aug}} = \begin{bmatrix} \dfrac{\omega_{\text{n}}^2}{\varepsilon^2} & \dfrac{2\zeta\omega_{\text{n}}}{\varepsilon} & 1 & -\dfrac{\omega_{\text{n}}^2}{\varepsilon^2} & -\dfrac{2\zeta\omega_{\text{n}}}{\varepsilon} \end{bmatrix}.$$

The control law is achieved through performing a special coordinate decomposition (Theorem 3.1 in [49]) on Equation 2.10, then choose a feedback matrix to stabilize the resulting system, and finally transform the feedback matrix back to the normal coordinate. And the ε becomes a design parameter for adjusting the bandwidth of the closed loop system. ω_{n}, ζ are the parameters that determine the desired pole locations of the

infinite zero structure of (2.10) through

$$p_{\text{character}}(s_{\text{L}}) = s_{\text{L}}^2 + 2\zeta\omega_{\text{n}}s_{\text{L}} + \omega_{\text{n}}^2. \tag{2.12}$$

Theoretically, it is possible to achieve arbitrarily fast response when ε is small enough. However, due to the requirements on time-scale separation of the inner and outer loop, it is wise to limit the bandwidth of the outer loop to be much smaller than that of the inner loop. The design procedure of the cascaded controller can be summarized as:

1. Find inner loop controlled output $\boldsymbol{q}$ and its relationship to force $\boldsymbol{T}_{\Sigma}$.
2. Design the inner loop controller and measure the closed inner loop's bandwidth.
3. Perform dynamic inversion to simplify the nonlinear model, find virtual input and $\boldsymbol{T}_{\Sigma}^{-1}$.
4. Design the outer loop controller based on the simplified linear model with a much smaller bandwidth compared to the inner loop.

2.4 Conclusion

In this chapter, an MGCS for RUAV is introduced and studied. The MGCS is designed for the trajectory-based guidance method, where the process starts from the user specification of high-level tasks via the HMI, then a mission management system is responsible for translating the user specific missions into a series of waypoints that could be transmitted and understood by the vehicle. Then a two layer guidance module leads the vehicle to the point of interest or follows the path of design. The first layer of guidance is a global path planner whose main task is to examine the connectivity information of the environment. In our design, in order to fulfill the requirement of online computation, the dynamics of the vehicle is ignored in the global planner. Therefore, the result is a series of connected line segments that link the vehicle to the final target. Then a local planner produces translational reference for the vehicle's controller based on the hint from the global planner. After this, based on the dynamic properties of most rotorcraft vehicles, a cascaded controller is implemented to track the reference generated by the guidance unit. The cascade control structure is successfully implemented on various vehicles. Further, the translational controller is also generalizable because the outer loop model of the vehicle is simplified to a double integrator after dynamic inversion. The system serves as a testbed for many other advanced control and planning algorithms. The proposed system has been used in various competitions and projects [34, 13, 46].

In the next two chapters, trajectory generation and guidance algorithms for offline and online scenarios would be discussed accordingly.

Chapter 3

Offline Trajectory Generation Algorithm

Many applications, such as stage performance, camera trajectory planning, and SLAM algorithm verification, require pre-planned paths with precise maneuvers. Despite the rapid development of online trajectory generation methods, some tasks still require an offline algorithm with human involvement. For example, only human designer is capable of planning 'beautiful' trajectories for UAVs to execute during an entertainment event.

In this chapter, an offline trajectory planning algorithm is developed targeting these purposes. Section 3.1 discusses the requirements of dynamic feasibility and its usage as optimization constraints. Section 3.2 provides an overview of the algorithm by introducing the requirements and the mathematical tools. Problem formulation and numerical solution to the problem are covered in Section 3.3. Finally, algorithm implementation on RUAVs for calligraphy writing is presented in Section 3.4.

3.1 Dynamic Feasibility

As mentioned in Section 2.2.2, one important requirement imposed on reference trajectory is the dynamic feasibility. Since the inner loop is assumed to be significantly faster than the outer loop, a trajectory is said to be dynamically feasible enough if it can guarantee the constraints and continuity of the combined thrust vector $\boldsymbol{T}_\Sigma$. From Equation(2.3), it gives:

$$\boldsymbol{T}_\Sigma(\boldsymbol{q}) = m_{\mathrm{q}}(\ddot{\boldsymbol{p}} - \boldsymbol{g}) - \boldsymbol{d}_{\mathrm{air}}(\boldsymbol{v}_{air}) \tag{3.1}$$

by assuming $\boldsymbol{d}_{\mathrm{air}}$ is continuous and $\boldsymbol{v}_{\mathrm{air}} = -\boldsymbol{v}$ (no wind in the environment), it gives:

$$\boldsymbol{T}_\Sigma(\boldsymbol{q}) = m_{\mathrm{q}}(\ddot{\boldsymbol{p}} - \boldsymbol{g}) - \boldsymbol{d}_{\mathrm{air}}(-\dot{\boldsymbol{p}}) \tag{3.2}$$

To guarantee the continuity of $\boldsymbol{T}_\Sigma$, $\boldsymbol{p}$ is required to have at least $\mathbf{C}_2$ continuity which equivalently makes the planning model as a triple integrator. For the constraints on $\boldsymbol{T}_\Sigma$, conservative measures are taken:

1. Find the set of achievable combined thrust $\mathbb{T}_\Sigma$ by examine the vehicle's non-linear model.
2. Plug the gravity $\boldsymbol{g}$ back and create $\mathbb{T}_{\mathrm{g}} = \{\boldsymbol{T} + \boldsymbol{g} \mid \boldsymbol{T} \in \mathbb{T}_\Sigma\}$
3. Find a rectangular cuboid lies in the interior of $\mathbb{T}_{\mathrm{g}}$, where the cuboid can be characterized by
$$\begin{cases} T_{\min x} \leq T_{\mathrm{g}_x} \leq T_{\max x} \\ T_{\min y} \leq T_{\mathrm{g}_y} \leq T_{\max y} \\ T_{\min z} \leq T_{\mathrm{g}_z} \leq T_{\max z} \end{cases}$$
4. For each axis i, find $a_{\max i},\ a_{\min i},\ v_{\max i},\ v_{\min i}$ which satisfy:

   $T_{\min i} \leq m(a_i - d_{\mathrm{air}}(-v_i)) \leq T_{\max i}$

   for all $a_i \in [a_{\min i},\ a_{\max i}], v_i \in [v_{\min i},\ v_{\max i}]$

Through this procedure, the constraints on $\boldsymbol{T}_\Sigma$ can be decoupled into each individual axis and expressed as the derivatives of $\boldsymbol{p}$. The effectiveness of our approach is proven through various simulations and real flight experiments, which will be discussed in

the following sections. The advantage of the decoupled constraints lies in the planning stage where the linear constraint is achievable. The trajectory generation would be more difficult if the coupled constraints are involved.

3.2 Algorithm Overview

3.2.1 Requirement Analysis

Vehicles are usually required to travel in a smooth and stable manner. In other words, abrupt changes of the combined thrust $\boldsymbol{T}_\Sigma$ shall be prevented. Equation 3.1 shows the derivatives of $\boldsymbol{p}$ shall be minimized to achieve the task.

Take derivation of Equation 2.3, the derivative of $\boldsymbol{T}_\Sigma$ is shown to be related to the trajectory's jerk which is the third derivative of the $\boldsymbol{p}$. Also, it has been proven in [11] that the upper bound of a quad-rotor's body angular rate is related to the magnitude of the trajectory's jerk. Therefore, for a smooth maneuver, a minimum jerk trajectory is considered. Interestingly, human beings also adopt minimum jerk trajectory when moving their limbs [51]. Combining the requirements on both dynamic feasibility and smoothness, the trajectory needs to be:

1. At least $\mathbf{C}_2$ continuous.

2. Satisfying the decoupled constraints of $a_i \in [a_{\min i},\ a_{\max i}], v_i \in [v_{\min i},\ v_{\max i}]$ for each axis i.

3. Able to minimize the derivatives of the position $\boldsymbol{p}$ up to jerk.

4. Able to specify an initial and end state (commonly as hover state) for a safe experiment.

Further, the algorithm should be as efficient as possible, since it is commonly used in GCS and mission manager for path designing purpose. Due to the dynamics of the targeting system (a triple integrator in this case), the dimension of the optimization space becomes infinite if the planning variables are chosen as the inputs of the integrator. Therefore, primitives are used to describe the trajectory and help to formulate the optimization problem in finite dimension.

3.2.2 Clamped Normalized Uniform B-Spline (NUBS)

In this thesis, a clamped Normalized Uniform B-Spline (NUBS) is considered as the primitive. By combining the optimal smoothing and interpolating method in [52, 53]

with a time vector optimization procedure, smooth 3D flight trajectory can be generated. The clamped spline is considered during the formulation so that the flying path is clamped by (tangent to) the first and last legs formed by the control points to match human intuition. An effort to add constraints on derivatives had also been made[1]. The algorithm has been successfully integrated into the MGCS with multiple real flight experiments.

B-spline is named after its choice of primitives, the Bezier curves which are commonly used for animators and motion planners. The classical definition of a B-spline is the following recursive form:

$$\begin{aligned} N_{i,0}(\varpi) &= \begin{cases} 1, \text{ if } \varpi_i \leq \varpi < \varpi_{i+1}, \\ 0, \text{ otherwise.} \end{cases} \\ N_{i,\rho}(\varpi) &= \frac{\varpi - \varpi_i}{\varpi_{i+\rho} - \varpi_i} N_{i,\rho-1}(\varpi) + \frac{\varpi_{i+\rho+1} - \varpi}{\varpi_{i+\rho+1} - \varpi_{i+1}} N_{i+1,\rho-1}(\varpi) \end{aligned} \tag{3.3}$$

where $N_{i,p}(\varpi)$ forms the base function of the generally defined B-splines and $V_{\mathrm{b}} = [\varpi_0, \varpi_1, \varpi_2, ...]$ is the knot vector of B-splines.

NUBS is a special type of B-spline which gets its name after the normalized knot vector. The normalized knot vector is equally segmented as $[0, 1, 2, 3, \ldots, m]$ and the base function of a NUBS can be written as [55]

$$N_{j,i}(s) = \begin{cases} 1, & \text{if } i = j = 0, \\ \frac{1-s}{i} N_{0,i-1}(s), & \text{if } j = 0, i \neq 0, \\ \frac{s}{i} N_{i-1,i-1}(s), & \text{if } j = i > 0, \\ \frac{i-j+s}{i} N_{j-1,i-1}(s) + \frac{1+j-s}{i} N_{j,i-1}(s), & \text{if } j = 1, ..., i-1, \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

where s is called the path parameter spanning through $[0, m]$. Thus, a trajectory in

[1]The author extended the work in [53] for the derivative-constrained trajectory based on a clamped-spline, but later found the same method had already been covered in [54] with a more general proof. Nevertheless, the addition of constraints on derivatives is covered for completeness.

3-dimensional space can then be expressed as

$$S_{\mathrm{k}}(s) = \sum_{i=1}^{M} \boldsymbol{c}_i B_{\mathrm{k}}(s - i + 1), \quad \boldsymbol{c}_i \in \mathbb{R}^3, \tag{3.5}$$

where

$$B_{\mathrm{k}}(s) = \begin{cases} N_{k-j,k}(s-j), \ j \leq s < j+1, \\ 0, \qquad\qquad \text{otherwise.} \end{cases} \tag{3.6}$$

and $j = 0, ..., k$. Here, k represents the order of spline, M is a design parameter which denotes the number of control points, and $\boldsymbol{c}_i$ is the control point in 3-D space. The decoupled constraints make it possible to solve the problem on each axis individually. Further, the flight path is preferred to be clamped by the first and last legs of control points to match human intuition. With the requirement of $\mathbf{C}_2$ continuity, a cubic clamped NUBS becomes our final choice. The proposed B-spline has a knot vector as $V_{\mathrm{knot}} = [0, 0, 0, 0, 1, 2, 3, ..., m-1, m, m, m, m]$. An open cubic B-spline with M control points will have a knot vector of $M + 4$ entries. If it becomes clamped, 6 more control points are added for specifying the initial and end boundary conditions. Correspondingly, V_{knot} now grows to $M + 10$ members with $m = M + 3$. For a dynamic system, it is more important to study its property regarding the real time t instead of the path parameter s. For the ease of further analysis and problem solving, a linear relation between t and s is assigned as in [52]:

$$\frac{s}{t} = \frac{M+3}{T_{\mathrm{true}}} = \alpha \tag{3.7}$$

where T_{true} is a tunning factor representing the total time of the trajectory. With Equation(3.7), there is also:

$$\frac{\mathrm{d}s}{\mathrm{d}t} = \alpha \tag{3.8}$$

Due to the difference in the time segmentation of V_{knot} among the first and last four elements, the first and last three base functions would also be different from the others. They can be calculated as two parts from Equation 3.3, the first part is the common base that excludes the first three and last three elements of all the base functions. Denoted as

$B_3(\varpi)$, it is given by

$$B_3(\varpi) = \begin{cases} \frac{1}{6}\varpi^3, & \text{if } \varpi \in [0,1), \\ \frac{1}{6} - \frac{1}{2}(\varpi-1)^3 + \frac{1}{2}(\varpi-1)^2 + \frac{1}{2}(\varpi-1), & \text{if } \varpi \in [1,2), \\ \frac{1}{6} + \frac{1}{2}(\varpi-3)^3 + \frac{1}{2}\varpi-3)^2 - \frac{1}{2}(\varpi-3), & \text{if } \varpi \in [2,3), \\ -\frac{1}{6}(\varpi-4)^3, & \text{if } \varpi \in [3,4), \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

whereas the second part which consists of the first and last three bases. The first three are:

$$N_{0,3}(\varpi) = \begin{cases} (1-\varpi)^3, & \text{if } \varpi \in [0,1), \\ 0, & \text{otherwise.} \end{cases}$$

$$N_{1,3}(\varpi) = \begin{cases} \varpi(1-\varpi)^2 + \frac{\varpi(4-3\varpi)(2-\varpi)}{8}, & \text{if } \varpi \in [0,1), \\ -\frac{(\varpi-2)^3}{4}, & \text{if } \varpi \in [1,2), \\ 0, & \text{otherwise.} \end{cases} \tag{3.10}$$

$$N_{2,3}(\varpi) = \begin{cases} \frac{\varpi^2(-3\varpi+4)}{4} + \frac{\varpi^2(3-\varpi)}{6}, & \text{if } \varpi \in [0,1), \\ \frac{\varpi(\varpi-2)^2}{4} + \frac{(3-\varpi)(-2\varpi^2+6\varpi-3)}{6}, & \text{if } \varpi \in [1,2), \\ \frac{\varpi^2(-3\varpi+4)}{4} + \frac{\varpi^2(3-\varpi)}{6}, & \text{if } \varpi \in [2,3), \\ 0, & \text{otherwise.} \end{cases}$$

while the last three are the reversed version of Equation 3.10:

$$\begin{aligned} N_{M+3,3}(\varpi) &= N_{2,3}(-\varpi + M + 3), \\ N_{M+4,3}(\varpi) &= N_{1,3}(-\varpi + M + 3), \\ N_{M+5,3}(\varpi) &= N_{0,3}(-\varpi + M + 3). \end{aligned} \tag{3.11}$$

By using the base function in Equation 3.9, 3.11 and 3.10 , the clamped cubic NUBS is given as

$$S_3(s) = \sum_{i=-2}^{0} \boldsymbol{c}_i N_{i+2,3}(s) + \sum_{i=1}^{M} \boldsymbol{c}_i B_3(s-i+1) + \sum_{i=M+1}^{M+3} \boldsymbol{c}_i N_{i+2,3}(s) \quad (3.12)$$

Noted in 3.12, a total of $M+6$ control points exist, where the first three and last three are used to determine the boundary conditions. To simplify the expression, a new basis symbol Γ is defined as:

$$\Gamma_{i,3}(s) \;=\; \begin{cases} N_{i,3}(s), & \text{if } i \in \{0,1,2,M+3,M+4,M+5\}, \\ B_3(s-i+3), & \text{otherwise.} \end{cases} \quad (3.13)$$

with which the clamped cubic NUBS can be written as

$$S_3(s) = \sum_{i=0}^{M+5} \boldsymbol{\tau}_i \Gamma_{i,3}(s), \boldsymbol{\tau}_i = \boldsymbol{c}_{i-2} \quad (3.14)$$

where $\boldsymbol{\tau}$ is a shifted representation of $\boldsymbol{c}$.

3.3 Solving of Minimum Jerk Trajectory

3.3.1 Problem formulation

Now, the clamped spline is expressed in a single equation (Equation 3.14) which fits the optimal smoothing and interpolating splines formulation in [53]. The formulation gives a quadratic optimization problem for the minimum jerk trajectory. It also shares a similar idea to the work appeared in [10] which uses polynomials. As mentioned in Section 3.2, the independent optimization variable $\boldsymbol{c}_i$ is solved on the $x-$, $y-$ and $z-$ axis separately, the following discussion in this section assumes a single axis back ground. The scalars c and τ represent the equivalents of $\boldsymbol{c}$ and $\boldsymbol{\tau}$ on a single axis respectively.

As in [53], if the data to be fitted is given in the following form

$$\begin{aligned} D_{\mathrm{s}} &= [d_0, d_1, ..., d_{I_{\mathrm{d}}}]^{\mathrm{T}}, \\ D_{\mathrm{time}} &= [t_0, t_1, ..., t_{I_{\mathrm{d}}}]^{\mathrm{T}}, \end{aligned} \quad (3.15)$$

where D_{s} is a vector of one dimensional (1D) data points for which the trajectory shall approach, D_{time} is the vector of time indicating at what time each of the above data points are reached, and I_{d} is the length of both vectors, then the problem for achieving the minimum jerk is equivalent to minimize the following cost function:

$$J_{\mathrm{s}} = w_{\mathrm{g}} \int_{-\infty}^{\infty} j_{\mathrm{s}}^2(t)dt + \sum_{i=1}^{I_{\mathrm{d}}} (S_3(\alpha t_i) - d_i)^2, \tag{3.16}$$

with w_{g} as a weighting factor, and

$$j_{\mathrm{s}}(t) = \sum_{i=0}^{M+5} \frac{d^3}{dt^3} \tau_i \Gamma_{i,3}(s), \quad \tau_i \in \mathbb{R}, \tag{3.17}$$

which is the third derivative of Equation 3.14 on a single axis. However, for flight path design, most of the time, the D_{time} won't be given. And an unexperienced guess usually gives trajectory of low quality. For example, the total execution time might be too long, or the trajectory fails to reflect user's design very well. In such a case, the value of D_{time} also becomes a programming variable and Algorithm 5 is used solve it.

Equation 3.16 consists of two parts, the first part stands for the cost on jerk

$$\int_{-\infty}^{\infty} j_{\mathrm{s}}^2(t)dt$$

and the second part represents the cost for deviating from the input data points D_{s}

$$\sum_{i=1}^{I_{\mathrm{d}}} (S_3(\alpha t_i) - d_i)^2$$

In [52], a quadratic cost function for minimizing second derivative is considered. Using its technique, the first part can be rewritten in the form of $T_{\mathrm{s}}^T G_{\mathrm{s}} T_{\mathrm{s}}$ where $T_{\mathrm{s}} = [\tau_0, \tau_1, ..., \tau_{M+5}]^{\mathrm{T}}$ is called the control point vector and G_{s} being a square matrix with each of its element g_{s} at row i column j as

$$g_{\mathrm{s}}(i,j) = \alpha^5 \int_{-\infty}^{\infty} \frac{d^3\Gamma_{i,3}(s)}{ds^3} \frac{d^3\Gamma_{j,3}(s)}{ds^3} ds \tag{3.18}$$

and the second part can be expressed as

$$\sum_{i=1}^{I_{\mathrm{d}}}(S_3(\alpha t_i)-d_i)^2=(H\tau-D_{\mathrm{s}})^{\mathrm{T}}(H\tau-D_{\mathrm{s}}) \tag{3.19}$$

and

$$H=\begin{bmatrix}\Gamma_{0,3}(\alpha t_1) & \Gamma_{1,3}(\alpha t_1) & \dots & \Gamma_{M+5,3}(\alpha t_1)\\ \Gamma_{0,3}(\alpha t_2) & \Gamma_{1,3}(\alpha t_2) & \dots & \Gamma_{M+5,3}(\alpha t_2)\\ \vdots & \vdots & \ddots & \vdots\\ \Gamma_{0,3}(\alpha t_I) & \Gamma_{1,3}(\alpha t_I) & \dots & \Gamma_{M+5,3}(\alpha t_I)\end{bmatrix} \tag{3.20}$$

Note that the dimension of vector T_{s} and D_{s} are $M+6$ and I_{d} respectively. Substitute Equations 3.19, 3.20 into 3.16, there is

$$J_{\mathrm{smin}}=\min_{T_{\mathrm{s}}}\left\{w_{\mathrm{g}}T_{\mathrm{s}}^{\mathrm{T}}G_{\mathrm{s}}T_{\mathrm{s}}+(HT_{\mathrm{s}}-D_{\mathrm{s}})^{\mathrm{T}}(HT_{\mathrm{s}}-D_{\mathrm{s}})\right\} \tag{3.21}$$

which is clearly a quadratic form regarding to T_{s}. Though not used in the software implementation, a closed-form solution can be given if no other constraints are considered. As mentioned before, the first and last three elements in D_{s} are used to determine the boundary conditions. Thus part or all of their values can be calculated based on these conditions and fixed separately. Correspondingly, T_{s} is separated into the pre-fixed part $T_{\mathrm{s}}^{\mathrm{F}}$ and the programmable part $T_{\mathrm{s}}^{\mathrm{P}}$ by a transformation matrix U_{s} as:

$$U_{\mathrm{s}}\begin{bmatrix}T_{\mathrm{s}}^{\mathrm{F}}\\ T_{\mathrm{s}}^{\mathrm{P}}\end{bmatrix}=T_{\mathrm{s}} \tag{3.22}$$

Then, Equation (3.21) can be rewrite as

$$\begin{aligned}J_{\mathrm{smin}} \;=\; \min_{T_{\mathrm{s}}^{\mathrm{P}}}\Bigg\{&\begin{bmatrix}T_{\mathrm{s}}^{\mathrm{F}}\\ T_{\mathrm{s}}^{\mathrm{P}}\end{bmatrix}^{\mathrm{T}} w_{\mathrm{g}}U_{\mathrm{s}}^{\mathrm{T}}G_{\mathrm{s}}U_{\mathrm{s}}\begin{bmatrix}T_{\mathrm{s}}^{\mathrm{F}}\\ T_{\mathrm{s}}^{\mathrm{P}}\end{bmatrix}\\ &+(HU_{\mathrm{s}}\begin{bmatrix}T_{\mathrm{s}}^{\mathrm{F}}\\ T_{\mathrm{s}}^{\mathrm{P}}\end{bmatrix}-D_{\mathrm{s}})^{\mathrm{T}}(HU_{\mathrm{s}}\begin{bmatrix}T_{\mathrm{s}}^{\mathrm{F}}\\ T_{\mathrm{s}}^{\mathrm{P}}\end{bmatrix}-D_{\mathrm{s}})\Bigg\}\end{aligned} \tag{3.23}$$

which can be further simplified to

$$J_{\mathrm{s}\min} = \min_{T_{\mathrm{s}}^{\mathrm{P}}} \left\{ \begin{bmatrix} T_{\mathrm{s}}^{\mathrm{F}} \\ T_{\mathrm{s}}^{\mathrm{P}} \end{bmatrix}^{\mathrm{T}} (w_{\mathrm{g}} U_{\mathrm{s}}^{\mathrm{T}} G_{\mathrm{s}} U_{\mathrm{s}} + U_{\mathrm{s}}^{\mathrm{T}} H^{\mathrm{T}} H U_{\mathrm{s}}) \begin{bmatrix} T_{\mathrm{s}}^{\mathrm{F}} \\ T_{\mathrm{s}}^{\mathrm{P}} \end{bmatrix} \right. \\ \left. -2 D_{\mathrm{s}}^{\mathrm{T}} H U_{\mathrm{s}} \begin{bmatrix} T_{\mathrm{s}}^{\mathrm{F}} \\ T_{\mathrm{s}}^{\mathrm{P}} \end{bmatrix} + D_{\mathrm{s}}^{\mathrm{T}} D_{\mathrm{s}} \right\} \tag{3.24}$$

Assign matrices $R_{\mathrm{s}} = w_{\mathrm{g}} U_{\mathrm{s}}^{\mathrm{T}} G_{\mathrm{s}} U_{\mathrm{s}} + U_{\mathrm{s}}^{\mathrm{T}} H^{\mathrm{T}} H U_{\mathrm{s}}$, $S_{\mathrm{s}} = D_{\mathrm{s}}^{\mathrm{T}} H U_{\mathrm{s}}$ and split them according to the dimension of $T_{\mathrm{s}}^{\mathrm{F}}$ and $T_{\mathrm{s}}^{\mathrm{P}}$, there is

$$R_s = \begin{bmatrix} R_{\mathrm{s}}^{\mathrm{FF}} & R_{\mathrm{s}}^{\mathrm{FP}} \\ R_{\mathrm{s}}^{\mathrm{PF}} & R_{\mathrm{s}}^{\mathrm{PP}} \end{bmatrix} \text{ and } S_{\mathrm{s}} = [S_{\mathrm{s}}^{\mathrm{F}}, S_{\mathrm{s}}^{\mathrm{P}}] \tag{3.25}$$

Substitute Equation 3.25 into 3.24, it gives

$$J_{\mathrm{s}\min} = \min_{T_{\mathrm{s}}^{\mathrm{P}}} \Big\{ {T_{\mathrm{s}}^{\mathrm{F}}}^{\mathrm{T}} R_{\mathrm{s}}^{\mathrm{FF}} T_{\mathrm{s}}^{\mathrm{F}} + {T_{\mathrm{s}}^{\mathrm{F}}}^{\mathrm{T}} R_{\mathrm{s}}^{\mathrm{FP}} T_{\mathrm{s}}^{\mathrm{P}} + {T_{\mathrm{s}}^{\mathrm{P}}}^{\mathrm{T}} R_{\mathrm{s}}^{\mathrm{PF}} T_{\mathrm{s}}^{\mathrm{F}} \\ + {T_{\mathrm{s}}^{\mathrm{P}}}^{\mathrm{T}} R_{\mathrm{s}}^{\mathrm{PP}} T_{\mathrm{s}}^{\mathrm{P}} - 2 S_{\mathrm{s}}^{\mathrm{F}} T_{\mathrm{s}}^{\mathrm{F}} - 2 S_{\mathrm{s}}^{\mathrm{P}} T_{\mathrm{s}}^{\mathrm{P}} + D_{\mathrm{s}}^{\mathrm{T}} D_{\mathrm{s}} \Big\} \tag{3.26}$$

To solve for optimal $T_{\mathrm{s}}^{\mathrm{P}}$, simply taking the first derivative with respect to $T_{\mathrm{s}}^{\mathrm{P}}$ will yield a closed-form solution as

$$T_{\mathrm{s}}^{\mathrm{P}*} = {R_{\mathrm{s}}^{\mathrm{PP}}}^{-1} ({S_{\mathrm{s}}^{\mathrm{P}}}^{\mathrm{T}} - {R_{\mathrm{s}}^{\mathrm{FP}}}^{\mathrm{T}} T_{\mathrm{s}}^{\mathrm{F}}) \tag{3.27}$$

Since a clamped spline is used, the resulting flight path will always be tangent to the first and last legs formed by the control points.

3.3.2 Quadratic Programming

Though the closed form solution can be found, real world experience tells us that it is difficult to find the inverse of a large size matrix like $R_{\mathrm{s}}^{\mathrm{PP}}$ due to the numerical instability. Nonetheless, since the problem in Equation 3.21 represents the form of a quadratic optimization, and it is shown to be convex [52], it can be solved efficiently. Considerable efforts are devoted into the solving of these type of questions. Therefore, efficient and numerically robust algorithms and packages can be adopted such as *quadprog* [56]

from Matlab or *CPLEX* [57] from IBM. To get a typical quadratic optimization formulation, Equation 3.21 is rearranged into

$$J_{\mathrm{s}\min} = \min_{T_\mathrm{s}} \left\{ T_\mathrm{s}^\mathrm{T}(w_\mathrm{g} G_\mathrm{s} + H^\mathrm{T} H) T_\mathrm{s} - 2 D_\mathrm{s}^\mathrm{T} H \tau + D_\mathrm{s}^\mathrm{T} D_\mathrm{s} \right\} \tag{3.28}$$

The cost function is naturally convex by its formulation [52]. By Equation 3.18, matrix G_s is constructed in a way to satisfy

$$T_\mathrm{s}^T G_\mathrm{s} T_\mathrm{s} = \int_{-\infty}^{\infty} j_\mathrm{s}^2(t) dt \tag{3.29}$$

Because $j_\mathrm{s}^2(t)$ is an non-negative value, therefore its integration

$$\int_{-\infty}^{\infty} j_\mathrm{s}^2(t) dt \geq 0 \tag{3.30}$$

always holds. With Equation 3.17, 3.29 and 3.30 there is:

$$T_\mathrm{s}^T G_\mathrm{s} T_\mathrm{s} = \int_{-\infty}^{\infty} (\sum_{i=0}^{M+5} \frac{d^3}{dt^3} \tau_i \Gamma_{i,3}(s))^2 dt \geq 0 \tag{3.31}$$

Because T_s is just a collection of τ_i as $T_\mathrm{s} = [\tau_0, \tau_1, ..., \tau_{M+5}]^\mathrm{T}$, Equation 3.31 is equally saying that for any real value column vector T_s, the expression

$$T_\mathrm{s}^T G_\mathrm{s} T_\mathrm{s} \geq 0$$

holds true. Therefore, G_s is at least semi-positive definite (SPD). While $H^\mathrm{T} H$ is always SPD, for any positive value w_g, the term $w_\mathrm{g} G_\mathrm{s} + H^\mathrm{T} H$ is also SPD. Moreover, from Equation 3.30, if

$$\int_{-\infty}^{\infty} j_\mathrm{s}^2(t) dt = 0$$

then $j_\mathrm{s}(t) = 0$ constantly. Since $j_\mathrm{s}(t)$ stands for the jerk, the physical meaning it implies is a trajectory without any maneuvering at all. Clearly, such trajectory is not in the area of interest and it does not need any planning. With this in mind, Equation 3.30 now becomes

$$\int_{-\infty}^{\infty} j_\mathrm{s}^2(t) dt > 0 \tag{3.32}$$

And $T_{\mathrm{s}}^{T} G_{\mathrm{s}} T_{\mathrm{s}} > 0$.

The following discussion in this section focuses on limiting the derivatives of the trajectory[2].

The following 3 properties of B-spline [61] are utilized.

1. For a NUBS, its base function is always non-negative.

2. For a clamped NUBS $S_{\rho}(s)$ with M_{e} control points of any order ρ, its derivative is still a clamped NUBS with order $\rho - 1$ evaluated on a new set of knot vector, which can be represented as

$$\frac{dS_{\rho}(s)}{ds} = \sum_{i=0}^{M_{\mathrm{e}}-1} \tau_i^{\rho-1} \Gamma_{i+1,\rho-1}(s) \tag{3.33}$$

   where $\tau^{\rho-1}$ are the new control points for the reduced order B-spline and it can be calculated as

$$\tau_i^{\rho-1} = \frac{\rho}{\varpi_{i+\rho+1} - \varpi_{i+1}} (\tau_{i+1}^{\rho} - \tau_i^{\rho}) \tag{3.34}$$

   The new knot vector is gained by removing the first and last item from the original one.

3. The initial value of a clamped B-spline equals its first control point.

$$S_{\rho}(0) = \tau_0^{\rho} \tag{3.35}$$

Therefore, for a clamped B-spline $S_{\rho}(0)$ with

$$\tau_0^{\rho} = \tau_1^{\rho} = \tau_2^{\rho} = \cdots = \tau_{M_{\mathrm{e}}-1}^{\rho} = \tau_{\mathrm{C}}^{\rho}$$

according to Equation 3.34 there is

$$\tau_0^{\rho-1} = \tau_1^{\rho-1} = \cdots = \tau_{M_{\mathrm{e}}-2}^{\rho-1} = 0$$

with Equation 3.33 it gives

$$\frac{dS_{\rho}(s)}{ds} = 0$$

[2]The discussion assumes a clamped spline. But the same method with a more general proof had already been covered by Fujioka in [54].

which means the value of $S_\rho(s)$ is a constant. Further, with Equation 3.35, the following conclusion holds

$$\begin{aligned} S_\rho(s) &= \sum_{i=0}^{M_\mathrm{e}-1} \tau_i^\rho \Gamma_{i,\rho}(s) = \tau_\mathrm{C}^\rho \\ \text{if} \quad & \tau_0^\rho = \tau_1^\rho = \tau_2^\rho = \cdots = \tau_{M_\mathrm{e}-1}^\rho = \tau_\mathrm{C}^\rho \end{aligned} \tag{3.36}$$

From Equation.3.33, the nth derivative of $S_\rho(s)$ can be represented as

$$\frac{d^n S_\rho(s)}{ds^n} = \sum_{i=0}^{M_\mathrm{e}-n} \tau_i^{\rho-n} \Gamma_{i+n,\rho-n}(s)$$

Combining with Equation 3.8, the nth time derivative of $S_\rho(s)$ is then

$$\frac{d^n S_\rho(s)}{dt^n} = \alpha^n \sum_{i=0}^{M_\mathrm{e}-n} \tau_i^{\rho-n} \Gamma_{i+n,\rho-n}(s)$$

Since the base functions $\Gamma_{i+n,\rho-n}(s)$ are always non-negative, it gives

$$\frac{d^n S_\rho(s)}{dt^n} = \alpha^n \sum_{i=0}^{M_\mathrm{e}-n} \tau_i^{\rho-n} \Gamma_{i+n,\rho-n}(s) \geq \alpha^n \sum_{i=0}^{M_\mathrm{e}-n} \tau_\mathrm{C} \Gamma_{i+n,\rho-n}(s)$$

$$\text{if} \quad \tau_i^{\rho-n} \geq \tau_\mathrm{C}, \quad \forall i \in [0, M_\mathrm{e} - n]$$

With Equation 3.36, it is equivalent as

$$\begin{aligned} &\frac{d^n S_\rho(s)}{dt^n} = \alpha^n \sum_{i=0}^{M_\mathrm{e}-n} \tau_i^{\rho-n} \Gamma_{i+n,\rho-n}(s) \geq \alpha^n \tau_\mathrm{C} \\ &\text{if} \quad \tau_i^{\rho-n} \geq \tau_\mathrm{C}, \quad \forall i \in [0, M_\mathrm{e} - n] \end{aligned} \tag{3.37}$$

Therefore, if the nth time derivative of the clamped NUBS $S_\rho(s)$ is to be limited between $S_{\max}^{(n)}$ and $S_{\min}^{(n)}$, the sufficient condition is

$$S_{\min}^{(n)} \leq \alpha^n \tau_i^{\rho-n} \leq S_{\max}^{(n)}, \quad \forall i \in [0, M_e - n] \tag{3.38}$$

With the result in Equation 3.38, the last job is to find $\tau_i^{\rho-n}$ for the nth derivative.

In case of a clamped cubic NUBS in Equation 3.14, taking the first time derivative

of the B-spline trajectory yields

$$\frac{dS_3(s)}{dt} = \sum_{i=0}^{M+4} \boldsymbol{\eta}_i \Gamma_{i+1,2}(s) \tag{3.39}$$

where

$$\boldsymbol{\eta}_i = \alpha \frac{3}{\varpi_{i+4} - \varpi_{i+1}} (\boldsymbol{\tau}_{i+1} - \boldsymbol{\tau}_i) \quad \text{with} \quad \boldsymbol{\eta}, \boldsymbol{\tau} \in \mathbb{R}^3 \tag{3.40}$$

For ease of analysis, scalar η is denoted as the equivalent of $\boldsymbol{\eta}$ on a single axis and $E_\mathrm{s} = [\eta_0, \eta_1 \dots \eta_{M+4}]^\mathrm{T}$ denotes a vector of η on that corresponding axis. By substituting ϖ with the knot vector of the clamped cubic NUBS V_knot, matrix that project T_s into E_s is formed as

$$K_\mathrm{s} = \alpha \begin{bmatrix} -3 & 3 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & -\frac{3}{2} & \frac{3}{2} & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & \dots & 0 \\ \vdots & & & & \ddots & & & & \vdots \\ 0 & 0 & \dots & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & -\frac{3}{2} & \frac{3}{2} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & -3 & 3 \end{bmatrix}$$

which gives

$$E_\mathrm{s} = K_\mathrm{s} T_\mathrm{s} \tag{3.41}$$

By taking derivation on Equation (3.39), it arrives at

$$\frac{d^2 S_3(s)}{dt^2} = \sum_{i=0}^{M+3} \boldsymbol{\gamma}_i \Gamma_{i+2,1}(s) \tag{3.42}$$

Here

$$\boldsymbol{\gamma}_i = \alpha \frac{2}{\varpi'_{i+3} - \varpi'_{i+1}} (\boldsymbol{\eta}_{i+1} - \boldsymbol{\eta}_i) \tag{3.43}$$

where ϖ' is the knot vector by removing the first and last members from V_{knot} for clamped cubic NUBS. In fact, it is the knot vector for a clamped quadratic NUBS and can be written in the form of $[0, 0, 0, 1, 2, 3, \dots]$. Again, scalar γ is denoted as the equivalent of $\boldsymbol{\gamma}$ on a single axis and $\Lambda_{\mathrm{s}} = [\gamma_0, \gamma_1 \dots \gamma_{M+3}]^{\mathrm{T}}$ denotes a vector of γ on that corresponding axis. Substituting the new knot vector, the matrix

$$K'_{\mathrm{s}} = \alpha \begin{bmatrix} -2 & 2 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & \dots & 0 \\ \vdots & & & & \ddots & & & & \vdots \\ 0 & 0 & \dots & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & -2 & 2 \end{bmatrix}$$

can be derived with

$$\Lambda_{\mathrm{s}} = K'_{\mathrm{s}} E_{\mathrm{s}} \tag{3.44}$$

With Equation 3.44 and 3.41, a transformation between T_{s} and Λ_{s} is made possible:

$$\begin{aligned} \Lambda_{\mathrm{s}} &= K'_{\mathrm{s}} K_{\mathrm{s}} T_{\mathrm{s}} \\ &\equiv L T_{\mathrm{s}} \end{aligned} \tag{3.45}$$

where matrix L is

$$L = \alpha^2 \begin{bmatrix} 6 & -9 & 3 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & \frac{3}{2} & \frac{-5}{2} & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \vdots & & & & \ddots & & & & \vdots \\ 0 & 0 & \dots & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & \frac{3}{2} & \frac{-5}{2} & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 6 & -9 & 3 \end{bmatrix}.$$

Then, the velocity constraints can be projected into the space of control points vector T_{s} as

$$\dot{S}_{3\min} \leq K_{\mathrm{s}i} T_{\mathrm{s}} \leq \dot{S}_{3\max}, \quad \forall i = \{0, 1, ..., M+4\}, \tag{3.46}$$

where $K_{\mathrm{s}i}$ denotes the ith row of K_{s}. The acceleration constraints then can be projected into the constraints of control points vector T_{s} as

$$\ddot{S}_{3\min} \leq L_i T_{\mathrm{s}} \leq \ddot{S}_{3\max}, \quad \forall i = \{0, 1, ..., M+3\}, \tag{3.47}$$

where L_i denotes the ith row of L.

For the boundary conditions, one should fix part or all of the first and last three elements of T_{s}. This is formed as linear equality constraint

$$A_{\mathrm{eq}} T_{\mathrm{s}} = b_{\mathrm{eq}} \tag{3.48}$$

For example, if the vehicle is to start and end at a hover state, the first three control points shall be equal, and the last three control points shall also be equal.

Equations 3.28, 3.46, 3.47 and 3.48 form a typical convex quadratic programming problem which can be solved numerically in an efficient manner [58]. In Figure 3.1, it shows a single axis solution with derivative constraints.

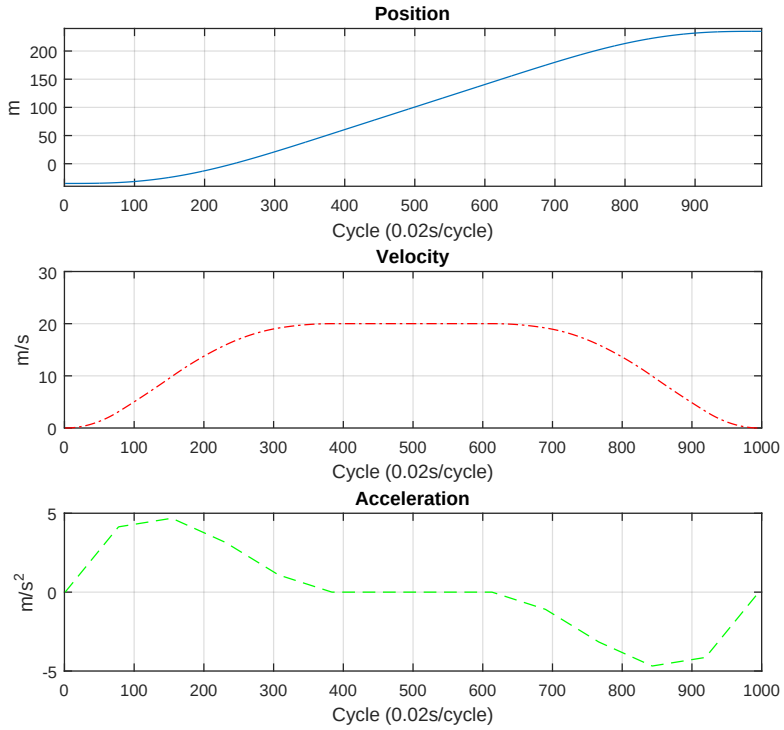


Figure 3.1: B-spline trajectory: Limit acceleration $5\,\mathrm{m/s^2}$ and velocity $20\,\mathrm{m/s}$

3.3.3 Time Vector Optimization

For many cases, the vector D_{time} is not explicitly given. And it becomes necessary to generate the time vector D_{time} while iteratively minimizing the jerk trajectory. Similar idea can be found in [10] with an implementation on polynomials. Let $\Upsilon_i = t_{i+1} - t_i$, which stands for the time difference between two adjacent members of D_{time}, a new optimization problem is formulated as

$$\min_{D_\varpi} f_\varpi(D_\varpi), \quad \text{with } D_\varpi = [\Upsilon_0 \dots \Upsilon_{I_\mathrm{d}-1}] \quad \text{s.t. } \Upsilon_i > 0. \tag{3.49}$$

A gradient descent method can be utilized for solution. The gradient of function f_ϖ is calculated by small perturbation method

$$\nabla f_\varpi = \frac{f_\varpi(D_\varpi + h g_\varpi) - f_\varpi(D_\varpi)}{h} \tag{3.50}$$

where h is a small number at the level of 10^{-6}, g_ϖ is a perturbation vector. With the numerically obtained gradient, the gradient descent method is performed using backtrack line search. Unlike the discussion in previous section, here the optimization includes

all three axis, though they are still done separately. Denote $\boldsymbol{D}_{\mathrm{s}} = [D_{\mathrm{s}x}, D_{\mathrm{s}y}, D_{\mathrm{s}z}]$ as a 3D version of D_{s} in Equation 3.15, and $\boldsymbol{T}_{\mathrm{s}} = [T_{\mathrm{s}x}, T_{\mathrm{s}y}, T_{\mathrm{s}z}]$ as a 3D version of T_{s}. The algorithm for performing time vector optimization is given as

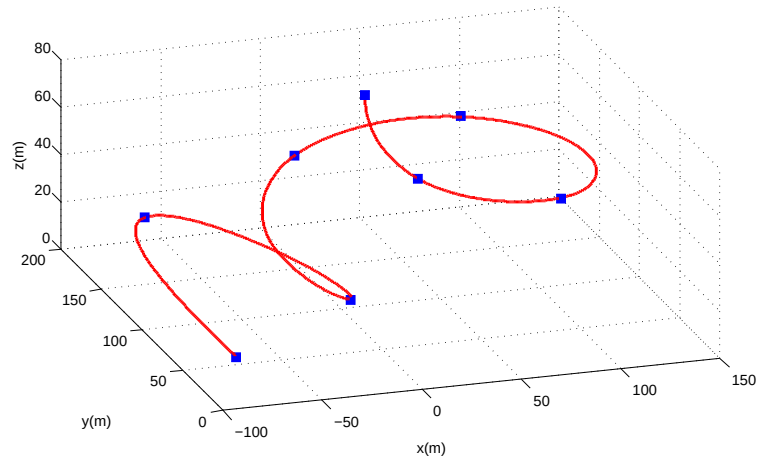
Algorithm 5 Optimization for time segmentation

Input: $D_{\varpi_{\mathrm{ini}}}$, $\boldsymbol{D}_{\mathrm{s}}$,$Step_{\mathrm{ini}}$
Output: D^{*}_{ϖ}, $\boldsymbol{T}^{*}_{\mathrm{s}}$
$D_{\varpi} \leftarrow D_{\varpi_{\mathrm{ini}}}$
$(f_{\varpi}, \boldsymbol{T}_{\mathrm{s}}) = \mathrm{QOptimize}(D_{\varpi}, \boldsymbol{D}_{\mathrm{s}})$
while Terminal condition not satisfied **do**
    $grad = \mathrm{Perturbe}(D_{\varpi}, \boldsymbol{T}_{\mathrm{s}})$
    $StepLength = Step_{\mathrm{ini}}$
    for i from 1 to K **do**
        $D^{\mathrm{new}}_{\varpi} = D_{\varpi} + StepLength \cdot grad$
        $(f^{\mathrm{new}}_{\varpi}, \boldsymbol{T}^{\mathrm{new}}_{s}) = \mathrm{QOptimize}(D^{\mathrm{new}}_{\varpi}, \boldsymbol{D}_{\mathrm{s}})$
        if $f^{\mathrm{new}}_{\varpi} \leq f_{\varpi} - StepLength \cdot grad \cdot grad'$ **then**
            $f_{\varpi} = f^{\mathrm{new}}_{\varpi}$
            $D_{\varpi} \leftarrow D^{\mathrm{new}}_{\varpi}$
            $\boldsymbol{T}_{\mathrm{s}} \leftarrow \boldsymbol{T}^{\mathrm{new}}_{\mathrm{s}}$
            BREAK
        else
            $StepLength = StepLength \cdot 0.9$
$D^{*}_{\varpi} \leftarrow D_{\varpi}$
$\boldsymbol{T}^{*}_{\mathrm{s}} \leftarrow \boldsymbol{T}_{\mathrm{s}}$

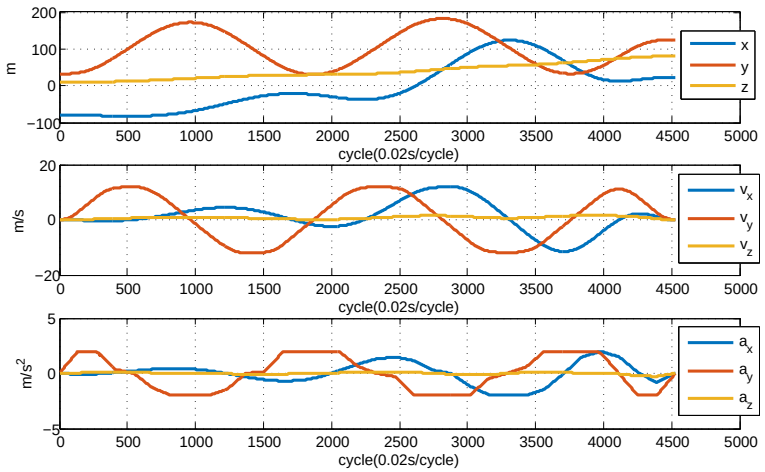
Function $(f_{\varpi}, \boldsymbol{T}_{\mathrm{s}}) = \mathrm{QOptimize}(D_{\varpi}, \boldsymbol{D}_{\mathrm{s}})$
$(f_x, T_{\mathrm{s}_x}) = \mathrm{QuadConvexOptimize}(D_{\varpi}, D_{\mathrm{s}x}, Constraint_x)$
$(f_y, T_{\mathrm{s}_y}) = \mathrm{QuadConvexOptimize}(D_{\varpi}, D_{\mathrm{s}y}, Constraint_y)$
$(f_z, T_{\mathrm{s}_z}) = \mathrm{QuadConvexOptimize}(D_{\varpi}, D_{\mathrm{s}z}, Constraint_z)$
$f_{\varpi} = f_x + f_y + f_z + K_{\mathrm{time}} \sum_{i=0}^{I_{\mathrm{d}}-1} \Upsilon_i$
$\boldsymbol{T}_{\mathrm{s}} = [T_{\mathrm{s}_x}, T_{\mathrm{s}_y}, T_{\mathrm{s}_z}]$

Here, the function $\mathrm{QuadConvexOptimize}()$ solves for the single axis optimization problem presented in Section 3.3.2. Its output are the minimum values of cost function (f_x, f_y, f_z) and the corresponding control point vectors (T_{s_x}, T_{s_y}, T_{s_z}). Whereas function $\mathrm{Perturbe}()$ is an numerical implementation of Equation 3.50. Using the proposed algorithm, the trajectory is further smoothed over the time vector, and the cost from each axis is also combined. To start the search process, an initial guess of T_{s} can be set by equally split the time. The algorithm is capable to provide the smooth 3D flight trajectory. An example of such trajectory flying through several 3D waypoints performing zig-zag climbing and spiral maneuver is given in Figure 3.2. In this

example, only the position of the waypoint is given. Due to the dynamics of the rotorcraft, the horizontal and vertical constraints differ from each other. The horizontal constraints are $v_{hmax} = 12m/s, a_{hmax} = 2m/s^2$ whereas the vertical constraints are $v_{vmax} = 1.5m/s, a_{vmax} = 0.8m/s^2$. On the other hand, there are missions only require the vehicle to reach an endpoint but with no end state velocity and acceleration constraints. It can be achieved by adjusting the equality constraints. An example is given in Figure 3.3 where the maximum horizontal velocity is changed to $5m/s$. In Figure 3.4, it shows a 2D flight path where the end state is not constrained. The flight path is clamped (tangent to) the first and last legs formed by the control points following human intuition.



(a) Flight path



(b) Trajectory

Figure 3.2: 3D Trajectory obtained through time vector optimization

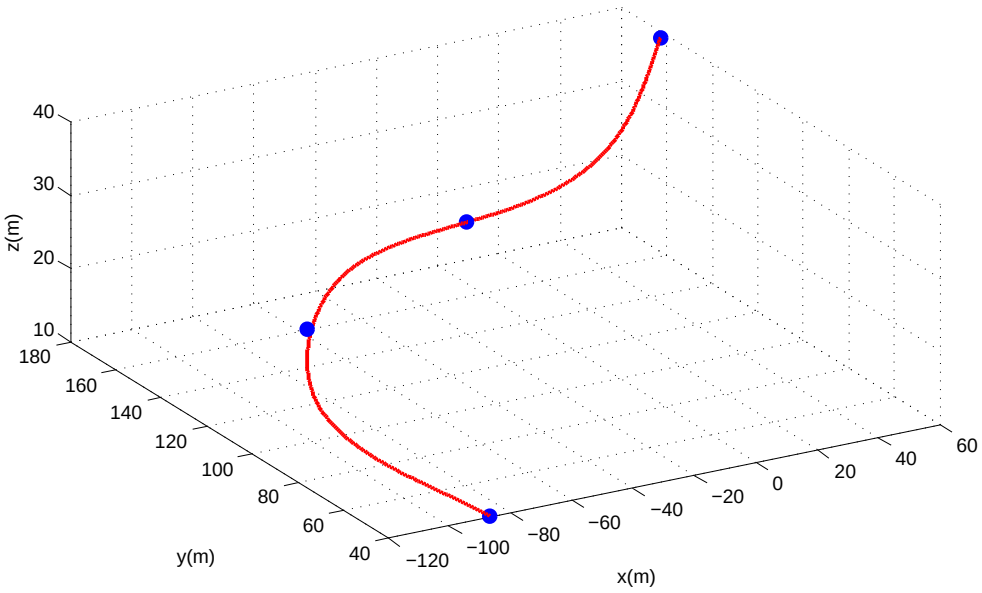
3.3.4 Reconstructing of Trajectory

After solving the trajectory optimization problem and obtaining the corresponding $T_{\mathrm{s}}{}^{*}$ for each axis, the trajectory needs to be re-constructed through Equation 3.14. Like the optimization, the reconstruction is done separately for each axis. But the base functions are defined recursively, it would be inefficient to reconstruct all the base functions and evaluate the trajectory. Here, the de Boor's algorithm (see Algorithm 6) is adopted for its ability to generate discretized trajectory of arbitrary frequency [60].

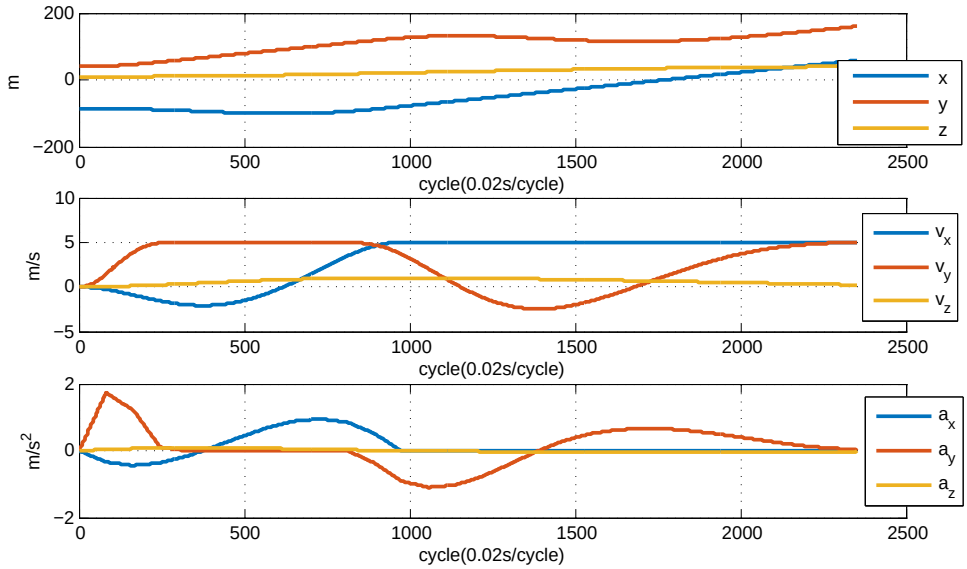
Algorithm 6 de Boor's algorithm

Input: Path parameter s_{o}, knot vector V_{knot}, control point vector T_{s}, order k_{o}
Output: Value of $S_{k_{\mathrm{o}}}(s_{\mathrm{o}})$
$k =$ The largest number satisfy $V_{\mathrm{knot}}(k) \leq s_0$
$X(:,1) = T_s(k - k_o + 1 : k, 1)$
for n from 1 to k_{o} **do**
    for m from n to k_{o} **do**
        $N_m = s_{\mathrm{o}} - V_{\mathrm{knot}}(k - k_{\mathrm{o}} + m)$
        if $N_m == 0$ **then**
            $weight = 0$
        else
            $c_{\mathrm{s}} = V_{\mathrm{knot}}(k + m - n + 1) - V_{\mathrm{knot}}(k - k_{\mathrm{o}} + m)$
            $weight = \dfrac{N_m}{c_{\mathrm{s}}}$
        $X(m,n) = (1 - weight) \cdot X(m-1, n-1) + weight \cdot X(m, n-1)$
$S_{k_{\mathrm{o}}}(s_{\mathrm{o}}) = X(k_{\mathrm{o}}, k_{\mathrm{o}})$

Note this algorithm translates the recursive definition of B-spline into cascaded loops. For the case of 3D splines, the algorithm is executed separately on each axis. Since $S_{k_{\mathrm{o}}}(s_{\mathrm{o}})$ is defined on the path parameter, to acquire the reference states regarding to time, Equation 3.7 is considered. The trajectory is now redefined as $S_{k_{\mathrm{o}}}(\alpha t_{\mathrm{o}})$ where $s_{\mathrm{o}} = \alpha t_{\mathrm{o}}$. Sometimes, the derivatives of the trajectory are also needed, the control points defined in Equation 3.41 and 3.44 can be used together with the clamped NUBS of reduced order to achieve the desired derivatives.

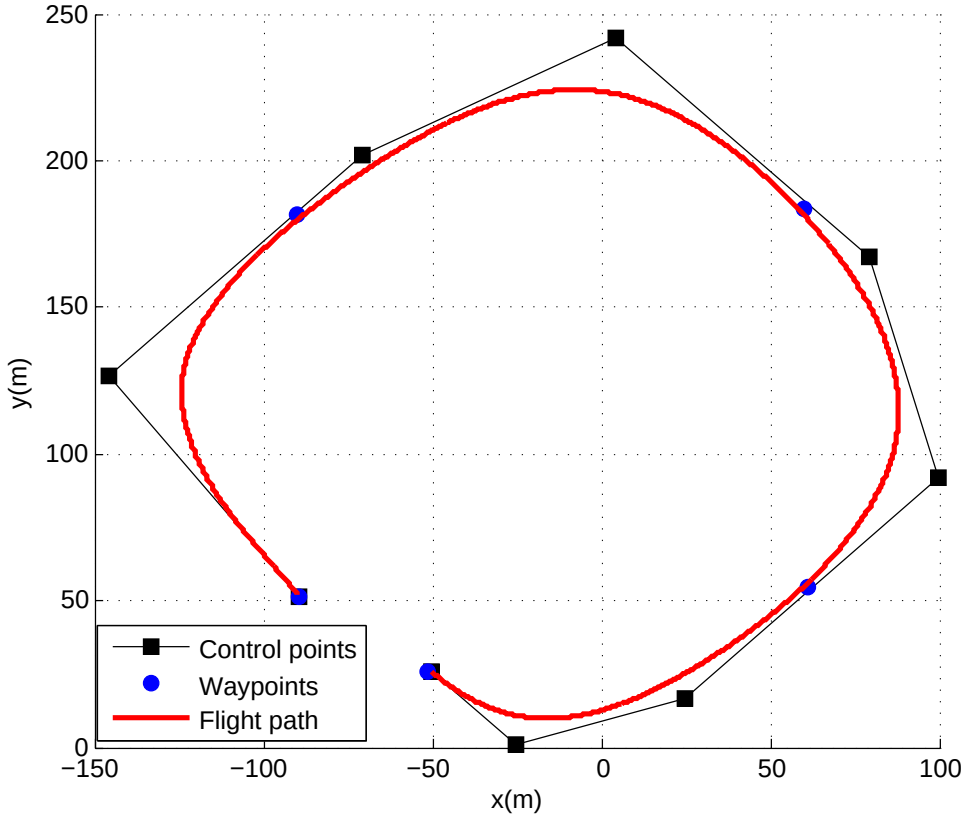


(a) Flight path

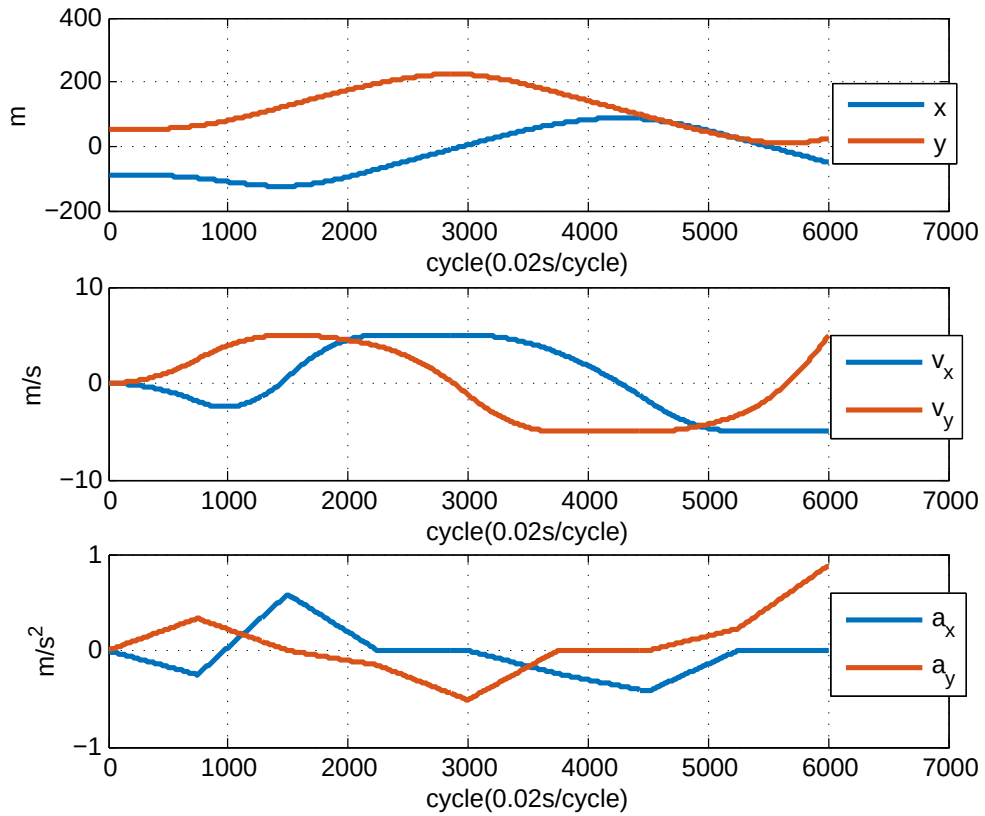


(b) Trajectory

Figure 3.3: Flight mission with end position constraint only



(a) Flight path



(b) Trajectory

Figure 3.4: Flight mission with no end constraint

3.4 UAV Calligraphy

Due to the involvement of various optimization techniques in the proposed B-spline method, it is mainly used for offline trajectory designing with powerful desktop computers. One interesting application is to generate trajectories that guide quad-copters to write calligraphy. The system consists of a touch screen HMI which allows the user to draw or write and a quad-rotor carrying the Chinese calligraphy brush beneath it. Together with a mission management system located on a separate desktop that connects the HMI to the quad-rotors and a VICON positioning system for feedback control. The quad-rotor will draw the content acquired from the HMI on a $150\,\mathrm{cm}\times150\,\mathrm{cm}$ paper. The quad-rotor adopted the cascaded control structure and utilize the RPT outer loop control law in Section 2.3. The system design is adopted from the MGCS in Chapter 2. The system overview of the UAV calligraphy system is illustrated in Figure 3.5. The human input is recorded discretely as a series of 2D points. However, they are not used directly as the vehicle's reference due to two reasons:

1. The human writing data sampled from the touch screen is noisy.

2. The dynamics of human's writing does not match the one of quad-rotor. For example, operator could write very fast on the touch screen thus creating problem for quad-rotor to track his input directly.

In our implementation, the handwriting is first extracted and represented as a series of waypoints. Then, the proposed B-spline trajectory generator produces the trajectory that both fits the vehicle's dynamics and matches the original handwriting. In order to sample the human input, a split and merge algorithm [62] is used to extract the characteristic points of handwriting. Here, characteristic points refer to the starting, ending and turning points that forms the skeleton of each stroke. The algorithm is illustrated in Figure 3.6 and consists of 5 steps:

1. Connect the first and last points A and B to form a straight line $A-B$.

2. Iterate the points between A and B, and find the point C which has the longest distance d_{longest} to the line $A-B$.

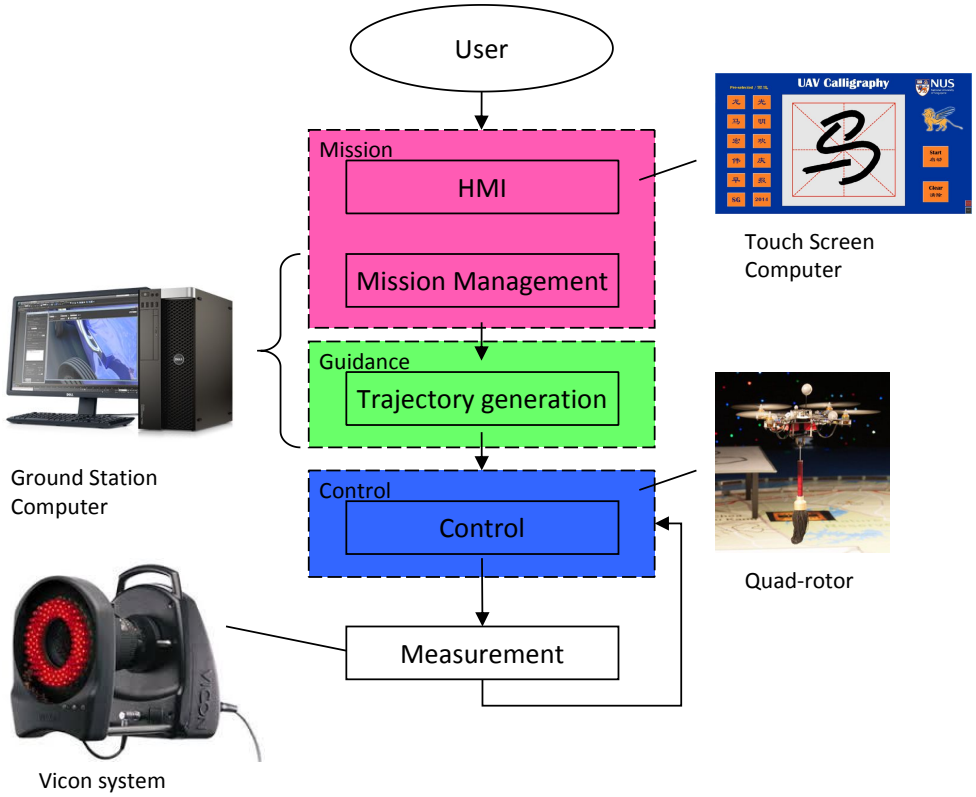


Figure 3.5: UAV calligraphy system

3. If the distance d_{longest} is smaller than a certain threshold, then points A and B are stored as the characteristic points. Algorithm stops.

4. Else, all three points A, B and C are stored as the characteristic points. The algorithm runs recursively for points between the line $A - C$ and the line $C - B$.

5. If d_{longest} falls below the threshold for all line segments, then the algorithm stops.

The extracted 2D characteristic points are used as reference for the B-spline trajectory generation algorithm. They can be considered as a 2D version of D_{s} in Equation 3.15 without time vector. For a smoother reference, Algorithm 5 is executed to generate the final trajectory. In order to start the gradient descent procedure, an initial guess of time vector is constructed by equally distributing time across D_{ϖ}. The resulting trajectories that resemble the user's handwriting are given in Figure 3.7. A comparison between results obtained before and after time vector optimization is made in Figure 3.8. Through time vector optimization, the cost will arrive at a new local minimum which is smaller than the cost induced by the initial time vector. In other words, the trajectory is

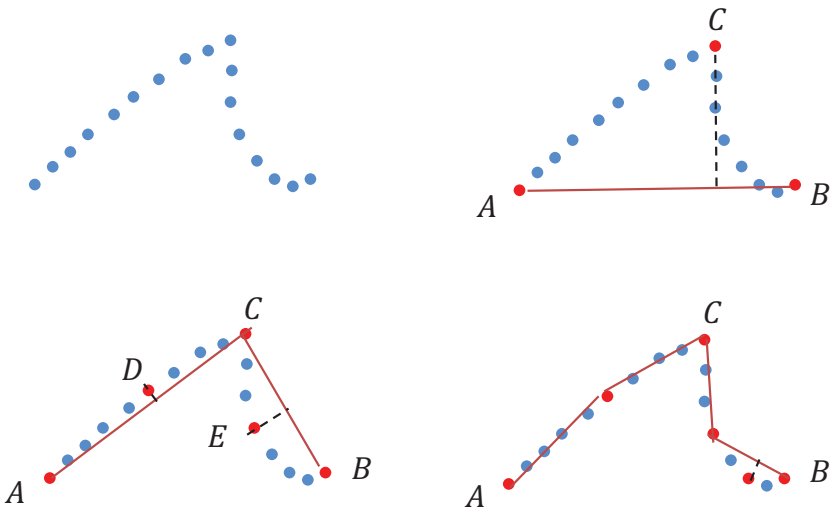
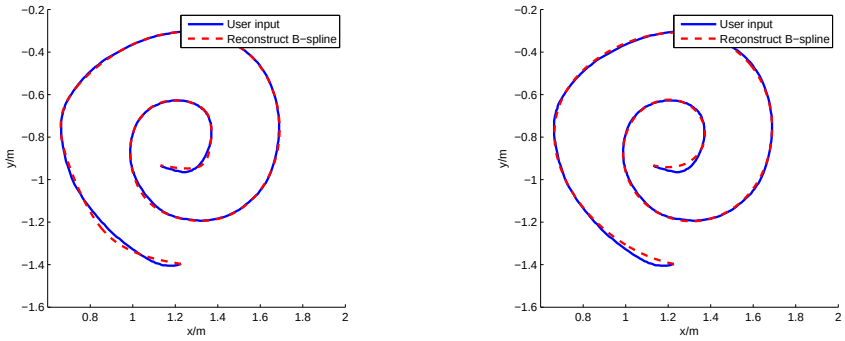


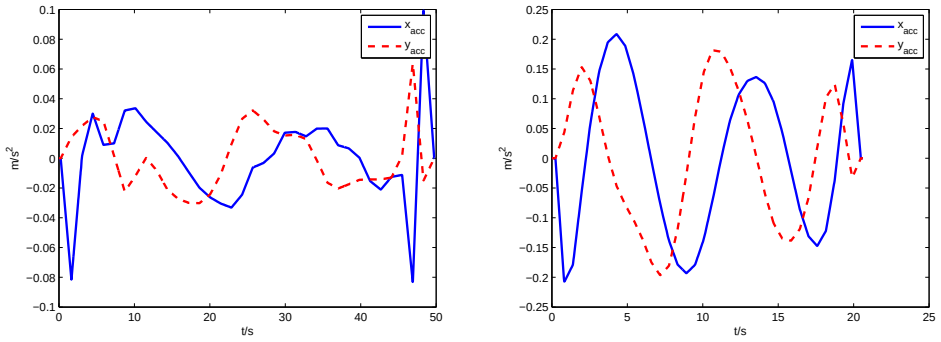
Figure 3.6: Split-and-merge sequence on continuous line segments

further smoothen and resembles the original user input more accurately. For the case of a vortex drawing (see Figure 3.7 and 3.8), the resulting acceleration reference (Figure 3.8) becomes much smoother after time vector optimization. In another case of writing a Chinese character *Guang*, the interpolation accuracy of the generated trajectory (Figure 3.9) is improved. In the experiments of drawing reconstruction, the B-spline trajectory generator takes about 3 to 10 seconds to interpolate the user's input (a typical Chinese character) on a desktop workstation. Then the quad-rotor writes the letter down by tracking the reference. Since the dynamics of the vehicle has been well considered in the planning stage, and the derivatives are limited to prevent pendulum effect of the calligraphy brush, an accurate tracking performance is achieved (see Figure 3.11). The proposed control law produces small error while countering the disturbances from the brush touching the paper. Finally, a comparison between the user's input, the generated trajectory, and the final written character by quad-rotor is made in Figure 3.12. Here, it writes the Chinese character *Cheng*. The above results have been published as in author's paper [13].



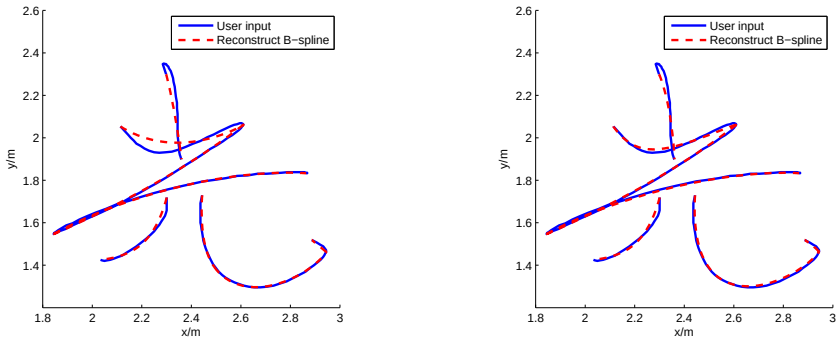
(a) Without time re-segmentation (b) With time re-segmentation

Figure 3.7: User input and generated spline of vortex drawing



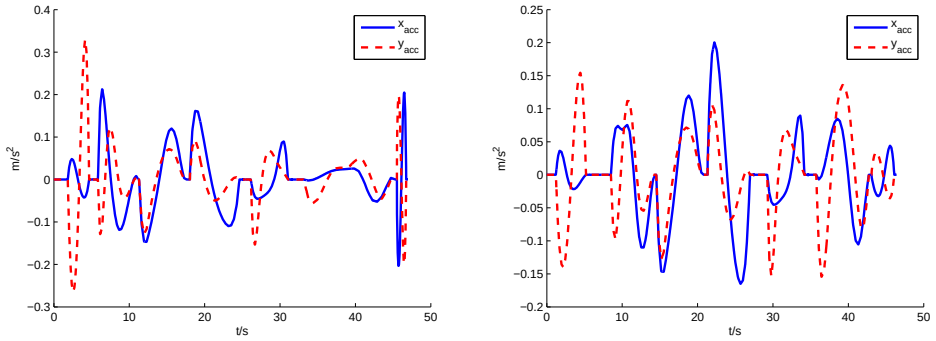
(a) Without time re-segmentation (b) With time re-segmentation

Figure 3.8: Generated spline's acceleration of vortex drawing



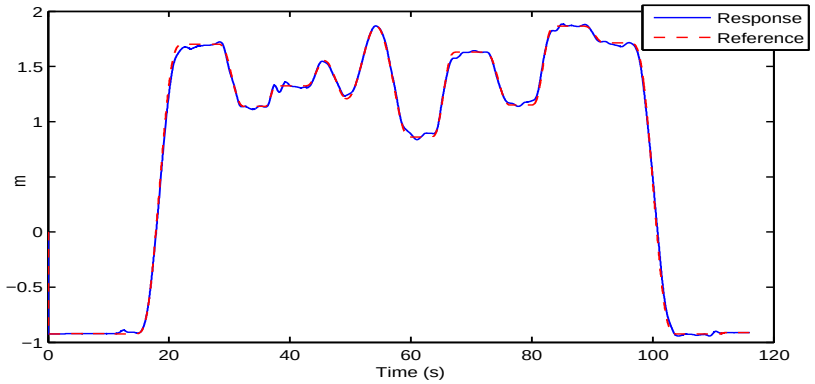
(a) Without time re-segmentation (b) With time re-segmentation

Figure 3.9: User input and generated spline of Chinese character *Guang*

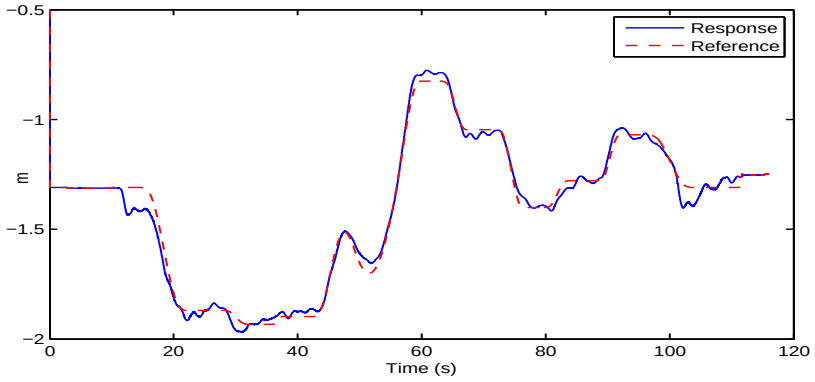


(a) Without time re-segmentation (b) With time re-segmentation

Figure 3.10: Generated spline's acceleration of Chinese character *Guang*

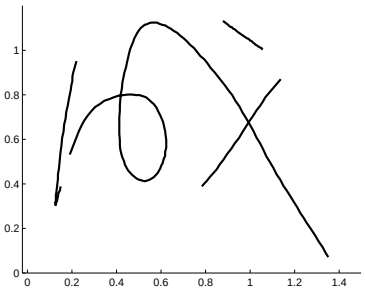


(a) x axis

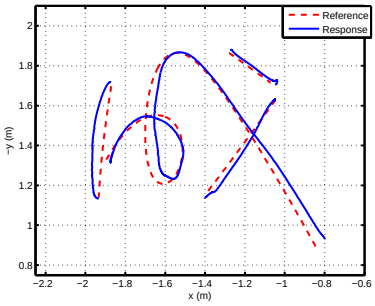


(b) y axis

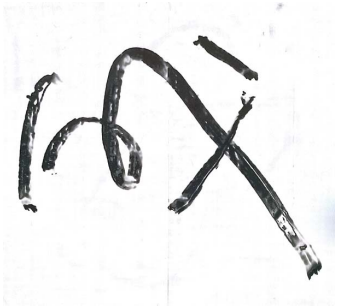
Figure 3.11: Tracking performance with real vehicle writing on paper



(a) User handwritten input



(b) Generated reference and its response



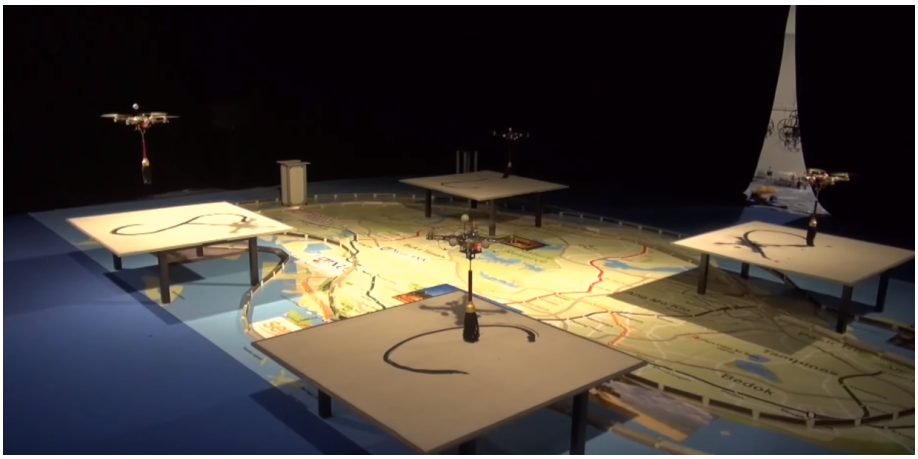
(c) Writing result

Figure 3.12: Comparison between user input, reference and outcome

3.5 Conclusion

In this chapter, a minimum derivatives B-spline trajectory planning method is presented, which produces trajectories with delicate maneuvers upon solving. This method typi-

(a) Multi drone light show



(b) UAV calligraphy

Figure 3.13: Demonstrations and stage performances

cally takes several seconds to finish with the time vector optimization. The resulting trajectory can be examined by designers or automated algorithms to determine its fitness. It becomes part of a reference generation toolbox for preparing demonstrations and stage performances (see Figure 3.13).

Many current popular trajectory generation methods for RUAV [10, 19] require iterative programming to produce derivative limited trajectory. The adopted algorithm only needs one quadratic programming for that. And a time vector optimization process is used to further smoothen the reference. It is also more suitable for RUAV compared to the interpolation based method widely adopted by CNC machining [63]. For methods

like [63], the computational complexity would increase due to the prolonged prediction horizon resulted from the low acceleration and jerk limits of RUAVs. Further, compared to more traditional path following based methods, the algorithm is capable of generating complex paths that the vehicle could track with high accuracy.

In the next chapter, online trajectory generation algorithms are presented, which considers to use low-cost platforms to perform complex tasks.

Chapter 4

Online Trajectory Generation Algorithm

In this chapter, online trajectory generation algorithms are covered with the special consideration for low-cost platforms. They are capable of generating dynamically feasible reference in the order of microseconds by utilizing the principals of sliding mode control (SMC). Section 4.1 introduces an online velocity reference generation algorithm for semi-autonomous flight with its application on safe fly zone mission. The online position reference generation method is depicted in Section 4.2 for fully-autonomous tasks. Both sections cover discussion on command filters and the TPBVP solvers that could 'predict' the future vehicle states without forward simulation.

Though many more advanced online trajectory generation algorithms exist, the covered method is more suitable for RUAVs with the limited computational power to execute missions that need accuracy in reference tracking. For example, many industrial inspection tasks require the vehicle to travel in the vicinity of obstacles and inside user-defined safe-fly-zone, while application like camera trajectory planning requires accurate velocity tracking.

4.1 Online Velocity Trajectory

4.1.1 State Constrained Sliding Mode Control

The algorithm presented in the Chapter 3 provides smooth trajectory with limited derivatives. However, the involvement of numerical optimization limits its realtime application on low-cost platforms. A typical example of such a vehicle is the K-lion quad-rotor (see Figure 4.1). The vehicle carries less than $500\,\mathrm{g}$ of payload and is equipped with a

Figure 4.1: K-Lion micro quad-rotor

PixHawk flight controller [64] which has a $168\,\mathrm{MHz}$ Cortex M4F Central Processing Unit (CPU). The storage system has a Random-access Memory (RAM) of $256\,\mathrm{kB}$ and Read-only Memory (ROM) of $2\,\mathrm{MB}$. All the operating system, sensor filtering, communication, mission management and trajectory generation rely on the same onboard computer. As a comparison, a typical laptop now equips with a $2.4\,\mathrm{GHz}$ quad-core CPU, $8\,\mathrm{GB}$ RAM, and $1\,\mathrm{TB}$ of hard disk. It is very difficult to implement a numerical optimization based trajectory generator with such a setup due to its limited ROM space. On the other hand, the dynamics of the rotorcraft do not become simpler with the decreasing in size and weight. By reviewing the dynamic feasibility in Section 3.1, the minimum requirements for the online trajectory generation algorithm are reduced to:

1. Maintain continuity on the acceleration reference.

2. Allow both the acceleration and velocity of the trajectory to be limited.

In order to implement an online and realtime trajectory generator on a micro-size rotorcraft such as the one in Figure 4.1, the above requirements are further simplified

Figure 4.2: A Futaba RC controller

into

$$\begin{aligned} |v_x| \leq v_{\max x} \quad |v_y| \leq v_{\max y} \quad |v_z| \leq v_{\max z} \\ |a_x| \leq a_{\max x} \quad |a_y| \leq a_{\max y} \quad |a_z| \leq a_{\max z} \\ |u_{\mathrm{j}_x}| \leq u_{\mathrm{j}\max x} \quad |u_{\mathrm{j}_y}| \leq u_{\mathrm{j}\max y} \quad |u_{\mathrm{j}_z}| \leq u_{\mathrm{j}\max z} \end{aligned} \tag{4.1}$$

where $u_{\mathrm{j}_x}, u_{\mathrm{j}_y}, u_{\mathrm{j}_z}$ are the jerk (derivative of acceleration) on corresponding axis. Since the jerks are now limited, the continuity of acceleration is guaranteed. With the simplified requirements, a sliding mode control (SMC) based trajectory generator is developed. The proposed algorithm is capable of achieving velocity command tracking while using the minimum amount of computational power.

This section focuses on generating the trajectory that tracks the user's velocity command. This is commonly referred to a semi-auto flight mode among the Radio Control (RC) hobby community. A commonly used RC flight controller is shown in Figure 4.2. The joysticks pointed by red arrows are the main control sticks that are responsible for sending commands to the vehicle at high frequency (typically more than $40\,\mathrm{Hz}$). Commonly, the commands sent by the controller are given as the reference to the inner loop's controlled outputs. For the operator to fly the rotorcraft in such a mode, a constant visual feedback on the vehicle's attitude is needed. Otherwise, tasks like hovering or path following become impossible. On the contrary, if the control sticks commands are mapped to velocity reference, a more user-friendly flight experience is achieved. For straight line flying, a simple pull or push to the control stick is enough. Whereas for hovering, just leaving the control stick at zero position will do the trick. In such a

mode, the operator could focus more on controlling the onboard pieces of equipment to process actual tasks. With the control structure in Section 2.3 and the feasibility requirements in Section 4.1.1, it is obvious that the trajectory generator is subjected to both input and state constraints. Considering the computational limitations, an SMC based trajectory generation technique is adopted. Consider a SISO double integrator system described by

$$\begin{aligned} \dot{v} &= a \\ \dot{a} &= u_{\mathrm{j}} \end{aligned} \tag{4.2}$$

where v stands for velocity, a stands for acceleration and u_{j} is the jerk. The trajectory generation law needs to provide reference trajectory that steers the v to some

$$v_{\mathrm{ref}} \in [-v_{\max}, v_{\max}]$$

while satisfying

$$\begin{aligned} -a_{\max} \leq a \leq a_{\max} \\ -u_{\mathrm{j}_{\max}} \leq u_{\mathrm{j}} \leq u_{\mathrm{j}_{\max}} \end{aligned}$$

To fulfill this objective, a discontinuous control law from [65] is defined as

$$u_{\mathrm{j}}(t) = \begin{cases} -u_{\mathrm{j}_{\max}} & if \quad (v,a) \in S_1 \cup (S_4 \cap S_5) \cup S_7 \\ u_{\mathrm{j}_{\max}} & if \quad (v,a) \in S_2 \cup (S_3 \cap S_6) \cup S_8 \end{cases} \tag{4.3}$$

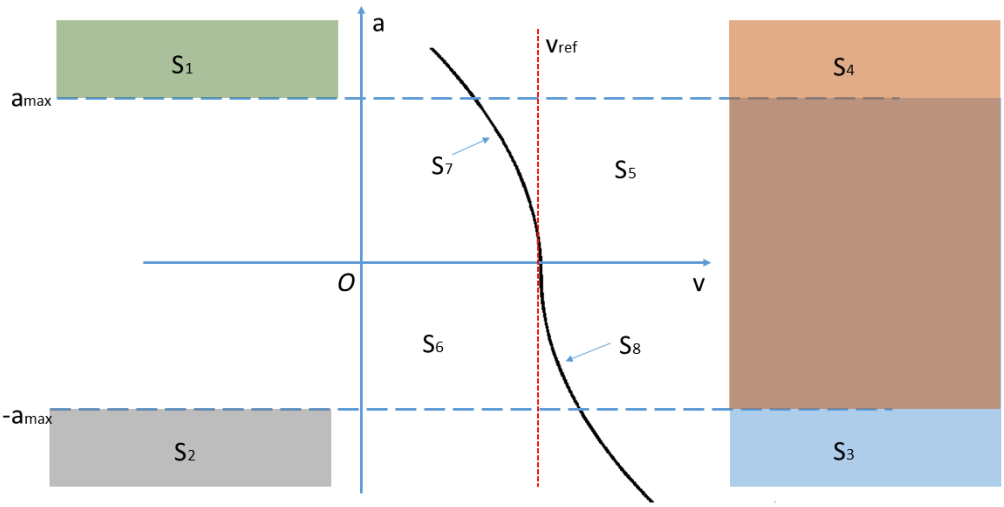


Figure 4.3: Plot of regions 1 to 8

where

$$S_1 \triangleq \{(v,a) : a \geq a_{\max}\}$$

$$S_2 \triangleq \{(v,a) : a \leq -a_{\max}\}$$

$$S_3 \triangleq \{(v,a) : a \leq a_{\max}\}$$

$$S_4 \triangleq \{(v,a) : a \geq -a_{\max}\}$$

$$S_5 \triangleq \{(v,a) : v + \frac{a\,|a|}{2u_{\mathrm{j}_{\max}}} > v_{\mathrm{ref}}\}$$

$$S_6 \triangleq \{(v,a) : v + \frac{a\,|a|}{2u_{\mathrm{j}_{\max}}} < v_{\mathrm{ref}}\}$$

$$S_7 \triangleq \{(v,a) : v + \frac{a\,|a|}{2u_{\mathrm{j}_{\max}}} = v_{\mathrm{ref}} \text{ and } a > 0\}$$

$$S_8 \triangleq \{(v,a) : v + \frac{a\,|a|}{2u_{\mathrm{j}_{\max}}} = v_{\mathrm{ref}} \text{ and } a < 0\}$$

The regions S1 to S8 can also be represented graphically, as illustrated in Figure 4.3. From Figure 4.3, it is clear that S_7 and S_8 forms a sliding mode. The purpose of the control law is then to steer the system onto the sliding mode. The proof on stability and convergence is given in [65]. Though the control law seems complex, the physical meaning it implies is rather straight forward, which can be summarized as

1. If the acceleration a violate its constraints. Manipulate u_{j} to bring it inside. This stands for the region S_1 and S_2.

2. From the current state, simulate steering the acceleration down to zero, and check the resulting end velocity which is denoted as v_{end}. And it can be expressed as

$$v_{\text{end}} = v + \frac{a\,|a|}{2u_{\text{j}_{\max}}}$$

If $v_{\text{end}} > v_{\text{ref}}$, it means even the acceleration is steered to zero immediately, the resulting end velocity is too large. In order to decrease the end velocity, control action $u_{\text{j}} = -u_{\text{j}_{\max}}$ should be taken providing the acceleration does not violate its constraints (not to violate condition 1). This result stands for the region $S_4 \cap S_5$.

If $v_{\text{end}} < v_{\text{ref}}$, it means the resulting end velocity is too small. Therefore, control action $u_{\text{j}} = u_{\text{j}_{\max}}$ should be taken to increase the end velocity providing the acceleration does not violate its constraints. This result stands for the region $S_3 \cap S_6$.

Finally, if $v_{\text{end}} = v_{\text{ref}}$, it implies the correct control action is just to steer the acceleration to zero. Hence the value of u_{j} depends on the current acceleration which stands for region S_7 and S_8.

Once the physical meaning is fully understood, an efficient implementation of the control law is proposed. To minimize the consumption of CPU resources, instead of scanning through S_1 to S_8 and searching the correct region that system belongs to, the algorithm adopts a nested checking procedure.

Algorithm 7 State constrained SMC for double integrator

1: Input: v_{ref}, $a_{\max}$, $u_{\text{j}_{\max}}$, v, a
2: Output: u_{j}^*
3: $v_{\text{end}} = v + \dfrac{a\,|a|}{2u_{\text{j}_{\max}}}$
4: $a_{\text{cruise}} = \text{sign}(v_{\text{ref}} - v_{\text{end}}) \cdot a_{\max}$
5: $u_{\text{j}}^* = u_{\text{j}_{\max}} \cdot \text{sign}(a_{\text{cruise}} - a)$

Line 3 calculates the end velocity and differentiate whether the current state is in S_5 or S_6. Line 4 determines the desired acceleration target a_{cruise}. Since a_{cruise} is limited, it effectively checks the region of S_1 to S_4. Note if $v_{\text{ref}} = v_{\text{end}}$ then $a_{\text{cruise}} = 0$, the regions S_7 and S_8 are inspected by the last line of the code.

The simulation results of the proposed implementation are shown in Figure 4.4

and Figure 4.5 with $a_{\max} = 2$ and $u_{\mathrm{j}\max} = 2$. In Figure 4.4, a system with initial acceleration violating the state constraint has been successfully stabilized by bring it to $v = 0,\ a = 0$. In Figure 4.5, a system starts at $v = 0,\ a = 0$ is steered to $v = 5,\ a = 0$. All the state and input constraints are fulfilled with our implementation.

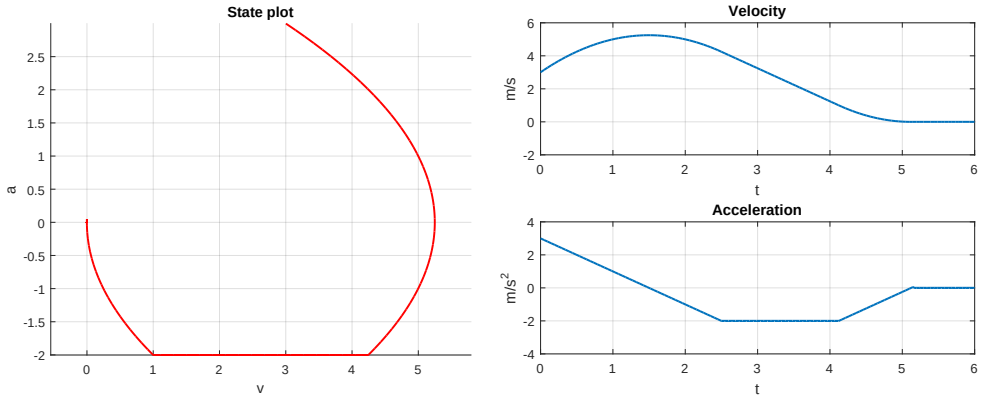


Figure 4.4: Velocity trajectory: initial acceleration violate limits

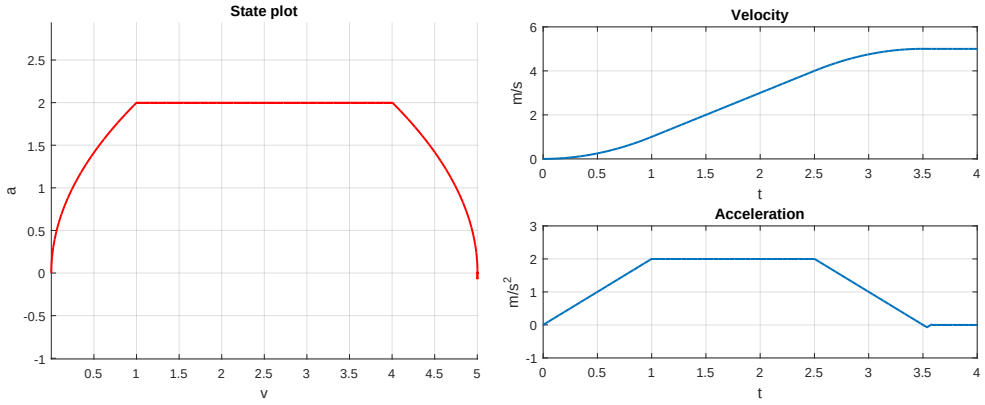
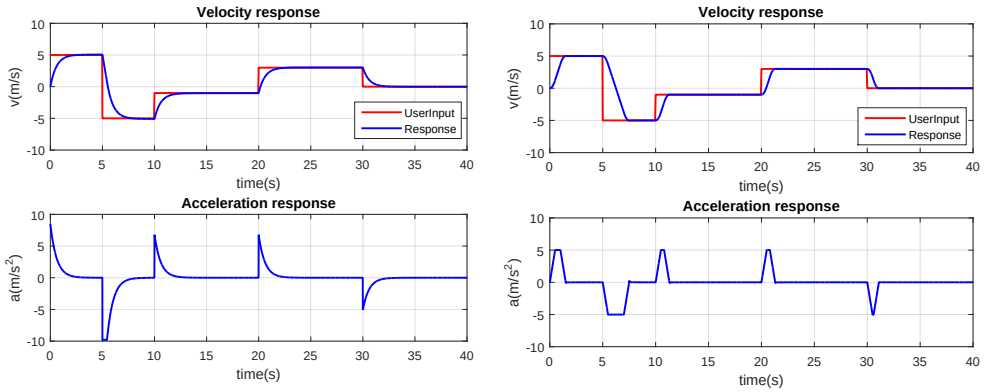


Figure 4.5: Velocity trajectory: steering to velocity target other than zero

4.1.2 Velocity Command Filter

Due to the requirement on system robustness, Algorithm 7 is modified into a command generator (command filter) rather than an outer loop controller.

The relationship between the command generator and the controller is shown in Figure 4.7. The command generator takes in the raw user command and generates dynamically feasible trajectories that can be tracked by the controller. A reference governor is responsible for checking the feasibility of the trajectory, resetting or modifying the user input when necessary. For example, if the human operator leads the vehicle to



(a) Without command generator (b) With command generator

Figure 4.6: Tracking of velocity reference

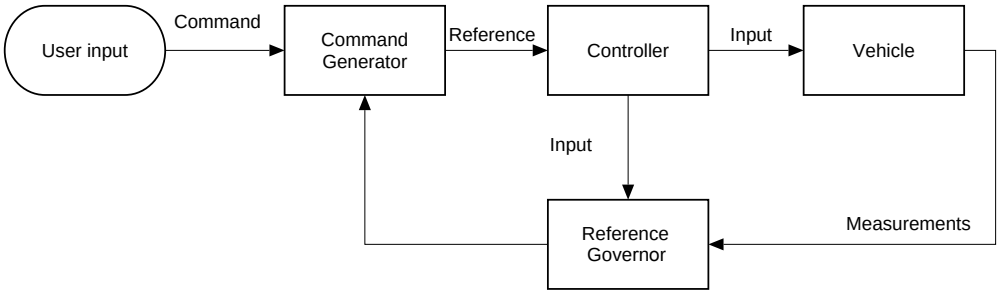


Figure 4.7: Command generating system

fly out of the safety mission zone, his command will be overwritten to avoid a safety hazard. The reference governor is discussed in detail in Section 4.1.5 and 4.2.4. The command generator is given in Algorithm 8. It is quite similar to Algorithm 7 except the usage of internal states (p_{I}, v_{I} and a_{I}) that turns the system into a filter. Function $\mathrm{ForwardSimulate}()$ generates filtered reference for position, velocity and acceleration (p_{rft}, v_{rft}, a_{rft} respectively) using an internal triple integrator model at a predefined sampling time t_{samp}. The advantages of utilizing the command generator are:

1. It results in a smoother flight experience by generating a continuous and limited acceleration profile. Figure 4.6 illustrates the simulation results of velocity reference tracking based on a quad-rotor outer loop model. In Figure 4.6(a) the user input is passed directly to the vehicle's velocity controller whereas in Figure 4.6(b) it is filtered by the command generator. Though the tracking performance is acceptable for both cases, the acceleration response without command filter suggests a sudden change on $\boldsymbol{T}_{\Sigma}$ when the user switches the velocity command.

Algorithm 8 State constrained velocity command generator

Input: v_{ref}, $a_{\max}$, $u_{\mathrm{j}_{\max}}$, v, a, p, t_{samp}
Output: p_{rft} v_{rft} a_{rft}
if !Initialized **then**
    $v_{\mathrm{I}} = v$
    $p_{\mathrm{I}} = p$
    $a_{\mathrm{I}} = a$
    Initialized = TRUE;
$v_{\mathrm{end}} = v_{\mathrm{I}} + \dfrac{a_{\mathrm{I}} \left| a_{\mathrm{I}} \right|}{2u_{\mathrm{j}_{\max}}}$
$a_{\mathrm{cruise}} = \mathrm{sign}(v_{\mathrm{ref}} - v_{\mathrm{end}}) \cdot a_{\max}$
$u_{\mathrm{j}}^{*} = u_{\mathrm{j}_{\max}} \ \mathrm{sign}(a_{\mathrm{cruise}} - a_{\mathrm{I}})$
$(p_{\mathrm{I}}, v_{\mathrm{I}}, a_{\mathrm{I}}) = \mathrm{ForwardSimulate}(p_{\mathrm{I}}, v_{\mathrm{I}}, a_{\mathrm{I}}, u_{\mathrm{j}}^{*}, t_{\mathrm{samp}})$
$p_{\mathrm{rft}} \leftarrow p_{\mathrm{I}}$
$v_{\mathrm{rft}} \leftarrow v_{\mathrm{I}}$
$a_{\mathrm{rft}} \leftarrow a_{\mathrm{I}}$

On the other hand, the one with command filter provides reference profile with limited acceleration and jerk, implying a smooth transform on $\boldsymbol{T}_{\Sigma}$.

2. It generates not only velocity but also position and acceleration reference, which helps to improve the performance of RPT control. Moreover, it means the same controller discussed in Section 2.3 can be used for velocity command tracking without any modification. It helps to reduce the design complexity of controller switching.

3. It can provide future information on the vehicle's state without forward simulation. The technique is explained with details in Section 4.1.3. With this information, various qualities of the trajectory could be examined efficiently.

Frame Translation

In real flight practice of RUAV under semi-auto mode, the operator would generate commands for the vehicle's velocity (in 3D space) and the heading angular rate. In tradition, these commands are assigned in a manner which is related to the vehicle's heading angle. For example, when the operator commands the vehicle to fly forward, the intention behind is flying in the same direction of vehicle's current heading. However, since the vehicle's position and velocity measurements are provided in the global coordinate, a frame translation is necessary to achieve the desired result. In our implementation, three coordinate systems are included. The global inertia frame $\mathcal{G}$, the

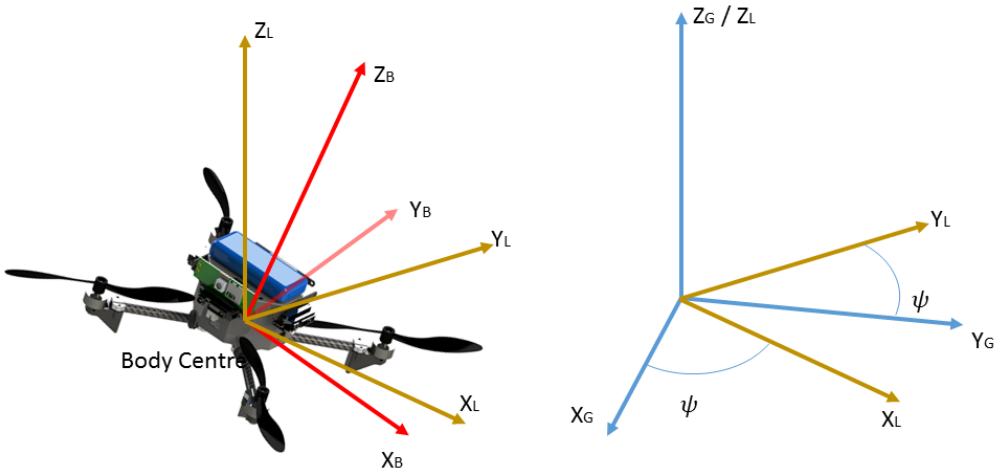


Figure 4.8: Coordinates utilized by the system

vehicle body frame $\mathcal{B}$ and an intermediate frame $\mathcal{L}$. Euler angles are used to define the roll, pitch and yaw (ϕ, θ, ψ respectively). The rotation matrix from $\mathcal{B}$ to $\mathcal{G}$ is given by ${}^{\mathrm{G}}\mathbf{R}_{\mathrm{B}} = {}^{\mathrm{G}}\mathbf{R}_{\mathrm{L}}\,{}^{\mathrm{L}}\mathbf{R}_{\mathrm{B}}$. The yaw rotation that affects the user's control is included in ${}^{\mathrm{G}}\mathbf{R}_{\mathrm{L}}$. During the semi-auto flight mode, the command generator should calculate reference in $\mathcal{L}$ whereas the translational controller does feedback control in $\mathcal{G}$. Therefore, the filtered references are first transformed through ${}^{G}\mathbf{R}_{L}$ before being passed to the translational controller. Experiment with a quad-rotor (see Figure 4.9) is performed. The platform comes with a tip-to-tip size of $1.7\,\mathrm{m}$, a maximum lift-off weight of $12\,\mathrm{kg}$. It could be piloted using RC transmitter under manual/semi-auto mode or perform fully automated mission through a handheld GCS. It possesses higher requirements on the guidance and control system compared to other similar platforms due to the larger size and heavier weight. Further, in the onboard implementation, techniques in Section 4.1.3 and 4.1.5 are combined into the velocity command filter so that a safe-fly-zone is guaranteed.

Flight experiment data is shown in Figure 4.10. Here, the heading angle is locked at around 45 degrees. The RC signal in Figure 4.10 corresponds to the forward speed command. The velocity reference shown in Figure 4.10 has already been transformed into the global coordinate. The human RC signal is well captured by the generated reference which is dynamically feasible and helps to improve the tracking performance. The maximum velocity tracking error is smaller than $0.5\,\mathrm{m/s}$ where the highest speed reaches $10\,\mathrm{m/s}$. With the proposed algorithm, long-range-semi-auto flight becomes

Figure 4.9: BlackLion-168 quad-rotor

possible. The data in Figure 4.11 illustrates a $5\,\mathrm{km}$ flight experiment where the vehicle cannot be visually observed.

4.1.3 Double Integrator TPBVP

For many situations, it is desirable to predict the future states of the vehicle at a given time instance. This is traditionally achieved by forward simulation which is time-consuming especially when the simulated time scale is long. For low-cost RUAVs, the onboard computational power is too weak for such approach.

The solution is a TPBVP solver which generates the entire trajectory that steers a double integrator from any initial state to the desired velocity v_{ref} with zero acceleration. Then a feedback controller is applied to track the trajectory. And the future state is now expressed as a *tunnel* around the reference with an estimated tracking error. When the trajectory is dynamically feasible, the tracking error would be small.

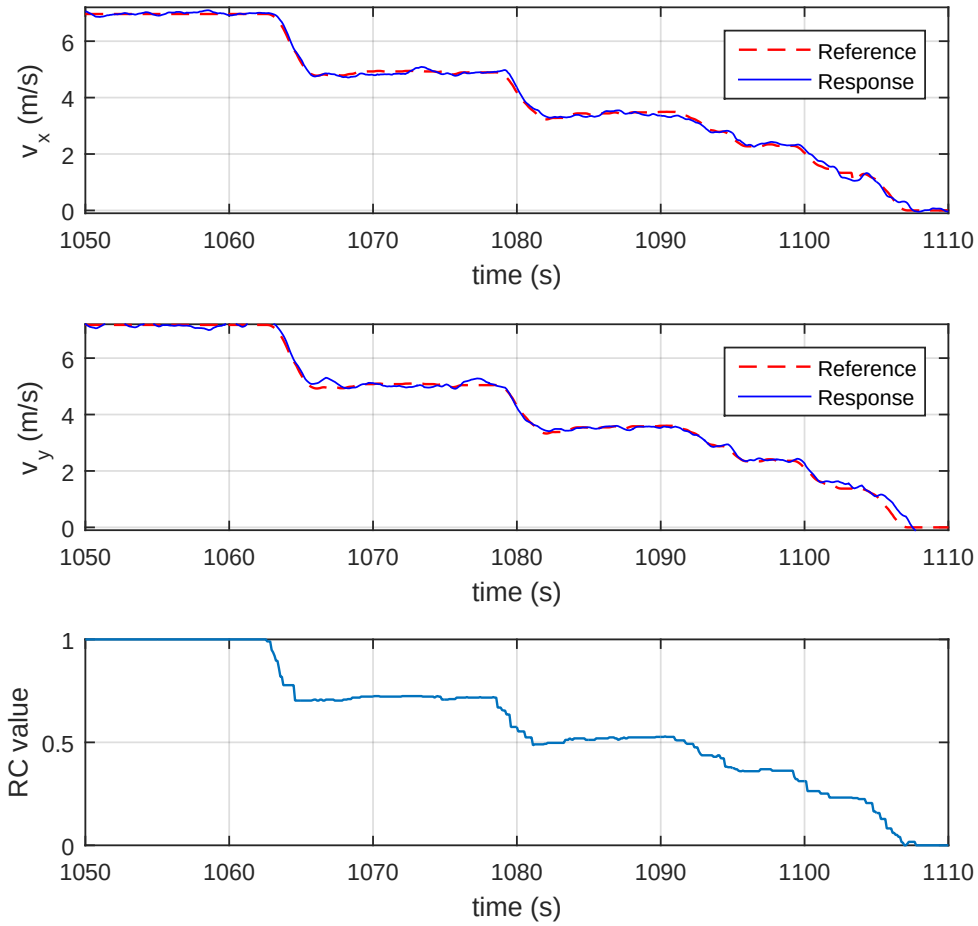


Figure 4.10: Experiment data for velocity tracking

The optimization target and the constraints are described as

$$
\begin{aligned}
&\min_{u_{\mathrm{j}}(t),t\in[0,T]} \left\{ J = \textstyle\int_{t=0}^{T} dt \right\} \ s.t. \\
&v(0) = v_0, \quad v(T) = v_{\mathrm{ref}} \\
&a(0) = a_0, \quad a(T) = 0 \\
&\dot{v}(t) = a(t) \\
&\dot{a}(t) = u_{\mathrm{j}}(t) \\
&-a_{\max} \leqslant a(t) \leqslant a_{\max}, \qquad \forall t \in [0,T] \\
&-u_{\mathrm{j}_{\max}} \leqslant u_{\mathrm{j}}(t) \leqslant u_{\mathrm{j}_{\max}}, \quad \forall t \in [0,T]
\end{aligned}
\tag{4.4}
$$

For such a problem, the input of the double integrator system u_{j} can only be chosen between $-u_{\mathrm{j}_{\max}}$, $u_{\mathrm{j}_{\max}}$ and 0. This strategy is often called bang-zero-bang control. With such a choice, the acceleration profile would form a trapezoidal or wedge shape

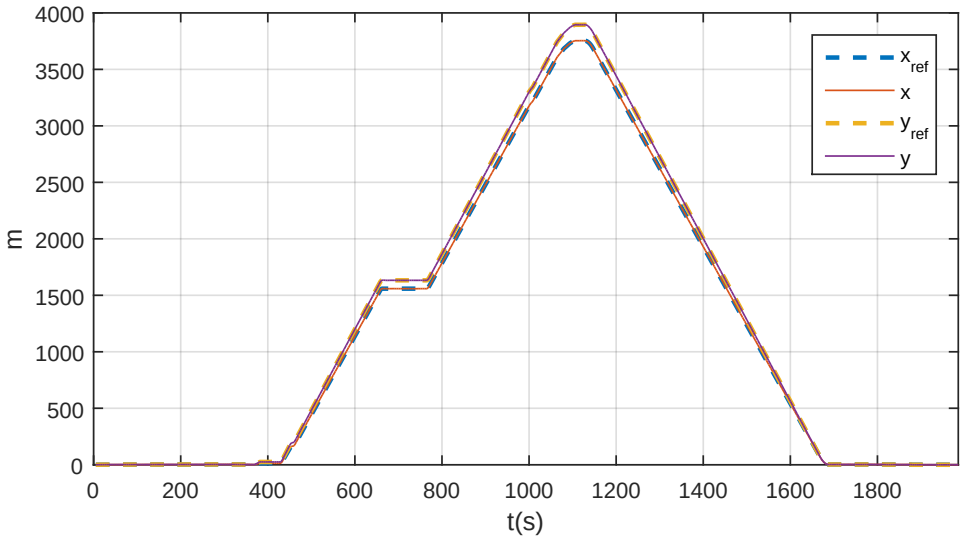


Figure 4.11: Experiment data for long distance flight

that consists of at most three phases as shown in Figure 4.12. The method is based on [59] with an additional zero acceleration cruise direction considered. An efficient implementation is given in Algorithm 9. Extension as guidance algorithms is covered in the following sections. Denote function

$$R_{\mathrm{v}}(t, v_0, a_0, u_{\mathrm{j}}) = v_0 + a_0 \cdot t + \frac{1}{2} u_{\mathrm{j}} \cdot t^2$$

These three phases could be expressed as

- The acceleration increasing (AI) phase:

$$v(t) = R_{\mathrm{v}}(t - 0, v_0, a_0, u_{\mathrm{j_{acc}}}) \quad \text{for } 0 \leq t < t_1$$

- The acceleration constant (AC) phase:

$$v(t) = R_{\mathrm{v}}(t - t_1, v_1, d_{\mathrm{a}} \cdot a_{\max}, 0) \quad \text{for } t_1 \leq t < t_2,$$

- And the acceleration decreasing (AD) phase:

$$v(t) = R_{\mathrm{v}}(t - t_2, v_2, a_2, u_{\mathrm{j_{dec}}}) \quad \text{for } t_2 \leq t < t_3$$

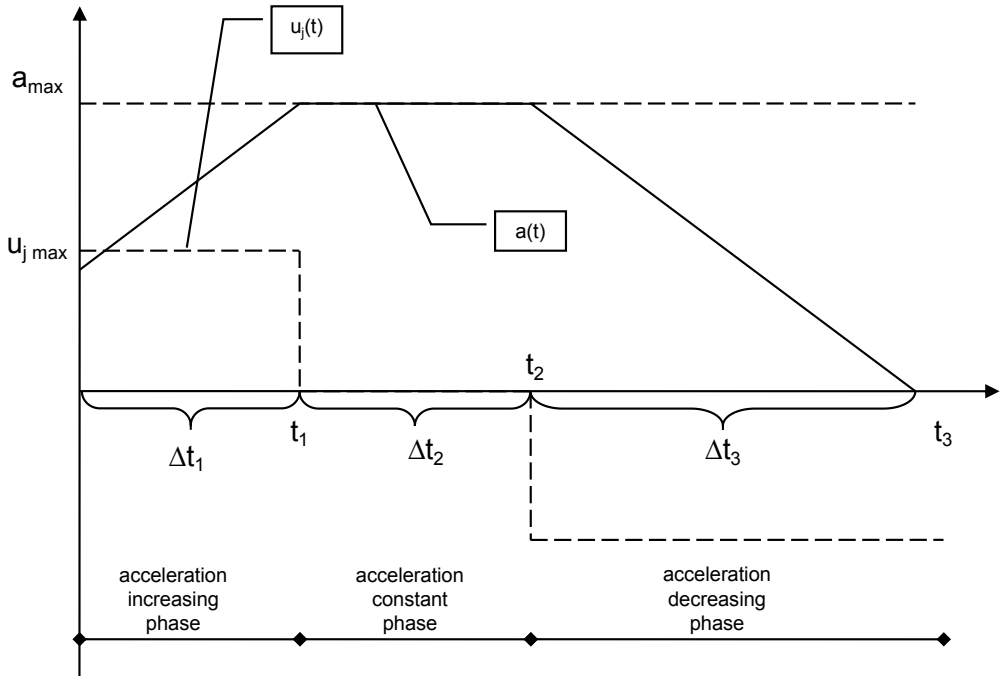


Figure 4.12: Trapezoidal acceleration profile
A trapezoidal acceleration profile, the corresponding jerk is discontinuous but bounded.

where

$$d_{\mathrm{a}} = \mathrm{sign}(v_{\mathrm{ref}} - v_0 - \frac{a_0 \left|a_0\right|}{2u_{\mathrm{j}_{\max}}})$$

is the acceleration cruising direction.

In our implementation, the value of d_{a} can be either $-1, 0, 1$. This helps to resolve some numerical issues introduced by the floating number. When $d_{\mathrm{a}} = 0$, the trapezoidal profile contains only the first phase. The detail can be found in Algorithm 9. Since u_{j} can only be $-u_{\mathrm{j}_{\max}}$, $u_{\mathrm{j}_{\max}}$ or 0, its value is determined if its sign can be solved for each phase. The desired acceleration during AC phase can only be $a_{\max}$, 0 or $-a_{\max}$. Then the method in Algorithm 7 can be used to determine its value as $d_{\mathrm{a}} \cdot a_{\max}$. Correspondingly, the value of u_{j} during the AI and AD phases can be determined as

$$u_{\mathrm{j}_{\mathrm{acc}}} = u_{\mathrm{j}_{\max}} \cdot \mathrm{sign}(d_{\mathrm{a}} \cdot a_{\max} - a_0)$$
$$u_{\mathrm{j}_{\mathrm{dec}}} = u_{\mathrm{j}_{\max}} \cdot \mathrm{sign}(0 - d_{\mathrm{a}} \cdot a_{\max})$$

With all the value of u_{j} properly set. The task is then to find the duration of each phase. Assume the time spent in the three phases are Δt_1, Δt_2 and Δt_3 correspondingly (see

Figure 4.12), it is now possible to express the velocity change of each phases as

$$\begin{aligned}
\Delta v_{\mathrm{acc}} &= a_0 \Delta t_1 + \frac{1}{2} u_{\mathrm{j}_{\mathrm{acc}}} \Delta t_1^2 \\
\Delta v_{\mathrm{dec}} &= d_{\mathrm{a}} \cdot a_{\max} \Delta t_3 + \frac{1}{2} u_{\mathrm{j}_{\mathrm{dec}}} \Delta t_3^2 \\
\Delta v_{\mathrm{constant}} &= v_{\mathrm{ref}} - v_0 - \Delta v_{\mathrm{acc}} - \Delta v_{\mathrm{dec}}
\end{aligned} \tag{4.5}$$

Then, the value of Δv_{acc} and Δv_{dec} is first computed under the assumption that the system reaches the desired $d \cdot a_{\max}$ and immediately steered to zero acceleration. This process gives

$$\begin{aligned}
\Delta t_1 &= \frac{|d_{\mathrm{a}} \cdot a_{\max} - a_0|}{u_{\mathrm{j}_{\max}}} \\
\Delta t_3 &= \frac{|-d_{\mathrm{a}} \cdot a_{\max}|}{u_{\mathrm{j}_{\max}}}
\end{aligned} \tag{4.6}$$

Substitute Equation 4.6 into 4.5, it is then possible to calculate the value of Δv_{acc}, Δv_{dec} and $\Delta v_{\mathrm{constant}}$. If $\Delta v_{\mathrm{constant}} > 0$ and $d_{\mathrm{a}} \neq 0$, the value of Δt_2 is simply

$$\Delta t_2 = \frac{\Delta v_{\mathrm{constant}}}{d_{\mathrm{a}} \cdot a_{\max}}$$

When $d_{\mathrm{a}} = 0$, there is $\Delta t_2 = 0$. And the time durations of all phases are found. However, if the solved $\Delta t_2 < 0$, it implies there will be no AC phase and the desired acceleration $d_{\mathrm{a}} \cdot a_{\max}$ cannot be achieved. For this case, the acceleration profile in Figure 4.12 will become a wedge shape (see Figure 4.13) with $\Delta t_2 = 0$. The task now becomes to solve Δt_1, Δt_3 together with a new variable a_{reach} which represents the reachable acceleration after AI phase. The solution to the following equations

$$\begin{aligned}
\Delta t_1 &= \frac{|a_{\mathrm{reach}} - a_0|}{u_{\mathrm{j}_{\max}}} \\
\Delta t_3 &= \frac{|a_{\mathrm{reach}}|}{u_{\mathrm{j}_{\max}}} \\
\Delta v_{\mathrm{acc}} &= a_0 \Delta t_1 + \frac{1}{2} u_{\mathrm{j}_{\mathrm{acc}}} \Delta t_1^2 \\
\Delta v_{\mathrm{dec}} &= a_{\mathrm{reach}} \Delta t_3 + \frac{1}{2} u_{\mathrm{j}_{\mathrm{dec}}} \Delta t_3^2 \\
\Delta v_{\mathrm{acc}} &+ \Delta v_{\mathrm{dec}} + v_0 = v_{\mathrm{ref}}
\end{aligned} \tag{4.7}$$

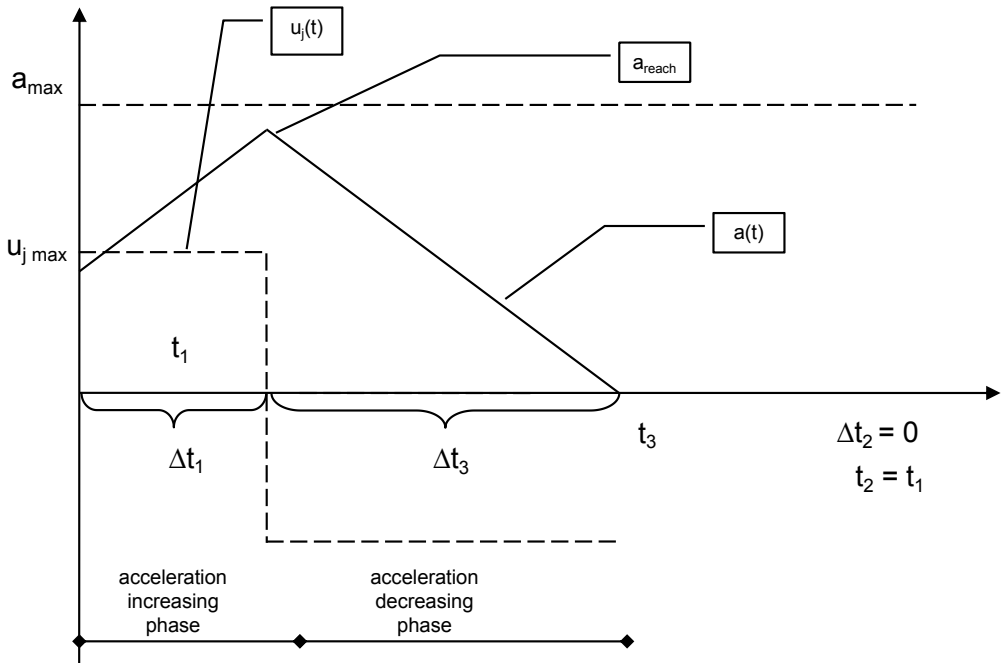


Figure 4.13: Wedge acceleration profile
A wedge acceleration profile, the corresponding jerk is discontinuous but bounded.

gives

$$a_{\mathrm{reach}} = d_{\mathrm{a}} \cdot \sqrt{d_{\mathrm{a}} \cdot u_{\mathrm{j}_{\max}} \cdot (v_{\mathrm{ref}} - v_0) + \frac{1}{2} a_0^2}$$

and hence the Δt_1 and Δt_3. The pseudo code for solving double integrator TPBVP is given in Algorithm 9. With the solved $u_{\mathrm{j}}(t)$, Δt_1, Δt_2, Δt_3 and all the initial conditions, the whole trajectory can be reconstructed using piecewise polynomials.

To demonstrate the method's capability, a comparison between the state trajectory obtained through solving the TPBVP and forward simulation is shown in Figure 4.14 and 4.15. In Figure 4.14, the trajectories start from $v_0 = 1,\ a_0 = 1$ with $a_{\max} = 2$ and $u_{\mathrm{j}_{\max}} = 2$. In Figure 4.15, the trajectories start from $v_0 = -1,\ a_0 = 4$ with the same state and input constraints which has been violated by the initial condition. In both figures, the state trajectories from the forward simulation and the closed-form solution resemble each other (provide the frequency of the forward simulation is high enough).

The time consumption of acquiring a future state through TPBVP solving is always constant since the TPBVP gives the entire state transfer policy. On the other hand, the forward simulation takes more time when the predicted state is further into the future. For applications that require the prediction of states in the far future, like safety zone flight and obstacle avoidance, the realtime performance would be difficult to achieve

Algorithm 9 Velocity target TPBVP solver

Input: v_0, a_0 ,$a_{\max}$,$u_{\mathrm{j}_{\max}}$, v_{ref}

Output: t_1,t_2,t_3,j_1,j_3

$v_{\mathrm{end}} = v_0 + \dfrac{a_0 \left|a_0\right|}{2u_{\mathrm{j}_{\max}}}$

$d_{\mathrm{a}} = \mathrm{sign}(v_{\mathrm{ref}} - v_{\mathrm{end}})$

$a_{\mathrm{cruise}} = d_{\mathrm{a}} \cdot a_{\max}$

$\Delta t_1 \leftarrow abs(a_{\mathrm{cruise}} - a_0)/u_{\mathrm{j}_{\max}}$

$u_{\mathrm{j}_{\mathrm{acc}}} \leftarrow u_{\mathrm{j}_{\max}} \cdot \mathrm{sign}(a_{\mathrm{cruise}} - a_0)$

$v_1 \leftarrow v_0 + a_0 \cdot \Delta t_1 + 0.5 \cdot u_{\mathrm{j}_{\mathrm{acc}}} \cdot \Delta t_1^2$

$\Delta t_3 \leftarrow abs(-a_{\mathrm{cruise}})/u_{\mathrm{j}_{\max}}$

$u_{\mathrm{j}_{\mathrm{dec}}} \leftarrow u_{\mathrm{j}_{\max}} \cdot \mathrm{sign}(-a_{\mathrm{cruise}})$

$\bar{v_3} \leftarrow a_{\mathrm{cruise}} \cdot \Delta t_3 + 0.5 \cdot u_{\mathrm{j}_{\mathrm{dec}}} \cdot \Delta t_3^2$

$\bar{v_2} \leftarrow v_{\mathrm{ref}} - v_1 - \bar{v_3}$

if $d_{\mathrm{a}} = 0$ **then**

    $\Delta t_2 \leftarrow 0$

else

    $\Delta t_2 \leftarrow \bar{v_2}/a_{\mathrm{cruise}}$

if $\Delta t_2 < 0$ **then**

    $a_{\mathrm{cruise}} \leftarrow d_{\mathrm{a}} \cdot \sqrt{d_{\mathrm{a}} \cdot u_{\mathrm{j}_{\max}} \cdot (v_{\mathrm{ref}} - v_0) + 0.5 \cdot a_0^2}$

    $\Delta t_1 \leftarrow abs(a_{\mathrm{cruise}} - v_0)/u_{\mathrm{j}_{\max}}$

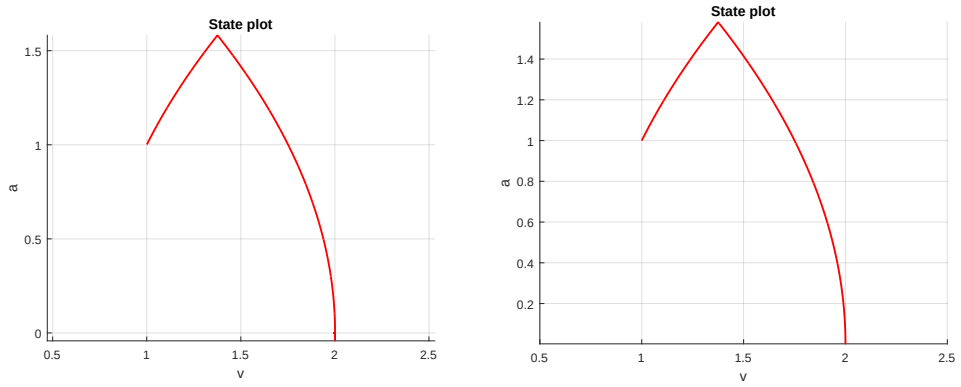
    $\Delta t_2 \leftarrow 0$

    $\Delta t_3 \leftarrow abs(-a_{\mathrm{cruise}})/u_{\mathrm{j}_{\max}}$

$t_1 \leftarrow \Delta t_1$

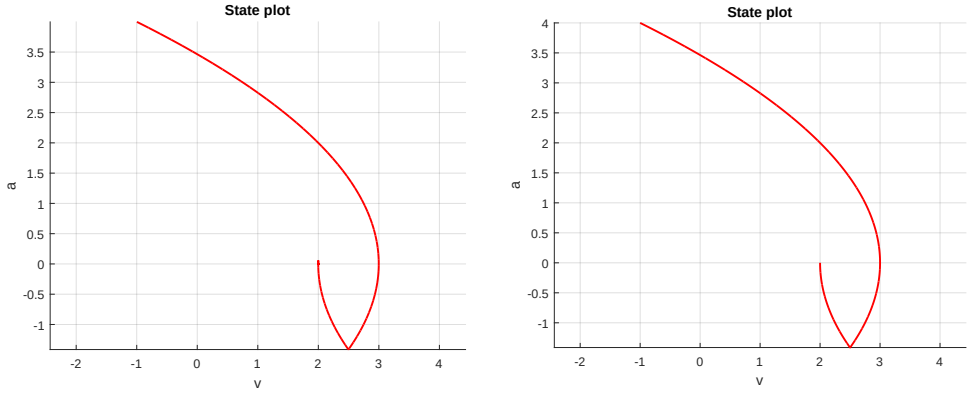
$t_2 \leftarrow \Delta t_1 + \Delta t_2$

$t_3 \leftarrow \Delta t_1 + \Delta t_2 + \Delta t_3$



(a) Forward simulation (b) Closed form solution

Figure 4.14: Comparison between forward simulation and TPBVP: no constraints violation



(a) Forward simulation (b) Closed form solution

Figure 4.15: Comparison between forward simulation and TPBVP: initial state violate constraints

if the forward simulation approach is adopted. A comparison between solving TPBVP and forward simulation is given in Table 4.1. Both the velocity and position trajectory are compared. It shows clearly the forward simulation takes more time as the predicted state grows further into the future.

4.1.4 Time Synchronization

For multi-dimensional cases, the simplest way is to calculate the trajectories using Algorithm 9 for each dimension separately. However, it is sometimes desired for the velocity of each dimension to reach the target value at the same time. This is referred as the time synchronization problem. As in [59], the idea is to prolong each dimension's reaching time to a common value. According to [66], the reaching time for a dimension

Table 4.1: Time consumption of predicting a future state

	TPBVP (Velocity)	FS (Velocity)	TPBVP (Position)	FS (Position)
State after 5 s	<1 µs	120 µs	<3 µs	400 µs
State after 15 s	<1 µs	350 µs	<3 µs	1200 µs
State after 30 s	<1 µs	690 µs	<3 µs	2430 µs

FS: Forward Simulation

with zero final acceleration can be extended to any value as long as it is larger than the minimum reaching time. Therefore, the steps of synchronizing all dimensions can be written as

1. Calculate the trajectory using Algorithm 9 for each dimension separately.

2. Find the dimension with longest reaching time t_{fl}.

3. Extend other dimension's reaching time to t_{fl}.

Now, an algorithm is needed to extend a given trajectory's reaching time. When the reaching time is prolonged, all acceleration profile will become trapezoidal shaped and the acceleration at its AC phase a_{co} needs to be found. By solving the following equations using results in [67]

$$
\begin{aligned}
\Delta t_1 &= \frac{|a_{\mathrm{co}} - a_0|}{u_{\mathrm{j}_{\max}}} \\
\Delta t_3 &= \frac{|a_{\mathrm{co}}|}{u_{\mathrm{j}_{\max}}} \\
\Delta v_{\mathrm{acc}} &= a_0 \Delta t_1 + \frac{1}{2} u_{\mathrm{j}_{\mathrm{acc}}} \Delta t_1^2 \\
\Delta v_{\mathrm{dec}} &= a_{\mathrm{co}} \Delta t_3 + \frac{1}{2} u_{\mathrm{j}_{\mathrm{dec}}} \Delta t_3^2 \\
\Delta v_{\mathrm{constant}} &= v_{\mathrm{ref}} - v_0 - \Delta v_{\mathrm{acc}} - \Delta v_{\mathrm{dec}} \\
\Delta t_2 &= \frac{\Delta v_{\mathrm{constant}}}{d_{\mathrm{a}}\ a_{\mathrm{co}}} \\
\Delta t_1 + \Delta t_2 + \Delta t_3 &= t_{\mathrm{fl}}
\end{aligned}
\tag{4.8}
$$

the value of a_{co} is found to be

$$a_{\mathrm{co}} = \begin{cases} \frac{1}{2} d_{\mathrm{a}} (b - \sqrt{b^2 - 4 d_{\mathrm{a}} \cdot u_{\mathrm{j}_{\max}} \Delta v - 2 a_0^2}) & \text{if } t_{\mathrm{fl}} < t_{\mathrm{lim}} \\ \frac{d_{\mathrm{a}}^3 \Delta v - d_{\mathrm{a}}^2 a_0^2 / 2 u_{\mathrm{j}_{\max}}}{d_{\mathrm{a}} \cdot t_{\mathrm{fl}} - a_0 / u_{\mathrm{j}_{\max}}} & \text{if } t_{\mathrm{fl}} \geq t_{\mathrm{lim}} \end{cases} \tag{4.9}$$

where

$$\Delta v = v_{\mathrm{ref}} - v_0$$

$$b = a_{\max} \cdot t_{\mathrm{fl}} + d_{\mathrm{a}} \cdot v_0$$

$$t_{\mathrm{limP}} = \frac{\Delta v}{a_0} + \frac{d_{\mathrm{a}} \cdot a_0}{2 u_{\mathrm{j}_{\max}}}$$

$$t_{\mathrm{lim}} = \begin{cases} t_{\mathrm{limP}} & \text{if } t_{\mathrm{limP}} \geq 0 \\ \infty & \text{if } t_{\mathrm{limP}} < 0 \end{cases}$$

with the value of a_{co} it is now possible to find the values of Δt_1, Δt_2 and Δt_3 through Equation 4.8. As for j_1 and j_3, they can be written as

$$j_1 = u_{\mathrm{j}_{\max}} \cdot \mathrm{sign}(a_{\mathrm{co}} - a_0)$$
$$j_3 = u_{\mathrm{j}_{\max}} \cdot \mathrm{sign}(-a_{\mathrm{co}})$$

And a trajectory can be easily reconstructed through the piecewise polynomials with the value of Δt_1, Δt_2, Δt_3, j_1, j_3 and all the initial conditions. An example of a system with three dimensions reaching their target velocity at the same time is shown in Figure 4.16. In this figure, each dimension starts with different initial conditions, has different constraints and their target velocities are $10\,\mathrm{m/s}$, $20\,\mathrm{m/s}$ and $30\,\mathrm{m/s}$ respectively.

4.1.5 Safe Fly Zone

Consider a mission which should be performed in a safety zone, where the zone is surrounded by tall buildings or trees. A fail-safe mechanism is required to prevent the user issuing commands which may potentially take the vehicle outside the zone and crash into the buildings. To achieve this, a simple MPC structure is adopted. The idea can be described as

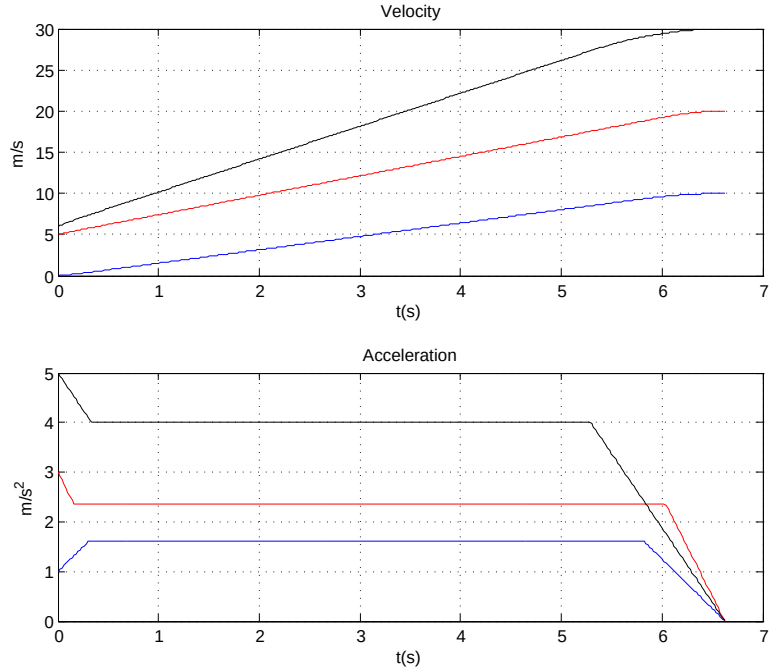


Figure 4.16: Time synchronized velocity trajectory

1. Predict the state reached by following the operator's velocity command for a short T_{h} seconds.

2. From the state in Step 1, immediately drop the velocity to zero and calculate the stop position.

3. Extract the trace of the vehicle through Step 1 and 2.

4. If the trace is outside the safety zone, the operator's command is nullified and replaced with a zero velocity command.

Here, the T_{h} is a tuning variable that helps to control the cautiousness of the vehicle. The trajectory of the vehicle during Step 1 and 2 can be obtained by Algorithm 9 whereas the trace is acquired through sampling the position trajectory into a series of line segments. To check whether the trace is outside the safety zone, each line segment is searched for intersection with the safety zone polygon. Here, it is not necessary to find the exact location of the intersection, only its existence matters. Since the polygon is also constructed with a series of line segments, the task is now to check the existence of intersection between two line segments. Efficient method in Algorithm 10 is adopted.

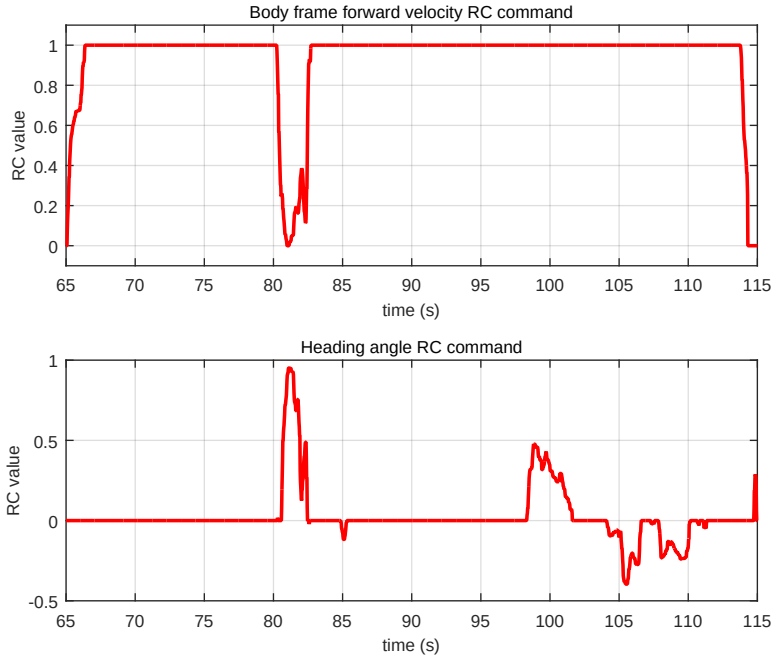
Algorithm 10 Intersection test for two line segments

1: Input: line segment (A, B) and (C, D) where A, B, C, D are 2D points
2: Output: isIntersect
3: isIntersect = $\text{ccw}(A, C, D)! = \text{ccw}(B, C, D)$ *and* $\text{ccw}(A, B, C)! = \text{ccw}(A, B, D)$
4: ---
5: **Function** isCCW = $\text{ccw}(P_1, P_2, P_3)$
6: isCCW = $(P_3.y - P_1.y)(P_2.x - P_1.x) > (P_2.y - P_1.y)(P_3.x - P_1.x)$

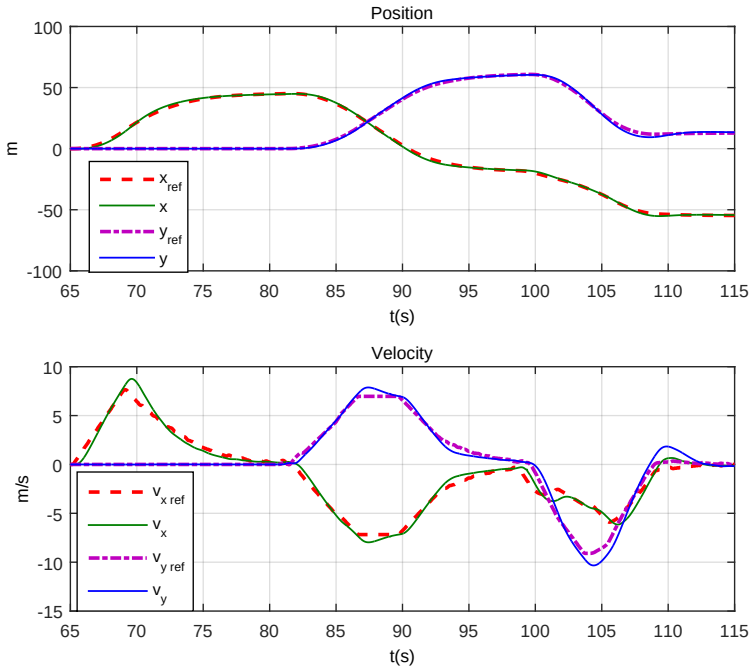
Further, the algorithm is also used to help determine whether the end point is inside the safety zone through the ray casting method. Finally, divide and conquer methods can be used to reduce the time complexity of searching if the safety zone polygon is complex.

Simulation experiment of the vehicle stopped by the safe zone boundary is given. In Figure 4.17, the deceleration trace of the vehicle is shown. And in Figure 4.18, the RC controller's command is compared with the generated reference and vehicle's response. From 105 s, the vehicle starts to decelerate to avoid crossing the safe fly zone boundary while the user still issues full speed forward velocity command. The issued command is overwritten by the guidance system and steers the vehicle to stop in front of the boundary.

Figure 4.17: Safe fly zone: vehicle trace



(a) RC inputs



(b) Vehicle response

Figure 4.18: Safe fly zone: commands and response

4.2 Online Position Trajectory

4.2.1 Position Command Filter

Though the semi-auto flight mode is popular among RC community, many other applications require fully autonomous flight based on pre-planned missions. As discussed in Chapter 2, the mission is usually specified by combinations of waypoints which consists of a position in the global coordinate and the corresponding tasks. To execute a given mission, the basic element is to guide the vehicle to a specific point of interest. Unlike the semi-auto mode, velocity command is no longer provided, and the algorithm shall generate all acceleration, velocity and position reference. One dimensional case is first illustrated where the optimization target and constraints are given as

$$
\begin{aligned}
&\min_{u_{\mathrm{j}}(t), t\in[0,T]} \left\{ J = \textstyle\int_{t=0}^{T} dt \right\} \ s.t. \\
&p(0) = p_0, \quad p(T) = p_{\mathrm{ref}} \\
&v(0) = v_0, \quad v(T) = 0 \\
&a(0) = a_0, \quad a(T) = 0 \\
&\dot{p}(t) = v(t) \\
&\dot{v}(t) = a(t) \\
&\dot{a}(t) = u_j(t) \\
&-v_{\max} \leqslant v(t) \leqslant v_{\max}, \qquad \forall t \in [0, T] \\
&-a_{\max} \leqslant a(t) \leqslant a_{\max}, \qquad \forall t \in [0, T] \\
&-u_{\mathrm{j}_{\max}} \leqslant u_{\mathrm{j}}(t) \leqslant u_{\mathrm{j}_{\max}}, \quad \forall t \in [0, T]
\end{aligned}
\tag{4.10}
$$

The problem is similar to the one in Equation 4.4 except that the dynamic constraints now describe a triple instead of a double integrator with extra limitation on its velocity. Though the TPBVP solver (see Section 4.2.5) can be used to solve the problem, a flight mission might encounter constantly changing target, such as the following of a moving ground vehicle. A position command filter which only generates the next cycle reference is proposed to save computational power. Let Algorithm 9 be denoted as a function

$$
(t_1, t_2, t_3, j_1, j_3) = \mathrm{VelParamGen}(v_0, a_0, a_{\max}, u_{\mathrm{j}_{\max}}, v_{\mathrm{ref}}) \tag{4.11}
$$

And express the piecewise polynomial describing the trajectory returned by Algorithm 9 as a function

$$(p_{\mathrm{sp}}, v_{\mathrm{sp}}, a_{\mathrm{sp}}) = \mathrm{StateExam}(v_0, a_0, p_0, t_1, t_2, t_3, j_1, j_3, t_{\mathrm{sp}}) \tag{4.12}$$

whereas its inputs are the initial states, parameters to restore the control value $u_{\mathrm{j}}(t)$ and a specific time t_{sp}; outputs are the position, velocity and acceleration at the given time t_{sp}. Here, $v_0,\ a_0,\ p_0$ are the initial velocity, acceleration and position; $t_1,\ t_2,\ t_3$ are the time of AI, AC and AD phases; j_1, j_3 are the jerk input during AI and AD phases. Then, the pseudo code for the state constrained position command filter can be illustrated by Algorithm 11.

Algorithm 11 State constrained position command filter

Input: $p_0, v_0, a_0, v_{\max}, a_{\max}, u_{\mathrm{j}_{\max}}, t_{\mathrm{samp}}, p_{\mathrm{ref}}$
Output: $u_{\mathrm{j}}^*, a_{\mathrm{rft}}, v_{\mathrm{rft}}, p_{\mathrm{rft}}$
if !Initialized **then**
    $p_{\mathrm{I}} = p_0$
    $v_{\mathrm{I}} = v_0$
    $a_{\mathrm{I}} = a_0$
    Initialized = TRUE
$(t_1, t_2, t_3, j_1, j_3) = \mathrm{VelParamGen}(v_{\mathrm{I}}, a_{\mathrm{I}}, a_{\max}, u_{\mathrm{j}_{\max}}, 0)$
$(p_{\mathrm{sp}}, v_{\mathrm{sp}}, a_{\mathrm{sp}}) = \mathrm{StateExam}(v_{\mathrm{I}}, a_{\mathrm{I}}, p_{\mathrm{I}}, t_1, t_2, t_3, j_1, j_3, t_3)$
if $p_{\mathrm{sp}} > p_{\mathrm{ref}} + p_\delta$ **then**
    $v_{\mathrm{cruise}} = -v_{\max}$
else if $p_{\mathrm{sp}} < p_{\mathrm{ref}} - p_\delta$ **then**
    $v_{\mathrm{cruise}} = v_{\max}$
else
    $v_{\mathrm{cruise}} = 0$
$(t_1, t_2, t_3, j_1, j_3) = \mathrm{VelParamGen}(v_{\mathrm{I}}, a_{\mathrm{I}}, a_{\max}, u_{\mathrm{j}_{\max}}, v_{\mathrm{cruise}})$
$(p_{\mathrm{I}}, v_{\mathrm{I}}, a_{\mathrm{I}}) = \mathrm{StateExam}(v_{\mathrm{I}}, a_{\mathrm{I}}, p_{\mathrm{I}}, t_1, t_2, t_3, j_1, j_3, t_{\mathrm{samp}})$
$u_{\mathrm{j}}* = j_1$
$a_{\mathrm{rft}} = a_{\mathrm{I}}$
$v_{\mathrm{rft}} = v_{\mathrm{I}}$
$p_{\mathrm{rft}} = p_{\mathrm{I}}$

The idea behind Algorithm 11 can be described as: try slowing down to full stop from the current state immediately, if it stops before the target $p_{\mathrm{sp}} < p_{\mathrm{ref}} - p_\delta$, then the desired velocity v_{cruise} shall be positive; if it overshoot the target $p_{\mathrm{sp}} > p_{\mathrm{ref}} + p_\delta$, v_{cruise} shall be negative; otherwise, v_{cruise} is zero. Here the p_δ is a small positive value added to prevent chattering caused by discretization. Its value is much smaller than the practical accuracy of the vehicle. With the desired velocity, Algorithm 9 is utilized to generate

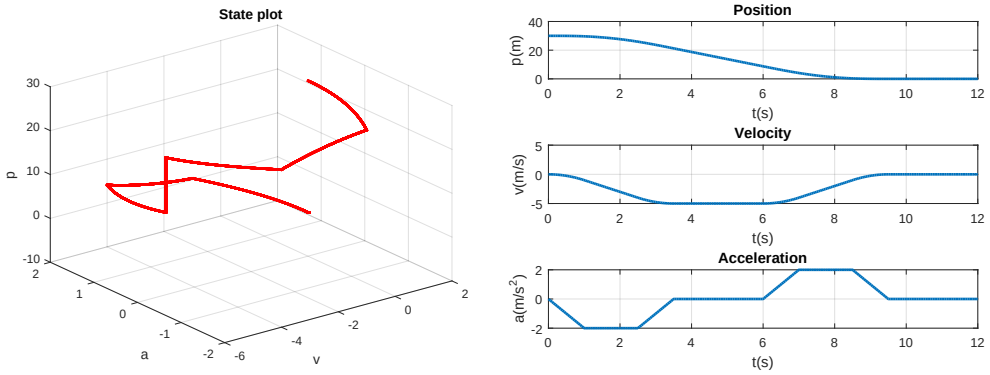


Figure 4.19: Position trajectory: all initial states within constraints

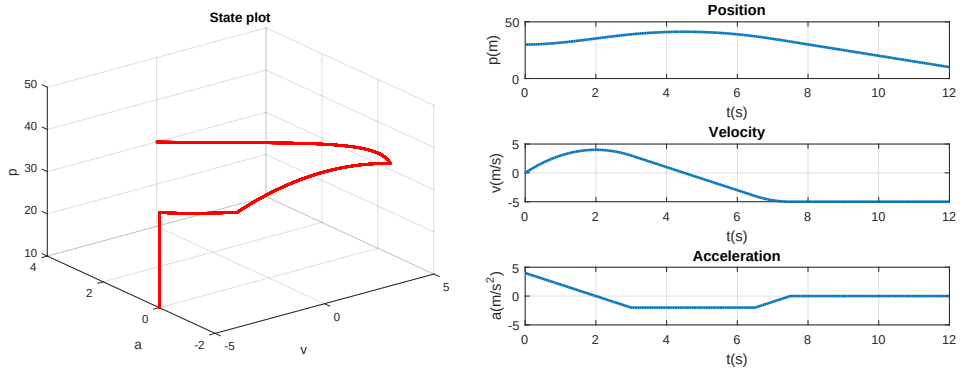


Figure 4.20: Position trajectory: initial acceleration violates constraints

the references $a_{\mathrm{rft}}, v_{\mathrm{rft}}$ and p_{rft} after a short sampling period t_{samp}. Moreover, due to the constraints on velocity and the minimum time optimization target, the magnitude of v_{cruise} is always chosen to be $v_{\max}$ when it is not zero. Some examples of executing the proposed command generator algorithm are given in Figure 4.19, 4.20 and 4.21. For all cases, the state constraints and the target position are the same as $u_{\mathrm{j}\max} = 2, a_{\max} = 2, v_{\max} = 5, p_{\mathrm{ref}} = 0$. In Figure 4.19, the initial condition is $a_0 = 0, v_0 = 0, p_0 = 30$, the resulting trajectories successfully steer the system to its target without violating any of the constraints. In Figure 4.20, the initial condition is $a_0 = 4, v_0 = 0, p_0 = 30$ which violates the acceleration constraint. Nevertheless, the resulting trajectory still manages to fulfill the task. Further, the acceleration has been brought down from its initial value into the region of constraint and no longer leaves it anymore. In Figure 4.21, the initial condition is $a_0 = 4, v_0 = -6, p_0 = 30$ which violates both velocity and acceleration constraints. Like the case in Figure 4.20, both velocity and acceleration are lead into the constraint region while the state trajectory marching towards its target.

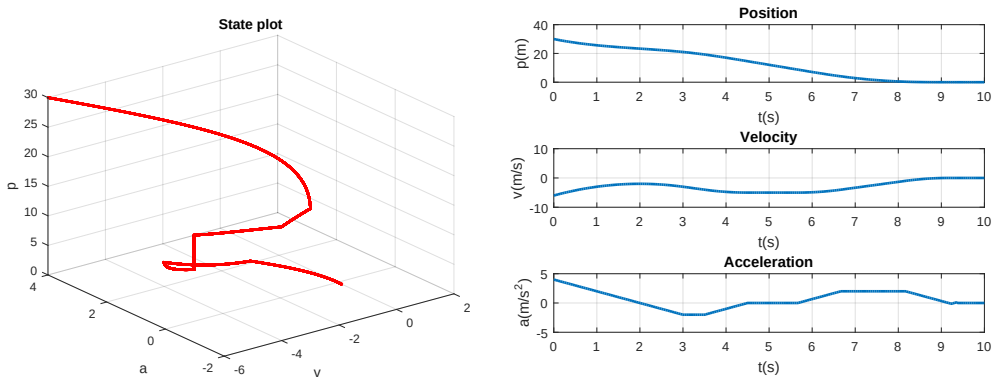


Figure 4.21: Position trajectory: initial acceleration and velocity both violate constraints

Finally, an example is given to illustrate the usage of Algorithm 11 as command filter for multiple positional inputs (see Figure 4.22). No matter how far away the target position is, the trajectory always steers the system towards its goal while fulfilling the state and input constraints. Algorithm 11 might cause small overshoot at the end of the trajectory, but the overshoot is very small given the typical RUAV's state limits. With $u_{\mathrm{j}_{\max}} = 5, a_{\max} = 5, v_{\max} = 10$, the proposed algorithm is estimated to use around $3.84\,\mu\mathrm{s}$[1] for each cycle on the flight controller with the $168\,\mathrm{MHz}$ Cortex M4F CPU. Though a rough estimation, considering the translational controller usually works at 50–$200\,\mathrm{Hz}$, the computational burden added to the system by the position command filter is minor.

4.2.2 Geometric Path Following by Coordinate Projection

Many tasks require the following of 2D geometric paths such as environment reconstruction, aero filming, and industrial inspection. During the process, the altitude of the vehicle is allowed to be changed independently regarding its horizontal position. In this subsection, a coordinate projection method is proposed for this purpose.

Single Segment Path

Considering the case in Figure 4.23, a vehicle is to follow the line segment formed by two waypoints WP_i and WP_{i+1} with an arbitrary initial condition. A new coordinate

[1]The actual profiling is done on the laptop with an Intel I5-3340M CPU at 2.7GHz. The cycle time on the laptop is $0.12\,\mu\mathrm{s}$. Scaling the time according to CPU frequency (2.7GHz to 168MHz) with a possible overhead, the cycle time on the flight controller is roughly estimated as $3.84\,\mu\mathrm{s}$.

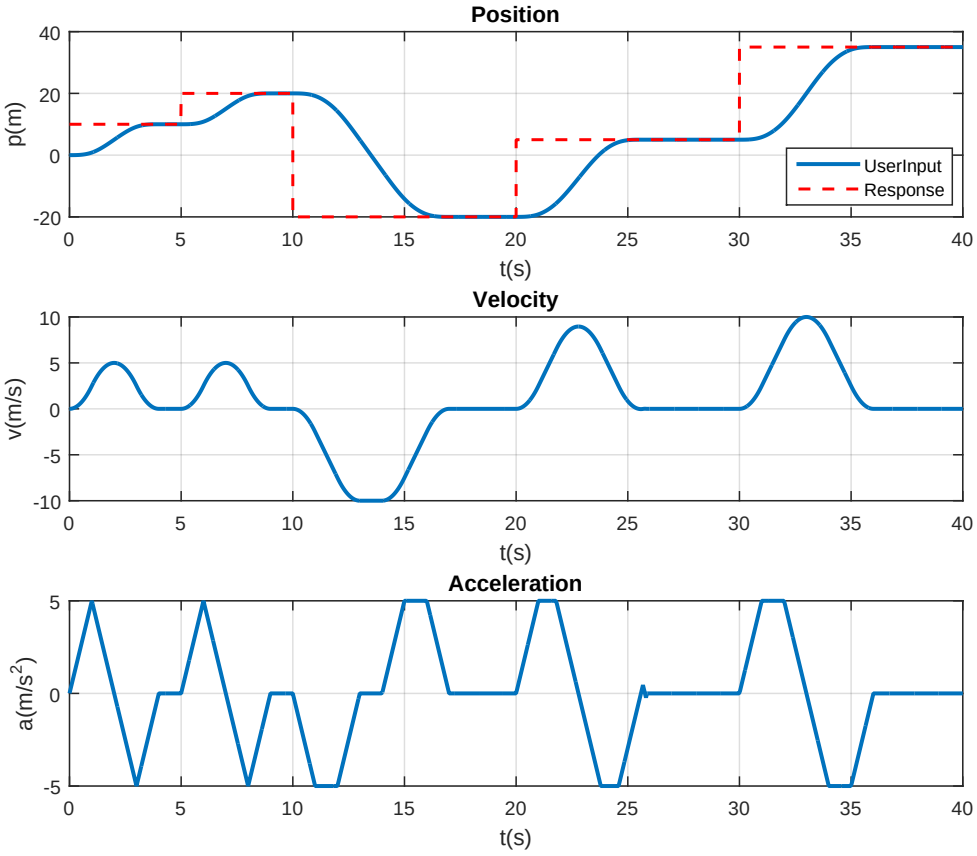


Figure 4.22: Filtered reference for position command

$\mathcal{W}$ is built with its origin at WP_i and its $x-$ axis pointing towards WP_{i+1}. With this coordinate, the path following problem can be solved by

1. Transfer all the vehicle's initial states into coordinate $\mathcal{W}$.
2. In the $x-$ axis, steer the vehicle to $x_{\mathrm{ref}} = \|\mathrm{WP}_i - \mathrm{WP}_{i+1}\|$.
3. In the $y-$ axis, steer the vehicle to $y_{\mathrm{ref}} = 0$.
4. For each axis, the position command filter or the position TPBVP solver (Section 4.2.5) can be executed individually.

The resulting simulation trajectory of the vehicle following a single line segment path using the position command filter is illustrated in Figure 4.24. As shown in the figure, the reference converges to the straight line path from its initial position and stops at waypoint 1. Further, if the vehicle is on a point to point mission and is to stop at each point, only the reference along the $x-$ axis is planned while the $y-$ axis reference is

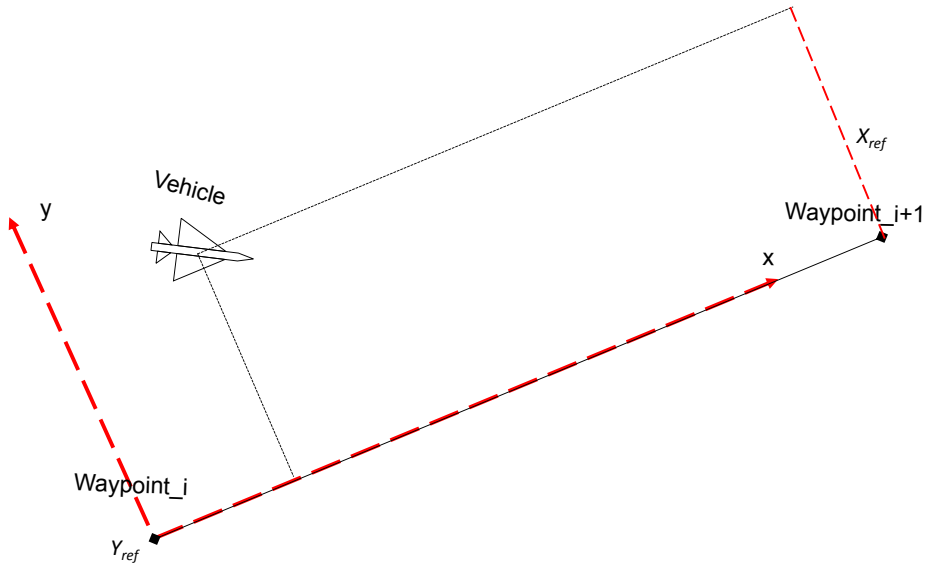


Figure 4.23: Coordinates during following 2D path

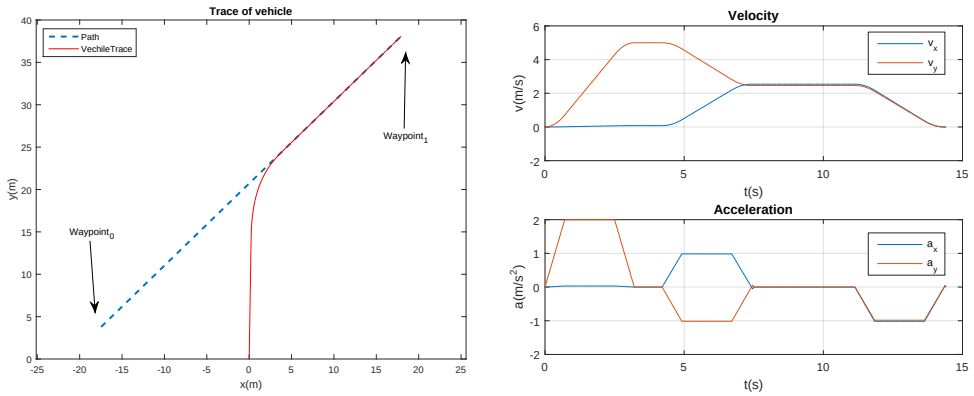


Figure 4.24: Outcome of line segment path following algorithm

always zero which saves computational power.

Multi Segment Path

On the other hand, the mission might consist of a series of connected line segments. If the above method is applied directly, the vehicle will enter hovering state at each waypoint. It can be overcome with switching to the next line segment before a full stop. The switching criteria can be defined by users, such as a reaching distance. Once the distance from the vehicle to a waypoint is smaller than the reaching distance, it automatically switches to the next line segment. Alternatively, if the switching criteria is not set, the following steps can be used to achieve a fast fly-by via each waypoint.

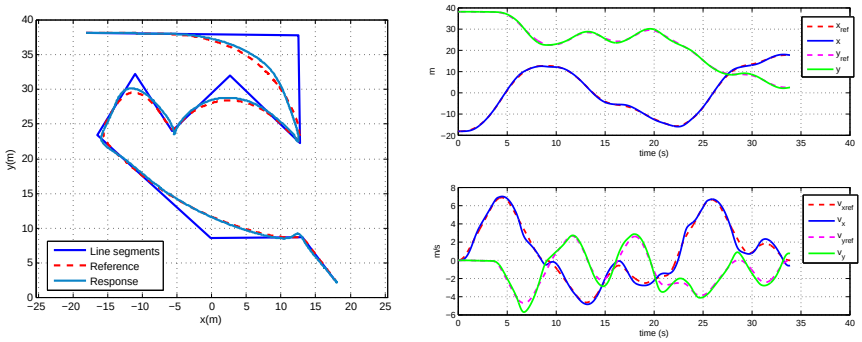


Figure 4.25: Following of complex path using the position command filter

1. Simulate an immediate braking (reduce speed to 0) along $x-$ axis. Denote the stop position as p_{stop_x}.

2. Check whether p_{stop_x} is very close to overshoot the target x_{ref}.

3. If it does, immediately switch to the next line segment.

The proposed strategy makes it more efficient to follow a path consists of many segments (see simulation results in Figure 4.25). The vehicle now can take the fly-by approach instead of hovering at each waypoint. The method can also be used to design curved path reference.

Circle and Spiral Path

By projecting the vehicle into a polar frame around a virtual center, a circle or spiral path can be easily generated. The limits on angular speed, angular acceleration and angular jerk shall be calculated so that they do not violate the vehicle's dynamic. Again, the position command filter or the position TPBVP solver can be adopted for the polar coordinate. Further, the coordinate projection idea can be combined with traditional path following methods and allow the vehicle to determine its heading angle automatically while accepting online specified speed command. This is helpful in tasks like the racing event filming.

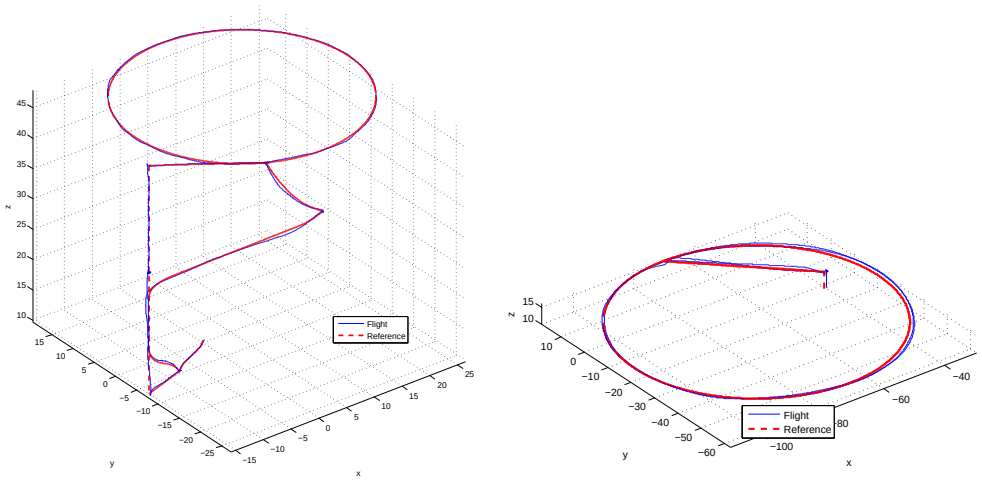
Reference Designing Toolbox

With the coordinate projection method as the base, and combining the B-spline trajectory generator, the velocity command filter, the position command filter, a jerk limited

TPBVP solver [68], the author implemented a reference designing toolbox for planning and visualizing trajectory. It allows the user to input a series of waypoints and adjusting the smoothness of the trajectory as well as the maximum flying velocity. It serves as a useful tool for developing and testing various guidance, control and localization algorithms as well as platforms.

Real Flight Experiment

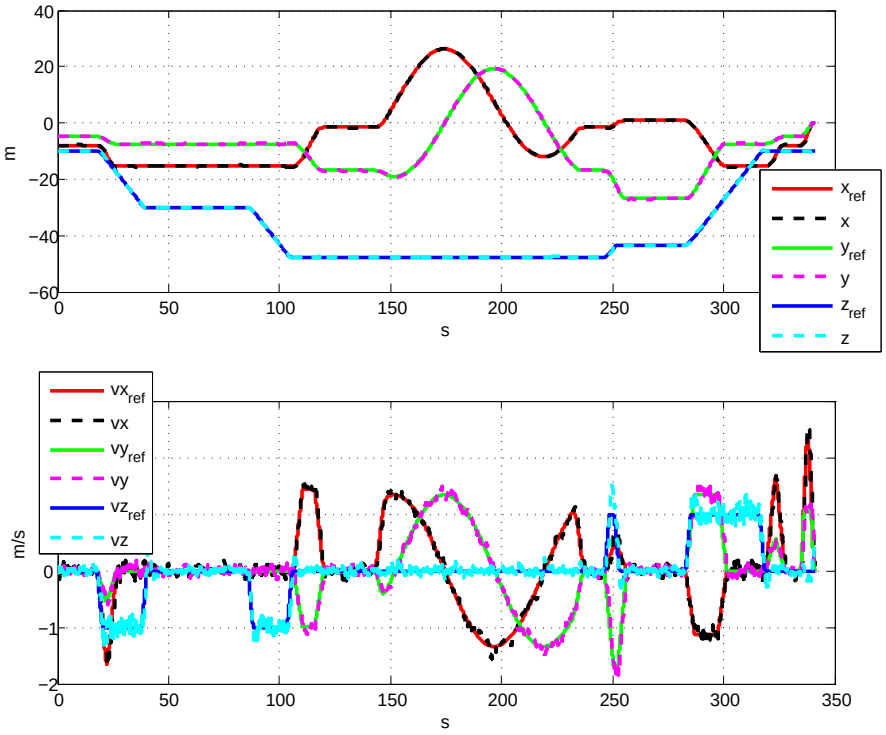
Here, four different real flight experiments are covered to demonstrate the tracking performance. Flight A and B show the vehicle's performance in combined missions. Flight C and D show the vehicle's performance during fast flight and among strong wind disturbances. The combined mission is shown in Figure 4.26. The tracking performance is shown in Figure 4.27 while the data is analyzed in Table 4.2. In flight experiment A, sub-meter path following performance is achieved with smaller top speed. While in flight experiment B, a small deviation from the circular path is maintained during high speed flight. Though the combined missions are generated before flight using the above reference designing toolbox, its execution could be interrupted and rearranged at any moment by online trajectory generation. The idea of realtime path modification is further extended to an obstacle avoidance system in Chapter 5.



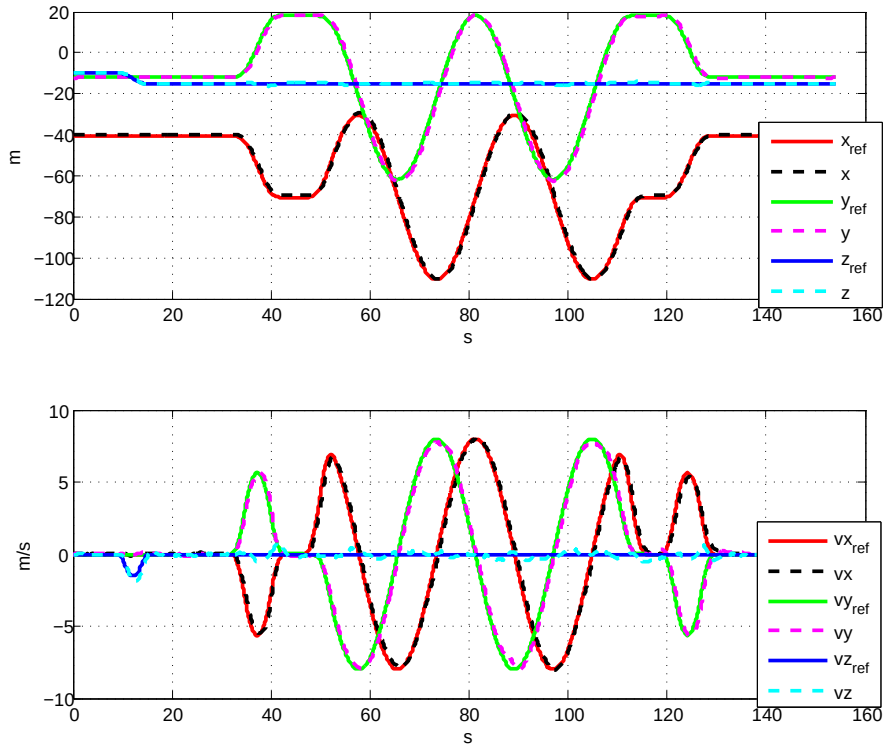
(a) Flight A (b) Flight B

Figure 4.26: Complex mission: vehicle trace

Fast flight and disturbance rejection experiments are shown in Figure 4.28. The experiment data is shown in Figure 4.29. The data analysis is shown in Table 4.3. In flight



(a) Flight A

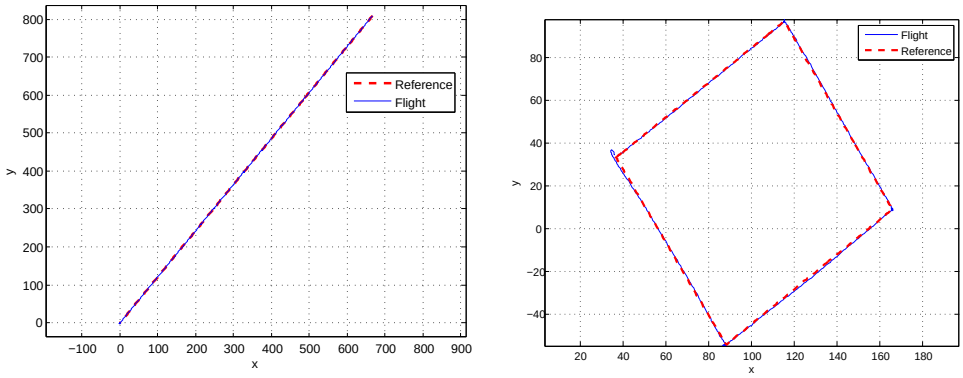


(b) Flight B

Figure 4.27: Complex mission: reference and response

Table 4.2: Experiment data of complex mission

	Flight A	Flight B
Wind speed	$\approx 2\,\mathrm{m/s}$	$\approx 4\,\mathrm{m/s}$
Top speed	$2.7\,\mathrm{m/s}$	$8\,\mathrm{m/s}$
Max of velocity error norm	$0.6\,\mathrm{m/s}$	$1.12\,\mathrm{m/s}$
Max of position error norm	$0.9\,\mathrm{m}$	$3.26\,\mathrm{m}$
Average of velocity error norm	$0.13\,\mathrm{m/s}$	$0.41\,\mathrm{m/s}$
Average of position error norm	$0.23\,\mathrm{m}$	$1.74\,\mathrm{m}$
Circle path max deviation (2D)	$0.4\,\mathrm{m}$	$1.22\,\mathrm{m}$
Circle path max deviation (3D)	$0.41\,\mathrm{m}$	$1.31\,\mathrm{m}$

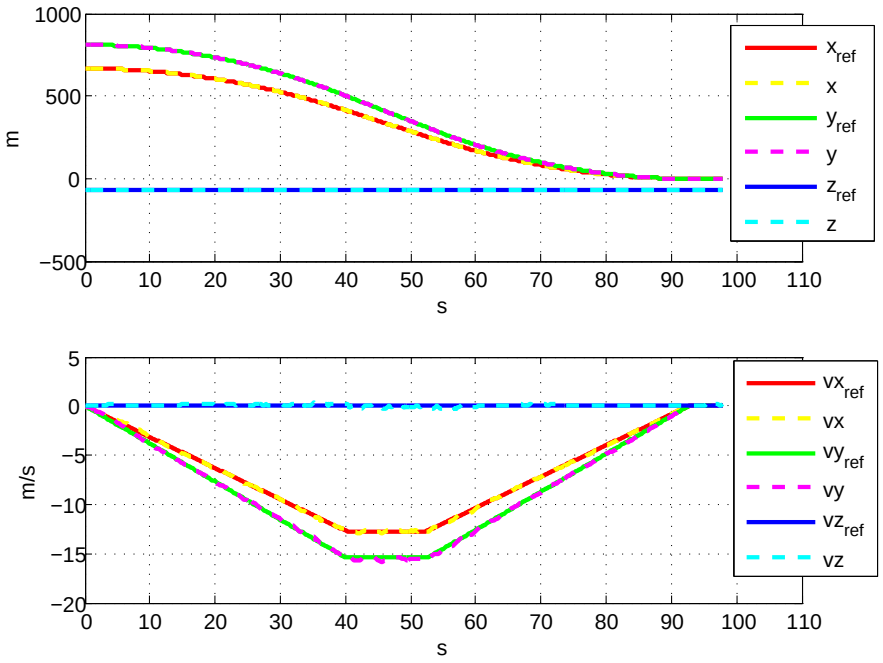


(a) Flight C (b) Flight D

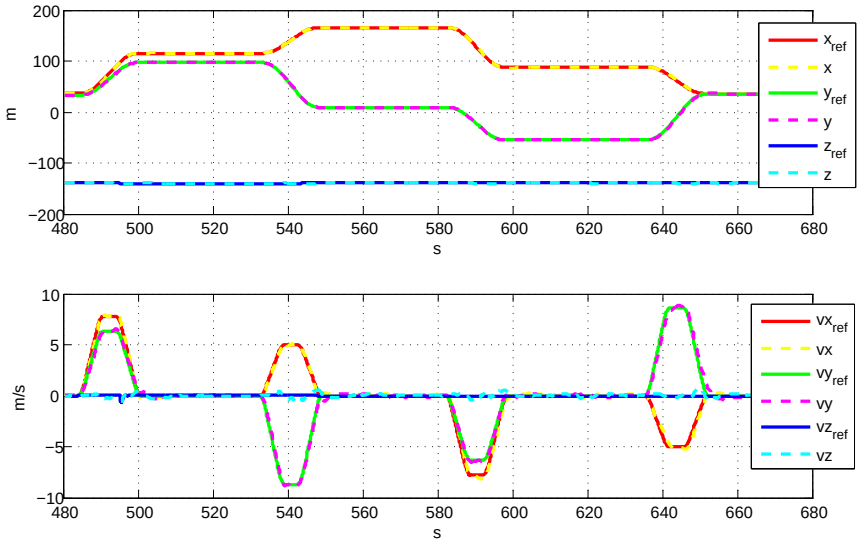
Figure 4.28: Fast flight and wind disturbance: vehicle trace

C, the vehicle traveled along a straight flight path with a maximum speed of $20\,\mathrm{m/s}$ and resulted in very small tracking and path deviation error. In flight D, the vehicle covered a square path with top speed of $10\,\mathrm{m/s}$ subject to very strong wind disturbances. Thanks to a well-designed control law and a dynamically feasible reference trajectory, performance is ensured even under the disturbance created by the wind and air drag during fast speed flight. In the experiments, better performance is achieved compared to several traditional techniques reported in [69]. As shown in the fast flight experiment, it effectively prevents the overshooting of path end through online generation of reference command. The reference is limited up to jerk which implies smoother flight under frequent maneuvering compared to acceleration limited method [14]. State transition from arbitrary initial state is also enabled instead of selecting trims and maneuvers from a library of precomputed motion primitives [70]. The cost is less aggressive maneuvers.

Predefined mission with online issued speed command is also possible. In Figure 4.30, the vehicle is made to follow horse track to film racing events. The flight path is



(a) Flight C



(b) Flight D

Figure 4.29: Fast flight and wind disturbance: reference and response

Table 4.3: Experiment data of fast flight with disturbance rejection

	Flight C	Flight D
Wind speed	$\approx 5\,\mathrm{m/s}$	$\approx 12\,\mathrm{m/s}$
Top speed	$20\,\mathrm{m/s}$	$10\,\mathrm{m/s}$
Max of velocity error norm	$0.55\,\mathrm{m/s}$	$1.37\,\mathrm{m/s}$
Max of position error norm	$1.42\,\mathrm{m}$	$2.84\,\mathrm{m}$
Average of velocity error norm	$0.17\,\mathrm{m/s}$	$0.28\,\mathrm{m/s}$
Average of position error norm	$0.59\,\mathrm{m}$	$0.67\,\mathrm{m}$
Straight path max deviation (2D)	$0.95\,\mathrm{m}$	$2.8\,\mathrm{m}$
Straight path max deviation (3D)	$1.0\,\mathrm{m}$	$2.81\,\mathrm{m}$

predefined by a series of waypoints, but the marching speed is controlled by the user to catch up with the horses. The maximum vehicle speed during the process reaches $24\,\mathrm{m/s}$ (80% of the theoretical max speed) with an average of position error norm around $2.97\,\mathrm{m}$. To my best knowledge, it is one of the fastest following-type mission performed by a quadrotor. The tracking performance is reasonably good considering the extreme flight speed and the frequent maneuvering for target following.

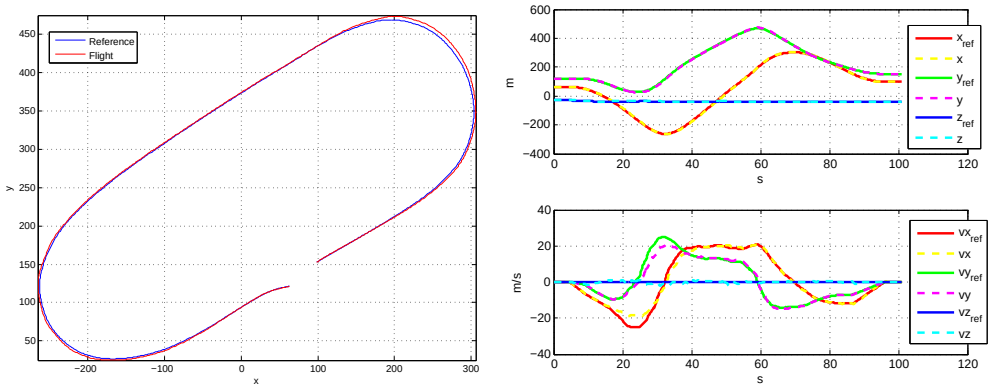


Figure 4.30: Mission with user controlled speed

4.2.3 Following Moving Ground Target

The method used to follow a geometric path could also be used to track a moving ground target. The idea comes from the target aiming guidance. A new coordinate system $\mathcal{D}$ is built by connecting the LOS between the vehicle to the target (see Figure 4.31). The task to follow the moving target can be decomposed in $\mathcal{D}$. In the $x-$ axis, x_{ref} is set as the distance between target and vehicle whereas on the $y-$ axis y_{ref} is zero. Then the position command filter can be executed on each axis to generate the

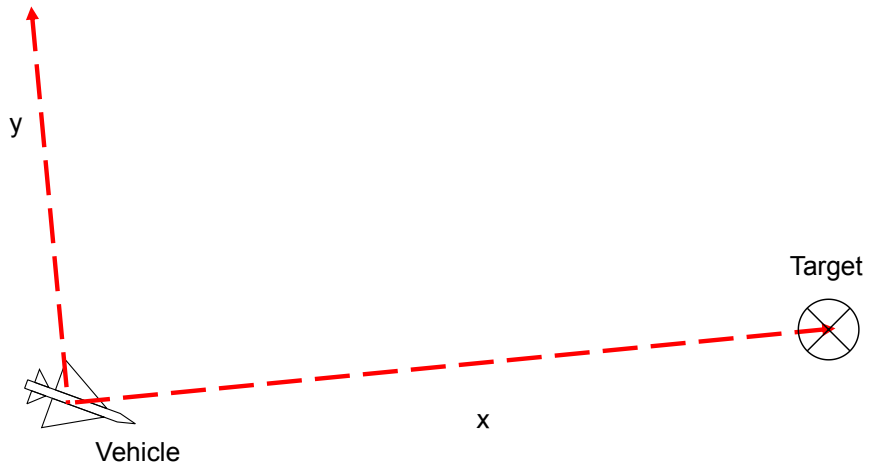


Figure 4.31: LOS coordinate system connecting vehicle to target

desired reference trajectory. An simulated example depicting the vehicle following a ground target moving in C shape path is shown in Figure 4.32. The proposed method cannot follow a moving target with zero position tracking error. But it is not crucial for tasks like monitoring and inspection. To achieve zero position error, an interception calculation is needed as introduced in [38].

4.2.4 Reference Governor

A common problem for all trajectory tracking control is the handling of saturation. Though the methods presented in this section could limit the possibility of saturation due to dynamically infeasible reference, the saturation could still happen due to system malfunction, extra strong disturbance or actuator damage. In such a case, the reference provided by command generator must be reset to guarantee the input saturation is not violated. Based on the controller design in Figure 2.14, an reference governor is introduced and illustrated in Figure 4.33. From Equation 2.11, the controller's output before the saturation can be expressed as

$$\boldsymbol{u}_{\text{out}} = \boldsymbol{a}_{\text{ref}} + K_{\text{p}}(\boldsymbol{p}_{\text{ref}} - \boldsymbol{p}) + K_{\text{v}}(\boldsymbol{v}_{\text{ref}} - \boldsymbol{v}) \tag{4.13}$$

where

$$K_p = F_{\text{aug}}(1)$$
$$K_v = F_{\text{aug}}(2)$$

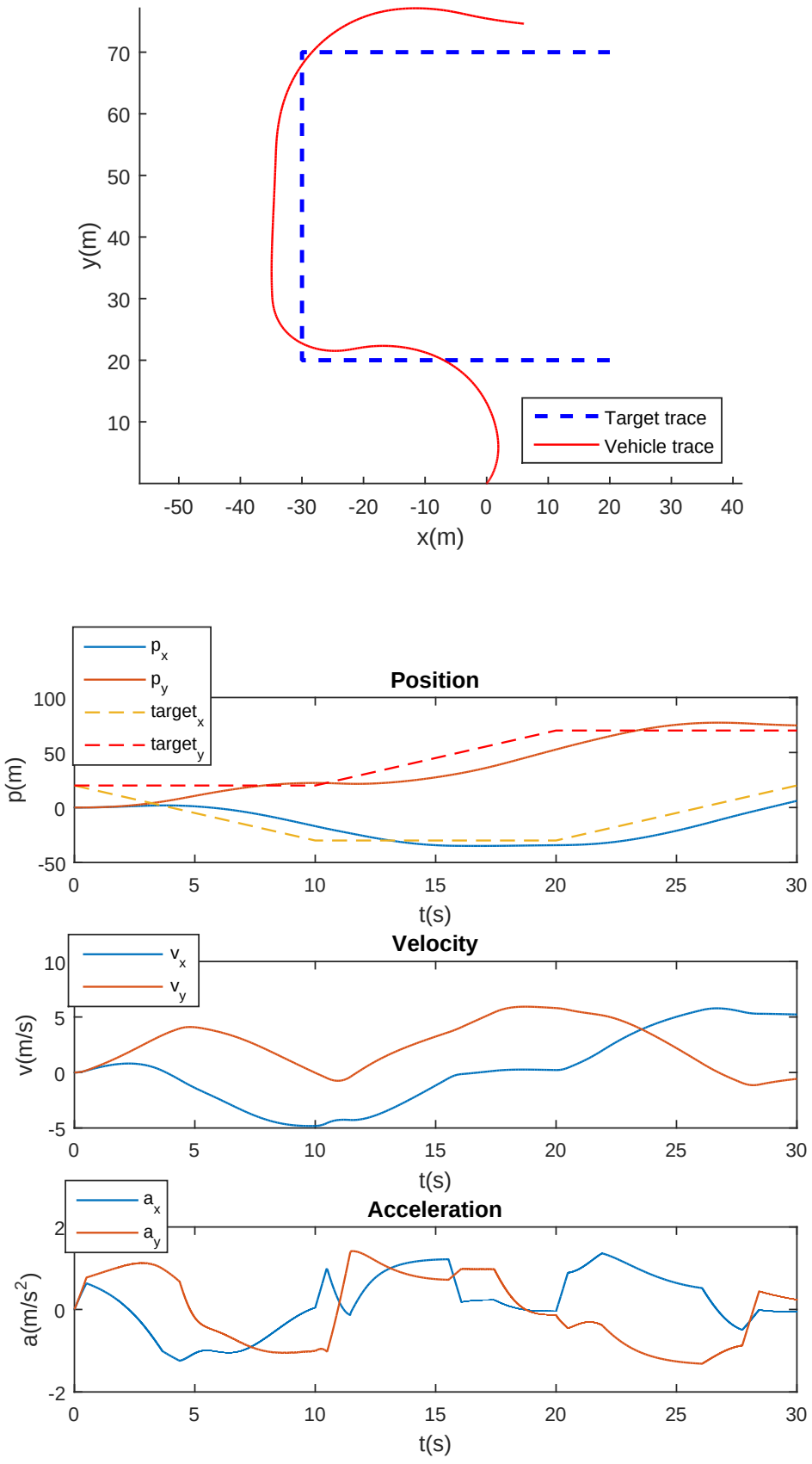


Figure 4.32: Following a moving ground target

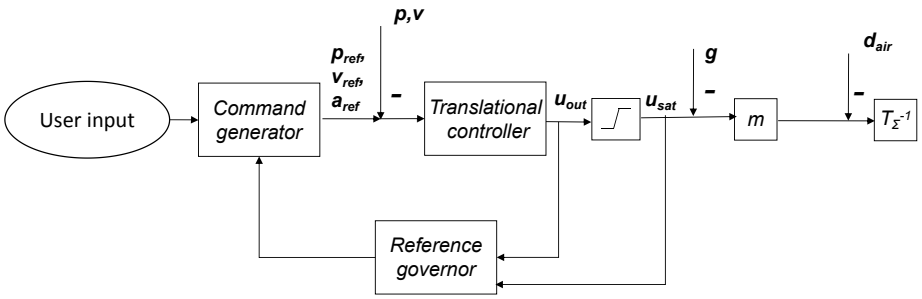


Figure 4.33: Reference governor with translational controller

Denote the saturated value of $\boldsymbol{u}_{\mathrm{out}}$ as $\boldsymbol{u}_{\mathrm{sat}}$, the task is then to design an Algorithm to reset the value of $\boldsymbol{p}_{\mathrm{ref}}, \boldsymbol{v}_{\mathrm{ref}}$ when the saturation is triggered. Consider the following case, if a vehicle at hovering is suddenly blown away by strong wind which causes its input saturation. The desired strategy is to counter the disturbance with maximum effort and return to its desired point with constrained states when disturbance is reduced. With the presented situation in mind, the following algorithm is developed. Firstly, it is needed to find the value of

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1 & 0 & 0 \\ 0 & \beta_2 & 0 \\ 0 & 0 & \beta_3 \end{bmatrix}$$

such that

$$\begin{aligned} \boldsymbol{p}_{\mathrm{d}} &= \boldsymbol{\beta}(\boldsymbol{p}_{\mathrm{ref}} - \boldsymbol{p}) + \boldsymbol{p} \\ \boldsymbol{v}_{\mathrm{d}} &= \boldsymbol{\beta}(\boldsymbol{v}_{\mathrm{ref}} - \boldsymbol{v}) + \boldsymbol{v} \\ \boldsymbol{a}_{\mathrm{d}} &= \boldsymbol{\beta}\boldsymbol{a}_{\mathrm{ref}} \end{aligned} \tag{4.14}$$

would satisfy

$$\boldsymbol{u}_{\mathrm{sat}} = \boldsymbol{a}_{\mathrm{d}} + K_{\mathrm{p}}(\boldsymbol{p}_{\mathrm{d}} - \boldsymbol{p}) + K_{\mathrm{v}}(\boldsymbol{v}_{\mathrm{d}} - \boldsymbol{v}) \tag{4.15}$$

Subtract Equation 4.15 from 4.13, the value of $\boldsymbol{\beta}$ on each individual axis i can be calculated as

$$\beta_i = \frac{u_{\mathrm{sat}i}}{u_{\mathrm{out}i}} \tag{4.16}$$

The value of $\boldsymbol{\beta}$ is then used to reset the command generator's internal states. Note that when $\boldsymbol{\beta} = 0_{3\times3}$, the output from translational controller is 0. Further, when $\boldsymbol{\beta} = I_{3\times3}$, the value of u_{out} will not be modified. To reset the command generator's internal state

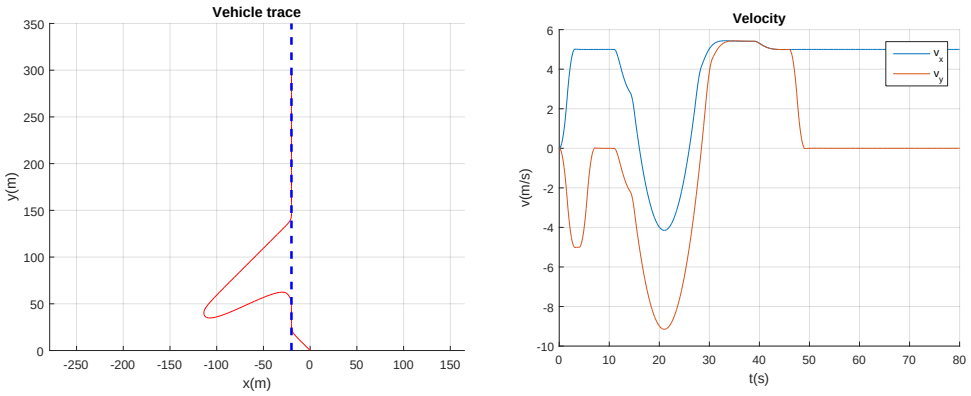


Figure 4.34: Disturbance rejection with reference governor

$\boldsymbol{p}_{\mathrm{I}} = [p_{\mathrm{I}x}, p_{\mathrm{I}y}, p_{\mathrm{I}z}]^{\mathrm{T}}, \boldsymbol{v}_{\mathrm{I}} = [v_{\mathrm{I}x}, v_{\mathrm{I}y}, v_{\mathrm{I}z}]^{\mathrm{T}}$ and $\boldsymbol{a}_{\mathrm{I}} = [a_{\mathrm{I}x}, a_{\mathrm{I}y}, a_{\mathrm{I}z}]^{\mathrm{T}}$, the method of back calculation for integrator anti windup can be utilized here as

$$\begin{aligned}
\boldsymbol{p}_{\mathrm{I}} &= \boldsymbol{p}_{\mathrm{I}} - k_{\beta 1}(I_{3\times 3} - \boldsymbol{\beta})(\boldsymbol{p}_{\mathrm{I}} - \boldsymbol{p}) \\
\boldsymbol{v}_{\mathrm{I}} &= \boldsymbol{v}_{\mathrm{I}} - k_{\beta 2}(I_{3\times 3} - \boldsymbol{\beta})(\boldsymbol{v}_{\mathrm{I}} - \boldsymbol{v}) \\
\boldsymbol{a}_{\mathrm{I}} &= \boldsymbol{a}_{\mathrm{I}} - k_{\beta 3}(I_{3\times 3} - \boldsymbol{\beta})\boldsymbol{a}_{\mathrm{I}}
\end{aligned}$$

where $k_{\beta 1,2,3} \in [0, 1]$ are design parameters. Though lacking theoretical proof, the possible benefit of the reference governor is depicted using simulation as in Figure 4.34 and 4.35. In the simulation, a vehicle is performing a path following task with its max velocity set as $5\,\mathrm{m/s}$ for both $x-$ and $y-$ axis, a strong wind gust starts at $11\,\mathrm{s}$ and ends at $33\,\mathrm{s}$ blows the vehicle away from the tracked path. For the case with reference governor, the vehicle returns to its desired path with a limited velocity around $5\,\mathrm{m/s}$ which is desired for the onboard sensors and measurement units. For the case without reference governor, the vehicle returns to the path with a maximum speed of $32\,\mathrm{m/s}$ (combining 2 axes) which could cause the measurement unit to fail. In the real flight scenario, a simpler practice is to limit the magnitude of $\boldsymbol{p}_{\mathrm{I}} - \boldsymbol{p}$ by having $\boldsymbol{p}_{\mathrm{I}} - \boldsymbol{p} = C_{lim} \cdot \frac{\boldsymbol{p}_{\mathrm{I}} - \boldsymbol{p}}{\|\boldsymbol{p}_{\mathrm{I}} - \boldsymbol{p}\|}$ when $\|\boldsymbol{p}_{\mathrm{I}} - \boldsymbol{p}\| > C_{lim}$. Here, the C_{lim} is the limited magnitude. The reference governor helps to limit the control input and saved the vehicle during the AVIC Cup 2015 competition when the position reference suddenly jump due to the mission manager malfunction.

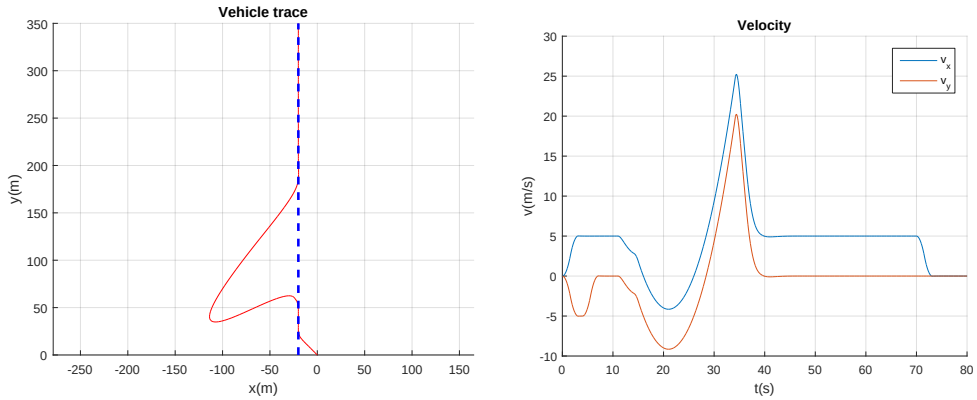


Figure 4.35: Disturbance rejection without reference governor

4.2.5 Triple Integrator TPBVP

Like the case in Section 4.1.5, sometimes it is desirable to know the whole trajectory without performing the forward simulation. From Equation 4.10, this is equivalent to find a TPBVP solver for a triple integrator. As proved in [66], such problem could be solved using seven motion segments where each of them can be described as

$$R_{\mathrm{p}}(t, p_0, v_0, a_0, u_{\mathrm{j}}) = p_0 + v_0 \cdot t + \frac{1}{2} a_0 \cdot t^2 + \frac{1}{6} u_{\mathrm{j}} \cdot t^3$$

These seven segments can be categorized into

1. Velocity increasing (VI) phase.

2. Velocity constant (VC) phase.

3. Velocity decreasing (VD) phase.

A typical 3-phase-7-segment trajectory is illustrated in Figure 4.36. For each segment, the value of u_{j} is either $u_{\mathrm{j}_{\max}}$, $-u_{\mathrm{j}_{\max}}$ or 0. The VI and VD phases would each takes at most 3 segments while the VC phase owns the remaining 1 segment. As shown in Figure 4.13 and 4.12, for each velocity changing phase, the acceleration profile could either be wedge (W) or trapezoidal (T) shaped. The trajectory in Figure 4.36 has a T-T type acceleration profile. Based on the shape of the acceleration profile, a decision tree method is proposed in [59, 66] to solve the TPBVP.

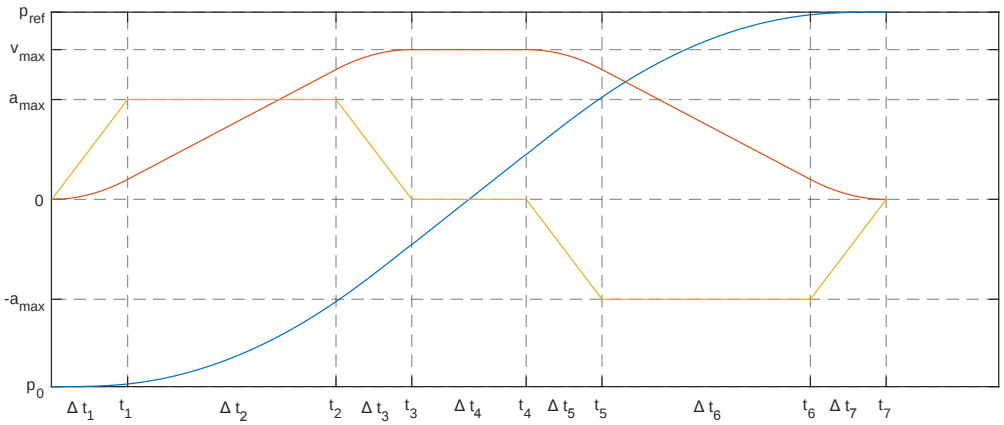


Figure 4.36: 7 segments of the point to point maneuver

Decision Tree Method

The decision tree method [59, 66] is first covered for completeness. A direct bisection method designed to reduce memory space consumption is discussed later. To solve the TPBVP, the first step is to determine the cruise direction by simulating an immediate brake from the current state. The braking trajectory can be solved using Algorithm 9 by setting the reference velocity to 0. The resulting stopping point p_{stop} is compared with the target p_{ref} to determine the cruise direction

$$d_{\text{p}} = \text{sign}(p_{\text{ref}} - p_{\text{stop}})$$

Then, the desired cruise velocity is

$$v_{\text{cruise}} = d_{\text{p}} \cdot v_{\text{max}}$$

Using its value, first try steer the system to v_{cruise} and immediately slowing down to zero speed. The resulting trajectory contains no VC phase and hence called *zero-cruise profile*. Since v_{cruise} is known, phase VI and VD can be solved through Algorithm 9 and the distance traversed during each phase (denoted as Δp_{inc} and Δp_{dec} correspondingly) can be obtained. Finally, the duration of cruising phase can be found as

$$t_{\text{cruise}} = \frac{p_{\text{ref}} - p_0 - \Delta p_{\text{inc}} - \Delta p_{\text{dec}}}{v_{\text{cruise}}}$$

If $t_{\text{cruise}} \geq 0$, our work is done, the whole position trajectory can be reconstructed as

1. VI phase: steer velocity to v_{cruise}.
2. VC phase: constant velocity v_{cruise} and holds for t_{cruise} seconds.
3. VD phase: steer velocity down to 0.

where the trajectories of VI and VD phases are already solved during constructing the *zero-cruise profile*.

However, if $t_{\text{cruise}} < 0$ which means v_{cruise} cannot be achieved, an intermediate state needs to be found to connect the VI and VD phases. The aim is to provide the final trajectory by solving the following equations

$$\begin{aligned}
&a_k = a_{k-1} + u_{\text{j}_k}\Delta t_k && k = 1...7 \\
&v_k = v_{k-1} + a_{k-1}\Delta t_k + \frac{1}{2}u_{\text{j}_k}\Delta t_k^2 && k = 1...7 \\
&a_7 = 0 \\
&v_7 = 0 \\
&\Delta t_4 = 0 \\
&p_7 = p_0 + \textstyle\sum_{k=1}^{7} R_{\text{p}}(\Delta t_k, v_{k-1}, a_{k-1}, u_{\text{j}_k})
\end{aligned} \tag{4.17}$$

where the variables a_k, v_k and p_k stands for the acceleration, velocity and position reached after segment k respectively. Δt_k is the duration of the kth segment (see Figure 4.36). The value of u_{j_k} within kth segment is determined by

$$u_{\text{j}} = [\pm, 0, \pm, 0, -, 0, +] \cdot d_{\text{p}} \cdot u_{\text{j}_{\max}}$$

The signs of u_{j} depend on

- The shape of acceleration profile, being either W-W, W-T, T-W or T-T type.
- The sign of peak accelerations during VI and VD phases. If they have the same sign, it will result in a *double deceleration* profile and the first VI phase becomes a VD phase. Otherwise, it results in a *canonical* profile as in Figure 4.36.

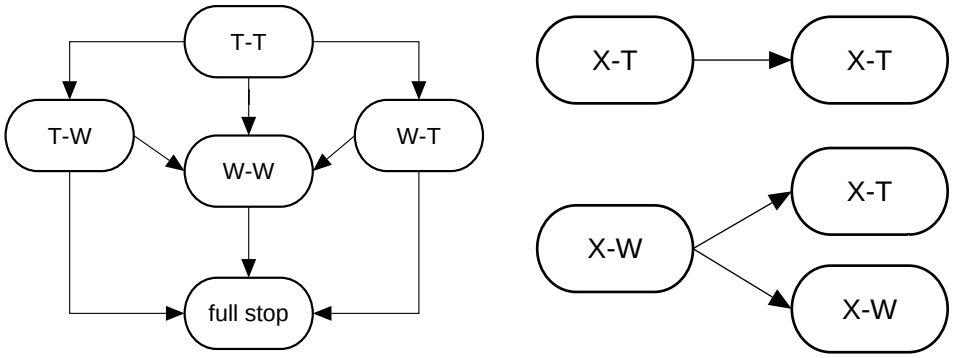
There are eight different acceleration profiles based on the classification on its shapes (W-W, W-T, T-W or T-T) and canonical types (*canonical* or *double deceleration*). For

each cases, unique solution to Equation 4.17 exists. The strategy is to find u_{j} by qualitatively deciding the acceleration profile among the eight possible cases and solve Equation 4.17. To determine the correct acceleration profile, the starting point is the previously computed *zero-cruise* profile. Since the v_{cruise} can no longer be reached, a velocity v_{reach} with smaller magnitude needs to be found. It can be achieved by decreasing the integral of $a(t)$ during the VI phase, i.e. by reducing the area covered by the piecewise acceleration profile in VI phase. To maintain a fixed final velocity, the same amount of area has to be removed from the velocity decreasing phase. Hence, to find the correct case, a decision tree is build to prune the acceleration profile repeatedly into the next shorter profile until the resulting end position p_7 no longer overshoot the target. The pruning process is described by the following Algorithm from [59]

Algorithm 12 Pruning of canonical profile

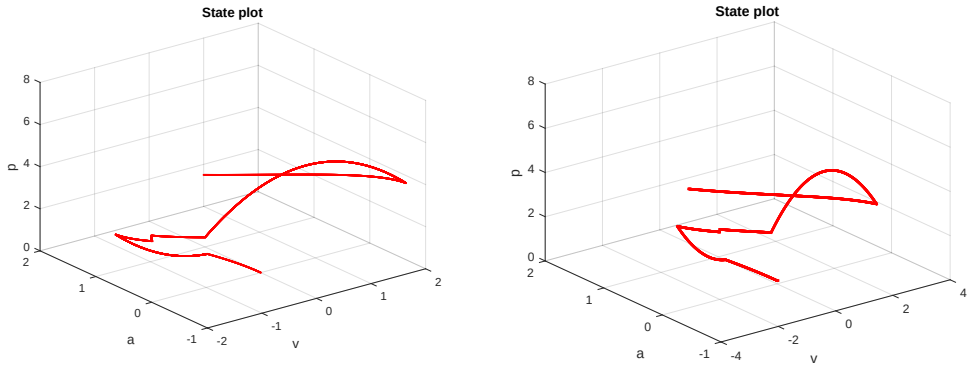
1: Compute the area Δv_{acc} and Δv_{dec} of which needs to be cut from velocity increasing and decreasing phase to transform the shape of acceleration profile.
2: Remove the smaller area from both phases and yield the next shorter profile case according a decision tree in Figure 4.37(a).
3: **if** The resulting stop point p_7 leads to an overshoot **then**
4: $\quad Continue$
5: **else**
6: $\quad$ The previous case is the correct one.
7: $\quad Break$
8: Solve Equation 4.17 numerically by substituting correct parameters obtained from the profile shape.

Once the correct case is obtained, a unique set of equations can be found with all signs fixed, then the solution can be computed numerically. On the other hand, if the *zero-cruise* profile ends with double deceleration. The pruning happens by shifting the area from the first VD phase into the second VD phase. The decision tree of pruning a double deceleration profile is given in Figure 4.37(b). A comparison between the state trajectories obtained from the TPBVP solver and the forward simulation is made. In Figure 4.38, a triple integrator is set with initial and final conditions $v_0 = 1,\ a_0 = 2,\ p_0 = 1.5,\ p_{\mathrm{f}} = 0$, constrained by $a_{\max} = 0.8,\ u_{\mathrm{j}_{\max}} = 2,\ v_{\max} = 2$. The state trajectories obtained from the TPBVP solver and the forward simulation resembles each other (provided the frequency of forward simulation is high enough).



(a) Canonical decision tree (b) Double deceleration decision tree

Figure 4.37: Decision trees for selecting the correct case
W: wedge shape, T: trapezoidal shape, X: ether T or W



(a) TPBVP solver (b) Forward simulation

Figure 4.38: State trajectory obtained by triple integrator TPBVP solver and the forward simulation

Direct Bisection Search Method

The solving of TPBVP is made very efficient through the decision tree. A dedicated solver (Reflexxes motion library [68]) is built by the author of [66]. However, the method by [59] suffers from numerical problem, and the Reflexxes library is difficult to be used directly in our low-cost flight controller (PixHawk) due to its memory space limitation. Therefore, the author proposed a direct bisection method to reduce the memory space consumption at the expense of less functions[2] compared to the Reflexxes motion library. Using the function definitions in Equation 4.11 and 4.12, the method is described in Algorithm 13. The main idea of our approach can be summarized as follows: Starting from the *zero-cruise* profile, track its first phase (VI for canonical

[2]The implemented onboard library only covers 1 DOF situation.

zero-cruise profile, VD for double deceleration *zero-cruise* profile) for t_{pb} seconds then immediately steer the velocity down to zero. If the stop point overshoots the target, then t_{pb} is reduced. Alternatively, if it undershoots the target, t_{pb} needs to be increased. The value of t_{pb} is adjusted through bisection.

In Algorithm 13, line 3–6 calculates the cruise direction and the desired cruise velocity. Line 7–12 solve the *zero-cruise* profile and check whether a cruise phase exists. If so, the cruise time is obtained through line 13. Otherwise, a bisection search (line 16–29) on t_{pb} is performed. For each bisection cycle, the first phase of a *zero-cruise* profile is executed for t_{pb} seconds (line 21) and the system is steered to full stop immediately (line 22–23). The resulting stop point is compared with the target to determine the bisection direction (line 24–27). If the error between these two points is small enough, the search terminates. The adjusting of t_{pb} is through the bisection between 0 and the duration of first phase from the *zero-cruise* profile. For the canonical profile, if the time duration of the first VI phase is reduced, so will be the v_{reach}. This is equivalent to reducing area from both VI and VD phases. Similarly, for double deceleration profile, if the duration of the first VD phase is reduced, the second VD phase spends more effort on deceleration. This is the same as an area shifting from the first phase to the second phase.

Using the outputs from Algorithm 13, the trajectory could be reconstructed at any given time t. The reconstruction is split into three different phases:

1. When $0 \leq t < t_{\mathrm{pb}}$, the trajectory is described by

   $$\mathrm{StateExam}(v_0, a_0, p_0, t_{1\mathrm{a}}, t_{2\mathrm{a}}, t_{3\mathrm{a}}, j_{1\mathrm{a}}, j_{3\mathrm{a}}, t).$$

2. When $t_{\mathrm{pb}} \leq t < t_{\mathrm{cruise}} + t_{\mathrm{pb}}$, the trajectory is described by

   $$\mathrm{StateExam}(v_{\mathrm{cruise}}, 0, p_{\mathrm{pb}}, 0, t_{\mathrm{cruise}}, 0, 0, 0, t - t_{\mathrm{pb}})$$

   where

   $$(p_{\mathrm{pb}}, v_{\mathrm{pb}}, a_{\mathrm{pb}}) = \mathrm{StateExam}(v_0, a_0, p_0, t_{1\mathrm{a}}, t_{2\mathrm{a}}, t_{3\mathrm{a}}, j_{1\mathrm{a}}, j_{3\mathrm{a}}, t_{\mathrm{pb}}).$$

3. When $t_{\mathrm{cruise}} + t_{\mathrm{pb}} \leq t$, the trajectory has two cases. If $t_{\mathrm{cruise}} > 0$, it is then written as

Algorithm 13 Triple integrator TPBVP solver, bisection method

Input: $p_0, v_0, a_0, v_{\max}, a_{\max}, u_{\mathrm{j}_{\max}}, p_{\mathrm{ref}}$

Output: $t_{1\mathrm{a}}, t_{2\mathrm{a}}, t_{3\mathrm{a}}, j_{1\mathrm{a}}, j_{3\mathrm{a}}, t_{1\mathrm{b}}, t_{2\mathrm{b}}, t_{3\mathrm{b}}, j_{1\mathrm{b}}, j_{3\mathrm{b}}, t_{\mathrm{pb}}, v_{\mathrm{cruise}}, t_{\mathrm{cruise}}$

$(t_1, t_2, t_3, j_1, j_3) = \mathrm{VelParamGen}(v_0, a_0, a_{\max}, u_{\mathrm{j}_{\max}}, 0)$

$(p_{\mathrm{sp}}, v_{\mathrm{sp}}, a_{\mathrm{sp}}) = \mathrm{StateExam}(v_0, a_0, p_0, t_1, t_2, t_3, j_1, j_3, t_3)$

$d_{\mathrm{p}} = \mathrm{sign}(p_{\mathrm{ref}} - p_{\mathrm{sp}})$

$v_{\mathrm{cruise}} = d_{\mathrm{p}} \cdot v_{\max}$

$(t_{1\mathrm{a}}, t_{2\mathrm{a}}, t_{3\mathrm{a}}, j_{1\mathrm{a}}, j_{3\mathrm{a}}) = \mathrm{VelParamGen}(v_0, a_0, a_{\max}, u_{\mathrm{j}_{\max}}, v_{\mathrm{cruise}})$

$(p_{\mathrm{f}}, v, a) = \mathrm{StateExam}(v_0, a_0, p_0, t_{1\mathrm{a}}, t_{2\mathrm{a}}, t_{3\mathrm{a}}, j_{1\mathrm{a}}, j_{3\mathrm{a}}, t_{3\mathrm{a}})$

$(t_{1\mathrm{b}}, t_{2\mathrm{b}}, t_{3\mathrm{b}}, j_{1\mathrm{b}}, j_{3\mathrm{b}}) = \mathrm{VelParamGen}(v_{\mathrm{cruise}}, 0, a_{\max}, u_{\mathrm{j}_{\max}}, 0)$

$(p_{\mathrm{fb}}, v, a) = \mathrm{StateExam}(v_{\mathrm{cruise}}, 0, p_{\mathrm{f}}, t_{1\mathrm{b}}, t_{2\mathrm{b}}, t_{3\mathrm{b}}, j_{1\mathrm{b}}, j_{3\mathrm{b}}, t_{3\mathrm{b}})$

$t_{\mathrm{cruise}} = 0$

if $\mathrm{sign}(p_{\mathrm{fb}} - p_{\mathrm{ref}}) \cdot d_{\mathrm{p}} \leq 0$ **then**

    $t_{\mathrm{cruise}} = \dfrac{|p_{\mathrm{ref}} - p_{\mathrm{fb}}|}{v_{\mathrm{cruise}}}$

    $t_{\mathrm{pb}} = t_{3\mathrm{a}}$

else

    $t_{\mathrm{cruise}} = 0$

    $tH = t_{3\mathrm{a}}$

    $tL = 0$

    for *counter* $= 1 : N$ **do**

        $t_{\mathrm{pb}} = (tH + tL)/2$

        $(p_{\mathrm{pb}}, v_{\mathrm{pb}}, a_{\mathrm{pb}}) = \mathrm{StateExam}(v_0, a_0, p_0, t_{1\mathrm{a}}, t_{2\mathrm{a}}, t_{3\mathrm{a}}, j_{1\mathrm{a}}, j_{3\mathrm{a}}, t_{\mathrm{pb}})$

        $(t_{1\mathrm{b}}, t_{2\mathrm{b}}, t_{3\mathrm{b}}, j_{1\mathrm{b}}, j_{3\mathrm{b}}) = \mathrm{VelParamGen}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, a_{\max}, u_{\mathrm{j}_{\max}}, 0)$

        $(p_{\mathrm{fb}}, v, a) = \mathrm{StateExam}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, p_{\mathrm{pb}}, t_{1\mathrm{b}}, t_{2\mathrm{b}}, t_{3\mathrm{b}}, j_{1\mathrm{b}}, j_{3\mathrm{b}}, t_{3\mathrm{b}})$

        if $\mathrm{sign}(p_{\mathrm{fb}} - p_{\mathrm{ref}}) \cdot d_{\mathrm{p}} < 0$ **then**

            $tL = t_{\mathrm{pb}}$

        else

            $tH = t_{\mathrm{pb}}$

        if $|p_{\mathrm{fb}} - p_{\mathrm{ref}}| < \epsilon$ **then**

            break

Table 4.4: Comparison between decision tree and direct bisection method

	Decision Tree	Direct bisection
Average solving time	2.2 µs	2.4 µs
Size of library file	5072 kB	931 kB
Size of exe file	281 kB	19 kB

$$\mathrm{StateExam}(v_{\mathrm{cruise}}, 0, p_{1\mathrm{b}}, t_{1\mathrm{b}}, t_{2\mathrm{b}}, t_{3\mathrm{b}}, j_{1\mathrm{b}}, j_{3\mathrm{b}}, t - (t_{\mathrm{cruise}} + t_{\mathrm{pb}}))$$

where

$$p_{1\mathrm{b}} = p_{\mathrm{pb}} + v_{\mathrm{cruise}} \cdot t_{\mathrm{cruise}}.$$

On the other hand, if there is no cruising phase, it is described by

$$\mathrm{StateExam}(v_{\mathrm{pb}}, a_{\mathrm{pb}}, p_{\mathrm{pb}}, t_{1\mathrm{b}}, t_{2\mathrm{b}}, t_{3\mathrm{b}}, j_{1\mathrm{b}}, j_{3\mathrm{b}}, t - (t_{\mathrm{cruise}} + t_{\mathrm{pb}})).$$

A comparison between the direct bisection method and the decision tree method from [68] is made. The problem is set to solve a single axis TPBVP. The maximum velocity is set as 10, the maximum acceleration is 1.5 and the maximum jerk is 1.0. The target point is always set as the origin. The initial condition covers $p_0 \in [-20, 20], v_0 \in [-20, 20], a_0 \in [-10, 10]$ with a 0.05 increment. 256 million trajectories are generated using both methods. The comparison is conducted on a Windows 10 (64 bit) desktop computer with the Intel I5-4670 CPU running at 3.4 GHz. The resulting file size, and the average solving time are given in Table 4.4. Though slower than the decision tree method, the proposed approach reduces the file size and makes it easier to be used on hardware with tight memory space. The stability of the direct bisection method is also examined during this process. Though it passes all the tests, a mathematical proof of convergence still lacks. Following the technique in [59], a safety mechanism shall be added to utilize the acceleration limited trajectory in case the direct bisection method failed to give an answer.

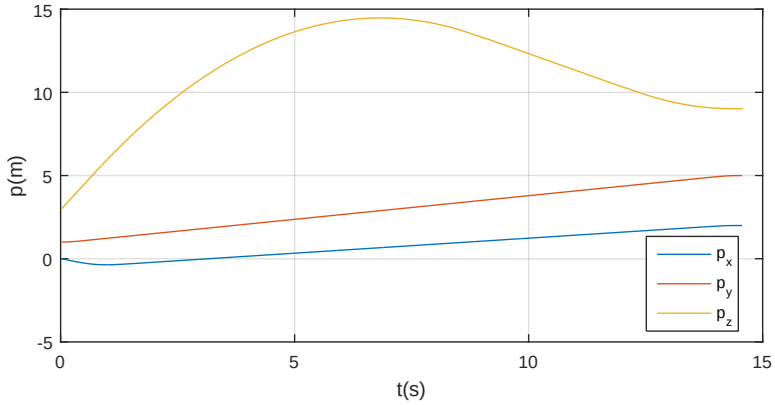
Time Synchronization

If the vehicle is required to reach a certain 3D point at the same time, a time synchronization between the three axis is needed. The idea is straightforward, find the axis with longest reaching time and prolong the reaching time of other two axises. Moreover, the inoperative time region described in [66] shall be considered during selecting the

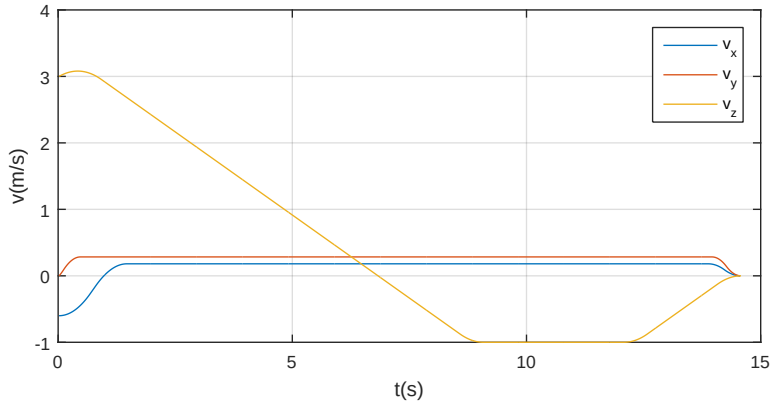
longest reaching time. The increase in reaching time is done by reducing the magnitude of v_{reach} and filling up the resulting gap between the stop point and p_{ref} by creating a VC cruise phase. Like the one-dimensional example, decision trees can be built to obtain the final trajectory as in [66]. However, bisection method might also be used. First, the v_{reach} is initialized as v_{cruise}, then the following bisection searching steps is executed

1. Calculate the distance traveled during reaching v_{reach} from current state and from v_{reach} to zero. Denote this traveled distance as Δp_{reach} and the total time consumed as t_{reach}.
2. Calculate the time of VC phase as $t_{\text{cruise}} = \dfrac{p_{\text{ref}} - \Delta p_{\text{reach}} - p_0}{v_{\text{reach}}}$.
3. Calculate the total time of consumption as $t_{\text{tot}} = t_{\text{reach}} + t_{\text{cruise}}$
4. If $t_{\text{cruise}} < 0$ or $t_{\text{tot}} < t_f$, reduce the magnitude of v_{reach} through bisection.
5. If $t_{\text{tot}} > t_{\text{f}}$, increase the magnitude of v_{reach} through bisection.

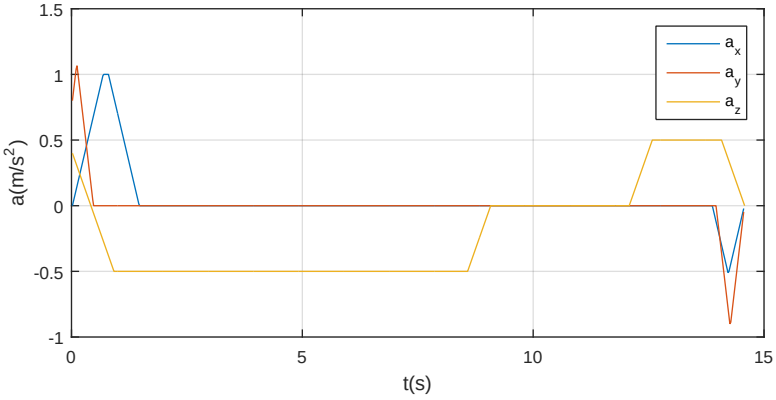
A time synchronized trajectory among three axis is shown in Figure 4.39. The initial conditions are $\boldsymbol{p}_0 = [0, 1, 3]$, $\boldsymbol{v}_0 = [-0.6, 0, 3]$, $\boldsymbol{a}_0 = [0, 0.8, 0.4]$, the position target is $\boldsymbol{p}_{\text{ref}} = [2, 5, 9]$ and the constraints are $\boldsymbol{v}_{\max} = [5, 3, 1]$, $\boldsymbol{a}_{\max} = [1, 2, 0.5]$, $\boldsymbol{u}_{\text{j}_{\max}} = [1.5, 3, 1]$. Though the initial conditions, end target and constraints on each axis are all different, a time synchronized target reaching is achieved.



(a) Position



(b) Velocity



(c) Acceleration

Figure 4.39: Time synchronized solution of triple integrator TPBVP

4.3 Conclusion

Online trajectory generation algorithms are proposed in this chapter with special consideration for platforms with low computational power. The online velocity command filter handles the RC transmitter command while producing dynamically feasible reference to the translational controller. The online position command filter is designed to provide state constrained reference that guides the vehicle in fully-autonomous mode. They are utilized in competitions and demonstrations (see Figure 4.40). Further, the jerk

(a) Auto water collection (b) UAV powerline inspection

Figure 4.40: Applications utilize the command generators

limited TPBVP solvers are also discussed. This is helpful for analyzing the whole trajectory in real time. The double integrator TPBVP has been successfully implemented on a safe-fly-zone application. On the other hand, in order to reduce the memory space consumption, a direct-bisection method is proposed to solve the triple integrator TPBVP. The result is a smaller file than the one provided by [68], and it is easier to be programmed onto the PixHawk flight controller.

In the next Chapter, the details of the obstacle-strewn environment guidance system are presented with real flight experiment data.

Chapter 5

Guidance in Obstacle-Strewn Environment

For unmanned vehicles to operate safely, the ability to navigate through the obstacle-strewn environment is crucial. It is the key to many other higher level tasks, such as flocking, exploration, and search-rescue mission. Moreover, in most cases, the cluttered environment is unknown until it is in the range of the vehicle's onboard sensors. Therefore, a vehicle must be able to perform *sensor-guided-sensor-guarded* guidance in such scenario to pursue its target while avoiding possible collisions. Again, the trajectory led guidance system in Chapter 2 can be utilized to solve the presented issue. A new *perception* module is introduced as a sub-system of the navigator unit. Its task is to translate the sensor information into data structures that could be understood and utilized by the guidance system. The approaches to the perception and guidance phases vary mainly due to the difference in vehicle types, sensors, targeted environments, and applications. In this chapter, let us focus on developing a trajectory generation based guidance system for RUAV in GPS-denied and obstacle-strewn environments.

Most successful obstacle avoidance strategies see their usage in ground vehicles and robots. Unfortunately, these algorithms are hard to be implemented directly to RUAVs due to limited payload and computational power. Therefore, an algorithm that consumes less memory and CPU time need to be developed, which is the subject of study in this chapter.

To save memory usage, a rolling map structure is used to handle the infinitely large

environment with limited memory space. The online position trajectory generator presented in Section 4.2.5 is used to achieve a reliable and efficient planning. Further, an algorithm is proposed to split the guidance problem into a series of TPBVPs so that the vehicle could travel smoothly on its way to the target.

The rest of the chapter is organized as follows: In Section 5.1, the hardware system and platform are introduced. The selection and construction of hardware heavily affect the design of the algorithm. Section 5.2 presents the perception module and the algorithms for analyzing the environment, whereas Section 5.3 discusses the various algorithms utilized in the guidance level, which are capable of generating smooth trajectories in real time to lead the vehicle to fly through obstacles in the unknown environment[1]. The experimental results and analysis are given in Section 5.4. Finally, the extension of the algorithm to relaxed formation and flocking of multiple RUAVs in the obstacle-strewn environment is discussed.

[1]Some figures are adopted from author's paper [33] for illustration purpose

5.1 System and Platform

Fuselage

The platform adopted to develop the obstacle-strewn environment guidance method is a BlackLion-068 octocopter shown in Figure 5.1. It utilizes an X8 configuration which consists of 8 motors mounted on an 'X' shaped frame with four sets of clockwise (CW) and counter-clockwise (CCW) rotating propellers. The two motors on the same arm have the same direction of rotation.

Figure 5.1: The X8 configuration octo-copter BL-068

The overall dimension is 26 cm in height and 68 cm from tip-to-tip including the propeller protection. The bare platform weighs about 1 kg including motors, propellers, and protections while the maximum lift-off weight is around 4 kg. The compact size and large payload of the UAV make it an idea platform to test various algorithms.

Sensors and Avionics

The platform comes with an attitude loop flight controller, an Intel Next Unit of Computing (NUC) computer with Intel I5 CPU running Robot Operating System (ROS). A TeraRanger One distance sensor with a maximum range of $14\,\mathrm{m}$ and accuracy of $\pm 2\,\mathrm{cm}$ is used for height measurement. A $30\,\mathrm{m}$ Hokuyo laser range finder is mounted on top of the vehicle for 2D environment sensing.

With the setup of sensors, the author designed obstacle avoidance algorithm focusing on flat surface maneuver for maximum safety. Also, due to the extra onboard NUC

which largely increase the onboard memory, more freedom is given to the algorithm design. For example, in solving position trajectory (triple integrator) TPBVP, the decision tree method [68] is preferred over direct bisection method since it is not worthy of sacrificing CPU time to conserve memory space now.

Mission System

The MGCS of the platform follows the structure proposed in Chapter 2. The arrangement of MGCS modules to system hardwares is shown in Figure 5.2. The GCS serves as

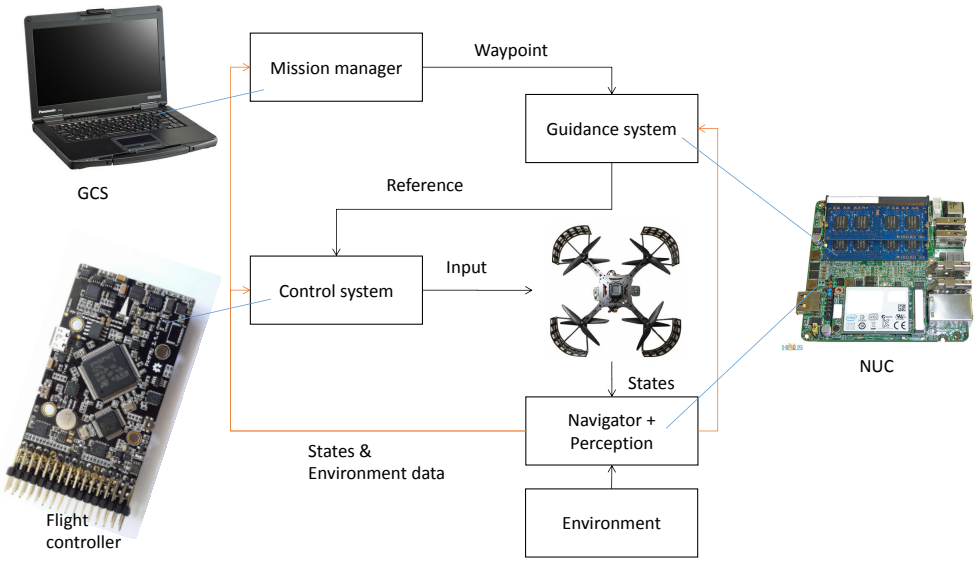


Figure 5.2: System design and module assignment

the mission planner to transfer task waypoints to the vehicle. The guidance, navigation and perception algorithm are realized on the NUC due to their relative high computational requirement while a dedicated flight controller running realtime OS guarantees the performance of the reference tracking.

5.2 Environment Perception

In this section, let us first discuss the localization and mapping methods to harvest the environmental information as an efficient data structure which could be utilized by the guidance system. The major tasks of the perception module are

- Localization of vehicle
- Generation of environmental map

which falls into the region of SLAM technique. SLAM is commonly used by vehicles in the GPS-denied environment. Its implementation depends on the equipped sensor such as RGB-D camera or laser range finder. SLAM with RGB-D camera was first covered in [71] and improved by [72]. By matching sparse visual features associate with depth information to obtain an initial estimate of the relative frame transformation, which is then refined by using iterative closest point (ICP). On the other hand, 2D laser range finder provides more accurate distance measurements in a 2D plane. SLAM with 2D laser data have been extensively studied. Open source packages like GMapping [73] and Hector-SLAM [74] are both capable for indoor environments. Algorithms describing SLAM in the forest like environment are covered in [34]. In this thesis, Hector-SLAM is used for the indoor environment. But the mapping process is heavily modified to suit our needs for keeping and analyzing the environment information. To facilitate an efficient planning, the map data structure should be able to

1. Efficiently determine the distance between a given point to its closest obstacle.
2. Evaluate a given trajectory by its clearness to the obstacle.

The former is used to check the distance between vehicle and obstacles at any given moment, whereas the second one is used for collision prediction and trajectory selection.

The task is to guide the vehicle safely to its target in the cluttered environment rather than reconstruct the map. Therefore, a grid based rolling map is chosen to store the world information. It is essentially a grid map but allowing information to exceed the map's boundary and re-enter through the other side. Algorithm 14 illustrates the transformation from a real world position to the gird coordinate in the rolling map.

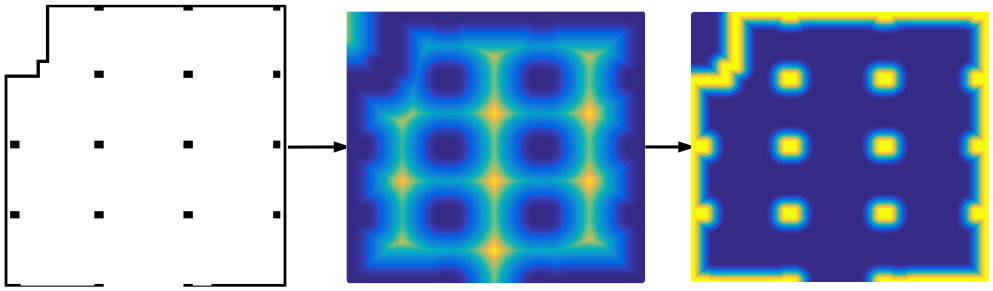


Figure 5.3: Cost map generation process.

Algorithm 14 Mapping from real position to rolling map grid

1: Input: Real world position p
2: Output: Grid on rolling map g
3: $p_{\text{discret}} \leftarrow \text{floor}((p - Origin)/GridSize + 0.5)$;
4: $g \leftarrow \text{positiveModulo}(p_{\text{discret}}, MapSize)$

The $Origin$ stands for global coordinate's origin point and it is mapped to the grid position $(0, 0, 0)$. The width of each grid is denoted as $GridSize$ and the positiveModulo() function performs the rolling action by shifting point outside the map's boundary to its positive modulo position in the map. $MapSize$ denotes the total grid number on each axis, it must be larger than the vehicle's sensor range otherwise for each scan the information will lose its continuity. Each grid g is represented by three integer $g.x, g.y, g.z$ to express its position in the grid map. Every data point captured by the laser scanner is first transformed from the vehicle's body fame $\mathcal{B}$ to the global coordinate $\mathcal{G}$ and then into the gird position through Algorithm 14.

During the map building, each grid is given a cost value based on its closeness to an obstacle. First, a method based on the Euclidean distance transformation (EDT) [75] is used to obtain the distance map. It runs a filter either through the whole map or around each obstacle grid to calculate the needed distance information. With the distance information, a cost value could be assigned to each grid. The process is illustrated in Figure 5.3. The leftmost figure shows an occupancy map which carries only binary information. The middle figure demonstrates an EDT process, the darker the grid, the closer it is to an obstacle. In the rightmost figure, the cost value is assigned to each grid based on a Gaussian function (The higher the grid's cost value, the closer it is to an obstacle).

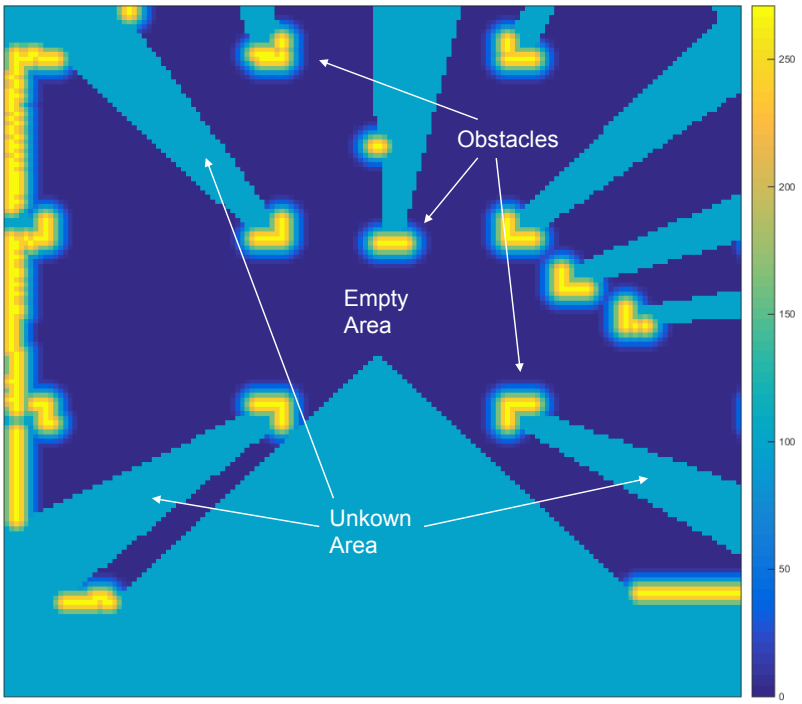


Figure 5.4: A cost map generated by laser scanner.

Figure 5.4 shows a typical grid map obtained from laser data with cost already assigned. The obstacle center has a cost of 300 and the grids around it have costs based on Gaussian function. In Figure 5.4, according to the cost value, the area can be classified into three types: the obstacles with high values, the empty area which has cost value close to zero and the unknown area with a medium cost. For convenience, the cost value of each grid is denoted as $\mathrm{cost}(g)$ and the distance to its nearest obstacle is written as $\mathrm{EDT}(g)$. The resulting map can be directly used for graph-based searches, such as the A-star algorithm. For onboard implementation, the EDT is performed around each newly added obstacle with an effective radius that is larger than the size of the vehicle.

Another problem of mapping are the moving obstacles. If the sensed obstacles are projected onto the map and accumulated forever, a moving object would leave its trace like a wall in the grid map. This would block the open path and render the map useless. Common solution to the problem is to reduce the cost value of each grid gradually through every scan as

$$\mathrm{cost}(g) = \mathrm{cost}(g) \cdot k_{\mathrm{g}}$$

where $0 < k_{\mathrm{g}} < 1$. The history information of the map is slowly washed away including

the trace left by moving obstacles. However, it also cleans the history information of static obstacles. To solve this problem, it is noticed for sensors like cameras, sonars, and laser scanners, they could only detect LOS object. And the space between the sensor and the data point is always empty. Therefore, it is desired to *clean* these areas or at least lower their cost value. To find all grids on the line segment connecting the sensor and the data point, the voxel traversal method [76] is implemented in Algorithm 15. Here, the function $point2grid()$ in line 3 and 4 represents Algorithm 14. For a line segment

Algorithm 15 Voxel traversal

1: Input: 2 points $p_{\text{start}}, p_{\text{end}}$
2: Output: Grids covered by the line segment, L
3: $g_0 \leftarrow \text{point2grid}(p_{\text{start}})$
4: $g_1 \leftarrow \text{point2grid}(p_{\text{end}})$
5: $StepX \leftarrow \text{sign}(g_1.x - g_0.x)$
6: $StepY \leftarrow \text{sign}(g_1.y - g_0.y)$
7: $Dx \leftarrow \text{abs}(g_1.y - g_0.y)$
8: $Dy \leftarrow \text{abs}(g_1.y - g_0.y)$
9: $tDeltaX \leftarrow 2Dy$
10: $tDeltaY \leftarrow 2Dx$
11: $tMaxX \leftarrow Dy$
12: $tMaxY \leftarrow Dx$
13: $g \leftarrow g_0$
14: **loop**
15: $L.\text{append}(g)$
16: **if** $g.x = g1.x \; and \; g.y = g1.y$ **then**
17: $break$
18: **if** $tMaxX < tMaxY$ **then**
19: $g.x \leftarrow g.x + StepX$
20: **else**
21: $g.y \leftarrow g.y + StepY$

with end points $p_{\text{start}}, p_{\text{end}}$, the algorithm returns a list of girds L that is covered by it. Since it is a rolling map, there exists no boundary and the algorithm only terminates when it reaches the line segment's end.

Finally, the procedures for mapping the environment is given in Algorithm 16.

Algorithm 16 Mapping

1: Input: List of obstacles $Obsl$, sensor position p_{s}
2: Output: Grid map Map with distance information
3: $Obsl_{\mathrm{fit}} \leftarrow$ empty list
4: $OccMap \leftarrow$ empty grid map with all member initialized as FALSE
5: **for** each Obstacle $Obs \in Obsl$ **do**
6: **if** $\mathrm{OccMap}(\mathrm{point2grid}(Obs.position)) =$ FALSE **then**
7: $Obsl_{\mathrm{fit}}$.append(Obs)
8: $\mathrm{OccMap}(\mathrm{point2grid}(Obs.position)) =$ TRUE
9: **for** each Obstacle $Obs \in Obsl_{\mathrm{fit}}$ **do**
10: $Grids \leftarrow \mathrm{VoxelTraversal}(p_{\mathrm{s}}, Obs.position)$
11: **for** each Grid $g \in Grids$ **do**
12: $Map(g) \leftarrow$ UNOCCUPIED
13: **for** each Obstacle $Obs \in Obsl_{\mathrm{fit}}$ **do**
14: $Map(\mathrm{point2grid}(Obs.position)) \leftarrow$ OCCUPIED
15: $\mathrm{DistanceTransform}(Map)$

In line 5–8, the algorithm first performs a discrete filtering on its input by projecting the raw data point into the grids. If two data points fall into the same grid, the later one is ignored. The procedure largely reduces the amount of data points. Then in line 9–12, the empty area between sensor and the obstacle is cleaned through voxel traversal. Finally, all the obstacle points are added back with the distance information around them. With the updated map, it becomes easy to check the distance of any point (including the vehicle's current position) to the closest obstacle by just accessing the corresponding grid's cost value.

On the other hand, in order to evaluate the clearness of a trajectory, it is turned into *foot print* which is a list of grids the vehicle covers when following the trajectory. This involves the size and orientation of the vehicle. Luckily, for an X shaped octo-copter with a similar width, length and height, it could be treated as a sphere in 3D or a circle in 2D with radius r_{v}. Note r_{v} must be the longest distance from the center of the vehicle to its edge. For a trajectory S_{t} to be collision free, it is required

$$\mathrm{Dist}_{\min}(S_{\mathrm{t}}) = \min_{g \in G_{\mathrm{s}}} \{EDT(g) - r_{\mathrm{v}}\} > 0$$

where G_{s} is the list of grids covered by S_{t}. An approximation method is then utilized to extract G_{s}. Since the trajectory could be in the form of polynomials, splines or even experience data, a general solution to exactly find the trace of the trajectory on the

grid map is time-consuming. In our implementation, all trajectories are approximated by a series of line segments, which is acquired by sampling the original trajectory at discrete time instances followed by a split and merge method. Afterward, G_{s} is obtained by voxel traversal for each line segment. The process of finding G_{s} is illustrated in Algorithm 17.

Algorithm 17 Trajectory to grids

1: Input: Trajectory S_{t}
2: Output: List of grids covered by S_{t}, G_{s}
3: $P_{\mathrm{list}} \leftarrow \mathrm{DiscreteSampling}(S_{\mathrm{t}})$
4: $L_{\mathrm{linseg}} \leftarrow \mathrm{Split\&merge}(P_{\mathrm{list}})$
5: **for** each Line segment $\in L_{\mathrm{linseg}}$ **do**
6: $\quad G_s.\mathrm{append}(\mathrm{VoxelTraversal}(\text{Line segment}))$

Here, the P_{list} stands for the list of points in the configuration space generated through time sampling, and L_{linseg} is a list of line segments returned by the split and merge process. The advantage of the voxel traversal algorithm over traditional Bresenham's method is reflected in this stage. With Bresenham's algorithm, it does not return all the grids covered by the line segment and leads to weaker checking condition. The difference between Bresenham's algorithm and the adopted one is highlighted in Figure 5.5. Note the adopted algorithm returns all the grids under the line segment while

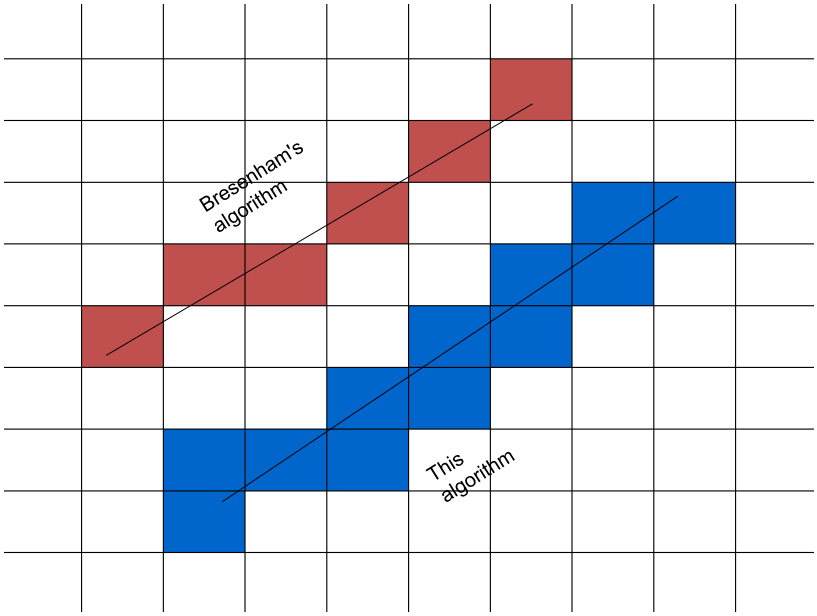


Figure 5.5: Comparison between Bresenham's algorithm
The adopted algorithm provides more safety by returning all grids covered by the line segment.

Bresenham's algorithm ignores those covered only a little. This might be dangerous if the ignored grid happens to be an obstacle. Lastly, by checking the cost value of each grid $g \in G_{\mathrm{s}}$, one could determine the safeness of the trajectory and the closest distance to any obstacles. It helps efficient planning and checking in the guidance level.

5.3 Planning and Guidance

5.3.1 Problem Overview

The problem of guidance in the obstacle-strewn environment, like many other engineering issues, can be written in the form of mathematical optimization. For a linear time-invariant discrete-time system

$$\begin{cases} \boldsymbol{x}_{k+1} &= \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k \\ \boldsymbol{y}_k &= \mathbf{C}\boldsymbol{x}_k \end{cases} \tag{5.1}$$

where A, B and C are system matrix, $\boldsymbol{x}_k$, $\boldsymbol{u}_k$ and $\boldsymbol{y}_k$ are the state, input and output of the system at time step k respectively. Denoting $\boldsymbol{U} = [\boldsymbol{u}_0, \boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_{N-1}]$, our task can be described as

$$\begin{aligned} \min_{\boldsymbol{U}} \left\{ J(\boldsymbol{U}) = {\boldsymbol{x}_N}'\mathbf{P}\boldsymbol{x}_N + \sum_{k=0}^{N-1} (\boldsymbol{x}_k'\mathbf{Q}\boldsymbol{x}_k + \boldsymbol{u}_k'\mathbf{R}\boldsymbol{u}_k) \right\} \; s.t. \\ \boldsymbol{x}_0 &= \boldsymbol{x}_{\text{start}} \\ \boldsymbol{x}_N &= \boldsymbol{x}_{\text{goal}} \\ \boldsymbol{x}_{k+1} &= \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k \\ \boldsymbol{y}_k &= \mathbf{C}\boldsymbol{x}_k \\ \boldsymbol{x}_{\min} \leqslant \boldsymbol{x}_k \leqslant \boldsymbol{x}_{\max}, &\quad \forall k \in [0, N] \\ \boldsymbol{u}_{\min} \leqslant \boldsymbol{u}_k \leqslant \boldsymbol{u}_{\max}, &\quad \forall k \in [0, N-1] \\ \boldsymbol{x} \notin \mathbb{O}, &\quad \forall k \in [0, N] \end{aligned} \tag{5.2}$$

where $\boldsymbol{x}_{\text{start}}, \boldsymbol{x}_{\text{goal}}$ are the initial and end states of the vehicle which correspond to the current state and target in our problem. The $\mathbb{O}$ is a subset of the configuration space that is related to obstacles and human set boundaries.

Until now, there is no explicit solution to the presented optimization problem. The best effort includes numerical methods, but its performance is subjected to the non-convexity of the optimization space and complexity of the dynamics. Furthermore, direct solution to these large dimension problems usually suffers from the numerical instability and could potentially generate a control sequence that would damage the hardware. Consequently, double validation of the resulting control sequences is needed

and further increases the computational burden. In this work, a two-step approach is adopted to simplify and solve the optimization problem. Though, the global optimality is no longer guaranteed, satisfying performance is achieved with real vehicle experiments.

5.3.2 Task Decomposition

Our guidance system consists of a global planner and a trajectory generator as presented in Chapter 2. The global planner extracts the connectivity information from the environment with ignored system dynamics whereas the trajectory planner focuses on producing the dynamically feasible reference for the controller. Since the rotorcraft is holonomic and capable of moving in all directions, it is simplified as a checker model on the grid map with relatively low speed (smaller than $5\,\mathrm{m/s}$). For a 2D map case, the checker could move from one grid to all of its eight neighbors at left, right, up, down, top left, bottom left, top right and bottom right or moving in straight lines. With all the simplifications and assumptions, the classical A-star algorithm or the modified RRT algorithm presented in Section 2.2.1 can be used to generate a safe path way consists of a series of line segments. Since a 2D sensor is adopted, the A-star method is preferred. A typical path way is shown in Figure 5.6.

With the path way acquired, dynamically feasible trajectories need to be designed. The most popular method [10] generates trajectory through a polynomial spline representation and quadratic optimization. The advantage of this approach lies in its ability to handle the multiple *key frames* problem directly. Here, a *key frame* means an equality constraint in the form of $\boldsymbol{x}_k = \boldsymbol{x}_{\mathrm{chosen}}$ for the optimization problem in Equation 5.2. To produce collision free solution, *key frames* are positioned along the path way to force the trajectory to stay close-by. The algorithm in Chapter 3 illustrates the formulation of optimization problem using B-splines and the UAV calligraphy example demonstrates the following of a geometric shape by assigning *key-frames*. The method usually produces high-quality trajectories with the minimum jerk or minimum snap property. However, its performance is limited by the quality of quadratic program solver. In this thesis, the triple integrator TPBVP solver is adopted with a receding horizon control (RHC) strategy to generate dynamically feasible trajectory. The end velocity and acceleration of

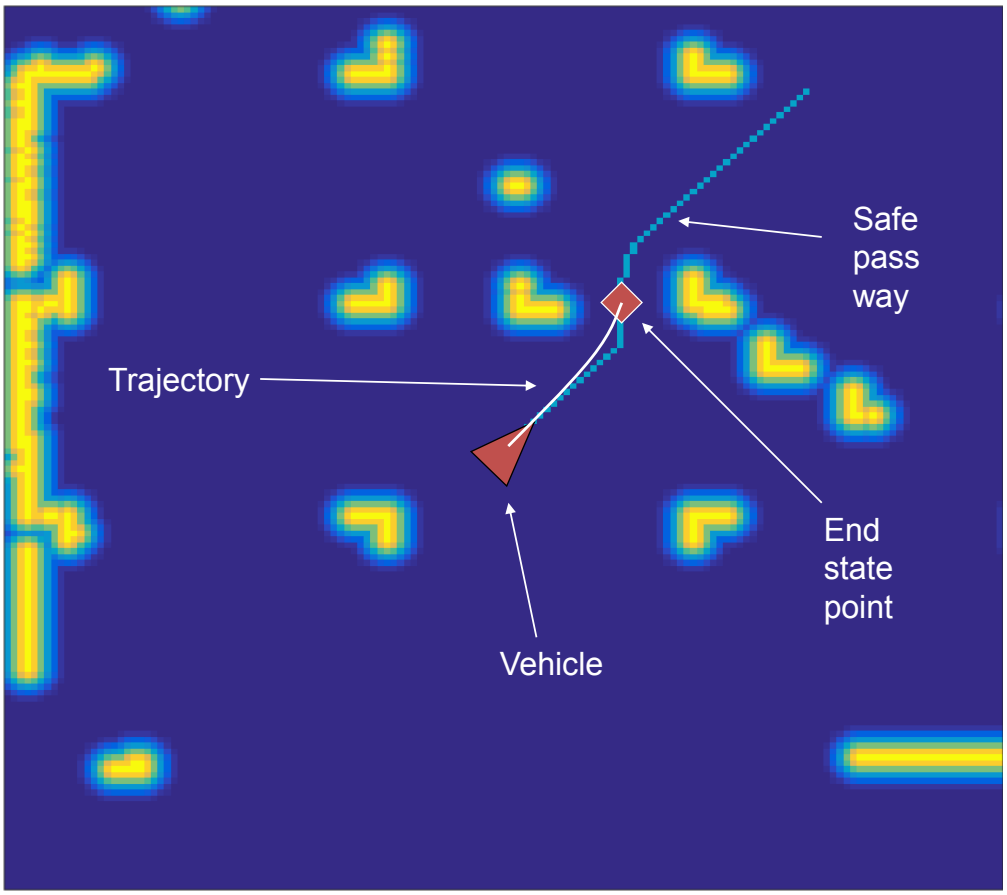


Figure 5.6: Safe path way and generated trajectory
The safe pass way is shown as green line segments. The end state point is picked around the last point stay line-on-sight to the start point.

each TPBVP are always zero. A continuous non-stop maneuver is achieved by switching to a new trajectory before reaching the end of the current one. However, if the algorithm fails due to unexpected reason, the vehicle will at least stop with a safe hover. This is commonly referred as model predictive equilibrium point control (MPEPC) strategy and frequently used for vehicle path planning [77]. Further, the generated trajectory should always stay in the vicinity of the safe path way. Here, the synchronization techniques [66, 78] can be applied to prevent large deviation from the safe path way. Also, the geometric path following technique in Section 4.2.2 can be used. A rotated coordinate is created based on the safe path way's location and orientation (see Figure 5.7). By solving TPBVP in the rotated axises $x'-$ and $y'-$ independently, a trajectory that converges to the safe path way is achieved. The target position is chosen around the end point of the safe path way's first segment. The procedures involved in each guidance

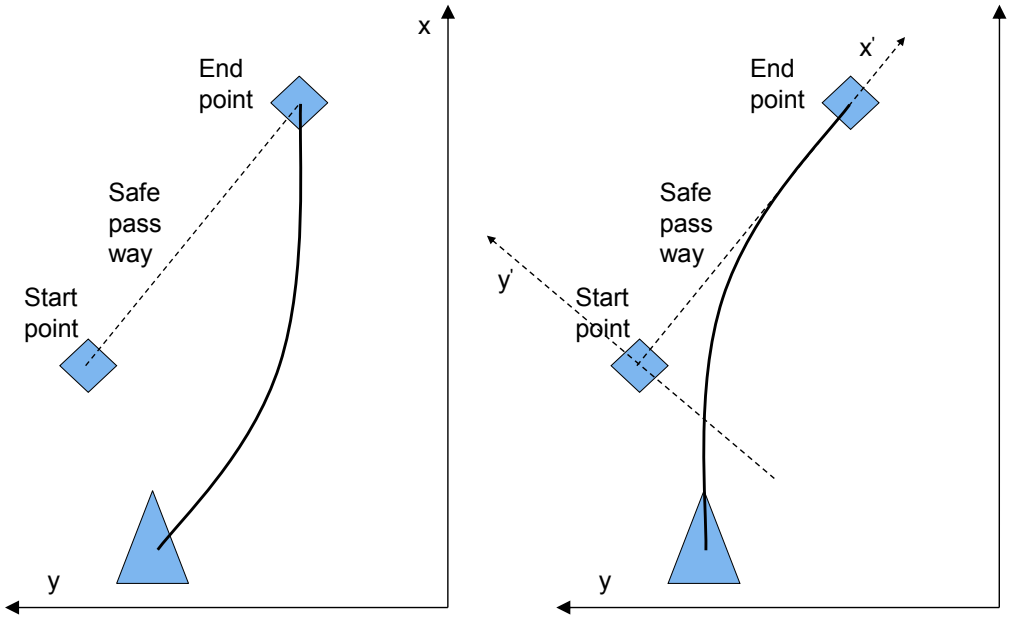


Figure 5.7: Trajectory generated independently on each axis in global frame vs rotated frame

cycle are

1. Update the environment information.

2. Generate safe path way through the global planner.

3. Formulate TPBVP along the first line segment of the safe path way.

4. Solve the TPBVP for reference trajectory.

5. Check whether the trajectory is safe, if not, randomly sample for an evasion trajectory.

6. Start tracking the new trajectory.

The above steps are repeated every t_{m} seconds with t_{m} shorter than the full time duration of the trajectory. With a new plan generated that frequently, the computational burden is heavy. Moreover, due to the nature of the proposed TPBVP solver, constant switching of trajectory could decrease the tracking control performance during aggressive maneuver because of the discontinuity on the jerk.

5.3.3 Event Triggered Trajectory Switching

To improve the trajectory quality and save computational resources, an event triggering guidance strategy is proposed. Unlike the MPEPC which plans and adopts new reference every time, it only switches to new trajectory when certain conditions are satisfied. The pseudo code for event triggered guidance is given in Algorithm 18. The current

Algorithm 18 Event triggered guidance cycle

```
1:  lastTarget ← CurrentPosition
2:  loop
3:      if Goal reached then
4:          break
5:      if CT is invalid then
6:          ET ← ETSearching()
7:          CT ← ET
8:      else
9:          if CT is in deceleration phase then
10:             PathWay ← A * Searching(lastTarget)
11:             EndState ← localTargetSearch(PathWay)
12:             P ← randomSample(EndState)
13:             MinDist ← 0
14:             for each point p ∈ P do
15:                 PT ← BVPS(CurrentState, p)
16:                 Dist ← MinDistToObstacle(PT)
17:                 if Dist > MinDist then
18:                     if PT is valid then
19:                         CT ← PT
20:                         MinDist ← Dist
21:                         lastTarget ← p
```

trajectory is denoted as CT, a possible trajectory as PT and the emergency trajectory as ET. The $lastTarget$ records the last end point for TPBVP which is also the beginning of connectivity search. During each cycle, the algorithm first exams the availability of current trajectory (line 5–7). If it is not collision free, an evading trajectory is searched and the vehicle immediately switch to it. Otherwise, if the current trajectory also happens to enter the deceleration phase, it switches to a newly generated trajectory. Firstly, the safe path way is obtained through connectivity search. Then a $\mathrm{localTargetSearch}()$ is performed to find the end point of the safe path way's first segment. A random sampling is performed around this end point to produce a list of sampled points. For each sampled point, a trajectory is generated and the one with maximum clearance is adopted (line 12–21). The $\mathrm{BVPS}()$ function stands for the position trajectory TPBVP solver

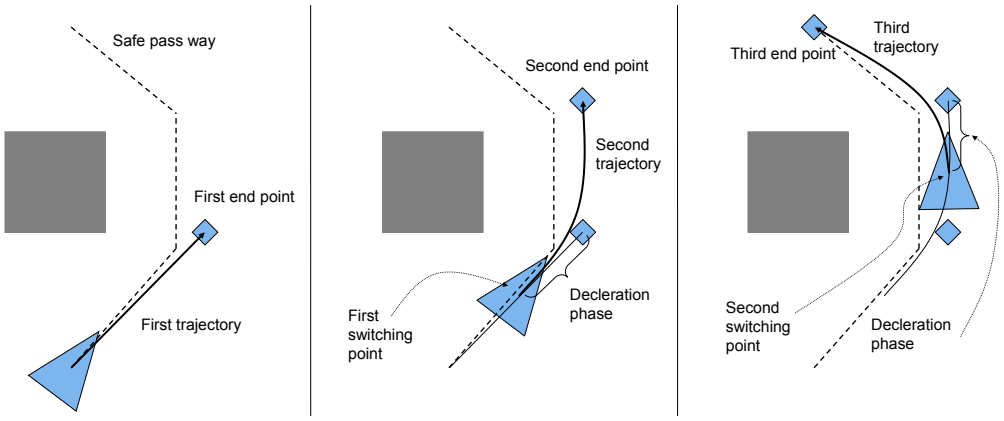


Figure 5.8: Event based trajectory switching
Step 1 (left): Safe pass way is generated, the first end point is picked around the first sharp turn. Step 2 (middle): vehicle starts to pick new end point when it is about to enter the deceleration phase of the first trajectory. Step 3 (right): by repeating the process in Step 2, vehicle could reach its target.

whereas the $\mathrm{MinDistToObstacle}()$ checks the minimum distance from a trajectory to any obstacle. By this algorithm, the planning is performed only under two conditions

1. The current trajectory is not collision free.
2. The current trajectory is about to finish and comes to hover.

which dramatically reduces the times of planning and reference switching. To consider the tracking error, the validity of trajectory reference depends on

1. Whether itself is collision-free.
2. Whether the vehicle's tracking response is collision-free.

To check the condition 2, the vehicle's tracking response needs to be predicted. When the tracking error is small, it could be done using experience formula. Otherwise, a fast forward simulation is conducted. Further, during the validity verification process in line 5, the radius used for collision checking is larger than the actual vehicle size r_{v} so that it is safe under the assumption of tracking error. In line 18, the checking radius is even larger than that in line 5, so that the adopted trajectory in line 19 will not become invalid in line 5 due to small tracking error. The process of the proposed trajectory switching method is illustrated in Figure 5.8. In case that CT is invalid, $\mathrm{ETSearching}()$ is performed to acquire an evading trajectory. The searching process

is illustrated in Algorithm19. During evading trajectory searching, trajectories aimed

Algorithm 19 Avoiding trajectory searching algorithm

```
1:  Success ← FALSE
2:  PathWay ← A ∗ Searching(currentPosition)
3:  EndState ← localTargetSearch(PathWay)
4:  P ← randomSample(EndState)
5:  MinDist ← 0
6:  for each point p ∈ P do
7:      PT ← BVPS(CurrentState, p)
8:      Dist ← MinDistToObstacle(PT)
9:      if Dist > MinDist then
10:         if PT is valid then
11:             CT ← PT
12:             MinDist ← Dist
13:             lastTarget ← p
14:             Success ← true
15: if Success = FALSE then
16:     Increase acceleration and jerk limit
17:     P_rs ← randomSample(currentPosition)
18:     for each point p ∈ P_rs do
19:         PT ← BVPS(CurrentState, p)
20:         if PT is valid then
21:             CT ← PT
22:             break
23:     Decrease acceleration and jerk limit
```

to the target but also away from collision (line 2–14) are first examined. Note that the beginning of connectivity search is now the vehicle's current position instead of the last end point of TPBVP. If these trajectories are not applicable, the vehicle is possibly in a dangerous situation. The limit on jerk and acceleration is increased to allow a more aggressive maneuver and the vehicle focuses purely on obstacle avoidance (line 15 – 23). With Algorithm 19, the vehicle now possesses the capability of handling obstacles that suddenly appear (or captured by the sensor), and dynamic obstacles such as other RUAVs or human beings.

A comparison between the event triggered and the RHC strategy is made. The RHC method is executed at 5 Hz, and planning path from the vehicle's current position at each cycle. For each trial, the testing scenario is to guide the vehicle from the same starting point to the same end point in the same simulation environment for both strategies. The comparison is made through several trials with different start and end points. The resulted position and velocity profiles are given in Figure 5.9, and the averaged data

of target reaching time and CPU consumption is given in Table 5.1. For reference, the simulation environment's 2D map is shown in Figure 5.10.

Table 5.1: Comparison between event triggered switch and RHC

	Event trigger	RHC
Number of planning	6	71
Number of switching	4	71
Target reaching time	13.3 s	15.7 s
Average planner cycle time	6.1 ms	21.7 ms

As shown in Figure 5.9, the trajectory produced by RHC method consists of more unnecessary wobbling in its velocity profile. This is due to the lack of energy conservation term in the TPBVP formulation and the frequent switching of trajectory. On the other hand, the proposed method generates smoother trajectory and reaches the target faster. The average planner cycle time is also smaller using our method (see Table 5.1). Because the planning routine is executed only when triggered which leaves most planner cycles only run the perception routine.

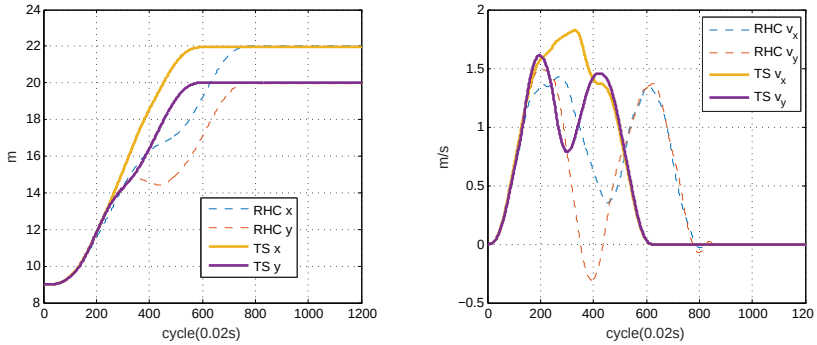
Though the performance of RHC method could be improved by introducing more complicated cost function and optimization tools, these algorithms increase the computational burden and system complexity. On the other hand, the RHC strategy (EPMPC method) is more suitable for dealing with the fast changing environment or task (like flying among many moving obstacles) at lower flight speed. Thus it is remained as an working option in the guidance module.

The TPBVP solver used in this method could also be substituted with any other algorithms as long as the resulting trajectory is dynamically feasible. In general, solving TPBVP is much faster than calculating the whole trajectory, thus more suitable to be used in an unknown environment with the realtime requirement.

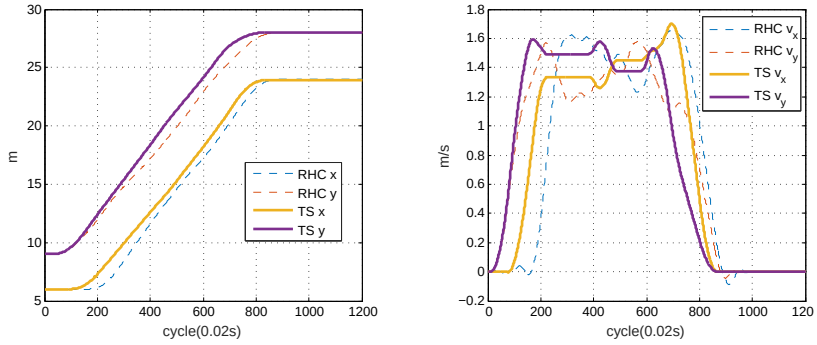
5.3.4 Multiple Waypoint Mission Management

With the proposed guidance system, multiple waypoint missions like searching and exploration is made possible. During the implementation of the indoor exploration system on the real vehicle, the following two problems need to be solved.

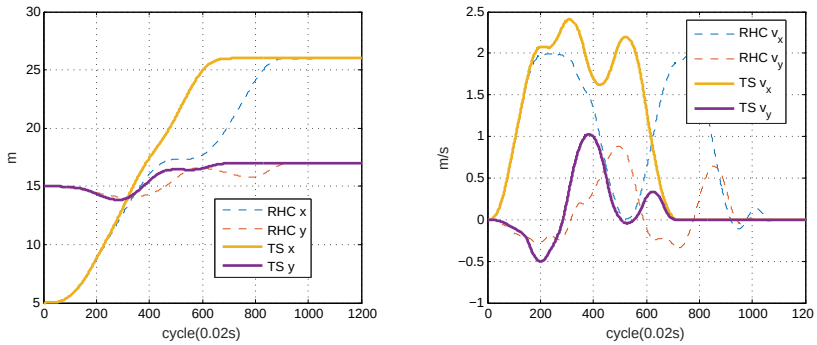
1. Sometimes, due to the lack of environmental knowledge, the mission manage-



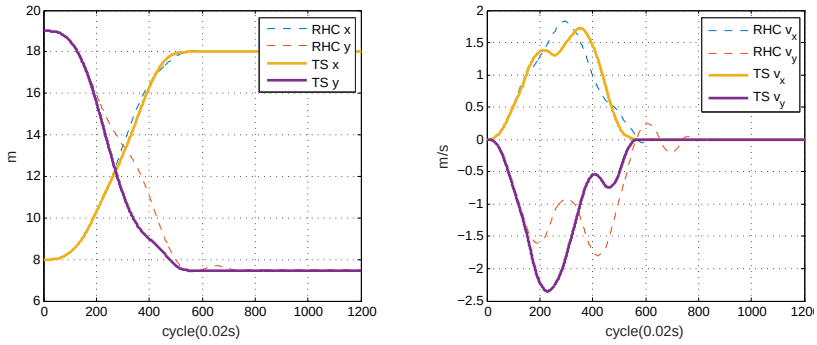
(a) Trail 1



(b) Trail 2



(c) Trail 3



(d) Trail 4

Figure 5.9: The RHC based strategy compared to the proposed trajectory switching (noted as TS) algorithm.

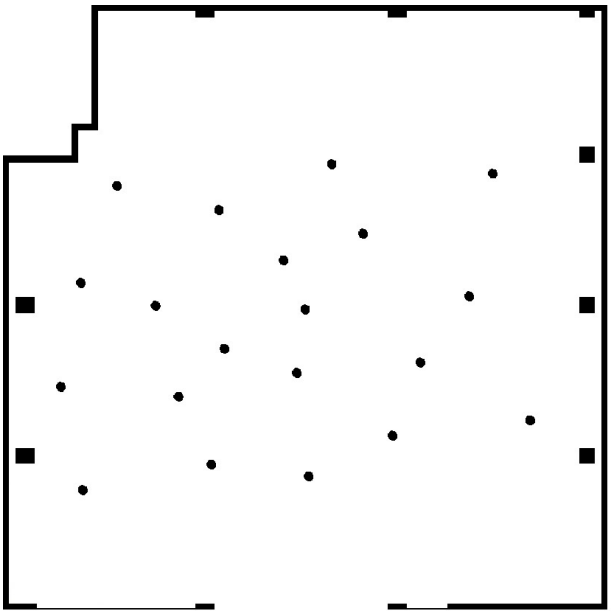


Figure 5.10: Simulation environment's 2D map for comparison between event triggered and RHC strategy.

ment system could allocate a waypoint that is entirely blocked by the obstacle. The vehicle needs to know when to give up on such waypoint and head to the next one.

2. The laser scanner only sees the forward 270 degrees of view thus it is dangerous for the vehicle to fly backward. The vehicle always needs to fly in a direction where it is covered by the laser scanner.

For the first problem, it is necessary to check whether a certain waypoint is reachable or not. This is done easily through the A-star search since it guarantees an optimal solution when any of them exists. If Algorithm 1 terminates while the CLOSED set does not contain the desired waypoint, then the specific waypoint cannot be reached. The mission management system then either give up on this waypoint or rewrite its position. The second problem is done by aligning the vehicle's heading direction towards its TPBVP end point. However, sometimes the vehicle might determine to reverse its direction suddenly, and the heading might fail to respond with enough speed. In this condition, the vehicle first find a safe position within its sensor range, create a temporary waypoint, fly to the waypoint and finally perform a hover turn to the desired direction. The hover turn mechanism is governed by a state machine so that the vehicle could resume its normal mission after the hover turn. An example of multi-waypoint exploration mission

consists of 3 waypoints is presented in Figure 5.11. The red squares are the targets 1–3 and the blue curve is the trace of the vehicle. Upon reversing of the marching direction, the mission management system directs a hover turn to make sure flying in sensor-covered area. Also, target 1 falls inside a square pillar thus cannot be reached. The mission management system then rewrites this target to a reachable position. In Figure

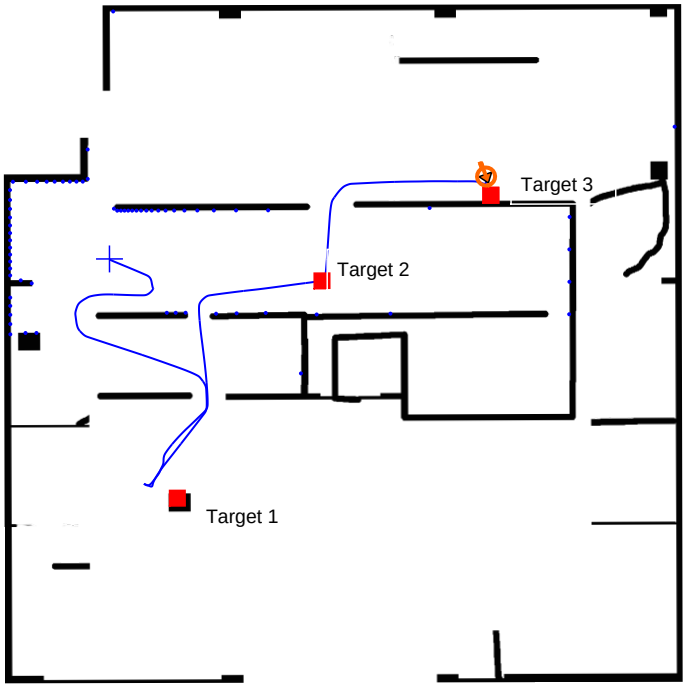


Figure 5.11: Exploration mission with multiple waypoints.

5.12, the result of map reconstruction in the 2014 IMAV competition fly-off is shown [79]. During the competition, the RUAV needs to identify objects while traveling in an office-like environment with many obstacles. The proposed guidance system is utilized in the competition, and the vehicle successfully explored the area and traveled to all the defined rooms.

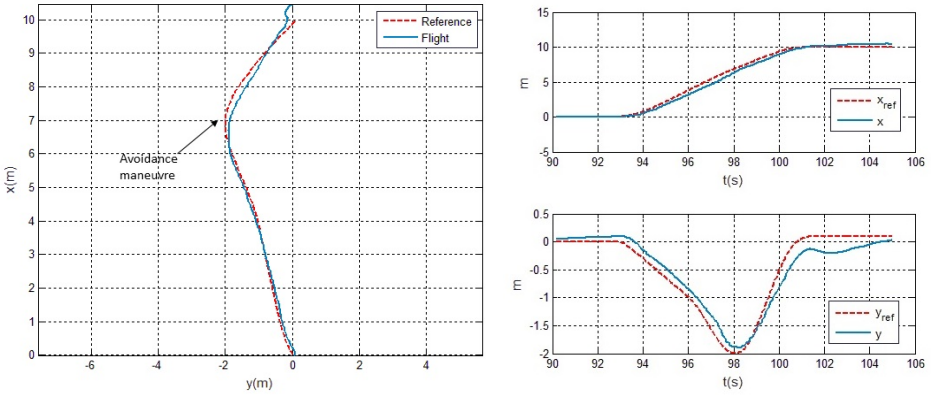
Figure 5.12: Result of map reconstruction in the IMAV competition fly-off

5.4 Flight Experiment

The experiment is performed by the BL-068 platform in an indoor clustered environment as depicted in Figure 5.13. In such an environment, the vehicle is guided safely

Figure 5.13: Experiment environment consists of pillars and other obstacles.

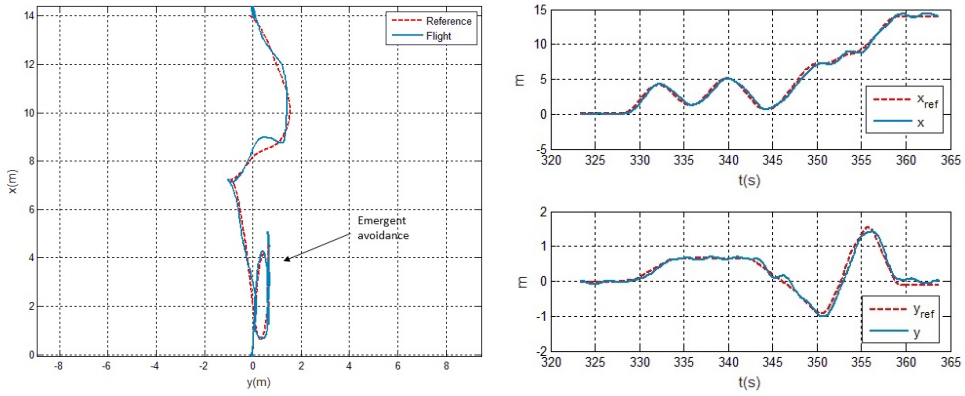
to fly through waypoints. Experiment data from real flight is presented in Figure 5.14 and 5.15. Figure 5.14 shows a typical avoidance maneuver adopted by the vehicle. The final target is 10 meters in front of the vehicle but blocked by an obstacle in between.



(a) Avoidance trajectory (b) Tracking control performance

Figure 5.14: Normal avoidance situation

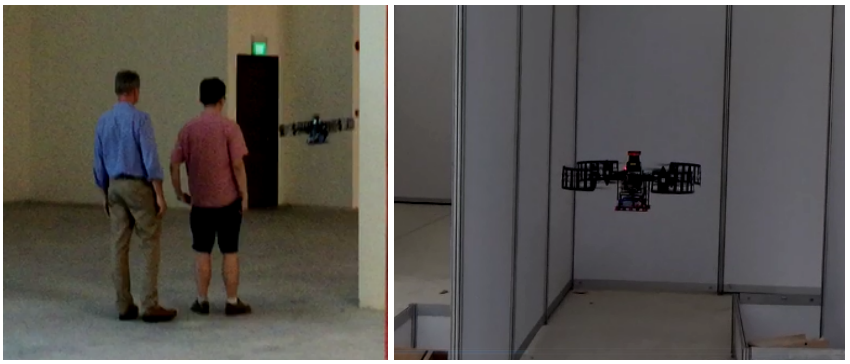
The maximum speed of the vehicle is more than $1.6\,\mathrm{m/s}$ during the flight. Figure 5.15 shows an emergent avoidance behavior. It was triggered by a faulty estimation of the obstacle location and caused the vehicle to fly too close to a pillar. When the vehicle realized the situation, it took emergent avoidance and prevented the collision. In this case, the avoidance trajectory is found without the need of increasing acceleration and jerk limit.



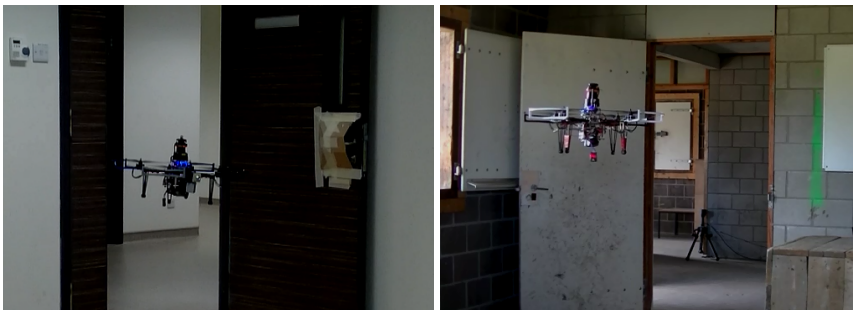
(a) Avoidance trajectory (b) Tracking control performance

Figure 5.15: Emergent avoidance

Beyond this test, which covers a GPS-less-forest-like environment, the proposed method is applied to various competitions and demonstrations. In Figure 5.16(a), it demonstrates the vehicle's capability of avoiding pedestrians by producing evading trajectory. Figure 5.16(c) shows the vehicle flying through a $90\,\mathrm{cm}$ wide door which is not much wider than the vehicle itself. Finally, Figure 5.16(b) and (d) shows the vehicle working in the indoor office like environment. In this kind of environment full of non-



(a) Avoid dynamic obstacle (b) Fly in narrow corridor



(c) Fly through narrow door (d) Fly in indoor office

Figure 5.16: Obstacle avoidance applications in various environment

convex obstacles, the global planning is crucial to prevent the vehicle from being stuck at the local minimum point. Finally, an indoor patrol mission is simulated as shown in Figure 5.17. The simulated vehicle is capable of covering the whole building without collision.

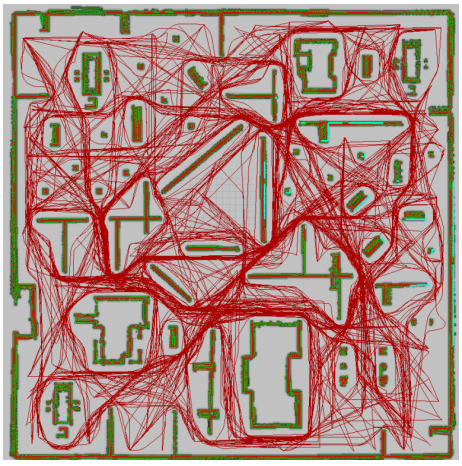


Figure 5.17: UAV patrol in simulated environment.

5.5 Relaxed Formation and Flocking

5.5.1 Relaxed Formation among Obstacles

In this Section, the usage of the proposed guidance method in multi agent case is discussed (also covered in author's paper [80]). Firstly, let us define the task to be

1. Vehicles need to form certain formation in the cluttered environment with arbitrary given initial states.

2. Vehicles need to move through the cluttered environment while trying to maintain the formation.

Since the vehicle cannot form the exact desired formation while moving in the obstacle-strewn environment, this behavior is named relaxed formation.

Using common sensors such as laser scanner, camera, and radar, it is easy to obtain the relative position of an obstacle, but rather difficult to obtain its accurate velocity since the later one involves classification and clustering. Therefore, a rough estimation of the neighbor agent's future route is done by increasing the cost value of its possible reaching area. Then the guidance method presented in the previous section is used to steer the vehicle towards its final goal. As shown in Figure 5.18, agent 0 first estimates the possible reaching area of its two neighbors (the shaded area), and assign the area with high-cost value. With the proposed A-star searching algorithm, the generated safe path way will naturally avoid the shaded area. Thus agent zero chooses the dashed path instead of squeezing through the incoming neighbors. In Figure 5.19, two simulated RUAV agent are trying to exchange their position in a cluttered environment. A sharp turn in agent 2's trajectory is resulted when it gave way to agent 1. Behavior such as reciprocal dance is not triggered in our case due to the existence of the global planner. The process is accomplished with no communication among agents.

A relaxed leader-follower formation is realized by continually change the follower's goal according to leader's position. Like in Figure 5.20, the follower (agent 0) is trying to stay on the left down corner to the leader (agent 1). When moving through obstacle-strewn environment, agent 0 then performs target tracking and collision avoidance all at the same time. It breaks out from the formation automatically when it is needed.

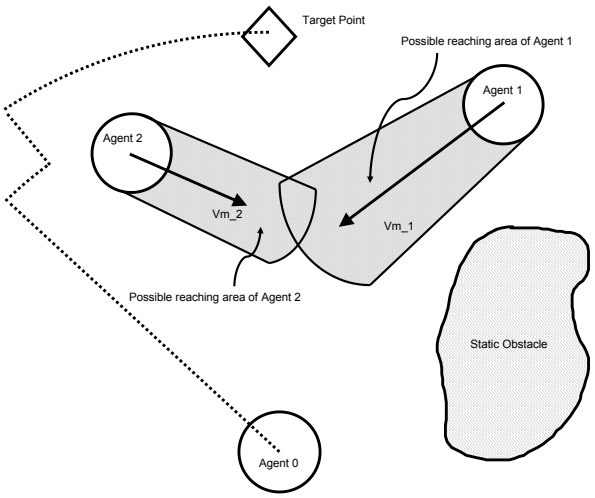


Figure 5.18: Possible reaching area of moving neighbor agent

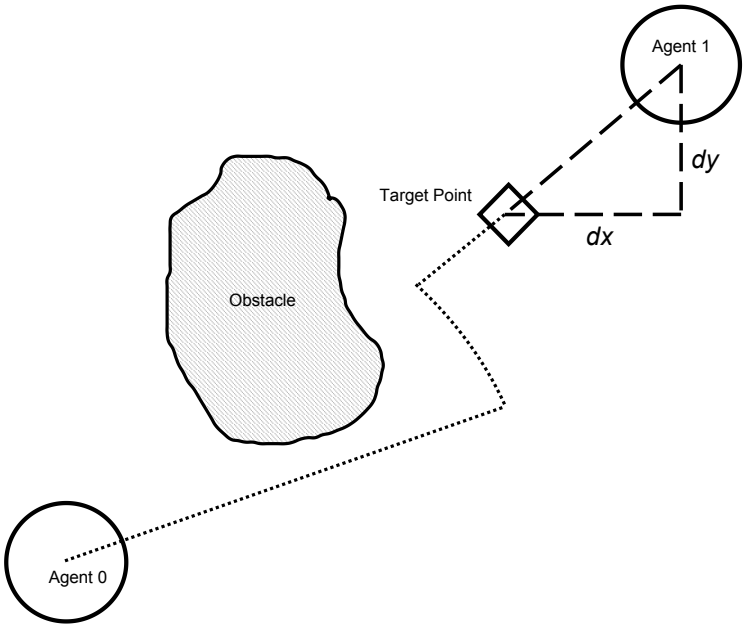


Figure 5.20: Follower trying to maintain the formation while performing obstacle avoidance

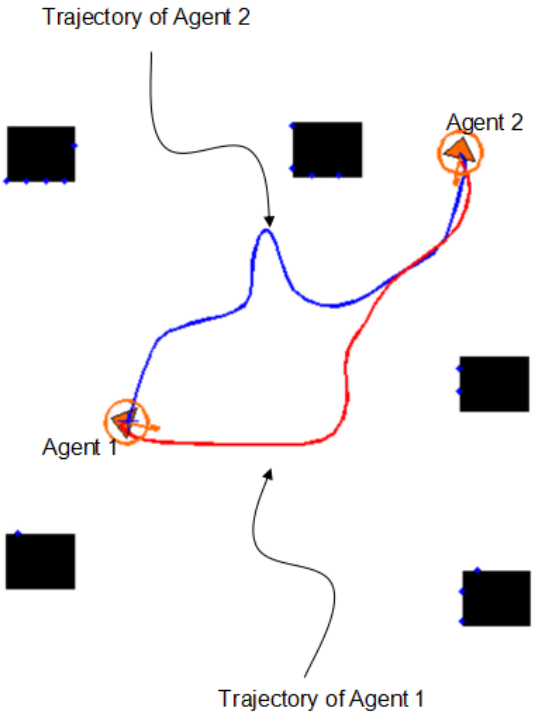


Figure 5.19: The trajectory of 2 agents exchanging their position in cluttered environment

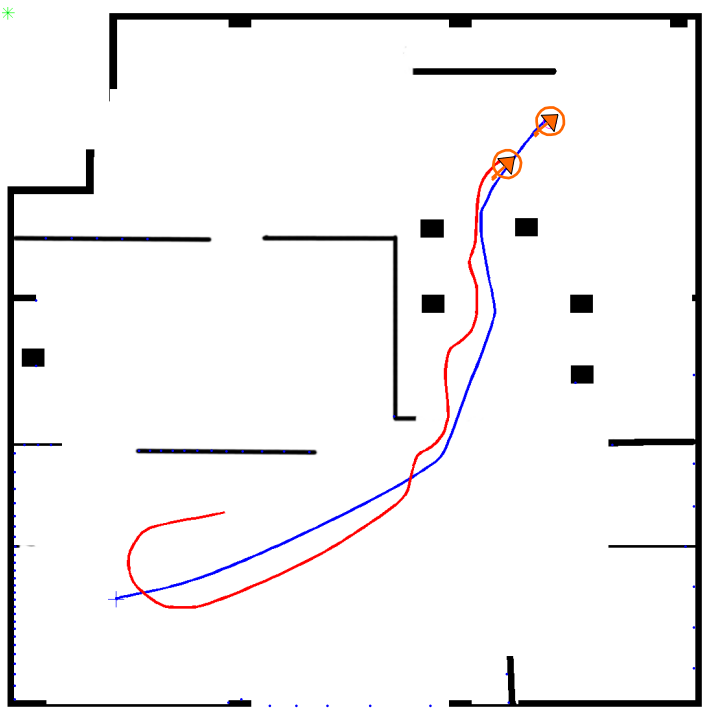


Figure 5.21: Leader follower moving formation in cluttered environment

In Figure 5.21, the trajectory of simulated 2-vehicle leader-follower formation flying through obstacles is illustrated. The follower (in red trajectory) is trying to stay at

the bottom-left corner of the leader (in blue trajectory) and that is the reason for it to reverse its direction at the beginning. The formation is broken down when they passed through the cluttered area and reformed immediately once given a chance.

5.5.2 Tandem Flocking

Simulation on a tandem flocking of multiple vehicles in an unknown forest environment is performed (also covered in author's paper [81]). The leader is responsible for finding ways to the final target while the followers walk behind each other one by one in a line. Each RUAV is equipped with an additional camera to detect and estimate its corresponding leader's relative position. The strategy of tandem flocking in cluttered environment goes as follows

1. The team leader flies towards its final target position through path planning and obstacle avoidance.

2. Each follower detects and estimates the position of its corresponding leader via the vision system.

3. A series of the corresponding leader's position are kept, and the instant target of the follower is selected among these positions. The instant target maintains a pre-specific distance to the latest known position of the leader which ensures cohesion and separation.

4. For each follower, flies towards the instant target while avoiding collision.

Since the camera comes with limited view angle, the heading of each follower always aims towards its own leader. Through out the process, the vehicles act independently and asynchronously from each other. A simulation of 6 vehicles flying in forest like environment is conducted. The maximum velocity of the leader is set as $v_{\max x} = v_{\max y} = v_{\max z} = 1\,\mathrm{m/s}$; whereas the maximum velocity of the follower is slightly higher as $v_{\max x} = v_{\max y} = v_{\max z} = 1.5\,\mathrm{m/s}$ so it could *catch up* the leading vehicle. In the global frame, the start positions of the leader is at $(46, 7)$ while its destination is at $(20, 44)$. The start position of the 5 followers are $(50, 7)$, $(55, 7)$, $(59, 7)$, $(63, 7)$ and $(67, 7)$ respectively as shown in Figure 5.22. Here UAV 1 is the leader and UAV

2–6 are the followers. The distance from the instant target to the latest known position of its corresponding leader is set as 3m. Figure 5.22 also illustrates the traces of the 6 vehicles during the flocking , where the red cubes represent the sensed obstacles and the numbers 1–6 denotes the initial position of each vehicle. Figure 5.23 shows the position and velocity trajectory of the 6 vehicles, the velocity references are well bounded considering the measurement units' limitation while the position references show the vehicles are away from each other avoiding mutual collision.

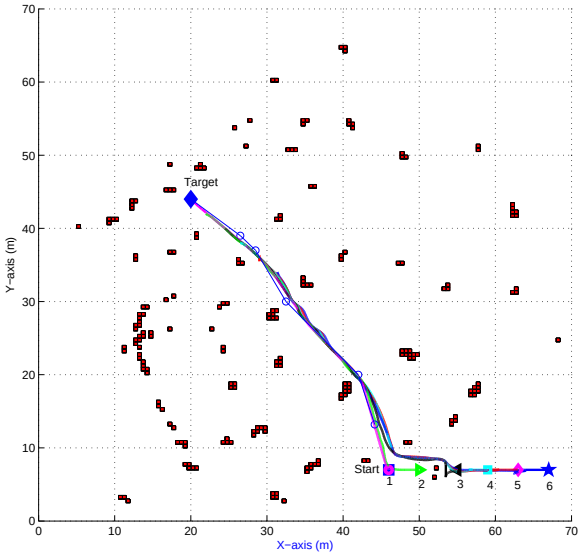
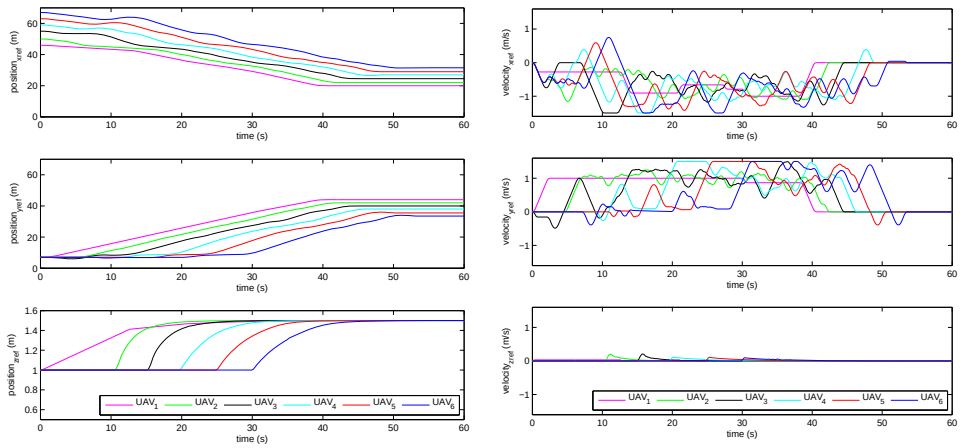


Figure 5.22: Trace of vehicles during flocking



(a) Position (b) Velocity

Figure 5.23: Reference trajectories during flocking

5.6 Conclusion

In this chapter, the author presents a guidance system that allows the vehicle to fly through obstacles. The proposed guidance system utilizes the structure of MGCS presented in Chapter 2. The overall planning problem is decomposed into a series of jerk limited TPBVP through the global planner. The verification and examination of the trajectory are based on a grid-based representation of the environment which facilitates $O(1)$ search of nearest obstacle distance. Moreover, to achieve the non-stop maneuver, an EPMPC strategy and the improved event triggered based trajectory switching method are proposed. In the event triggered method, a new cycle of planning is triggered only if the current trajectory is no longer safe, or the vehicle is about to stop. The proposed guidance system is tested under various environments with efficient and stable performance. It also handles dynamic obstacles by producing evading trajectory in realtime.

Further, the situation of multiple vehicle formation is discussed. Due to the usage of global planner which searches the connectivity information of the map, the proposed method helps to reduce the phenomenon related to local minimum and reciprocal dancing. Avoidance between agents and relaxed formation are demonstrated with simulation in the obstacle-strewn environment. Our approach does not require accurate neighbor information which makes it more straight forward for engineering implementation.

Finally, the guidance system can be improved by utilizing more efficient global planner, or TPBVP solver considering more optimization targets. Each submodule, the perception unit, the global and local planner can be modified without affecting the others. Thus it also serves as an excellent testbed for different SLAM, motion planning, cluster classification and vision algorithms.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, the author presented a successful approach to conduct trajectory leaded RUAV guidance and control in various scenarios. The major contribution lies at the implementation of the reference generation and guidance algorithms covered in Chapter 3 and 4, the guidance system in obstacle-rich environment covered in Chapter 5, and the MGCS. By solving convex quadratic programming, the B-spline based algorithm generates complicated trajectories through a time vector optimization, which are proved to be suitable for RUAVs with complex dynamics. Various performance and demonstrations are made possible by this algorithm, including the UAV calligraphy performance in 2014 Singapore Airshow and the multi-drone light show at Gardens by the Bay in 2016. Though still lacks the efficiency to be implemented on microprocessors like the flight controller of the micro quad-rotor in Figure 4.1, it is fast enough to be used on laptops for easy designing of the trajectory. On the other hand, in Chapter 4, online reference generation algorithms targeting inexpensive hardware are covered. The velocity reference generator is mainly used for semi-auto flying and utilized in applications like safe fly zone. The position reference generator is proposed for waypoint based guidance. With these algorithms, dynamically feasible reference could be generated in realtime for precise tracking control. The cascaded controller is then capable of giving accurate tracking performance in real flight experiment, which is crucial for tasks like aero filming and close inspection. These online reference generators are very

efficient, when used as command filter, a cycle can be finished in micro-second level even with low-power CPU like the Cortex M4F. Though the computational cost is increased when the TPBVP solver is involved (especially with direct bisection method), sub-millisecond performance can still be achieved with just a normal laptop.

With these trajectory generation tools, the author also proposed a guidance strategy to be used in the obstacle-rich environment. In Chapter 5, systematic introductions are given to each sub-module of the approach. The perception module translates the environmental information into the data structure that could be utilized by various planning and verification algorithms. The guidance module decomposes the task into a series of TPBVP. And the continuous movement can be achieved by an EPMPC strategy or the improved event triggered trajectory switching method. The proposed approach also considers the dynamic obstacles through rapid evasion trajectory searching. Finally, multiagent formation and flocking algorithm are built based on the proposed guidance system. To verify the generality and performance of our approach, experiments are conducted in environments that vary from forest to indoor office, with or without dynamic obstacles.

The author also implemented an MGCS for the experiment. An HCI translates the human input to waypoint based missions and uploads them to the vehicle wirelessly. The guidance system utilizes the above-mentioned methods to produce the dynamically feasible reference for the low-level controller. A control scheme is then proposed for general rotorcrafts. The MGCS is proven successful through various performances and competitions. Furthermore, it also serves as a testbed for different kinds of vehicles and algorithms in many other research projects.

6.2 Future Works

Possible improvements or extensions can be categorized into two major aspects. First, more sophisticated motion planning could be considered. Secondly, with inter-vehicle communication, a protocol based multiagent cooperation could be realized.

6.2.1 Towards Smarter Motion Planning

Through our study and experiment, the author found that the bottleneck of many robotic applications is a reliable online motion planning system, which can handle complex system dynamics and the capricious environment. Due to the difference in vehicle model, targeting scenario, a flexible, efficient and general solution is yet to be developed for motion planning problems. Conventional methods, which utilize dynamic programming, sampling search, numerical optimization or switching control, are prone to these problems. On the other hand, human beings are well suitable for a broad range of motion planning tasks, such as climbing, running and driving vehicles. The author thus proposes a possible structure that could be used by vehicles with different dynamics and targeting environments (see Figure 6.1). In this system, the key module will be the motion library, the motion optimizer, the controller and the reflection unit. The motion library stores a series of trajectories and is responsible for selecting the most appropriate one based on the measurements. The optimizer then utilizes the selected trajectory as an initial condition to execute an optimization process to fulfill various constraints based on the measurements. Then a controller, possibly dedicated to the selected trajectory from motion library would actuate the final system. Moreover, a reflection unit is introduced to handle events with fast dynamics.

Take ping-pong as an example. When the player learns a new move set such as forehand smash, a new nominal trajectory is added to the motion library. Then he might spend many hours training with this forehand smash movement to make sure he could hit an incoming ball at any angle, speed and spinning condition. This is equivalent as training of a neural network to perform the motion optimization. In the ping-pong example, the motion optimizer is responsible for adjusting the nominal smash movement so it could intercept the ball at the correct angle. The more training the player takes, the more input conditions the neural network could handle, and hence the better

the player performs the smash movement. Finally, if the player finds the ping-pong is coming towards his face with high speed, the reflection unit is turned on to avoid the ball. During this short period, the reflection unit takes over the body controller directly. To build this motion planning system, the nominal trajectory in motion library could be obtained through solving complex non-linear optimization problems offline. Then a neural network is established through simulated training to adjust the nominal trajectory efficiently. Finally, the reflection unit acts like an event-triggered control system to take over the actuator when necessary.

6.2.2 Towards Smarter Multi-vehicle System

To coordinate multiple vehicles, we have to solve

1. Construction of a synchronized coordinate system.
2. Establishment of a distributed traffic control protocol to enable efficient collision avoidance.
3. Implementation of a mission management protocol to facilitate online cooperation.

The first issue is to find a unified coordinate so that the vehicles could share their sensed information and target region. For example, if agent 0 finds a POI in its local coordinate system, it needs to broadcast the POI's location to other vehicles. However, since all other vehicles work in their local coordinate system, to correctly interpret this POI, they need to know their relative position to agent 0. Current available solutions are

1. Utilizing a third party measurement system such as GPS, Vicon, and UWB, to localize all the agents. These systems either have limitations on working environment or require the setup of multiple ground calibration unit.
2. Directly measuring neighbor agent's position using onboard sensors. This approach requires complex vision segmentation or classification algorithms which are still open problems.

Another possible approach is to utilize the features in SLAM algorithm to synchronize each vehicle's local coordinate. Assume that a set of common features can be captured

by two vehicles, a coordinate synchronization could be accomplished by communicating the feature's descriptor and its location in each vehicle's local coordinate. This approach requires no extra setup and is suitable for all environments. Further, since all agents are already running onboard SLAM, the feature comparison and mutual synchronization could be done without using too much onboard computational power.

The second issue is to establish an unobstructed traffic for the multiagent cooperation. If all the vehicles utilize a trajectory based guidance system, it is possible to employ a protocol to exchange the future trajectory among vehicles efficiently. With this information, planning could be more efficient. If each vehicle could be classified by several properties such as weight, mission type or remaining energy, it is possible to design a function to assign priority levels to each vehicle distributively. With other vehicles' future trajectories and priority levels, a distributed traffic control system could be designed to determine whether an agent should give way to its neighbors. A similar strategy is common among human drivers, whenever a driver sees an ambulance with alarm on, s/he then steers his vehicle away from the future path of the ambulance.

The third issue requires a discrete mission management system to handle various events during the cooperative process. For example, in a search-rescue mission, the system handles the behavior change between search and rescue.

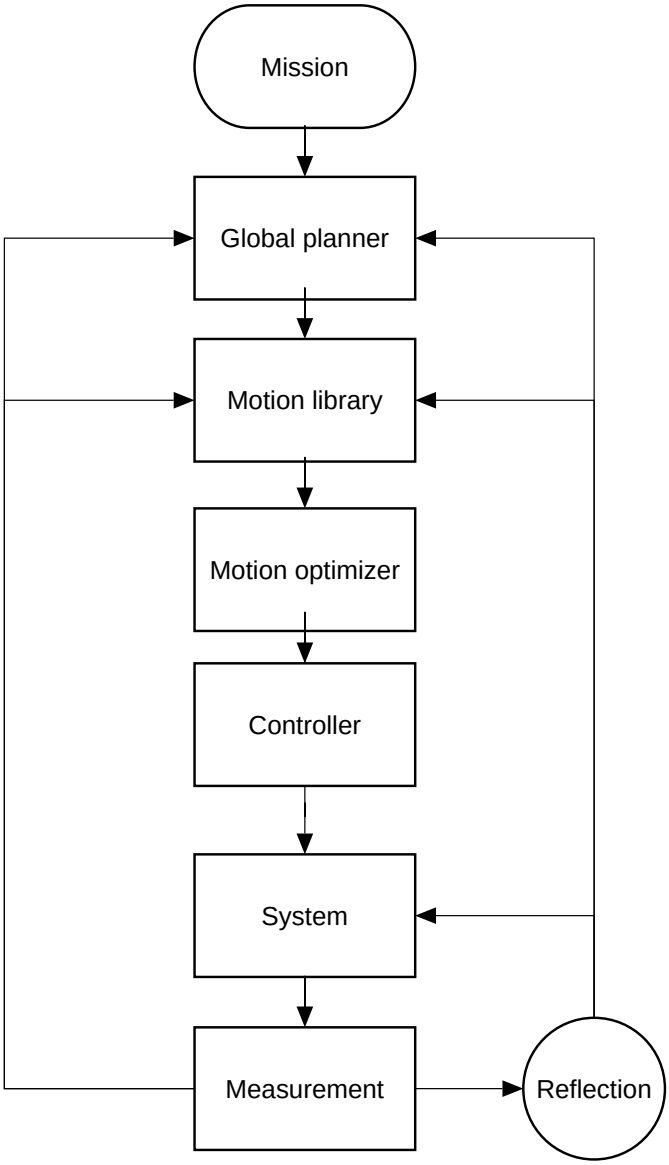


Figure 6.1: A possible motion planning structure for different systems

Bibliography

[1] M. S. Grewal, L. R. Weill, and A. P. AndrJay, *Global Positioning Systems, Inertial Navigation, and Integration*. John Wiley & Sons, Inc, Hoboken, New Jersey, United States, 2007.

[2] J. A. Farrell, *Aided Navigation: GPS with High Rate Sensors.* McGraw-Hill, Inc, 2008.

[3] C. S. Draper, W. Wrigley, G. Hoag, R. H. Battin, E. Miller, A. Koso, A. L. Hopkins, and W. E. Vander Velde, "Apollo guidance and navigation," tech. rep., Instrumentation Laboratory, Massachusetts Institute of Technology, Cambridge, MA, United States, 1965.

[4] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," tech. rep., The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, United States, 1992.

[5] E. C. Stewart, "Application of statistical theory to beam-rider guidance in the presence of noise II: modified wiener filter theory," tech. rep., Ames Aeronautical Lab, National Advisory Committee for Aeronautics, Moffett Field, CA, United States, 1955.

[6] V. Rajasekhar and A. G. Sreenatha, "Fuzzy logic implementation of proportional navigation guidance," *Acta Astronautica*, vol. 46, no. 1, pp. 17–24, Jan 2000.

[7] C. L. Lin and H. W. Su., "Intelligent control theory in guidance and control system design: an overview," *Proceedings of the National Science Council, Republic of China. Part A*, vol. 24, pp. 15–30, 2000.

[8] S. Park, J. Deyst, and J. How, "A new nonlinear guidance logic for trajectory tracking," in Proceedings of the *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Rhode Island, United States, Aug 2004.

[9] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard, "Vector field path following for miniature air vehicles," *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 519–529, Jun 2007.

[10] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in Proceedings of the *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, pp. 2520–2525, May 2011.

[11] M. Hehn and R. D'Andrea, "Real-Time Trajectory Generation for Quadrocopters," *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 877–892, Aug 2015.

[12] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," in Proceedings of the *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, pp. 3480–3486, Nov 2013.

[13] S. K. Phang, S. Lai, F. Wang, M. Lan, and B. M. Chen, "Systems design and implementation with jerk-optimized trajectory generation for UAV calligraphy," *Mechatronics*, vol. 30, pp. 65–75, September 2015.

[14] G. M. Hoffmann, S. L. Waslander and C. J. Tomlin, "Quadrotor helicopter trajectory tracking control," in Proceedings of the *AIAA Guidance, Navigation and Control Conference and Exhibit*, Hawaii, United States, Aug 2008.

[15] I. D. Cowling, O. A. Yakimenko, J. F. Whidborne, and A. K. Cooke, "A prototype of an autonomous controller for a quadrotor UAV," in Proceedings of the *2007 European Control Conference*, Kos, Greek, pp. 4001–4008, July 2007.

[16] Y. Bouktir, M. Haddad, and T. Chettibi, "Trajectory planning for a quadrotor helicopter," in Proceedings of the *16th Mediterranean Conference on Control and Automation*, Ajaccio, France, pp. 1258–1263, June 2008.

[17] T. J. Koo and S. Sastry, "Differential flatness based full authority helicopter control design," in Proceedings of the *38th IEEE Conference on Decision and Control*, Phoenix, United States, vol. 2, pp. 1982–1987, Dec 1999.

[18] A. Bry, C. Richter and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in Proceedings of the *16th International Symposium on Robotics Research*, Singapore, pp. 649–666, Dec 2013.

[19] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in Proceedings of the *2016 IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, pp. 1476–1483, May 2016.

[20] W. V. Loock, G. Pipeleers, and J. Swevers, "Time-optimal quadrotor flight," in Proceedings of the *2013 European Control Conference*, Zurich, Switzerland, pp. 1788–1792, July 2013.

[21] M. Hehn, R. Ritz, and R. D'Andrea, "Performance benchmarking of quadrotor systems using time-optimal control," *Autonomous Robots*, vol. 33, no. 1, pp. 69–88, Mar 2012.

[22] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 1, pp. 23–32, Feb 1992.

[23] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, Jun 1991.

[24] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in Proceedings of the *2008 IEEE International Conference on Robotics and Automation*, Pasadena, United States, pp. 1928–1935, May 2008.

[25] D. Delling, P. Sanders, D. Schultes, and D. Wagner, " Engineering Route Planning Algorithms," *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*, pp. 117–139, Springer, Berlin Heidelberg, Jun 2009.

[26] S. M. Lavalle, J. J. Kuffner, and Jr., "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, A K Peters/CRC Press, Apr 2000.

[27] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, no. 2, pp. 100–107, July 1968.

[28] A. Stentz, "Optimal and Efficient Path Planning for Partially Known Environments," *IEEE Transactions on Robotics*, vol. 21, no.3, pp. 354–363, June 2005.

[29] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," in Proceedings of the *1994 IEEE International Conference on Robotics and Automation*, San Diego, United States, pp. 203–220, May 2005.

[30] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, May 2011.

[31] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in Proceedings of the *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, pp. 1478–1483, May 2011.

[32] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in Proceedings of the *2015 IEEE International Conference on Robotics and Automation*, Seattle, United States, pp. 3067–3074, May 2015.

[33] S. Lai, K. Wang, H. Qin, J. Q. Cui, and B. M. Chen, "A robust online path planning approach in cluttered environments for micro rotorcraft drones," *Control Theory and Technology*, vol. 14, no. 1, pp. 83–96, Feb 2016.

[34] J. Q. Cui, S. Lai, X. Dong, and B. M. Chen, "Autonomous navigation of uav in foliage environment," *Journal of Intelligent & Robotic Systems*, pp. 1–18, Oct 2015.

[35] A. Datta, B. Roget, D. Griffiths, G. Pugliese, J. Sitaraman, J. Bao, L. Liu, and O. Gamard, "Design of a martian autonomous rotary-wing vehicle," *Journal of aircraft*, vol. 40, no. 3, pp. 461–472, May–Jun 2003.

[36] "Pd-100 black hornet prs." http://www.proxdynamics.com/pcroducts/pd-100-black-hornet-prs, 2012.

[37] F. Wang, P. Liu, S. Zhao, B. M. Chen, S. K. Phang, S. Lai, T. Pang, B. Wang, C. Cai, and T. H. Lee, "Development of an unmanned helicopter for vertical replenishment," *Unmanned Systems*, vol. 03, no. 01, pp. 63–87, Jan 2015.

[38] M. W. Mueller, M. Hehn, and R. D'Andrea, "A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation," *IEEE Transactions on Robotics*, vol. 31, pp. 1294–1310, Dec 2015.

[39] B. Yu, X. Dong, Z. Shi, and Y. Zhong, "Formation control for quadrotor swarm systems: Algorithms and experiments," in Proceedings of the *32nd Chinese Control Conference*, Xi'an, China, pp. 7099–7104, Jul 2013.

[40] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in Proceedings of the *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, pp. 1917–1922, Oct 2012.

[41] J. Chen, K. Su, and S. Shen, "Real-time safe trajectory generation for quadrotor flight in cluttered environments," in Proceedings of the *2015 IEEE International Conference on Robotics and Biomimetics*, Zhuhai, China, pp. 1678–1685, Dec 2015.

[42] M. Harada, H. Nagata, J. Simond, and K. Bollino, "Optimal trajectory generation and tracking control of a single coaxial rotor UAV," in Proceedings of the *AIAA Guidance, Navigation, and Control Conference*, Boston, United States, Aug 2013.

[43] QGC development team, "Qgroundcontrol opensource ground control station project," website http://qgroundcontrol.org/.

[44] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 6, pp. 707–717, Dec 1992.

[45] K. Peng, G. Cai, B. M. Chen, M. Dong, K. Y. Lum, and T. H. Lee, "Design and implementation of an autonomous flight control law for a UAV helicopter," *Automatica*, vol. 45, no. 10, pp. 2333–2338, 2009.

[46] F. Wang, K. Wang, S. Lai, S. K. Phang, B. M. Chen, and T. H. Lee, "An efficient UAV navigation solution for confined but partially known indoor environments," in Proceedings of the *11th IEEE International Conference on Control Automation*, Taichung, Taiwan, China, pp. 1351–1356, Jun 2014.

[47] F. Lin, K. Z. Y. Ang, F. Wang, B. M. Chen, T. H. Lee, B. Yang, M. Dong, X. Dong, J. Cui, S. K. Phang, B. Wang, D. Luo, K. Peng, G. Cai, S. Zhao, M. Yin, and K. Li, "Development of an unmanned coaxial rotorcraft for the darpa uavforge challenge," *Unmanned Systems*, vol. 01, no. 02, pp. 211–245, 2013.

[48] K. Wang, Y. Ke and B. M. Chen, "Development of autonomous hybrid UAV U-Lion with VTOL and cruise flying capabilities," Proceedings of the *2016 IEEE International Conference on Advanced Intelligent Mechatronics,* Banff, AB, pp. 1053–1060, 2016.

[49] T. H.Lee, B. M. Chen and V. Venkataramanan, *Hard Disk Drive Servo Systems*, 2nd Edition, Springer, New York, 2006.

[50] B. M. Chen, *Robust and H∞ Control*, Springer, New York, 2000.

[51] T. Flash and N. Hogans, "The coordination of arm movements: An experimentally confirmed mathematical model," *Journal of neuroscience*, vol. 5, pp. 1688–1703, 1985.

[52] H. Kano, H. Nakata and C. F. Martin, "Optimal curve fitting and smoothing using normalized uniform B-splines: a tool for studying complex systems", *Applied Mathematics and Computation*, vol. 159, no. 1, pp. 96-128, 2005.

[53] H. Kano, H. Fujioka and C. F. Martin, “Optimal smoothing and interpolating splines with constraints”, *Applied Mathematics and Computation*, vol. 218, no. 5, pp. 1831-1844, 2011

[54] H. Fujioka and H. Kano, “ Control theoretic B-spline smoothing with constraints on derivatives”, *IEEE 52nd Annual Conference on Decision and Control*, Firenze, Italy, 2013.

[55] K. Takayama and H. Kano, “A new approach to synthesizing free motions of robotic manipulators based on the concept of unit motion”, *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 3, pp. 453-463, 1995.

[56] “Matlab Mathworks, Inc.” http://www.mathworks.com/

[57] “IBM ILOG CPLEX Optimizer IBM, Inc.” https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[58] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer Series in Operations Research and Finicial Engineering, 2nd Edition, Springer, 2006.

[59] R. Haschke, E. Weitnauer, H. Ritter, “On-line planning of time-optimal, jerk-limited trajectories,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, pp. 3248 – 3253, 2008.

[60] C. de Boor, *A Practical Guide to Splines*, Springer, New York, 1978.

[61] C. K. Shene, Lecture notes on *Introduction to computing with Geometry*, https://www.cs.mtu.edu/ shene/COURSES/cs3621/NOTES/, Michigan Technological University, 2011.

[62] G. A. Borges and M. J. Aldon, “A split-and-merge segmentation algorithm for line extraction in 2d range images, Proceedings of the *15th International Conference on Pattern Recognition*, Barcelona, Spain, pp. 441-444, 2000.

[63] F. Lange and A. Albu-Schffer, ”Path-Accurate online trajectory generation for jerk-limited industrial robots,” *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 82–89, Jan 2016.

[64] L. Meier, P. Tanskanen, F. Fraundorfer and M. Pollefeys, “PIXHAWK: A system for autonomous flight using onboard computer vision,” *2011 IEEE International Conference on Robotics and Automation,* Shanghai, pp. 2992–2997, 2011.

[65] M. Rubagotti and A. Ferrara, “Second order sliding mode control of a perturbed double integrator with state constraints,” in Proceedings of the *2010 American Control Conference*, Baltimore, United States, pp. 985–990, June 2010.

[66] T. Kroger and F. M. Wahl, “Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events,” *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, Feb 2010.

[67] F. Ramos, M. Gajamohan, N. Huebel, and R. DAndrea, “Time-Optimal Online Trajectory Generator for Robotic Manipulators,” Institute for Dynamics Systems and Control, ETH, Zurich, Feb. 2013.

[68] T. Kroger, “Opening the door to new sensor-based robot applications – The Reflexxes Motion Libraries,” in Proceedings of *2011 IEEE International Conference on Robotics and Automation*, Shanghai, pp. 1–4, 2011.

[69] P. Vlez, N. Certad and E. Ruiz, “Trajectory Generation and Tracking Using the AR.Drone 2.0 Quadcopter UAV,” in Proceedings of *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics*, Uberlandia, pp. 73–78, 2015.

[70] C. L. Bottasso, D. Leonello and B. Savini, “Path Planning for Autonomous Vehicles by Trajectory Smoothing Using Motion Primitives”, *IEEE Transactions on Control Systems Technology*, vol. 16, no. 6, pp. 1152–1168, Nov 2008.

[71] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments,” *The International Journal of Robotics Research*, vol. 31 no. 5 pp. 647–663, Apr 2012.

[72] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, *RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments*, pp. 477–491, Springer, Berlin Heidelberg, 2014.

[73] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb 2007.

[74] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in Proceedings of *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, Kyoto, Japan, pp. 155–160, Nov 2011.

[75] P. F. Felzenszwalb and D. P.Huttenlocher, "Distance transforms of sampled functions," *Theory of Computing*, vol. 8, no. 19, pp. 415–428, 2012.

[76] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in Proceedings of the *1987 European Association for Computer Graphics*, Amsterdam, Netherlands, pp. 3–10, Aug 1987.

[77] J. J. Park, C. Johnson, B. Kuipers,"Robot navigation with model predictive equilibrium point control," Proceedings of the *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, pp. 4945–4952, 2012.

[78] T. Kroger, "Online Trajectory Generation: Straight-Line Trajectories," in *IEEE Transactions on Robotics,* vol. 27, no. 5, pp. 1010–1016, 2011.

[79] J. Cui, S. K. Phang, K. Ang, F. Wang, X. Dong, Y, Ke, S. Lai, K. Li, X. Li, F. Lin, J. Lin, P. Liu, T. Pang, B. Wang, K. Wang, Z. Yang and B. M. Chen, "Drones for coorperative search and rescue in post-disaster situation," Proceedings of the *Cybernetics and Intelligent Systems, Robotics, Automation and Mechatronics*, Angkor Wat, Cambodia, pp. 167–174, Aug 2015.

[80] S. Lai, J. Cui and B. M. Chen, "Relaxed formation in cluttered environments using rotor-craft UAV," Proceedings of the *2015 10th Asian Control Conference*, Kota Kinabalu, pp. 1–6, May 2015.

[81] F. Liao, J.Wang, R. Teo, Y. Hu, S. Lai, J. Cui, F. Lin, "Vision-based Autonomous Flocking Of Uavs In Unknown Forest Environment," Proceedings of the *12th International Conference On Control & Automation*, Kathmandu, Nepal, pp. 892–897, Jun 2016.

Chapter 7

List of Author's Publication

Journals

1. S. K. Phang, S. Lai, F. Wang, M. Lan and B. M. Chen, "Systems design and implementation with jerk-optimized trajectory generation for UAV calligraphy," *Mechatronics*, vol. 30, pp. 65–75, Jun 2015.

2. J. Cui, S. K. Phang, K. Ang, F. Wang, X. Dong, Y. Ke, S. Lai, K. Li, X. Li, J. Lin, P. Liu, T. Pang, K. Wang, Z. Yang, F. Lin and B. M. Chen, "Search and rescue using multiple drones in post-disaster situation", *Unmanned Systems*, vol. 4, no. 1, pp. 83–96, Feb 2016.

3. F. Wang, P. Liu, S. Zhao, B. M. Chen, S. K. Phang, S. Lai, T. Pang, B. Wang, C. Cai and T. H. Lee, "Development of an unmanned helicopter for vertical replenishment," *Unmanned Systems*, vol, 3, no. 1, pp. 63–87, Jan 2015

4. S. Lai, K. Wang, H. Qin, J. Q. Cui and B. M. Chen, "A robust online path planning approach in cluttered environments for micro rotorcraft drones," *Journal of Control Theory and Technology*, vol. 14, no. 1, pp. 83–96, Feb 2016.

5. M. Zhu, S. Lai, R. Boucher, B. M. Chen, X. Cheng and W. Kang, "Minimum Time Trajectory for Helicopter UAVs: Computation and Flight Test," *Applied Mathematical Sciences*, vol. 7, no. 130, pp. 6475–6487, 2013

6. J. Q. Cui, S. Lai, X. Dong and B. M. Chen, "Autonomous navigation of UAV in foliage environment," *Journal of Intelligent and Robotic Systems*, in press.

Conferences

1. S. K. Phang, Fei Wang, Kangli Wang, Shupeng Lai and Ben M. Chen, "An effective method for autonomous localization and navigation in unknown indoor environment using MAV," Proceedings of the *2015 International Micro Air Vehicle Conference and Competition*, Aachen, Germany, Sep 2015.

2. J. Cui, S. K. Phang, K. Ang, F. Wang, X. Dong, Y, Ke, S. Lai, K. Li, X. Li, F. Lin, J. Lin, P. Liu, T. Pang, B. Wang, K. Wang, Z. Yang and B. M. Chen, "Drones for coorperative search and rescue in post-disaster situation," Proceedings of the *Cybernetics and Intelligent Systems, Robotics, Automation and Mechatronics*, Angkor Wat, Cambodia, pp. 167–174, Aug 2015.

3. S. K. Phang, J. Cui, K. Ang, F. Wang, X. Dong, Y. Ke, S. Lai, K. Li, X. Li, F. Lin, J. Lin, P. Liu, T. Pang, B. Wang, K. Wang, Z. Yang and B. M. Chen, "Urban post-disaster search and rescue solutions with unmanned aircraft systems," Proceedings of the *2015 International Conference on Electronics, Information and Communication*, Singapore, pp. 91–92, Jan 2015.

4. K. Li, R. Huang, S. K. Phang, S. Lai, F. Wang, P. Tan, B. M. Chen and T. H. Lee, "Off-board vision odometry and control of an ultralight quadrotor MAV," Proceedings of the *2014 International Micro Air Vehicle Conference and Competition*, Delft, The Netherlands, pp. 50–57, Aug 2014.

5. F. Wang, P. Liu, S. Zhao, B. M. Chen, S. K. Phang, S. Lai and T. H. Lee, "Guidance, navigation and control of an unmanned helicopter for automatic cargo transportation," Proceedings of the *2014 Chinese Control Conference*, Nanjing, China, pp. 1013–1020, Jul 2014.

6. F. Wang, K. Wang, S. Lai, S. K. Phang, B. M. Chen and T. H. Lee, "An efficient UAV navigation solution for confined but partially known indoor environments," Proceedings of the *11th IEEE International Conference on Control & Automation*, Taichung, Taiwan, pp. 1351–1356, Jun 2014.

7. S. K. Phang, S. Lai, F. Wang, M. Lan and B. M. Chen, "UAV calligraphy," Proceedings of *the 11th IEEE International Conference on Control & Automation*,

Taichung, Taiwan, pp. 422–428, Jun 2014.

8. J. Cui, S. Lai, X. Dong, P. Liu, B. M. Chen and T. H. Lee, "Autonomous navigation of UAV in forest," Proceedings of the *2014 International Conference on Unmanned Aircraft Systems*, Orlando, FL, pp. 726–733, May 2014.

9. S. Lai, J. Cui and B. M. Chen, "Relaxed formation in cluttered environments using rotor-craft UAV," Proceedings of the *2015 10th Asian Control Conference*, Kota Kinabalu, pp. 1–6, May 2015.

10. K. Wang, S. Lai, J. Cui, Y. Ke and B. M. Chen, "Navigation of micro air vehicles with inconsistent GPS guidance and online path planning," Proceedings of the *2015 International Micro Air Vehicles Conference, Session 2A,* Aachen, Germany, September 2015.

11. K. Li, R. Huang, S. K. Phang, S. Lai, F. Wang, P. Tan, B. M. Chen and T. H. Lee, "Vision-based autonomous control of an ultralight quadrotor MAV," Proceedings of the *2014 International Micro Air Vehicle Conference and Competition*, Delft, the Netherlands, pp. 50–57, Aug 2014.

12. F. Liao, J.Wang, R. Teo, Y. Hu, S. Lai, J. Cui, F. Lin, "Vision-based Autonomous Flocking Of Uavs In Unknown Forest Environment," Proceedings of the *12th International Conference On Control & Automation*, Kathmandu, Nepal, pp. 892–897, Jun 2016.

13. F. Liao, S. Lai, Y. Hu, J. Cui, J.Wang, R. Teo, F. Lin, "3D Motion Planning Of Uavs In Gps-denied Unknown Forest Environment," Proceedings of the *2016 Intelligent Vehicles Symposium*, Gothenburg, Sweden, pp. 246–251, Jun 2016.

14. S. Lai, K. Wang, K. Li, B. M. Chen, "Path planning of rotorcrafts in unknown environment," Proceedings of the *35th Chinese Control Conference*, Chengdu, China, pp. 10900–10905, Jul 2016.

15. M. Lan, S. Lai, Y. Bi, H. Qin, J. Li, F. Lin, B. M. Chen, "BIT*-based Path Planning for Micro Aerial Vehicles," Proceedings of the *42nd Annual Conference of IEEE Industrial Electronics Society*, Florence, Italy, 2016, in press.