Design and Synthesis of a Hierarchical Hybrid Controller for Quadrotor Navigation

by

Xiaotong Zhang

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2015 by the
Graduate Supervisory Committee:

Georgios Fainekos, Chair
Heni Ben Amor
Aviral Shrivastava

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

There has been exciting progress in the area of Unmanned Aerial Vehicles (UAV) in the last decade, especially for quadrotors due to their nature of easy manipulation and simple structure. A lot of research has been done on achieving autonomous and robust control for quadrotors. Recently researchers have been utilizing linear temporal logic as mission specification language for robot motion planning due to its expressiveness and scalability. Several algorithms have been proposed to achieve autonomous temporal logic planning. Also, several frameworks are designed to compose those discrete planners and continuous controllers to make sure the actual trajectory also satisfies the mission specification. However, most of these works use first-order kinematic models which are not accurate when quadrotors fly at high speed and cannot fully utilize the potential of quadrotors.

This thesis work describes a new design for a hierarchical hybrid controller that is based on a dynamic model and seeks to achieve better performance in terms of speed and accuracy compared with some previous works. Furthermore, the proposed hierarchical controller is making progress towards guaranteed satisfaction of mission specification expressed in Linear Temporal Logic for dynamic systems. An event-driven receding horizon planner is also utilized that aims at distributed and decentralized planning for large-scale navigation scenarios. The benefits of this approach will be demonstrated using simulations results.

DEDICATION

To My Parents

ACKNOWLEDGMENTS

I will like to express my deepest appreciation and gratitude to my advisor and committee chair, Dr. Georgios Fainekos, for giving me an opportunity to work on this exciting research topic and for providing invaluable amount of ideas, mentoring, support and patience overseeing my research. I would like to thank the committee members, Dr. Heni Ben Amor, Dr. Aviral Shrivastava and Dr. Yinong Chen for agreeing to be a part of my committee and providing ideas and feedback. I would like to thank my lab mates Adel Donkachi, Kangjin Kim, Bardh Hoxha, and Wei Wei for providing support. Finally, I would like to thank my parents for always being supportive and encouraging me with their best wishes, without whom this journey would have been very difficult.

This thesis work has been partially supported by NSF CPS 1446730. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Motivation and Challenges

Quadrotors have gained more and more attention in civilian applications as their manufacturing and maintenance cost went down to acceptable levels in the past few years. Compared with manned aerial vehicles, quadrotors have unique advantages in many areas. To list a few, quadrotors are widely used in:

- Aerial Surveillance: These light-weight, easy-maneuvered, and low-cost flying devices are almost the ideal tools for carrying missions in dangerous areas that are hard for human to reach. For example, the DJI Inspire quadrotor was recently used to record volcano eruptions in Vanuatu, and got by far the most close-up video of an erupting volcano [1]. Other aerial vehicles have never been able to get this close to a live volcano.

- Commercial Transportation: Traditional manned heavy helicopters are only used for high-value commercial applications due to the high cost, complex maintenance, and long-term pilot training. Quadrotors greatly simplified this thanks to its simplified structure and autonomous control software. Amazon and DHL have launched projects seeking to use quadrotors to deliver packages [2].

- Search and Patrol: Such operations are time-consuming and often dangerous for manned aerial vehicle. Thanks to their light-weighted structure, quadrotors could carry more fuel and equipment to work for longer time and be more efficient. The

Canadian mounties have announced that a police quadrotor equipped with thermal sensors, for the first time, successfully saved a driver's life in a remote woody area [3].

However, in most of such applications, quadrotors are controlled remotely by human pilots, which significantly limits its applicability and utilization in diverse applications. Flying quadrotors controlled by unprofessional pilots have caused a lot of controversial issues regarding privacy and safety [4]. Many privacy places, even the White House, were reported to have found unidentified quadrotor's intrusion. Also, many pilots of passenger airlines have reported seeing quadrotors near planes landing routes. For these reasons, a robust and accurate controller is much more desirable and essential for quadrotors compared with other robot applications.

A lot of research has been done in designing automated controllers that are robust and accurate enough to replace the remote pilots and accomplish the desired missions autonomously. In my opinion, the challenges in designing such controllers can be summarized in the following three aspects:

1) Finding a language that could formally define complex mission specifications and a planner which could automatically and efficiently generate plans that satisfy these specifications.

2) A continuous controller for the quadrotor's non-linear high-order dynamics that is able to follow reference commands from the higher level planner.

3) An interface that composes the high-level discrete planer and the low-level continuous controller, such that the behavior of the quadrotor satisfies the high-level specifications.

1.2 Contributions

The main contribution of this thesis is the design and implementation of a new kind of hierarchical hybrid controller for quadrotors. Compared with the previous approaches that used a first-order kinematic model such as [5] and [6], it is designed to accommodate a high-order dynamic model of the quadrotor's behavior in order to achieve better performance in terms of velocity and safety. Linear Temporal Logic (LTL) is used as the mission specification language and a LTL receding horizon planning method [7] is adopted to generate discrete plans that satisfy the mission specification. Compared with global planning, receding horizon method does not rely on global data and thus is more efficient for large-scale distributed path planning. To compose the high-level discrete plans and actual physical system of a quadrotor, the approximate simulation relations method [8] is used heuristically in order to improve the chances that the actual behavior of the quadrotor also satisfies the high-level specification. In later sections, the benefits of the proposed control synthesis framework will be shown by simulation results. A local planer is also proposed to fly through areas with higher resolution.

A software-in-the-loop simulator for quadrotor is also developed to test and verify the controller's design. It is based on the quadrotor model in Robotic toolbox [9] in MATLAB/Simulink. It receives commands from the controller and updates the states of the quadrotor in Simulink. It communicates through UDP with the controller.

1.3 Structure of the Thesis

This thesis introduces a new design of a hierarchical hybrid controller for quadrotors and its benefits compared with some previous approaches. This thesis is structured according to the following outline:

Chapter 2: Reviews some fundamental background knowledge. Several models and notations which are used later in this thesis are also introduced.

Chapter 3: Reviews previous works that this thesis is fundamentally based on and discusses their relationship to this work. A review of related works is also included.

Chapter 4: Describes the problems to be solved in this thesis.

Chapter 5: Describes the structure of the hierarchical hybrid controller as well as the roles of each component and how they cooperate with each other to full fill the mission specification.

Chapter 6: Presents some simulation results of this design and it compares the proposed controller design with previous research to show its benefits.

Chapter 7: Gives a conclusion and several possible future directions in this area of research.

CHAPTER 2

BACKGROUND

This chapter presents some background concepts, definitions, and notations that this thesis is built upon. Notations used in this chapter will be remain consistent later in this thesis. It will be helpful to read through this chapter in order to better understand the later contents.

2.1 Quadrotor Model and Dynamics

Figure 1 shows a classic model of a quadrotor. The coordinate frame in the picture is defined as the body frame $\mathcal{B}$, which is fixed to the quadrotor, and its three axis are hereby denoted by $x_{\mathcal{B}}$, $y_{\mathcal{B}}$, and $z_{\mathcal{B}}$ respectively. In contrast, the world frame $\mathcal{W}$ is fixed to the ground and its three axis are denoted by $x_{\mathcal{W}}$, $y_{\mathcal{W}}$, and $z_{\mathcal{W}}$.

A quadrotor has 6 degrees of freedom (DOF) which are its movement along the $x_{\mathcal{B}}$, $y_{\mathcal{B}}$, $z_{\mathcal{B}}$ and its rotation along $x_{\mathcal{B}}$, $y_{\mathcal{B}}$, and $z_{\mathcal{B}}$ (pitch, roll, and yaw). The only inputs to the system are the rotation velocities of the four rotors (ω_1, ω_2, ω_3, ω_4) which provide lift and maneuverability.

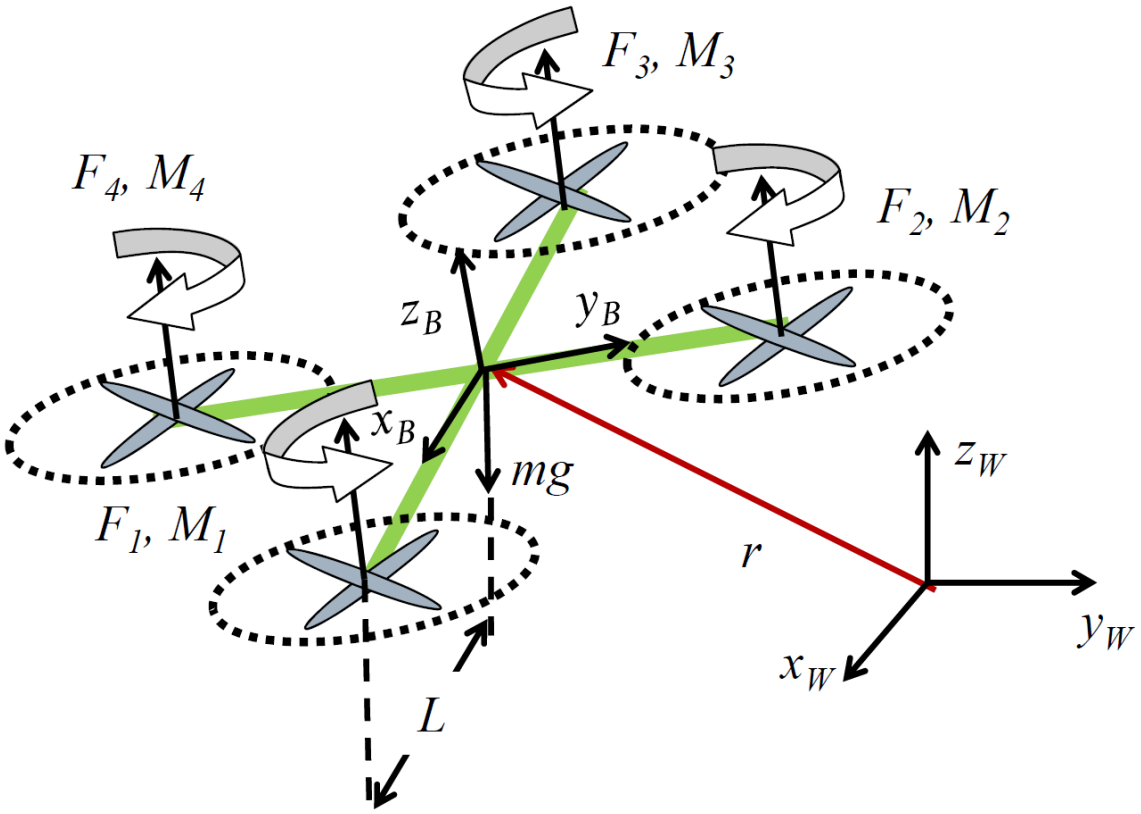


Figure 1 Typical Model of a Quadrotor

According to classic control theory, having equal or more control inputs than outputs is a necessary condition to achieve fully decoupled control. In a quadrotor there are six degrees of freedom while it only has four inputs. This means some maneuvers are not controllable or coupled with other maneuvers. The rest part of this section will introduce the dynamics and basic control methods for a quadrotor.

A rotor i with angular velocity ω_i can generate corresponding thrust F_i and moment M_i that are approximately described by the following equations:

- $F_i = k_F {\omega_i}^2$
- $M_i = k_M {\omega_i}^2$

where k_F and k_M are constant parameters that are determined by the rotor's size and shape. The generated thrust vectors of each of the four rotors are always parallel to $z_{\mathcal{B}}$, and the direction of the moment is determined by the rotation direction. In Figure 1, rotors 1 & 3 rotate counter-clockwise and generate clockwise moments, while rotors 2 & 4 rotate clockwise and thus generate counter-clockwise moments. A quadrotor can achieve complex maneuvers by changing the angular velocity of the four rotors.

A quadrotor controls its attitude by the following equation:

$$I\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}L(F_2 - F_4)\\ L(F_3 - F_1)\\ M_1 - M_2 + M_3 - M_4\end{bmatrix} - \begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \times I\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \qquad (1)$$

where ϕ, θ, ψ are quadrotor's roll, pitch, and yaw angles, L is the distance between the center of a rotor to the center of gravity, I is the quadrotor's inertia matrix that is approximately a diagonal matrix due to its symmetric structure:

$$I = \begin{bmatrix}I_{xx} & 0 & 0\\ 0 & I_{yy} & 0\\ 0 & 0 & I_{zz}\end{bmatrix}$$

A quadrotor can control its movement by the following equation:

$$m\ddot{r} = \begin{bmatrix}0\\ 0\\ -mg\end{bmatrix} + R\begin{bmatrix}0\\ 0\\ \Sigma F_i\end{bmatrix} \qquad (2)$$

where m is the quadrotor's mass, $r = [r_x \quad r_y \quad r_z]^T$ is the position matrix, and R is the transition matrix from $\mathcal{B}$ to $\mathcal{W}$:

$$\begin{bmatrix} cos\psi\ cos\theta - sin\phi\ sin\psi\ sin\theta & -cos\phi\ sin\psi & cos\psi\ sin\theta + cos\theta\ sin\phi\ sin\psi \\ cos\theta\ sin\psi + cos\psi\ sin\phi\ sin\theta & cos\phi\ cos\psi & sin\psi\ sin\theta - cos\psi\ cos\theta\ sin\phi \\ -cos\phi\ sin\theta & sin\phi & cos\phi\ cos\theta \end{bmatrix}$$

From the above analysis, it's obvious that the value of $F_2 - F_4$ has effect on roll angle ϕ, $F_3 - F_1$ has effect on pitch angle θ, and $M_1 - M_2 + M_3 - M_4$ has effect on the yaw angle ψ. Changing the positon can be achieved by changing orientation angles and total thrusts. To make it convenient for future analysis, the following mapping is applied:

$$u_1 = k_F(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)$$

$$u_2 = k_F(\omega_2^2 - \omega_4^2)$$

$$u_3 = k_F(\omega_1^2 - \omega_3^2)$$

$$u_4 = k_M(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)$$

Then, the former equations can be changed to the following equations:

$$\Sigma_q = \begin{cases} \ddot{r}_x = -(cos\phi\ sin\theta\ cos\psi + sin\theta\ sin\phi) \cdot \frac{u_1}{m} \\ \ddot{r}_y = -(cos\phi\ sin\theta\ sin\psi - sin\phi\ cos\psi) \cdot \frac{u_1}{m} \\ \ddot{r}_z = g - (cos\varphi\ cos\theta) \cdot \frac{u_1}{m} \\ \ddot{\theta} = \dot{\phi}\dot{\psi}\left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right) + \frac{L}{I_{xx}}u_2 \\ \ddot{\varphi} = \dot{\theta}\dot{\psi}\left(\frac{I_{yy} - I_{zz}}{I_{xx}}\right) + \frac{L}{I_{xx}}u_2 \\ \ddot{\psi} = \dot{\phi}\dot{\theta}\left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right) + \frac{L}{I_{yy}}u_3 \end{cases} \quad (3)$$

This model will be referred to as Σ_q for the rest of this thesis. As it can be seen, it is a non-linear high-order system. A simple and common way to analyze this model is to linearize the model at near-hover state where ϕ, θ, and ψ are small. Though it is simple

for implementation and analysis, it's only accurate at near-hover state and could cause large error when operating at high speed. In this work, feed-back linearization is applied to control a quadrotor which is covered in the next section.

2.2 Feedback Linearization and Attitude Control

According to [10], for a nonlinear system defined as:

$$x = \mathrm{f}(x) + \mathrm{g}(\mathrm{x})\mathrm{u}$$

$$\mathrm{y}=\mathrm{h}(x)$$

where f and g are smooth vector fields and h is an infinitely differentiable function, there exists a function $u = \alpha(x) + \beta(x)v$ such that the output y and new input v are linearized.

As it is mentioned in the previous section, changing a quadrotor's movement can be achieved by changing its orientation and overall thrust. The rest of this section will present an attitude controller using feedback linearization [11].

First, define the states, inputs, and outputs of the altitude controller. The state is defined as $x_p = [\phi \quad \theta \quad \psi \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T$, input $u_q = [u_1 \quad u_2 \quad u_3 \quad u_4]^T$, and output $y_p = [u_1 \quad \phi \quad \theta \quad \psi\ ]^T$.

Then, define the following feedback linearization to obtain a linear system, as it is done in [11]

$$\begin{aligned} u_2 &= f_2(\dot{\varphi}, \dot{\theta}, \dot{\psi}) + u_2^* \\ u_3 &= f_3(\dot{\varphi}, \dot{\theta}, \dot{\psi}) + u_3^* \\ u_4 &= f_4(\dot{\varphi}, \dot{\theta}, \dot{\psi}) + u_4^* \end{aligned} \qquad (4)$$

where

$$
\begin{aligned}
f_2(\dot{\varphi},\dot{\theta},\dot{\psi}) &= \frac{I_{xx}}{L}\left(K_2\dot{\varphi} - \dot{\theta}\dot{\psi}\frac{I_{yy}-I_{zz}}{I_{xx}}\right)\\
f_3(\dot{\varphi},\dot{\theta},\dot{\psi}) &= \frac{I_{yy}}{L}\left(K_3\dot{\theta} - \dot{\varphi}\dot{\psi}\frac{I_{zz}-I_{xx}}{I_{yy}}\right)\\
f_4(\dot{\varphi},\dot{\theta},\dot{\psi}) &= I_{zz}\left(K_4\dot{\psi} - \dot{\varphi}\dot{\theta}\frac{I_{xx}-I_{yy}}{I_{zz}}\right)
\end{aligned}
\qquad (5)
$$

K_2, K_3, K_4 are parameters. Here, a linearized system is obtained in the following form:

$$
\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}K_2\dot{\varphi} + \frac{L}{I_{xx}}u_2^*\\ K_3\dot{\theta} + \frac{L}{I_{yy}}u_3^*\\ K_4\dot{\psi} + \frac{1}{I_{zz}}u_4^*\end{bmatrix} \qquad (6)
$$

Then, a linearized and decoupled system $\Sigma_p = (A_p, B_p, C_p, D_p)$ is obtained:

$$
\begin{aligned}
\dot{x}_p &= A_p x_p + B_p u_p\\
y_p &= C_p x_p + D_p u_p
\end{aligned}
\qquad (7)
$$

and

$$
A_p = \begin{bmatrix}0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\\0&0&0&K_2&0&0\\0&0&0&0&K_3&0\\0&0&0&0&0&K_4\end{bmatrix}\ B_p = \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&\frac{L}{I_{xx}}&0&0\\0&0&\frac{L}{I_{yy}}&0\\0&0&0&\frac{1}{I_{zz}}\end{bmatrix}
$$

$$
C_p = \begin{bmatrix}0&0&0&0&0&0\\1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\end{bmatrix}\ D_p = \begin{bmatrix}\frac{1}{m}&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}
$$

where $u_p = [u_1 \quad u_2^* \quad u_3^* \quad u_4^*]^T$ is the new input vector after feedback linearization. Note that u_1 is actually not used in attitude control, it is here to make the format consistent.

With the feedback linearized and decoupled system, it will be very easy to define a proportional control law. Taking the control of roll angle as an example, it will be feasible to have $u_2^* = w_2(\varphi_{des} - \varphi)$, where φ_{des} is the reference command. This forms a closed-loop system and its behavior is determined by the parameter pair (K_2, w_2), and it's not hard to get its transfer function:

$$F(s) = \frac{X_4(s)}{X_{4des(s)}} = \frac{1}{Ixx/(Lw_2) \cdot s^2 - (K_2 I_{xx})/(Lw_2) \cdot s + 1}$$

which is a second-order transfer function. To make it stable, K_2 must be negative to make its poles on the left-half plane. With the PID control design function in the Control System Toolbox [12] and the transfer function above as the design plant, it is not hard to get parameters with promising results. For the control on the roll angle, having $K_2 = -80$ and $w_2 = 414.51$ can achieve 0.11 second of settle time and 0 % overshoot.

2.3 Decidable Hybrid System

Hybrid system is a kind of system that consists of a combination of continuous and discrete states [13], [14], [15], [16]. A hybrid system can be formally defined as:

$$HS = (\mathcal{X}, L, X_0, I, f, T),$$

where $\mathcal{X} \subseteq \mathbb{R}^N$ is the set of continuous state, L is the set of discrete locations (also called cells in this work), $X = \mathcal{X} \times L$ is the overall continuous-discrete state space. Here, $X_0 \subseteq X$

is the set of initial states, I is the function that assigns $x \in \mathcal{X}$ to some $l \in L$ such that x lies inside the region labeled by l. $T \subset L \times \mathcal{X} \times L$ is the set of discrete transitions and $f: L \rightarrow (\mathcal{X} \rightarrow T\mathcal{X})$ specifies the continuous flow (vector fields) in each location.

Hybrid system is a major modeling framework for a large class of systems like air-traffic management systems, self-driving vehicles, and robotics. A hybrid system is decidable if there exists a computation procedure that could verify any property in the system within a finite number of steps. One key property that needs to be verified in the areas of motion planning is reachability and avoidance. This is the problem of determining whether the system could reach a set of states (targets) while avoid reaching another set of states (obstacles).

Calin Belta et al in [17] proposed a framework that synthesizes a rectangular multi-affine hybrid system which is bisimilar [17] with its discrete quotient transition system $DS = (L, L_0, T_d)$, where L has the same meaning as it is in HS, L_0 is the set of initial discrete states and $T_d \subseteq L \times L$ is the discrete transition system. The bisimulation property was first introduced in [18] and [19], it indicates whether two systems are equivalent in reachability properties.

In this work, spaces are divided into non-overlapping N-dimensional rectangles characterized as a tuple (a, b), where $a = (a_1, a_2, \cdots a_N)$, $b = (b_1, b_2, \cdots b_N)$, and $a_i < b_i$ for $i = 1,2, \cdots N$. For each rectangle, the multi-affine function vector field f is assigned as:

$$f(x_1, x_2, \cdots, x_N) = \sum_{(v_1,\cdots,v_M)\in V_N} \prod_{k=1}^{N} \left(\frac{x_k - a_k}{b_k - a_k}\right)^{\zeta_k(v_k)} \cdot \left(\frac{b_k - x_k}{b_k - a_k}\right)^{1-\zeta_k(v_k)} f(v_1, \cdots, v_N)$$

and

$$\dot{x} = f(x)$$

where $x = (x_1, \cdots, x_N)$ is the continuous position, $V_N = \prod_{i=1}^{N}\{a_i, b_i\}$ is the set of 2^N vertices, and function $\zeta_k : \{a_k, b_k\} \to \{0,1\}$ is the indicator function that $\zeta_k(a_k) = 0, \zeta_k(b_k) = 1$. Hybrid systems with this multi-affine function can be informally explained as follows: the value of this function is uniquely determined by its values at its vertices. If all the vectors at the vertices "point out" of a specific facet, then any continuous trajectory following this vector field will go through the same facet. If all the vectors at the vertices "point inside" the rectangle, then all continuous trajectories will stay inside the rectangle. Thus, such HS and its corresponding DS are bi-similar. A formal proof and a more detailed discussion can be found in [5] and [20].

2.4 Linear Temporal Logic (LTL)

Temporal logic provides tools for reasoning standard logic statements over time. It is commonly used in formal verification tools in robot motion planning and for requirements of computer programs. As a formal statement tool, its statements are precise and unambiguous. It is also similar to structured English syntax, making it very easy to understand. Linear Temporal Logic (LTL) is a kind of temporal logic whose syntax contains the following temporal operations such as always (G), eventually (F), next (X) and until (U), and the standard logic operators like negation ($\neg$), disjunction ($\vee$),

conjunction (∧), and implication (⇒). LTL formulas are defined according to the following grammar:

$$\phi \ ::= \pi \mid \neg\phi \mid \phi \vee \phi \mid \phi \mathcal{U} \phi$$

where ϕ is a LTL formula, Π is the set of atomic propositions, and $\pi \in \Pi$ is an atomic proposition. In real applications, LTL can be used to describe complex behaviors for robots. The following are several examples for some frequently used specifications:

- Coverage: the LTL formula $F\pi_1 \wedge F\pi_2 \wedge F\pi_3 \wedge \cdots \wedge F\pi_n$ specifies that robot should eventually visit region $\pi_1, \pi_2, \cdots, \pi_n$ with no particular order.
- Reachability and avoiding obstacles: the LTL formula $\neg(\pi_1 \vee \pi_2 \vee \pi_3 \vee \cdots \vee \pi_n)\mathcal{U}\ target$ specifies that before robot reaches its target *target*, it should avoid regions $\pi_1, \pi_2 \cdots \pi_n$.
- Non-strict Sequencing: the LTL formula $F(\pi_1 \wedge F(\pi_2 \wedge F(\pi_3)))$ specifies that the robot should reach region $\pi_1, \pi_2,$ and π_3 in that order (without preceding sequences like $\pi_1, \pi_3, \pi_2, \pi_3$.

It has been proven that any LTL formula can be converted to Büchi automaton [21] defined as:

$$B \coloneqq (S_B, S_{B0}, T_B, \Sigma_B, F_B)$$

where S_B is the set of states, S_{B0} is the set of initial states, $T_B \subseteq S_B \times \Sigma_B \times S_B$ is the non-deterministic transition function, Σ_B is the input alphabet, and $F_B \subseteq S_B$ is the set of accepting states. There are already several publicly available tools that can achieve this like

ltl2ba [22]. Figure 3 shows the generated Büchi automaton using $ltl2ba$ from LTL formula:

$$\phi_l = \boldsymbol{GF}\, photo \wedge \boldsymbol{G}\,(photo \Rightarrow \boldsymbol{X}\, upload) \wedge \boldsymbol{G}(upload \Rightarrow \boldsymbol{X}\, photo) \quad (8)$$

where the request $photo$ lies in cell (3,1) and the request $upload$ lies in cell (5,10), as it is shown in Figure 2.

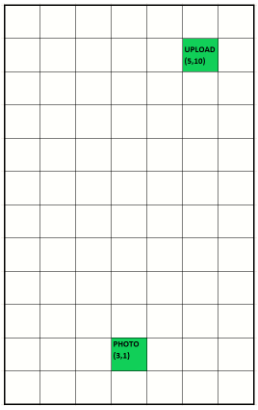


Figure 2 An Environment for Formula 8

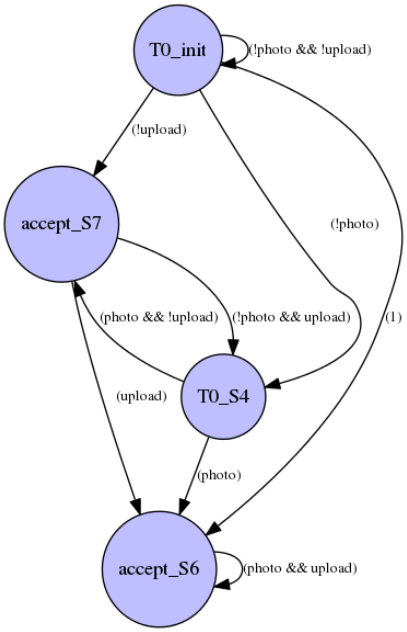


Figure 3 Buchi Automaton Captured from Formula 8

A common and proven method to use LTL as planning tool is to form the product automaton P from the product of the Büchi automaton B and a weighted transition system T. The weighted transition system T can be defined as:

$$T \coloneqq (S_T, s_{T0}, T_T, h_T, w_T)$$

where S_T is the set of discrete states, s_{T0} is the initial state, $T_T \subseteq S_T \times S_T$ is the transition function, $h_T : S_T \to 2^{\Pi}$ is the labeling function giving a state the corresponding atomic propositions, and $w_T: T_T \to \mathbb{N}$ is the function assigning non-negative weight to each transition. The product automaton P can be defined as:

$$P \coloneqq B \otimes T = (S_P, S_{p0}, T_P, w_P, F_P)$$

where $S_p \subseteq S_T \times S_B$ is the set of states, S_{p0} is the set of initial state that $\{(s_T, s_B) | (s_B, h_T(s_T), s_B') \in T_B, s_B \in S_{B0}, s_T \in S_{T0}\}$, $T_p = \{((s_T, s_B), (s_T', s_B')) | (s_T, s_T') \in T_T, (s_B, s_B') \in T_B\}$ is the transition function, $w_P = w_T(q_T, q_t')$ is the weight function, and $F_P = \{(s_T, s_B) | s_B \in F_B\}$ is the set of accepting states. This way, the product transition automaton captures both the mission specification information and transition information. Figure 3 shows the Büchi automaton generated from the LTL formula 8. Planning based on P will satisfy the requirements of both the LTL formula and the transition system. Figure 4 shows the global transition system in which the weight of transition is the smallest manhattan distance and Figure 5 is its product with the previous Buchi automaton. Figure 3, Figure 4, and Figure 5 are obtained from the codes that come with [7].

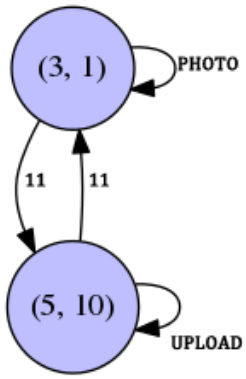


Figure 4 Global Transition System for Environment in Figure 2

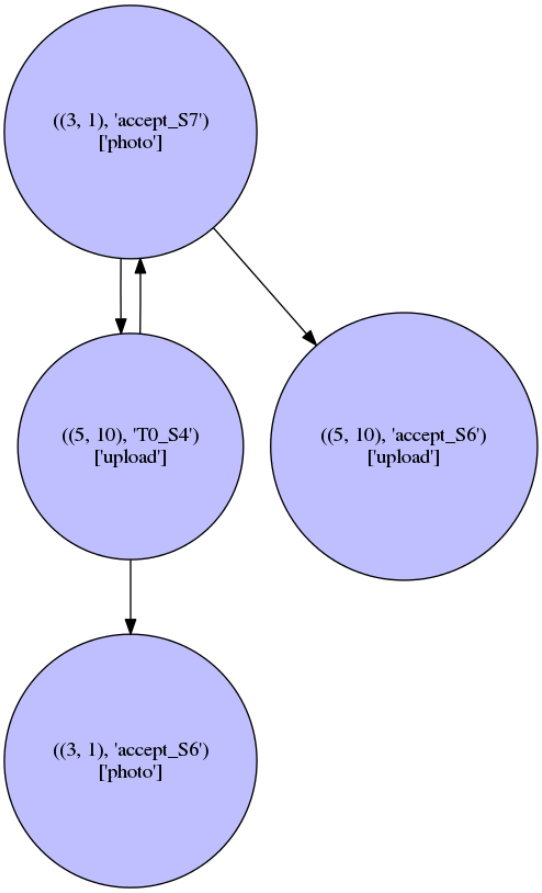


Figure 5 Global Product Transition System Based on Equation 9 and Figure 4

2.5 Approximate Simulation Relations

Approximate simulation relations, as it was introduced in [23] and [24], allows to characterize a simple approximate system from the actual complex system that can be used to simplify the controller design. The approximate system can be lifted to the actual system through an interface function characterized by a simulation function and the errors between the trajectories of the two systems are guaranteed to be bounded. A detailed discussion and applications can be found in [25], [24], and [26]. Methodologies for designing such interfaces are described in [27].

Generally, systems using approximate simulation relations contain at least two layers. The first layer is an abstract model which is a simplified version of the real system that is used to meet the high-level specifications. The second layer is a higher fidelity model that is closer to the real system. Considering the following two models:

$$\Sigma: \begin{cases} \dot{x}(t) = f(x(t), u(t)) \\ y(t) = h(x(t)) \end{cases}$$

where $x(t) \in X \subseteq \mathbb{R}^n$ is the state of the system, X_I is the set of initial states and $x(0) \in X_I, u(t) \in U \subseteq \mathbb{R}^q$ is the input of the system, $y(t) \in Y \subseteq \mathbb{R}^k$ is the output of the system and

$$\Sigma': \begin{cases} \dot{z}(t) = g(z(t), v(t)) \\ y'(t) = k(z(t)) \end{cases}$$

where $z(t) \in Z \subseteq \mathbb{R}^m, Z_I$ is the set of initial states of the abstract system and $z(0) \in Z_I$, $v(t) \in V \subseteq \mathbb{R}^p$ is the input of the system and , $y'(t) \in Y \subseteq \mathbb{R}^k$ is the output of the system.

To apply approximate simulation relations, Σ' must be a complete approximate subsystem of precision δ of Σ. A complete subsystem can be defined as follows: for any initial state such that $x(0) = x_0 \in X_I$ of Σ, there is an initial state $z(0) = z_0 \in Z_I$ of Σ' such that for every state trajectory $z(t)$ starting at z_0 of Σ', there always exists a state trajectory $x(t)$ of Σ starting at x_0 satisfying $\|h(x(t) - k(z(t))\| \leq \delta$ for all $t \geq 0$.

The approximate simulation relations can be defined as follows:

Definition 1: A relation $\mathcal{R} \subseteq \mathbb{R}^m \times \mathbb{R}^n$ is an approximate simulation relation of precision δ between Σ' and Σ if for all $(z_0, x_0) \in \mathcal{R}$,

1) $\|h(x_0) - k(z_0)\| \leq \delta$
2) For any state trajectory $z(t)$ of Σ' starting at $z(0) = z_0$ there exists a state trajectory $x(t)$ of Σ starting at $x(0) = x_0$ and satisfying: $\forall t \geq 0, \big(x(t), z(t)\big) \in \mathcal{R}$.

Such approximate simulation relations can be achieved by using a class of functions called simulation functions [12]. A simulation function is a non-increasing positive function that bounds the distance between the trajectories of Σ and Σ'.

Definition 2: A function $V : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}^+$ is a simulation function between Σ and Σ' for all $(z, x) \in \mathbb{R}^m \times \mathbb{R}^n$ that satisfies:

$V(z, x) \geq \|k(z) - h(x)\|$, and

$$\sup_{v \in V} \inf_{u \in U} \left(\dot{V}\big(x(t), z(t)\big)\right) = \sup_{v \in V} \inf_{u \in U} \left(\frac{\partial V(z,x)}{\partial z} g(z, v) + \frac{\partial V(z,x)}{\partial x} f(x, u)\right) \leq 0$$

This definition can be interpreted as follows: $v(t)$ can be considered as a disturbance signal and $u(t)$ as a control signal, then for any disturbance signal, there can always be a control signal that the simulation function is non-increasing. Knowledge of $v(t)$ can be used to construct $u(t)$. Fainekos at el in [27] present a method using the knowledge of $v(t), z(t)$, and $x(t)$ to form an interface that achieves a maximum tracking error bounded by $2v_{max}$ where v_{max} is the maximum value of $v(t)$. This work will be reviewed with details later in Section 3.

Then, since

$$\|k(z_0) - h(x_0)\| \leq \sqrt{V(z_0, x_0)} \leq \delta, (z_0, x_0) \in \mathcal{R}.$$

it can be inferred that a relation defined by

$$\mathcal{R} = \{(z, x) \in \mathbb{R}^m \times \mathbb{R}^n \mid V(z, x) \leq \delta^2\}$$

is also an approximate relation of precision δ between Σ' and Σ.

CHAPTER 3

RELATED WORK

This thesis describes a framework that aims to solve the problem of controlling a quadrotor modeled by second-order dynamics to satisfy specifications expressed in linear temporal logic. There is already a lot of related work that solves part of the problem. This section reviews some previous works that are closely related or directly used in this thesis.

3.1 Temporal Logic Motion Planning for Dynamic Robots

Fainekos et al in [3] presented a framework solving the problem of temporal logic motion planning for general ground robots modeled by second order dynamics using approximate relations and Temporal Logic over the Reals (RTL) [28].

Consider a second-order dynamic model of a ground robot Σ_r described as:

$$\Sigma_r: \begin{cases} \dot{x}(t) = y(t), x(t) \in X, x(0) \in X_0 \\ \dot{y}(t) = u(t), y(t) \in \mathbb{R}^2, y(0) = [0\ 0]^T \end{cases}$$

where $x(t)$ is the position of the robot, $y(t)$ is its velocity, and $u(t)$ is the input command, which is also robot's acceleration, and $u(t) \in U = \{\mu \in \mathbb{R}^2 | \|\mu\| \leq u_{max}\}$. Its states are defined as $\theta(t) = [x^T(t)\ \ y^T(t)]^T$. Then they defined the following notation:

$$\dot{\theta}(t) = A\theta(t) + Bu(t), x(t) = C_x\theta(t), y(t) = C_y\theta(t).$$

Also, consider an abstracted first-order kinematic model Σ_r':

$$\Sigma_r': \dot{z}(t) = v(t), t \geq 0, z(0) \in Z_0$$

where $z(t) \in Z \subseteq \mathbb{R}^2$ is the position of the robot in this kinematic model, and $v(t) \in V = \{\mathcal{v} \in \mathbb{R}^2 | \|\mathcal{v}\| \leq v_{max}\}$ is its input representing its velocity.

Then they proposed an interface function and a simulation function that $\Sigma r'$ is an approximate relation to Σ_r with bounded error proportional to v_{max}.

Suppose α satisfies the following inequation:

$$\frac{v_{max}}{2}\left(1 + \left|1 - \frac{1}{\alpha}\right| + \frac{2}{\sqrt{\alpha}}\right) \leq u_{max}, \alpha > 0 .$$

Then, the approximate relation given by:

$$\mathcal{V} = \{(z, \theta) \in \mathbb{R}^2 \times \mathbb{R}^4 | \mathcal{S}(z, \theta) \leq 4\mathcal{v}_{max}^2\}$$

is an approximate relation of precision $2\ v_{max}$ between $\Sigma_r{}'$ and Σ_r . $\mathcal{S}(z, \theta) = \max(\mathcal{Q}(z, \theta), 4v_{max}^2)$ where

$$\mathcal{Q}(z, \theta) = \|C_x\theta - z\|^2 + \alpha\left\|C_x\theta - z + 2C_y\theta\right\|^2.$$

Then, function

$$u_r(v, z, \theta) = \frac{v}{2} + \frac{-1 - \alpha}{4\alpha}(C_x\theta - z) - C_y\theta$$

is the associated interface function.

One of the most valuable part of this conclusion is that the maximum tracking error between the dynamic model and the kinematic model is bounded to two times the maximum velocity of the kinematic model. This means that the tracking accuracy can be improved by reducing the kinematic model's maximum velocity. Also, the tracking error

will grow together with robot's velocity, forcing designers to make a trade-off between speed and accuracy.

3.2 Receding Horizon Planning

Ulusoy and Belta in [7] present a new kind of receding horizon planner for automatically controlling a robot satisfying both the global and local mission specifications. Consider a decomposed environment $\mathcal{E}$ defined as:

$$\mathcal{E} = (\mathcal{C}, \mathcal{S}, \mathcal{L}_s, \mathcal{D}, \mathcal{L}_d(t))$$

where $\mathcal{C} = \{c_{x,y,z} \,|0 \leq x < p, 0 \leq y < q, 0 \leq z < r\}$ is a $p \times q \times r$ matrix of cubic cells, $\mathcal{S}$ is the set of global static requests, $\mathcal{L}_s: \mathcal{C} \to \mathcal{S}$ is a map from a cubic cell to global static requests located in this cell, $\mathcal{D}$ is the set of local dynamic requests, and $\mathcal{L}_d(t): \mathcal{C} \to \mathcal{D}$ is a time-varying map from a cubic cell to local dynamic requests occurred in this cell at the time t. The global mission specification ϕ_g is expressed in LTL and is defined over the set of global static request $\mathcal{S}$ defined in $\mathcal{E}$, it dictates the global motion of the robot in the environment. The local mission specification consists of a priority function $prio: \mathcal{D} \to \mathbb{N}$ where lower number means higher priority, and a regular expression ϕ_l over the set of local dynamic request $\mathcal{D}$, it specifies how the vehicle should respond to the local dynamic requests detected within the sensor range. The generated trajectory by this method is guaranteed to be shortest in manhattan distance if the target is within the sensor range or there is no obstacle between current position and target. Figure 6 demonstrates its execution procedure.

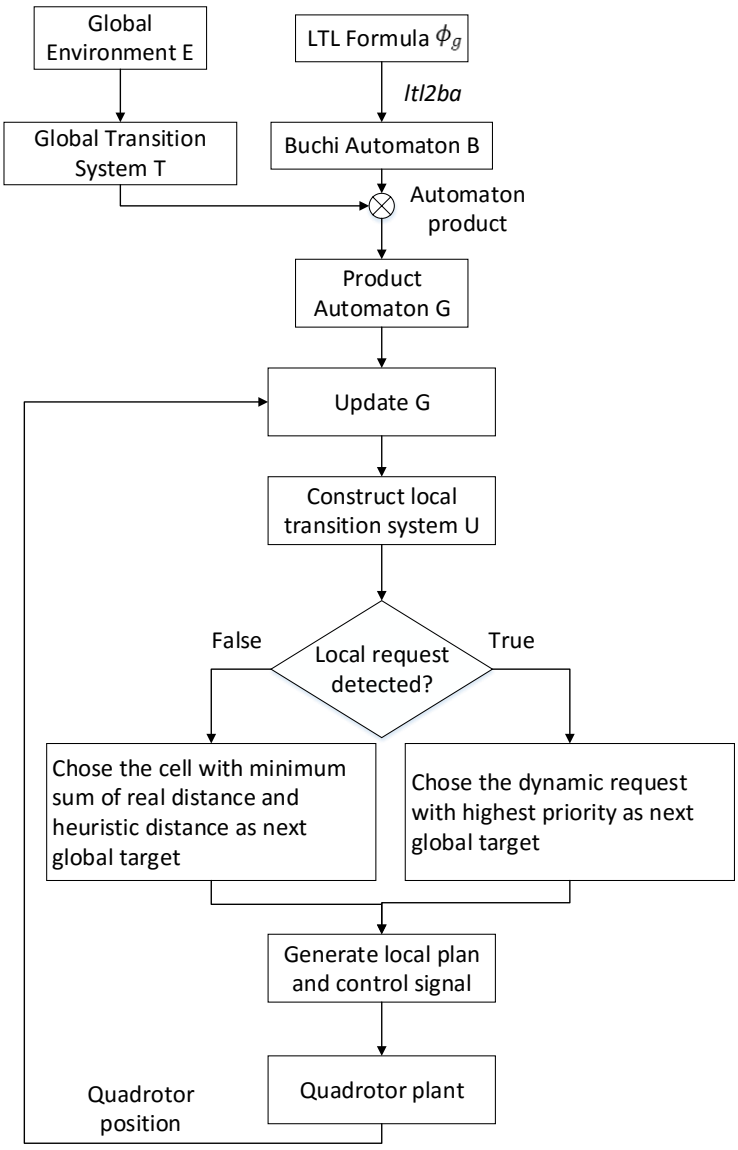


Figure 6 Execution Process of the Receding Horizon LTL Planning Algorithm

In Figure 6, the algorithm first translates the LTL formula ϕ_g to a Buchi automaton B. Then, it takes the product of the Buchi automaton B with the global transition system T to generate a global product automaton G, which captures the motion between cells with global requests and global mission specifications. Then, a local transition system U which covers areas within the sensor range is constructed and a local path within U is calculated which satisfies both the global mission specification and the local mission specifications,

and is able to avoid the unsafe areas detected in U. It will first serve the local requests with the highest priority if there are any, then head to the closest global request cell if it is in the sensor range, or head to a cell on the border of the sensor range that has the minimum sum of real distance and heuristic distance. Here, the real distance is the distance between the current position and the target border cell, and the heuristic manhattan distance is between the border cell and the next global target.

The receding horizon method generates a local discrete trajectory instead of just the next neighboring cell, which will be helpful when generating a continuous trajectory from it (we will show this in the next section). Also, its computation does not rely on all the global environment data, but only on environment data within its sensor range. These are very desirable properties when flying UAVs in real world where the environment data is enormous and the computation power is limited.

As the sensor range is limited when compared to the size of the global environment, it is possible that there is no satisfying local discrete plan within the sensor range. This problem is not handled in the current theory as presented in [7], as it is demonstrated in Figure 7 and Figure 8.

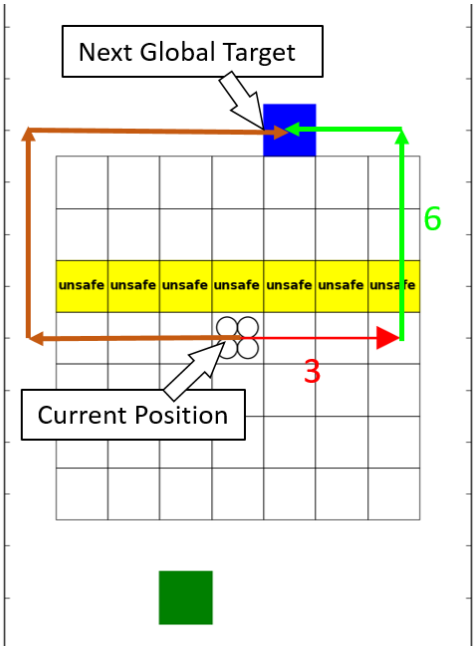


Figure 7 Situation with no Valid Local Plan

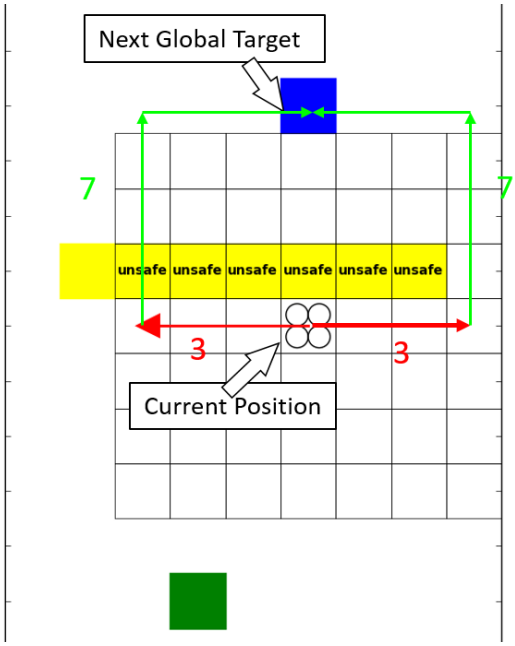


Figure 8 Situation with no Valid Local Plan

In Figure 7, the grid is the local planning area with distance 3 and the blue cell is the next global target. The shown local path calculated by the local planner with the

minimum sum of the real distance (the red arrow) and heuristic distance (green arrow) is not valid while there is a feasible alternative plan (brown arrow) that the local planer cannot find. Then, the quadrotor moves to the right cell, as in Figure 8 and then it has to re-plan. A problem is raised here that there are two local plans with the same summation of real distance and heuristic distance. Thus the behavior of this algorithm is nondeterministic. If it chose to go left, it will go back to the situation in Figure 7 and get trapped in a live-lock situation and cannot move forward.

One solution to handle this is by adding a higher-level resolver with global environment information. As the calculation of such a resolver will involve global environment data, it should not be called too frequently, it will only be called when local planning has reached to a live-lock situation.

Another solution is to increase the the environment's resolution. As it is mentioned before, the environment is decomposed into cubic cells which will inevitably lose some geometric data. A solid obstacle in a coarsely decomposed environment may found "apertures" if the decomposition resolution is increased. This will be shown in detail in Section 5.3.

CHAPTER 4

PROBLEM FORMULATION

Consider a non-linear quadrotor model Σ_q and the corresponding feedback linearized concrete model Σ, as it is described in Section 2.1 and 2.2, and a decomposed environment

$$\mathcal{E} = (\mathcal{C}, \mathcal{S}, \mathcal{L}_s, \mathcal{D}, \mathcal{L}_d(t))$$

where $\mathcal{C} = \{c_{x,y,z} \,|0 \leq x \leq p, 0 \leq y \leq q, 0 \leq z \leq r\}$ is an $p \times q \times r$ matrix of cubic cells, $\mathcal{S}$ is the set of static requests, $\mathcal{L}_s: \mathcal{C} \to \mathcal{S}$ is a map from a cubic cell to static requests located in this cell, $\mathcal{D}$ is the set of dynamic requests, and $\mathcal{L}_d(t): \mathcal{C} \to \mathcal{D}$ is a time-varying map from a cubic cell to dynamic requests occurred in this cell at the time t. The construction of the linearized concrete model Σ will be described in detail in later section.

The goal of this thesis is to design a hierarchical hybrid controller that generate control signal $u(t)$ for Σ such that its resulting state trajectory $x(t)$ satisfies both the global mission specification ϕ_g expressed in LTL and local specifications ϕ_l that is the same as defined in Section 3.2.

A hierarchical synthesis approach is used to achieve this objective. The hierarchical structure consists of a high-level planner that generates discrete plans satisfying all global and local specifications, a low-level controller handling continuous controls that makes sure its output could track the input command, and a synthesis framework using the approximate simulation relation method which guarantees the trajectory of the concrete model Σ can track its abstracted kinematic model Σ' with a bounded error.

Thus, the problem can be described as follows: *Given an environment $\mathcal{E}$, a non-linear model Σ_q of a quadrotor, global requirement ϕ_g and local requirement ϕ_l, design a hierarchical hybrid controller for Σ_q such that its trajectories satisfy all the specifications.*

The following example is based on the example in [7] but extended to 3D environment with more complexed obstacles, as it is shown in Figure 9.

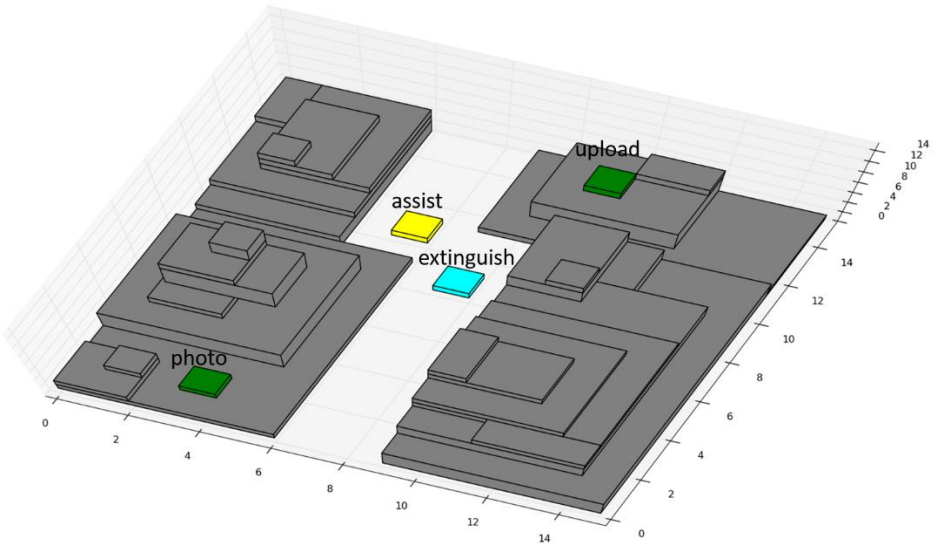


Figure 9 A Surveillance Environment

In Figure 9, the green cells are global requests which include $photo$ in cell (3,1,2) and $upload$ in cell (9,13,5) respectively, the yellow cell is the local dynamic request $assist$ in cell (5,10,1), the cyan cell is the local dynamic request $extinguish$ in cell (7,8,1), and the grey cells are the obstacles. The global static mission specification is the same as Formula 8 and the local dynamic request is defined as:

$$\phi_l \coloneqq (extinguish|assist)^*, prio(assist) = 1, prio(extinguish = 0) \quad (9)$$

which can be interpreted as: Serve the assist and extinguish requirement if they are detected within the sensor range and extinguish has higher priority than assist.

CHAPTER 5

HIERARCHICAL HYBRID CONTROLLER DESIGN

This chapter presents and analyzes the design of a hierarchical hybrid controller following the approaches described in Sections 2.3 and 2.5 as well as how it is adapted to control a quadrotor which is a complex non-linear system. Another design based on traditional PID controller is also presented. Their performance comparison will be shown in Section 6.2.

5.1 Controller Based on PID Algorithm

Constructing controllers for complex physical systems have been one of the biggest challenges in the area of motion planning. There have been a lot of proposed solutions to achieve this goal and PID control has become one of the most popular control algorithms in the past decades due to its simplicity and efficiency. According to [29], 90% of industrial controllers are based on the PID algorithm.

The advantage of PID control is that it does not require much knowledge of the control target. The design and parameter tuning can be done by analyzing the input and output of the target system. The recent booming of "plug-and-play" PID controllers and automated parameter optimization tools have made it even more convenient to use PID control [30].

Figure 10 presents the block diagram of the structure of a PID control system for a quadrotor. In the block diagram, Σ_q is the non-linear model of the quadrotor described by Equation 3, r is the position of the quadrotor and u_1, u_2, u_3, u_4 are the input to Σ_q. The receding horizon LTL planner P_{RH} is based on the algorithm presented in Section 3.2 and

v is the calculated output of the planner by constructing local multi-affine vector fields as it is shown in Section 2.3. The calculated v, representing the desired velocity of the quadrotor, serves as the input to the kinematic model Σ_k of the quadrotor defined as:

$$\Sigma_k: \begin{cases} \dot{x}_k(t) = A_k x_k(t) + B_k v_k(t) \\ y_k(t) = C_k x_k(t) + D_k v_k(t) \end{cases} \quad (10)$$

where $x_k(t) \in X_{k0} \subseteq \mathbb{R}^3$ is the position vector in the 3D environment for this kinematic model, $v_k(t) \in V = \{v \in \mathbb{R}^3 | \|v\| \leq v_{max}\}$ is the input velocity for this system, $y_k(t) \in X_{k0} \subseteq \mathbb{R}^3$ is the output, and A_k, B_k, C_k, D_k are defined as follows:

$$A_k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} C_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} D_k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

It is already proven in [17] that the output trajectory $y_k(t)$ of Σ_k is guaranteed to satisfy the mission specification in the planner P_{RH}.

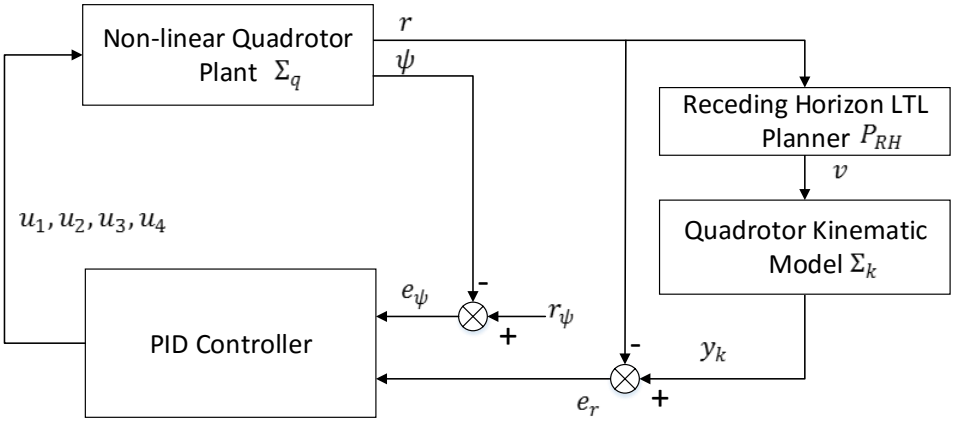


Figure 10 PID Controller Block Diagram

Then, $e_r = [e_x, e_y, e_z]^T$, which is the tracking error between the position of Σ_q and Σ_k, and e_ψ, which is the error between the yaw angle ψ in Σ_q and the reference yaw angle

r_ψ, are fed into the PID controller. Here, r_ψ is set to 0 in this work because there is no requirement on the yaw angle and this do not have any effect on the quadrotor's position control [11]. However, it is fully capable of tracking reference command if it is necessary in future works. The output of the PID controller is calculated using the following equations:

$$\begin{cases} u_1 = k_{pz}e_z + k_{dz}\,\dot{e}_z + k_{iz}\int e_z dt \\ u_2 = k_{py}e_y + k_{dy}\,\dot{e}_y + k_{iy}\int e_y dt \\ u_3 = k_{px}e_x + k_{dx}\,\dot{e}_x + k_{ix}\int e_x dt \\ u_4 = k_{p\psi}e_\psi + k_{d\psi}\,\dot{e}_\psi + k_{i\psi}\int e_\psi dt \end{cases}$$

where k_{p*} ,$*= \{x, y, z, \psi\}$, are the PID parameters. With the help of the *Simulink Design Optimization* toolbox [31] in MATLAB/Simulink, tuning these parameters has become a much easier job. The PID parameters are trained by having the quadrotor's non-linear model tracking a ramp signal, which simulates the controller's work situations where it has to track the trajectory of the quadrotor's abstract model Σ_k that flies with constant velocity. Section 6.2 contains more details of the optimization and the results can be found in Table 1.

5.2 Hierarchical Hybrid Controller

Although the PID control algorithm is easy to use, it does not give any formal guarantee on its dynamic tracking error which is crucial for safety. It is believed in this work that a hierarchical controller, as it is proposed and applied in [24] and [27], will give better performance and guarantee for safety. In this work, progress are made towards that direction. Namely, It is established here that if the hierarchical controller in [27] is used,

then experimentally better performance can be achieved than the PID architecture in Section 5.1.

Hierarchical control design has been proven to be an efficient approach for controlling complex non-linear systems. The advantage of the hierarchical structure is that each layer handles a simpler local problem coordinated by higher layers which will not override its decisions. This way, a complex problem is broken into a series of simpler problems and organized in a hierarchical structure. The hierarchical structure of this work is very similar to the approach in [27], but extended to controlling quadrotors. The hierarchical control architecture is shown in Figure 11.

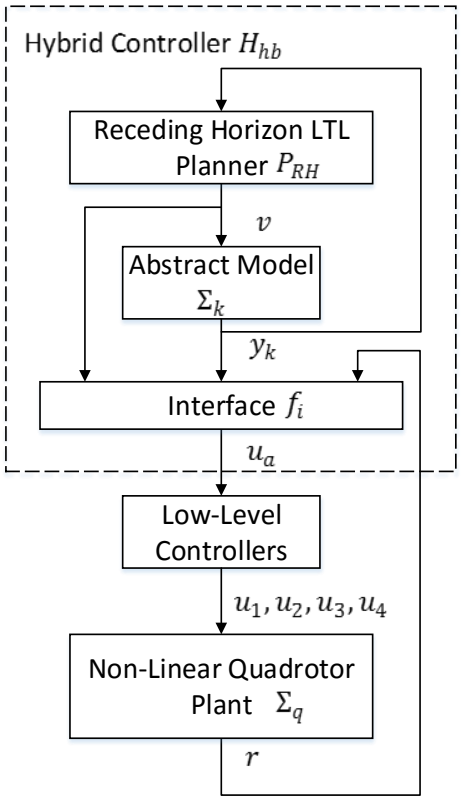


Figure 11 Hierarchical Structure

There are two layers in this hierarchical structure.

The first layer includes a receding horizon LTL planner P_{RH} as proposed in [7] and an abstract kinematic model of the quadrotor Σ_k as in Equation 9. The reason of adding an extra abstract model is to simplify the planning problem which otherwise is an undecidable problem, in general. These two parts are exactly the same with those in Section 5.1.

The following is an interface function f_i that takes the actual position of the quadrotor r, the position in the kinematic model y_k, and the desired velocity from the vector field v as input and computes the desired acceleration vector u_a as output. The generated acceleration vector u_a serves as the input to the next layer. The interface function f_i is used as computed in [27], which means that for a second order dynamic model with $\ddot{r} = u_a$ the tracking error $\|r - y_k\|$ is bounded by $2v_{max}$. Note though that the interface function f_i should be computed as described in [24]. Here, this work shows that the interface from [27] still achieves good experimental performance, albeit without formal guarantees. These three parts forms the hybrid controller H_{hb}.

The second layer consists of a series of low-level controllers and the non-linear quadrotor plant. In [27], the concrete model is a second-order dynamic model taking the acceleration command as input directly, which assumes that the acceleration could change immediately. This works for low-speed ground vehicles since the time duration that the torque is applied to the wheels is short enough to be ignored. But for a quadrotor, this assumption can no longer hold as the acceleration control is achieved by attitude control which has longer time constant. Thus, extra controllers are needed to achieve the desired acceleration command u_a from the interface function, as it is shown in Figure 12.

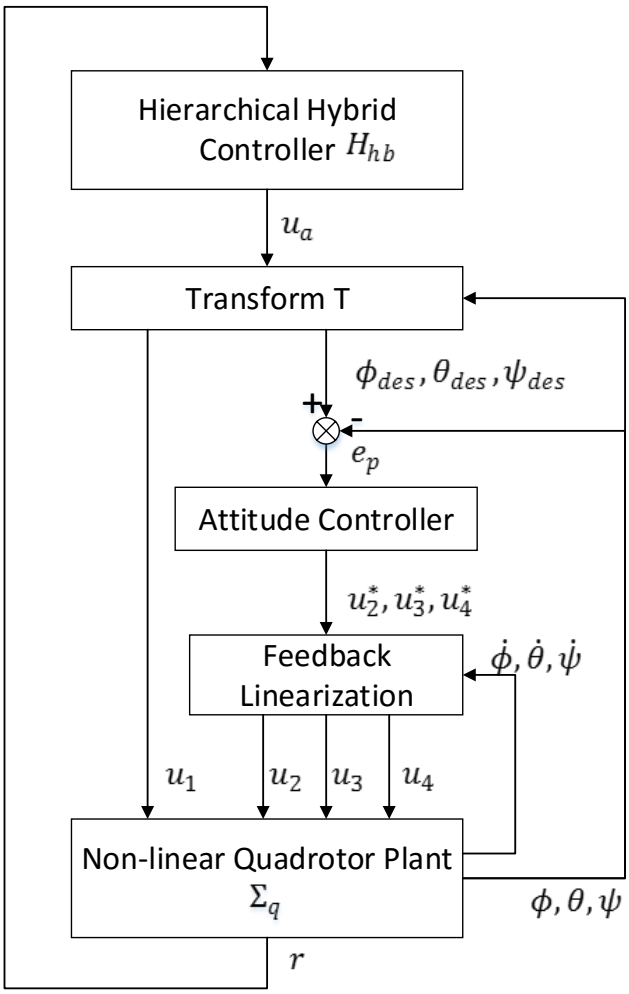


Figure 12 Low-level Controllers

Here, the hybrid controller H_{hb} is the part in the dashed rectangle in Figure 11, T is the transformation function that transfers the given acceleration vector u_a into the corresponding desired orientation angles $\phi_{des}, \theta_{des}, \psi_{des}$ and total thrust u_1. Assuming that the desired yaw angle ψ_{des} is 0, the relationship between u_a, u_1, and the desired orientation angles at static state can be described by the following equation [11]:

$$u_a = [u_{ax}, u_{ay}, u_{az}]^T = \begin{bmatrix} \frac{u_1 \sin(\theta_{des}) \cos(\varphi_{des})}{m} \\ \frac{u_1 \sin(\varphi_{des})}{m} \\ \frac{u_1 \cos(\theta_{des}) \cos(\varphi_{des})}{m} - g \end{bmatrix} \quad (10)$$

where g is the gravity acceleration and m is the mass of the quadrotor. Given the desired acceleration and current orientation angles, u_1 and the desired orientation angles can be calculated using the following equations:

$$\begin{cases} \phi_{des} = \text{atan}\left(\frac{u_{ay}}{u_{az} + g}\right) \\ \theta_{des} = \text{atan}\left(\frac{u_{ax}}{(u_{az} + g)/\cos(\phi_{des})}\right) \\ \psi_{des} = 0 \\ u_1 = \frac{(u_{az} + g) * m}{\cos(\phi) * \cos(\theta)} \end{cases}$$

Where ϕ and θ are current roll and pitch angles. Function 1 is the corresponding Matlab implementation:

Function 1: Transfer an acceleration vector to corresponding $u_1, \phi_{des}, \theta_{des}, \psi_{des}$

```
1  function [phi, theta, psi, u1]= fcnT(ax,ay,az,cur_phi, cur_theta)
2      g=9.81;
3      M=4;
4      phi=atan(ay/(az+g));
5      theta = atan(ax/((az+g)/cos(phi)));
6      psi = 0
7      u1=(az+g)*M/(cos(cur_phi)*cos(cur_theta));
```

where variable phi, theta, psi, and u1 represent the desired row, pitch yaw, and overall thrust respectively, ax, ay, and az are components of u_a, cur_phi, and cur_theta are the current row and pitch angles. As the yaw angle does not has any impact on the position control of a quadrotor and to simplify the analysis, the yaw angle is set to 0 throughout this

work. Also, in line 7, u1 is calculated using the current row and pitch angle instead of the desired angles. This is to ensure the dynamic stability of the control on the z-axis as the actual orientation angles cannot converge to the desired angles immediately.

Then, the calculated u_1 is directly fed to the non-linear quadrotor plant, $\phi_{des}, \theta_{des},$ and ψ_{des} are used to calculate e_p which is the difference between the desired angles and current angles. Then, the attitude controller takes e_p as input to calculate the feedback-linearized inputs $u_2^*, u_3^*, and\ u_4^*$, as it is shown in Equation (4) and (6). In the end, the rest of the control signals $u_2, u_3,$ and u_4 are calculated through the feedback linearization controller based on Equation 5.

One crucial issue that must be mentioned here is that the interface function f_i is utilized here in order to make the second-order dynamic model being able to track the corresponding kinematic model with a bounded error. But for the design described in this section, the existence of the non-linear transformation function T makes the whole system non-linear (the feedback linearization is only done for the nested attitude control). These controllers can only make the system behave like a second-order dynamic system by adjusting the quadrotor's orientation angles and thrust to follow the acceleration command, which will add extra dynamics to the system. Thus, for this design, the established bound in [27] is not theoretically guaranteed. The actual tracking error will be analyzed through simulation results in Chapter 6.

5.3 Local Plan Resolver

In the receding horizon method, a local plan within the quadrotor's sensor range needs to be calculated. The target of the local plan is either a global request cell in P, if it is within

the sensor range, or the cell that has the closest heuristic manhattan distance. But it is possible that there is no such trajectory that leads to that target. Figure 13 shows this situation.

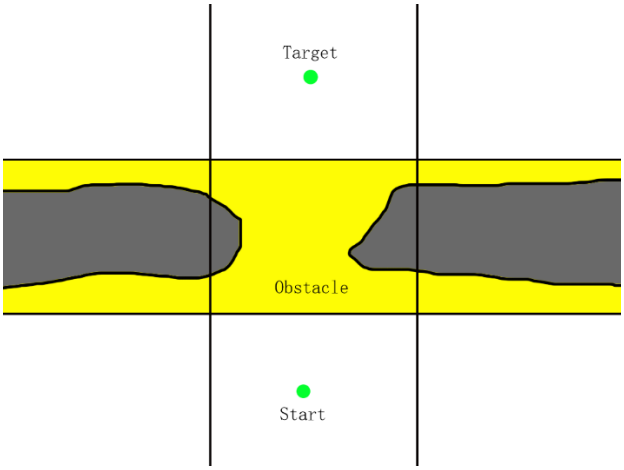


Figure 13 Situation when no Local Plan is Found

This is caused by the limited resolution of the environment $\mathcal{E}$ which results in some geometric data being lost. Thus, one possible solution is to increase the resolution in order to try to find a trajectory to the target. Figure 14 shows such an example.

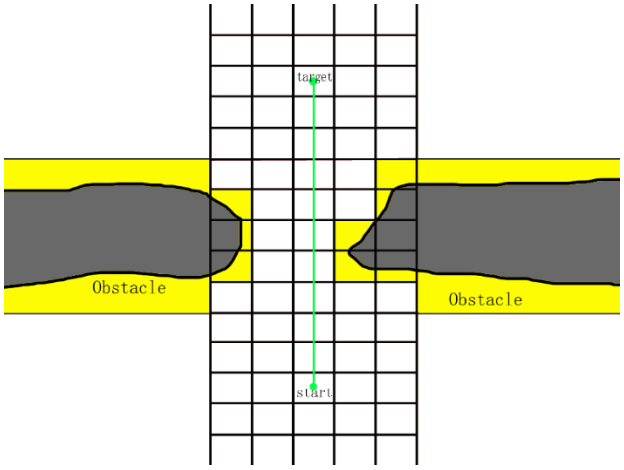


Figure 14 Path Found after Increasing Partition Resolution

The environment's resolution will be reduced after the quadrotor has reached the new target. As the new target in the high resolution environment is a subset of the old target in the original environment, the receding horizon planner can resume execution.

The advantage of this design is that it gives more flexibility to the trade-off problem between speed and safety that is mentioned in Section 2.5. The use of the approximate simulation relations method brings a bound for the tracking error, which is an important property for safety. By enlarging the obstacles, the value of this bound will ensure that the quadrotor will not hit the obstacles. Also, as the bound is proportional to the maximum planned velocity, it becomes possible to make sure that the continuous trajectory satisfies the mission specification in the planner by adjusting the velocity. The quadrotor can fly faster when there is more open space and fly slower when there is higher resolution and many obstacles to improve accuracy and safety.

5.4 Putting Everything Together

At this point, the design has a receding horizon planner, a continuous controller for the quadrotor and a synthesis framework using the approximate simulation relation method. Now it's time to put all these parts together to make them work with each other. One main challenge is that both the planner and the continuous controller need to check and calculate commands frequently, thus they should work in parallel. Algorithm 1 and Algorithm2 demonstrate the work procedure of the implementation in pseudocode.

Algorithm 1: System Initialization

```
1  Fetch the environment  env
2  Fetch the quadrotor configuration quad
3  Construct the global transition system, Buchi
     automaton, and product automaton
```

```
 4   Construct quadrotor controller quad_controller using
 5
 6   //State variable initialization
 7   cid = initial cell id
 8   state = initial quadrotor state // can be: fly, hover,
 9   route = None
10   fly_mode = normal
```

Algorithm 2: Main Control Loop

```
 1   System Initialization()
 2   while true:
 3       cid=quad_controller.get_cell()
 4       if route is None or cid is not route[0]:
 6           if route[0] or route[1] is high resolution
 7               //Switch to high-resolution mode
 8               fly_mode = slow
 9               quad.env,          planner.env          =
10               route = planner.genPlan()
11           else:
12               //Switch to normal mode
13               fly_mode = normal
14               quad.env, planner.env = env.normal_env()
15               route = planner.genPlan()
16           //Check validation
17           if route is None:
18               state = error
19               break;
20       if fly_mode is slow:
21           quad_controller.velocity = slow
22       else:
23           quad_controller.velocity = normal
24       quad_controller.fly_route(route)
```

Algorithm 1 handles the initialization of state variables, the environment, and the controller. The off-line calculation of the Buchi automaton, global transition system and global product transition system are also done at this stage.

Algorithm 2 demonstrates the high-level controller that handles different situations. As calling a receding horizon method to update the plan consumes much more computation power (generating local transition system, checking the global product system, and calling Dijkstra's algorithm to calculate a local path), it is only called when the quadrotor enters a new cell. To achieve fast velocity while ensuring safety, the environment has two decomposition systems, the normal environment and high-resolution environment. The high-resolution environment is only activated when at least one of the first two cells contains a higher resolution decomposition. Otherwise, the environment will switch back to the normal decomposition.

As it was presented before, the receding horizon method cannot guarantee that the generated path is valid and it will fail in two scenarios: the established target within the sensor range is not reachable and the established target leads the quadrotor to "linger" around. This is due to the limited range of its sensor which could not look far enough to find a better solution. One way to solve this problem is to conduct a global trajectory planning using A* algorithm. As it was mentioned in the previous section, one advantage of using the receding horizon method is that it does not rely on the global data for planning. Thus, the global trajectory planning is only called when there is no valid local plan that could make progress. Before calling the global trajectory planner, the distance between the local target and the nearest next target in P will be recorded as d*. The quadrotor will follow the global trajectory until its distance to the next target in P is smaller than d*, which means that quadrotor has moved out of the "trap" where it cannot find a valid local plan. Note that when the quadrotor is following the global trajectory, it cannot react to local dynamic requests.

Here, one problem that still remains open is the detection of the failure of receding horizon LTL algorithm. Based on current observations, the quadrotor can be trapped in a live-lock situation that move back and forth. But this is not theoretically proved and there can be other situation as well. For this reason, this problem is not yet solved in this work.

CHAPTER 6

IMPLEMENTATION AND SIMULATION

This section will describe the implementation of the design in Section 5.2 and make comparison with the PID only approach in Section 5.1 in order to demonstrate the benefits of this design.

6.1 Implementation

The hybrid controller is implemented in Python and the acceleration controller as well as the quadrotor simulator are implemented using MATLAB/Simulink. The hybrid controller will calculate the desired acceleration command and send it to the acceleration controller. Then the acceleration controller will generate the rotation commands for the four rotors that will follow the command and compute the simulated states of the quadrotor and send to the hybrid controller. The two parts communicate through UDP channels and work cooperatively to accomplish the mission. 1.3

The hybrid controller contains two parts that execute in parallel: the high level planner that executes in 10 Hz and the low level controller that executes in 200Hz. The two parts cooperate through reading and writing protected variables, including the current discrete trajectory and the quadrotor states, position and commands. The high level planner will re-plan and update the current discrete-trajectory when the quadrotor enters a new cell. If the quadrotor has reached the last cell in the discrete trajectory and have not received any command from the planner, it will hover at the center point of the last cell.

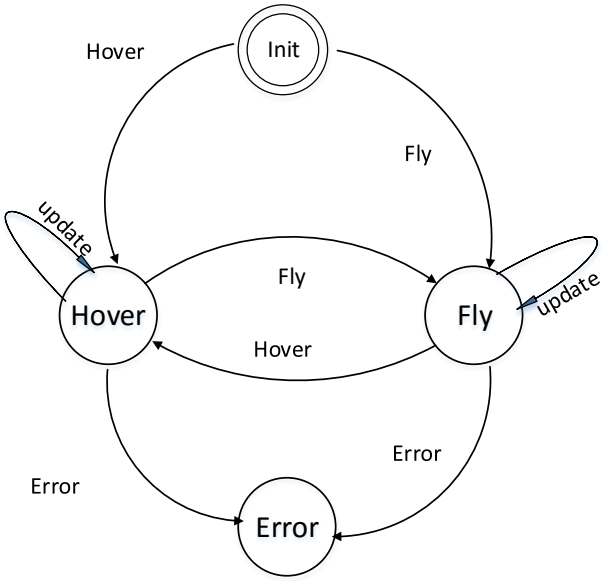


Figure 15 State Machine for the Planner and the Controller

6.2 Comparison with the PD Controller

This section presents the simulation results of having a quadrotor fly through a pre-defined discrete trajectory with different velocities to check its safety. The results will also be compared with the approach in Section 5.1 which uses a simple PD controller instead of the approximate simulation method. The two simulations are performed with the same quadrotor platform, same acceleration limit, and follow the same assigned discrete trajectory. In Figure 16 to Figure 23, the green lines are the trajectory of the abstract model Σ' and the red lines are the actual trajectory of the non-linear quadrotor model Σ_q. The yellow lines mark the differences between the two trajectories at the same time.

The results of using approximate simulation relations are shown in Figure 16 to Figure 23:

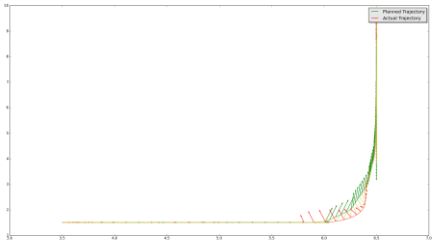


Figure 16 Trajectory on X-Y Plane at Velocity 0.4

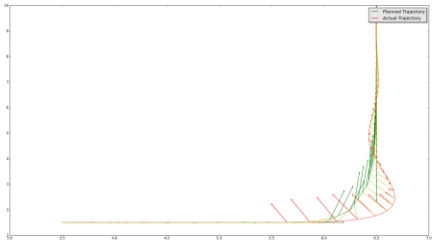


Figure 17 Trajectory on X-Y Plane at Velocity 0.8

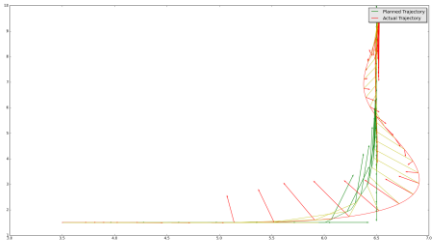


Figure 18 Trajectory on X-Y Plane at Velocity 1.2

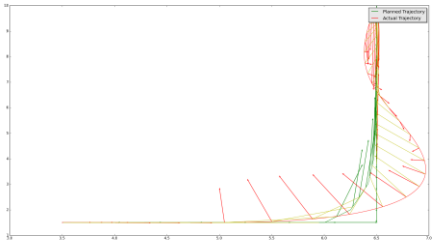


Figure 19 Trajectory on X-Y Plane at Velocity 1.4

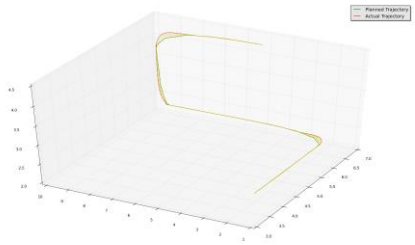


Figure 20 Trajectory at Velocity 0.4

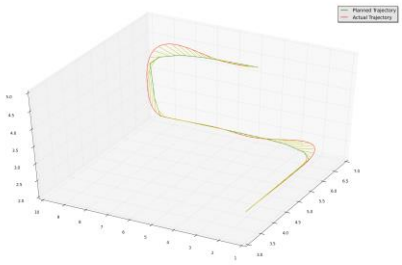


Figure 21 Trajectory at Velocity 0.8

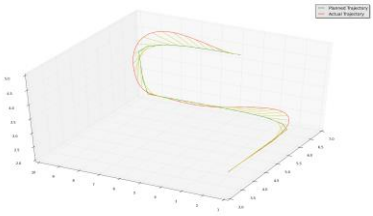


Figure 22 Trajectory at Velocity 1.2

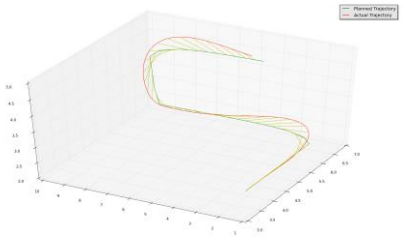


Figure 23 Trajectory at Velocity 1.4

The figures on the right panel are the actual 3D trajectories and the figures on the left panel are their projection onto the X-Y plane. As it can be seen, the tracking error increases with the velocity. The maximum velocity the approximate simulation method can achieve is 1.4 without flying out of the assigned cells.

One common design approach for a PID controller was discussed in Section 5.1. But the *Simulink Design Optimization* tool failed to converge during the parameter optimization for this approach. Up to this point, this work has not been able to figure out the reason of the failure and the possibility of a software bug or an improper parameter cannot be excluded. However, this work did succeed to optimize the parameter following the approach in [7], where the PD controller is used to generate an acceleration control signal:

$$\ddot{r}^{des} = k_p e + k_d \dot{e}$$

Here, $\ddot{r}^{des}$ is the computed acceleration command, e is the tracking error between the quadrotor's actual position and the abstract model's position at the same time, and k_p and k_d are the proportional and derivative parameters respectively. As the performance of the PD controller highly relies on the parameter tuning, the controller is trained to track a ramp signal with the slope set to 0.25 using the *Simulink Design Optimization* and got the following set of parameters described in Table 1.

Table 1 PID Parameters after Optimization

	k_p	k_d
x-axis	1.061	0.911
y-axis	1.061	0.911
z-axis	0.966	1.061

Figure 24 shows the design interface and output after the parameter optimization, where the blue lines are the position tracking error at different optimization iterations and the bold black lines are the upper and lower bounds for the tracking error. It took about 2 hours to converge the search of parameters, but this time is subject to change with different initial condition and computer performance.

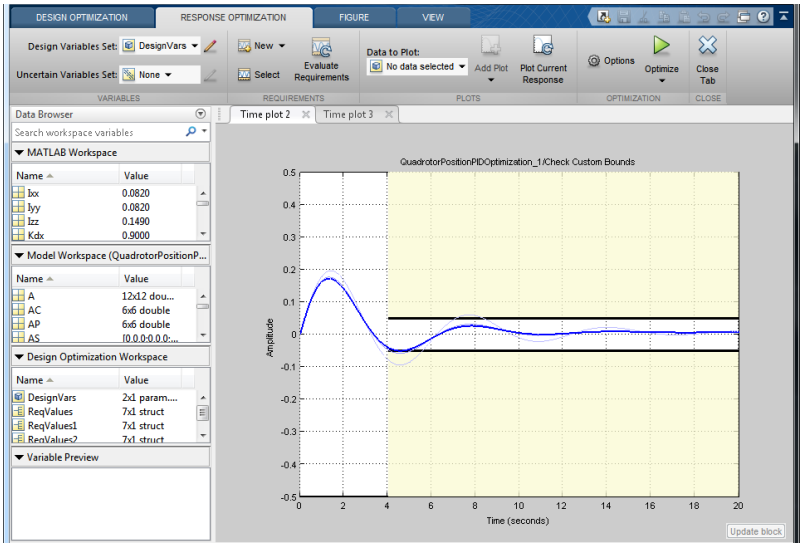


Figure 24 Output after Parameter Optimization

As it can be seen, the quadrotor could track a ramp signal with almost 0 static error and a very small dynamic error at the starting time. The simulation results of using the PD controller are shown in Figure 25Figure 32.

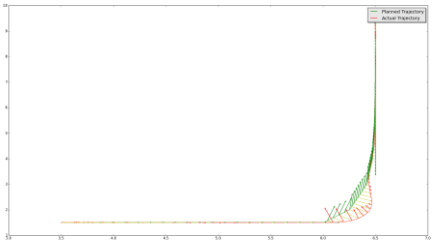


Figure 25 Trajectory on X-Y Plane at Velocity 0.4

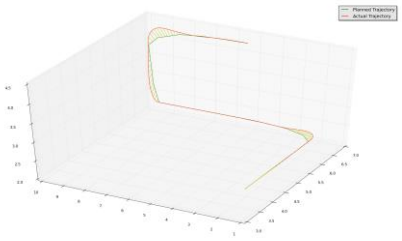


Figure 29 Trajectory with PD Controller at Velocity 0.4

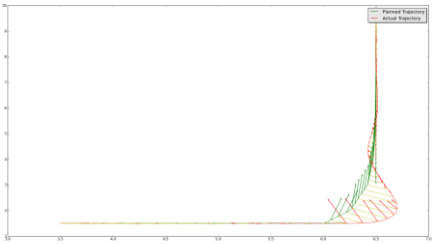


Figure 26 Trajectory on X-Y Plane at Velocity 0.6

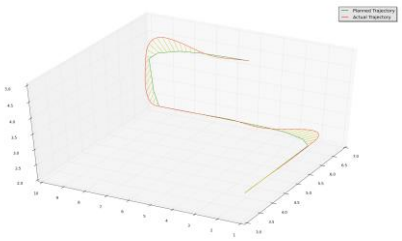


Figure 30 Trajectory with PD Controller at Velocity 0.6

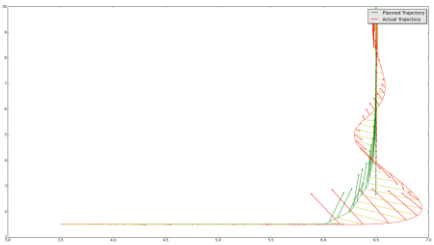


Figure 27 Trajectory on X-Y Plane at Velocity 0.8

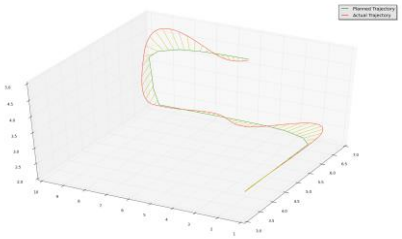


Figure 31 Trajectory with PD Controller at Velocity 0.8

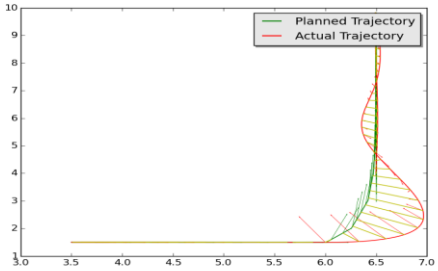


Figure 28 Trajectory on X-Y Plane at Velocity 0.9

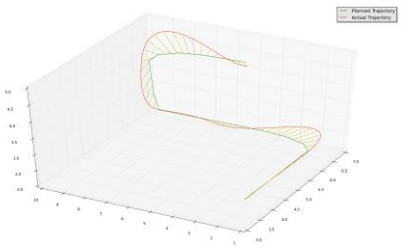


Figure 32 Trajectory with PD Controller at Velocity 0.9

The maximum velocity that can be achieved without flying out of the assigned cells is 0.9.

The simulation results have shown that the approach using the approximate simulation relations achieved better performance in terms of speed and accuracy compared to the old approach using the PD controller. Also, the approximate simulation relation can provide a tracking error bound, while the PD controller cannot. However, this cannot be formally proved with the current approximate simulation relation. Experimentally, the approximate simulation is better than the PD controller. Note however that the performances of both methods highly rely on the choice of parameters, and thus it is hard to find the optimal parameters.

Figure 33 shows that the bound set by the approximate simulation method is not violated during the execution.

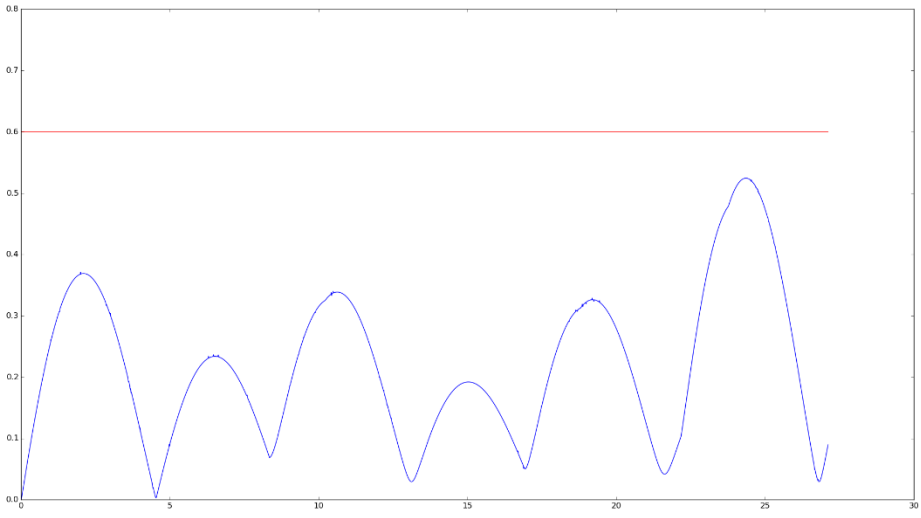


Figure 33 Tracking Error at Velocity 0.3

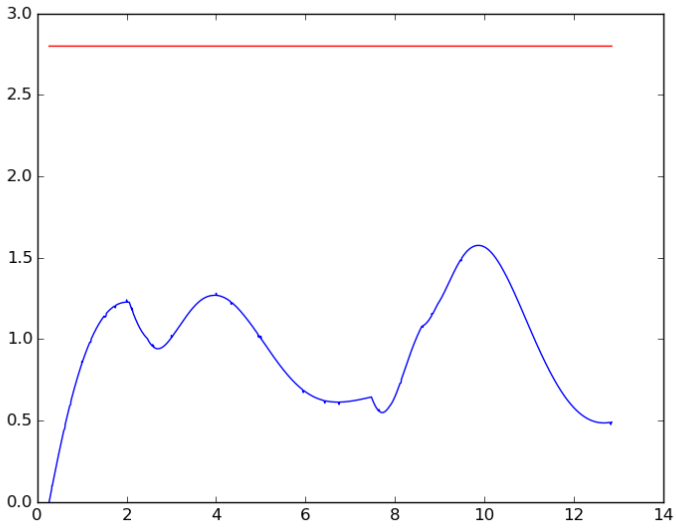


Figure 34 Tracking Error at Velocity 1.4

Figure 33 and Figure 34 are plotted with the same trajectory as the previous simulations at velocity 0.3 and 1.4 respectively. The red lines are the established bound $2v_{max}$ and blue lines are the actual tracking error between the abstract model and concrete model. The increase of the tracking error is caused by the change of directions, and after that it will tends to get back to zero. This, from another aspect, verifies the correctness of the approximate simulation relation method.

Another set of comparisons between the two methods is done to show that at the same velocity, the approach proposed in this work achieved higher accuracy and thus it could fly through more critical environments. Figure 35 and Figure 36 show the environment and the trajectories of the two methods with the same mission, which is to visit the three global requirements (the green cells) in order.

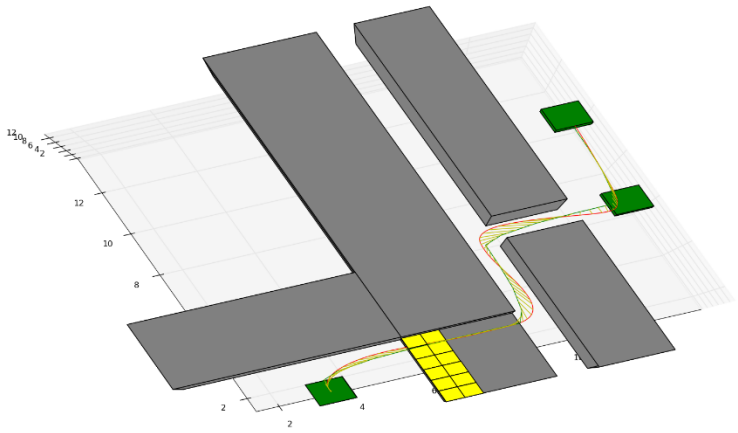


Figure 35 Trajectory with Approximate Simulation Method at Velocity 0.8

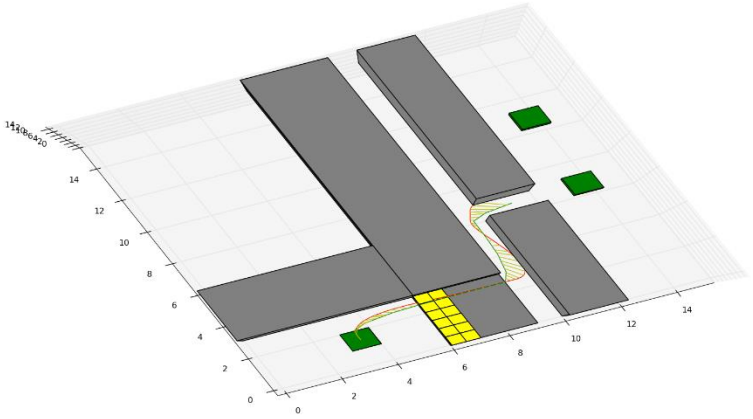


Figure 36 Trajectory with PD Controller at Velocity 0.8

As it is shown, the trajectory with the approximate simulation method successfully flew through the narrow space and reached the target. On the other hand, the trajectory with the PD controller failed to make the second turn and crashed into the obstacles (the gray cells).

6.3 System Integration Testing

The third experiment aims to verify the integration with the planner and its functionality in satisfying the high-level mission specifications. The following scenario is constructed: a quadrotor has to take and upload photos (photo, upload) of a natural disaster repeatedly in a valley and also it has to extinguish fires and assist personnel during the way if such requests are detected within its sensor range. The quadrotor cannot upload before taking a new photo and cannot take a new photo before having uploaded the old one. The environment that this test is carried on is the same as the one described in Figure 9 and the global and local mission specifications are the same as Formula 8 and Formula 9.

The following picture shows the trajectory for this mission which satisfies the defined specification.

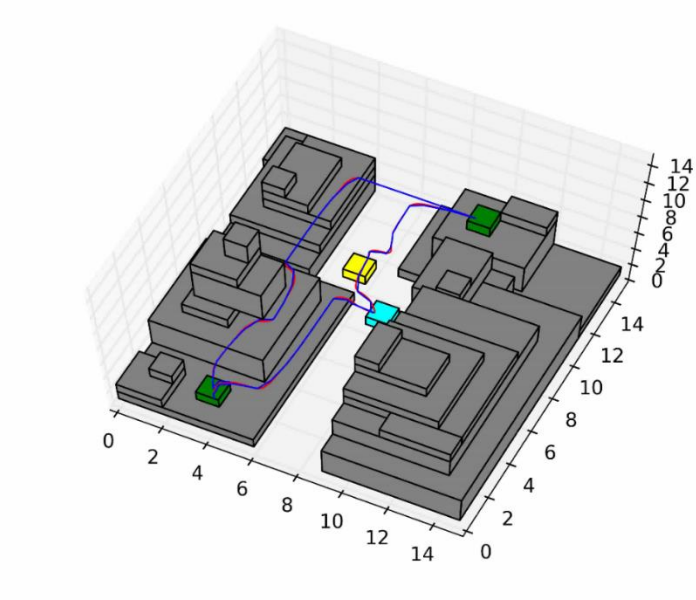


Figure 37 Environment and Mission Trajectory

In the figure above, the gray blocks are obstacles that the quadrotor should avoid. Green cells are global requests (photo or upload), the cyan and yellow cells are local dynamic requests (extinguish and assist, respectively), the blue line is the trajectory of the abstract kinematic model and the red line is the trajectory of the concrete non-linear dynamic model. The quadrotor was initially located in the “photo” cell in the right-bottom corner, and planned a trajectory to the next global request “upload”. During the flight to “upload”, it found local dynamic requests “assist” and “extinguish” and served them on the way. Then, it went on to the “upload”. After uploading the photos, it headed back to the “photo” location.

Figure 38 demonstrates its trajectories more clearly. As it can be seen, the actual trajectory(red) is very close to the abstract kinematic model’s trajectory(green). Large tracking errors only occur when changing the direction.

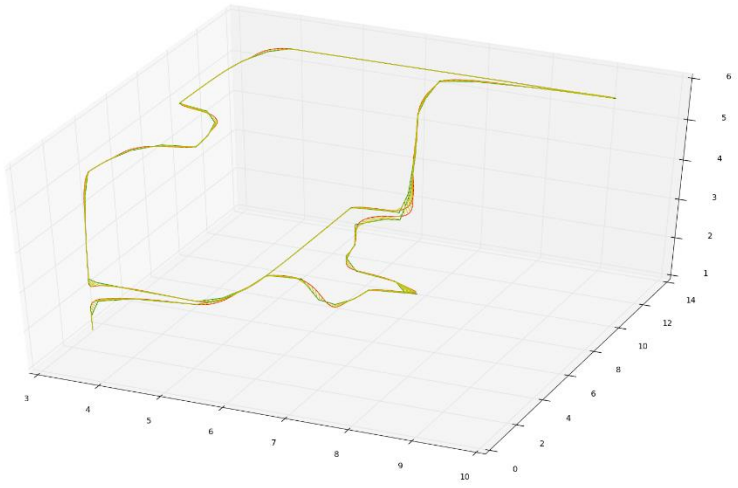


Figure 38 Mission Trajectory.

Figure 39 compares the tracking error and the bound established by the approximate simulation relation method. The bound is never violated during the whole process. Large tracking errors still happened when changing the direction, and the error tends to be stabilized at about 0.15. Also, the bound established is pretty loose as the tracking error is much smaller than the calculated bound. To further show that the bound established in [27] is experimentally not violated, a series of experiments are carried with 10 different routes including extreme conditions like spiral rising and u-turn at different velocities. Figure 40 to Figure 42 show the results of these simulations and their actual trajectories can be found in APPENDIX A .

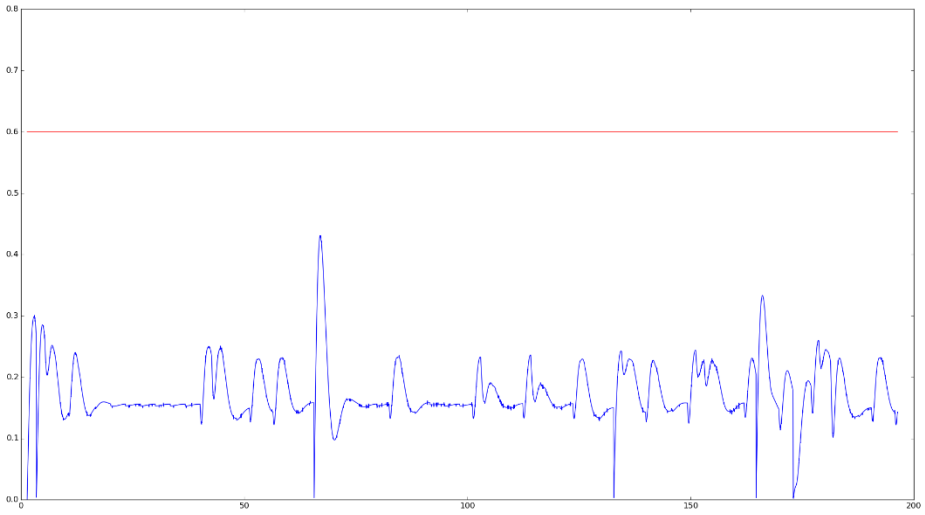


Figure 39 Tracking Error over Time.

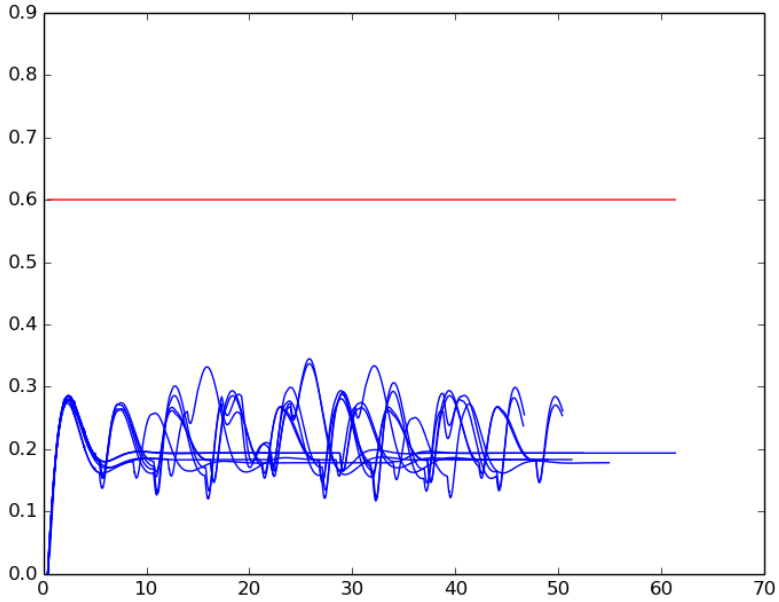


Figure 40 Tracking Errors over Time at Velocity 0.3

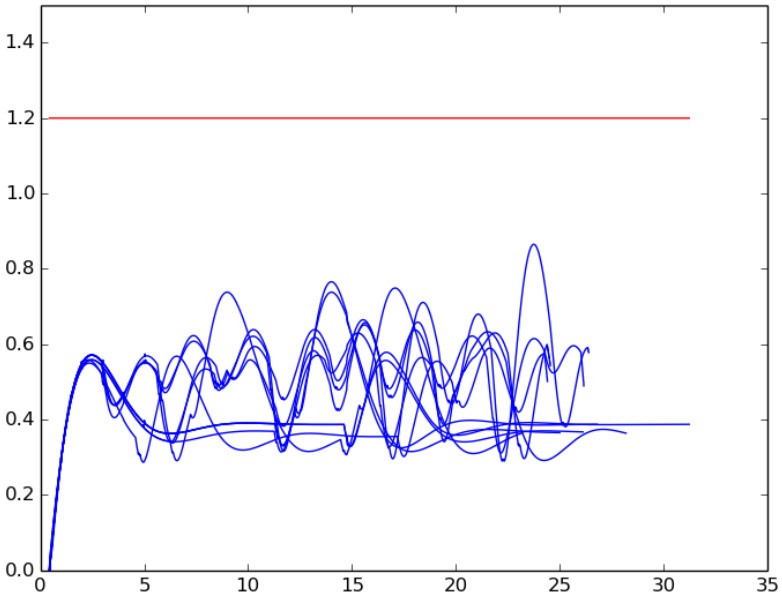


Figure 41 Tracking Errors over Time at Velocity 0.6

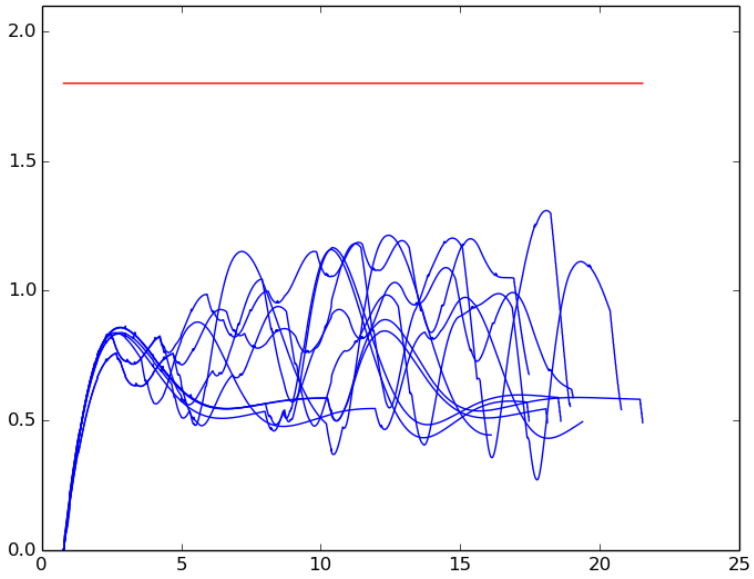


Figure 42 Tracking Errors over Time at Velocity 0.9

The last simulation experiment aims to verify the controller's ability to detect and generate local plans to fly through areas with high-resolution decomposition. The result is shown in Figure 43.

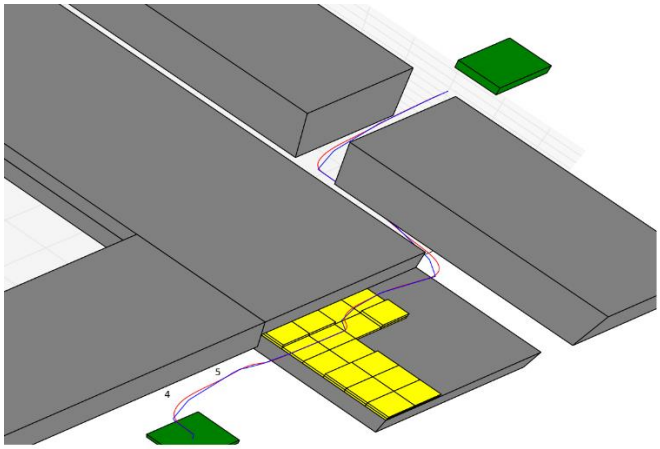


Figure 43 Quadrotor Fly Through High Resolution Area

The yellow cells are the obstacles with higher resolution. As it can be seen, the quadrotor generated a local plan and flew through it. After leaving this area, it recovered its initial environment decomposition.

CHAPTER 7

CONCLUSION

In this paper, a hierarchical hybrid controller for a dynamic quadrotor model is presented that experimentally guarantees the quadrotor's behavior satisfies a high-level mission specification. A receding horizon method is used for planning a discrete trajectory, feedback linearization is applied to quadrotor's non-linear model to achieve attitude and acceleration control, and the approximate simulation relation method is deployed for synthesis of the discrete planner and continuous controller. In addition, a local planner is also proposed to solve the problem when no local plan is found. Based on the simulation results, this approach achieved better performance in terms of speed and accuracy compared to previous approaches using a PD controller.

However, the interface used in this work is designed for a general second-order dynamic model which does not cover the quadrotor's orientation dynamics. A transformation that transfers acceleration vector to the quadrotor's orientation and an attitude controller are necessary to serve as adapters between the general second-order dynamic model and the actual non-linear model of the quadrotor. Thus, the tracking error bound described in [27] is not formally guaranteed. But, the bound is not violated based on multiple simulation results.

Future work can concentrate on the following topics:

- In [32], two feedback linearization laws are proposed to generate a linearized model of the quadrotor, which can be used to calculate interface functions with

guaranteed tracking error bound, as it is shown in [25]. The next step of this work is to utilize this approach to achieve a formal bound.

- Improve the bound between the distance of the abstract model and concrete model. As it is shown in this thesis, the bound is proportional to the maximum velocity of the abstract model, and this bound is very loose as it is shown in the simulation results. Having a tighter bound could significantly improve the controller's speed and accuracy.
- Improve the ability of resisting disturbances. In real applications, a controller cannot be described as robust if it cannot handle disturbances like wind, change of payload, and even change of physical structures. There are already several works on this topic. In [33], a framework that takes the wind and fuel level into account is presented. In [34], a method that controls a quadrotor with a cable suspended load is proposed.
- Economic physical implementation. One major reason of quadrotor's increased popularity is its reduced cost and this trend will not stop. In [35], the authors tried to use commercial available components from smartphones to build a quadrotor which achieved promising performances compared with those using professional parts. Future work should concentrate on implementing the results of this thesis on such a quadrotor.

REFERENCES

[1] "Into an Active Volcano with the Inspire 1," 17 2 2015. [Online]. Available: https://www.dji.com/newsroom/news/into-an-active-volcano-with-the-inspire-1.

[2] "Drone delivery is already here — and it works," 30 11 2015. [Online]. Available: http://www.marketwatch.com/story/drone-delivery-is-already-here-and-it-works-2015-11-30.

[3] "Canadian mounties claim first person's life saved by a police drone," 10 5 2013. [Online]. Available: http://www.theverge.com/2013/5/10/4318770/canada-draganflyer-drone-claims-first-life-saved-search-rescue.

[4] M. C. Heatherly, "Drones: The American Controversy," Journal of Strategic Security, vol. 7.4, p. 25, 2014.

[5] Fainekos, Georgios E., Hadas Kress-Gazit, and George J. Pappas, "Hybrid controllers for path planning: A temporal logic approach.," Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on. IEEE, pp. 4885-4890, 2005.

[6] Fainekos, Georgios E., Hadas Kress-Gazit, and George J. Pappas, "Temporal logic motion planning for mobile robots.," Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference, pp. 2020-2025, 2005.

[7] Ulusoy, Alphan, and Calin Belta, "Receding horizon temporal logic control in dynamic environments.," The International Journal of Robotics Research, vol. 33.12 , pp. 1593-1607, 2014.

[8] Girard, Antoine, and George J. Pappas., "Hierarchical control using approximate simulation relations."," Decision and Control, pp. 264-269, 2006.

[9] P. Corke, Robotics, Vision & Control: Fundamental Algorithms in Matlab, Springer, 2011.

[10] S. Sastry, Nonlinear Systems: Analysis, Stability, and Control, Springer Science & Business Media, 2013.

[11] H. Voos, "Nonlinear Control of a Quadrotor Micro-UAV using Feedback-Linearization," Mechatronics, 2009. ICM 2009. IEEE International Conference, pp. 1-6, 2009.

[12] "PID Control Design with Control System Toolbox," Mathworks, [Online]. Available: http://www.mathworks.com/videos/pid-control-design-with-control-system-toolbox-68748.html.

[13] A. Alur, T. Henzinger, E. Sontag, Hybrid Systems III: Verification and Control, Springer-Verlag, 1996.

[14] T. Henzinger, S. Sastry, Eds., Hybrid Systems: Computation and Control, Springer, 1998.

[15] F. Vaandrager, J. van Schuppen, Eds., Hybrid Systems: Computation and Control, Springer, 1999.

[16] N. Lynch, B. Krogh, Eds., Hybrid Systems: Computation and Control, Springer, 2000.

[17] Belta, Calin, and Luc CGJM Habets., "Constructing decidable hybrid systems with velocity bounds," Decision and Control, 2004. CDC. 43rd IEEE Conference on., vol. 1, pp. 467-472, 2004.

[18] D. M. R. Park, Concurrency and automata on infinite sequences, Springer-Verge, 1980.

[19] R. Miller, Communication and Concurrency, Pretice Hall, 1989.

[20] C. Belta, L. Habets, "Control of rectangular multi-affine systems with applications to genetic networks," IEEE Transactions on Automatic Control, 2004.

[21] E. M. Clarke, O. Grumberg, D. Peled, Model checking, MIT press, 1999.

[22] P. Gastin, D Oddoux, "Fast LTL to Buchi automata translation," Processing of the 13th international conference on computer aided verification., pp. 53-65, 2001.

[23] A. Girard, J. Pappas, "Approximation Metrics for Discrete and Continuous Systems," Automatic Control, IEEE Transactions, vol. 52, no. 5, pp. 782-798, 2007.

[24] A. Girard, G. J. Pappas, "Hierarchical control system design using approximate simulation.," Automatica, pp. 3727-3732, 2009.

[25] A. Girard, G.J. Pappas, "Approximate hierarchies of linear control systems.," Decision and Control, 2007 46th IEEE Conference., pp. 3727-3732, 2007.

[26] A. Girard, G. J. Pappas, "Approximate bisimulation relations for constrained linear systems.," Automatica, pp. 1307-1317, 2007.

[27] Fainekos, G. E., Girard, A., Kress-Gazit, H., & Pappas, G. J., "Temporal logic motion planning for dynamic robots," Automatica, vol. 45(2), pp. 343-352, 2009.

[28] M. Reynolds, "Continuous Temporal Models.," AI 2001: Advances in Artificial Intelligence, pp. 414-425, 2011.

[29] W. S. Levine, Ed. Piscataway, "PID Control," in The Control Handbook, IEEE Press, 1996, p. 198–209.

[30] Ang, Kiam Heong, Gregory Chong, Yun Li, "PID control system analysis, design, and technology," Control Systems Technology, pp. 559-576, 2005.

[31] "Simulink Design Optimization," Mathworks, [Online]. Available: http://www.mathworks.com/products/sl-design-optimization/.

[32] Bonna, R., and J. F. Camino., "Trajectory Tracking Control of a Quadrotor Using Feedback Linearization".

[33] C. Yoo, R. Fitch, S. Sukkarieh, "Online task planning and control for fuel-constrained aerial robots in wind fields," The International Journal of Robotics Research, 2015.

[34] P. Cruz, M. Oishi, R. Fierro, "Lift of a cable-suspended load by a quadrotor: A hybrid system approach.," American Control Conference(ACC), pp. 1887-18992, 2015.

[35] G. Loianno, G. Cross, C. Qu, Y. Mulgaonkar, J. Hesch, V. Kumar, "Flying Smartphones: Automated Flight Enabled by Consumer Electronics.," 2015.

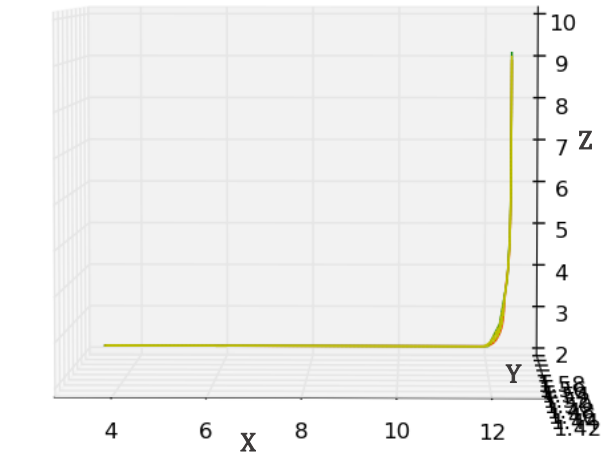
[36] Michael, Nathan; Mellinger, D.; Lindsey, Q.; Kumar, V., "The GRASP Multiple Micro-UAV Testbed,," Robotics & Automation Magazine, IEEE , vol. 17, pp. 56,65, 2010.

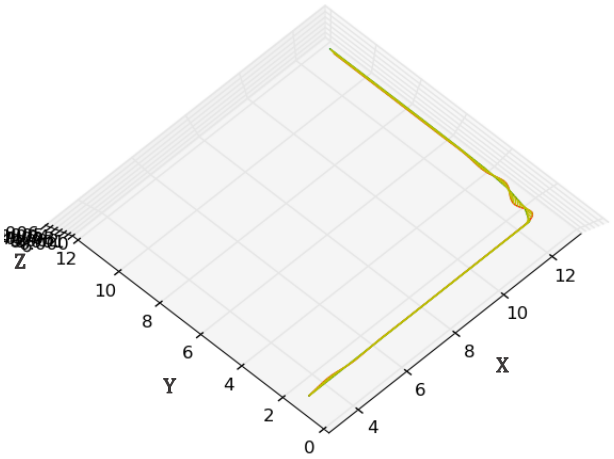
APPENDIX A

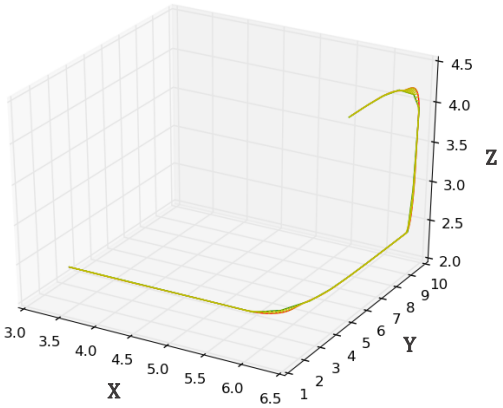
SIMULATIONS FOR TRACKING ERROR ANALYSIS

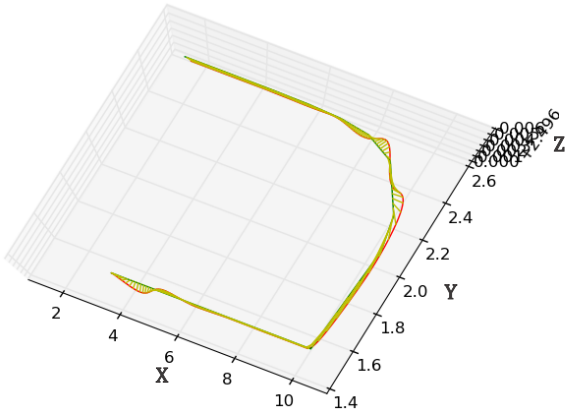
In Section 6.3, Figures Figure 40 to Figure 42 shows the tracking error over time of having a quadrotor fly through 10 different assigned trajectories with three different velocities. To make these simulation results more convincing, this section presents the actual trajectories of these simulations.

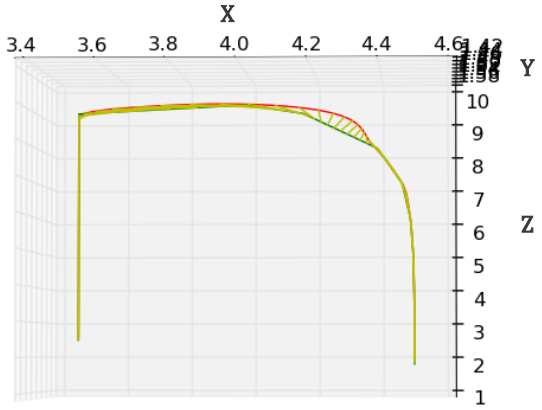
The following 10 figures shows the 10 actual trajectories at velocity 0.3, all trajectories consists of 1*1*1 cubic cells starting at (3, 1, 2):

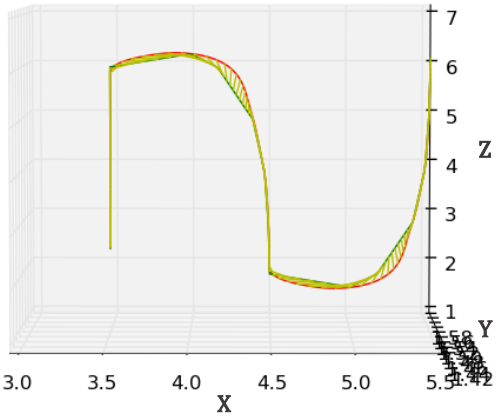


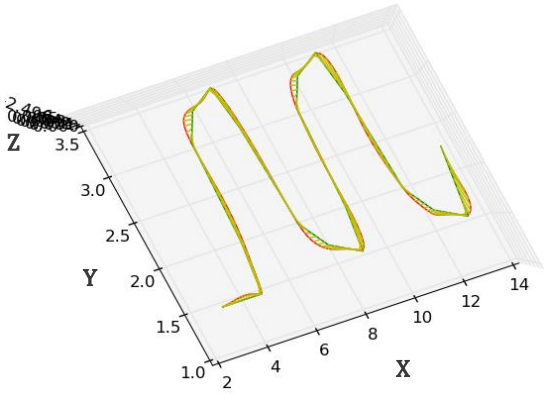


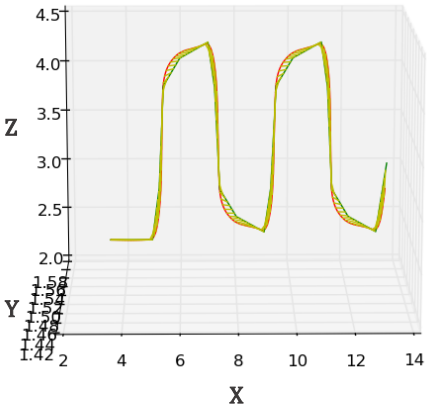

4.5
4.0
3.5
Z
3.0
2.5
2.0
10
9
8
7
6
5
4
3
2
1
Y
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
X

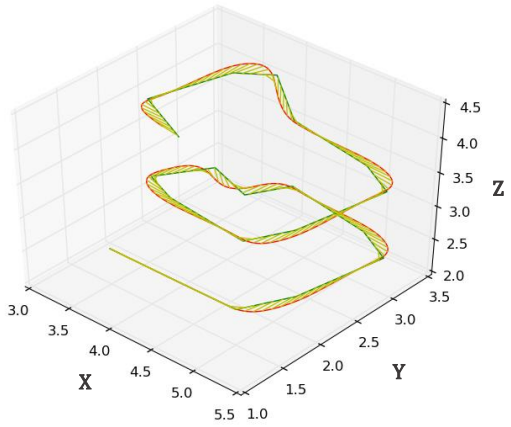

Z
2.6
2.4
2.2
2.0
Y
1.8
1.6
1.4
2
4
6
X
8
10

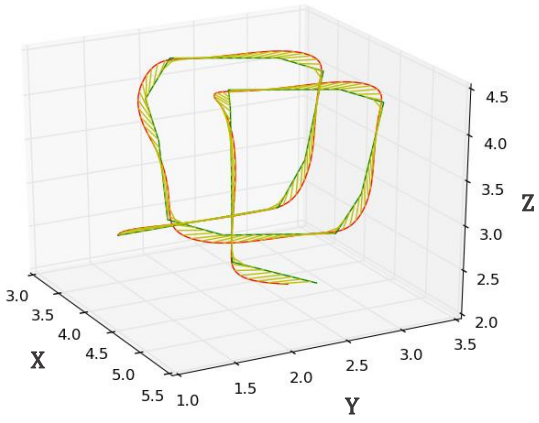

X
3.4
3.6
3.8
4.0
4.2
4.4
4.6
Y
10
9
8
7
Z
6
5
4
3
2
1


7
6
5
4
3
2
1
Z
Y
3.0
3.5
4.0
4.5
5.0
X


Z
3.5
3.0
2.5
Y
2.0
1.5
1.0
2
4
6
8
10
12
14
X


4.5
4.0
3.5
Z
3.0
2.5
2.0
Y
2
4
6
8
10
12
14
X


4.5
4.0
3.5
3.0
2.5
2.0
Z
3.5
3.0
2.5
2.0
1.5
1.0
Y
3.0
3.5
4.0
4.5
5.0
5.5
X


4.5
4.0
3.5
3.0
2.5
2.0
Z
3.5
3.0
2.5
2.0
1.5
1.0
Y
3.0
3.5
4.0
4.5
5.0
5.5
X