DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

LTL Motion Planning with Collision Avoidance for A Team of Quadrotors

ZIWEI XU

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING

LTL Motion Planning with Collision Avoidance for A Team of Quadrotors

Master Thesis

Ziwei Xu

Examiner: Dr. D. V. Dimarogonas
Supervisor:Christos Verginis

July 5, 2016

Abstract

Linear Temporal Logic (LTL), as one of the temporal logic, can generate a fully automated correct-by-design controller synthesis approach for single or multiple autonomous vehicles, under much more complex missions than the traditional point-to-point navigation.
In this master thesis, a framework which combines model- checking-based robot motion planning with action planning is proposed based on LTL formulas. The specifications implicitly require both sequential regions for multi-agent to visit and the desired actions to perform at these regions while avoiding collision with each other and fixed obstacles. The high level motion and task planning and low level navigation function based collision avoidance controller are verified by nontrivial simulation and implementation on real quadcopter in Smart Mobility Lab.

Key Words: Finite transition system(FTS), Linear Temporal Logic(LTL) formula, Büchi automaton(BA), Optimal path, Robotic operating system(ROS)

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, with the development of technology, more and more autonomous robot such as unmanned aerial vehicle(UAV), domestic robots, autonomous cars appear in our daily life. One of the most important feature of autonomous robots is that they are expected to comprehend user's command like go to certain place or finish certain task without or with as less human intervention as possible.

All the robotics issues we mentioned above related to two fundamental problems: task and motion planning. Nowadays, temporal-logical-based task and motion planing has gain more and more attention. Linear Temporal Logic (LTL), as one of the temporal logic, can generate a fully automated correct-by-design controller synthesis approach for single or multiple autonomous vehicles, under much more complex missions than the traditional point-to-point navigation [1].

Therefore, we will focus on the LTL based task and motion planing for multi-agent in this thesis. The high level motion and task planning and low level collision avoidance controller is verified by nontrivial simulation and implementation on real quadcopter in Smart Mobility Lab.

1.2 Related Work

Temporal logical model checking is an automatic verification technique for the finite state systems which was developed independently by Clarke and Emerson[2] and by Queille and Sifakis[3] in early 1980's.

Linear Temporal Logic (LTL) specifications, as one of the temporal logic model checking method, was first introduced by Amir Pnueli[4] in 1977's. In this paper, the formal system which is particular suitable for reasoning about concurrent programs was first proposed. A general LTL model checking process which satisfied the automata-theoretic model checking criterion[5]

can be described as below :

1. Define the transition system model M containing traces over the set of atomic propositions.
2. Let specification φ be a formula over the set of atomic propositions.
3. Check that $M \models \varphi$:
   - Translate the specification $\neg\varphi$ into a Büchi automaton $A_{\neg\varphi}$ and develop the synchronized product of $A_{\neg\varphi}$ and model M as $A_{M,\neg\varphi}$
   - check whether $A_{M,\neg\varphi}$ is empty. Namely, check if there is a trace that is accepted by $A_{M,\neg\varphi}$.
     - If such trace exists, return FALSE and regards this trace as a counter example.
     - If no such trace exists, return TRUE.

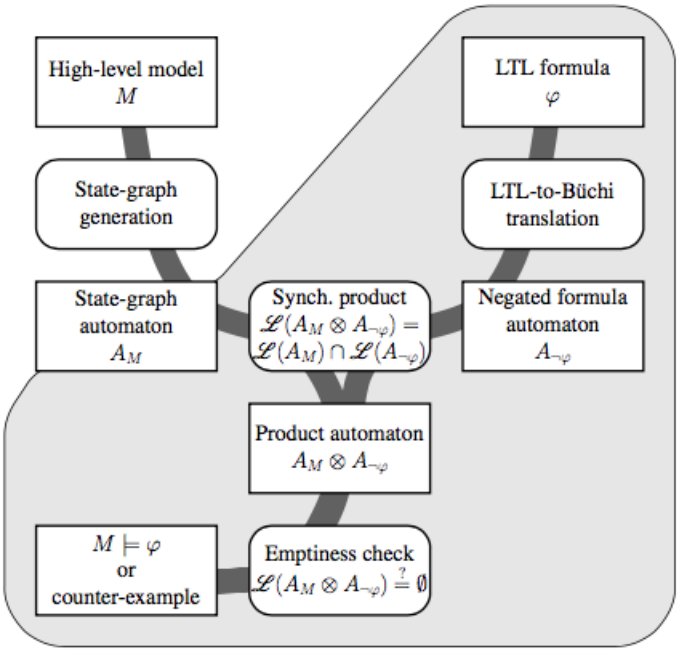

Figure 1.1 The automata-theoretic approach to LTL model checking

In which the most common used algorithm for $A_{M,\neg\varphi}$ nonemptiness check is Depth-first search (DFS) and the algorithm complexity is linear in sum of vertex and transition.

We can classify the LTL model checker into two types: explicit model checker and symbolic model checker. The most commonly used explicit LTL model checker includes SPIN[6] and SPOT[7]. Each model checker has its own characteristics, for example, SPIN verification models put emphasis on proving the correctness of process interactions, and attempt to abstract as much as possible from internal sequential computations, while the SPOT which relies on Transition-based Generalized Büchi Automata (TGBA) can be combined and interfaced with third party tools to build a model checker.

However, all of them construct the explicit state space model and develop the synchronized product $A_{M,\neg\varphi}$ such that $\mathscr{L}(A_{M,\neg\varphi}) = \mathscr{L}(M) \cap \mathscr{L}(A_{\neg\varphi})$ and $|A_{M,\neg\varphi}| = \mathscr{O}|M| \cdot |A_{\neg\varphi}|$, where $|\cdot|$ denotes the size of the automaton which concern about both number of state and the transition between different state. However, when it comes to verify the nonemptiness of $A_{M,\neg\varphi}$, the computation time required to find the related trace will grow exponentially with the number of system components which is one of the major practical limitation of model checking. Besides, the state explosion problem is inevitable in the worst case. Fortunately, a lot of techniques have been developed to mitigate this kind of problem for certain types of systems over last several years. For instance, partial order reduction which use asynchronous composition of processes to analysis the independence of state in the system[8], and analyzing simulation and equivalence of programs[9] and the use of rigorous argument [10].

On the other hand, the symbolic model checkers, such as CadenceSMV[11], NuSMV [12], and VIS[13] which represent the mode of system symbolicly also address the state explosion problem. Theoretically, all symbolic model checkers use all most the same symbolic translation for LTL specifications which is described in [14] though there are some slight differences on further optimization. The method they used to analyze the state space is called binary decision diagrams (BDDs)[15]. BDD was introduced by McMillan in 1992[16] and it provides a compress representation of state space by using Boolean formulas so that there is no explicit state graph and we can use syntactically small equations to represent large sets of states.

Therefore, the main difference between explicit model checker and symbolic model checker is that latter one represents the state as the result of a logical equation and reduce the memory space significantly.

1.3 Thesis Outline

In this section I will describe the outline and my main contribution to each chapter briefly.

Chapter 2	Graph theory and dynamic modeling of quadrator with manipulator are introduced
Chapter 3	LTL based motion planning for multi-agent case as well as high level collision avoidance is introduced
Chapter 4	Local independent task for multi-agent case is discussed
Chapter 5	Both nontrivial simulation and real implementation in ROS are presented
Chapter 6	Conclusion and further improvement

Chapter 2

Background

2.1 Graph Theory

A graph consists of several nodes where some pair of node are connected by links. We usually called the nodes as vertices and the links between vertices as edges. Normally, a graph can be represented by formula G = (V, E) comprising a set V of vertices together with a set E of edges.

The graph can be classified into several types: undirected graph, directed graph, weighted graph and so on.

- the undirected graph is a graph in which edges have no orientation. The edge (a, b) is identical to the edge (b, a)
- A directed graph or digraph is a graph in which edges have orientations. That is, we need to distinguish edge (a, b) and edge (b, a)
- A weighted graph is a graph in which a number (the weight) is assigned to each edge. Both undirected and undirected graph can become a weighted graph

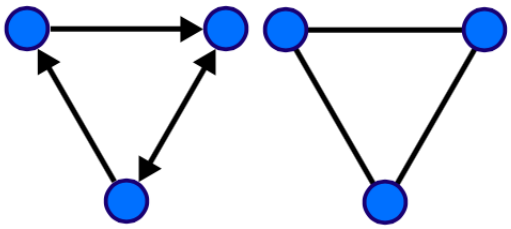

Figure 2.1 Directed graph(left) and undirected graph(right)

The two basic path search algorithm in graph are Depth-first search (DFS)[17] and Breadth-first search (BFS)[18]. DFS starts at the arbitrary

node in the graph as root and explores as far as possible along each branch before backtracking, while BFS explores the neighbor nodes first, before moving to the next level neighbors. The algorithm in detail are described in Algorithm1 and Algorithm2.

Algorithm 1 Depth-first-search

Require: Graph $G = (V, E)$, root node s

function DFS (G)
    for each vertex $u \in G.V$ **do**
        $u.color = WHITE$
        $u.\pi = NIL$
    end for
    $time = 0$
    for each vertex $u \in G.V$ **do**
        if $u.color == WHITE$ **then**
            $DFS - VISIT(G.u)$
        end if
    end for
end function
function DFS-VISIT(G, u)
    $time = time + 1$
    $u.d = time$
    $u.color = GRAY$
    for each $v \in G.Adj[u]$ **do**
        if $v.color == WHITE$ **then**
            $v.\pi = u$
            $DFS - VISIT(G, v)$
        end if
    end for
    $u.color = BLACK$
    $time = time + 1$
    $u.f = time$
end function

Algorithm 2 Breadth-first-search

Require: Graph $G = (V, E)$, root node s

function BFS (G, s)
  for each vertex $u \in G.V - \{s\}$ **do**
    $u.color = WHITE$
    $u.d = \infty$
    $u.\pi = NIL$
  end for
  $s.color = GRAY$
  $s.d = 0$
  $s.\pi = NIL$
  $Q = Empty$
  $ENQUEUE(Q, s)$
  while $Q! = Empty$ **do**
    $u = DEQUEUE(Q)$
    for each $v \in G.Adj[u]$ **do**
      if $v.color == WHITE$ **then**
        $v.color = GRAY$
        $v.d = u.d + 1$
        $v.\pi = u$
        $ENQUEUE(Q, v)$
      end if
    end for
    $u.color = BLACK$
  end while
end function

2.2 Modeling of Quadrotor with Manipulator

This section has derived the dynamic model of the UAV with manipulator in a symbolic matrix by using Euler-Lagrangian formalism which has laid a solid foundation for further study eg: developing controller for picking and dropping. The whole system for UAV with manipulator with respect to reference frame is illustrated in figure 2.2

2.2.1 Kinematic Model

We define the yaw-pitch-roll Euler Angles w.r.t the fixed frame of earth inertia frame as $\boldsymbol{\eta} = [\varphi \quad \theta \quad \psi]$ Then we can derive the rotation matrix:

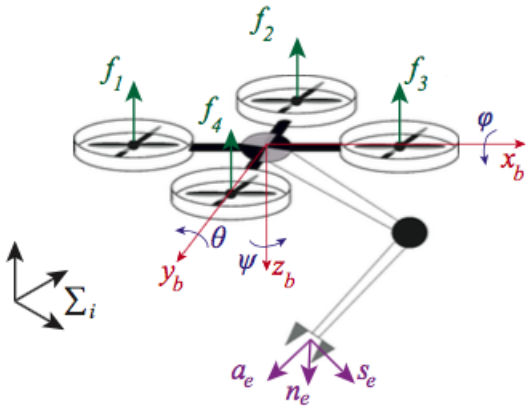

Figure 2.2 Cartesian coordinate system for UAV with manipulator [19]

$$\begin{aligned}\mathbf{R}_B^I &= \mathbf{R}_X(\varphi)\mathbf{R}_Y(\theta)\mathbf{R}_Z(\psi) \\ &= \begin{bmatrix} c\theta c\psi & -c\theta s\psi & s\theta \\ c\varphi s\psi + c\psi s\varphi s\theta & c\varphi c\varphi - s\theta s\psi s\varphi & -s\varphi c\theta \\ s\varphi s\psi - c\psi c\varphi s\theta & s\varphi c\psi + c\theta s\psi s\varphi & c\varphi c\theta \end{bmatrix}\end{aligned} \tag{2.1}$$

where $\mathbf{R}_B^I$ denotes the rotation matrix $\mathbf{R}$ of body frame B w.r.t inertia frame I.

If we denotes the absolute position of the UAV, i.e. the position of body frame B w.r.t the inertia frame I expressed in inertia frame I as $\boldsymbol{P}_{B/I}^I = [x \quad y \quad z]$, then the absolute linear velocity of the UAV can be expressed as:

$$\dot{\boldsymbol{P}_{B/I}^I} = \mathbf{R}_B^I \dot{\boldsymbol{P}_{B/I}^B} \tag{2.2}$$

In view of the orthogonality of $\mathbf{R}$, one has the relation

$$\mathbf{R}\mathbf{R}^{\mathbf{T}} = \mathbf{I} \tag{2.3}$$

Differential it with respect to time and set

$$\begin{aligned}S(\boldsymbol{\omega}_{B/I}^I) &= \dot{\mathbf{R}_B^I}\,(\mathbf{R}_B^I)^T \\ &= \begin{bmatrix} 0 & -\dot{\theta}\,s\varphi - \dot{\psi}\,c\varphi c\theta & \dot{\theta}\,c\varphi - \dot{\psi}\,c\theta s\varphi \\ \dot{\theta}\,s\varphi + \dot{\psi}\,c\varphi c\theta & 0 & -\dot{\varphi} - \dot{\psi}\,s\theta \\ \dot{\psi}\,c\theta s\varphi - \dot{\theta}\,c\varphi & \dot{\varphi} + \dot{\psi}\,s\theta & 0 \end{bmatrix}\end{aligned} \tag{2.4}$$

where $\boldsymbol{\omega}_{B/I}^I$ denotes angular velocity of body frame B w.r.t inertia frame$I$ in inertia frame I and $ck(sk)$ denotes $cosk(sink)$.

Therefore,

$$\boldsymbol{\omega}_{B/I}^{I} = \begin{bmatrix} \dot{\varphi} + \dot{\psi}\, s\theta \\ \dot{\theta}\, c\varphi - \dot{\psi}\, c\theta s\varphi \\ \dot{\theta}\, s\varphi + \dot{\psi}\, c\varphi c\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 & s\theta \\ 0 & c\varphi & -c\theta s\varphi \\ 0 & s\varphi & c\varphi c\theta \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.5}$$

Set

$$\begin{bmatrix} 1 & 0 & s\theta \\ 0 & c\varphi & -c\theta s\varphi \\ 0 & s\varphi & c\varphi c\theta \end{bmatrix} = \boldsymbol{\Gamma}_{\eta} \tag{2.6}$$

$$\boldsymbol{\omega}_{B/I}^{I} = \boldsymbol{\Gamma}_{\eta} \dot{\boldsymbol{\eta}}_{B} \tag{2.7}$$

Also,

$$\boldsymbol{\omega}_{B/I}^{B} = (\boldsymbol{R}_{B}^{I})^{T} \boldsymbol{\omega}_{B/I}^{I} = (\boldsymbol{R}_{B}^{I})^{T} \boldsymbol{\Gamma}_{\boldsymbol{\eta}}\, \dot{\boldsymbol{\eta}} = \boldsymbol{Q}_{B}\, \dot{\boldsymbol{\eta}} \tag{2.8}$$

Taking the representation singularities of above matrix into consideration, then $\theta \neq \pm k\frac{\pi}{2}$, with $k = 1, 3, 5...$

Therefore, according to direct kinematics, any vector in body frame B can be expressed in inertia frame I by using the following matrix:

$$\boldsymbol{A}_{B} = \begin{bmatrix} \boldsymbol{R}_{B}^{I} & \boldsymbol{P}_{B/I}^{I} \\ \boldsymbol{0}^{T} & 1 \end{bmatrix} \tag{2.9}$$

Noticing that in our case a manipulator with n rigid links is attached to the aerial vehicle. In order to complete the series of task i.e. grab, hold, move, we need to know the position of end-effector. It was previously illustrated that the pose of a point in body frame w.r.t the inertia frame is described by the position vector of the origin and the vectors of body frame attached to the inertia frame. Hence, let us start with more general case. By denoting the the position of the center of mass of the link i with $i = 1, 2, ..., n$ w.r.t the inertia frame I as $\boldsymbol{P}_{li/I}^{I}$, the following relationship holds

$$\boldsymbol{P}_{li/I}^{I} = \boldsymbol{P}_{B/I}^{I} + \boldsymbol{R}_{B}^{I} \boldsymbol{P}_{li/I}^{B} \tag{2.10}$$

Similarly, we can derive the the angular velocity of the center of mass of the link i w.r.t the inertia frame I as

$$\boldsymbol{\omega}_{li/I}^{I} = \boldsymbol{\omega}_{B/I}^{I} + \boldsymbol{R}_{B}^{I} \boldsymbol{\omega}_{li/I}^{B} \tag{2.11}$$

Considering the manipulator geometric Jacobian, we have

$$\dot{\boldsymbol{P}}_{li/B}^{B} = \boldsymbol{J}_{P}^{(li)}\, \dot{\boldsymbol{q}} \qquad \boldsymbol{\omega}_{li/B}^{B} = \boldsymbol{J}_{o}^{(li)}\, \dot{\boldsymbol{q}} \tag{2.12}$$

By taking (2.11) and (2.12) into account and differential (2.10) w.r.t time yield

$$\begin{aligned}\dot{\boldsymbol{P}}^I_{li/I} &= \dot{\boldsymbol{P}}^I_{B/I} + \dot{\boldsymbol{R}}^I_B\, \boldsymbol{P}^B_{li/I} + \boldsymbol{R}^I_B\, \dot{\boldsymbol{P}}^B_{li/I} \\ &= \dot{\boldsymbol{P}}^I_{B/I} + \boldsymbol{S}(\boldsymbol{\omega}^I_{B/I})\boldsymbol{R}^I_B\boldsymbol{P}^B_{li/I} + \boldsymbol{R}^I_B\boldsymbol{J}^{(li)}_P\,\dot{\boldsymbol{q}} \\ &= \dot{\boldsymbol{P}}^I_{B/I} - \boldsymbol{S}(\boldsymbol{P}^I_{li/I})\boldsymbol{\omega}^I_{B/I} + \boldsymbol{R}^I_B\boldsymbol{J}^{(li)}_P\,\dot{\boldsymbol{q}} \end{aligned} \tag{2.13}$$

and

$$\boldsymbol{\omega}^I_{li/I} = \boldsymbol{\omega}^I_{B/I} + \boldsymbol{R}^I_B\boldsymbol{J}^{(li)}_o\,\dot{\boldsymbol{q}} \tag{2.14}$$

Now, let us focus on a more specific problem: the linear velocity and angular velocity of end-effector. Based on what we derived above, we can easily get this problem done:

Let $\boldsymbol{P}^I_{E/I}$ be the position of end-effector w.r.t the inertia frame I and $\boldsymbol{\omega}^I_{li/I}$ be the angular velocity of end-effector w.r.t the inertia frame I. we can get

$$\begin{aligned}\dot{\boldsymbol{P}}^I_{E/I} &= \dot{\boldsymbol{P}}^I_{B/I} - \boldsymbol{S}(\boldsymbol{P}^I_{E/I})\boldsymbol{\omega}^I_{B/I} + \boldsymbol{R}^I_B\boldsymbol{J}^{(E)}_P\,\dot{\boldsymbol{q}} \\ &= \dot{\boldsymbol{P}}^I_{B/I} - \boldsymbol{S}(\boldsymbol{P}^I_{E/I})\boldsymbol{\Gamma}_\eta\dot{\boldsymbol{\eta}}_B + \boldsymbol{R}^I_B\boldsymbol{J}^{(E)}_P\,\dot{\boldsymbol{q}} \end{aligned} \tag{2.15}$$

and

$$\boldsymbol{\omega}^I_{E/I} = \boldsymbol{\omega}^I_{B/I} + \boldsymbol{R}^I_B\boldsymbol{J}^{(E)}_o\,\dot{\boldsymbol{q}} = \boldsymbol{\Gamma}_\eta\dot{\boldsymbol{\eta}}_B + \boldsymbol{R}^I_B\boldsymbol{J}^{(E)}_o\,\dot{\boldsymbol{q}} \tag{2.16}$$

If we define the generalized joints vector as $\boldsymbol{\xi} = \begin{bmatrix}\boldsymbol{P}^I_{B/I}{}^T & \boldsymbol{\eta}^T_B & \boldsymbol{q}^T\end{bmatrix}^T$ which is $n_\zeta = 6 + n$ dimension. Then

$$\dot{\boldsymbol{P}}^I_{E/I} = \begin{bmatrix}\boldsymbol{I}_3 & -\boldsymbol{S}(\boldsymbol{P}^I_{E/I})\boldsymbol{\Gamma}_\eta & \boldsymbol{R}^I_B\boldsymbol{J}^{(E)}_P\end{bmatrix}\dot{\boldsymbol{\xi}} = \boldsymbol{J}_P(q)\,\dot{\boldsymbol{\xi}} \tag{2.17}$$

$$\boldsymbol{\omega}^I_{E/I} = \begin{bmatrix}\boldsymbol{0}_3 & \boldsymbol{\Gamma}_\eta & \boldsymbol{R}^I_B\boldsymbol{J}^{(E)}_o\end{bmatrix}\dot{\boldsymbol{\xi}} = \boldsymbol{J}_o(q)\,\dot{\boldsymbol{\xi}} \tag{2.18}$$

Therefore, the configuration space of end-effector can be expressed as

$$\dot{\boldsymbol{\upsilon}}_E = \begin{bmatrix}\boldsymbol{J}_P(\boldsymbol{q}) \\ \boldsymbol{J}_P(\boldsymbol{o})\end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\,\dot{\boldsymbol{\xi}} \tag{2.19}$$

where $\boldsymbol{J}(\boldsymbol{q})$ is $(6 \times n_\zeta)$ Jacobian matrix.

2.2.2 Dynamic Model

The dynamical modeling of UAV with manipulator is based on the method *Lagrange formulation* with which the equations of motion can be derived in a systematic way independently of the reference coordinate frame.

Thinking of the total *kinetic energy* and *potential energy* of the system $\mathcal{T}$ and $\mathcal{U}$, then *Lagrangian* of the mechanical system can be defined as a function of the generalized joints vector $\boldsymbol{\xi}$:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{2.20}$$

The Lagrange equations are expressed by:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\xi}_i} - \frac{\partial \mathcal{L}}{\partial \xi_i} = \tau_i \qquad i = 1, ..., n_\xi \tag{2.21}$$

where τ_i is the generalized force associated with the generalized joints i.

Based on the kinematics part, we can derive the kinetic energy $\mathcal{T}_b$ of UAV and the kinetic energy $\mathcal{T}_{li}$ of manipulator link i.

$$\begin{aligned}\mathcal{T}_b &= \frac{1}{2} m_B (\boldsymbol{P}^{\dot{I}}_{B/I})^T \boldsymbol{P}^{\dot{I}}_{B/I} + \frac{1}{2} (\boldsymbol{\omega}^I_{B/I})^T \boldsymbol{I}^I_B \boldsymbol{\omega}^I_{B/I} \\ &= \frac{1}{2} m_B (\boldsymbol{P}^{\dot{I}}_{B/I})^T \boldsymbol{P}^{\dot{I}}_{B/I} + \frac{1}{2} \dot{\boldsymbol{\eta}_B}^T \boldsymbol{Q}_B{}^T \boldsymbol{I}^B_B \boldsymbol{Q}_B \dot{\boldsymbol{\eta}_B}\end{aligned} \tag{2.22}$$

$$\begin{aligned}\mathcal{T}_{li} &= \frac{1}{2} m_{li} (\boldsymbol{P}^{\dot{I}}_{li/I})^T \boldsymbol{P}^{\dot{I}}_{li/I} + \frac{1}{2} (\boldsymbol{\omega}^I_{li/I})^T \boldsymbol{I}^I_{li} \boldsymbol{\omega}^I_{li/I} \\ &= \frac{1}{2} m_{li} (\boldsymbol{P}^{\dot{I}}_{li/I})^T \boldsymbol{P}^{\dot{I}}_{li/I} + \frac{1}{2} (\boldsymbol{\omega}^I_{li/I})^T \boldsymbol{R}^I_B \boldsymbol{R}^B_{li} \boldsymbol{I}^{li}_{li} \boldsymbol{R}^{li}_B \boldsymbol{R}^B_I \boldsymbol{\omega}^I_{li/I}\end{aligned} \tag{2.23}$$

where $\boldsymbol{I}^I_B$ and $\boldsymbol{I}^I_{li}$ represent the inertia matrix of AUV and link i w.r.t inertia frame respectively.

Therefore, according to equation(2.13)(2.14)(2.22) and(2.23), the total *kinetic energy* can be written as:

$$\mathcal{T} = \mathcal{T}_b + \sum_{i=1}^{n} \mathcal{T}_{li} = \frac{1}{2} \dot{\boldsymbol{\xi}}^T \boldsymbol{B}(\boldsymbol{\xi}) \dot{\boldsymbol{\xi}} \tag{2.24}$$

where $\boldsymbol{B}(\boldsymbol{\xi})$ is $(n_\xi \times n_\xi)$ inertia matrix which is:

- *symmetric*
- *positive definite*
- *configuration-dependent*

The component of $\boldsymbol{B(\xi)}$ can be derived as:

$$\boldsymbol{B}_{11} = (m_B + \sum_{i=1}^{n} m_{li})\boldsymbol{I}_3$$

$$\boldsymbol{B}_{22} = \boldsymbol{Q}_B^T \boldsymbol{I}_B^B \boldsymbol{Q}_B + \sum_{i=1}^{n} (m_{li} {\boldsymbol{\Gamma}_\eta}^T \boldsymbol{S}(\boldsymbol{P}_{li/I}^I)^T \boldsymbol{S}(\boldsymbol{P}_{li/I}^I)\boldsymbol{\Gamma}_\eta + \boldsymbol{Q}_B^T \boldsymbol{R}_{li}^B \boldsymbol{I}_{li}^{li} \boldsymbol{R}_B^{li} \boldsymbol{Q}_B)$$

$$\boldsymbol{B}_{33} = \sum_{i=1}^{n} (m_{li} {\boldsymbol{J}_P^{(li)}}^T \boldsymbol{J}_P^{(li)} + {\boldsymbol{J}_O^{(li)}}^T \boldsymbol{R}_{li}^B \boldsymbol{I}_{li}^{li} \boldsymbol{R}_B^{li} \boldsymbol{J}_O^{(li)})$$

$$\boldsymbol{B}_{12} = {\boldsymbol{B}_{21}}^T = -\sum_{i=1}^{n} (m_{li} \boldsymbol{S}(\boldsymbol{P}_{li/I}^I)\boldsymbol{\Gamma}_\eta)$$

$$\boldsymbol{B}_{13} = {\boldsymbol{B}_{31}}^T = \sum_{i=1}^{n} (m_{li} \boldsymbol{R}_B^I \boldsymbol{J}_P^{(li)})$$

$$\boldsymbol{B}_{23} = {\boldsymbol{B}_{32}}^T = \sum_{i=1}^{n} (\boldsymbol{Q}_B^T \boldsymbol{R}_{li}^B \boldsymbol{I}_{li}^{li} \boldsymbol{R}_B^{li} \boldsymbol{J}_O^{(li)} - m_{li} {\boldsymbol{\Gamma}_\eta}^T \boldsymbol{S}(\boldsymbol{P}_{li/I}^I)^T \boldsymbol{R}_B^I \boldsymbol{J}_P^{(li)})$$

The potential energy of the system consists of the potential energy associated to UAV and the sum of potention energy associated to link i which can be given by:

$$\mathcal{U} = m_B g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \boldsymbol{P}_{B/I}^I + g \sum_{i=1}^{n} \left(m_{li} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (\boldsymbol{P}_{B/I}^I + \boldsymbol{R}_B^I \boldsymbol{P}_{li/B}^B) \right) \tag{2.25}$$

where $g = 9.8 m/s^2$ is the gravity acceleration value and $[0 \quad 0 \quad 1]^T$ represents that the gravity acts along the z axes of inertia frame.

Then, based on the total kinetic and potential energies we have derived in **??** and **??**, by computing Lagrange equations in **??** and taking the Christoffel symbols of the first type [20] we can get the dynamic modeling of whole system as

$$\boldsymbol{B(\xi)}\ddot{\boldsymbol{\xi}} + \boldsymbol{C}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}})\dot{\boldsymbol{\xi}} + \boldsymbol{g(\xi)} = \boldsymbol{u} + \boldsymbol{u_{ext}} \tag{2.26}$$

In which, $\boldsymbol{u}$ is a $(n_\xi \times 1)$ vector representing the generalized input forces. $\boldsymbol{g(\xi)} = (\partial \mathcal{U}(\boldsymbol{\xi})/\partial \boldsymbol{\xi})^T$ and $\boldsymbol{u_{ext}}$ represents the effects caused by external generalized force and $\boldsymbol{C}$ is a $(n_\xi \times n_\xi)$ matrix in which

$$c_{ij} = \sum_{k=1}^{n_\xi} \frac{1}{2} \left(\frac{\partial b_{ij}}{\partial \xi_k} + \frac{\partial b_{ik}}{\partial \xi_j} + \frac{\partial b_{jk}}{\partial \xi_i} \right) \dot{\xi}_k$$

Chapter 3

Robot Motion Planning

In this chapter we will develop a high level motion planner framework for multi-agent case. In particular, we will construct the abstraction of robot motion in workspace first. Then we need to develop the LTL formula and related büchi automaton based on the complex robot task specification. Last but not the least, we will do synchronized product of finite transition system and büchi automaton and search the optimal path.

3.1 Discretized Abstraction

We will use a weighted finite transition system(FTS)[21] to describe the behavior of a robot within a workspace in this section.

3.1.1 Partition of Workspace

Suppose we partition the initial workspace into N regions, denote by $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, where $\pi_i \cap \pi_j = \emptyset$, if $i \neq j$. There are different decomposition schemes available depends on control strategy and robot dynamic [22] [23] [24] such as triangles, rectangle, polygon and sphere. The partition naturally include the Boolean proposition $\Psi_r = \{\Psi_{r1}, \Psi_{r2}, ...\Psi_{rn}\}$ which has the following property:

$$\Psi_{ri} = \begin{cases} Ture & if \quad p \in \pi_i \\ False & if \quad p \notin \pi_i \end{cases} \tag{3.1}$$

where p means current position of robot.

In addition, we also would like to define the what properties can be satisfied as region π_i. Then, we defined another set of Boolean proposition $\Psi_p = \{\Psi_{p1}, \Psi_{p2}, ..., \Psi_{pn}\}$ to denote the properties at every region.

3.1.2 Continuous Dynamic of Robot

Although we decompose the workspace into several independent component, we should also take the geometric setting of the robot into consideration so that the planned motions can be executable in the physical world[25].

Normally, the continuous dynamical of the robot motion is often formalized using a function $p : [0, T] \to \mathcal{S}$ that satisfies an ordinary differential equation of the form:

$$\dot{p(t)} = f(p(t), u(t)) \tag{3.2}$$

Where $p(t)$ is the position of robot at time t. $u : [0, T] \to \mathcal{U}$ for all $t \in [0, T]$. $\mathcal{S} \in \mathbb{R}^n$ and $\mathcal{U} \in \mathbb{R}^n$ are state space and control space respectively.

Besides, there is another constraint for controller $\mathcal{U}$:

$$u(t) = \mathcal{U}(\pi_i, \pi_j) \tag{3.3}$$

Which means that when robot move from current region π_i to next region π_j within finite time T, controller $\mathcal{U}$ need to ensure the position of robot $p(t) \in (\pi_i \cup \pi_j)$at any time $t \in [0, T]$.

3.1.3 Weighted FTS

Based on section 3.1.1 and 3.1.2, we can abstract the robot motion among the region $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, where we use boolean proposition r_i represents robot is now in region π_i for $i = 1, 2, .., n$. Therefore, we can represent the robot motion by a weighted finite transition system.

Definition 3.1. An **atomic proposition**(AP) is a boolean statement or assertion that must be true or false.

Definition 3.2. The weighted FTS is a

$$\mathcal{M} = \{\Pi_\mathcal{M}, Act_\mathcal{M}, \to_\mathcal{M}, \Pi_{\mathcal{M}0}, \Psi_\mathcal{M}, L_\mathcal{M}, W_\mathcal{M}\} \tag{3.4}$$

- $\Pi_\mathcal{M}$ is a set of state and $\Pi_\mathcal{M} = \{\pi_i, i = 1, 2, ..., n\}$
- $Act_\mathcal{M}$ is a set of action we need to move from on region to another.
- $\to_\mathcal{M}$ is a set of transition between states and $\to_\mathcal{M} \subseteq \Pi_\mathcal{M} \times Act_\mathcal{M} \times \Pi_\mathcal{M}$.
- $\Pi_{0\mathcal{M}}$ is a set of initial state.
- $\Psi_\mathcal{M}$ is a set of atomic proposition and $\Psi_\mathcal{M} = \Psi_r \cup \Psi_p$.
- $L_\mathcal{M}$ is a label function $L_\mathcal{M} : \Pi_\mathcal{M} \to 2^{\Psi_\mathcal{M}}$ indicate the set of atomic proposition satisfied at each state.
- $W_\mathcal{M}$ is the weight of each transition $\in \to_\mathcal{M}$, in our case $\pi_i \to_\mathcal{M} \pi_j$ we simply define the weight of $\to_\mathcal{M}$ as the straight-line distance $dist(\pi_i, \pi_j)$ between two states.

3.2 LTL Based Specification

3.2.1 Syntax and Semantics

Linear Temporal Logic(LTL) formula over the set of atomic proposition AP can be constructed by using following syntax:

$$\varphi ::= True \quad | \quad a \quad | \quad \varphi_1 \wedge \varphi_1 \quad | \quad \neg\varphi \quad | \quad \bigcirc \varphi \quad | \quad \varphi_1 \cup \varphi_2 \tag{3.5}$$

where $a \in AP$ and $\wedge$(and), $\neg$(not), $\bigcirc$(next), $\cup$(until)

Based on LTL syntax above, we can derive other useful operators such as:

- $\vee(or) := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- $\Rightarrow (implication) := \neg\varphi_1 \vee \varphi_2$
- $\Diamond(eventually) := True \cup \varphi$
- $\Box(always) := \neg\Diamond\neg\varphi$
- $\Diamond\Box\varphi$: 'eventually forever φ'
- $\Box\Diamond\varphi$: 'infinitely often φ'

LTL formula can specify the property of path. In other words, it specifies a set of infinite words $Words(\varphi)$ over 2^{AP} that satisfied the formula φ where

$$Words(\varphi) = \{\sigma \in (2^{AP}) \quad | \sigma \models \varphi\} \tag{3.6}$$

In which $\sigma = \sigma_0\sigma_1\sigma_2....$

Definition 3.3. The semantics of LTL can be defined as follow:

- $\sigma \models True$
- $\sigma \models a$ iff $\sigma_0 \models a$
- $\sigma \models \varphi_1 \wedge \varphi_2$ iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$
- $\sigma \models \neg\varphi$ iff $\sigma \not\models \varphi$
- $\sigma \models \bigcirc\varphi$ iff $\sigma[1...] = A_1A_2A_3... \models \varphi$
- $\sigma \models \varphi_1 \cup \varphi_2$ iff $\exists j \geq 0, \sigma[j...] \models \varphi_2$ and $\sigma[i...] \models \varphi_1$, for all $0 \leq i < j$

In order to have a better understanding of LTL formula, an example is provided:

Example 3.1. $(\Box\Diamond A) \wedge (\Box A \Rightarrow \bigcirc(\neg A \cup B))$ means infinity often visit region A, besides, never visit A again before finishing visiting region B.

3.2.2 Büchi Automaton

From section 3.2.1 we know that the LTL formula specifies a set of infinite words $Words(\varphi)$ over 2^{AP}. Therefore, we need to find a kind of automaton that is suited for this infinite word $Word(\varphi)$ – Nondeterministic Büchi Automaton(NBA).

Definition 3.4. Given a LTL formula φ, there exist a NBA over 2^{AP} corresponding to φ, where the NBA can be denoted as a tuple

$$\mathcal{A}_\varphi = (Q, 2^{AP}, \delta, Q_0, F) \tag{3.7}$$

- Q is a set of states
- 2^{AP} is an alphabet
- δ is a transition relation which means $Q \times 2^{AP} \to 2^Q$
- Q_0 is a set of initial states
- F is set of accept state

Definition 3.5. For a NBA $\mathcal{A}_\varphi$, if there exists a language $\mathcal{L}(\mathcal{A}_\varphi) = Words(\varphi)$ in which some accept state occur infinitely often, we can say that$\mathcal{L}(\mathcal{A}_\varphi)$ is an accepting run in $\mathcal{A}_\varphi$[21].

Definition 3.6. The accepting run of a büchi automaton has the 'prefix-suffix' structure where prefix starts from the initial state $q_0 \in Q_0$ to one of accept state $q_f \in F$ and only visit once while the suffix loop is repeated infinitely.

$$\tau = \tau_{pre}(\tau_{suf})^\omega \tag{3.8}$$

There are several algorithms can be used to convert the LTL formula into a NBA which has already been reviewed in chapter 1. In this thesis, we directly use the method develop by Denis Oddoux and Paul Gastin called LTL2BA[26]. LTL2BA uses generalized Büchi automaton as an transitional step, generating a very weak alternating co-Büchi automaton and then transforms it into a Büchi automaton.

Example 3.2. Given a LTL formula $(\Box\Diamond a)\wedge(\Box\Diamond b)\wedge(\Box\Diamond c)\wedge(\Box(a \Rightarrow \bigcirc b))$, derive the corresponding NBA.

The NBA derived by LTL2BA is shown in figure 3.1 where the boolean expression for example $(!a \,\&\, \&b)$ denoting the expression $\{b\}$ and $\{b, c\}$

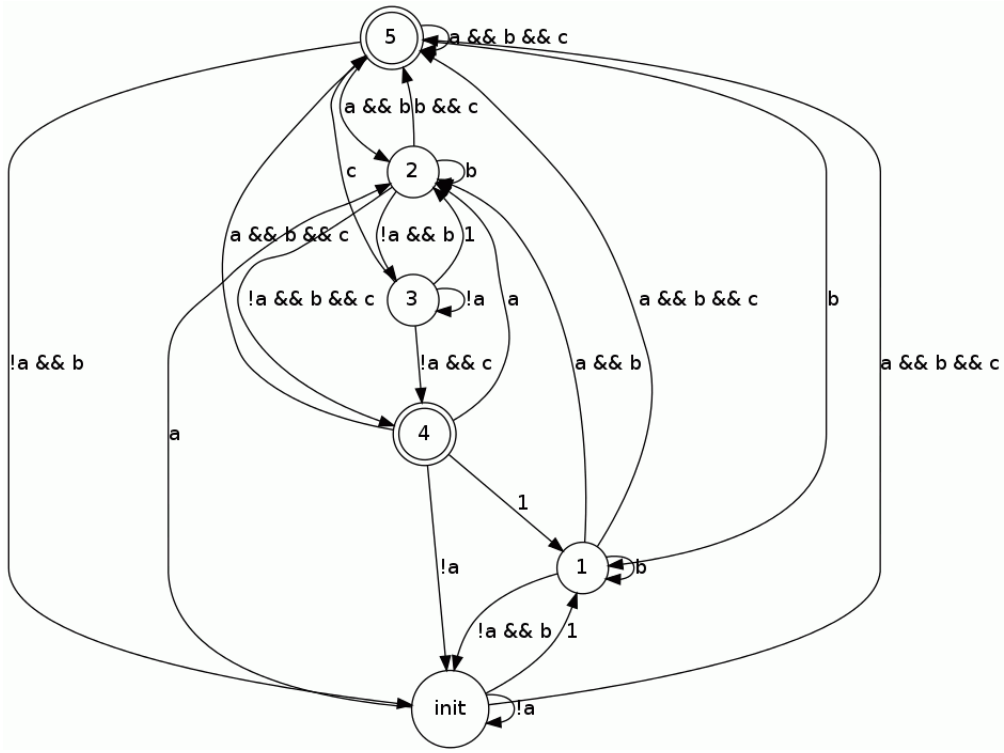

Figure 3.1 NBA corresponding to example 3.2

3.3 Optimal Path Planning

Based on the weighted finite transition system(wFTS) that can be used to describe the behavior of a robot within a workspace and the LTL formula based Nondeterministic Büchi Automaton(NBA), we would like to derived the synchronized product of TS and NBA that satisfied the task specification and find the optimal path that minimize the cost during whole process.

3.3.1 Product Büchi Automaton

Definition 3.7. The synchronized product of wFTS $\mathcal{M}$ and NBA $\mathcal{A}_\varphi$ can be defined by a tuple

$$\mathcal{A}_p = \mathcal{M} \otimes \mathcal{A}_\varphi = \{Q', 2^{AP}, \delta', Q'_0, F', W_p\} \tag{3.9}$$

- Q' is the set of states where $Q' = \Pi_\mathcal{M} \times Q = \{(\pi, q) \in Q' | \pi \in \Pi_\mathcal{M}$ and $q \in Q\}$
- 2^{AP} is atomic proposition set

- δ' is the transition function between different states where $(\pi_j, q_n) \in \delta'(\pi_i, q_m)$ iff $\rightarrow_{\mathcal{M}} \subseteq \pi_i \times Act_{\mathcal{M}} \times \pi_j$ and $q_n \in \delta(q_m, L_{\mathcal{M}}(\pi_j))$
- Q'_0 is a set of initial state where$(\pi, q) \in Q'_0$ iff $\pi \in \Pi_0$ and there exists $q_0 \in Q_0 \cap q \in \delta(q_0, L_{\mathcal{M}}(\pi))$
- $\mathcal{F}'$ is the set of accept sates where $\mathcal{F}' = \{(\pi, q) | q \in \mathcal{F} \text{ and } \pi \in \Pi_{\mathcal{M}}\}$
- W_p is the weight function of edge where $W_p\{(\pi_i, q_m), (\pi_j, q_n)\} = W_{\mathcal{M}}(\pi_i, \pi_j)$

Since $\mathcal{A}_p$ is still a büchi automaton[21], the accepting run can be defined similar as $\mathcal{A}_\varphi$. Therefore, our task in the next step is to search the optimal run that has the structure like:

$$\mathcal{R} = \mathcal{R}_{pre}(\mathcal{R}_{suf})^\omega \tag{3.10}$$

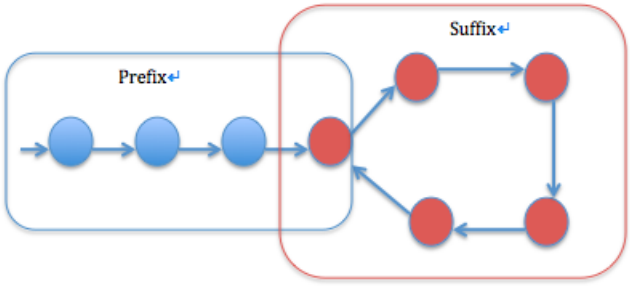

Figure 3.2 Accepting run consists of prefix and suffix

3.3.2 Shortest Path Search

Finding the optimal path planing based on product automaton $\mathcal{A}_p$ means that we need to find the prefix(from initial states to some certain accept state) and the suffix(the loop starts from accept states and end at the same state) as well as the best combination of prefix and suffix that can minimize the total cost(the sum of edge weight along the shortest path).

There are several algorithms can be used to solve this kind of shortest path problem such as Dijkstra's [27], Bellman Ford [28], A^* search [29], Floyd Warshall [30]. Considering the characteristic of these algorithm, Dijkstra algorithm would be the classic option in our case.

Dijkstra solves the one-all shortest path problem which means it can determine the shortest path from initial state q_0 to all the other state in the graph. In order to speed up the algorithm, we modified the algorithm slightly: as long as all the states in the targets are visited, we stop the algorithm since rest of the states unvisited are worthless to us. The process of searching shortest prefix is described by algorithm 3 and algorithm 4

Algorithm 3 Dijkstra_shortest_prefix

Require: Product automaton $\mathcal{A}_p$, initial state $q_0 \in Q'_0$
Ensure: The minimum distance $dist[q]$ from particular node q to q_0 and $pred[q]$ of node q along the shortest path from q_0 to q.

```
1: function DIJKSTRA_SHORTEST_PREFIX(A_p, q_0, targets)
2:     tovisit = []
3:     visited = []
4:     targets = F'
5:     for q ∈ Q' do
6:         d[q] = ∞
7:         tovisit.append(q)
8:     end for
9:     d[q_0] = 0
10:    pre[q_0] = NIL
11:    while tovisit ≠ ∅ ∧ targets ≠ ∅ do
12:        u = Extract_Min(tovisit)
13:        visited.append((u, d[u]))
14:        tovisit.pop(u)
15:        for v ∈ post[u] do
16:            if d[v] > d[u] + W'(u, v) then
17:                d[v] = d[u] + W'(u, v)
18:                pred[u] = v
19:            end if
20:        end for
21:        if u ∈ targets then
22:            targets.pop(u)
23:        end if
24:    end while
25:    return dist, pred
26: end function
```

Algorithm 4 Compute_path

Require: $(dist, pred)$ from Dijkstra_shortest_prefix, initial state q_i, accept state q_j
Ensure: shortest path between q_i and q_j

```
function COMPUTE_PATH(q_i, q_j)
    path = []
    u = q_i
    while pred[u] ≠ q_j do
        path.append((u, pred[u]))
        u = pred[u]
    end while
    path.append((u, q_j))
    return path
end function
```

However, when it comes to the problem 'finding the shortest suffix loop', things become a little complicated since for dijistra algorithm, the shortest path from q_s to q_s is 0. Actually, we can consider about one step previous state before accept state $pre[q_s]$, then we can use the dijistra algorithm to compute the shortest path from q_s to $pre[q_s]$ and finally add $(pre[q_s], q_s)$ into path.

Algorithm 5 Dijkstra_shortest_suffix

Require: Product automaton $\mathcal{A}_p$, accept state $q_s \in F'$
Ensure: Shortest path $\mathcal{R}_{suf}$ from q_s to q_s

```
path = []
targets = pre(q_s)
dist, pred = Dijkstra_shortest_prefix(A_p, q_s, targets)
for p ∈ pre(q_s) do
    path.append(compute_path(q_s, p))
    path.append((p, q_s))
end for
return Suffix path R_suf
```

After we generating both shortest prefix and suffix loop, we can search the optimal path by find the best combination of prefix and suffix. Since the suffix loop will be run infinite times, we should put more emphasis on the cost of suffix when we talking about the minimization of total cost. Therefore we need a weighting parameter β to adjust cost function:

$$\mathcal{W}_{total} = \mathcal{W}_{\mathcal{R}_{pre}} + \beta \mathcal{W}_{\mathcal{R}_{suf}} \tag{3.11}$$

.

Chapter 4

Local Task Planning and Multi-agent Case

In this chapter, we will first develop the complete robot model including both robot motion and action. Then we will consider the multi-agent case in which we will design independent task for each agent, discuss collision avoidance for high level planner as well as the hybrid controller for realizing the collision avoidance planner by using navigation function.

4.1 Local Task Planning

4.1.1 Robot Action Model

There are several standard automated action planners such as STRIPS [31], Planning Domain Definition Language(PDDL) [32], Action description language(ADL) [33]. Given the initial state and the specification of target state, they evaluate whether a action is executable by analyzing the precondition and postcondition of the action [34]. We will use the same approach in [34]

Before we formalized the action model, the simple example is provided

Example 4.1. There is a task: pick ball A at region 4. Let's consider this problem by using precondition and postcondition. The result is sown in figure 4.1

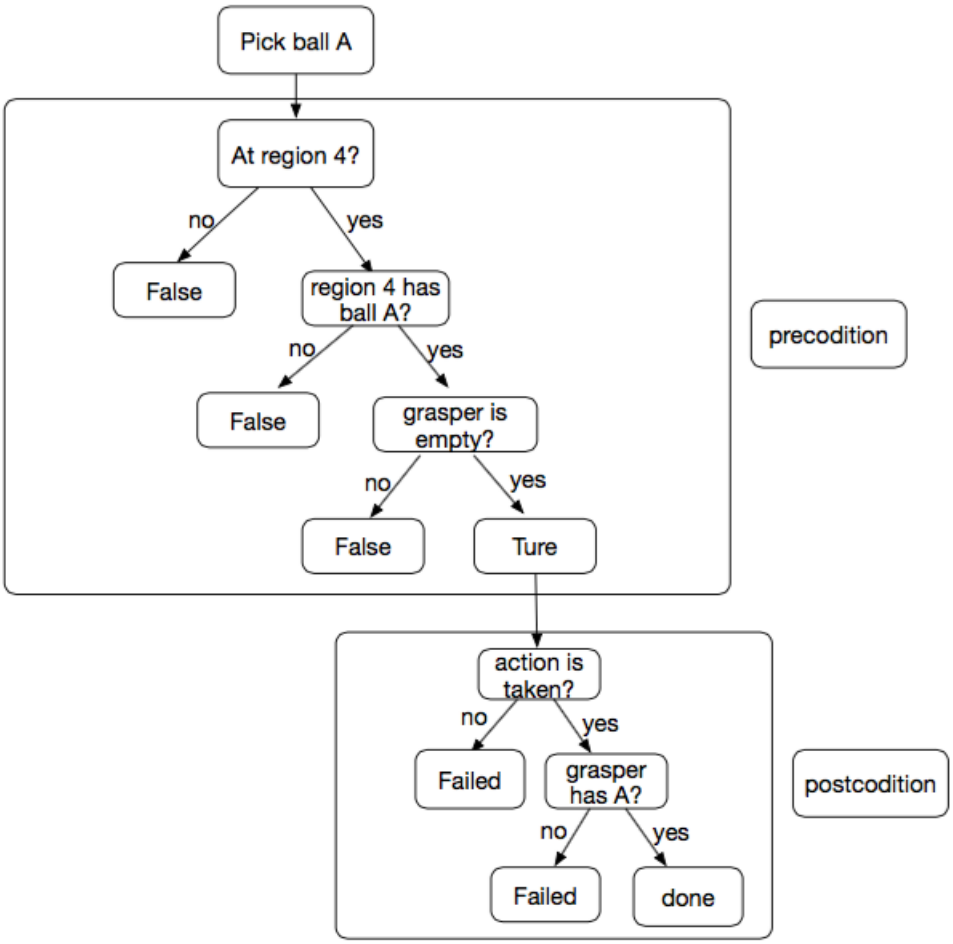

Figure 4.1 Illustration for example 4.1

Therefore, we can denote the set of action performed by robot as $Act = \{Act_0, Act_1, Act_2, ..., Act_n\}$, where Act_0 means there is no action performed. Besides, we introduce other three sets of proposition:

- $\Psi_{\mathcal{M}} = \Psi_r \cup \Psi_p$, the atomic propositions of workspace introduce in Chapter 3.

- $\Psi_s = \{\Psi_{s1}, \Psi_{s2}, ..., \Psi_{sj}\}$, the atomic proposition used to represent different internal state of robot, for example,'robot is holding ball A','robot's grasper is empty' and so on.

- $\Psi_b = \{\Psi_{b1}, \Psi_{b2}, ..., \Psi_{bk}\}$, Ψ_{bk} is verification of action k. Only if action k is preformed, $\Psi_{bk} = True$

Based on $\Psi_{\mathcal{M}}, \Psi_s, \Psi_b$, we can describe the precondition(pred) and post-condition(posd) as follow:

The precondition function:

$$Act \times 2^{\Psi_{\mathcal{M}}} \times 2^{\Psi_s} \rightarrow True/False \tag{4.1}$$

which means before we take the action, we need to consider about the internal state of robot and property of the current region, if both condition is satisfied, then the action can be taken ($Ture$). Note that for Act_0 the precondition is $True$.

The postcondition function:

$$Act \times (2^{\Psi_s} \times \Psi_b) \rightarrow (2^{\Psi_s} \times \Psi_b) \tag{4.2}$$

which means that after the action is taken, internal state of robot and verification of action will change. For example, after action 'pick ball A' is implemented, the internal state of robot change from 'grasper is empty' to 'robot is holding A', besides, the Ψ_{bk} corresponding to task 'pick ball A' is activated to be $True$ while other $\Psi_{bj} \in \Psi_b$ are $False$.

Note that here we use Ψ_b instead of 2^{Ψ_b} because that each time there will be only one action performed so that we can reduce the size of automaton significantly.

Definition 4.1. Given $\Psi_{\mathcal{M}}, \Psi_s, \Psi_b$ and Act, the robot action model can be defined as

$$\mathcal{A} = \{\Pi_{\mathcal{A}}, Act_{\mathcal{A}}, \Psi_{\mathcal{M}}, \rightarrow_{\mathcal{A}}, \Pi_{\mathcal{A}0}, \Psi_{\mathcal{A}}, L_{\mathcal{A}}, W_{\mathcal{A}}\} \tag{4.3}$$

- $\Pi_{\mathcal{A}} \subseteq (2^{\Psi_s} \times \Psi_b)$ is a set of all internal state and verification of action.
- $Act_{\mathcal{A}}$ is a set of actions which the robot is capable of.
- $\Psi_{\mathcal{M}}$ is the property of each region in workspace.
- $\rightarrow_{\mathcal{A}}$ is the transition relation denoted by $\rightarrow_{\mathcal{A}} \subseteq \pi_{\mathcal{A}} \times act_{\mathcal{A}} \times 2^{\Psi_{\mathcal{M}}} \times \pi'_{\mathcal{A}}$ only if the following condition hold:
  - $\pi_{\mathcal{A}}, \pi'_{\mathcal{A}} \in \Pi_{\mathcal{A}}$ and $act_{\mathcal{A}} \in Act_{\mathcal{A}}$
  - $pred(act_{\mathcal{A}}, 2^{\Psi_{\mathcal{M}}}, \Psi_{\mathcal{M}}) \rightarrow True$
  - $posd(act_{\mathcal{A}}, \pi_{\mathcal{A}}) \rightarrow \pi'_{\mathcal{A}}$
- $\Pi_{\mathcal{A}0} \subseteq (2^{\Psi_s} \times \Psi_{b0})$ is a set of initial state
- $\Psi_{\mathcal{A}} = \Psi_s \cup \Psi_b$ is the set of all atomic proposition in model
- $L_{\mathcal{A}}$ is the label of state which equals to $\pi_{\mathcal{A}}$
- $W_{\mathcal{A}}$ is the weight function which measures the cost of edges $\rightarrow_{\mathcal{A}}$

4.1.2 Complete Robot Model

Now, we have derived the robot motion model in Chapter 3:

$$\mathcal{M} = \{\Pi_{\mathcal{M}}, Act_{\mathcal{M}}, \rightarrow_{\mathcal{M}}, \Pi_{\mathcal{M}0}, \Psi_{\mathcal{M}}, L_{\mathcal{M}}, W_{\mathcal{M}}\} \tag{4.4}$$

as well as the robot task model in Chapter4:

$$\mathcal{A} = \{\Pi_{\mathcal{A}}, Act_{\mathcal{A}}, \Psi_{\mathcal{M}}, \rightarrow_{\mathcal{A}}, \Pi_{\mathcal{A}0}, \Psi_{\mathcal{A}}, L_{\mathcal{A}}, W_{\mathcal{A}}\} \tag{4.5}$$

We can develop the complete robot model for both motion and action.

Definition 4.2. The complete robot model can be described as a tuple:

$$\mathcal{C} = \mathcal{M} \otimes \mathcal{A} = \{\Pi_{\mathcal{C}}, Act_{\mathcal{C}}, \rightarrow_{\mathcal{C}}, \Pi_{\mathcal{C}0}, \Psi_{\mathcal{C}}, L_{\mathcal{C}}, W_{\mathcal{C}}\} \tag{4.6}$$

- $\Pi_{\mathcal{C}} = \Pi_{\mathcal{A}} \times \Pi_{\mathcal{M}}$ is the set of state
- $Act_{\mathcal{C}} = Act_{\mathcal{A}} \cup Act_{\mathcal{M}}$ is the set of all actions
- $\rightarrow_{\mathcal{C}} \subseteq \Pi_{\mathcal{C}} \times Act_{\mathcal{C}} \times \Pi_{\mathcal{C}}$ is the transition function following the rules below:
  - $(\pi_{\mathcal{M}}, \pi_{\mathcal{A}}) \xrightarrow{act_{\mathcal{M}}} (\pi'_{\mathcal{M}}, \pi_{\mathcal{A}})$ iff $\pi_{\mathcal{M}} \xrightarrow{act_{\mathcal{M}}} \pi'_{\mathcal{M}}$ and $\pi_{\mathcal{A}} \xrightarrow{act_{\mathcal{A}0}} \pi_{\mathcal{A}}$
  - $(\pi_{\mathcal{M}}, \pi_{\mathcal{A}}) \xrightarrow{act_{\mathcal{A}}} (\pi_{\mathcal{M}}, \pi'_{\mathcal{A}})$ iff $\pi_{\mathcal{A}} \times act_{\mathcal{A}} \times {}_{\mathcal{M}}(\pi_{\mathcal{M}}) \times \pi'_{\mathcal{A}} \subset \rightarrow_{\mathcal{A}}$
- $\Pi_{\mathcal{C}0} = \Pi_{\mathcal{M}0} \times \Pi_{\mathcal{A}0}$ containing the initial region and initial action of robot.
- $\Psi_{\mathcal{C}} = \Psi_{\mathcal{M}} \cup \Psi_{\mathcal{A}}$ representing all the atomic proposition of whole model, including $\Psi_r, \Psi_p, \Psi_s, \Psi_b$
- $L_{\mathcal{C}} = L_{\mathcal{M}} \cup L_{\mathcal{A}}$ is the label function.
- $W_{\mathcal{C}}$ is the weight function where:
  - $W_{\mathcal{C}}\{(\pi_{\mathcal{M}}, \pi_{\mathcal{A}}), (\pi'_{\mathcal{M}}, \pi_{\mathcal{A}})\} = W_{\mathcal{M}}(\pi_{\mathcal{M}}, \pi'_{\mathcal{M}})$
  - $W_{\mathcal{C}}\{(\pi_{\mathcal{M}}, \pi_{\mathcal{A}}), (\pi_{\mathcal{M}}, \pi'_{\mathcal{A}})\} = W_{\mathcal{A}}(\pi_{\mathcal{A}}, \pi'_{\mathcal{A}})$

So far, we have developed the whole structure for single motion and action planner. Given the complete robot model $\mathcal{C}$ and LTL specification based automaton $\mathcal{A}_\psi$, we can build the synchronized product and find the optimal path satisfied requirement by using the algorithm introduced in Chapter 3.

4.2 Multi-agent Case

When the task specification become complicated and there is time constrain for the task implementation or the work requires performing different action by different kinds of robot , it's obvious that we can not use only one robot to finish all the task. Therefore, it is necessary do discuss the multi-agent case.

4.2.1 Centralized Motion and Task Planning

Suppose we have N agents in total. For each agent $i \in \{1, 2, ..., n\}$, we model the motion of agent as:

$$\mathcal{M}^i = \{\Pi_{\mathcal{M}}, Act^i_{\mathcal{M}}, \rightarrow^i_{\mathcal{M}}, \Pi^i_{\mathcal{M}0}, \Psi^i_{\mathcal{M}}, L^i_{\mathcal{M}}, W^i_{\mathcal{M}}\} \tag{4.7}$$

The definition rule is same as single agent case in Chapter 3. Since all the agents share the same workspace, transition state $\Pi_{\mathcal{M}}$ should be the same, but they may have different transition relation and initial state.

Considering the collision avoidance,that is, there can not be two or more agents in one region at same time, we can generate the product motion model as:

$$\mathcal{M} = \{\Pi_{\mathcal{M}}, Act_{\mathcal{M}}, \rightarrow_{\mathcal{M}}, \Pi_{\mathcal{M}0}, \Psi_{\mathcal{M}}, L_{\mathcal{M}}, W_{\mathcal{M}}\} \tag{4.8}$$

- $\Pi_{\mathcal{M}}$ is the set of state as defined in previous chapter
- $Act_{\mathcal{M}}$ is a set of action
- $\rightarrow_{\mathcal{M}}$ represents the transition relation which can be defined as
  - the example is give in two agent case
  - $(\pi_i, \pi_j) \xrightarrow{act_{\mathcal{M}}}_{\mathcal{M}} (\pi'_i, \pi'_j)$ iff $\pi'_i \subseteq post(\pi_i) \cap \pi'_j \subseteq post(\pi_j)$ and $\pi'_i \neq \pi'_j$
- $\Pi_{\mathcal{M}0} = \prod_{i=1}^{N} \Pi^i_{\mathcal{M}0}$ is the set of initial region.
- $\Psi_{\mathcal{M}} = \cup_{i=1}^{N} \Psi^i_{\mathcal{M}}$ is the atomic proposition of whole model
- $L_{\mathcal{M}} = \cup_{i=1}^{N} L^i_{\mathcal{M}}$ is the label function.
- $W_{\mathcal{M}}$ is the weight function where:
  - $W_{\mathcal{M}}\{(\pi_i ... \pi_j), (\pi'_i ... \pi'_j)\} = \sum_{i=1}^{N} W^i_{\mathcal{M}}(\pi_i, \pi'_i)$

if we define the action model of agent i as

$$\mathcal{A}^i = \{\Pi^i_{\mathcal{A}}, Act^i_{\mathcal{A}}, \Psi^i_{\mathcal{M}}, \rightarrow^i_{\mathcal{A}}, \Pi^i_{\mathcal{A}0}, \Psi^i_{\mathcal{A}}, L^i_{\mathcal{A}}, W^i_{\mathcal{A}}\} \tag{4.9}$$

The product action model of whole system can be simply defined as:

$$\mathcal{A} = \cup_{i=1}^{N} \mathcal{A}^i \tag{4.10}$$

Then the following steps: product motion and action model, LTL based automaton, synchronized product automaton and the optimal run search base on the same method introduced in previous section.
Note: in order to have a better understanding of multi-agent process, the specific implementation including partial code will be illustrated in chapter 5.

4.2.2 Navigation Function based Control Strategy

After finishing the high level path generating, we need to make sure the hybrid controller can be executed properly following the optimal path. Therefore,there are at least three preconditions we need to consider about: robot need to work within workspace; robot should never go across the forbidden area; and agents should avoid colliding with each other.The worst case condition may happening during the execution is represented in figure4.24.34.4

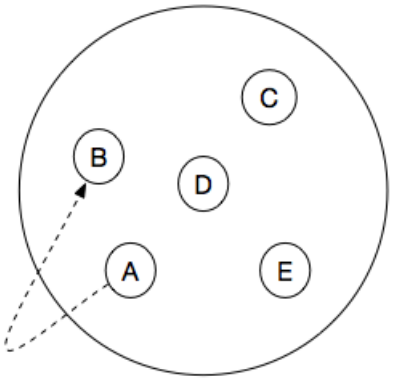

Figure 4.2 Agent fly outside the workspace

In order to avoid these kind of situation, we introduce the navigation function presented in [35].Assuming that the robot satisfied the single-integrator dynamics:

$$\dot{q} = u \tag{4.11}$$

where q and u represents the position and velocity of robot. According to Koditschek and Rimon, navigation function is a analytic real valued map

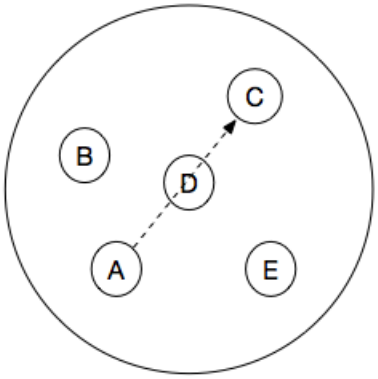

Figure 4.3 Agent flies from A to C while going across the forbidden area D

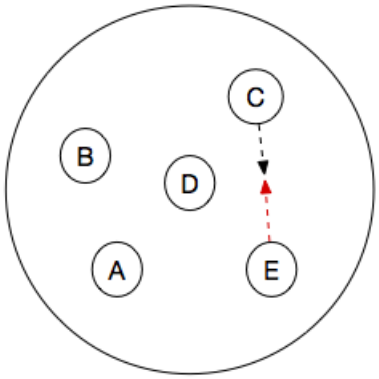

Figure 4.4 Collision between agent1(red) and agent2(black)

with a integrated gradient vector field, which can generate curves from any initial point to the destination in the sphere free space ($\mathscr{F}$) without colliding obstacles. The free space has the following definition.

$$\mathscr{F} \triangleq \mathscr{W} - \bigcup_{j=1}^{M} \mathscr{O}_j \tag{4.12}$$

Where $\mathscr{W}$ is the workspace regarded as a large sphere with radius ρ_0.

$$\mathscr{W} \triangleq \{q \in E^n : ||q - q_0||^2 \leq \rho_0^2\} \tag{4.13}$$

and $\mathscr{O}_j$ means the j_{th} obstacle. Each obstacle has a center point q_j and a radius ρ_j. So, $\mathscr{O}_j$ can be written in following format, and M is the number of the obstacles.

$$\mathscr{O}_j \triangleq \{q \in E^n : ||q - q_j||^2 \leq \rho_j^2\}, j = 1...M \tag{4.14}$$

Furthermore, the obstacles are strictly in the work space and non-intersect. It indicates that,

$$||q_j|| + \rho_j < \rho_0, j = 1...M$$
$$||q_i - q_j|| > \rho_i + \rho_j, j$$

Moreover, the destination point $q_d \notin \mathscr{O}_j, j = 1...M$

The proposed navigation function, $\varphi : \mathscr{F} \to [0, 1]$ (a mapping from the free space to [0,1]), is a combination of three functions, i.e. function composition.

$$\begin{aligned} \varphi &\triangleq \sigma_d \circ \sigma \circ \hat{\varphi} \\ &\triangleq \sigma_d[\sigma(\hat{\varphi})] \end{aligned} \tag{4.15}$$

Where $\hat{\varphi}$ is polar, almost everywhere Morse and analytic; obviously, it can reach the peak on $\partial\mathscr{F}$. Furthermore, we divide the set $\mathscr{F}$ into "good" subset and "bad" subset. The destination points belongs to "good" subset since there are negative gradient lines leading to it. Meanwhile, all the boundary points belongs to the "bad" subset. The cost of points in "bad" subset should be high. So, construct a fraction function with numerator term (γ) and denominator term (β). γ term includes the destination points while β term is consist of boundary points. As a result, $\hat{\varphi}$ has following format.

$$\hat{\varphi} \triangleq \frac{\gamma}{\beta} \tag{4.16}$$

Where $\gamma : \mathscr{F} \to [0, \infty)$ is

$$\gamma \triangleq \gamma_d^k, k \in \mathbb{N}; \tag{4.17}$$
$$\gamma_d \triangleq \|q - q_d\|^2 \tag{4.18}$$

and $\beta : \mathscr{F} \to [0, \infty)$ is

$$\beta \triangleq \prod_{j=0}^{M} \beta_j \tag{4.19}$$
$$\beta_0 \triangleq \rho_0^2 - \|q - q_0\|^2 \tag{4.20}$$
$$\beta_j \triangleq \|q - q_j\|^2 - \rho_j^2, j = 1...M \tag{4.21}$$

And the function σ transform the $[0, \infty)$ to $[0, 1]$. σ should be monotonically increasing on $[0, \infty)$ to make the set of critical points of $\hat{\varphi}$ and $\sigma(\hat{\varphi})$. So, it can be defined in following format.

$$\sigma(x) \triangleq \frac{x}{1+x} \tag{4.22}$$

Theoretically, the function composition $\sigma(\hat{\varphi})$ is non-degenerate on $\mathscr{F}$ except at the destination (q_d). To change the destination to a non-degenerate critical point, the function σ_d is needed. It has the coefficient k which is same coefficient in $\hat{\varphi}$. Also, the function σ_d transform the interval $[0, 1]$ to $[0, 1]$.

$$\sigma_d(x) \triangleq x^{\frac{1}{k}}, k \in \mathbb{N} \tag{4.23}$$

Finally, the navigation function on a valid free space $\mathscr{F}$, which has a designed positive integer k for any finite obstacles and for any destination point in the $\mathscr{F}$, is in following format.

$$\begin{aligned} \varphi &= \sigma_d \circ \sigma \circ \hat{\varphi} \\ &= \frac{\gamma_d}{(\gamma_d^k + \beta)^{1/k}} \end{aligned} \tag{4.24}$$

In more general case – multi-agent case, we need to slightly modify the original navigation function[36]:

$$\varphi_i = \frac{\gamma_{di} + f_i}{((\gamma_{di} + f_i)^k + G_i)^{1/k}} \tag{4.25}$$

where φ_i denotes the navigation function for robot i, G_i for agent i represents the relative position with other agents while f function will be defined later.

The reason why we need the G function in multi-agent case is that, we need to consider different collision condition. For example, agent i only collide with one agent or it collide with all other agent at same time which is illustrated in figure 4.5

The G_i function is given as:

$$G_i = \prod_{l=1}^{n_L^i} \prod_{j=1}^{n_{Rl}^i} (g_j)_l \tag{4.26}$$

where n_L^i means the number of level and n_{Rl}^i means the number of relations in level l for agent i.

$$(g_j)_l = (b_j)_l + \frac{\lambda (b_j)_l}{(b_j)_l + (\tilde{b_j})_l^{1/h}} \tag{4.27}$$

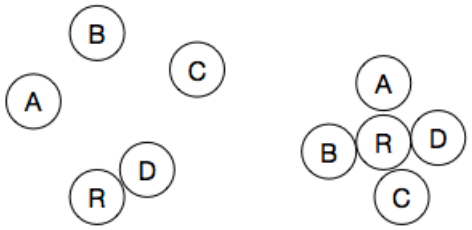

Figure 4.5 level 1: collide with one agent(left), level 4: collide with 4 agent(right)

where $(b_j)_l$ denotes the sum of all $\beta_{\{R,i\}}$in the j-th case in level l

$$(b_j)_l = \sum_{i \in j} \beta_{\{R,i\}} \tag{4.28}$$

for example, in figure 4.5, the 2-th case in level 2 can be

$$(b_2)_3 = \beta_{\{R,A\}} + \beta_{\{R,B\}} + \beta_{\{R,C\}}$$

and $(\tilde{b_j})_l$ denotes the complementary set of relations of level-l

$$(\tilde{b_j})_l = \prod_{m \in {R_l}^c} (b_m)l \tag{4.29}$$

for example, in figure 4.5, the five agent case, for agent R, the complementary set of 2-th case in level 2 is

$$\begin{aligned}
(\tilde{b_2})_3 &= (b_1)_3 \cdot (b_3)_3 \cdot (b_4)_3 \\
(b_1)_3 &= \beta_{\{R,A\}} + \beta_{\{R,B\}} + \beta_{\{R,D\}} \\
(b_3)_3 &= \beta_{\{R,A\}} + \beta_{\{R,C\}} + \beta_{\{R,D\}} \\
(b_4)_3 &= \beta_{\{R,B\}} + \beta_{\{R,C\}} + \beta_{\{R,D\}}
\end{aligned}$$

f_i is used to make sure the navigation function φ_i attains positive values in proximity situations even when agent i has already reached its destination. Both of them will be illustrated later.The f function is defined as:

$$f_i(G_i) = \begin{cases} a_0 + \sum_{j=1}^{3} a_j {G_i}^j, & G_i \leqslant X \\ 0, & G_i > X \end{cases} \tag{4.30}$$

where a_0, a_1, a_2, a_3 are a set of parameters to make sure when $G_i \to 0$, f_i is maximized while f_i is minimized when $G_i = X$

The navigation function is mathematically correct. Let the velocity of robot follow the negated gradient $u = - \bigtriangledown \varphi$, the navigation function can approach the destination point q_d to make $\bigtriangledown\varphi(q_d) = 0$, since q_d is a non-degenerate local minimum of φ. It supports a simple and valid motion planning algorithm. In other word, there exists a curve path for quadrotor to arrive the destination point in a valid free space without any collision from almost any starting position in $\mathscr{F}$

Chapter 5

Implementation

In this chapter we will verify the LTL based motion and action planner by a series of simulation and real quadcopter experiment. The block diagram representation of this planning work is given in figure5.1

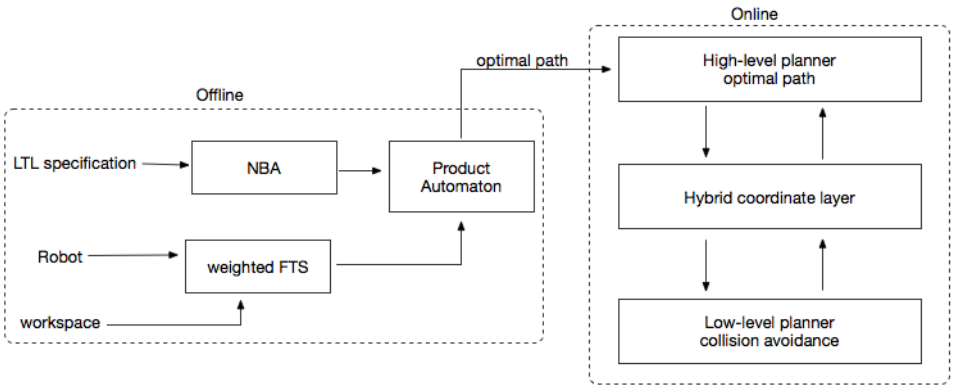

Figure 5.1 LTL based planning framework.

The algorithms for high-level planner and low-level planner are discussed in chapter 3 and chapter 4 separately. Then we need the hybrid coordination layer to communicate with each layer.

we generate the optimal path $\mathcal{R} = \pi_{\mathcal{R},0}\pi_{\mathcal{R},1}...$ based on algorithm 3,4,5. For each pair of transition $(\pi_{\mathcal{R},i}, \pi_{\mathcal{R},i+1})$ there exists an action $Act_{\mathcal{C}}$ where $\pi_{\mathcal{R},i} \times Act_{\mathcal{C}} \times \pi_{\mathcal{R},i+1} \subseteq \rightarrow_{\mathcal{C}}$. According to the $Act_{\mathcal{C}}$, corresponding controller is activated until the condition is satisfied. The detail of hybrid controller is illustrated in algorithm 6.

Algorithm 6 Hybrid_coordination_layer

Require: optimal path $\mathcal{R} = \pi_{\mathcal{R},0}\pi_{\mathcal{R},1}...$
Ensure: Low level control strategy
1: Follow the optimal path sequence $(\pi_{\mathcal{R},i}, \pi_{\mathcal{R},i+1}) \in \mathcal{R}$ for $i \in 0, 1, 2, ..., n-1$ repeat following step:
2: Since $\pi_{\mathcal{R},i} \times Act_{\mathcal{C}} \times \pi_{\mathcal{R},i+1} \subseteq \rightarrow_{\mathcal{C}}$
3: **if** $Act_{\mathcal{C}} \in Act_{\mathcal{A}}$ and $Act_{\mathcal{C}} = act_{\mathcal{A},k}$ **then**
4: Activate corresponding action controller $\mathcal{K}$ until $\Psi_{b,k}$ is True
5: **end if**
6: **if** $Act_{\mathcal{C}} \in Act_{\mathcal{M}}$ **then**
7: Navigation function is activated until the position of robot $p \in \pi_{\mathcal{R},i+1}$
8: **end if**

5.1 Simulation

In order to have a better visualization of results, we only study the 2D case. However, the model the method we proposed can be applied in N dimension.Considering the real number of quadcopter in Smart Mobility Lab, we will discuss the two quadcopters case in this chapter.

Considering the workspace given in figure 5.2

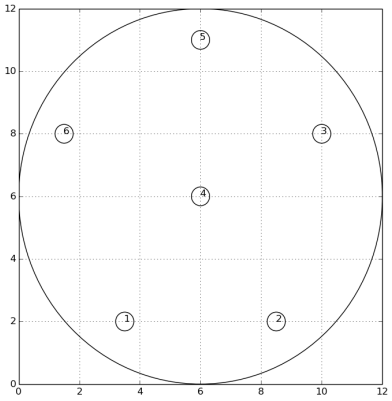

Figure 5.2 Abstraction of Workspace

The workspace is bounded in region $\pi_0 = (6, 6), r = 6$. The regions of interest are $\pi_1 = (3.5, 2), \pi_2 = (8.5, 2), \pi_3 = (8, 10), \pi_4 = (6, 6), \pi_5 = (6, 11), \pi_6 = (1.5, 8)$. Besides, we assume the region of interest and quadcopter have the same radius $r_state = r_agent = 0.3$. Each region is supposed to connected with any other region and the cost of edge is measured by the

geometric distance between them.Each region for different agent has different labels as show in figure 5.3

```
A_init_pose = (3.5,2)
A_node_dict ={
              (3.5,2): set(['a1', 'basketa']),
              (8.5,2): set(['a2']),
              (10,8): set(['a3']),
              (6,6): set(['a4']),
              (6,11): set(['a5']),
              (1.5,8): set(['a6','balla'])
              }

B_init_pose=(8.5,2)
B_node_dict ={
              (3.5,2): set(['b1']),
              (8.5,2): set(['b2','ballb']),
              (10,8): set(['b3']),
              (6,6): set(['b4']),
              (6,11): set(['b5','basketb']),
              (1.5,8): set(['b6'])
              }
```

Figure 5.3 Labels for Different Region

The motion model of each agent $\mathcal{M}^i$can be generate by:

```
class MotionFts(DiGraph):
    def __init__(self, node_dict, symbols, ts_type):
        DiGraph.__init__(self, symbols=symbols, type=ts_type, initial=set())
        for (n, label) in node_dict.iteritems():
            self.add_node(n, label=label, status='confirmed')


    def add_full_edges(self,unit_cost=1):     #Each region is supposed to connected with any other region
        for f_node in self.nodes_iter():
            for t_node in self.nodes_iter():
                dist = distance(f_node, t_node)
                if (f_node, t_node) not in self.edges():
                    self.add_edge(f_node, t_node, weight=dist*unit_cost)


    def set_initial(self, pose):        #s set of initial state
        init_node = self.closest_node(pose)
        self.graph['initial'] = set([init_node])
        return init_node
```

Figure 5.4 Motion Model for Single Agent

Therefore, the product motion model of two agents can be derived by **class MultiProdMot**

```
class MultiProdMot(DiGraph):
    def __init__(self, A1_motion, A2_motion):
        DiGraph.__init__(self, A1=A1_motion, A2=A2_motion, initial=set(), type='MultiProdMot')

    def build_full(self):
        for f_A1_node in self.graph['A1'].nodes_iter():
            for f_A2_node in self.graph['A2'].nodes_iter():
                if (f_A1_node != f_A2_node): #A and B can't be in same region at same time
                    f_prod_node = self.composition(f_A1_node, f_A2_node)
                    for t_A1_node in self.graph['A1'].successors_iter(f_A1_node):
                        for t_A2_node in self.graph['A2'].successors_iter(f_A2_node):
                            if (t_A1_node != t_A2_node):    #A and B can't be in same region at same time
                                t_prod_node = self.composition(t_A1_node,t_A2_node)
                                weight1=self.graph['A1'][f_A1_node][t_A1_node['weight']
                                weight2=self.graph['A2'][f_A2_node][t_A2_node['weight']
                                #update edges and weight
                                self.add_edge(f_prod_node, t_prod_node, weight=weight1+weight2)

    def composition(self, A1_node, A2_node):
        prod_node = (A1_node, A2_node)
        if not self.has_node(prod_node):
            new_label=self.graph['A1'].node[A1_node]['label'].union(self.graph['A2'].node[A2_node]['label'])
            self.add_node(prod_node, A1=A1_node, A2=A2_node, label=new_label, status='confirmed')
            if ((A1_node in self.graph['A1'].graph['initial']) and
                (A2_node in self.graph['A2'].graph['initial'])):
                self.graph['initial'].add(prod_node)
        return prod_node
```

Figure 5.5 Product Motion model

As for the action model, assuming agent A and agent B are capable of three action separately: $Act^1_{\mathcal{A}0} = None, Act^1_{\mathcal{A}1} = pick\ balla, Act^1_{\mathcal{A}2} = drop\ balla$ and $Act^2_{\mathcal{A}0} = None, Act^1_{\mathcal{A}1} = pick\ ballb, Act^2_{\mathcal{A}2} = drop\ ballb$, then the action dictionary can be initialized as:

```
action_dict={
             'picka': (100, 'balla', set(['picka']), 'A'),
             'dropa': (60, 'basketa', set(['dropa']), 'A'),
             'pickb': (100, 'ballb', set(['pickb']), 'B'),
             'dropb': (60, 'bastekb', set(['dropb']), 'B')
            }
```

Figure 5.6 Initialization of Action Dictionary

Where the second attribute of each key is the precondition of each action and the last attribute denotes the agent that a responsible for this action.

Therefore, the action model can be built by **class ActionModel**:

```python
class ActionModel(object):
    # action_dict = {act_name: (cost, guard_formula, label, executor)}
    def __init__(self, action_dict):
        self.raw = action_dict
        self.action = dict()
        for act_name, attrib in action_dict.iteritems():
            cost = attrib[0]
            guard_formula = attrib[1]
            guard_expr = parse_guard(guard_formula)
            label = attrib[2]
            agent=attrib[3]
            self.action[act_name] = (cost, guard_expr, label,agent)
        self.action['None']=(1, parse_guard('1'), set(),'N')
    # determine whchi act can be executed at current region
    def allowed_actions(self, ts_node_label):
        allow_action = set()
        for act_name, attrib in self.action.iteritems():
            if (attrib[1].check(ts_node_label)):
                allow_action.add(act_name)
        return allow_action
```

Figure 5.7 Action Model

Based on the product model and action model, we can generate the complete multi-agent model as:

```python
class MotActModel(DiGraph):
    def __init__(self, mot_fts, act_model):
        DiGraph.__init__(self, region=mot_fts, action=act_model, initial=set(), type='MotActModel')

    def composition(self, reg, act):
        prod_node = (reg, act)
        if not self.has_node(prod_node):
            new_label = self.graph['region'].node[reg]['label'].union(self.graph['action'].action[act][2])
            self.add_node(prod_node, label=new_label, region=reg, action=act)
            if ((reg in self.graph['region'].graph['initial']) and (act == 'None')):
                self.graph['initial'].add(prod_node)
        return prod_node

    def build_initial(self):
        for reg_init in self.graph['region'].graph['initial']:
            init_prod_node = self.composition(reg_init, 'None')

    def build_full(self):
        for reg in self.graph['region'].nodes_iter():
            for act in self.graph['action'].action.iterkeys():
                prod_node = self.composition(reg, act)
                # actions
                label = self.graph['region'].node[reg]['label']
                for act_to in self.graph['action'].allowed_actions(label):
                    prod_node_to = self.composition(reg, act_to)
                    if self.graph['action'].action[act_to][3]=='A':
                        active_agent='A'
                    elif self.graph['action'].action[act_to][3]=='B':
                        active_agent='B'
                    elif self.graph['action'].action[act_to][3]=='N':
                        active_agent='None'

                    self.add_edge(prod_node, prod_node_to, weight=self.graph['action'].action[act_to][0],
                     active_agent=active_agent,label= 'stay')
                # motions
                for reg_to in self.graph['region'].successors_iter(reg):
                    if reg_to != reg:
                        prod_node_to = self.composition(reg_to, 'None')
                        if (prod_node[0][0] != prod_node_to[0][0]) and (prod_node[0][1] != prod_node_to[0][1]):
                            active_agent='All'
                        elif (prod_node[0][0] != prod_node_to[0][0]) and (prod_node[0][1] == prod_node_to[0][1]):
                            active_agent='A'
                        elif (prod_node[0][0] == prod_node_to[0][0]) and (prod_node[0][1] != prod_node_to[0][1]):
                            active_agent='B'
                        self.add_edge(prod_node, prod_node_to, weight=self.graph['region'][reg][reg_to]['weight'],
                         active_agent=active_agent,label= 'goto')
```

Figure 5.8 Complete Multi-agent Model

Note that we discuss the case when agent stay at same region, which kind

of action can be executed at this region as well as when $[region] \neq [region_to]$, which agent has changed the region. This kind of labels are very important because after we finding the optimal path, each node $\pi_{\mathcal{R}i}$ will have the similar format as $((2.5, 8), (6, 11), 'picka')$ which is a representation for all agents at the same step. We need to extract the path for each agent, such as in figure 5.9

```
#for A
self.pre_plan_1 = []
self.pre_plan_1.append(self.line[0][0][0])
for ts_edge in self.pre_ts_edges:
    if (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'stay') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] != 'B'):
        self.pre_plan_1.append(ts_edge[1][1]) # action
    elif (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'goto') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] == 'B'):
        self.pre_plan_1.append('None') # motion
    else:
        self.pre_plan_1.append(ts_edge[1][0][0])
self.suf_plan_1 = []
for ts_edge in self.pre_ts_edges:
    if (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'stay') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] != 'B'):
        self.suf_plan_1.append(ts_edge[1][1]) # action
    elif (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'goto') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] == 'B'):
        self.suf_plan_1.append('None') # motion
    else:
        self.suf_plan_1.append(ts_edge[1][0][0])

#for B
self.pre_plan_2 = []
self.pre_plan_2.append(self.line[0][0][1])
for ts_edge in self.pre_ts_edges:
    if (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'stay') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] != 'A'):
        self.pre_plan_2.append(ts_edge[1][1]) # action
    elif (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'goto') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] == 'A'):
        self.pre_plan_2.append('None') # motion
    else:
        self.pre_plan_2.append(ts_edge[1][0][1])
self.suf_plan_2 = []
for ts_edge in self.pre_ts_edges:
    if (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'stay') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] != 'A'):
        self.suf_plan_2.append(ts_edge[1][1]) # action
    elif (product.graph['ts'][ts_edge[0]][ts_edge[1]]['label'] == 'goto') and
    (product.graph['ts'][ts_edge[0]][ts_edge[1]]['active_agent'] == 'A'):
        self.suf_plan_2.append('None') # motion
    else:
        self.suf_plan_2.append(ts_edge[1][0][1])
```

Figure 5.9 Extract Path for each agent

Case 1

Task Definition

In this case, we define the task for two agents as: **Agent A pick balla at region 6 and drop balla in the basketa in region 1, while agent B pick ballb at region 2 and drop ballb in the basket b in region 5 as well as visits region 3. All the tasks should be done infinitely often and never go to region 4** which can be written as LTL formula in

python:

$$([] <> (picka)) \ \&\& \ ([] <> (pickb)) \ \&\& \ ([] <> (b3)) \ \&\& \ ([](!b4) \ \&\& \ (!a4))$$
$$([]((picka) -> X(!(picka)Udropa))) \ \&\& \ ([](pickb -> X(!(pickb)Udropb))) \tag{5.1}$$

Hence, the path we generate as

- For agent A
  - prefix A: region 1 → region 6→ 'picka' → 'None' → region 1 → 'dropa' → region 2
  - surfix A: region 6 → 'picka' → 'None' → region 1 → 'dropa' → region 2
- For agent B
  - prefix B:region 3 → region 2 → 'None' → 'pickb' → region 5 → 'None' → 'dropb' → region 3
  - surfix B: region 2 → 'None' → 'pickb' → region 5 → 'None' → 'dropb' → region 3

The results are show in figure 5.10 and 5.11. For simplicity, only prefix is generated on figure.

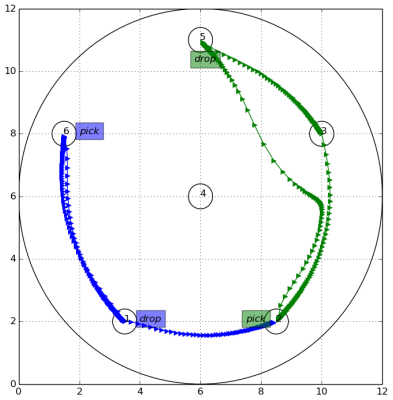

Figure 5.10 case1 with direction

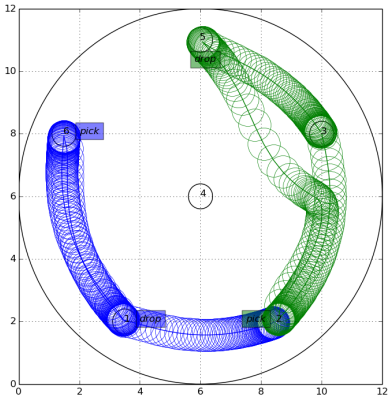

Figure 5.11 case1 with quads area

where the left one shows the direction of path while the right shows the area of the quadcopter so that we can know that both agent A and B never reach the forbidden region 4.

Case 2

There is another case we simulated with one quadcopter has specific action which the other doesn't.

Task Definition

In this case, the motion and action are defined as: **Agent B go to region 1, region 2, region 5 infinitely often. Agent A needs to pick ball at region 6 and drop ball at region 2 and report the task message at region 5 infinitely often.**

Note: Agent B needs to go to region 5 check message and send message to A so that A can pick the ball. After A drop the ball, it needs to go to region 5 report the status and then B will go to region 5 check message again.

Hence, the path we generate as

- For agent A
  - prefix A: region 1 → region 6 → 'pick' → region 2 → 'drop' → region 5
  - surfix A: region 6 → 'pick' → region 2 → 'drop' → region 5
- For agent B
  - prefix B: region 2 → region 5 → region 1 → region 3 → region 1 → region 2
  - surfix B: region 5 → region 1 → region 3 → region 1 → region 2

The results are show in figure 5.12 and 5.13. For simplicity, only prefix is generated on figure.

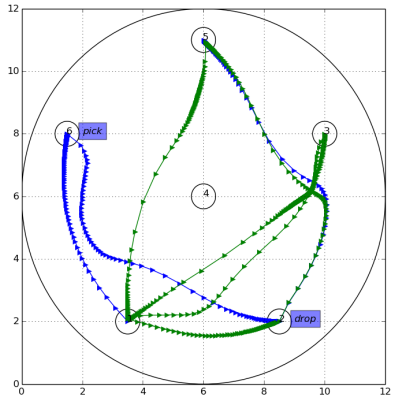

Figure 5.12 case2 with direction

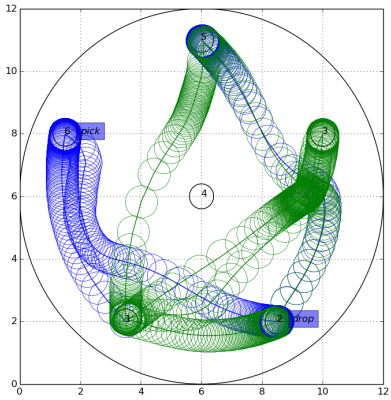

Figure 5.13 case2 with quads aera

5.2 Real Quadcopter based Experiment

In this section, we will focus on the real quadcopter experiment. Due to the limit of space, we reduce the radius of each state while keep the ratio between quadcopter and workspace the same as real condition.

Note two experiments implemented in this part are based on one real quadcopter and one visual quadcopter.

Since the basic framework of ROS has been built in Smart Mobility Lab, the main idea of this experiment is getting familiar with ROS framework so that we can establish desired the ros node file and subscrib the position information of each quadcopter from corresponding topic and publish the desired speed to speed controller as well as modify a set of launch files.

Real vs Virtual case 1

Consider about the following case:

Both agent A and agent B need to visit region 1, region 2 and region 3 infinitely often while avoid forbidden region 4

The initial state of agent A is region 1 and the initial state of agent B is region 2. The optimal path is found as:

- agent A: region 1 → region 2 → region 3 → region 1 → region 2 → region 3 →....
- agent B: region 2 → region 3 → region 1→ region 2 → region 3 → region 1 →....

Hence, the result is shown in figure 5.14

From result 1 we can easily tell that blue one is real quadcopter since there are a lot of measurement noise during the experiment. However, we can also find that the navigation function is quite robust during the real experiment.

Real vs Virtual Case 2

In order to further verify the validity of navigation function, due to the limited of space, we only consider the one step move: let agent A and agent B switch their position to see whether they can avoid each other automatically.

The result is show in figure 5.15. We notice that when it comes to the most trick problem: switch position with each other, two agents will adjust the position and distance between each other and will not collide with each other and other fixed obstacles.

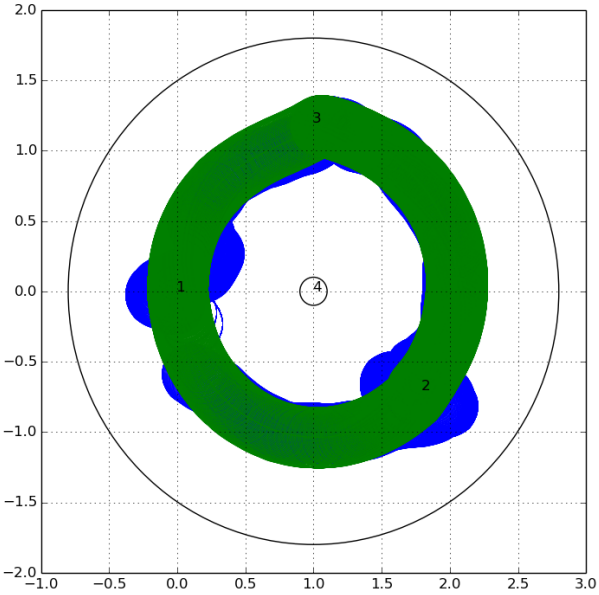

Figure 5.14 Real case 1 result

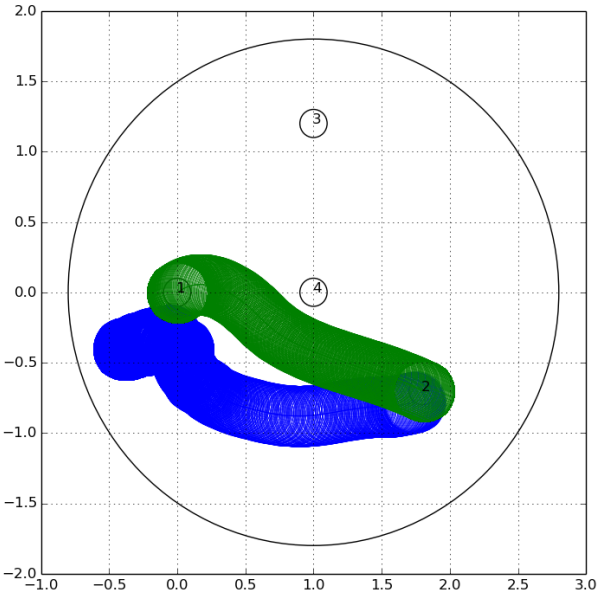

Figure 5.15 Real case 2 result

Two Real Quads Case

Since in figure 5.15 we can see that two quadcopters are very closed to each other when they want to switch position, we decide not to do the switch position case in two real quadcopters case.(The the air turbulence and measurement error may cause the crash and ruin two quadcopter.)

Therefore, we re-implement the **Real vs Virtual Case 1** and the result is shown in figure 5.16 and 5.17

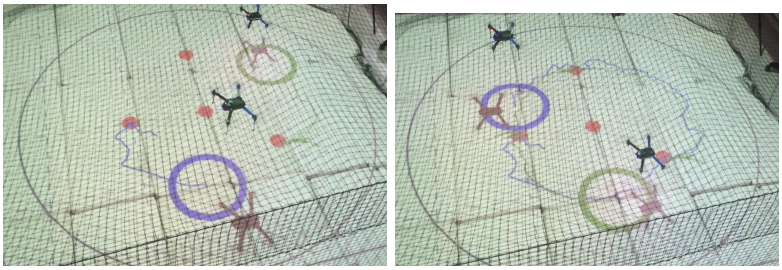

Figure 5.16 Two Real Quads Case online simulation

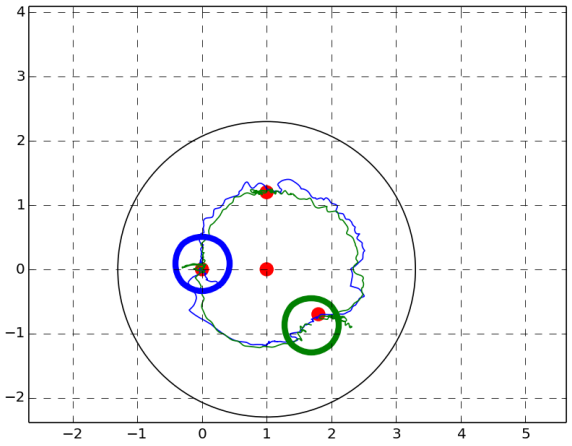

Figure 5.17 Two Real Quads Case online simulation

We can see that the measurement noise is quite large but the navigation function is still robust enough to achieve collision avoidance.

Chapter 6

Discussion

In this chapter, I will make a brief conclusion about what has been done and discuss about future work.

6.1 Conclusion

My main contribution in this master thesis is focus on following aspects:

- Completed dynamic modeling of quadcopter wiht manipulator.
  - Firstly we derived the kinematic model of quadcopter with manipulator.Then the dynamical modeling is based on the method Lagrange formulation with which the equations of motion can be derived in a systematic way independently of the reference coordinate frame. However, due to the limit of time and lack of real manipulator with actuator, we didn't design the controller in this case. But the dynamic model provide a solid foundation for further study.
- High level motion and task planning.
  - In this part we focus on the high level motion and action planner framework for multi-agent case. In particular, we constructed the abstraction of robot motion in workspace and derived the action model first. Then given the LTL formula we can find related buÌĺchi automaton based on the complex robot task specification. Thirdly, we did synchronized product of finite transition system and buÌĺchi automaton and search the optimal path. Last but not the least, we discussed the multi-agent case.

- Low level navigation function based controller.
  - Based on the generated high level path, we need to make sure the hybrid controller can be executed properly following the optimal path while avoid collide with obstacles. We introduced the original navigation function which can generate curves from any initial point to the destination in the sphere free space without colliding obstacles as well as discussed the more general case dealing with multi-agent case(the dynamic obstacle case)
- Simulation and real quadcopter experiment.
  - Firstly, we further illustrated the implementation detail in multi-agent case, especially the combination of motion and action model as well as the precondition and postcondition configuration. Then we verify the validity of this kind of LTL based motion and action planning algorithm in python simulation. In real quadcopter experiment, due to the lack of manipulator, we only discuss the multi-agent motion planning.

6.2 Future Work

In this master thesis, we discuss about symbolic LTL planning. However, there are still some challenges to be solved.

- Büchi automaton generated by LTL2BA is sensitive to the order in LTL formula. Therefore, a new way to develop corresponding büchi automaton may be necessary.
  - In this master thesis, given specific LTL formula, we use the software LTL2BA to generate corresponding büchi automaton. The problem is that, for different LTL formula that represents the same task, for exmaple,we have the task 'visit region A and region B infinitely often',we don't care about visit order, but $(\square\diamondsuit A) \wedge (\square\diamondsuit B)$ and $(\square\diamondsuit B) \wedge (\square\diamondsuit A)$ will have different büchi automaton which may cause one has short path while the other has a longer path. Therefor, it will be difficult to find the shortest path for a long LTL formula. I may find another way to generate the LTL formula.

- On the other hand, I may put more emphasis on the decentralized task planning and study the multi-agent systems with dependent local tasks. That is, we will discuss more complex task that can't be implemented by one agent and need several agent coordinate with each other.

- Last but not the least, I may continue studying the multi-agent(more that two agent) navigation function. In this case, we need to discuss different collision condition: how many agent collide. Then we need to develop the G function we mentioned in chapter 4 to solve this problem. We also need to add the f compensate function to make the whole system more flexible: make sure the agent which already reach its target also can coordinate with other agent and give way to other agent to reach their target region.

Bibliography

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] E. M. Clarke and E. A. Emerson, *Design and synthesis of synchronization skeletons using branching time temporal logic*. Springer, 1981.

[3] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in cesar," in *International Symposium on Programming*, pp. 337–351, Springer, 1982.

[4] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pp. 46–57, IEEE, 1977.

[5] M. Y. Vardi, "Automata-theoretic model checking revisited," in *Verification, Model Checking, and Abstract Interpretation*, pp. 137–150, Springer, 2007.

[6] S. Eker, J. Meseguer, and A. Sridharanarayanan, "The maude ltl model checker," *Electronic Notes in Theoretical Computer Science*, vol. 71, pp. 162–187, 2004.

[7] A. Duret-Lutz and D. Poitrenaud, "Spot: an extensible model checking library using transition-based generalized büchi automata," in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(mascots 2004). Proceedings. the IEEE Computer Society's 12th Annual International Symposium on*, pp. 76–83, IEEE, 2004.

[8] D. Peled, "All from one, one for all: on model checking using representatives," in *Computer Aided Verification*, pp. 409–423, Springer, 1993.

[9] R. Milner, *An algebraic definition of simulation between programs*. Citeseer, 1971.

[10] C. B. Jones, "Specification and design of (parallel) programs.," in *IFIP Congress*, pp. 321–332, 1983.

[11] K. L. McMillan, "The smv language," *Cadence Berkeley Labs*, pp. 1–49, 1999.

[12] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, 2000.

[13] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, *et al.*, "Vis: A system for verification and synthesis," in *Computer Aided Verification*, pp. 428–432, Springer, 1996.

[14] E. Clarke, O. Grumberg, and K. Hamaguchi, "Another look at ltl model checking," in *Computer Aided Verification*, pp. 415–427, Springer, 1994.

[15] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang, "Symbolic model checking: 1020 states and beyond," *Information and computation*, vol. 98, no. 2, pp. 142–170, 1992.

[16] K.L.McMillan, "Symbolic model checking : an approach to the state explosion problem," *Ph.D. thesis*, 1992.

[17] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[18] R. Zhou and E. A. Hansen, "Breadth-first heuristic search," *Artificial Intelligence*, vol. 170, no. 4, pp. 385–408, 2006.

[19] V. Lipiello and F. Ruggiero, "Cartesian impedance control of a uav with a robotic arm," in *10th IFAC Symposium on Robot Control*, 2012.

[20] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[21] C. Baier, J.-P. Katoen, *et al.*, *Principles of model checking*, vol. 26202649. MIT press Cambridge, 2008.

[22] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2689–2696, IEEE, 2010.

[23] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 3, pp. 55–64, 2011.

[24] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Motion and action planning under ltl specifications using navigation functions and action description language," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 240–245, IEEE, 2013.

[25] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI Communications*, no. Preprint, pp. 1–12.

[26] P. Gastin and D. Oddoux, "Fast ltl to büchi automata translation," in *Computer Aided Verification*, pp. 53–65, Springer, 2001.

[27] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[28] J. Bang-Jensen and G. Z. Gutin, *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[29] W. Zeng and R. Church, "Finding shortest paths on real road networks: the case for a*," *International Journal of Geographical Information Science*, vol. 23, no. 4, pp. 531–543, 2009.

[30] T. M. Chan, "More algorithms for all-pairs shortest paths in weighted graphs," *SIAM Journal on Computing*, vol. 39, no. 5, pp. 2075–2089, 2010.

[31] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[32] M. Fox and D. Long, "Pddl+: Modelling continuous time-dependent effects," in *Proc. 3rd International NASA Workshop on Planning and Scheduling for Space*, Citeseer, 2002.

[33] E. P. Pednault, "Adl: Exploring the middle ground between strips and the situation calculus," in *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pp. 324–332, Morgan Kaufmann Publishers Inc., 1989.

[34] M. Guo, "Hybrid control of multi-robot systems under complex temporal tasks," 2015.

[35] D. E. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary," *Advances in Applied Mathematics*, vol. 11, no. 4, pp. 412–442, 1990.

[36] D. V. Dimarogonas, S. G. Loizou, K. J. Kyriakopoulos, and M. M. Zavlanos, "Decentralized feedback stabilization and collision avoidance of multiple agents," *NTUA, http://users. ntua. gr/ddimar/TechRep0401. pdf, Tech. Report*, 2004.

TRITA TRITA-EE 2016:113
ISSN 1653-5146