공학석사학위논문

쿼드로터 무인기의 경로추정을 위한 역 강화학습 제어기법

Inverse Reinforcement Learning Control for Trajectory Tracking of a Quadrotor UAV

2014 년 2 월

서울대학교 대학원

기계항공공학부

최 승 원

쿼드로터 무인기의 경로추정을 위한 역 강화학습 제어기법

Inverse Reinforcement Learning Control for Trajectory Tracking of a Quadrotor UAV

지도교수 김 현 진

이 논문을 공학석사 학위논문으로 제출함

2013 년 12 월

서울대학교 대학원

기계항공공학부

최 승 원

최승원의 공학석사 학위논문을 인준함

2013 년 12 월

위 원 장 기 창 돈

부위원장 김 현 진

위 원 오 성 회

Inverse Reinforcement Learning Control for Trajectory Tracking of a Quadrotor UAV

A Dissertation

by

SEUNGWON CHOI

Presented to the Faculty of the Graduate School of
Seoul National University
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE

School of Mechanical & Aerospace Engineering

Seoul National University

Supervisor : Associate Professor H. Jin Kim

February 2014

to my

MOTHER, FATHER, and BROTHER

with love

Abstract

Inverse Reinforcement Learning Control for Trajectory Tracking of a Quadrotor UAV

Seungwon, Choi
Department of Mechanical & Aerospace Engineering
The Graduate School
Seoul National University

The main purpose of this thesis is to imitate the demonstrations of a quadrotor UAV flown by an expert pilot. First, we collect a data set of several demonstrations by an expert for a certain task which we want to learn. We extract a representative trajectory from the dataset. Hidden Markov model (HMM) and dynamic time warping (DTW) are used for obtaining the trajectory. We extract the sequence of state and input data. But a direct use of the input data can cause the danger in stability. For that reason, a controller is required. We design a reinforcement learning controller with reward function of linear quadratic form. To track the extracted trajectory well, an inverse reinforcement learning algorithm is suggested. Using particle swarm optimization (PSO), the reward function that minimizes the trajectory tracking error is learned. With the simulation and experiment applied to a quadrotor UAV, the successful imitation result is presented.

Keyword : Apprenticeship learning, Quadrotor trajectory tracking, Imitation learning, Trajectory learning, Inverse reinforcement learning

Student Number : 2012-20710

Table of Contents

List of Figures

1

Introduction

Apprenticeship learning, also called learning from demonstration (LfD), is learning a complex task by observing a demonstration of an expert. This framework has been considered for various robotic applications [1]-[7]. It is difficult for a robot to autonomously accomplish a tough task such as aerobatic movement of a helicopter [1], humanoid robot tasks [2],[3], or maneuvering of a quadruped robot in extreme terrain [5]. The demonstrations for the tasks done by an expert can enhance the performance by extracting a trajectory or reward function.

In robotics, a task is often defined as a trajectory and a controller is designed for a robot to follow the trajectory. For a satisfying performance of the complex tasks, the trajectory considered with the robot dynamics is needed. When a teacher is available, a feasible trajectory can be extracted based on the real movements of the robot [1]-[3]. We call this procedure as trajectory learning in this thesis. Instead of learning a trajectory, by obtaining a reward function in reinforcement learning (RL) setting, the robot also can imitate the expert's demonstrations [4]-[7]. This is called inverse reinforcement learning [8],[9]. In the data of the demonstrations, the expert's private reward function is hidden. Inverse reinforcement learning algorithm learns the latent reward function to imitate the performance of the expert. We combine both concepts.

The main purpose of this thesis is to imitate the demonstrations of a quadrotor UAV performed by an expert pilot. First, we extract a representative trajectory from the demonstrations

for a certain tasks. The trajectory is time-specified and includes the state and input informations. Because a direct use of the desired input data can degrade the stability, a controller is needed. We designed a reinforcement learning controller for tracking performance [10]. Additionally we use the concept of inverse reinforcement learning. We suggest an simulation-based inverse reinforcement learning algorithm to find a reward function which minimizes the trajectory tracking error using particle swarm optimization (PSO) [11], a reward function is a linear quadratic form.

1.1 Literature review

Related works on this paper are the trajectory learning and the inverse reinforcement learning. Here, trajectory learning means extracting the desired trajectory and inverse reinforcement learning means the process of finding a good reward function from the learned trajectory.

In [1], they use multiple demonstrations to learn the helicopter aerobatics. They proposed a trajectory learning algorithm using hidden Markov model (HMM) and dynamic time warping (DTW). A method of imitating the humanoid tasks by learning the trajectory with Gaussian mixture regression (GMR) is presented in [2]. In [1],[2], they designed optimal controllers to reproduce the demonstration with the learned trajectory. In [3], an online quantum mixture regression algorithm was suggested for learning the trajectory of a robotic manipulator with high degree of freedom.

In [4], a reward function is obtained using an optimal control techniques for imitating the swing up task of pendulum. In [8],[9], an inverse reinforcement learning algorithm with Markov decision process (MDP) has been proposed. Based on this algorithm, several researches have been progressed [5],[6]. In [5], the algorithm was applied to the locomotion of a quadruped robot in extreme terrain. In [6], they considered a problem for leaning from multiple strategies. Multiple strategies mean the existence of the several reward functions. An optimal reward function was driven from different reward function. In [7], imitating problem from the demonstration of dissimilar robot was solved by extracting the context-dependent reward function using Gaussian mixture model.

1.2 Thesis contribution

The contributions of this paper are the following: first, we apply the trajectory learning algorithm using HMM and DTW to a quadrotor UAV. Considering quadrotor dynamics, a state and input trajectory is learned from a dataset of demonstrations performed by an expert for certain maneuver. For tracking the learned trajectory we design a reinforcement learning controller. Next, we propose an inverse reinforcement learning algorithm using PSO. Based on simulation, a proper reward function is learned to make the quadrotor fly similarly with the extracted trajectory. Finally using the desired trajectory and reward function, we perform the experiment to validate the overall approach.

1.3 Thesis outline

The rest of this paper is organized as follows. In Chapter 2, the quadrotor dynamics is presented. In Chapter 3, a reinforcement learning based feedback controller is designed and its derivation is included. In Chapter 4, a trajectory learning algorithm on quadrotor (Section 4.1) and an inverse reinforcement learning algorithm (Section 4.2) are introduced. In Chapter 5 and Chapter 6, simulation and experiment results are shown. Finally concluding remarks are included in Chapter 7.

2

Quadrotor dynamics

The equation of motion of a quadrotor with the mass m and the moment of inertia J is

$$m\ddot{X} = mge_3 - TRe_3 \tag{2.1}$$

$$\dot{\Omega} = -\Omega \times J\Omega + M \tag{2.2}$$

where X is the position of the quadrotor in the inertial frame, Ω is the angular velocity in the body frame, g is the gravitational acceleration, T is the input thrust in the direction of $-z_B$, M is the input moment in x-y-z axis, and e_3 is the unit vector of z axis. The configuration is in Fig. 2.1. The subscript O means the inertial frame and B means the body frame. The detail of quadrotor dynamics are included in [12], [13].

2.1 Translational dynamics

Translational dynamics of the quadrotor is presented in Eq. (2.1). It can be represented as

$$\begin{aligned} m\ddot{x} &= -T(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) \\ m\ddot{y} &= -T(\cos\phi\sin\theta\sin\psi - sin\phi\cos\psi) \\ m\ddot{z} &= mg - T\cos\phi\cos\theta \end{aligned} \tag{2.3}$$

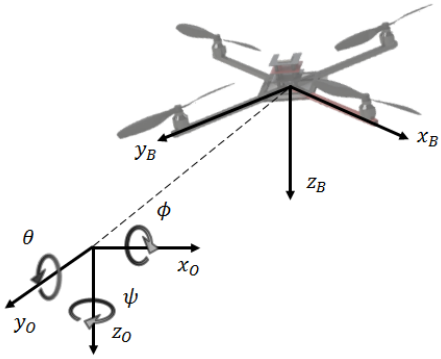

Figure 2.1: Quadrotor configuration

Here, x, y, z are the position and ϕ, θ, ψ are the Euler angle in the inertial frame. With the assumption of small Euler angle which means the quadrotor flies near hover, the translational dynamics can be linearized into,

$$\begin{aligned} m\ddot{x} &= -mg\theta \\ m\ddot{y} &= mg\phi \\ m\ddot{z} &= -\delta T \end{aligned} \tag{2.4}$$

where $\delta T = T - mg$.

2.2 Attitude dynamics

Attitude dynamics of the quadrotor is shown in Eq. (2.2). As in the Section 2.1, using small Euler angle assumption, it can be simplified into

$$\begin{aligned} J_{xx}\ddot{\phi} &= (J_{yy} - J_{zz})\dot{\theta}\dot{\psi} + M_\phi \\ J_{yy}\ddot{\theta} &= (J_{zz} - J_{xx})\dot{\psi}\dot{\phi} + M_\theta \\ J_{zz}\ddot{\psi} &= (J_{xx} - J_{yy})\dot{\phi}\dot{\theta} + M_\psi \end{aligned} \tag{2.5}$$

where J_{xx}, J_{yy}, J_{zz} are the moment of inertia and M_ϕ, M_θ, M_ψ are the input moment generated from the motors along the x, y, z axis. Also if the Euler angular rates are small, the attitudinal

dynamics can be linearized into

$$
\begin{aligned}
J_{xx}\ddot{\phi} &= M_{\phi} \\
J_{yy}\ddot{\theta} &= M_{\theta} \\
J_{zz}\ddot{\psi} &= M_{\psi}.
\end{aligned}
\tag{2.6}
$$

3

Reinforcement learning controller

Markov decision process (MDP) is a common framework for reinforcement learning. In our setting, MDP is used to represent quadrotor dynamics as follows. The set of states are the state of the quadrotor and the set of actions are the input of the quadrotor. The state transition occurs with the following stochastic, discrete linear model,

$$\bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k + w_k \tag{3.1}$$

where $\bar{x}$ is the state vector, $\bar{u}$ is the input vector, A and B are system matrices, and w_k is the gaussian noise with zero mean at iteration k. We assume the reward function to be a linear quadratic form:

$$r_k = \bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k \tag{3.2}$$

r_k is the reward at time step k. Q and R are the positive definite matrix. Value function V_k is defined as the expected sum of discounted reward function.

$$E\left[V_k\right] = E\left[\sum_{i=k}^{\infty} \gamma^{i-k} r_i\right] = \sum_{i=k}^{\infty} E\left[\gamma^{i-k} r_i\right] \tag{3.3}$$

where γ is the discount factor and has the value between 0 and 1 and E is an operator for expectation. Bellman optimality equation can be obtain from Eq. (3.3).

$$E\left[V_k\right] = E\left[\bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k\right] + E\left[\gamma V_{k+1}\right] \tag{3.4}$$

Because the system model is linear and the reward function has the form of linear quadratic, assume that

$$V_k = \bar{x}_k^T P \bar{x}_k \tag{3.5}$$

$$\bar{u}_k = -K\bar{x}_k \tag{3.6}$$

Here, P is the positive definite matrix and K is the optimal gain matrix. Subsituting the Eqs. (3.1), (3.5) and (3.6) into Eq. (3.4),

$$\begin{aligned} E\left[\bar{x}_k^T P \bar{x}_k\right] &= E\left[\bar{x}_k^T Q \bar{x}_k + \bar{x}_k^T K^T R K \bar{x}_k\right] \\ &\quad + E\left[\gamma \bar{x}_k^T (A - BK)^T P (A - BK) \bar{x}_k\right] \end{aligned} \tag{3.7}$$

3.1 Policy iteration

Policy iteration is a method to find the optimal policy by iterating two processes consisting of policy evaluation and policy improvement. Policy evaluation is to update the value function with a given policy. In our work, P is newly evaluated in given K through policy evaluation. In the policy improvement, the optimal policy, K, is updated using the new value function, P.

First, we will derive the update rule for policy evaluation at iteration j, with P^j and K^j. From Eq. (3.7),

$$\begin{aligned} E\left[\bar{x}_k^T P^{j+1} \bar{x}_k\right] &= E\left[\bar{x}_k^T Q \bar{x}_k + \bar{x}_k^T (K^j)^T R K^j \bar{x}_k\right] \\ &\quad + E\left[\gamma \bar{x}_k^T (A - BK^j)^T P^j (A - BK^j) \bar{x}_k\right] \end{aligned} \tag{3.8}$$

Because of E is linear operator, the new P^{j+1} is driven directly from Eq. (3.8).

$$P^{j+1} = Q + (K^j)^T R K^j + \gamma (A - BK^j)^T P^j (A - BK^j) \tag{3.9}$$

Next, with the newly updated P^{j+1}, from the Eq. (3.7),

$$\begin{aligned} E\left[\bar{x}_k^T P^{j+1} \bar{x}_k\right] &= E\left[\bar{x}_k^T Q \bar{x}_k + \bar{x}_k^T (K^{j+1})^T R K^{j+1} \bar{x}_k\right] \\ &\quad + E\left[\gamma \bar{x}_k^T (A - BK^{j+1})^T P^{j+1} (A - BK^{j+1}) \bar{x}_k\right] \end{aligned} \tag{3.10}$$

Policy, K^{j+1}, which optimizes the Eq. (3.10) is the improved value with updated value function. Because the linear quadratic form is a scalar quantity and both E and trace operator, tr, are linear operator, $E\left[\bar{x}^T P \bar{x}\right] = tr\left(E\left[\bar{x}^T P \bar{x}\right]\right) = E\left[tr\left(\bar{x}^T P \bar{x}\right)\right]$ holds. And by the cyclic property of the

trace operator, $E\left[tr\left(\bar{x}^T P\bar{x}\right)\right] = E\left[tr\left(P\bar{x}\bar{x}^T\right)\right]$ also holds. With those properties, Eq. (3.10) becomes,

$$\begin{aligned} tr\left(P^{j+1}\chi\right) &= tr\left(Q\chi + (K^{j+1})^T R K^{j+1}\chi\right) \\ &\quad + tr\left(\gamma(A - BK^{j+1})^T P^{j+1}(A - BK^{j+1})\chi\right) \end{aligned}$$

where $\chi = E\left[\bar{x}\bar{x}^T\right]$. Taking the derivative by K^{j+1},

$$\left[RK^{j+1} - \gamma B^T P^{j+1}\left(A - BK^{j+1}\right)\right]\chi = 0$$

$$K^{j+1} = \gamma\left(R + \gamma B^T P^{j+1} B\right)^{-1} B^T P^{j+1} A \tag{3.11}$$

By repeating this update, an improved policy is obtained.

Now we summarize the policy iteration. First, P^0 and K^0 should be initialized, $j = 0$. Second, we evaluate P^{j+1} using Eq. (3.9) with P^j and K^j from previous iteration. Next, improved K^{j+1} is obtained using Eq. (3.11) from newly updated P^{j+1}. Then repetition of the above two processes until P and K converge yields the optimal policy K. The convergence was proved in [14].

3.2 Trajectory tracking controller

In our problem, the trajectory tracking controller is needed. Because the desired trajectory was extracted from the trajectory learning algorithm, it is feasible,

$$\bar{x}^*_{k+1} = A\bar{x}^*_k + B\bar{u}^*_k \tag{3.12}$$

where $\bar{x}^*$ is the desired state vector and $\bar{u}^*$ is the desired input vector. Assume that,

$$\tilde{x}_k \triangleq \bar{x}_k - \bar{x}^*_k,\ \tilde{u}_k \triangleq \bar{u}_k - \bar{u}^*_k$$

The error dynamics is given,

$$\tilde{x}_{k+1} = A\tilde{x}_k + B\tilde{u}_k + w_k \tag{3.13}$$

Then, the same derivation in Section 3.1 is possible. With the optimal gain K,

$$\begin{aligned} r_k &= \tilde{x}_k^T Q\tilde{x}_k + \tilde{u}_k^T R\tilde{u}_k \\ \tilde{u}_k &= -K\tilde{x}_k \\ \bar{u}_k &= \bar{u}^*_k - K\left(\bar{x}_k - \bar{x}^*_k\right) \end{aligned} \tag{3.14}$$

3.3 Quadrotor controller

A quadrotor is under-actuated system with highly coupled states. In general, controller of the quadrotor is composed of position and attitude controller, because the attitude of the quadrotor is vulnerable to disturbance. In our work, we design a controller with RL controller in Section 3.1 for control the position and yaw angle, and a classical PID controller for roll and pitch angle control. The configuration of the controller is in Fig. 3.1. Because most of the tasks of quadrotor is performed near hover, we can use the linear dynamics, Eqs. (2.4) and (2.6).

$$\bar{x} \triangleq \left[\ x\ y\ z\ \phi\ \theta\ \psi\ \dot{x}\ \dot{y}\ \dot{z}\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}\ \right]^T$$
$$\bar{u} \triangleq [\ T\ M_\phi\ M_\theta\ M_\psi\ ]^T$$

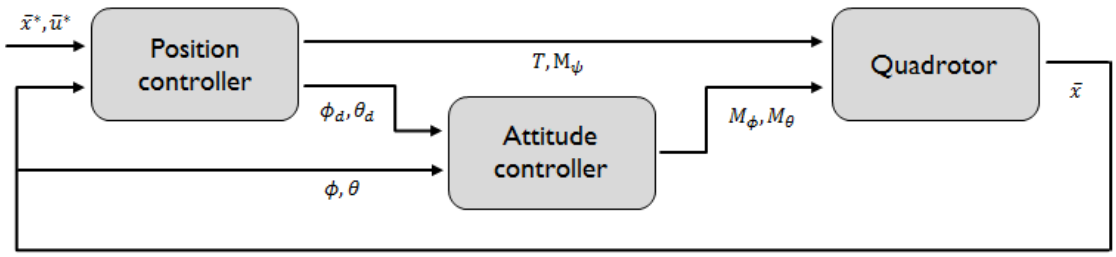

Figure 3.1: Quadrotor controller diagram

As in Fig. 3.1, the output of the RL controller are the T, M_ψ, ϕ_d, and θ_d. The linear dynamics used in RL controller is

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\psi} \end{pmatrix} = \begin{pmatrix} -g\theta_d \\ g\phi_d \\ -\delta F/m \\ M_\psi/J_{zz} \end{pmatrix} \tag{3.15}$$

Using Euler discretization with sampling time dt, discrete linear dynamics can be obtained.

With the RL controller derived in Section 3.2,

$$\begin{pmatrix} T \\ M_\psi \\ \phi_d \\ \theta_d \end{pmatrix}_k = \begin{pmatrix} T^* \\ M_\psi^* \\ \phi^* \\ \theta^* \end{pmatrix}_k - K \left[\begin{pmatrix} x \\ y \\ z \\ \psi \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\psi} \end{pmatrix}_k - \begin{pmatrix} x^* \\ y^* \\ z^* \\ \psi^* \\ \dot{x}^* \\ \dot{y}^* \\ \dot{z}^* \\ \dot{\psi}^* \end{pmatrix}_k \right] \tag{3.16}$$

ϕ_d and θ_d are used in the roll and pitch attitude controller. We use a classical PID controller as follows,

$$\begin{aligned} M_\phi &= -K_p(\phi - \phi_d) - K_d(\dot{\phi}) - K_i \int (\phi - \phi_d)\mathrm{d}t \\ M_\theta &= -K_p(\theta - \theta_d) - K_d(\dot{\theta}) - K_i \int (\theta - \theta_d)\mathrm{d}t \end{aligned} \tag{3.17}$$

4

Inverse reinforcement learning control

In this chapter, trajectory learning algorithm (Section 4.1) and inverse reinforcement learning algorithm (Section 4.2) are introduced. The purpose of trajectory learning algorithm is to extract the representative trajectory among the demonstrations. Let the demonstration trajectory, $\bar{y}$, and the representative trajectory, $\bar{z}$, be,

$$\bar{y}_j^l \triangleq \begin{bmatrix} \bar{x}_j^l \\ \bar{u}_j^l \end{bmatrix} \tag{4.1}$$

$$\bar{z}_t \triangleq \begin{bmatrix} \bar{x}_t^* \\ \bar{u}_t^* \end{bmatrix} \tag{4.2}$$

for $j = 0, ..., N^l - 1,\ l =, 1..., L,\ t = 0, ..., T^* - 1$. $\bar{x}$ is the state vector, $\bar{u}$ is the input vector, N^l is the maximum time step for each demonstration, L is the total demonstration number, and T^* is the maximum time step for the trajectory. We use the probabilistic model to obtain the intended trajectory $\bar{z}$ (Section 4.1).

A simulation-based inverse reinforcement learning algorithm is suggested to make the quadrotor fly similarly to the trajectory which is learned in trajectory learning algorithm. Trajectory tracking RL controller we designed in Chapter 3 is used. We assume that the reward function is a linear quadratic form. The object of the algorithm is obtain the reward function which minimizes the trajectory tracking error. The details are given in Section 4.2.

4.1 Trajectory learning algorithm

The concept of trajectory learning algorithm is considering the demonstrations as the observations of one intended trajectory. It is expressed as

$$\bar{z}_{t+1} = f\left(\bar{z}_t\right) + w_t^{\bar{z}}, \quad w^{\bar{z}} \sim \mathcal{N}(0, \Sigma^{\bar{z}}) \tag{4.3}$$

$$\bar{y}_j^l = h(\bar{z}_{\tau_j^l}) + w_t^{\bar{y}^l}, \quad w^{\bar{y}^l} \sim \mathcal{N}(0, \Sigma^{\bar{y}^l}) \tag{4.4}$$

$$\tau_j^l \sim \mathbb{P}(\tau_{j+1}^l | \tau_j^l) \tag{4.5}$$

where $w^{\bar{z}}$ and $w^{\bar{y}}$ are the system and observation noise with gaussian distribution and τ_j^l is the time index of the observation in j-th time step of k-th observations. τ is assumed to be under the multinomial distribution. We use the Eqs. (2.3), (2.5) and Euler discretization for $f(\bar{z})$.

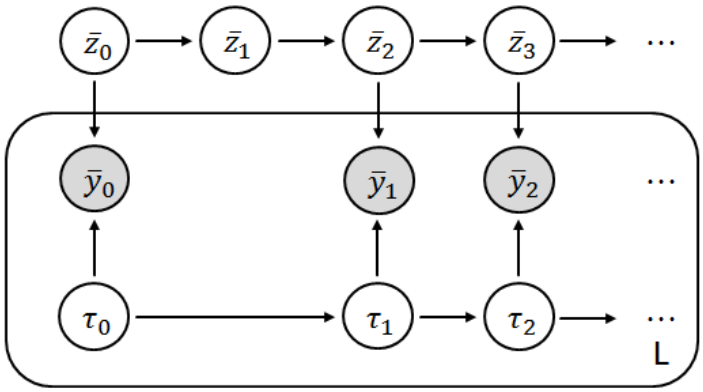

Figure 4.1: Concept of trajectory learning algorithm

It can be also represented in graphical model in Fig. 4.1. Circles are random variables and filled circles mean observed variables.

We want to find the hidden trajectory $\bar{z}_t$ which maximizes the log-likelihood,

$$\max_{\tau, \Sigma^{(\cdot)}} \log \mathbb{P}(\bar{y}, \tau; \Sigma^{(\cdot)}) \tag{4.6}$$

However, it is difficult to optimize the likelihood over Σ and τ at once. We maximize Eq. (4.6) through two EM algorithms. First, we update the covariance matrix $\Sigma^{(\cdot)}$ with fixed τ (Subsection 4.1.1). Next, we reassign τ for new $\Sigma^{(\cdot)}$ (Subsection 4.1.2).

4.1.1 EM algorithm with hidden Markov model

With fixed τ, Fig. 4.1 becomes a form of standard hidden Markov model (Fig. 4.2).

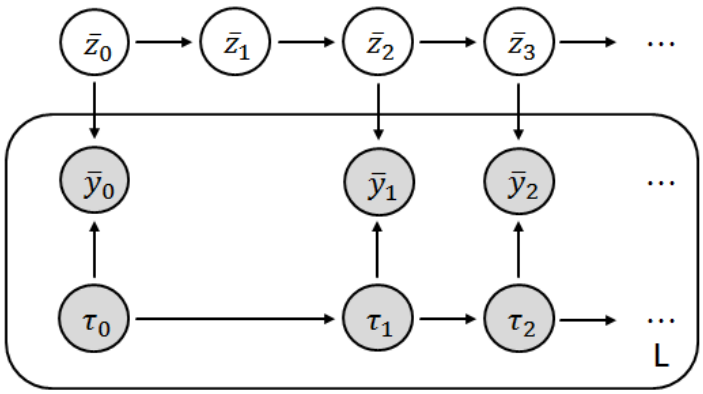

Figure 4.2: Hidden Markov model

In this subsection, covariance which maximizes the Eq. (4.6) will be updated. First in the E-step, the distribution, $\mathcal{N}(\bar{\mu},\ \Sigma)$, of the latent variable, $\bar{z}$ is evaluated with the current $\Sigma^{(\cdot)}$ and τ by using extended Kalman filter and smoother. The M-step uses the distribution to update the covariance matrix $\Sigma^{(\cdot)}$. We have

$$\begin{aligned} \bar{z}_{t+1} &= f(\bar{z}_t) + w_t^{\bar{z}}, \quad w^{\bar{z}} \sim \mathcal{N}(0, \Sigma^{\bar{z}}) \\ \bar{y}_{t+1} &= h(\bar{z}_t) + w_t^{\bar{y}}, \quad w^{\bar{y}} \sim \mathcal{N}(0, \Sigma^{\bar{y}}) \end{aligned}$$

In the E-step, extended Kalman filter computes the $\mu_{t+1|t}$, $\Sigma_{t+1|t}$, $\mu_{t|t}$, $\Sigma_{t|t}$ through forward prediction and measurement update along time t. Let the F_t and H_t be

$$\begin{aligned} F_t &= \left.\frac{\partial f}{\partial \bar{z}}\right|_{\bar{\mu}_{t|t}} \\ H_t &= \left.\frac{\partial h}{\partial \bar{z}}\right|_{\bar{\mu}_{t|t-1}} \end{aligned}$$

In the prediction step,

$$\begin{aligned} \bar{\mu}_{t+1|t} &= f(\bar{\mu}_{t|t}) \\ \Sigma_{t+1|t} &= F_t \Sigma_{t|t} F_t^T + \Sigma^{\bar{z}} \end{aligned} \tag{4.7}$$

In the filtering step, with the observations $\bar{y}$,

$$\begin{aligned}
\bar{r}_t &= \bar{y}_t - h(\bar{\mu}_{t|t-1}) \\
S_t &= H_t \Sigma_{t|t-1} H_t^T + \Sigma^{\bar{y}} \\
K_t &= \Sigma_{t|t-1} H_t^T S_t^{-1} \\
\bar{\mu}_{t|t} &= \bar{\mu}_{t|t-1} + K_t \bar{r}_t \\
\Sigma_{t|t} &= (I - K_t H_t) \Sigma_{t|t-1}
\end{aligned} \tag{4.8}$$

Finally in the smoothing process, $\bar{\mu}_{t|T}$ and $\Sigma_{t|T}$ are computed through the backward pass.

$$\begin{aligned}
L_t &= \Sigma_{t|t} F_t^T \Sigma_{t+t|t}^{-1} \\
\bar{\mu}_{t|T} &= \bar{\mu}_{t|t} + L_t(\bar{\mu}_{t+1|T} - \bar{\mu}_{t+1|t}) \\
\Sigma_{t|T} &= \Sigma_{t|t} + L_t(\Sigma_{t+1|T} - \Sigma_{t+1|t}) L_t^T
\end{aligned} \tag{4.9}$$

Using the extended Kalman filter and smoother, Eqs. (4.7),(4.8), and (4.9), the distribution of the latent variable $\bar{z}$ can be evaluated. In the M-step, we can use the computed distributions to update the covariance matrix $\Sigma^{\bar{z}}$, $\Sigma^{\bar{y}}$. Here, we redefine the F_t, H_t as

$$\begin{aligned}
F_t &= \left.\frac{\partial f}{\partial \bar{z}}\right|_{\bar{\mu}_{t|T}} \\
H_t &= \left.\frac{\partial h}{\partial \bar{z}}\right|_{\bar{\mu}_{t|T-1}}
\end{aligned}$$

The update equations are the following.

$$\begin{aligned}
\delta\bar{\mu}_t &= \bar{\mu}_{t+1|T} - f(\bar{\mu}_{t|T}) \\
\delta\bar{y}_t &= \bar{y}_t - h(\bar{\mu}_{t|T})
\end{aligned}$$

$$\begin{aligned}
\Sigma^{\bar{z}} = \frac{1}{T} \sum_{t=0}^{T-1} \{ & \delta\bar{\mu}\delta\bar{\mu}^T + \Sigma_{t+1|T} - \Sigma_{t+1|T} L_t^T F_t^T \\
& - F_t L_t \Sigma_{t+1|T} + F_t \Sigma_{t|T} F_t^T \} \\
\Sigma^{\bar{y}} = \frac{1}{T+1} \sum_{t=0}^{T} \{ & \delta\bar{y}_t \delta\bar{y}_t^T + H_t \Sigma_{t|T} H_t^T \}
\end{aligned} \tag{4.10}$$

With the assumption of fixed time index, we can learn the most likely hidden trajectory over covariance matrix $\Sigma^{(\cdot)}$. Then we can use the computed quantities to update the time index.

4.1.2 EM algorithm with dynamic time warping

The approach in this subsection is not needed when the exact time indices of all observations are known. When they are unknown, the observations should be aligned properly. Dynamic time warping (DTW) is a sequence alignment algorithm by measuring similarity between two different signals. We align the time index which maximizes the log-likelihood, Eq. (4.6).

$$\begin{aligned}
\bar{\tau} &= \arg\max_{\tau} \log \mathbb{P}(\bar{y}, \tau; \Sigma) \\
&= \arg\max_{\tau} \log \mathbb{P}(\bar{y}; \bar{\mu}_{t|T}, \tau)\mathbb{P}(\bar{\mu}_{t|T})\mathbb{P}(\tau) \\
&= \arg\max_{\tau} \log \mathbb{P}(\bar{y}; \bar{\mu}_{t|T}, \tau)\mathbb{P}(\tau) \\
&= \arg\max_{\tau} \sum_{l=1}^{L} \sum_{j=0}^{N^l-1} [\log p(\bar{y}_j^l | \bar{z}_{\tau_j^l|T}, \tau_j^l) + \log p(\tau_j^l | \tau_{j-1}^l)]
\end{aligned} \tag{4.11}$$

We define a new quantity $s = 0, ..., N^l - 1$, $t^{'} \in \{t-3,\ t-2,\ t-1\}$, and $Q(s,t)$,

$$\begin{aligned}
Q(s,t) =& \log p(\bar{y}_s | \bar{z}_{\tau_{s|T}}, \tau_s = t) \\
&+ \max_{t^{'}} \left[\log p(\tau_s = t | \tau_{s-1}) + \sum_{j=0}^{s-1} \{\log p(\bar{y}_j | \bar{z}_{\tau_{j|T}}, \tau_j) + \log p(\tau_j | \tau_{j-1})\} \right] \\
=& \log p(\bar{y}_s | \bar{z}_{\tau_{s|T}}, \tau_s = t) + \max_{t^{'}} \left[\log p(\tau_s = t | \tau_{s-1} = t^{'}) + Q(s-1, t^{'}) \right]
\end{aligned} \tag{4.12}$$

For $s = 0, ..., N^l - 1$, we compute $Q(s,t)$. From the maximum value of $Q(N^l - 1, t)$ for each observation, we can assign the new τ_js by finding which $t^{'}$ is used for maximization in Eq. (4.12). Used $t^{'}$s are the time index difference which maximizes the log-likelihood between two adjacent observation data. With the reassigned observations, we compute the distribution of τ using standard maximum likelihood estimation for multinomial distribution.

Through Subsection 4.1.1 and 4.1.2, we can obtain most likely hidden trajectory over observations which maximizes the log-likelihood, Eq. (4.6). The trajectory includes the time-specified state and input data. We call this trajectory as desired trajectory in the next section and it is denoted as $(\bar{x}^*, \bar{u^*})$.

4.2 Inverse reinforcement learning algorithm

In this section, we propose an inverse reinforcement learning algorithm. We assume that the reward function is a linear quadratic form.

$$r_k = \tilde{x}_k^T Q \tilde{x}_k + \tilde{u}_k^T R \tilde{u}_k \tag{4.13}$$

The objective of the algorithm is obtain $Q,\ R$ which minimizes the trajectory tracking error.

$$\arg\min_{Q,R} J = \arg\min_{Q,R} \sum \{W_{\bar{x}}^T (\bar{x}_{k+1} - \bar{x}_{k+1}^*)^2 + W_{\bar{u}}^T (\bar{u}_k - \bar{u}_k^*)^2\} \tag{4.14}$$

where $W_{\bar{x}}$ and $W_{\bar{u}}$ are the weight matrix for each state and input, $\bar{x}_{k+1}$ and $\bar{u}_k$ is calculated in simulation using the trajectory tracking RL controller with the specific Q and R. For minimizing Eq. (4.14), we utilize particle swarm optimization (PSO). PSO is a computational method to improve the candidate solutions iteratively. PSO does not need the gradient of the objective function which is hard to obtain for stochastic system. However it does not guaranteed to find the global optimum. For applying PSO, several particles over search space and a quality which should be optimized are needed. Because we want to find Q and R that minimize J in Eq. (4.14), the value of Q and R becomes the particle and we measures the quality of each particle using J. The followings are the detail of the algorithm.

1. Initialize the position of several candidate particles,

$$w_i = [Q_1, ..., Q_n, R_1, ...R_m] \tag{4.15}$$

   for $i = 1, ..., n_{cand}$. Particles are spread throughout search space which is composed of the values of Q and R. The element of w_i should be larger than zero.

2. Simulate the quadrotor using the trajectory tracking RL controller in Chapter 3, Eqs. (3.16) and (3.17) for $k = 1, ..., T^*$.

$$\bar{u}_k = \bar{u}_k^* - K(\bar{x}_k - \bar{x}_k^*)$$

3. Calculate the cost function for each particle,

$$J_i = \sum_{k=0}^{T^*-1} \{W_{\bar{x}}^T (\bar{x}_{k+1} - \bar{x}_{k+1}^*)^2 + W_{\bar{u}}^T (\bar{u}_k - \bar{u}_k^*)^2\} \tag{4.16}$$

   $W_{\bar{x}}$ and $W_{\bar{u}}$ are the weight matrix.

4. Evaluate the local best known position w_i^{lb} for each particle. Here, best known position means the position when the particle has minimum cost, J.

5. Evaluate the global best known position w_i^{gb} among all particles.

6. Update the position of the candidate particles,

$$w_i = w_i + 2c_1(w_i^{lb} - w_i) + 2c_2(w_i^{gb} - w_i) \tag{4.17}$$

   c_1 and c_2 are the random numbers between 0 and 1.

7. Repeat 2. to 6. until J converges.

After running the algorithm, w_i^{gb} is the learned value of reward function Q and R which minimizes the trajectory tracking error.

Some properties of the algorithm are:

- The large number of particles has a chance to find better optimal solution. But it takes more computational time.

- Because the same ratio of Q and R has same role, the actual magnitude of their initial value can vary.

- The weight matrices $W_{\bar{x}}$ and $W_{\bar{u}}$ can be easily chosen by using the value which is inverse proportional to the unit of each state and input.

5

Simulation

5.1 Simulation setup

To validate the trajectory learning and inverse reinforcement learning algorithms we perform two simulations: way-point tracking and circuit tracking. Fig. 5.1 and Fig. 5.2 indicates the trajectory which the robot should follow for each simulations. We collect the demonstration data using conventional PID controller in the simulation environment. 5 and 10 demonstrations are used for way-point tracking and circuit tracking simulation respectively. We set the different maximum time for each demonstration to check out the effect of DTW in trajectory learning algorithm. The quadrotor model parameters used in simulation are in table 5.1.

Parameter	Symbol	Value
Mass	m	0.8kg
Inertia tensor (x)	J_{xx}	$0.008\text{kg} \cdot \text{m}^2$
Inertia tensor (y)	J_{yy}	$0.008\text{kg} \cdot \text{m}^2$
Inertia tensor (z)	J_{zz}	$0.016\text{kg} \cdot \text{m}^2$

Table 5.1: Quadrotor model parameters in simulation

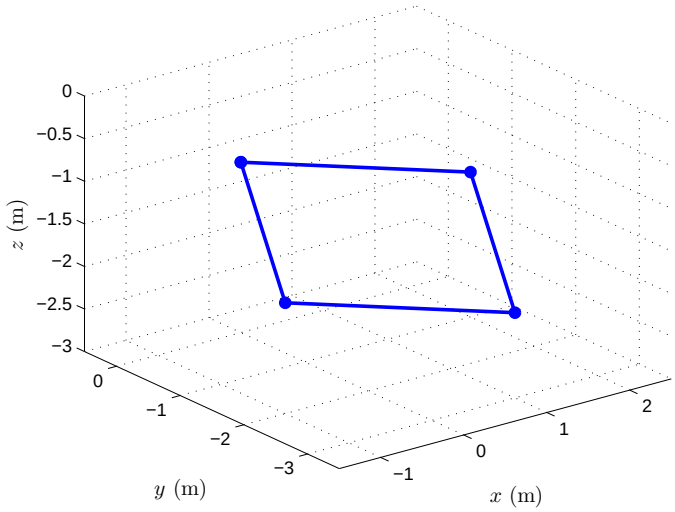

Figure 5.1: Trajectory of way-point tracking

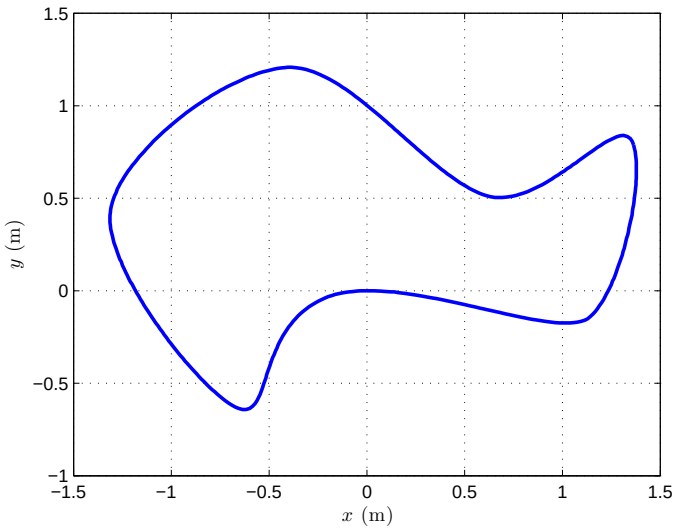

Figure 5.2: Trajectory of circuit tracking

5.2 Way-point tracking simulation result

First, Fig. 5.3 $\sim$ Fig. 5.8 show the way-point tracking simulation result. Fig. 5.3 $\sim$ Fig. 5.5 represent the result of trajectory learning algorithm. In Fig. 5.3, we can see the effect of DTW. While it seems that the trajectory which does not use DTW is more similar to the mean value of the demonstrations, the slope of the trajectory with DTW is similar to those of demonstrations. It means that the trajectory includes the exact behavior with demonstrations regardless of time difference by using DTW. In the following simulation and experiment, DTW is always applied. Fig. 5.4 plots the demonstration data and learned trajectory in 3-D space. Fig. 5.5 displays all state and input of demonstration and extracted trajectory.

Inverse reinforcement learning algorithm learns the reward function which minimizes the trajectory tracking error. Fig. 5.6 presents the convergence of the cost function, Eq. (4.16). It means that the proper reward function which minimizes the error locally is learned. With the reward function and RL controller in Chapter 3, the performance of the quadrotor is in Fig. 5.7 and Fig. 5.8. We can see that the quadrotor follows the extracted trajectory well.

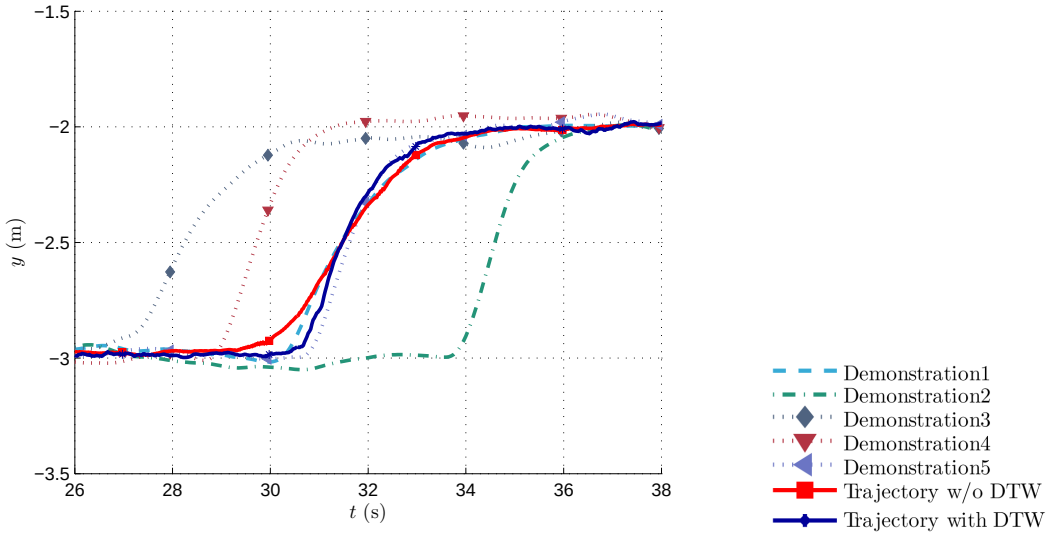

Figure 5.3: Effect of dynamic time warping (DTW)

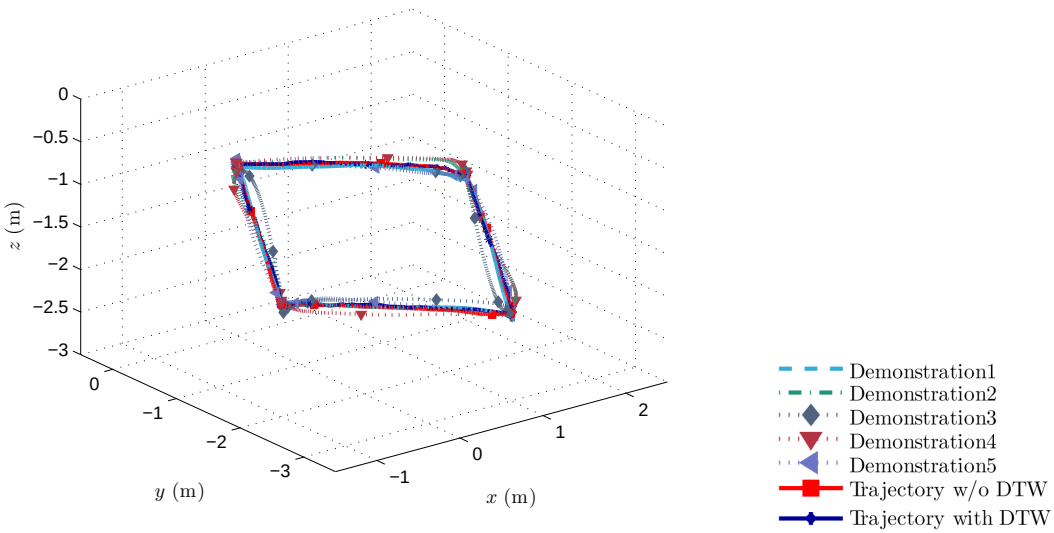

Figure 5.4: Result of trajectory learning for way-point tracking (1)

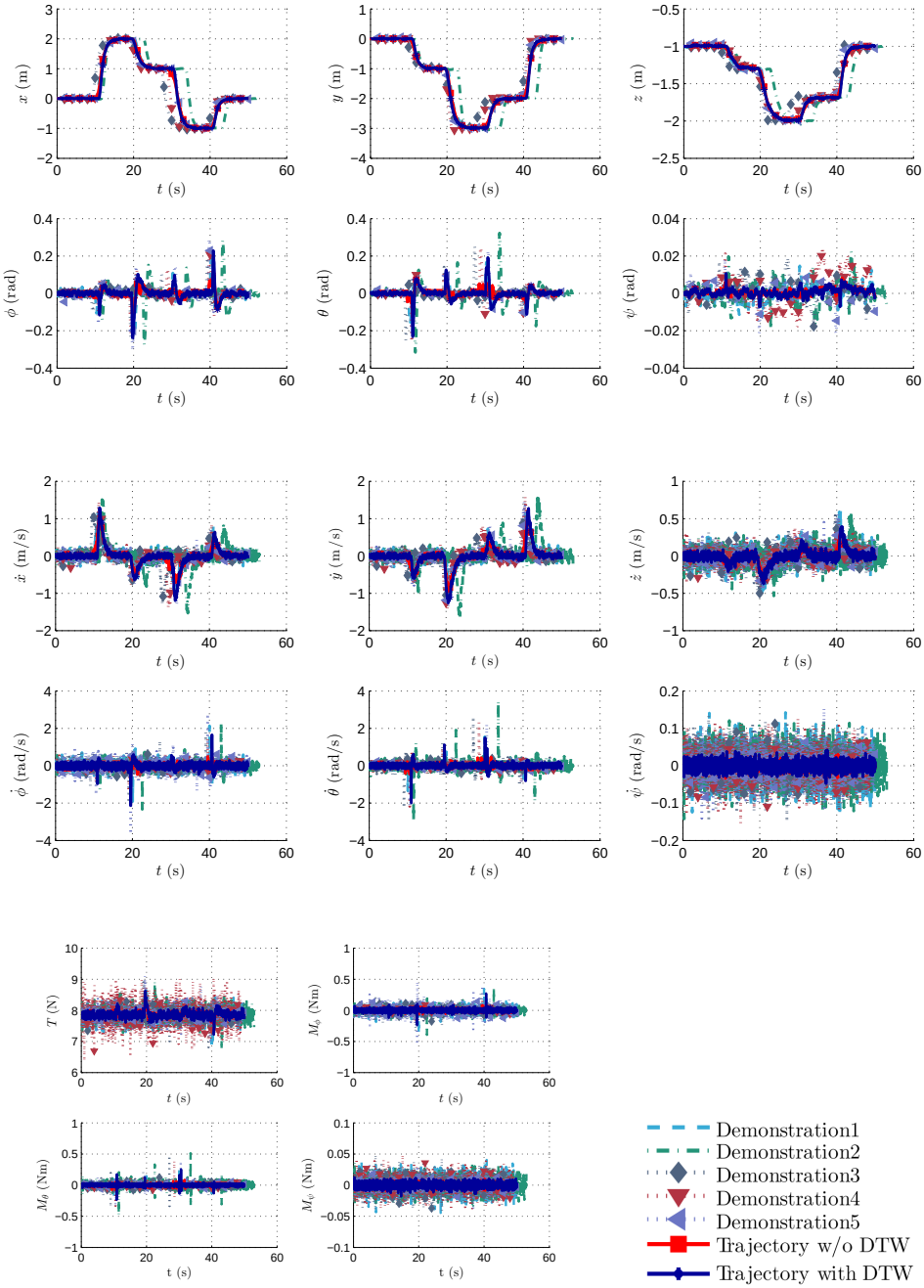

Figure 5.5: Result of trajectory learning for way-point tracking (2)

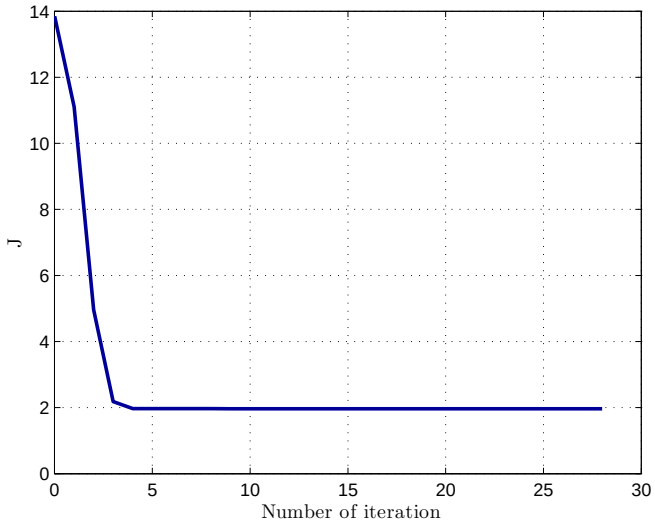

Figure 5.6: Cost in inverse reinforcement learning algorithm for way-point tracking

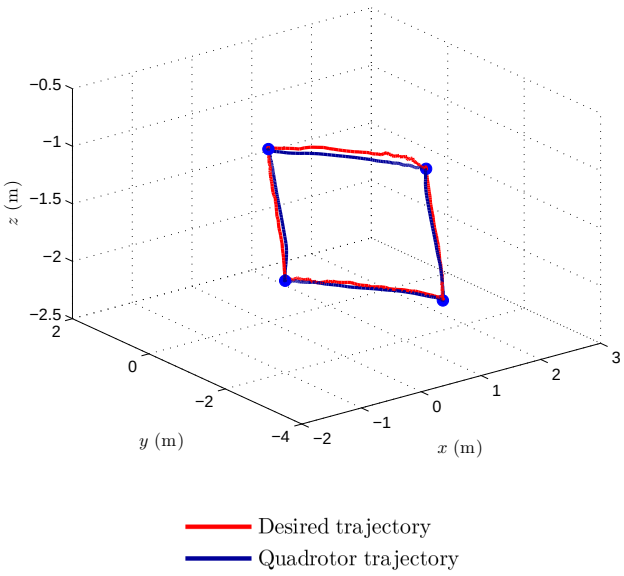

Figure 5.7: Performance of quadrotor with learned reward function for way-point tracking (1)

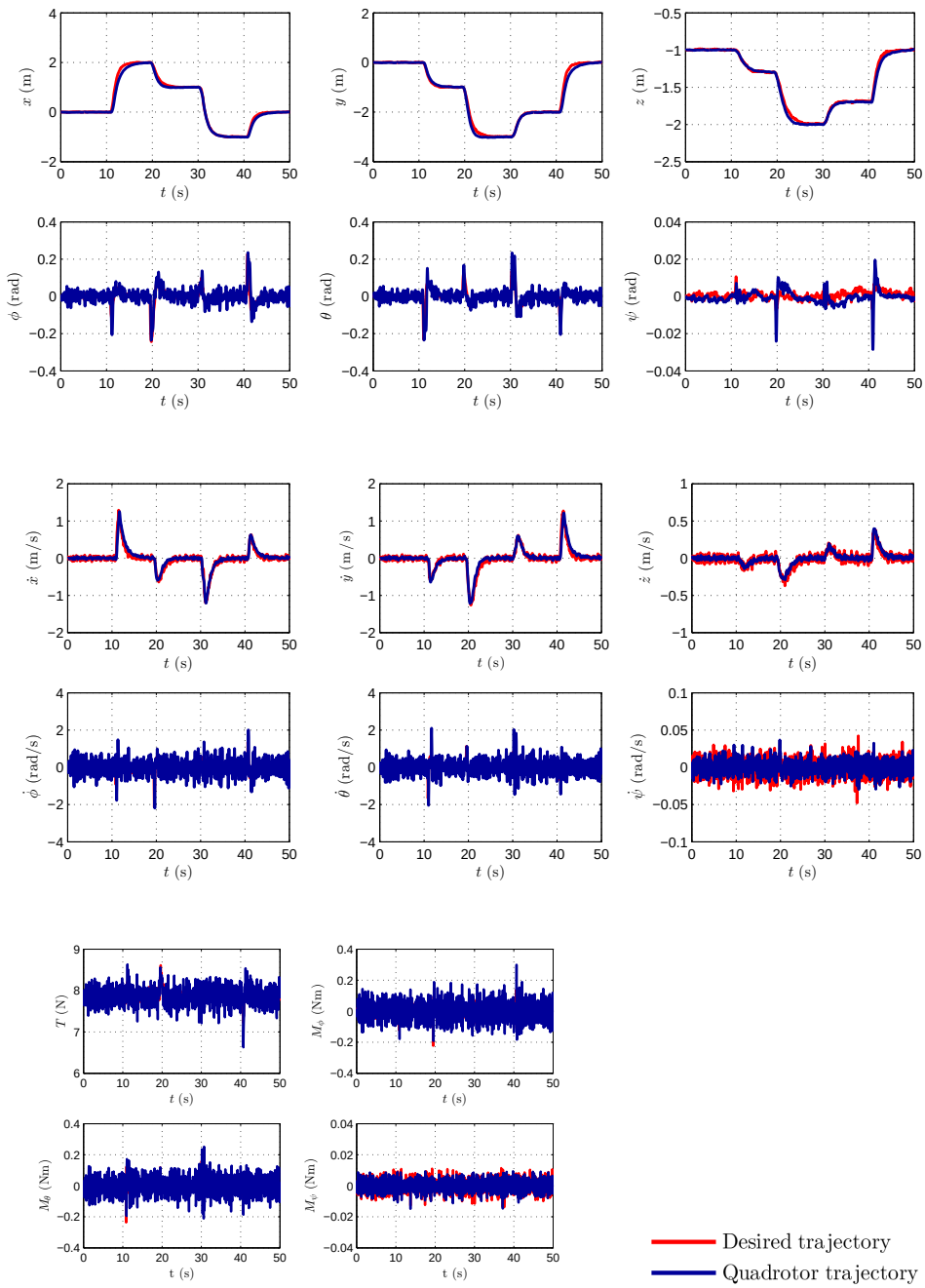

Figure 5.8: Performance of quadrotor with learned reward function for way-point tracking (2)

5.3 Circuit tracking simulation result

Same as a way-point tracking simulation, Fig. 5.9 $\sim$ Fig. 5.13 present the result of trajectory learning and inverse reinforcement learning algorithm for circuit tracking. In Fig. 5.9 and Fig. 5.10, a representative trajecoty is extracted from 10 demonstrations. Using Kalman filter and smoother, variances in demonstrations and noise are reduced during the trajectory includes overall tendency of the demonstrations.

Fig. 5.11 shows the cost function, Eq. (4.16), when inverse reinforcement learning algorithm runs. We can see that the cost function converges to small value. Fig. 5.12 and Fig. 5.13 indicate that the trajectory tracking error is small, that is the reward function is well learned. For the simulation, the apprenticeship learning algorithm in this thesis works well. In the next section, we will perform an experiment to validate the applicability of the algorithm in a real quadrotor system.

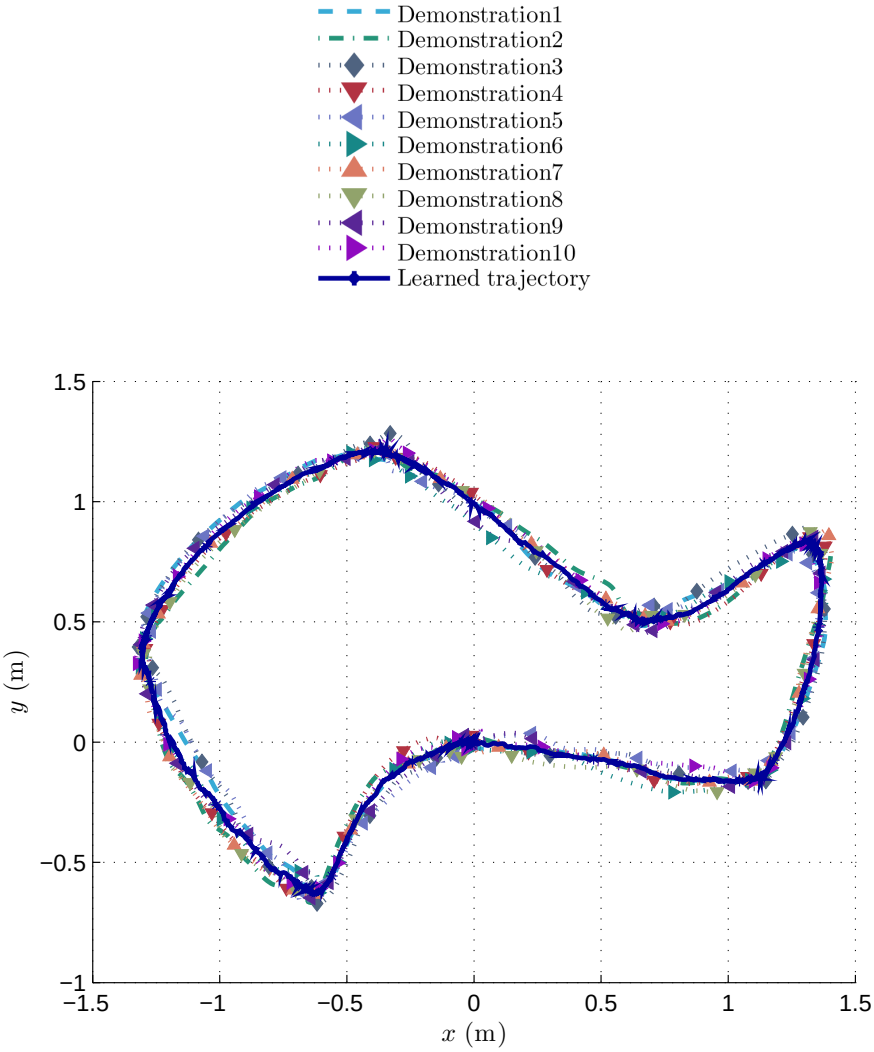

Figure 5.9: Result of trajectory learning for circuit tracking (1)

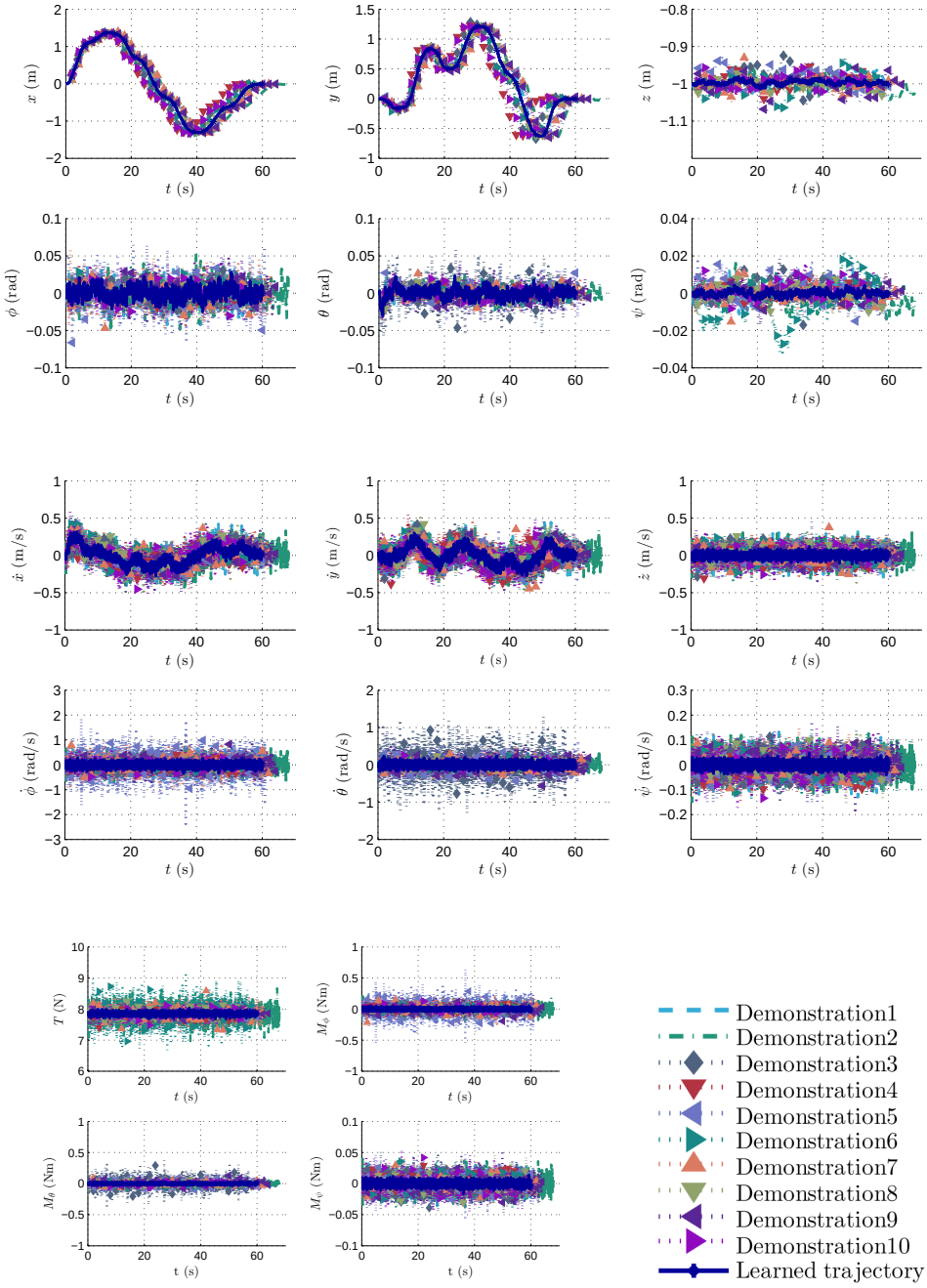

Figure 5.10: Result of trajectory learning for circuit tracking (2)

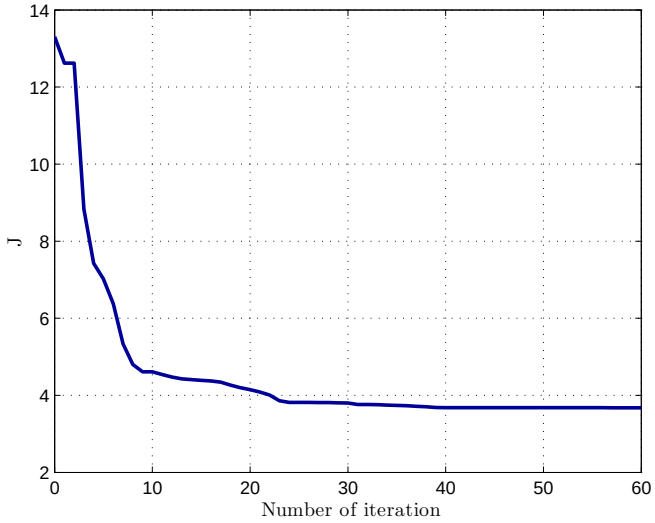

Figure 5.11: Cost in inverse reinforcement learning algorithm for circuit tracking

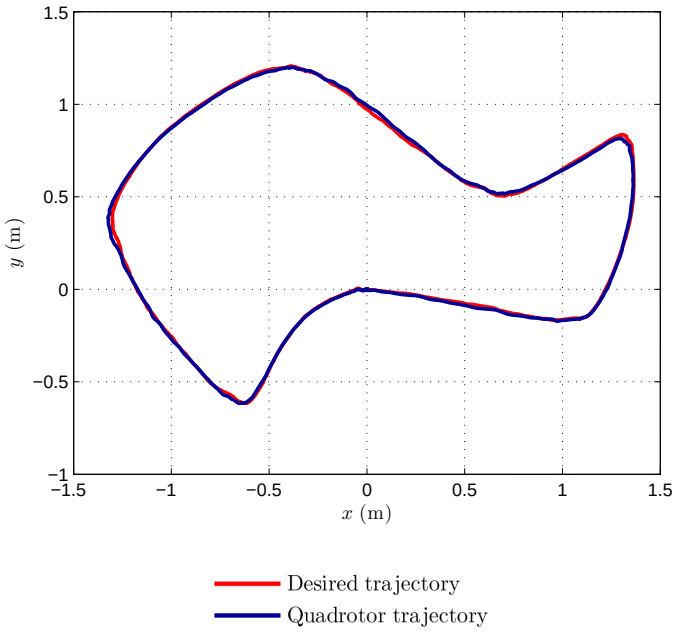

Figure 5.12: Performance of quadrotor with learned reward function for circuit tracking (1)

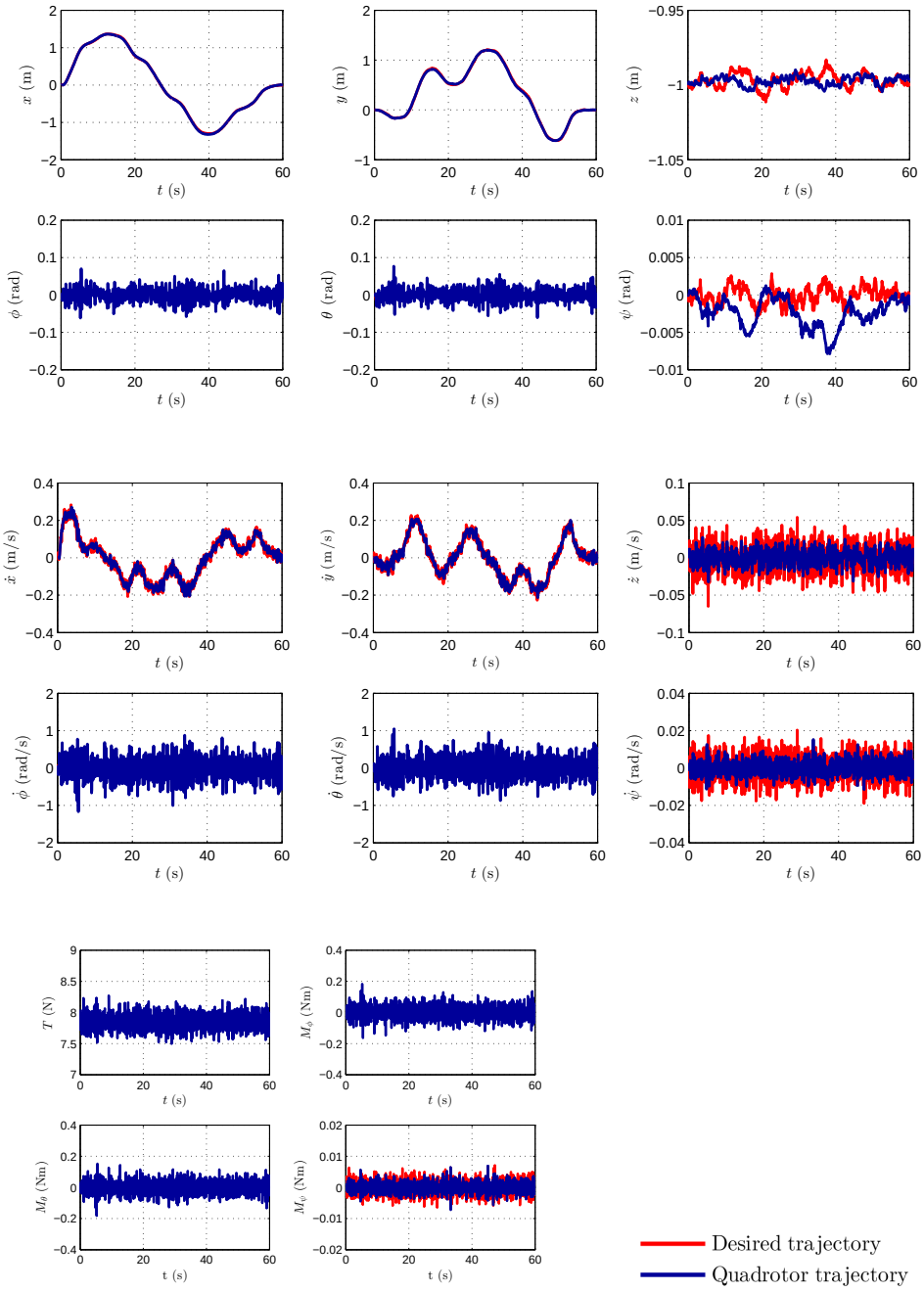

Figure 5.13: Performance of quadrotor with learned reward function for circuit tracking (2)

6

Experiment

6.1 Experiment setup

The hardware setup for experiment appears in Fig. 6.1. First, Vicon, a motion capture system estimates the position of the quadrotor. In the ground station, the position data from vicon is transmitted to an onboard computer of quadrotor through wifi module. On the quadrotor, the onboard computer, AHRS (Attitude and heading reference system), and ESC (Electric speed controller) with motors are equipped. AHRS is composed of gyro sensor, accelerometer, and magnetometer. It estimates the attitude (Euler angle and angular rate) of the quadrotor. Onboard computer receives position and attitude of the quadrotor from vicon and AHRS. Using those data, control inputs are computed. The control inputs are used for activating the motor through ESC. Control sequence is represented in Fig. 6.2. As mentioned before, the controller consists of position controller and attitude controller. The position controller runs at 25Hz using position data from Vicon. Attitude data from AHRS and desired roll and pitch angle from the position controller are used in the attitude controller at 200Hz frequency. The controller computes the desired rpm for each motors to control the quadrotor. Through ESC, 4 motors rotate as commanded by the controller. The quadrotor used in experiment has 1.15kg mass, (0.012,0.012,0.024)kg $\cdot$ m^2 inertia tensor, and 0.23m arm length. The task for experiment is tracking the eight(8)-shaped trajectory two times maintaining the altitude. seven demonstrations are performed by a pilot.

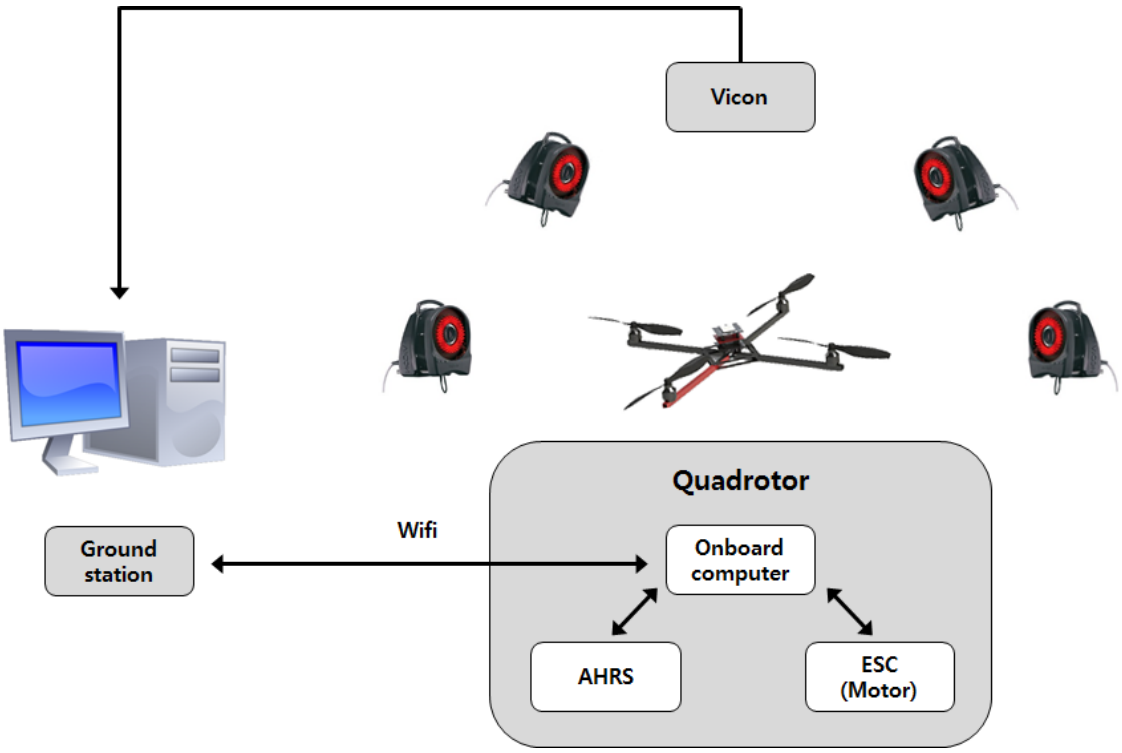

Figure 6.1: Hardware setup configuration for experiment

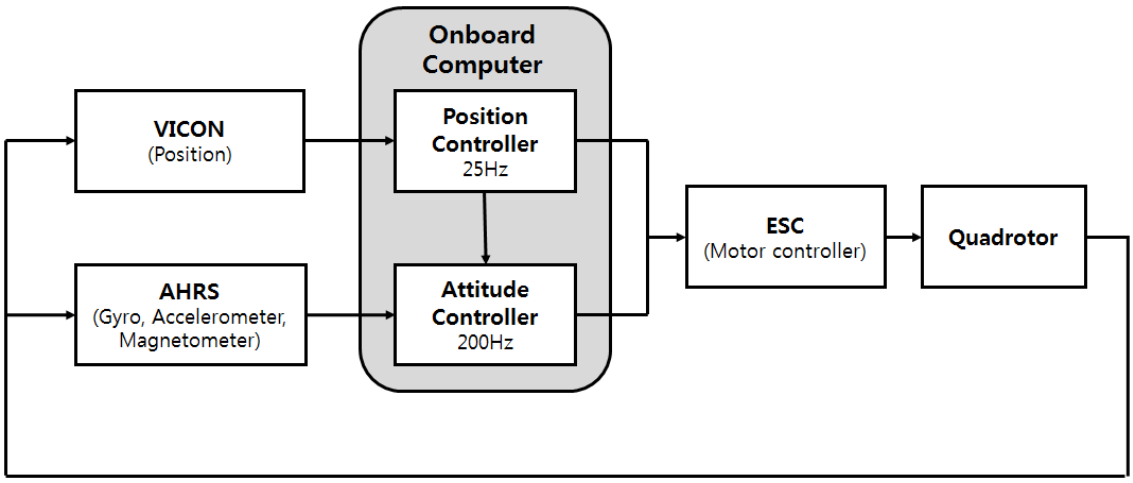

Figure 6.2: Configuration of the control system

6.2 Experiment result

Experiment results are shown in Fig. 6.3 $\sim$ Fig. 6.7. Fig. 6.3 and Fig. 6.4 represents the result of trajectory learning algorithm with real flight data. In the Fig. 6.5, we can see that the reward function which minimizes the trajectory tracking error is learned. With the extracted trajectory and learned reward function, we perform the task using the same quadrotor which is used for demonstrations. The results are in Fig. 6.6 and Fig. 6.7. Althogh the suggested inverse reinforcement learning algorithm in this thesis is based on simulation, the experiment result is acceptable. The altitude, z only has some noticeable error. It's because the relation between desired thrust and the thrust of motor is influenced by the battery. It can be reduced by improving the control structure such as adding error integration term. By applying the introduced algorithm to quadrotor, we can imitate the demonstrations of eight-shaped trajectory tracking.

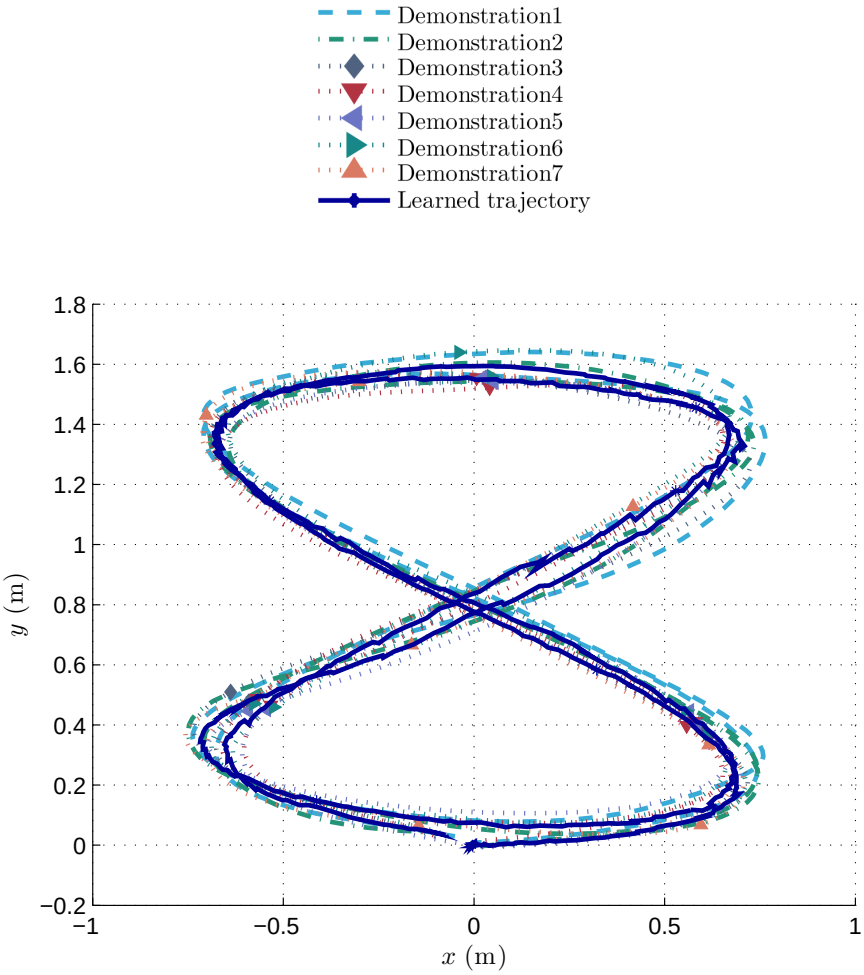

Figure 6.3: Result of trajectory learning for experiment (1)

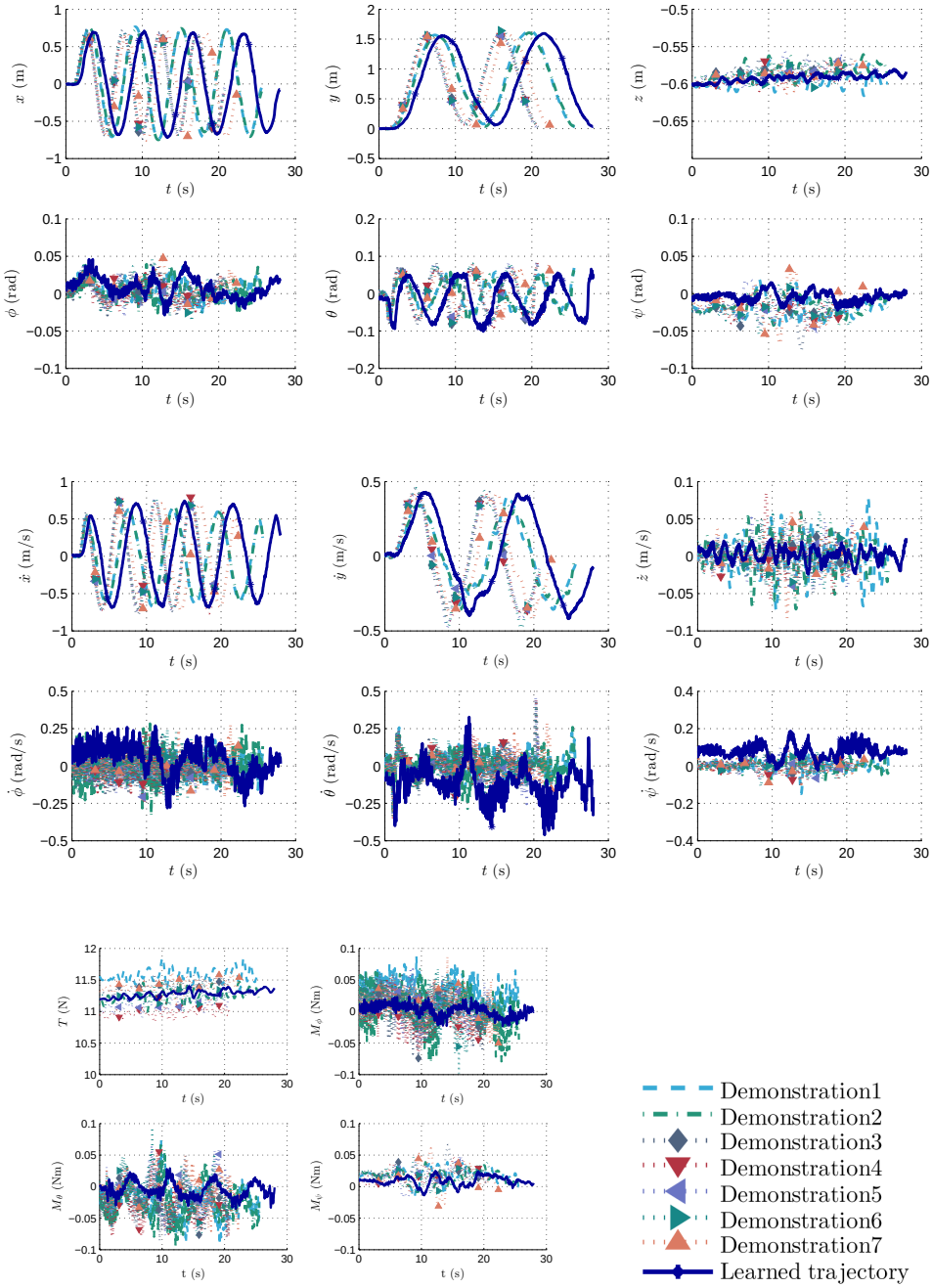

Figure 6.4: Result of trajectory learning for experiment (2)

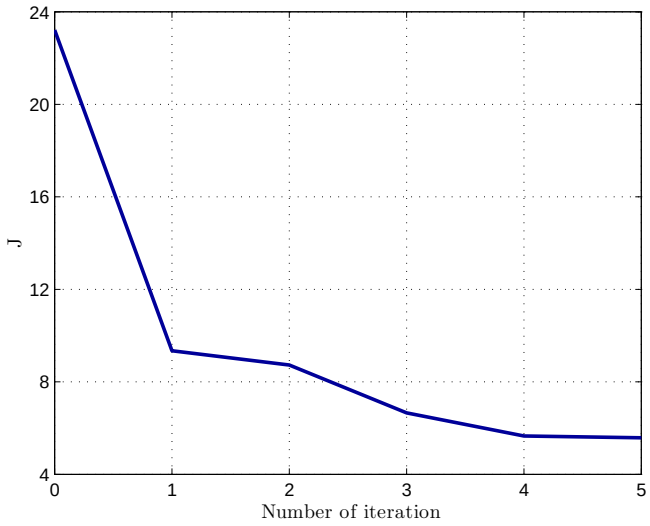

Figure 6.5: Cost in inverse reinforcement learning algorithm for experiment

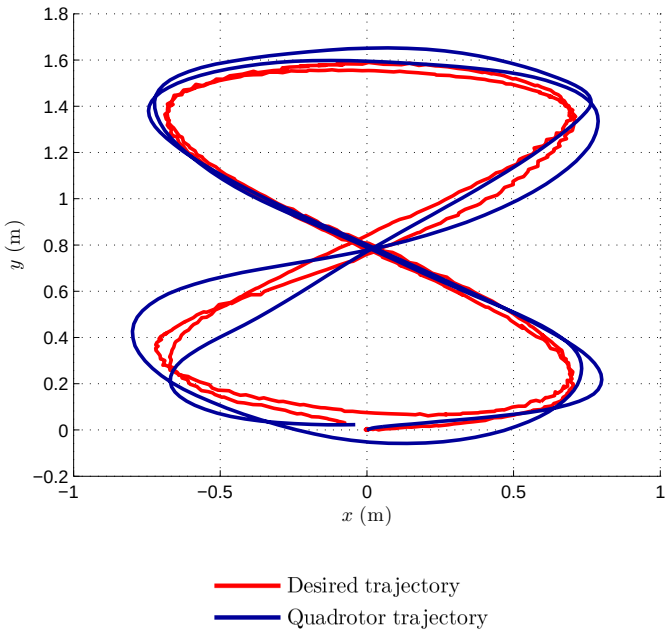

Figure 6.6: Performance of quadrotor with learned reward function for experiment(1)

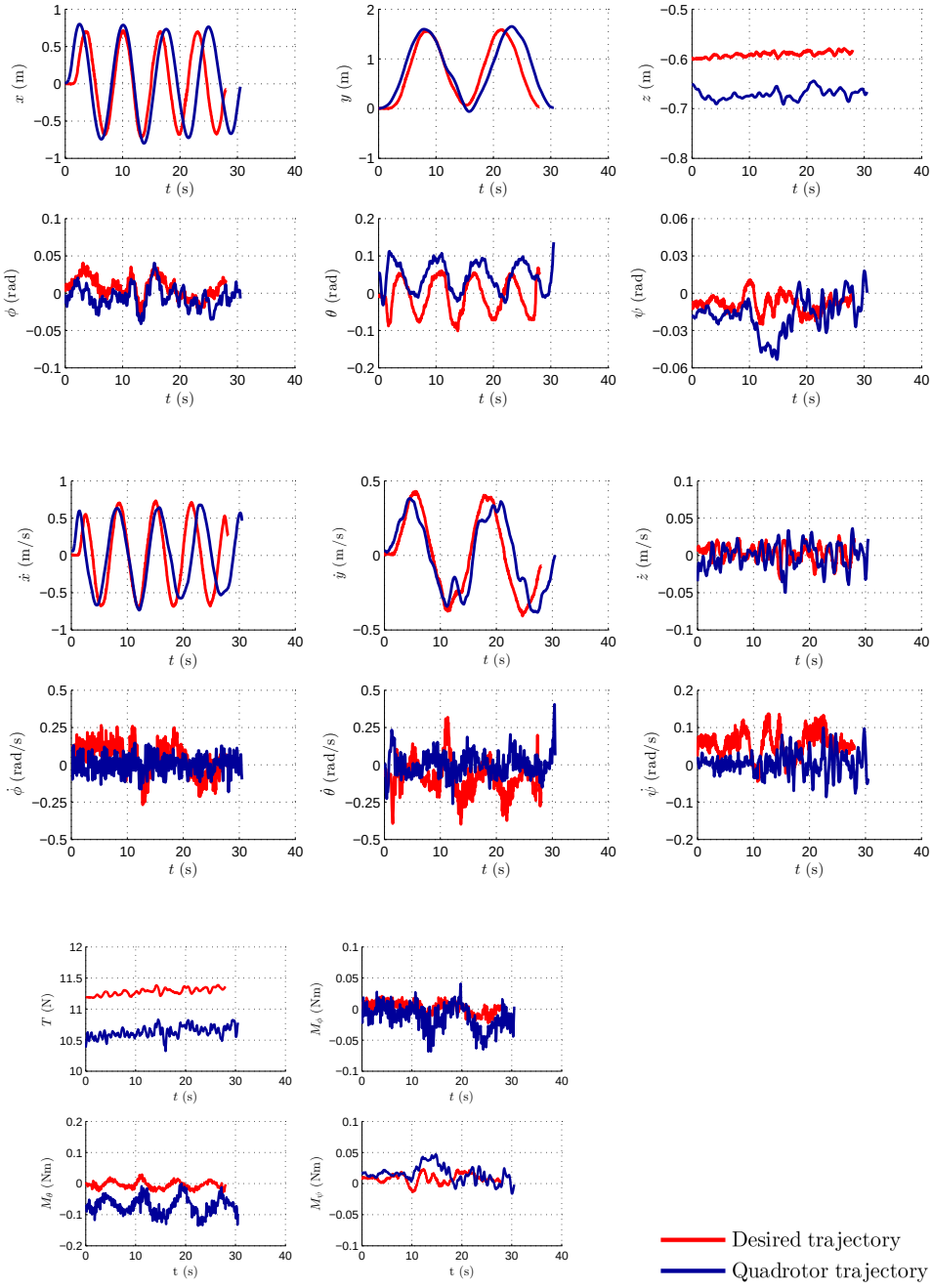

Figure 6.7: Performance of quadrotor with learned reward function for experiment(2)

7 Conclusion

In this thesis, we proposed an apprenticeship learning algorithm to imitate the demonstrations and applied it to a quadrotor UAV. To imitate the demonstrations, we divide the process into two parts; first, we extract a representative trajectory from demonstrations which a quadrotor should follow to behave like demonstrations. Then the controller learns the reward function which follows the extracted trajectory perfectly. The former part is called trajectory learning and the latter is inverse reinforcement learning. Through simulation and experiment results, we validate the algorithms. Using trajectory learning algorithm using hidden Markov model (HMM) and dynamic time warping (DTW), the most likely trajectory which describes the demonstrations well is learned. The inverse reinforcement learning algorithm with particle swarm optimization (PSO) let the reinforcement learning based controller learn the good reward function to track the extracted trajectory. In simulation and experiments, successful imitation results are obtained. The quadrotor flies similar to demonstrations with the learned trajectory and reward function. Overall we can conclude that the proposed apprenticeship learning works well for the quadrotor UAV. As future work, system identification or improved control structure can improve the algorithm, because the optimization of reward function is performed in the simulation environment.

References

[1] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Jounal of Robotics Research*, vol. 29, no. 13, pp. 1608-1639, 2010.

[2] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART B*, vol. 37, no. 2, pp. 286-298, 2007.

[3] D. Korkinof, and Y. Dimiris, "Online quantum mixture regression for trajectory learning by demonstration," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[4] C. G. Atkeson, S. Schaal, "Robot learning from demonstration," *Proceedings of the 14th International Conference on Machine Learning, ICML'97*, 1997.

[5] J. Z. Kolter, P. Abbeel, A. Y. Ng, "Hierarchical apprenticeship learning with application to quadruped locomotion," *Proceedings of the Advances in Neural Information Processing, NIPS'08*, 2008.

[6] A. K. Tanwani, and A. Billard, "Transfer in inverse reinforcement learning for multiple strategies," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[7] M. S. Malekzadeh, D. Bruno, S. Calinon, T. Nanayakkara, and D. G. Caldwell, "Skills transfer across dissimilar robots by learning context-dependent rewards," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[8] A. Y. Ng, and S. Russell, "Algorithms for inverse reinforcement learning," *Proceedings of the 17th International Conference on Machine Learning*, 2000.

[9] P. Abbeel, and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," *Proceedings of the 21st International Conference on Machine Learning*, 2004.

[10] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control," *IEEE Control Systems Magazine*, Vol. 32, Issul. 6, pp. 76-105, 2012.

[11] J. Kennedy, and R. Eberhart, "Particle Swarm Optimization," *Proceedings of the 1995 IEEE International Conference on Neural Networks*, vol. 4, pp. 1942-1948, 1995.

[12] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying," *Lausanne Polytechnic University*, 2007.

[13] H. B. Lee, "Trajectory tracking of a quadrotor UAV using geometric-based backstepping control," *Seoul National University*, 2013.

[14] G. A. Hewer, "An iterative technique for the conputation of the steady state gains for the discrete optimal regulator," *IEEE Transactions on Automation Control*, 1971.

국 문 초 록

본 논문에서는 전문가로부터 구현된 쿼드로터의 시범 데이터를 모방하는 것을 목표로 한다. 먼저 학습하고자 하는 임무를 동일하게 수행하는 여러번의 데이터를 수집한다. 전문가의 시범을 모방하기 위하여 먼저 여러 시범 데이터를 잘 나타내는 하나의 경로를 추출한다. 본 논문에서는 hidden Markov model (HMM)과 dynamic time warping (DTW)를 이용하여 경로를 학습하는 알고리즘을 소개하였다. 이 때 쿼드로터가 추종해야 할 상태와 그 때의 제어 입력 값을 학습한다. 이 제어 입력을 바로 사용할 경우 쿼드로터의 안정성을 보장할 수 없기 때문에 추가적인 제어기가 필요하다. 본 논문에서는 선형 이차 형태의 보상함수를 갖는 강화학습 기반 제어기를 설계하였다. 쿼드로터가 학습된 경로를 잘 추종하도록 역 강화학습 알고리즘을 제안하여 궤적 추종 오차를 최소화 하는 보상함수를 학습하였다. 이 때 입자 군집 최적화 (PSO) 방법을 이용하여 학습을 진행하였다. 시뮬레이션과 실험을 통하여 소개한 알고리즘을 검증하고 이를 쿼드로터에 적용하여 성공적인 모방 결과를 얻었다.

주요어 : 도제 학습, 모방 학습, 쿼드로터, 경로 학습 알고리즘, 역 강화학습 알고리즘

학번 : 2012-20710