Department of Informatics

Trajectory Learning for Highly Aerobatic Unmanned Aerial Vehicle

Master Thesis

Maja Celine Sevaldson

August 14, 2012

Abstract

The dynamics of fixed wing planes are well understood within the conventional flight envelope. The situation is different in the case of aerobatic maneuvers with a large angle of attack, such as perching and vertical hover. In such maneuvers the airflows around the plane are unpredictable making it difficult to create accurate dynamic models, which would normally be needed for the design of conventional controllers. Yet human RC pilots are able to fly these maneuvers with fixed wing planes.

Apprenticeship learning provides a promising solution to the problem of automating highly aerobatic maneuvers. It allows the maneuver to be learned from demonstration flights done by a human RC pilot rather than relying on an accurate dynamics model.

The focus of this thesis is on a specific issue in apprenticeship learning, namely how to infer the trajectory the pilot intended to fly from a set of suboptimal demonstration trajectories. Such a trajectory can be used as a target trajectory for an autonomous controller. A trajectory learning algorithm that has shown promising results in automation of aerobatic helicopter flight is applied to a fixed wing UAV platform.

The algorithm is tested on two different maneuvers; A straight line of level flight, and a vertical hover maneuver. In the case of both maneuvers the algorithm learned the intended trajectory without prior knowledge about the trajectory.

In order to collect training data for the trajectory learning task, an appropriate platform and data acquisition system are needed. This thesis therefore also presents the development of a fixed wing UAV platform for research on automation of aerobatic flight.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Thesis overview 1
1.2.1 Supporting contributions 2
1.2.2 Thesis outline 2
1.3 Related work 3
1.3.1 Stanford Autonomous Helicopter 3
1.3.2 Aerobatic maneuvers 4

2 Background **7**
2.1 Hidden Markov Models 7
2.1.1 Discrete Markov Processes 7
2.1.2 Hidden Markov Model 8
2.2 Expectation-Maximization 10
2.2.1 Maximum Likelihood Estimation 11
2.2.2 EM 11
2.3 Extended Kalman Smoother 12
2.3.1 Discrete Kalman filter 12
2.3.2 Extended Kalman filter 13
2.3.3 Kalman smoother 13
2.4 Dynamic Time Warping 14

3 Trajectory Learning **17**
3.1 State modelling 17
3.1.1 Time alignment 18
3.2 The trajectory learning algorithm 19
3.2.1 E-step for hidden trajectory 20
3.2.2 M-step for hidden trajectory 20
3.2.3 Time alignment 21
3.3 Dynamic Model 22
3.3.1 Equations of motion 22
3.4 Improvements of the model 23

4 Experimental platform **25**
4.1 Plane 26
4.1.1 Dynamical properties of the plane 27
4.1.2 Modifications to the airframe 29

4.1.3 Motor 29
4.1.4 Radio Control System 30
4.2 Microcontrollers, Sensors and Data Acquisition 31
4.2.1 ArduPilot Mega 32
4.2.2 Inertial Measurement Unit 32
4.2.3 GPS 33
4.2.4 Barometric pressure sensor 34
4.2.5 Sonar 36
4.2.6 Pilot Inputs 36
4.2.7 Data Acquisition and Storage 37
4.3 Test environment 37
4.4 Alternative Positioning Systems 39
4.5 Simulator 41

5 Experiments and Results **43**
5.1 Preliminary tests 43
5.1.1 GPS accuracy 43
5.1.2 Outdoor flight tests 44
5.2 Collecting training data 45
5.2.1 Linear flight and banked turns 46
5.2.2 Hovering with transitions 47
5.3 Pre-processing 47
5.3.1 Selecting good demonstrations 49
5.3.2 BPM filtering and calibration 49
5.3.3 Localization of Position Data 51
5.3.4 IMU data 52
5.3.5 Pilot control inputs 55
5.4 Results 57
5.4.1 Time alignment 57
5.4.2 Straight line of level flight 59
5.4.3 Vertical hover 62
5.4.4 Model Prediction Error 64

6 Conclusion **67**
6.1 Discussion 67
6.1.1 Platform 67
6.1.2 Time alignment 68
6.1.3 Trajectory learning 68
6.2 Future work 69
6.3 Conclusion 70

List of Figures

2.1 Three state Markov model of the weather 9
2.2 Hidden Markov Model of crazy soda dispenser 9
2.3 Dynamic time warping. The figure is adapted from [31]. 14
2.4 Dynamic time warping distance matrix. The figure is borrowed from [40]. 15

3.1 Model of the relation between the hidden trajectory and one of the demonstrations. The coloured nodes are unobserved. . 19
3.2 Model of the state when τ_j^k is fixed. Coloured nodes are unobserved. 19

4.1 Expert pilot flying the plane in vertical hover. 25
4.2 Electrifly Yak 54 3D 26
4.3 Reinforcement of the motor mount 30
4.4 Measured thrust as a function of control input. 30
4.5 7 channel Futaba RC system 31
4.6 Diagram of DAQ system 32
4.7 Electronics mounted on the plane 33
4.8 ArduPilot Mega board 34
4.9 MicroStrain 3DM-GX3 IMU 34
4.10 MediaTek 3329 GPS 35
4.11 Bosch Sensortech BMP085 digital pressure sensor inside plastic enclosure 35
4.12 Maxbotix ultrasonic rangefinder sonar 36
4.13 Outdoor test space. 37
4.14 Indoor flight in the CEPSUM gym. 38
4.15 The Stinger Dome of Concordia University in Montreal 38
4.16 Principle of stereo computer vision 39
4.17 2D tracking based on colour tresholding 40

5.1 On ground GPS test at McGill downtown campus. The red line marks the actual path travelled, while the yellow line shows the logged GPS data. 43
5.2 Comparison of GPS and BPM altitude data 44
5.3 In air tests of DAQ system and GPS logging. 45
5.4 Hand launch of the plane for a training data collection flight in the Stinger Dome. 46
5.5 Mark-up of the Stinger Dome field for collecting training data. 47

5.6 Vertical hover in the Stinger Dome. 48
5.7 Vertical hover. 48
5.8 East - North plot of a good linear flight demonstration. 49
5.9 Position plot and pitch angle from a good hover demonstration 50
5.10 BPM data is filtered using a 3rd order Butterworth low-pass filter with 1 Hz cut-off frequency. 50
5.11 Uncalibrated BPM data and Sonar reference signal 51
5.12 Uncalibrated BPM data and Sonar reference at the two regions where calibration points were selected. 51
5.13 Calibrated BPM data and Sonar reference signal 52
5.14 Zoomed in view of the two calibration regions after calibration 52
5.15 IMU frame and body frame of the plane. 53
5.16 Aileron deflection as a function of input pulse width 55
5.17 Elevator deflection as a function of input pulse width 56
5.18 Rudder deflection as a function of input pulse width 56
5.19 Thrust in Newtons vs PWM command 57
5.20 Linear velocities ($\dot{n}\dot{e}\dot{d}$) from the linear flight. 58
5.21 Angular Velocities around the plane z-axis (r). 58
5.22 East-North position of level flight (meters) 59
5.23 Position and linear velocity of the plane during a straight line of level flight. 60
5.24 Orientation during level flight (Euler Angles). 60
5.25 Angular velocities pqr (rad/sec), level flight. 61
5.26 Control inputs, level flight. 61
5.27 Hover maneuver. 62
5.28 Hover maneuver - Linear velocities and position. 63
5.29 Angular Velocities, hover flight (rad/sec). 63
5.30 Orientation during hover (Euler angles). 63
5.31 Control inputs, hover maneuver. 64
5.32 Model prediction error in linear accelerations ($\dot{u}\dot{v}\dot{w}$) during straight flight. 65
5.33 Model prediction error in angular accelerations ($\dot{p}\dot{q}\dot{r}$) during straight flight. 65
5.34 Model prediction error in linear accelerations ($\dot{u}\dot{v}\dot{w}$) during the hover maneuver. 65
5.35 Model prediction error in angular accelerations ($\dot{p}\dot{q}\dot{r}$) during the hover maneuver. 65

List of Tables

4.1 Detailed weight breakdown of the experimental platform . . . 27
4.2 Moments and products of inertia of Yak54 with Data Acquisition system. 28
4.3 Parameters used in calculation of rotational damping moments. 29

Acknowledgements

All thanks to God, the Exalted, the Most High. Without His mercy and blessings I could never have finished this thesis. Then thanks to my family, especially my husband Moustafa, who provides me with tremendous amounts of love and support. Thanks to my main supervisor Prof. Meyer Nahon who took me on as his student even though I am not a Mechanical Engineer. I learned a lot from working in his lab at McGill University, and the experience of studying in Montreal literally changed my life. Thanks to Kyrre Glette who agreed to co-supervise me from across the ocean. A great thanks is due to Patrick Thompson from the Udisco hobby-shop, for lending us his expertise in RC-planes. He piloted our plane in all the flight tests, some of which were conducted during rather unfriendly Montreal winter temperatures. And last but not least, thanks to all my friends and colleagues at McGill for an excellent working environment as well as countless cups of ice-cream and coffee.

Chapter 1

Introduction

1.1 Motivation

Many automation tasks requires a robot to follow a specified trajectory. Hand crafting such a trajectory is a non-trivial task for most robotic systems operating in a real world environment. The trajectory needs to adhere to the dynamics of the system, as well as correspond to the desired maneuver. The latter can be difficult for complex maneuvers, and the former requires accurate dynamics models that are valid for the entire trajectory. However, when humans learns a task we do not necessarily require someone to write down an accurate task description including any of the dynamics equations involved. Rather it is easier for us to learn the task if an expert demonstrates it to us. This is the principle of apprenticeship learning; Learning by demonstration.

Fixed wing unmanned aerial vehicles (UAVs) are interesting platforms for apprenticeship learning. Some UAVs are capable of highly aerobatic maneuvers such as perching and vertical hover. While the dynamics of fixed wing aircrafts within the conventional flight envelope are fairly well understood, the situation changes drastically at angles of attack closer to 90°. In such maneuvers the wings loose their lift, and the airflows around the plane are complex and unpredictable [45]. Because of the complexity of the dynamics, hand crafting trajectories for such maneuvers is difficult, if not impossible. Nevertheless experienced RC pilots are able to fly them. Thus it makes for an interesting apprenticeship learning problem to have a fixed wing UAV automate a trajectory demonstrated by a human RC pilot.

1.2 Thesis overview

This thesis focuses on a specific issue in apprenticeship learning. Given a set of suboptimal demonstration trajectories, what is the trajectory the human expert intended to demonstrate? If this trajectory is found it can be used as a target trajectory for an autonomous controller, possibly surpassing the human expert in performance of the demonstrated task. This is the first step towards an ultimate goal of automating highly aerobatic maneuvers with a fixed wing UAV.

Promising results have been presented by Abbeel et al. [2] using this approach to automate air-shows for an autonomous helicopter. The work of this thesis aims to apply techniques presented by Abbeel et al. to a highly aerobatic fixed wing UAV platform. While fully automating maneuvers is outside the scope of the thesis, the focus is set on inferring an intended target trajectory from a set of expert demonstrations. This trajectory can later be used as a target trajectory in an autonomous controller.

In order to collect training data for the trajectory learning, a fixed wing platform capable of recording the state of the plane as well as the pilot control inputs is needed. Such a platform could later be used for automating the maneuver as well. A large part of the work in this thesis has been devoted to developing a suitable platform.

The goals of the thesis can be summarized as:

- Apply trajectory learning techniques to highly aerobatic maneuvers with a fixed wing platform.

- Develop a suitable fixed wing platform for apprenticeship learning of highly aerobatic maneuvers.

1.2.1 Supporting contributions

The research was conducted in the lab of Professor Meyer Nahon, which is part of the Center for Intelligent Machines at McGill University. Other students in the research group has made supporting contributions to the work. Dimitri Poliderakis and Karen Bodie did preliminary research for development of the platform as well as some investigation of potentially suitable flight simulators as parts of their undergraduate honours thesis. Jean Froundjian developed a prototype of the ground based vision system described in section 4.4 as part of his Masters project. Ryan Caverly helped with practical work on experiments for characterizing the relationship between control inputs and actuator response during a summer internship. Waqas Khan is currently working on his PhD thesis developing a dynamics model of the fixed wing platform, that is accurate outside the conventional flight envelope. There is no overlap between this thesis and the work of Khan other than a shared platform.

1.2.2 Thesis outline

The remainder of this chapter will present related work. Chapter 2 gives an overview of background theory related to the trajectory learning algorithm. Chapter 3 presents the trajectory learning algorithm and the dynamic model of our fixed wing platform. Chapter 4 gives a detailed description of the platform. Chapter 5 presents the results of the trajectory learning and describes the preliminary testing and preprocessing, and chapter 6 provides a discussion of the results as well as the conclusion and suggestions for future work.

1.3 Related work

There has been much research in the field of apprenticeship learning, also known as imitation learning, learning by demonstration, or programming by demonstration.

Atkeson and Schaal used a form of apprenticeship learning to make a robotic arm learn tasks by watching a human demonstrate them [3]. Tidemann and Öztürk presented a modular connectionist architecture for apprenticeship learning [47], and used the architecture to enable a simulated robot to learn dance moves from demonstrations by a human dancer. Tedrake et al. implemented a different form of apprenticeship learning where a robot does not learn from a human, rather it learns dynamic walking from demonstrations by a passive dynamic walker [46]. Calinon et al. presents apprenticeship learning for learning trajectories of robotic tasks, and use time alignment of demonstrations, but they do not incorporate the dynamics of the system.

The work in this thesis is highly based on the work in autonomous helicopter aerobatics by Abbeel et al. [2]. Their helicopter autonomously performs full air-shows including maneuvers that only excellent human pilots are able to perform. This research is very similar to what we are trying to achieve on a fixed wing platform, as the helicopter learns trajectories from a set of demonstration trajectories from flights by a human RC pilot. These trajectories are then used as a target trajectory for the autonomous controller. The following section will describe the work of Abbeel et al. in more detail, as it has been central for this thesis.

1.3.1 Stanford Autonomous Helicopter

In the research of Abbeel et al. [2] aggressive aerobatic maneuvers are learned from demonstration flights by a human expert. The authors combine apprenticeship learning with a form of model predicative control for non-linear systems. The apprenticeship learning comprises inferring an intended target trajectory from a set of demonstrations.

A dynamic programming algorithm called Dynamic Time Warping (DTW), known from the field of speech recognition, is applied to align the demonstrations in time. The demonstrations are modelled as noisy subsampled observations of the hidden intended trajectory, in the form of a hidden Markov model (HMM). Parameters of the HMM, such as distributions of the hidden trajectory are computed using the Expectation Maximization algorithm. An extended kalman smoother is used for state estimation together with a basic dynamic model of the helicopter. The basic model is a parametric model where the parameters are learned from flight data that is not specific to the maneuver that is to be learned. The model is improved by incorporating model prediction errors as bias terms, and by applying locally weighted regression along the trajectory to more accurately represent the dynamics of the helicopter for the specific manuever.

After inferring the intended trajectory from the demonstrations the trajectory is used as a target trajectory for an autonomous controller. The controller consists of an off-line base controller and an on-line adaptive controller. First a limited horizon Linear Quadratic Regulator (LQR) is run offline to compute the optimal control policy for flights under ideal conditions. To account for external disturbances such as wind, an adaptive receding horizon LQR controls the helicopter online during the autonomous flight. If the online LQR fails to converge to a solution in time, the computed offline LQR policy is used.

Abbeel et al. presents promising results where the autonomous helicopter demonstrates entire air-shows consisting of aggressive aerobatic maneuvers such as rolls, flips, tic-tocs and auto-rotation landing. Their algorithm is capable of incorporating flight data from autonomous flights as training data for further training, resulting in a performance better than the human expert that initially demonstrated the maneuvers. Furthermore the algorithm can also incorporate expert advice as prior knowledge about the trajectory.

This thesis relies heavily on the work of Abbeel et al., applying the trajectory learning algorithm presented in [2] to a fixed wing platform rather than a helicopter. While Abbeel et al. incorporate expert advice for the target trajectory, the work presented here does not assume any prior knowledge of the trajectory other than the given demonstrations.

1.3.2 Aerobatic maneuvers

A considerable amount of research has been done on automation of aerobatic maneuvers with fixed wing UAVs. Much of this research is on controllers capable of controlling the plane in the nonlinear dynamics regime at high angles of attack. This section presents research done on automation of perching and hover maneuvers.

Perching Interesting work on perching includes that of Cory and Tedrake [10, 45, 11] who exploits nonlinear dynamics in development of a perching controller for their fixed wing glider that can land on a wire. Desbiens and Cutosky [25] demonstrates perching and landing on vertical surfaces such as a brick wall. While Cory and Tedrake uses a motion capture system to track the pitching trajectory, Desbiens and Cutosky found that for their purpose it was enough to measure the distance to the wall and by this trigger the perching maneuver at the exact right time. The maneuver itself is achieved open-loop.

Hurst et al. [18] offers a different solution to the problem of modelling the non-linear dynamics related to the perching maneuver. They divide the non-linear trajectory into windows of nearly linear behaviour and combined these models to approximate the full non-linear model. In this work the authors use a morphing plane that can change its airframe to maintain controllability during maneuvers that could otherwise result in a stall. The use of a morphing plane has advantages in controllability, it comes with the cost of higher complexity. In the work of this thesis a choice was made to use a regular off-the-shelf airframe because these planes are already capable of highly aerobatic maneuvers.

Hovering A fixed wing UAV can be made to hover like a helicopter by pitching up into a vertical position. While pitching up, the plane reaches a stall where the wings no longer contribute to the lift. The aircraft needs a large amount of thrust to break out of the stall, and UAVs capable of hovering typically have a large thrust to weight ratio. Green and Oh [15, 14, 16] developed a controller for autonomous hovering capable of transitioning from level flight to hover and maintaining a stable hover for several minutes. Myrand-Lapierre et al. [32] also describe strategies to transition between these two flight modes. Johnson et al. [21, 22] developed an adaptive controller using dynamic inversion adapted by neural networks for the same maneuver.

While the trajectory learning algorithm used in this thesis is tested using the vertical hover maneuver, the focus is on the trajectory learning itself rather than control, which sets apart this work from the previously mentioned research on autonomous hover.

Chapter 2

Background

Hidden Markov Models (HMMs), Expectation Maximization (EM), Kalman smoothing and Dynamic Time Warping (DTW) are central elements used in the trajectory learning algorithm. This chapter aims to give a brief introduction to the theory behind these techniques, while the next chapter will focus on how the techniques are applied in the specific case of trajectory learning for a fixed wing UAV.

2.1 Hidden Markov Models

The theory behind Markov processes were first presented by Andrei A. Markov for the use of modelling letter sequences in Russian literature [27], while Hidden Markov Models (HMMs) were introduced by Baum et al. around 1970 [5]. A hidden Markov model is a probabilistic function of a Markov process. The trajectory learning algorithm described in chapter 3 models the state of the plane along the trajectory using Hidden Markov Models (HMMs).

2.1.1 Discrete Markov Processes

Markov models can be used to model systems which at any time is in one of N states $s_1, s_2, ..., s_N$ [37]. The system transitions from one state to the other (or to the same state) at a regularly spaced time interval, and according to a given set of state transition probabilities. The time instances between the state transitions is denoted as $t = [1 : T]$, and the state at time t is denoted q_t. In the case of discrete Markov processes we assume the probability that the current state of the system is state $s_n, \quad n \in [1 : N]$ only depends on the previous state, not on all the earlier states. We also assume that this probability is independent of time. The state transition probabilities can be written as [37]:

$$a_{ij} = P(q_t = S_j | q_{t-1} = S_i), \quad 1 \leq i, j \leq N \tag{2.1}$$

The state transition probabilities obey standard stochastic constraints, and thus have the properties

$$a_{ij} \geq 0, \qquad \sum_{j=1}^{N} a_{ij} = 1 \tag{2.2}$$

Consider an example from [37], on modelling weather using a Markov model: Once a day the weather is observed as being in one of three states: sun, cloudy or rain/snow. The transition probabilities between the states are shown in figure 2.1. And can be written in matrix form as follows:

$$A = a_{ij} = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.2 & 0.6 & 0.2 \\ 0.3 & 0.3 & 0.4 \end{bmatrix} \tag{2.3}$$

Given this weather model, we can now calculate the probability that the weather next week is for instance "sun-sun-rain-rain-cloudy-sun-cloudy", if we know the weather is sunny on Monday. Stating the problem more formally we can say s_1 = sun, s_2 = cloudy and s_3 = rain/snow and the observation sequence $O = \{s_1, s_1, s_2, s_2, s_3, s_1, s_3\}$, with initial condition $\pi = s_1$. The probability of O given the model is

$$\begin{aligned} P(O|Model) = &\quad P(s_1, s_1, s_2, s_2, s_3, s_1, s_3|Model) \\ = &\quad P(s_1) * P(s_1|s_1) * P(s_2|s_1) * P(s_2|s_2) \\ &\quad * P(s_3|s_2) * P(s_1|s_3) * P(s_3|s_1) \\ = &\quad \pi * a_{11} * a_{12} * * a_{22} * a_{23} * a_{31} * a_{13} \\ = &\quad 1 * 0.8 * 0.1 * 0.6 * 0.2 * 0.3 * 0.1 \\ = &\quad 2.88 * 10^{-4} \end{aligned} \tag{2.4}$$

2.1.2 Hidden Markov Model

The Hidden Markov Model (HMM) is an extension of Discrete Markov Model where the states are not directly observable. Rather the result of the states can be observed, but not the process that generated this result.

[26] gives the example of a crazy soda machine that switches randomly between a tendency to dispense coke and a tendency to dispense ice tea. If the machine would dispense coke every time a coin is inserted in the coke tendency state, and ice tea in the ice tea tendency state, then the state would be directly observable and we could model it as a regular Markov model. But in the case of the crazy machine we can not observe the states directly. In either state the machine will dispense coke, ice tea or lemonade. The different kinds of soda are the observable results of the

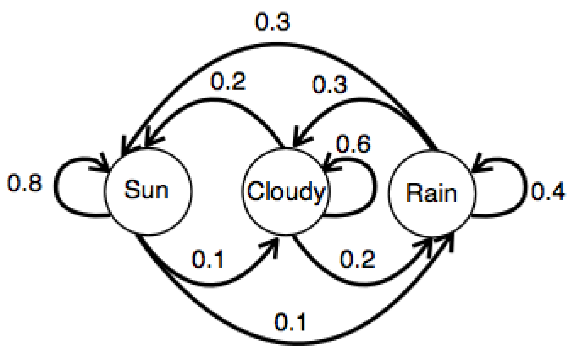


Figure 2.1: Three state Markov model of the weather

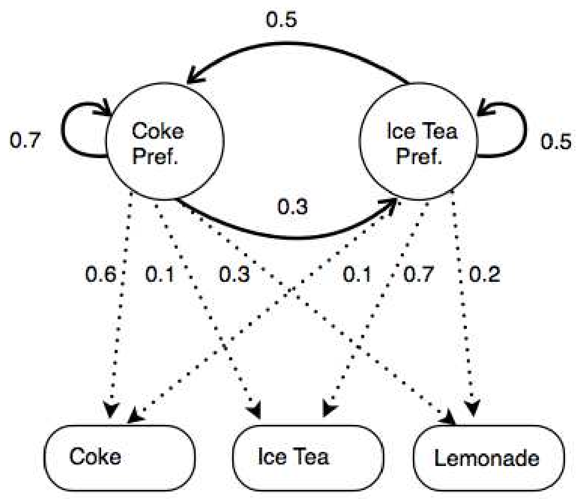


Figure 2.2: Hidden Markov Model of crazy soda dispenser

hidden states. Figure 2.2 shows the state transition probabilities and the output probabilities of the crazy machine.

The probability of a certain sequence of outputs is the sum of the probabilities for each possible state sequence that produces the output. As an example, given the model shown in figure 2.2 calculate the probability of observing the output sequence $O = \{Coke, Lemonade\}$ when the machine always starts in the coke preferring state.

$$P(O|Model) = 0.6 * 0.7 * 0.3 + 0.6 * 0.3 * 0.2 = 0.162 \tag{2.5}$$

The general form of HMM can be written as follows [26]:

Set of states	$S = \{s_1, s_2, ..., s_N\} = \{1, 2, ..., N\}$
Set of possible outputs	$K = \{k_1, k_2, ..., k_M\}$
Initial state probabilities	$\Pi = \{\pi_i\}, i \in S$
State transition probabilities	$A = \{a_{ij}\} = P(q_t = S_j \mid q_{t-1} = S_i), i, j, \in S$
Output emission probabilities	$B = \{b_{ik}\}, i \in S, k \in K$
State sequence	$X = (x_1, x_2, ..., x_{T+1})$
Output sequence	$O = (o_1, o_2, ..., o_T)$

In the above example the set of states consist of the coke preferring state (s_1), and the ice tea preferring state (s_2). $K = \{coke, Ice\ Tea, Lemonade\}$. $\Pi = \{1, 0\}$, because we know the machine always starts in state s_1.

$A = \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix}$ and $B = \begin{bmatrix} 0.6 & 0.1 & 0.3 \\ 0.1 & 0.2 & 0.7 \end{bmatrix}$.

There are three basic problems for HMMs that need to be solved for the HMM to be useful in real world applications [37]:

1. Given the observation sequence $O = (o_1, o_2, ..., o_T)$ and the model $\lambda = (A, B, \pi)$, how can we compute the probability $P(O|\lambda)$?

2. Given the observation sequence O and the model λ, how can we find the state sequence $Q = (q_1, q_2, ..., q_T)$ that best explains the observations?

3. How do we adjust the model parameters $\lambda = (A, B, \pi)$ to maximize $P(O|\lambda)$?

Problem 1 is the problem of evaluating how well the model fits the observed sequence. It is useful as a scoring measure when a model is to be selected from a set of models. This is the problem that was solved in the above example of a crazy soda machine.

Problem 2 is the problem of uncovering the hidden states of the model. Several different optimality criteria can be imposed depending on the application, since there is no "correct" state sequence to be found.

Problem 3 is the problem of "training" the HMM; i.e. optimizing the model parameters to best describe how a given observation sequence is generated. This is the problem of the HMM that models the states in the trajectory learning algorithm, and the problem can be solved using The Expectation-Maximization algorithm described next.

2.2 Expectation-Maximization

The Expectation-Maximization (EM) algorithm was first presented in 1977 [12]. It is used in a wide range of applications and has been described in several books and papers over the years, see for instance [29, 6, 49]. EM iteratively computes the Maximum Likelihood (ML) when the observations can be described as incomplete data. The trajectory learning algorithm uses a form of nested EM where both parameters for the HMM modelling the state is trained, and the time alignment indices that maps the states of the demonstrations to the intended trajectory are estimated. This section first provides a brief introduction to Maximum Likelihood Estimation (MLE), before describing the EM algorithm.

2.2.1 Maximum Likelihood Estimation

MLE is commonly used to estimate parameters of statistical models [33].

The probability density function (PDF) $p(x|\Theta)$ specifies the probability distribution of the data x given the parameters Θ. p could be a set of gaussian distributions and Θ could be the means and variances of the distributions.

If we have a data set $X = \{x_1, x_2, ..., x_N\}$ that is sampled from a distribution $p(X|\Theta)$, then the likelihood of the parameters given the data is [6]:

$$L(\Theta|X) = \prod_{i=1}^{N} p(x_i|\Theta) \tag{2.6}$$

In MLE we seek to find the parameters Θ' that maximizes the likelihood function $L(\Theta|X)$ i.e.:

$$\Theta' = \underset{\Theta}{\arg\max} L(\Theta|X) \tag{2.7}$$

Often the logarithm $log(L(\Theta|X))$ is maximized instead because it is analytically easier.

2.2.2 EM

Maximum likelihood assumes that all the data is observable. This is not always an applicable assumption, as in the case of HMMs where the observed data is a probabilistic function of a hidden state sequence. EM is an algorithm for estimating maximum likelihood of incomplete data sets. However the algorithm is also used in cases where the data set is in fact complete, by assuming there are hidden parameters. When working with incomplete data sets we need to adjust the likelihood function to account for the joint probability density of the observable and the hidden data.

Formally, the data set X is assumed to be incomplete, and a complete data set $Z = (X, Y)$ is assumed to exist. Furthermore a joint density function is specified [6]:

$$p(Z|\Theta) = p(X, Y|\Theta) = p(Y|X, \Theta)p(X|\Theta) \tag{2.8}$$

The complete data likelihood function is given by equation 2.9. Y can be considered as a random variable, while X and Θ are constant.

$$L(\Theta|Z) = L(\Theta|X, Y) = p(X, Y|\Theta) \tag{2.9}$$

The EM algorithm consists of two steps, an expectation step (E-step) where model parameters are estimated, and a maximization step (M-step) where $L(\Theta|Z)$ is maximized using the current estimate of model parameters. These two steps are iterated until convergence at a (possibly local) maximum.

The form of the estimation in the E-step depends on the application. In the case of this thesis the estimation is done using an extended kalman smoother. The maximization step estimates Θ' by maximum likelihood (see section 2.2.1) as though the estimated complete data Z provided by the kalman smoother is the real complete data. This maximization can be described as follows:

$$\Theta' = \arg\max_{\Theta} L(\Theta|Z) \tag{2.10}$$

2.3 Extended Kalman Smoother

The kalman filter was first presented by R.E. Kalman in 1960 [23]. It is a predictor-corrector type estimator that minimizes the estimated error covariance. The conditions for the filter to work optimally rarely exist, nevertheless it works well in many applications and is widely used in tracking, motion prediction and fusion of data from multiple sensors. [7] provides a gentle introduction to kalman filters, other references include [28, 42, 13]. Welch and Bishop maintains a website [17] with useful tutorials, references and research related to the Kalman filter.

An extended kalman smoother is used as a state estimator in the E-step of the EM used by the trajectory learning algorithm. It estimates the mode and distribution of the hidden states of the HMM, namely the intended trajectory.

2.3.1 Discrete Kalman filter

The discrete Kalman filter is the Kalman filter in its original form as presented in [23]. In this form both measurements and estimates are discrete in time.

Consider a linear system governed by the following equation:

$$x_k = Ax_{k-1} + Bu_k + w_k - 1 \tag{2.11}$$

And the measurements $z \in \mathbb{R}^m$ of the form

$$z_k = Hx_k + v_k \tag{2.12}$$

where w_k is the process noise, and v_k is the measurement noise. A in equation 2.11 is a n x n matrix that relates the state x at the previous time step to the state at the current time step. B is a n x l matrix that relates a control input $u \in \mathbb{R}$ to the state. H in equation 2.12 is a m x n matrix that relates the state to the measurement z.

The kalman filter can be thought of as a form of feedback control. It iterates two steps; A time update step and a measurement update step (feedback step). In the time update step a priori state estimate and error covariance is obtained as shown in equation 2.13 . In the measurement update step (equation 2.14) the Kalman gain is first calculated, then the state estimate is updated with the measurements z_k, and finally the error covariance is updated. The steps are iterated, recursively conditioning the estimates on all previous measurements. In the following equations $\hat{x}_k^-$ denotes the a priori state estimate at step k, while $\hat{x}_k$ is the posteriori state estimate. Similarly P_k^- represents the a priori estimate error covariance, and P_k is the a posteriori estimate error covariance [7].

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_k \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned} \tag{2.13}$$

$$\begin{aligned} K_k &= \frac{P_k^- H^T}{HP_k^- H^T + R} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ P_k &= (I - K_k H)P_k^- \end{aligned} \tag{2.14}$$

2.3.2 Extended Kalman filter

The discrete kalman filter described in the previous section is only applicable to linear systems. The extended Kalman filter (EKF) extends the theory to non-linear systems by linearizing about the current mean and covariance. The estimation is linearized using partial derivatives of the process and measurement functions. The time update and measurement update equations are modified as shown in equation 2.15 and 2.16. A_k and W_k in equation 2.15 are the process Jacobians at time step k, and Q_k is the process noise covariance at time step k. f and h in the measurement update equations (2.16) are both non linear functions. f relates the state at the previous time-step $k-1$ to the state at the current time-step k, while h relates the state x_k to the measurement z_k. H_k and V_k are measurement Jacobians at time step k, and R_k is the measurement noise covariance equation [7].

$$\begin{aligned} \hat{x}_k^- &= f(\hat{x}_{k-1}, u_k) \\ P_k^- &= A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \end{aligned} \tag{2.15}$$

$$\begin{aligned} K_k &= \frac{P_k^- H_k^T}{H_k P_k^- H_k^T + V_k R_k V_k^T} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-)) \\ P_k &= (I - K_k H_k)P_k^- \end{aligned} \tag{2.16}$$

2.3.3 Kalman smoother

The Kalman smoother differs from the Kalman filter in that it conditions the smoothed distributions with respect to all the measurement data, whereas the Kalman filter only uses the measurements up until the current time-step. The Kalman smoother equations (equation 2.17) are recursively applied to the results of the Kalman filter [41]. The subscript $k|K$ denotes a value conditioned on all the measurements $z_{1:K}$, while $k|k$ denotes values only conditioned on measurements until and including the current time step k. The smoothed distributions are calculated recursively starting from the last time step $k = K$.

$$\begin{aligned}
\hat{x}_{k|K} &= \hat{x}_{k|k} + L_k(x_{k+1|K} - \hat{x}_{k+1|K}) \\
P_{k|K} &= P_{k|k} + L_k(P_{k+1|K} - P_{k+1|k})L_k^T \\
L_k &= \frac{P_{k|k}A^T}{P_{k+1|k}}
\end{aligned} \tag{2.17}$$

2.4 Dynamic Time Warping

Dynamic Time Warping (DTW) is used to compare time dependent sequences. It has been used extensively in speech recognition, where words need to be recognized regardless of the speed of the speech, and the speed of the speech of one person may vary non-linearly over time. However the technique is general and can be applied to any pair of sequences that require alignment in time.

The training data used to train the trajectory learning algorithm consist of a set of demonstration flight trajectories. In order to compare the demonstrations they need to be aligned in time similar to how speech patterns are aligned in speech recognition. DTW is applied to find the time mapping between the demonstration trajectories and the hidden intended trajectory as described in section 3.1.1.

Consider the two time dependent sequences $A = [a_1, a_2, ..., a_I]$ and $B = [b_1, b_2, ..., b_J]$, where both sequences are sampled at the same constant sample rate. The goal is to align A and B in time by mapping the features a_i to the most similar features b_j where $i \in [1 : I]$ and $j \in [1 : J]$ [40]. Figure 2.3 illustrates the principle.

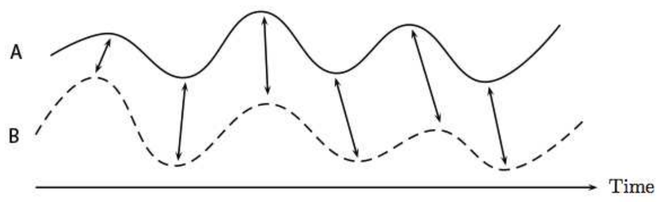


Figure 2.3: Dynamic time warping. The figure is adapted from [31].

In order to compare the features we need a distance function $d(a, b)$ that measures the similarity between two features. The distance measure function varies with the application. $d(a, b)$ should give a low value if a and b are similar to each other, and a high value if they differ much.

By evaluating the distance of every pair a_i, b_j we obtain the distance matrix $D \in \mathbb{R}^{IxJ}$, where $D(i, j) = d(a_i, b_j)$. The matrix D can be viewed as in figure 2.4, where sequence A is developed along the i-axis and sequence B along the j-axis. We want to find the alignment that maps features of sequence A to the most similar features of sequence B; i.e. an alignment that minimizes the distance measure between the two sequences. This is the sequence $C = [c_1, c_2, ..., c_K]$, which in [40] is called the warp path.

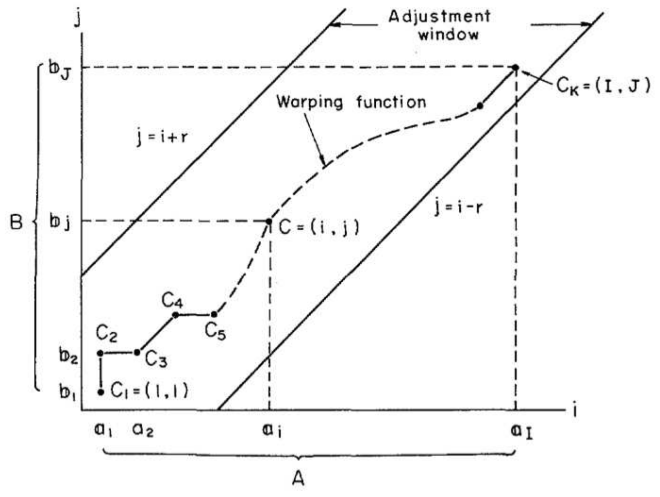


Figure 2.4: Dynamic time warping distance matrix. The figure is borrowed from [40].

The more the warp path deviates from the diagonal of matrix D, the more changes have been done in the time alignment. If the warp path equals the diagonal of D then there are no changes. Certain conditions are placed on the warp path C. These conditions include monotonicity, continuity, boarder conditions and windowing. Equation 2.18 defines the monotonicity condition.

$$\begin{gathered} i_{k-1} \le i_k \\ j_{k-1} \le j_k \end{gathered} \tag{2.18}$$

The continuity condition is written as follows:

$$\begin{gathered} i_k - i_{k-1} \le 1 \\ j_k - j_{k-1} \le 1 \end{gathered} \tag{2.19}$$

Equation 2.18 and 2.19 gives us the following relation between two consecutive points:

$$c_{k-1} = \begin{cases} & (i_k, j_k - 1) \\ & (i_k - 1, j_k - 1) \\ or & (i_k - 1, j_k) \end{cases} \tag{2.20}$$

The boundary conditions are given by

$$\begin{gathered} c_1 = (1, 1) \\ c_K = (I, J) \end{gathered} \tag{2.21}$$

An adjustment window of length r is chosen to limit the time warping as shown in figure 2.4. The window condition is formally written:

$$|i_k - j_k| \leq r \tag{2.22}$$

[40] defines the time normalized difference between A and B as follows:

$$D(A,B) = \frac{1}{\sum_{k=1}^{K} w_k} \quad \min_F \left[\sum_{k=1}^{K} d(c_k) * w_k \right] \tag{2.23}$$

This equation can be efficiently solved by dynamic programming by recursing backwards from the last time step until the beginning of the time series.

w_k in equation 2.23 is a weighting coefficient. The form of w_k depends on the manner which A and B are aligned to each-other. Equation 2.24 gives the weighting coefficient for the case where B is aligned to the time axis of A, while equation 2.25 is valid if A and B are both aligned to an imaginary axis l.

$$w_k = i_k - i_{k-1} \tag{2.24}$$

$$w_k = (i_k - i_{k-1}) + (j_k - j_{k-1}) \tag{2.25}$$

Chapter 3

Trajectory Learning

The goal of the trajectory learning algorithm is to infer the trajectory that the pilot most likely intended to fly, by examining trajectories from real demonstration flights repeatedly demonstrating the same maneuver. The demonstration trajectories are modelled as sub-optimal noisy observations of the hidden intended trajectory. This follows from the idea that even an expert pilot will never be able to fly a maneuver perfectly, but the different demonstrations may be sub-optimal at different sections of the trajectory. In that case the set of suboptimal demonstrations together can be used to infer the intended trajectory. The trajectory learning in this thesis is highly based on the work of Abbeel et al. [2]. The approach is the same, but a different dynamics model is needed since Abbeel et al. applies the techniques to a helicopter platform while this thesis aims to apply the same methods to flight data from a fixed wing UAV.

3.1 State modelling

M demonstration trajectories of length N^k, for $k = 0, 1, \dots, M-1$ are given as input to the algorithm. Abbeel et al. [2] reports $M = 5$ to be sufficient for most maneuvers, provided the demonstration trajectories are of good quality. A trajectory is represented as a sequence of states s_j^k and control inputs u_j^k. The states and control inputs are combined into a single observation vector $y_j^k = \begin{bmatrix} s_j^k \\ u_j^k \end{bmatrix}$, for $j = 0, 1, \dots, N^k - 1$ and $k = 0, 1, \dots, M-1$. The observations are assumed to be governed by a hidden intended trajectory $z_t = \begin{bmatrix} s_t \\ u_t \end{bmatrix}$ for $t = 0, 1, \dots, T-1$, where T is the length of the hidden trajectory.

An initial state distribution $z_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ for the hidden trajectory is given, and the state at next time step is estimated as

$$z_{t+1} = f(z_t) + \omega_t^{(z)}, \quad \omega_t^{(z)} \sim \mathcal{N}(0, \Sigma^{(z)}) \tag{3.1}$$

where $f(z_t)$ is a basic dynamics model of the plane as described in section 3.3, and $\omega^{(z)}$ is a zero mean gaussian random variable representing uncertainties of the dynamics model.

To allow for variations in time between the hidden intended trajectory and the demonstrations, the demonstration trajectories are modelled as subsampled noisy observations of the hidden trajectory. A hidden trajectory length of twice the average length of the demonstrations gives sufficient resolution [2].

3.1.1 Time alignment

No human pilot is capable of flying the exact same trajectory with the exact same timing in repeated demonstrations, even if the conditions were perfect with no external disturbance. This means the trajectories are not only suboptimal with respect to the features of the intended trajectory, but they are also possibly (and most likely) warped along the time axis. The maneuver might be started at different times in the different demonstrations, and the duration of features in the maneuver might differ between demonstrations. In other words there exists a non-linear mapping between each observed demonstration trajectory and the hidden intended trajectory, and in order to compare the demonstrations they first need to be aligned in time. This is done using the dynamic programming algorithm known as Dynamic Time Warping (DTW) that was described in section 2.4. DTW computes time alignment indices τ_j^k that maps the observations y_j^k to the states z_t of the hidden trajectory. In other words, the model assumes

$$y_j^k = z_{\tau_j^k} + \omega_j^{(y)}, \quad \omega_j^{(y)} \sim \mathcal{N}(0, \Sigma^{(y)}) \tag{3.2}$$

The noise term $\omega_j^{(y)}$ models inaccurate sensor readings as well as errors caused by the human pilot's imperfect demonstrations.

τ_j^k is not directly observable, and is modelled with the following distribution using parameter d_i^k:

$$P(\tau_{j+1}^k | \tau_j^k) = \begin{cases} d_1^k & if \quad \tau_{j+1}^k - \tau_j^k = 1 \\ d_2^k & if \quad \tau_{j+1}^k - \tau_j^k = 2 \\ d_3^k & if \quad \tau_{j+1}^k - \tau_j^k = 3 \\ 0 & \quad otherwise \end{cases} \tag{3.3}$$

Figure 3.1 illustrates how the relationship between the observations and the hidden trajectory are modelled when τ_j^k is unobserved. As can be seen from the illustration, the model is quite complicated because the time alignment indices τ_j^k are unknown. In this case each node of the observed demonstration trajectory y_j^k can be associated with several nodes of the hidden intended trajectory z_t. If however τ_j^k was known, the state model would be significantly less complicated. This case is illustrated by an example in figure 3.2. Note that the superscript k has been omitted for simplicity, and the figure illustrates the relationship between one demonstration y_j^k and the hidden trajectory z_t. There are M demonstration trajectories and M time alignment index sequences that all relate to the same hidden trajectory in a similar manner to the illustrations.

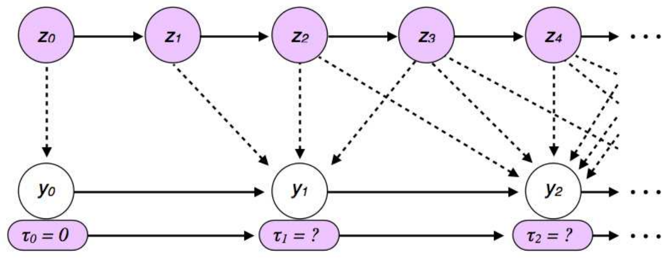


Figure 3.1: Model of the relation between the hidden trajectory and one of the demonstrations. The coloured nodes are unobserved.

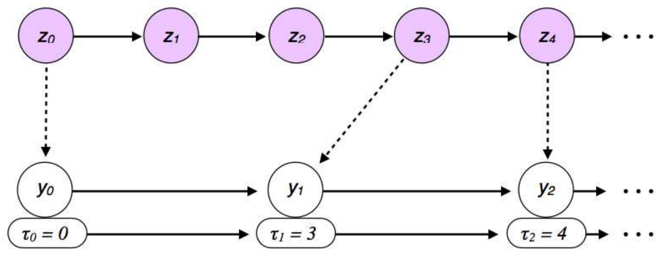


Figure 3.2: Model of the state when τ_j^k is fixed. Coloured nodes are unobserved.

3.2 The trajectory learning algorithm

The trajectory learning algorithm computes the most likely intended trajectory from a set of demonstration trajectories that are possibly warped along the time axis. More formally the algorithm approximately solves the maximization [2]:

$$\max_{\tau,\Sigma^{(\cdot)},d} logP(y,\tau;\Sigma^{(\cdot)},d) \tag{3.4}$$

Where y is the observations (i.e. demonstration trajectory), τ is the time-alignment indices, d is the time-index transition probabilities, and $\Sigma^{(\cdot)}$ is the covariance matrices. The intended trajectory z is estimated to be the mode of the distribution found in the maximization of equation 3.4

If the time alignments τ_j^k are fixed, the parameters d_i^k can be estimated in closed form. Furthermore estimating the covariance matrices $\Sigma^{(\cdot)}$ becomes a standard HMM parameter learning problem, where the parameters can be learned using the EM algorithm. In order to leverage the simplifications made possible by a fixed τ_j^k, Abbeel et al. proposes an alternating optimization procedure [2]. Covariances and the distributions of the hidden states are first estimated, assuming fixed time indices τ_j^k. In the second phase τ_j^k and transition probability parameters d_i^k are found on the basis of the current estimates of z. The resulting time indices are then used to re-estimate the covariances and distributions of z, and the process is repeated until convergence. The technique can be viewed as a nested EM summarized in the following 5 steps, where step 2-5 are iterated:

1. initialize $\Sigma^{(\cdot)}, d_i^k$, and τ_j^k to hand chosen values. Abbeel et al. reports

$\Sigma^{(\cdot)} = I, d_i^k = \frac{1}{3}$, and $\tau_j^k = j\frac{T-1}{N^k-1}$ as typical choices [2].

2. Hidden trajectory E-step: Find the distributions $\mathcal{N}(\mu_{t|T-1}, \Sigma_{t|T-1})$ of the hidden states z_t based on the current estimates of $\Sigma^{(\cdot)}$ and τ_j^k, by running an extended kalman smoother.

3. Hidden trajectory M-step: Update the covariances $\Sigma^{(\cdot)}$ by the standard EM update.

4. Time alignment E-step: fix z to $\mu_{t|T-1}$ obtained by the kalman smoother, and use dynamic time warping to find the τ that maximizes $P(z, y, \tau)$.

5. Time alignment M-step: estimate d from τ using standard maximum likelihood estimates.

The following subsections will treat the different steps of the algorithm in detail.

3.2.1 E-step for hidden trajectory

The E-step for inferring the hidden trajectory uses an extended Kalman smoother as described in section 2.3 to estimate the states z_t. Recall the a priori estimate $\hat{x}_t^-$ in the update equations 2.15 of the extended Kalman filter. $\hat{x}_t^-$ is a function that relates the state at the previous time step and the control inputs to the state at the current time-step, $f(\hat{x}_{t-1}, u_t)$. This is the dynamics model of the plane, which is described in section 3.3.

First an extended Kalman filter computes $\Sigma_{t|t}$ and $\mu_{t|t}$, which is the covariance and mode of the distribution of z_t based on the observations at time-steps until and including the current time-step. In the process $\Sigma_{t+1|t}$ and $\mu_{t+1|t}$, namely the parameters of the distribution of z_{t+1} given only the observations up until the current time-step t, are calculated as well. Then the results of the Kalman filtering is used in the Kalman smoother to calculate $\Sigma_{t|T-1}$ and $\mu_{t|T-1}$; i.e. the mode and covariance given the observations at all time-steps. The algorithm has now estimated the distribution $z_t \sim \mathcal{N}(\mu_{t|T-1}, \Sigma_{t|T-1})$, and z_t is estimated to be equal to the mode $\mu_{t|T-1}$ for the rest of the EM iteration.

3.2.2 M-step for hidden trajectory

z_{t+1} and y_{t+1} are modelled as follows:

$$z_{t+1} = f(z_t) + \omega_t^{(z)}, \quad \omega_t^{(z)} \sim \mathcal{N}(0, Q) \tag{3.5}$$

$$y_{t+1} = h(z_t) + \nu_t^{(z)}, \quad \nu_t^{(z)} \sim \mathcal{N}(0, R) \tag{3.6}$$

After the Kalman smoother computes the distribution $z_t \sim \mathcal{N}(\mu_{t|T-1}, \Sigma_{t|T-1})$ in the E-step, Q and R of equations 3.5 and 3.6 can be updated using the standard update of EM applied to nonlinear systems with Gaussian noise [2]:

$$\begin{aligned} \delta\mu_t &= \mu_{t+1|T-1} - f(\mu_{t|T-1}) \\ A_t &= Df(\mu_{t|T-1}) \\ L_t &= \frac{\Sigma_{t|t} A_t^T}{\Sigma_{t+1|t}} \\ P_t &= \Sigma_{t+1|T-1} - \Sigma_{t+1|T-1} L_t^T A_t^T - A_t L_t \Sigma_{t+1|T-1} \\ Q &= \frac{1}{T} \sum_{t=0}^{T-1} \delta\mu_t \delta\mu_t^T + A_t \Sigma_{t|T-1} A_t^T + P_t \end{aligned} \tag{3.7}$$

$$\begin{aligned} \delta y_t &= y_t - h(\mu_{t|T-1}) \\ C_t &= Dh(\mu_{t|T-1}) \\ R &= \frac{1}{T} \sum_{t=0}^{T-1} \delta\mu_t \delta\mu_t^T + C_t \Sigma_{t|T-1} C_t^T \end{aligned} \tag{3.8}$$

3.2.3 Time alignment

In the E-step for time alignment the demonstrations are aligned in time by optimizing the time alignment index τ for each demonstration separately using Dynamic Time Warping (described in section 2.4. The goal is to compute the τ that maximizes the log likelihood of the current set of parameters [2]:

$$\tau = \arg\max_{\tau} logP(z, y, \tau; \Sigma^{(\cdot)}, d) \tag{3.9}$$

As discussed earlier, d is assumed to be fixed, and optimized separately from τ. This assumption simplifies the maximization to:

$$\begin{aligned} \tau &= \arg\max_{\tau} logP(y|z,\tau)P(z)P(\tau) \\ &= \arg\max_{\tau} logP(y|z,\tau)P(\tau) \end{aligned} \tag{3.10}$$

which in turn can be expanded to:

$$\tau = \arg\max_{\tau} \sum_{k=0}^{M-1} \sum_{j=0}^{N^k-1} [\ell(y_j^k | z_{\tau_j^k} + \ell(\tau_j^k | \tau_{j-1}^k)] \tag{3.11}$$

z in the above equations was found in the E-step for hidden trajectory (section 3.2.1) and is the mode of the distribution computed by the kalman smoother. The summations in equation 3.11 are independent of each other. In other words the log likelihood of τ can be computed separately for each of the M demonstrations.

A quantity $Q(s,t)$ is defined as the maximum obtainable value of the first $s+1$ terms of the inner summation in equation 3.11, where $\tau_s = t$ [2]:

$$Q(0,t) = \ell(y_0 | z_{\tau_0}, \tau_0 = t) + \ell(\tau_0 = t), \quad for\ s = 0 \tag{3.12}$$

$$
\begin{aligned}
Q(s,t) = {} & \ell(y_0|z_{\tau_0}, \tau_0 = t) \\
& + \max_{\tau_1,\dots,\tau_{s-1}} [\ell(\tau_s = t|\tau_{s-1}) \\
& + \sum_{j=0}^{s-1} [\ell(y_j|z_{\tau_j}, \tau_j) + \ell(\tau_j|\tau_{j-1})]], \quad for\ s > 0
\end{aligned}
\tag{3.13}
$$

Equation 3.13 can be written recursively as follows:

$$
\begin{aligned}
Q(s,t) = {} & \ell(y_0|z_{\tau_0}, \tau_0 = t) \\
& + \max_{t'} [\ell(\tau_s = t|\tau_{s-1} = t') + Q(s-1, t')]
\end{aligned}
\tag{3.14}
$$

where $t' \in \{t-3, t-2, t-1\}$. The alignment window condition is enforced by only computing $Q(s,t)$ if $2s - C \le t \le 2s + C$, where C is fixed.

In the M-step for time alignment the time index transition probability parameters d are estimated from the new τ using maximum likelihood estimation.

3.3 Dynamic Model

The dynamic model is the function $f(z_t)$ in equation 3.1, and is used for state estimation by the EM algorithm as described in section 3.2.1 and 3.2.2. Only an approximate dynamic model is needed [2]. This model can be learned from data as in the work of Abbeel et al. [2], or it can be modelled in a more traditional way as is done in this thesis. This section provides details on the dynamics model used with the learning algorithm for the fixed wing UAV described in chapter 4. The dynamics model predicts angular accelerations $(\dot{p}, \dot{q}, \dot{r})$ and linear accelerations $(\dot{u}, \dot{v}, \dot{w})$ based on the current state and control input. Angular and linear velocities, position, and orientation are obtained through simple Euler integration of the accelerations.

3.3.1 Equations of motion

The dynamic model assumes the equations of motion given in equation 3.15. These equations are based on the model used by Abbeel et al. [2], but have been modified to account for the differences between a helicopter and a fixed wing platform.

$$
\begin{aligned}
\dot{u} &= vr - wq - d_x/m + g_x/m + F/m \\
\dot{v} &= wp - ur - d_y/m + g_y/m \\
\dot{w} &= uq - vp - d_z/m + g_z/m - l \\
\\
\dot{p} &= qr(I_{YY} - I_{ZZ})/I_{XX} - B_X|p|p/I_{XX} + M_a/I_{XX} \\
\dot{q} &= pr(I_{ZZ} - I_{XX})/I_{YY} - B_Y|q|q/I_{YY} + M_e/I_{YY} \\
\dot{r} &= pq(I_{XX} - I_{YY})/I_{ZZ} - B_Z|r|r/I_{ZZ} + M_r/I_{ZZ}
\end{aligned}
\tag{3.15}
$$

where p, q, r are the angular velocities around the x,y and z-axis respectively. u, v, w are the linear velocities in body frame of the aircraft. d_x, d_y and d_z are the drag forces. l is the lift and F is the thrust. $-B_X|p|p, -B_Y|q|q$

and $-B_Z|r|r$ are the rotational damping moments in roll, pitch and yaw. Note that rather than squaring the angular velocities they are multiplied with their absolute value, in order to keep the signs intact. M_a, M_e and M_r are the rotational moments governed by aileron, elevator and rudder deflections. m is the mass of the plane, and I_{xx}, I_{yy}, I_{zz} are the inertial moments of the x, y and z-axis of the plane. The products of inertia I_{xy}, I_{xz} and I_{yz} are assumed to be 0. The moments are divided by m for the linear accelerations and I_{xx}, I_{yy} or I_{zz} for the angular accelerations in order to convert the moments into forces. $qr(I_{YY} - I_{ZZ}), pr(I_{ZZ} - I_{XX})$ and $pq(I_{XX} - I_{YY})$ are the inertial coupling terms. g_x, g_y and g_z are the gravity force in body frame of the plane. The transformation of gravity from inertial to body frame is given by the following equation, where θ represents the pitch angle of the plane and ψ denotes the yaw angle.

$$\begin{aligned} g &= 9.81 \\ g_x &= -mg\sin\theta \\ g_y &= mg\cos\theta\sin\psi \\ g_z &= mg\cos\theta\cos\psi \end{aligned} \tag{3.16}$$

The specific values of the parameters with respect to the fixed wing platform used in this thesis are given in section 4.1.1 on page 27.

3.4 Improvements of the model

The Kalman smoother calculates the model prediction error (MPE) of the dynamics model at each time-step. The dynamic model can be improved for a specific trajectory by incorporating the MPE as bias terms. Denoting the bias terms as $\beta = \{\beta_u, \beta_v, \beta_w, \beta_p, \beta_q, \beta_r\}$, equation 3.1 becomes:

$$z_{t+1} = f(z_t) + \beta_t + \omega_t^{(z)}, \quad \omega_t^{(z)} \sim \mathcal{N}(0, \Sigma^{(z)}) \tag{3.17}$$

Chapter 4

Experimental platform

The goal of the trajectory learning algorithm is to learn a trajectory for a specific maneuver from demonstration flights controlled by a human pilot. To achieve this we need an experimental platform that is familiar to the pilot, that is capable of highly aerobatic maneuvers as well as capable of recording the state of the plane during the demonstration flights. Figure 4.1 shows the pilot flying the plane inside a gymnasium.

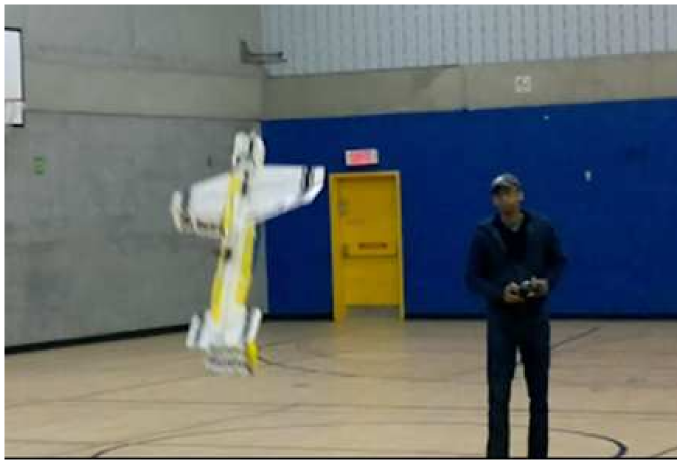


Figure 4.1: Expert pilot flying the plane in vertical hover.

Much of the research done on automating aerobatic maneuvers makes use of custom build airframes [10, 11, 44, 18, 45] sometimes with added degrees of freedom, as in the work of Hurst et al. in [18]. We chose an off-the-shelf fixed wing RC plane because these planes are already capable of highly aerobatic maneuvers. They are naturally intuitive to an experienced RC pilot, and development time is saved as the plane is designed by the manufacturer.

We represent the state of the plane by position, orientation, linear acceleration and angular velocity. The plane was fitted with a custom data acquisition (DAQ) system in order to collect and record this data. Included in the DAQ system is a sensor suite consisting of an Inertial Measurement Unit (IMU), a Global Positioning System (GPS) module, a barometric pressure sensor, and a sonar.

The data was collected indoors to avoid the effects of wind. One challenge when conducting indoor testing is that most indoor test spaces lack GPS visibility. We eventually solved this problem by conducting our test flights in the Stinger Dome at Concordia University. This is a temporary inflatable dome where GPS visibility is comparable to outside. We also looked into the possibility of using an alternative positioning method, and started development of a ground based vision system as described in section 4.4.

The trajectory learning algorithm finds a trajectory that can be used as a target for a controller in an autonomous scenario. An accurate simulator is valuable in development of such a controller. It is useful to develop the controller in simulation before testing it on a real flight, as repeated test-runs under predictable conditions can be performed without the time and monetary costs of rebuilding the plane after every crash. Section 4.5 provides an overview of flight simulators that could be used for this purpose.

4.1 Plane

The plane used in our experiments is the Electrifly by Great Planes Yak54 3D RC plane shown in figure 4.2. It is made of styrofoam with carbon fiber control surface push-rods and landing gear. The weight of the airframe without servos, motor or any electronics is 130 grams. Its length is 0.95 meters and it has a wingspan of 0.94 meters. The small size and low weight makes it ideal for indoor flight. This is an off-the-shelf airframe designed for RC aerobatics. It is easy to build and relatively cheap which is of benefit as crashes are inevitable, even for an experienced RC pilot. Table 4.1 gives a detailed weight breakdown of all the components on the plane.

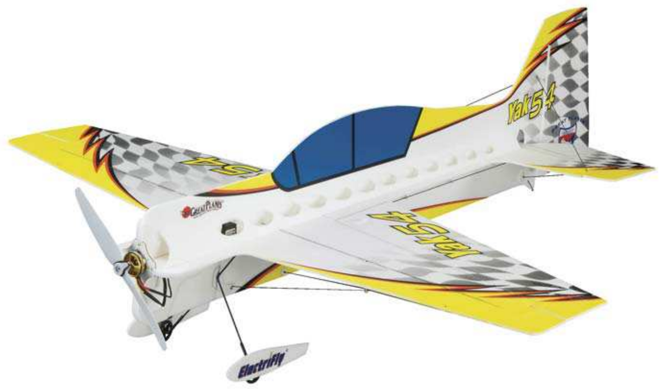


Figure 4.2: Electrifly Yak 54 3D

A RC plane is normally operated by transmitting control inputs as radio signals from a transmitter. These signals are received by the radio receiver aboard the plane. The servos and the motor controller are connected directly to the receiver.

Component	Mass
Airframe	130g
Motor	54g
Electronic Speed Controller	31g
Propeller	13g
Servos	3 x 11g = 33g
RC receiver	9g
ArduPilot Mega	16g
Inertial Measurement Unit	12g
GPS	7g
SD-module	13g
Sonar	5g
Pressure Sensor	1g
Battery	73g
Cables, extra glue and reinforcements	21g
Total	418g

Table 4.1: Detailed weight breakdown of the experimental platform

Our plane uses three servos which actuate the deflection of the control surfaces by pushing or pulling thin carbon fiber rods. One servo actuates both ailerons by the use of a differential arm. The other two servos operate the rudder and the elevator. A motor controller controls the speed of the motor. The motor controller and the servos take their inputs in the form of pulse width modulated (pwm) signals.

4.1.1 Dynamical properties of the plane

The dynamical properties of the plane depends on the shape, size and weight of the plane. This section describes the properties specific to the Yak54 with fitted electronics and gives specific values for the parameters of the dynamic model presented in section 3.3.

The total mass of the plane is 0.418kg as shown in table 4.1.

The moments of inertia is the resistance of the plane to angular accelerations, similar to how mass can be seen as an object's resistance to linear accelerations. They were calculated using CAD-software and are given in table 4.2. The products of inerta I_{XY}, I_{XZ} and I_{YZ}, are small and thus assumed to be zero in the dynamic model of the plane.

Rotational moments. The aileron, elevator and rudder deflections cause roll, pitch and yaw moments respectively. The following sign convention was used when calculating the moments[39]:

- A positive aileron deflection angle is given when the right aileron is pointing down, and leads to a negative roll moment M_a that makes the right wing go up.
- Positive elevator deflection means the elevator points down, and leads to a negative pitch moment M_e that pitches the nose of the plane down.
- Positive rudder deflection means rudder moved to the left. It leads to a negative yaw moment M_r which moves the nose left.

M_a, M_e and M_r were calculated as:

$$\begin{aligned} M_a &= (-0.03554\delta_a^2 + 0.04795\delta_a)u^2 \\ M_e &= (-0.02698\delta_e^2 + 0.03641\delta_e)u^2 \\ M_r &= 0.05490\delta_r u^2 \end{aligned} \tag{4.1}$$

Where δ_a, δ_e, and δ_r are the deflections of aileron, elevator and rudder in radians, and u is the linear velocity along the x-axis of the plane.

I_{XX}	$0.00222 kgm^2$
I_{YY}	$0.01676 kgm^2$
I_{ZZ}	$0.01815 kgm^2$
I_{XY}	$-0.00012 kgm^2$
I_{XZ}	$-0.00015 kgm^2$
I_{YZ}	$0.00001 kgm^2$

Table 4.2: Moments and products of inertia of Yak54 with Data Acquisition system.

Rotational damping moments. The damping moments are calculated assuming the surfaces of the plane are plain flat plates. In the case of the Yak54 foam plane this is an accurate assumption. Equation 4.2 shows the calculation of the rotational damping moments, and table 4.3 explains the parameters used in the calculation. B_x dampens the roll moment and is assumed to be affected by the wings, tail and the body of the fuselage. B_y dampens the pitch moment, and is mostly affected by the wing and the tail. B_z dampens the yaw moment and is assumed to only be affected by the body of the fuselage.

$$\begin{aligned} B_x &= \rho C_D l_2 * 0.0064 + \rho C_D l_3 * 0.0004 + \rho C_D R_1 * 0.000016402 \\ B_y &= \rho C_D R_2 * 0.0001233 + \rho C_D R_3 * 0.020042 \\ B_z &= \tfrac{1}{2}\rho C_D l_1 * 0.0528678 \end{aligned} \tag{4.2}$$

Parameter	Value	Physical explanation
ρ	1.2041	Density of air at 20 degrees
C_D	1.28	Drag coefficient of a flat plate
l_1	0.18	Height of fuselage in meters
l_2	0.21	Chord at the midpoint of each wing
l_3	0.14	Chord at the midpoint of each tail
R_1	0.9	Length of the body
R_2	0.4	Half the length of the wing
R_3	0.2	Half the length of the tail

Table 4.3: Parameters used in calculation of rotational damping moments.

The drag forces along the x, y and z-axis of the plane are

$$\begin{aligned} d_x &= 0.03852u^2 \\ d_y &= 0.114v^2 \\ d_z &= 0.199w^2 \end{aligned} \tag{4.3}$$

where u, v and w are the linear velocities in the body frame of the plane.

4.1.2 Modifications to the airframe

Some modifications had to be done for the plane to be able to fly aerobatic maneuvers with the added weight of the electronics.

These include reinforcement of the motor mount, reinforcement of the airframe structure, as well as reinforcement and lengthening of the landing gear.

The motor mount was reinforced by steel epoxy to be able to hold the strong motor, as shown in figure 4.3. The landing gear was extended to make room for a 11-inch propeller, and reinforced by carbon fiber rods. The fuselage and the wings of the plane also needed reinforcement in the area where the landing gear is attached. The added mass increases the pressure at this point during landing.

4.1.3 Motor

When collecting training data we were primarily interested in high Angle-of-Attack (AoA) maneuvers such as vertical hover. This maneuver requires a thrust-to-weight ratio greater than one. Since all the lift will be generated by the propeller in the vertical flight regime [15]. With the added electronics the total weight of the plane is 418g. We chose the Rimfire 400 brushless motor, which according to our tests produces maximum 10N (with control input $pw = 2100\mu s$). This gives us a maximum thrust-to-weight ratio of $\frac{10N}{0.418kg*9.81m/s^2} = 2.44 : 1$. Our plane requires a thrust of more than $0.418kg * 9.81m/s^2 = 4N$ to successfully perform the hover maneuver. The thrust produced by the motor at different speeds (i.e. at different control input pulse widths) was measured in the lab, and the resulting plot is shown in figure 4.4.

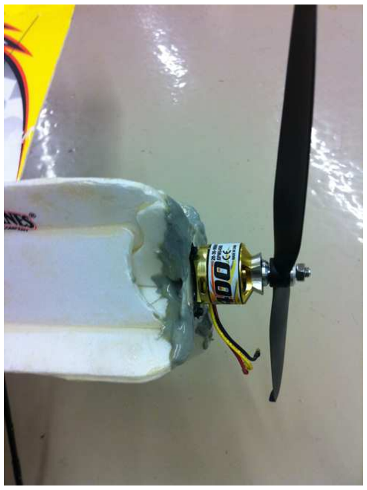


Figure 4.3: Reinforcement of the motor mount

To give the pilot some operating room it is beneficial if the required thrust can be achieved with the control-stick actuated half-way or less ("mid-stick"), this is equivalent to a control input pulse width of $1500\mu s$ or less. From figure 4.4 we can see that the RimFire 400 produces 6.54N thrust mid-stick ($pw = 1500\mu s$). This gives a thrust-to-weight ratio of $\frac{6.54N}{0.418kg*9.81m/s^2} = 1.60:1$ which is sufficient for vertical hover.

The speed of the motor is controlled by an Electrifly SS-25 Electronic Speed Controller (ESC).

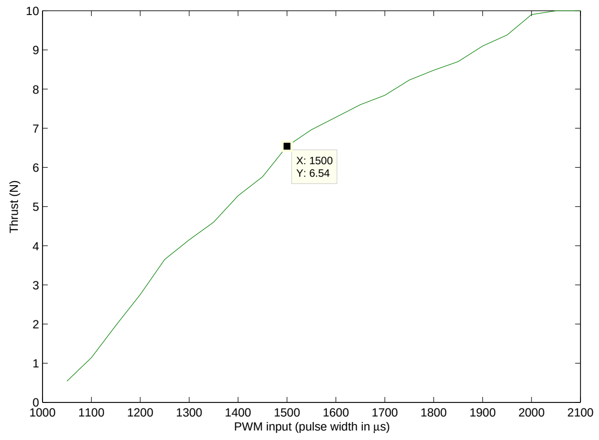


Figure 4.4: Measured thrust as a function of control input.

4.1.4 Radio Control System

The Futaba T7C Transmitter and R617FS Receiver is used for radio control of the plane. This is a 7 channel Digital Proportional RC System that operates at 2.4GHz. While only four channels are strictly needed for controlling the plane (Elevator, Rudder, Ailerons and Thrust), the extra channels give us the flexibility to add more features such as turning on and off on board data logging using a switch on the transmitter. This RC system also complies with standards of the RC hobbyist community, and ensures that an experienced RC pilot will achieve the expected response when controlling the plane.

The control inputs from the transmitter are received by the radio receiver aboard the plane. Y-connectors simultaneously channel the received signal to the actuators and to the DAQ-board for logging. Figure 4.5 shows the transmitter and receiver.

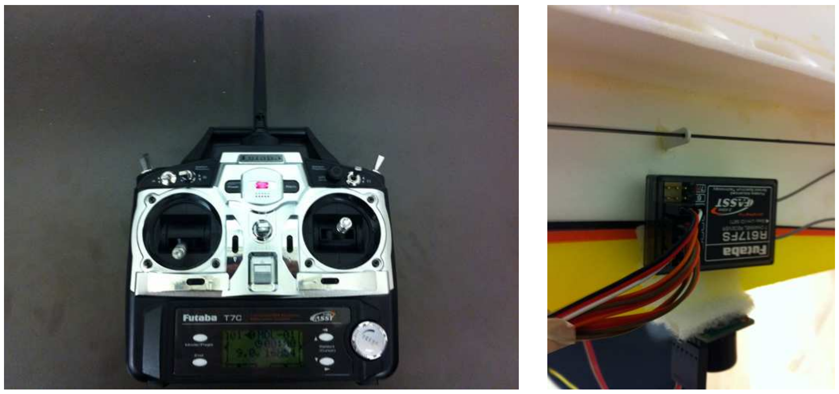

(a) Transmitter (b) Receiver

Figure 4.5: 7 channel Futaba RC system

4.2 Microcontrollers, Sensors and Data Acquisition

The Data Acquisition (DAQ) system was implemented on an ArduPilot Mega board. It is responsible for collecting and logging sensor-data and control inputs. The sensor suite consists of an Inertial Measurement Unit (IMU) that provides orientation and angular velocity, a downward-pointing sonar that provides a reference altitude, a barometric pressure module (BPM) measuring altitude and a Global Positioning System (GPS) module that provides latitude/longitude position. Linear accelerations can be calculated based on the GPS and BPM data. A SD-Card module is used to enable on-board logging of data. The diagram in figure 4.6 gives an overview of the DAQ system.

The servos controlling ailerons, elevator and rudder, as well as the motor controller receive their signals directly from the R/C receiver to ensure no noise disturbance from the data acquisition electronics. Y-connectors are used to feed the input signals simultaneously to the data acquisition board and the actuators. The electronics were carefully distributed and mounted on the fuselage of the plane in order to balance the center of gravity, as shown in figure 4.7.

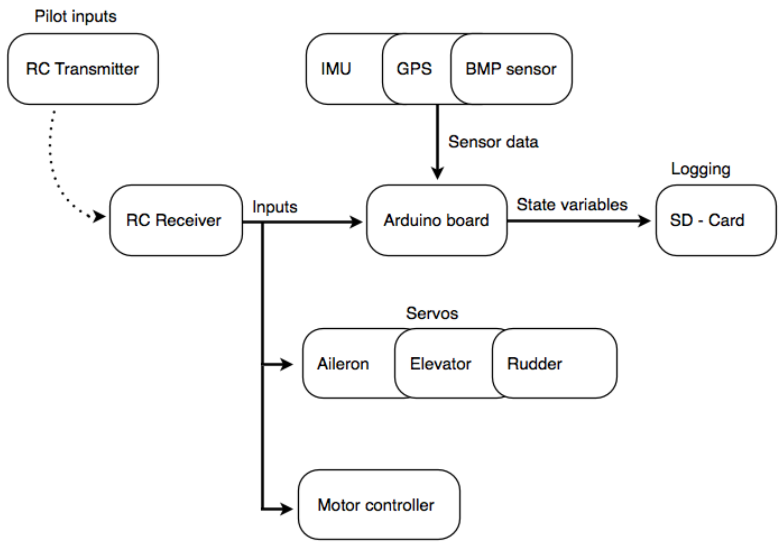


Figure 4.6: Diagram of DAQ system

4.2.1 ArduPilot Mega

The ArduPilot Mega is an Arduino compatible autopilot board developed by DIY Drones. It is based on a 16MHz Atmega1280 processor. A separate circuit (multiplexer chip and ATMega328 processor) can be used to transfer control from the RC system to the autopilot and back again. This setup allows for the main controller to restart without interrupting the RC-signals from the transmitter. The ATMega1280 features 128kB Flash memory, 8kB SRAM and 4kB EEPROM. It has a 8 bit AVR RISC-based microcontroller, and operates at maximum 16MHz. The ArduPilot board comes with a 6-pin GPS connector, 16 analog inputs (with a 10 bit AD converter for each input), and 40 digital input/outputs. It includes 4 serial ports and an SPI interface. Timers are used to enable inputs and outputs of PWMs. Figure 4.8 shows a picture of the ArduPilot Mega board.

The microcontrollers on the ArduPilot Mega are compatible with the Arduino programming environment. The Arduino programming language is a simple open source language for programming microcontrollers. It is based on C++, and C++ libraries can be used directly in Arduino code. Arduino simplifies the programming of the microcontroller by providing a high level programming API hiding low level register manipulations.

4.2.2 Inertial Measurement Unit

The orientation of the plane as well as the linear accelerations and angular velocities are acquired using the MicroStrain 3DM-GX3 25 OEM Inertial Measurement Unit (IMU) shown in figure 4.9. This IMU is a 6 DOF orientation sensor which features an integrated 32-bit low power processor, 17-bit resolution Gyroscopes, Accelerometers, and Magnetometers. The Measurement range of the accelerometers is reported to be $\pm 5g$, while the

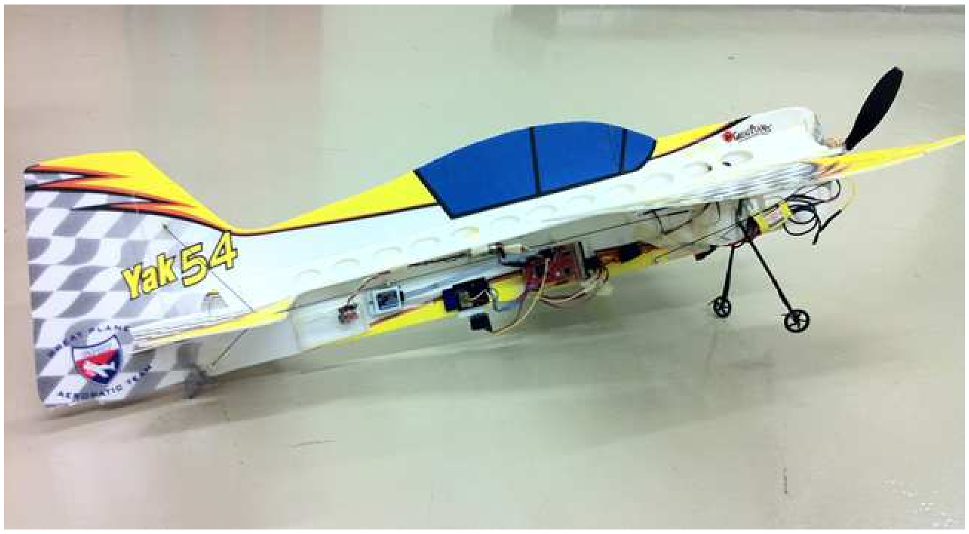


Figure 4.7: Electronics mounted on the plane

gyroscopes have a range of $\pm 300°/sec$ and the built in magnetometers have a range of $\pm 2.5 Gauss$. The maximum output data rate of the IMU is 1kHz.

The IMU is connected to the ArduPilot board through a serial port, and the DAQ software communicates with it using a simple and well documented protocol. Details on the 3DM-GX3 communications protocol can be found in [30].

4.2.3 GPS

We use the MediaTek 3329 Patch-On-Top (POT) GPS module pictured in figure 4.10. This is a small and light weight GPS. The dimensions of the GPS including the breakout board it is mounted on are 30mm x 16mm x 6mm and it weighs 7 grams including the connector cable. This GPS module is compatible with the ArduPilot board, and Arduino libraries for communicating with the module is available. The maximum output data rate is 10Hz.

The GPS guarantees an 2D accuracy of 3m in single GPS mode. This appears too coarse for our application, but it is worth mentioning that the measure is a worst case scenario of absolute accuracy. Our tests with the GPS show that relative accuracy is much better, usually within one meter. It is possible to achieve higher accuracy using a Dual GPS (DGPS), where an additional GPS antenna is placed in a known location on the ground. The MediaTek 3329 data sheet reports 2D accuracy of 2.5m in this configuration. We chose to use the single GPS solution. The increase of accuracy with DGPS of only 0.5 meters does not justify the added complexity to the system.

A problem with GPS 3D positioning is that accuracy of altitude measurements is worse than accuracy in the horizontal plane, generally by a factor of 10. In our case this implies an accuracy of 30m which is not useful. Tests with the GPS showed satisfactory 2D position sensing, while

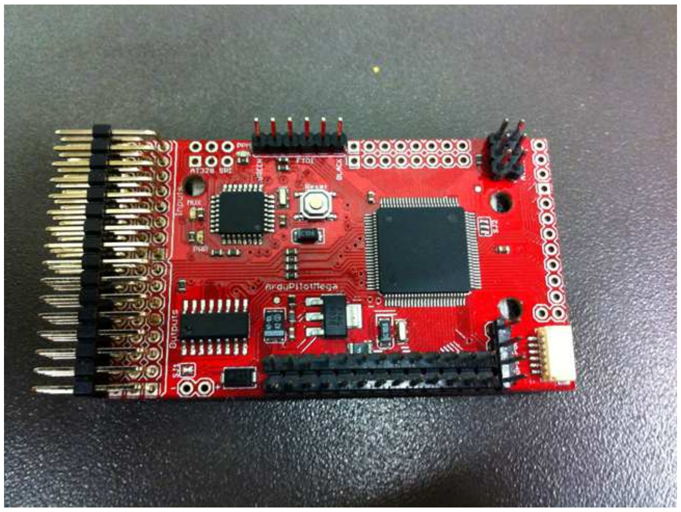


Figure 4.8: ArduPilot Mega board

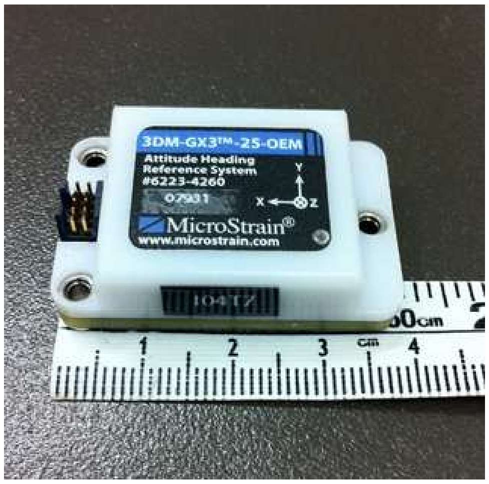


Figure 4.9: MicroStrain 3DM-GX3 IMU

altitude was unreliable. In order to get more accurate altitude readings we incorporated a Barometric Pressure sensor as described next.

4.2.4 Barometric pressure sensor

The Bosch Sensortech BMP085 digital pressure sensor was incorporated to compensate for the poor accuracy of GPS altitude measurements. The sensor module (including breakout board) measures 15.2mm x 15.2mm. The pressure range of the sensor is reported to be 300 - 1100hPa, which corresponds to 9000m above sea level to 500m below sea level [8].

The BPM085 has an incorporated temperature sensor to allow for temperature compensation of the pressure measurement. The altitude in meters above sea level can be calculated using the international barometric

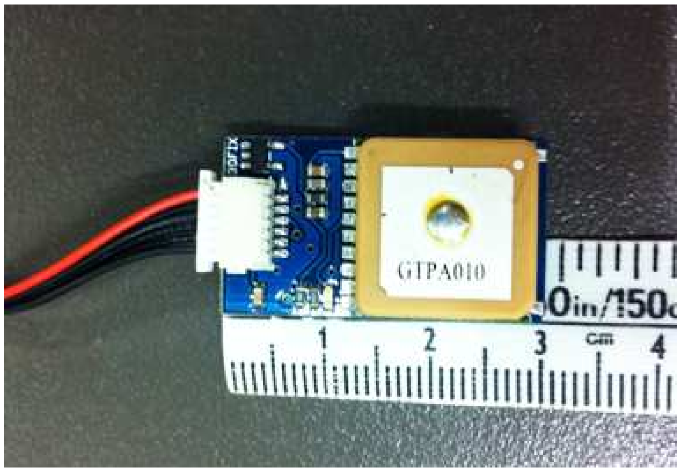


Figure 4.10: MediaTek 3329 GPS

formula:

$$h = 44330(1 - (\frac{p}{p_0})^{\frac{1}{5.255}}) \tag{4.4}$$

Where p is the measured pressure after temperature compensation, and p_0 is pressure at sea level. The pressure at sea level can be obtained by a local weather forecast, but the standard atmospheric pressure, 1013.25hPa is commonly used as an approximation.

Adafruit Industries has developed an open source C++ library specifically to the BMP085 sensor [19]. This library can be used to read pressure and temperature, as well as calculate the altitude according to equation 4.4.

A plastic enclosure is mounted around the sensor to protect it from variations in pressure due to prop-wash as shown in figure 4.11.

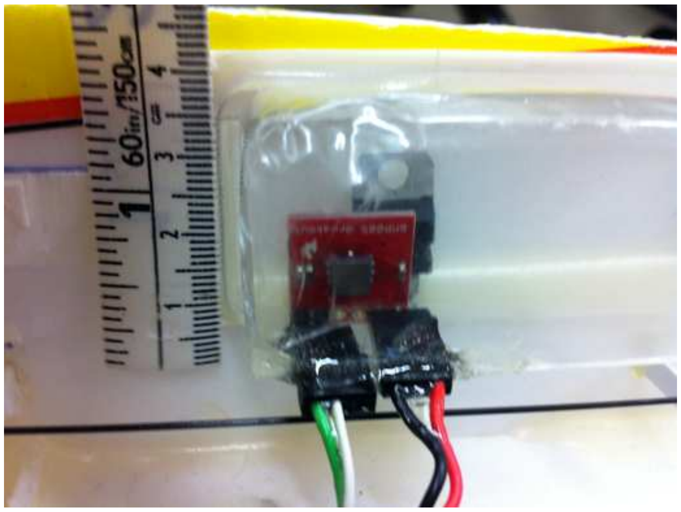


Figure 4.11: Bosch Sensortech BMP085 digital pressure sensor inside plastic enclosure

While the relative accuracy of the BPM sensor is good, $\pm 0.2m$, the absolute accuracy is only $\pm 3m$. In order to get a more accurate absolute altitude we calibrate the BPM data using the on-board sonar.

4.2.5 Sonar

The Maxbotix ultrasonic rangefinder sonar is included in the sensor suite. While we are not relying on the sonar directly in the current configuration it is used as a means of calibrating the BPM altitude. In the current configuration the sonar is mounted underneath the plane, pointing down. This setup allows us to measure distance to ground at low altitudes in near level flight. Figure 4.12 shows a picture of the rangefinder sonar.

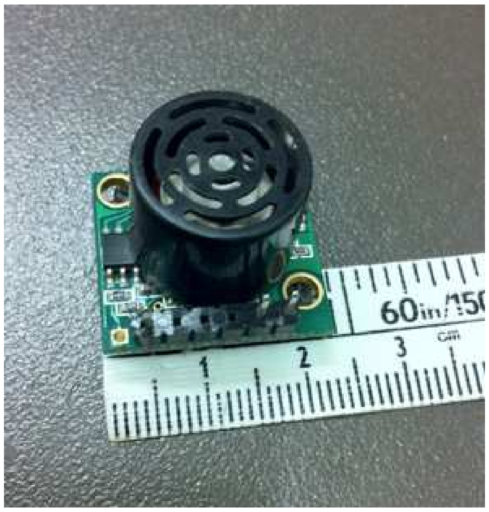


Figure 4.12: Maxbotix ultrasonic rangefinder sonar

The sonar has a maximum range of 6m, while the plane often flies at altitudes higher than this. The orientation of the plane also changes rapidly during flight making it difficult to determine which direction the sonar is actually pointing. These factors make the sonar unreliable as a stand-alone means of altitude measurement, but it can be used to calibrate the BPM altitude as described in section 5.3.2.

In autonomous flight, a sonar could also be used as a means of obstacle detection. A sonar pointing forward could detect obstacles and trigger vertical hover or other obstacle avoiding behaviour. Desbiens and Cutowski successfully used a forward-pointing sonar to detect walls in [25].

4.2.6 Pilot Inputs

Control inputs from the pilot are received by the radio-receiver on the plane. From the receiver the signal is connected directly both to the actuators and to the ArduPilot board by the means of y-connectors. The control input pulse width is measured in micro seconds by the DAQ system on the ArduPilot board, and recorded in the flight log together with the sensor data.

4.2.7 Data Acquisition and Storage

The DAQ software is written entirely in the Arduino programming language. The software reads input from all the sensors and the RC receiver, and logs the data to a SD-Card at approximately 10Hz. The logged data includes a timestamp from the microcontroller (in milliseconds since the start of the program), control input pulse widths, sonar distance, IMU orientation in the form of a rotation matrix, IMU linear accelerations and angular velocity, IMU magnetometer vector, GPS latitude, longitude and altitude, GPS ground speed, ground corse and timestamp, as well as BPM temperature, pressure and altitude.

A DF Robot SD-Card module is used to log data from the demonstration flights. In an autonomous configuration the SD-Card module could be used to log the flight for analysis purposes or load configuration scripts. The SD-Card module is connected to the ArduPilot board using the SPI port.

4.3 Test environment

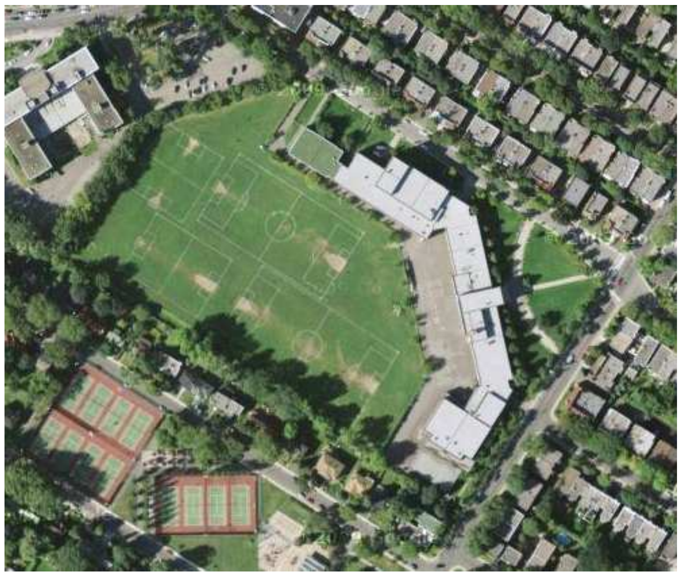


Figure 4.13: Outdoor test space.

Preliminary tests were performed outdoors on the fields of École Royale Vale in Montreal. A satellite photo of the test space is shown in figure 4.13. During outdoor testing it was found that even small wind gusts have a large impact on the control of the plane. If training data is collected under such conditions the recorded pilot inputs will mainly reflect the pilot's effort to compensate for the wind. To avoid collecting data that is affected by wind, it is desirable to conduct the demonstration flights indoors.

Early indoor tests were done in the CEPSUM gym of University of Montreal. Figure 4.14 shows a photo from one of the flights in this gym. The gym we used is a basketball field the size of 15 x 30 meters. This is smaller than ideal, but sufficient to perform the hover maneuver provided the plane is small and capable of slow flight. The greatest downside of collecting training data in the cepsum gym is lack of GPS visibility. This is true for most indoor test spaces. Position information is critical to determining the trajectory of the plane, and other means of recording the plane's position was investigated as described in section 4.4 on the next page.

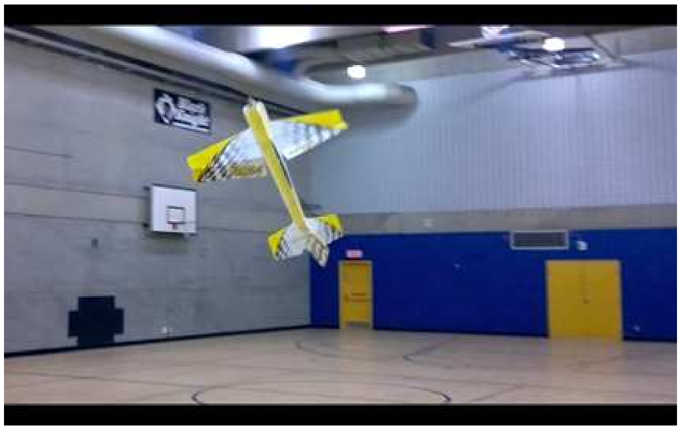


Figure 4.14: Indoor flight in the CEPSUM gym.

Figure 4.15 shows the Stinger Dome of Concordia University in Montreal. This is a temporary dome that cover four soccer fields of size 55 x 30 meters. The walls and roof of the dome are inflatable and does not block GPS visibility. This was the ideal test space for our project with windless conditions, GPS visibility and enough space to perform aerobatic maneuvers.

Figure 4.15: The Stinger Dome of Concordia University in Montreal

4.4 Alternative Positioning Systems

As we need to log the position of the plane, some research was also done on developing a ground based vision system. The idea was to use two or more cameras on the ground to track the plane inside the gym. To achieve this some distinguishable feature of the plane such as colour, needs to be located in each of the camera images. Once a feature is recognized the coordinates in the vertical plane can be tracked by simply translating the x,y coordinates of the feature in the image to real world coordinates. The depth D can be calculated according to equation 4.5 where $f = f_L = f_R$ is the focal length of the camera lens, b is the distance between the two cameras (known as the baseline), while V_L and V_R are the distance from the center of the image to the point in each image, as shown in figure 4.16 [9] .

$$D = \frac{bf}{V_L - V_R} \tag{4.5}$$

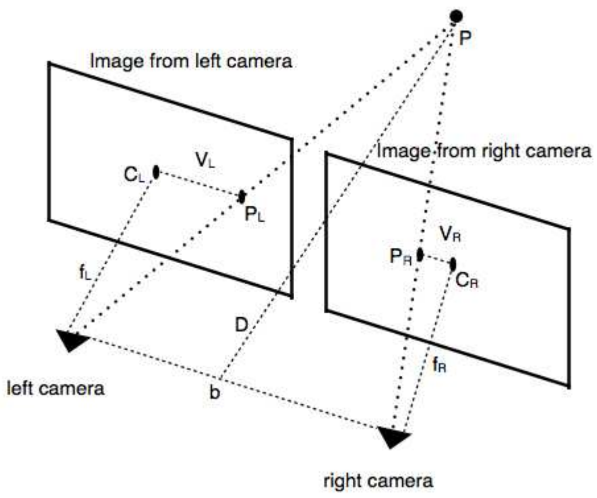


Figure 4.16: Principle of stereo computer vision

The vision system was based on the OpenCV open source library for image processing. A foundation for the vision system was developed in C++ and basic colour tracking was implemented. We were not able to achieve the desired accuracy in 3D tracking within the time frame of the project, but 2D tracking functionality was successfully implemented and might be useful in future work on this or a similar project. Figure 4.17 shows 2D tracking based on colour tresholding using the ground based vision system.

One of the challenges with a ground based vision system is that the tracking data needs to by synchronized with the sensor data collected with the onboard DAQ system. Synchronizing the data manually is time demanding, and it would be useful to find a way to automate this task in the future.

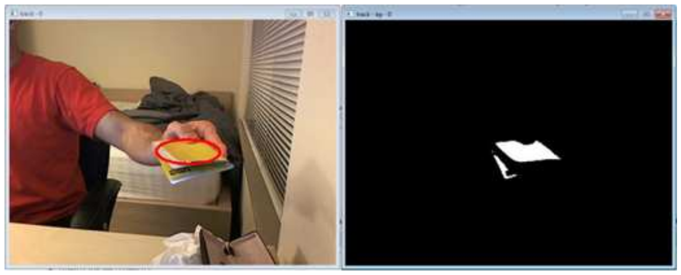


Figure 4.17: 2D tracking based on colour tresholding

As alternatives to the ground based vision system we also investigated commercially available solutions. The OptiTrack and ViCon motion capture systems, Ubisense indoor positioning system and the cricket positioning system developed at MiT were evaluated with respect to price, accuracy and ease of setup.

OptiTrack from Natural Point [35] makes use of motion capture cameras to track objects in three dimensions. this systems offers millimetre precision at a frame rate of 100FPS, and tracks orientation and velocities as well as position. OptiTrack covers a maximum of 6 x 6 meters, which is too small for tracking a motorized plane. Furthermore these cameras would need to be calibrated, and the system is more suitable for a permanent setup. Because of the lack of a permanent test space to fly in we needed a system that would be easy to set up. This is also the most expensive solution we investigated.

The Ubisense Real Time Location System (RTLS) [48] uses receiver stations placed in the corners of the test-space and calculates position using Time Difference of Arrival (TDoA) and Angle of Arrival (AoA) of ultra-wideband (UWB) pulses from a transmitter tag on the target. This system can cover large areas, only limited by the amount of receivers installed. Ubisense is typically used to track people inside large buildings such as hospitals. The accuracy is reported to be 15 cm for stationary objects, and depends on the number and placement of receiver stations. For objects moving at speeds faster than walking the accuracy drops to 1 meter, and the faster the speed the worse the accuracy. The same issues of installation and calibration applies as with the OptiTrack motion capture system.

Finally we considered the Cricket system developed at MiT [36]. Cricket is a low cost, small size and high accuracy indoor positioning system that uses a combination of RF and ultrasound to track an object. Each cricket works both as a transmitter and a receiver. Several crickets are placed at known locations covering the test space and one cricket is placed on the object that is to be tracked. Even though this system is more light weight and might be easier to set up than the others, it still requires us to calibrate the system before every test. While the other systems are off-the-shelf solutions, cricket is open source software and hardware, and some development would be needed on our part to make it work with our project.

4.5 Simulator

While we did not use a simulator for this project It could be useful for future work, for instance when developing a controller. Some research was done on potential flight simulators for this use, and this section provides a brief overview of the flight simulators Flight Gear, X-Plane and Charles River RC-simulator (crrcsim) for simulating a radio controlled plane. Requirements for a good simulator include the ability to model RC-planes with sufficient realism, logging of state variables, possibility to modify or create controllers and customize plane models and transparency.

Charles River is the only simulator in this comparison that is specifically designed for RC-planes. However, the dynamics model the simulator is based upon, the LARCSim, was developed by NASA as a model for full size airplanes [20]. Charles River does not model torque or prop-wash effects, which are essential for realistic simulation of highly aerobatic maneuvers.

FlightGear implements several flight dynamics models. It was originally based upon LARCsim, later JSBsim was implemented and gives support to rockets and lighter than air airplanes. UIUC is a dynamics model based on LARCSim but includes effects of icing. Yasim is the most current flight dynamics model used with FlightGear using different calculation methods than LARCsim and JSBsim. [34]

X-Plane is a flight simulator developed by Laminar Research. It offers realistic simulation of the flight model of a large range of aircraft. The dynamics model of X-plane is based on blade element analysis. Controllers and plane models are easily customized and the simulators can be interfaced with Matlab and Simulink [38].

FlightGear and X-Plane are not designed to show the point of view of an RC pilot (on the ground, tracking the plane). This seems to be the major reason these two programs are not used more frequently for RC training.

Out of the simulators reviewed X-plane seems like the most suitable for the purpose of developing a controller to automate a RC plane. This simulator offers advanced physics modelling, it is easily customized and offers out of the box logging as well as interface to Matlab and Simulink.

Chapter 5

Experiments and Results

This chapter describes the experiments that were conducted as well as their results. Preliminary tests of the experimental platform are first presented before describing the flights for collecting training data for the trajectory learning algorithm. The raw data require pre-processing as described in section 5.3. Finally the results of the trajectory learning are presented.

5.1 Preliminary tests

The preliminary tests served to verify sensor accuracy and improve the experimental platform as described next.

5.1.1 GPS accuracy

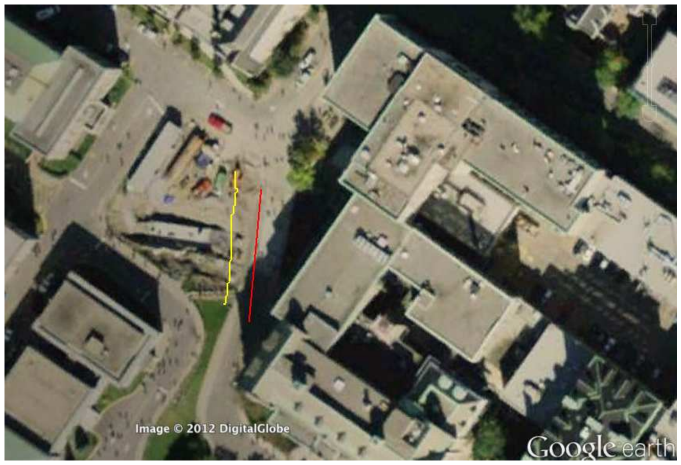


Figure 5.1: On ground GPS test at McGill downtown campus. The red line marks the actual path travelled, while the yellow line shows the logged GPS data.

Ground tests were conducted to determine whether the accuracy of the on-board GPS was adequate for our needs. The data sheet of the MediaTek 3329 GPS module reports a horizontal accuracy of 3m [1]. This is too coarse for our application, however it is a worst case measure of absolute position. The relative position accuracy is potentially better. In other words a GPS plot of a flight may have an offset with respect to global position, but still reflect the flown trajectory accurately in a local frame. Out-door tests were conducted to test the horizontal accuracy of the GPS as well as the overall functionality of the custom build DAQ system. Figure 5.1 shows a Google earth plot of an early GPS ground test. In this test the plane was carried by a person walking along the red line in the plot. The yellow line shows the logged GPS data. While the logged data was off by a few meters in absolute accuracy, the relative accuracy seems adequate. It is worth mentioning that the surrounding buildings at the test space could affect the accuracy of the location data negatively, and that better absolute accuracy might be achieved in a large open space.

The data sheet does not report vertical accuracy of the GPS, and this is generally much worse than horizontal accuracy. According to [4] GPS vertical accuracy is 27.7 meters, which is inadequate for our use. The bad accuracy was confirmed by analyzing the GPS data collected on our indoor flights. Figure 5.2 shows a plot of the GPS altitude and BPM altitude from a hover demonstration flight in the Stinger Dome of Concordia University. The roof of the Stinger Dome is approximately 13 meters at its highest point, while the GPS reports altitudes of nearly 50 meters. Similar observations were made for all flights. BPM altitude was preferred over GPS altitude based on these observations.

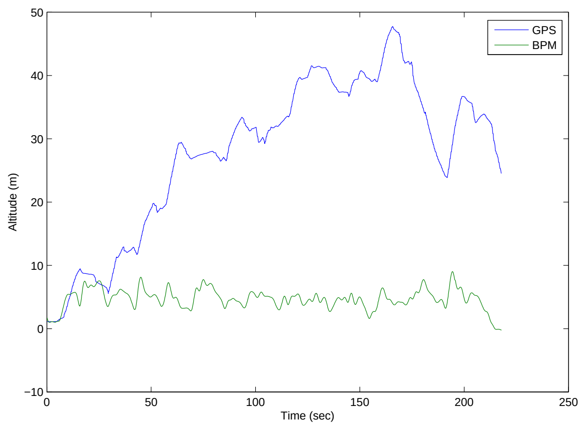


Figure 5.2: Comparison of GPS and BPM altitude data

5.1.2 Outdoor flight tests

A set of outdoor flight-tests was conducted to test the flyability of the plane with the mounted electronics, and to test the onboard DAQ system during real flight. The modifications to the airframe is a result of these early tests as weak points of the airframe was discovered. It also became clear that tests were best preformed indoors after our pilot reported that large corrections in the control inputs were needed for relatively calm wind gusts. Figure 5.3 shows a google earth plot of the GPS data from one of the outdoor flights. The plot closely resembles the actual flight path and indicates that the absolute accuracy is good in open areas.

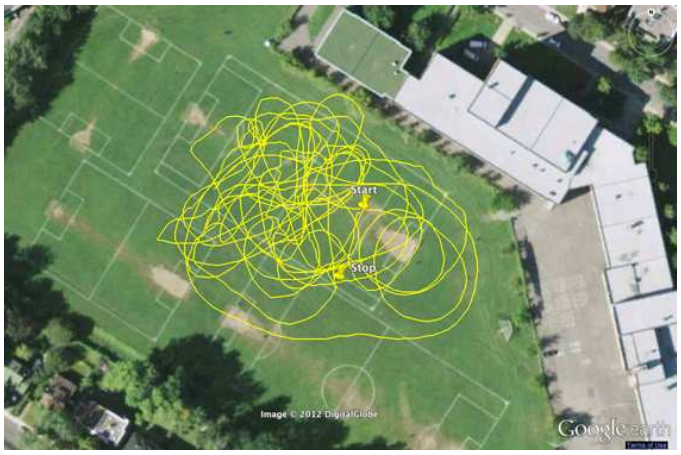


Figure 5.3: In air tests of DAQ system and GPS logging.

5.2 Collecting training data

Training data for the trajectory learning algorithm was collected in the Stinger Dome of Concordia University in Montreal. The goal was to have the algorithm learn an intended trajectory from a set of demonstration flights done by a human RC pilot. Such a trajectory can later be used as a target trajectory for a high level controller automating aerobatic maneuvers. Abbeel et al [2] recommends that at least 10 demonstrations should be performed, from which the 5 most successful repetitions can be selected by hand.

To define the state of the plane along the trajectory the learning algorithm assumes we have:

- The input commands from the RC receiver in the form of pulse width modulations (PWM).
- Linear velocities in inertial frame.
- Angular rates in body frame of the aircraft.
- North, East, Down (NED) position.
- Orientation in the form of quaternions.

The on-board data acquisition system of the YAK54 logs the timestamp of the samples from the micro-controller and from the GPS, input commands from the radio receiver in the form of PWM values for throttle, elevator, ailerons and rudder, linear accelerations, angular rates, magnetometer vector, and rotation matrix from the IMU, distance in centimetres from the sonar, latitude, longitude, altitude, ground speed, and ground course from the GPS, and temperature, pressure and altitude from the BPM.

Two sets of training data were collected. The first maneuver was linear flight with banked turns, and is used as a simple training problem. The second maneuver is the more challenging vertical hover. Both maneuvers are described in more detail next. Videos were recorded of all the flights to allow for visual comparison with the logged data. Videos of both maneuvers are available for viewing at http://ifi.uio.no/majacs/masters.

Figure 5.4: Hand launch of the plane for a training data collection flight in the Stinger Dome.

5.2.1 Linear flight and banked turns

The first demonstration performed was simple linear flight with banked turns. Before the flight the plane was placed on the ground at the pre-defined start point, where the logging was switched on to create an initial reference log. This ensures that we get an updated initial reading of the sensors at zero position. The recorded initial inputs from the radio receiver shows any trimming done by the pilot on field. Altitude readings from the BPM and sonar sensors determines a zero reference point for altitude calculations. Latitude and longitude information from the GPS are used as a zero point in the local NED coordinate system. And the initial GPS coordinates are used in the transformation from global to local coordinates in the pre-processing of the data as described in section 5.3.

After the reference log was recorded the plane was picked up and hand launched as shown in figure 5.4. Once the plane reached a stable altitude the pilot flew along the oval path marked up in figure 5.5. A total of 10 circuits were completed before the pilot landed the plane close to the starting point. The logging was kept on for the entire flight until the plane had landed and was stable on the ground. Five sections of linear flight was chosen from this data as training data for the trajectory learning.

The linear flight with banked turns maneuver was used as an early test case for the learning algorithm. The plane is within the level flight regime

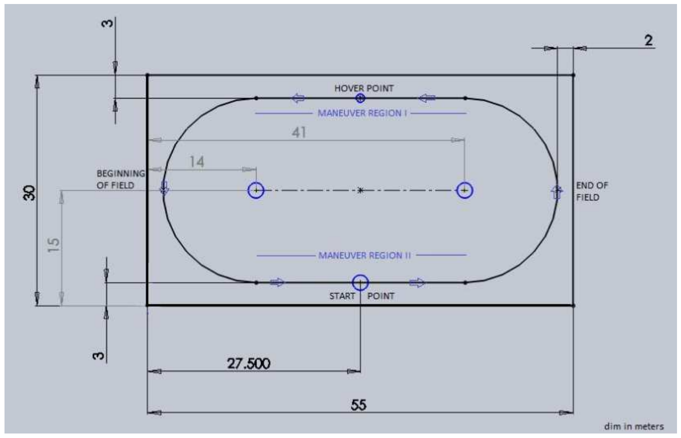


Figure 5.5: Mark-up of the Stinger Dome field for collecting training data.

during the entire flight, and the flight path can be broken down to linear or curved paths for even simpler learning problems.

5.2.2 Hovering with transitions

The second set of demonstrations consisted of hovering including transitions in and out of the hover. As with the linear flight the plane was first placed on the ground at the starting position to create an updated reference log. This was done to account for any changes that might have affected the sensors or servos due to drift or damage to the plane after crashes.

The plane was again hand launched and the plane followed the same oval path as in the linear flight. A point on the flight path was defined as the hover point as shown in figure 5.5. At this point the plane was brought to a vertical hover, and the hover was maintained for a couple of seconds before the pilot transitioned back to level flight and continued along the oval flight path. Figure 5.6 shows a picture of the plane in vertical hover, and the full maneuver is illustrated in figure 5.7.

The hover maneuver provides a more interesting learning problem for the learning algorithm, as the plane maneuvers outside of the level flight regime. While in hover the plane is pitched up 90 degrees, and the air-flows during the transition in and out of the maneuver is highly unpredictable [45]. For this reason it is difficult to hand-craft a flyable trajectory, and learning the trajectory from demonstration flights provides a promising solution.

The hover was performed 10 times, and the five best demonstrations were selected during pre-processing.

5.3 Pre-processing

The raw data collected from the demonstration flights needs pre-processing before it can be used as training data. The trajectory learning algorithm expects state input of the following format:

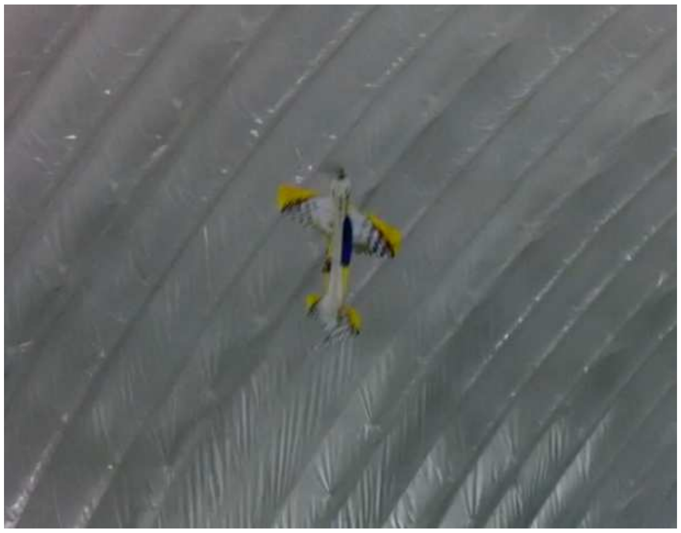


Figure 5.6: Vertical hover in the Stinger Dome.

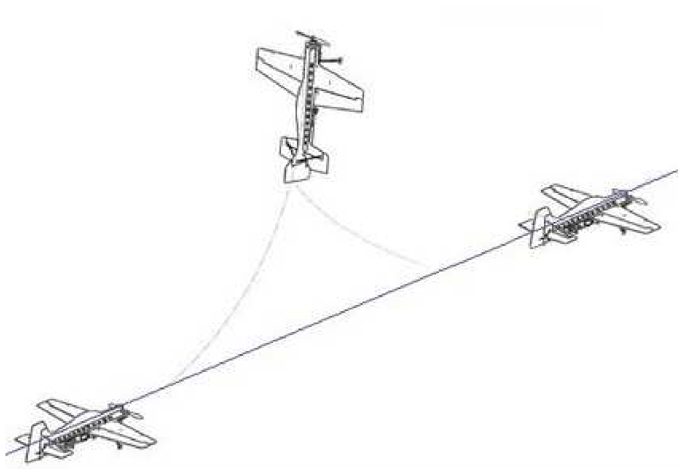


Figure 5.7: Vertical hover.

$<\dot{n}\dot{e}\dot{d}><NED><pqr><quaternion><control\,inputs>$

The BPM data was first filtered and calibrated as shown in section 5.3.2. Then local NED cooridinates are calculated from GPS and calibrated BPM data as described in section 5.3.3. Differentiating the NED position data gives the linear velocities in inertial frame ($\dot{n}\dot{e}\dot{d}$). The orientation requires transformation from the rotation matrix representation in the raw data to the quaternion representation expected by the trajectory algorithm. The transformation from rotation matrix to quaternion is given in section 5.3.4. The control inputs from the pilot was recorded in the form of pulse widths, and need conversion to deflection angles as described in section 5.3.5.

An important part of the pre-processing is to select good demonstrations from the set of collected training data. This process is done manually as described next.

5.3.1 Selecting good demonstrations

In the initial flights, a new log was created for each demonstration, but this approach proved not ideal and was abandoned. The logging is controlled by the pilot, and the continous log switching provides a distraction. As a result the start and stop point of the log files will vary in the best case, in the worst case the pilot may forget to switch on the logging at all.

Based on the experience with log switching a decision was made to keep the logging running for the entire flight. Because the demonstrations are all combined into one file, the different demonstrations need to be identified in the raw log-file. Once the demonstration regions are located, the best five are selected for further processing. This was done manually by examining the position and orientation data and comparing with videos of the flight.

From the demonstrations of linear flight with banked turns, we looked for regions of the log where the plane is flying as close to a straight line as possible. A position plot of one such demonstration is shown in figure 5.8.

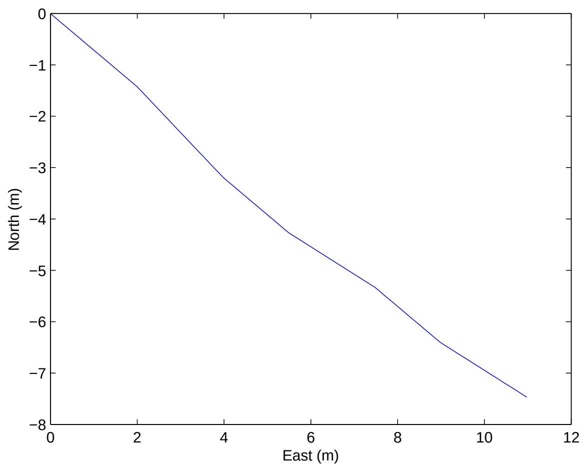
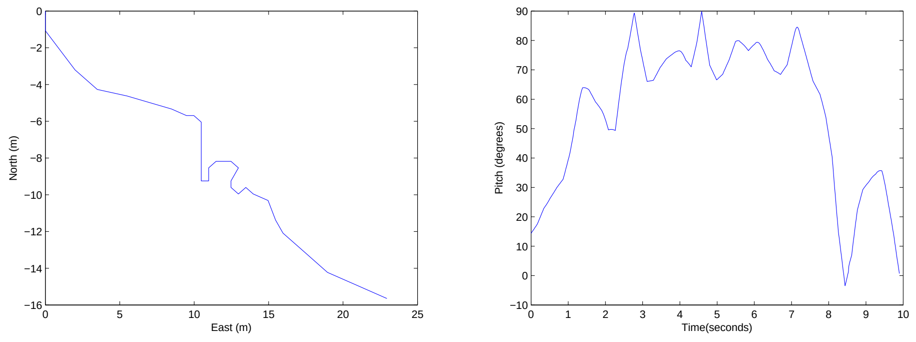


Figure 5.8: East - North plot of a good linear flight demonstration.

In good hover demonstrations the plane does smooth transitions between level flight and hover, it keeps a pitch angle close to 90 degrees for 4 or more seconds, and stays close to a steady position during the hover. Figure 5.9 shows the position and pitch angle of the plane during a good hover demonstration.

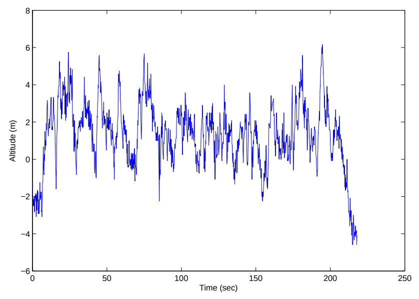
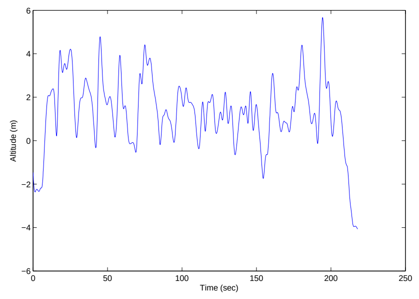
5.3.2 BPM filtering and calibration

The raw data from the barometric pressure sensor is subject to noise, and needs filtering before further use. We applied a 3rd order Butterworth low-pass filter with a cut-off frequency of 1Hz. Figure 5.10 shows the BPM signal before and after filtering.



(a) East-North Position (b) Pitch angle

Figure 5.9: Position plot and pitch angle from a good hover demonstration

(a) Raw BPM data (b) Filtered BPM data

Figure 5.10: BPM data is filtered using a 3rd order Butterworth low-pass filter with 1 Hz cut-off frequency.

The BPM data is calculated based on the standard sea level pressure and gives an inaccurate absolute altitude. Furthermore the sensor is subject to drift over time. Figure 5.11 shows the uncalibrated BPM data and the Sonar reference readings from the hover demonstrations described in section 5.2.2.

At first glance the sonar data looks unusable. That is due to the fact that for most of the flight the plane flew at an altitude greater than 6 meters, which is beyond the range of the sonar. There are however regions where the sonar data is reliable; i.e., when the altitude is in the range 0.17m - 6m and the orientation of the plane is near level.

The sonar data points from reliable regions were used as references to calibrate the BPM height. Two points along the trajectory were selected manually, and BPM offset was calculated by subtracting the BPM reading from the sonar reading at the calibration points. The offset was added to the BPM data within the region of each calibration point.

Calibration points were found manually by closely inspecting the sonar data as well as the orientation of the plane. Figure 5.12 gives a zoomed in view of the graph in figure 5.11. The two graphs show the regions where suitable calibration points were located. For the hover demonstrations

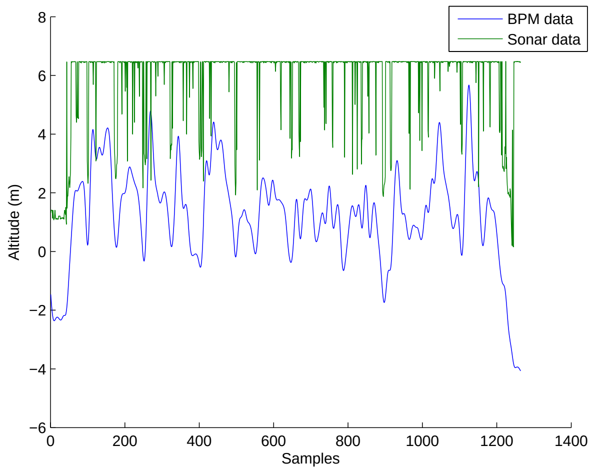
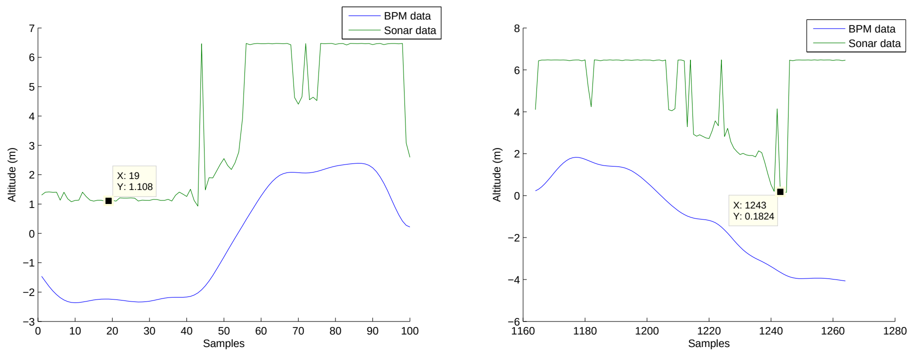


Figure 5.11: Uncalibrated BPM data and Sonar reference signal

calibration points were chosen at samples 19 and 1243. The offset calculated at sample 19 was added to the BPM signal from sample 1 to 1132. The offset at sample 1243 was added to the BPM signal from sample 1133 to the end.

The same procedure was used to calibrate BPM data from the linear flight.



(a) Calibration point a, sample 19

(b) Calibration point b, sample 1243

Figure 5.12: Uncalibrated BPM data and Sonar reference at the two regions where calibration points were selected.

The calibrated BPM data is shown in figure 5.13 and 5.14. The data corresponds better to the sonar data at reliable data points after calibration.

5.3.3 Localization of Position Data

The GPS data is recorded in global Latitude North and Longitude East degrees and needs to be converted to a localized NED frame. The transformation is simplified using the flat earth assumption. This will not

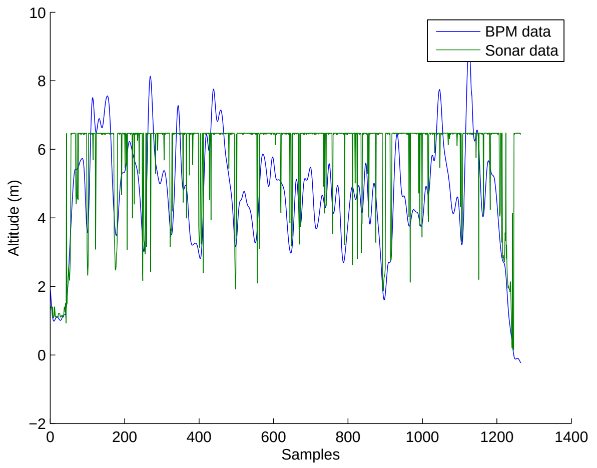
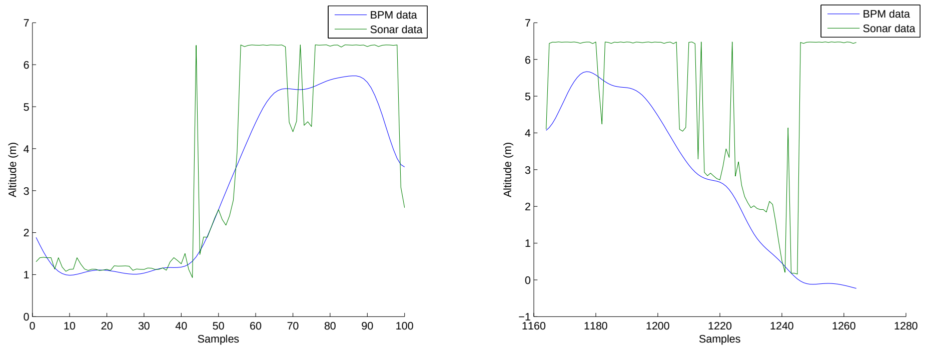


Figure 5.13: Calibrated BPM data and Sonar reference signal



(a) Region of calibration point at sample 19 (b) Region of calibration point at sample 1243

Figure 5.14: Zoomed in view of the two calibration regions after calibration

introduce large errors due to the small area normally covered by an RC-plane. NED coordinates are calculated as shown in equation 5.1, where r is the mean radius of the earth in meters, ϕ and λ are latitude and longitude in radians, h is altitude in meters and ϕ', λ' and h' are the respective local references.

$$\begin{aligned} n &= r(\phi - \phi') \\ e &= r cos(\phi)(\lambda - \lambda') \\ d &= -(h - h') \end{aligned} \tag{5.1}$$

5.3.4 IMU data

Singularities Euler angles is commonly used to represent orientations in aerospace research. A limitation of this representation is that singularities occur at angles of ±90° [24]. In the hover maneuver the goal is to keep the plane pitched up at exactly 90°, and the limitations of the Euler angle representation could introduce significant errors. To avoid the problem of singularities we log the entire direct cosine matrix (dcm) representing the orientation of the IMU with respect to inertial frame.

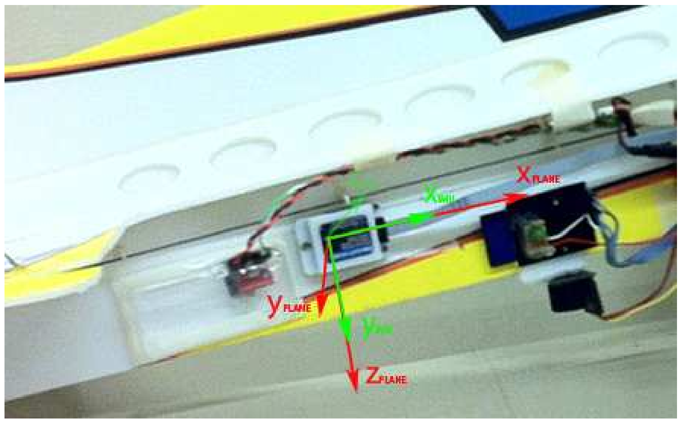


Figure 5.15: IMU frame and body frame of the plane.

IMU offset The IMU is placed on the side of the fuselage of the plane as shown in figure 5.15. The orientation calculated by the IMU assumes a reference frame where the Z-axis points down, the X-axis points forward and the Y-axis points to the side from the IMU. This would correspond to the body frame of the plane only if the IMU was mounted horizontally with the correct side up. Because the IMU is placed vertically, there is an offset between the IMU frame and the body frame of the plane. The offset corresponds to a rotation of 90° around the X-axis of the plane, and is accounted for by the following rotation matrix [43]

$$R_p^{imu} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{5.2}$$

The orientation is logged as a rotation matrix that represents a rotational transformation from IMU frame to Inertial frame, R^i_{imu}. We can view the IMU offset as a rotational transformation from the plane body frame to the IMU frame about the plane x-axis, R_p^{imu}. We want a rotation matrix R_p^i, representing a rotational transformation from the body frame of the plane to inertial frame. R_p^i can be composed from the rotations represented by R^i_{imu} and R_p^{imu} as follows [43]:

$$R = R_p^i = R^i_{imu} * R_p^{imu} \tag{5.3}$$

The angular velocities are transformed from IMU Frame (pqr_{imu}) to the body frame of the plane (pqr_p) by multiplying with the offset matrix R_p^{imu} [43].

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = R_p^{imu} * \begin{bmatrix} p \\ q \\ r \end{bmatrix}_{imu} \tag{5.4}$$

Quaternion representation While the rotation matrix representation avoids the problems of singularities it is memory intensive from a computer programming perspective. While Euler angle representation require 3 elements to be stored, the rotation matrix requires 3x3 elements. A more compact representation that avoids singularities would be ideal.

The quaternion is an example of such representation. It consist of a 3 element vector component $[q_X, q_Y, q_Z]^T$ and a scalar component q_W. The scalar component represents the rotation while the vector component gives the direction of the rotation axis. There are two common conventions when representing quaternions, with the scalar component as the first or as the last element. We use the latter representation, $Q = [q_x, q_y, q_z, q_w]^T$.

Our IMU does not output quaternions directly (current versions of the MicroStrain 3DM-GX3 IMU has an updated firmware that offer this feature, but this was not yet available at the time of purchase of our IMU). And a transformation from matrix to quaternion representation is necessary. The relationship between matrix and quaternion is given in [24] as :

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{12} \\ r_{21} & r_{22} & r_{22} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} 2q_w^2 - 1 & 2q_x q_y & 2q_x q_z \\ +2q_x^2 & +2q_w q_z & -2q_w q_y \\ & & \\ 2q_x q_y & 2q_w^2 - 1 & 2q_y q_z \\ -2q_w q_z & +2q_y^2 & +2q_w q_x \\ & & \\ 2q_x q_z & 2q_y q_z & 2q_w^2 - 1 \\ +2q_w q_y & -2q_w q_x & +2q_z^2 \end{bmatrix} \tag{5.5}$$

From equation 5.5, assuming R is orthogonal we get:

$$\begin{aligned} 4q_w q_x &= r_{23} - r_{32} \\ 4q_w q_y &= r_{31} - r_{13} \\ 4q_w q_z &= r_{12} - r_{21} \\ 4q_w^2 - 1 &= tr(R) \end{aligned} \tag{5.6}$$

q_w can then be calculated as follows:

$$q_w = (1/2)\sqrt{m_{11} + m_{22} + m_{33} + 1} \tag{5.7}$$

Once we have q_w the vector component of the quaternion can easily be calculated based on the relationships in equation 5.6

$$\begin{aligned} q_x &= (m_{23} - m_{32})/4q_w \\ q_y &= (m_{31} - m_{13})/4q_w \\ q_z &= (m_{12} - m_{21})/4q_w \end{aligned} \tag{5.8}$$

5.3.5 Pilot control inputs

The control inputs are given in the form of PWM commands that are fed directly to the servos. In order to simulate the effect of the control inputs on the state, these commands need to be converted to deflection angles. The control input ranges from pulse widths of $900\mu s$ to $2100\mu s$, but the mapping between control input and deflection angle is not necessarily linear, and it can be different for the different control surfaces. A control surface deflection model for each control surface is thereby necessary.

In order to map the control inputs to corresponding deflection angles an experiment was conducted where different control inputs were given to the plane and the resulting deflection angle was measured.

Aileron deflection. The aileron deflection is a quadratic function of the input pulse width. Figure 5.16 shows a plot of the aileron deflection data together with a 2nd order polynomial fit given by:

$$\delta_a = (9.1945 * 10^{-07}) u_a^2 - 0.0048 u_a + 5.2353 \tag{5.9}$$

where δ_a is the deflection angle in radians and u_a is the aileron input

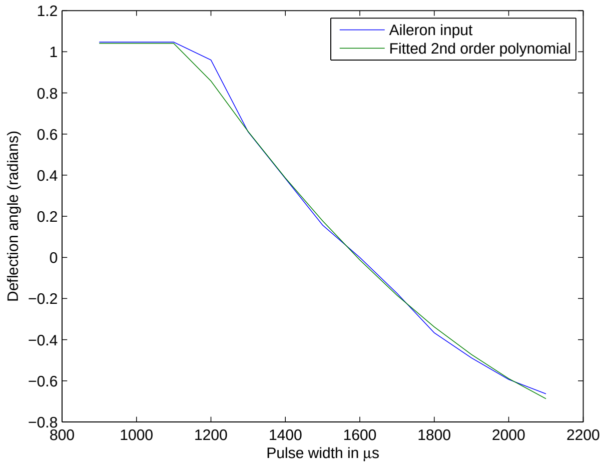


Figure 5.16: Aileron deflection as a function of input pulse width

pulse width in μs. In the experiment the aileron deflection angle reached its maximum value at a pulse width of $1150\mu s$ and its minimum angle at $2100\mu s$, in the servo model we therefore constrain the aileron inputs to these limits.

Elevator deflection. The elevator deflection is a linear function of the input pulse width. Figure 5.17 shows a plot of the elevator deflection data together with a fitted linear function given by:

$$\delta_e = \frac{u_e - 900}{1200} * (1.2387 + 1.2051) - 1.2387 \tag{5.10}$$

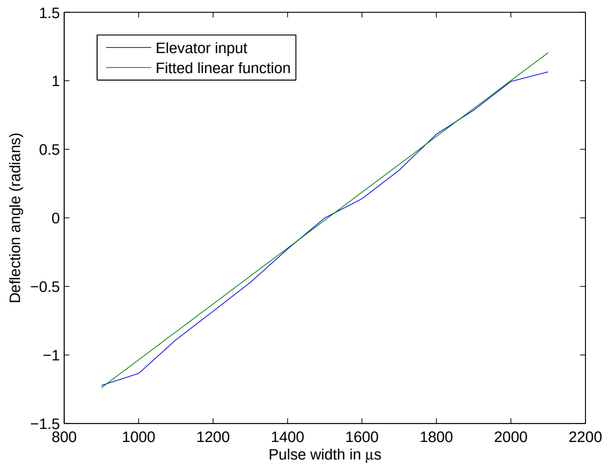


Figure 5.17: Elevator deflection as a function of input pulse width

where δ_e is the deflection angle in radians and u_e is the elevator input pulse width in μs. The elevator commands are limited to the pulse width range $900\mu s$ to $2100\mu s$, which is equal to the total range of the control inputs.

Rudder deflection. The rudder deflection is a linear function of the input pulse width. Figure 5.18 shows a plot of the rudder deflection and the fitted linear function given by:

$$\delta_r = \frac{u_e - 1000}{1000} * (1.344 + 1.2564) - 1.344 \tag{5.11}$$

where δ_r is the deflection angle in radians and u_r is the rudder input pulse

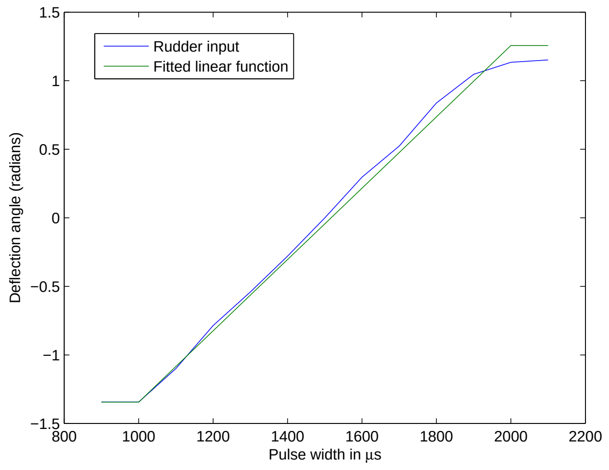


Figure 5.18: Rudder deflection as a function of input pulse width

width in μs. The rudder inputs are limited to the range $1000\mu s$ to $2000\mu s$.

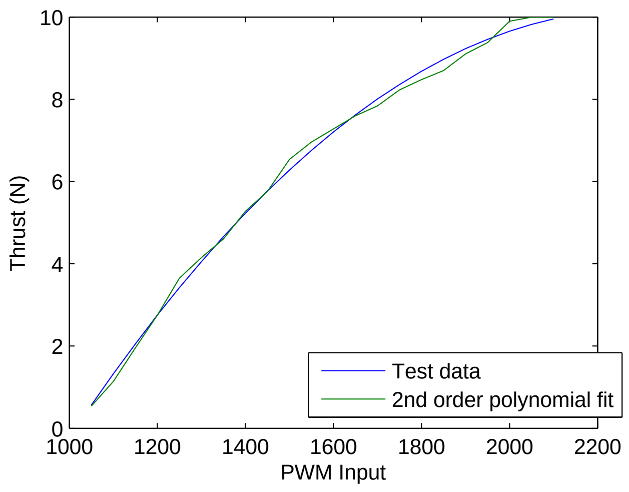


Figure 5.19: Thrust in Newtons vs PWM command

Thrust. The throttle commands are also given as PWM signals, and need to be converted to Newtons. Another experiment was conducted to measure the thrust at different throttle commands. The thrust was measured while the motor was mounted on a stable surface. Figure 5.19 shows the measured relationship between thrust and PWM input, together with a second order polynomial fit given by the following equation:

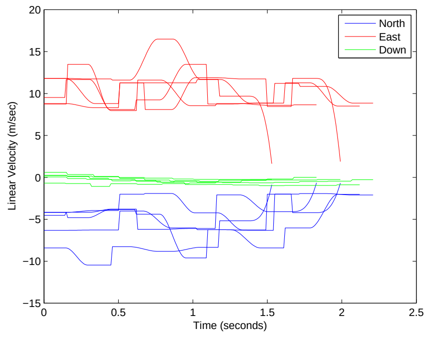
$$F = 6.2628 * 10^{-6} u^2 + 0.0287 u - 22.6335 \tag{5.12}$$

5.4 Results

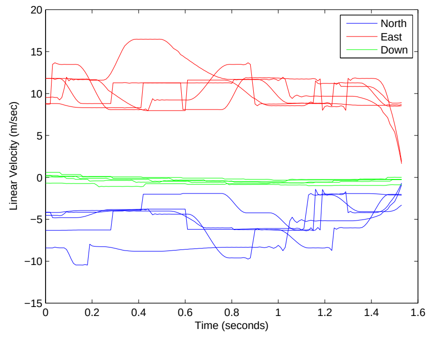
The trajectory learning algorithm was tested on two maneuvers, a straight line of level flight and a vertical hover. The training data for the straight line was collected in the "linear flight with banked turns" demonstrations described in section 5.2.1, and the vertical hover data was collected as described in section 5.2.2. First the results of the time alignment of demonstrations are presented, then the learned trajectories for the level flight and the hover flight, and finally the model prediction errors.

5.4.1 Time alignment

The demonstration trajectories are aligned in time using DTW as described in section 3.1.1. Figure 5.20 shows the linear velocities from the level flight before and after alignment. After the alignment all demonstration trajectories have the same length in time, and similar features of the demonstration trajectories are aligned. The latter is more obvious in the example in figure 5.21 which shows the angular velocity around the z-axis (i.e. the yaw-rate, r) of the hover flight.

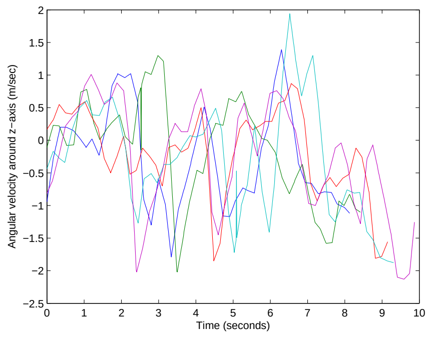


(a) Before alignment

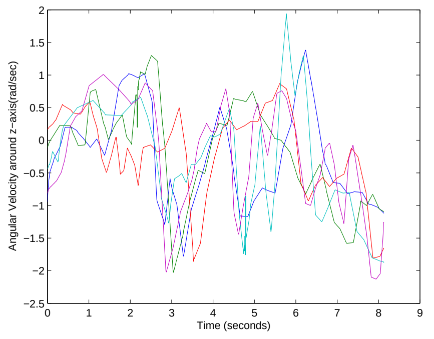


(b) After alignment

Figure 5.20: Linear velocities ($\dot{n}\dot{e}\dot{d}$) from the linear flight.



(a) Before alignment



(b) After alignment

Figure 5.21: Angular Velocities around the plane z-axis (r).

5.4.2 Straight line of level flight

The first maneuver used to test the trajectory learning algorithm was a straight line of level flight. Training data for this manuever are regions of linear flight in the "linear flight with banked turns" demonstration set. Figure 5.22 shows an East-North plot of the position of the plane during the maneuver. The learned trajectory is closer to a straight line than any of the demonstrations, and is achieved without supplying previous knowledge about the trajectory to the algorithm.

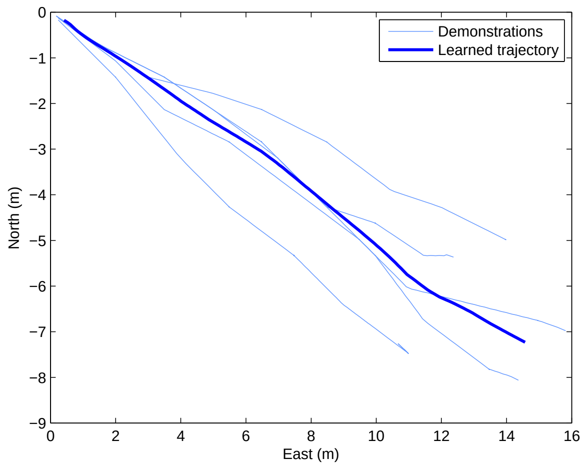


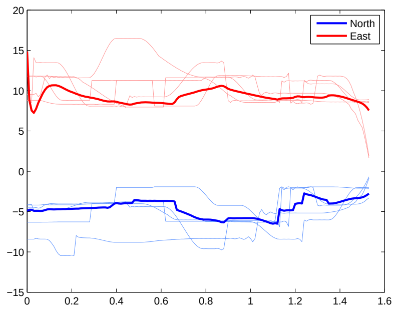
Figure 5.22: East-North position of level flight (meters)

As shown in figure 5.23, the altitude is fairly constant and the altitude velocity is close to zero. This is consistent with a straight line flight flown at constant height. Note that the altitude and altitude velocity is here represented as height above ground for convenience. In reality the North East Down convention is used in the flight logs, and thus the altitude would appear to be negative.

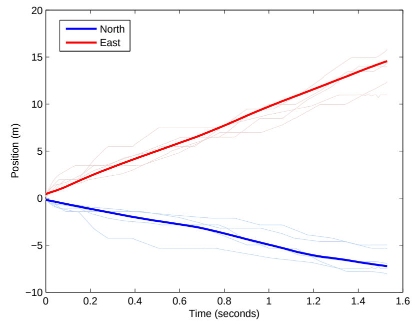
The Euler angles plotted in figure 5.24 shows that the orientation of the learned trajectory together with the aligned demonstrations. The shifting roll and turning yaw angles of the demonstrations reflect that the line is flown in a constricted space. After flying the straight line the plane entered into a left turn, thus the plane rolled right wing up and started turning its nose left. The pitch angle is also increased towards the end of the demonstrations, which slows down the speed of the plane before the turn. Since this is a consistent pattern in all the demonstrations the learning algorithm interprets it as part of the maneuver. Note that the yaw angle is kept more stable in the learned trajectory than in any of the demonstration trajectories before preparing for the turn.

As can be seen from figure 5.25, the angular velocities correspond to the changes of orientation in figure 5.24.

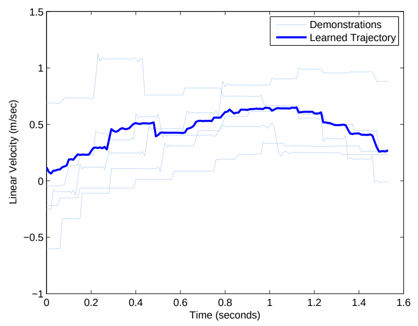
A pattern corresponding to the Euler angle changes can be seen in the control surface deflections as well. Figure 5.26(a) shows a positive aileron deflection towards the end of the trajectory which by convention



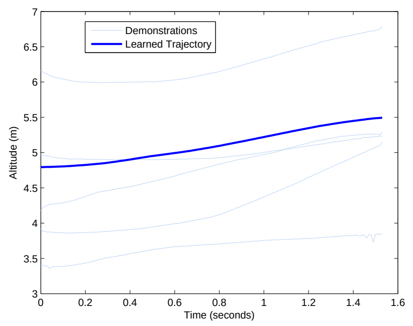
(a) North and East velocity (meters/sec)



(b) North and East position (meters)



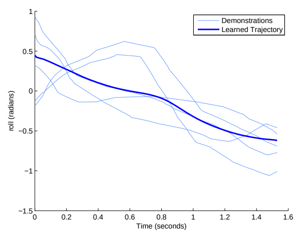
(c) Altitude velocity (meters/sec)



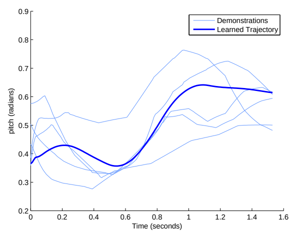
(d) Altitude (meters)

Figure 5.23: Position and linear velocity of the plane during a straight line of level flight.

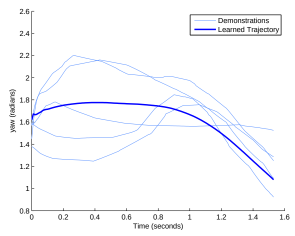
corresponds to right aileron down, and leads to a right wing up roll moment [39]. The rudder deflection is slowly increased as shown in figure 5.26(d), which means the rudder is moved to the left, turning the nose of the plane left as well. Figure 5.26(b) and (c) shows the elevator is moved down (pitching the nose of the plane up) and the thrust is decreased, to slow down before the turn.



(a) Roll (radians)



(b) Pitch (radians)



(c) Yaw (radians)

Figure 5.24: Orientation during level flight (Euler Angles).

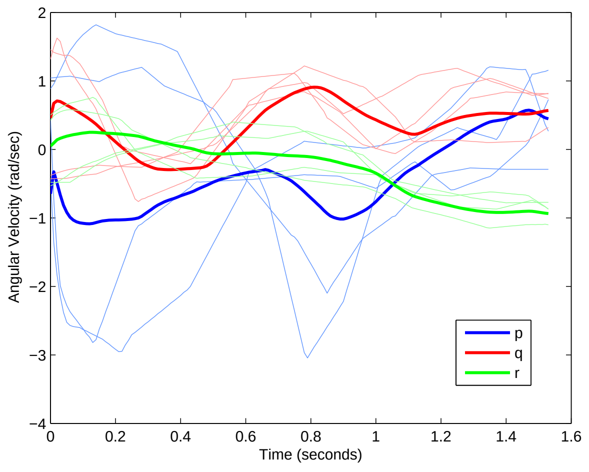
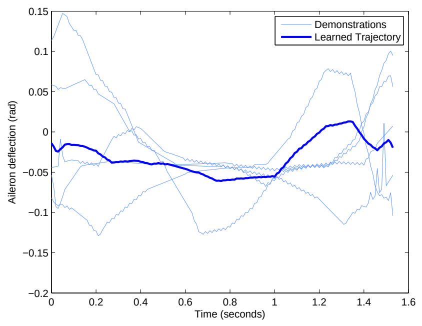
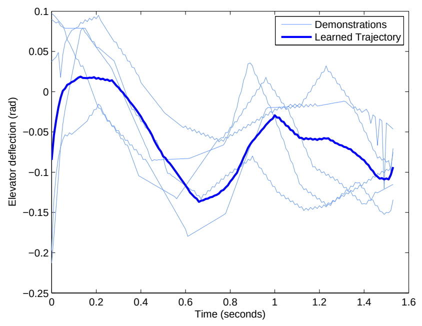


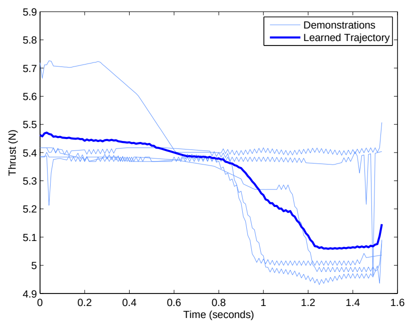
Figure 5.25: Angular velocities pqr (rad/sec), level flight.



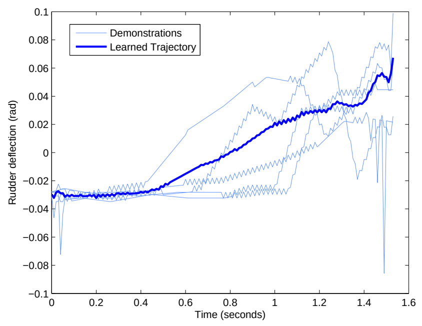
(a) Aileron deflection (radians)



(b) Elevator deflection (radians)



(c) Thrust in Newton



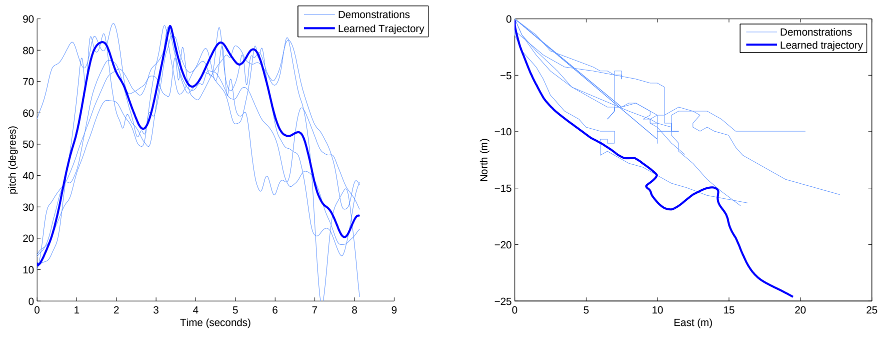
(d) Rudder deflection (radians)

Figure 5.26: Control inputs, level flight.

5.4.3 Vertical hover

The vertical hover maneuver gives a more challenging learning problem as the trajectory is more complex and longer in time. It is also more challenging to the human pilot, which means we are less likely to have repeatedly good demonstrations. Still, the algorithm successfully inferred the intended hover trajectory.

Figure 5.27 gives a plot of the pitch angle and position of the plane during the hover maneuver. The blue thick line shows the learned trajectory, while the thin lines show the demonstrations. As can be seen from the plot, the learned trajectory makes the plane pitch up to 83 degrees and maintains a pitch angle between 55 and 89 degrees for 5.5 seconds before pitching down to resume level flight.

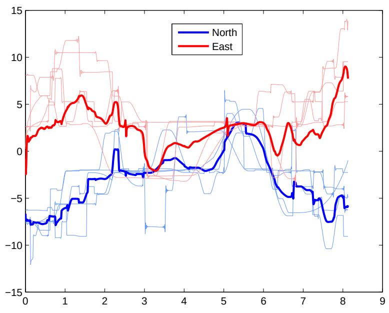


(a) Pitch (degrees) (b) East-North position (meters))

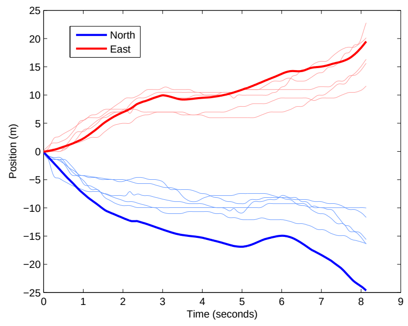
Figure 5.27: Hover maneuver.

The plane moves further south in the learned trajectory than in any of the demonstration trajectories. This is because the position must not only be the most likely position considering the demonstration trajectories, it must also adhere to the dynamics of the plane, and thus the position also depends on the most likely linear velocities. The learned linear velocity in North direction explains the deviating North position. As shown in figure 5.28 the change in position corresponds to the learned linear velocities. Figure 5.4.3 and (d) shows that the plane looses altitude during the hover, which is consistent with visual observations of the maneuver.

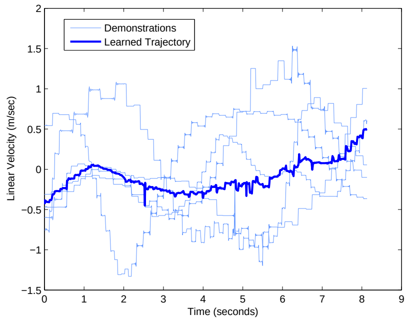
Figure 5.29 shows the angular velocities during the hover maneuver, and figure 5.30 shows the orientation in radians. It is a consistent pattern that the plane rolls right wing up as the pilot is establishing control in the hover, this can be seen in the peak at around 3 seconds in figure 5.30(a). The yaw is affected as well, as can be seen in figure 5.30(c).



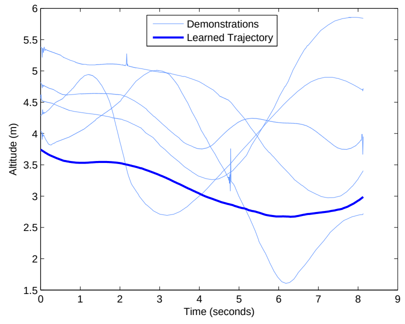
(a) North and East velocities (m/sec)



(b) North and East position (meters)

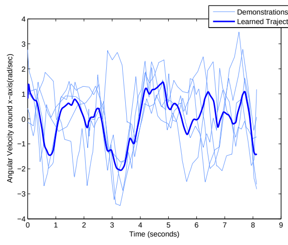


(c) Altitude velocity (m/sec)

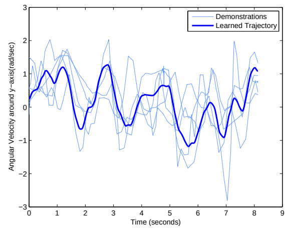


(d) Altitude (meters)

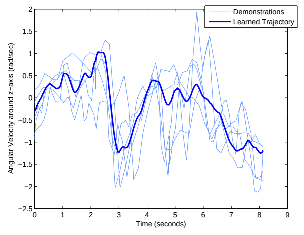
Figure 5.28: Hover maneuver - Linear velocities and position.



(a) p

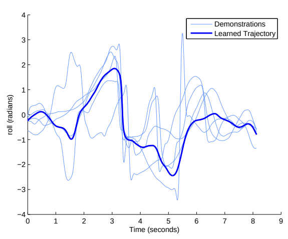


(b) q

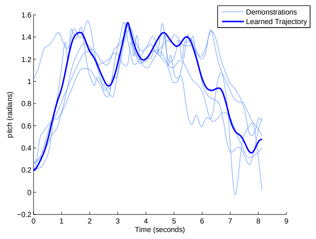


(c) r

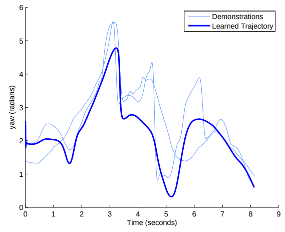
Figure 5.29: Angular Velocities, hover flight (rad/sec).



(a) Roll



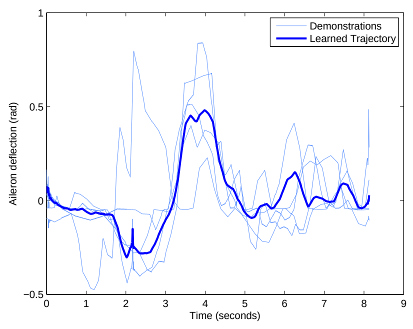
(b) Pitch



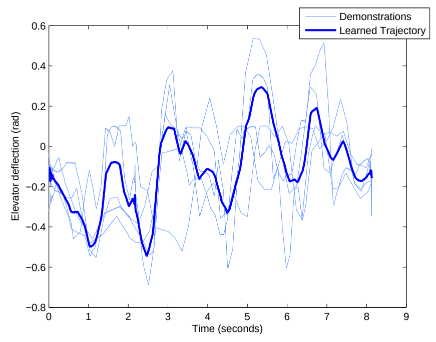
(c) Yaw

Figure 5.30: Orientation during hover (Euler angles).

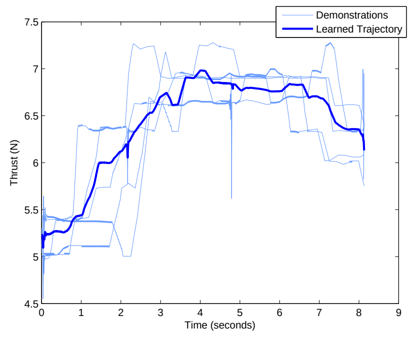
Figure 5.31(c) shows a great increase in thrust during the hover maneuver. This is needed because the wings have no lift at such a high pitch angle, and the entire lift of the plane must be generated by the propeller. The aileron and rudder deflections in figure 5.31(a) and 5.31(d) shows the compensations done by the pilot to establish control over the plane in the vertical hover. The elevator deflection in figure 5.31(b) shows the control of the pitch angle to make the plane maintain a near vertical position during the hover. In general the deflection angles are greater for the hover maneuver than for the level flight, this is due to the fact that the plane flies at a slower speed.



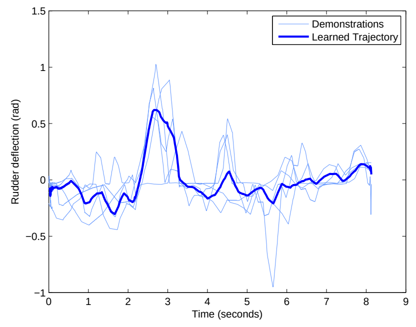
(a) Aileron deflection (radians)



(b) Elevator deflection (radians)



(c) Thrust (Newton)

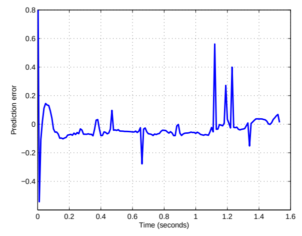


(d) Rudder deflection (radians)

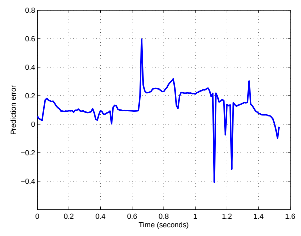
Figure 5.31: Control inputs, hover maneuver.

5.4.4 Model Prediction Error

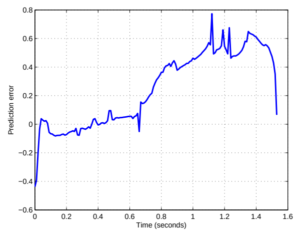
The learning algorithm calculates model prediction error along the trajectory, for the simple dynamics model. The model is then adapted by using the prediction error as bias terms. Figure 5.32 and 5.33 shows the prediction error in linear and angular accelerations for the straight line level flight trajectory. Figure 5.34 and 5.35 shows the errors in $\dot{u}\dot{v}\dot{w}$ and $\dot{p}\dot{q}\dot{r}$ along the hover trajectory.



(a) $\dot{u}$

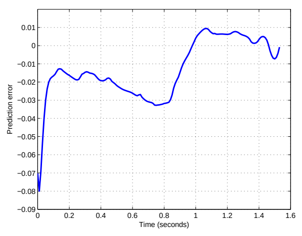


(b) $\dot{v}$

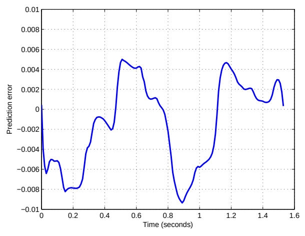


(c) $\dot{w}$

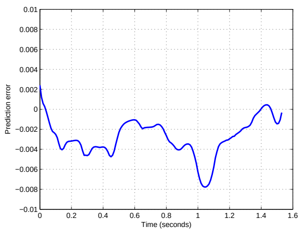
Figure 5.32: Model prediction error in linear accelerations ($\dot{u}\dot{v}\dot{w}$) during straight flight.



(a) $\dot{p}$

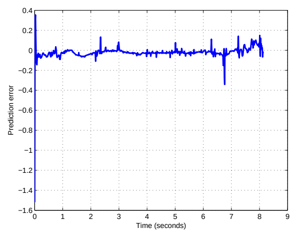


(b) $\dot{q}$

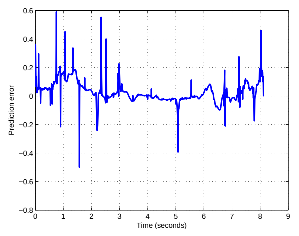


(c) $\dot{r}$

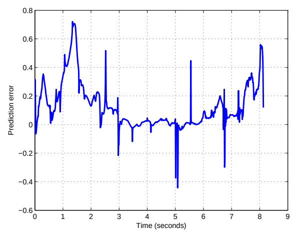
Figure 5.33: Model prediction error in angular accelerations ($\dot{p}\dot{q}\dot{r}$) during straight flight.



(a) $\dot{u}$

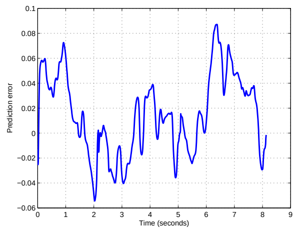


(b) $\dot{v}$

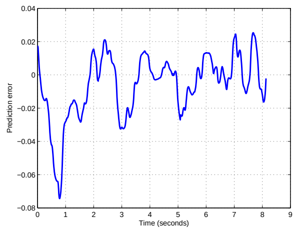


(c) $\dot{w}$

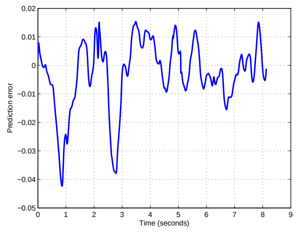
Figure 5.34: Model prediction error in linear accelerations ($\dot{u}\dot{v}\dot{w}$) during the hover maneuver.



(a) $\dot{p}$



(b) $\dot{q}$



(c) $\dot{r}$

Figure 5.35: Model prediction error in angular accelerations ($\dot{p}\dot{q}\dot{r}$) during the hover maneuver.

Chapter 6

Conclusion

This chapter provides the conclusion of the thesis. First the results provided in chapter 5 and the developed UAV platform are discussed, then suggestions for future work are given as well as the final conclusion.

6.1 Discussion

The focus of this thesis was on development of a fixed wing UAV research platform as well as the implementation of trajectory learning to infer the intended trajectory from a set of demonstration trajectories. In the following the results are discussed, beginning with the development of the platform.

6.1.1 Platform

A radio-controlled fixed wing UAV platform with a custom built data acquisition system was developed, as described in chapter 4. This platform was used to collect training data for the trajectory learning algorithm.

The choice of an airframe made of styrofoam as opposed to more durable materials makes the platform fragile, usually requiring repairs after every flight. However the repairs are fairly simple, and once the most highly stressed areas of the plane were reinforced, the platform proved surprisingly durable. The greatest benefit of a foam platform is the light weight. The hover maneuver requires a high thrust-to-weight ratio as mentioned in section 4.1.3, at the same time the plane should be flyable at low speeds in order to be suitable for indoor testing. The light weight of the styrofoam makes it possible to use a relatively small motor and still generate a sufficient amount of thrust.

The developed platform is able to collect all the sensor data needed to describe the state of the plane during demonstration flights, and record it on-board the plane.

6.1.2 Time alignment

The dynamic time warping works well for aligning the demonstration as shown in section 5.4.1. If we look at the state variables separately it may seem as though the algorithm sometimes misaligns features at certain points along the trajectory. Consider for instance the aligned demonstrations plotted in figure 5.31(a) on page 64 where the large peak in Aileron deflection intuitively seems misaligned with the other demonstrations. However, the algorithm finds the time indices that provides the best alignment considering *all* the state variables, and can not assign separate indices for each variable. In the case of the deviant aileron deflection peak, a corresponding misalignment is only found in the roll angle in figure 5.30(a), while the time alignment is good in the majority of the state variables.

6.1.3 Trajectory learning

The trajectory learning algorithm successfully inferred the intended trajectory in the case of both the demonstrated maneuvers. In this section the results of the learning are discussed, considering the level flight and the hover maneuver separately.

Level flight. The plot in figure 5.22 on page 59 shows that the learned trajectory in the case of the level flight is closer to a straight line than any of the demonstration trajectories. This is achieved without incorporating prior knowledge about the trajectory. As can be seen from the plot of the orientation in figure 5.24 the learned trajectory also maintains a more stable orientation than any of the demonstrations.

Features that are seen consistently in a majority of the demonstrations are interpreted as part of the maneuver. In the level flight maneuver this led to the algorithm learning the preparation for the left turn following the linear flight. This can be prevented either by considering a smaller part of the trajectory (i.e. only the part that is not affected by the following turn), performing the demonstrations in a larger space, or incorporating prior knowledge of the orientation.

Hover. Figure 5.27 on page 62 shows the pitch angle and the position of the plane during the hover maneuver. The intended trajectory is successfully learned by the trajectory learning algorithm. Intuitively the pitch angle should be as close to 90 degrees as possible for the duration of the hover, and the plane should maintain stable roll and yaw angles. The orientation plot in figure 5.30 shows the fluctuations in roll and yaw is a consistent pattern of the demonstrations, and thus they are learned by the trajectory learning algorithm.

While the human pilot needs these adjustments in oder to establish control of the plane, it could be possible for an autonomous controller to minimize the fluctuations. Prior knowledge in the form of constraints on the orientation during the hover could be incorporated to possibly achieve a more stable target trajectory for such a controller.

Model prediction error The model prediction error shown in section 5.4.4 is the prediction errors of the basic dynamic model along the trajectory. The prediction errors are used as bias terms to improve the calculation of the accelerations in the dynamic model along the trajectory. Without providing the bias terms the learning algorithm was unable to learn the intended trajectory.

Trajectory. As a controller is not yet implemented at this stage it is not possible to guarantee the flyability of the trajectory. However, analyzing the data shows that the trajectory is promising as it adheres to the dynamics of the plane, and is similar to the demonstrated maneuvers. Furthermore it succeeded in capturing the intended trajectory in both maneuvers without prior knowledge of the trajectory.

6.2 Future work

The implementation of the trajectory learning algorithm and the development of a suitable platform are the first steps towards automating highly aerobatic maneuvers with a fixed wing UAV through apprenticeship learning. The next step would be to develop an autonomous controller that can follow the learned target trajectory. Abbeel et al. use a combination of a finite and receding horizon linear quadratic regulator for this purpose, but other controllers may be suitable as well. An accurate flight simulator would be useful in the controller development to reduce the time and cost of re-building the plane after crashes.

Prior knowledge can be incorporated in order to learn target trajectories that are superior to the trajectories demonstrated by a human pilot. Such prior knowledge can for instance be in the form of constraints on the position or orientation of the plane.

For the case of autonomous flight the platform needs a few adjustments. These adjustments include receiving the control inputs from the ArduPilot board rather than the RC receiver and implementation of switching between manual RC control and autonomous control. More sensors may be added as well, such as sonars for obstacle detection.

As described in section 5.3.1 the switching of the logging provides a distraction to the RC pilot, as it is controlled from the radio transmitter. Our solution was to keep the logging on for the entire flight. Another solution could be to separate the log switching from the control of the plane, so that it can be controlled separately. Depending on the implementation this solution may require more hardware mounted to the plane to allow wireless communication other than the radio control.

The problem of an alternative solution to GPS position measurements for indoor flights where GPS is unavailable remain unsolved, and should be researched further.

6.3 Conclusion

This thesis presents the implementation of apprenticeship learning of trajectories for highly aerobatic maneuvers using an off the shelf radio-controlled fixed wing aircraft, and shows that the trajectory learning algorithm presented in [2] can be applied to fixed wing UAVs by providing a suitable dynamic model.

Secondly an experimental platform for research on apprenticeship learning for fixed wing UAVs was developed, including a custom designed data acquisition system that logs the state variables of the plane as well as the control inputs from the pilot during demonstration flights. The platform is easily extendable for use in fully autonomous flights.

These contributions are the first steps towards automating highly aerobatic maneuvers with a fixed wing aircraft through the means of apprenticeship learning.

Bibliography

[1] *MEDIATEK - 3329 Datasheet*, rev.a03 edition, 2003.

[2] P. Abbeel, A. Coates, and A.Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[3] C.G. Atkeson and S. Schaal. Robot learning from demonstration. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 12–20. MORGAN KAUFMANN PUBLISHERS, INC., 1997.

[4] Y. Bar-Shalom, X.R. Li, T. Kirubarajan, and J. Wiley. *Estimation with applications to tracking and navigation*. Wiley Online Library, 2001.

[5] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[6] J.A. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4:126, 1998.

[7] G. Bishop and G. Welch. An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, 8:27599–3175, 2001.

[8] Bosch Sensortec. *BMP085 Digital Pressure Sensor*, datasheet, rev. 1.0 edition, July 2008.

[9] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008.

[10] R. Cory and R. Tedrake. Experiments in fixed-wing uav perching. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2008.

[11] R.E. Cory. Supermaneuverable perching. 2010.

[12] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[13] A. Gelb. *Applied optimal estimation.* MIT press, 1979.

[14] W.E. Green and P.Y. Oh. Autonomous hovering of a fixed-wing micro air vehicle. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2164–2169. IEEE, 2006.

[15] W.E. Green and P.Y. Oh. A fixed-wing aircraft for hovering in caves, tunnels, and buildings. In *American Control Conference, 2006*, page 6. IEEE, 2006.

[16] W.E. Green and P.Y. Oh. A hybrid mav for ingress and egress of urban environments. *Robotics, IEEE Transactions on*, 25(2):253–263, 2009.

[17] Gary Bishop Greg Welch. The kalman filter, http://www.cs.unc.edu/ welch/kalman/.

[18] A. Hurst, A. Wickenheiser, and E. Garcia. Localization and perching maneuver tracking for a morphing uav. In *Position, Location and Navigation Symposium, 2008 IEEE/ION*, pages 1238–1245. IEEE, 2008.

[19] Adafruit Industries. C++ library for the bmp085 sensor, https://github.com/adafruit/adafruit-bmp085-library.

[20] E.B. Jackson. Manual for a workstation-based generic flight simulation program (larcsim) version 1.4. 1995.

[21] E.N. Johnson and S.K. Kannan. Adaptive trajectory control for autonomous helicopters. *Journal of Guidance Control and Dynamics*, 28(3):524–538, 2005.

[22] E.N. Johnson, M.A. Turbe, A.D. Wu, S.K. Kannan, and J.C. Neidhoefer. Flight test results of autonomous fixed-wing uav transitions to and from stationary hover. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference Exhibit, Monterey, CO*, 2006.

[23] R.E. Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[24] J.B. Kuipers. *Quaternions and rotation sequences.* Princeton university press Princeton, NJ, USA:, 1999.

[25] A. Lussier Desbiens and M.R. Cutkosky. Landing and perching on vertical surfaces with microspines for small unmanned air vehicles. *Journal of Intelligent and Robotic Systems*, 57(1):313–327, 2010.

[26] C.D. Manning, H. Schütze, and MITCogNet. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[27] AA Markov. An example of statistical analysis of the text of" evgenii onegin" illustrating the linking of events into a chain. *Izv. Imp. Akad. Nauk, Ser*, 6:153–162, 1913.

[28] P.S. Maybeck. *Stochastic models, estimation and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Pr, 1979.

[29] G.J. McLachlan and T. Krishnan. *The EM algorithm and extensions*, volume 274. Wiley New York, 1997.

[30] MicroStrain, Inc., 459 Hurricane Lane Williston, VT 05495 United States of America. *3DM-GX3® Data Communications Protocol*, 2010.

[31] M. Müller and Ltd MyiLibrary. *Information retrieval for music and motion*, volume 6. Springer Berlin, 2007.

[32] V. Myrand-Lapierre, A. Desbiens, E. Gagnon, F. Wong, and E. Poulin. Transitions between level flight and hovering for a fixed-wing mini aerial vehicle. In *American Control Conference (ACC), 2010*, pages 530–535. IEEE, 2010.

[33] I.J. Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1):90–100, 2003.

[34] A.R. Perry. The flightgear flight simulator. In *2004 USENIX Annual Technical Conference, Boston, MA*, 2004.

[35] Natural Point. Optitrack motion capture system, http://www.naturalpoint.com/optitrack.

[36] N.B. Priyantha. *The cricket indoor location system*. PhD thesis, Massachusetts Institute of Technology, 2005.

[37] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[38] Laminar Research. X-plane 10 desktop manual, http://www.x-plane.com/support/manuals/desktop/.

[39] JB Russell. *Performance and stability of aircraft*. Butterworth-Heinemann, 1996.

[40] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49, 1978.

[41] S. Sarkka, A. Vehtari, and J. Lampinen. Time series prediction by kalman smoother with cross-validated noise density. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 1653–1657. IEEE, 2004.

[42] H.W. Sorenson. Least-squares estimation: from gauss to kalman. *Spectrum, IEEE*, 7(7):63–68, 1970.

[43] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot modeling and control*. John Wiley & Sons New York, NY, USA:, 2006.

[44] RH Stone and G. Clarke. Optimization of transition manoeuvres for a tail-sitter unmanned air vehicle (uav). In *Australian Aerospace International Congress [online paper], http://www. aeromech. usyd. edu. au/uav/twing/pdfs/AIAC_paper_final. pdf, Paper*, volume 105, 2001.

[45] R. Tedrake, Z. Jackowski, R. Cory, J.W. Roberts, and W. Hoburg. Learning to fly like a bird. *Massachusetts Institute of Technology Computer Science and Artificial Intelligence Lab*, pages 1–7, 2009.

[46] R. Tedrake, T.W. Zhang, and H.S. Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2849–2854. IEEE, 2004.

[47] A. Tidemann and P. Öztürk. Learning dance movements by imitation: A multiple model approach. *KI 2008: Advances in Artificial Intelligence*, pages 380–388, 2008.

[48] Ubisense. Real time location system, http://www.ubisense.net/.

[49] CF Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.