Controller Design for UAV Tracking Specific Object

A Thesis Presented

by

Xiangyu Dong

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Communication, Control, and Signal Processing

Northeastern University
Boston, Massachusetts

July 2016
I dedicate this to my girlfriend and all the women in my family, for their kindness and selflessness; their endless support and devotion will always be remembered.
Contents

List of Figures vi
List of Tables viii
List of Acronyms ix
Acknowledgments x
Abstract of the Thesis xi

1 Introduction 1
  1.1 Category of Unmanned Aerial Vehicle (UAV) ................................. 1
  1.2 Use of UAV ......................................................... 3
  1.3 Computer vision – Tracking ........................................... 4
  1.4 What interests us? .................................................... 4

2 Device and Platform 5
  2.1 Device ................................................................. 5
    2.1.1 The UAV ....................................................... 5
    2.1.2 Computing Machine ......................................... 6
  2.2 Platform ............................................................. 6
    2.2.1 Robot Operating System (ROS) ............................... 6
    2.2.2 Tracker ......................................................... 7
    2.2.3 MATLAB ....................................................... 7
  2.3 Summary ............................................................ 8

3 Specification of this System 9
  3.1 AR.Drone Mechanism ................................................ 9
  3.2 On-board Cameras .................................................. 10
  3.3 Tracker and Captured Frames ..................................... 11
  3.4 Sampling time and delay .......................................... 12
  3.5 Sensor data in use ............................................... 15
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Fixed-wings UAV</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Helicopter UAV</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Multi-rotor UAV</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Data flow diagram</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Schematics of the Parrot AR.Drone (Body coordinate frame)</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Original picture captured by the drone’s front camera</td>
<td>11</td>
</tr>
<tr>
<td>3.3</td>
<td>Drone’s output frame after enabling tracker</td>
<td>12</td>
</tr>
<tr>
<td>3.4</td>
<td>Target lost caused by moving the object outside the camera range</td>
<td>13</td>
</tr>
<tr>
<td>3.5</td>
<td>Target lost caused by rotating the object at some degree to the camera</td>
<td>13</td>
</tr>
<tr>
<td>3.6</td>
<td>Runtime distribution of 1000 loops</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>Coordinate system from captured frame</td>
<td>17</td>
</tr>
<tr>
<td>4.2</td>
<td>Description in coordinate system if object gets closer</td>
<td>18</td>
</tr>
<tr>
<td>4.3</td>
<td>Perfect tracking description in coordinate system</td>
<td>18</td>
</tr>
<tr>
<td>5.1</td>
<td>One test data for the area signal</td>
<td>21</td>
</tr>
<tr>
<td>5.2</td>
<td>Closed-loop system of horizontal movement with proportional controller</td>
<td>22</td>
</tr>
<tr>
<td>5.3</td>
<td>System response – Horizontal movement by P controllers with different parameter</td>
<td>22</td>
</tr>
<tr>
<td>5.4</td>
<td>Closed-loop system of heading movement with proportional controller</td>
<td>24</td>
</tr>
<tr>
<td>5.5</td>
<td>System response – Self-rotation movement by P controllers with different parameter</td>
<td>24</td>
</tr>
<tr>
<td>5.6</td>
<td>Closed-loop system of vertical movement with proportional controller</td>
<td>25</td>
</tr>
<tr>
<td>5.7</td>
<td>System response – Lifting movement by P controllers with different parameter</td>
<td>25</td>
</tr>
<tr>
<td>5.8</td>
<td>System identification – Horizontal movement input and output data</td>
<td>28</td>
</tr>
<tr>
<td>6.1</td>
<td>Displacement controller – Altitude</td>
<td>31</td>
</tr>
<tr>
<td>6.2</td>
<td>Displacement controller – Heading</td>
<td>32</td>
</tr>
<tr>
<td>6.3</td>
<td>Displacement controller – Left and right / Back and forth</td>
<td>33</td>
</tr>
<tr>
<td>6.4</td>
<td>Horizontal displacement controller with rotation matrix implementation</td>
<td>34</td>
</tr>
<tr>
<td>6.5</td>
<td>Head turning tracking block diagram</td>
<td>35</td>
</tr>
<tr>
<td>6.6</td>
<td>Horizontal moving tracking block diagram</td>
<td>36</td>
</tr>
<tr>
<td>6.7</td>
<td>System response – Horizontal movement by displacement P controllers</td>
<td>37</td>
</tr>
<tr>
<td>6.8</td>
<td>System response – Self-rotation movement by displacement P controllers</td>
<td>37</td>
</tr>
</tbody>
</table>
6.9 System response – lifting movement by displacement P controllers

7.1 Top view explanation of the tracking method we use
7.2 Heading identification – input and output data
7.3 Three heading outputs from experiment, modified model and original model
7.4 Altitude identification – input and output data
7.5 Three altitude outputs from experiment, modified model and original model
7.6 Data density – heading direction to frame location
7.7 Data density – altitude direction to frame location
7.8 Tracking system model – heading tracking
7.9 Tracking system model – altitude tracking
List of Tables

A.1 ROS useful commands ................................................. 52
List of Acronyms

**UAV** Unmanned Aerial Vehicle. An aircraft without a human pilot aboard. The flight of UAVs may operate with various degrees of autonomy: either under remote control by a human operator, or fully or intermittently autonomously, by onboard computers.

**ROS** Robot Operating System. An operating system which provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

**TLD** Tracking-Learning-Detection. TLD is an award-winning, real-time algorithm for tracking of unknown objects in video streams. The object of interest is defined by a bounding box in a single frame. TLD simultaneously tracks the object, learns its appearance and detects it whenever it appears in the video. The result is a real-time tracking that often improves over time.

**LTI** Linear time-invariant.
Acknowledgments

I would like to express the deepest gratitude to my advisor Professor Mario Sznaier for his many valuable guidance throughout the past two years during my master program. Taking his control courses is one of the greatest experience in my life and doing research in his laboratory is another. Without his cultivation and support, this thesis would never have been possible.

Furthermore, I would like to acknowledge with much appreciation my committee member Professor Felipe Pait who revised this thesis and gave me important suggestion and feedback. Without him, this thesis would not be this much perfect.

I would also like to thank Xikang Zhang, who provided great insights and ideas for advancing this research and encouraged me to march forward; Jose Lopez, who gave me significant assistance on the foundation construction of this research; Yongfang Cheng, who promptly brought me the theoretical solutions of some critical problems of the research.

In addition, a thank you for all my lab-mates in the Northeastern Robust System Laboratory. It is these people who created and kept a well-established academic atmosphere together. It is these people who make people here to become one family.
Abstract of the Thesis

Controller Design for UAV Tracking Specific Object

by

Xiangyu Dong

Master of Science in Electrical and Computer Engineering

Northeastern University, July 2016

Dr. Mario Sznaier, Advisor

The prosperity of [UAV] brings a new lifestyle to aviation enthusiasts, since all kinds of new features have been devised and added to [UAV]. One significant improvement is putting HD camera on-board to enable aerial photography. In recent years, computer vision also becomes a fast growing academic discipline. Object detection and object tracking are hot topics in this field. This thesis developed and tested the idea of [UAV] controller design combined with object tracking so that the [UAV] is able to follow one specific object automatically. In this application, [ROS] has been used as a communication platform between the [UAV] and the computer, and we implement the Tracking-Learning-Detection (TLD) tracker into this platform. System identification has been performed to find the [UAV] motion models according to its movement in different directions. Based on these models and the feedback signals from the TLD tracker, different feedback loops are established. Various velocity controllers for the inner loop and displacement controllers for the outer loop have been designed separately with respect to corresponding directions. After comparing the results from experiment, we choose the best controller which allows the [UAV] to follow the object with a fast and accurate tracking response.
Chapter 1

Introduction

UAV has been a hot topic for nearly two decades. At the beginning, UAV is only designed for military use and it is a model-plane remotely controlled by those enthusiastic amateur who are interested in airplane. But nowadays it is becoming more and more popular in our everyday life. The prospect of UAV market is promising since many of the high-tech companies want to combine some of their products to the UAV so that they can make even better products.

1.1 Category of UAV

There are three types of UAV in the mainstream: fixed wings, helicopter and multi-rotor.

Fixed wings just as its name implies, the wings are fixed as shown in Figure 1.1. The lift force is created by air flow going through the wings. The dynamic system of them is similar to the Boeing-737 and Airbus-320. The advantages of this kind of UAV is they have longest time of endurance and highest load factor among the three. It is better for long range monitoring. The disadvantages are they need long taxi-way to take off and land.

Helicopter usually have two rotors. A typical one is shown in Figure 1.2. A large one on the top, which provides the force to make the helicopter move and a small one on the tail, which keep the helicopter stable and prevent spinning caused by the large rotor. The good thing for helicopter is it can perform vertical take-off and landing while the bad thing is it has complex mechanical structure so that the maintenance cost and operating expense are relatively high.

Multi-rotor refers to the type of aerial vehicles with multiple rotors in the same horizontal plane. They can have three, four, six or eight rotors as shown in Figure 1.3. Those with four rotors are also called ‘quadrotor’, the common ones that have very simple mechanism. Four motors connecting
CHAPTER 1. INTRODUCTION

Figure 1.1: Fixed-wings UAV

Figure 1.2: Helicopter UAV
CHAPTER 1. INTRODUCTION

Figure 1.3: Multi-rotor UAV

to four rotors separately with simple control to all four motors can make it work. Pros for them also include vertical take-off and landing. Cons are these aircraft consume more power to fly, so they has shorter flight duration time. This also leads to less load they can carry.

The quadrotor has so many names. Drone is one of them, since the rotors make buzz noise while flying like male bee. The other is quadcopter, combined word from ‘quadrotor’ and ‘helicopter’. We may use different names in the following chapters but they all refer to the same thing.

1.2 Use of UAV

As we all know, UAV has many military use. The fixed wings is more widely used for unmanned surveillance aircraft while the multi-copter usually works as individual combat weapon.

What’s more interesting is, UAV especially the quadrotor, can take aerial photography. Many of the reality show has been videoed with them and grand landscape scenes in most of the blockbusters from HBO and Hollywood are also captured by the camera on-board; Since the prosperous market of online shopping, the delivery service becomes much more important these days. Thus, the new idea of delivery drone becomes a hot topic. DHL and FedEx are now doing research for delivering of packages with drones. Amazon is testing their future delivery system – Prime Air and their slogan is “30 Minute Delivery”.

3
CHAPTER 1. INTRODUCTION

[UAV] is also used for monitoring in wild animal protection area and for supervision of various plants in greenhouse.

1.3 Computer vision – Tracking

Computer vision is a thriving topic in computer engineering these years. One useful tool focuses on how to detect a target in a set of pictures (a video stream) and lock on it. This target should have some properties that others do not have, thus the algorithm could figure out this target with a confident level.

Once the target is reliably locked on, it is possible to track this target.

1.4 What interests us?

If we can combine the quadrotor with tracking, it will follow certain object while flying. Therefore, an autonomous flying camera could be created. Whether having picnic with family outside or climbing a cliffs in the mountain, this flying camera would capture those important moments automatically.

To make it real, the first thing is implementing the quadrotor using computers to control it based on the on-board camera so that it can track specific object.
Chapter 2

Device and Platform

The main purpose of this chapter is to describe what hardware and software we use. And how we combine them to work as we desired.

2.1 Device

2.1.1 The UA V

In this project, we choose the Parrot AR.Drone 2.0 to be the controlled UAV. We have done plenty of research for which UAV we should use at the very beginning of this project. Some competitive candidates include DJI Phantom 3, DJI Phantom 4 and Parrot AR.Drone 1.0.

Both DJI Phantom 3 and Phantom 4 have very good stability and high resolution on-board camera. The Phantom 4 can capture 4K video or 12MP pictures while the Phantom 3 can capture 2.7K video. There is a video showing Phantom 3 flying above the cloud. It can be found in [1]. However, these two quadrocopters are designed for outdoor recreation, so they do not have indoor hulls. As we are programming and flying the drone in our lab, which is a enclosure space, it is better to have some preventive action for the drone. We realize it is often the case that programming mistakes could result in drone crashing or their rotors hitting something and breaking unexpectedly. We also check the prices which is $499 for Phantom 3 and $1399 for Phantom 4. If it crashes and requires regularly replacing parts or sometimes buying a new one, it will become less cost-effective.

We see the AR.Drone 1.0 is mature product with on-board sensors such as three-axis gyro, pressure sensor, ultrasound sensor for ground altitude measurement and so on. These sensors can
generate enough feedback signals for controller design. But this product is launched in the year 2010. Now it is not easy to get a brand new one and replacing accessories are difficult to find.

Therefore, we take AR.Drone 2.0 into consideration. It inherits most features from AR.Drone 1.0 and has some improvements. Furthermore, it has indoor hull and the price for it is $249. All these feature makes it the best candidate.

2.1.2 Computing Machine

In order to control the UAV via a computer, we need a workstation or a high performance laptop to ensure reliable Wi-Fi connection and sufficient computation rate. All the data and experiments we have below are generated by a workstation in the Northeastern Robust Control Lab. It has Intel Xeon E5540 CPU, 24GB DDR3 memory and a Wi-Fi card supporting IEEE 802.11n with maximum net data rate of 300 Mbits/s.

In the final stage, we try to check the portability of this system. Therefore we transplant all the work from the workstation to a powerful laptop with Intel Core i7-6700HQ CPU and 16GB DDR4 memory. The result is promising and we get almost the same performance as that using the workstation.

However, when we try to test it via a laptop with even lower hardware configuration, especially those with a ultra low voltage CPU such as Intel Core i7-4500U, we don’t get the performance as we expect. The lag from computation has a strong negative impact on the controller so that the UAV system is no longer stable.

2.2 Platform

2.2.1 ROS

This project uses the ROS for the base platform so that each component has the ability to communicate with one another. We use the ‘indigo’ version of ROS and we use a Ubuntu 14.04 LTS operating system in order to provide fully support to it. For more information about ROS please check out [2].

We use ardrone_autonomy as it provides easy setup ROS driver for Parrot AR.Drone 2.0. With this, we can obtain real time sensors data from the drone and send commands to it. This driver act as an independent package in the ROS workspace. For information about ardrone_autonomy, see [3].
CHAPTER 2. DEVICE AND PLATFORM

2.2.2 Tracker

This project uses the TLD algorithm as the tracker. It is one of the most important parts to generate the outer loop feedback signals enabling the drone to follow object and an advantage of this tracker is its learning capability which means it does not need any prior information of the tracked object. It uses a bounding box around the object while tracking. The output of the tracker is the bounding box location in each frame with its height and width captured by the drone’s on-board camera.

During our test, we find its real-time performance is relatively acceptable comparing to other tracking algorithms but there is room for improvement. The TLD exists in the ROS workspace in the form of an independent package. Instead of changing the code inside the package, we improve its outputs in MATLAB by adding a linear prediction with constant velocity, in case of short time target losing caused by the tracker.

For more details about TLD algorithm, check out [4].

2.2.3 MATLAB

We use MATLAB as the computation terminal which can get all data from different ROS packages, filter these data, design the control law and the controller, generate the control action from the controller and send the control action signals back to the ROS packages. In this thesis, most of our works are achieved by using MATLAB.

MATLAB has new toolbox named ‘Robotic System Toolbox’ which was introduced to MATLAB in the version of 2015a. It is this toolbox that provide fully support for ROS and enable easy connection and communication between ROS and MATLAB.

MATLAB also support Object-Oriented Programming. In this project, we generate different files for storing different classes destination. The main script instantiate these class first and call the method inside it to get data and do computation.

These classes include:

- ar_drone.m – communicating with the ardrone_autonomy.
- tld_tracker.m – communicating with the tracker
- XYcontrol.m – generate control action signals.
- XYcollect.m – collecting history data for each flying experiment.
CHAPTER 2. DEVICE AND PLATFORM

Note here, we introduce a data collecting class in order to plot and analyze the collected data easily. By using this method of coding, the readability of our codes has been significantly improved.

2.3 Summary

At the end of this chapter, we use the Figure 2.1 to depict what components have been used and the data flow between each component so that readers can get a clear understanding of our system.

Figure 2.1: Data flow diagram
Chapter 3

Specification of this System

In this chapter, we give a brief explanation about the feasibility of this project and mainly talk about the two different tracking ideas about it.

3.1 AR.Drone Mechanism

The drone movements are similar to those of a conventional helicopter. But the movement of drone is achieved by changing and adjusting the rotation rate of four propellers that causes movement. Figure 3.1 shows the drone movement schematics and details are explained below:

- **Pitch** – Rotational movement along transversal axis $y$ resulting in a transnational movement on $x$ axis. It can be achieved by increasing the speed of engines $M_1$ and $M_2$, while decreasing the speed of engines $M_3$ and $M_4$ (or vice-versa) to generating movement on $x$ axis positive (or negative) direction.

- **Roll** – Rotational movement along longitudinal axis $x$ resulting in a transnational movement on $y$ axis. It can be achieved by increasing the speed of engines $M_2$ and $M_3$, while decreasing the speed of engines $M_1$ and $M_4$ (or vice-versa) to generating movement on $y$ axis positive (or negative) direction.

- **Yaw** – Rotational movement along $z$ axis. It can be achieved by increasing the speed of engines $M_2$ and $M_4$, while decreasing the speed of engines $M_1$ and $M_3$ (or vice-versa) resulting in an overall clockwise (or counter-clockwise) torque, without changing overall upwards thrust or balance.
CHAPTER 3. SPECIFICATION OF THIS SYSTEM

- Throttle – Translational movement on $z$ axis. The drone ascends or descends. It can be achieved by equally increasing or decreasing the speed of all four engines.

![Schematics of the Parrot AR.Drone (Body coordinate frame)](image)

Figure 3.1: Schematics of the Parrot AR.Drone (Body coordinate frame)

The drone accepts above four commands as inputs while flying. Thus, the drone can travel to any certain point in space. Through this, it has the ability to track a moving object.

Note that we are using the original indoor foam hull to avoid unnecessary damage while doing experiment. According to the manual, in this case, the drone weights 420g in total. Therefore all our experiments, data analysis and system design are based this weight.

3.2 On-board Cameras

The AR.Drone has two on-board cameras, one pointing forward (called front camera) and one pointing downward (called bottom camera).

The front camera is our main tool to capture live video stream for tracker to locate the object. Although this camera support 720p video stream, we choose the resolution of $640 \times 360$. Thus the tracker can process less data in order to accelerate the computation speed for this real time
system without losing too much useful information from the video stream. A snapshot of the video stream is shown on Figure 3.2.

Figure 3.2: Original picture captured by the drone’s front camera

The bottom camera is designed for the purpose of drone stabilization. Unlike AR.Drone 1.0, the second version does not provide access to the bottom video stream. Thus its only job is to recognize patterns on the floor in order to guide the internal stabilizer to control the drone while flying. The drone has its own algorithm to use the bottom camera frame signal, but we do not know what exactly happen there. In this case, we will ignore the effect to the whole system generated by this input. We will also treat it as Linear time-invariant (LTI) system.

3.3 Tracker and Captured Frames

There is a object made of cardboard with conspicuous patterns in the Figure 3.2. We assume this is the target we want to track. To get the position of it, the bounding box in tracker has been created. After that, tracker returns real time varying bounding box shown in Figure 3.3 [TLD] algorithm use probability method to locate the object after it has been selected.

If the color of bounding box is blue, it means the tracker has much more confidence that this is the target being selected. If the color is yellow, it means the tracker is not very confident of it.
Target lost, if the bounding box is missing, it means either the object is outside of frames shown in Figure 3.4 or the object is misinterpreted by the tracker and causes object missing shown in Figure 3.5. Note here, the TLD performance improves over the time while recognizing the selected object.

3.4 Sampling time and delay

ROS communicates with the drone at the rate of 15Hz by configuring the Drone Update Frequency on ardrone_autonomy. In MATLAB, we run the computation part in loops. Within a single loop, a set of commands is executed in a sequence. They are: Update data from the drone; Compute the control action using these data; Send the control action back to the drone.

Since this is a real-time system, we assume the time for flying the drone is the same as it for executing the program in MATLAB. Thus, we use ‘tic’ and ‘toc’ command in MATLAB to get the loop time and use this as the estimation of the system’s sampling time.

During one experiment, the program loop time is varying from 0.01 to 0.25 second as shown in Figure 3.6 with the average of 0.035. Multiple experiments show that the average loop times are from 0.03 to 0.07 second.

Therefore, we cannot guarantee fixed sampling time of this system. We treat it as contin-

Figure 3.3: Drone’s output frame after enabling tracker
CHAPTER 3. SPECIFICATION OF THIS SYSTEM

Frame: 2910, Posterior 0.00, FPS: 86.77,

Figure 3.4: Target lost caused by moving the object outside the camera range

Frame: 800, Posterior 0.00, FPS: 99.75,

Figure 3.5: Target lost caused by rotating the object at some degree to the camera
CHAPTER 3. SPECIFICATION OF THIS SYSTEM

uous system. We assume the 0.05 second sampling time for those cases requiring fixed sampling rate, for example, we use 0.05 second to interpolate the data in order to meet the data requirement in MATLAB Identification Toolbox.

Integrator of one signal is obtained by the summation of each value of this signal multiplying the loop time separately while differentiator is achieved by the difference of two adjacent values of the signal dividing by the loop time.

We use 0.26 second as the time delay in this system. In [5], the time delay via Wi-Fi connection between drone and computer has been set to 0.26 second in its AR.Drone Simulink Model. As we lack precise methods to measure the time delay parameter, we decide to use 0.26 for this project.

Figure 3.6: Runtime distribution of 1000 loops
CHAPTER 3. SPECIFICATION OF THIS SYSTEM

3.5 Sensor data in use

Except for data from the on-board camera, we also use data from the on-board sensors for our system. They include:

- Euler angles – The three-axis gyro gives us the roll, pitch and yaw angles with the precision of 6 degree. During the experiments, we find the roll and pitch angles are more precise than the yaw angle.
- Altitude – The ultrasound sensor measures the distance from the ground. Its accuracy is 1mm.
- Linear velocities – AR.Drone uses its own on-board sensors and its bottom camera to give estimations of its velocities in three directions with accuracy of 1mm/s.

For full specifications of AR.Drone sensor data, see Reading from AR-Drone section in [3].
Chapter 4

Tracking Strategy

For tracking an object, it means the drone can follow it and move if the object moves. More precisely speaking, from the drone’s perspective, we want the object to always stay in the center of drone’s front camera.

4.1 Distance error signal

We can see in Figure 3.3, the object is not in the center of the camera. Therefore, it is reasonable to generate error signals indicating the distance from the object center to the camera center.

We can see the frame captured by the camera as two dimensional data. It is appropriate to transfer Figure 3.3 to the coordinate system and retrieve the data. After the transformation, we get Figure 4.1. The tracked object is given by the bounding box. We define the center of it as object center with its location \((a, b)\) in the frame. And we also define the camera center as \((320, 180)\). Note here, the resolution of each frame from the drone is 640 x 360. Thus, the pixel point \((320, 180)\) is the camera center.

The error signal is defined by camera center coordinate minus object center coordinate and then normalize them within the rage of \(-0.5\) to \(0.5\). The horizontal and vertical error signals are defined respectively in (4.1) and (4.2).

\[
x_{\text{error}} = (320 - a)/640 \\
y_{\text{error}} = (180 - b)/360
\]  

\(\text{(4.1)}\) 

\(\text{(4.2)}\)
CHAPTER 4. TRACKING STRATEGY

Figure 4.1: Coordinate system from captured frame

Note here, frames are two-dimensional data. It is easy to manipulate them by transferring the two-dimensional data to two one-dimensional data. In the same sense, the error is a vector at first denoting by the orange solid line in Figure 4.1 and we get two vectors orthogonal to each other through vector decomposition denoting by the orange dotted line in Figure 4.1.

4.2 Area error signal

The idea ‘objects are becoming smaller as their distance from the observer increases’ can be introduced for measuring the relative depth in the frame. The bounding box has width and length. And these two variables are changing based on the object outline. Figure 4.2 depicts how bounding box change if the object is closer to the observer. The dotted box reveals the original area of the object while the solid box implies the current area of it.

Thus, area of the tracked object can be achieved by multiplying the width and length of the bounding box. Furthermore, TLD can give us an initial area data of the object after system initialization. We can compute the ratio of these two areas to see if the object is getting closer or farther to the observer.

Therefore, the area error signal $S_{error}$ is defined by (4.3). Note here, $S_{error}$ is neither a
CHAPTER 4. TRACKING STRATEGY

Figure 4.2: Description in coordinate system if object gets closer

Figure 4.3: Perfect tracking description in coordinate system
CHAPTER 4. TRACKING STRATEGY

normalized signal nor a linear signal.

\[ S_{\text{error}} = \frac{(S_{\text{initial}} - S_{\text{current}})}{S_{\text{initial}}} \]  \hspace{1cm} (4.3)

4.3 Tracking idea using error signals

Object-tracking is achieved by using these signals as feedback and design controllers to enable the drone to move in desired direction so that these errors decreased. Thus, the drone can follow the object. \( x_{\text{error}} \) illustrates the horizontal errors that leads to the drone’s horizontal movement while \( y_{\text{error}} \) expresses the vertical errors that leads to the drone’s vertical movement.

Object center (a,b) goes to camera center (320,180) indicating perfect tracking. The desired result can be described in Figure 4.3. At this point, both \( x_{\text{error}} \) and \( y_{\text{error}} \) equal to 0. \( S_{\text{error}} \) can be used to keep the distance between the drone and the object. If \( S_{\text{error}} = 0 \), it means the drone is keeping the distance from the object as it originally is.
Chapter 5

Simple Controller Design

The ultimate goal for a control system is to minimize the error signals depicting in Chapter 4. In control theory, error signal refer to the input of the controller. There are two properties to determine if the control system is good or not. One is stability meaning in a finite time, the error signals converge to some small value (usually 0). The other is performance meaning the less time the system needs for the error signals to converge, the better performance it has.

5.1 The problem of area error signal

The area error signal has been defined in Section 4.2 and we original want to exploit it for maintaining the space distance between the drone and the object. But in fact, we find the area signal which is used to generate the area error signal is not good enough during different tests.

In one test, we put an object which has a certain space distance from the drone for 12 seconds and then we move it close to the drone with another certain space distance for 16 seconds. The actual area signal is recorded in Figure 5.1 in red while the signal we are expecting is in blue.

As we can see, when the object is far from the drone, this signal is acceptable. However, if the object is close enough to the drone, the signal becomes very noisy. We notice in the computer screen that the length and width of the bounding box always have sudden changes which leads to unusual jumps of the area signal. We have tried many ways to filter the signals. But the results are not promising. It either introduces too much delay or lacks the performance. Since this thesis is concentrated on controller design, this challenging problem is left here and still open.

At this point, it is wise to give up translational movement on $x$ axis generated by pitch angle. Thus, we begin to send zero pitch command to the drone to make it stay in this direction and
prevent it from moving back and forth.

5.2 Proportional controllers

Section 3.1 describe the drone mechanism and how to achieve different movement. Notations from there have been used through this chapter. This is the beginning part of controller design, so we do not have any system model. We start our design by tune parameters during experiments and we close the loop by using a proportional (or in other word, a pure gain) controller for movement in different directions. During each experiments, we set the space distance from the object to the drone to be 3 meters.

5.2.1 Translational movement on $y$ axis

The $x_{error}$ in each frame described in Section 4.1 can be reduced by doing translational movement on $y$ axis. For simple explanation, we say horizontal translational movement. Note here, we send zero command to make the drone keep its heading direction towards the object. Figure 5.2 depicts the closed loop system of this movement. In this block diagram, we do not take the real
CHAPTER 5. SIMPLE CONTROLLER DESIGN

Figure 5.2: Closed-loop system of horizontal movement with proportional controller

Figure 5.3: System response – Horizontal movement by P controllers with different parameter.
world position of the drone into consideration. We only care about if the object center is close to the camera center.

We implement this algorithm to the system. Test it for 20 seconds and record the data. We have tune the parameter during numerous experiments and put three set of data here which has a certain representativeness among all. Figure 5.3 shows these three sets of data with different proportional parameters 0.2, 0.3 and 0.4. Note in the figure, there are some discontinuous jumps. They indicate short-time object lost and if this happens, zero command will be sent to the drone.

When $P$ is 0.4, the system response is oscillating implying unstable system. When $P$ equals 0.2 and 0.3, the trajectories are slowly converging. 0.3 for $P$ has better performance than 0.2. However, those performance are still deficient.

5.2.2 Rotational movement along $z$ axis

The $x_{error}$ can also be reduced by doing rotational movement along $z$ axis. We called it drone self-rotation movement. In this case, the drone is rotating towards the object. Note here, we send zero roll command to make the drone keep its horizontal position. Figure 5.4 shows the closed loop system of the yaw movement. In the same as previous part, we do not take the real world position of the drone into consideration.

With the same experiment setting as that in Section 5.2.1 we select three sets of data displaying in Figure 5.5. It shows these three sets of data with different proportional parameters 0.5, 1.0 and 1.5. 1.5 for $P$ has the best performance among all three.

However, we notice that even if the system is stable, as time goes on, the signal often leaves the envelope-curve area and goes back again instead of staying inside the envelope-curve area forever. Since we only operate this drone with inputs in certain range, we treat it as an LTI system. Besides we have not found the source of any disturbance, so we believe this behavior is not common if the system is stable. One potential reason could be the delay of this system.

5.2.3 Translational movement on $z$ axis

The $y_{error}$ can only be reduced by doing translational movement on $z$ axis. We called it vertical movement. In this case, the drone is doing lifting movement to follow the object. Figure 5.6 shows the closed loop system of the lifting movement. We do not take the real world position of the drone into consideration either.
CHAPTER 5. SIMPLE CONTROLLER DESIGN

Figure 5.4: Closed-loop system of heading movement with proportional controller

Figure 5.5: System response – Self-rotation movement by P controllers with different parameter.
CHAPTER 5. SIMPLE CONTROLLER DESIGN

Figure 5.6: Closed-loop system of vertical movement with proportional controller

Figure 5.7: System response – Lifting movement by P controllers with different parameter.
CHAPTER 5. SIMPLE CONTROLLER DESIGN

This time we still use the same experiment setting as before, and we also select three sets of data portrayed in Figure 5.7. It shows these three sets of data with different proportional parameters 0.5, 1.0 and 1.5 while 1.5 for P has the best performance.

However, the unusual behavior described in Section 5.2.2 also happens here. The signal goes outside of the envelope-curve area unexpectedly. Same as the previous experiment, no source of noise have been detected.

5.2.4 Summary

The experiments above show that using this approach to control the drone is feasible. The whole system works as we expect but the performance is not good enough. After tuning the parameters for some time, we begin to think by only using a P controller, the performance can no longer be improved. To improve the performance, an alternative way to do this is that we can first identify system models and use the knowledge in control theory to design better controllers for the drone.

5.3 First attempt – System identification

At this point, we start to seeking the system models. We plan to do this, by collecting the flying data and using system identification toolbox to obtain the mathematical model.

5.3.1 Identification procedure

First, we try do identify the horizontal translational movement model in Figure 5.2. Distance from the drone to the object is set to be 3 meters and starting position in the camera is set to be around 265 pixel. An varying roll command needs to be created and sent to the drone as the identification input signal. And we catch the horizontal pixel position of the target from each frame as the identification output signal.

An input roll command signal has been generated. This signal has multiple separated positive and negative impulses with amplitude of 0.2 for one second while zeros have been put into the gap of adjacent two impulses as illustrates in Figure 5.8 upper part. This signal makes the drone move leftward for a short-time and stay, then move rightward for a short-time and stay and repeat it.

Note here, we use several impulses as input for a number of reasons. First, it can be seen from experiment that drone movement is less sensitive to input signals when it goes below 0.1 but
CHAPTER 5. SIMPLE CONTROLLER DESIGN

more sensitive when it goes beyond 0.7 which means the input to the drone and its output movement do not have a strong proportional relation. For example, the movement caused by input of 0.05 for 1 second is much less than one fifth of the movement caused by input of 0.25 for 1 second while the input of 0.8 for 1 second gives much more displacement than twice the input of 0.4 for 1 second. Second, a large input could potentially makes the object go outside of the drone camera’s visual field, and so as a input with relatively long action time. If the object is outside the camera frame, the signal becomes useless. Therefore, instead of sinusoid or square signals, impulses have been selected as the system identification input signal. It will be an informative signal if the amplitude is inside the range of 0.1 to 0.7.

5.3.2 Result analysis

We do each experiment for more than one minutes to get as much information of the system as we can. Two experiment data have been expressed in the lower part of Figure 5.8.

We send all zeros to produce zero movement in order direction including yaw, pitch and lift while doing these experiments. Since we try to get the horizontal movement model, we do not want the results to be affected by other dimensional movement.

However, as observed experimentally, sending zero command to the drone does not necessarily mean it can keep its position at these dimensions. Coupling effects occur while sending non-zero roll command for the purpose of identification while put others to zero. And as the time of these experiments goes longer, the drone gets farther to the original starting point in these directions.

Take one experiment result for example, the starting space distance from drone and object is 3 meters by our setting. But after 60 seconds experiment, the space distance is about 6 meters which means the drone has a backward movement, even if sending zero pitch command to it during the whole experiment. In some other experiments, the drone slightly changing its heading direction has been detected with zero yaw command sent.

Note here, the results illustrated in Figure 5.8 is the data got from two experiments. Take a close look at these two results, and it can be seen that with the same input signal, the system has different response especially the amplitude difference in the time 40 to 50 second and the response follows. Therefore, the response is unpredictable at some degree.

The problems described above have significant negative effects on the process of system identification. We also try this method for finding models of movement in other directions. Nonetheless, these problems still exist. Thus, first, we believe doing system identification from drone
Figure 5.8: System identification – Horizontal movement input and output data
CHAPTER 5. SIMPLE CONTROLLER DESIGN

input directly to the camera outputs is challenging and second, we need to get rid of the coupling effects which means doing system identification in one direction while keeping its location in other directions.
Chapter 6

Improved Controller Design

In the development-kit of [5], their Simulink model was designed for way-point tracking. Way-point tracking focuses on the distance from the starting point. Instead of velocity, displacement control has been used. And the input of the system is the desired positions not the velocities. Inspired by this, we introduce displacement controller to our system.

6.1 Reason for displacement control

At first, we think the drone’s inputs consisting of roll, pitch, yaw and lift signals are all velocities, because as soon as we send zero signals to the drone, it stops to move on corresponding direction.

And during the experiment illustrated in Section [5.3] we find out that same input to the same system does not generate the same output. This maybe causes by some dynamics of the drone, but we have no clue. Right now, we can not compensate the output. The only thing we can do is to treat it as noise, but this noise is large enough to influence the whole system and to directly cancel the noise is hard to achieve. However, if we change the system input from velocities with noise to displacement by adding an inner loop feedback, this problem can be solved.

Displacement control can also solve the random movement problem demonstrated in Section [5.3.2]. By sending zero velocity to the drone in one direction, it will have random movement caused by coupling condition if the drone is moving in other directions. But if we have displacement control, random movement will be compensated by the inner feedback loop. In order to make the drone stay in one direction, at this time we think it is much more reliable to send zero displacement rather than zero velocity signals.
Nonetheless, there is a disadvantage for introducing the displacement control. Note here this displacement is the real world position. But our ultimate goal for tracking is to minimize the distance from the camera center to the object center. This distance is inside the frame and it is affected by the space distance between the drone and the object. It is difficult to match the real-world displacement in meter with frame displacement in pixel exactly.

### 6.2 Feasibility for displacement control

In this section, displacement controllers have been developed in the order of altitude, heading, left and right movement, back and forth movement. Sensor data explained in Section 3.5 have been used for them.

#### 6.2.1 Displacement controller for altitude

The drone can detect the altitude from the ground. Therefore, we compare the altitude reference input to the altitude feedback using their difference as the error. Then the error goes through the controller and works as velocity input to manipulate the drone flying to the desired altitude. Figure 6.1 is the block diagram of the Altitude displacement controller.

![Block diagram of Altitude displacement controller](image)

Figure 6.1: Displacement controller – Altitude

Here, we use a pure gain as the controller. After tuning, $P_{\text{altitude}} = 1.1$ works best.

#### 6.2.2 Displacement controller for heading

The drone can collect the heading data in degree. Firstly, the initial heading direction in degree has been collected. The heading reference is the difference between the initial heading and
the desired heading direction. In other words, the heading reference indicates how much we want the drone to turn. Therefore, the error is generated and goes through the controller and works as velocity input to manipulate the drone turning to the desired direction. Figure 6.2 is the block diagram of the Heading displacement controller.

Figure 6.2: Displacement controller – Heading

Note here, we also use a pure gain as the controller. After tuning, $P_{\text{heading}} = 1.5$ gives the best performance. And there is a constrain for the heading data from the drone, that its value is varying $-179^\circ$ to $180^\circ$.

6.2.3 Displacement controller for left and right movement

The drone can give an estimation of velocity in mm/s on its translational movement on $y$ axis, or in other word, leftward and rightward movement corresponding to its body frame. We get the feedback displacement by adding an integrator after this signal. Thus, we can compare the reference input to the displacement feedback using their difference as the error. Then the error goes through the controller and works as velocity input to the drone flying to the desired leftward or rightward position. Figure 6.3 describes this in details.

Note here, a pure gain controller has been used again for this. After tuning, $P_{\text{LeftRight}} = 0.3$ works best.
6.2.4 Displacement controller for back and forth movement

The drone can give an estimation of velocity in mm/s on its translational movement on $x$ axis, or in order word, backward and forward movement corresponding to its body frame. This movement is similar to that in the direction translational movement on $y$ axis. Figure 6.3 describes this in details.

Note here, same as before, a pure gain controller has been used. After tuning, $P_{BackForth} = 0.3$ gives the best performance.

6.3 Rotation matrix

During experiment, a drawback has been detected for the displacement controllers designed in Section 6.2.3 and Section 6.2.4. The drone’s horizontal movement is performed based on its body coordinate system instead of the inertial coordinate system. In this case, if the drone has a rotation, it is difficult to let it arrive the desired position in space.

However, this problem can be solved by introducing an rotation matrix which transfer the movement on body coordinate system to it on inertial coordinate system. Since this problem only happens in the two horizontal translations, a 2 by 2 rotation matrix is needed. The rotation angle is defined by $\theta_{rot} = (yaw_{current} - yaw_{initial}) \times \pi/180$. And the rotation matrix is defined by \((6.1)\).
CHAPTER 6. IMPROVED CONTROLLER DESIGN

\[ R(\theta_{rot}) = \begin{bmatrix} \cos \theta_{rot} & -\sin \theta_{rot} \\ \sin \theta_{rot} & \cos \theta_{rot} \end{bmatrix} \] (6.1)

During this transformation procedure, we have two-step work. First, we need to transfer the sensor velocities data from body frame to inertial frame to do the integration. Second, we need an inverse transformation to give the input velocities to drone in its body frame. The forward transformation is defined by (6.2) and the inverse one is defined by (6.3).

\[
\begin{bmatrix} v_{x_{\text{inertial}}} \\ v_{y_{\text{inertial}}} \end{bmatrix} = \begin{bmatrix} \cos \theta_{rot} & -\sin \theta_{rot} \\ \sin \theta_{rot} & \cos \theta_{rot} \end{bmatrix} \begin{bmatrix} v_{x_{\text{body}}} \\ v_{y_{\text{body}}} \end{bmatrix}
\] (6.2)

\[
\begin{bmatrix} v_{x_{\text{body}}} \\ v_{y_{\text{body}}} \end{bmatrix} = \begin{bmatrix} \cos \theta_{rot} & -\sin \theta_{rot} \\ \sin \theta_{rot} & \cos \theta_{rot} \end{bmatrix}^{\top} \begin{bmatrix} v_{x_{\text{inertial}}} \\ v_{y_{\text{inertial}}} \end{bmatrix}
\] (6.3)

Therefore, we have to put the two horizontal movement models together. The combined model is shown in Figure 6.4

![Diagram](image)

Figure 6.4: Horizontal displacement controller with rotation matrix implementation

6.4 Tracking with displacement control

Now we have to confront the tracking disadvantage for introducing the displacement control claimed in Section 6.1.
CHAPTER 6. IMPROVED CONTROLLER DESIGN

6.4.1 Strategy

It is hard to mapping the frame distance in pixel to real-world distance in meters. But recall the previous strategy that we use the distance error in each frame as velocity input to the drone. Now we need a displacement input. Since we get displacement by integrating velocity, we can use a similar way which is created the displacement input by integrating the error in each frame. The back and forth displacement has been set to zero, since we prove the area signal is not reliable to make the drone move back and forth to track the object in Section 5.1 and in this case, we want the drone to keep its location in this direction.

This strategy can be seen as a trick to avoid taking the transferring from pixels to meters into consideration. Thus, we have now have two tracking method. One is tracking by turning the drone’s head. The other is let the drone doing horizontal movement. The block diagram in Figure 6.5 and Figure 6.6 describes the two tracking methods respectively. Note here, the subsystems inside Figure 6.5 and Figure 6.6 are the displacement control models we designed in previous section which are Figure 6.1, Figure 6.2 and Figure 6.4.

Figure 6.5: Head turning tracking block diagram
6.4.2 Experiment results

We use a pure gain controllers here for these. The distance between the drone and the object has been set to 3 meters. With different gains, different results have been shown here.

Figure 6.6 portrays the horizontal movements with the displacement tracking. It expresses three sets of data with different proportional parameters 1.0, 1.2 and 1.4 while 1.2 for P has the best performance. We can see they are much more stable and the amplitude of oscillation is less compared to the trajectory in Figure 5.3. Also, they have shorter rising time.

Figure 6.7 depicts the self-rotation movements with the displacement tracking. It expresses three sets of data with different proportional parameters 1.0, 1.2, 1.4 and 1.4 for P gives the best performance. As we see, the performance is similar with them in Figure 5.5. But, the rising time is a bit longer.

Figure 6.9 portrays the horizontal movements with the displacement tracking. It expresses three sets of data with different proportional parameters 1.0, 1.2 and 1.4 while 1.2 for P has the best performance. We can see the amplitude of oscillation is smaller compared to the trajectory in Figure 5.7. They also have shorter rising time.
CHAPTER 6. IMPROVED CONTROLLER DESIGN

Figure 6.7: System response – Horizontal movement by displacement P controllers

Figure 6.8: System response – Self-rotation movement by displacement P controllers
Figure 6.9: System response – lifting movement by displacement P controllers
Chapter 7

Approximate Models

With the help of the displacement feedback, we can now get better environment for the system identification procedure. We are about to do it again in this chapter.

7.1 The preferred tracking method

In Section 6.4.1 we describe two tracking methods, head turning tracking outlined in Figure 6.5 and horizontal moving tracking portrayed in Figure 6.6.

Since we do not have reliable signal to make the drone follow the object by moving back and forth, and coupling effects occur mostly between horizontal movements (the back and forth movement and left and right movement), we skip these by making the drone stay where it is in the horizontal plane during tracking. And this is obtained by keeping sending zero displacement signals for the drone’s horizontal movements.

In this case, the tracking we use can be illustrated in Figure 7.1. The drone can rise up or descend to track the object and always turning its head towards it.

7.2 Second attempt – System identification

We want to identify the model from the drone’s input to the output in the camera frame. This time, we break this procedure into two parts. First part is getting the model from input to the drone’s real-world location and second part is from the real-world location to the camera frame. As stated above, we are doing this procedure for only two direction movements of the drone, the lifting and the heading.


7.2.1 Models from input to real-world location

In [5], they are doing way-point tracking using the same drone. In their simulation part, it provides model information about the drone movement. There are two models we find useful. The one is the drone turning model from input to head-turning output in rad expressed in (7.1). The other is the drone lifting model from input to altitude change output in meter specified in (7.2).

\[ G_{yaw} = \frac{1.2653}{s + 0.005879} \] (7.1)

\[ G_{altitude} = \frac{0.1526s + 5.1529}{s^2 + 5.8200s + 1.375 \times 10^{-11}} \] (7.2)

If their models are correct, we can use the model information for our research. With these model information, we are doing a model validation to see if they work for the drone we have. Depending on the validation results, we either modify the model a bit to fit the drone or doing system identification again if the model is not accurate at all.
CHAPTER 7. APPROXIMATE MODELS

7.2.1.1 Heading Model

In order to validate the model, we need to do experiment of turning the drone’s head and collect the data.

The signal described in Figure 7.2 upper part has been used as the yaw input. They can be seen as several impulses with amplitude of 0.3 for 1 second. And the lower part illustrates the output result in degree getting from the drone sensor in experiments.

We set the initial value of the output to zero by subtracting the original initial value from each output data. And we change the unit of measurement from degree to rad by multiplying \( \pi \) and divided by 180.

And then, we use the MATLAB command ‘lsim’ to simulate the output from yaw model (7.1) with the same input as in the experiment. We put the experiment output and the simulation output together in Figure 7.3.

After comparing the results from the experiment and the original model, we know that both of them have approximately the same rising time but they don’t share the same amplitude. We also understand that, even if they are all AR.Drone 2.0, they maybe have slightly different model. The original model is therefore invalided but we can try to do some modification.

In this case, it is reasonable to add a gain to the original model to fit the experiment data. And we find a gain of 1.6 fits them best. Therefore, a yaw model with a gain 1.6 to the original one is specified in (7.3). And the simulation with the same input as before has been performed to the modified model. The result is shown in Figure 7.3.

\[
G_{\text{yaw}} = \frac{2.0245}{s + 0.005879} \quad (7.3)
\]

7.2.1.2 Altitude Model

The same procedure as Section 7.2.1.1 has been performed to find the altitude model.

The signal described in Figure 7.4 upper part has been used as the lift input. It is similar to the yaw identification input but with a different amplitude of 0.5 for 1 second. And the lower part illustrates the output result in meters getting from the drone altitude sensor in experiments.

We set the initial value of the output to zero by subtracting the original initial value from each output data. And we change the unit of measurement from millimeter to meter by multiplying 1/1000.

41
CHAPTER 7. APPROXIMATE MODELS

Figure 7.2: Heading identification – input and output data
CHAPTER 7. APPROXIMATE MODELS

Figure 7.3: Three heading outputs from experiment, modified model and original model

And then, we use the MATLAB command ‘lsim’ to simulate the output from altitude model (7.2) with the same input as in the experiment. The results have been put together in Figure 7.5.

From the figure, we can modify the original model to make it predict the output more precisely. After analyzing, multiplying a scalar is an acceptable way. This time, the scalar has been chosen for 0.9 and the model becomes (7.4). The simulation results of this model is included in Figure 7.5.

\[
G_{\text{altitude}} = \frac{0.1373s + 4.6376}{s^2 + 5.8200s + 1.375 \times 10^{-11}}
\]  

(7.4)

7.2.2 Models from real-world location to camera frame

In this section, we are going to find the model from the real-world location to the camera frame location. As these models vary depending on the distance from the drone to the object, we fix this distance to be 3 meters. Since we can not measure this distance in the real-time, the following models we create only works with the constrain of three meter distance and its neighborhood.

Thus, it is rational to assume these models as different scalars. The following subsections demonstrate how to find these two scalars mapping the real-world location to the camera frame location receptively.
CHAPTER 7. APPROXIMATE MODELS

Figure 7.4: Altitude identification – input and output data
CHAPTER 7. APPROXIMATE MODELS

7.2.2.1 From heading direction to horizontal displacement in frame

We are going to let the drone doing some rotation and record the heading direction and the frame location. Since we assume the model is a scalar, we obtain this scalar by computing the frame location changes in pixel divided by the heading direction changes in rad.

Several positive and negative step signal as input and the yaw displacement model in Figure 6.2 have been used for this experiment. The result is expressed in Figure 7.6. Note here, in this experiment, the drone is turning left and right and we care about the locations in real world and in camera frame, not the experiment time. Therefore, the data plotted on the horizontal axis is the heading direction in rad while the data on the vertical axis is the frame location in pixel.

As observed, the area with higher density of data points indicates higher probability of data reliability. The two extreme areas have been circled as illustrated Figure 7.6 with the center points of (0.1, 60) and (0.8, 500).

Thus, this scalar model from heading direction difference in rad to frame location difference in pixel is calculated by (7.5).
CHAPTER 7. APPROXIMATE MODELS

![Diagram](image)

**Figure 7.6**: Data density – heading direction to frame location

\[
k_{\text{rad2pixel}} = \frac{\Delta_{\text{pixel}}}{\Delta_{\text{rad}}} = \frac{500 - 60}{0.8 - 0.1} = 628.5714 (\text{pixel/rad}) \quad (7.5)
\]

### 7.2.2.2 From altitude to vertical displacement in frame

We make the drone do a set of up and down movement and record the altitude and the frame location. Same as before, we obtain the scalar model by computing the frame location changes in pixel divided by the altitude changes in rad.

The altitude displacement model in Figure 6.1 with an input signal consisting of several positive and negative steps has been used for this experiment. The experiment data is shown in Figure 7.7. The data plotted on the horizontal axis is the altitude from the floor in meter and the data on the vertical axis is the frame location in pixel.

Areas with the top two density have been circled in Figure 7.7 and their center locations are (0.43, 163) and (1.08, 287). In this way, the scalar model from altitude change to frame location change can be computed by (7.6).
CHAPTER 7. APPROXIMATE MODELS

Figure 7.7: Data density – altitude direction to frame location

\[
k_{\text{meter2pixel}} = \frac{\Delta_{\text{pixel}}}{\Delta_{\text{meter}}} = \frac{287 - 163}{1.08 - 0.43} = 190.7692 (\text{pixel/meter}) \tag{7.6}
\]

7.3 Summary for the two models

In Section 7.2.1 and Section 7.2.2 we demonstrate the model from drone input to real-world location change and the model from real-world location change to camera frame location change of the distance of three meters.

Thus, we can combine these two models to get the model from drone input to the camera frame location. And the delay describable in Section 3.4 has been brought in the model.

7.3.1 The heading tracking model

The block diagram of our heading tracking system model is specified in Figure 7.8. Note the drone can only accept input turning signal from -1 to 1, so there is an saturation element.
CHAPTER 7. APPROXIMATE MODELS

7.3.2 The altitude tracking model

The block diagram of our altitude tracking system model is specified in Figure 7.9. Note the drone can only accept input lifting signal from -1 to 1, so there is an saturation element.

Figure 7.9: Tracking system model – altitude tracking
Chapter 8

Conclusion

Up to now, the best controllers we have are the proportional controllers implemented in the outer loop. After multiple experiments of parameter adjustment, we ensure that they are the best proportional controllers. The results and data are shown in Chapter 6.

Chapter 7 creates an mathematical model for this system. These models we have are good tools for future controller design and can be used to do the simulation before implement the controllers to the real system.

Right now, we have the ability to give a demo using the device and platform specified in Chapter 2. Appendix A is the instruction for how to do this.

We have tried many other controllers such as the PD and PID controllers. After applying them to our system, they do not have the performance improvement we expect. We believe the bottleneck could be the system delay of 0.26 second. It is hard to design better controllers with system delay. Our main future work is now focused on how to deal with this system delay including identifying where it is from and finding ways to eliminate the delay.

This concludes my thesis.
Bibliography

[1] taofang, “Dji phantom 3 altitude test - above the clouds + fireworks,” https://www.youtube.com/watch?v=wmaXMhoa1xY


Appendix A

How to Do the Demo

A.1 How to Configure the System

Before reading this document, it is strongly suggested to check *How to use the AR Drones* written by Jose Lopez. Its content includes the basic idea of connecting the AR.Drone to the computer via Wi-Fi.

All the codes required for running this demo have been packed and stored in the Linux Machine at the Northeastern Robust System Lab.

A.1.1 Install ROS System

Follow the instructions at the link [here](#).

ROS Version Indigo is suggested.

A.1.2 Creating the Workspace

Follow the instructions at the link [here](#) and take a look at the section *Create a ROS Workspace* using catkin.

A.1.3 Ardrone-autonomy Installation

Follow the instructions at the link [here](#).

A.1.4 Required File Operation

Add the workspace `/devel/setup.bash` to `/bashrc`
APPENDIX A.  HOW TO DO THE DEMO

Go to ‘ros_tracker_packages’ folder. Unpack the tld_tracker.tar and copy the whole ‘tld_tracker’ folder in the ‘src’ (not the first ‘tld_tracker’ folder) to the workspace ‘src’ folder and do the catkin_make again in command line at the workspace path.

A.2 Some useful commands in command line

These commands are used to check the installation procedure in readiness for starting the demo.

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>• echo $ROS_PACKAGE_PATH</td>
<td>check package path</td>
</tr>
<tr>
<td>• source /catkin_ws/devel/setup.bash</td>
<td>change default path</td>
</tr>
<tr>
<td>• roscd ardrone_autonomy</td>
<td>check the path if correct</td>
</tr>
</tbody>
</table>

Table A.1: ROS useful commands

A.3 How to Start the Demo

If all the procedures above are done successfully, follow the steps below. Note here, for all the commands below, type them in order and in each separated command line tab.

A.3.1 Initialize ROS

Type the command in a new tab:
roscore

A.3.2 Connect to drone (after your drone is connect to the computer via Wi-Fi)

To run at 15Hz (Data sending out) (Preferred), type the command in a new tab:
rosrun ardrone_autonomy ardrone_driver _realtime_navdata:=True _navdata_demo:=1

To run at 200Hz (Data sending out), type the command in a new tab:
rosrun ardrone_autonomy ardrone_driver _realtime_navdata:=True _navdata_demo:=0

A.3.3 Two command for checking status

Type each command in a new tab:
rostopic echo /ardrone/navdata
rostopic hz /ardrone/navdata

A.3.4 Start TLD and choose a target

Type the command in a new tab:
rosrun tld_tracker tld_tracker

A.3.5 Switch to MATLAB

Run the script named run_this in the folder named matlab_script_from_Xiangyu.