Stabilization of a Roll-Tilt Camera on an Autonomous Quadrotor Helicopter

by

William Nathan Pickeral

Submitted to the Department of Mechanical Engineering on May 6, 2011, in partial fulfillment of the requirements for the degree of Bachelor of Science in Mechanical Engineering

Abstract

Harmful algal blooms are becoming an increasingly difficult problem to deal with, particularly in Singapore. The Center for Environmental Sensing and Modeling (CENSAM) has developed a network of autonomous vehicles to find blooms when they occur. The problem is that finding blooms, which are often transient in nature, can be difficult, particularly with slow-moving underwater and surface vehicles. Autonomous “quadrotor” helicopters are being utilized to visually survey large areas to spot these blooms while they are occurring. Here we develop a model for implementing servo motor controlled camera stabilization on these autonomous vehicles. The need for camera stabilization arises because video footage is monitored continuously while the onboard GPS is controlling the motion of the quadrotor. The operator of the quadrotor may not want to look in the direction that the GPS controller would like to guide the vehicle. We explore implementing a system that gives the operator the ability to control the camera, yet maintain the autonomous nature of the quadrotor. We develop two models for the rotations involved in stabilizing the position and orientation of the camera against the motion of the vehicle it is mounted on. We use these models to investigate the limitations this type of active stabilization would impose on our quadrotor and GPS controller, and discuss the next steps in integrating it into our system.

Thesis Supervisor: Nicholas M. Patrikalakis
Title: Kawasaki Professor of Engineering
Acknowledgments

I would like to thank Professor Nicholas Patrikalakis and Josh Leighton for their guidance and support on this project, as well as the MIT Mechanical Engineering Department.

I also wish to thank my family and friends for their support during the research and writing process - in particular Ashley Perko and Bonnie Blackburn for helping solve some difficult math.

The work described in this thesis was funded in part by the Singapore National Research Foundation (NRF) through the Singapore-MIT Alliance for Research and Technology (SMART) Center for Environmental Sensing and Modeling (CENSAM).
# Contents

1 Introduction .......................................................... 11
   1.1 Harmful Algal Blooms ........................................... 11
   1.2 Use of UAVs in Detecting HABs ................................. 12
   1.3 Stabilization of the Visual Data ............................... 13

2 Hardware .............................................................. 15
   2.1 Quadrotor ......................................................... 15
      2.1.1 Degrees of freedom ....................................... 16
      2.1.2 Sensors .................................................... 16
   2.2 Camera Mount .................................................. 17

3 Kinematics ............................................................ 19
   3.1 Background on Rotations ....................................... 19
      3.1.1 Rotating About X,Y, or Z ................................. 19
      3.1.2 Rotating About a Vector ................................ 20
   3.2 Modeling Actuations ............................................ 21
   3.3 Camera Correction Without Yaw Control ...................... 23
      3.3.1 Arbitrary Pitch and Roll of the Quadrotor ............. 23
      3.3.2 Roll Correction ............................................ 25
      3.3.3 Tilt Correction ............................................. 25
      3.3.4 Error in the Roll and Tilt Vectors ....................... 27
   3.4 Camera Correction with Yaw Control ......................... 30
4 Implementing Control

4.1 Comparison of 2 DOF and 3 DOF Models .......................... 38
4.2 Computation on Board the Quadrotor ............................. 39
4.3 Limitations on the GPS Controller ................................. 40
4.4 User Control of the Camera ........................................ 41

5 Conclusions and Recommendations ................................. 45

A Mathematica Code for Finding Correction Angles in the 2 DOF
System ............................................................... 49

B Mathematica Code for Finding Correction Angles in the 3 DOF
System ............................................................... 55
List of Figures

2-1  Ascending Technologies’ AscTec Pelican (http://www.asctec.de) . . . 15
2-2  Roll-Tilt camera mount . . . . . . . . . . . . . . . . . . . . . . . . . . 17

3-1  Definition of coordinate system . . . . . . . . . . . . . . . . . . . . . 20
3-2  Arbitrary pitch and roll of the quadrotor . . . . . . . . . . . . . . . 22
3-3  Camera vectors in initial position and after arbitrary pitch and roll . 24
3-4  Camera vectors before and after roll correction of \( n \) . . . . . . . . . 26
3-5  Camera vectors before and after a camera tilt of \( m \) . . . . . . . . . 28
3-6  Camera vectors in initial position and after arbitrary pitch and roll . 29
3-7  Camera vectors before and after a yaw of \( \gamma \) . . . . . . . . . . . . . 32
3-8  Camera vectors before and after a roll of \( n \) . . . . . . . . . . . . . . 33
3-9  Camera vectors before and after a tilt of \( m \) . . . . . . . . . . . . . . 34
3-10 Camera vectors in initial position and after arbitrary pitch and roll . 35

4-1  Difference in initial and final tilt vectors in 2 DOF system, \( m_0 = 45 \) . 38
4-2  Difference in initial and final tilt vectors in 2 DOF system, \( m_0 = 10 \) . 40
4-3  Camera roll as a function of quadrotor pitch and roll, \( m_0 = 45 \) . . . . 41
4-4  Camera tilt as a function of quadrotor pitch and roll, \( m_0 = 45 \) . . . . 42
Chapter 1

Introduction

1.1 Harmful Algal Blooms

Changing conditions have made Harmful Algal Blooms (HABs) an increasing concern in recent years. HABs are occurring more frequently and their impact on marine life, as well as local economics can be devastating. In January of 2010, fish farms near Pasir Ris and Pulau Ubin in Singapore were hit by an algal bloom. Over 200,000 fish were killed, wiping out the stock of nearly 13 farms and causing over millions in economic damages. The deaths were due to decreased oxygen levels in the water as a result of plankton blooms [Quek and Lim, 2010, Sim, 2010].

Although observations of HABs have become more frequent, we still know little about the cause of these blooms. One of the primary goals for the Center for Environmental Sensing and Modeling (CENSAM, http://censam.mit.edu/) is to better understand the mechanisms and conditions that cause algal blooms, allowing us to place fish farms in areas less susceptible to damage by the algae. CENSAM is currently working to associate easily measurable environmental variables (temperature, salinity, and dissolved oxygen) with algal blooms [Ooi et al., 2010]. Unfortunately, in order to measure the conditions of the algal blooms, we must first find them. This can difficult due to the localized and transient nature of the blooms [Richardson, 1996].

CENSAM has developed a network of Autonomous Surface Vehicles (ASVs) and Autonomous Underwater Vehicles (AUVs) to study the conditions of the Singapore
coastal zone [Patrikalakis et al., 2010]. These platforms can be used to gather data about the blooms, but locating them still remains difficult. A third platform, is being developed to help locate HABs. Small autonomous “quadrotor” helicopters are being used to visually cover large areas and spot algal blooms quickly. Slower surface and underwater vehicles can then be deployed to the area.

1.2 Use of UAVs in Detecting HABs

Surface and underwater vehicles are traditionally used to observe conditions that indicate algal blooms such has pH, conductivity, concentration of Chlorophyll-a, dissolved oxygen, and concentration of nutrients needed for algae growth [Ooi et al., 2010]. However, there is little data available on the waters near Singapore, making it difficult to build a model for predicting their location and gather data on the behaviors of the blooms. The use of Unmanned Arial Vehicles (UAVs) is well suited for the task of locating algal blooms in the Johor Strait of Singapore due to the fact that HABs in this region are expected to appear as dark brown patches on the surface of the water; allowing for visual detection. In addition the blooms in the Johor Strait are short lived, lasting only a few hours. This means there is a need to be able to survey large areas in a short period of time.

To accomplish the task of finding algal blooms, UAVs are launched from surface vessels near candidate areas predetermined from models using past data. These areas are based of A GPS controller is used to follow a predetermined flight path and a human operator reviews pictures or video capture by the UAV. When a potential algal bloom is found, the UAV is able to maintain its GPS coordinates and relay its coordinates to ASVs and AUVs, allowing them to collect measurements of the area.

Typically a fixed wing aircraft would be used for large area surveillance missions, particularly in harsh environments, but the need to launch and land on small boats necessitated the use of a helicopter. The choice of a helicopter also allows the UAV to hover in one location and monitor a bloom. The helicopter’s heading can be independently controlled, simplifying the camera mount and allowing for pan control of the
camera without an added motor. A quadrotor helicopter was chosen because they are mechanically simpler than a standard helicopter and require little maintenance. They also have simpler controls, as their flight performance is completely symmetric. The use of a camera mounted on a quadrotor and autonomous GPS controls introduced the need to stabilize the camera in order for a human operator to easily monitor the video sent from the quadrotor.

1.3 Stabilization of the Visual Data

While there is a manual control option available for safety reasons, in normal conditions the GPS controller allows the quadrotor to operate fully autonomously, following waypoints. As the GPS controller causes the pitch and roll of the quadrotor to change, the direction and orientation of the camera changes. This undesirable change in camera position may cause a human operator to lose track of an area they are monitoring, particularly if the angles of rotation were large. To correct for variation introduced by pitching and rolling of the quadrotor, we need to be able to tilt and roll the camera frame independently of the vehicle. Even after introducing two degrees of freedom in the camera control, it is still only possible to stabilize the camera's position, or its orientation. In order to correct both simultaneously, we will need a third degree of freedom. Fortunately, quadrotors are designed in such a way that the yaw of the vehicle can be independently controlled.

We will examine the kinematics involved in determining the necessary correction commands for our particular setup, and how we can implement this with the design limitations imposed by the above mission.
Chapter 2

Hardware

2.1 Quadrotor

There are many commercially available quadrotors. For our purposes, we have chosen to use Ascending Technologies’ AscTec Pelican. The Pelican, shown in 2-1, has a payload capacity of 500g and a flight time of about 20 minutes at maximum speeds of 10 m/s.

Figure 2-1: Ascending Technologies’ AscTec Pelican (http://www.asctec.de)
2.1.1 Degrees of freedom

Quadrotors have four fixed rotors, with adjacent motors turning in opposite directions. This creates a zero net torque about yaw axis, meaning that yaw stabilization is not needed. Pitch, roll, and yaw are all induced by varying the relative speed of two of the four propellors, creating mismatched torques between those two propellors. Yaw is induced by varying two adjacent rotors, and pitch and roll are induced by varying opposite motors (ones that share a common axis). In this manner, yaw can be controlled independently of the other two motions, allowing us to use a two degree of freedom (2 DOF) camera mount and still correct for both position and orientation of the camera.

Initial pitch, roll, and yaw will be treated as our input variables. Any change to the pitch and roll will be treated as noise that we want to correct for. Tilt and roll of the camera, as well as yaw, will be treated as our outputs.

2.1.2 Sensors

The AsTec Pelican, in its standard configuration, is equipped with a full range of sensors. The vehicle has accelerometers and gyroscopes to measure all three angles of rotation, as well as a GPS sensor, compass, and pressure sensor. Pitch, roll, thrust, and yaw commands can be sent to the vehicle using a high level interface, and real time values can be read. For stabilizing the camera, we are concerned with reading the pitch, roll, and yaw variables. We than want control the yaw of the quadrotor as well as send servo commands to the tilt and roll motors on the camera mount. Current orientation of the quadrotor will be read, the correction calculated, and command signals sent to the camera mount and quadrotor.

With the onboard controller, we are also able to set limits on the pitch, roll, and yaw rates; this is useful in that the quadrotor is much more agile than the servos controlling the camera mount. Setting hard limits on the rate of rotation of the vehicle will ensure that the camera mount is always faster than the helicopter.
2.2 Camera Mount

In order to keep the structure of the quadrotor out of the field of view of the camera, the camera is mounted at a 45 degree rotation from the axes of the quadrotor about the yaw axis of the quadrotor (i.e. half way between the two axes of the quad rotor). The camera mount has two Graupner DS 3068 Servo motors to control the tilt and roll of the camera. Tilt is limited to rotation from 0° to 90° (horizontal to vertical). Roll is limited to +/-20°. The tilt servo is mounted to the roll arm so that rolling causes a change in the tilt axis. Figure 2-2 shows the camera mount. The roll servo is connected to the mount through a three bar linkage. The actual motion, for this model is complex; however near a roll angle of 0 the lever ratio from this particular setup is about 1:3. While each servo can move at 0.262 rad/sec, the roll’s lever ratio slows it to a speed closer to 0.087 rad/sec.

![Figure 2-2: Roll-Tilt camera mount](image)

Since the camera rotates about a different axis from that of the quadrotor, a small amount of translation will occur even after we attempt point the camera in the correct direction. However, since the quadrotor is translating anyways (that is the purpose of pitching and rolling the vehicle), we will only be concerned with pointing the camera
in the correct direction and keeping it level. For this reason, we will model the point of rotation the same for both the camera and the quadrotor.
Chapter 3

Kinematics

In order to accomplish the correction necessary to keep the camera pointed in the same direction, we will need to model the rotations and determine a solution that places the camera vector in its original position. To start, it is useful to define our coordinate systems. As shown in figure 3-1, coordinate system 0 is our reference coordinate system and the same as our body-fixed coordinate system when the quadrotor is in its initial position. Coordinate systems 1 and 2 are body-fixed to the vehicle; coordinate system 3 is body fixed to the camera so that \( \hat{x}_3 \) is in the direction of the camera. Coordinate system 2 is rotated 45° from 1, so that \( \hat{x}_2 = \hat{x}_3 \) if the camera is tilted 0°. This makes \( \hat{x}_2 \) the axis about which the camera rolls, and \( \hat{y}_2 \) the axis about which the camera tilts. The “quadrotor” pitches about \( \hat{y}_1 \) and rolls about \( \hat{x}_1 \).

3.1 Background on Rotations

Considering most of the difficulty in this problem involves rotating, it useful to review some basic linear algebra and rotation matrices.

3.1.1 Rotating About X,Y, or Z

To start, many rotations can be modeled as rotations about an axis. Equations 3.1, 3.2, and 3.3 are rotations about x, y, and z respectively.
Figure 3-1: Definition of coordinate system

\[
R_x(\theta) = \begin{pmatrix}
1 & 0 & 0 \\
0 & \cos(\theta) & -\sin(\theta) \\
0 & \sin(\theta) & \cos(\theta)
\end{pmatrix}
\] (3.1)

\[
R_y(\theta) = \begin{pmatrix}
\cos(\theta) & 0 & \sin(\theta) \\
0 & 1 & 0 \\
-\sin(\theta) & 0 & \cos(\theta)
\end{pmatrix}
\] (3.2)

\[
R_z(\theta) = \begin{pmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{pmatrix}
\] (3.3)

3.1.2 Rotating About a Vector

Many times, however, it easier to express a rotation about an arbitrary axis. To do this we first need to express our axis as a vector, \( \hat{u} = \{u_x, u_y, u_z\}^T \). Then a rotation of \( \theta \) about \( \hat{u} \) can be expressed by:
\[ R_u(\theta) = \begin{pmatrix}
    \cos \theta + u_x^2(1 - \cos \theta) & u_xu_y(1 - \cos \theta) - u_z \sin \theta & u_xu_z(1 - \cos \theta) - u_y \sin \theta \\
    u_xu_y(1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2(1 - \cos \theta) & u_yu_z(1 - \cos \theta) - u_x \sin \theta \\
    u_xu_z(1 - \cos \theta) + u_y \sin \theta & u_yu_z(1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2(1 - \cos \theta)
\end{pmatrix} . \tag{3.4} \]

### 3.2 Modeling Actuations

Using the above rotation matrices it is possible to develop a model of the rotations in our system. There are two distinct options available in implementing our correction. We could choose to simply control the camera servos and accept the small amount of error in our orientation, or we could opt to add the extra complexity and implement yaw control as well. We will examine both scenarios.

In either case, the first step is determining the current position of the quadrotor. We are given pitch, roll and yaw angles from the accelerometers and gyroscopes on the quadrotor. From this we must extrapolate the rotation that ended us in this position.

We will assume that yaw angle is a variable we control, and that it has not changed. Fig. 3-2 shows coordinate systems 0 and 1 after an arbitrary pitch, \( \alpha \), and roll of the quadrotor, \( \beta \). \( \alpha \) is the angle between the projection of \( \hat{z}_1 \) into the \( \hat{x}_0-\hat{z}_0 \) plane and \( \hat{z}_0 \). \( \beta \) is the angle between the projection of \( \hat{z}_1 \) into the \( \hat{y}_0-\hat{z}_0 \) plane and \( \hat{z}_0 \). It is difficult to define a rotations based on these two values as they do not describe intuitive rotations.

We define two new variables, \( \theta \) and \( \phi \) to better locate our system. With some trigonometry, we can relate \( \alpha \) and \( \beta \) to \( \theta \) and \( \phi \). The result is:

\[ \tan \theta = \frac{\tan \alpha}{\tan \beta} \tag{3.5} \]

\[ \tan \phi = \sqrt{\tan^2 \alpha + \tan^2 \beta}. \tag{3.6} \]
Figure 3-2: Arbitrary pitch and roll of the quadrotor
\( \theta \) and \( \phi \) allow us to define our new position with a few rotations. Namely, coordinate system 0 is a rotation of \( \phi \) about \( \hat{x}_0 \) after rotating \( \hat{x}_0 \) about \( \hat{z}_0 \) by \( \theta \). This can be expressed in equation 3.7, where \( \hat{z}_1^0 \) is the \( z_1 \) vector expressed in the 0 coordinate system (superscripts denote the coordinate of reference).

\[
\hat{z}_1^0 = R_z(-\theta)R_x(\phi)R_z(\theta)\hat{z}_0
\] (3.7)

### 3.3 Camera Correction Without Yaw Control

#### 3.3.1 Arbitrary Pitch and Roll of the Quadrotor

Once we can express the pitch and roll of the quadrotor as rotations, we are able to also express the camera vector in our initial condition, and after an arbitrary pitch and roll of the quadrotor. Figure 3-3 shows the camera vector, camera roll axis, and camera tilt axis before and after an arbitrary vehicle pitch and roll. The camera vector, \( \hat{x}_3^2 \), can initially be expressed in coordinate system 2 as \( \hat{x}_3^2 = (\cos m_0, 0, -\sin m_0)^T \) where \( m_0 \) is the initial camera tilt. We can use a rotation about \( \hat{z}_0 \) to express the vector in coordinate system 0, but it is more useful to leave it in coordinate system 2 for the time being.

We can then express the camera vector after a pitch and roll, in coordinate system 2, by using the same rotations as in equation 3.7. To cut down on the overall number of rotations, it is simplest to work in a coordinate system defined by rotating coordinate system 0 about \( \hat{z}_0 \) by \( \theta \). We call this new system, coordinate system 4. We can move from 2 to 4 by rotating about \( \hat{z}_0 \) by \( \frac{\pi}{4} - \theta \). We then rotate by \( \phi \), so the new camera vector in coordinate system 4 is \( \hat{x}'_3 \), where the prime denotes a rotation. This vector can be expressed as:

\[
\hat{x}'_3 = R_x(\phi)R_z(\frac{\pi}{4} - \theta)\hat{x}_3^2
\] (3.8)
Figure 3-3: Camera vectors in initial position and after arbitrary pitch and roll
3.3.2 Roll Correction

Now that we have an expression for the rotated camera vector, we need to correct for the pitch and roll of the quadrotor by rotating about the camera roll and tilt axes. Due to the design of the mount, the camera tilt axis is affected by rolling the camera, so it makes sense to roll the camera first. Figure 3-4 shows the important vectors prior to roll correction and after a roll correction of \( n \).

We want to roll about \( \hat{x}'_2 \) in coordinate system 4, so if we can express \( \hat{x}'_2 \) as a vector in coordinate system 4 we can rotate \( \hat{x}'_3 \) about that vector. Fortunately, the rotations in equation 3.7 can be used to do exactly that, namely:

\[
\hat{x}'_2 = R_x(\phi)R_z(\frac{\pi}{4} - \theta)\hat{x}_2
\] (3.9)

We can then express the camera roll, in coordinate system 4, by to rotating about \( \hat{x}'_2 \) by \( n \). So our new camera vector is then:

\[
\hat{x}''_3 = R_x'(n)R_x(\phi)R_z(\frac{\pi}{4} - \theta)\hat{x}'_3
\] (3.10)

3.3.3 Tilt Correction

The final step is to tilt correct the camera. Figure 3-5 shows the camera vectors before and after tilt correction. We are tilting about \( \hat{y}'_2 \), in coordinate system 4. That is, we are tilting about the original tilt axis after it has been rotated by the pitch and roll of the quadrotor as well as the roll of the camera, and we want to continue working in coordinate system 4. The vector around which we tilt, \( \hat{y}''_2 \), can than be expressed as:

\[
\hat{y}''_2 = R_x'(n)R_x(\phi)R_z(\frac{\pi}{4} - \theta)\hat{y}_2
\] (3.11)

Our final step is then to rotate our camera vector about \( \hat{y}''_2 \), using rotation about an arbitrary vector. This gives us a final camera vector, \( \hat{x}'''_3 \), in coordinate system 4 of :
Figure 3-4: Camera vectors before and after roll correction of $n$
\[
\hat{x}_3''' = R_{z_2}(m)R_{x_2}(n)R_x(\phi)R_z\left(\frac{\pi}{4} - \theta\right)\hat{x}_3^2
\]  
(3.12)

The only problem with equation 3.12 is that our final camera vector is expressed in coordinate system 4, and our initial camera vector was expressed in coordinate system 2. We solve this by rotating about \(\hat{z}_0\) by \(\theta - \frac{\pi}{4}\), so that:

\[
\hat{x}_3'' = R_z(\theta - \frac{\pi}{4})R_{y_2}(m)R_{x_2}(n)R_x(\phi)R_z\left(\frac{\pi}{4} - \theta\right)\hat{x}_3^2
\]  
(3.13)

We can then compare our initial camera vector and final camera vectors to solve for \(n\) and \(m\).

\[
\hat{x}_3'' = \hat{x}_3^2
\]  
(3.14)

In fact the rotations in equation 3.13 can be used to express any vector in coordinate system 2 after the full set of rotations. So it is possible to examine the error in camera position.

### 3.3.4 Error in the Roll and Tilt Vectors

Figure 3-6 compares the initial and final positions of the camera vector as well as the roll and tilt vectors. Using the 2 DOFs above, we are able to exactly place the final camera vector (solid purple line) over the initial camera vector (solid black line). In other words, the position of the camera is corrected for. However, because we only had 2 DOFs we were left with some error, \(\epsilon\) in the final roll and tilt vectors (dotted purple lines) as compared to the initial roll and tilt vectors (dotted black line). This means that the orientation of the camera was not completely corrected. We would end up pointing at the right object, but a horizontal line in the original camera frame would end up slanted in the final camera frame. It would also be possible to rotate in such a way that the orientation is correct, but the position is incorrect. However, it is impossible to completely correct both the position and orientation of the camera. To do so we must look at a solution involving 3 DOFs.
Figure 3-5: Camera vectors before and after a camera tilt of $m$
Figure 3-6: Camera vectors in initial position and after arbitrary pitch and roll
3.4 Camera Correction with Yaw Control

In order to position and orient our camera, we need a third degree of freedom. Since the yaw of the quadrotor is completely decoupled from any other motion of the vehicle, it is convenient to use this as our third degree of freedom. To include yaw, \( \gamma \), in our model, the analysis is similar to above. The main difference is that yaw affects both the tilt axis and roll axis, so we must take this into account before applying these corrections. Figures 3-7 to 3-10 show the rotations necessary to correct the camera using our 3 degrees of freedom after an arbitrary roll and pitch is applied to the vehicle.

The process involves finding the initially rotated camera vector, then yawing about the quadrotor’s new z axis, and finally applying camera roll and tilt corrections as before. We yaw the quadrotor before applying the camera rotations because yaw affects both of the camera rotations. We will continue to work in coordinate system 4.

Finding the initially uncorrected camera vector is the same as in the previous section, so equation 3.8 still applies. The second step involves finding \( \hat{z}_2' \), the vector about which the yaw occurs:

\[
\hat{z}_2' = R_x(\phi)R_z\left(\frac{\pi}{4} - \theta\right)\hat{z}_2 \tag{3.15}
\]

Once we know this we can rotate other vectors in coordinate system 4 about this vector. We can then write our roll, tilt, and camera vectors with this added rotation.

\[
\hat{x}_2'' = R_{z_2} (\gamma) R_x (\phi) R_z \left(\frac{\pi}{4} - \theta\right) \hat{x}_2 \tag{3.16}
\]

\[
\hat{y}_2''' = R_{z_2} (\gamma) R_y (\gamma) R_x (\phi) R_z \left(\frac{\pi}{4} - \theta\right) \hat{y}_2 \tag{3.17}
\]

\[
\hat{x}_3''' = R_{z_3} (\gamma) R_y (\gamma) R_x (\phi) R_z \left(\frac{\pi}{4} - \theta\right) \hat{x}_3 \tag{3.18}
\]

Again our final and initial camera vectors are expressed in two different coordinate systems.
systems, so to compare them we must rotate one into the other. The total rotation is:

\[
\hat{x}_3^{''''} = R_z(\theta - \frac{\pi}{4}) R_y(m) R_x(n) R_x(\gamma) R_z(\phi) R_z(\frac{\pi}{4} - \theta) \hat{x}_3^2
\] (3.19)

Finally, we need to compare not only the camera vectors, but the initial and final tilt vectors as well to have enough information to solve for all three angles. To get our final tilt vector, we apply the same rotations as in 3.19:

\[
\hat{y}_2^{'''} = R_z(\theta - \frac{\pi}{4}) R_y(m) R_x(n) R_x(\gamma) R_z(\phi) R_z(\frac{\pi}{4} - \theta) \hat{y}_2
\] (3.20)

The vectors we want to compare are:

\[
\hat{x}_3^{''''} = \hat{x}_3^2
\] (3.21)

\[
\hat{y}_2^{'''} = \hat{y}_2
\] (3.22)

Figure 3-10 shows the initial and final vectors. Notice that our roll, tilt, and camera vectors are all aligned after the corrections. We can successfully position and orient our camera.
Figure 3-7: Camera vectors before and after a yaw of $\gamma$
Figure 3-8: Camera vectors before and after a roll of $n$
Figure 3-9: Camera vectors before and after a tilt of $m$
Figure 3-10: Camera vectors in initial position and after arbitrary pitch and roll
Chapter 4

Implementing Control

Now that we have two valid models for the rotations of the mount and the quadrotor, we need to implement a control system for the camera mount. We first need to determine a way to solve for the desired angles, then we need to determine which model is best for implementation. Finally, we need to find a way to implement that model with the limitations of the quadrotor.

Unfortunately, our solution involving 2 DOFs and our solution involving 3 DOFs are both non-linear with many cross terms between the variables we would like to solve for. An analytical solution is simply not feasible, so we must opt to use a numerical solver. Fortunately, there are many programs available with numerical analysis packages. We have used Wolfram Mathematica and the FindRoot method to solve our system. This method uses a modified version of Newton’s Method to find the roots of a given system. Appendices A and B show the Mathematica code used to find solutions in either case.

The main difficulty in using a numerical solver is that for each situation we have more equations than unknowns. For the 2 DOF model, we have 3 equations and 2 unknowns. This comes from the fact that the camera vector has 3 components. For our 3 DOF system we are correcting both the camera vector and the roll vector. Both of these vectors have 3 components, giving us 6 equations in 3 unknowns. In the 2 DOF system, it is possible to solve all 3 combinations of the 3 equations and 2 unknowns and choose the best solution. See lines 59-93 in Appendix A. For the
3 DOF model, it is too computationally intensive to solve all 20 combinations. We implement a method of iteratively solving the system with three components and checking to see if the error is zero for both vectors. If the error is nonzero, we change two of the components we are using and try again. We are able to find a solution in all cases by the second iteration. See lines 90-100 of B.

4.1 Comparison of 2 DOF and 3 DOF Models

While the 2 DOF system is computationally much simpler to solve and therefore less time consuming to deal with, it has one major drawback. With this model we can confidently solve for either position or orientation, but in finding one accurately we create an error in the other. Figure 4-1 shows $\epsilon$, the difference in angle between the initial and final tilt vector. This plot was generated using an initial camera tilt of $45^\circ$, and used a range of $-45^\circ$ to $+45^\circ$ for both $\alpha$ and $\beta$ (the pitch and tilt of the quadrotor). In most cases (particularly for small perturbations) the difference is negligible.

\[
\begin{array}{c}
\end{array}
\]

Figure 4-1: Difference in initial and final tilt vectors in 2 DOF system, $m_0 = 45$
On the plot there is a line where $\epsilon$ becomes much larger than elsewhere on the plot. This line is parallel to the line were only roll correction is necessary ($\alpha = \beta$). It occurs at a distance of approximately $m_0$ from the roll only line and can be described approximately by the equation $\beta = \alpha - m_0\sqrt{2}$. This large error occurs from the fact that a large roll correction is needed in the direction opposite of what would reduce the error between the initial and final tilt vectors. This causes a larger difference in the tilt vectors than with no roll correction. Intuitively there are many solutions that would allow us to reduce this difference in the tilt vectors, but all of these solutions would result in a small error in the camera vectors. We could choose a compromise solution that allows us to have a small error in both the tilt vectors and the camera vectors, but a better solution is to include yaw control. This would allow us to move both the camera vector and the roll vector back into their original position.

While in most cases this is not an issue because the differences in initial and final tilt vectors are less than $20^\circ$, there will always be a small portion of quadrotor rotations that would cause us problems. This issue becomes most apparent at initial camera angles close to zero, as shown in figure 4-2. The large discrepancies occur at small quadrotor rotations that are likely to occur. Choosing to work with the more difficult 3 DOF system and integrating yaw control into our model will allow us to remove these problem areas from our model.

### 4.2 Computation on Board the Quadrotor

While the quadrotor does have an onboard computer, it is limited in computational ability. For this reason it is not practical to try to solve this problem onboard the vehicle. To implement the correction, we would use a three dimensional lookup table. Prior to flight, the solutions would be found for all possible initial conditions. We would solve the system, varying the initial camera tilt and the pitch and tilt of the quadrotor to a $1^\circ$ or $5^\circ$ resolution. We would then build a lookup table with these solutions and interpolate between solutions for angles that are not in the lookup table. Each cell in the table would contain our angles. This assumes that we do not want
to give control of camera roll to the user. If we were to implement a user-controlled camera, we would need a fourth dimension to our lookup table (initial camera roll).

### 4.3 Limitations on the GPS Controller

If we were to use the 2 DOF system, we would have to limit pitch and tilt to certain directions if the camera were initially close to horizontal, due to the large peaks that occur in $\epsilon$ at certain values in our 2 DOF model. The 3 DOF model would only be limited by the physical limitations of the camera mount design, i.e. the roll would be limited to $\pm 20^\circ$. Figures 4-3 and 4-4 plot the necessary tilt and roll corrections as a function of the quadrotor motions.

We can use these plots to find limits for pitch and roll. Bounding by a roll of $\pm 20^\circ$ we can show that we need to satisfy $\beta < 25 - \alpha$ and $\beta > -25 - \alpha$. Bounding by a tilt of $\pm 45^\circ$ (our initial tilt was $45^\circ$, so we are free to move $45^\circ$ in either direction) we can show that we need to satisfy $\beta < \alpha + 65$ and $\beta > \alpha - 65$. These equations give us
limits we can program into our GPS controller. However, while the slopes stay the same, the intercepts change based on the initial position of our camera. The white dotted lines in figures 4-3 and 4-4 are the necessary bounding boxes for $m_0 = 45^\circ$.

4.4 User Control of the Camera

Ultimately we would want to give the human operator pan and tilt control of the camera. This is difficult because it imposes another set of rotations on our system. However, neither one of our models assumed our initial camera vector or tilt vectors were special. As long we kept careful track of our coordinate systems, and our desired positions, it should be possible to implement. We just need to keep careful track of the camera tilt angle. The pan angle should not be a problem because we just need
to reset our reference frame so that its x-axis stays inline with the x-axis of the body fixed frame.

The larger issue is limiting the rates involved in the rotations of the quadrotor to ensure the camera can stay positioned throughout the rotation. The fact that our camera mount is 45° out of alignment with the vehicle is not ideal for this; this prevents a one-to-one correlation of the rotation angles. However, we can look at extrema of our bounding box to gain insight into the necessary rate limits. At a pitch of -45° and a roll of 20° from the quadrotor, we would need 43.96° of tilt correction and 17.94° of roll. This would take the roll servo 3.6 seconds at its maximum speed and the tilt servo 2.9 seconds. Using the slower motor, we would need to limit the pitch of the vehicle to 0.21 rad/sec and the roll to 0.096 rad/sec. Notice that we were limited by the slower servo, so the less distance that servo would need to move,
the faster we would be able to move. In fact the fastest point is when we rotate the quadrotor about the tilt axis and only need to correct by tilting. In this case, we could pitch and tilt as fast as 0.2 rad/sec. However, if we were in line with the roll axis we would need to limit both vehicle rates to 0.06 rad/sec. While less extreme angles would allow for faster rotations, limiting the rotations to the slowest rates (0.06 rad/sec) is sensible.
Chapter 5

Conclusions and Recommendations

With increasing frequency of algal blooms, and higher economic and wildlife costs of said blooms, it becomes increasingly important that we are able to understand and find harmful algal blooms. Current mechanisms of detecting blooms rely on large amounts of past data to try and predict where the blooms are going to occur. However, this method is not always successful, particularly in places like Singapore where blooms can last less than an hour. Fortunately, it is possible to quickly spot some blooms by visually surveying large areas of the ocean. The Center for Environmental Sensing and Modeling, has already proven that quadrotor helicopters can be successfully used to find potentially harmful blooms and direct deployment of surface and underwater vehicles to the quadrotors GPS location.

In this thesis, we work on improving the user experience with cameras mounted to the quadrotors by developing a model for stabilizing the field of view of a camera mounted to an autonomous quadrotor. The pitch and roll of an autonomously controlled vehicle, is potentially distracting to a human operator working with the footage sent from the quadrotor. This distraction could be large enough to prevent the operator from finding a harmful bloom. For this reason we developed two models to attempt to correct for the pitch and roll of a quadrotor, keeping a constant camera orientation and a level field of view. One model uses only a roll-tilt servo controlled camera mount and is limited to either correcting orientation or maintaining a level view. A second model, incorporating quadrotor yaw as a third control variable allows
us to simultaneously orient the camera and level the field of view.

Both of these models have limitations, as the quadrotor is able to pitch and roll over much larger angles than we are able to correct for with our current mount design. The quadrotor is also able to rotate at much faster rates than servo motors on the camera mount. For this reason we must add artificial constraints to both the rates and angles of the GPS controller.

The model without yaw control, however has one major limitation that is not found in the model with yaw control. Without yaw control there is a region where the roll correction moves the vector about which we tilt in the direction opposite to what we desire. This creates a large difference between the initial and final tilt vectors, effectively orienting the camera differently before and after the rotations. This is corrected for by incorporating yaw control, allowing us to both orient camera and level the field of view.

Implementation of these models still needs to occur. However, it hopefully should not be too difficult, as the servo motors have a built-in feedback control. Work needs to be done to transform angle data into the correct pulse widths to send the servos PWM commands. Lookup tables also need to be generated, so that current pitch and roll can be read from the sensors and the proper correction can be found without computation. Implementing yaw control can be done using the existing heading controller on the quadrotor. Work also needs to be done to make sure these stabilization techniques can also work with user control of the camera. Operators should have the ability to pan and tilt the camera while the quadrotor is moving.

Overall, we present a usable model for implementing camera stabilization on an autonomous quadrotor helicopter that will be used to help find algal blooms off the coast of Singapore.
References


C. Quek and J. Lim. 200,000 fish in farms off Pasir Ris dead. The Straits Times, Jan 2010.


M. Sim. Plankton bloom hits Pulau Ubin fish farms. The Straits Times, Jan 2010.
Appendix A

Mathematica Code for Finding Correction Angles in the 2 DOF System

(*camera vector*)
a[ct1_]=\{\{\text{Cos}[ct1]\},\{0\},\{-\text{Sin}[ct1]\}\};

(*roll vector*)
b=\{\{1\},\{0\},\{0\}\};

(*tilt vector*)
d=\{\{0\},\{1\},\{0\}\};

(*Initial camera tilt*)
ct=45*\text{Pi}/180;

(*Rotation Matrix for Theta and Phi*)
RzRot[zRt_]=\{\{\text{Cos}[zRt],\text{Sin}[zRt],0\},\{\text{Sin}[zRt],\text{Cos}[zRt]
\],0\}\};
Rphi[PHI_]=\{\{1,0,0\},\{0,\text{Cos}[PHI],\text{Sin}[PHI]\}\};

49
alpha = Pi/6;
beta = -Pi/4;

(* If both alpha and beta are zero, theta and phi are undefined. We check for this, and than solve *)
If [alpha==0 && beta==0, theta=0; phi=0,
(* Easier angles to work with *)
theta = ArcTan[Tan[alpha], Tan[beta]];
phi = ArcTan[1, Sqrt[Tan[alpha]^2 + Tan[beta]^2]]);

(* Angle between A coord system and axis of rotation *)
zRot = theta - Pi/4;

(* New camera roll axis *)
camRollAxis = Rphi[phi].RzRot[-zRot].b;
ux1 = camRollAxis[[1, 1]];
uy1 = camRollAxis[[2, 1]];
uz1 = camRollAxis[[3, 1]];

(* Calculate camera roll after transformation in original coordinate system *)
tensor = {{ux1^2, ux1*uy1, ux1*uz1}, {ux1*uy1, uy1^2, uy1*uz1}, {ux1*uz1, uy1*uz1, uz1^2}};
skew = {{0, -uz1, uy1}, {uz1, 0, -ux1}, {-uy1, ux1, 0}};
I1 = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

(* Camera Roll Transformation in original coordinates *)
RcamRoll[cR_] = I1*Cos[cR] + Sin[cR]*skew + (1 - Cos[cR])*tensor;
Clear [ux, uy, uz, tensor, skew];

(*New camera tilt axis. Should be a function of cR?*)
camTiltAxis [cR_] = RcamRoll [cR]. Rphi [phi]. RzRot[−zRot]. d;
cta = camTiltAxis [cR];

ux [cR_] = cta [[1, 1]];
uy [cR_] = cta [[2, 1]];
uz [cR_] = cta [[3, 1]];

tensor [cR_] = {{ux [cR]^2, ux [cR]*uy [cR], ux [cR]*uz [cR]},
                {ux [cR]*uy [cR], uy [cR]^2, uy [cR]*uz [cR]},
                {ux [cR]*uz [cR], uy [cR]*uz [cR], uz [cR]^2}};
skew [cR_] = {{0, −uz [cR], uy [cR]},
              {uz [cR], 0, −ux [cR]},
              {−uy [cR], ux [cR], 0}};

RcamTilt [cT_, cR_] = I1*Cos [cT] + Sin [cT]*skew [cR] + (1 − Cos [cT]) *
tensor [cR];

(*Write an equation for final x vector*)
X = Inverse [RzRot[−zRot]]. RcamTilt [cT, cR]. RcamRoll [cR]. Rphi [phi]
    . RzRot[−zRot]. a[ct];

(*Solve 3 Times, using all possible combinations of vector components*)
S = FindRoot [{X[[3, 1]] == A1[[3, 1]], X[[1, 1]] == A1[[1, 1]]},
             {cR, 0.1, −Pi, Pi}, {cT, 0.1, −Pi, Pi}, MaxIterations -> 500];
camR1 = S [[1, 2]];
camT1 = S [[2, 2]];
S = FindRoot[{X[[2, 1]] == A1[[2, 1]], X[[1, 1]] == A1[[1, 1]]}, {cR, 0.1, -Pi, Pi}, {cT, 0.1, -Pi, Pi}, MaxIterations -> 500];
camR2 = S[[1, 2]]; camT2 = S[[2, 2]];  
S = FindRoot[{X[[3, 1]] == A1[[3, 1]], X[[2, 1]] == A1[[2, 1]]}, {cR, 0.1, -Pi, Pi}, {cT, 0.1, -Pi, Pi}, MaxIterations -> 500];
camR3 = S[[1, 2]]; camT3 = S[[2, 2]];  
(* Calculate the Error in all 3 cases *)
a1 = Inverse[RzRot[-zRot]].RcamTilt[camT1, camR1].RcamRoll[camR1].Rphi[phi].RzRot[-zRot].a[ct];
d1 = Inverse[RzRot[-zRot]].RcamTilt[camT1, camR1].RcamRoll[camR1].Rphi[phi].RzRot[-zRot].d; b1 = Inverse[RzRot[-zRot]].RcamTilt[camT1, camR1].RcamRoll[camR1].Rphi[phi].RzRot[-zRot].b;
ae1 = ArcCos[A1[[1, 1]]*a1[[1, 1]] + A1[[2, 1]]*a1[[2, 1]] + a1[[3, 1]]*a1[[3, 1]]]*180/Pi

a2 = Inverse[RzRot[-zRot]].RcamTilt[camT2, camR2].RcamRoll[camR2].Rphi[phi].RzRot[-zRot].a[ct];
d2 = Inverse[RzRot[-zRot]].RcamTilt[camT2, camR2].RcamRoll[camR2].Rphi[phi].RzRot[-zRot].d; b2 = Inverse[RzRot[-zRot]].RcamTilt[camT2, camR2].RcamRoll[camR2].Rphi[phi].RzRot[-zRot].b;
ae2 = ArcCos[A1[[1, 1]]*a2[[1, 1]] + A1[[2, 1]]*a2[[2, 1]] + a2[[3, 1]]*a2[[3, 1]]]*180/Pi
\[ a_3 = \text{Inverse} [RzRot[-zRot]]. RcamTilt[camT3, camR3]. RcamRoll[camR3]. Rphi[phi]. RzRot[-zRot]. a[ct] ; \]
\[ d_3 = \text{Inverse} [RzRot[-zRot]]. RcamTilt[camT3, camR3]. RcamRoll[camR3]. Rphi[phi]. RzRot[-zRot]. d \;
\]
\[ b_3 = \text{Inverse} [RzRot[-zRot]]. RcamTilt[camT3, camR3]. RcamRoll[camR3]. Rphi[phi]. RzRot[-zRot]. b \;
\]
\[ a_3 = \text{ArcCos} [A1[[1, 1]]*a_3[[1, 1]] + A1[[2, 1]]*a_3[[2, 1]] + a_3[[3, 1]]*a_3[[3, 1]]]*180 / \text{Pi} \]

(* Check to see that no solutions have an imaginary component, if they do throw them out*)

\[ \text{If} [\text{Im}[a_1] != 0 , a_1 = 180 , a_1 = a_1] ; \]
\[ \text{If} [\text{Im}[a_2] != 0 , a_2 = 180 , a_2 = a_2] ; \]
\[ \text{If} [\text{Im}[a_3] != 0 , a_3 = 180 , a_3 = a_3] ; \]

(* Choose best case, based on position*)

\[ \text{If} [a_1 <= a_2 , \]
\[ \quad \text{If} [a_1 <= a_3 , \text{camT=} camT1 ; \text{camR=} camR1 ; \text{Print} [1] , \text{camT=} camT3 ; \text{camR=} camR3 ; \text{Print} [3]] , \]
\[ \quad \text{If} [a_2 <= a_3 , \text{camT=} camT2 ; \text{camR=} camR2 ; \text{Print} [2] , \text{camT=} camT3 ; \text{camR=} camR3 ; \text{Print} [3]] \]
\]

(* Calculate best case vectors*)

\[ X = \text{Inverse} [RzRot[-zRot]]. RcamTilt[camT, camR]. RcamRoll[camR]. Rphi[phi]. RzRot[-zRot]. a[ct] \]
\[ d_1 = \text{Inverse} [RzRot[-zRot]]. RcamTilt[camT, camR]. RcamRoll[camR]. Rphi[phi]. RzRot[-zRot]. d \;
\]
\[ b_1 = \text{Inverse} [RzRot[-zRot]]. RcamTilt[camT, camR]. RcamRoll[camR]. Rphi[phi]. RzRot[-zRot]. b \;
\]
(* Calculate errors, using dot product *)

\[
\text{ArcCos}[A1[[1,1]]*X[[1,1]] + A1[[2,1]]*X[[2,1]] + A1[[3,1]]*X[[3,1]]]*180/\text{Pi}
\]

\[
\text{ArcCos}[d1[[1,1]]*d[[1,1]] + d1[[2,1]]*d[[2,1]] + d1[[3,1]]*d[[3,1]]]*180/\text{Pi}
\]

\[
\text{ArcCos}[b1[[1,1]]*b[[1,1]] + b1[[2,1]]*b[[2,1]] + b1[[3,1]]*b[[3,1]]]*180/\text{Pi}
\]

(* Final Solution is camR and camT *)
Appendix B

Mathematica Code for Finding Correction Angles in the 3 DOF System

(*camera vector*)
2 a[ct1_] = {Cos[ct1], {0}, {-Sin[ct1]}};

(*roll vector*)
4 b = {{1}, {0}, {0}};

(*tilt vector*)
6 d = {{0}, {1}, {0}};

(*Yaw Vector*)
8 e = {{0}, {0}, {1}};

(*Initial camera tilt*)
ct = 45*Pi/180;

(*Rotation Matrix for Theta and Phi*)
14 RzRot[zRt_] = {{Cos[zRt], -Sin[zRt], 0}, {Sin[zRt], Cos[zRt], 0}, {0, 0, 1}};
Rphi[PHI_]={'{1,0,0}', '{0,Cos[PHI],-Sin[PHI]}', '{0,Sin[PHI],Cos[PHI]}'};

alpha=0;
beta=Pi/12;

(*If both alpha and beta are zero, theta and phi are undefined. We check for this, and then solve*)
If[alpha==0 && beta==0,theta=0;phi=0,
theta=ArcTan[Tan[alpha],Tan[beta]];
phi=ArcTan[1,Sqrt[Tan[alpha]^2+Tan[beta]^2]];

(*Angle between coord system 4 and axis of rotation*)
zRot=theta-Pi/4;

(*Find new yaw axis*)
camYawAxis=Rphi[phi].RzRot[-zRot].e;
ux=camYawAxis[[1,1]];
uy=camYawAxis[[2,1]];
uz=camYawAxis[[3,1]];

(*Calculate camera yaw after transformation in original coordinate system*)
tensor={
{ux^2,ux*uy,ux*uz},
{ux*uy,uy^2,uy*uz},
{ux*uz,uy*uz,uz^2}
};

skew={
{0,-uz,uy},
{uz,0,-ux},
{-uy,ux,0}
};
I1={
{1,0,0},
{0,1,0},
{0,0,1}
};

(*Camera Yaw Transformation in original coordinates*)
RcamYaw[GAM]=I1*Cos[GAM]+Sin[GAM]*skew+(1-Cos[GAM])*tensor;
Clear [ux, uy, uz, tensor, skew];

(*New camera roll axis*)
ux1=camRollAxis[[1,1]];
uy1=camRollAxis[[2,1]];
uz1=camRollAxis[[3,1]];

(*Calculate camera roll after transformation in original coordinate system*)
tensor[GAM]={
{ux1^2,ux1*uy1,ux1*uz1},
{ux1*uy1,uy1^2,uy1*uz1},
{ux1*uz1,uy1*uz1,uz1^2}};
skew[GAM]={
{0,-uz1,uy1},
{uz1,0,-ux1},
{-uy1,ux1,0}};
I1={
{1,0,0},
{0,1,0},
{0,0,1}};

(*Camera Roll Transformation in original coordinates*)
RcamRoll[cR,GAM]=I1*Cos[cR]+Sin[cR]*skew[GAM]+(1-Cos[cR])*tensor[GAM];

Clear[ux,uy,uz,tensor,skew];

(*New camera tilt axis.*)
cta=camTiltAxis[cR,GAM];
ux[cR,GAM]=cta[[1,1]];
uy[cR,GAM]=cta[[2,1]];
uz[cR, GAM] = cta[[3, 1]];  

tensor[cR, GAM] = {
    {ux[cR, GAM]^2, ux[cR, GAM]*uy[cR, GAM], ux[cR, GAM]*uz[cR, GAM]},
    {ux[cR, GAM]*uy[cR, GAM], uy[cR, GAM]^2, uy[cR, GAM]*uz[cR, GAM]},
    {ux[cR, GAM]*uz[cR, GAM], uy[cR, GAM]*uz[cR, GAM], uz[cR, GAM]^2}
};  

skew[cR, GAM] = {
    {0, -uz[cR, GAM], uy[cR, GAM]},
    {uz[cR, GAM], 0, -ux[cR, GAM]},
    {-uy[cR, GAM], ux[cR, GAM], 0}
};  

RcamTilt[cT, cR, GAM] = I1*Cos[cT] + Sin[cT]*skew[cR, GAM] + (1 - Cos[cT])*tensor[cR, GAM];  

(*Find Camera Vector and Roll Vector after full transformation*)  


(*Choose 3 of the 6 vector components and solve for 3 angles*)  

S = FindRoot[
    {X[[1, 1]] == A1[[1, 1]], X[[2, 1]] == A1[[2, 1]], D1[[1, 1]] == d[[1, 1]]},
    {cR, 0.1, -Pi, Pi}, {cT, 0.1, -Pi, Pi}, {GAM, 0.1, -Pi, Pi}, MaxIterations -> 500, DampingFactor -> 6];  

camR = S[[1, 2]];  
camT = S[[2, 2]];  

GAMMA = S[[3, 2]];  

(*Calculate Transformed Vectors*)
\( X_1 = \text{Inverse} \left[ \text{RzRot} \left[ -\text{zRot} \right] \right] \cdot \text{RcamTilt} \left[ \text{camT}, \text{camR}, \text{GAMMA} \right] \cdot \text{RcamRoll} \left[ \text{camR}, \text{GAMMA} \right] \cdot \text{RcamYaw} \left[ \text{GAMMA} \right] \cdot \text{Rphi} \left[ \phi \right] \cdot \text{RzRot} \left[ -\text{zRot} \right] \cdot a \left[ \text{ct} \right] \);

\( d_1 = \text{Inverse} \left[ \text{RzRot} \left[ -\text{zRot} \right] \right] \cdot \text{RcamTilt} \left[ \text{camT}, \text{camR}, \text{GAMMA} \right] \cdot \text{RcamRoll} \left[ \text{camR}, \text{GAMMA} \right] \cdot \text{RcamYaw} \left[ \text{GAMMA} \right] \cdot \text{Rphi} \left[ \phi \right] \cdot \text{RzRot} \left[ -\text{zRot} \right] \cdot d \);

\( b_1 = \text{Inverse} \left[ \text{RzRot} \left[ -\text{zRot} \right] \right] \cdot \text{RcamTilt} \left[ \text{camT}, \text{camR}, \text{GAMMA} \right] \cdot \text{RcamRoll} \left[ \text{camR}, \text{GAMMA} \right] \cdot \text{RcamYaw} \left[ \text{GAMMA} \right] \cdot \text{Rphi} \left[ \phi \right] \cdot \text{RzRot} \left[ -\text{zRot} \right] \cdot b \);

(* Calculate Error *)

\( a_{e1} = \text{ArcCos} \left[ A_1 \left[ \left[ 1,1 \right] \right] \cdot X_1 \left[ \left[ 1,1 \right] \right] + A_1 \left[ \left[ 2,1 \right] \right] \cdot X_1 \left[ \left[ 2,1 \right] \right] + A_1 \left[ \left[ 3,1 \right] \right] \cdot X_1 \left[ \left[ 3,1 \right] \right] \right] \cdot 180 / \pi \);

\( d_{e1} = \text{ArcCos} \left[ d_1 \left[ \left[ 1,1 \right] \right] \cdot d \left[ \left[ 1,1 \right] \right] + d_1 \left[ \left[ 2,1 \right] \right] \cdot d \left[ \left[ 2,1 \right] \right] + d_1 \left[ \left[ 3,1 \right] \right] \cdot d \left[ \left[ 3,1 \right] \right] \right] \cdot 180 / \pi \);

\( b_{e1} = \text{ArcCos} \left[ b_1 \left[ \left[ 1,1 \right] \right] \cdot b \left[ \left[ 1,1 \right] \right] + b_1 \left[ \left[ 2,1 \right] \right] \cdot b \left[ \left[ 2,1 \right] \right] + b_1 \left[ \left[ 3,1 \right] \right] \cdot b \left[ \left[ 3,1 \right] \right] \right] \cdot 180 / \pi \);

(* If error is nonzero, try different set of equations *)

\( \text{If} \left[ a_{e1} > 0.001 \quad || \quad d_{e1} > 0.001, \right. \)

\( S = \text{FindRoot} \left[ \left\{ X \left[ \left[ 1,1 \right] \right] = A_1 \left[ \left[ 1,1 \right] \right], X \left[ \left[ 3,1 \right] \right] = A_1 \left[ \left[ 3,1 \right] \right], D_1 \left[ \left[ 2,1 \right] \right] = d \left[ \left[ 2,1 \right] \right], \right\}, \{ \text{cR}, 0.1, -\pi, \pi \}, \{ \text{cT}, 0.1, -\pi, \pi \}, \{ \text{GAMMA}, 0.1, -\pi, \pi \}, \text{MaxIterations} \rightarrow 500, \text{DampingFactor} \rightarrow 6 \right]; \)

\( \text{camR} = S \left[ \left[ 1,2 \right] \right]; \)

\( \text{camT} = S \left[ \left[ 2,2 \right] \right]; \)

\( \text{GAMMA} = S \left[ \left[ 3,2 \right] \right]; \)

(* Calculate Final Vectors *)

\( X = \text{Inverse} \left[ \text{RzRot} \left[ -\text{zRot} \right] \right] \cdot \text{RcamTilt} \left[ \text{camT}, \text{camR}, \text{GAMMA} \right] \cdot \text{RcamRoll} \left[ \text{camR}, \text{GAMMA} \right] \cdot \text{RcamYaw} \left[ \text{GAMMA} \right] \cdot \text{Rphi} \left[ \phi \right] \cdot \text{RzRot} \left[ -\text{zRot} \right] \cdot a \left[ \text{ct} \right] \);

\( d_1 = \text{Inverse} \left[ \text{RzRot} \left[ -\text{zRot} \right] \right] \cdot \text{RcamTilt} \left[ \text{camT}, \text{camR}, \text{GAMMA} \right] \cdot \text{RcamRoll} \left[ \text{camR}, \text{GAMMA} \right] \cdot \text{RcamYaw} \left[ \text{GAMMA} \right] \cdot \text{Rphi} \left[ \phi \right] \cdot \text{RzRot} \left[ -\text{zRot} \right] \cdot d \)
\[ b1 = \text{Inverse} \left[ \text{R}z\text{Rot}\left[ -z\text{Rot} \right] \right] \cdot \text{R}c\text{amTilt}\left[ \text{camT, camR, GAMMA} \right] \cdot \text{R}c\text{amRoll}\left[ \text{camR, GAMMA} \right] \cdot \text{R}c\text{amYaw}\left[ \text{GAMMA} \right] \cdot \text{R}\phi\left[ \text{phi} \right] \cdot \text{R}z\text{Rot}\left[ -z\text{Rot} \right] \cdot b \]

\((* \text{Calculate Final Error}*)\)

\[ a_e1 = \text{ArcCos}\left[ a1 \left[ \left[ 1,1 \right] \right] \cdot x\left[ \left[ 1,1 \right] \right] + a1 \left[ \left[ 2,1 \right] \right] \cdot x\left[ \left[ 2,1 \right] \right] + a1 \left[ \left[ 3,1 \right] \right] \cdot x\left[ \left[ 3,1 \right] \right] \right] \cdot 180/\text{Pi} \]

\[ d_e1 = \text{ArcCos}\left[ d1 \left[ \left[ 1,1 \right] \right] \cdot d\left[ \left[ 1,1 \right] \right] + d1 \left[ \left[ 2,1 \right] \right] \cdot d\left[ \left[ 2,1 \right] \right] + d1 \left[ \left[ 3,1 \right] \right] \cdot d\left[ \left[ 3,1 \right] \right] \right] \cdot 180/\text{Pi} \]

\[ b_e1 = \text{ArcCos}\left[ b1 \left[ \left[ 1,1 \right] \right] \cdot b\left[ \left[ 1,1 \right] \right] + b1 \left[ \left[ 2,1 \right] \right] \cdot b\left[ \left[ 2,1 \right] \right] + b1 \left[ \left[ 3,1 \right] \right] \cdot b\left[ \left[ 3,1 \right] \right] \right] \cdot 180/\text{Pi} \]

\((* \text{Final Solution is camR, camT, and GAMMA}*)\)