Stabilization of a Tumbling Quadcopter UAV
Implementing and Testing of Geometric Control on ArduPilot Firmware

Submitted in partial fulfillment of the requirements
of the degree of

Master of Technology

by

Arjun Sadananda
(Roll No. 213236002)

Supervisors:

Prof. Ravi N Banavar
Prof. Kavi Arya

Systems & Control Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
2023

Declaration

I declare that this written submission is original and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented, fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited or from whom proper permission has not been taken when needed.

Date:

Arjun Sadananda

Roll No. 213236002

Contents

Chapter 1

Introduction

1.1 Background

The project aims to implement and test the nonlinear geometric controller described in Lee et al. (2010a) and Lee et al. (2010b)*(Control of Complex Maneuvers for a Quadrotor UAV using Geometric Methods on SE(3) - Taeyoung Lee, Melvin Leok, and N. Harris McClamroch)* on a Quadcopter Unmanned Aerial Vehicle (UAV). The control objective chosen is to "Stabilise a Tumbling Quadcopter UAV". Stabilise here means to achieve zero angular and linear velocity. Tumbling here means to initialize the UAV with "large" non-zero angular and linear velocity.

The present version of ArduPilot Firmware ArduPilotDevTeam (????b) is already capable of doing this to some extent. Therefore additionally the aim of the project also includes a comparison of the "performance" of the geometric controller and the present version of ArduPilot Firmware (4.4.x).

ArduPilot is one of the most widely used open-source flight controller firmware for autonomous drones. (Betaflight and iNav are the most widely used flight controller firmware in

the FPV (First-Person View) and drone racing communities, which also use a similar controller at the core). ArduPilot Firmware uses an advanced implementation of PID controller to achieve a wide variety of "flight modes" to achieve varying levels of autonomy. At the core of all the advanced "flight modes", lies the attitude controller, where a P controller converts the Euler angle errors into desired rotation rates followed by a PID controller to convert the rotation rate error into high-level motor commands. ArduPilotDevTeam (????a)

The first step in introducing Geometric Control in the firmware will be to introduce a new attitude controller described as ATTITUDE CONTROLLED FLIGHT MODE in Lee et al. (2010a) with $R_d(t) = e \in SO(3)$ in ArduPilot. But this only achieves zero angular velocity and allows the drone to drift with non-zero linear velocities. Therefore the next phase will be to introduce the VELOCITY CONTROLLED FLIGHT MODE described in Lee et al. (2010a) with $v_d(t) = 0 \in \mathbb{R}^3$. Finally, an Altitude Control (or partial POSITION CONTROLLED FLIGHT MODE) can be introduced to additionally achieve a desired altitude. The three flight modes limited to the specific commands described above correspond to three flight modes implemented in ArduPilot named: STABILIZE, FLOWHOLD, and ALTHOLD modes (partial POSHOLD) respectively. The final mode that we shall integrate Geometric Control into is what is known as THROW mode in ArduPilot.

1.2 Motivation

Geometric control is concerned with the development of control systems for dynamic systems evolving on nonlinear manifolds that cannot be globally identified with Euclidean spaces. By characterizing the geometric properties of nonlinear manifolds intrinsically, geometric control techniques provide unique insights into control theory that cannot be obtained from dynamic models represented using local coordinates. The proposed controller in Lee et al. (2010a) exhibits the following unique features:

- It guarantees almost global tracking features of a quadrotor UAV as the region of attraction almost covers the attitude configuration space SO(3). Such global stability analysis on the special Euclidean group of a quadrotor UAV is unprecedented(2010).
- It is coordinate-free. Therefore, it completely avoids singularities, complexities, discon-

tinuities, or ambiguities that arise when using local coordinates or quaternions.

Therefore the minimum expectation in the "stabilization of tumbling quadcopter UAV" problem is to have a "larger" domain of attraction as compared to the existing ArduPilot Firmware. It will also be interesting to compare the transient behaviors of the systems with the geometric controller and the present ArduPilot Controller. Additionally,

- Hybrid control structures between different tracking mode is robust to switching conditions due to the almost global stability properties. Therefore, aggressive maneuvers of a quadrotor UAV can be achieved in a unified way, without need for complex reachability analyses.

Which is good for future extension of this work to more complex problems.

1.3 Stage I Report Outline

Stage 1 of this project focused on the Literature Survey and Hardware & Software setup. The literature survey mainly concentrates on the controller proposed in [Lee et al. (2010a) and Flight Controller Firmware documentation. This report summarises the work done so far in 2 more chapters that follow. Chapter 2 shall cover the theory or the review of literature on the Geometric Controller and the ArduPilot Controller. Chapter 3 shall cover the Hardware and Software setup for the implementation phase. Finally, Chapter 4 shall summarise and present the future work for this project.

This page was intentionally left blank.

Chapter 2

Review of Literature

2.1 Dynamical System

This section describes the mathematical model of the quadrotor UAV rigid body dynamics, defined on the configuration space SE(3). This is introduced in Lee et al. (2010a) as a basis for the analysis.

Constants:

Define:

$m \in \mathbb{R}$ the total mass

$d \in \mathbb{R}$ the "drone radius": distance of the center of each motor to the center of mass

$J \in \mathbb{R}^3$ the inertia matrix with respect to body-fixed frame

Configuration Manifold

The configuration of this quadrotor UAV is defined by the location of the center of mass and the attitude with respect to the inertial frame. Therefore, the configuration manifold is the special

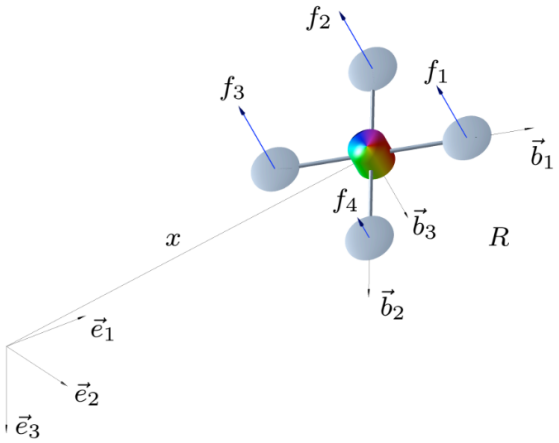


Fig. 1. Quadrotor model

Euclidean group $\mathsf{SE}(3) = \mathbb{R}^3 \rtimes SO(3)$, which is the semidirect product of $\mathbb{R}^3$ and the special orthogonal group $\mathsf{SO}(3) = \{R \in \mathbb{R}^{33} | R^T R = I, detR = 1\}$. Define:

x the position vector of the center of mass, (origin of b frame) w.r.t s frame.

R the rotation matrix from the body-fixed frame to the inertial frame.

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \in \mathbb{R}^3 \qquad R = \begin{bmatrix} | & | & | \\ b_1 & b_2 & b_3 \\ | & | & | \end{bmatrix} \in \mathsf{SO}(3)$$

b_i is the transformed body frame coordinates in NED frame coordinates {North-East-Down Frame}

Velocity - Tangent Space

Define:

v the velocity vector of the center of mass in inertial frame

Ω the angular velocity in body-fixed frame

$\mathfrak{so}(3) \cong T_e SO(3)$

$$\mathbf{v} = \begin{pmatrix} v^1 \\ v^2 \\ v^3 \end{pmatrix} \in \mathbb{R}^3 \qquad \Omega = \begin{pmatrix} \omega^1 \\ \omega^2 \\ \omega^3 \end{pmatrix} \in \mathbb{R}^3 \cong \mathfrak{so}(3) \ni \begin{bmatrix} 0 & -\omega^3 & \omega^2 \\ \omega^3 & 0 & -\omega^1 \\ \omega^2 & \omega^1 & 0 \end{bmatrix} = \hat{\Omega}$$

the *hat map* $\hat{}: \mathbb{R}^3 \to \mathfrak{so}(3)$ is defined by the condition that $\hat{x}y = x \times y$ for all $x, y \in R$

Control Input Remapping (Kinematics)

This is also related to what is called "MOTOR MIXING" in AP_Motors Library of ArduPilot.

$$\begin{bmatrix} f \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -d & 0 & d \\ d & 0 & -d & 0 \\ -c_{\tau f} & c_{\tau f} & -c_{\tau f} & c_{\tau f} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

This mapping from thrust f_i to f and $M \in \mathbb{R}^3$ is invertible when $8c_{\tau f}d^2$ (which is true for all quadcopter UAVs). Therefore we use f and $M \in \mathbb{R}^3$ as the control inputs.

Equations of Motion

The equations of motion of the quadrotor UAV can be written as

$$\dot{x} = v$$
$$m\dot{v} = mge_3 = fRe_3$$
$$\dot{R} = R\hat{\Omega}$$
$$J\dot{\Omega} + \Omega \times J\Omega = M$$

Linear Dynamical System

To do a theoretical analysis of the ArduPilot's Controller it would be a good idea to also look at the model of Quadcopter UAV on $(x, y, z, \psi, \theta, \phi) \in \mathbb{R}^6$ using Euler (Tait Bryan) Angles described in:

Castillo Garcia et al. (2023)*Stabilization of a Mini Rotorcraft with Four Rotors By Pedro Castillo, Rogelio Lozano, and Alejandro Dzul*

Which is a paper on experimental implementation and comparison of LQR and nonlinear control laws (described in Teel (1992)*Global stabilization and restricted tracking for multiple integrators with bounded controls - Andrew R. Teel*). This is not summarised here, since it is not important in the Stage 1 Report.

2.2 Defining Flight Modes

Since the quadrotor UAV has four inputs, it is possible to achieve asymptotic output tracking for at most **four quadrotor UAV outputs**. The quadrotor UAV has **three translational and three rotational degrees of freedom**; it is not possible to achieve asymptotic output tracking of both the attitude and position of the quadrotor UAV. This motivates us to introduce several flight modes. Each flight mode is associated with a specified set of outputs for which exact tracking of those outputs define that flight mode.

Geometric Controller Flight Modes The three flight modes considered in the paper [Lee et al. (2010a) are:

- Attitude controlled flight mode: the outputs are the attitude of the quadrotor UAV and the controller for this flight mode achieves asymptotic attitude tracking.
- Position controlled flight mode: the outputs are the position vector of the center of mass of the quadrotor UAV and the controller for this flight mode achieves asymptotic position tracking.
- Velocity controlled flight mode: the outputs are the velocity vector of the center of mass of the quadrotor UAV and the controller for this flight mode achieves asymptotic velocity tracking.

ArduPilot Flight Modes

Copter has 25 flight built-in flight modes, 10 of which are regularly used. ArduPilotDevTeam (????a) The table shows the ArduPilot Flight modes that are relevant to the objective of this project (and two others are listed here just because they are interesting, Flip and Acro)

Symbol Definition

- \- Manual control
- \+ Manual control with limits & self-level
- s Pilot controls climb rate
- A Automatic control

Mode	Alt Ctrl	Pos Ctrl	Summary
Stabilize	-	+	Self-levels the roll and pitch axis
Alt Hold	s	+	Holds altitude and self-levels the roll & pitch
FlowHold	s	A	Position control using Optical Flow
Loiter	s	s	Holds altitude and position, uses GPS for movements
PosHold	s	+	Like loiter, but manual roll and pitch when sticks not centered
Simple			An add-on to flight modes to use pilot's view instead of yaw orientation
Throw	A	A	Holds position after a throwing takeoff
Flip	A	A	Rises and completes an automated flip
Acro	-	-	Holds attitude, no self-level

Table 2.1: ArduPilot Flight Modes

2.3 Controller Design

2.3.1 ATTITUDE CONTROLLED FLIGHT MODE

Control Objective

An arbitrary smooth attitude tracking command $R_d(t) \in SO(3)$ is given as a function of time. As a first approach to stabilization of a tumbling UAV, this can be simplified to track a constant $e \in SO(3)$. Note that this doesn't define any translational motion objective. The corresponding angular velocity command is obtained by the attitude kinematics equation:

$$\boxed{t \to R_d(t) \in SO(3)} \qquad \boxed{\hat{\Omega}_d = R_d^T \dot{R}_d}$$

For the first approach to stabilising a tumbling quadcopter, this would simplify to;

$$\boxed{R_d = e \in SO(3)} \qquad \boxed{\hat{\Omega}_d = 0}$$

Error Function

First, define the real-valued error function on $\mathsf{SO}(3) \times \mathsf{SO}(3)$:

$$\psi(R, R_d) = \frac{1}{2} tr[I - R_d^T R] \qquad \psi(R, I) = \frac{1}{2} tr[I - R]$$

This function is locally positive-definite about $R = R_d$ within the region where the rotation angle between R and R_d is less than 180$\circ$. This set can be represented by the sublevel set of ψ where $\psi < 2$ which almost covers SO(3).

When variation of R is represented as $\delta R = R\hat{\eta}$ for $\eta \in \mathbb{R}^3$ the derivative of the error function is given by

$$\mathbf{D}_R \Psi(R, R_d).R\hat{\eta} = -\frac{1}{2} tr[R_d^T R \hat{\eta}] = \frac{1}{2}(R_d^T R - R^T R_d)^\vee.\eta = e_R.\eta$$

We use $-\frac{1}{2} tr[\hat{x}\hat{y} = x^T y$. From this, the **Attitude Tracking Error** is chosen to be

$$\boxed{\mathbb{R}^3 \ni e_R = \frac{1}{2}(R_d^T R - R^T R_d)^\vee} \qquad \boxed{\mathbb{R}^3 \ni e_R = \frac{1}{2}(R - R^T)^\vee}$$

The *vee map* $\vee : \mathfrak{so}(3) \to \mathbb{R}^3$ is the inverse of the hat map.

It is an interesting observation that if we look at the exponential map from the Lie algebra $\mathfrak{so}(3)$ to its Lie group $\mathsf{SO}(3)$, and represent the rotation in terms of axis ω and angleθ as follows $R = I + sin(\theta)\hat{\omega} + (1 - cos(\theta)\hat{\omega}^2$. The error function is $1 - cos(\theta)$ and the attitude tracking error is $sin(\theta)$

The tangent vectors $\dot{R} \in \mathsf{T}_R\mathsf{SO}(3)$ and $\dot{R} \in \mathsf{T}_{R_d}\mathsf{SO}(3)$ cannot be directly compared since they lie in different tangent spaces. We transform $\dot{R}_d$ into a vector in $\mathsf{T}_R\mathsf{SO}(3)$, and we compare it with $\dot{R}$ as follows $\dot{R} - \dot{R}_d(R_d^T R)$. Therefore we choose the **Angular Velocity Tracking Error** as

$$\boxed{\mathbb{R}^3 \ni e_\Omega = \Omega - R^T R_d \Omega_d} \qquad \boxed{\mathbb{R}^3 \ni e_\Omega = \Omega}$$

We can show that e_Ω is the angular velocity of the rotation matrix R_d^T, represented in the body-fixed frame, since $d/dt(R_d^T R) = (R_d^T R)\hat{e_\Omega}$

Geometric Attitude Tracking Controller

$$\boxed{M = -k_R e_R - k_\Omega e_\Omega + \Omega \times J\Omega - J(\hat{\Omega} R^T R_d \Omega_d - R^T R_d \dot{\Omega}_d)}$$

$$\boxed{M = -k_R e_R - k_\Omega e_\Omega + \Omega \times J\Omega}$$

In this attitude-controlled mode, it is possible to ignore the translational motion of the quadrotor UAV; consequently, the reduced model for the attitude dynamics are given by using the controller expression in the Rotational Kinematics and Dynamics equations of motion. The paper states the result that $(e_R, e_\Omega) = (0,0)$ is an exponentially stable equilibrium of the reduced closed loop dynamics. An estimate of the domain of attraction is obtained for which the quadrotor attitude lies in the sublevel set $L_2 = \{R \in \mathsf{SO}(3) | \phi(R, R_d) < 2\}$ for a given R_d. More explicitly, the attitudes that lie outside of the region of attraction are of the form $exp(\pi \hat{s}) R_d$ for some $s \in \mathsf{S}^2$. Since they comprise a two-dimensional manifold in the three-dimensional SO(3), we claim that the presented controller exhibits almost global properties in SO(3).

The corresponding ArduCopter Flight mode is: ***STABILIZE MODE*** where the thrust is left to the pilot. But, this is not exactly true, because in this mode the pilot provides roll and pitch angles (w.r.t to yaw orientation) and yaw rate instead of yaw angle. There is another mode in ArduCopter called ***SIMPLE MODE*** which defines the desired roll and pitch in the pilot/global frame. ArduCopter's controller is discussed in Section 2.4.

Additionally, we could implement an Altitude Controller to track a desired altitude x_{3d}, as follows:

$$f = \frac{k_x(x_3 - x_{3d}) + k_v(\dot{x_3} - x_{3d}) + m\dot{g} - m\ddot{x_{3d}}}{e_3 . R e_3}$$

But before we get to position control the first objective for this problem statement is to stabilise the velocity. Therefore instead of next looking at the POSITION CONTROLLED FLIGHT MODE we are interested in VELOCITY CONTROLLED FLIGHT MODE.

2.3.2 VELOCITY CONTROLLED FLIGHT MODE

Control Objective

An arbitrary velocity tracking command and the desired direction of the first body-fixed axis is given

$$\boxed{t \to v_d(t) \in \mathbb{R}^3} \qquad b_{1d}(t) \in \mathsf{S}^2$$

which simplifies to $\boxed{v_d = 0 \in \mathbb{R}^3}$ for stabilisation of a tumbling drone.

Velocity Tracking Error is given by $\boxed{e_v = v - v_d}$

The **nonlinear controller** for the velocity-controlled flight mode is given by

$$\boxed{f = (k_v e_v + mge_3 - m\dot{v_d}).Re_3} \quad \text{whichreducesto} \quad \boxed{f = (k_v e_v + mge_3).Re_3} \text{if} \quad v_d = 0$$

$$\boxed{M = -k_R e_R - k_\Omega e_\Omega + \Omega \times J\Omega - J(\hat{\Omega} R^T R_c \Omega_c - R^T R_c \dot{\Omega}_c)}$$

where k_v, k_R, k_Ω are positive constants, and following the prior definition of the attitude error and the angular velocity error

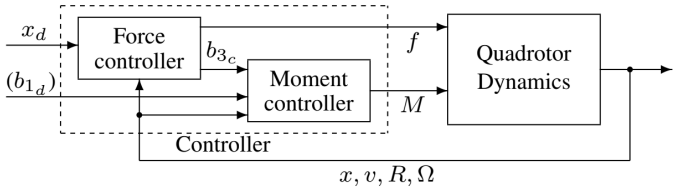
$$e_R = \frac{1}{2}(R_c^T R - R^T R_c)^\vee \qquad e_\Omega = \Omega - R^T R_c \Omega_c$$

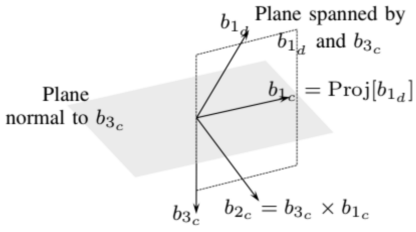
and $R_c(t) \in \mathsf{SO(3)} and \Omega_c \in \mathbb{R}^3$ are constructed as:

$$\boxed{R_c = [b_{1c}; b_{3c} \times b_{1c}; b_{3c}]}, \qquad \hat{\Omega}_c = R_c^T \dot{R}_c$$

where $b_{3c} \in \mathsf{S}^2$ is defined by

$$b_{3c} = \frac{-k_v e_v - mge_3 + m\dot{v}_d}{\| - k_v e_v - mge_3 + m\dot{v}_d \|} \qquad \boxed{b_{3c} = \frac{-k_v e_v - mge_3}{\| - k_v e_v - mge_3 \|}}$$





and $b_{1c} \in \mathsf{S}^2$ is chosen to be orthogonal to b_{3c}, thereby guaranteeing $R_c \in \mathsf{SO}^3$

$$b_{1c} = Proj[b_{1d}] - \frac{b_{3c} \times b_{1d}}{\|b_{3c} \times b_{1d}\|}$$

For stabilisation of Tumbling UAV we could choose $b_{1d} = e_1$ or $b_{1d} = e_2$ or to relax the control objective $b_{1d} = \frac{e_3 \times b_1}{\|e_3 \times b_1\|}$ or equivalently in this case $b_{1d} = b_1$.
We assume that $\|k_v e_v + mge_3 - m\dot{v_d}\| \neq 0$ and $\|mge_3 - m\dot{v_d}\| < B$. where B is a positive constant.

NOTE: The Stabilization version corresponds to the "FLOW HOLD" mode on ArduPilot. Additionally, the general mode is closer to LOITER MODE on ArduPilot than POSHOLD but we may be more interested in POSHOLD since we do not want to rely on GPS for position estimation.

2.4 ArduPilot Controller

2.4.1 Copter Attitude Control

The ArduPilot Developers' Documentation explains the controller in some depth.

Figure 2.1 is a high-level diagram showing how the attitude control is done in ArduCopter.

Figure 2.2 describes what is done for each axis. A P controller converts the angle error (the difference between the target angle and actual angle) into a desired rotation rate followed by a PID controller to convert the rotate rate error into a high level motor command. The "square root

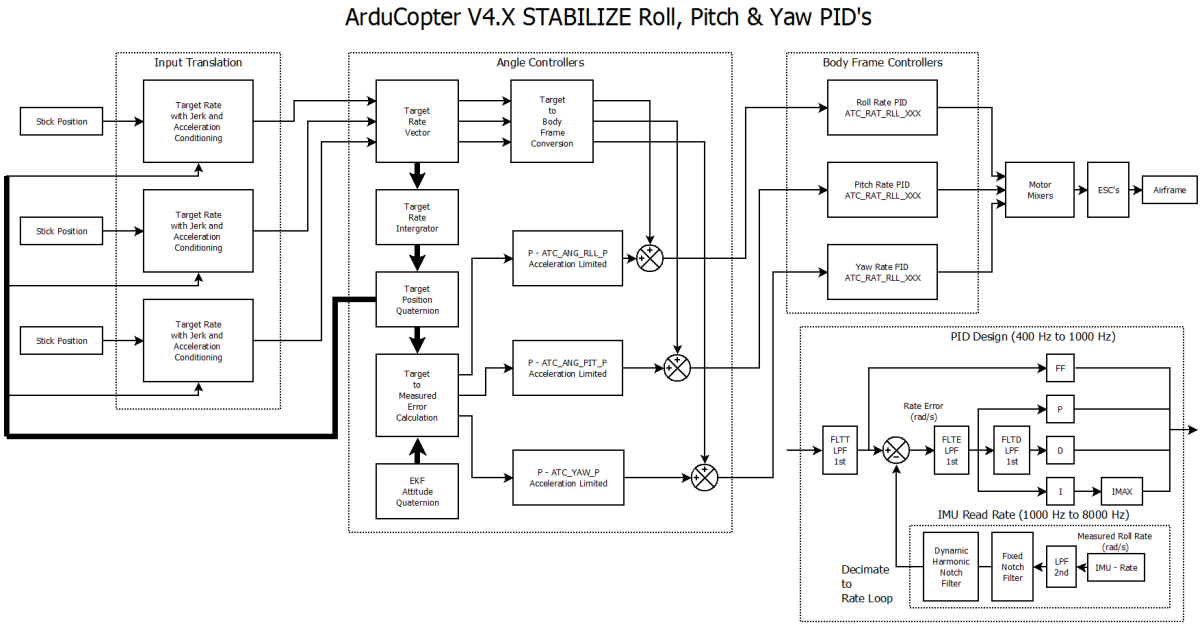


Figure 2.1: Test image include graphix.

controller" portion of the diagram shows the curve used with the angle control's P controller. Figure 2.3 shows the code path followed from pilot input down to PWM output.

NOTE: This is only part of the picture, the code exploration of the ArduPilot controller is ongoing and will be presented in the next stage of this project.

On every update (i.e. 400hz on Pixhawk, 100hz on APM2.x) the following happens:

- the top level flight-mode.cpp's "update_flight_mode()" function is called. which in turn calls the appropriate <flight mode>_run() function.
- the <flight mode>_run function is responsible for converting the user's input into a lean angle, rotation rate, climb rate, etc that is appropriate for this flight mode.
- the last thing the <flight mode>_run function must do is pass these desired angles, rates etc into Attitude Control and/or Position Control libraries. The AC_AttitudeControl library provides 5 possible ways to control the attitude of the vehicle, the most common 3 are described below.
  - angle_ef_roll_pitch_rate_ef_yaw(): this accepts an "earth frame" angle for roll and pitch, and an "earth frame" rate for yaw.
    NOTE: STABILIZE MODE

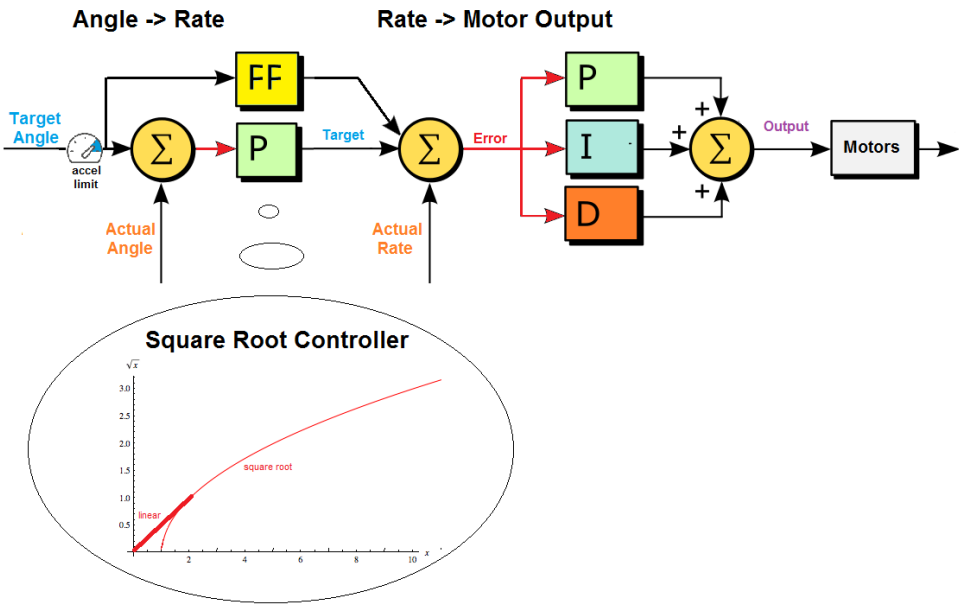
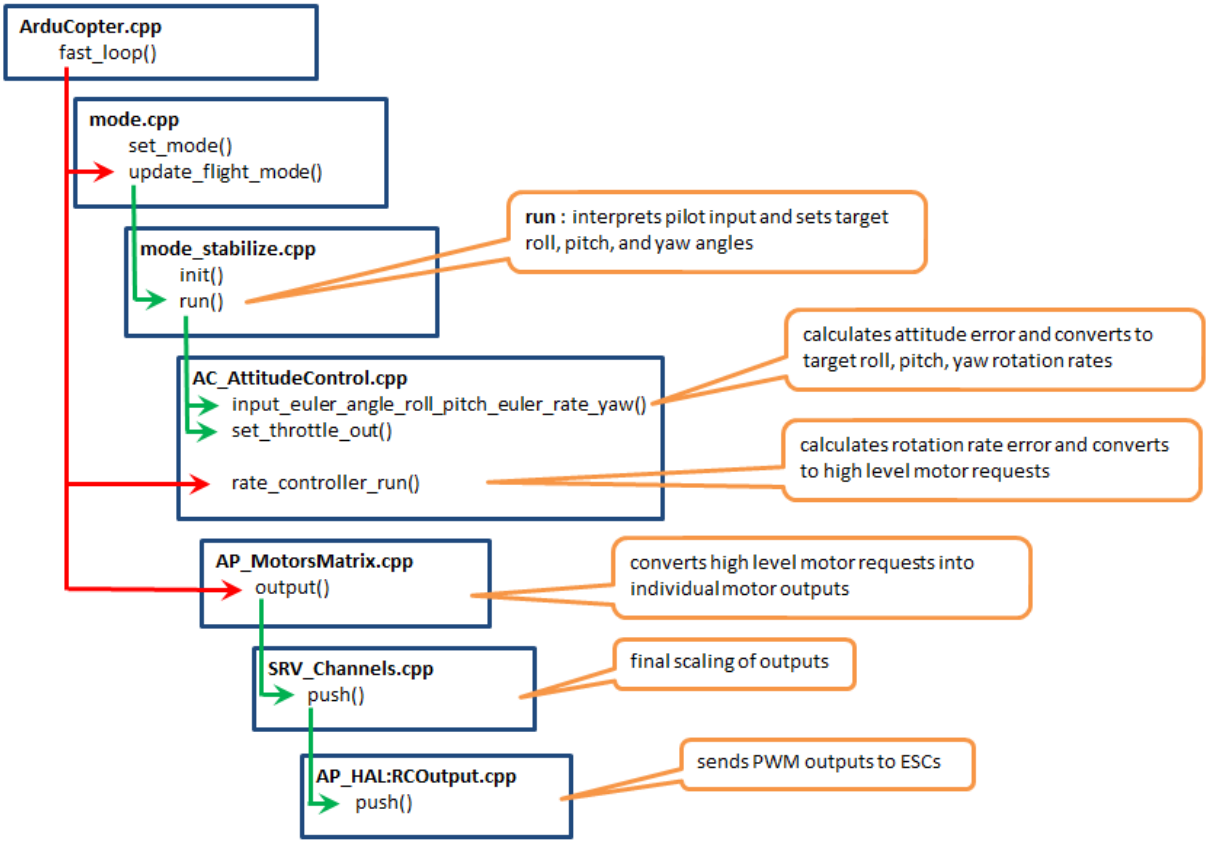


Figure 2.2: Test image include graphix.



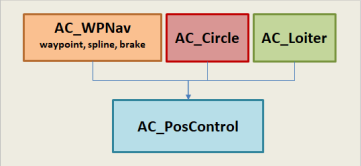


Figure 2.3: Test image include graphix.

  - angle_ef_roll_pitch_yaw(): this accepts “earth frame” angles for roll, pitch and yaw. NOTE: ATTITUDE CONTROLLED FLIGHT MODE
  - rate_bf_roll_pitch_yaw(): this accepts a “body frame” rate (in degrees/sec) for roll pitch and yaw.

- After any calls to these functions are made the AC_AttitudeControl::rate_controller_run() is called. This converts the output from the methods listed above into roll, pitch and yaw inputs which are sent to the AP_Motors library via it’s set_roll, set_pitch, set_yaw and set_throttle methods.

2.4.2 Copter Position Control and Navigation

AC_PosControl is the core library that is used by all the more autonomous modes that require position control.

- Separate interfaces for horizontal (X and Y axis) control and vertical (Z-axis) control. These interfaces are separated because some flight modes (like AltHold mode) only require the Z-axis controller

- Layered PID controllers are used

  - XY axis uses a Position P to convert position error to a target velocity. A velocity PID converts velocity error into a desired acceleration which is then converted to a desired lean angle which is then sent into the attitude control library.
  - The Z axis uses a Position P controller to convert position error to a target vertical velocity (aka climb rate). A Velocity P controller converts velocity error to a desired acceleration. An Acceleration PID converts acceleration error into a desired throt-

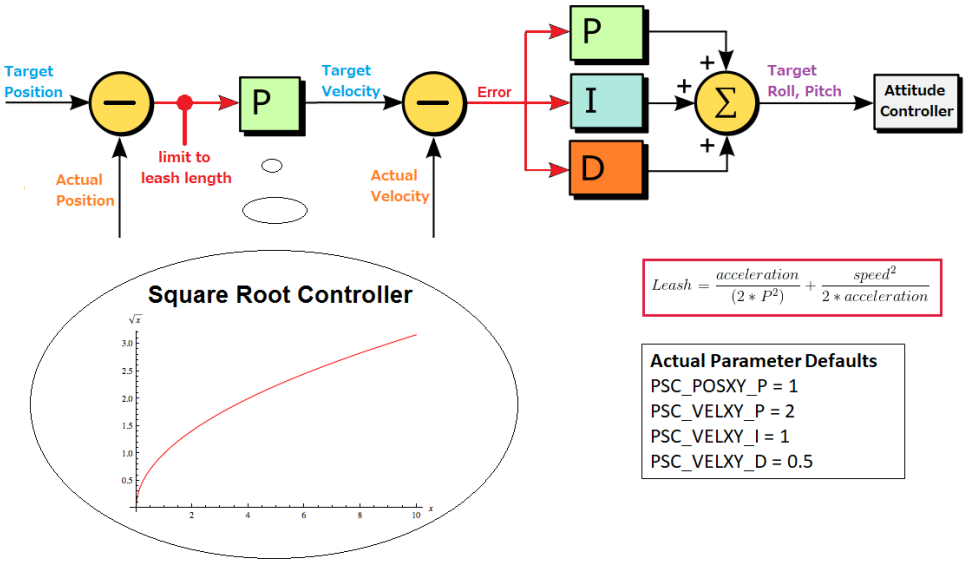


Figure 2.4: Test image include graphix.

tle which is then sent into the attitude control library (which mostly just passes it through to the low level motors library)

- AC_PosControl also includes a 3D velocity controller and a 3D Position+Velocity controller

Attached in the appendix is a code snippet and some comments about the state machine in throw_mode.cpp file. This is the mode that comes closest to achieving the control objective of this project.

This page was intentionally left blank.

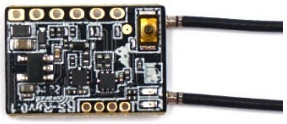
Chapter 3

Hardware & Software Setup

On the Hardware and Software setup side of the project, there have been a few iterations and learnings from each iteration.

First Build

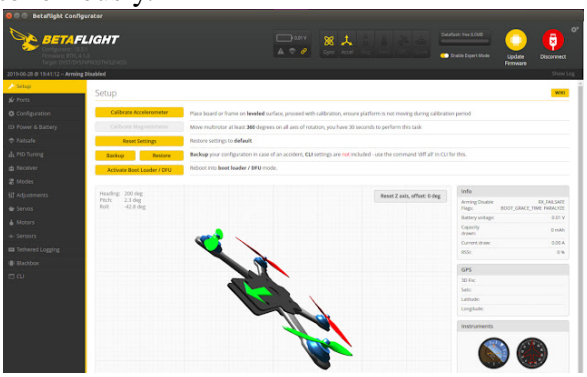
The first build was the minimal build. With the bare minimum requirements to achieve the desired result.




Components: GEP-12A-F4 Flight Controller 12A ESC, STM32F411, BMI270/42688, No Baro, 25.5×25.5mm, M2, GR1105 5000kV, Prop Length 3”, Pitch, 1.5”, 2 blade, Archer-RS, Transmitter FrSky X9 Lite

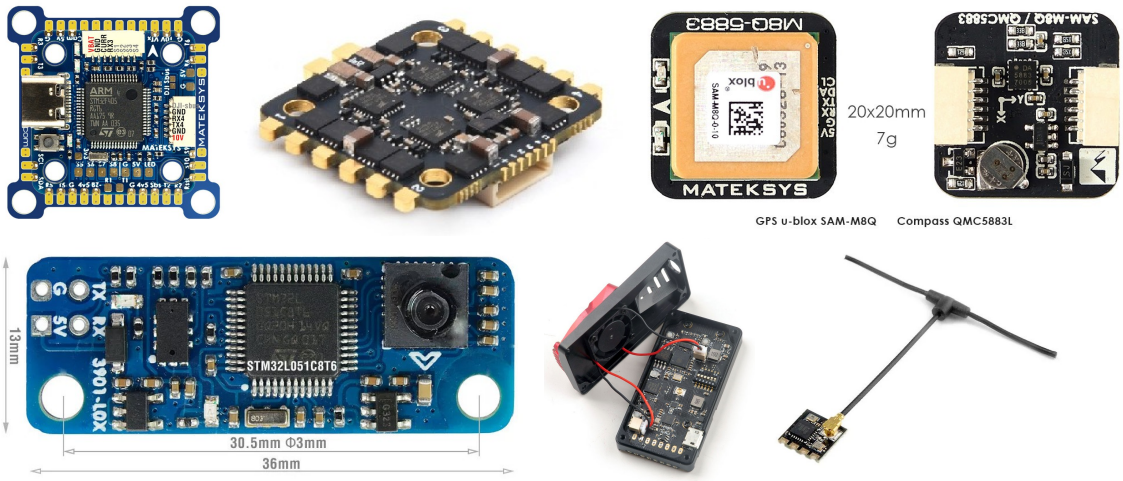
This drone was flashed with the Betaflight firmware and was flown in what Betaflight calls ANGLE MODE (same as STABILIZE MODE). With some piloting skills, this drone was already capable of stabilizing from tumbling initial conditions. However, efforts put into

further work on this drone were stopped early on, in order to incorporate more sensors required to achieve the control objective completely autonomously.

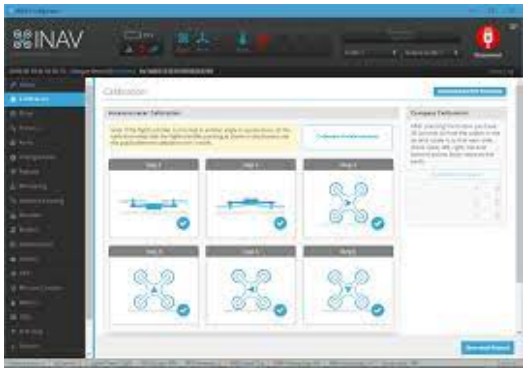


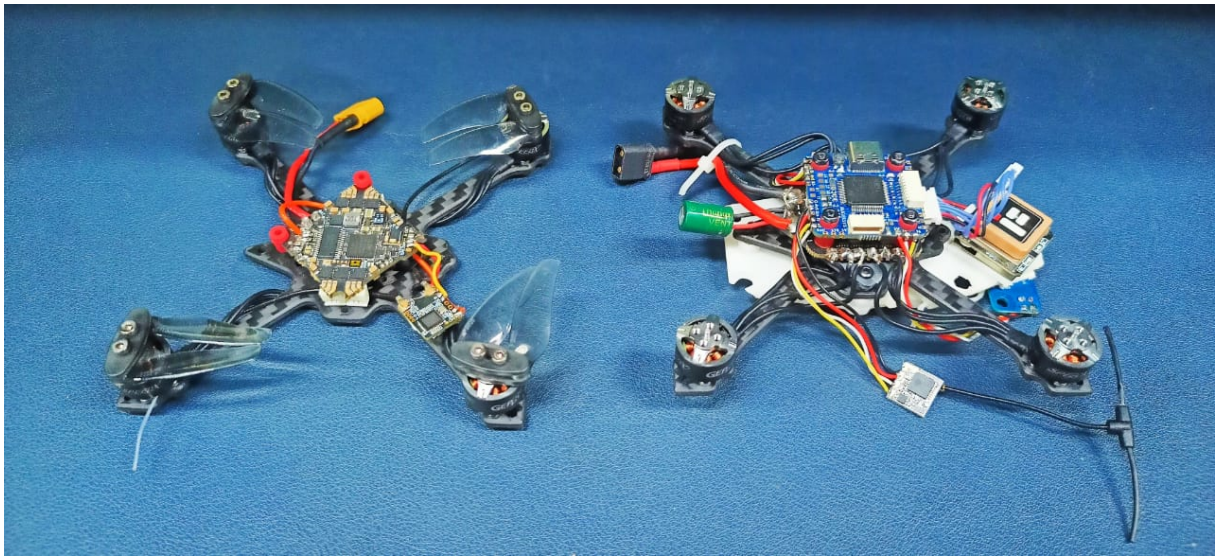
Second Build

The next iteration incorporated all the sensors one would need to achieve this goal with minimum/no piloting skills.

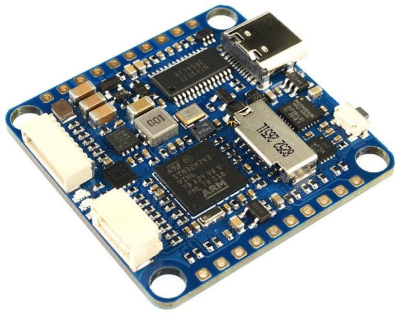


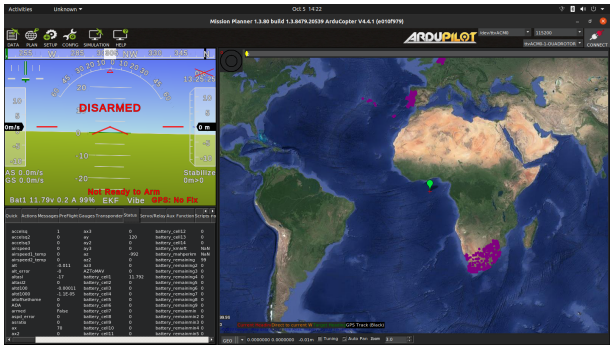
This drone was flashed with iNav firmware which supported more sensor interfacing than Betaflight. But on further investigation, it was decided to switch to ArduPilot for multiple reasons. One of the top reason was that despite appearing more intimidating at first, it has much better documentation for developers to learn the firmware code.

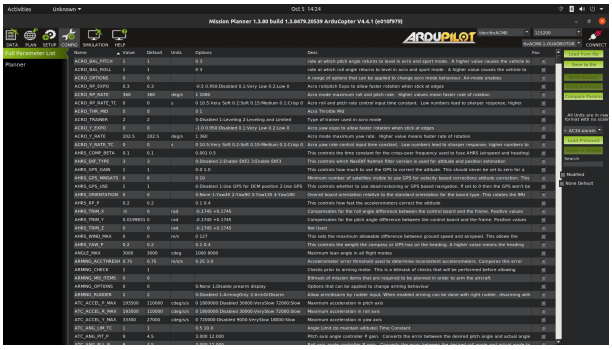




Ongoing Build The final iteration in progress is an upgrade of all the specs, better flight controller, better sensors, better mechanical design, etc.







This page was intentionally left blank.

Chapter 4

Summary and Future Work

To summarise, in Stage 1 of this project, we have done all the preparatory/background work needed to Implement and Test Geometric Control on ArduPilot Firmware. We looked at the dynamics of a quadcopter UAV on the SE(3) manifold. We went on to clearly define the control objective and the different flight modes. Finally, in the literature review of geometric control, we looked at the actual controller in two flight modes that we will need for the Stabilisation of a tumbling drone. Next, we looked at the existing approaches in ArduCopter to achieve the desired objective. We looked at the different flight modes in ArduPilot and briefly looked at the two core control modules of ArduCopter: "Attitude Control" and "Position Control". We also iterated through a few hardware and software setups to better achieve the desired objective.

The next part of this project involves completing the Hardware and Software setup for the third iteration and implementing the ArduCopter's Controller to achieve stabilisation of a tumbling quadcopter UAV.

Next, we shall start by implementing Geometric Attitude Control on hardware. Eventually, we shall create a new control flight mode in ArduPilot to implement the full objective of this project. We'll then need to devise and implement a strategy to "test"/compare the two controllers by choosing and logging the relevant data from the flight controller.

That will conclude the M.Tech. Project of Stabilising a Tumbling Quadcopter UAV using Geometric Control.

Appendix A

ArduPilot Firmware

A.1 mode_throw.cpp

Throw State Machine (Code not shown)

Throw_Disarmed - motors are off

Throw_Detecting - motors are on and we are waiting for the throw

Throw_Uprighting - the throw has been detected and the copter is being uprighted

Throw_HgtStabilise - the copter is kept level and height is stabilised about the target height

Throw_PosHold - the copter is kept at a constant position and height

```
switch (stage) {

case Throw_Disarmed:

// prevent motors from rotating before the throw is detected unless
   enabled by the user
if (g.throw_motor_start == PreThrowMotorState::RUNNING) {
```

```
7  motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
       GROUND_IDLE);
8  } else {
9  motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::SHUT_DOWN)
       ;
10 }
11
12 // demand zero throttle (motors will be stopped anyway) and continually
       reset the attitude controller
13 attitude_control->reset_yaw_target_and_rate();
14 attitude_control->reset_rate_controller_I_terms();
15 attitude_control->set_throttle_out(0,true,g.throttle_filt);
16 break;
17
18 case Throw_Detecting:
19
20 // prevent motors from rotating before the throw is detected unless
       enabled by the user
21 if (g.throw_motor_start == PreThrowMotorState::RUNNING) {
22 motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
       GROUND_IDLE);
23 } else {
24 motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::SHUT_DOWN)
       ;
25 }
26
27 // Hold throttle at zero during the throw and continually reset the
       attitude controller
28 attitude_control->reset_yaw_target_and_rate();
29 attitude_control->reset_rate_controller_I_terms();
30 attitude_control->set_throttle_out(0,true,g.throttle_filt);
31
32 // Play the waiting for throw tone sequence to alert the user
33 AP_Notify::flags.waiting_for_throw = true;
34
35 break;
36
37 case Throw_Wait_Throttle_Unlimited:
38
```

```
39 // set motors to full range
40 motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
       THROTTLE_UNLIMITED);
41
42 break;
43
44 case Throw_Uprighting:
45
46 // set motors to full range
47 motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
       THROTTLE_UNLIMITED);
48
49 // demand a level roll/pitch attitude with zero yaw rate
50 attitude_control->input_euler_angle_roll_pitch_euler_rate_yaw(0.0f, 0.0f
       , 0.0f);
51
52 // output 50% throttle and turn off angle boost to maximise righting
       moment
53 attitude_control->set_throttle_out(0.5f, false, g.throttle_filt);
54
55 break;
56
57 case Throw_HgtStabilise:
58
59 // set motors to full range
60 motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
       THROTTLE_UNLIMITED);
61
62 // call attitude controller
63 attitude_control->input_euler_angle_roll_pitch_euler_rate_yaw(0.0f, 0.0f
       , 0.0f);
64
65 // call height controller
66 pos_control->set_pos_target_z_from_climb_rate_cm(0.0f);
67 pos_control->update_z_controller();
68
69 break;
70
71 case Throw_PosHold:
```

```
72
73 // set motors to full range
74 motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
       THROTTLE_UNLIMITED);
75
76 // use position controller to stop
77 Vector2f vel;
78 Vector2f accel;
79 pos_control->input_vel_accel_xy(vel, accel);
80 pos_control->update_xy_controller();
81
82 // call attitude controller
83 attitude_control->input_thrust_vector_rate_heading(pos_control->
       get_thrust_vector(), 0.0f);
84
85 // call height controller
86 pos_control->set_pos_target_z_from_climb_rate_cm(0.0f);
87 pos_control->update_z_controller();
88
89 break;
90 }
```

References

ArduPilotDevTeam, ????a.
URL https://ardupilot.org/dev/

ArduPilotDevTeam, ????b. Arduplane, arducopter, ardurover, ardusub source. https://github.com/ArduPilot/ardupilot.git.
URL https://github.com/ArduPilot/ardupilot.git

Castillo Garcia, P., Lozano, R., Dzul, A., 10 2023. Stabilization of a mini-rotorcraft having four rotors. Vol. 3. pp. 2693 – 2698 vol.3.

Lee, T., Leok, M., McClamroch, N. H., 2010a. Control of complex maneuvers for a quadrotor uav using geometric methods on se (3). arXiv preprint arXiv:1003.2005.

Lee, T., Leok, M., McClamroch, N. H., 2010b. Geometric tracking control of a quadrotor uav on se (3). In: 49th IEEE conference on decision and control (CDC). IEEE, pp. 5420–5425.

Teel, A. R., 1992. Global stabilization and restricted tracking for multiple integrators with bounded controls. Systems & control letters 18 (3), 165–171.

This page was intentionally left blank.

Acknowledgments