Simulator Development of a Quadcopter

A project present to
The Faculty of the Department of Aerospace Engineering
San Jose State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Yibo Challinger

May, 2018

approved by

Dr. Long K.Lu
Faculty Advisor

ABSTRACT

SIMULATOR DEVELOPMENT OF A QUADCOPTER

by Yibo Challinger

A real-time high-fidelity software simulation is highly beneficial for controller development and testing. This paper focuses on the development of a dynamic model of a quadcopter using SymPy to derive equations of motion and generate C code. The vehicle used for flight testing is a self-assembled quadcopter. A series of tests are carried out on the flight vehicle to measure physical parameters including moments of inertia and electrical characteristics of motors. Simulation data is validated against flight data. This project utilizes open-source software, which makes the results completely accessible to the academic, hobby and industrial communities.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Prof. Long K. Lu for his continuous advising for this project.

Also, I would like to thank my husband for being a great partner in everything!

Table of Contents

CHAPTER

LIST OF TABLES

LIST OF FIGURES

Nomenclature

P	number of poles of a motor
λm	peak flux linkage due to permanent magnet
Id,Iq	d- and q- axis components of stator current
Ld,Lq	d- and q- axis stator self-inductance
R	motor resistance
ω	motor angular velocity (in rad/sec)
ι	motor torque
F	motor thrust
u	voltage input
Km	motor constant

CHAPTER 1

INTRODUCTION

1.1 Motivation

A quadcopter is a multirotor propelled by four rotors. Due to their simple mechanism, versatile application and major cost reduction compared to traditional helicopters, multirotors are making the transition from a niche hobby to a commercial tool, with numerous applications including videography, video surveillance, delivery, inspection, and precision agriculture. An accurate, reliable software simulation is highly beneficial for rotorcraft development. Wei et al. [1] captured the system identification of a quadcopter at a certain trimming condition using CIFER. Similarly, Schreurs et al. [2] derived the system identification of a quadcopter using test data. However, a linearized model of a quadcopter does not describe the full dynamics. This project is to create an accurate simulator based on the full-blown model of a quadcopter.

1.2 Literature Review

1.2.1 Quadcopter

Multirotors are first built and tested around 1900, such as Gyroplane No. 1 quadcopter by Jacques and Louis Brequet, Oehmichen No. 2 quadcopter, de Bothezat quadcopter [3]. However, when computers and good electric motors did not yet exist, multirotor lost its popularity to single main rotor helicopter in terms of natural stability and simplicity of mechanism [3].

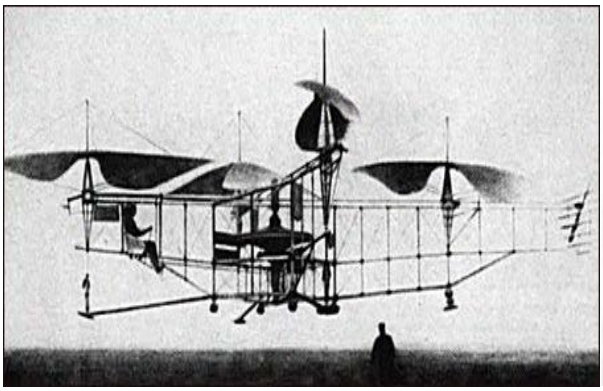

Figure 1.1: Oehmichen No.2 Quadcopter [3]

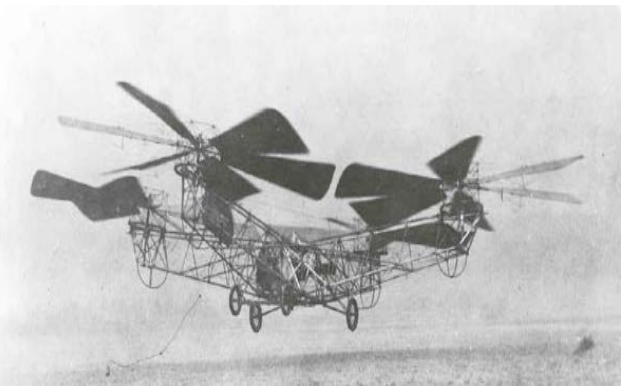

Figure 1.2: de Bothezat Quadcopter [3]

Owing to the dramatic advances of microprocessors, sensors and global positioning systems, modern multirotors are stabilized and controlled with the help of feedback control systems, which makes multirotors the most studied and utilized rotorcraft.

1.2.3 Methods of Orientation

There are many methods to describe the orientation of a rigid body with respect to a fixed coordinate system. The most widely used one is the Euler method because of its readability. However, modern methods of orientation for both aircraft and spacecraft are transitioning to the use of quaternions. Quaternions are redundant and therefore, must be re-normalized as numerical error builds up and less human readable than Euler angles. However, quaternions eliminate the

gimbal locks introduced by Euler method. Also, quaternions are more compact and more computationally efficient than rotation matrices. Figure 1.3 depicts one of the two singularity points in Euler method.

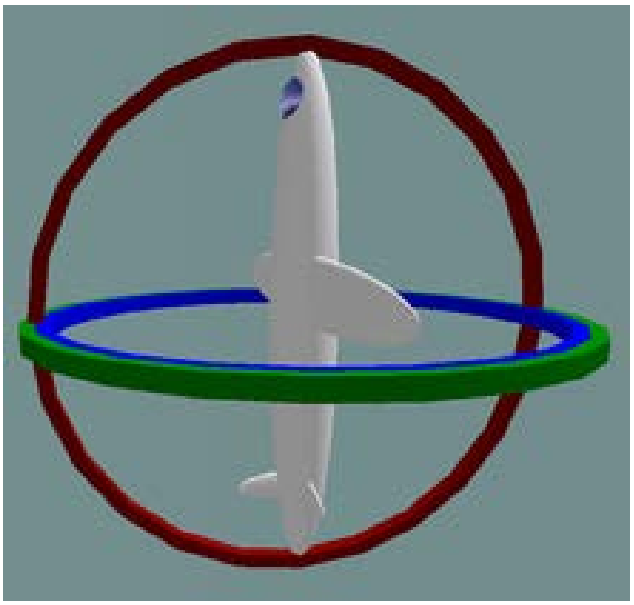

Figure 1.3: Illustration of a singularity in Euler method [17]

$$\boldsymbol{q} = \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix}$$

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(2(q_0q_1 + q_2q_3),\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \\ \operatorname{asin}(2(q_0q_2 - q_3q_1)) \\ \operatorname{atan2}(2(q_0q_3 + q_1q_2),\ q_0^2 + q_1^2 - q_2^2 - q_3^2) \end{bmatrix} \quad (1.1)$$

Equation 1.1 shows the conversion between Euler angles and quaternions. Quaternions q is in the form of $[q0, q1, q2, q3]^T$. [φ, θ, ψ] denote roll, pitch, yaw in Euler angles 3-2-1 sequence intrinsic, respectively [4].

1.2.4 Software

This project utilizes Sympy and PyDy for simulation and visualization. SymPy is a Python library providing symbolic math and computer algebra capabilities. SymPy also provides a mechanics package, which provides a clean application programmer interface (API) for programmatically describing the free-body diagram of a system of rigid bodies, and for applying Kane's method to derive symbolic equations of motion. The Kane's method API accepts a list of rigid body objects and a list of forces. In SymPy, RigidBody objects have a reference frame (a ReferenceFrame object), a center of mass, which is a Point object, a scalar mass, and a moment of inertia tensor, which is an Inertia object. Forces are applied either to points or to reference frames - torques. This API allows the creation of flexible and extensible simulations of mechanical systems. PyDy, short for Python Dynamics, extends SymPy output to numerical domain for system integration and visualization [5].

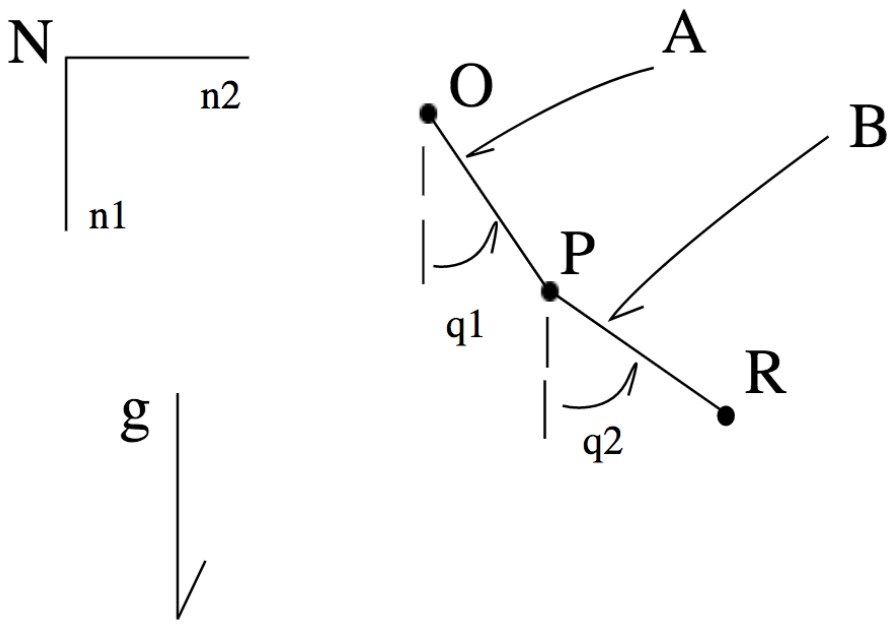

Figure 1.4: Double pendulum example [6]

A simple example to illustrate the use of Pydy is the double pendulum problem. Based on the geometry of the system, there are two body frames, A and B frame. q1 and q2 are used to define the locations of these two frames. Combined with tangential velocities of points P and R, the relative motion of the system can be fully described. Kane's method uses kinematic and kinetic equations to solve for mass and forcing matrices, which are used to obtain the equations of motion of the system.

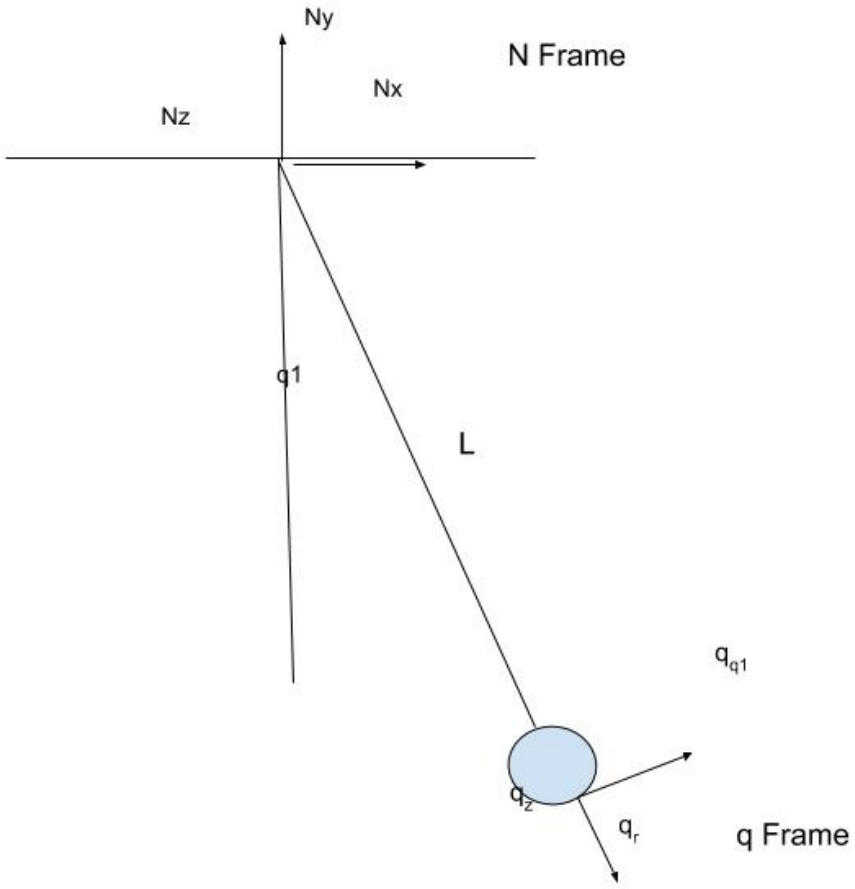

Figure 1.5: A single pendulum

The derivation in Table 1 is based on the setup in Fig. 1.4. The left column in Table 1 is the analytical solution from applying Golden Rule for vector differentiation, whereas the right column is the analytical solution derived from SymPy. The result from SymPy is consistent with mathematical solution for a single pendulum example.

Table 1.1: Validation of SymPy with a single pendulum

Derivation	
${}^{N}r^{Q} = L\ q_{r}$ ${}^{N}w^{Q} = q1d\ q_{z}$ ${}^{N}v^{Q} = {}^{Q}d/dt\ {}^{N}r^{Q} + {}^{N}w^{Q} \times {}^{N}r^{Q} = l\ q1d\ q_{q1}$ ${}^{N}a^{Q} = {}^{Q}d/dt\ {}^{N}v^{Q} + {}^{N}w^{Q} \times {}^{N}v^{Q} = l\ q1dd\ q_{q1} = lq1d^{2}\ q_{r}$ $F^{Q} = -mg \sin(q1)\ q_{q1} + mg \cos(q1)\ q_{r} - T\ q_{r}$ q1dd = -g*sin(q1)/l	from sympy import symbols from sympy.physics.mechanics import * q1 = dynamicsymbols('q1') q1d = dynamicsymbols('q1', 1) u1 = dynamicsymbols('u1') u1d = dynamicsymbols('u1', 1) l, m, g = symbols('l m g') N = ReferenceFrame('N') A = N.orientnew('A', 'Axis', [q1, N.z]) A.set_ang_vel(N, u1 * N.z) O = Point('O') P = O.locatenew('P', l * A.x) O.set_vel(N, 0) P.v2pt_theory(O, N, A) ParP = Particle('ParP', P, m) kd = [q1d - u1] FL = [(P, m * g * N.x)] BL = [ParP] KM = KanesMethod(N, q_ind=[q1], u_ind=[u1], kd_eqs=kd) (fr, frstar) = KM.kanes_equations(FL, BL) kdd = KM.kindiffdict() mm = KM.mass_matrix_full fo = KM.forcing_full qudots = mm.inv() * fo qudots = qudots.subs(kdd) qudots.simplify() mechanics_printing() mprint(qudots)
Results	

q1dot = u1 u1dot = -g*sin(q1)/l	q1dot = u1 u1dot = -g*sin(q1)/l

1.2.5 Motor Model

Ohm [8] demonstrates the measurements of motor parameters m, R and the derivation of the torque of a permanent magnet synchronous motor.

$$\tau = .75 * P (\lambda m * Iq + (Ld - Lq) Iq * Id) \quad (1.2)$$

Values of Iq, Id are retrieved from OpenMotorDrive. [7]

$$Iq = \frac{V}{\sqrt{3}} - \frac{\omega P \lambda m}{2 R} \quad (1.3)$$

$$Id = 0 \quad (1.4)$$

Substitute Eq. 1.3 and Eq. 1.4 in Eq. 1.2 to get torque and voltage relation in Eq. 1.5.

$$\tau = .75 * P * \frac{\lambda m}{R} * (\frac{V}{\sqrt{3}} - 7\omega * \lambda m) \quad (1.5)$$

1.2.6 Simulator

The simulator is intended to model the open loop dynamic behaviors of a quadcopter. Previous works on designing simulators of a quadcopters have been done in MATLAB by Gheorghiţă et al. [9] and Bresciani [10]. Bresciani's simulator also accounts for the actuator (motor) dynamics.

González et al. [11] designed a quadcopter simulator with linear quadratic regulator (LQR) controller for attitude stabilization in Python library. The open loop dynamics used are described in Eq. 1.5. The states in Eq. 1.5 are position, velocity, quaternion and angular

$$\dot{x} = \frac{d}{dt}\begin{bmatrix} p \\ \dot{p} \\ \boldsymbol{q} \\ \omega \end{bmatrix} = \begin{bmatrix} \dot{p} \\ \boldsymbol{q} \otimes \dfrac{F_{th}}{m} \otimes \boldsymbol{q}^* + \overline{g} \\ \dfrac{1}{2}\boldsymbol{q} \otimes \omega \\ J^{-1}\left(\tau - \omega \times J\,\omega\right) \end{bmatrix} \tag{1.5}$$

velocity, where position and velocity are with respect to the Earth frame and quaternions and angular velocity are with respect to the body frame. Inputs are thrust and torque, denoted as F_{th} and [11].

In full quaternion based attitude control for a quadcopter, Fresk and Nikolakopoulos [4] designed a non-linear P^2 controller in Fig. 1.4. The controller consists of an inner loop proportional gain P_ω for angular velocity and an outer loop proportional gain P_q for attitude control [4].

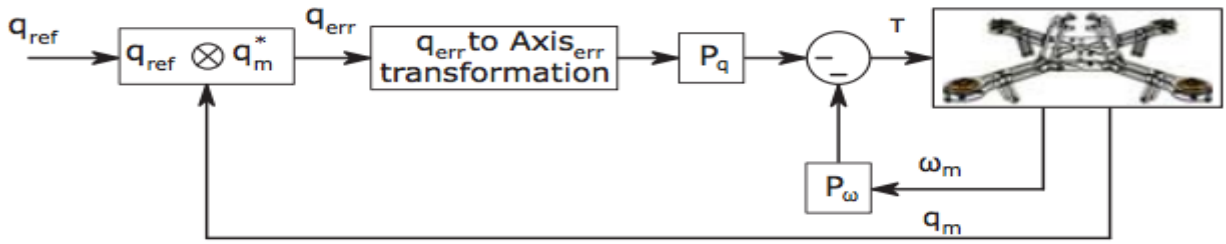

Figure 1.6: Block diagram of the full non-linear P^2 quaternion based control scheme [4]

1.2.7 High-fidelity modeling

Aerodynamic effects on a quadcopter are often oversimplified. The dynamic model of a quadcopter loses its accuracy where the impact of aerodynamic effects become significant. There are three main effects. Firstly, total thrust depends on the free stream velocity and angle of attack of the propellers. The second effect is the blade flapping introduced by the different thrust on each rotating blade of a propeller. The advancing blade experiences a larger velocity relative to

the free stream than the retreating blade and therefore the advancing blade produces more lift than the retreating blade. The rotor plane is tilted away from the direction of motion as in Fig. 1.5. The deflected thrust generates a longitudinal thrust. This thrust can produce a moment if the center of gravity of the vehicle does not align up with rotor plane vertically. Furthermore, the tilted blade generates a moment on the motor hub that is proportional to the stiffness of the propeller. The third effect is that the vehicle body interferes with the flow of the rotor [12].

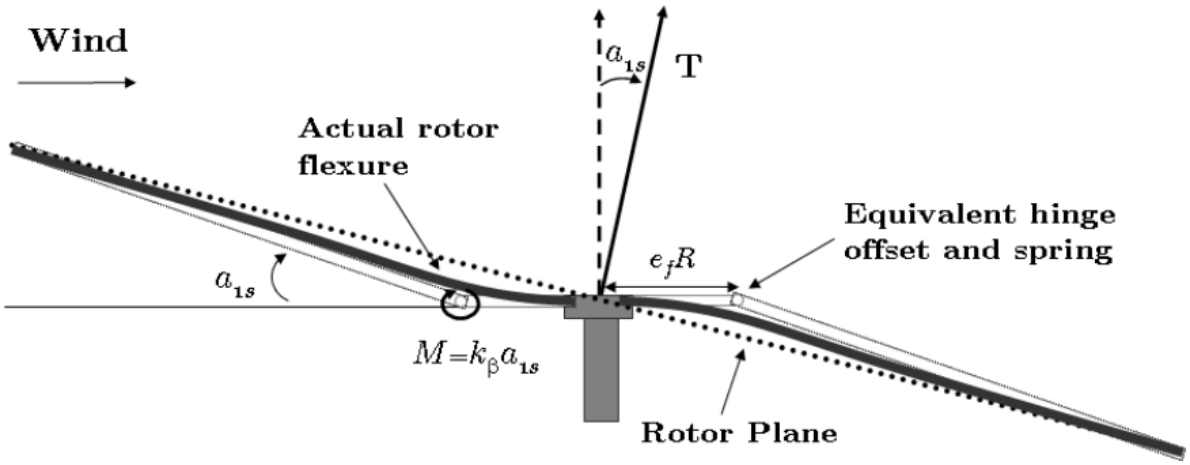

Figure 1.7: Rotor blade modelling blade flapping effect [12]

Another aerodynamic effect that is generally neglected in quadcopter dynamics is drag. If the propeller is divided into several sections, assuming drag coefficient is constant, then the drag force on each section is:

$$\Delta F = 0.5\ \rho\ C_D\ S\ V^2 \qquad (1.6)$$

The torque due to drag is:

$$\iota = \sum \Delta F * r = \sum 0.5\ \rho\ C_D\ S\ V^2 r = 0.5\ \rho\ C_D\ S\ (\omega\ r)^2 r = b\ \omega^2 \qquad (1.7)$$

Therefore, drag torque is proportional to angular velocity squared [13].

1.3 Problem Description

This project focuses on the development of a high-fidelity dynamic model of a multirotor and visualization in a 3D environment. This dynamics simulation is intended to accurately model actuator dynamics, aerodynamics, certain aeroelastic effects and ground interactions, allowing rapid development and testing of control algorithms. The proposed research will heavily utilize open-source software making the results completely accessible to the academic, hobby and industrial communities, which is one of the major and novel contributions of this project.

1.4 Methodology

Table 2.1: List of software used for modelling and simulation

Software	Description	Purpose
SymPy	Python symbolic math library	Derive symbolic equations of motion, generate C code.
PyDy	Python dynamics library based on SymPy	Integrate equations of motion in Python and visualize in web browser for testing purposes
Eigen	C++ linear algebra library	Used to solve equations of motion and integrate in real time or faster than real time

The symbolic equations of motion of the multirotor system are derived using SymPy, resulting in a set of nonlinear ordinary differential equations. C code is generated from the symbolic equations of motion, and a C++ library is employed to integrate the differential equations. An open-source, web-based 3D platform called CesiumJS will be used for

visualization in future work. A series of tests have been carried out on a flight vehicle to measure physical parameters such as moments of inertia and electrical characteristics of motors. Simulation results are compared with validation data from a real flight.

CHAPTER 2

DERIVATION OF MODEL

2.1 Description of bodies and frames

The proposed model of a quadcopter is comprised of thirteen rigid bodies - the quadcopter body, four rotors and eight propeller blades. Each body has a reference frame. Including Earth frame (referred as the N frame), there are thirteen frames used to describe the dynamics of a copter. Vehicle body frame is represented with quaternions. Each rotor frame is constrained such that it rotates about the z axis of the vehicle frame. The blade deflections only occur in the y-z plane of motors as in Fig. 2.1.

The 37 states in the model are quaternions, positions, linear velocities, angular velocities, motor angles, motor angular velocities, blade angles and blade angular velocities. Quaternions and motor angles are used for describing the rotation of body frame relative to N frame and the rotation of motor frame to body frame, respectively. Angular velocities and motor angular velocities describe the relative rotational motions of body frame and motor frame, respectively. Positions and linear velocities are described in N frame. Blade angles are used for describing the rotation of blade frame relative to motor frame. Blade angle velocities describe the relative rotational motion of blade frame. System inputs are motor torques in terms of voltages. Dynamics breaks into two parts, kinematics and kinetics. Kinematics is the geometry of motion described in mathematics, whereas kinetics is force geometry relations. Both parts are needed to calculate equation of motions. Specifically, the kinetics used in this model include gravity, thrusts, motor torques, and reaction torque from the motors. Kane’s method is used in SymPy to

solve equation of motions. Kane's method uses kinematics and kinetics equations of the system to calculate mass and forcing matrices.

The state vector is set to be [quat0 quat1 quat2 quat3 pos0 pos1 pos2 motor_theta0 motor_theta1 motor_theta2 motor_theta3 blade_theta0 blade_theta1 blade_theta2 blade_theta3 blade_theta4 blade_theta5 blade_theta6 blade_theta7 omega0 omega1 omega2 vel0 vel1 vel2 motor_omega0 motor_omega1 motor_omega2 motor_omega3 blade_omega0 blade_omega1 blade_omega2 blade_omega3 blade_omega4 blade_omega5 blade_omega6 blade_omega7]^T.

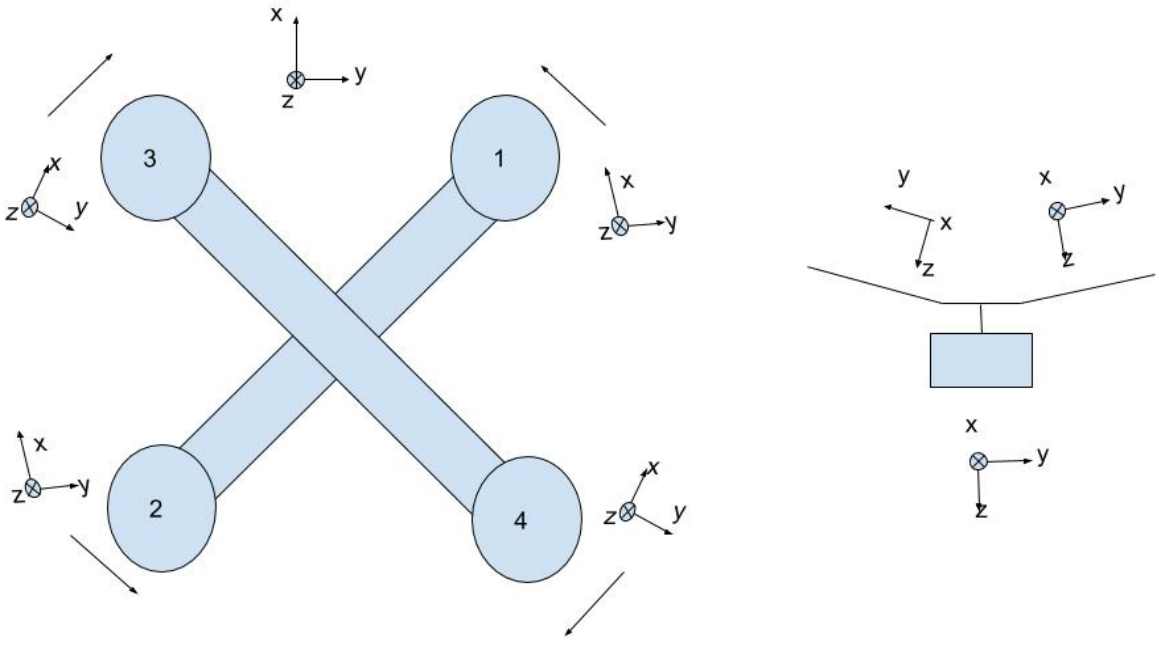

Figure 2.1: Quadcopter frame and motor frames (left), rotor frame (right)

The quadcopter setup is consistent with ardupilot convention in Fig. 2.1. In this setup, motors 1 and 2 rotate counterclockwise and motors 3 and 4 rotate clockwise. Figure 2.1 also

indicates the coordinates of the vehicle body frame and motor frames. The quadcopter has an X-shaped frame with equal arm lengths. Motors 1 and 2 spin counterclockwise, while motors 3 and 4 spin counterclockwise. Hence, clockwise props are used on motors 1 and 2, whereas counterclockwise props are used on motors 3 and 4 to produce positive thrust. Furthermore, motor frames are defined such that positive motor rotation produces positive thrust.

The model is intended to be accurate in all flight conditions. Therefore, axial and tangential speeds of blades are obtained to accurately model differential thrust on the advancing and retreating blades as blades flap up and down.

2.2 Propeller model description

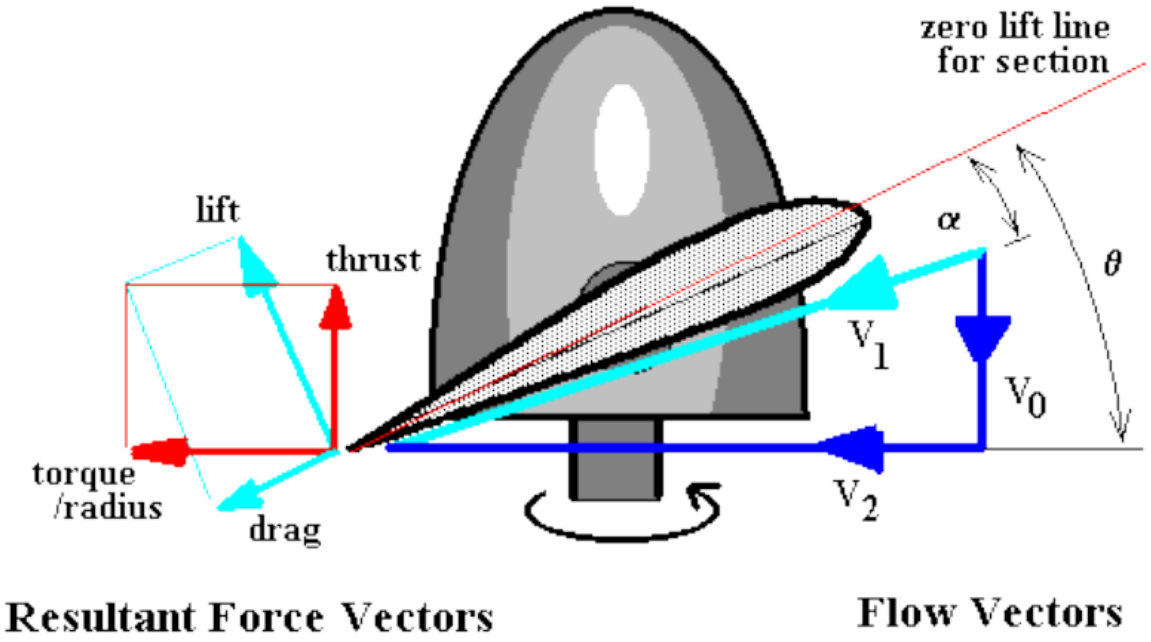

Figure 2.2: Aerodynamic forces on a blade [14]

V0 is axial flow velocity at propeller, V2 is angular flow velocity, and V1 is the sum of velocities. Parameter a is axial inflow factor, and b is angular inflow factor. is angle of attack and is blade pitch angle. Parameters α and θ represent angle of attack and propeller pitch angle, respectively.

$$V0 = V_{\infty}\ (1+a) \qquad (2.1)$$

$$V2 = \Omega\, r\, (1-b) \qquad (2.2)$$

$$V1 = \sqrt{V0^2 + V2^2} \qquad (2.3)$$

$$\alpha = \theta - \arctan\left(\frac{V0}{V2}\right) \qquad (2.4)$$

Thin airfoil theory is used for estimating the aerodynamic coefficients of a blade. Thrust and torque are in the axial and tangential direction of a rotor disk, respectively. Thrust (T) and torque (Q)/radius are the vector sum of lift and drag forces.

$$\Delta T = 0.5\ \rho\ V^2\ c\ (C_L\ \cos(\varphi) - C_D\ \sin(\varphi))\ dr \qquad (2.5)$$

$$\Delta Q = 0.5\ \rho\ V^2\ c\ (C_D\ \cos(\varphi) + C_L\ \sin(\varphi))\ r\ dr, \qquad (2.6)$$

$$\text{where } \varphi = \arctan\left(\frac{V0}{V2}\right) \qquad (2.7)$$

Conservation of momentum requires the changes in thrust and torque to be conserved. Therefore, they are also constrained by the following equations.

$$\Delta T = \rho\ 4\ \pi\ r\ V^2_{\infty}\ (1+a)\ a\ dr \qquad (2.8)$$

$$\Delta Q = \rho\ 4\ \pi\ r^3\ V_{\infty}\ (1+a)\ b\ \Omega\ dr \qquad (2.9)$$

Because these final forms of thrust and torque all contain unknown variable a and b, the system of equations are solved through iterations. Angle of attack determines the lift and drag. Lift and drag determine thrust and torque (2.5), (2.6). The conservation of momentum determines the inflow factor and swirl factor (2.8), (2.9). Inflow and swirl factors change the axial and angular velocities (2.1), (2.2), hence, angle of attack (2.4). [14]

The propeller blade in Fig. 2.2 flaps up and rotates forward. While the methodology of thrust and torque derivation using blade element conservation holds true for blades that flap up and down and rotate forward and backward, these equations are not generalized for implementation. Corrections have to be made for each scenario for a complete blade model.

Chapter 3

IMPLEMENTATION OF MODEL

3.1 Visualization in PyDy

Once the system is numerically solved using SymPy, PyDyViz is utilized to create a 3D visualization. PyDyViz is short for Python Dynamics Visualizations. It generates browser based simulations for PyDy framework.

PyDy visualization API has Python modules references, which include Shapes, ReferenceFrame, Cameras, Lights and Scene, and JavaScript functions references, which include Canvas, canvas/initialize.js, canvas/addObjects.js, and canvas/animate.js.

Figure 3.1 shows the starting location of a quadcopter in a simulation. The quadcopter is created with a total of 4 different types of visualization frames, namely, body frame, arm frame, motor frame and blade frame. The quadcopter uses north-east-down coordinates in Newtonian frame, PyDyViz uses red, green and blue axes to represent north-east-up, respectively.

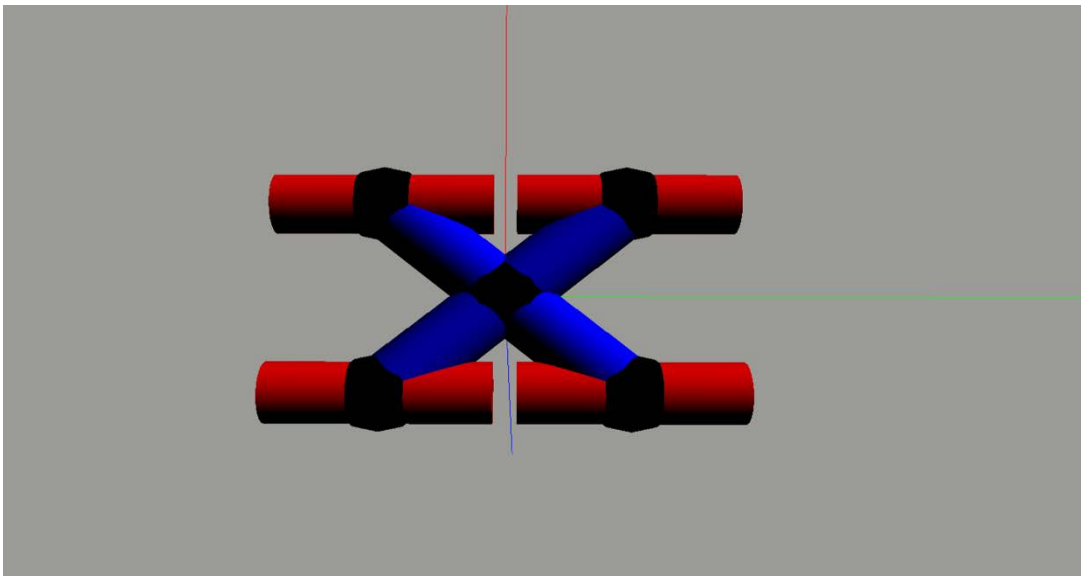

Figure 3.1: Visualization of a quadcopter displayed in a web browser

3.2 Measurement of Parameters

3.2.1 Mass property measurement

Mass of vehicle, rotor full assembly, and motor assembly without propeller are measured to be 3.08 kg, .117 kg, and .075 kg, respectively.

The moment of inertia measurement was performed on the fully assembled vehicle using a bifilar pendulum. Two parallel wires are used to suspend the vehicle. A small moment is applied parallel to the wires and the period of oscillation is measured. The moments of inertia are calculated using the following equation:

$$I = \frac{mgd^2}{4L\omega^2} \qquad (3.1)$$

where m is the mass of the measured vehicle, d is the distance between two filars, L is the length of a filar, and is the angular frequency calculated from the measured period of oscillation. [16]

Table 3.1 Moment of inertia measurement of the quadcopter

Quadcopter	Mass (kg)	Filar Distance (m)	Filar length (m)	Period (s)	Moment of inertia (kg*m2)
x axis	3.08	.54	1.33	1.32	0.073
y axis	3.08	.54	1.33	1.49	0.093
z axis	3.08	.45	1.33	2.30	0.154

3.2.2 Propeller measurement

Blade stiffness is calculated from applied torque and angular displacement. The applied torque (T) is proportional to the angular displacement (θ) of one side/end with respect to the other.

$$T = k * \theta \quad (3.2)$$

where k is stiffness. Displacement angle is 0.0043 for the torque of 0.22 N*m, so the stiffness of the blade is 51.8 N*m.

3.3 Overview of Eigen

To run the simulation in real-time, C code is generated with codegen. The nonlinear system is solved numerically with Eigen. Eigen is a C++ template library for linear algebra. Cholesky decomposition is chosen to solve the equations of motion. Table 3.2 compares the advantages and disadvantages of each decomposition in Eigen in terms of requirements, speed and accuracy.

Table 3.2: Decompositions in Eigen [15]

Decomposition	Method	Requirements on the matrix	Speed (small -to-me dium)	Speed (large)	Accuracy
PartialPivLU	partialPivLu()	Invertible	++	++	+
FullPivLU	fullPivLu()	None	-	- -	+++
HouseholderQR	householderQr()	None	++	++	+
ColPivHouseholderQR	colPivHouseholder Qr()	None	+	-	+++
FullPivHouseholderQR	fullPivHouseholder Qr()	None	-	- -	+++
CompleteOrthogonalD ecomposition	completeOrthogona lDecomposition()	None	+	-	+++
LLT	llt()	Positive definite	+++	+++	+
LDLT	ldlt()	Positive or negative semidefinite	+++	+	++
BDCSVD	bdcSvd()	None	-	-	+++
JacobiSVD	jacobiSvd()	None	-	- - -	+++

3.4 4th order Runge-Kutta Integration Method

4th order Runge-Kutta method is used for integrating equations of motion to get states. RK4 estimates the next value using the weighted average of four increments, where each increment is the product of time interval and an estimated slope specified by Eq. 3.6 to Eq. 3.9 [18]. The simulator uses an integration time step of 1e-4 seconds.

$$\frac{dy}{dx} = f(t,y) \qquad (3.3)$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (3.4)$$

$$t_{n+1} = t_n + h \quad (3.5)$$

$$k_1 = h\, f(t_n, y_n) \quad (3.6)$$

$$k_2 = h\, f\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \quad (3.7)$$

$$k_3 = h\, f\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \quad (3.8)$$

$$k_4 = h\, f(t_n + h, y_n + k_3) \quad (3.9)$$

Chapter 4

VALIDATION OF MODEL

4.1 Hardware setup

The vehicle used for flight testing is a self-assembled quadcopter. Table 4.1 lists all the components of the quadcopter. A series of tests were carried out on the flight vehicle to measure physical parameters including moments of inertia and electrical characteristics of motors.

Table 4.1 Quadcopter components

Component	Specifics
Frame	enRoute QC730
Motors	enRoute 5828-340KV
Motor drive	Prototype field-oriented control with UAVCAN interface
Propellers	18 inch diameter, pitch unspecified
Battery	Hobbyking Multistar 6-cell 10Ah Lithium Polymer
Telemetry	RFDesign RFD900u
RC Receiver	FRSky D4R-II
Autopilot	Hex Technologies Pixhawk 2 Cube running ArduCopter
GPS	Hex Technologies Here+

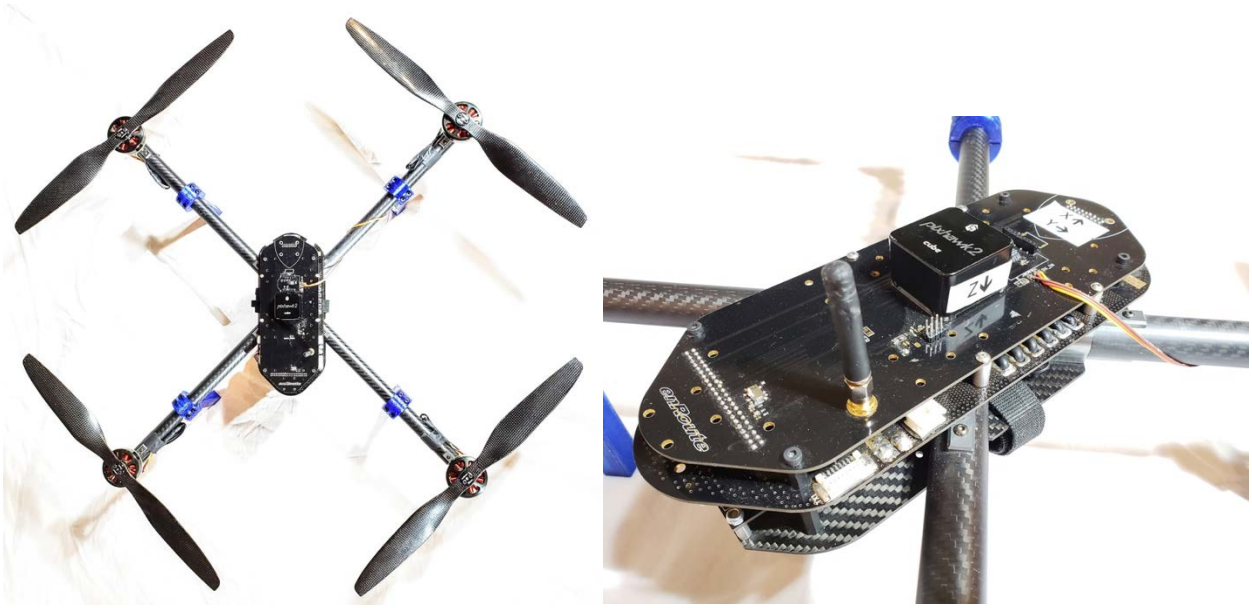

Figure 4.1: Quadcopter setup

Pixhawk 2 sensors include an Invensense MPU 9250 as main accel and gyro, ST Micro 16-bit gyroscope, ST Micro 14-bit accelerometer/compass (magnetometer), MEAS barometer. Accelerometers, gyroscopes and magnetometers are used to measure linear acceleration, rotational rate and magnetic field, respectively. Each of the three axes has one accelerometer, one gyroscope and one magnetometer to measure vehicle's movements in all three directions. Pixhawk 2 interfaces include 2x CANamong others. Motor speed controller provides motor speed feedback and input voltage over UAVCAN.

4.2 Test plan

The test plan for the quadcopter model includes the following maneuvers: hover, climbs and descents, roll, pitch, yaw doublets. Data logging will include rotor speed, input voltage from ESC, attitude, and angular velocity from Pixhawk, linear velocity from onboard GPS.

Dynamic parameters, lift and drag coefficients, and blade pitch are manually tuned to fit the flight test data. The propeller blade is cambered, so the blade pitch is slightly increased to account for absolute angle of attack. Lift coefficient slope is estimated with thin airfoil theory.

4.2 Hover Test

The vehicle was hovered. The motor rotation speeds and input voltages were averaged over the flight to be 252 rad/sec, 9.4 Volts. The model has motor speeds of 245 rad/sec under the same voltage inputs.

4.3 Climbs and Descents

The blade element momentum theory model does not converge correctly when descending, therefore blade element model is used with a fixed inflow velocity in the simulator. Flight testings are conducted at steady climb rate of 2, 4, 6 m/s. Motor speeds from the simulator are consistent with flight vehicle with a maximum of 6% uncertainty in Table 4.2.

Table 4.2: Comparisons of motor speed between vehicle and simulator

Climb rate	Motor speed, flight vehicle	Motor speed, simulator
0 m/s	252 rad/s	245 rad/s
2 m/s	265 rad/s	275 rad/s
4 m/s	280 rad/s	300 rad/s
6 m/s	335 rad/s	337 rad/s

4.4 Roll, Pitch, Yaw

The same motor input voltages are used in the simulator and flight testing for roll, pitch and yaw maneuvers for model validation. Motor speed outputs and angular velocities in all three directions are compared between the model and fight data in Fig. 4.2, Fig. 4.3, and Fig. 4.4.

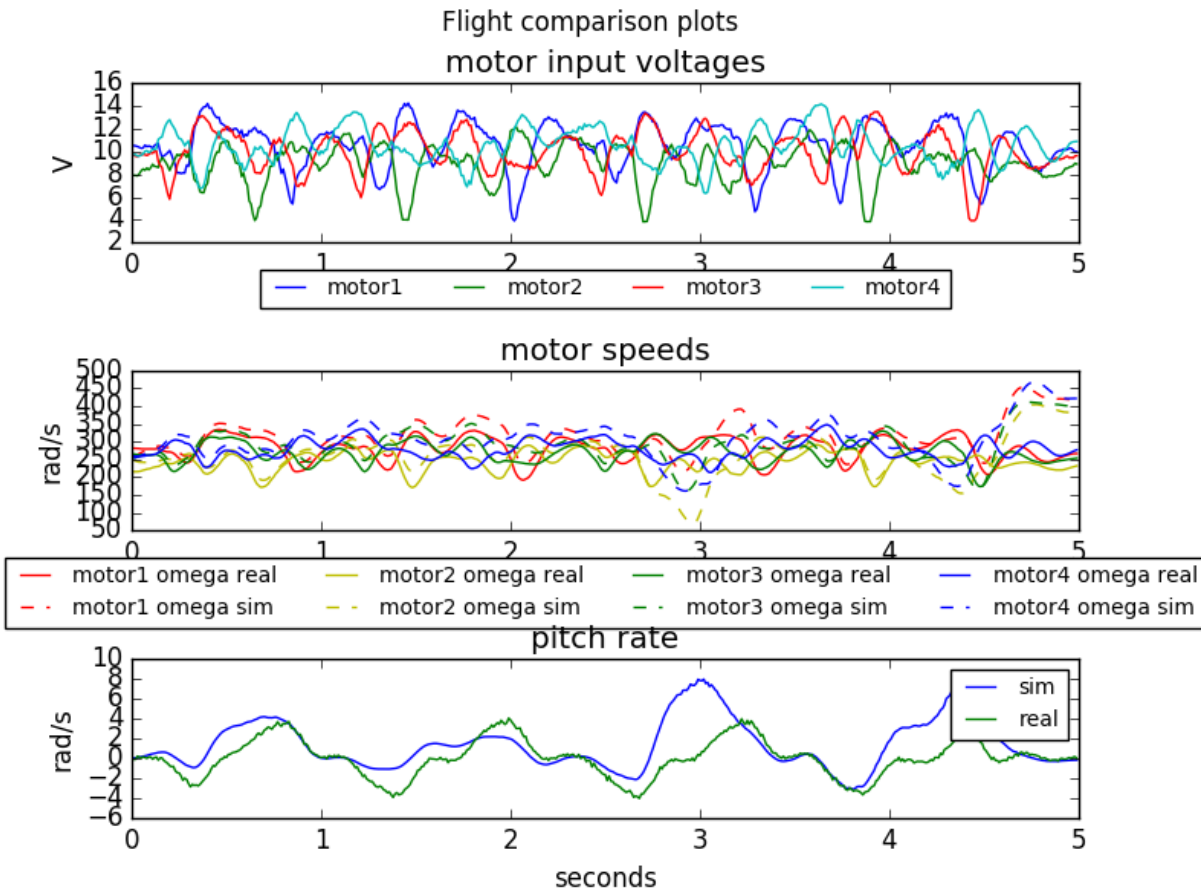

Figure 4.2: Pitch comparison

Figure 4.2 shows the motor input voltages, motor speeds between flight data and simulator and pitch comparison in a pitch maneuver. Motor speed comparisons plot validates the motor model used in the simulator. Pitch rates are consistent between vehicle model and flight data in general, although a disturbance can be seen at 3 seconds.

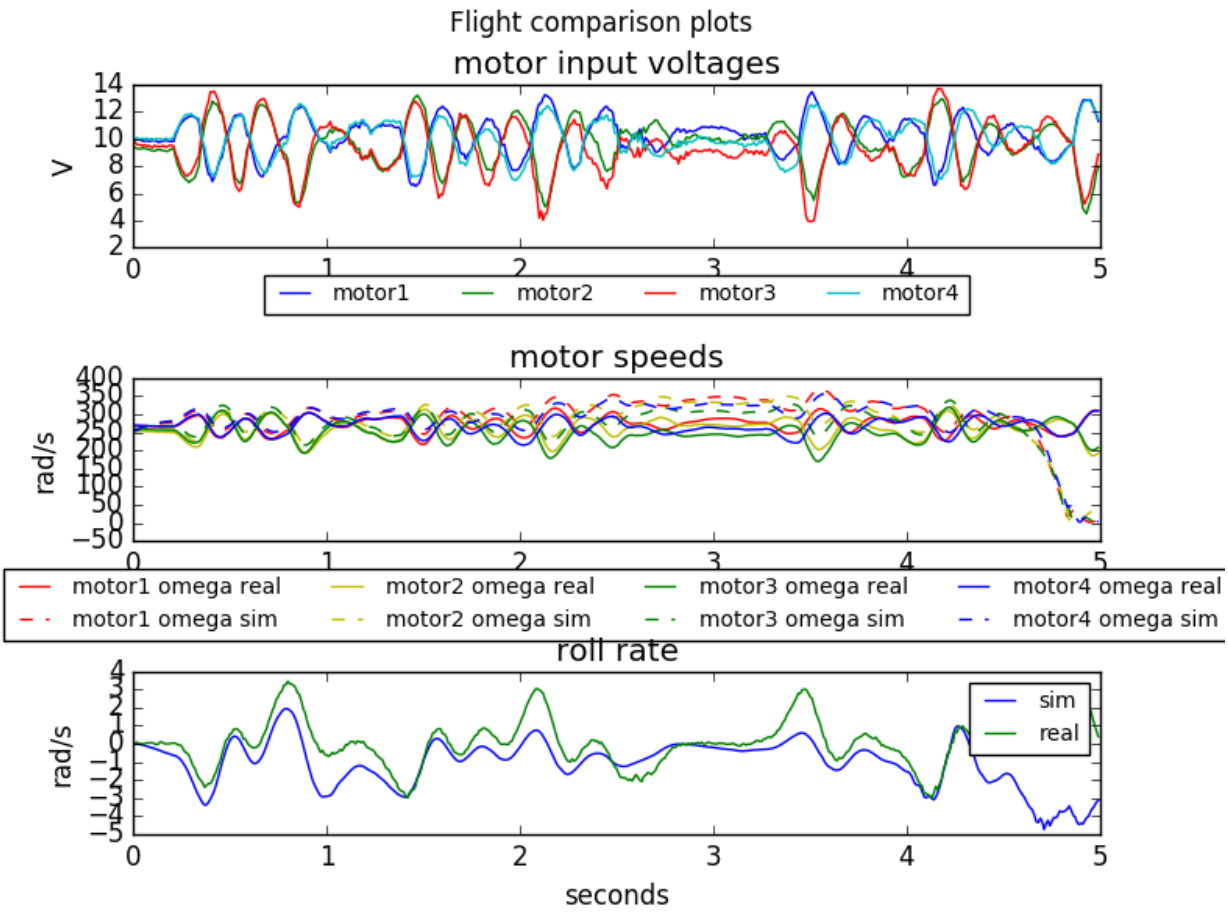

Figure 4.3: Roll comparison

Figure 4.3 demonstrates the same plots for roll testing. Motor speeds and roll rates are highly consistent under this maneuver.

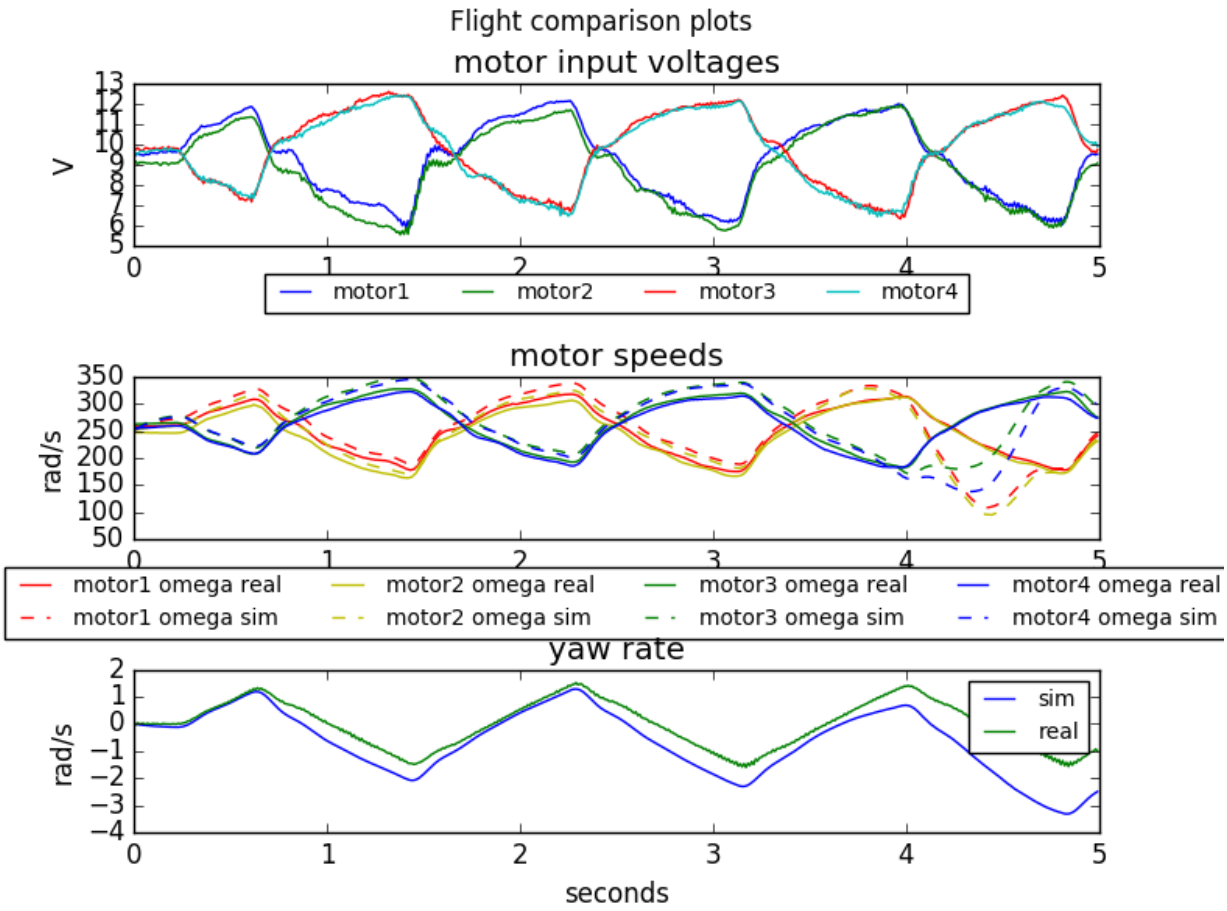

Figure 4.4: Yaw comparison

Similarly, Fig. 4.4 is for yaw testing and motor speeds and yaw rates are highly consistent.

Chapter 5

CONCLUSION & FUTURE WORK

This project explored the development of a real-time high-fidelity simulation, visualization and flight testing of an enroute quadcopter. The simulator developed in this project provides a platform for controller development and testing of quadcopters. All the softwares are open-source and available at https://github.com/yibochallinger/quadcoptersimulator.

In this project, an accurate dynamic model of a quadcopter was developed using SymPy and C code generation. The model includes mathematical modelling of permanent magnet synchronous motors and aerodynamic modelling of blade-flapping effects. A non-real time 3D visualization of a quadcopter was created with PyDyViz. Furthermore, flight test maneuvers, including hover, climbs & descents, pitch, roll and yaw, were conducted on an enroute quadcopter for model validation.

Future work will focus on parameter fitting with propeller thrust and torque measurements using a thrust stand and dynamometer, live 3D visualization in CesiumJS through websocket++ and integration into the Ardupilot open-source autopilot. In addition, a more generalized propeller model can also improve the overall accuracy of the simulation.

References

[1] Wei, W., Tischler, M., and Cohen, K., "System Identification and Controller Optimization of a Quadrotor UAV," *Proceedings of the AHS International's 71st Annual Forum and Technology Display*, Virginia Beach, VA, 2015.

[2] Schreurs, R. J. A. *et al*., "Open loop system identification for a quadrotor helicopter system," *2013 10th IEEE International Conference on Control and Automation (ICCA)*, Hangzhou, 2013, pp. 1702-1707.

[3] "History of Quadcopters and Multirotors," *KROSSBLADE AEROSAPCE* [online database], http://www.krossblade.com/history-of-quadcopters-and-multirotors/ [retrieved 28 Sep. 2017].

[4] Fresk, E., and Nikolakopoulos, G., "Full Quaternion Based Attitude Control for a Quadrotor," *European Control Conference*, July 17-19, 2013.

[5] "PyDy: Multibody Dynamics with Python," [online database], http://www.pydy.org [retrieved 20 September 2017].

[6] "A Double Pendulum Example," [online database], http://www.pydy.org/examples/double_pendulum.html [retrieved 13 December 2017].

[7] "OpenMotorDrive," [online database], https://github.com/OpenMotorDrive/openmotordrive/blob/master/src/motor.c [retrieved 1 May 2018].

[8] Ohm, D. Y., "Dynamic model of PM synchronous motors," Drivetech, Inc., Blacksburg, VA, May 2000.

[9] Gheorghiță, D., Vîntu, I., Mirea, L., and Brăescu, C., "Quadcopter Control System Modelling and Implementation," *International Conference on System Theory, Control and Computing (ICSTCC)*, October 14-16, 2015.

[10] Bresciani, T., "Modelling, Identification and Control of a Quadrotor Helicopter," A Master's thesis of the Department of Automatic Control, Lund University, Lund, Sweden, 2008.

[11] González, H. A., Escobar, J. C., and García, P. C. "Quadrotor Quaternion Control," *2015 International Conference on Unmanned Aircraft Systems*, June, 2015.

[12] Hoffmann, G. M., Huang, H., Waslander, S. L., and Tomlin, C. J., "Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment," *AIAA Guidance, Navigation and Control Conference and Exhibit*, Hilton Head, South Carolina, USA, 2007.

[13] Gibiansky, A., "Quadcopter dynamics, simulation and control," [online database], http://andrew.gibiansky.com/downloads/pdf/Quadcopter%20Dynamics,%20Simulation,%20and%20Control.pdf [retrieved 28 Sep. 2017].

[14] "Blade Element Theory for Propellers," *Aerodynamics for Students* [online database], http://www.aerodynamics4students.com/propulsion/blade-element-propeller-theory.php [retrieved 1 May 2018].

[15] "Linear algebra and decompositions," [online database], https://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html [retrieved 1 May 2018].

[16] Habeck, J., and Seiler, P., "Moment of Inertia Estimation Using a Bifilar Pendulum," A report of UMTC Undergraduate Research Presentations and Papers (UROP), University of Minnesota, Minnesota, USA, 2016.

[17] "Gimbal Lock," [online database], https://en.wikipedia.org/wiki/Gimbal_lock [retrieved 1 May 2018].

[18] Suli, E., and Mayers, D., "Runge-Kutta methods," *An Introduction to Numerical Analysis*, Cambridge University Press, Cambridge, 2003, pp. 325-329.