Team No. 212

Problem B

Quadcopter Stability in Wind

Abstract

In this article, we estimate the maximum windspeed that allows the quadcopter to stay within 20cm from the target. We use python to build a physical simulation system, a thrust control system that can control the quadcopter to react to the wind and maintain its stability and a wind simulator . By inputting two basic patterns of wind(periodic and noise) into our simulation with different magnitude, we get the average time that the quadcopter is able to stay within range. We find that periodic wind poses greater threats to safety operation than the dominant wind with noise.

Keywords: Quadcopter dynamics, Linear Control, Newton-Euler method, Controller design, Wind simulation, Stability analysis

Contents

1 Introduction

Quadcopter unmanned aerial vehicles (UAVs) are used widely both in industry and in academic research nowadays. Some applications with precise operation requires high stability of the platform.

Wind is the major disturbance source effecting the stability of the UAVs. The real wind usually has time-dependent magnitude and direction, which can easily threaten the safety of the equipment. Thus the maximum wind speed that allows safe operation is an important parameter to estimate the UAV performance.

In this article, we are going to develop a simulation method to determine the maximum wind speed that can allow the drone to stay within 20 cm of the target.

In Model section, the physics simulation system is first proposed, which models the dynamic of the drone given the wind. Then is the thrust controller, which generates the best response given the wind and the drone's current info. Finally, the wind simulator is proposed, which can both generate periodic or noised wind.

In Results section, we will first give the brief summary of our model's simulation result and give the maximum wind speed that allows safe operation. Then, the detailed data will be given with clear image illustration to validate our statement.

Finally,we will draw conclusions and discuss the advantages and and advantages of our model and give comments on how to improve the model.

2 Model

2.1 Problem Overview

In this article, we are going to estimate the maximum windspeed that allows the quadcopter to stay within 20cm from the target. We will first simplify the quadcopter dynamics model to reduce the calculation complexity in algorithm. Based on the rigid body dynamics, we then derive equations of motions and then use python to build a physical simulation system for the quadcopter. We then develop a thrust control system that can control the quadcopter to react to the wind and maintain its stability. By inputting basically two patterns of wind(periodic and noise) into our simulation with different magnitude, we run the simulation repeatly and get the average time that the quadcopter is able to stay within desired range. If the time is large enough, the corresponding windspeed is allowed for the safe operation. The maximum windspeed can be found accordingly.

Figure 1: Quadcopter overview.

2.2 Assumption and Definition

2.2.1 Quadcopter model simplification

The drone we used in our model has a total mass of 1.5kg, and is powered by four rotors which can each generate a thrust up to 7 N.

The major mass of the quadcopter is concentrated on the platform in the middle. Therefore, in order to simplify the dynamics equation of motion during the operation of the quadcopter, we assume that the platform in the center is a homogeneous sphere with a radius $r = 10cm$ and a mass $m = 1kg$. Four rotors are arranged in a square, each with a mass $m_r = 0.125kg$, and a distance of $d = 50cm$ from the sphere center. We assume that the four rods connecting the rotor to the central sphere are light enough that it can be ignored in the dynamics model. The model frame is showed in Figure 2.

As for staying within 20 cm, as our model, including the thrust control system algorithm, is rough, if the quadcopter is able to stay within 20 cm around the target for 30 s, it is believed that if the optimized control algorithm is given, the quadcopter can stay for as long as possible. Thus, the objective of the simulation is to find the maximum wind speed that the quadcopter can resist to stay within 20 cm around the target for 30 s.

2.2.2 Aerodynamics of the model

According to the study on the propeller aerodynamics (1), the aerodynamic effects applied to the rotor are evaluated by integrating, along each rotor blade, the aerodynamic force per surface increment.

By integrating the lift per surface increment, we obtain the lift force LF (or

thrust)

$$\mathbf{F_L} = \rho c R^3 \omega^2 C_{L\alpha} \left(\frac{\alpha_0}{3} - \frac{\bar{w} + L\left(\varepsilon_1 q - \varepsilon_2 p\right)}{2R|\omega|} \right) \hat{\mathbf{z}_B} \tag{1}$$

where R is the radius of the rotor blade, ρ is the air density, U is the airspeed, c is the blade width, $C_{L\alpha}$ is the lift coefficient, α_0 its pitch angle at rest. The coefficients ε_1 and ε_2 depend on the rotor under consideration.

The first item with ω^2 is far more larger than other items. Thus, the equation for the lift force can be safely reduced to

$$\mathbf{F_L} = \frac{1}{3} \rho c R^3 \omega^2 C_{L\alpha} \alpha_0 \hat{\mathbf{z}_B} \tag{2}$$

Here the lift coefficient $C_{L\alpha}$ can be approximate by the average pitch angle $\overline{\alpha}$ (3) of the blade.

$$C_{L\alpha} = c_l sin(\overline{\alpha}) cos(\overline{\alpha}). \tag{3}$$

In our model, we set $\rho = 1.225 kg/m^3$, $c = 3cm$, $R = 5cm$, $\overline{\alpha} = 0.78$, thus

$$\mathbf{F_L} = 0.01\omega^2$$

.

For the drag force on the rotors, we neglect the interactions of the vehicle dynamics onto the aerodynamics effects, thus the drag force caused by the rotors is zero.

For the drag force on the middle sphere

$$\mathbf{F_d} = \frac{1}{2} \rho A \|\boldsymbol{u_r}\| \boldsymbol{u_r} C_d, \tag{4}$$

where $A = \pi r^2$ is the cross section area of the sphere, $\boldsymbol{u_r} = \boldsymbol{w} - \dot{\boldsymbol{\xi}}$ is the velocity of wind relative to the quadcopter in the ground frame and C_d is the drag coefficient. Here we set $C_d = 1$.

2.3 Physics simulation system

The dynamics of the quadcopter's motion is approached with rigid body mechanics in (2). We adopted notations from the paper but also made necessary simplifications to the model. The quadcopter has 6 degrees of freedom, 3 for translational motion and 3 for rotation. Both the ground frame and the body frame are considered for solving the motion of the quadcopter. The axes of the body frame x_B, y_B and z_B are indicated in Figure2. The origin of the body frame is located at the center of mass of the quadcopter, which is the center of the central sphere in our model.

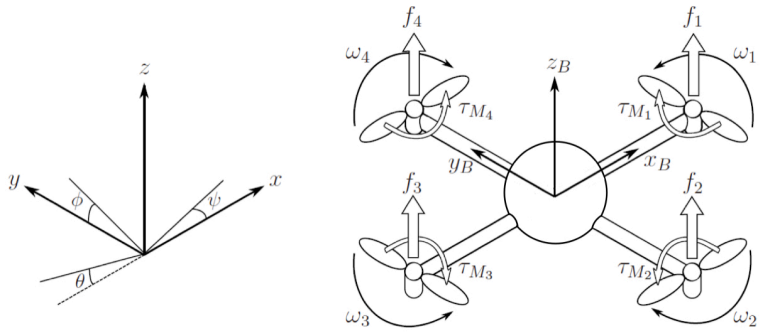


Figure 2: The simplified frame for a quadcopter.

In the ground frame, the Cartesian coordinates x, y, z is denoted by $\boldsymbol{\xi}$. The posture of the quadcopter is described with Euler angle $\boldsymbol{\eta}$ with components ϕ, θ and ψ, as shown in Figure2.

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

In the body frame, we denote the linear velocity with $\boldsymbol{V}_B$ and the angular velocities with $\boldsymbol{\nu}$.

$$\boldsymbol{V}_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

The rotation matrix from the body frame to the inertial frame is an orthogonal matrix

$$\boldsymbol{R} = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix}$$

The transformation matrix for angular velocities from the inertial frame to the body frame is $\boldsymbol{W}_\eta$, and from the body frame to the inertial frame is $\boldsymbol{W}_\eta^{-1}$, as shown in

$$\dot{\boldsymbol{\eta}} = \boldsymbol{W}_\eta^{-1}\boldsymbol{\nu}, \quad \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi / C_\theta & C_\phi / C_\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{5}$$

$$\boldsymbol{\nu} = \boldsymbol{W}_\eta \dot{\boldsymbol{\eta}}, \quad \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta C_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{6}$$

in which $T_x = \tan(x)$. The matrix $\boldsymbol{W}_\eta$ is invertible if $\theta \neq (2k-1)\phi/2$, $k \in \mathbf{Z}$

In our simplified model of the quadcopter, the inertia of its body (excluding the rotors) can be obtained as the inertia of tensor of a sphere:

$$\boldsymbol{I} = \frac{1}{5} m r^2 \boldsymbol{I}_3 \tag{7}$$

The angular velocities the four rotors are denoted by ω_i $(i = 1, 2, 3, 4)$. For simplicity the air drag arising from the rotation and angular acceleration of the rotors are omitted.

We use T to denote the combined force given by the rotors in the z_B direction. In the body frame, the combined force is

$$\boldsymbol{T}_B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \quad \text{where} \quad T = \sum_{i=1}^{4} f_i = k \sum_{i=1}^{4} \omega_i^2 \tag{8}$$

Also, the torque caused by the thrust is

$$\boldsymbol{\tau}_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} d(f_4 - f_2) \\ d(f_3 - f_1) \\ 0 \end{bmatrix} \tag{9}$$

The physical significance of the torque is clear: by altering the difference between f_4 and f_2, rolling movement can be obtained; by altering the difference between f_3 and f_1, one can obtain pitch movement. The acquirement of yaw movement is a little more subtle, and we'll soon notice that yaw movement arises from gyroscopic force Γ in Newton-Euler equations.

The translational and rotational motion of the quadcopter can be fully described with Newton-Euler equations. For translational motion, it's easier to use the ground frame:

$$m\ddot{\boldsymbol{\xi}} = \boldsymbol{G} + \boldsymbol{R}\boldsymbol{T}_B + \boldsymbol{F}_{\text{wind}} \tag{10}$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix} C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi S_\theta C_\phi - C_\psi S_\phi \\ C_\theta C_\phi \end{bmatrix} \tag{11}$$

where F_{wind} is the drag force given in Equation4. For rotational movement, we can obtain the angular acceleration with

$$\boldsymbol{I}\dot{\boldsymbol{\nu}} + \boldsymbol{\nu} \times (\boldsymbol{I}\boldsymbol{\nu}) + \boldsymbol{\Gamma} = \boldsymbol{\tau} \tag{12}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{I}^{-1}\left(-\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx}p \\ I_{yy}q \\ I_{zz}r \end{bmatrix} - I_r \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega_\Gamma + \boldsymbol{\tau}\right) \tag{13}$$

where $\omega_\Gamma = \omega_1 - \omega_2 + \omega_3 - \omega_4$ and I_r is the moment of inertia of each rotor relative to its axis of symmetry.

With the angular acceleration in body frame obtained, we can derive that in the ground frame with $\boldsymbol{W}_\eta$:

$$\begin{aligned} \ddot{\boldsymbol{\eta}} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{W}_\eta^{-1}\boldsymbol{\nu}\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{W}_\eta^{-1}\right)\boldsymbol{\nu} + \boldsymbol{W}_\eta^{-1}\dot{\boldsymbol{\nu}} \\ &= \begin{bmatrix} 0 & \dot{\phi}C_\phi T_\theta + \dot{\theta}S_\phi/C_\theta^2 & -\dot{\phi}S_\phi C_\theta + \dot{\theta}C_\phi/C_\theta^2 \\ 0 & -\dot{\phi}S_\phi & -\dot{\phi}C_\phi \\ 0 & \dot{\phi}C_\phi/C_\theta + \dot{\phi}S_\phi T_\theta/C_\theta & -\dot{\phi}S_\phi/C_\theta + \dot{\theta}C_\phi T_\theta/C_\theta \end{bmatrix}\boldsymbol{\nu} + \boldsymbol{W}_\eta^{-1}\dot{\boldsymbol{\nu}} \end{aligned} \tag{14}$$

2.4 Thrust control system

Our thrust control system is designed as an linear optimization problem for f_1, f_2, f_3, f_4. In each step, our goal is to

- maximize the component of the velocity in the direction that points to the origin.
- maximize the component of angular acceleration in the direction opposite to the current angular velocity.

In other words, our goal is to

$$\text{maximize } (\dot{\boldsymbol{\xi}} + \boldsymbol{\xi}\Delta t)\cdot(-\boldsymbol{r}) \text{ subject to } f_i \leq 7\text{N } (i = 1, 2, 3, 4) \tag{15}$$

and

$$\text{maximize } (\dot{\boldsymbol{\eta}} + \ddot{\boldsymbol{\eta}}\Delta t)\cdot(-\boldsymbol{\eta}) \text{ subject to } f_i \leq 7\text{N } (i = 1, 2, 3, 4) \tag{16}$$

with Equation10, Equation13 and Equation14 these two optimization goals can be reduced to

$$\text{minimize } (\boldsymbol{R}\boldsymbol{T}^B)\cdot\boldsymbol{\xi} \tag{17}$$

and

$$\text{minimize } (\boldsymbol{W}_\eta^{-1}\boldsymbol{I}^{-1}\boldsymbol{\tau})\cdot\boldsymbol{\eta} \tag{18}$$

The gyroscopic forces $\boldsymbol{\Gamma}$ can introduce non-linear terms of the thrust forces, but they are neglected because of their small magnitude compared with $\boldsymbol{\tau}$.

Finally our problem has the form of minimizing $\sigma(f_1 + f_2 + f_3 + f_4)$ and minimizing $w_1(f_4 - f_2) + w_2(f_3 - f_1)$

where

$$\sigma = \boldsymbol{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \boldsymbol{\xi}, \tag{19}$$

$$w_1 = \boldsymbol{W}_{\eta}^{-1} \boldsymbol{I}^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \boldsymbol{\eta}, \tag{20}$$

and

$$w_2 = \boldsymbol{W}_{\eta}^{-1} \boldsymbol{I}^{-1} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \cdot \boldsymbol{\eta}, \tag{21}$$

As a linear optimization problem, The pseudocode for controlling is given in Algorithm1, where ϵ, δ and ζ are parameters for controlling each adjustment.

Algorithm 1 Thrust Control algorithm

procedure UPDATE THRUST CONTROL
- $f_1 \leftarrow f_1 - \text{sgn}(\sigma)\epsilon$
- $f_2 \leftarrow f_2 - \text{sgn}(\sigma)\epsilon$
- $f_3 \leftarrow f_3 - \text{sgn}(\sigma)\epsilon$
- $f_4 \leftarrow f_4 - \text{sgn}(\sigma)\epsilon$
- $f_4 \leftarrow f_4 - \text{sgn}(w_1)\delta$
- $f_2 \leftarrow f_2 + \text{sgn}(w_1)\delta$
- $f_3 \leftarrow f_3 - \text{sgn}(w_2)\zeta$
- $f_1 \leftarrow f_1 + \text{sgn}(w_2)\zeta$

2.5 Wind pattern

Real wind conditions are usually not steady, characterized by constant changes in direction and magnitude. The main characteristics of wind speed are randomness, intermittentness and sudden change. Therefore, when simulating the wind field, we decompose the wind speed vector into dominant wind $\boldsymbol{w}_d$, periodic wind $\boldsymbol{w}_p$, and random wind $\boldsymbol{w}_r$.

2.5.1 Dominant wind

Since wind is relatively continuous to time in most cases, we define a dominant wind speed vector $\boldsymbol{w}_r$ within the time range of our simulation test as a constant vector. For convenience, we set it along the x axis.

$$\boldsymbol{w}_d = k\hat{i}. \tag{22}$$

2.5.2 Periodic wind

In order to better simulate the dynamic changes of real wind, we introduce a periodically changing wind based on the dominant wind, which is specifically expressed as a sinusoidal wind speed vector with initial phase superimposed on the dominant wind in three directions.

$$\begin{cases} w_{px} = u\left(t - t_{bx}\right) W_x sin\left(2\pi\left(\dfrac{t - t_{bx}}{T_{px}}\right)\right), \\ w_{py} = u\left(t - t_{by}\right) W_y sin\left(2\pi\left(\dfrac{t - t_{by}}{T_{py}}\right)\right), \\ w_{pz} = u\left(t - t_{bz}\right) W_z sin\left(2\pi\left(\dfrac{t - t_{bz}}{T_{pz}}\right)\right). \end{cases} \tag{23}$$

Here $u(t - t_{bi})$ is the step function that switches from 0 to 1 when $t = t_{bi}$, $\boldsymbol{T}_{pi}$ is the period of the periodic wind from the ith coordinate and t_{bi} is the begin time of the sinusoidal wind from ith coordinate.

2.5.3 Random noise

Considering different disturbance in the open environment, the random wind component $\overrightarrow{\boldsymbol{w_r}}$ is also added into windspeed vector. The random wind here is simulated by adding a noise into the dominant wind. The added random noise follows the Gaussian distribution with the probability density

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{24}$$

where μ is the mean and σ the standard deviation. The function has its peak at the mean, and its "spread" increases with the standard deviation. This implies that the noise added into the dominant wind is more likely to return to the mean value, rather than those far away. For each step of time, the wind vector is incremented by the noise

$$\begin{cases} w_{rx}(t + \Delta t) = w_{rx} + \Delta w_{rx,n}, \\ w_{ry}(t + \Delta t) = w_{ry} + \Delta w_{ry,n}, \\ w_{ry}(t + \Delta t) = w_{rz} + \Delta w_{rz,n}. \end{cases} \tag{25}$$

Here $\boldsymbol{w}_{r,i}$ is the random wind component along the ith coordinate, $\Delta \boldsymbol{w}_{ri,n}$ is the increment following the Guassian normal distribution added to the i th random wind component each step of time in the iteration.

However, because the noise we introduced is the vector sum of three one-dimensional random walk along x, y, z coordinate, the random windspeed will gradually drifting from its mean magnitude 0. To avoid the noise to change the

magnitude of the dominant windspeed, a correction item in direct proportion to the current random windspeed is introduced into the iteration. Thus, Eq.(25) can be refined to

$$\begin{cases} w_{rx}(t+\Delta t) = w_{rx} + \Delta w_{rx,n} - C_I w_{rx}, \\ w_{ry}(t+\Delta t) = w_{ry} + \Delta w_{ry,n} - C_I w_{ry}, \\ w_{ry}(t+\Delta t) = w_{rz} + \Delta w_{rz,n} - C_I w_{rz}. \end{cases} \tag{26}$$

Here C_I is the coeffient introduced to eliminate the drift. The detail method is introduced in section 2.4.

Combining the dominant, periodic and random windspeed, the real wind condition can be approached, In our physics simulation, the windspeed input will be chosen from the vector sum of these components

$$\boldsymbol{w} = \boldsymbol{w}_d + \boldsymbol{w}_p + \boldsymbol{w}_r. \tag{27}$$

3 Results

3.1 Dominant wind with random noise

In order to find the maximum wind speed that complies with our definition of the problem in Section2.2.1, wind patterns with different speed distributions were tested in our simulation environment. We aim to find the maximum wind speed that enables the quadcopter to stay around the target within 20 cm for 30 s. For this reason, the simulation for each wind model is suspended once the distance from the origin reaches 20cm. t_20 for each wind model is obtained as the average of 10 simulations under the same condition. The simulation data is summarized in Table1.

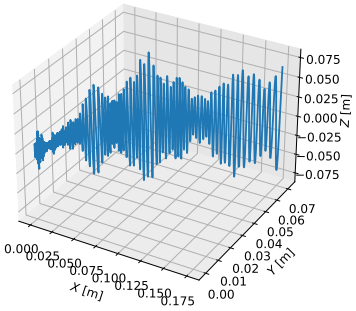
Referring from the table, we can roughly find the maximum wind speed $v_M = 5$ m/s.

Wind Speed [m/s]	Deviation of the Wind Speed [m/s]	t_{20} [s]
2	0.01	45.55
2	0.11	36.59
2	0.21	28.27
2	0.31	23.58
2	0.41	22.11
2	0.51	18.79
2	0.61	12.10
3	0.01	48.70
3	0.11	35.68
3	0.21	31.36
3	0.31	15.73
3	0.41	17.83
3	0.51	15.02
3	0.61	12.75
4	0.01	53.68
4	0.11	45.48
4	0.21	30.09
4	0.31	20.69
4	0.41	16.35
4	0.51	14.25
4	0.61	12.85
5	0.01	54.13
5	0.11	54.89
5	0.21	45.45
5	0.31	15.63
5	0.41	23.66
5	0.51	13.81
5	0.61	10.56
6	0.01	60.59
6	0.11	53.93
6	0.21	26.38
6	0.31	21.37
6	0.41	19.43
6	0.51	10.83
6	0.61	8.41
7	0.01	42.67
7	0.11	28.90
7	0.21	34.13
7	0.31	21.08
7	0.41	20.81
7	0.51	13.50
7	0.61	9.95

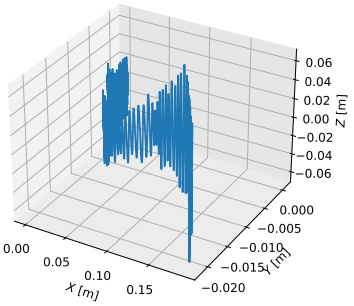
Table 1: Simulation Data

To make our simulation more convincing, four sample simulation trajectories are plotted in Figure 3. Each subfigure was simulated under wind speed of 5 m/s and deviation of 0.1m/s. The trajectories consist of positions of center of mass during the simulations. These plots demonstrate how the control system manages to keep the quadcopter near the origin when it's subject to the wind.

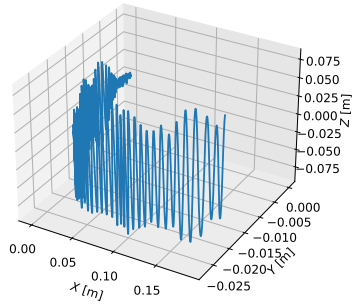
The wind field is also visualized in Figure 4 to give an clear intuition of how it varies in direction and magnitude through time. The dataset of wind speed in the figure has deviation of 0.2 m/s. Each line represents the wind speed at an instant. The color of the end point of the lines vary continuously from deep brown to pale yellow.



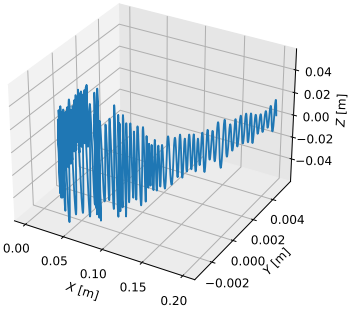
(a) Trajectory 1



(b) Trajectory 2



(c) Trajectory 3



(d) Trajectory 4

Figure 3: Sample Trajectories for Wind Speed = 5 m/s

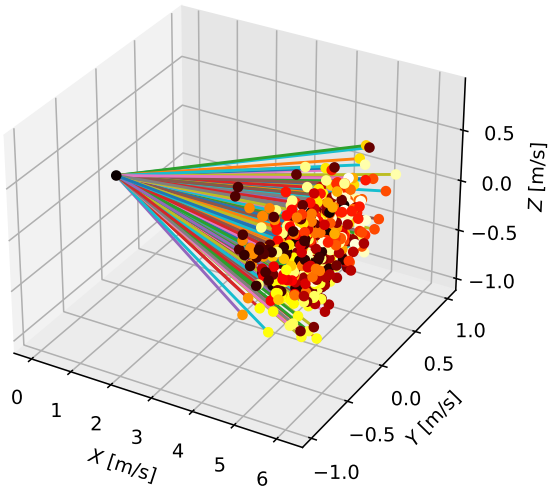


Figure 4: Visualization of Wind

3.2 Periodic wind

From Fourier analysis we know that any periodical wind can be decomposed into a series of wind with sinusoidal velocity components. Figure 6 exhibits the possible wind speed with sinusoidal velocity components.

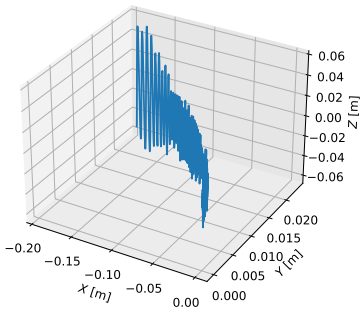
Therefore the study of periodical wind with sinusoidal velocity components is crucial as it can help us predicate the behavior of the quadcopter under more general periodical wind patterns. In this section we observe periodic wind with sinusoidal velocity component in the x-axis direction. In order to find the maximum wind speed that complies with our definition of the problem in Section2.2.1, wind with different amplitude and frequency were tested in our simulation environment. Again, t_20 denotes the time at which the distance from the origin reaches 20cm. The simulation data is summarized in Table2 and Table3.

Referring from the table, we can conclude that our control model cannot handle wind with varying magnitude as perfectly as it does with dominant wind patterns. With fixed amplitude, the stability becomes worse at the frequency of the change of wind's velocity increases. However, we can also notice that the stability is less sensitive to increasement of frequency at lower amplitudes.

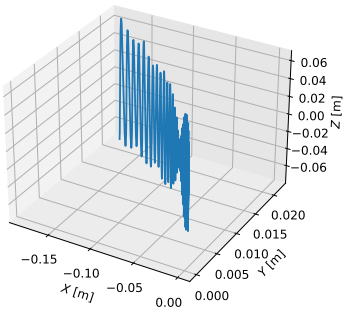
Under frequency $f = 0.01$ Hz, the performance reaches our expectation only when the amplitude of the velocity of wind is no larger than 0.95 m/s. When the

frequency becomes larger, the quadcopter will be able to keep itself stable under lower amplitudes.

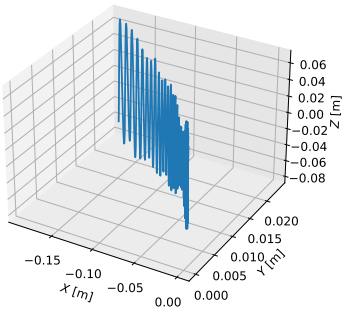
For periodic wind, four sample simulation trajectories are plotted in Figure 5. Each subfigure was simulated under amplitude 1.2m and frequency 0.01 Hz. The trajectories consist of positions of center of mass during the simulations. These plots demonstrate how the control system manages to keep the quadcopter near the origin when it's subject to periodical wind.



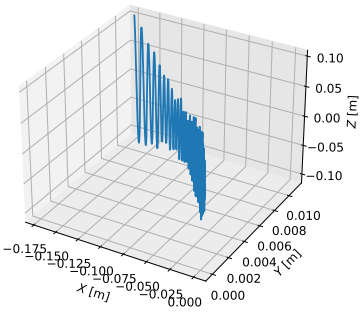
(a) Trajectory 1



(b) Trajectory 2



(c) Trajectory 3



(d) Trajectory 4

Figure 5: Sample Trajectories for Wind Speed = 5 m/s

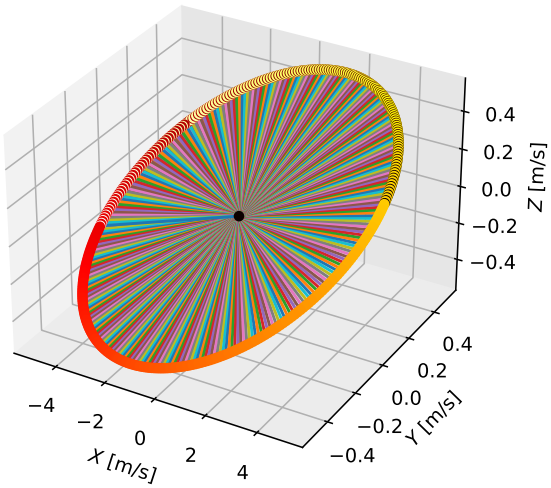


Figure 6: Visualization of Wind

Speed Amplitude [m/s]	frequency [Hz]	t_{20} [s]
1.001	0.01	26.54
1.001	0.02	20.50
1.001	0.03	18.08
1.001	0.04	16.60
1.001	0.05	15.78
1.001	0.060	15.28
1.001	0.070	15.27
2.001	0.01	17.73
2.001	0.02	13.45
2.001	0.03	11.63
2.001	0.04	10.53
2.001	0.05	9.72
2.001	0.060	9.12
2.001	0.070	8.66
3.001	0.01	14.66
3.001	0.02	10.67
3.001	0.03	8.99
3.001	0.04	8.070
3.001	0.05	7.47
3.001	0.060	7.00
3.001	0.070	6.630

Table 2: Periodic Wind Simulation Data Table 1

Speed Amplitude [m/s]	frequency [Hz]	$t_{20}[s]$
0.7	0.001	44.86
0.7	0.011	32.46
0.7	0.021	24.67
0.7	0.031	21.93
0.7	0.041	21.05
0.7	0.051	21.05
0.7	0.061	21.31
0.75	0.001	44.58
0.75	0.011	30.70
0.75	0.021	24.09
0.75	0.031	21.29
0.75	0.041	20.19
0.75	0.051	19.96
0.75	0.061	20.20
0.8	0.001	44.26
0.8	0.011	30.09
0.8	0.021	23.26
0.8	0.031	20.24
0.8	0.041	19.24
0.8	0.051	18.76
0.8	0.061	19.01
0.85	0.001	47.57
0.85	0.011	27.78
0.85	0.021	21.95
0.85	0.031	19.73
0.85	0.041	18.33
0.85	0.051	17.88
0.85	0.061	17.89
0.9	0.001	45.60
0.9	0.011	27.46
0.9	0.021	21.31
0.9	0.031	19.03
0.9	0.041	17.83
0.9	0.051	17.12
0.9	0.061	17.08
0.95	0.001	47.43
0.95	0.011	26.25
0.95	0.021	20.77
0.95	0.031	18.52
0.95	0.041	17.11
0.95	0.051	16.38
0.95	0.061	16.00

Table 3: Periodic Wind Simulation Data Table 2

4 Discussion

4.1 Conclusion

In this paper, we proposed a physics simulation system to simulate

- the dynamics of the quadcopter;
- a thrust control system which controls the thrusts of the quadcopter, keeping the quadcopter near the origin;
- a wind simulator to simulate winds that vary in time and magnitude through time.

These three parts are combined to find the maximum wind speed, which is

- for dominant wind with noise,
- for periodic wind.

We find that the maximum windspeed allowed for the quadcopter to stay within the desired range for at least $30s$ when the dominant wind with noise was applied is $5m/s$.

For the periodic wind pattern, our control model cannot handle wind with varying magnitude as perfectly as it does with dominant wind patterns. With fixed amplitude, the stability becomes worse at the frequency of the change of wind's velocity increases.

Under frequency $f = 0.01$ Hz, the performance reaches our expectation only when the amplitude of the velocity of wind is no larger than $0.95m/s$.

We can accordingly conclude that periodic wind poses greater threats to safety operation than the dominant wind with noise.

4.2 Advantages

Though it's a simplified model, our simulation has many advantages.

1. The physics simulation system is accurate.

   As the rigid body dynamics theorem is used comprehensively, our physics simulation system will have the exact behavior as the according simplified model in reality.

2. The thrust control algorithm is original and has strong physics interpretation.

   The thrust control algorithm was derived purely by solving the Newton-Euler equations and extracting terms with linear dependence on the thrust

from the accelerations (both translational and angular). Thus, compared to the standard PID control algorithm (4), it does not only have far more less parameters to adjust, but also a much clearer connection to the model it controls.

3. The wind simulator simulates different types of winds and is adjustable.

   Through the simulation mathematical models of the three parts of wind speed that compose the wind field, dominant wind, periodic wind and random wind are used to simulate the change process of wind speed over a period of time, so that the randomness, intermittentness and the mutant characteristics of the real wind field under external interference can be considered.

4.3 Limitation and possible refinement

Due to time limit, our model still has some limitations that need further refinement.

1. The shape of the quadcopter.

   In this article, we simplify the major part of the quadcopter as a sphere and omit the structure connecting the rotors and the middle platform. However, actually the moment of inertia of the rod structures also matters in the equations of rigid body dynamics. However, in our model, due to the symmetry, this additional item can be replaced by changing the mass distribution of our model and will not effect the final results.

2. The omitted Gyro force

   In our thrust control system, we neglect the Gyro force when the angular speed is changing according to time. However, the neglect is reasonable because the magnitude of the force is 1-2 order smaller than other force, e.g. lift force or drag force acting on the middle sphere. Also, this force is considered in our physics simulation to accurately predict the motion of the quadcopter.

3. Air drag To reduce the complexity of calculation, we omitted the drag force acting on four rotors. According to the study on the aerodynamics of the propellers(1), the drag force mainly depends on the translational velocity of the quadcopter and the angular speed of pitch, roll and yaw. The simulation can be improved by adding these items into our functions of force in the programs.

References

[1] P. Bristeau, P. Martin, E. Salaün and N. Petit, “The role of propeller aerodynamics in the model of a quadrotor UAV,” 2009 European Control Conference (ECC), Budapest, 2009, pp. 683-688, doi: 10.23919/ECC.2009.7074482.

[2] Luukkonen, Teppo. ”Modelling and control of quadcopter.” Independent research project in applied mathematics, Espoo 22 (2011).

[3] R. Gill and R. D’Andrea, ”Propeller thrust and drag in forward flight,” 2017 IEEE Conference on Control Technology and Applications (CCTA), Mauna Lani, HI, 2017, pp. 73-79, doi: 10.1109/CCTA.2017.8062443.

[4] J. Li and Y. Li, ”Dynamic analysis and PID control for a quadrotor,” 2011 IEEE International Conference on Mechatronics and Automation, Beijing, 2011, pp. 573-578, doi: 10.1109/ICMA.2011.5985724.

A Source Code

Listing 1: Source Code

```
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import sys

g = 9.80665
# maxmimum thrust of each rotor
thrustMax = 7


# keep thrust in reasonable range
def _thresholdControl(thrust):

    return max(0, min(thrust, thrustMax))


# gravitational acceleration in ground frame
gravity = [0, 0, - g]
# mass of the drone
mass = 1.5
# distance between the center of the drone and each rotor
d = 0.5
# constant ratio in relation between angular speed of rotor
# and the thrust it produces
# f = k * omega**2
rotorAngularSpeedConstant = 0.01


class Velocities:
    def __init__(self, vx, vy, vz, vpsi, vtheta, vphi):
        self.vx = vx
        self.vy = vy
        self.vz = vz
        self.vpsi = vpsi
```

```
        self.vtheta = vtheta
        self.vphi = vphi


class Accelerations:
    def __init__(self, ax, ay, az, apsi, atheta, aphi):
        self.ax = ax
        self.ay = ay
        self.az = az
        self.apsi = apsi
        self.atheta = atheta
        self.aphi = aphi


class Control:
    thrust1 = [] # in the direction of (1,1)
    thrust2 = [] # in the direction of (1,-1)
    thrust3 = [] # in the direction of (-1,-1)
    thrust4 = [] # in the direction of (-1,1)

    def __init__(self, t1, t2, t3, t4):
        self.thrust1.append(t1)
        self.thrust2.append(t2)
        self.thrust3.append(t3)
        self.thrust4.append(t4)


class Wind:
    x = 0
    y = 0
    z = 0

    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def toList(self):
        return [self.x, self.y, self.z]
```

```
class BlowingWind:
    wind = Wind(0, 0, 0)
    speed = 0
    std = 0
    t = 0

    def __init__(self, speed, std):
        self.speed = speed
        self.std = std
        self.wind.x = speed

    def generate(self):
        self.t += 0.01
        f = 0.01
        self.wind.x += np.random.normal(0, self.std) - 1 / 4 * (
            self.wind.x - self.speed)
        self.wind.y += np.random.normal(0, self.std) - 1 / 4 *
            self.wind.y
        self.wind.z += np.random.normal(0, self.std) - 1 / 4 *
            self.wind.z
        return self.wind

    def toList(self):
        return [self.wind.x, self.wind.y, self.wind.z]


class SinusoidalWind:
    wind = Wind(0, 0, 0)
    t = 0

    def __init__(self, ax, ay, az, freq):
        self.ax = ax
        self.ay = ay
        self.az = az
        self.phix = np.pi / 2
        self.phiy = 0.233
        self.phiz = 0.343
```

```
        self.freq = freq

    def toList(self):
        return [self.wind.x, self.wind.y, self.wind.z]

    def generate(self):
        self.t += 0.01
        self.wind.x = self.ax * np.sin(2 * np.pi * self.freq *
            self.t + self.phix)
        self.wind.y = self.ay * np.sin(2 * np.pi * self.freq *
            self.t + self.phiy)
        self.wind.z = self.ay * np.sin(2 * np.pi * self.freq *
            self.t + self.phiz)
        return self.wind


class Drone:
    psi = [0] # yaw; z
    theta = [0] # pitch; y
    phi = [0] # roll; x
    x = [0]
    y = [0]
    z = [0]
    ax = 0
    ay = 0
    az = 0
    ms = 0 # mass of center sphere
    rs = 0 # radius of center sphere
    mr = 0 # mass of each rotor
    Ir = 0 # moment of inertia of each rotor
    deltaT = 0.05
    control = Control(0, 0, 0, 0)
    breakflag = 0

    def __init__(self, deltaT, ms, rs, mr, Ir, speed):
        self.deltaT = deltaT
        self.ms = ms
        self.rs = rs
        self.mr = mr
```

```
        self.Ir = Ir
        self.control = Control(0,0,0,0)
        self.breakflag = 0
        self.psi = [0] # yaw; z
        self.theta = [0] # pitch; y
        self.phi = [0] # roll; x
        self.x = [0]
        self.y = [0]
        self.z = [0]
        self.ax = 0
        self.ay = 0
        self.az = 0
        # speed: x-component of the velocity of wind at t = 0
        temp_thrust = np.sqrt((self._wind2force(Wind(speed, 0, 0)
           ))[0] ** 2 + (gravity[2] * mass) ** 2) / 4
        self.control.thrust1.append(temp_thrust)
        self.control.thrust2.append(temp_thrust)
        self.control.thrust3.append(temp_thrust)
        self.control.thrust4.append(temp_thrust)
        self.theta.append(- np.arctan(np.abs(self._wind2force(
           Wind(speed, 0, 0))[0]) \
                                / (g * mass)))

    def _wind2force(self, wind):
        # drag coefficient
        cd = 0.5

        # density of air
        rho = 1.293

        # cross section
        A = np.pi * self.rs ** 2

        # velocity of CoM
        v = self.velocities()
        u = [v.vx, v.vy, v.vz]

        # relative velocity in ground frame
        ur = np.array(u) - np.array(wind.toList())
```

```
        # relative speed
        sr = np.linalg.norm(ur)

        # air drag in ground frame
        F = - 1 / 2 * rho * A * sr * ur * 0.5

        return F # a function which gives force considering wind
            speed

    def _updateControl(self):
        distance = np.linalg.norm([self.x[-1], self.y[-1], self.z
            [-1]])
        if (distance >= 0.2):
           self.breakflag = 1

        qualifier = max(distance, 0.13)
        thrust1 = self.control.thrust1[-1]
        thrust2 = self.control.thrust2[-1]
        thrust3 = self.control.thrust3[-1]
        thrust4 = self.control.thrust4[-1]

        p1 = 80 * qualifier
        p2 = 90 * qualifier
        p3 = 80 * qualifier

        thrust1 -= np.sign(self.sigma) * p1
        thrust2 -= np.sign(self.sigma) * p1
        thrust3 -= np.sign(self.sigma) * p1
        thrust4 -= np.sign(self.sigma) * p1

        thrust4 -= np.sign(self.w1) * p2
        thrust2 += np.sign(self.w1) * p2
        thrust3 -= np.sign(self.w2) * p3
        thrust1 += np.sign(self.w2) * p3

        thrust1 = _thresholdControl(thrust1)
        thrust2 = _thresholdControl(thrust2)
        thrust3 = _thresholdControl(thrust3)
```

```
        thrust4 = _thresholdControl(thrust4)

        self.control.thrust1.append(thrust1)
        self.control.thrust2.append(thrust2)
        self.control.thrust3.append(thrust3)
        self.control.thrust4.append(thrust4)

    # def update(self,wind): # The wind is supposed to be blown
         in the x direction

    def velo(self, list):
        return (list[-1] - list[-2]) / self.deltaT

    def acc(self, list):
        if len(list) >= 3:
            return (list[-1] - 2 * list[-2] + list[-3]) / self.
                deltaT ** 2
        else:
            return 0

    def velocities(self):
        if len(self.x) == 1:
            return Velocities(0, 0, 0, 0, 0, 0)
        else:
            return Velocities(self.velo(self.x), self.velo(self.y)
                , self.velo(self.z), self.velo(self.psi),
                             self.velo(self.theta), self.velo(self.
                                phi))

    def accelerations(self):
        if len(self.x.len) <= 2:
            return Accelerations(0, 0, 0, 0, 0, 0)
        else:
            return Accelerations(self.acc(self.x), self.acc(self.y
                ), self.acc(self.z), self.acc(self.psi),
                                 self.acc(self.theta), self.acc(self
                                    .phi))

    def update(self, wind):
```

```
        # base transition matrix
        rot = self.rotationMatrix()

        # Force caused by wind in ground frame
        fWind = self._wind2force(wind)

        # wind force in drone frame
        wind_drone = np.matmul(np.transpose(rot), fWind)

        # gravitational acceleration in drone frame
        g_drone = np.matmul(np.transpose(rot), gravity)

        # current thrust forces in ground frame
        f1 = self.control.thrust1[-1]
        f2 = self.control.thrust2[-1]
        f3 = self.control.thrust3[-1]
        f4 = self.control.thrust4[-1]

        # total upward force in the drone frame
        T = f1 + f2 + f3 + f4

        # total upward force vector in the drone frame
        T_drone = np.transpose([0, 0, T])

        # get the first-order derivative of the coordinates in
            the ground frame
        v = self.velocities()

        # translational acceleration in ground frame by Newton
            II
        translationalAcceleration_ground = gravity + (1 / mass) *
             np.matmul(rot, T_drone) + (1 / mass) * fWind

        # extract angular derivatives of Euler angles and
            convert to drone frame
        angularVelo_ground = [v.vphi, v.vtheta, v.vpsi]
        translationalVelo_ground = [v.vx, v.vy, v.vz]
        omega = np.matmul(self.angularVeloToDrone(),
            angularVelo_ground)
```

```
        # torques provided by the rotors in the drone frame
        tau = [d * (f4 - f2), d * (f3 - f1), 0]

        # from the thrust forces, obtain the angular velocities
            of the rotors in the drone frame
        rotorOmega1 = self.rotorAngularSpeed(f1)
        rotorOmega2 = self.rotorAngularSpeed(f2)
        rotorOmega3 = self.rotorAngularSpeed(f3)
        rotorOmega4 = self.rotorAngularSpeed(f4)
        # the angular velocity term in gyroscope torque
        omegaTau = rotorOmega1 - rotorOmega2 + rotorOmega3 -
            rotorOmega4

        # angular acceleration in drone frame by Euler-Newton
            equation
        angularAcceleration_drone = np.matmul(np.linalg.inv(self.
            sphereInertia()), - np.cross(omega, np.matmul(
            self.sphereInertia(), omega)) - self.Ir * np.cross(
                omega, [0, 0, 1]) * omegaTau + tau)

        # update the vx, vy, vz, x, y, z according to the
            derivatives obtained above
        translationalVelo_ground_new = self.deltaT *
            translationalAcceleration_ground +
            translationalVelo_ground
        self.x.append(self.x[-1] + self.deltaT *
            translationalVelo_ground_new[0])
        self.y.append(self.y[-1] + self.deltaT *
            translationalVelo_ground_new[1])
        self.z.append(self.z[-1] + self.deltaT *
            translationalVelo_ground_new[2])

        # update the Euler angles
        # first get the new angular velocities in the drone
            frame
        omega_new = omega + self.deltaT *
            angularAcceleration_drone
```

```
        # switch to ground frame
        angularVelo_ground_new = np.matmul(np.linalg.inv(self.
            angularVeloToDrone()), omega_new)

        self.phi.append(self.phi[-1] + self.deltaT *
            angularVelo_ground_new[0])
        self.theta.append(self.theta[-1] + self.deltaT *
            angularVelo_ground_new[1])
        self.psi.append(self.psi[-1] + self.deltaT *
            angularVelo_ground_new[2])

        # update acceleration
        self.ax = self.acc(self.x)
        self.ay = self.acc(self.y)
        self.az = self.acc(self.z)

        # scalars for controlling
        self.sigma = rot[0][2] * self.x[-1] + rot[1][2] * self.y
            [-1] + rot[2][2] * self.z[-1]
        wi = np.matmul(np.linalg.inv(self.angularVeloToDrone()),
            np.linalg.inv(self.sphereInertia()))
        self.w1 = np.inner(angularVelo_ground_new, wi[:, 0])
        self.w2 = np.inner(angularVelo_ground_new, wi[:, 1])

        # update control signal
        self._updateControl()


    # base transition matrix between the drone frame and the
        ground frame
    def rotationMatrix(self):
        cps = np.cos(self.psi[-1])
        sps = np.sin(self.psi[-1])
        cph = np.cos(self.phi[-1])
        sph = np.sin(self.phi[-1])
        ct = np.cos(self.theta[-1])
        st = np.sin(self.theta[-1])
        mat = [[cps * ct, cps * st * sph - sps * cph, cps * st *
            cph + sps * sph],
```

```
            [sps * ct, sps * st * sph + cps * cph, sps * st *
               cph - cps * sph],
            [-st, ct * sph, ct * cph]]
        return mat

    # transformation for angular velocity
    # result * (angular velocity in ground frame)
    # = (angular velocity in drone frame)
    # W_{\eta}
    def angularVeloToDrone(self):
        cps = np.cos(self.psi[-1])
        sps = np.sin(self.psi[-1])
        cph = np.cos(self.phi[-1])
        sph = np.sin(self.phi[-1])
        ct = np.cos(self.theta[-1])
        st = np.sin(self.theta[-1])
        mat = [[1, 0, -st],
               [0, cph, -sph],
               [0, -sph, ct * cph]]
        return mat

    def derivativeOfAngularVeloDroneToGround(self, omega):
        vphi = omega[0]
        vtheta = omega[1]
        vpsi = omega[2]
        cps = np.cos(self.psi[-1])
        sps = np.sin(self.psi[-1])
        cph = np.cos(self.phi[-1])
        sph = np.sin(self.phi[-1])
        ct = np.cos(self.theta[-1])
        st = np.sin(self.theta[-1])
        tt = np.tan(self.theta[-1])
        mat = [[0, vphi * cph * tt + vtheta * sph / ct ** 2, -
           vphi * sph * ct + vtheta * cph / ct ** 2],
               [0, - vphi * sph, - vphi * cph],
               [0, vphi * cph / ct + vphi * sph * tt / ct, - vphi
                  * sph / ct + vtheta * cph * tt / ct]]

    def sphereInertia(self):
```

```
        Ixx = 1 / 5 * self.ms * self.mr ** 2
        return [[Ixx, 0, 0], [0, Ixx, 0], [0, 0, Ixx]]

    def rotorAngularSpeed(self, thrust):
        return np.sqrt(np.abs(thrust / rotorAngularSpeedConstant)
            )
```