Quantifying the flight envelope of a multi-rotor aircraft under random wind forcing using Probabilistic collocation method

Kevin Silberberg[1]

[1]**University Of California, Santa Cruz**

ABSTRACT

This study introduces a method to quantify the operational flight envelope of multi-rotor drones subject to random wind disturbances using the Probabilistic Collocation Method (PCM). By modeling wind as independent Gaussian random variables in two dimensions, the study solves a random initial value problem to assess the drone's stability during takeoff and hovering. PCM enables efficient computation of the mean and variance of the drone's trajectory under varying wind intensities, offering a faster alternative to Monte Carlo simulations. Results reveal a critical threshold for wind variability, beyond which stable hovering cannot be guaranteed, highlighting a clear relationship between wind variability and flight stability. This framework provides valuable insights for mission planning in unpredictable weather conditions and establishes a computationally efficient method for evaluating the impact of control design on flight performance. Future work aims to explore the influence of payload changes, control modifications, and 3D spatial modeling on the drone's stability under random wind forcing.

Keywords: Probabilistic collocation method, Flight Envelope

NOMENCLATURE

t = time
$y(t)$ = motion in horizontal direction
$z(t)$ = motion in vertical direction
$\phi(t)$ = angle of rotation
$f(t)$ = force of lift orthogonal to airframe
$\tau(t)$ = rotational force in clockwise direction
$F_{d_{y,z}}$ = drag force in the y or z direction
C_d = drag coefficient
$\xi_{y,z}$ = wind speed
$v_{r_{y,z}}$ = air relative velocity
$u_{y,z,\phi}$ = system control output
$Kp_{y,z,\phi}$ = proportional gain for each state
$Ki_{y,z,\phi}$ = integral gain for each state
$Kz_{y,z,\phi}$ = derivative gain for each state
m = mass of the drone
g = force of gravity
I_{xx} = mass moment of inertia
L = length of the drone arm
$T_{L,R}$ = thrust produced by left or right motor
y_d, z_d = desired coordinate position

INTRODUCTION

Multi-rotor small unmanned aircraft systems (sUAS) have recently been employed in a wide range of industrial applications, including the inspection of various structures (e.g., space shuttles, wind turbines, nuclear power plants), the measurement of atmospheric parameters (e.g., wind speed, wind direction, temperature, pressure), and participation in search and rescue operations. Additionally, sUAS operating in stable hovering flight are increasingly used to capture atmospheric particulate matter by mounting a laser-gas analyzer aboard the drone. However, this approach can push the drone toward its payload limits and increase instability under uncertain wind conditions. This is often done in conjunction with either adding an additional sensor (e.g., a sonic anemometer) to measure wind velocity directly, or using sensor-free methods to infer the wind velocity vector from the drone's Inertial Measurement Unit (IMU) Neumann and Bartholmai (2015).

It is critical to understand the flight envelope and operational limitations of a given sUAS configuration—particularly regarding payload requirements, weight distribution, and control parameters under challenging conditions such as high winds. NASA has investigated off-nominal flight dynamics in multi-rotor drones, including scenarios where

the vehicle deviates significantly from its intended flight path due to controller-receiver communication failures (runaway events), extreme maneuvers that lead to unrecoverable flight regimes, or adverse environmental conditions such as vortex ring states (VRS) during rapid descent Foster and Hartman (2017).

In this study, we propose a method for quantifying the drone's flight envelope during takeoff and approach to stable hovering flight by modeling the wind as independent Gaussian random variables in each spatial dimension. We then solve the associated random initial value problem—where the drone takes off from a stationary position and attempts to hover at a desired location under random uniform forcing—using a probabilistic collocation method (PCM). By using PCM, the wind can, in fact, be modeled with any distribution, provided that the random variables in each spatial dimension are independent.

MODEL

For the purpose of this study, we make certain assumptions to simplify the aerodynamics of the equations of motion for a feedback-stabilized multirotor drone, considering only two spatial dimensions. However, the principles we apply can be extended to three dimensions, as demonstrated by the rigid body model outlined in González-Rocha et al. González-Rocha et al. (2019).

2D-Rigid Body Model

The equations of motion are represented as a system of first-order nonlinear time-invariant ordinary differential equations.

$$
\begin{aligned}
\dot{x}_1 &= x_4 \\
\dot{x}_2 &= x_5 \\
\dot{x}_3 &= x_6 \\
\dot{x}_4 &= \frac{1}{m}\left(F_{d_y} - u_1 \sin(x_3)\right) \\
\dot{x}_5 &= \frac{1}{m}\left(F_{d_z} + u_1 \cos(x_3) - mg\right) \\
\dot{x}_6 &= \frac{u_2}{I_{xx}}
\end{aligned}
$$

Under the mapping,

$$
\begin{aligned}
x_1 &= y(t) & & \\
x_2 &= z(t) & & \\
x_3 &= \phi(t) & u_1 &= f(t) \\
x_4 &= \dot{y}(t) & u_2 &= \tau(t) \\
x_5 &= \dot{z}(t) & & \\
x_6 &= \dot{\phi}(t) & &
\end{aligned}
$$

where $f(t)$, the force of thrust orthogonal to the airframe, is a linear combination of the thrust produced by the left and right motors.

$$f(t) = T_R + T_L$$

$\tau(t)$, the torque experienced by the aircraft about its center of mass, is the difference between the thrust produced by the left and right motors, multiplied by the distance of the actuator from the center of mass (the length of the actuator arm).

$$\tau(t) = L(T_R - T_L)$$

I_{xx}, the mass moment of inertia, is the total mass of the aircraft multiplied by the square of the length of the actuator arm.

$$I_{xx} = mL^2$$

Controller

To define a control law using a proportional-integral-derivative (PID) controller, we linearized the model about steady and stable flight, such that the tilt angle at equilibrium is zero. Notice that at $\phi_{eq}(t) = 0$, the aircraft would be parallel to the ground, and the total thrust produced by the motors would counteract the force of gravity acting on the airframe, i.e., $f_{eq}(t) = mg$. This condition allows the aircraft to hover at a desired coordinate position.

Given the above, the nonlinearities in the model become,

$$
\begin{aligned}
\sin(\phi_{eq}(t)) &= \phi_{eq}(t) \\
\cos(\phi_{eq}(t)) &= 1
\end{aligned}
$$

See equations (1), (2), and (3) in the appendix for the control law, where $u_{1_{ctrl}}$ and $u_{2_{ctrl}}$ are the inputs to the system. Here, ε is a small parameter that determines the time interval of the error accounted for in the input, and ϕ_d is the desired tilt angle required to achieve the desired position in the horizontal direction.

Drag and Random Wind Forcing

To induce translational motion on the drone that is independent of the drone's tilt angle, we must introduce a drag term in the model. We define the air-relative wind speed experienced by the drone as the difference between the drone's current speed and the wind speed along each spatial direction.

$$v_{r_y} = \frac{dy}{dt} - \xi_y \qquad \xi_y, \xi_z \sim \mathcal{N}(0, \sigma^2)$$
$$v_{r_z} = \frac{dz}{dt} - \xi_z$$

where $\xi_{y,z}$ are normally distributed independent random variables with a mean of zero and a varying standard deviation, σ, for each spatial component. Notice that in the context of a random variable exhibiting a constant forcing on the drone over time, if we sample both random variables independently, we obtain a uniform random vector field ξ with magnitude $||\xi|| = \sqrt{\xi_y^2 + \xi_z^2}$ and direction $\theta = \arctan\left(\frac{\xi_z}{\xi_y}\right)$.

Given the above, we define the quadratic drag as:

$$F_{d_y} = -C_d v_{r_y} \sqrt{v_{r_y}^2 + v_{r_z}^2},$$
$$F_{d_z} = -C_d v_{r_z} \sqrt{v_{r_y}^2 + v_{r_z}^2},$$

where the parameter C_d is a generalized drag coefficient that captures information such as the cross-sectional area of the drone, the density of the air, and other related factors. In reality, the effects of drag are asymmetric; however, for our purposes, we assume that the drag effects on the drone are equal in both spatial directions.

Notice that the parameter σ, the standard deviation of the normal distribution, is analogous to wind speed $||\xi||$ in our model. The larger the variance of the normal distribution about the mean of zero, the higher the probability of sampling a random vector with increased magnitude.

METHODS

To understand the flight limitations of a drone under varying wind conditions, or the maximal average wind speed that a drone can experience during flight before its stability about the desired position is no longer guaranteed, we introduce the probabilistic collocation method (PCM). This method allows us to calculate the average trajectories and the associated variance of many realizations of a drone flying in a uniform vector field to a maximal degree of exactness. By "maximal degree of exactness," we mean that the statistical properties (e.g., mean, variance, etc.) of the random initial value problem associated with the PID-controlled dynamical system can be computed exactly, down to machine precision.

Orthogonal Polynomials

The first step is to calculate the set of unique orthogonal polynomials associated with the probability density function (PDF) of the distributions used to model the wind. The moment problem for each random variable must be uniquely solvable and must therefore satisfy at least one of the following criteria (Venturi, 2014a, p. 42, Polynomial Chaos):

1. The PDF is compactly supported.
2. The moment-generating function
$$M(n) = \mathbb{E}\{e^{n\xi(w)}\}$$
exists and is finite in a neighborhood of $n = 0$.
3. $\xi(w)$ is exponentially integrable.
4. The sequence of moments $M_n = \mathbb{E}\{\xi^n\}$ satisfies
$$\sum_{n=0}^{\infty} \left(\frac{1}{M_{2n}}\right)^{\frac{1}{2n}} = \infty.$$

Once these conditions are satisfied, the Stieltjes Algorithm is used to determine the coefficients α_n and β_n, which uniquely define the set of monic orthogonal polynomials corresponding to the weight function $\mu(x)$ that is associated with the PDF $p_X(x)$ of each random variable.

In our case, if we fix $\sigma = \sigma^*$, then the weight function is given by

$$p_{Y,Z}(x, \sigma^{*2}) = \mu_{y,z}(x) = \frac{e^{-\frac{x^2}{2\sigma^{*2}}}}{\sqrt{2\pi\sigma^{*2}}},$$

where we truncate the normal distribution to be within the support $[-10, 10]$, which still integrates to 1 over the support.

Stieltjes Algorithm

The following algorithm finds $M+1$ orthogonal polynomials and the recurrence coefficients of the first M polynomials α_n, β_n which we can later use to find the gauss quadrature points which are the zeros of the polynomial of order $M+1$.

We define the inner product as

$$\langle p, q \rangle = \int_a^b p(x)q(x)\mu(x)dx$$

for numerical stability when calculating this integral if the measure $\mu(x)$ is supported on the interval

[a, b] then we map it to the standard interval [-1, 1] by using the coordinate transformation

$$x = \frac{b-a}{2}z + \frac{b+a}{2} \quad z = \frac{2}{b-a}\left(x - \frac{b+a}{2}\right)$$

where $z \in [-1, 1]$ and

$$\mu(z) = p_X\left(\frac{b-a}{2}z + \frac{b+a}{2}\right)$$

Once we have computed the Gauss quadrature points and weights, which we will outline in the next section, we can map the points back into the original domain [a, b] by performing the inverse transformation.

Given the above, we perform the following steps to calculate the family of orthogonal polynomials π_{n+1} and the coefficients α_n, β_n.

The coefficients can be computed by the ratio of inner products,

$$\alpha_n = \frac{\langle x\pi_n, \pi_n\rangle}{\langle \pi_n, \pi_n\rangle}$$
$$\beta_n = \frac{\langle \pi_n, \pi_n\rangle}{\langle \pi_{n-1}, \pi_{n-1}\rangle}$$

1. Initialize three arrays of length $M+2$ where the index of each array are from $n = 0, 1, ..., M+1$.
2. Set $n = 1$, $\beta_n = 0$, $\pi_{n-1}(x) = 0$, and $\pi_n(x) = 1$ and compute α_n.
3. Compute
$$\pi_{n+1}(x) = (x - \alpha_n)\pi_n(x) - \beta_n\pi_{n-1}(x)$$
4. With $\pi_{n+1}(x)$ and $\pi_n(x)$ compute β_{n+1}.
5. Repeat steps 2 through 4 until you have $M+1$ polynomial functions.

Gaussian Quadrature Points and Weights

With the coefficients α_n and β_n from the Stieltjes Algorithm, we can find the gauss quadrature points and weights by first constructing the following tridiagonal symmetric matrix of size $M \times M$,

$$\mathbf{S} = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & 0 & 0 & \cdots & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & 0 & \cdots & 0 \\ 0 & \sqrt{\beta_2} & \alpha_2 & \sqrt{\beta_3} & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \sqrt{\beta_{n-1}} & \alpha_{n-1} & \sqrt{\beta_n} \\ 0 & \cdots & 0 & 0 & \sqrt{\beta_n} & \alpha_n \end{bmatrix}$$

The gauss quadrature points z_j are zeros of the $M+1$ orthogonal polynomial calculated by the Stieltjes Algorithm and can be found easily by calculating the eigenvalues of the matrix $\mathbf{S}$ (Venturi, 2014b, p. 12, Sampling Methods). Each corresponding weight w_j is equal to the the first component of each normalized eigenvector $\mathbf{v}_{1,j}$ squared.

$$w_j = (\mathbf{v}_{1,j})^2 \int_{-1}^{1} \mu(z)dz$$

Probabilistic Collocation Method

Now that we have the Gauss points and weights corresponding to each random variable available, we construct a n-dimensional grid in $\mathbb{R}^{\ltimes}$ by taking the tensor product of all the points defined as

$$\{(z_{i_1}, z_{i_2}, ..., z_{i_n}) | i_1, i_2, ..., i_n \in \{1, 2, ..., M\}\}$$

So for example in our case of dimension $n = 2$ and our random variables being i.i.d. we can simply construct a matrix,

$$Y = \begin{bmatrix} z_1 & z_1 & \cdots & z_1 \\ z_2 & z_2 & \cdots & z_2 \\ \vdots & \vdots & \ddots & \vdots \\ z_j & z_j & \cdots & z_j \end{bmatrix} \quad Z = \begin{bmatrix} z_1 & z_2 & \cdots & z_M \\ z_1 & z_2 & \cdots & z_M \\ \vdots & \vdots & \ddots & \vdots \\ z_1 & z_2 & \cdots & z_M \end{bmatrix}$$

and flatten each matrix such that the columns of Y, Z are stacked on top of each other. Points in 2D would then be a Vector of size $M^2 \times 2$,

$$\mathbf{p}_i = \begin{bmatrix} Y_1 & Z_1 \\ Y_2 & Z_2 \\ \vdots & \vdots \\ Y_i & Z_i \end{bmatrix}$$

where $i = 1, 2, ..., M^2$

We can easily calculate weights corresponding to each point by taking the outer product

$$\mathbf{w}_i = w_j^T \cdot w_j$$

In Figure 1 we calculate each Gauss quad point and weight for the Normal distribution with mean

$\mu = 0.0$ and standard deviation $\sigma = 1.0$ and plot them as wind vectors emanating from the origin. Each vector is colored by its corresponding weight scaled by the natural log. Following the above procedure, we can find the relationship between the parameter σ, the standard deviation of the Normal distribution, and the magnitude of the average wind vector being applied to the drone.

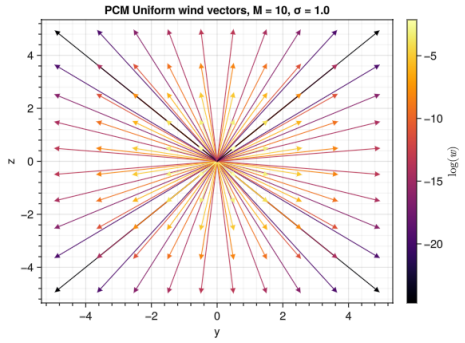

Figure 1. Gauss points derived from tensor product PCM with $M = 10$ points and fixing the normal distribution at $\mu = 0.0, \sigma = 1.0$, representing the uniform wind vector fields that we are subjecting the drone to, for each simulation. The vectors are colored by the natural log of their corresponding weights to show that the points closer to the mean have a higher contribution to the weighted average.

In Figure 2 we take find the weighted average of the Gauss quadrature points for varying σ. This shows a linear relationship between the spread of the normal distribution and the magnitude of the average uniform vector applied to the drone for all realizations.

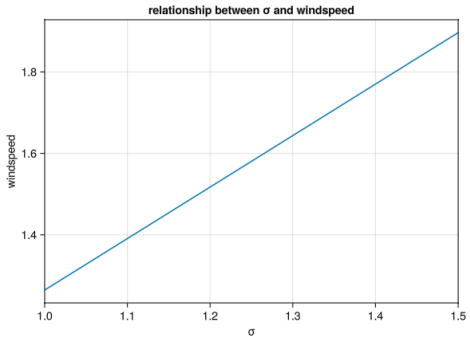

Figure 2. Linear relationship between the average magnitude of the random vector sampled from two independent Gaussian distributions with varying σ.

The equation of the line can then used to map the value of the standard deviation to the wind speed when characterizing the flight envelope of the drone.

We can calculate the mean PCM trajectory by taking the weighted average of the solution by numerically solving the initial value problem M number of times, and then calculating the weighted average of those points in each spatial dimension by,

$$\bar{x}_j = \left[\sum_{n=0}^{M} x_{j_n}(t)\right]^T \cdot \mathbf{w}_i \quad \text{for } j = 1,2,...,6$$

RESULTS

We vary the parameter σ and calculate the Gauss points and weights for each variation of the normal distribution with the number of orthogonal polynomials set to $M = 10$, then solve the initial value problem in time from $t \in [0,30]$ seconds with initial conditions $\mathbf{x}(0) = 0$ for all state variables for all Gauss points (in this case 100 times). This means the drone takes off from the origin and travels to its desired position, which we have set to $(y_d, z_d) = (1.0, 2.0)$.

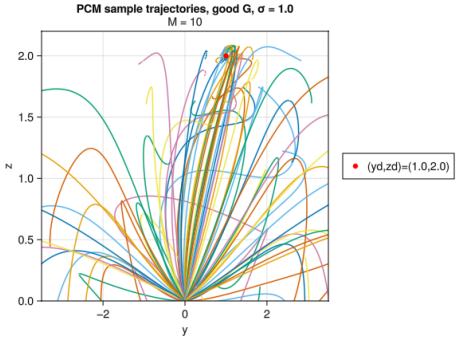

Figure 3. All 100 trajectories of the drone traveling from the origin at the initial condition, to the desired point $(y_d, z_d) = (1.0, 2.0)$ under varying uniform vector fields

In the absence of wind, the drone's path is stable and hovers at the desired coordinate position (see figure 5 in the appendix). If we set sigma to be $\sigma = 1.5$ and plot one mean PCM trajectory In Figure 3 we show each of the solution trajectories at every gauss point. We can see that with arbitrarily greater wind speeds the drone's trajectories deviates from the desired position.

Characterizing the Flight envelope

If we plot many mean trajectories in the same plot, we can see that around a σ of 0.61 corresponding to an average wind speed of 0.8 meters per second, the drone enters a flight regime where steady and

stable flight can not be guaranteed. This can be clearly seen in figure 4 where we plot many mean PCM trajectories.

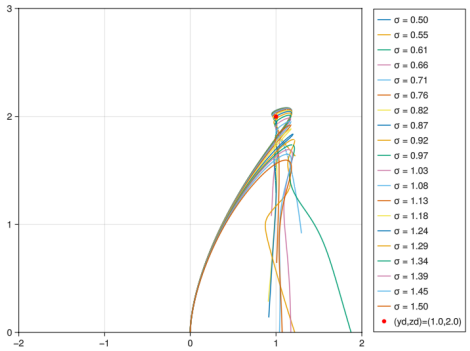

Figure 4. Mean PCM trajectories for varying sigma

DISCUSSION

This study introduced a novel method for characterizing the flight envelope of multi-rotor aircraft under the influence of random wind forcing. By employing the Probabilistic Collocation Method (PCM), we understand the impact of wind variability on the stability and control of small unmanned aircraft systems (sUAS) during takeoff and hover phases. The findings highlight key aspects of the drone's operational limits, offering a quantifiable relationship between wind variability, represented by the standard deviation σ of the normal distribution, and the drone's ability to maintain stable flight.

One of the most notable outcomes is the identification of a critical threshold for wind variability beyond which stable hovering can no longer be guaranteed. As illustrated, the drone's flight path deviates significantly from its desired position at higher values of σ, corresponding to stronger wind conditions. This method can be useful for mission planning, particularly in applications like atmospheric sampling or inspection in unpredictable weather conditions.

The use of PCM provided a computationally efficient means of estimating the mean and variance of the drone's trajectory, avoiding the need for exhaustive Monte Carlo simulations. This method's efficiency is particularly advantageous when assessing multiple flight configurations or testing different controller designs. By leveraging orthogonal polynomials and Gauss quadrature points, we achieved precise statistical characterizations of the system's response to random forcing.

For future work, we would like to introduce more variability into the system and determine how factors such as the weight of the drone, or changes to the mass moment of inertia effect the drones dynamics under uncertain wind conditions. This approach would allow us to systematically exploration of how changes in the control system parameters, payload weight, and aerodynamic coefficients influence the drone's stability under random wind forces.

REFERENCES

Foster, J. V. and Hartman, D. (2017). *High-Fidelity Multi-Rotor Unmanned Aircraft System (UAS) Simulation Development for Trajectory Prediction Under Off-Nominal Flight Dynamics*, chapter 3271. American Institute of Aeronautics and Astronautics.

González-Rocha, J., Woolsey, C. A., Sultan, C., and De Wekker, S. F. J. (2019). Sensing wind from quadrotor motion. *Journal of Guidance, Control, and Dynamics*, 42(4):836–852.

Neumann, P. P. and Bartholmai, M. (2015). Real-time wind estimation on a micro unmanned aerial vehicle using its inertial measurement unit. *Sensors and Actuators A: Physical*, 235:300–310.

Venturi, D. (2014a). Lecture notes am238, polynomial chaos. Canvas, University of California Santa Cruz.

Venturi, D. (2014b). Lecture notes am238, sampling methods. Canvas, University of California Santa Cruz.

APPENDIX

Figures

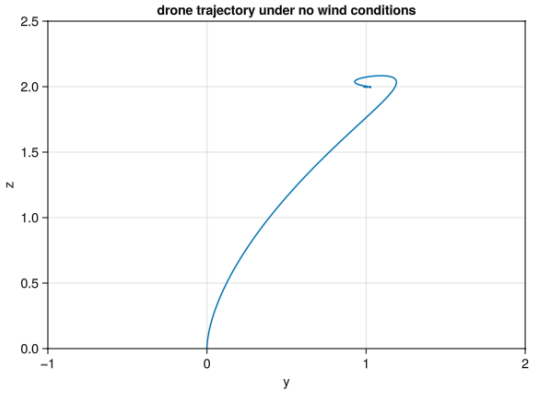

Figure 5. A single drone trajectory under no wind conditions.

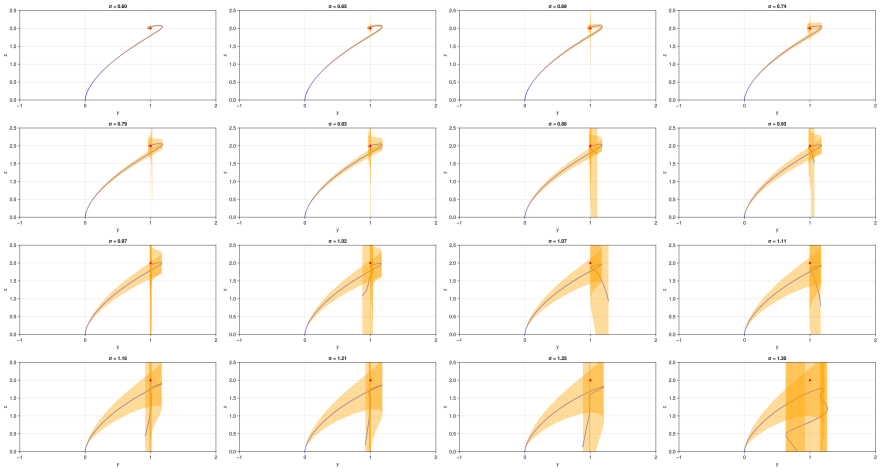

Figure 6. Mean PCM trajectories of the drone with varying σ in blue, and the trajectory variance in orange.

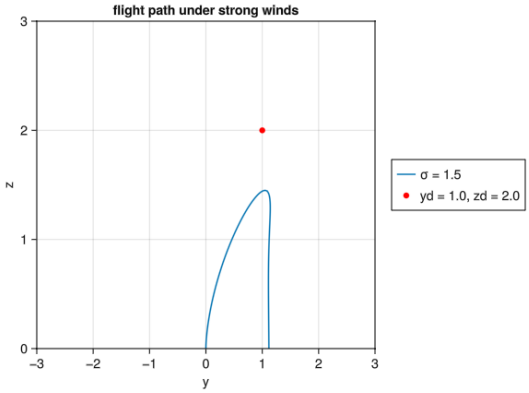

Figure 7. A mean PCM trajectory under strong wind conditions ($\sigma = 1.5$)

Equations

$$u_{1_{ctrl}} = -F_{d_z} + mg + m\left(K_{p_z}(z_d - x_2) + K_{i_z}\left(\int_{t-\varepsilon}^{t} z_d dt - \int_{t-\varepsilon}^{t} x_2 dt\right) - K_{d_z}x_5\right) \tag{1}$$

$$u_{2_{ctrl}} = I_{xx}\left(K_{p_\phi}(\phi_d - x_3) + K_{i_\phi}\left(\int_{t-\varepsilon}^{t} \phi_d dt - \int_{t-\varepsilon}^{t} x_3 dt\right) - K_{d_\phi}x_6\right) \tag{2}$$

$$x_{3_{ctrl}} = \phi_{ctrl} = \phi_d = \frac{1}{g}\left(\frac{F_{d_y}}{m} - \left(K_{p_y}(y_d - x_1)\right) + K_{i_y}\left(\int_{t-\varepsilon}^{t} y_d dt - \int_{t-\varepsilon}^{t} x_1 dt\right) - K_{d_y}x_4\right) \tag{3}$$

Code

```
using DifferentialEquations
using DynamicalSystems
using Printf
using GLMakie
using QuadGK
using StaticArrays
using LinearAlgebra
using Statistics

# weight function
Normal(x, u, sigma) = exp(-(x - u)^2/(2*sigma^2))/sqrt(2.0*pi*sigma^2)

# define an integral using gauss-kronrod quadrature rule
integ(x::Function, sup::SVector{2, Float64}) = quadgk(x, sup[1], sup[2];
 ↪  atol=1e-8, rtol=1e-8)[1]

function stieltjes(mu::Function, N::Int64, sup::SVector{2, Float64})
    M = N + 2  # Extend size to accommodate buffer
    n = 2      # Starting index for the recursion

    # Initialize orthogonal polynomials (pi_n) as functions
    pi = Vector{Function}(undef, M)
    pi[n-1] = x -> 0.0 * x^0.0  # pi_0(x) = 0
    pi[n] = x -> 1.0 * x^0.0    # pi_1(x) = 1

    # Initialize coefficient vectors alpha_n and beta_n
    alpha = Vector{Float64}(undef, M)
    beta = Vector{Float64}(undef, M)

    # Compute the first alpha coefficient (alpha_2)
    # alpha_2 = <x pi_1, pi_1> / <pi_1, pi_1>
    alpha[n] = integ(x -> x * pi[n](x) * pi[n](x) * mu(x), sup) / integ(x ->
     ↪  pi[n](x) * pi[n](x) * mu(x), sup)

    # Compute the next orthogonal polynomial pi_2
    # pi_2(x) = (x - alpha_1)pi_1(x) - beta_1pi_0(x)
    pi[n+1] = x -> (x - alpha[n]) * pi[n](x)

    for n in 3:M-1
        alpha[n] = integ(x -> x * pi[n](x) * pi[n](x) * mu(x), sup) /
         ↪  integ(x -> pi[n](x) * pi[n](x) * mu(x), sup)
        beta[n] = integ(x -> pi[n](x) * pi[n](x) * mu(x), sup) / integ(x ->
         ↪  pi[n-1](x) * pi[n-1](x) * mu(x), sup)
        pi[n+1] = x -> (x - alpha[n]) * pi[n](x) - beta[n] * pi[n-1](x)
    end
    return pi, alpha, beta
end
```

```
function guassQuad(mu::Function, M::Int64, sup::SVector{2, Float64})
    # get polynomials, alpha, and beta coefficients of the weight function
     ↪  mu
    polynomials, alpha, beta = stieltjes(mu, M, sup)

    # construct tridiagonal S Matrix
    S = Matrix{Float64}(undef, M, M)
    for idx in CartesianIndices(S)
        i, j = idx.I
        if abs(i - j) <= 1
            if i == j
                S[idx] = alpha[i + 1]
            elseif i == j - 1
                S[idx] = sqrt(beta[j + 1])
            elseif i == j + 1
                S[idx] = sqrt(beta[i + 1])
            end
        else
            S[idx] = 0.0
        end
    end
    return polynomials, eigvals(S), eigvecs(S)[1, :] .^ 2
end

function hermitePCM(M::Int64, mu::Float64, sigma::Float64)
    _, p, w = guassQuad(x->Normal(x, mu, sigma), M, SA[-10.0, 10.0])
    X = repeat(p, 1, M)
    Y = repeat(p', M, 1)
    # flatten the points and weights
    return hcat(vec(X), vec(Y)), vec(w*w')
end


getParams(windy::Float64, windz::Float64, Kpy::Float64, Kiy::Float64,
 ↪  Kdy::Float64, Kpz::Float64, Kiz::Float64, Kdz::Float64, Kpphi::Float64,
 ↪  Kiphi::Float64, Kdphi::Float64) = SVector{18, Float64}(
    9.81,     # gravity
    0.91,     # mass
    0.056875,# mass moment of inertia
    0.25,     # length
    5.328,    # max thrust
    0.25,     # drag coefficient
    1.0,      # integration time delay
    windy,
    windz,
    Kpy, Kiy, Kdy,
    Kpz, Kiz, Kdz,
    Kpphi, Kiphi, Kdphi
)

getInit() = SVector{6, Float64}(
    0.0,     # initial y position
    0.0,     # initial z position
    0.0,     # initial tilt position
    0.0,     # initial velocity in y
    0.0,     # initial velocity in z
    0.0      # initial velocity in phi
)

getDesired() = SVector{2, Float64}(
```

```
    1.0,    # yd
    2.0     # zd
)

int(x::Float64, t::Float64, eps::Float64) = quadgk(z -> x, t - eps, t, atol
 ↪  = 1e-6, rtol = 1e-6)[1]

getDrag(Cd::Float64, dy::Float64, dz::Float64, vy::Float64, vz::Float64) =
 ↪  SVector{2, Float64}(
    -Cd*(dy - vy)*sqrt((dy - vy)^2 + (dz - vz)^2),
    -Cd*(dz - vz)*sqrt((dy - vy)^2 + (dz - vz)^2)
)

function droneRule(u, p, t)
    yd, zd = getDesired()
    Fdy, Fdz = getDrag(p[6], u[4], u[5], p[8], p[9])
    # control law
    phid = (1.0/p[1])*((Fdy/p[2]) - (p[10]*(yd - u[1]) + p[11]*(int(yd, t,
     ↪  p[7]) - int(u[1], t, p[7])) - p[12]*u[4]))
    u1_ctrl = -Fdz + p[2]*p[1] + p[2]*(p[13]*(zd - u[2]) + p[14]*(int(zd, t,
     ↪  p[7]) - int(u[2], t, p[7])) - p[15]*u[5])
    u2_ctrl = p[3]*(p[16]*(phid - u[3]) + p[17]*(int(phid, t, p[7]) -
     ↪  int(u[3], t, p[7])) - p[18]*u[6])

    # clamp to max thrust
    TL = clamp(0.5*(u1_ctrl - u2_ctrl/p[4]), 0.0, p[5])
    TR = clamp(0.5*(u1_ctrl + u2_ctrl/p[4]), 0.0, p[5])
    u1 = TL + TR
    u2 = (TR - TL)*p[4]

    # outputs
    dx4 = (1.0/p[2])*(Fdy - u1*sin(u[3]))
    dx5 = (1.0/p[2])*(Fdz + u1*cos(u[3]) - (p[2]*p[1]))
    dx6 = (u2/p[3])
    return SVector{6, Float64}(u[4], u[5], u[6], dx4, dx5, dx6)
end

function solsize()
    # solve problem once to get size of solution
    prob = ODEProblem(droneRule, getInit(), (0.0, 30.0), getParams(0.0, 0.0,
     ↪  0.4, 0.1, 1.0, 0.4, 0.1, 1.0, 18.0, 1.0, 15.0))
    sol = solve(prob, Tsit5(), dt=1e-3, adaptive = false, abstol = 1e-8,
     ↪  reltol = 1e-8)
    length(sol.t)
end

function PCM(points::Matrix{Float64}, weights::Vector{Float64},
 ↪  gains::SVector{9, Float64}, Z::Int64)
    # create a vector of the same size of solution
    ysol = Matrix{Float64}(undef, size(points)[1], Z)
    zsol = Matrix{Float64}(undef, size(points)[1], Z)

    # solve the problem M^2 times
    for i in 1:size(points)[1]
        prob = ODEProblem(droneRule, getInit(), (0.0, 30.0),
         ↪  getParams(points[i, 1] ,points[i, 2], gains...))
        sol = solve(prob, Tsit5(), dt=1e-3, adaptive = false, abstol = 1e-8,
         ↪  reltol = 1e-8)
        ysol[i, :] = sol[1, :]
        zsol[i, :] = sol[2, :]
```

```
    end

    muy, muz = SA[ysol' * weights, zsol' * weights]
    sigmay, sigmaz = SA[((ysol .- muy').^2)' * weights, ((zsol .- muz').^2)'
     ↪  * weights]
    SA[muy, muz, sqrt.(sigmay), sqrt.(sigmaz)]
end

function buildPDF()
    Z = solsize()
    n = 4
    sigmas = LinRange(0.6, 1.3, n^2)
    M = 10
    yd, zd = getDesired()

    f = Figure()
    for i in 1:n
        for j in 1:n
            k = (i-1)*n+j
            ax = Axis(f[i, j],
                title = "sigma = $(@sprintf("%.2f", sigmas[k]))",
                xlabel="y",
                ylabel="z"
            )
            xlims!(ax, -1.0, 2.0)
            ylims!(ax, 0.0, 2.5)
            p, w = hermitePCM(M, 0.0, sigmas[k])
            muy, muz, sigmay, sigmaz = PCM(p, w, SA[0.4, 0.1, 1.0, 0.4, 0.1,
             ↪  1.0, 18.0, 1.0, 15.0], Z)
            lines!(ax, muy, muz, color = :blue)
            band!(ax, muy, muz.-sigmaz, muz.+sigmaz, alpha=0.4, color =
             ↪  :orange)
            scatter!(ax, yd, zd, color = :red)
        end
    end
    f
end
```