**1. PID:**

* PID control system is needed when some state variable is needed to be precisely set to a given final state
* It uses a simple feedback mechanism which tells the controller how much *output* to generate given only current state and final state
* 3 things are inferred from the given data - *current* error (proportional), *accumulated* error (integral), and *time rate* of error (derivative
* Using a linear combination of these values, a value is generated which is used to control the *control* variable, which often is the second time derivative of the state variable (for distance as state variable, control variable is acceleration)

**2. 2D Multirotor Control:**

* This was a very interesting problem to understand and solve (*as much as I could)*; it consists of just a drone in 2 dimensions which has 2 rotors
* The relevant dimensions which were given are - mass of drone, moment of inertia of drone about X axis, length of arms of the drone and of course the gravitational acceleration constant
* The relevant variables for my use to design the PID control system were - the current *state*, the destination *state,* the global *parameters* and the control variable*s* of *thrust* and *central moment about X axis* - inside the ‘controller.m’ file
* The current state was defined by the y, z and phi positions along with their rates of change, and the destination state was defined by the y and z positions along with their single and double time rates of change
* With all of this given, here is the final sketch of my solution:
  + Model out the physics of the problem (I hadn’t done this thoroughly in the seesaw problem) on paper and understand the relation between the variables and the functioning of the drone in terms of error analysis
  + Keep track of what differences are actually needed - between positions (y and z), between velocities (y and z), between *desired* and *current* angles (phi), and between rates of change of *desired* and *current* angles (phi) [**Note:** Desired angle is *not* given, it has been calculated also using PID]
  + In order, create 3 separate PID variables, PIDZ, PIDY and PIDPhi, and in them, store calculated PID values of their respective variables
  + With the physics done, it was determined that PIDZ uses error in Z and directly maps to *Thrust* output; PIDY uses error in Y and *calculates* the *desired* phi which is an input to PIDPhi; PIDPhi uses error in Phi and directly maps to *Moment* output
  + The sketch is now made in code too, after this only to tune the parameters one by one (6 parameters here)
  + A basic principle I applied was to tune it to *traj\_step* to see its basic functioning and then extend it bit by bit to the other trajectories provided

**3. Final Solution Intricacies:**

* Once again, my solution uses the given parameters to their greatest extent, making only the independent terms (acceleration of positions and angle) PID controlled, keeping mass and such terms independent, making it valid for any sort of change in the *sys\_params.m* file (*except gravity, once again don't change what nature has to offer you :P)*:
  + Each PID is the double time rate of change of its assigned variable, hence making the system independent and also making it all easily understandable [**Note:** This is done because a second order linear homogeneous differential equation of error is made, with coefficients of Kp and Kd; here the choice of Kp and Kd which converges error to 0 the fast is the solution]
  + The *Thrust* value has a minimum *MG/cos(phi)* term, which balances out the weight of the drone, along with it has the PIDZ term scaled by *M/cos(phi)*
  + The *Moment* value has 0 as minimum (since in stable equilibrium the drone is *not* rolling constantly), and along with it *Ixx\*PIDPhi,* since its directly proportional to the angular acceleration
  + The physics of the entire system has been explained in a separate image, the code is only an implementation of that model
  + The final solution has very minimal error for step and line trajectories, small deviations in the diamond and sine trajectories
  + It is also very fast, solves in under 4 seconds for the step, sine and line trajectories and under 10 seconds for the diamond trajectory [**Note:** Sample time in the images and the code itself is 10 seconds, because diamond trajectory was unable to complete in less than that time]

**4. Problems Faced (and solutions):**

* This task was very interesting to begin and do, but understanding the problem and to make use of the given information was a bit tough, and the tuning part was hell (to begin with)
* **Problem 1 - Understanding the given Files:**
  + Seeing a much higher number of files as compared to the seesaw task, it was a lot more confusing to browse through and understand what the problem statement was or which file to run first
  + Finally, I understood the basic structure of the entire given content, the basic structure of the drone and how to run the simulation
* **Problem 2 - Modeling the physics of the Drone:**
  + This was one of the hardest steps, because i kept getting stuck at one or the other point due to silly mistakes in calculating or due to lack of understanding *why* such a solution came out to be
  + A lot of googling and browsing through courses on Aerial Robotics, I came across some great explanations and models for a drone system in 2 dimensions
  + Using those, I confirmed the equations that I had derived by hand and proceeded to model the control variables to the PID values in theory
  + The point to note in this step was, why the theory talked about linearising the error function or the equations of motion, ofc in small angles approximation is good, but the angles weren’t all small to begin or end with, so I decided to use the equations as it is without forcible linearisation, which may be a cause of error too
* **Problem 3 - Modeling Control variables to PID:**
  + The main problem in this step was figuring out *how* am i supposed to use the given data to make the system work
  + Firstly, I had to understand what exactly does the PID map to, as in, for a state variable, what is the PID value’s significance, and how exactly does it map to the control variables
  + So, the PD (no integral) system is when we have a homogeneous second order differential of error with Kp and Kd as coefficients of *error* and *time rate of error*
  + In this, the second time rate of error is *desired double time rate of* ***state*** *- variable double time rate of* ***state***
  + This variable double time rate of state is basically what the PD system outputs, and this value is mapped out to the Control variable the mapping of which is found through modeling the system
  + After understanding this PD mapping, I had to decide what values go where, and how to find the missing values
  + The value of *phi\_c\_dot* was chosen to be 0 by me [*this variable is used in the physics image that i have attached*] by the logic that the commanded value of *phi* is constant, why would there be any desired rate of change in the desired final position of *phi* itself, as it was just another way to express the Y part of the system to the drone [*this assumption may be wrong, I am not too sure, but the final PID system worked for the most part]*
  + This left me with 6 variables to tune, 2 (Kp and Kd) for each Y, Z and Phi
* **Problem 4 - Tuning:**
  + Initially the tuning was completely blind in nature, with random increments to random constants, which led me to absolutely nothing
  + 2 things helped me in tuning (after hours of trying blind tuning), choosing *traj\_step* as the initial trajectory, and making *t\_total* 20 seconds in the *simulation\_2d.m* file
  + The step trajectory was a very basic point to point PID system so it was *comparatively* easy to play around with the tuning values for that, and since it was constrained motion in only Y axis, I could leave the PID constants for Z axis as 0 while I tuned the other 4
  + Giving myself additional time also helped me look at whether the given set of constant values converged or diverged the system to the desired point and basically make use of graph as an imaging tool for myself
  + Slowly I played around with constants for Y and Phi, and got satisfactory results
  + However a point of concern which still remains is that the constants of Z did not do much to alter the path in the Z axis for sine and diamond trajectories, all it did was help me remove the ‘*No complex values allowed*’ error; this may be causing some error in the functioning of the drone, but it wasn’t really visible as much and the final cumulative error result were also satisfactory

**5. References:**

* <https://ieeexplore.ieee.org/document/9909194>
* <https://www.coursera.org/learn/robotics-flight>
* <https://eudl.eu/pdf/10.4108/eetct.v9i31.1922>
* <https://liu.diva-portal.org/smash/get/diva2:1129641/FULLTEXT01.pdf>