**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. Seesaw Control:**

* The problem scene simply consists of a seesaw, a ball, and the torque generator at the pivotal center of the seesaw
* The relevant dimensions are given - mass of ball and seesaw, radius of ball, length of each arm of the seesaw and of course the gravitational acceleration constant
* The variables I played around with are the - current and final *state* of the ball-seesaw set, the global *parameters* which were set and the control variable of *torque* - inside the ‘control.m’ file given
* The *state* has been defined by 4 variables - position and velocity of the ball along the seesaw, and angle and angular velocity of seesaw about the pivotal center
* A simple sketch of the solution is as follows:
  + Create an error variable containing the difference between the current and desired states
  + Create a proportional variable which is a linear combination of error in position and angle; similarly create a derivative variable which is linear combination of error in velocity and angular velocity
  + Their sum gives the PID control value, which was taken as angular acceleration in the solution
  + The final output torque is sum of minimum torque required to keep ball at rest (M\*G\*X) and the control output scaled as torque using moment of inertia (I\*PID)
  + Finally, individual parameters were tuned to actually make the pid control work
  + **Note:** There is no Integral term in the PID system i made, because i was unable to store past values, however as a *jugaad* integral term, the minimum torque to keep ball at rest was used which varies with distance as it is

**3. Final Solution Intricacies:**

* My solution to the problem uses the given parameters to the greatest extent, that is, even changing values of given parameters in the *params.m* file will not have effect on the proper functioning of the PID control made (*except changing g, don't change what earth itself is giving you :P)* :
  + This was done by scaling the PID value by I (moment of inertia of ball-seesaw system *at* the specific moment), essentially making PID the angular acceleration of the system at the pivot
  + The final output torque had this scaled PID value (actually PD) taking care of the *proportional* and *derivative* part of the control system, and a *tauExternal* value, which is the torque exerted onto the seesaw by the ball, which acts as the *jugaad* *integral* term
  + Out of all this, the only *empirical* values which I have used are the constants for the PID
* The code of my solution is also in a neat matrix sort of manner, I don’t really know why I did that or why I am mentioning that, but matrix multiplication seemed like a cool thing to do so I did it anyways (the details are commented inside the *controls.m* file)
* It also simply doesn’t suffice to say *“yes i tuned the controller, now it works”* because the hard part *was* tuning the controller:
  + Starting with only proportional terms (X and Theta), I tuned them in such a manner that successful oscillations were able to take place at a reasonable speed
  + Next, I added the derivative terms one at a time (V then Omega), which made the ball stop at the desired location, albeit at a slow rate initially
  + Final step was simply tuning up/down the constants to make individual error terms as negligible as possible, which was done simply by looking at how the errors changed on changing individual constants

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

* The task seemed simple enough to begin with, but upon beginning it, there were lots of problems at hand, some simple, some hard, some mind numbingly boring due to repetitive nature *(tuning)*
* **Problem 1 - Downloading the Files given:**
  + After downloading and getting familiar with MATLAB, I started googling what *control.m, params.m,* etc mean in MATLAB
  + Having found no results online (because these are locally made files), I realized these must be files must be there on Teams, which they were
* **Problem 2 - Modeling the output Torque:**
  + From the beginning I realized that there must be a component which has *minimum* torque needed to balance the ball at the desired location, because at some point PID value will tend towards 0 (due to lack of *integral* term)
  + Hence I chose that as the *central* part of the output torque, which created lots of problems for me initially
  + My output ‘u’ looked like *tauExternal\*(1-PID)* which I thought would yield appropriate outputs (which it still might because there were several other problems at hand while i modeled the Torque which were later solved) thinking that “*we need* at least *tauExternal amount, and as long as there is error, apply more/less torque according to the direction of the error in order to push the ball in desired direction”*
  + Similarly other models of ‘u’ looked like *tauExternal - tauMax\*PID (*scaled value of output torque needed to balance ball at the end of the seesaw), etc
  + Finally I thought of some way to make PID constants more general by considering making PID itself the *angular acceleration* of the system at the pivot, so no matter what change occurs in mass, radius, etc, the tuning will be appropriate, hence ‘u’ looked finally looked like *tauExternal - I\*PID*
  + This initially did not work either but was more general looking than the other attempts so I stuck with it
* **Problem 3 - Proportional Tuning:**
  + With the model of torque ready, I started tuning the proportional constant for X *alone*; this of course lead to disaster but I couldn’t make sense of it - the ball was either moving too fast towards the desired location and then jumping out of the seesaw, or moving slowly towards it and *still* jumping out of the seesaw
  + Spending hours on *KpX* alone, I realized that this must be because Theta is also something to consider, because when error is small, net excess torque is small too, but the angle made due to the previous application of torque is still significant to the ball, hence to set that back a Theta proportional *KpTheta* was needed
  + This seemed to work in making the system oscillate between 2 points with the desired state in the middle, and with some tinkering around i managed to make the oscillations as fast as possible without a large offset in positions
* **Problem 4 - Derivative Tuning:**
  + Slowly I added the derivative terms too, starting with *KdX* term, this made the system tend towards the desired location but again was very slow in doing so, I had to increase the time span significantly to see it finally converging
  + Increasing *KdX* did improve but the system was still oscillating significantly without much damping, so I decided to add the *KdTheta* constant for Theta to try and make it ‘even’ and that helped a lot
  + Turns out, just how there needs to be a look out for how fast X is changing, there needs to be one for Theta too to make the system more damped
  + Finally, individually tuning each parameter led to quite an accurate control system
* **Problem 5 - Moment of Inertia:**
  + After having a very accurately tuned PID system, I checked it for various start points, all of which led to good results with error in orders of 10-2 , however the specific distance I used to tune had error in orders of 10-4 which left me dissatisfied
  + This was also problematic as I expected a smaller travel distance to have less error, but it had more error than the calibrated distance (albeit negligibly more)
  + I realized that the problem was with the *I* term I used in the expression for ‘u’, initially it was the *maximum* Moment of Inertia of the system, I made it to the *instantaneous* Moment of Inertia and re-tuned the parameters
  + This helped me minimize error for all values, and led to an expected decrease in error when considering smaller distances

**5. References:**

* <https://in.mathworks.com/help/matlab/ref/ode45.html#bu00_4l_sep_shared-odefun>
* <https://www.santoshrajkumar.com/files_public/projects/seesaw/see_saw_control.pdf>
* <https://www.norwegiancreations.com/2016/08/the-seesaw-part-2-basic-pid-theory-and-arduino-implementation/>
* <https://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method>
* <https://hackaday.com/2019/07/31/ping-pong-ball-makes-great-pid-example/>
* <https://roble.uno/control-pid-barra-y-bola-arduino/?v=028a54c4d521>
* <https://www.youtube.com/playlist?list=PL7CAABC40B2825C8B>