CS 7638: Artificial Intelligence for Robotics

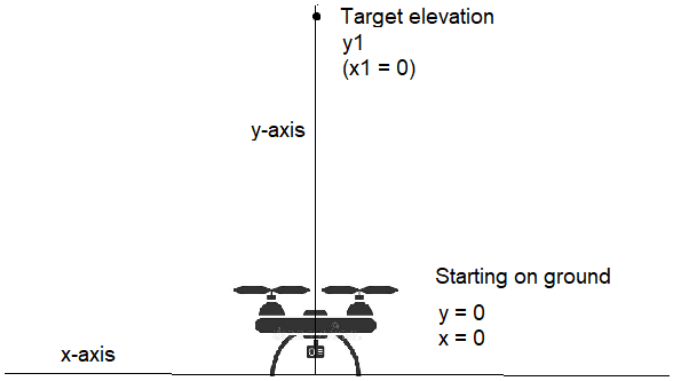
Drone Control (PID) Project

Fall 2021 - Deadline: Monday Nov 1st, Midnight AOE

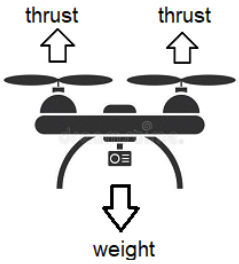
Project Description

A PID controller is a component of a feedback control system. The system (usually called "plant") is supposed to be maintained at a target "steady" state. For example, a car that needs to stay in the middle of a lane, or a thermostat that needs to maintain a particular temperature, etc. In this project you will be implementing a PID controller to guide a drone to a target elevation and horizontal position and keep it hovering at the target position for a specified time. The Drone starts on the ground with its rotors off. For simplicity we only consider a two dimensional world, and a dual rotor drone. We also don't consider any air drag or ground effects (https://www.rchelicopterfun.com/ground-effect.html) while simulating the drone.

Consider the simple case first, where the Drone only has to lift off vertically and achieve a target elevation and hover there.

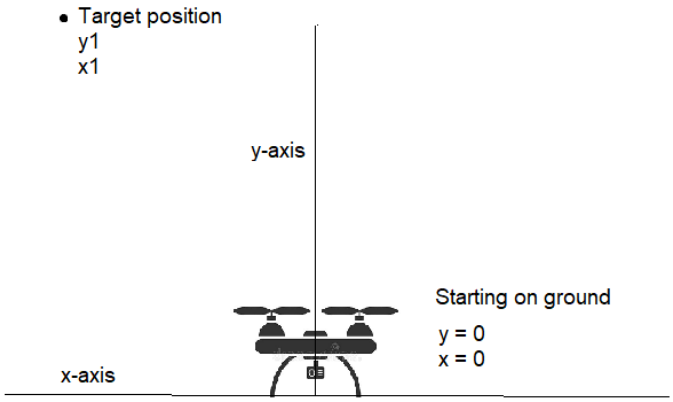


We have to induce a certain rpm (rotations per minute) equally in both the rotors. As the rpm of its rotors increase, they push the air down and this exerts a net upward force (thrust) on the Drone.

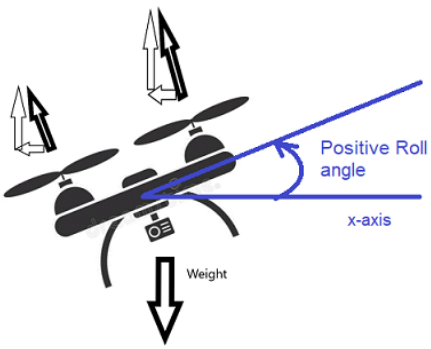


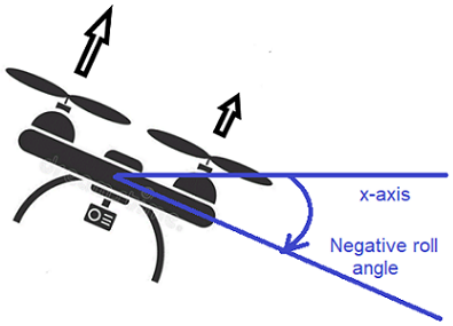
When this force equals the weight of the Drone, it starts to hover. Increasing the rpm beyond this point will lift the Drone up. To reach a certain elevation within some limited time, the thrust has to be sufficiently high. But as the drone nears the target, we have to start reducing the thrust until it is back equal to its weight ideally right at the target elevation. There, we have to maintain this thrust in order for it to hover for the desired period.

Now let's consider the situation where we also want to change the horizontal position of the drone, besides giving it some elevation.



For this we need to tilt the drone sideways, which is called Roll (measured in angle). This is done by varying the thrust (or rpm) between the rotors of the Drone. In a Dual Rotor Drone, if thrust on the right side is higher, the Drone will roll left. If thrust on the left side is higher, then the Drone will roll right.





When the drone is at rest, it has a roll angle of zero. A positive roll angle corresponds to a leftward tilt, and a negative roll angle corresponds to a rightward tilt.

Once the drone rolls in the appropriate direction, the horizontal component of thrust moves the drone in the horizontal direction. The Drone needs to have enough roll to reach the target x1 location in a specified timeframe (specified by the test case). Note that the vertical components of the thrusts keep the drone levitating. Combined, they should equal or exceed the weight of the drone to keep it hovering, or lift it up to the desired height.

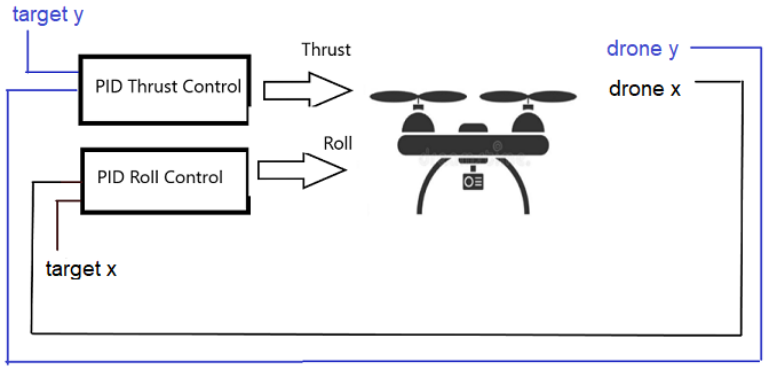
As it reaches close to the target x1, we need to reduce the roll until it reaches 0 degree, ideally exactly at the target. Then it has to hover there for a specified time. Note that in order to roll and move side-ways, the Drone needs to be at some elevation, so roll is applied along with some thrust.

We only need to return a single thrust and roll angle from the PID controller, and the simulator will take care of inducing the appropriate rpms in the rotors. If the roll angle is zero, the thrust is going to be equally distributed. Otherwise, it will be distributed according to the roll angle.

You will implement the PID controllers for thrust and roll in the drone_pid.py file. You will be provided a target elevation for the PID Thrust controller, and a target horizontal position for the PID Roll controller.

As a reminder, the PID control value (alpha) is calculated by the following formula:

alpha = tau_p * error + tau_i * error_integral + tau_d * error_derivative



There are three parts and six functions provided in the drone_pid class:

Part 1

part_1_a_pd_thrust - In this function, you have to implement the code for a PD controller for Thrust, but you don't have to find or tune the gain values for P and D. The test suite will call this function with pre-tuned gain values for specific target elevations, and your code is evaluated for correctness. In essence, here we save you the work of finding the optimal values. It is important to note that what you return from this function is the change in thrust, not a target thrust. The function is called from our Drone Simulator in a loop. At every step, there is a max change in rpm that can be applied by the Drone. This can be a positive or negative change. If the value returned by your implementation is going to cause the rpm to change outside of the max possible change in rpm per timestep, it is clipped at the max (or min if negative).

part_1_b_pd_roll - In this function, you have to implement the code for the PD controller for Roll. You don't have to find the P and D gain values in this function, they will be provided by the test suite. The return value is the change in roll angle. The function is called in a loop from the simulator. The change in roll at each step is also constrained by the max and min change in rpm at each rotor. As roll can't be applied without some elevation, this function is tested along with the part_1_a_pid_thrust function from the test suite.

Part 2

part_2_a_pd_thrust_only - This is the same as part_1_a_pd_thrust, except here the test suite will not pass the gain values. Instead you are expected to find and tune the P and D gain values. Your code is NOT expected to automatically find/tune these values, but you should find/tune the values by hand and hard code them into your function. This function is called alone by the test suite, i.e., roll is not tested with it.

part_2_b_pd_thrust - This function is the same as above, except that it will be tested with part_2_b_pd_roll function below. So you may find that you have to tune different values for the P and D gains for this function. Your code is NOT expected to automatically find/tune these values, but you should find/tune the values by hand and hard code them into your function.

part_2_b_pd_roll - This function returns the roll angle for the Drone. In this function, you will need to find and tune the values of P, I and D gains. The simulator calls this function along with part_2_b_pd_thrust function. You need to find and tune the P and D gain values for this function. Your code is NOT expected to automatically find/tune these values, but you should find/tune the values by hand and hard code them into your function.

Part 3

part_3_pid_thrust - In this function, you have to return a change in thrust, but you will have to find and tune an Integral gain value too, besides P and D values. In this function the drone has an error in its motor

which causes it to add a constant 2 rpms every time its thrust is changed. Again, your code is NOT expected to automatically find/tune the gain values for P, I and D, but you should find/tune the values by hand and hard code them into your function.

Control Saturation and Integral Wind-up

This section is particularly applicable to Part 3. In a feedback control system, control saturation occurs when an actuator (e.g. motor or steering angle) has reached its limit, but the steady state (i.e. the desired target, like target x, y position for drone) has not been reached. So the controller keeps increasing the output of the actuator. This in itself may not be a problem because many real systems will simply ignore the additional commands. However the problem occurs if there is an integral term in the controller. It will continue to accumulate more and more error (called Integral windup), as it is not able to release any of this error because the actuator is operating at its limit. When the system reaches the steady state, the error for P term is zero which will try to lower the actuator output. However, now the Integral term begins to unwind and it will continue to move the actuator in the same direction and take the system beyond the steady state. To handle this situation, you need to check if the control output is saturated and handle the integral term appropriately. In this assignment, we pass you a flag to indicate if control is saturated, but you will need to check for it in your code and handle the integral term appropriately. *Note*: This is not a requirement to get above 90% grade. You should be able to get a score of 90 or more if you just use an appropriate P, I and D gain values, and you score well on the other parts.

Submitting your Assignment

Your submission will consist of the drone_pid.py file (only) which will be uploaded to Canvas->GradeScope. Do not archive (zip,tar,etc) it. Your code must be valid python version 3 code.

We ask that you keep any testing code in a separate file that you do not submit. Your code should also NOT display a GUI or Visualization when we import or call your function under test.

Calculating your score

The max score for the entire project is 100. There are 5 test suites. Each test suite carries a max weighted score of 20 (1/5th of the total score). Each test suite has multiple test cases.

Each test case within a suite will have a continuous score (expressed as a percentage) depending on how close your drone is to the target position. If your drone position is within the error specified for that test case for the specified duration, you will get 100% for that test case. For each test suite, you will receive the average score of all test cases in the suite, weighted by 1/5. To get the full score of 20 for a test suite, you need to get 100% in each test case within that suite. *NOTE*: It is possible to get extra points in this assignment if your error is below the specified error for each test case. The max total score is capped at 101 for the whole project.

Testing Your Code

NOTE: The test cases in this project are subject to change.

We have provided a testing suite similar to the one we'll be using for grading the project, which you can use to ensure your code is working correctly.

We are using the Canvas->GradeScope autograder system which allows you to upload and grade your assignment with a remote / online autograder. You must submit your `drone_pid.py` file to Gradescope to receive credit.

We **are likely, but not required, to use the last grade you receive, (or your selected grade) via the Canvas->GradeScope autograder as your final grade** at our discretion. (See the "Online Grading" section of the Syllabus.)

Academic Integrity

You must write the code for this project alone. While you may make limited usage of outside resources, keep in mind that you must cite any such resources you use in your work (for example, you should use comments to denote a snippet of code obtained from StackOverflow, lecture videos, etc).

You must not use anybody else's code for this project in your work. We will use code-similarity detection software to identify suspicious code, and we will refer any potential incidents to the Office of Student Integrity for investigation. Moreover, you must not post your work on a publicly accessible repository; this could also result in an Honor Code violation [if another student turns in your code]. (Consider using the GT provided Github repository or a repo such as Bitbucket that doesn't default to public sharing.)