Controller Project
Ashutosh Zade
May 29, 2018

3D Control Architecture

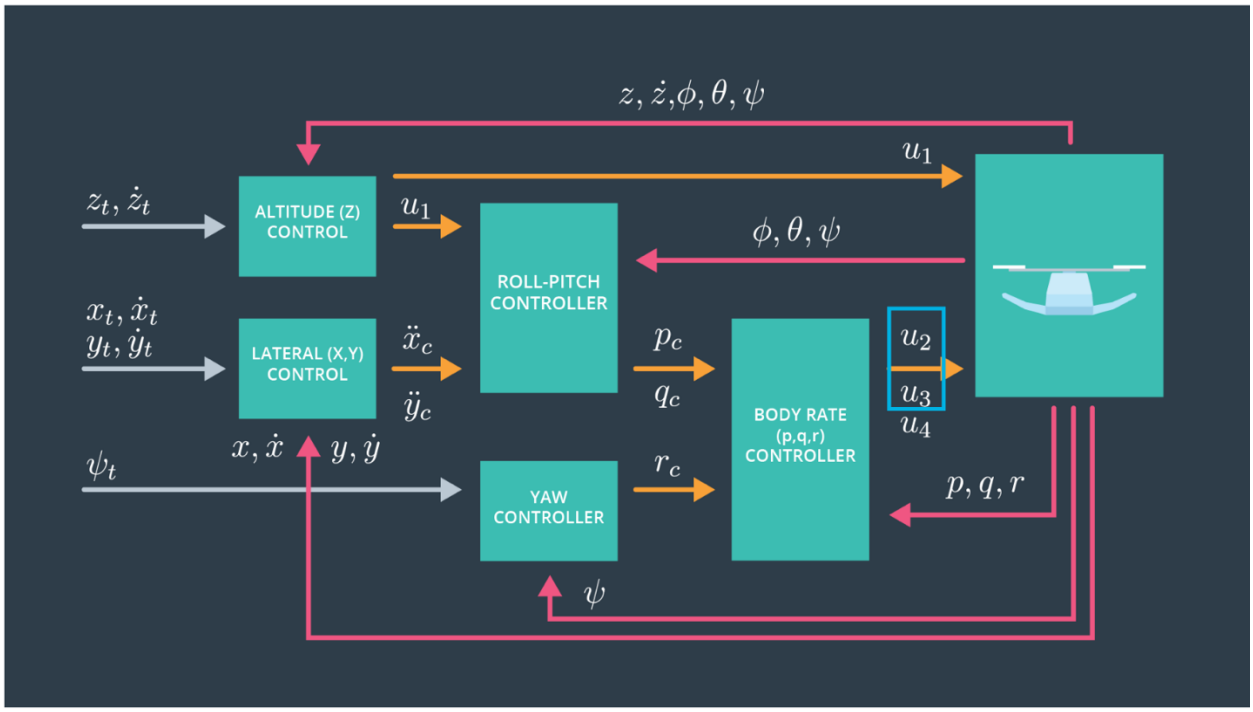

Figure (1) – source – See references.

Scene 1 Tuning

The goal of this effort to balance the Quadcopter by tuning the mass

default Quadcopter.Mass = 0.1

default minMotorThrust = 0.1
default maxMotorThrust = 4.5

Keeping max thrust the same the Mass needs to change to ~ 4.5 to balance the Quadcopter

m = Fthrust/g = 4.5/9.81 ~ .46

Results

Simulation #1807 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds

Simulation #1808 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds

Body rate and roll/pitch control (scenario 2)

** Parts of the section below are borrowed from readme.md **

Objective -- Quadcopter in scenario 2 is small intertial rotation speed along its roll axis and controller should stabilize the vehicle. The controller stabilizes rotational motion to bring back to level attitude.

Steps taken:

1. Implement body rate control
2. Implement the code in the function GenerateMotorCommands()
3. implement the code in the function BodyRateControl()
4. Tune kpPQR in QuadControlParams.txt to get the vehicle to stop spinning quickly but not overshoot

(Per project notes, once tuned further changes to Kpqr will be avoided)

If successful, you should see the rotation of the vehicle about roll (omega.x) get controlled to 0 while other rates remain zero. Note that the vehicle will keep flying off quite quickly, since the angle is not yet being controlled back to 0. Also note that some overshoot will happen due to motor dynamics!.

If you come back to this step after the next step, you can try tuning just the body rate omega (without the outside angle controller) by setting QuadControlParams.kpBank = 0.

5. Implement roll / pitch control We won't be worrying about yaw just yet.
6. Implement the code in the function RollPitchControl()
7. Tune kpBank in QuadControlParams.txt to minimize settling time but avoid too much overshoot

If successful you should now see the quad level itself (as shown below), though it'll still be flying away slowly since we're not controlling velocity/position! You should also see the vehicle angle (Roll) get controlled to 0.

Results:

Simulation #27 (../config/2_AttitudeControl.txt)
PASS: ABS(Quad.Roll) was less than 0.025000 for at least 0.750000 seconds
PASS: ABS(Quad.Omega.X) was less than 2.500000 for at least 0.750000 seconds

Following parameters are tuned to achieve required roll and Omega for this controller.

Angle control gains
#kpBank = 5 -- original
kpBank = 20.0

Angle rate gains
#kpPQR = 23, 23, 5 -- original
kpPQR = 95.0, 95.0, 6.0

The general guidelines followed as actual values can be 3-4 times the initial values.

Position/velocity and yaw angle control (scenario 3)

** Parts of the section below are borrowed from readme.md **

Objective -- Implement Altitude and Yaw control for the Quadcopter using Scene 3. This scenario creates 2 identical Quads, one offset from target point w/yaw = 0 and other offset from target point w/yaw = 45 degrees.

Steps taken:

1. Implement the code in the function LateralPositionControl()
2. Implement the code in the function AltitudeControl()
3. Tune parameters kpPosZ and kpPosZ
4. Tune parameters kpVelXY and kpVelZ

If successful, the quads should be going to their destination points and tracking error should be going down. However, one quad remains rotated in yaw.

Next is YawControl:

1. Implement the code in the function YawControl()
2. Tune parameters kpYaw and the 3rd (z) component of kpPQR
3. Tune position control for settling time. Yaw control requires a lot of control authority from a quadcopter and can really affect other degrees of freedom. Quadcopters with tilted motors have better yaw authority.

Below are the tuned parameters:

Position control gains
#kpPosXY = 1

#kpPosZ = 1
#KiPosZ = 20
kpPosXY = 30.0
kpPosZ = 20.0
kiPosZ = 40.0

Results:

PASS: ABS(Quad1.Pos.X) was less than 0.100000 for at least 1.250000 seconds
PASS: ABS(Quad2.Pos.X) was less than 0.100000 for at least 1.250000 seconds
PASS: ABS(Quad2.Yaw) was less than 0.100000 for at least 1.000000 seconds

Non-idealities and robustness (scenario 4)

Objective -- the goal is the make drones more robust by using Integral control

Scenario 4. This is a configuration with 3 quads that are all are trying to move one meter forward. However, this time, these quads are all a bit different:

1. The green quad has its center of mass shifted back
2. The orange vehicle is an ideal quad
3. The red vehicle is heavier than usual

Instructions suggest:

Run your controller & parameter set from Step 3. Do all the quads seem to be moving OK? If not, try to tweak the controller parameters to work for all 3 (tip: relax the controller).

Edit AltitudeControl() to add basic integral control to help with the different-mass vehicle.

Tune the integral control, and other control parameters until all the quads successfully move properly. Your drones' motion should look like this:

Results:

PASS: ABS(Quad2.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
PASS: ABS(Quad3.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds

Source location -- https://github.com/boldbinarywisdom/Aerial-controls

References:

(1) 3D control Architecture – FCND Term Lesson 14 Full 3D Control
(2) Motion primitives – Angela et. Al - http://flyingmachinearena.org/wp-content/publications/2011/schoellig_feasibility_of_motion_primitives.pdf
(3) Collective Thrust and angular acceleration – overleaf https://www.overleaf.com/read/thzntmhcqkkp#/63267348/
(4) Cascaded P controller - https://www.overleaf.com/read/bgrkghpggnyc#/61023787/
(5) Multi-copter PID tuning - https://docs.px4.io/en/advanced_config/pid_tuning_guide_multicopter.html
(6) Representing Attitude - https://www.astro.rug.nl/software/kapteyn/_downloads/attitude.pdf