

Aerial Robotics Week 4

**WEEK 04**

Welcome to Week 4! We have developed planar and three-dimensional dynamic models of the quadrotor. This week, you will learn about autonomous navigation and how to develop linear controllers for these three-dimensional dynamic models. With this knowledge, you will be required to implement PID controller for quadrotor in 3D.

**So, to recap what we have learnt so far:**

 In week 3, we discussed about quadrotor kinematics: Transformation, Rotation, Euler angle, Axis/angle representations for rotation also had a look at angular velocity vector.

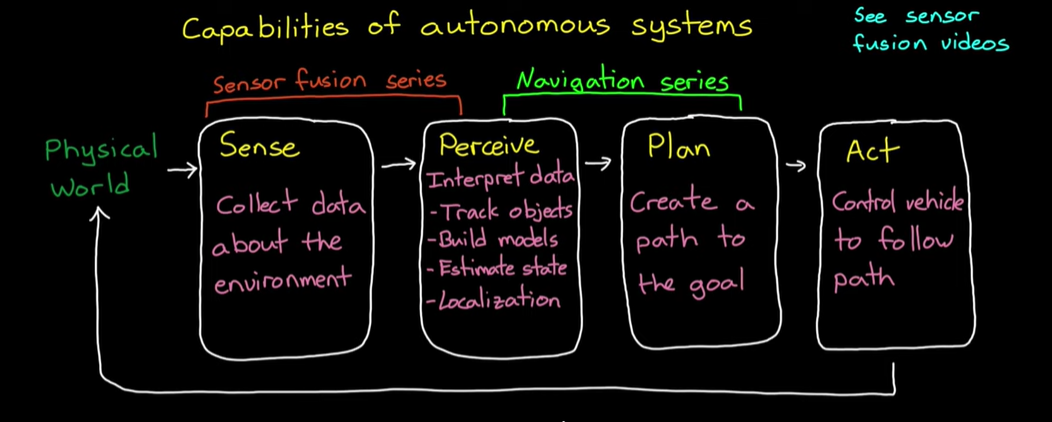
 Estimation of the drone's position and velocity of movement by six parameters (6 degrees of freedom).

 Then we understand how to write the equations of motion for a quadrotor.

# Autonomous navigation

Now we are going to explore the world of self-guided systems as we delve into developing controllers for models, enabling them to navigate and interact autonomously.

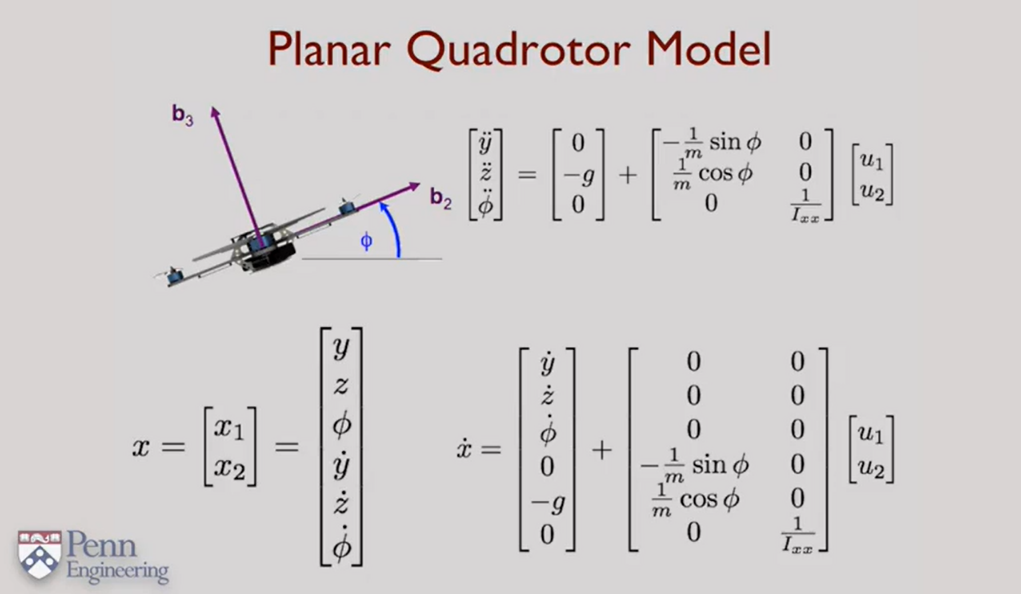
[https://www.youtube.com/watch?v=Fw8JQ5Q-ZwU&list=PLn8PRpmsu08rLRGrnF-S6T](https://www.youtube.com/watch?v=Fw8JQ5Q-ZwU&list=PLn8PRpmsu08rLRGrnF-S6TyGrmcA2X7kg&index=1) [yGrmcA2X7kg&index=1](https://www.youtube.com/watch?v=Fw8JQ5Q-ZwU&list=PLn8PRpmsu08rLRGrnF-S6TyGrmcA2X7kg&index=1)



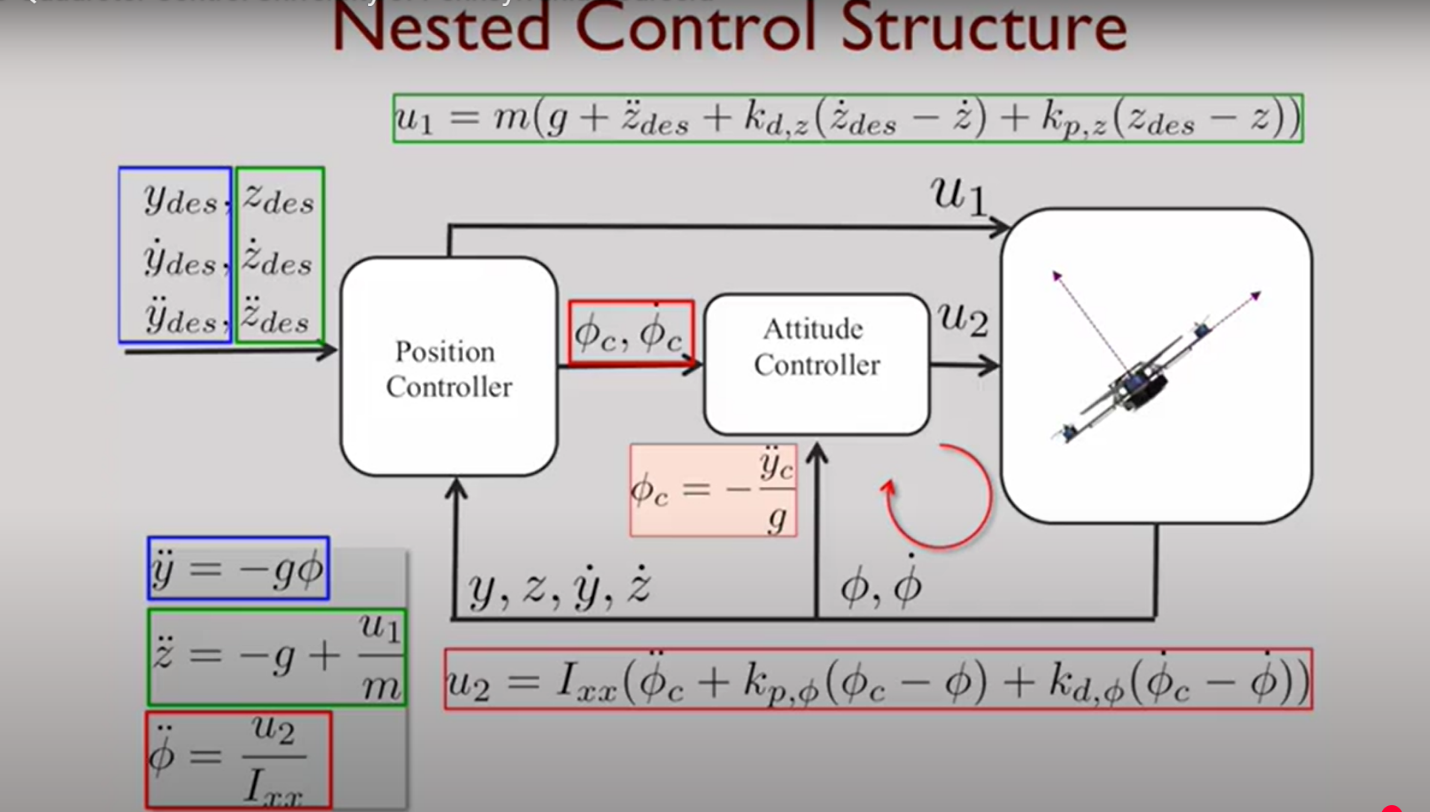


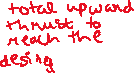
[https://www.youtube.com/watch?v=NrzmH\_yerBU&list=PLn8PRpmsu08rLRGrnF-S6T](https://www.youtube.com/watch?v=NrzmH_yerBU&list=PLn8PRpmsu08rLRGrnF-S6TyGrmcA2X7kg&index=2) [yGrmcA2X7kg&index=2](https://www.youtube.com/watch?v=NrzmH_yerBU&list=PLn8PRpmsu08rLRGrnF-S6TyGrmcA2X7kg&index=2)

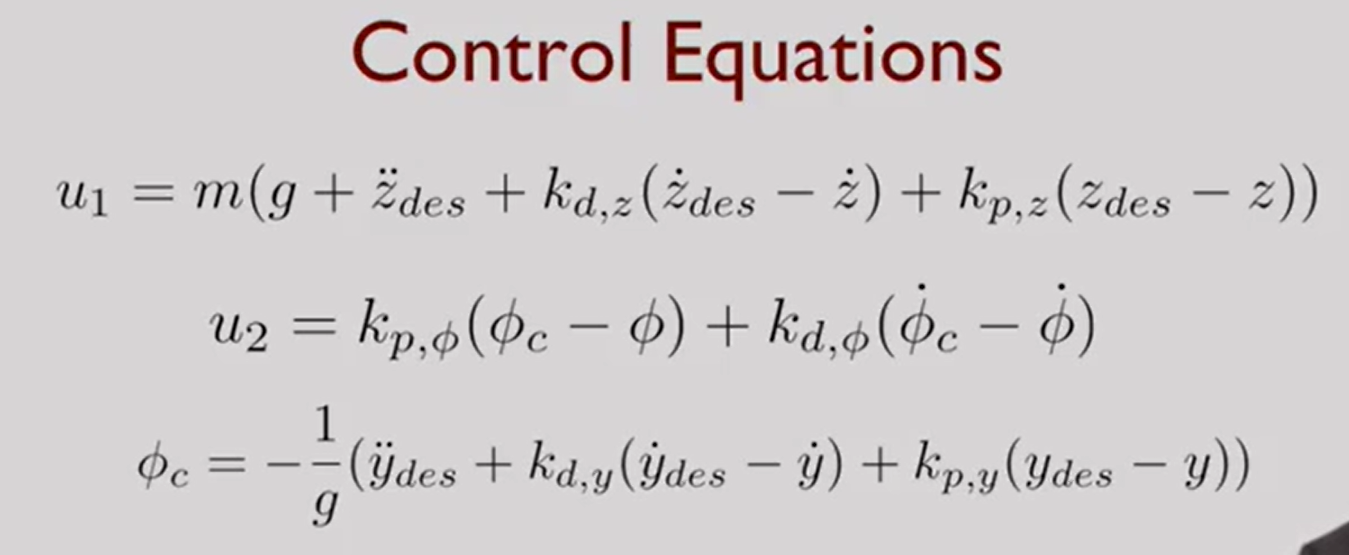
# 2D Quadrotor Controller

****

<https://youtu.be/uPsenrdsu28>

****



****



We'll again add a quick example of how a 2D controller looks with just P added to it, also We have chosen a path such that y\_des and z\_des don't come into picture and again no thresholds for thrust(So if they do you'll need to code that), have a look at it, tinker around, you'll obviously never attain the desired goal with the present controller, but it is encouraged to try and convert it into a PD controller (also the drill's same download the folder, open controller.m, have a look at the equations then run ‘*runsim*’ have a look at the results in it etc.

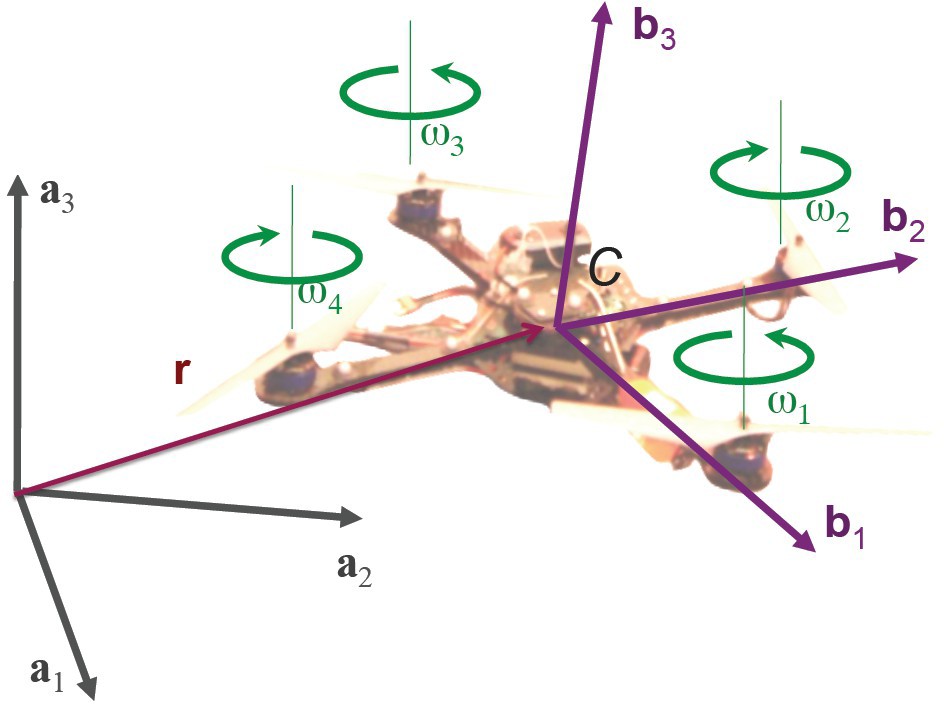
# 3D Quadrotor Control

[2D Controller.zip](https://www.notion.so/signed/https%3A%2F%2Fprod-files-secure.s3.us-west-2.amazonaws.com%2F7c4bdc63-bf66-4fe6-bde2-e046a8292fe8%2F807e006c-d07c-4f0f-846a-de17043daa16%2F2D_Controller.zip?table=block&id=1886ff4a-1782-4e92-86dc-7d928965db9c&spaceId=7c4bdc63-bf66-4fe6-bde2-e046a8292fe8&userId=00cf3b5e-cc19-4e39-a0ac-f0ca0381cf21&cache=v2)

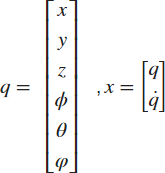
Having looked at the 2-dimensional quadrotor, we now turn our attention to the dynamics of the 3-dimensional model. For a better understanding follow the below video with the content

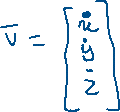
<https://youtu.be/wOoLiwAYN0M>

Recall that we have a body-fixed frame attached to the quadrotor and we have an inertial frame:



The state of the quadrotor consists of position and orientation. The position is the position-vector of the center-of-mass, and the roll, pitch and yaw angles tell us the orientation. This state is composed of a six-dimensional vector and its rate of change (in other words, the velocity).





The roll, pitch and yaw angles follow the usual convention: first the yaw above the zaxis,and then the roll & pitch as we've seen before. The angular velocity components in the body frame, p q and r, are related to the derivatives of the roll, pitch and yaw angles through this coefficient matrix:

*p* cos *θ*

*q* = 0

*r* sin *θ*

1. − cos *ϕ* sin *θ ϕ*˙
2. sin *ϕ θ*˙

0 cos *ϕ* cos *θ φ*˙

This set of equations tells us everything we need to know about the kinematics of the vehicle. What we’d like to do is to track arbitrarily specified trajectories in three-dimensions.

We assume we’re given a trajectory. This trajectory consists of a position vector that varies as a function of time, and a yaw angle, which also varies as a function of time:

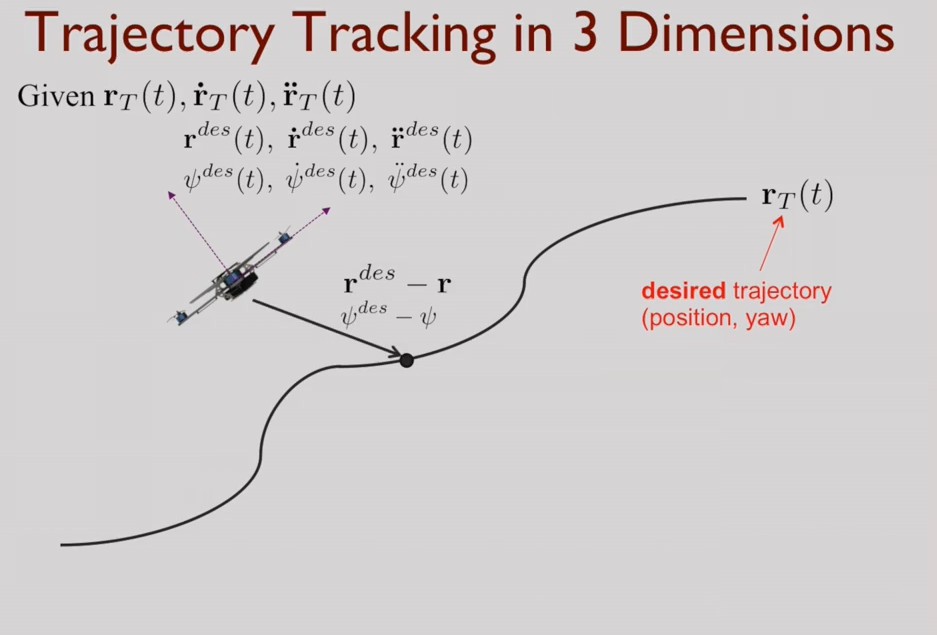
*x* (*t*)

*rT* (*t*) =

*y* (*t*)

*z* (*t*)

*φ* (*t*)



We want this four-dimensional vector to be differentiable, and we want to be able to obtain not only its derivative, but also its second derivative.

As before, we're interested in the difference between the desired position and the actual position. But now we're also interested in the difference between the desired yaw angle and the actual yaw angle. That gives us the error vector and its derivative:

*ep* = *rT* (*t*) − *r*



*ev* = *r*˙*T* (*t*) − *r*˙



And again, we want the error vector to go exponentially to zero.

( *r* ¨( *t*) − *r*¨ ) + *K e* + *K e* = 0

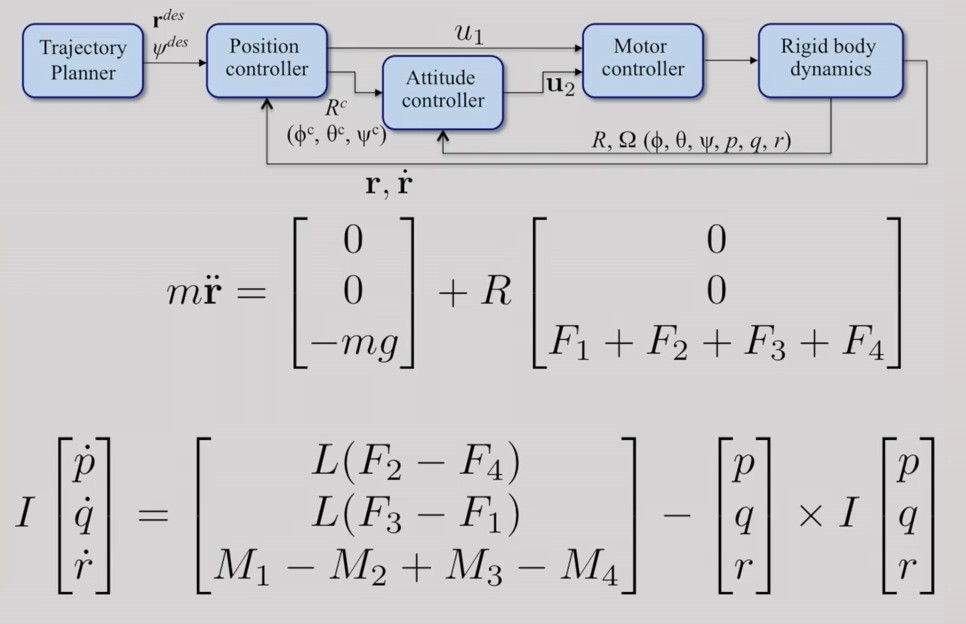


*T*

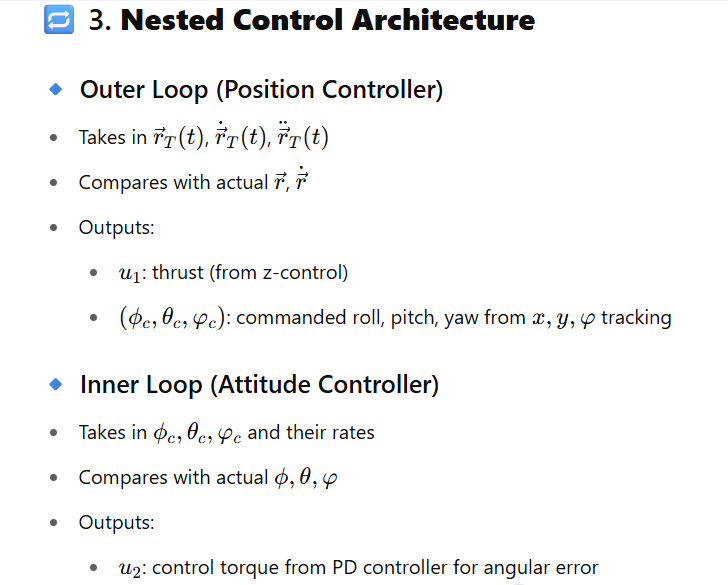
*c*

*d v*

*p p*

This lets us calculate the commanded acceleration, *r*¨*c* , whether it's the 2nd-derivative of the position vector, or the 2nd-derivative of the yaw angle. Let's take another look at the equations of motion, and the nested-feedback-loop we described for the two-dimensional quadrotor model:





As in the planar case, we have nested feedback loops. The inner loop corresponds to attitude control, and the outer loop corresponds to position control. In the inner loop, we specify the orientation either using a rotation matrix or a series of roll, pitch, and yaw angles. We’ll feedback the actual attitude and angular velocity, or the roll, pitch and yaw angles and the angular rates.

From that, we calculate the input *u*2. *u*2 is a function of the thrusts and moments that we get

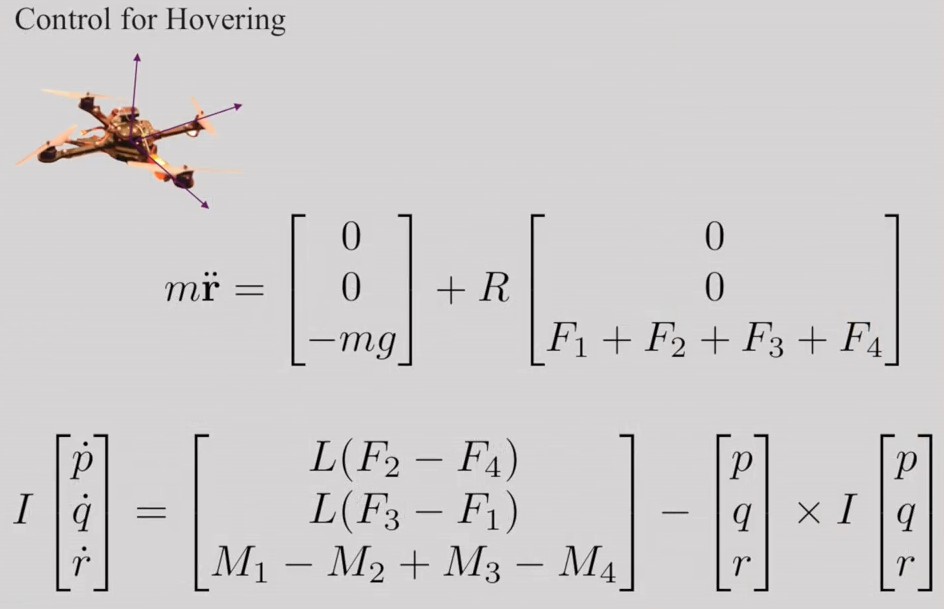
from the motors.

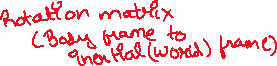
In the equations-of-motion at the bottom, we have to calculate the value of

*u*2 based on the desired attitude.

We now turn our attention to the outer-loop, which is a position feedback loop. In this loop, we take the specified position vector & specified yaw angle from the Trajectory Planner. We compare that with the actual position and velocity, and from that we calculate

*u*1, *u*1is essentially the sum of all the thrust forces.

Rather than looking at the general trajectory-following problem, let's initially focus on a very specific case, the case of hovering:



So, in hovering, the robot’s position and orientation are fixed. In other words, all velocities are zero. Further, the roll and pitch angles are also equal to zero. We want to consider small perturbations around the hover position. Accordingly, we'll linearize the dynamics around this current configuration.



In order for the hover position to be one of equilibrium, the input u2 has to be 0, the angular velocity components p, q, and r are equal to 0, and their derivatives are also equal to 0. Likewise, the sum of the thrusts has to compensate for the weight of the robot, but the yaw angle can be non-zero as long as it's fixed.

Linearizing around this point, means that we are assuming *u*1

is almost equal to mg. We assume the roll and pitch angles are close to zero, and we assume that the yaw angle is fixed, or close to fixed at a given value *φo*.

(*u*1 ∼ mg , *θ* ∼ 0, *ϕ* ∼ 0 , *φ* ∼ *φo*)

When we linearize the equations, we end up with these simplified equations:

*r*¨1 = *x*¨ = *g* (*θ* cos *ϕ* + *ϕ* sin *φ*)

*r*¨2 = *y*¨ = *g* (*θ* sin *φ* − *ϕ* cos *φ*)

We can now design the inner feedback loop, by simply specifying u2 using a proportional plus derivative controller:

*Kp*,*ϕ* (*ϕc* − *ϕ*) + *Kd*,*ϕ* (*pc* − *p*)

*u*2 =

*Kp*,*θ* (*θc* − *θ*) + *Kd*,*θ* (*qc*

— *q*)

*Kp*,*φ* (*φc* − *φ*) + *Kd*,*φ* (*rc* − *r*)

Here we assume that we have the commanded roll, pitch, and yaw angles and their derivatives. And all we require for feedback are the actual roll, pitch, and yaw angles, and their derivatives. Now, let's consider the outer feedback loop. If we linearize the equations-of-motion, the expressions for the first two components of acceleration can be written in the form shown below:

*r*¨1 = *x*¨ = *g* (*θ* cos *ϕ* + *ϕ* sin *φ*)

*r*¨2 = *y*¨ = *g* (*θ* sin *φ* − *ϕ* cos *φ*)

We can now turn our attention to the error equation describing the error in position, using these linearised equations of motion. Remember, we want this error to satisfy the second-order differential equation:

(*ri*,¨des − *r*¨*i*,*c* ) + *Kd* ,*i* (*ri* ,˙des − *r*˙*i*) + *Kp*,*i* (*ri*,des − *ri*) = 0

This equation contains terms to do with the desired trajectory, *ri*,des , and terms to do with the actual trajectory being followed,

*ri*. We will use this equation to calculate the commanded acceleration, *r*¨1,*c*. This commanded acceleration will, in turn, tell us what thrust we need to apply with the rotors.

*u*1 = *m* (*g* + *r*¨3,*c* )

Finally, we need to determine the commanded roll, pitch, and yaw using the linearized equations. If we know that the commanded acceleration needs to be in the X and Y direction, we can

calculate the roll and pitch angles and their derivatives.

*ϕ* = 1 (*r*¨

sin *φ* − *r*¨

cos *φ* )

*c g* 1,*c*

1

des

2,*c*

des

*θc* =

 (*r*¨1,*c* cos *φ*des + *r*2,*c* sin *φ*des )

*g*

We can determine the commanded yaw angles in a similar manner.

*φc* = *φ*des

So, you have developed planar and three-dimensional dynamic models of the quadrotor and linear controllers for these models.

So, this week we have understood 3-D Quadcopter Controller, it's equation of motion and plan any trajectory in 3D environment.

