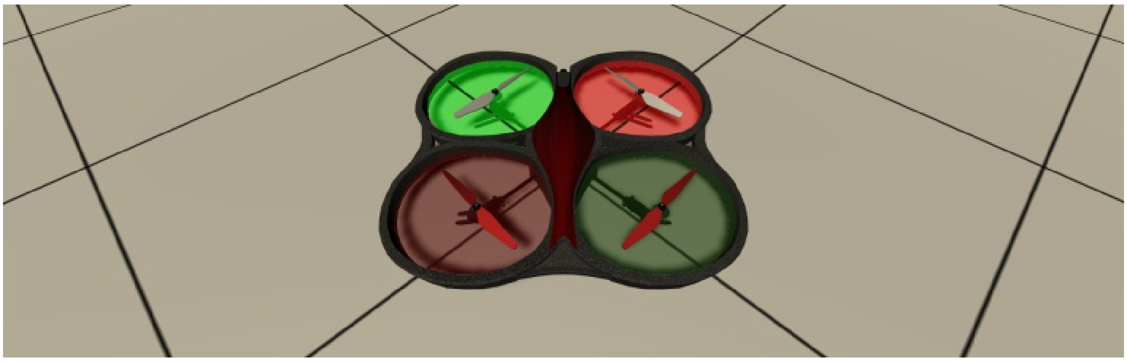
Project: Controls Lab

1. Project Summary

The goal of the project is to learn the fundamentals and implement a PID controller on a quadcopter.

This PID controller will be used in a ROS node for controlling the hover, attitude, and position of a quadcopter inside a Unity simulation environment.

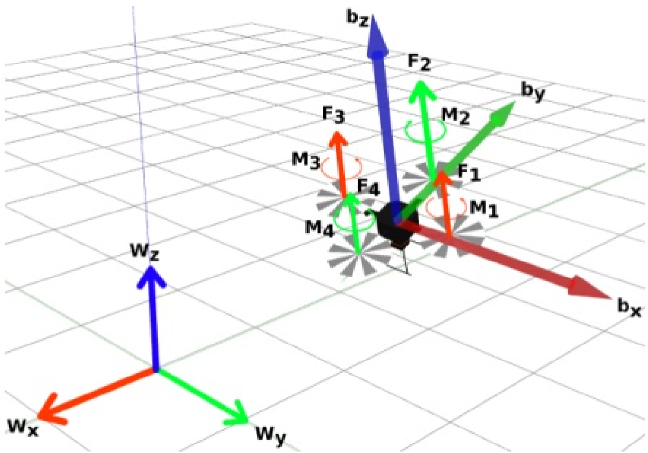


Objectives:

- Implement a PID Controller
- Implement a Hover Controller
- Implement a Attitude Controller
- Implement a Position Controller

2. Quadcopter Kinematics

To explore the kinematics of a quadcopter, there are two reference frames considered - **world frame** and **body frame**. The **world frame** is a fixed frame unlike the **body frame** whose origin is coincident with the *center of mass* of the quadcopter.



- **W_x**, **W_y**, **W_z** are the orthonormal vectors of the world frame. **b_x**, **b_y**, **b_z** are the orthonormal vectors of the body frame (or quadcopter).
- Thrusts generated by each rotor is labeled by **F_i**. The torque generated by each propeller is labeled by **M_i**.

A point **P** seen from the **body frame** can be represented in the **world frame** using a composite rotation matrix intrinsically formed i.e. ${}^{W}R_B = R_z(yaw) * R_y(pitch) * R_x(roll)$

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_W = {}^{W}_{B}R \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_B$$

2.1 Quadcopter Motion

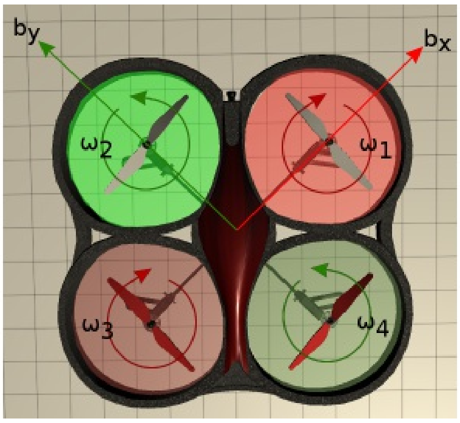
A rigid body in 3D space has six degrees of freedom. However, a quadrotor only has four motors to act as control inputs, thus a quadrotor is an example of an underactuated system. What this means is that a quadrotor cannot independently control each translational and rotational degree of freedom.

- **Pitch**: To translate the quadcopter forward/backward, we need to pitch the quadcopter along the y-axis of the body frame.
  Rotation by a positive pitch angle along **b_y**, translates it forward.
  Rotation by a negative pitch angle along **b_y**, translates it backward.

- **Roll**: To translate the quadcopter right/left, we need to roll the quadcopter along the x-axis of the body frame.
  Rotation by a positive roll angle along **b_x**, translates it right.
  Rotation by a negative roll angle along **b_x**, translates it left.

- **Vertical Motion**: Each rotor produces a thrust that is proportional to the angular velocity of the rotor. If the net motor thrust along the z-axis of the body frame is greater than the force of gravity, then the quadrotor will accelerate upwards. The converse holds true as well.

- **Yaw**: Motors 1 & 3 rotate clockwise; Motors 2 & 4 rotate counter-clockwise. Since torque produced is opposite to the direction of the propeller, torque from motors 1 & 3 rotate counter-clockwise while torque from motors 2 & 4 rotate clockwise. For a pure yawing motion, i.e., rotate without changing elevation, one pair of motors would increase their angular speeds by the same amount as the opposite pair would decrease theirs.



2.2 Quadcopter States

In general, a quadrotor requires 12 generalized coordinates to completely describe its position and orientation in 3D space (3 for position, 3 for orientation, and their time derivatives).

- `x` , `y` , `z` - Position of the quadrotor's center of mass.
- `ϕ` , `θ` , `ψ` - Orientation, in Euler angles, of {B} relative to {W}.
- `x_dot` , `y_dot` , `z_dot` - Linear velocities of the quadrotor's center of mass.
- `ϕ_dot` , `θ_dot` , `ψ_dot` - Angular velocities of {B} relative to {W}.

2.3 Calculating angular velocity at Hover

Calculating the nominal motor thrust to hover in an equilibrium configuration is very straightforward - it is simply a statics problem. If the sum of the forces acting in the $\mathbf{W_z}$ direction is equal to zero, then there is no acceleration in the z-direction of the world frame.

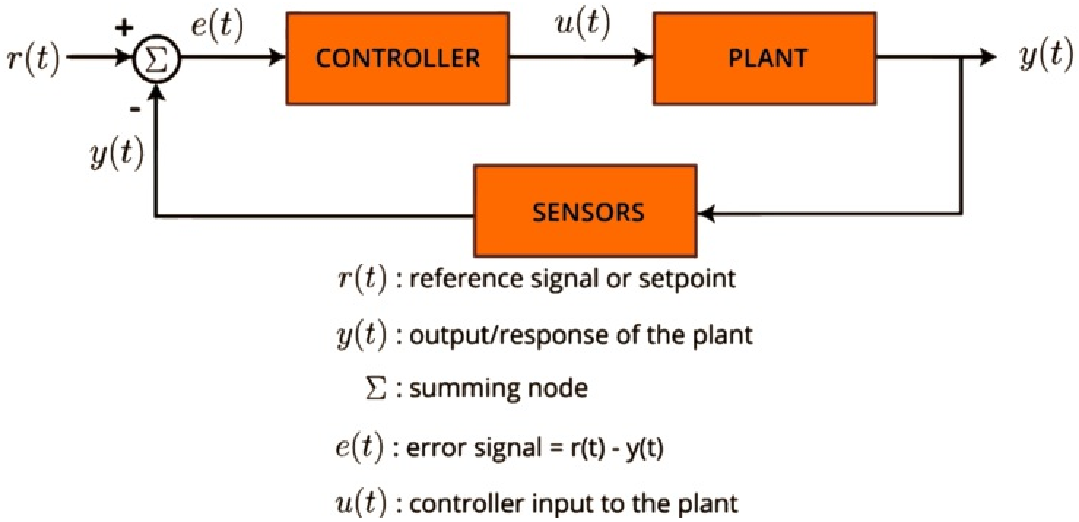
$$\begin{aligned}
\sum \mathbf{F_z} &= \mathbf{0} \\
-m\mathbf{g} + \mathbf{F_1} + \mathbf{F_2} + \mathbf{F_3} + \mathbf{F_4} &= \mathbf{0} \\
-mg\mathbf{W_z} + F_1\mathbf{b_z} + F_2\mathbf{b_z} + F_3\mathbf{b_z} + F_4\mathbf{b_z} &= \mathbf{0} \\
-mg\mathbf{W_z} + (F_1 + F_2 + F_3 + F_4)\,\mathbf{b_z} &= \mathbf{0} \\
\left(-mg\mathbf{W_z} + \sum_{i=1}^{4} F_i\mathbf{b_z}\right) \cdot \mathbf{W_z} &= \mathbf{0} \cdot \mathbf{W_z}
\end{aligned}$$

In equilibrium state, each motor thrust is the same, and the z-axis of {B} and {W} frames are parallel, hence their dot product is 1.

$$
\begin{aligned}
-mg + 4F_i &= 0 \\
-mg + 4k_F\omega_i^2 &= 0 \\
\omega_i &= \sqrt{\frac{mg}{4k_F}}
\end{aligned}
$$

3. PID Controller

In a closed-loop system, the objective is to track a desired set point by minimizing the error. The error is the difference between the desired set point and current measurement.



The Proportional-Integral-Derivative (PID) controller is the most ubiquitous controller used in the industry with over 97% systems having some form of PID within then.

PID is the **Controller** block in the figure above. It generates ***u(t)*** (control effort) based on ***e(t)*** (error) and feeds it to the **Plant** block. The plant produces the ***y(t)*** (output). The **Sensor** block senses the output and feeds it back to the **summing** node for re-calculating the error.

3.1 Parameter Definitions

- **Proportional** - The error is multiplied by a constant called proportional parameter (K_p) creating a control effort proportional to the error.

- **Integral** - The error is integrated and then multiplied by a constant called integral parameter (K_i) creating a control effort that keeps track of error's history.

- **Derivative -** The derivative of the error is computed for every time internal. The differential error is multiplied by a constant called derivative parameter (K_d) creating a control effort that can predict the error.

3.2 Parameter Effects

Changing the parameters K_p, K_i, K_d alters the behavior of the control effort, ***u(t)***, generated by the PID controller drastically. The table below summarizes the behavior of the control effort generated by the PID.

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
K_p	Decrease	Increase	Small Change	Decrease	Degrade
K_I	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor Change	Decrease	Decrease	No Effect in Theory	Improve if K_d Small

- **Rise time** - Time taken to first reach the desired value.
- **Overshoot** - The maximum offset from the desired value.
- **Settling time** - Time taken to get within 95-98% of desired value.
- **Steady-state error** - Difference between desired value and steady-state value.
- **Stability** - Ability to track the desired set point with negligible oscillations.

4. Positional Control of Quadcopter

The goal is to navigate a quadcopter stably to a desired position in 3D space. First, the quadcopter will need to achieve a stable hovering altitude with the help of the **Hover Controller**. After getting a stable hover, next the attitude of the quadcopter will be made stable using the **Attitude Controller**. Finally, within a cascaded fashion, the **Position Controller** will be implemented.

4.1 Hover Controller

The hover controller is responsible for making the quadcopter hover stably at a desired altitude. I used a PD controller with the following parameters:

- **Hover:**
  - K_p - 10.0
  - K_i - 0.0

- K_d - 10.0
- **Gravity** - On

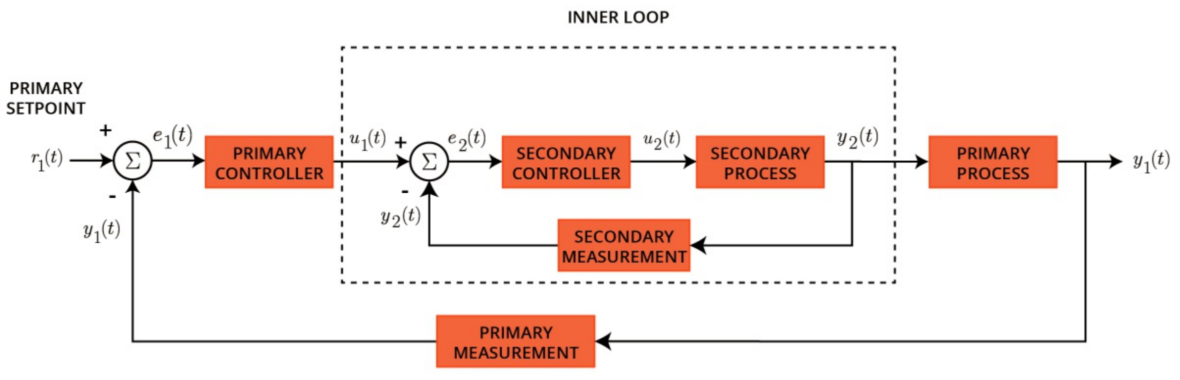
4.2 Attitude Controller

The attitude controller is responsible for maintaining the pose of the quadcopter at the desired orientation. Here too, I used a PD controller with the following parameters:

- **Roll:**
  - K_p - 2.0
  - K_i - 0.0
  - K_d - 2.0
- **Pitch:**
  - K_p - 2.0
  - K_i - 0.0
  - K_d - 2.0
- **Yaw:**
  - K_p - 5.0
  - K_i - 0.0
  - K_d - 5.0
- **Gravity** - On

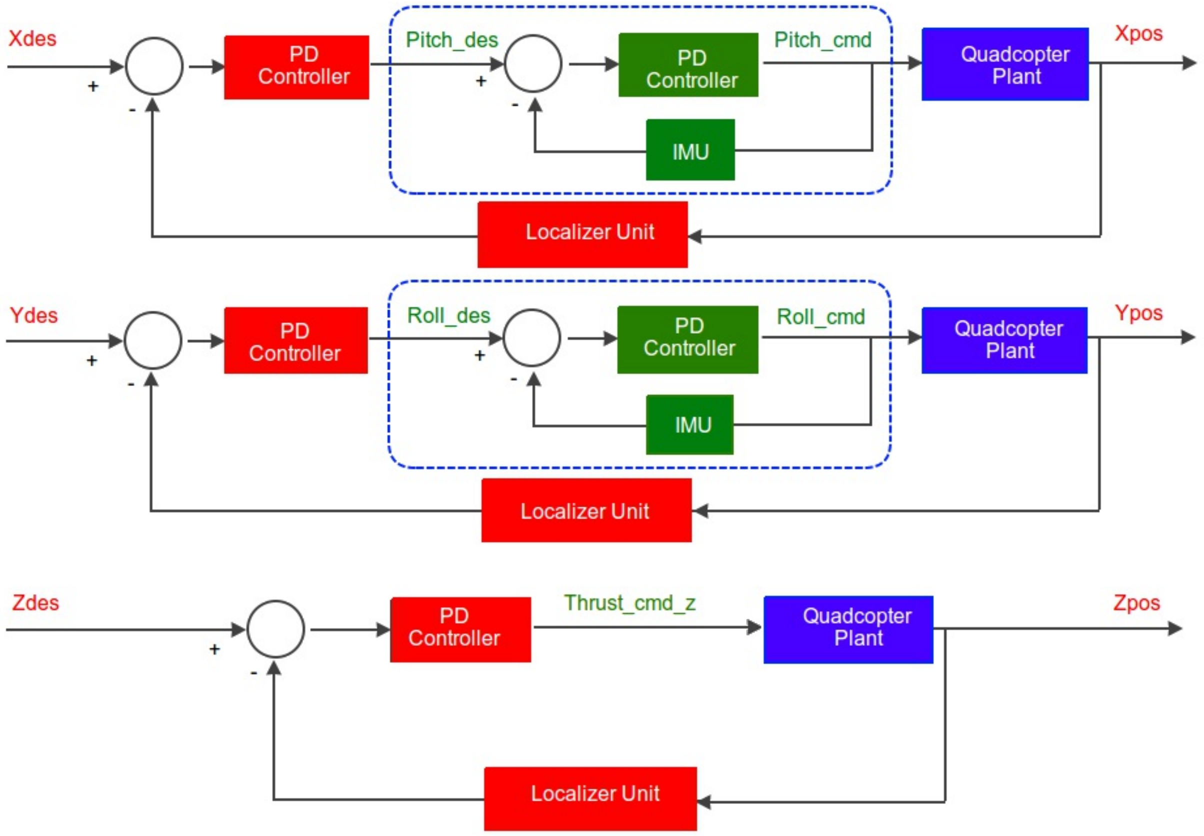
4.3 Cascaded Control

The cascade control is simply a closed-loop system within another closed-loop system. It can be thought of a loop within a loop. The general principle is for the inner loop to operate at a frequency at least 10x times faster than the outer loop.



For our position controller, the inner loop is the **Attitude Controller** while the **Position Controller** is the outer loop.

4.4 Position Controller



The position controller is responsible for translating the quadcopter to a desired position. I used a PD controller for $\mathbf{X_{des}}$, $\mathbf{Y_{des}}$, and $\mathbf{Z_{des}}$. The controller for $\mathbf{Z_{des}}$ is just the hover controller implemented above. I used a PD controller with the following parameters:

- **X_{des}:**
  - K_p - 0.75
  - K_i - 0.00
  - K_d - 1.50
- **Y_{des}:**
  - K_p - 0.75
  - K_i - 0.00
  - K_d - 1.50
- **Z_{des} (Hover):**
  - K_p - 10.0
  - K_i - 0.0
  - K_d - 10.0
- **Gravity** - On