Pre Lab 2
Giovanni Dal Lago

1)

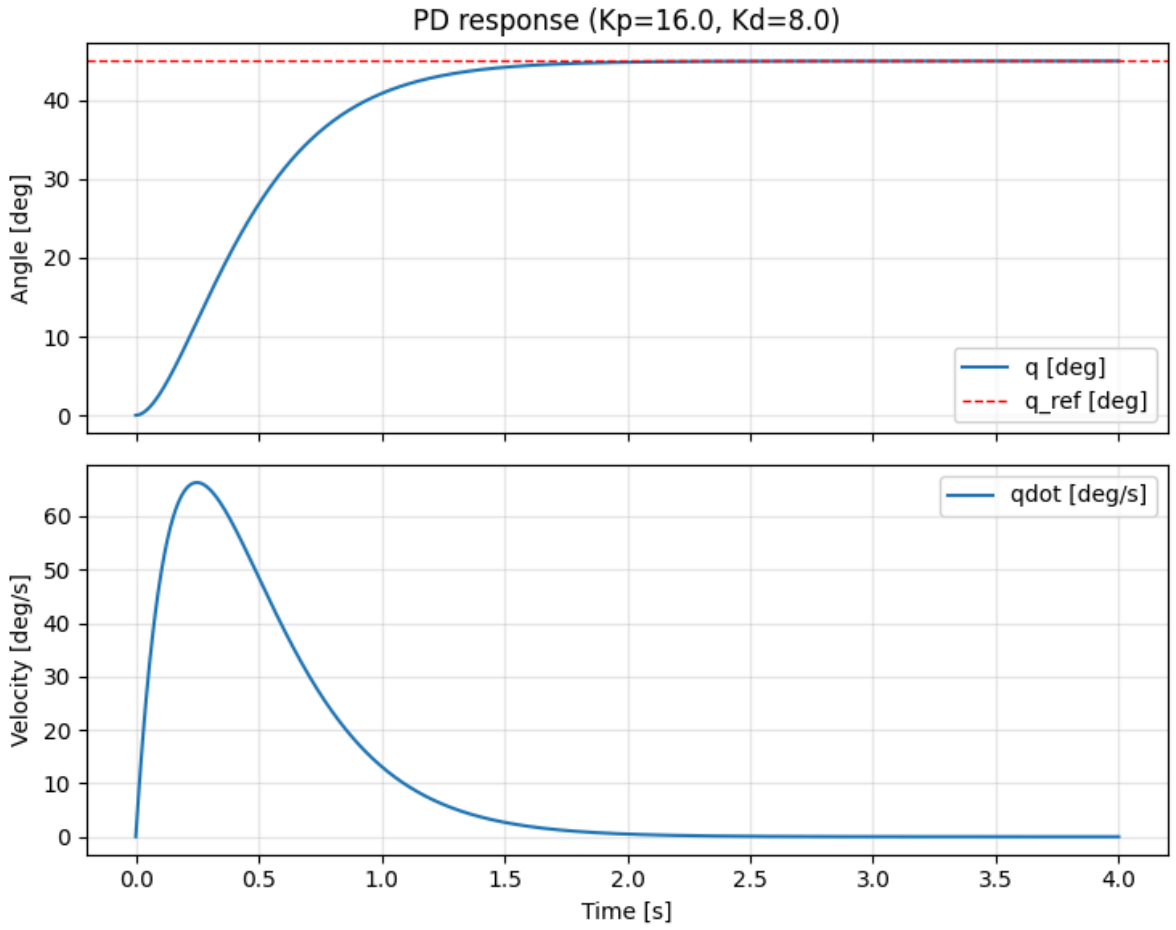
- The video starts with explaining what an object is

  then it explains:

- How to create a class
- What a class is
- What methods are and how to create them
- The `__init__` method
- Attributes and how to modify them
- How classes interact with each other
- Inheritance
- Class attributes
- Class methods
- Static methods

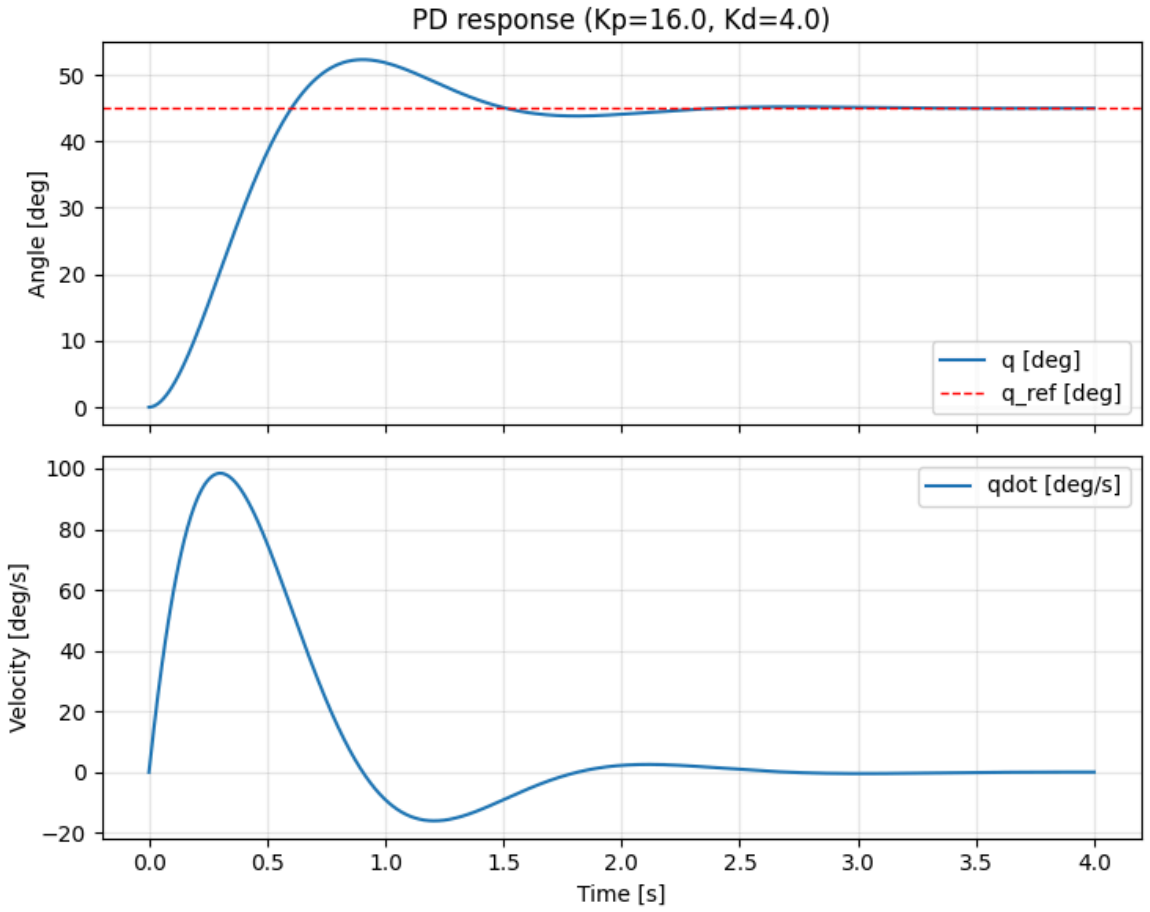
2)

```
for k in range(N - 1):
e = qref - q[k]
edot = qref_dot - qdot[k]
u = Kp * e - Kd * qdot[k]

qdot_next = qdot[k] + dt * u
q_next = q[k] + dt * qdot_next

qdot[k + 1] = qdot_next
q[k + 1] = q_next
```



The figure shows a **damped** system.

- The blue trace of q(t)q(t) rises toward the reference without exceeding it, so there is **no overshoot**.
- The rise is fast at the beginning, then slows down and settles, which is typical of an **overdamped or near critically damped** system where oscillations are suppressed by the derivative term.
- The velocity $\dot{q}(t)$ (bottom plot) has an initial peak and then quickly decays to zero, indicating that the motion's energy is dissipated



This response with Kp=16,Kd=4 is **underdamped**.

- The angle q(t)q(t) overshoots the red line, peaking above 50 degrees before settling back toward 45 degrees.
- The velocity plot shows a strong positive peak followed by a negative dip, confirming oscillatory behavior.
- Compared to the previous case (Kp=16,Kd=8), the weaker derivative gain reduces damping, so the system reacts faster but with overshoot and oscillations before stabilizing.

the full code:

```
import numpy as np

import matplotlib.pyplot as plt



def simulate_pd(q0_deg, qref_deg, Kp, Kd, T=4.0, dt=0.002, vel_limit=None):

N = int(np.ceil(T / dt)) + 1

t = np.linspace(0.0, T, N)

q = np.zeros(N)

qdot = np.zeros(N)



q[0] = float(q0_deg)

qdot[0] = 0.0

qref = float(qref_deg)

qref_dot = 0.0



for k in range(N - 1):

e = qref - q[k]

edot = qref_dot - qdot[k] # non usato

u = Kp * e - Kd * qdot[k]



qdot_next = qdot[k] + dt * u

q_next = q[k] + dt * qdot_next
```

```
qdot[k + 1] = qdot_next

q[k + 1] = q_next



return t, q, qdot



def plot_response(t, q, qdot, qref_deg, title, save_path=None):

fig, axes = plt.subplots(2, 1, figsize=(7.5, 6), sharex=True)

axes[0].plot(t, q, label="q [deg]")

axes[0].axhline(y=qref_deg, linestyle="--", color="red", linewidth=1,
label="q_ref [deg]")

axes[0].set_ylabel("Angle [deg]")

axes[0].set_title(title)

axes[0].grid(True, alpha=0.3)

axes[0].legend()



axes[1].plot(t, qdot, label="qdot [deg/s]")

axes[1].set_xlabel("Time [s]")

axes[1].set_ylabel("Velocity [deg/s]")

axes[1].grid(True, alpha=0.3)

axes[1].legend()



fig.tight_layout()

if save_path:

fig.savefig(save_path, dpi=160)

print(f"Saved {save_path}")

plt.show()
```

```
def main():

Kp_B, Kd_B = 16.0, 8.0

t, q, qdot = simulate_pd(q0_deg=0.0, qref_deg=45.0, Kp=Kp_B, Kd=Kd_B, T=4.0,
dt=0.002)

plot_response(t, q, qdot, qref_deg=45.0,

title=f"PD response (Kp={Kp_B}, Kd={Kd_B})",

save_path="prelab1_pd_B.png")



if __name__ == "__main__":

main()
```