ENGR 4421: Robotics II

Revisit PID Control

02/22/2022

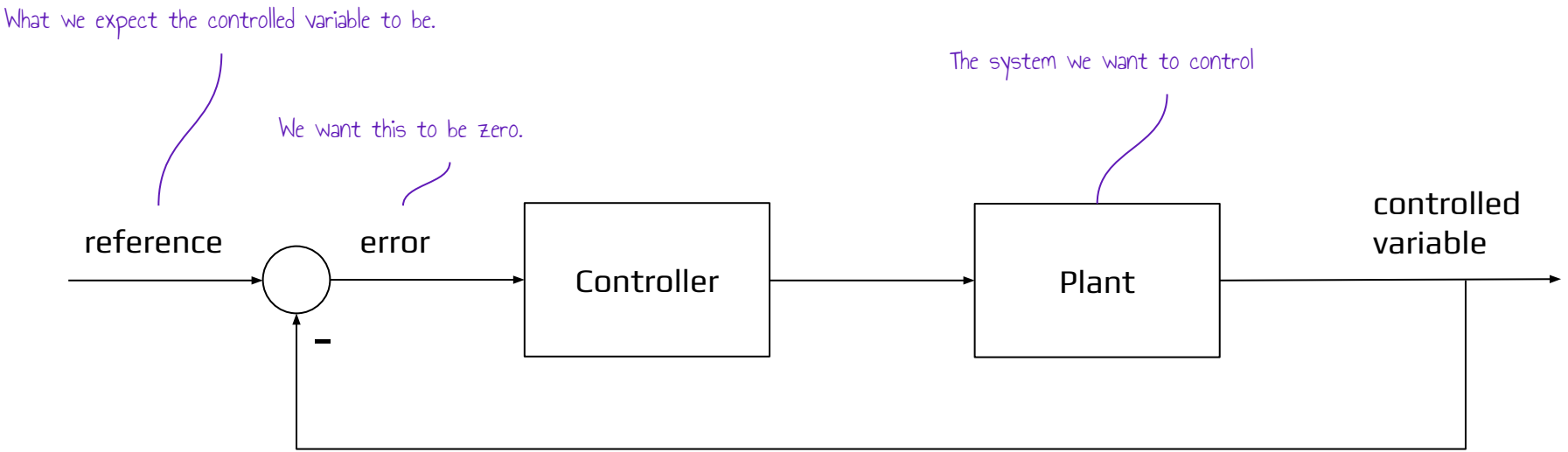
Outline

- Feedback Control Review
- Bang-Bang Control
- PID Control
  - Proportional Gain
  - Integral Gain
  - Derivative Gain

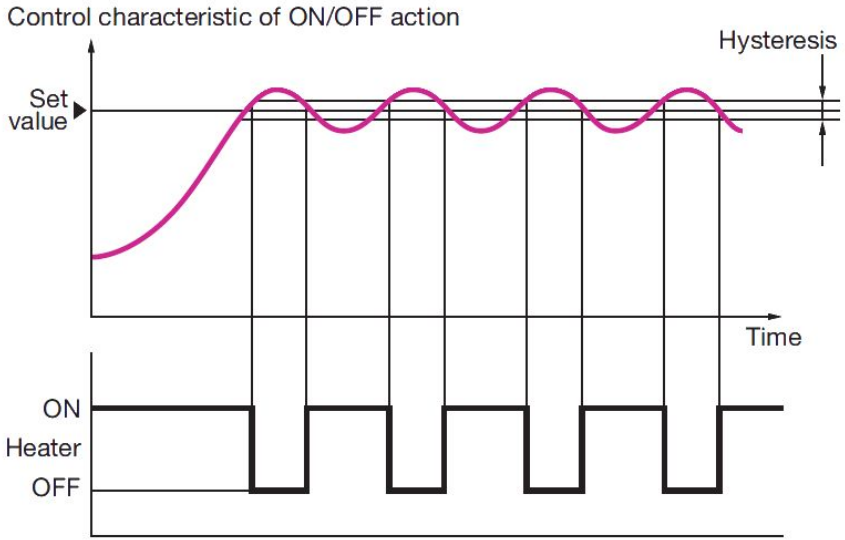
Some Control Scenarios

- A furnace trying to warm a room.
- A robotic arm trying to reach to a certain pose.
- A valve trying to control water flow..
- A cart trying to arrive to a destination point.
- A quadcopter trying to balance itself.
- A motor trying to reach desired speed.

Feedback Control



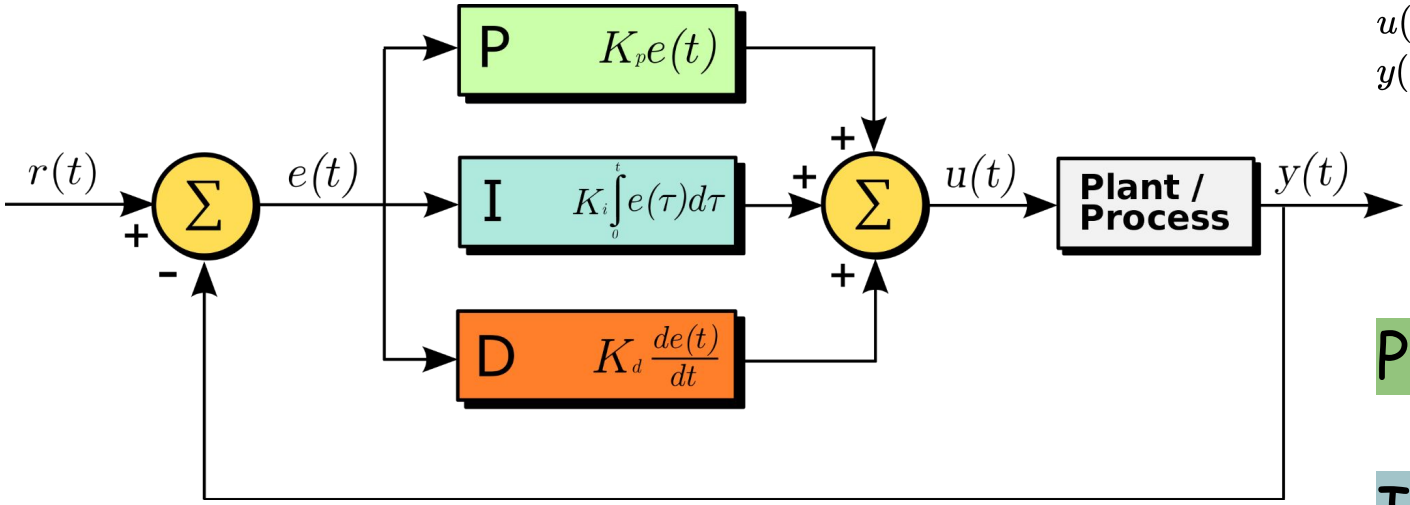
Bang-Bang Control



- On/Off control is simple.
- Set a reasonable hysteresis gap.
- Sudden high current/heating/expansion leads to wear-and-tear effect.

PID Controller

$r(t)$: *reference*
$e(t)$: *error*
$u(t)$: *control* signal
$y(t)$: *controlled variable*



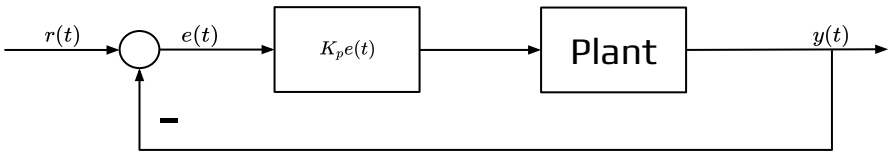
$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

P: present

I: past

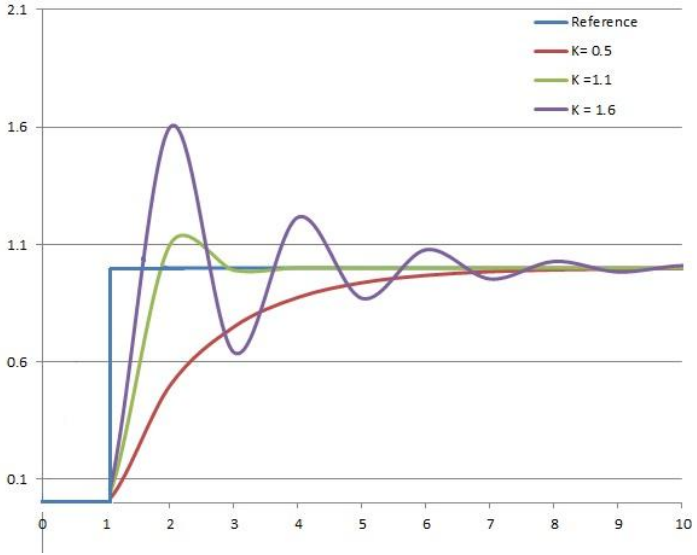
D: future

Proportional Gain



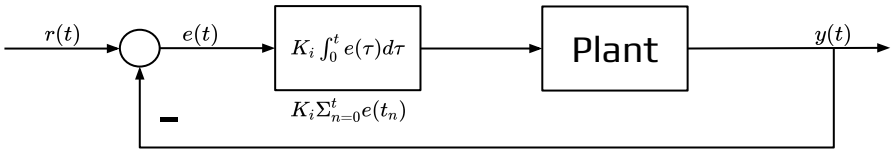

```
duty_cycle = 0.0
Kp = 0.5
reference_speed = math.pi  # rad/s
while True:
    sensed_speed = compute_speed_from_encoder(encoder_counter)
    error = reference_speed - sensed_speed
    proportional_increment = Kp*error
    duty_cycle = duty_cycle + proportional_increament
    if duty_cycle >= 1:
        duty_cycle = 1
    elif duty_cycle <= 0:
        duty_cycle = 0
```

Proportional Gain



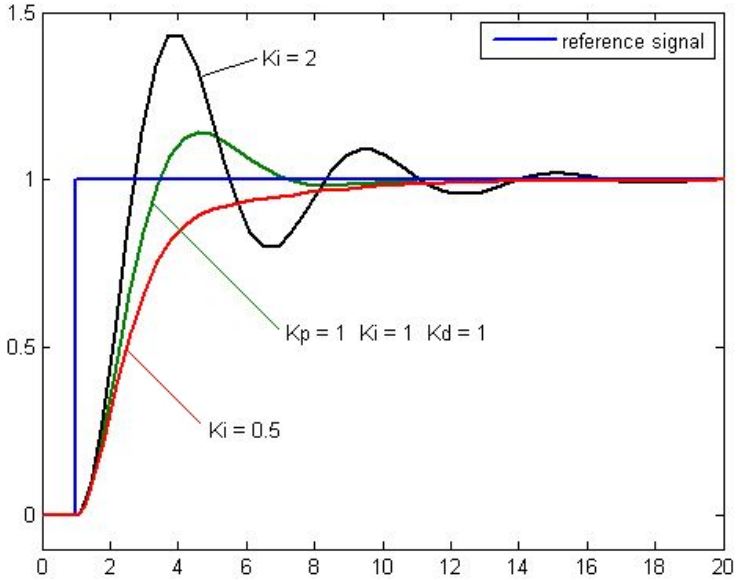
- Proportional gain is simple and can work out of the box in practice.
- Large Kp may cause oscillation.
- Output will have an offset from the set point if a non-zero input is required at that point.

Integral Gain



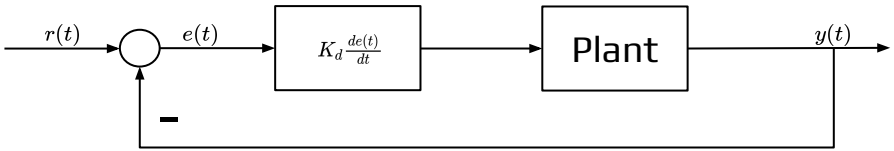

```
duty_cycle = 0.0
Kp = 0.5
Ki = 0.1
reference_speed = math.pi  # rad/s
error_buffer = []
while True:
    sensed_speed = compute_speed_from_encoder(encoder_counter)
    error = reference_speed - sensed_speed
    error_buffer.append(error)
    proportional_increment = Kp*error
    integral_increment = Ki*sum(error_buffer)
    duty_cycle = duty_cycle + proportional_increament \
                            + integral_increment
    if duty_cycle >= 1:
        duty_cycle = 1
    elif duty_cycle <= 0:
        duty_cycle = 0
```

Integral Gain



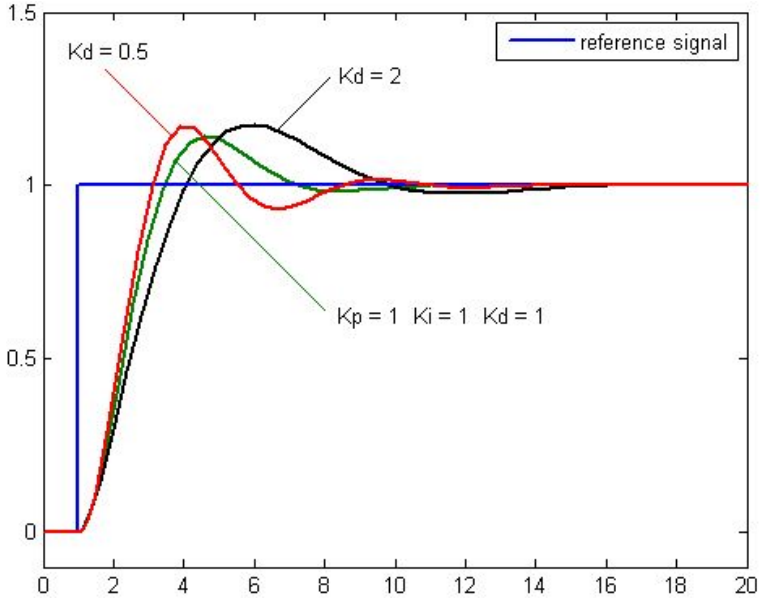
- Compare to Kp, Ki is usually a smaller factor.
- Need a reasonable error buffer size.
- You don't want to save errors at the beginning, set a threshold for Ki to kick in.
- Set point/reference may drastically changes, make sure to clean up the error buffer after that.

Derivative Gain




```
duty_cycle = 0.0
Kp = 0.5
Ki = 0.02
Kd = 0.1
reference_speed = math.pi  # rad/s
error = 0
last_error = 0
error_buffer = [last_error]
while True:
    sensed_speed = compute_speed_from_encoder(encoder_counter)
    error = reference_speed - sensed_speed
    error_buffer.append(error)
    proportional_increment = Kp * error
    integral_increment = Ki * sum(error_buffer)
    derivative_increment = Kd * (error_buffer[-1] - error_buffer[-2])
    duty_cycle = duty_cycle + proportional_increament \
                            + integral_increment \
                            + derivative_increment
    if duty_cycle >= 1:
        duty_cycle = 1
    elif duty_cycle <= 0:
        duty_cycle = 0
    # DO NOT FORGET TO UPDATE LAST ERROR!
    last_error = error
```

Derivative Gain



- Usually, Kp and Ki will do the job.
- Kd can help to stabilize the outcome.
- Set a time interval to calculate derivatives, the error exact one time step before could be noisy.

PID Putting together