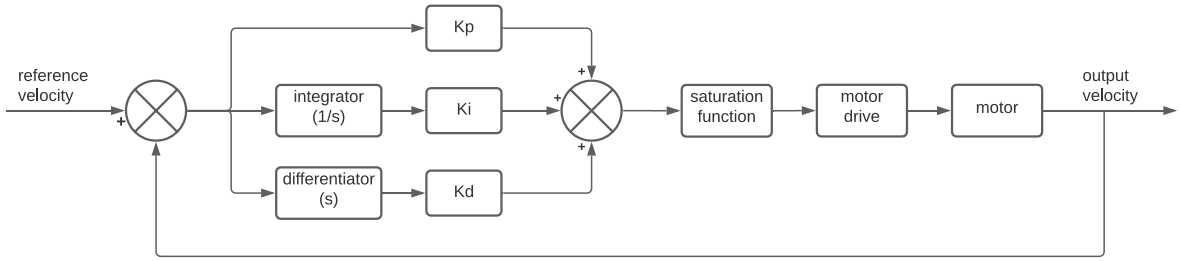
Simple PID controller (Python)

Control block diagram

A PID control algorithm with output saturation is used.



Implementation in Python

The PID controller was implemented in a class named **PIDController**. The code is shown in the next page.

Class Methods and Attributes:

```
class PIDControler(kp, ki, kd, lim_min, lim_max, T):
```

- `kp` – proportional gain
- `ki` – integral gain
- `kd` – derivative gain
- `lim_min` – min value for output saturation
- `lim_max` – max value for output saturation
- `T` – sample time
- `update_pid(reference, measurement)` – this function returns the PID controller output for a given pair of reference and measurement values

Two additional parameters representing the integrator and error memory are also included. (`i` and `e_prev`)

Python code for PID controller

```python
from dataclasses import dataclass


@dataclass
class PIDController:
    # PID gains
    kp: float
    ki: float
    kd: float

    # output saturation
    lim_min: float
    lim_max: float

    # sample time
    T: float

    # memory
    i: float = 0  # integrator
    e_prev: float = 0  # previous error

    def update_pid(self, reference: float, measurement: float):
        # error
        e = reference - measurement

        # proportional
        p = self.kp * e

        # integral
        self.i = self.i + self.ki * e * self.T

        # derivative
        d = self.kd * (e - self.e_prev) / self.T

        # update memory
        self.e_prev = e

        # calculating output and applying output saturation
        output = p + self.i + d
        if output > self.lim_max:
            output = self.lim_max
        if output < self.lim_min:
            output = self.lim_min

        return output
```

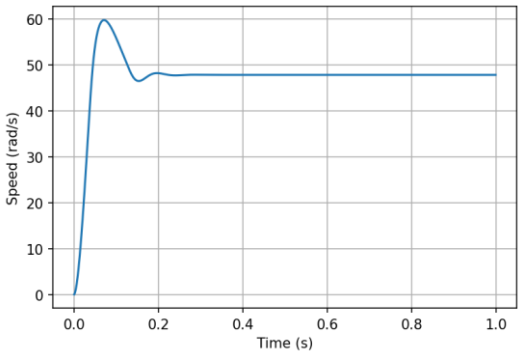
Testing and Tuning

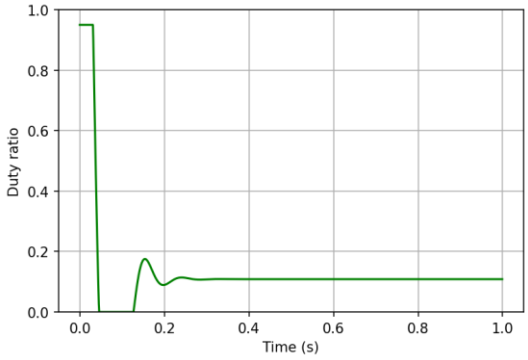
For testing the PID controller, an **armature controlled permanent magnet dc motor** (**24V**) was selected.

Assumptions - A reference speed of $\mathbf{10}\ \boldsymbol{rad/s}$ was used. The sample time was chosen as $\mathbf{1}\boldsymbol{ms}$, which roughly corresponds to the PWM frequency of an Arduino UNO ($1kHz$).

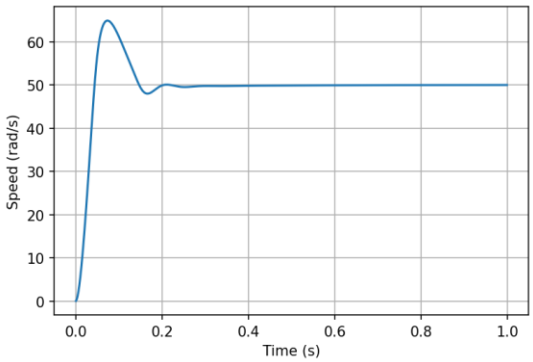
The matplotlib plotting library and the Python control systems library were used to obtain plots of the output response. These plots are given below. Plots of the time variation of both the **motor speed (plant output)** and **duty ratio (controller output)** are shown.

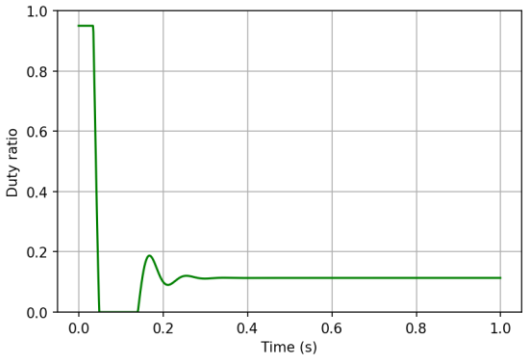
The plots for $\boldsymbol{kp = 0.05}$ are shown below. Here ki and kd were set to 0.





A steady state error is present; therefore, keeping $\boldsymbol{kp = 0.05}$, we introduce $\boldsymbol{ki = 0.015}$.





kd is introduced. The other two parameters are also tweaked to obtain a satisfactory response.

$\boldsymbol{kp = 0.1,\ ki = 0.5,\ kd = 0.004}$