AI for Mobile Robots
- CSIP5202 -

Feedback Control

Overview

- Feedback control vs Open Loop
- Binary Control
- Hysteresis
- Proportional Control
- PID Control
- Tuning a PID Controller

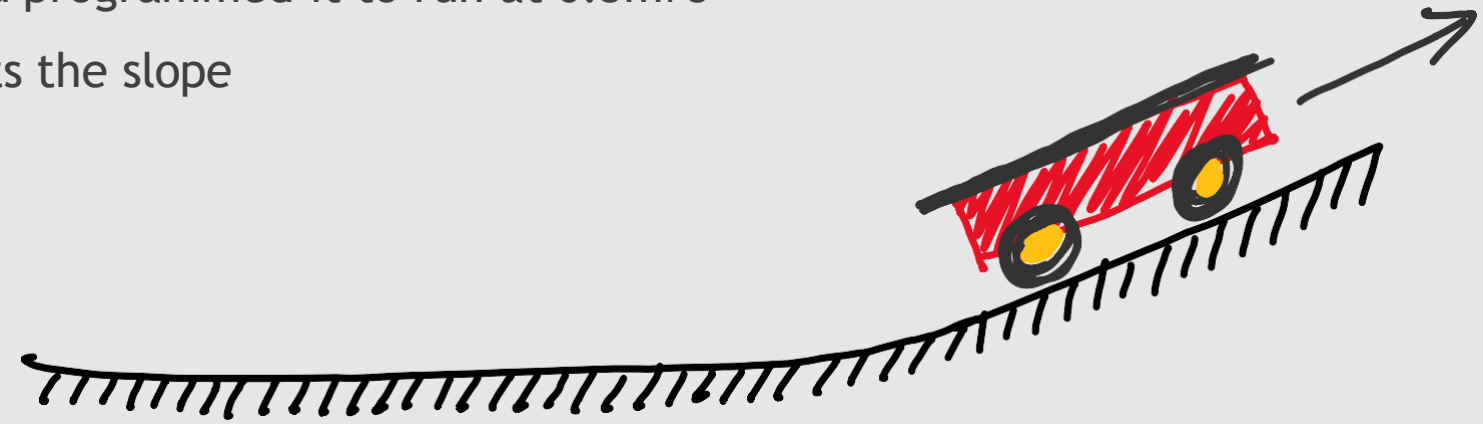
Feedback Control vs. Open Loop

- Imagine a wheel robot
- You want it to run at 0.5m/s

- In open loop control
  - Required motor power setting found empirically for a given situation
  - … but then…

Feedback Control vs. Open Loop

- If the situation changes slightly, e.g. path slopes up
- You programmed it to run at 0.5m/s
- Hits the slope



- Not enough power to run at 0.5m/s up a slope, so speed drops to a different value to which you wanted

Feedback Control vs. Open Loop

- The situations' changes can be due to various common reasons:
  - Battery discharging
  - Loosing/Gaining weight (e.g. picking up objects)
  - Terrain changes:
    - More or less friction
    - Surface Material
    - Surface Gradient
  - Etc

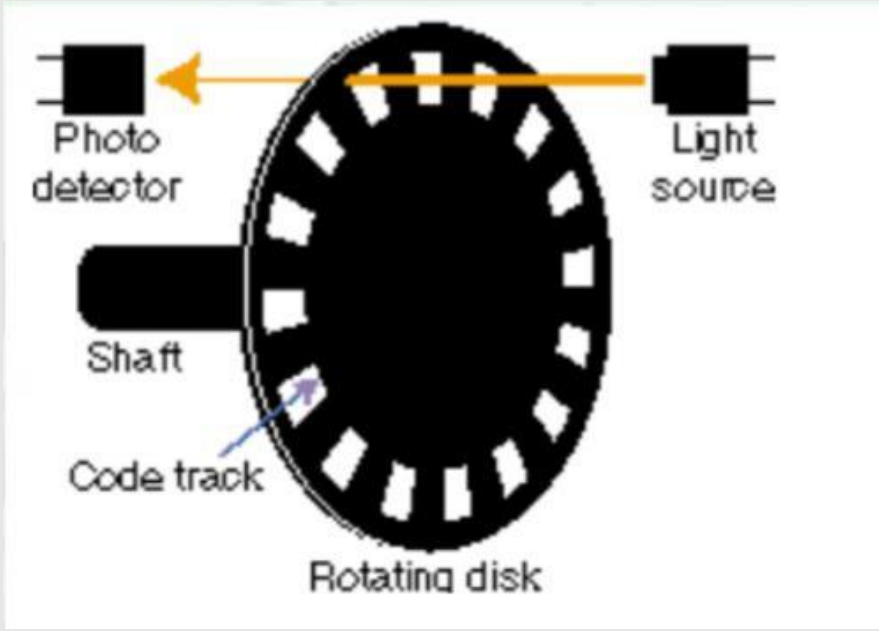
Feedback Control vs. Open Loop

- Solution:
- Measure how fast the robot is travelling
- Apply power accordingly using a transfer function:
  - Binary
  - Proportional
  - Proportional, Integral, Derivative
  - Other (fuzzy, AI based…)

A use for odometry sensors

- To measure the wheels' speed we can use an internal state sensor (encoder) just as the ones used in odometry:

- Each change of light level (pulse) represents a set rotation of the shaft



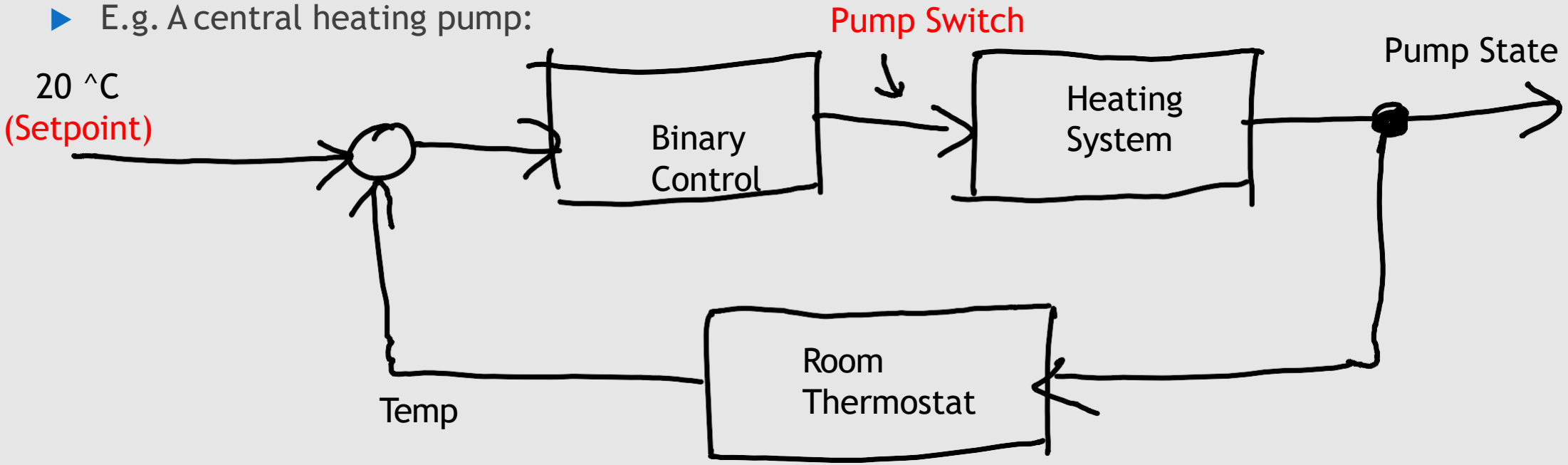
Feedback Control vs. Open Loop

- The iRobot provides the odometry ready calculated and converted to distance
- With the current simulator we cannot measure speed from sensors…
  - Can we measure it in another way?
  - Do we really need to measure the speed? (the maps do not provide slopes or battery changes)

Feedback Control

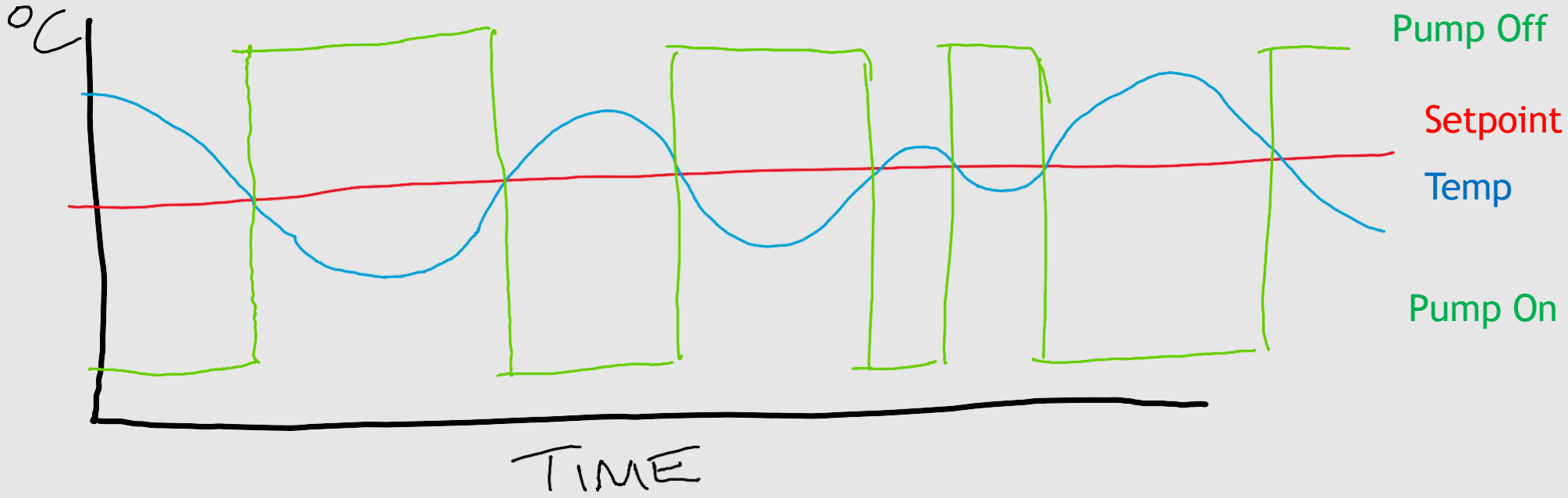
Binary Control

- Binary refers to actuators which can only be ON or OFF
- E.g. A central heating pump:



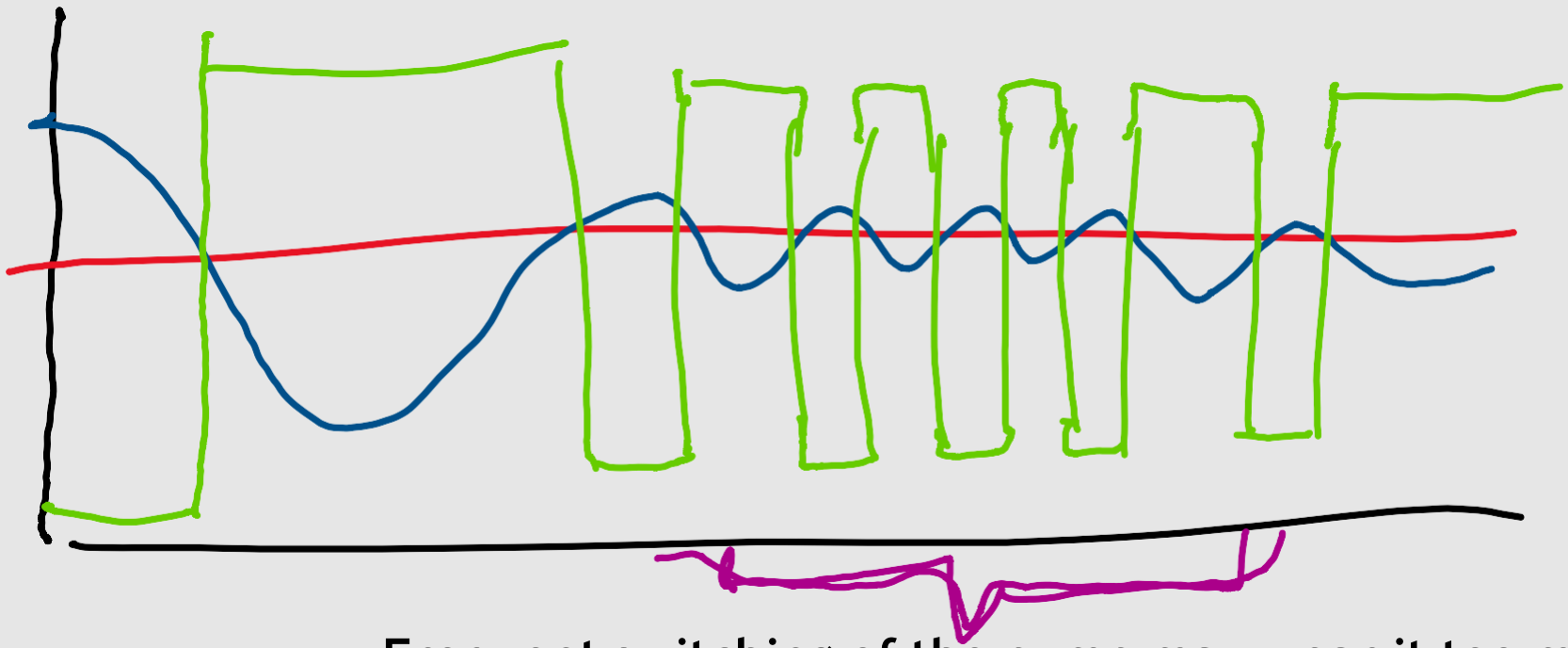
Binary Control

- Simple binary rules:
  - If temperature goes below set point turn heating pump ON
  - If temperature goes above set point turn heating pump OFF



Binary Control

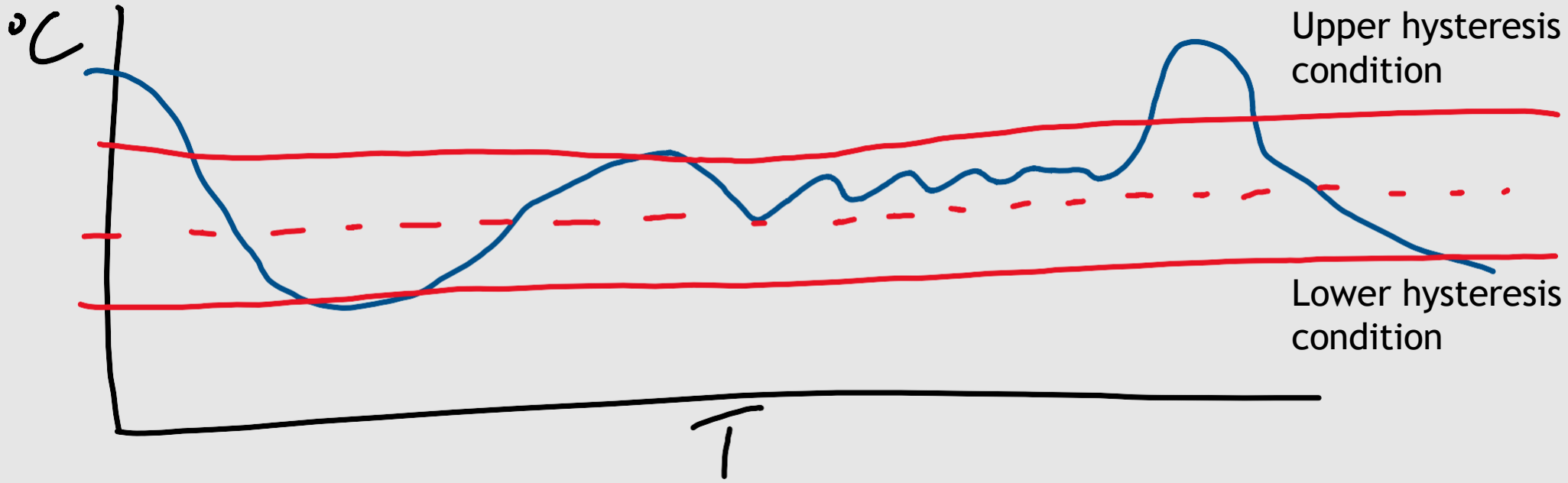
- Problems arise when the controlled variable moves quickly (or frequently) about the set point



Frequent switching of the pump may wear it too much!

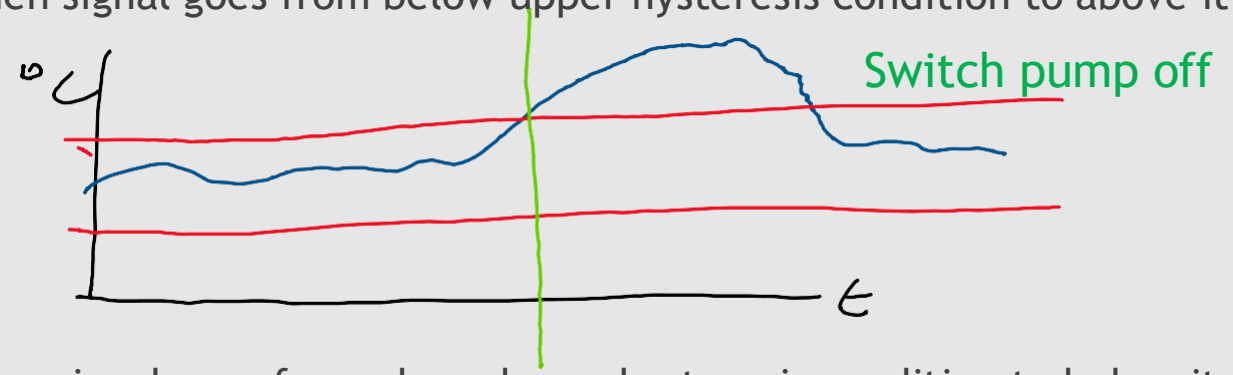
Hysteresis

- Add upper and lower bounds around the set-point, so that we have a region to allow for change without reacting needlessly

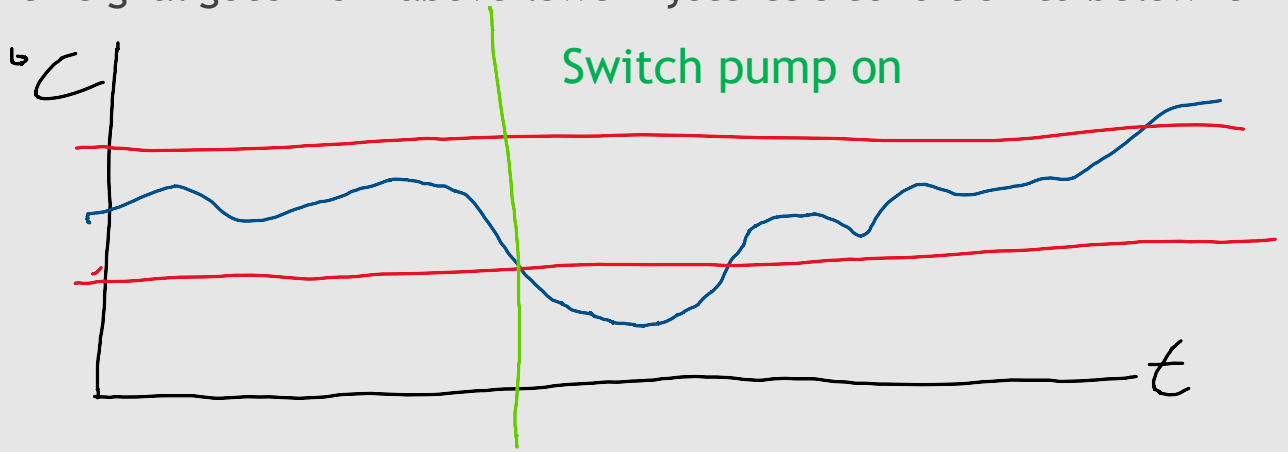


Hysteresis

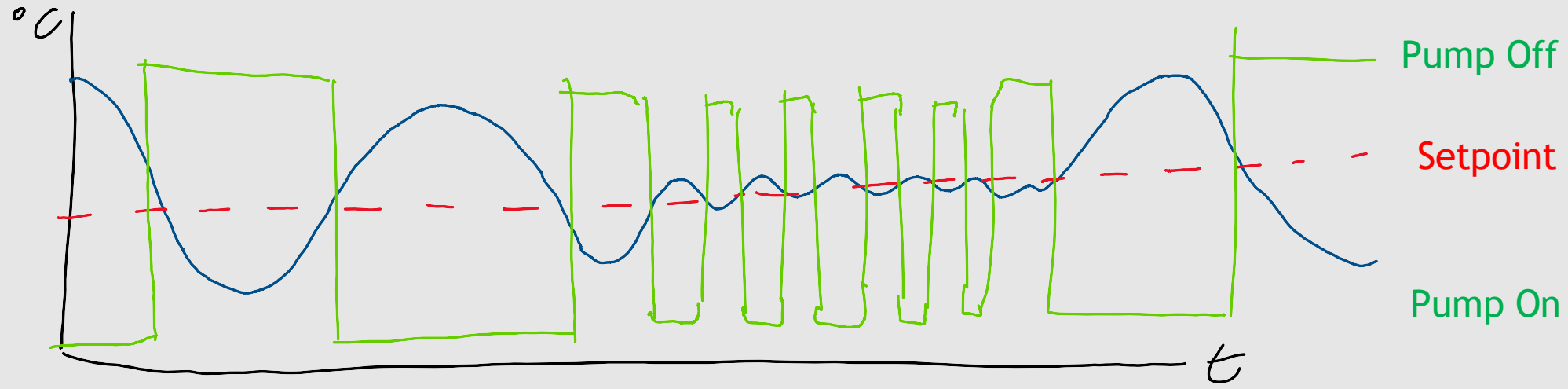
- Two rules:
  - Switch off when signal goes from below upper hysteresis condition to above it

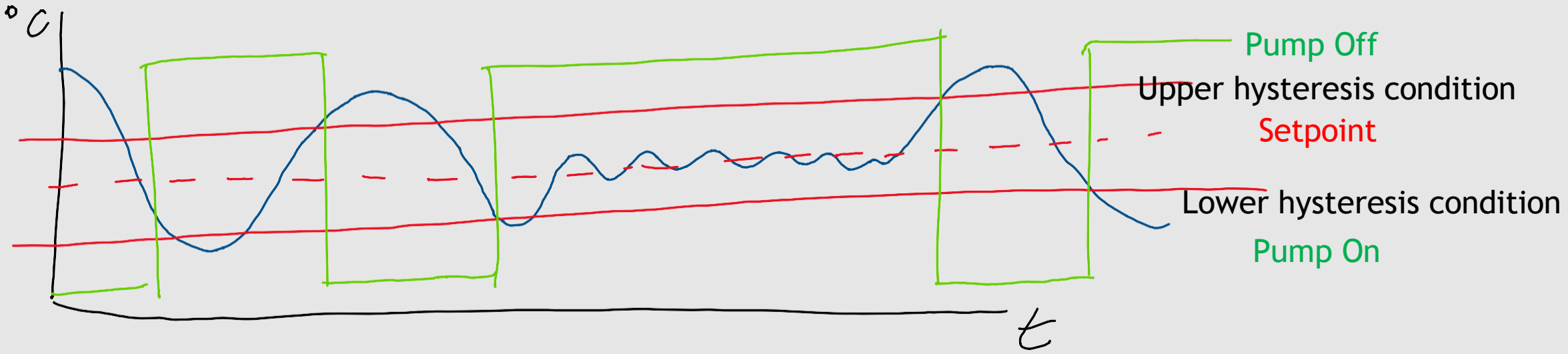


  - Switch on when signal goes from above lower hysteresis condition to below it



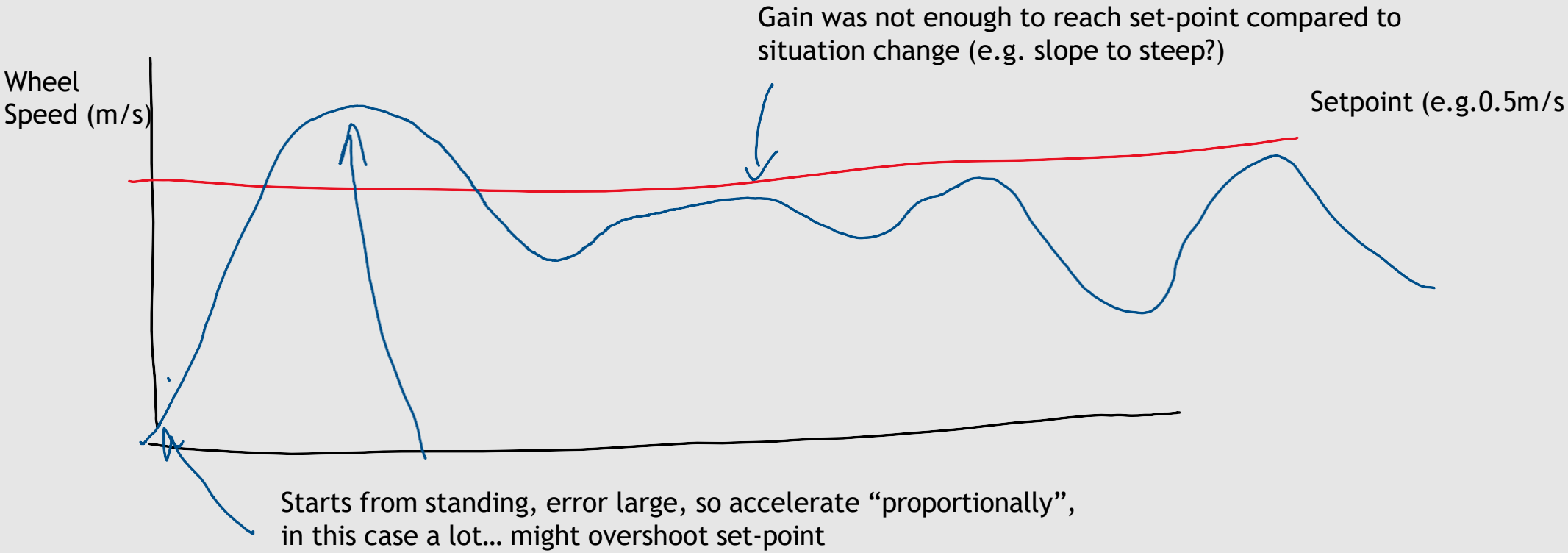
Hysteresis (without and with)





Proportional Control

- Control response (Output) is proportional to the error
- Remember our robot example



Proportional Control

- Control response is proportional to the error

- O = Np + Kp X e(t)
- Output = Nominal Power + Proportional Gain X Error

- Gain is set by the system engineer
- Say Kp = 0.65, and the error right now is 5
- The output signal is 0.65 X 5 = 3.25

- Nominal power = power of open loop controller

Feedback Control

- Feedback control used to adjust the output signal to the conditions
- Binary control:
  - On or Off systems
  - Hysteresis used to reduce switching around the set-point
- Proportional control:
  - Control response is proportional to the error

Feedback Control - PID

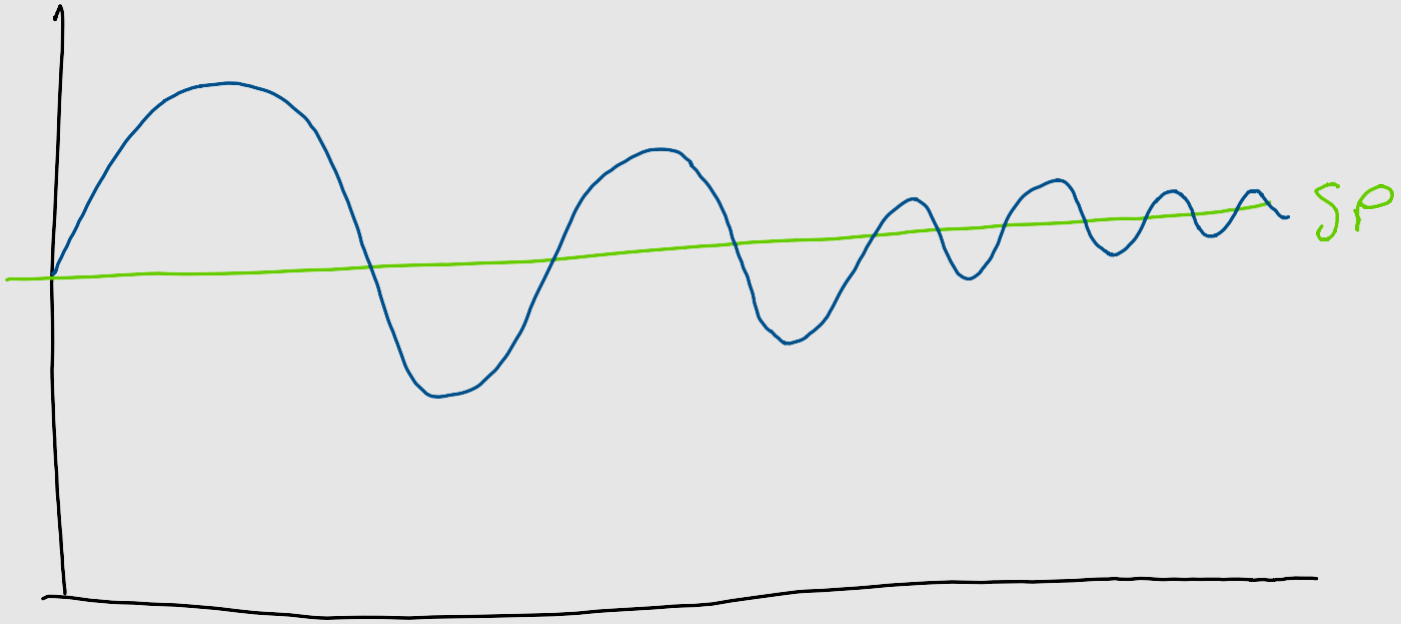
Why PID?

- In a real system you need to be able to adjust the set-point:
  - Speed is not just set to a specific value permanently
  - Need to be able to use any set-point

- With proportional control this isn’t efficient
- Also, in proportional control, when there’s no error, it means no output
- We want a output level to keep the error at zero and stay there

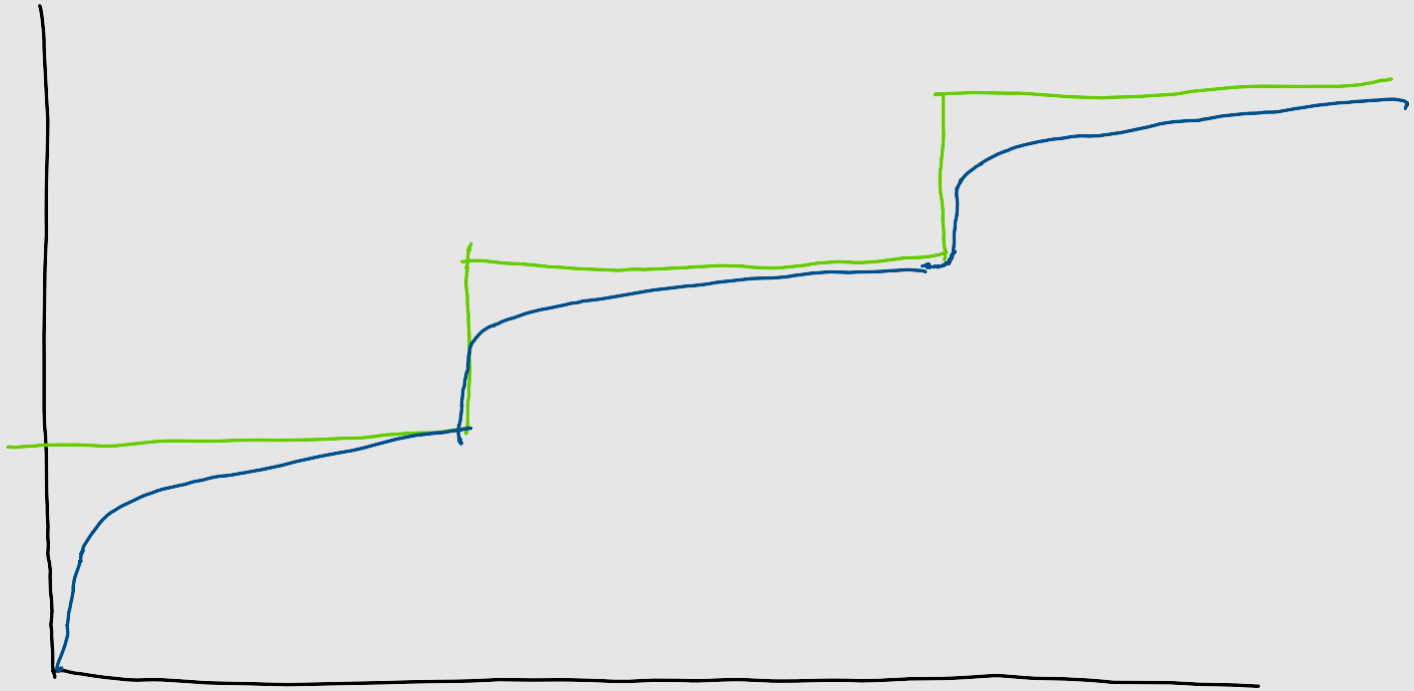
Why PID?

- When error = 0 no output
- With proportional control we keep having an error and correcting it



Why PID?

- We want to follow and stay at every set-point



PID – Proportional-Integral-Derivative

- Most widely used feedback control technique
  - Well understood
  - Well studied
  - Can adapt to different set-points
  - Faster dampening
- Controller made up of three elements:
  - Proportional
  - Integral
  - Derivative

PID

- The Proportional part
- Very simple – gets back to the set-point when there is an error

- $P = K_p \times e(t)$

- P is the output of the proportional controller
- Kp is the gain of the proportional controller
- e(t) is the error at time t

PID

- The Integral part
- Reduces long term errors (tries to keep output setting at 0 error)
- Can be seen as a type of ‘memory’

- $I = Ki \times \int e(\tau) \times d(\tau)$

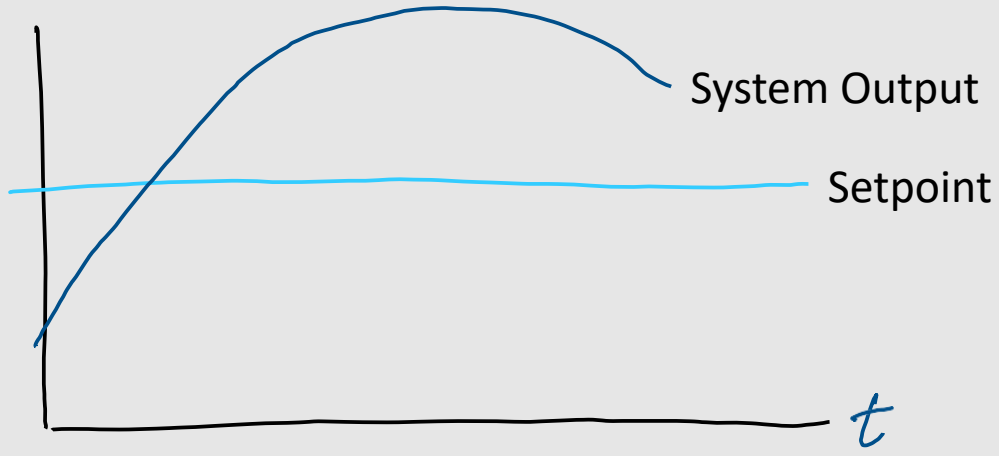
- I is the output of the integral controller
- Ki is the gain of the integral controller
- e is the error
- T is the integration variable (time interval)

PID

- Integral
- Implementing it:
  - Store the error for the last n control cycles
    - (how far back you want to remember?)
  - Sum errors
  - Threshold – make it a sensible value
  - Multiply by gain

PID

- So far…
- Proportional control based on instant error (right now)
- Integral control based on previous errors (memory)
- We need a way of thinking about what might be happening with the error's trend



PID

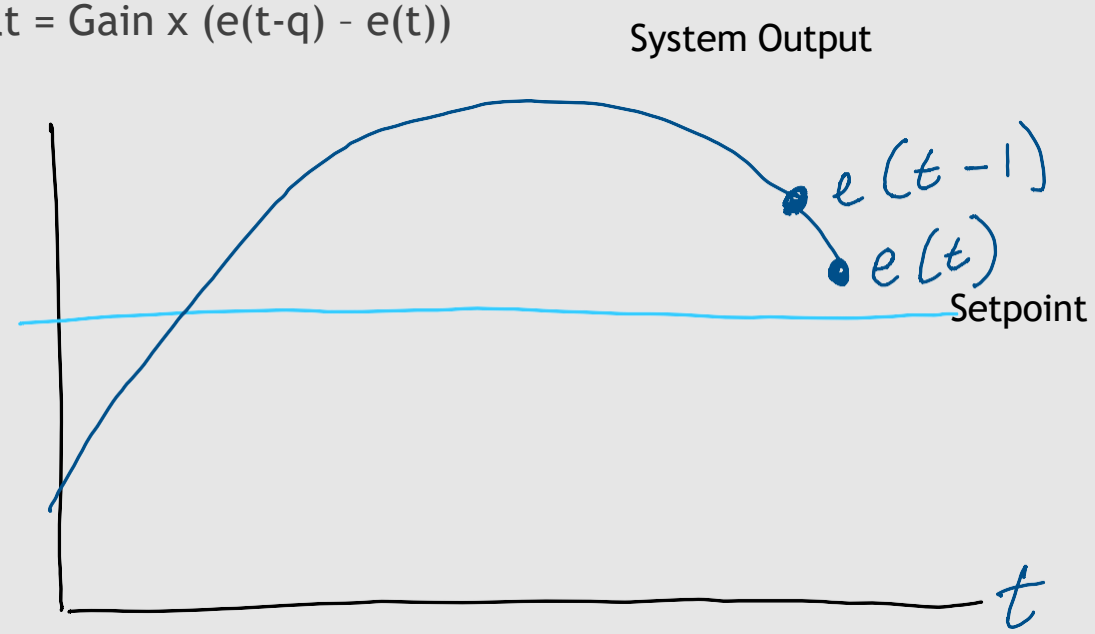
- The Derivative part
- Considers the rate of change (slope) of the error over time (e.g. if the error is large but decreasing we might want to correct less than if it's large and increasing, thus having a more stable controller)

- $D = K_d \times \frac{de(t)}{dt}$

- D is the output of the derivative controller
- Kd is the gain of the derivative controller
- e is the error
- t is the time

PID

- Derivative
- Implementing it:
  - Need to know direction and magnitude of error
  - Result = Gain x (e(t-q) – e(t))



PID

- Proportional, Integral, Derivative
- Putting it all together

- $O = K_p \times e(t) + K_i \times \int e(\tau) \times d(\tau) + K_d \times \frac{de(t)}{dt}$

- O = P + I + D

Tuning

- How do we find good values for the gains?

Tuning

- How do we find good values for the gains?

- Start by setting Kp to a low value say 1
- You may see the system oscillate
- What will this look like of the iRobot?

Tuning

- How do we find good values for the gains?

- Now tune Kd
- Start with a value 100 times Kp
- So Kd = 100
- Raise Kd until the system shows a fast, large oscillation
- Reduce Kd by a factor of 2 or 4

Tuning

- How do we find good values for the gains?

- Now tune Kp
- If the system is oscillating:
  - Drop Kp by a factor of 10 until oscillation stops
- If the system is not oscillating:
  - Increase Kp by a factor of 10 until oscillation begins
  - Drop Kp by a factor of 2 or 4
- Adjust by factor of less than two until it looks good

Tuning

- How do we find good values for the gains?

- Now tune Ki
- Try numbers around 0.0001 to 0.001
- Up them until by a factor of 10 until you get oscillation
- Drop by a factor or 2
- Fine tune

Recap

- Proportional, Integral, Derivative
  - P – fix current error
  - I – fix previous errors
  - D – predict and fix future errors
- Tuning
  - Tune D, then P and finally I
  - Ramp up factors until oscillation
  - Drop by factor of 2 or 4
  - Fine tune

Further Reading

- Read the paper: “PID without PhD” provided on LearningZone in this lecture’s material, it should help with this week’s lab work