Introduction to Robotics

Felipe P. Vista IV

Class Admin Matters

Grading

➢ Attendance 5%

Name (Original Name)	User Email	Join Time	Leave Time	Duration (Minutes)
[redacted]		4/12/2021 9:12	4/12/2021 10:14	62
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:12	4/12/2021 9:14	3
		4/12/2021 9:13	4/12/2021 9:13	1
		4/12/2021 9:13	4/12/2021 9:14	2
		4/12/2021 9:14	4/12/2021 9:14	1
		4/12/2021 9:14	4/12/2021 9:14	1
		4/12/2021 9:14	4/12/2021 10:14	60

Bad ZOOM User Name (Absent)

- Iphone → Not your name
- SiAko 202100001 → Wrong order
- SiAko → Name only
- 202100001 → ID Num only

ZOOM User Name (Present)

- University ID Num_Name
- 202100001 SiAko → GOOD (Present)

Name (Original Name)	User Email	Total Duration (Minutes)
[redacted]		62
		63
		62
		62
		63
		62
		63

Student Responsibilities

- Download/Install ZOOM app for online lecture
  - *Zoom profile must be your* ***OASIS ID+name*** *similar to OASIS*
  - *Ex.:* ***202061234 YourName***
  - ***If you are asked, but no reply, then you'll be out of zoom & mark absent***
- Regularly login, check OLD IEILMS for updates, notifications
  - *https://ieilmsold.jbnu.ac.kr*
  - *Presentations & lecture videos will be uploaded after class*
- Regularly check Kakao Group Chat for class
  - *Everybody must have a Kakao talk account*
  - *Search & add account "**botjok**", introduce yourself and name of class ("Robotics"), then you will be added to the group chat*

Intro To Robotics

CONTROL

Control

- Control Models
- On-Off Control
- Proportional (P) Controller
- Proportional-Integral (PI) Controller
- Proportional-Integral-Derivative (PID) Controller

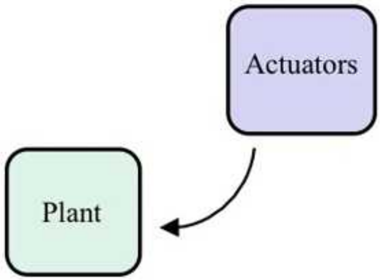
Intro

- Decisions
  - *What robotic algorithms has to make*
  - *Task given → take action (depend on data from sensors)*
  - *Ex: robot on tracks in a warehouse ()*
    - Sensors (navigate back & forth, detect & grasp object)

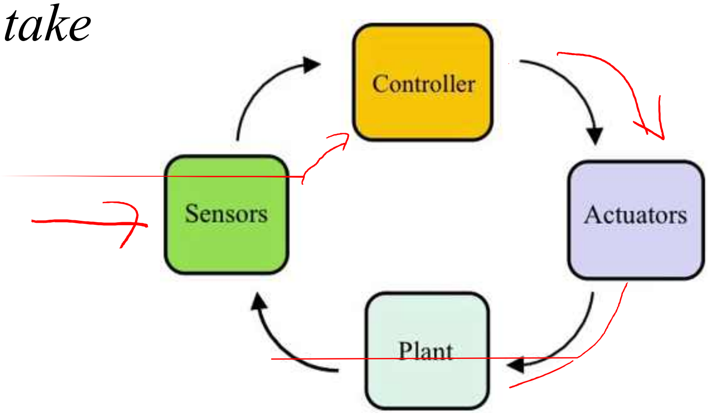
- Only extremely well-defined env tasks carried out w/o sensors
  - *Parts precisely placed in assembly line*
  - *But there would be obstacles in most environments, hence need for sensors*
- Control algorithms
  - *Adapt to small variations in environment*
  - *Sophisticated mathematical theory of control fundamental in robotics*

Control Models

- Two ways control algorithm can decide upon an action
- Open loop system
  - *Parameters of control algorithm are set*
  - *Parameters do not change while system is running*
- Closed loop system
  - *Sensors measure error between desired & actual states*
  - *Error used in deciding what action to take*



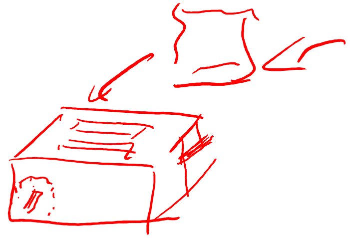
Open-loop system



Closed-loop system

Open Loop Control

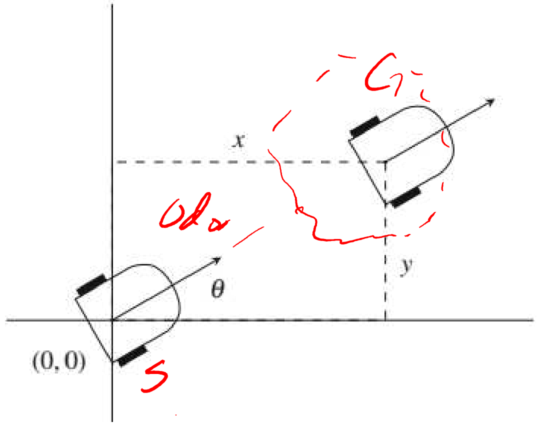
- **Ex. : Toaster (a semi autonomous machine)**
  - *Put bread → set timer → push lever down to start*
    - Result not guaranteed
  - *Timer too short: must toast again*
  - *Timer too long: burnt smell all around*



- Result uncertain because it is an open loop system
  - *Doesn't check if desired result is achieved*
  - *Open loop systems very familiar*
- **Ex. 2: Washing machine:**
  - *Set amt of water (& temp) + set duration of cycle + set amt detergent*
  - *But machine don't measure 'cleanliness' (???) & modify to obtain it*

Open Loop Control

- Mobile robot navigating through odometry alone
  - *Also an open loop control*
- Recall how distance was computed?
  - *Track motor power*
  - *How long it was running*

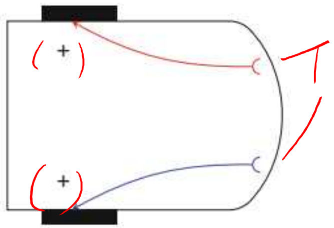


- Uncertainty in final calculated position
  - *Variations in speed of wheels*
  - *Surface where robot moves*

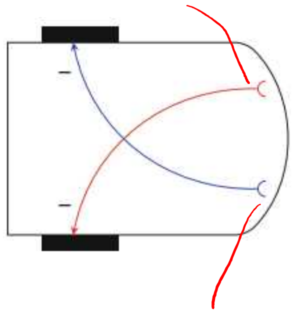
- How then to deal with this?
  In most applications:
  - *Odometry: First for navigating to vicinity of goal position, then*
  - *Sensors: move robot to precise goal position*

Closed Loop Control

- Closed loop control systems
  - *Used by robots to achieve autonomous behaviour*
- **Ex.: Braitenberg vehicles**

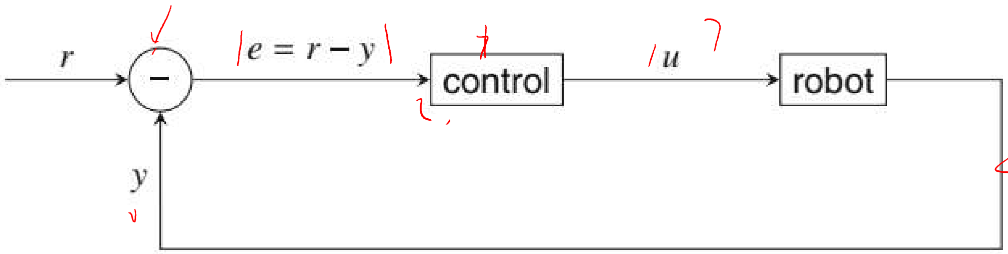


Specification(Attractive & Repulsive):
Object approach robot from behind → run away until out of range.



- Robot must measure distance from object
  - *Stop when distance is large enough*
- Motor power depends on measured distance
  - *But robot speed depends on motor power…*
    - *w/c changes distance to object → modify power setting → ….*
- This circular behaviour is cause of the term "closed loop"

Closed Loop Control



Closed loop control system

r : reference value

- Spec of robot task; Can't be directly used by robot, must be converted to ***u***
- **Ex.: Warehouse** (*robot pos relative to shelf, gripper arm distance from object to pick*)

u : control value

- **Ex.:** (*motor power, time motor running*)

e : error

- ***e*** = ***r*** - ***y***

y : output

- Actual state of robot
- **Ex.:** (*distance to the object*)

Also called **feedback control system**, *where **y** is fed back to control algo*

- *To compute control value **u***
- *Compared with **r** to compute error*
- *Error used to generate **u** that is input to the robot*

Period of a Control Algorithm

```
integer period                    // Duration of timer period
integer timer                     // Timer variable

1:  period ← . . .                           // Timer variable
2:  timer ← period                           // Initialize timer
3:  loop
4:    when timer-expired-event-occurs
5:       control algorithm                   // Run algorithm
6:       timer ← period                      // Reset timer

    // Operating System
7:    when hw-clock-interrupt-occurs
8:       timer ← timer − 1                   // Decrement the timer
9:       if timer = 0                        // Timer expires
10:        raise timer-expired-event         // raise an event
```

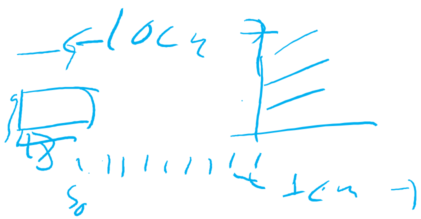
Algorithm 6.1 : Control Algorithm Outline

Period of a Control Algorithm

- Control algorithms run periodically   1-9, A-F 10 15   1 F 16 8 4 2 1
- A *timer variable* is initialized (Ex. Every 20 ms)
  - *Embedded computer's* hardware clock *'tick" at fixed intervals* → interrupt
  - *Interrupt* handled *by OS, decrement timer variable value* → *zero*
  - Timer expired *(zero)* → *event raised in sw (OS)* → *run control algo*
- Period of algorithm   TX: 5ms 4ms 3ms 2 1
  - *Important parameter of control system design*
- Period too short
  - *Waste computing resources* → *computer overloaded* → *commands to robot arrive too late*
- Period too long
  - *Robot not able to respond in time to correct errors in its motion*

Period of a Control Algorithm

- **Ex. : Robot approach an object 10cm away at 2cm/s**
- Control period of 1ms (waste computing resources)
  - *Move at only 0.002cm (0.02mm) per each 1ms cycle of algo*
  - *Changes in power over small distances not affect robot ability to fulfill task*
- On the other hand, 2s ~~2ms~~ (much worse)
  - *Robot move 4cm per each cycle → crash into object*



- Then approx. 0.25s (seems reasonable)
  - *Move 0.5cm, meaningful distance in approaching object*

p=0.5s ⇒ ?
s=1cm/s

- To find optimum period

$2\frac{cm}{s} \cdot 2s = 4cm$

  - *Experiment with various periods around this values*
  - *Achieve satisfactory behaviour that reduce computational cost*

Period of a Control Algorithm

- We now study sequence of four algorithms
  - *Each one building on the previous one*
  - *Providing more accurate control*
  - *But at more computational cost*
- In practice
  - *System designer chooses simplest algorithm*
  - *One that allows robot to fulfill its task*
- The algorithm specs for the robot
  - *Must approach object and stop at s distance in front of it*
  - *Distance measured by a proximity sensor*
  - *Robot speed controlled by setting motor power*

s

x

Control

- Control Models
- On-Off Control
- Proportional (P) Controller
- Proportional-Integral (PI) Controller
- Proportional-Integral-Derivative (PID) Controller

Ex. 1: On-Off Control

- Also called bang-bang algorithm

```
integer reference    // Reference Dist
integer measured     // Measured Dist
integer error        // Dist error
```

```
error ← reference – measured
if error < 0
   left-motor-power ← 50     // Fwd
   right-motor-power ← 50
if error = 0
   left-motor-power ← 0      // Stop
   right-motor-power ← 0
if error > 0
   left-motor-power ← -50    // Bwd
   right-motor-power ← -50
```

Algorithm 6.2 : On-Off Controller

- constant **reference**
  - *Distance from object to stop*
- var **measured**
  - *Distance measured by sensor*
- **error** *(Difference between the two)*
  - *(-) : robot too far away*
  - *(+) : robot too close*

Ex.: error

- *reference = **10cm**, measured = **20cm***
- *10 – 20 → -10*
- *Robot must move forward*

Ex. 1: On-Off Control

- Also called bang-bang algorithm

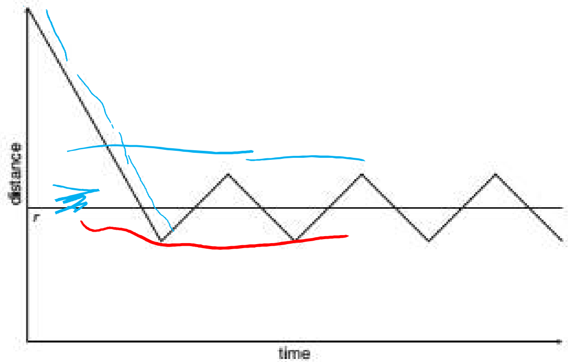
```
integer reference   // Reference Dist
integer measured   // Measured Dist
integer error      // Dist error

error ← reference – measured
if error < 0
   left-motor-power ← 50    // Fwd
   right-motor-power ← 50
if error = 0
   left-motor-power ← 0     // Stop
   right-motor-power ← 0
if error > 0
   left-motor-power ← -50  // Bwd
   right-motor-power ← -50
```

Algorithm 6.2 : On-Off Controller

Ex.: Robot approach at full speed

- *Take time sensor read/send data*
- *then error computed*
- If measured = reference (unlikely)
  - *Can't stop immediately → overrun*
  - *Robot backup full speed → overrun too*
- When *timer → run control algo again*
  - *Same result, oscillate (a)*
- *Unlikely robot stops at/near reference distance*



(a) On-Off Behaviour

Ex. 1: On-Off Control

- Also called bang-bang algorithm

```
integer reference   // Reference Dist
integer measured    // Measured Dist
integer error       // Dist error

error ← reference – measured
if error < 0
   left-motor-power ← 50  low  // Fwd
   right-motor-power ← 50
if error = 0
   left-motor-power ← 0     // Stop
   right-motor-power ← 0
if error > 0
   left-motor-power ← -50  // Bwd
   right-motor-power ← -50
```

Algorithm 6.2 : On-Off Controller

Further disadvantages:

- Frequent & abrupt reversal of direction
  - *Result to high acceleration*
  - *If controlling gripper arm*
    - Object carried may be damaged
- Cause high level wear & tear
  - *On motors*
  - *Other mechanical moving parts*

Control

- Control Models
- On-Off Control
- Proportional (P) Controller
- Proportional-Integral (PI) Controller
- Proportional-Integral-Derivative (PID) Controller

Ex. 2: Proportional (P) Controller

- Inspired by riding a bicycle to develop better algorithm

```
integer reference   // Reference Dist
integer measured    // Measured Dist
integer error       // Error
float   gain        // Proportional Gain
integer power       // Motor Power
```

```
1:  error ← reference – measured
                            // Distances
2:  power ← gain * error
                            // Control value
3:  left-motor-power ← power
4:  right-motor-power ← power
```

Algorithm 6.3 : Proportional Controller

Traffic light turn red while riding a bike

- Don't wait last moment → brake hard
  - *Fly off the bike!*
- Better thing is slow down gradually
  - *Stop pedaling (slow down) → brake gently (slow down more) → @ stop line going slowly : squeeze harder to stop fully*
- Algorithm can be expressed as:
  "Reduce your speed more as you get closer to the reference distance."

Ex. 2: Proportional (P) Controller

- Inspired by riding a bicycle to develop better algorithm

```
integer reference   // Reference Dist
integer measured    // Measured Dist
integer error       // Error
float   gain        // Proportional Gain
integer power       // Motor Power
```

```
1:   error ← reference – measured
                                // Distances
2:   power ← gain * error
                                // Control value
3:   left-motor-power ← power
4:   right-motor-power ← power
```

Algorithm 6.3 :
Proportional (P) Controller

Decrease in speed *inversely proportional* to distance to traffic light

$$\text{decrease in speed} \propto \frac{1}{\text{dist traffic light}}$$

- The closer to light, the more we slow down
- Factor of proportionality called ***gain***

Ex. 2: Proportional (P) Controller

- Inspired by riding a bicycle to develop better algorithm

```
integer reference   // Reference Dist
integer measured    // Measured Dist
integer error       // Error
float   gain        // Proportional Gain
integer power       // Motor Power

1:  error ← reference − measured
                       // Distances
2:  power ← gain * error
                       // Control value
3:  left-motor-power ← power
4:  right-motor-power ← power
```

Algorithm 6.3 : Proportional (P) Controller

Ex.:

- **reference** = ***100cm***, **gain** = ***-0.8***
- If robot 150cm away
  - **error** = *100 – 150 = -50*
  - **power** = *-0.8 * -50* = ***40***
- If robot overrun **reference** & **measured** = ***60***
  - **power** = -32 (backwards)

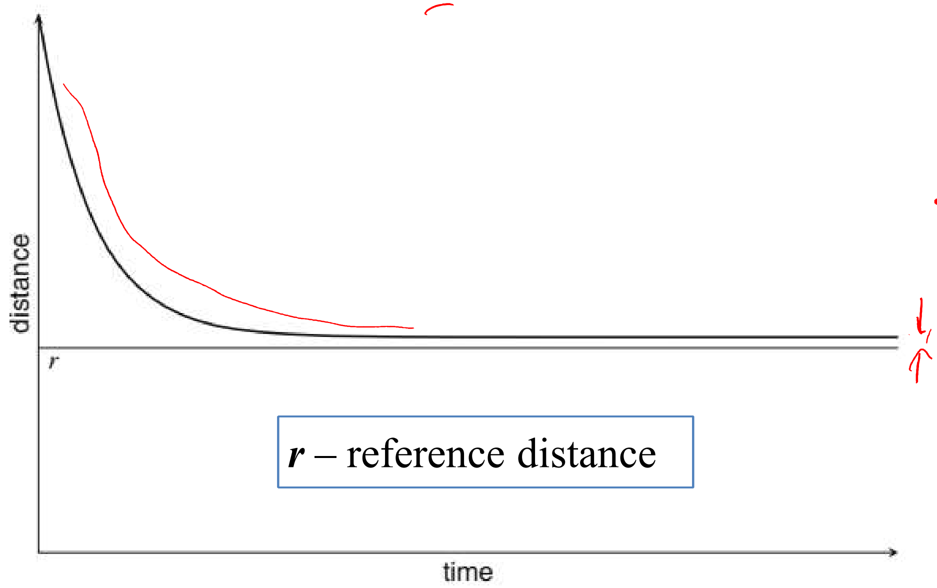
Distance	Error	Power
150	−50	40
125	−25	20
60	40	−32

Proportional controller (gain= -0.8)

Ex. 2: Proportional (P) Controller

- (a) shows change in motor power is smooth
  - *No rapid acceleration/deceleration*
  - *Response somewhat slow, still approach target distance*
- But, robot don't actually reach reference distance, why?

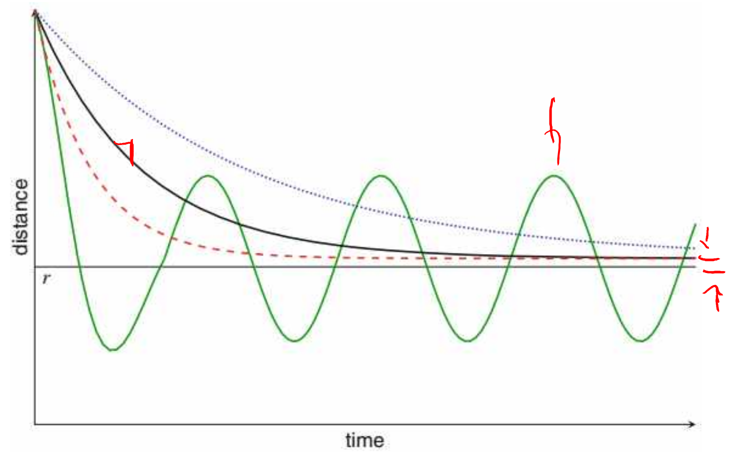
stable



(a) Behaviour of P-controller

Ex. 2: Proportional (P) Controller

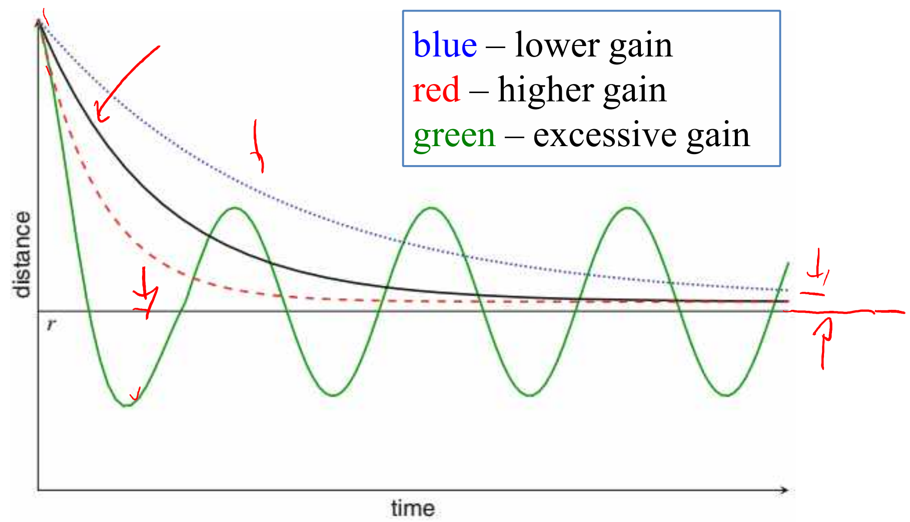
- But, robot don't actually reach reference distance, why?
  - *Consider robot very near reference distance*
  - *Theoretically: low power setting → slow movement → reach reference distance*
  - *Practical: very low power setting → cannot overcome internal friction (motors & connection to wheels) → robot stops moving*
- Increasing gain
  - *Can overcome the problem*
  - *Serious disadvantage with high gain*



(b) Effect of gain on P-controller

Ex. 2: Proportional (P) Controller

- Increasing gain can overcome the problem
  - *Higher gain : approach reference distance faster*
  - *Lower gain : approach reference distance slower*
  - *Excessive (too high) gain : P-controller act like on-off oscillating response*
    - Then controller is **unstable**



converge

(b) Effect of gain on P-controller

Ex. 2: Proportional (P) Controller

- Sometimes, reference can't be reached
  - *Even in an ideal system*
- Assume **object** moving away from our robot at constant speed
  - *max power (catch-up) → @measured is small: very low power (slower than object, hence will never catch up), OR*
  - *max power (catch-up) → @measured = reference : zero power (stop, but object is still moving) → robot will start again*
- This Start-and-Stop motion
  - *not intended goal in maintaining reference distance*

Ex. 2: Proportional (P) Controller

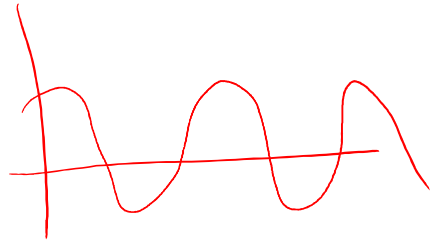
- Sometimes, reference can't be reached even in an ideal system
- Assume **object** moving away from our robot at constant speed

Ex.: Object moves = 20cm/s
- **reference** = 100cm, **gain** = -0.8
- If robot 150cm away, **power** = *40* **(catch up)**
- If robot 125cm away, **power** = *20* **(maintain)**
- If robot 115cm away, **power** = *8* **(back away)**

Distance	Error	Power
150	−50	40
125	−25	20
110	−10	8

Proportional controller moving object (gain= -0.8)

- Generally, robot stabilizes fixed dist from reference dist
- Reduce this error by increasing gain
  - *But reference distance will never be reached*
  - *Cause controller to be unstable*



Control

- Control Models
- On-Off Control
- Proportional (P) Controller
- **Proportional-Integral (PI) Controller**
- Proportional-Integral-Derivative (PID) Controller

Ex. 3: Proportional-Integral (PI) Controller

- PI-controller can achieve reference distance
  - *Even with friction or moving object*
  - *by taking into account accumulated error over time*
- While P-controller only take into account current error:

$$u(t) = k_p e(t) \quad \rightarrow P$$

- PI adds integral of error from start to current time of running algo

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\, d\tau \quad \rightarrow PI$$

- Separate gain factors for proportional & integral terms
  - *To allow flexibility in design of the controller*

Ex. 3: Proportional-Integral (PI) Controller

- Discrete approximation to continuous integral performed
  - *When implementing PI controller*

```
integer reference    // Reference Dist
integer measured     // Measured Dist
integer error        // Error
integer error-sum ← 0 //Cumulative Error
float   gain-p ← ...  // Proportional Gain
float   gain-i ← ...  // Integral Gain
integer power ← ...  // Motor power
```

```
1:  error ← reference – measured
                        // Distances
2:  error-sum ← error-sum + error
                        // Integral Term
3:  power ← gain-p * error +
      gain-i * error-sum // Control value
4:  left-motor-power ← power
5:  right-motor-power ← power
```

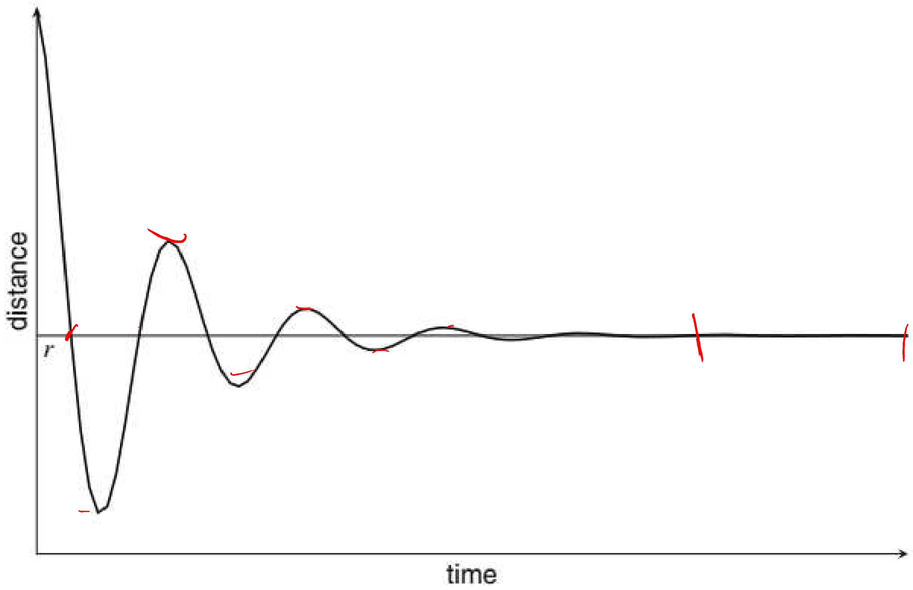
Algorithm 6.4 : PI Controller

With friction or moving object:

- Error will be integrated
  - Cause higher motor power → converge to reference distance
- Problem is
  - Error integration start from initial state (robot far from object)
  - Integral term already large value (when approaching reference)
  - Must move past reference → errors of opposite sign (to reduce error value)
  - Can cause oscillations

Ex. 3: Proportional-Integral (PI) Controller

- Discrete approximation to continuous integral performed
  - *When implementing PI controller*



Behaviour of PI controller

With friction or moving object:

- Error will be integrated
  - Cause higher motor power → converge to reference distance
- Problem is
  - Error integration start from initial state (robot far from object)
  - Integral term already large value (when approaching reference)
  - Must move past reference → errors of opposite sign (to reduce error value)
  - Can cause oscillations

Control

- Control Models
- On-Off Control
- Proportional (P) Controller
- Proportional-Integral (PI) Controller
- Proportional-Integral-Derivative (PID) Controller

Ex. 4: Proportional-Integral-Derivative (PID) Controller

- When throwing/kicking a ball to moving player
  - *Do not throw to player's current position*
    - When ball arrives at pos you threw it, player already gone
  - *Instead, estimate where new position will be & throw/kick ball there*
- Similarly, robot carrying parcel to moving trolley
  - *Put down parcel exact time at estimated pos of trolley*

- Control algorithm
  - *Can't be On-Off, P-Controller*
    - Consider only current error
  - *Same with PI-Controller*
    - Consider only current + previous errors
  - *We have to consider future error*

Ex. 4: Proportional-Integral-Derivative (PID) Controller

- To estimate future error
  - *Rate of change of error can be taken into account*
  - *If small rate of change: put parcel just before trolley approaches*
  - *If large rate of change: put parcel much earlier*
- Mathematically
  - *Rate of change expressed as a derivative*
  - *PID controller adds an additional term to* **P** *and* **I** *terms*

$\frac{d}{dt}$

$$u(t) = k_p e(t) + k_i \int_{\tau=0}^{t} e(\tau)\, d\tau + k_d \frac{de(t)}{dt}$$

- Differential approximated by difference bet previous & current errors
  - *In the implementation of a PID controller*

Ex. 4: Proportional-Integral-Derivative (PID) Controller

- Differential approximated by difference bet previous & current errors
  - *In the implementation of a PID controller*

```
integer reference   // Reference Dist
integer measured  // Measured Dist
integer error          // Error
integer error-sum ← 0 //Cumulative Error
integer prev-error ← 0 // Prev Error
integer error-diff ← 0  // Error Difference
float     gain-p ← ...  // Proportional Gain
float     gain-i ← ...   // Integral Gain
float     gain-d ← ...  // Derivative Gain
integer power ← ...  // Motor power
```

```
1:   error ← reference – measured
                                       // Distances
2:   error-sum ← error-sum + error
                                       // Integral Term
3:   error-diff ← error – prev-error
                                    // Differential Term
4:   prev-error ← error// Save curr error
5:   power ← gain-p * error +   P
       gain-i * error-sum +   I   D
       gain-d * error-diff   // Control value
6:   left-motor-power ← power
7:   right-motor-power ← power
```

Algorithm 6.5 : PID Controller

Ex. 4: Proportional-Integral-Derivative (PID) Controller

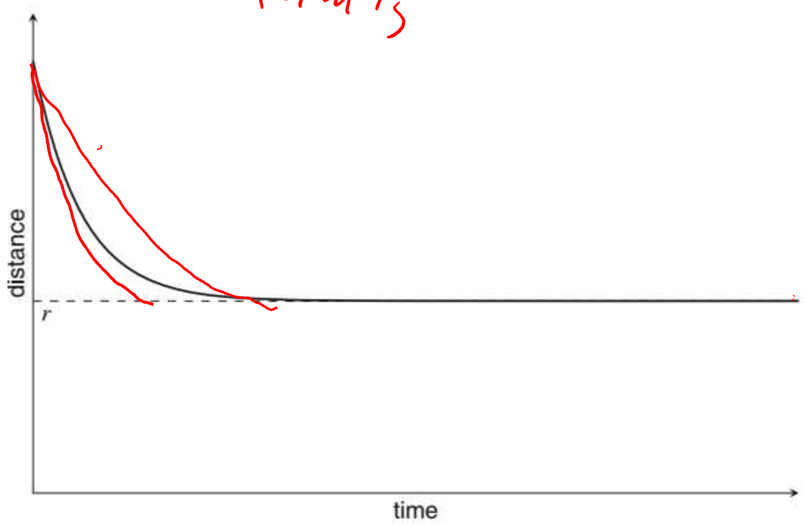
- Plot shows smooth & rapid convergence to reference
- Gains of PID controller must be balanced
  - *If gains for P & I too high : oscillations can occur*
  - *If gain for D is too low : controller reacts to short bursts of noise*

$K_P = ?$

$K_I = ?$

$K_D = ?$

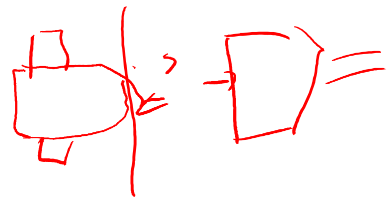
"tuning"



PID Controller Behaviour

[0.8 0.3 0.4

[0.9 1.2 0.4



Summary

- **Good** control algorithm
  stop & go
  - *Converge rapidly to desired result while avoiding abrupt motion*
  - *Computationally efficient & don't need constant tuning*
  - *Adapt to specific requirements of system & task* , V = c ; ΔV
  - *Correctly function in varying environmental conditions*
- Discussed **four algorithms**
  - *From the impractical On-Off algorithms → algorithms that combine proportional, integral & derivative terms*
- **Functions** of the terms
  - *Proportional terms : ensure large errors → rapid convergence to reference*
  - *Integral terms : ensure reference can actually be attained*
  - *Derivative terms : algorithm becomes more responsive*

Thank you.