Control Theory

Maya Nasr
MIT-IIT Robotics Program 2017

Measure X axis Voltage

To measure X axis voltage

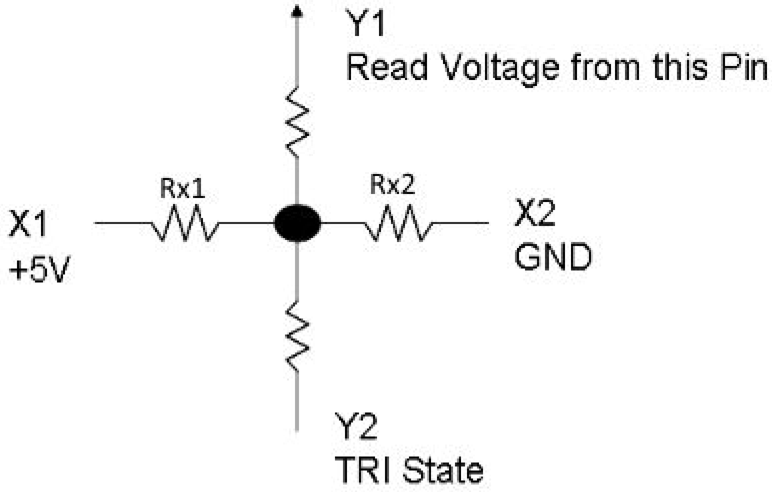
a. We are going to measure voltage on Y1
-> set Y1 pin as INPUT

b. Make Y2 Tristate (remove its influence from circuit)
-> set Y2 as INPUT but LOW

c. Form a voltage divider in X1(+5V) and X2(GND)
-> set X1 as OUTPUT but HIGH
set X2 as OUTPUT but LOW

d. Read the ADC from Y1 pin (analogRead)

Reading of X axis touch position Rx1 and Rx2 Varry according to touch position so the voltage at Y1



blog.circuits4you.com

Similarly Measure Y axis Voltage

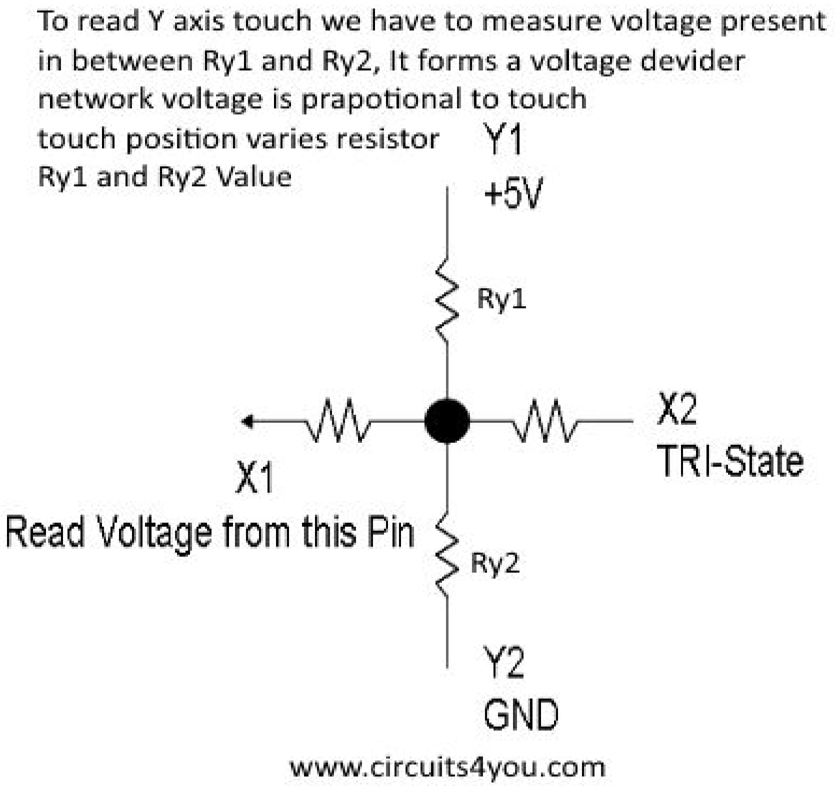
To measure Y axis voltage

a. We are going to measure voltage on X1

b. Make X2 Tristate (remove its influence from circuit)

c. Form a voltage divider in Y1(+5V) and Y2(GND)

d. Read the ADC from X1 pin (analogRead)



Touchscreen Arduino Code

Write an arduino code that gives the X and Y coordinates of a touch point.

Don't forget to:

- Define your Touch screen connection: (Y+ is A0, X+ is A1, Y- is A2 and X- is A3)

Touchscreen Arduino Code

```
4- Wire Touchscreen Connections
=================================*/
//Define your Touch screen connections

#define XM A3
#define YM A2
#define XP A1
#define YP A0

void setup()
{
   Serial.begin(9600);
}
```

```
void loop()
{
  int X,Y; //Touch Coordinates are stored in X,Y variable
   pinMode(YP,INPUT);
   pinMode(YM,INPUT);
   digitalWrite(YP,LOW);
   pinMode(XP,OUTPUT);
   digitalWrite(XP,HIGH);
   pinMode(XM,OUTPUT);
   digitalWrite(XM,LOW);
   X = (analogRead(YP)); //Reads X axis touch position

   pinMode(XP,INPUT);
   pinMode(XM,INPUT);
   digitalWrite(XM,LOW);
   pinMode(YP,OUTPUT);
   digitalWrite(YP,HIGH);
   pinMode(YM,OUTPUT);
   digitalWrite(YM,LOW);
   Y = (analogRead(XP)); //Reads Y axis touch position

  //Display X and Y on Serial Monitor
   Serial.print("X = ");
   Serial.print(X);
   Serial.print(" Y = ");
   Serial.println(Y);
   delay(100);
}
```

Overview

- What is a PID controller
- P- Proportional Control
  - Examples/Disadvantages
- I- Integral Control
  - Examples/Disadvantages
- D- Derivative Control
  - Examples/Disadvantages
- PID Control
- The basic P controller
- Ball on Beam Lab

PID Control

- PID stands for **P**roportional-**I**ntegral-**D**erivative controller

PID Control

- PID stands for **P**roportional-**I**ntegral-**D**erivative controller

- It is one of the basic types of controllers

PID Control

- PID stands for **P**roportional-**I**ntegral-**D**erivative controller
- It is one of the basic types of controllers
- Over-simplified way of ***What it is:***

It's a way to get motors to do what you want them to do efficiently and smoothly.

PID Control

- Use what you know from your sensors to compute an “intelligent” motor output.

PID Control

- Use what you know from your sensors to compute an "intelligent" motor output.

- Think about it as the idea that you need to slow down as you get close so you don't overshoot the target.

The Problem

- Suppose you have a motor controlled robotic arm

The Problem

- Suppose you have a motor controlled robotic arm
- You want to use this robotic arm to lift a drink to your mouth.

The Problem

- Suppose you have a motor controlled robotic arm
- You want to use this robotic arm to lift a drink to your mouth.
- You can’t just randomly give the motor voltage until it arrives at your mouth

The Problem

- Suppose you have a motor controlled robotic arm
- You want to use this robotic arm to lift a drink to your mouth.
- You can't just randomly give the motor voltage until it arrives at your mouth
- Might end up lighting your robotic arm on fire, or spilling the drink, or hitting yourself with it, etc.

The Problem

- Suppose you have a motor controlled robotic arm
- You want to use this robotic arm to lift a drink to your mouth.
- You can't just randomly give the motor voltage until it arrives at your mouth
- Might end up lighting your robotic arm on fire, or spilling the drink, or hitting yourself with it, etc.

What you need is some kind of "control"

P — Proportional Control

P — Proportional Control

- Your voltage is related to speed (may be nonlinearly depending on controllers, motors, friction, etc).

P — Proportional Control

- Your voltage is related to speed (may be nonlinearly depending on controllers, motors, friction, etc).

- Assume that the general relationship that an increase in voltage generally results in an increase in speed.

P — Proportional Control

- Your voltage is related to speed (may be nonlinearly depending on controllers, motors, friction, etc).
- Assume that the general relationship that an increase in voltage generally results in an increase in speed.
- Want to do is avoid spilling the drink, by causing the arm to overshoot the target value.

P — Proportional Control

- **Step one** is to measure the difference between where the arm is (far from your mouth) and where you want it to be (at your mouth). Call this "**error**".

P — Proportional Control

- **Step one** is to measure the difference between where the arm is (far from your mouth) and where you want it to be (at your mouth). Call this "**error**".

- **Step two** is give the motor voltage **P**roportional to the error:

  i.e. supply more voltage further away, and less voltage as you come near.

Disadvantages of P Control

- After you do this, you might see that your drink asymptotically approaches your mouth, but doesn't get there.

Disadvantages of P Control

- After you do this, you might see that your drink asymptotically approaches your mouth, but doesn't get there.

- Due to frictional effects and mathematical reasons, the speed approaches zero faster than the voltage does, and two things may occur:

    1. The motor draws a lot of current (inefficient)

    2. The drink remains out of reach of your mouth — it **undershoots**.

I — Integral Control

I — Integral Control

- Measure not only the error of the arm, but also **how much that error has changed** since the last time you checked.

I — Integral Control

- Measure not only the error of the arm, but also **how much that error has changed** since the last time you checked.
- If the error has not changed as much as you wanted it to, add a little voltage.

I — Integral Control

- Measure not only the error of the arm, but also **how much that error has changed** since the last time you checked.
- If the error has not changed as much as you wanted it to, add a little voltage.
- Repeat this over and over again, really fast, and now you have what approximately amounts to an **I**ntegration.

I — Integral Control

- Measure not only the error of the arm, but also **how much that error has changed** since the last time you checked.
- If the error has not changed as much as you wanted it to, add a little voltage.
- Repeat this over and over again, really fast, and now you have what approximately amounts to an **I**ntegration.

Now you've got enough voltage!

Disadvantages of I Control

And yet, two things may occur:

1. As these little bits of voltage stack up, you get enough energy per charge to get that arm going again — but you might hit yourself with the drink. You've **overshot.**

Disadvantages of I Control

And yet, two things may occur:

1. As these little bits of voltage stack up, you get enough energy per charge to get that arm going again — but you might hit yourself with the drink. You've **overshot.**

2. As you shoot past your target error value, P and I become negative, so your arm switches direction and goes the other way… And overshoots, moving too low, causing the values to switch signs, so it comes back up and hits you in the face again, and then drops too low… You've got an **oscillation.**

D — Derivative Control

Now you're getting frustrated by not being able to get you drink!

D — Derivative Control

Now you’re getting frustrated by not being able to get you drink!

- You want to do is damp things down, chill them out a bit.

D — Derivative Control

Now you're getting frustrated by not being able to get you drink!

- You want to do is damp things down, chill them out a bit.
- What you do now is measure how fast the error is changing — take its **D**erivative.

D — Derivative Control

Now you're getting frustrated by not being able to get you drink!

- You want to do is damp things down, chill them out a bit.
- What you do now is measure how fast the error is changing — take its **D**erivative.
- If the error is changing too fast, you cut the voltage. You can do this very quickly because you're a robot.

PID Control

- The **D**erivative control compensates for the overshoots of the **I**ntegral control.

PID Control

- The **D**erivative control compensates for the overshoots of the **I**ntegral control.

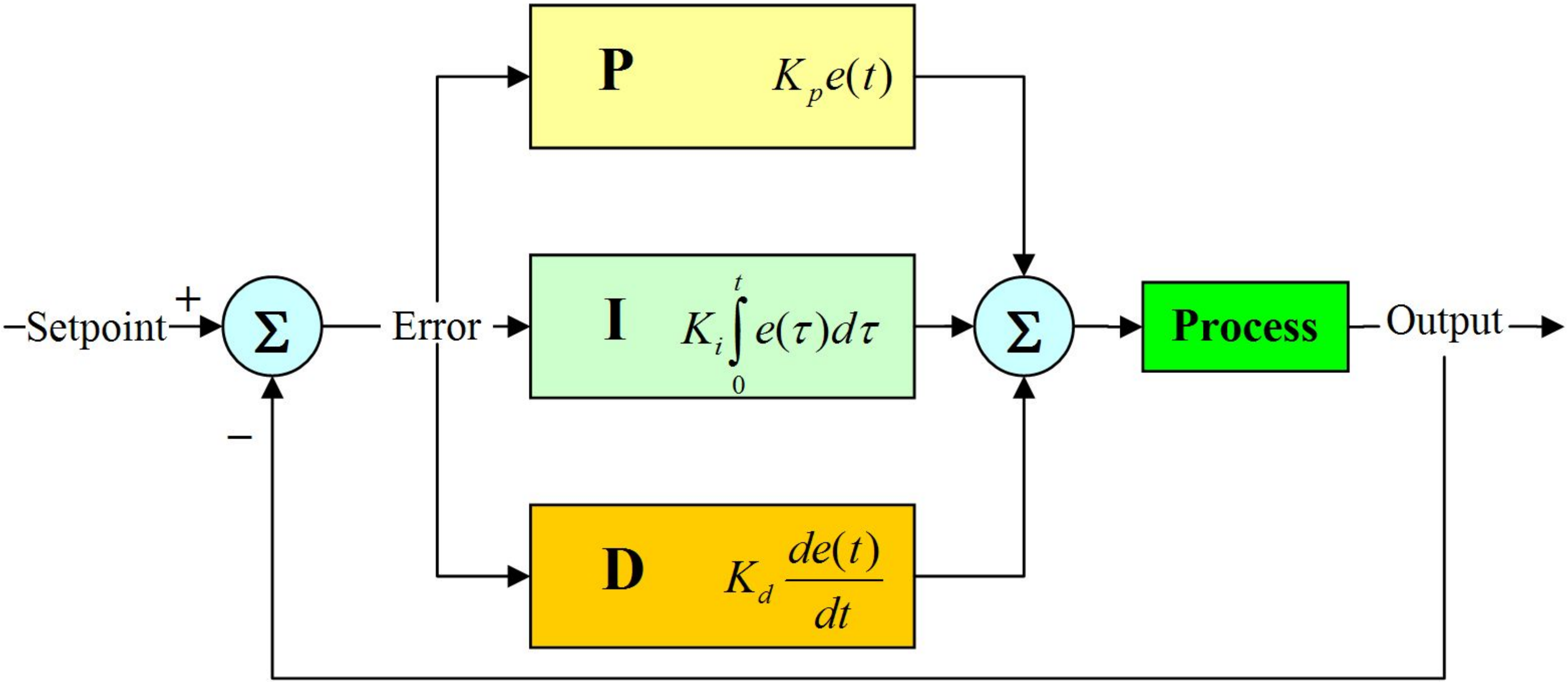
- **I**ntegral control compensates for the undershoots of the **P**roportional control.

PID Control

- The **D**erivative control compensates for the overshoots of the **I**ntegral control.

- **I**ntegral control compensates for the undershoots of the **P**roportional control.

- The combination of the three (not always needed) results in a smooth, efficient motion that draws exactly as much current as it needs to. This is called **PID Control!**


Setpoint
+
−
Σ
Error
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau) d\tau$
D
$K_d \frac{de(t)}{dt}$
Σ
Process
Output

PID Example 2

Want to drive a robot forward 10 metres, and then stop.

P — Proportional Control

- A measure of how far you are away from your goal.

P — Proportional Control

- A measure of how far you are away from your goal.

- The larger the P value, the larger the motor output should be, since you have farther to go.

P — Proportional Control

- A measure of how far you are away from your goal.

- The larger the P value, the larger the motor output should be, since you have farther to go.

- In this example, measure the distance remaining between our current position and our goal of 10 metres.

D — Derivative Control

- How quickly you are moving towards your goal.

D — Derivative Control

- How quickly you are moving towards your goal.
- Unlike P, when D is higher you want to "pull back" your motor output. This keeps you from heading towards you target too quickly.

D — Derivative Control

- How quickly you are moving towards your goal.
- Unlike P, when D is higher you want to "pull back" your motor output. This keeps you from heading towards you target too quickly.
- If P is still large (ie you are far from your target) then it will "overpower" the D.

D — Derivative Control

- How quickly you are moving towards your goal.
- Unlike P, when D is higher you want to "pull back" your motor output. This keeps you from heading towards you target too quickly.
- If P is still large (ie you are far from your target) then it will "overpower" the D.
- In this example, measure be the speed we are traveling towards the goal.

I — Integral Control

- The integral is how long you have been away from your target.

I — Integral Control

- The integral is how long you have been away from your target.
- The idea is that the longer the robot has not made it to the target, the more power should be applied to get it there.

I — Integral Control

- The integral is how long you have been away from your target.
- The idea is that the longer the robot has not made it to the target, the more power should be applied to get it there.
- The most difficult part to incorporate. PD control alone can be very effective.

I — Integral Control

- The integral is how long you have been away from your target.
- The idea is that the longer the robot has not made it to the target, the more power should be applied to get it there.
- The most difficult part to incorporate. PD control alone can be very effective.
- I is the sum of the distances you are away each time. It can be calculated by adding up the P value from each loop.

Calculating the Motor Output

- The basic formula is: output = P * Kp + I * Ki - D * Kd

Calculating the Motor Output

- The basic formula is: output = P * Kp + I * Ki - D * Kd
- Where Kp, Ki, and Kd are values tweaked to get the proper result.

Calculating the Motor Output

- The basic formula is: output = P * Kp + I * Ki - D * Kd
- Where Kp, Ki, and Kd are values tweaked to get the proper result.
- There are some methods to calculate the values of Kp, Ki, and Kd, but it is generally more effective to find them by trial and error in robotics.

Method for Setting Values

- Start with Kp small and Ki, Kd both zero.

Method for Setting Values

- Start with Kp small and Ki, Kd both zero.
- Raise Kp until the robot is oscillating consistently around the target.

Method for Setting Values

- Start with Kp small and Ki, Kd both zero.
- Raise Kp until the robot is oscillating consistently around the target.
- Once this is accomplished, start increasing Kd until the robot stops oscillating.

Method for Setting Values

- Start with Kp small and Ki, Kd both zero.
- Raise Kp until the robot is oscillating consistently around the target.
- Once this is accomplished, start increasing Kd until the robot stops oscillating.
- Then add Ki until the robot stops within a desired range of the target.

P Controllers

- P controllers (K/ simple gain controllers) are the simplest closed-loop negative feedback controller.

P Controllers

- P controllers (K/ simple gain controllers) are the simplest closed-loop negative feedback controller.
- A K controller simply amplifies the error between the system's current position and the desired position and passes this as the input to the system.

P Controllers

- P controllers (K/ simple gain controllers) are the simplest closed-loop negative feedback controller.
- A K controller simply amplifies the error between the system's current position and the desired position and passes this as the input to the system.
- Examples of K controllers are RC servos.

P Controllers

- P controllers (K/ simple gain controllers) are the simplest closed-loop negative feedback controller.
- A K controller simply amplifies the error between the system's current position and the desired position and passes this as the input to the system.
- Examples of K controllers are RC servos.
- The K controller is useful for slow processes, i.e. systems that must change slowly. As the controller speeds up, the controller tends to overshoot the setpoint.

P Controllers

- P controllers (K/ simple gain controllers) are the simplest closed-loop negative feedback controller.
- A K controller simply amplifies the error between the system's current position and the desired position and passes this as the input to the system.
- Examples of K controllers are RC servos.
- The K controller is useful for slow processes, i.e. systems that must change slowly. As the controller speeds up, the controller tends to overshoot the setpoint.
- Additionally, K controllers are susceptible to offset, that is, the controller will approach the setpoint but will stop short due to the amount of control input needed to maintain a particular process state.

How to think of the PID Code

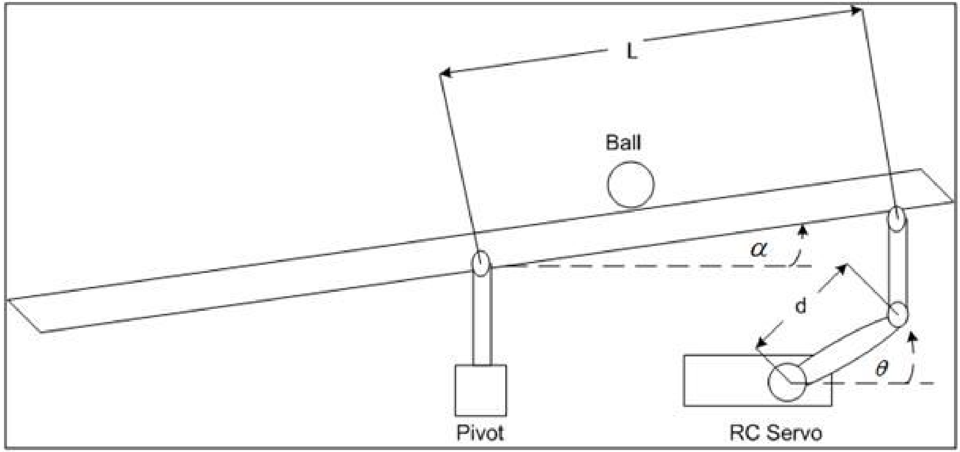
```
previous_error = 0
integral = 0
start:
  error = setpoint – actual_position
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto start
```

PID Code Lab

- Take an example of a car of speed V
- When you step on the accelerator, your car moves slowly, then faster, and faster still, until you let off the gas pedal.
- Write a general PID code in C to control the car's motion

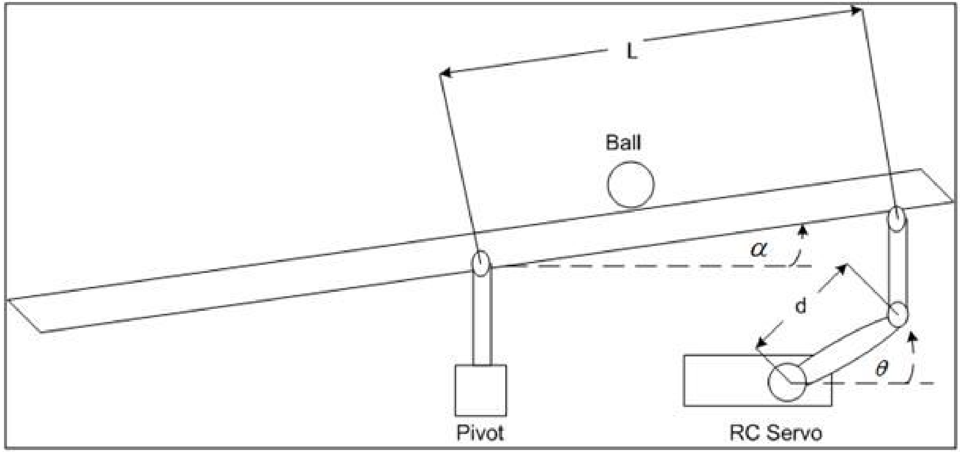
Balancing Ball on Beam Lab

- The system includes a ball, a beam, a motor and touch screen sensor.



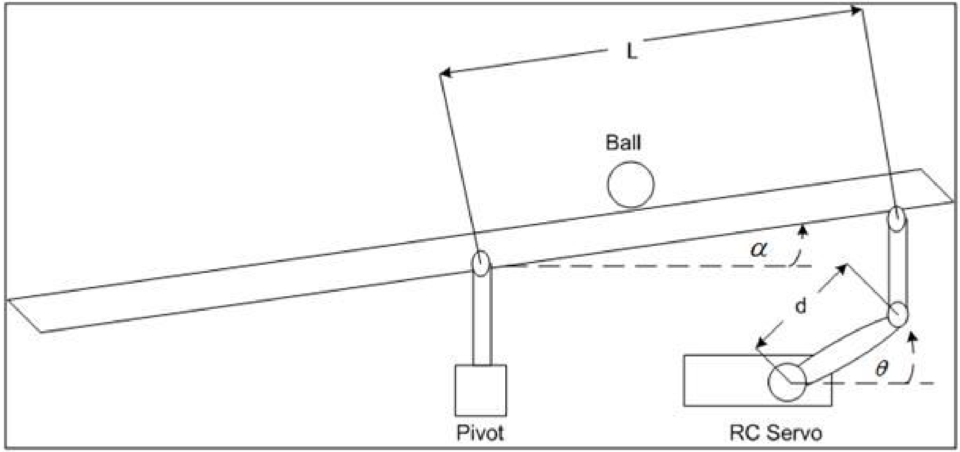
Balancing Ball on Beam Lab

- The system includes a ball, a beam, a motor and touch screen sensor.
- The basic idea is to use the torque generated from motor to the control the position of the ball on the beam.



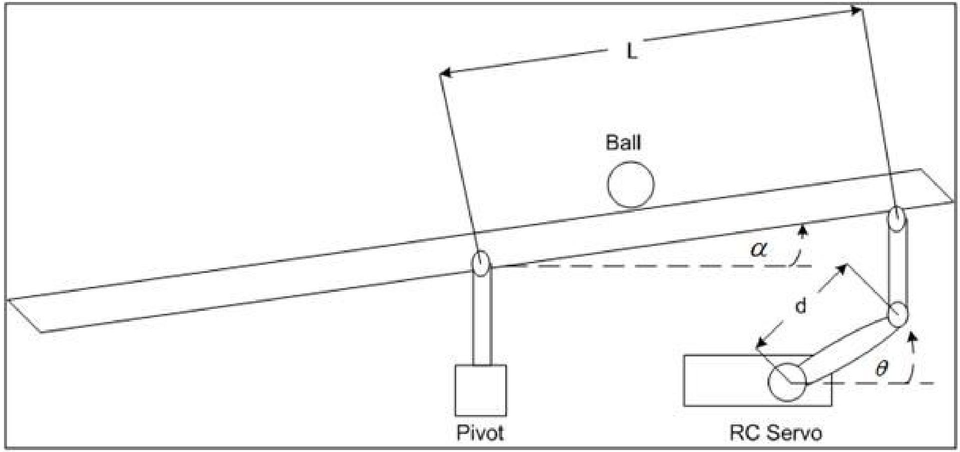
Balancing Ball on Beam Lab

- The system includes a ball, a beam, a motor and touch screen sensor.
- The basic idea is to use the torque generated from motor to the control the position of the ball on the beam.
- The ball rolls on the beam freely.



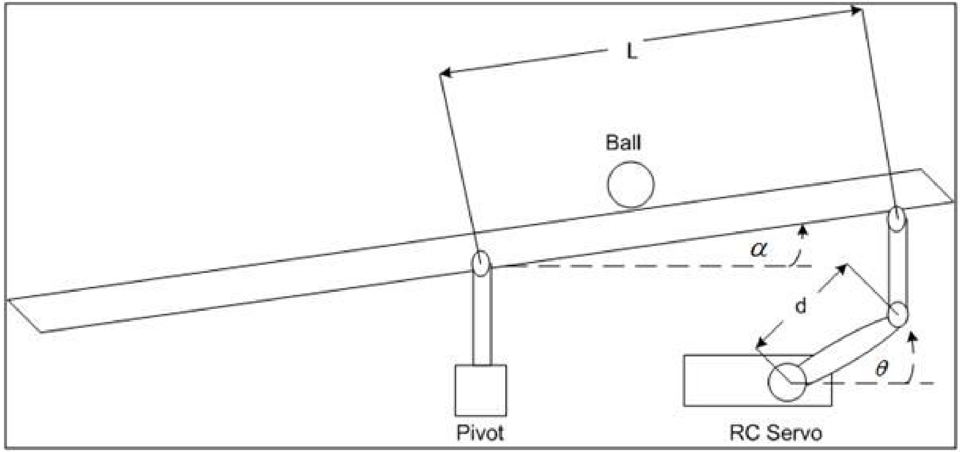
Balancing Ball on Beam Lab

- The system includes a ball, a beam, a motor and touch screen sensor.
- The basic idea is to use the torque generated from motor to the control the position of the ball on the beam.
- The ball rolls on the beam freely.
- The information from the sensor can be taken and compared with desired positions values.



Balancing Ball on Beam Lab

- The system includes a ball, a beam, a motor and touch screen sensor.
- The basic idea is to use the torque generated from motor to the control the position of the ball on the beam.
- The ball rolls on the beam freely.
- The information from the sensor can be taken and compared with desired positions values.
- The difference can be fed back into the controller, and then into the motor in order to gain the desired position.



Balancing Ball on Beam Lab

- The ball rolling up and down the beam

Balancing Ball on Beam Lab

- The ball rolling up and down the beam

- The beam rotating through its central axis.

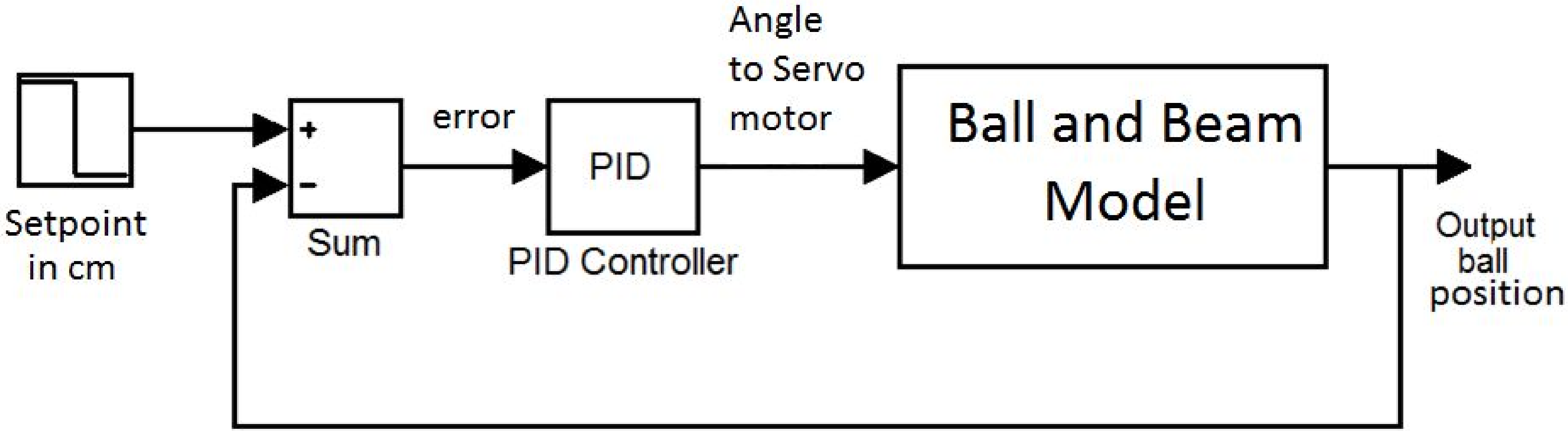
Balancing Ball on Beam Lab

- The ball rolling up and down the beam

- The beam rotating through its central axis.

- The aim of the system is to control the position of the ball to a desired reference point, and reject disturbances such as a push from a finger.

Balancing Ball on Beam Lab



Balancing Ball on Beam Lab

Processing Demo of Lab

Balancing Ball on Beam Lab

- Write an Arduino code to balance the ball on a beam.

Balancing Ball on Beam Lab

- Write an Arduino code to balance the ball on a beam.
- You can use the following Arduino libraries:

Balancing Ball on Beam Lab

- Write an Arduino code to balance the ball on a beam.
- You can use the following Arduino libraries:
  - Servo library `<Servo.h>`

Balancing Ball on Beam Lab

- Write an Arduino code to balance the ball on a beam.
- You can use the following Arduino libraries:
  - Servo library <`Servo.h`>
  - Touchscreen library "TouchScreen.h"
    - TouchScreen ts = TouchScreen(XP, YP, XM, YM, 711);
    - TSPoint p = ts.getPoint();
    - x -coordinate is(p.x); y -coordinate is (p.y);

Balancing Ball on Beam Lab

- Write an Arduino code to balance the ball on a beam.
- You can use the following Arduino libraries:
  - Servo library `<Servo.h>`
  - Touchscreen library "TouchScreen.h"
    - TouchScreen ts = TouchScreen(XP, YP, XM, YM, 711);
    - TSPoint p = ts.getPoint();
    - x -coordinate is(p.x); y -coordinate is (p.y);
  - PID library `<PID_v1.h>`

PID library <PID_v1.h>

PID()

Creates a PID controller linked to the specified Input, Output, and Setpoint.

PID library <PID_v1.h>

PID()

Creates a PID controller linked to the specified Input, Output, and Setpoint.

Syntax

PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction)

For reference, in a car, the Input, Setpoint, and Output would be the speed, desired speed, and gas pedal angle respectively.

PID library <PID_v1.h>

PID()

Creates a PID controller linked to the specified Input, Output, and Setpoint.

Syntax

PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction)

For reference, in a car, the Input, Setpoint, and Output would be the speed, desired speed, and gas pedal angle respectively.

Parameters

Input: The variable we're trying to control (double)
Output: The variable that will be adjusted by the pid (double)
Setpoint: The value we want to Input to maintain (double)
Direction: Direction the output will move when faced with a given error. DIRECT is most common.

Compute()

Description

Contains the pid algorithm. it should be called once every loop(). Most of the time it will just return without doing anything. At a frequency specified by SetSampleTime it will calculate a new Output.

Compute()

Description

Contains the pid algorithm. it should be called once every loop(). Most of the time it will just return without doing anything. At a frequency specified by SetSampleTime it will calculate a new Output.

Syntax

Compute()

Compute()

Description

Contains the pid algorithm. it should be called once every loop(). Most of the time it will just return without doing anything. At a frequency specified by SetSampleTime it will calculate a new Output.

Syntax

Compute()

Parameters

None

Returns

True: when the output is computed
False: when nothing has been done

SetMode()

Description

Specifies whether the PID should be on (Automatic) or off (Manual.) The PID defaults to the off position when created.

SetMode()

Description

Specifies whether the PID should be on (Automatic) or off (Manual.) The PID defaults to the off position when created.

Syntax

```
SetMode(mode)
```

SetMode()

Description

Specifies whether the PID should be on (Automatic) or off (Manual.) The PID defaults to the off position when created.

Syntax

SetMode(mode)

Parameters

mode: AUTOMATIC or MANUAL

SetOutputLimits()

Description

The PID controller is designed to vary its output within a given range.

Syntax

```
SetOutputLimits(min, max)
```

Balancing Ball on Beam Lab Code Steps

- Include the libraries you're using
- Define servo pin, Kp, Ki, Kd, Setpoint, Input, Output, and ServoOutput
- Initialize your PID object and your Servo
- Start the `setup()`
- After you finish your `setup()`, move to the `loop()`