WEIGHT BALANCING SYSTEM USING ARDUINO

Components:

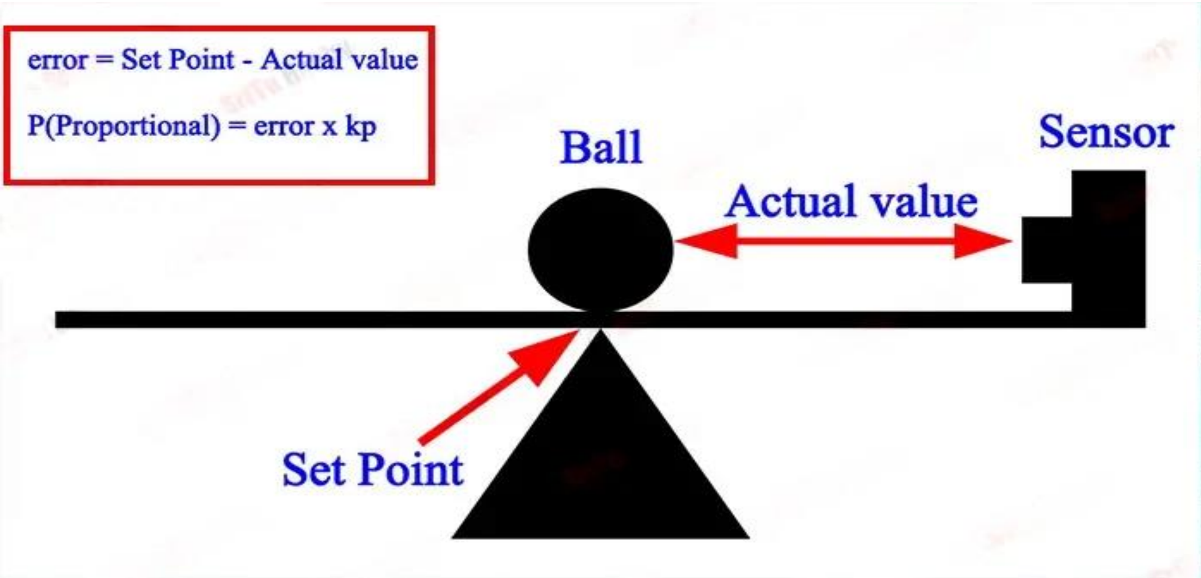
- Arduino UNO board
- Ultrasonic sensor
- Servo motor
- Foam board
- Card board
- Jumper wire
- Iron stick
- Ball

PID (Proportional Integral Derivative)

- **P (Proportional)** — This Proportional controller gives a proportional output to the error. For that, the error must be multiplied by a constant called kp. The correct value should be found by entering different values for the Kp constant. We can get the error by subtracting the actual value from the set point. Through this, we can stabilize the system to some extent.

- **I (Integral)** — The steady-state errors can be detected through this controller. For that, the error must be integrated. Then, must multiply that value by a constant called ki. This correct ki value should be found using different values.

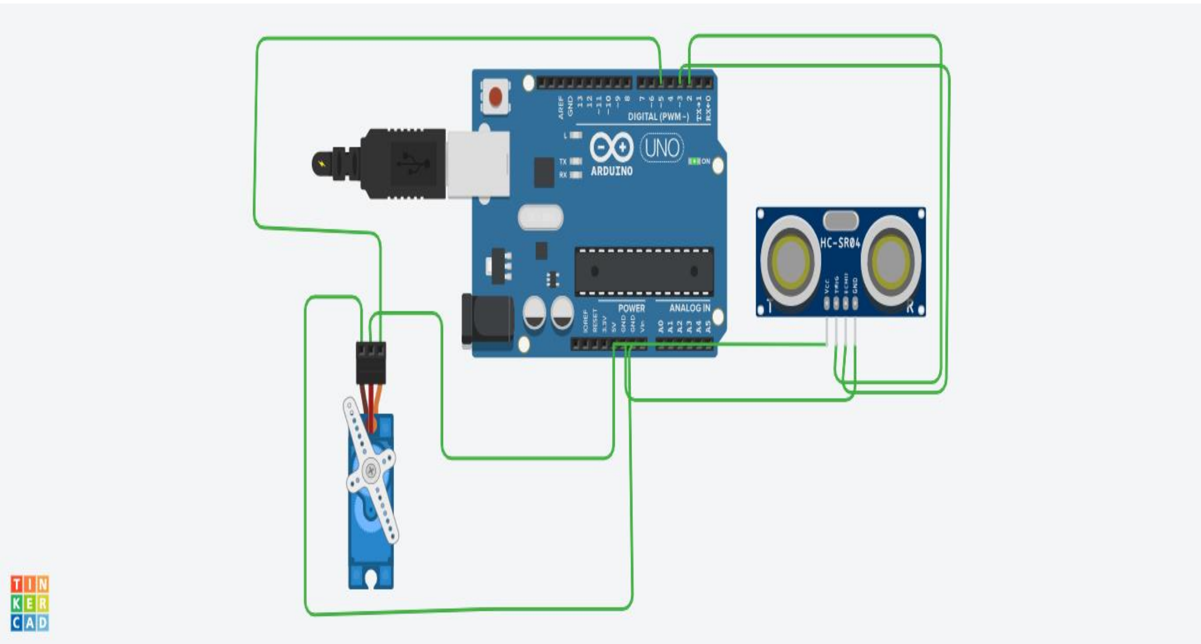
- **D (Derivative)** — This derivative controller eliminates the problem of overshoot caused by the proportional controller. For that, the error must be differentiated. Then, that value must be multiplied by a constant kd. This correct kd value should be found using different values



$$\text{error} = \text{Set Point} - \text{Actual value}$$

$$\text{PID} = \text{error} \times kp + ki \times \int (\text{error})dt + kd \times \frac{d(\text{error})}{dt}$$

CIRCUIT DIAGRAM



Tuning Process

1. **Upload the Code**:
    - Upload the code to your Arduino and open the Serial Monitor.
2. **Observe the Behavior**:
    - Observe how the servo motor responds to changes in the distance.
    - Note if the servo oscillates, reaches the setpoint, or takes too long to stabilize.
3. **Adjust `kp`**:
    - If the servo is too slow, increase `kp`.
    - If the servo oscillates too much, decrease `kp`.
4. **Adjust `ki`**:
    - If there is a steady-state error (the servo doesn't quite reach the setpoint), increase `ki`.
    - If the system becomes too oscillatory, decrease `ki`.
5. **Adjust `kd`**:
    - If the system oscillates and you want to dampen the response, increase `kd`.
    - Be cautious as too high a `kd` can lead to excessive damping or noise sensitivity.

Iterative Tuning

Continue to iteratively adjust `kp`, `ki`, and `kd` based on the observed behavior until the system responds as desired. Here's a summary of the expected changes with each parameter adjustment:

- **Increase `kp`**: Faster response but more oscillation.
- **Increase `ki`**: Reduces steady-state error but can cause more oscillation.
- **Increase `kd`**: Reduces oscillation and improves stability but can slow down the response.

Final Tuning

Once you find a set of `kp`, `ki`, and `kd` values that provide a stable and accurate response, you can finalize your PID controller settings. The values might need to be fine-tuned over several iterations to achieve the best performance.

Length of the balance is 10 inch

CODE:

```
#include <Servo.h>
Servo servo;

#define trig 2
#define echo 3

#define kp 0
#define ki 0
#define kd 0

double priError = 0;
double toError = 0;

void setup() {
  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);
  servo.attach(5);
  Serial.begin(9600);
  servo.write(50);

}
void loop() {
  PID();
  //  int a = distance();
  //  Serial.println(a);
}

long distance () {
  digitalWrite(trig, LOW);
  delayMicroseconds(4);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  long t = pulseIn(echo, HIGH);
  long cm = t / 29 / 2;
  return cm;
}

void PID() {
  int dis = distance ();


  int setP = 15;
  double error = setP - dis;
```

```
  double Pvalue = error * kp;
  double Ivalue = toError * ki;
  double Dvalue = (error - priError) * kd;

  double PIDvalue = Pvalue + Ivalue + Dvalue;
  priError = error;
  toError += error;
  Serial.println(PIDvalue);
  int Fvalue = (int)PIDvalue;


  Fvalue = map(Fvalue, -135, 135, 135, 0);


  if (Fvalue < 0) {
    Fvalue = 0;
  }
  if (Fvalue > 135) {
    Fvalue = 135;
  }

  servo.write(Fvalue);
}
```

HARDWARE SETUP: