How to create a PID DC motor position controller

# Introduction

Control Engineers are tasked with creating a way of controlling systems, whether its mechanical, chemical, electrical, etc. This task is achieved by creating controllers that decide how to influence a system to achieve the desired outcome. One of the simples examples of controllers are PID controllers such as the one in the cruise control feature of modern day cars. The cruise control takes in a desired speed (reference value) and the actual speed (feedback) to accelerate the car to reach the desired speed. Just like this, there are many more applications for PID controllers in everything from airplanes to animatronics and even thermostats in your house. By the end of this instructional guide, you will have a basic understanding of what a PID controller is and have created a position controller for a DC motor, just like a controls engineers would use for a robotics application.

**Throughout this guide you will:**

* Learn the basic theory behind PID controllers
* Build a demonstration motor and feedback assembly
* Program a PID controller for the assembly

**Completion time:** 20 to 45 minutes.

**Minimum skills required:**

* Basic soldering skills
* Familiarity with programming Arduino microcontrollers
* Past 3D printing experience
* A conceptual understanding of integration and differentiation

|  |  |
| --- | --- |
| **Materials:** | **Tools:** |
| 1 – Arduino Nano (microcontroller)  4 – P160KNP-0FC25B5K (Linear Potentiometers)  1 – MD5-2445 (12V DC motor)  1 – Pololu Jrk 21v3 (motor driver)  1 – YB1203000-USB (Power Bank)  1—20 AWG Wire (Black, Red, Yellow, Blue, Orange)  1 – 3D printed model  1 – Coaxial female connector | 1 – Soldering Iron  1 – Rosin Core Solder  1 – Wire Strippers  1 – Computer with Arduino IDE  1 – Mini USB cable |

Contents

[Introduction 1](#_Toc3437932)

[Dangers, Warnings, and Cautions 4](#_Toc3437933)

[Dangers Involved with these instructions 4](#_Toc3437934)

[Warning Involved with these instructions 4](#_Toc3437935)

[Caution Involved with these instructions 4](#_Toc3437936)

[Theory 5](#_Toc3437937)

[1. Mechanical Assembly 7](#_Toc3437938)

[1.1. Identify the components needed for the mechanical assembly 7](#_Toc3437939)

[1.2. Assemble the motor-feedback subassembly 7](#_Toc3437940)

[1.3. Assemble the base-dials subassembly 8](#_Toc3437941)

[1.4. Final Mechanical Assembly 8](#_Toc3437942)

[2. Electrical Assembly 9](#_Toc3437943)

[2.1. Identify the materials and tools required for the electrical assembly 9](#_Toc3437944)

[1.1. Solder power and signal wires to the potentiometers 10](#_Toc3437945)

[1.2. Solder power wires to the motor 10](#_Toc3437946)

[1.3. Connect the motor to the motor controller 10](#_Toc3437947)

[1.4. Solder the potentiometers to the Arduino 11](#_Toc3437948)

[1.5. Solder the Arduino and motor controller 11](#_Toc3437949)

[1.6. Solder the coaxial connector to the motor controller 11](#_Toc3437950)

[1.7. Connect battery pack to the coaxial connector, powering the system 11](#_Toc3437951)

[2. Code 12](#_Toc3437952)

[2.1. Open the Arduino IDE an create a New Sketch 12](#_Toc3437953)

[3.2 Initialize all the constants required 12](#_Toc3437954)

[3.3 Initialize all the variables required 12](#_Toc3437955)

[3.4 Measure the value of the gains 13](#_Toc3437956)

[3.5 Measure the Error 13](#_Toc3437957)

[3.6 Calculate the PID input 13](#_Toc3437958)

[3.7 Actuate the motors 14](#_Toc3437959)

[3.8 Display the data 14](#_Toc3437960)

[3.9 Set pins and start communication 14](#_Toc3437961)

[3.10 Create the main loop 14](#_Toc3437962)

[3.11 Upload code to the Arduino Nano 14](#_Toc3437963)

[4. Tuning 15](#_Toc3437964)

[4.1 Setting Kp Value 15](#_Toc3437965)

[4.2 Setting Ki Value 16](#_Toc3437966)

[4.3 Setting Kd Value 16](#_Toc3437967)

[5. Conclusion 17](#_Toc3437968)

[6. Attachments 18](#_Toc3437969)

[Attachment 1: Wiring Diagram 18](#_Toc3437970)

[Attachment 2: Transcribed Code 19](#_Toc3437971)

[7. Cited Images 21](#_Toc3437972)

# Dangers, Warnings, and Cautions

These instructions can be a great way to gain exposure in the field of controls engineering, but it must be done safely. Below there is a list of dangers, warnings, and cautions. Dangers are those that could cause serious injury or even death. Warnings are those that may cause injury if done improperly and cautions are those that can damage the final product or equipment. Read the followings in detail to be aware of the risks involved with these instructions. In addition to this warning, there will reminders on the steps that are associated with each of these categories to remind you of the risks. Every danger will be written in red text, every warning will be in red-orange text and every caution will be written in orange text.

## Dangers Involved with these instructions

|  |  |
| --- | --- |
| Situation | Precaution |
| * Unattended soldering irons are a fire hazard | * Turn off or disconnect soldering iron when not in use |
| * Stripped wires powered by mains voltage are an electric shock hazard | * Inspect all connections and wires before powering any mains voltage device |

## Warning Involved with these instructions

|  |  |
| --- | --- |
| Situation | Precaution |
| * The flux in the solder can boil and expel hot solder | * Wear safety glasses when soldering |
| * The soldering iron is hot and a burning hazard | * Be careful when soldering and hold the soldering iron solely by the handle |
| * Short circuits can cause elements to become a burning hazard | * Before powering a device make sure all connections are properly made |

## Caution Involved with these instructions

|  |  |
| --- | --- |
| Situation | Precaution |
| * Parts from the 3d printer are weaker perpendicular to layer lines and stresses can break them | * Do not force parts to fit. If they do not friction fit, use some sandpaper to loosen the tolerances and retry. |
| * Mistakes in the code can cause parts to rotate in the wrong direction and break | * Ensure that the electrical connections and the code are in unison |

# Theory

A screenshot of a cell phone

Description generated with very high confidence

Figure 1: Block Diagram of a PID Controller

Consider this problem: you are tasked with finding a way of regulating the temperature inside your house and keeping it at the desired temperature. During spring and winter, you want to keep the house warm while during summer and fall you want to keep the house cool. Where would you start?

This is a very similar problem to what controls engineers face when they are tasked with controlling a process. A process is defined as any physical operation that we are interested in controlling. The process converts a given input into an output, which is governed by the physics of the process, also referred to as “the system”. An example of this would be the relationship between the accelerator and the speed of a vehicle. As the accelerator is pressed down, the speed increases proportionally to the input. To control a process, engineers use a controller to provide the appropriate inputs for the process to ensure that the output of the process is the desired output.

One of the simplest models for a controller is what is known as a PID controller. PID is an acronym for Proportional Integral Derivative controller. This controller works by finding the error between the desired output, also referred to as the “state of a system”, and the actual output. This error in the system is then transformed by four mathematical operations to produce an appropriate input to the process. Going back to the heating example, if the temperature in your house is 70F yet the desired temperature is 75F, there is a 5F error. The error can also be negative, and it simply means that the desired temperature is less than the actual temperature. A simpler way of thinking about it is that the error is a measure of how far and in what direction the actual measurement must move to equal the reference.

The first of the three operations is a simple multiplication. The error in the system is multiplied by a constant called the proportional constant, Kp. This proportional constant takes the state error and converts it to an input to the process. The error in temperature in this example was calculated to be 5F. This error would be multiplied by the constant Kp to give the expression 5\*Kp.

The second operation is taking a time derivative of the error. A time derivative is the rate of change of the error with respect to time, or how quickly is the error changing. Suppose that for the heating example the temperature at 6 am inside of the house is 60F and at noon is 65F. The error at 6 am will be (75-60) F or 15F while the error at noon will be (75-65) F or 10F. This means that the change in the error was (10-15) F or -5 F while the change in time was (12 pm – 7 am) or 5 hrs. The time derivative of the error will then be -5/5 [F/hr.] or -1 [F/hr.]. This time derivative is then multiplied by a constant Kd to give the expression -Kd [F/hr.].

The third operation is a time integral of the error. A time integral is a combination of a sum and a multiplication in which the error at any point in time is multiplied by a very small increment of time and then added to the previous multiplication of error and time increment. The integral measures how long the system has been at a given state. In our example, we can measure the temperature every second and multiply the measurement by a second to then add it to the previous sum. For this example, let’s assume that the calculated integral has a value of 2 [F\*hr.]. This result is then multiplied by a constant Ki to give the expression 2Ki [F\*hr.].

After the three operations are performed on the error measurement, the three resulting expressions are added together to yield the following expression: (5Kp - 1Kd + 2Ki). The final expression will be the input to the process which will minimize the error in the system and ultimately reach the reference value, but before this can happen the variables Kp, Ki, and Kd must be set.

With this crash course in PID controller design, you will be able to properly tune the values for the PID controller and achieve a stable system with proper response time.

# Mechanical Assembly

## Identify the components needed for the mechanical assembly

* + 1. Identify the 3D printed components

|  |  |  |  |
| --- | --- | --- | --- |
| A close up of a device  Description generated with high confidence | A close up of a device  Description generated with very high confidence | A picture containing sky  Description generated with very high confidence | A close up of a device  Description generated with very high confidence |
| Figure 2: Base | Figure 3: Feedback Mount | Figure 4: Pointer | Figure 5: Motor Mount |

* + 1. Identify the purchased components

|  |  |
| --- | --- |
| A close up of a device  Description generated with high confidence | A close up of a white background  Description generated with high confidence |
| Figure 6: Linear Potentiometer | Figure 7: 12V DC motor |

## Assemble the motor-feedback subassembly

|  |  |
| --- | --- |
| * + 1. Press fit the motor into the motor mount     2. Press fit linear potentiometer into feedback mount     3. Press fit motor mount into the pointer     4. Press fit linear potentiometer into the pointer | Figure 8: motor-feedback subassembly exploded view |
| The motor-feedback subassembly will look like this once steps 1.2.1 – 1.2.4 are completed | Figure 9: motor-feedback subassembly |

## Assemble the base-dials subassembly

|  |  |
| --- | --- |
| * + 1. Press fit the linear potentiometers into the base | Figure 10: base-dials subassembly exploded view |
| The base-dials subassembly will look like figure 11 once step 1.3.1 is completed | Figure 11: base-dials subassembly |

## Final Mechanical Assembly

|  |  |
| --- | --- |
| * + 1. Press fit the motor-feedback subassembly onto the posts on the base | Figure 12: final mechanical assembly exploded view |
| The final assembly will look like figure 13 once step 1.4.1 is completed | Figure 13: final mechanical assembly |

# Electrical Assembly

## Identify the materials and tools required for the electrical assembly

* + 1. Identify the purchased components

|  |  |  |  |
| --- | --- | --- | --- |
| 12V 3000mAh DC Lithium Ion Battery Pack |  |  |  |
| Figure 14: Power Bank | Figure 15: 20 AWG Wire | Figure 16: Arduino Nano | Figure 17: Motor Driver |
|  |  |  |  |
|  | Figure 18: Rosin Core Solder | Figure 19: coaxial connector |  |

* + 1. Identify the tools

|  |  |
| --- | --- |
|  |  |
| Figure 20: Wire Strippers | Figure 21: Soldering Iron |

2.1. Solder power and signal wires to the potentiometers

|  |  |
| --- | --- |
| Solder red, blue and negative wires to the three linear potentiometers as shown in figure 22 | A picture containing object, mirror  Description generated with very high confidence  Figure 22: potentiometers with soldered wires |



## Solder power wires to the motor

|  |  |
| --- | --- |
| Solder red and black wires to the terminals on the back of the motor as shown in figure 23 | Figure 23: motor with soldered wires |

## Connect the motor to the motor controller

|  |  |
| --- | --- |
| Connect the motor wires A and B to the screw terminals A and B on the motor controller | Figure 24: wiring of the motor controller |

## Solder the potentiometers to the Arduino

|  |  |
| --- | --- |
| Solder the signal wires (blue) to the analog pins of the Arduino   * A0 = Kp potentiometer * A1 = Ki potentiometer * A2 = Kd potentiometer * A3 = Feedback potentiometer | Figure 25: Wiring of the potentiometers to the Arduino |

## Solder the Arduino and motor controller

|  |  |
| --- | --- |
| Solder pin D5 from the Arduino to the TX pin on the motor controller | Figure 26: wiring of the Arduino to the motor controller |

## Solder the coaxial connector to the motor controller

|  |  |
| --- | --- |
| * Center terminal of the coaxial connector to the 12V terminal on the motor controller * Outside terminal of the coaxial connector to the GND terminal on the motor controller * Reverse polarity will damage the board | Figure 27: wiring for the power bank and |

## Connect battery pack to the coaxial connector, powering the system

# Code

|  |  |
| --- | --- |
| Open the Arduino IDE an create a New Sketch |  |
| Initialize all the constants required The controller needs set pins to read and write information to. By defining the pins beforehand, the variable name will take the place of its value and make the code more readable and easier to change if needed | int P\_pin = A0;  int I\_pin = A1;  int D\_pin = A2;  int Feedback = A3;  int freq = 10;  int Motor = 5; |
| * 1. Initialize all the variables required   For the controller to operate, there are several required measurements that change through the code. These measurements are the variables throughout the code.  The first set of measurements needed are the gains (Kp, Ki, Kd) along with the feedback (Position), reference (middle) and error (err). The reference (middle) is the average value between the leftmost and the rightmost position.  The P, I and D variables are the values of the input associated with the three branches of the controller.  The currentTime, elapsedTime, and previousTime are variables used to keep track of time and to calculate the D and I portion of the controller. The out variable is the final input from the PID controller.  The PWM variable is the duty cycle of the PWM wave based on the out variable and the PWM\_mapped variable is the PWM variable normalized and centered around 127.5, the 0 value for the motor controller. | float Kp;  float Ki;  float Kd;  float Position;  float err;  float left\_edge = 0;  float right\_edge = 900;  float middle = 0.5\*(left\_edge+right\_edge);  float P;  float I;  float D;  float currentTime;  float elapsedTime;  float previousTime;  float lastError;  float out;  int PWM; |
| Measure the value of the gains The gains function measures the value of the gain dials and normalizes it to be used in the rest of the code. | void gains()  {  Kp = map(analogRead(P\_pin),0,1024,0,5);  Ki = map(analogRead(I\_pin),0,1024,0,1);  Kd = map(analogRead(D\_pin),0,1024,0,5);  } |
| Measure the Error The error function finds the difference between the reference position (middle) and the actual position which is read through the analogRead function. This difference can be either positive or negative depending on what side the pointer is. | float error()  {  Position = analogRead(Feedback);  err = middle - Position;  return err;  } |
| Calculate the PID input The variable currentTime stores the time since the code started while previousTime stores this same value after the PID calculation is done.  Elapsed time calculates how much time has passed since the last PID calculation.  The error function from the previous step is utilized to find the error in position  This error is then saved as lastError to later find the change in error  P calculation: The error is multiplied by the proportional gain  I calculation: The error is multiplied by the time interval between calculations and the integral constant  D calculation: The rate of change of the error is found by dividing the change in error by the change in time. This is then multiplied by the derivative constant  The final output of this function is the sum of the P, I and D calculations. | void PID()  {  currentTime = millis();  elapsedTime =(currentTime - previousTime);    err = error();  P = Kp \* err;  I += Ki \* err \* elapsedTime;  D = Kd \* (err - lastError)/elapsedTime;  out = P+I+D;    lastError = err;  previousTime = currentTime;  } |
| Actuate the motors The motor driver is activated through a PWM signal.  The output from the PID controller is normalized and centered around 127.5 to communicate clockwise and counterclockwise rotation to the motor driver. | void motor()  {  PWM = map(out,-1000,1000,0,255);  analogWrite(Motor,PWM);  } |
| Display the data The data function sends to the computer the value of the gains, error and PWM signal in a table for monitoring through the serial monitor. | void data()  {  Serial.print(Kp);  Serial.print("\t");  Serial.print(Ki);  Serial.print("\t");  Serial.print(Kd);  Serial.print("\t");  Serial.print(err);  Serial.print("\t");  Serial.print(PWM);  Serial.println("\t");  } |
| Set pins and start communication The setup runs once and continues through the code. In the setup, the pins for the gains and communication with the computer is initialized. | void setup()  {  pinMode(P\_pin,INPUT);  pinMode(I\_pin,INPUT);  pinMode(D\_pin,INPUT);  pinMode(Feedback,INPUT);  Serial.begin(9600);  } |
| Create the main loop The loop runs indefinitely through the functions that are in it. It calculates the gain, calculates the PID inputs, actuates the motor and publishes the data. These are the stages of the block diagram in functions | void loop()  {  gains();  PID();  motor();  data();  } |
| Upload code to the Arduino Nano |  |

# Tuning

PID controllers are highly sensitive to the gain values and drastically affect the response and stability of the system. The response of a system is how the system will approach the reference value. A good rule of thumb is striving to achieve a quick response without passing the reference, also called overshoot. The stability of the system can be thought of how well the system can stay at a given reference despite there being errors in measurements or external inputs. The desired response for this project would reach a vertical orientation quickly without overshooting. Likewise, the desired stability would be being able to push the pointer to either side and having it return to center. The next steps will guide you through the tuning process of the PID controller. Through this section, you will set new values for the different gains, test the system and iterate until the desired system response and stability is achieved.

## Setting Kp Value

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Figure 28: Non-ideal tuning for Kp | Figure 29: Better tuning for Kp | Figure 30: Adequate Kp tuning |

* + 1. Position the Kp dial to a non-zero value
    2. Test the value
    3. Increase or decrease the value based on the response
    4. Repeat steps 4.1.1. – 4.1.3. until the response behaves like figure 30

## Setting Ki Value

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Figure 31: Non-ideal tuning for Ki | Figure 32: Better tuning for Ki | Figure 33: Adequate Ki tuning |

* + 1. Position the Kd dial to a non-zero value
    2. Test the value
    3. Increase or decrease the value based on the response
    4. Repeat steps 4.2.1. – 4.2.3. until the response behaves like figure 30

## Setting Kd Value

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Figure 34: Non-ideal tuning for Kd | Figure 35: Better tuning for Kd | Figure 36: Adequate Kd tuning |

* + 1. Position the Kd dial to a non-zero value
    2. Test the value
    3. Increase or decrease the value based on the response
    4. Repeat steps 4.3.1. – 4.3.3. until the response behaves like figure 30

# Conclusion

Congratulations! If you are reading this, you have successfully created a PID DC motor position controller. The system response should be fast with little to no overshoot and robust enough to avoid measurement inconsistency and external inputs. This project can be further expanded by making the setpoints change as a function of time and observing how the tuned controller responds to changing a changing reference instead of a fixed reference.

If you are interested in learning more about control engineers and the work they do I encourage you to visit Brian Douglas’s YouTube channel, Control Systems Lectures, in which he explains what controls engineering is and provides a working understanding of the topic.

Link – youtube.com/controllectures

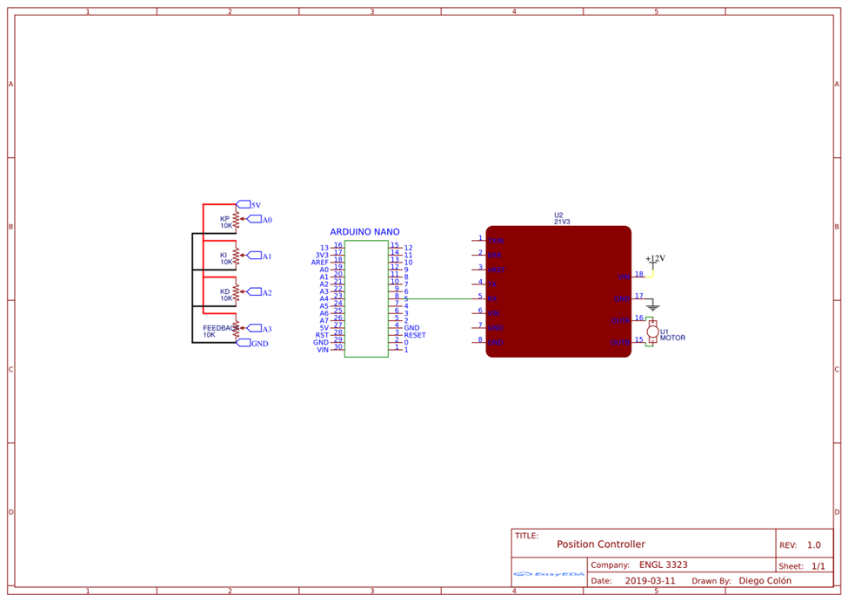
If you had difficulties with the programming of the Arduino, or want to learn more about them, visit the reference section of the Arduino website for detailed explanations of the syntax and the different functions used in this instructional guide.

Link - [www.arduino.cc/reference/en/](http://www.arduino.cc/reference/en/)

Hopefully this has been an interesting and learning experience and now have a better understanding of what a control engineer does.

# Attachments

## Attachment 1: Wiring Diagram



## Attachment 2: Transcribed Code

// Constants

int P\_pin = A0;

int I\_pin = A1;

int D\_pin = A2;

int Feedback = A3;

int Motor = 5;

// Variables

float Kp;

float Ki;

float Kd;

float Position;

float err;

float left\_edge = 0;

float right\_edge = 900;

float middle = 0.5\*(left\_edge+right\_edge);

float P;

float I;

float D;

float currentTime;

float elapsedTime;

float previousTime;

float lastError;

float out;

int PWM;

int PWM\_mapped;

// Functions

void gains(){

Kp = map(analogRead(P\_pin),0,1024,0,5);

Ki = map(analogRead(I\_pin),0,1024,0,1);

Kd = map(analogRead(D\_pin),0,1024,0,5);}

float error(){

Position = analogRead(Feedback);

err = middle - Position;

return err;

}

void PID(){

currentTime = millis();

elapsedTime =(currentTime - previousTime);

err = error();

P = Kp \* err;

I += Ki \* err \* elapsedTime;

D = Kd \* (err - lastError)/elapsedTime;

out = P+I+D;

lastError = err;

previousTime = currentTime;}

void motor(){

PWM = map(out,-1000,1000,0,255);

analogWrite(Motor,PWM);}

void data(){

Serial.print(Kp);

Serial.print("\t");

Serial.print(Ki);

Serial.print("\t");

Serial.print(Kd);

Serial.print("\t");

Serial.print(err);

Serial.print("\t");

Serial.print(PWM);

Serial.println("\t");}

// Code

void setup() {

pinMode(P\_pin,INPUT);

pinMode(I\_pin,INPUT);

pinMode(D\_pin,INPUT);

pinMode(Feedback,INPUT);

Serial.begin(9600);}

void loop() {

gains();

PID();

motor();

data();}

# Cited Images

Figure 14:

“Rechargeable 12V 3000mAh DC Lithium-Ion Battery Pack, 12V/5V Dual Output External Battery Power Bank.” *TalentCell*, www.talentcell.com/products/12v-battery/12v-3000mah-dc-lithium-ion-battery-pack.html.

Figure 15:

Holt, R. W. “20 AWG Colored Wire Spools.” *Amazon*, 21 Feb. 2019, www.amazon.com/dp/B07G2GLKMP/ref=sspa\_dk\_detail\_1?th=1.

Figure 16:

“Arduino Nano.” *Arduino*, store.arduino.cc/usa/arduino-nano.

Figure 17:

“Jrk 21v3.” *Pololu*, www.pololu.com/product/1392/resources.

Figure 18:

Inductiveload. “60-40 Solder Roll.” *Wikipedia*, 16 Sept. 2006, commons.wikimedia.org/wiki/File:60-40\_Solder.jpg.

Figure 19:

Meise, Martin. “Hohlstecker Und Hohlbuchse.” *Wikipedia*, 23 Dec. 2009, commons.wikimedia.org/wiki/File:Hohlstecker\_und\_Hohlbuchse\_5,5x2,5.jpg.

Figure 20:

HetheMeow. “Wire Strippers.” *Wikipedia*, 8 Nov. 2018, commons.wikimedia.org/wiki/File:Wire\_strippers\_5.jpg.

Figure 21:

Eugster, Simon A. “Weller Soldering Station.” *Wikipedia*, 20 Aug. 2009, commons.wikimedia.org/wiki/File:Soldering\_Station\_Weller\_2.jpeg.

Figure 24:

“Jrk 21v3.” *Pololu*, www.pololu.com/product/1392/resources.

Figure 28 – Figure 36:

Physicsch. “PID Compensation Animated.” *Wikipedia*, 28 May 2015, commons.wikimedia.org/wiki/File:PID\_Compensation\_Animated.gif.