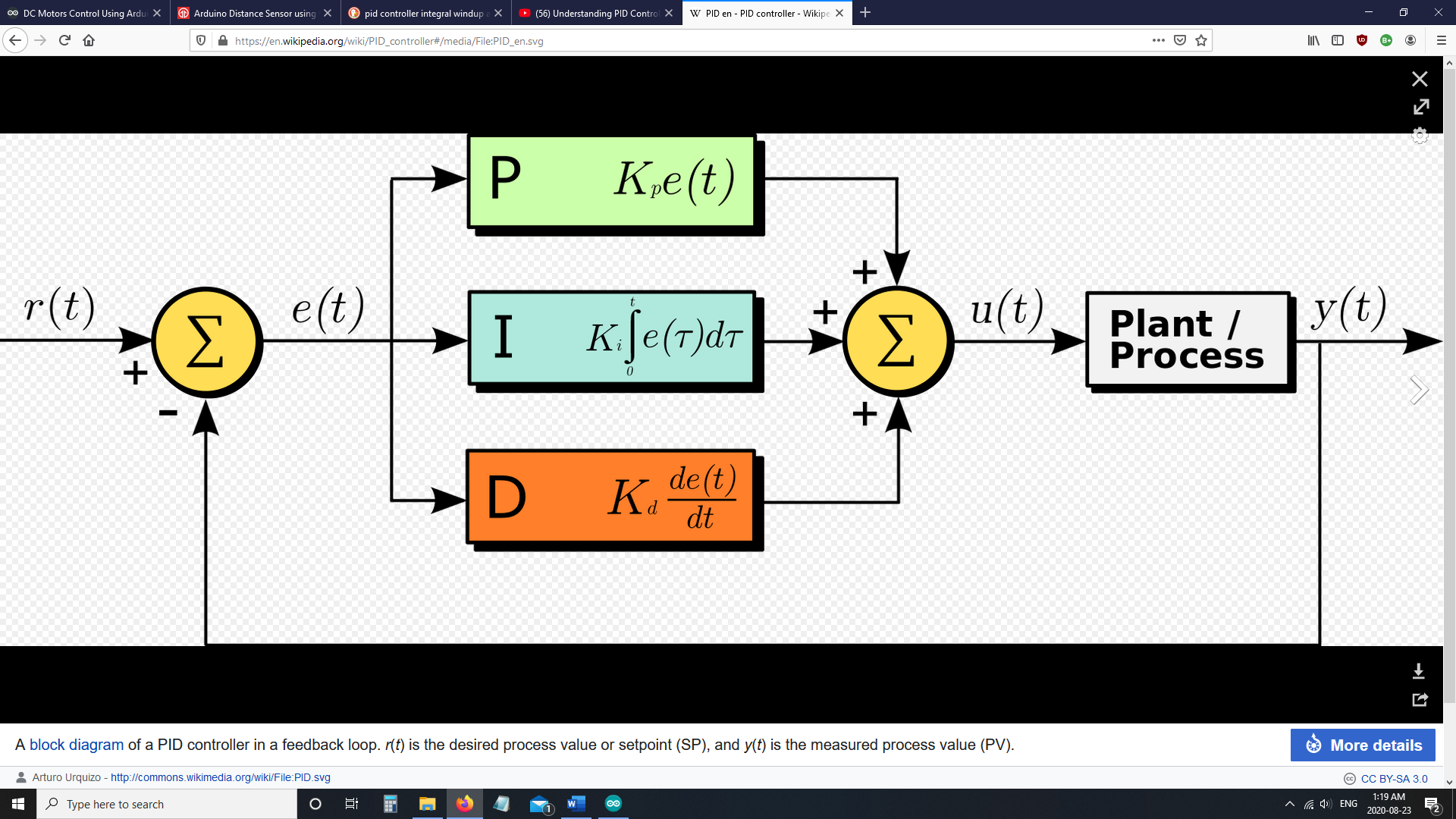
PID Controller

Description:

The goal of this project is to create a simple PID controller using an Arduino, ultrasonic sensor, and DC motor. A PID controller is a widely used control systems control loop. The P in PID controller stands for proportional which uses a feedback control signal that is linearly proportional to the error, the error being a set point minus the current output. In the diagram of the PID controller below, the error is the loop from y(t) that is subtracted from the set point r(t). The result is called proportional feedback. The I in PID stands for integral which is the feedback control signal that is linearly proportional to the integral of the error. The way the integral works is it “looks at the past” by taking the all the previous errors and sums them together. The D in PID stands for derivative. The goal of the derivative is to “predict the future” which dampens the overshoot from the integral. As the error is increasing the derivative increases and while the error decreases the derivative decreases. The output of the PID controller is calculated by adding the values of the proportional term, integral term, and derivative term, all times their respective weighting factors. The output of the added PID components, u(t), is called the control variable which adjusts the error over time by changing a component of the system (ex. changing how open a valve is, RPM of the motor, etc.), called the actuator. The process is the central component the PID controller which is to be controlled. The plant is a combination of the process and the actuator. The output of the plant/process, y(t), is the process variable which reacts to the change of the control variable. The goal of the process variable is to make it equal to the set point. Also, the proportional term, integral term, and derivative term are all multiplied by their own weighting factor (kp, ki, and kd) which are used to tune the controller to get a desired controller output. There are many ways to tune the controller, but popular methods include Ziegler-Nichols tuning method and automatic tuning by a software program like MATLAB.

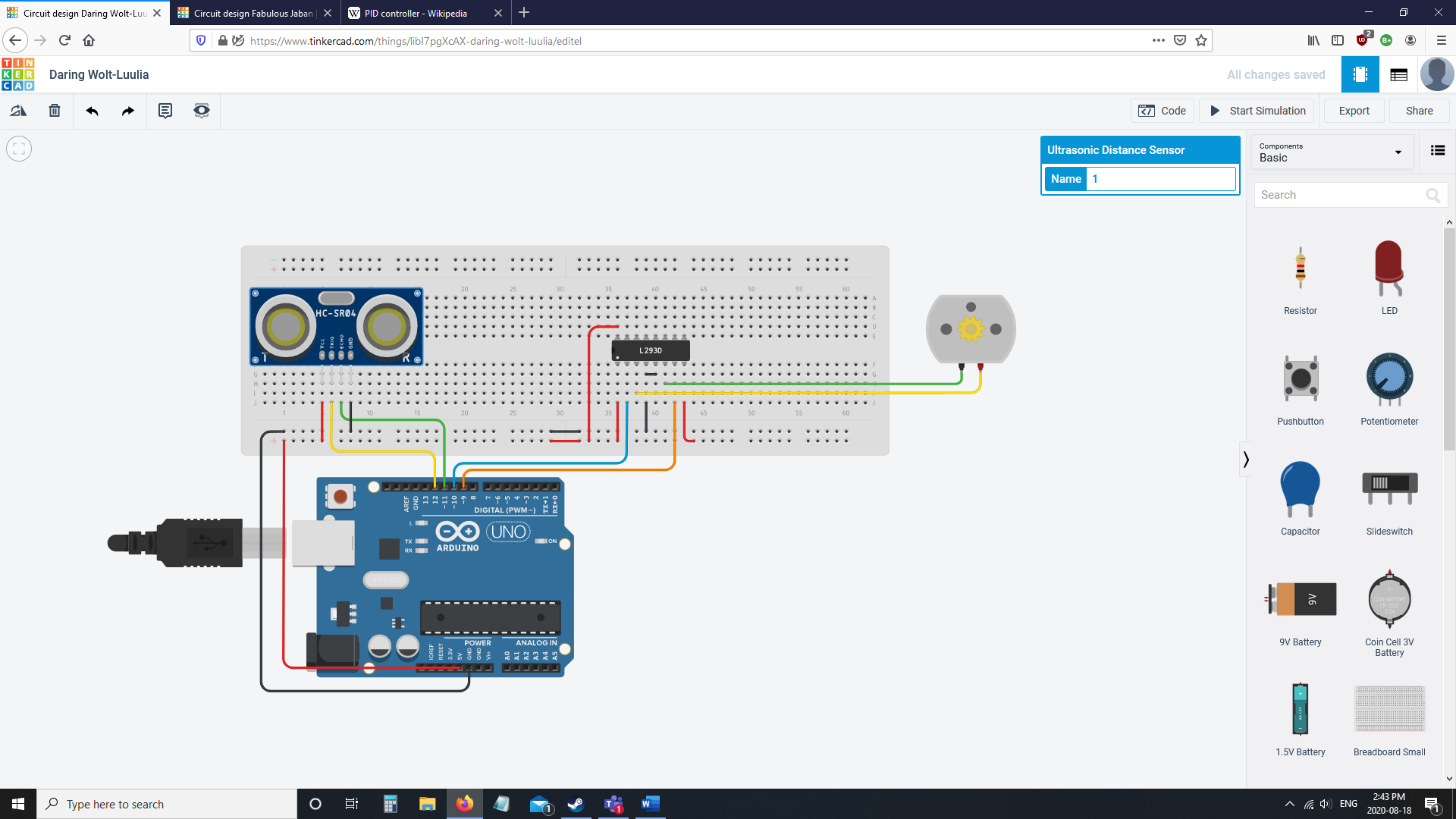
For this project you will be making a PI controller (although the derivative calculations are kept in the code to show how they would be implemented) to control motor speed to reach the ideal distance from the ultrasonic sensor. The derivative is not used in this project because the signal noise from the ultrasonic sensor is noisy, making the derivative useless. If you want to see this for yourself then send the derivative value to the serial monitor or serial plotter and you will see all changes in the derivative value will be neutralized by an equal change of the derivative in the opposite sign.

PID Controller Block Diagram:



[1]"PID controller", *En.wikipedia.org*, 2020

Wiring Diagram for Project:



The Arduino code is in the .ino file attached with this file. The serial monitor and serial plotter are set to show the PI controller output before limiting it but other outputs are the distance an object is from the ultrasonic sensor (in cm), the PWM value sent to the motor, the calculations for each part of the PI controller (including D to show it doesn’t work for this project), and the output after it is limited.

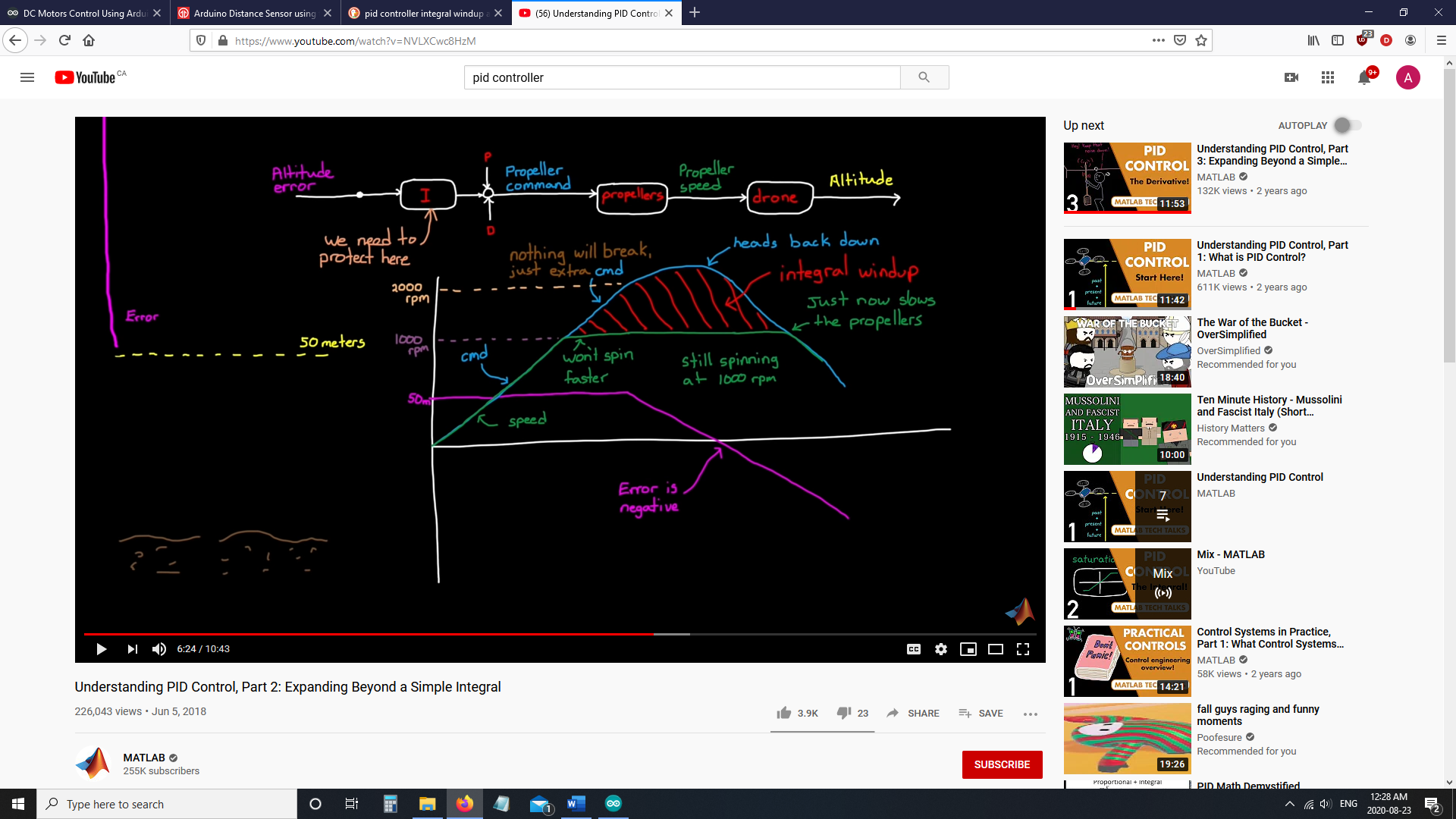
Note 1: Sending too many values to the serial monitor may have unforeseen consequences (ex. the ultrasonic sensor reading does not work) so be sure to limit the amount of data to be printed by the serial monitor.

Note 2: The code is currently set up to display only the output before saturation to the serial plotter. If you would like to add more variables for the serial plotter to display, please click [here](https://arduinogetstarted.com/tutorials/arduino-serial-plotter) to see how to do so.

The code:

|  |  |
| --- | --- |
|  | <math.h> will be used to include the absolute value function to provide leniency for the ultrasonic sensor. |
|  | These variables are used for the ultrasonic sensor’s distance input. The cm variable will be used for the PI controller’s input. |
|  | These variables are used to select the pins that will output to the DC motor and the value sent to the DC motor. |
|  | The “time” variables are used to calculate the integral and, in other projects, the derivative.  The error variable is the error and lastError is used for the derivative (the change in error from the last error over elapsedTime. The output variable is the output of the controller. The setPoint variable is the set point (in this project it is the ideal location away from the ultrasonic sensor). The cumError and rateError variables are the integral and derivative variables, respectively. |
|  | The posSaturation and negSaturation variables are the saturation values of the PID controller. A saturation value is the point the plant cannot follow the command it is given (in this case the value that is above the maximum PWM that can be sent). In this example there are positive and negative saturation points whose absolute values are equal to allow the DC motor to have an equal RPM in both directions (ex. positive values spin motor clockwise and vice versa). The posMax and negMax variables are the max PWM to the motor (values will be mapped later). |
|  | Setup for the ultrasonic sensor, serial output, and pins of motor output. The set point of the desired position is chosen with the setPoint variables in setup. |
|  | The first part of the loop() function is a do while loop that gets a value from the ultrasonic sensor. The parameters are choosing an acceptable distance reading for the ultrasonic sensor (it is set to 75cm in this example). You can adjust the parameters for acceptable distance readings to whatever you want but, at best, the ultrasonic sensor only reads values from 2cm to 400cm (assuming the one you are using is a high quality one). The Serial.print() is left commented in the code if you would like to see the distance of an object from the ultrasonic sensor (in cm). |
|  | Continuing on in the loop() function is the PID calculations and PWM output to the motor that is calculated in computePID() function. The input for the PI controller is the ultrasonic sensor reading. If the output variable is closer to the sensor than the set point is then the output is greater than 0 and the INA motor pin will be pulsed and vice versa. To change the output to the spectrum of the PWM pulses in analogWrite(), the [map()](https://www.arduino.cc/reference/en/language/functions/math/map/) function is used to map the value from 0 to posMax/negMax to 0 to 255 (the range of PWM pulses). |
|  | The return\_cm() function is used to get the ultrasonic sensor reading and make the cm variable equal the reading. The ultrasonic sensor has some signal noise which may make the reading a slightly different reading than is accurate. To reduce this problem an if statement is used to see if the reading is less than or equal to 1cm away from the set point. If the reading is less than or equal to 1cm away from the set point than the reading becomes equal to the set point. The reading of the ultrasonic sensor is not stored in a local variable and sent as a return value to the loop() because using the cm global variable makes the function faster, which will reduce the amount of overshoot in the PI controller (overshoot will be explained later). |
|  | The computePID() function computes the PI controller calculations. The inp parameter variable is the distance input from the ultrasonic sensor. The millis() function is used to determine the current amount of time the program has been running. The elapsedTime variable computes how much time it has been since the computePID() was last called, which will be used for the integral (and derivative if it was used). |
|  | The error variable is used for the error, calculating the P in PI. Next, the integral is calculated by adding error \* elaspedTime to all previous error \* elaspedTime values. In the code there are times when the integral doesn’t do and remains the same value. At these times the integral is saturating, meaning that it has reached the maximum value that the motor can be sent. The derivate is the rateError variable and is calculated using the change in error divided by the change in time. |
|  | Output is calculated using the error and cumError times their respective weighting factors (which are the k variables). The derivative kd variable is 0 which makes the derivative value not included in the controller. Serial.println() is then used to print the output of the PI controller. If the output is exceeding either of the saturation values than the output gets changed to the saturation value. Finally, the error and time are saved as the previous error and previous time and the output is returned. |

Integral windup: When a PID controller’s (or in this case a PI controller’s) output has exceeded the saturation value this is called integral windup. For this project, when integral windup occurs the output of the PI controller to the motor will be faster than motor’s max RPM, the integral will make the motor spin at max RPM even if the distance gets closer to the set point, and the motor will start to decrease it’s RPM once the integral has decreased to the motor’s max RPM (see the chart below).



[5]B. Douglas, "Understanding PID Control", *Youtube*, 2020

**References**

[1]"PID controller", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/PID\_controller#Loop\_tuning. [Accessed: 25- Aug- 2020].

[2]A. Ramamoorthy, "Understanding PID Loop Process Control", *Vertech.com*, 2020. [Online]. Available: https://www.vertech.com/blog/understanding-pid-loop-process-control-part-1. [Accessed: 25- Aug- 2020].

[3]"Arduino - Serial Plotter | Arduino Tutorial", *Arduino Getting Started*, 2020. [Online]. Available: https://arduinogetstarted.com/tutorials/arduino-serial-plotter. [Accessed: 25- Aug- 2020].

[4]"map() - Arduino Reference", *Arduino.cc*, 2020. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/math/map/. [Accessed: 25- Aug- 2020].

[5]B. Douglas, "Understanding PID Control", *Youtube*, 2020. [Online]. Available: https://www.youtube.com/watch?v=wkfEZmsQqiA&list=PLn8PRpmsu08pQBgjxYFXSsODEF3Jqmm-y. [Accessed: 25- Aug- 2020].

[6]A. Tutorial, "Arduino PID Control Tutorial - Microcontroller Tutorials", *Microcontroller Tutorials*, 2020. [Online]. Available: https://www.teachmemicro.com/arduino-pid-control-tutorial/. [Accessed: 25- Aug- 2020].

[7]"How to Use an Ultrasonic Sensor", *Arduino Project Hub*, 2020. [Online]. Available: https://create.arduino.cc/projecthub/MisterBotBreak/how-to-use-an-ultrasonic-sensor-181cee. [Accessed: 25- Aug- 2020].

[8]"How to Use the L293D Motor Driver - Arduino Tutorial", *Instructables.com*, 2020. [Online]. Available: https://www.instructables.com/id/How-to-use-the-L293D-Motor-Driver-Arduino-Tutorial/. [Accessed: 25- Aug- 2020].

[9]G. Franklin, J. Powell and A. Emami-Naeini, *Feedback control of dynamic systems*, 7th ed. Upper Saddle River, NJ: Pearson Higher Education Inc., 2015.