Digital Implementation of the PID Control Using Arduino

Final course Report of ME411 Mechatronics I by

PURVANG LAPSIWALA UIN 662689378 Illinois, Chicago

December 2017

Submitted to Dr. Sabri Cetinkunt

University of Illinois at Chicago

SUMMARY

This experiment aims to simulate an analog PID controller output using an Arduino microcontroller. For this purpose, we need to design a Simulink model for PID controller. An error signal will be passed as an input from function generator to an ADC channel of the microcontroller and the PWM output signal of this microcontroller is connected to the low pass filter in order to convert it into true analog signal. This analog signal then can be read by an oscilloscope. This experiment prominently elucidates applying the theory of PID controller utilizing an Arduino microcontroller and provides the basic understanding of Arduino's applications in variety of processes.

For the optional part, the PID control algorithm is written in C language in Arduino IDE as an interrupt service routine (ISR), and a timer to generate interrupt to execute the PID algorithm at a rate of 100 Hz. By performing this experiment, we were able to understand the digital implementation of PID Controller in Simulink and in Arduino IDE, effect of different gains on the controller, how does auto generate C code from Simulink works.

OBJECTIVE

The objective of the experiment was to understand the Arduino R3 Uno board along with its software and hardware interface circuits, learn the software development environment tool (IDE) for a microcontroller, understand the Simulink support package provided for Arduino and usage of different blocks in it like Analog input, PWM blocks, Data conversion block, design of PID control using Arduino microcontroller to test error input signal from function generator, developing the control algorithm in Simulink and C.

LIST OF COMPONENTS

Oscilloscope, Power supply, Function Generator, Resistor, Jumper wires, Breadboard, Capacitor, Arduino, Laptop.

The resistance value is found from the following table:

Color	Digit	Multiplier	Tolerance (%)
Black	0	10^0 (1)	
Brown	1	10^1	1
Red	2	10^2	2
Orange	3	10^3	
Yellow	4	10^4	
Green	5	10^5	0.5
Blue	6	10^6	0.25
Violet	7	10^7	0.1
Grey	8	10^8	
White	9	10^9	
Gold		10^{-1}	5
Silver		10^{-2}	10
(none)			20

THEORY

Our first object to simulate an analog PID using digital microcontroller.

A System is designed in order to produce desired output from the given inputs to that system. However, a plant which runs in real world always doesn't produce expected results because of different losses like due friction. This is where the concept of PID controllers contributes, to adjust the system parameters in order

to get the desired output in spite of any alterations or disturbances in the input to the system. Controller reduces the error in input signal to give a desired output and this is achieved by using proportional (P), derivative (D), integral (I) action.

Each action has its own gain value which are tuned to give the desired output.

- Proportional Controller (Kp): The output is error signal scaled with some value.
- Integral Controller (Ki): As time marches forward the error over time is continuously summed and scaled with some value to give desired output.
- Derivative Controller (Kd): This nothing but rate of change of the error signal.

There is no hard and fast rule saying that system must contain all the actions, everything mainly depends on the applications. Most common ones are PI, I, PID generally real-world systems contain a natural damping in them.

PID tuning is an important part of the discussion. Following are the methods for tuning a PID: Cohen-Coon Method (Open-loop Test), Ziegler-Nichols Method (Closed-loop P-Control Test), Tyreus-Luyben Method (Closed-loop P-Control test), and Autotune Method (Closed-loop On-Off test).

In this experiment we have followed a manual tuning method, which involves numerous attempts of trial and error.

ARDUINO

Arduino is open-source computer hardware and software community that designs and manufactures microcontroller-based kits for building digital devices and interactive objects that can sense and control objects in the physical world. In this Experiment we are using a specific Arduino Board named UNO R3.

Arduino has set of function libraries that help us in executing our desired process. Some functions and useful components of Arduino that are used in this project are explained in brief as follows:

Interrupts - Arduino board is essentially a microcontroller and it can handle many devices. In order to do so the device needs to communicate to the Arduino board. There are two methods for this purpose. Out of which we have used the method called Interrupts. Where, when a device needs the microcontroller service, the device notifies it by sending an interrupt signal to the microcontroller. When microcontroller receives the interrupt signal, it stops whatever it was doing and serves the device.

Timers -A timer is a built in hardware inside Arduino controller. It is just like a clock and can be used to measure time events. It can generate different types of interrupts order to control the controller operation. There are different timers depending on specific microprocessor.

For this project I have used a timer called Timer1 and the following functions:

- Timer1 - Timer 1 is a 16 bit timer. In the Arduino world the Servo library uses Timer1 on Arduino UNO.
- noInterrupts() - It basically prevents any other interrupts from interfering. Some functions will not work while interrupts are disabled, and incoming communication may be ignored.
- Interrupts() - It enables interrupts back again. Interrupts allow certain important tasks to happen in the background and are enabled by default.
- pinMode() - This function assigns a specific pin as an input or output port.

- Timer1.initialize() - This is called first before calling any other function. It repeats the code for every period specified. Its default period value is one second.
- Timer1.attachInterrupt() - It interrupts a function specified in parenthesis.
- analogRead() - This function takes analog reading from the pin specified in parenthesis.
- map() - It re-maps a number from one range to another. That is, a value of from input Low would get mapped to output Low, a value of from input High to output High.
- analogWrite() - It writes analog value i.e. PWM wave to a pin. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite()**.**

Many of the above functions can be seen in use when the Simulink auto generates the code and deploys to hardware.

The basic logic in of PID control in C is we take consider the input analog reading to be current error, proportional control will be direct scaling of it, for integral control we add previous all error with current error and scale it, for derivative control we subtract the current error and previous error and scale it, and finally add all the outputs to give the control signal and now previous error becomes the current error.

To develop PID function for Arduino, I have specified the following terms to write C code:

- Input variables: When the controller first turns all the specified input variables are set to zero initially.
- Sample Time- We are using timer library to generate interrupt to execute the PID algorithm at rate of 100Hz. This way the PID function get called every 10miliseconds.
- Map Input: The Arduino board contains 10bit-analog to digital convertor which maps the input voltage between 0 and 5 to 0 and 1023. The 'map' function maps this input analog value from 10-bits to 8-bits (0 to 255) analog output (PWM).
- Integral term: Integral term is defined in the way it sums up all the previous error. Anti-Wind up is added to prevent saturation of integrator, integral term was specified with output limits. Hence, when the integral

term adds to value greater than the maximum output the integral term was set to zero (0).

- Output saturation- To prevent saturation of output, output term was specified with output limits. Hence, when the output term goes beyond that limit output was set to maximum output (255).

SIMULINK

- In Simulink we build our model following the same logic used in C coding but added advantage we have is we can use the continuous blocks for P, I, D individual rather than implementing them in discrete from. One precaution we need to take is that the blocks we use for PID only accept data type double but the data we send as input is are 16bit integer so we need to use a data conversion block to change values to double then do our calculations on PID and then we write them over we give them as 8bit integer using another data conversion block.

- Also, we don't declare saturation of control signal as it will be checked by the PWM output block internally.

- We need to first download the Arduino support package for the Simulink form the internet and then build our model.

- We begin our model by keeping Arduino Analog input block and specify the pin. This basically means we will read the analog input data from the specified pin (this was done by analogRead function in C). After reading the data we send it to data conversion block and get double variable data type. We send this data to individual P, I, D block which apply the respective control action on the input signal. Then output of each control action is summed and sent to data conversion block to convert data to 8bit integer then sent Arduino PWM Analog output with specified pin number. Also sampling time for all blocks is set to 10ms.

After building the model we change simulation target to run on hardware, and select the board as Arduino and select the COM port and then deploy the model to hardware, during this process from the model auto C code is generated and implemented in Arduino directly.

We should also enable overrun detection so that any previous written code currently working on Arduino will be erased.

We can also add a scope at the output to check the results, in Simulink itself.

SOME IMPORTANT BLOCKS

ANALOG INPUT

Measure the voltage of an analog pin relative to the analog input reference voltage on the Arduino hardware. Output the measurement as a 10-bit value that ranges from 0 to 1023. If the measured voltage equals the ground voltage, the block output emits 0. If the measured voltage equals the analog reference voltage, the block output emits 1023. Parameters are Pin Number, Sample Time.

PWM

Use pulse-width modulation (PWM) to change the duty-cycle of square-wave pulses output by a PWM pin on the Arduino hardware. PWM enables a digital output to provide a range of different power levels, like that of an analog output. The value sent to the block input determines the width of the square wave, called duty-cycle that the target hardware outputs on the specified PWM pin. The range of valid input values is 0 to 255. The block input inherits the data type of the upstream block, and internally converts it to uint8. Sending the maximum value, 255, to the block input produces 100% duty-cycle, which results in full power on a PWM pin. Sending the minimum value, 0, to the block input produces 0% duty-cycle, which results in no power on a PWM pin. Sending an intermediate value to the block input produces a proportional duty-cycle and power output on a PWM pin. For example, sending 204 to the block input produces 80% duty cycle and power (204/255 = 0.8). Sending out-of-range values, such as 500 or -500, to the block input has the same effect as sending the maximum or minimum input values. Parameters of the block is Pin Number.

DATA TYPE CONVERSION

Convert input signal to specified data type. The Data Type Conversion block converts an input signal of any Simulink data type to the data type that you specify. The input can be any real- or complex-valued signal. If the input is real, the output is real. If the input is complex, the output is complex. This is a Complex block with many parameters given to control so that it gives more control to the user. Some parameters are Data type assistant, Input and Output to have equal, Integer rounding mode, Saturate integer on over flow, Sample time, Output Max and Min, Mode, Output Data type etc.

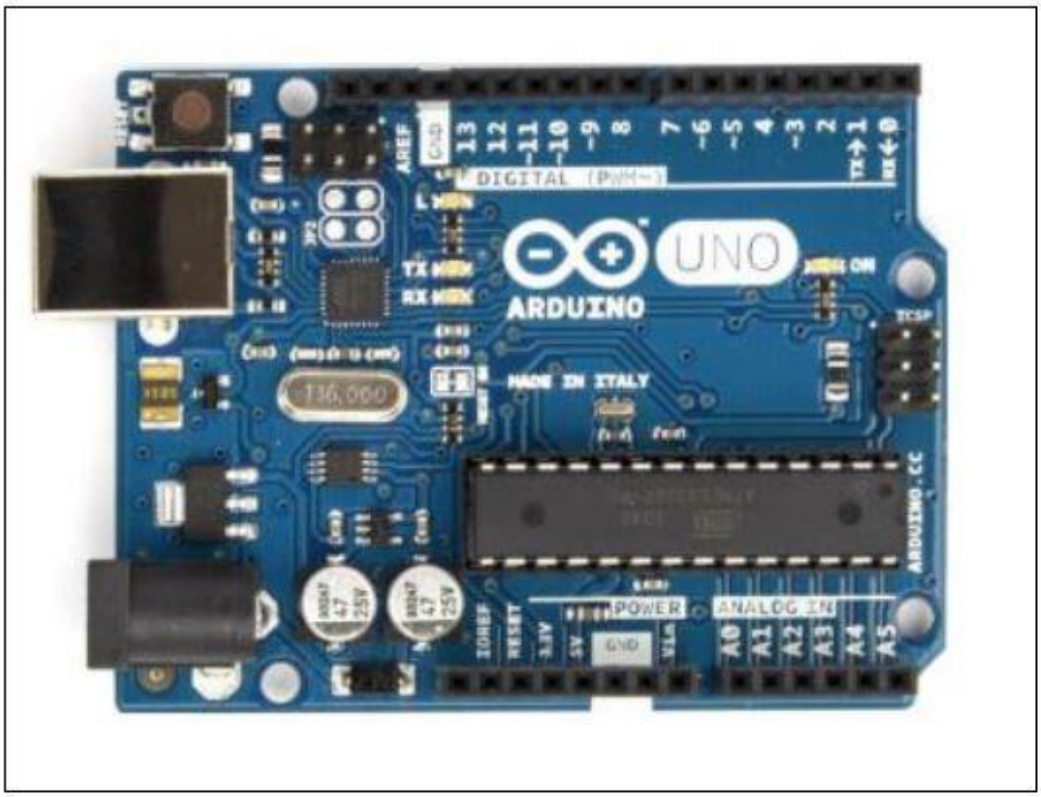
PROCEDURE

- Arduino kit was procured from the internet and was studied in detail in order to understand its hardware operations.
- Arduino IDE was installed and small C program on Blink LED was tested to verify if the board is working and understand the process of writing the C code and how to implement in hardware
- Next Matlab and Simulink with its Arduino support package was installed and small example on Blink LED using pulse generator was tested to understand how to build a model and deploy it to the hardware.
- Next a PID model was built in Simulink and coded in C (for optional part) as explained in the report.
- Low pass RC filter was built using R = 100 Kilo Ohms and C = 123 Nano Faraday, this circuit helps us to filter out all high frequencies from the PWM output and gives a true analog output. Cut off frequency fc = 13 Hz, time constant is RC = 0.0123 seconds. Resistor and capacitor is connected in series in this order and output between the capacitor and resistor is measured and connected to oscilloscope.
- Function Generator Connections: Positive terminal from the signal generator is connected to the potentiometer and ground from the potentiometer and Negative terminal from signal generator are connected to common ground. Output from the potentiometer is taken and attached to Arduino Analog input pin A0.

- Circuit Connections: The Arduino PWM output pin 5 is connected to the input of the Low pass filter and output of the low pass filter to oscilloscope. Analog ground from the Arduino board is connected to common ground on the bread board.
- Oscilloscope: One reading is taken from the output of potentiometer and output of low pass filter and grounds of the oscilloscope are attached to the common ground on the bread board.
- After setting up the circuit, Arduino was connected to the laptop and Simulink model was deployed into it.
- Sine, square and ramp signal were given as input to the Arduino and output reading were taken. (Amplitude – 5Vpp, Offset – 2.5V, Frequency – 1Hz).
- For different Kp, Ki, Kd, results were taken and peak to peak voltage were measured.

FIGURES

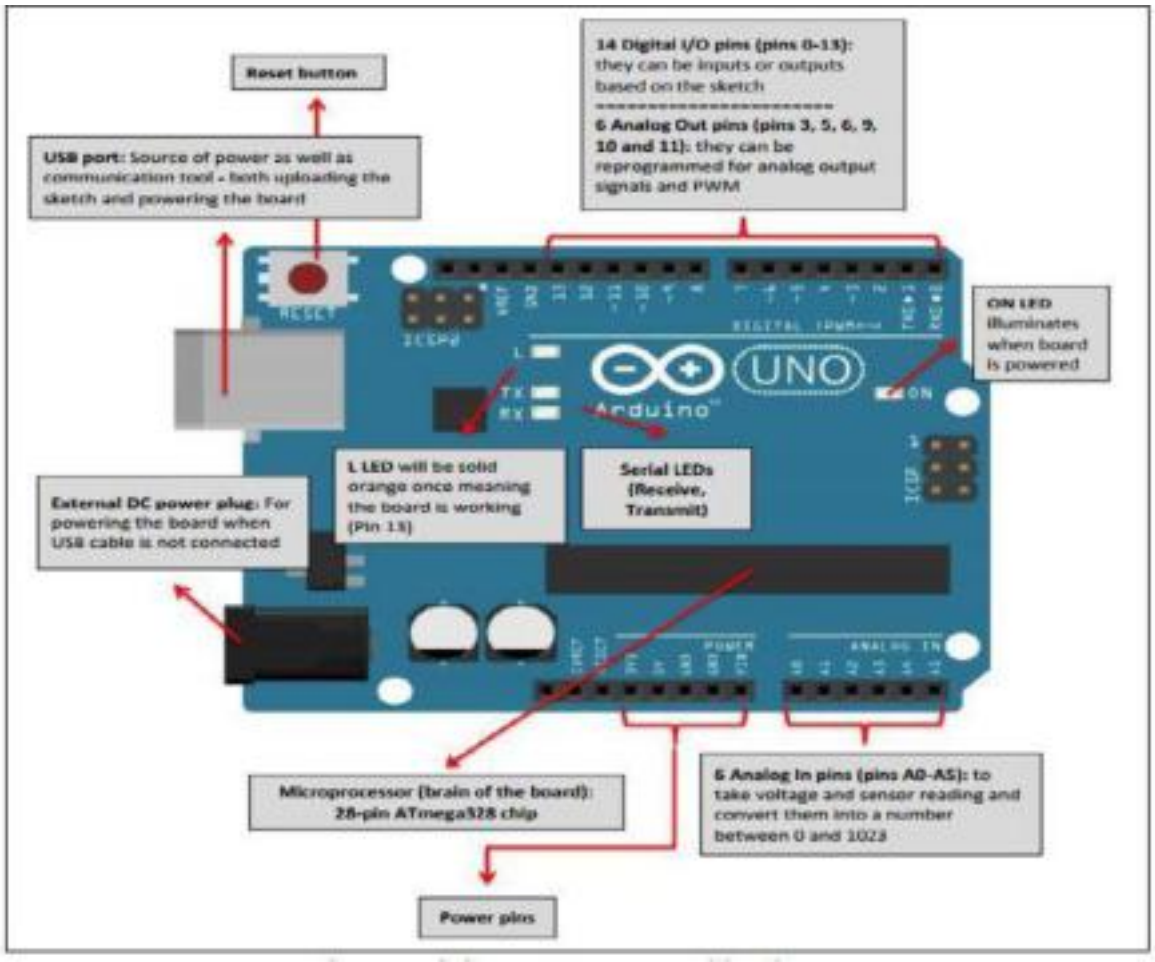
ARDUINO



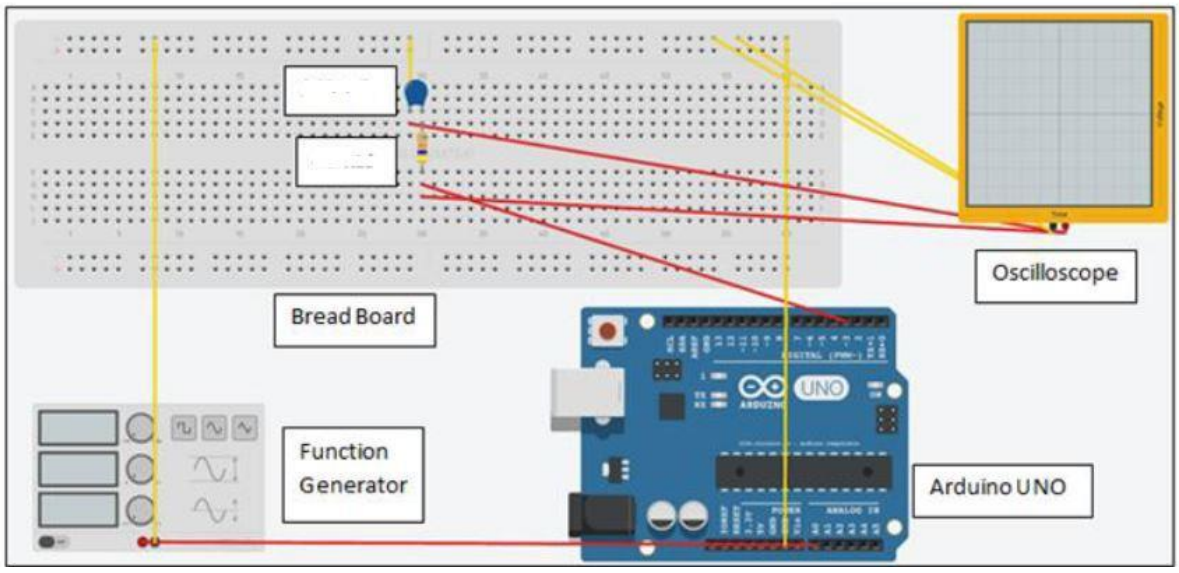
SPECIFICATIONS OF ARDUINO

Microcontroller chip	**ATmega 328**
Operating Voltage	**5V**
Input Voltage	**7-12V**
Digital I/O Pins	**14 (6 PWM)**
Analog Input Pins	**6**
Clock Speed	**16 MHz**
Memory	**32KB flash**
Length	**68.6 mm (3")**
Width	**53.4 mm (2")**
Weight	**25 g**

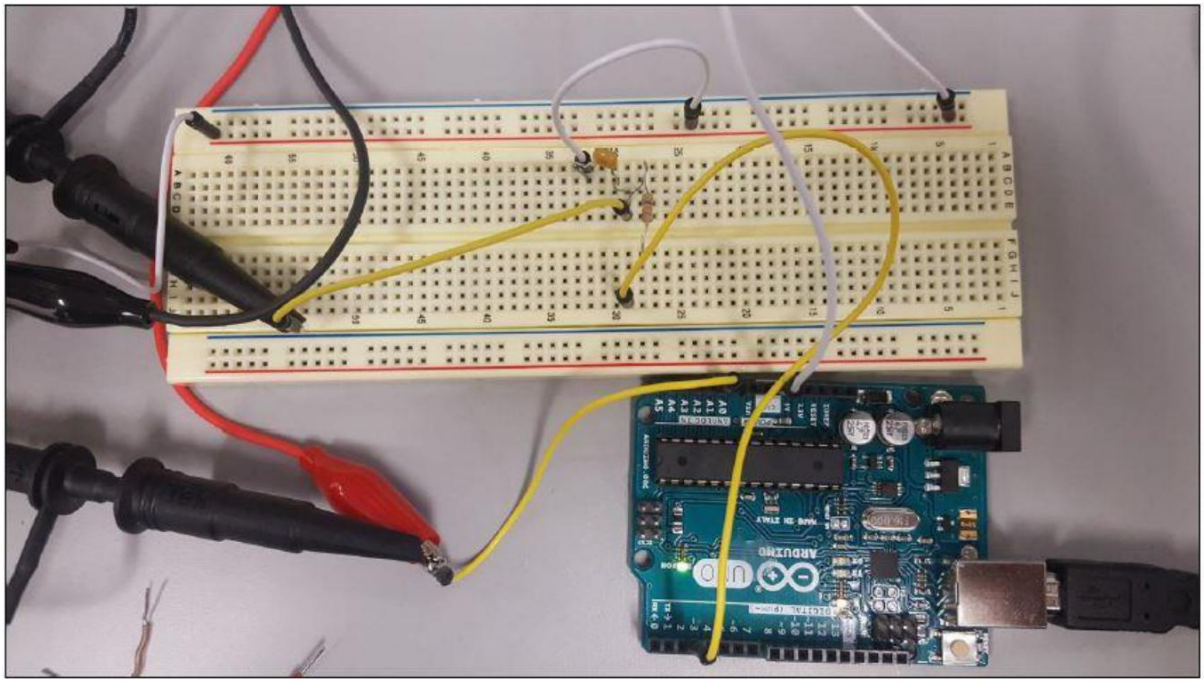
ARDUINO BOARD DETAILS

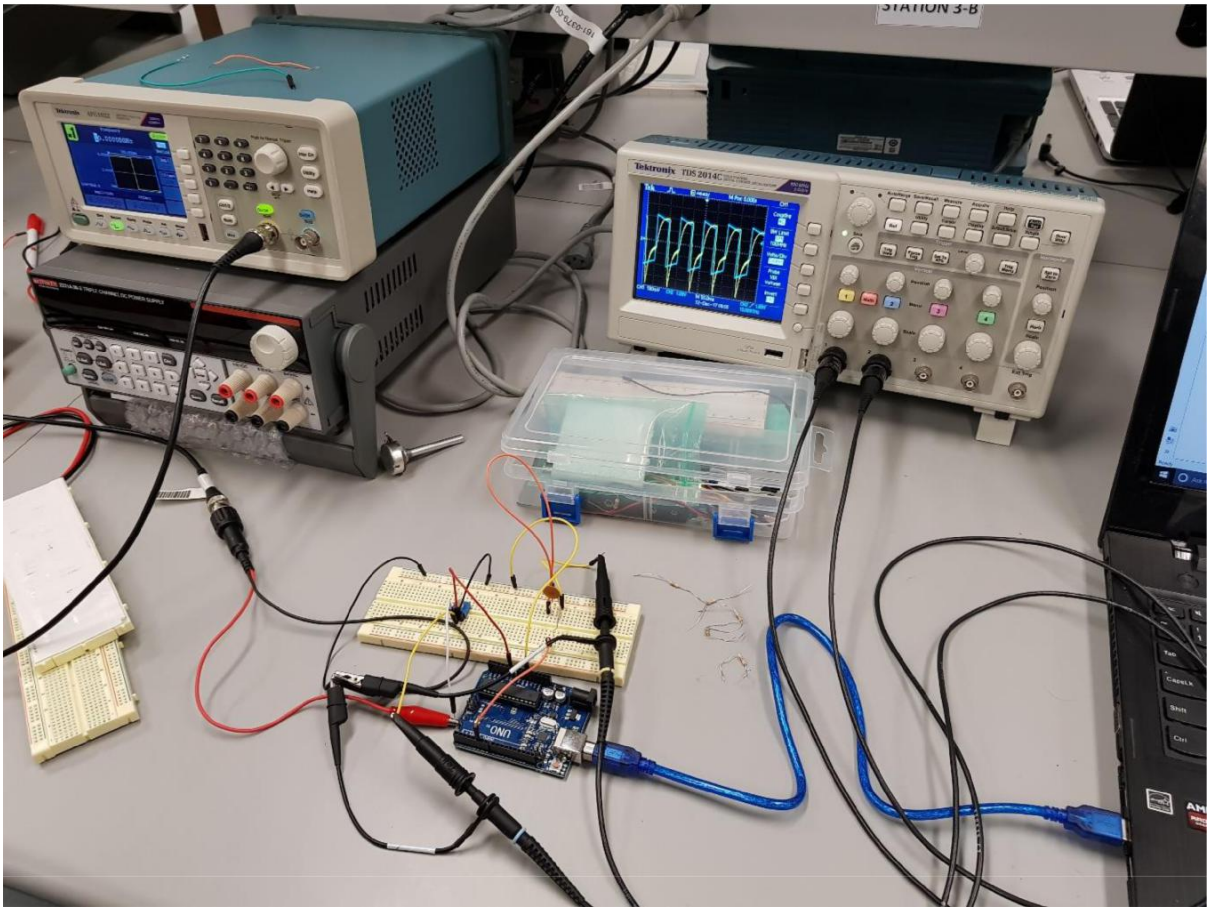


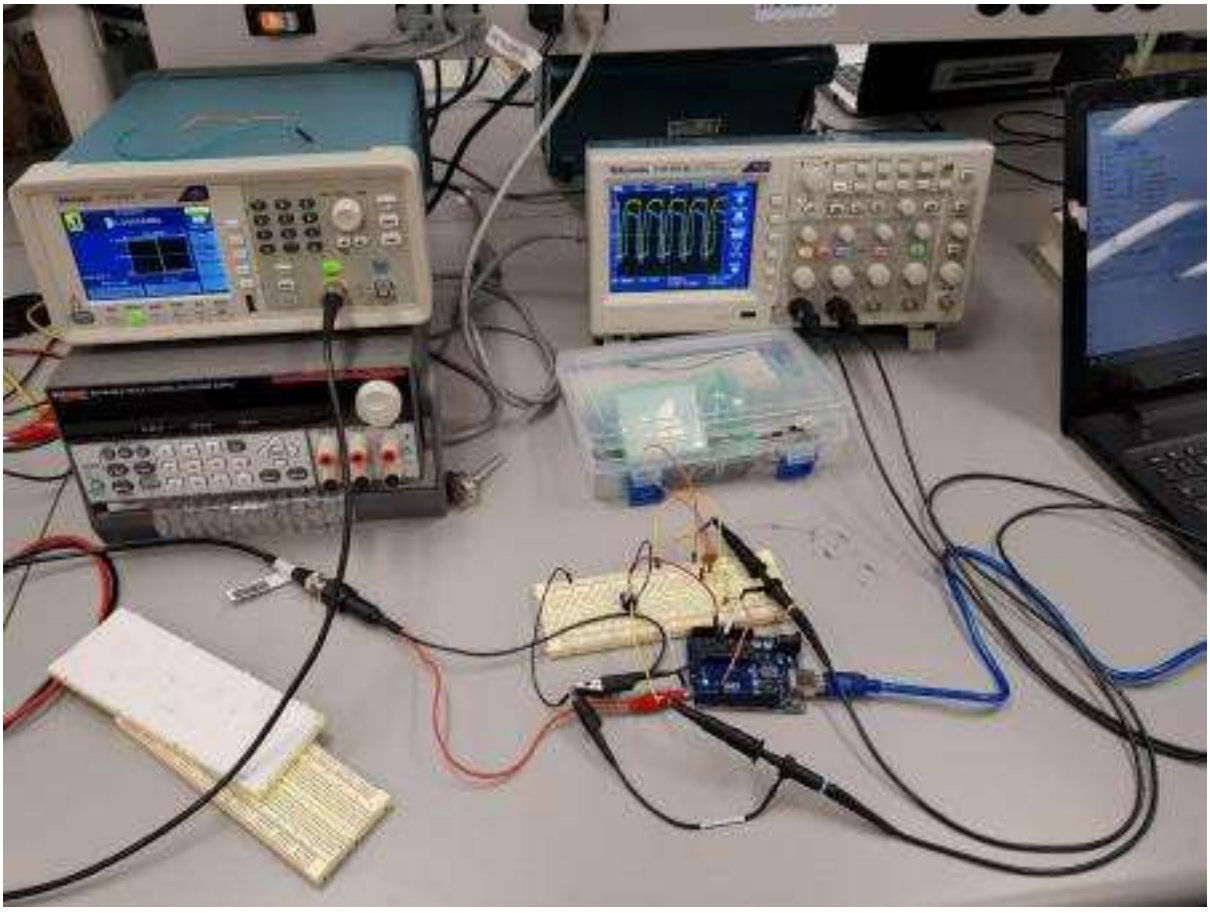
CIRCUIT SCHEMATIC

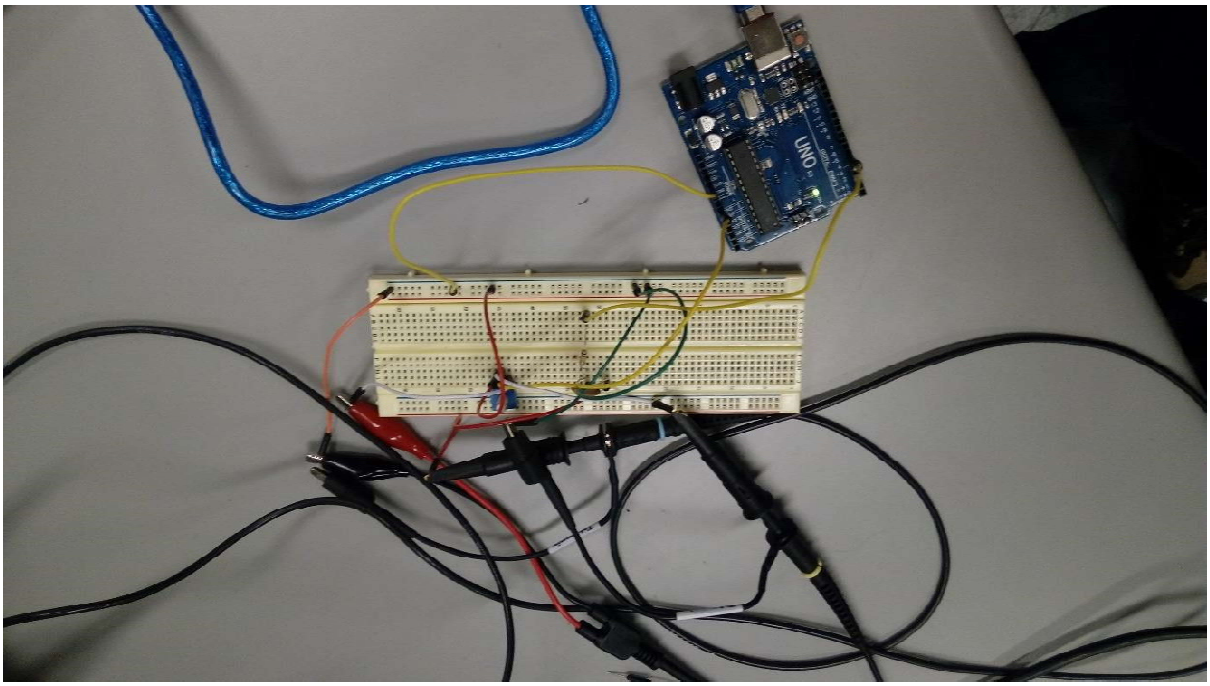


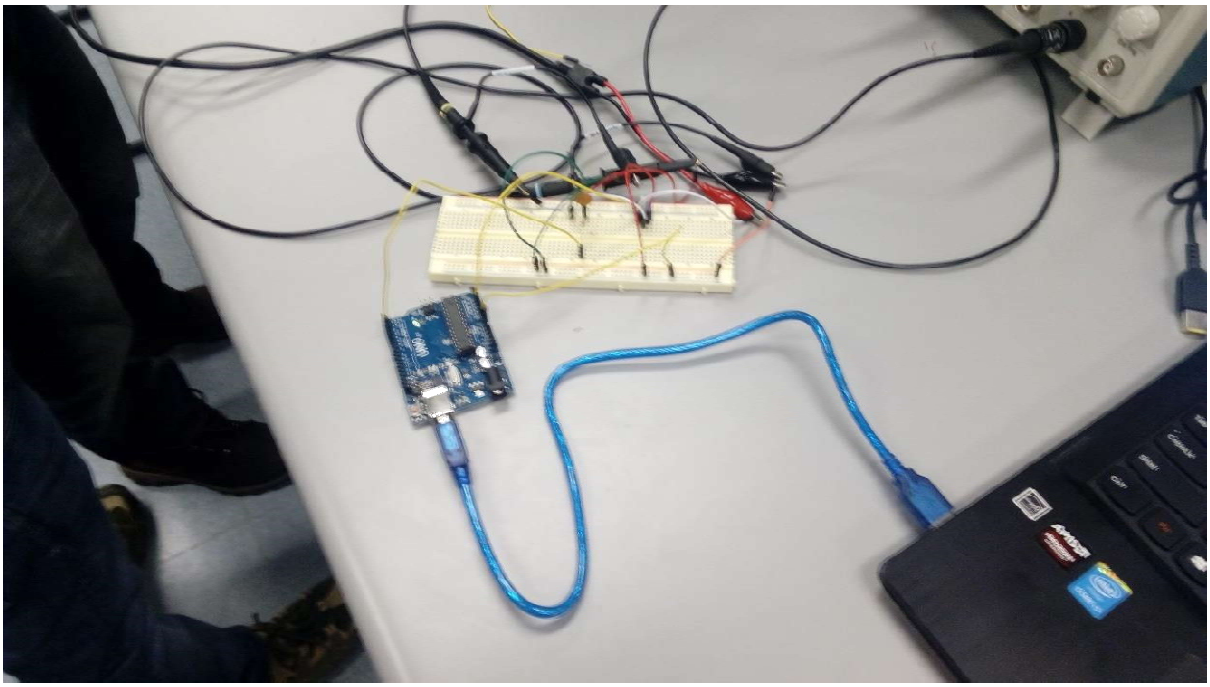
CURCUIT

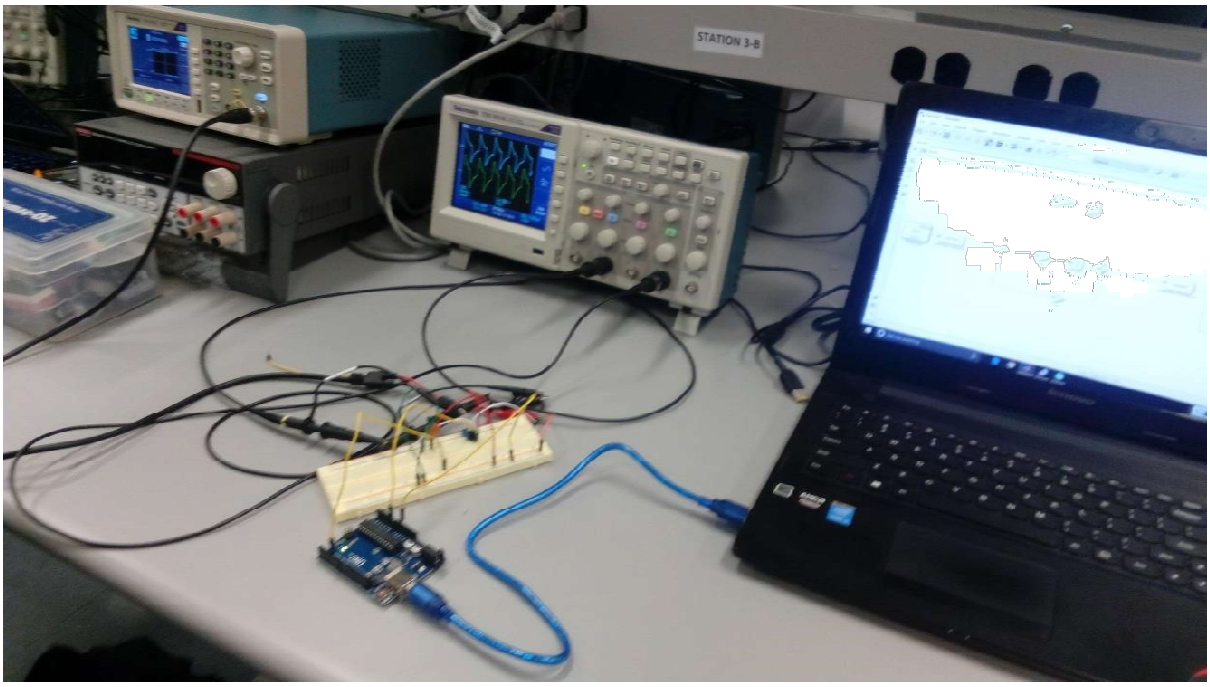


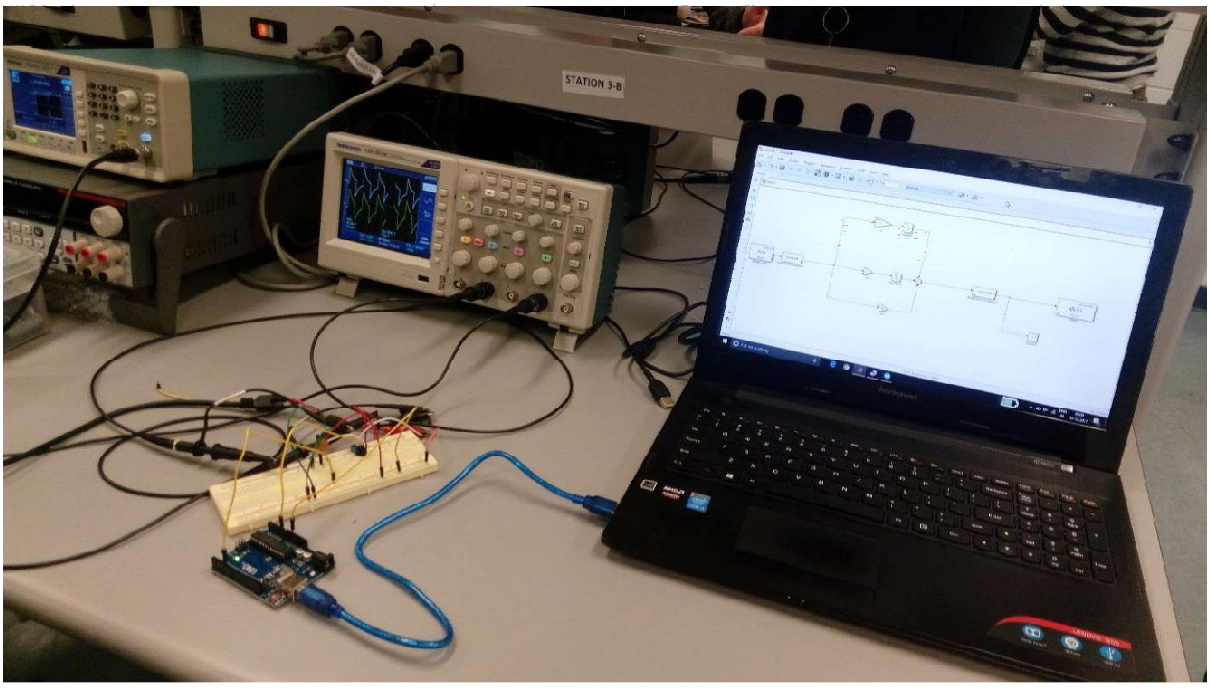





UNO

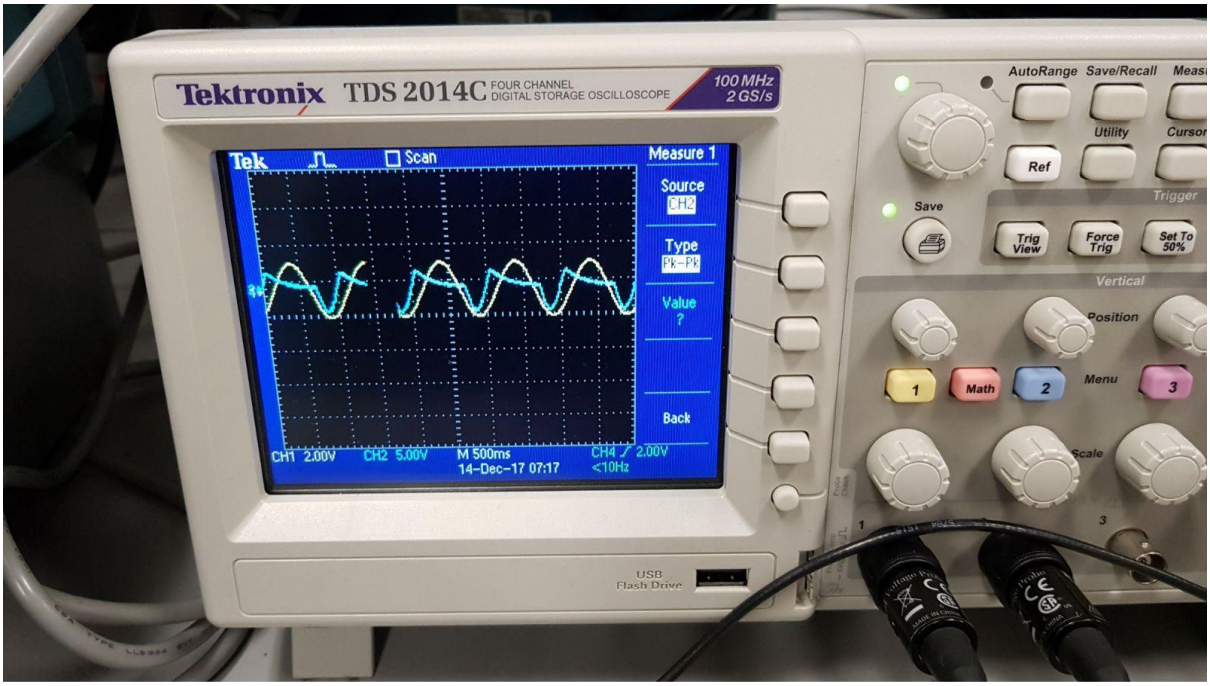



STATION 3-B

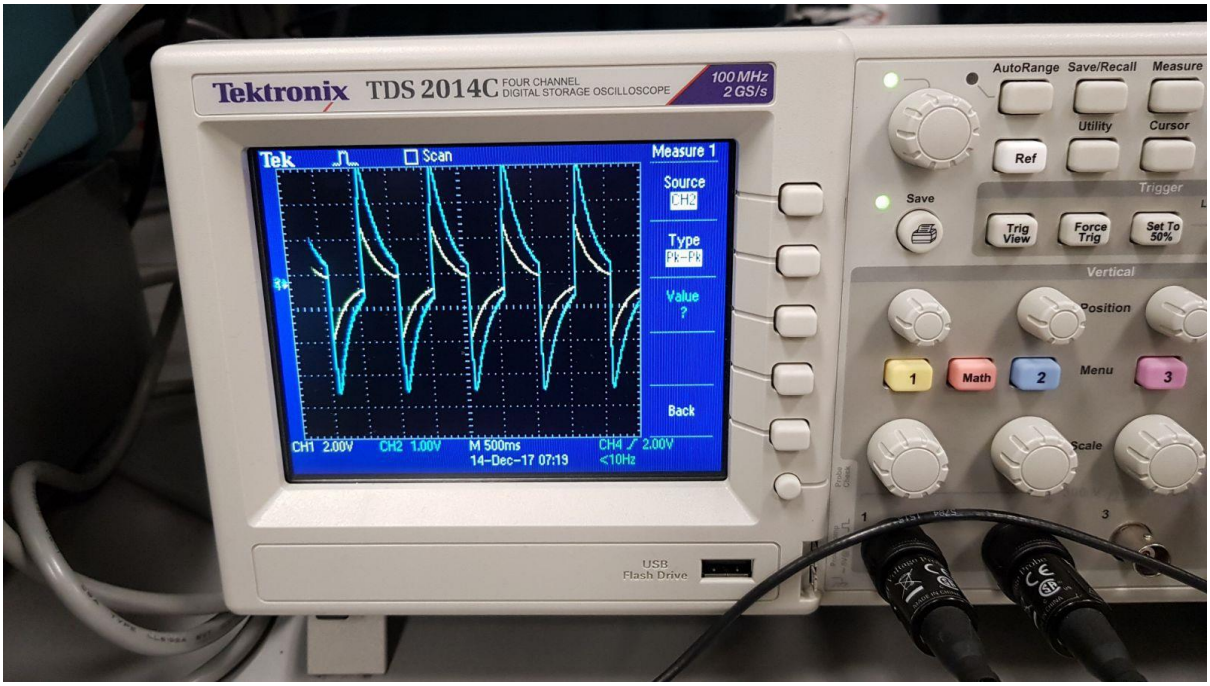

STATION 3-B

SIMULINK RESULTS

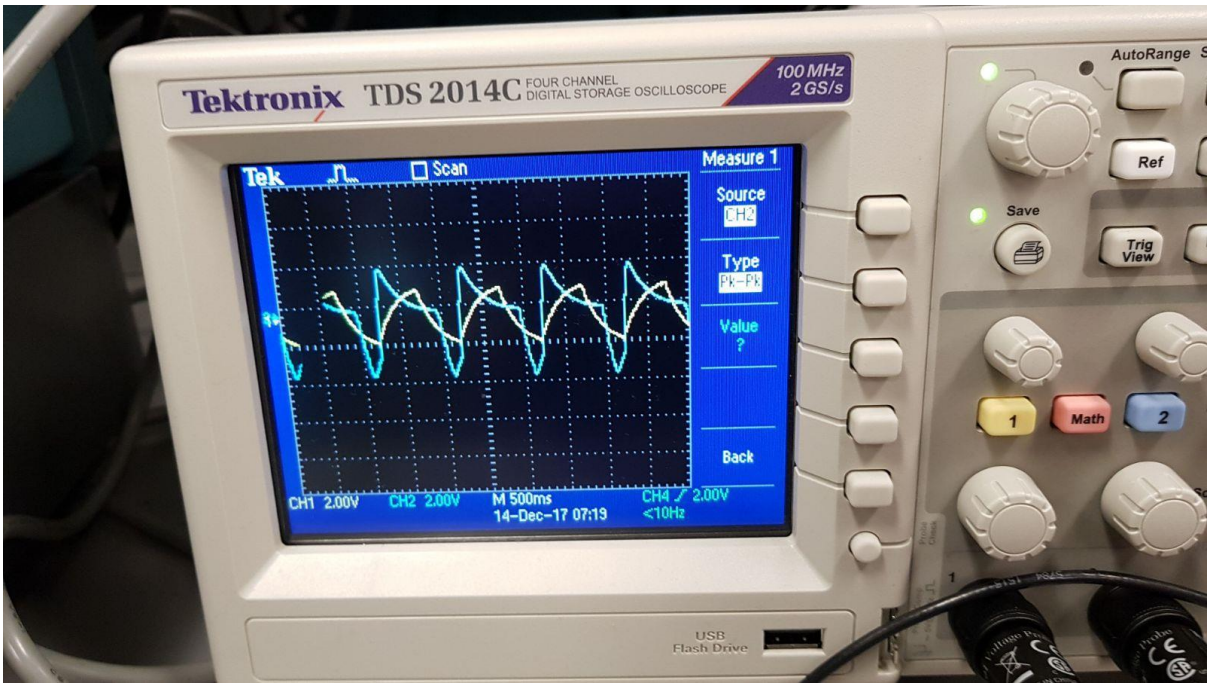
Kp = 1,
Ki = 0.005,
Kd = 0.15,
Sine Wave



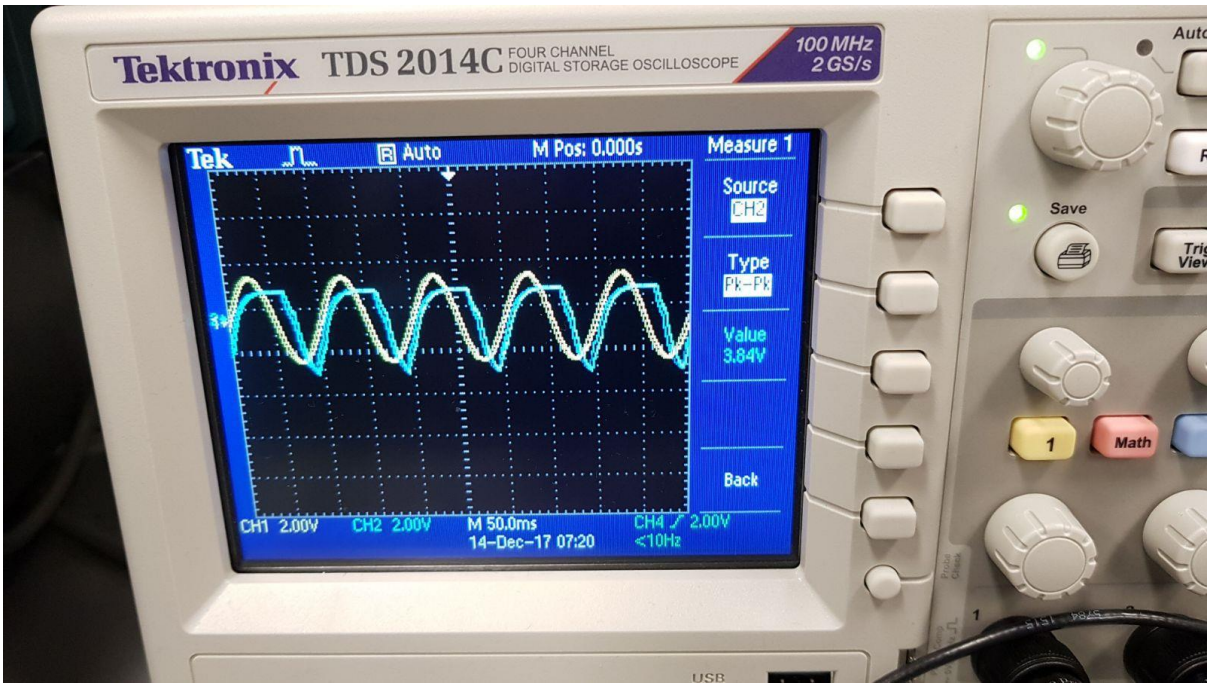
Kp = 1,
Ki = 0.005,
Kd = 0.15,
Square Wave



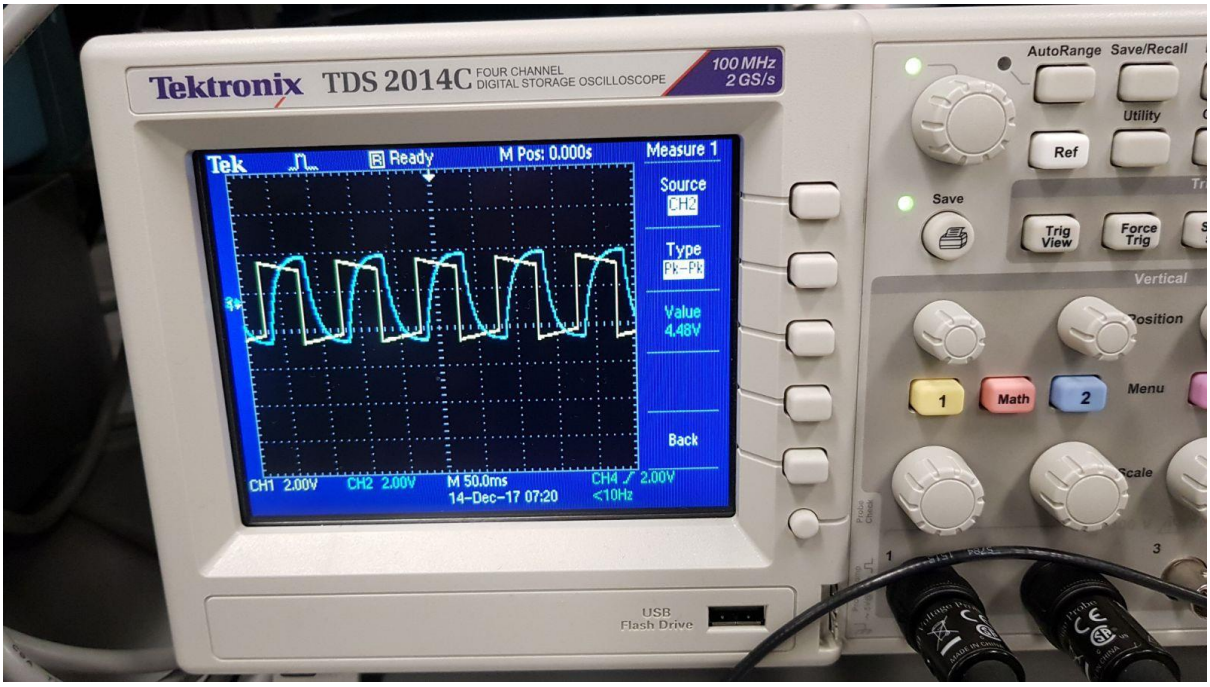
Kp = 1,
Ki = 0.005,
Kd = 0.15,
Ramp Wave



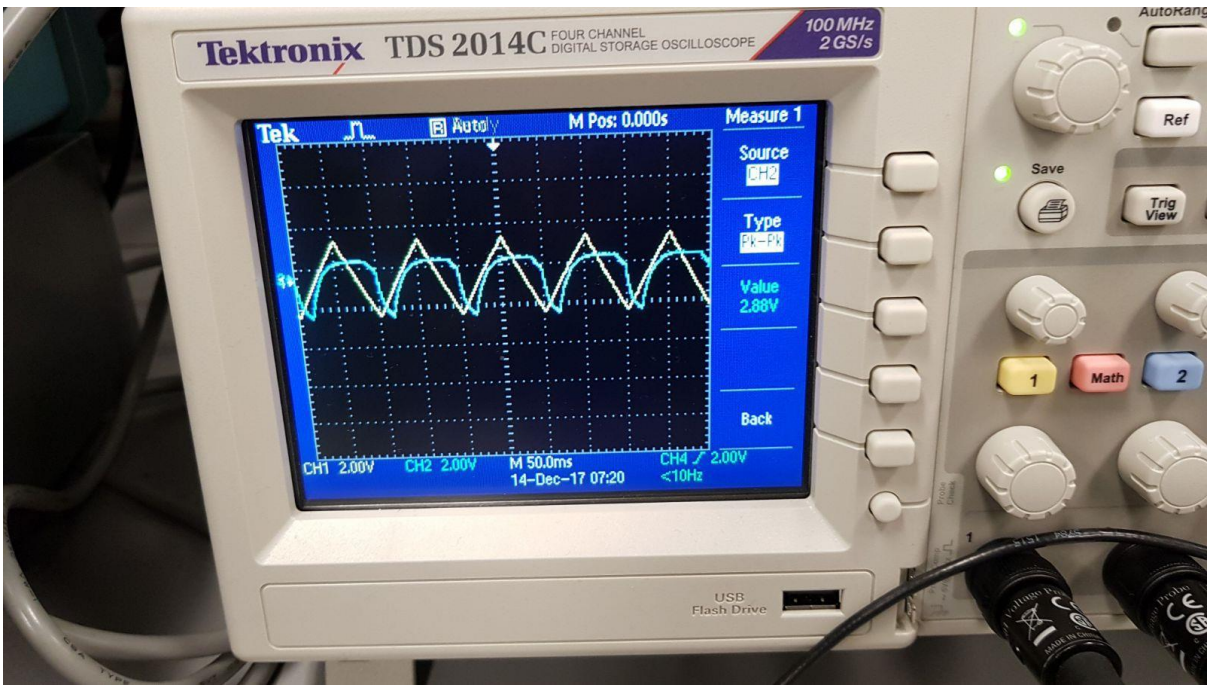
Kp = 1,
Ki = 0,
Kd = 0,
Sine Wave



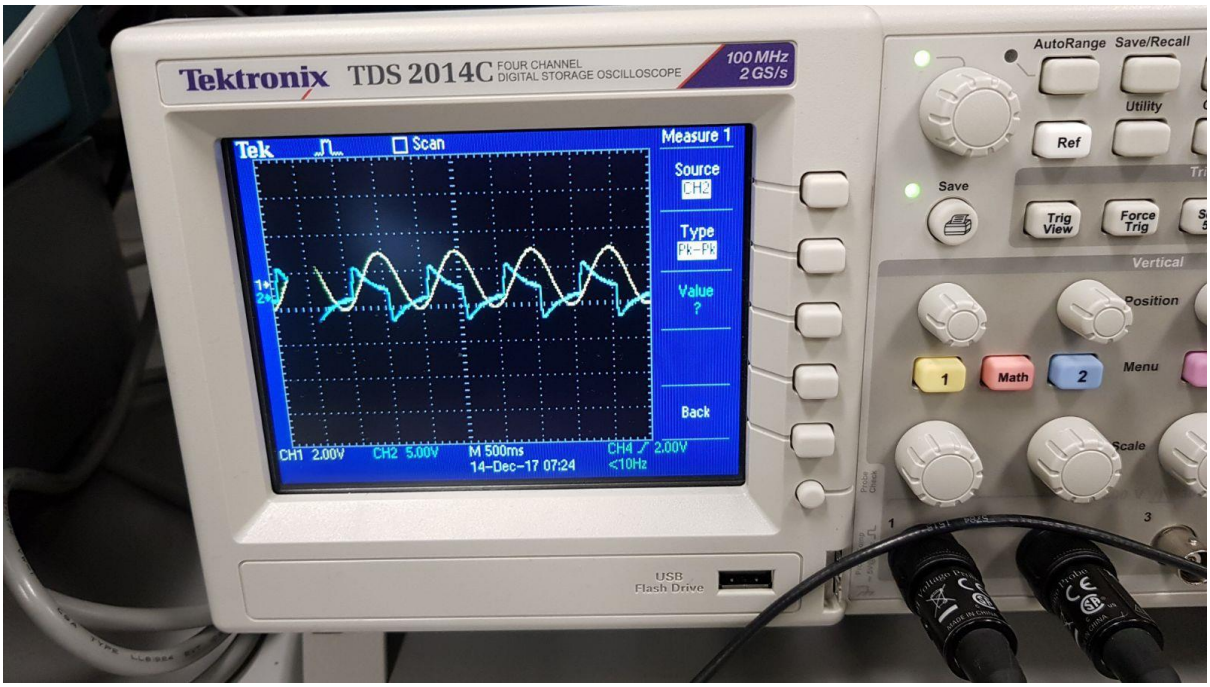
Kp = 1,
Ki = 0,
Kd = 0,
Square Wave



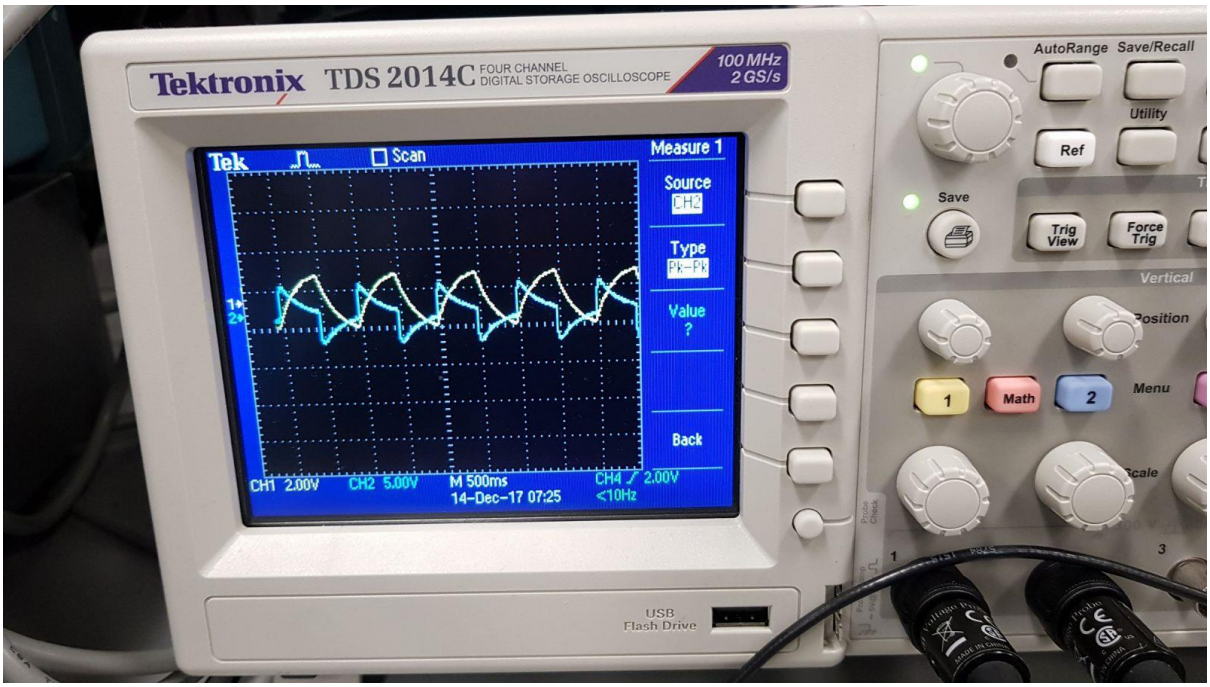
Kp = 1,
Ki = 0,
Kd = 0,
Ramp Wave



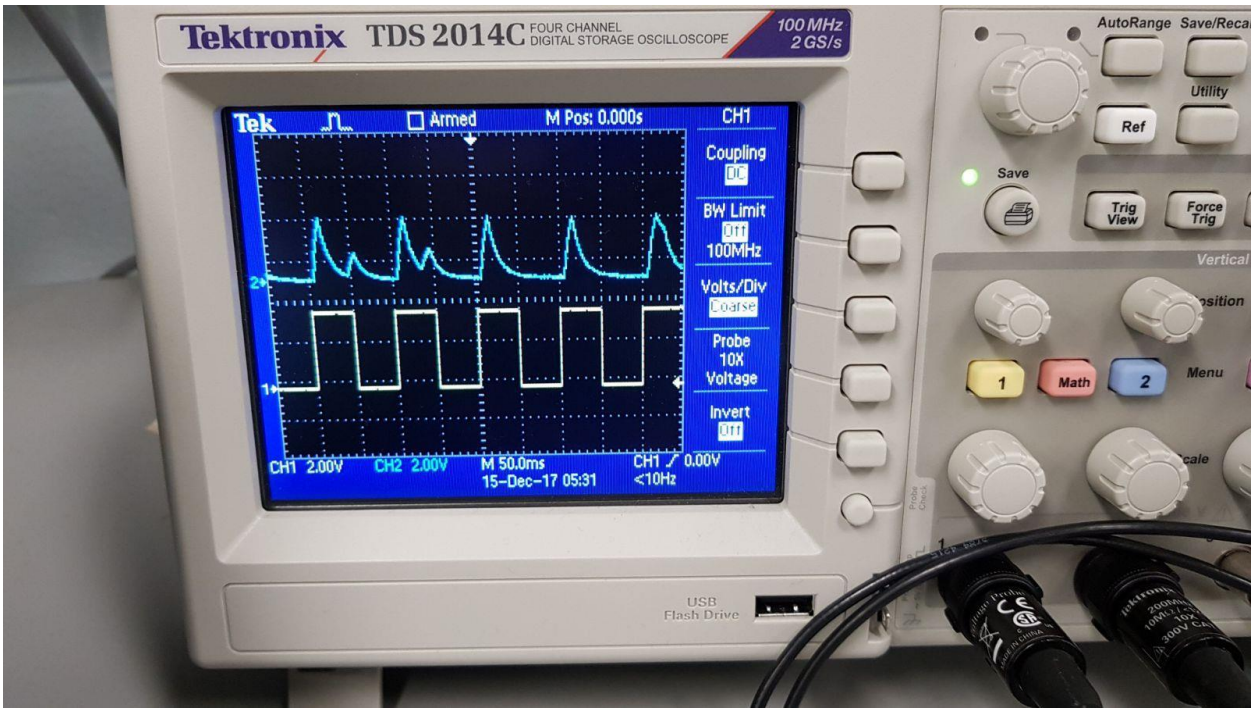
Kp = 0,
Ki = 0,
Kd = 1,
Sine Wave



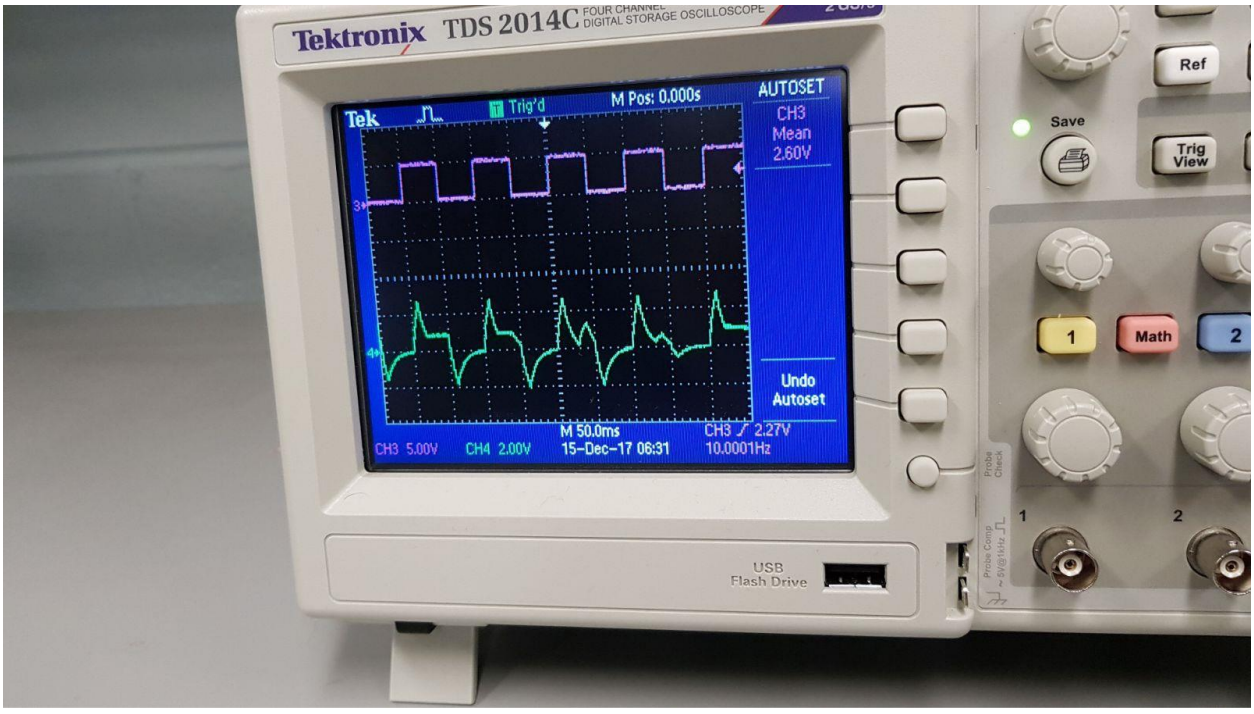
Kp = 0,
Ki = 0,
Kd = 1,
Ramp Wave



Kp = 0,
Ki = 0,
Kd = 1,
Square Wave



Kp = 1,
Ki = 0,
Kd = 0.5,
Square Wave



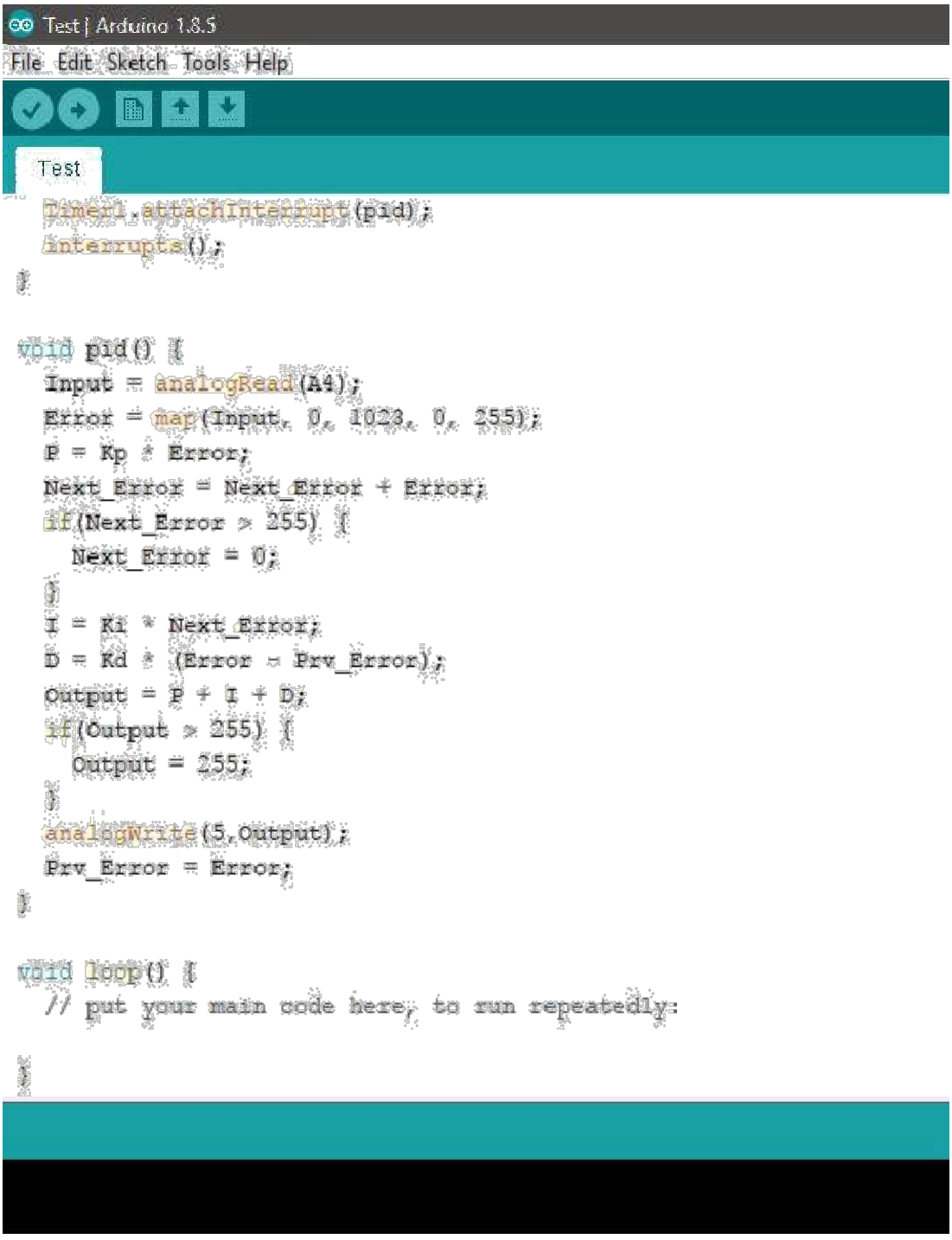
C CODE USING ARDUINO IDE

Test | Arduino 1.8.5

File Edit Sketch Tools Help

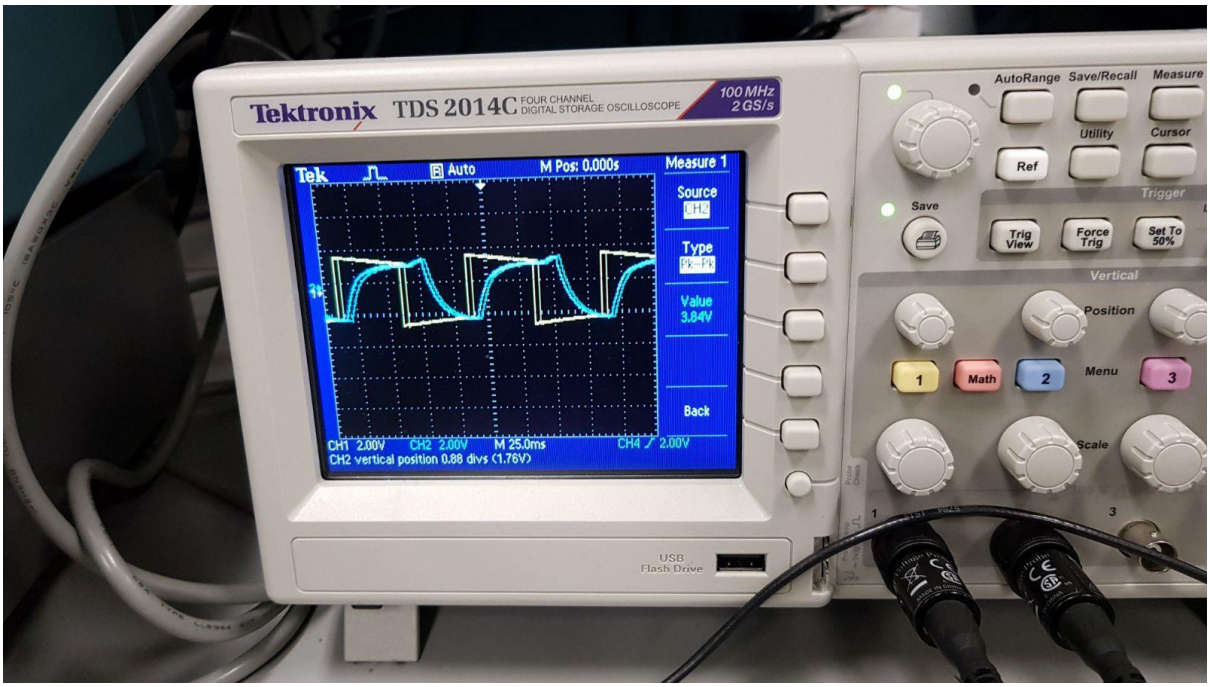
Test

```
#include <TimerOne.h>
float Input = 0;
float Error = 0;
float Prv_Error = 0;
float Next_Error = 0;
float P = 0;
float I = 0;
float D = 0;
float Output = 0;
float Kp = 2;
float Kd = 0.005;
float Ki = 0.01;

void setup() {
  // put your setup code here, to run once:
  noInterrupts();
  pinMode(A4, INPUT);
  pinMode(5, OUTPUT);
  Timer1.initialize(10000);
  Timer1.attachInterrupt(pid);
  interrupts();
}

void pid() {
  Input = analogRead(A4);
  Error = map(Input, 0, 1023, 0, 255);
```

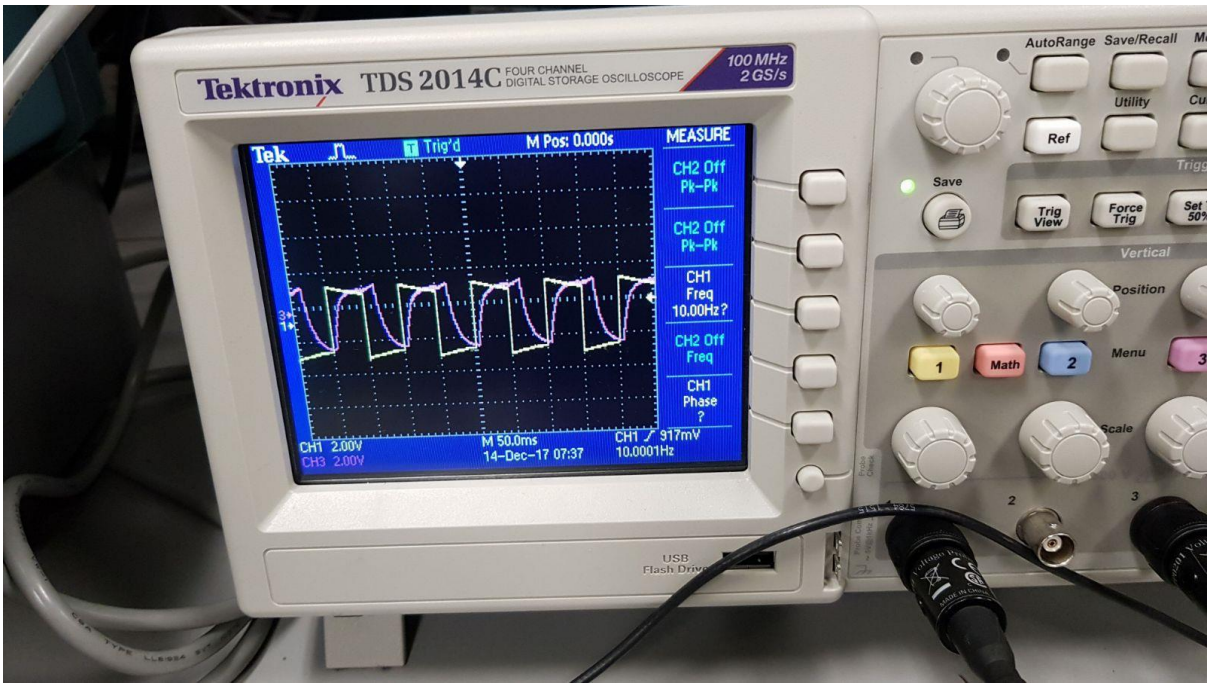

Test | Arduino 1.8.5
File Edit Sketch Tools Help
Test
Timer1.attachInterrupt(pid);
interrupts();
}
void pid() {
Input = analogRead(A4);
Error = map(Input, 0, 1023, 0, 255);
P = Kp * Error;
Next_Error = Next_Error + Error;
if(Next_Error > 255) {
Next_Error = 0;
}
I = Ki * Next_Error;
D = Kd * (Error - Prv_Error);
Output = P + I + D;
if(Output > 255) {
Output = 255;
}
analogWrite(5,Output);
Prv_Error = Error;
}
void loop() {
// put your main code here, to run repeatedly:
}

C CODE RESULTS USING IDE

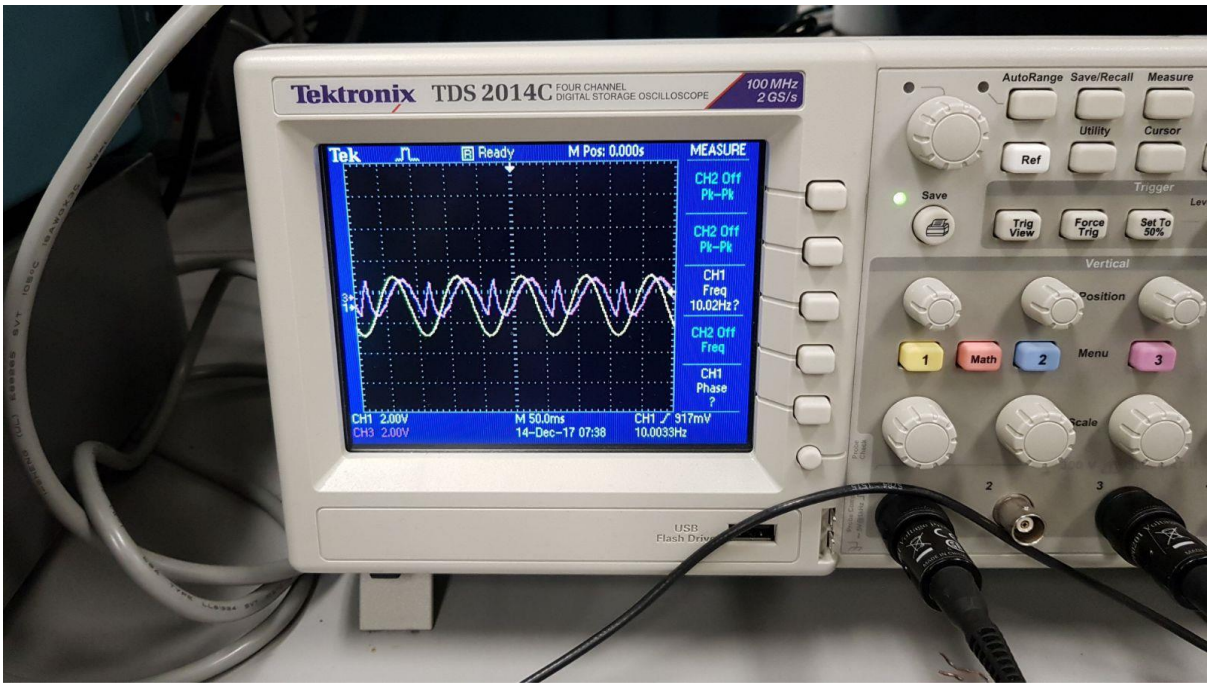
Kp = 1,
Ki = 0.005,
Kd = 0.15,
Square Wave



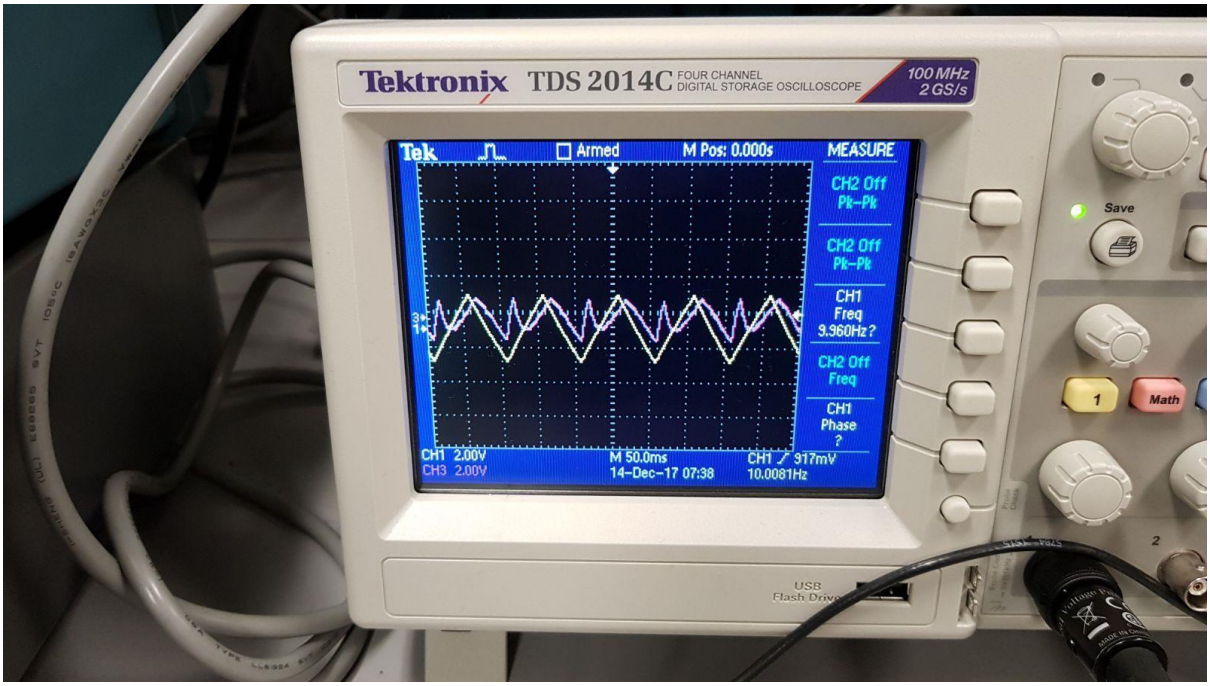
Kp = 1,
Ki = 0.002,
Kd = 0.2,
Square Wave



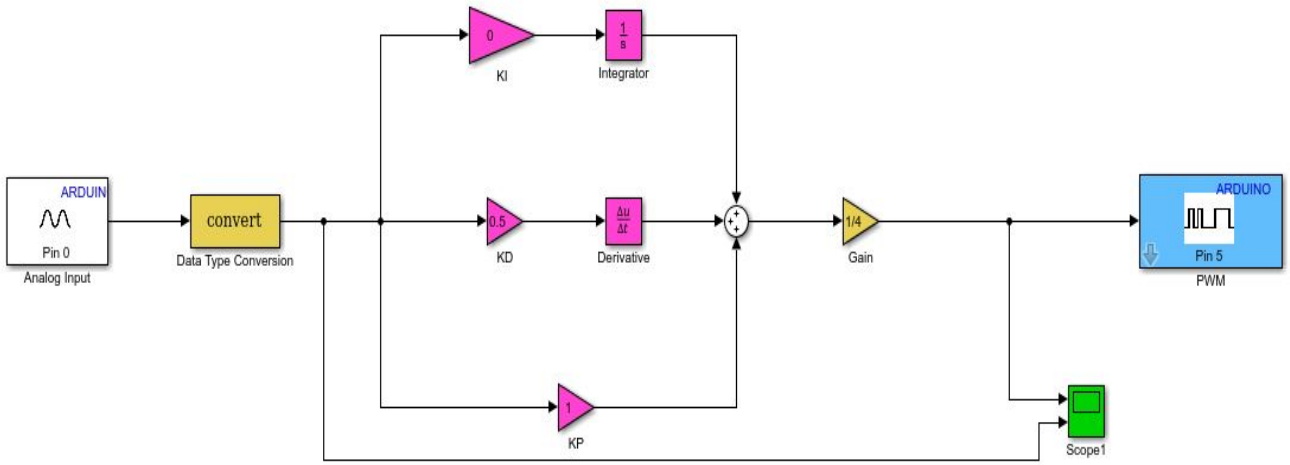
Kp = 1,
Ki = 0.02,
Kd = 0.2,
Sine Wave



Kp = 1,
Ki = 0.002,
Kd = 0.2,
Ramp Wave



SIMULINK MODEL



CONCLUSION

- For Kp = 1, and rest of the gains are zero the output signal is ideally the same as the input signal but there is certain damping in the system of the wave due to the Low Pass RC circuit. (Both for C code and Simulink)
- For Kd = 1, and the rest of the gains are zero the output signal varies with respect to the rate of the change of the error signal. The spikes are expected in output due to sudden rate of change of the input signal. We can also see approximate square wave as output for ramp input signal.(Both for C code and Simulink)
- For Ki = 1 and the rest of the gains are zero, the output signal keeps on accumulating as per the integral control definition. The C code used in Arduino, includes an anti- wind up of the integral function. Hence the output signal should reset if the error signal reaches its maximum limit (255 in this case). The spikes in the output signal are expected due to the error signal value crosses the maximum capacity, also sometimes it involves a lot of spikes because of the anti- wind up operation which resets the value of the function due to the error value exceeding 255. But in Simulink since I have not used

any anti-wind up since, the output eventually adds and saturates and gives output as straight line.

- For Kp = 1, Ki = 0.005, Kd = 0.15, the output signal is very similar to the input, with the slope different due to differential controller. Since the Ki value is low, there is not much saturation of the error but after a certain time period, the output signal spikes due to the anti-wind up of the integral control (for the C code) and the outputs, keep alternating with a regular time interval. But for in Simulink, since there is no prevention for wind up, output eventually saturates.

ADDITIONAL OBSERVATIONS

- It was observed that the proportional gain could not be increased as the input voltage was 5V and further increasing the Kp will result in redundancy, since the output will not be affected, and it will saturate for values higher than 5V, this happens because Arduino can only be operated between 0 to 5V. (Both for C code and Simulink)
- The changes in the Kd, resulted in the sharpness of the curve when the input error was a square wave function. Higher was the value of Kd, quicker was the descend in the output curve along with a sharper curve. (Both for C code and Simulink)
- The Integral gain contributed in flickering of the output curve. Higher the Ki value, more would the output signal flicker. This was also partly due to the anti-wind up of the integral control in C code. In Simulink for higher values of Ki output saturated with in no time and became a straight line, as there was no prevention for anti-wind up.

PROBLEM FACED

- The capacitor chosen earlier was not charging fast enough w.r.t to the sampling time period. This problem was solved by choosing a capacitor with lower capacitance and also taking lower value of resistance, as it decreased the time constant of the RC circuit which in decreases the time required for the capacitor to charge and reach the output peak value.
- In C coding for the Arduino, faced issues with installing Timer1.r11 library finally solved through going through the trouble shooting docs provided the Arduino community.
- The oscilloscope and Function generator was faulty because the input square was not showing as in the same form on the scope. This error was taken into consideration as mentioned in all the readings and results mentioned above in this experiment.
- While building the Simulink model we faced an issue regarding data type mismatch, as Analog input of the Arduino gives 10 bit integer and but PID model takes data type double. This problem was solved by using Data Conversion Block present in Simulink.
- Finally, precaution needs to be taken to set the Simulink environment to external and mention the COM port to which Arduino is connected explicitly to Simulink, as it can't detect that automatically. Also, we need to enable overrun detection so that program whenever new program is burnt on Arduino previous program will be erased.

REFERENCES

- https://www.mathworks.com/help/supportpkg/arduino/examples/getting-started-with-arduino-hardware.html (Math Works)
- https://www.arduino.cc/en/Guide/HomePage (Arduino)
- https://www.mathworks.com/help/supportpkg/arduino/ref/pwm.html(Math Works)
- https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/ (Arduino)
- https://www.mathworks.com/help/supportpkg/arduino/ref/analoginput.html (Math Works)
- Resistor Color Codes : Color Codes - Electronics Textbook", Allaboutcircuits.com http://www.allaboutcircuits.com/textbook/reference/chpt-2/resistor-color-codes/.
- https://www.mathworks.com/help/simulink/slref/datatypeconversion.html (Math Works)
- https://uic.blackboard.com/webapps/blackboard/execute/modulepage/view?course_id=_125696_1&cmp_tab_id=_151487_1&mode=view (Blackboard ME411Mechatronis)
- https://www.wikipedia.org/