The design task:

Motor/stering Simulation:

- **A 2nd-order type 1 system used to simulate motor**
- **Scale output voltage to present -60º and +60º**
- **Dominant time constant to be 0.2-0.5 seconds**
- **Output voltage should be attenuate or amplified to match the specification of the data acquisition system**

Real-time controller design:

- **Reference signal should be time-varying and changeable**
- **Motor position should go from 0° - 50° within 1 second with less than 10% overshoot.**
- **External signal to switch between automatic and manual model.**
- **Reset to bring output to zero.**
- **Control signal should be displayed.**

Date: 29/09/2017, Friday

Today's log summary:

- Make general plans
- Motor modeling
- Do some research on 2nd-order type system
- Find out the circuit to represent motor

General design plan:

Three states for this design tasks – 1. Motor modeling 2. simulation and 3. Implementation

1. Modeling state

- Model motor
- Find out the representation of the modeling result with electronic components

2. Simulation stage:

- motor will be simulated and tested on Simulink.
- On Simulink, PID control will be applied and PID parameters will be tuned.

3. Implementation stage:

- Make modeling motor with electronic components and test.
- PID control with Arduino.

Since we are not given a real motor to control, the most important thing at first, is to make one with electronics components as required. We need to know what kind of transfer function of a typical DC motor and then implement the transfer function by combining a serial of amplifier circuit.

Motor modeling

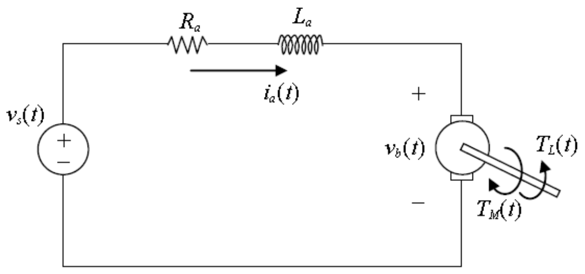


Fig 1-1 typical DC motor equivalent circuit [1]

A typical DC motor system structure is Depicted in Fig 1-1.

R_a : Armature resistance
L_a : winding leakage inductance
$v_s(t)$: voltage source
$v_b(t)$: back emf voltage.
$i_a(t)$: armature current
$\omega(t)$: angular velocity of the rotor
$T_m(t)$: generated motor torque
J_{eq} : rotor moment of inertia

According to KVL, voltage equation of the circuit is described as

$$R_a i_a(t) + L_a \frac{di_a(t)}{dt} + v_b(t) = v_s(t) \qquad (1\text{-}1)$$

$\mathrm{v_b}(t)$ is proportional to $\omega(\mathrm{t})$

$$\mathrm{v_b}(t) = k_b \omega(t) \qquad (1\text{-}2)$$

$\mathrm{T_M}(t)$ is proportional to $\mathrm{i_a}(\mathrm{t})$

$$\mathrm{T_M}(t) = k_T i_a(t) \qquad (1\text{-}3)$$

$$k = k_t = k_b \qquad (1\text{-}4)$$

Thus, Revise equation (1-1)

$$R_a i_a(t) + L_a \frac{di_a(t)}{dt} + k\omega(t) = v_s(t) \qquad (1\text{-}5)$$

Assume there is no external torque and neglect friction and dumping, then

$$T_m(t) = J_{eq}\left(\frac{d^2}{dt^2}\theta_m(t)\right) \qquad (1\text{-}6)$$

Where $\mathrm{J_{eq}}$ is the rotor moment of inertia.

Combine (1-3), (1-4), (1-5) and (1-6), the dynamic equation of the DC motor is

$$L_a \frac{di_a(t)}{dt} + R_a i_a(t) + k\omega(t) = v_s(t) \qquad (1\text{-}7)$$

$$k i_a(t) = J_{eq}\left(\frac{d^2}{dt^2}\theta_m(t)\right) \qquad (1\text{-}8)$$

Since the electrical time constant is smaller than mechanical time constant, neglect term $\frac{di_a(t)}{d_t}$, in (1-7).

$$i_a(t) = \frac{1}{R_a} v_s(t) - \frac{k}{R_a}\omega(t) \qquad (1\text{-}9)$$

Based on the above analysis, the model can be descripted in input-output transfer function. Because the control goal is to control motor to a desired angle. Output of the transfer function should be

$$\theta_m(t) = \int_0^t \omega(\tau)dt \qquad (1\text{-}10)$$

lead to

$$\dot{\theta_m}(t) = \omega(t)$$

Perform Laplace transfer of (1-6) and (1-8)

$$i_a = \frac{J_{eq} s^2 \theta_m(s)}{k} \qquad (1\text{-}11)$$

$$v_s(s) - \frac{R_a J_{eq} s^2 \theta_m(s)}{k} - k\theta_m(s) = 0 \qquad (1\text{-}12)$$

Finally, we get the open-loop transfer function of a typical motor.

$$G(s) = \frac{\theta_m(s)}{v_s(s)} = \frac{k}{R_a J_{eq} s^2 + k^2 s} \qquad (1\text{-}13)$$

Circuit representation

Op-amp is an amplifier with infinite Gain and Bandwidth when used in the Open-loop mode with typical DC gains of well over 100dB. [2]

Two key features should be kept in mind when dealing with Op-amp – infinite input impedance and zero input offset voltage. More specifically, no current will flow into either "-" or "+" input and "$V_1 = V_2$"

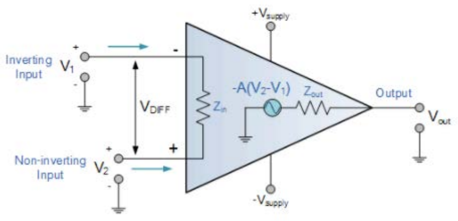


Fig 1.2 typical amplifier layout

The transfer function (1-3) can be rewritten as

$$\mathrm{G(s)} = \frac{\theta_m(s)}{v_s(s)} = \frac{k}{As(Bs+1)} = \frac{1}{As} \cdot \frac{1}{Bs+1} \qquad (1\text{-}14)$$

Where $\mathrm{A} = \frac{1}{k}$, $B = \frac{R_a J_{eq}}{k^3}$.

(1-14) can be formed by cascading two Op-amp (Operational Amplifier) circuits – integrator amplifier circuit and inertia amplifier circuit.

$\frac{1}{As}$ corresponds to an Integrator Amplifier circuit, whose circuit is as below.

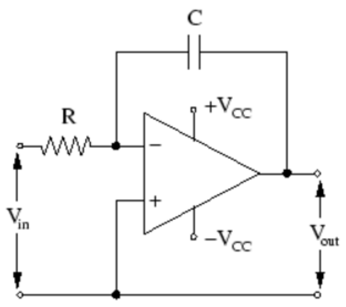


Fig 1.3 Integrator Amplifier circuit

Its transfer function is

$$G_1(s) = \frac{V_{out}}{V_{in}} = -\frac{1}{RCs} \tag{1-15}$$

$\frac{1}{Bs+1}$ corresponds to an Inertia Amplifier circuit, whose circuit is as below.

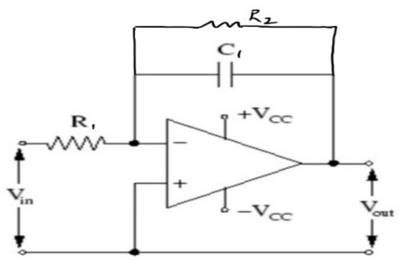


Fig 1.4 Inertia Amplifier Circuit

Its transfer function is

$$G_2(s) = \frac{V_{out}}{V_{in}} = -\frac{1}{R_1\left(C_1 s + \frac{1}{R_2}\right)} = -\frac{1}{\frac{R_1}{R_2}(C_1 R_2 s + 1)} \tag{1-16}$$

Cascade the two circuits (product of (1-15) and (1-16))

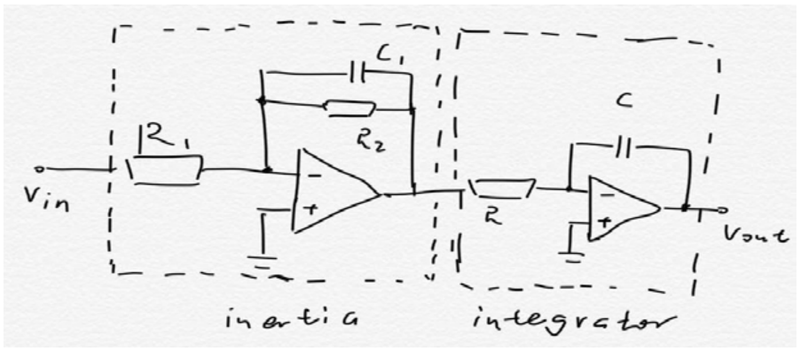


Fig 1.5 Motor modeling circuit

we get 2nd-order type transfer function

$$G_3(s) = \frac{V_{out}}{V_{in}} = \frac{1}{\frac{RCR_1}{R_2} s(C_2 R_2 + 1)} \tag{1-17}$$

Date: 2-3/10/2017, Monday &Tuesday

Today's log summary:

- Motor specification configuration
- Simulation (motor and control)
- Make motor circuit and test it.

As we got the motor model represented with electronic circuit, components specification should be configured. To test whether the designed circuit might work or not, simulation on motor circuit is necessary. If the simulation result is not desirable, then modification should be made before making the hardware circuit, as modification on hardware circuit is quite tricky. Once we get desired simulation result, a "real" motor will be made on a bread board for testing.

Motor parameters configuration:

The only recommendation or requirement for the modeling motor is to make DTC (dominant time constant) between 0.2 and 0.5s and it can be calculated as below.

$$\mathrm{DTC} = \mathrm{R}_1 \cdot C_1 \qquad (2\text{-}1)$$

The maximum capacitor available in the lab is 820 nF, which is chosen as C_1. Thus, R_1 resistance is chosen from 243.9 kΩ to 609.76 kΩ. To make the motor model's response slower, 500 kΩ is chosen for R_2. Other parameters of components are as follow:

$$R_1 = R_2 = 1M\Omega$$

$$C = 10\,nF$$

Simulation (motor and control)

Motor circuit simulation

Motor circuit is simulated under "Simscape → Electrical" library. Simulation circuit is as shown in Fig 2.1

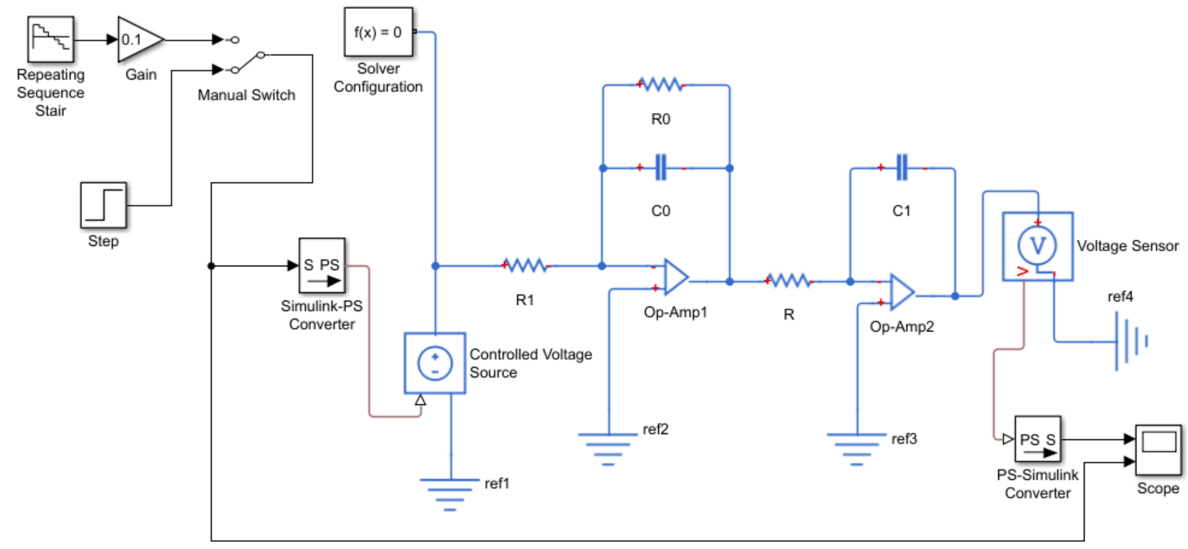


Fig 2.1 motor circuit simulation

In the simulation, parameters of each components (resistors and capacitors) are configured by a m file, of which content is as below:

Ts = .001;
R = 1000;
R0 = 500;
R1 = 1000;
C0 = 820;
C1 = 10;

Where, Ts is the filtering time constant in "Slover configuration" block, which is required in each physical network represented by a connected Simscape block diagram. [3] Its function is to specify the solver parameters needed by the model before simulation begins.

Simulink-PS Converter functions as to convert unitless Simulink signal from Step resource or Repeating Sequence Stair to Physical signal. Then the ideal voltage resource is controlled by the input signal to output ideal voltage to the motor input. The output voltage is sensed by a Voltage Sensor and then converted back to unitless signal for Scope observation.

Input a step signal with final value 5. Output signal is linear as shown in Fig 2.2.

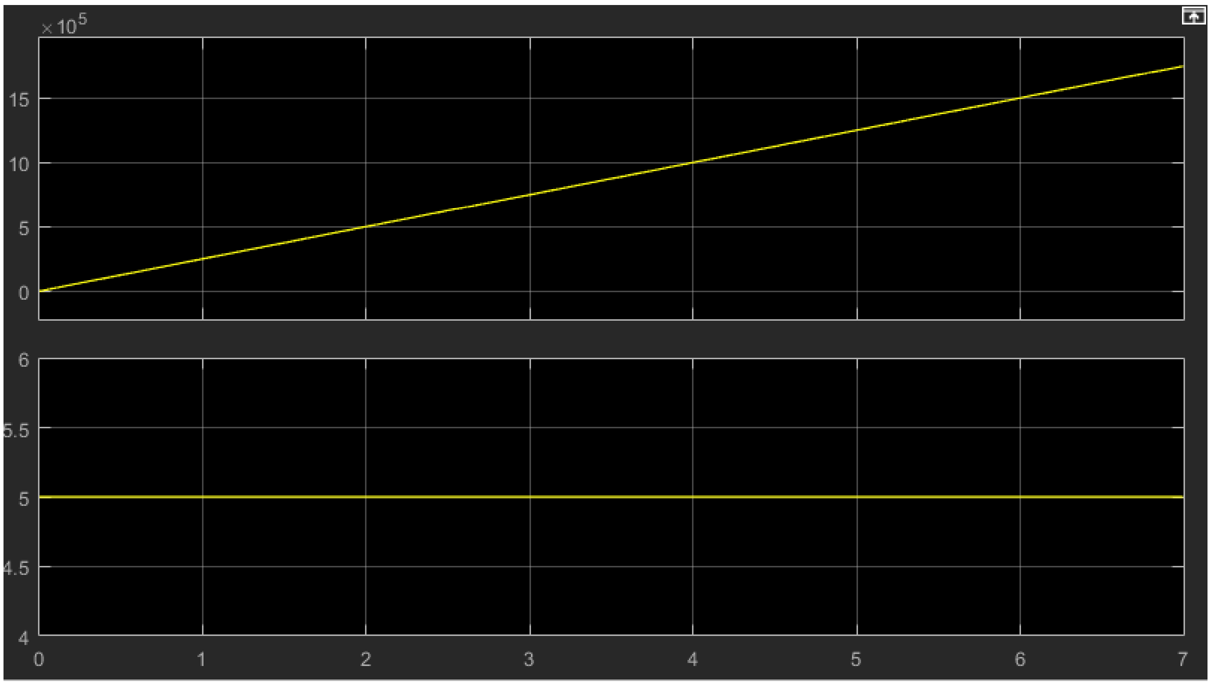


Fig 2.2 Model motor output with step input of final value 5

Decrease step output from 5 to 1. We get a series of linear output but with different slopes in proportion to input. The output signals might not be accurate but can tell how the relationship between input and output. And generally, the motor model's response is quire fast.

Output with "Repeating Sequence Stair" input value of "0.3 0.2 0.1 0 -0.1 -0.2 -0.3" is shown in Fig 2.3.

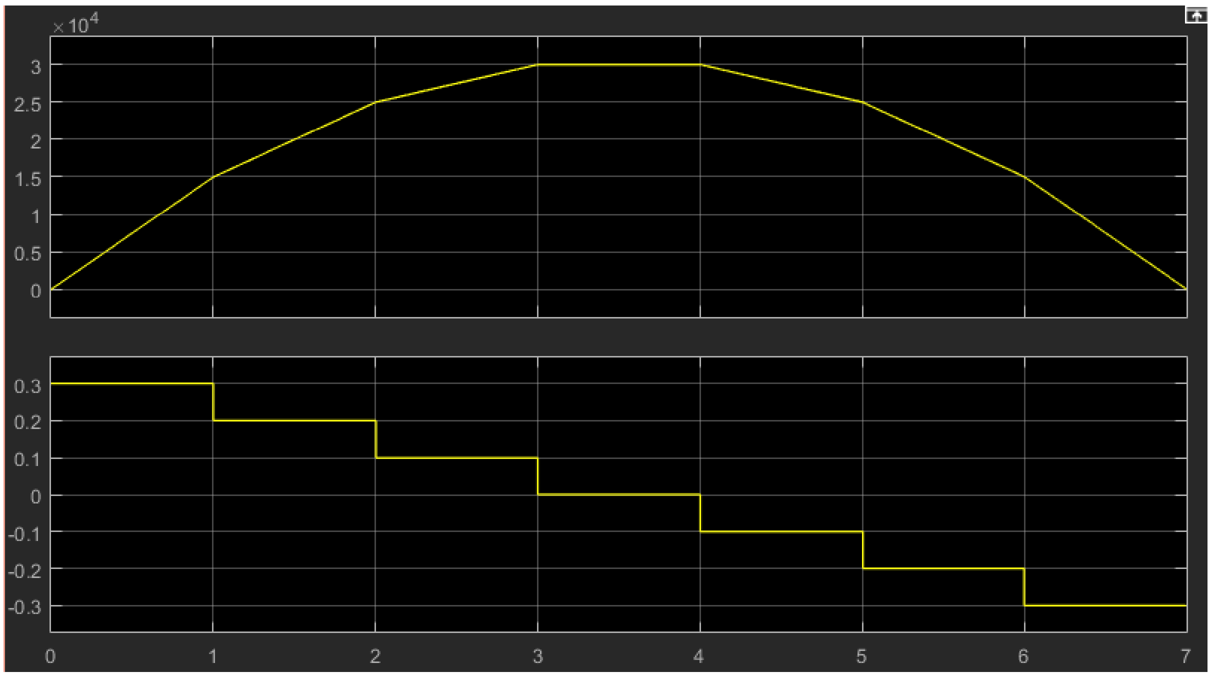


Fig 2.3 "motor" output of repeating sequence stair input

As we can see, with “0” input from 3s to 4s, the output will hold on and with negative input from 4s to 7s, the output will decrease. Thus, the “motor” can be controlled “forward rotate” or “backward rotate” with positive input or negative output.

Motor control loop simulation

In motor control, PID controller (proportional-integral-derivative controller) is widely used because of its flexibility and reliability.

The control block diagram is as shown below:

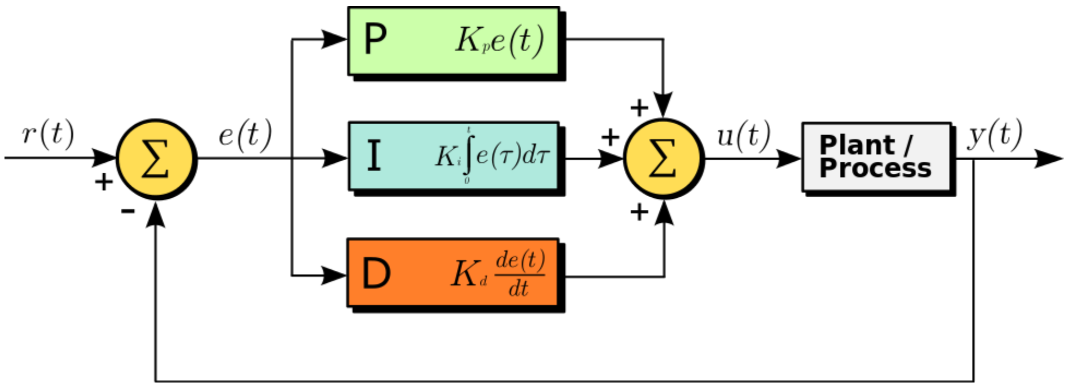


Fig 2.4 PID control block diagram [4]

The principle of PID control is that it generates correction signal (u(t)) based on proportional, integral and derivative terms to an error signal (e(t)) which is continuously calculated between a setpoint and feedback value. With output (y(t)) continuously feedbacked, the system continuously corrects itself until it approaches its setpoint with desired setpoint.

Control function derives from Fig 2.4 is

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \qquad (2\text{-}2)$$

$$\mathrm{e(t)} = r(t) - y(t) \qquad (2\text{-}3)$$

Where

u(t) : correction variable

$\mathrm{K_p}, \mathrm{K_i}, K_d$: proportional, integral and derivative terms

$\mathrm{e(t)}$: error variable

Simulation blocks for PID control is as follow:

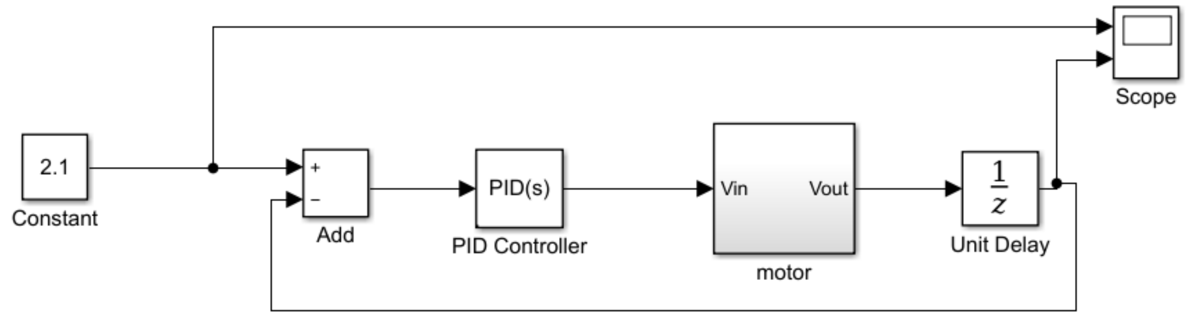
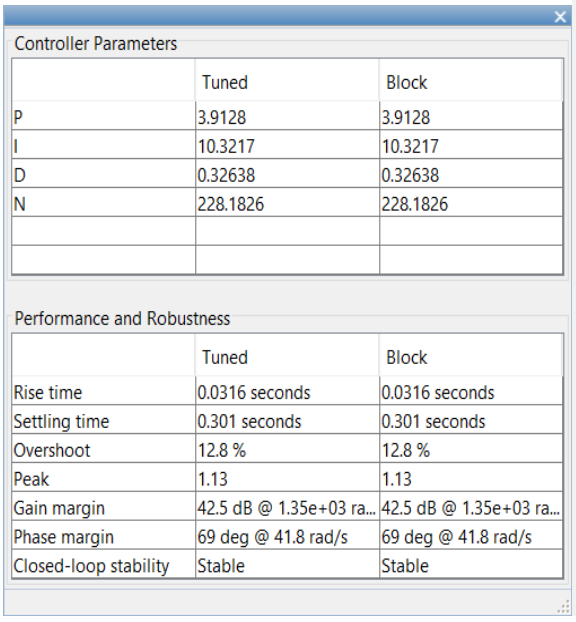


Fig 2.5 PID control blocks

Among them, motor block is a subsystem created by packing the motor modeling circuit in Fig 2.1. Unit delay here is to introduce a delay to break the algebraic loop but the function will be the same. PID controller block is to calculate correction variable. P, I, D parameters can be tuned with the help of PID Tuner.

While the simulation result is a little different from that shown in the Tuner.



Controller Parameters

	Tuned	Block
P	3.9128	3.9128
I	10.3217	10.3217
D	0.32638	0.32638
N	228.1826	228.1826

Performance and Robustness

	Tuned	Block
Rise time	0.0316 seconds	0.0316 seconds
Settling time	0.301 seconds	0.301 seconds
Overshoot	12.8 %	12.8 %
Peak	1.13	1.13
Gain margin	42.5 dB @ 1.35e+03 ra...	42.5 dB @ 1.35e+03 ra...
Phase margin	69 deg @ 41.8 rad/s	69 deg @ 41.8 rad/s
Closed-loop stability	Stable	Stable

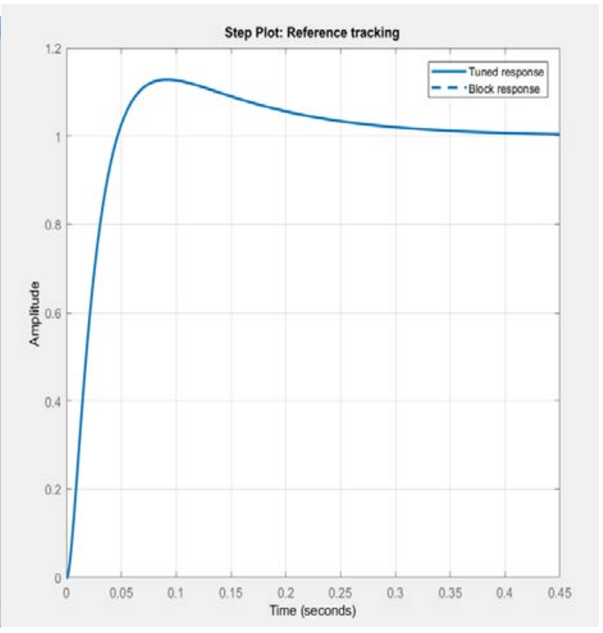


Fig 2.6 Tuner results (a) parameters (b) response graph

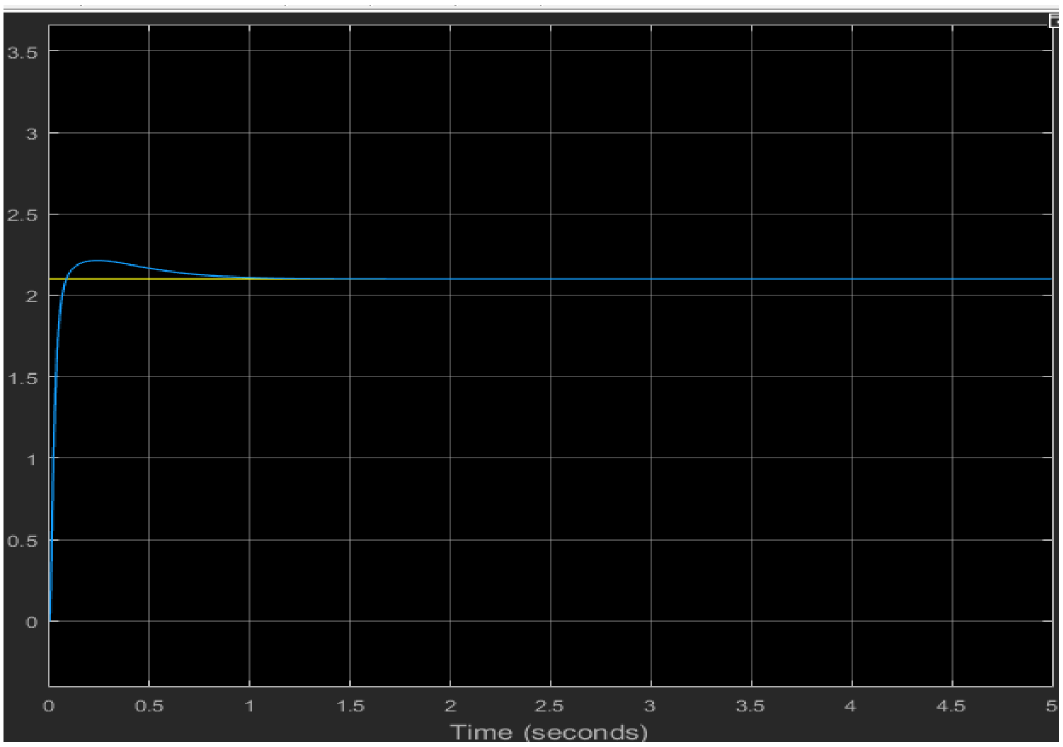


Fig 2.7 motor control simulation result

Simulation result shows that, with P = 3.9128, I = 1.3217, D = 0.32638, the controller is able to control the output to reach 2.1 with less than 10% overshoot and around 0.8s setting time, which meets the requirement. While overshoot in PID Tuner is 12.8%.

Motor circuit testing

Amplifier in Fig 2.1 is replaced by an amplifier IC – LM324N, which has four independent, high-gain, internally frequency compensated Op-amps designed to operate from a single power supply over a wide range of voltages [5] (3-32V).

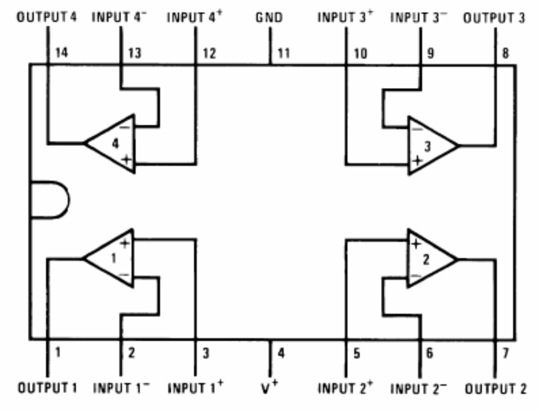


Fig 2.8 LM324N foot pin

Connect the circuit according to Fig 2.1, and the real circuit is as below.

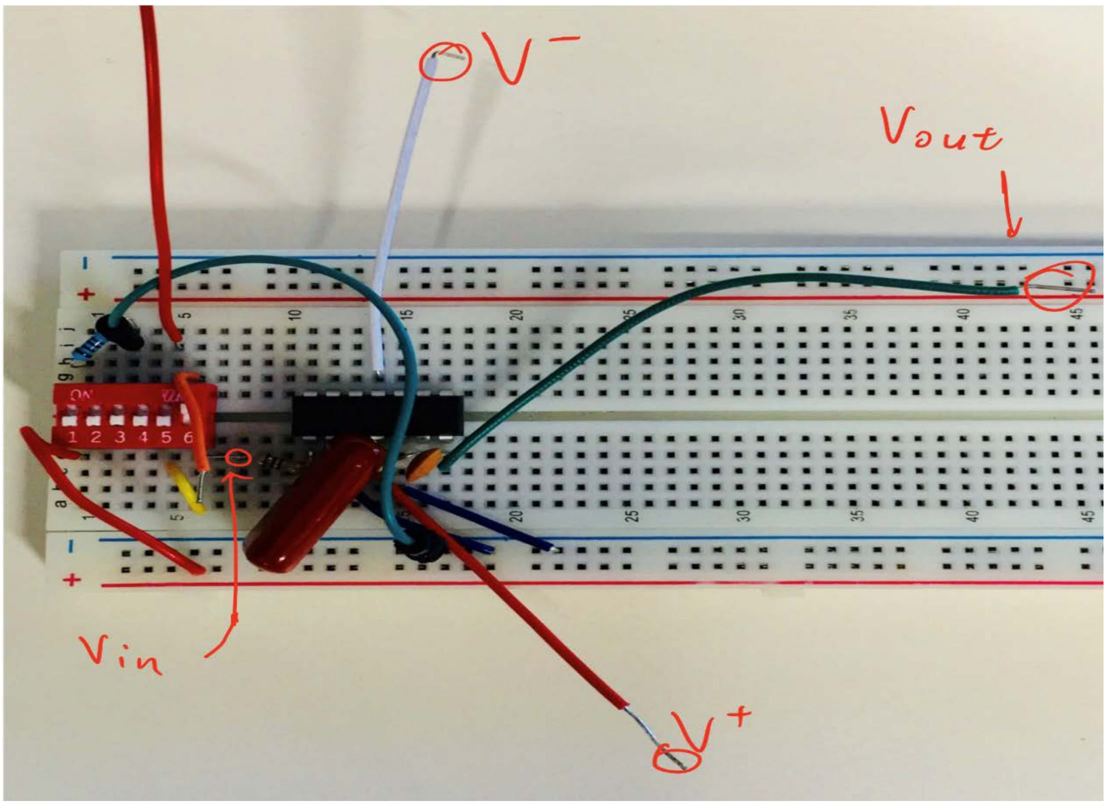


Fig 2.9 Modeling motor circuit

Given V+ = 12V, V- = -12V. Vin is input from DC power supply, Vout is sampled by Arduino and then scaled and transmitted by series port for observation. One advantage to observe from Arduino series plotter is that it can plot more past data and show them on bigger screen. While oscillator screen is quite small and may not be able to observe all the past data.

Gradually increase input voltage to 1.5V, and then turn to "0". Output is as below

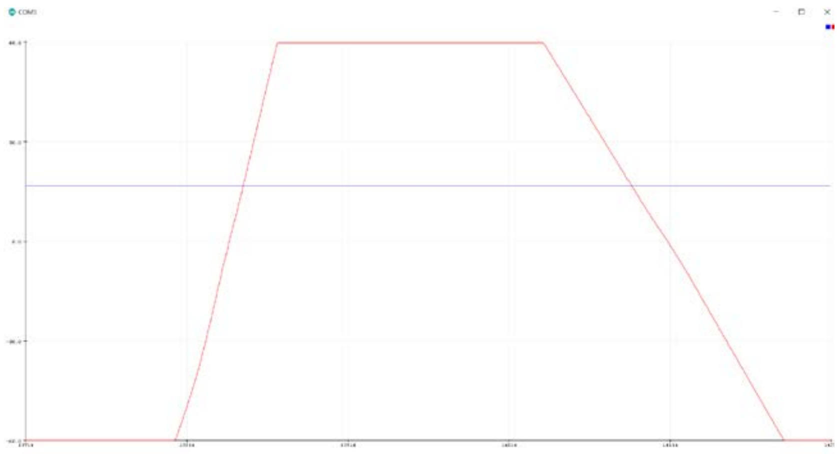


Fig 2.10 modeling motor circuit test

From Fig 2.10, the discharging process is slower than charging process. "0" input can cause output decreases, which means PWM input can dynamically control the output.

Date: 06/10/2017, Friday

Today's log summary:

- Code Arduino for PID control
- Tune P, I, D parameters to meet requirement

Coding

Coding becomes much easier when there is an Arduino PID library online. Several functions will be explained for understanding how the codes work.

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

//create a PID controller (C++ class) linked to the specified Input, Output, and Setpoint. Where "Direction" tells whether to subtract (reverse) or add (direct) when an error is given instead of being calculated.

SetSampleTime()

//tell PID controller how often to correction signal be calculated. Unit is millisecond. This is done (in *Compute()*) not by an interrupt function but by judging time change. If it is larger than sampling time, then execute calculation or do nothing.

SetMode()

//tell PID controller to start (Automatic) or stop (Manual).

Compute()

//Calculate correction signal using PID algorithm. Until time change is larger than sampling time set by *SetSampleTime(),* output will be calculated.

Once the output from PID controller is available. It will be used to control the duty of PWM by function *analogWrite(pin, value).*

Another coding issue is about how to represent output to angle from -60° to 60°. There are 16 10-bit ADC on Mega2560 but they can only measure voltage from 0 to 5V. Measurement can be done by *AnalogRead()* function, whose return value is from 0 - 1023. Thus, to map 0 - 1024 to -60 -60, we have

$$\text{valueMapped} = \frac{120}{1024} \times (value - 511) \qquad (3\text{-}1)$$

Both setpoint value from DAQ and measured output from modeling motor circuit will be scaled by (3-1). Codes in Arduino is as:

```
Setpoint_scale = 0.1171875*(Setpoint - 522);
 angle_out = 0.1171875*(Input-512);
```

Tuning

Although PID algorithms is quite simple, tuning the PID parameters is quite tricky. k_p influences system's response – the bigger, the faster. While if k_p is too big, proportional control will create unacceptable overshoot, collapsing system stability. k_i functions to determine system's steady-state-error. If control accuracy is the control system's major concern, k_i should be adjusted bigger. But too big k_i tends to lead to "integral saturation", causing too much overshoot. k_d is to reduce over shoot or down shoot, making system mores stable. It works like "estimating" error tendency and correct it in advance. As a trade-off, it will slow down system response, decreasing setting time. In all, there is balance between the three parameters.

One solution of tuning is an experimental method. Its step is as follow:

1. Set k_i and k_p to 0 and gradually increase k_d.
2. until the system oscillates, reduce k_d by a factor of 2-4.
3. Make k_p to be approximately 1% of k_d.
4. Gradually increase k_p till oscillate.
5. Attenuate k_p by a factor of 2-4.
6. Set k_i to approximately 1% of k_p.
7. Increase k_i till oscillate
8. Reduce k_i by a factor of 2-4. [6]

Input reference signal is generated from DAQ. It generates a series of Repeating Sequence Stair signal [1 3 4 5 3 2 3 4] with sampling time equating to 3s. Jumping from 1 to 3, step of 2 can be mapped as angle jumping from 0 – 50°.

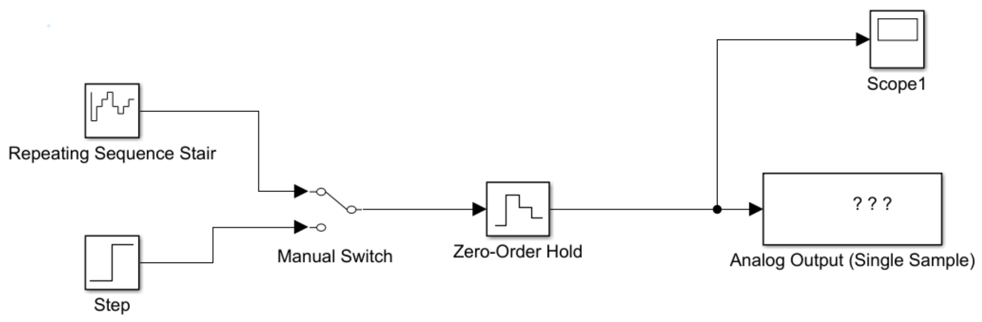


Fig 3.1 reference signal generator

Final tuned k_p, k_i, k_d are 0.08, 0.003, and 0.01 respectively and meet the requirement.

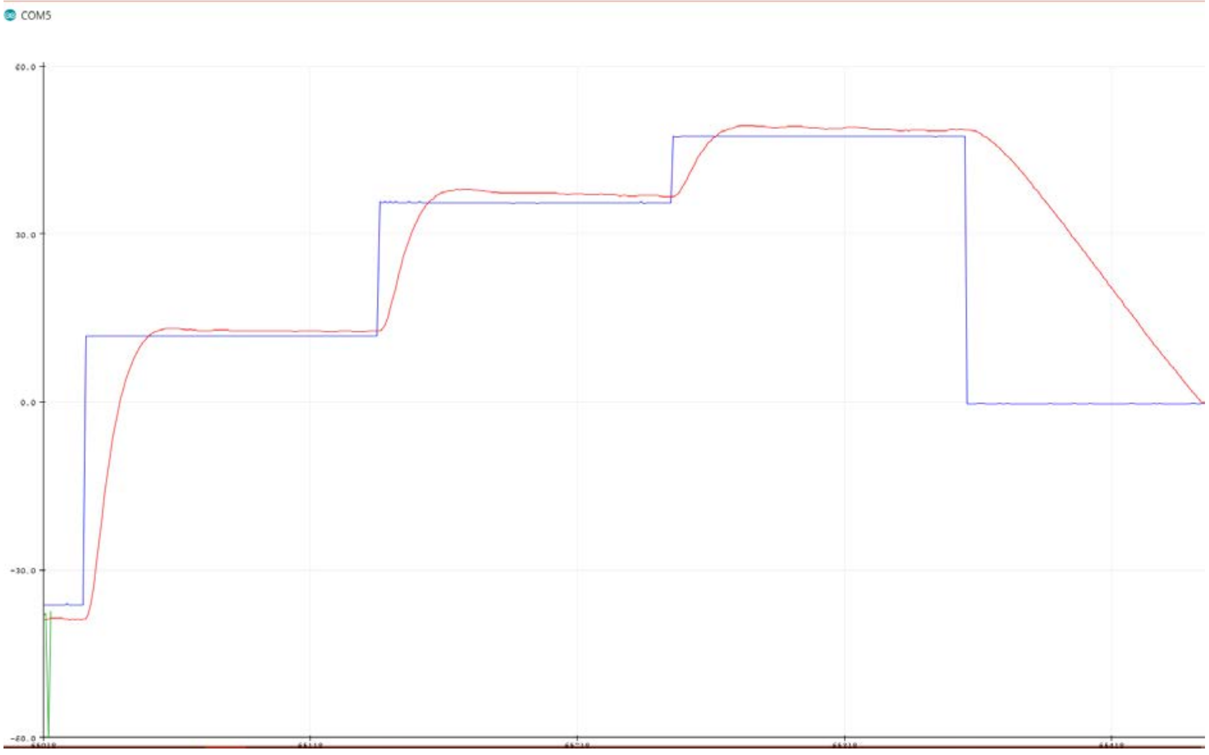


Fig 3.2 tracking results

Each stair lasts for 3s and the maximum step is 50° (first stair in the graph). Thus, we know, setting time is approximately 0.9s and overshoot 6%.

Date: 09/10/2017

Today's log summary:

- Reset function implementation
- Manual function implementation

A 6-bit switch Dip switch is put on the bread board for switching between "Automatic Model", "Manual Model" and "reset".

Rest function

A simple external circuit is created to generate "reset" signal, as shown in Fig 4.1.

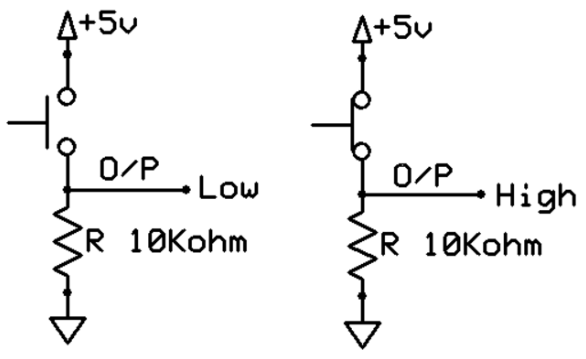


Fig 4.1 reset signal generating circuit

The push button can be replaced by a DIP switch. The reset port is scanned in the loop. In an if statement, once HIGH signal is scanned, it will set the setpoint to zero.

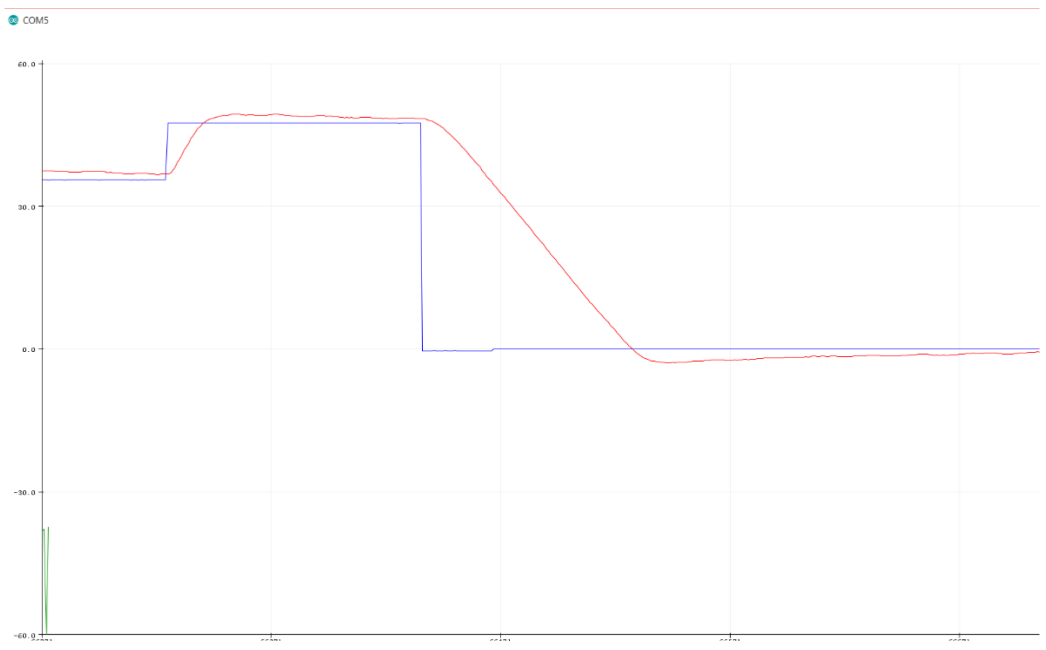


Fig 4.2 Rest signal testing.

Manual model

Two ways to understand manual model:

1. Manually set a new setpoint. The feedback loop still exists, and motor will track the new setpoint.
2. Manually control the motor. The feedback loop will be broken, and take control of the motor input

The first solution is extremely easy – in Fig 3.1, put the switch to the Step source side and tap in a new value. Then the “motor” will track the new setpoint.

The second one is a little bit tricky but not that complex. Solution is to measure a changeable voltage input from DC resource and then scale the input voltage to 0-255. Then the scaled value is then used to change the PWM duty. However, mapping input (0-1023) to 0-255 by scaler 4.0118 is not desirable. Just a small voltage will cause modeling motor output rise sharply. By trial and error, 0.005 is desirable.

*MANU_OUTPUT = 0.005 * analogRead(MANU_INPUT);*

In manual model, PWM is output from a port different from that in auto model.

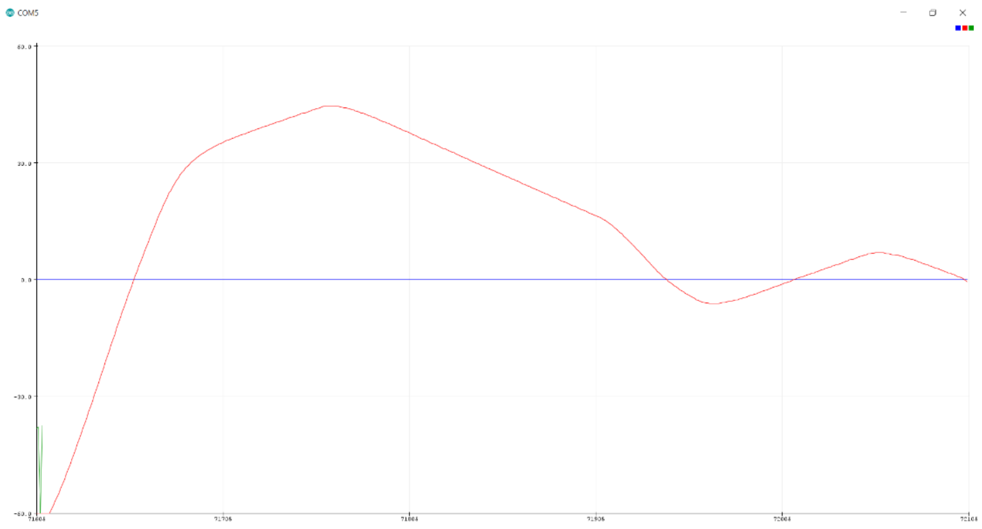


Fig 4.3 manual control in manual model

Date: 09/10/2017

Today’s log summary:

- Summary: outcome and lessons
- Future improvement

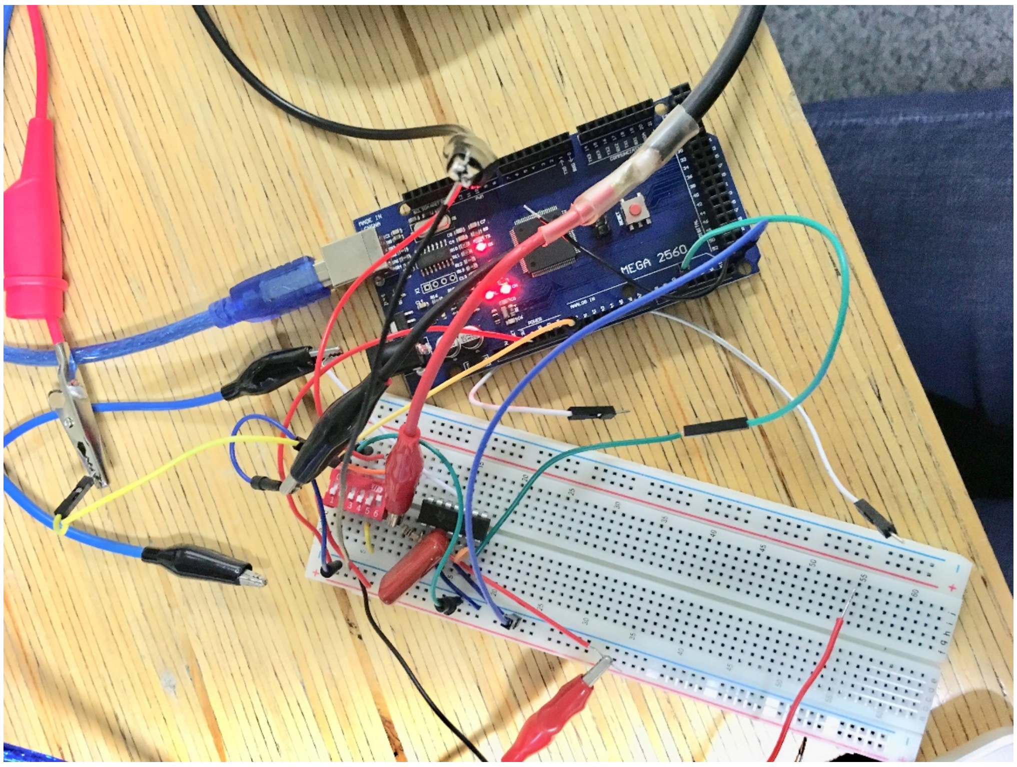


Fig 5.1 final work

Lessons:

- using library can accelerate development procedure.
- Simulink simulation can not 100% guarantee the result, but can tell a general map. So, more work need to be done in hardware testing.
- Do not weld a circuit until there are plenty of components or quite confident on the design, otherwise it will waste lots of time.
- Keep components connected solidly and layout well-organized can save much time on hardware debug.

Future improvement:

- Weld the circuit after all work.
- Use H-bridge to create inverse voltage to accelerate discharging period. It also isolates motor power supply from Arduino board.

Total cost

Name	Quantity	Price/AUD	Total/AUD
Mega2560 board with cable (bought in China)	1	6	6
Dip switch	1	2	2
Bread board	1	2.5	2.5
			13.5

Reference

[1] Prof Yon-Ping Chen, “Dynamic System Simulation and Implementation”, NCTU Department of Electrical and Computer Engineering 2015 Spring Course
[2] Operational Amplifiers Summary, http://www.electronics-tutorials.ws/
[3] Solver Configuration, https://www.mathworks.com/
[4] PID controller, https://en.wikipedia.org
[5] LMx24-N, LM2902-N Low-Power, Quad-Operational Amplifiers, http://www.ti.comt
[6] What are good strategies for tuning PID loops? https://robotics.stackexchange.com
[7] Push Button Switch Interfacing with Arduino UNO, https://www.maxphi.com