THE UNIVERSITY OF SYDNEY

School of Electrical and Information Engineering

STUDENT PLAGIARISM: COURSEWORK – POLICY AND PROCEDURE

COMPLIANCE STATEMENT

LABORATORY WORK

I certify that:

(1) I have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure*;

(2) I understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

(3) the Work that I record in my logbook or have marked will be solely the work of my lab partner and myself. All measurements, calculations, and other Work will be done during the assigned laboratory session unless otherwise noted. The Work will not include any material previously submitted for credit in any unit of study.

Name(s): Qizhang Li, Rijul Gupta, Lisa Mobilia

Signature(s): Qizhang Li Rijul

Date: 2017/6/4

Unit of Study for which Work will be submitted: ELEC 3204 project

ELEC3204 Project – The Spin Chamber

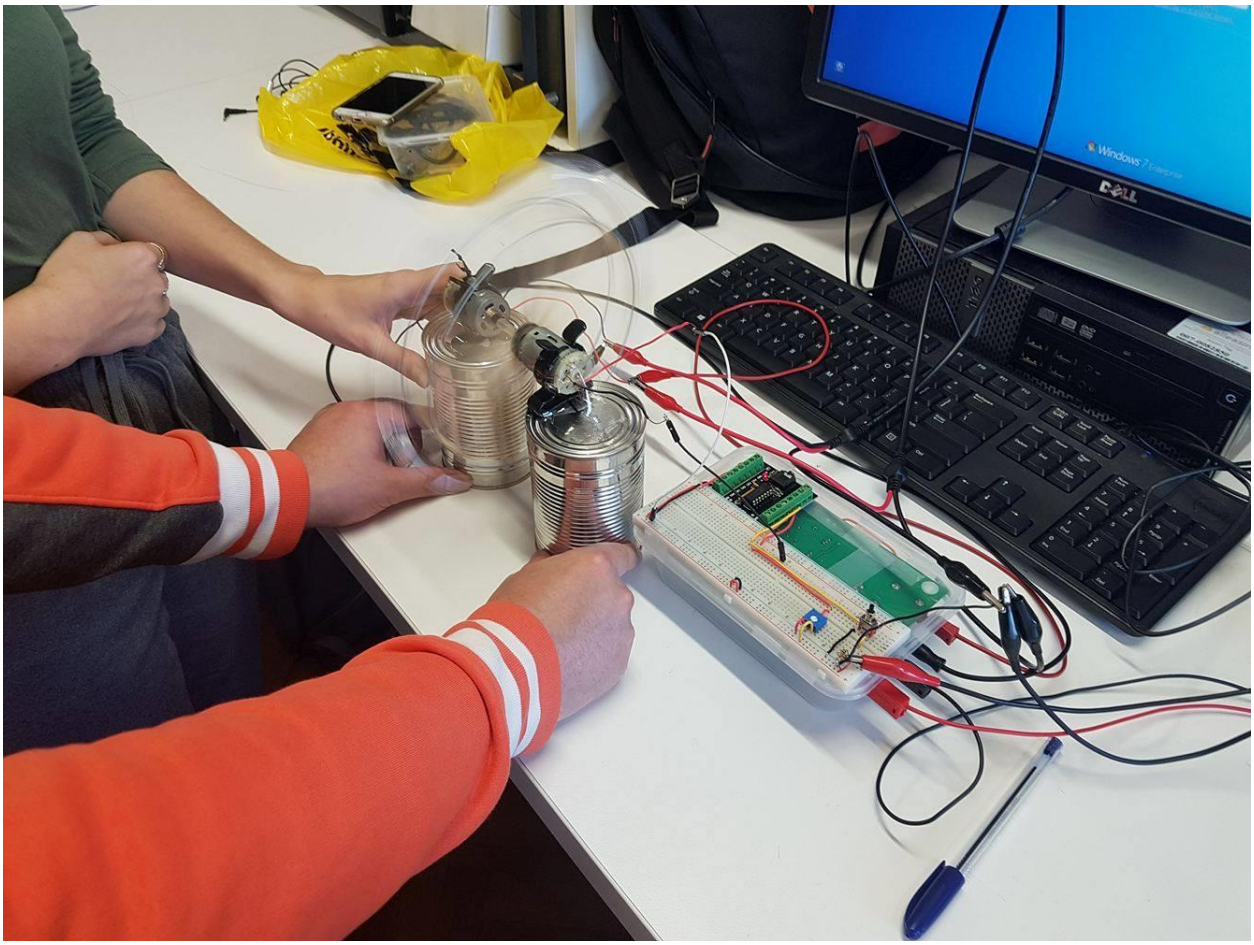
Group 3
Lisa Mobilia (470382152)
Qizhang Li (460452436)
Rijul Gupta (470230990)

Contribution

All the team members contributed equally in the assembly of the hardware as well as configuring, debugging and testing the software. We met at different occasions to discuss the plan for phases of the project as well as to work on it.

Project Description

Our implementation is based on the Human Centrifuge used by NASA to simulate high g environments for astronaut training. A person sitting in the chamber would experience a force proportional to the square of the speed that the chamber is rotating in, thereby allowing the simulation of high g environments. Cosmologists have long predicted that when we go inhabit space, space colonies would be made of such chambers as they provide an artificial sense of gravity and helps to escape weightlessness.



In this project, we have made use of 2 DC motors set in a feedback loop to simulate a controlled speed spin chamber. To make the frame we have used wooden rods arranged radially on each side of the spur gear and connected them laterally via more rods. The frame is further reinforced with wire to provide a sturdy implementation of the frame. To simulate different loads, we have used small metal thimbles to represent the chambers, which can be attached and detached as required. The different weights provide the means necessary to simulate varying degrees of load to test the feedback control.

Hardware Specifications

Controls

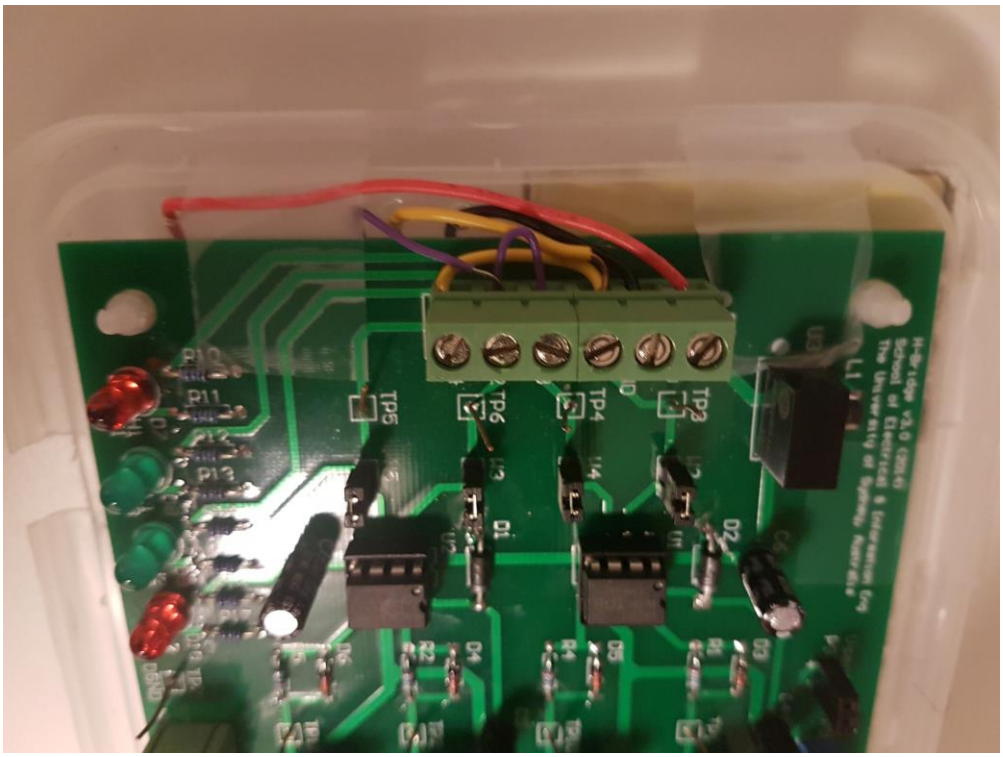
1. Main/Power switch – used to switch the whole device on/off.

   The Main Switch is driven at half the bus voltage by using a voltage divider to power its high port. This is done to ensure that the output voltage that is fed to ADC of PICAXE is always much less than 5 and hence it does not get damaged.

2. Mode switch – a potentiometer used to switch between different modes which force the system to operate at varying speeds through the PICAXE software.

Connections

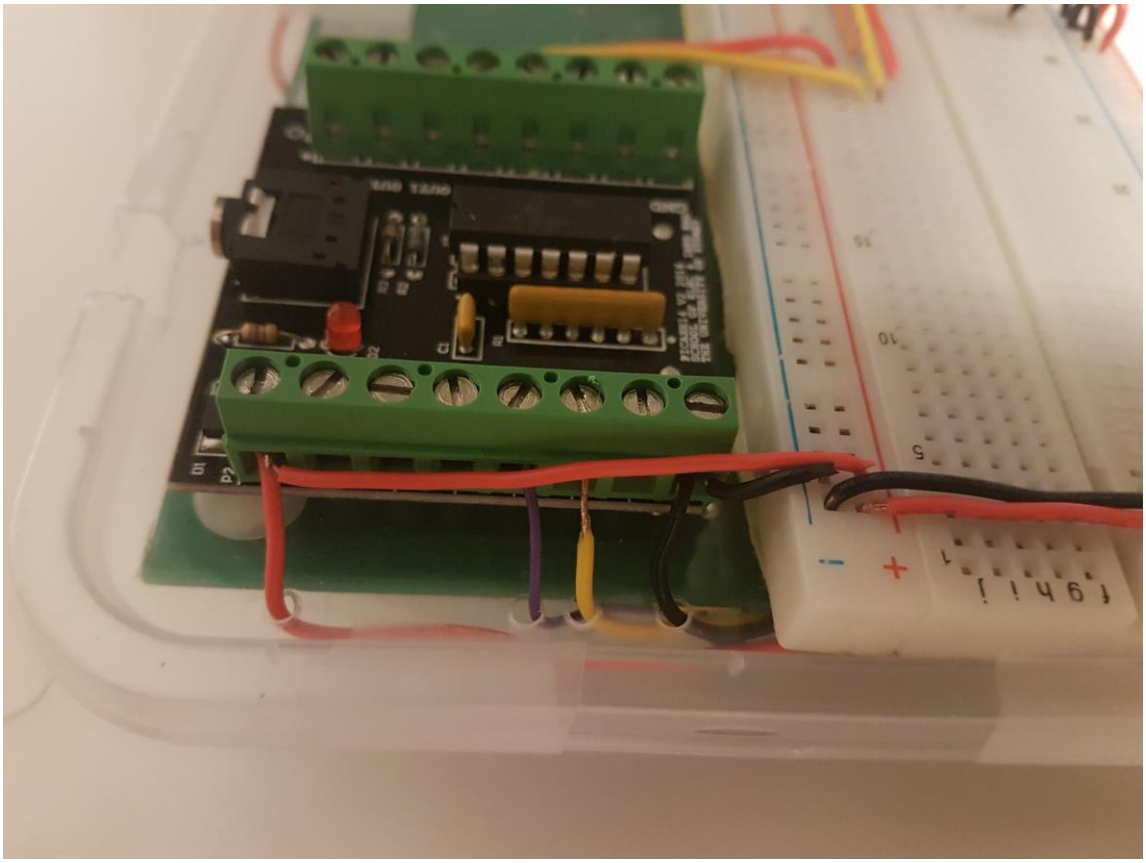
Connections on H Bridge



Color code:

1. Red: 5V
2. Black: Ground
3. Yellow: Q1, Q4
4. Purple: Q2, Q3

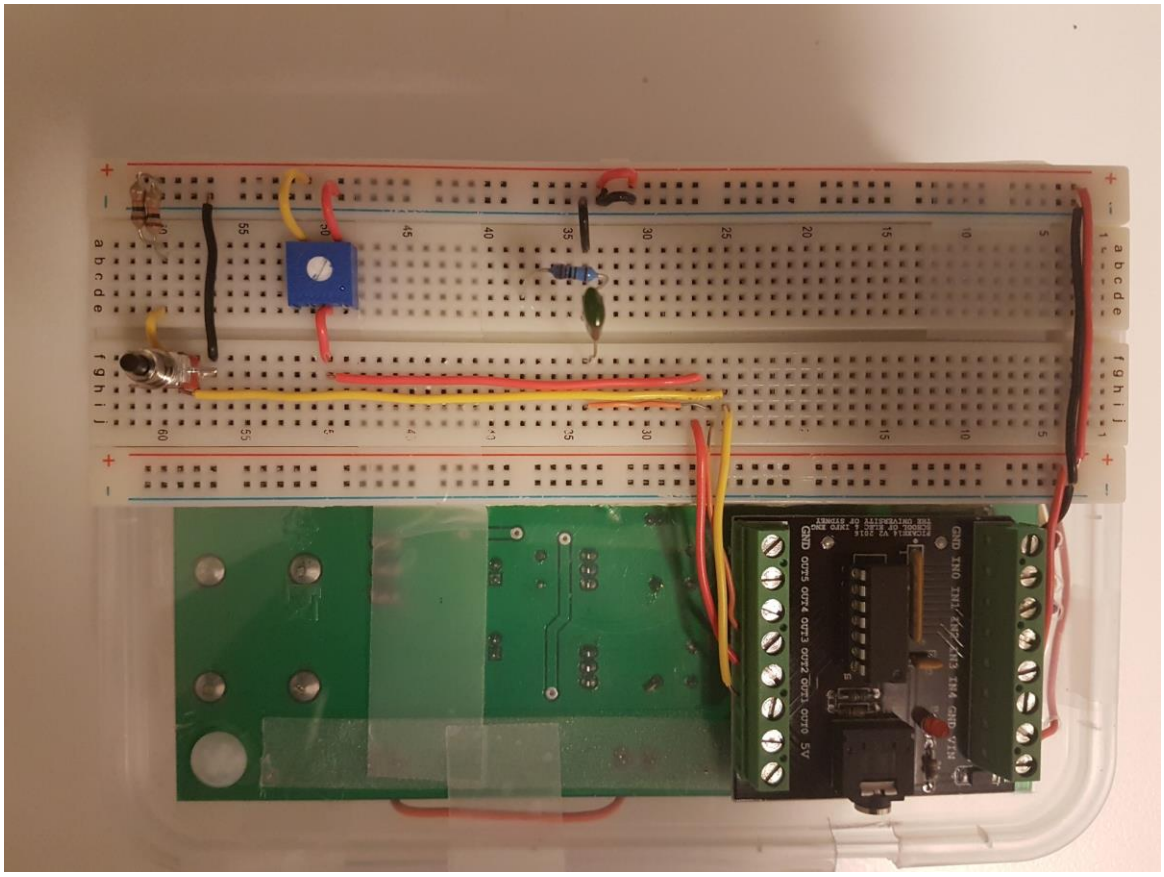
Connections on PICAXE



Color code:

1. Red: Vin
2. Black: Ground
3. Yellow: IN1
4. Purple: IN2

Connections on Breadboard and more on PICAXE



On PICAXE (Left port)

1. Orange – Feedback
2. Red – Mode Switch / Potentiometer
3. Yellow – Power Switch

Software Specifications

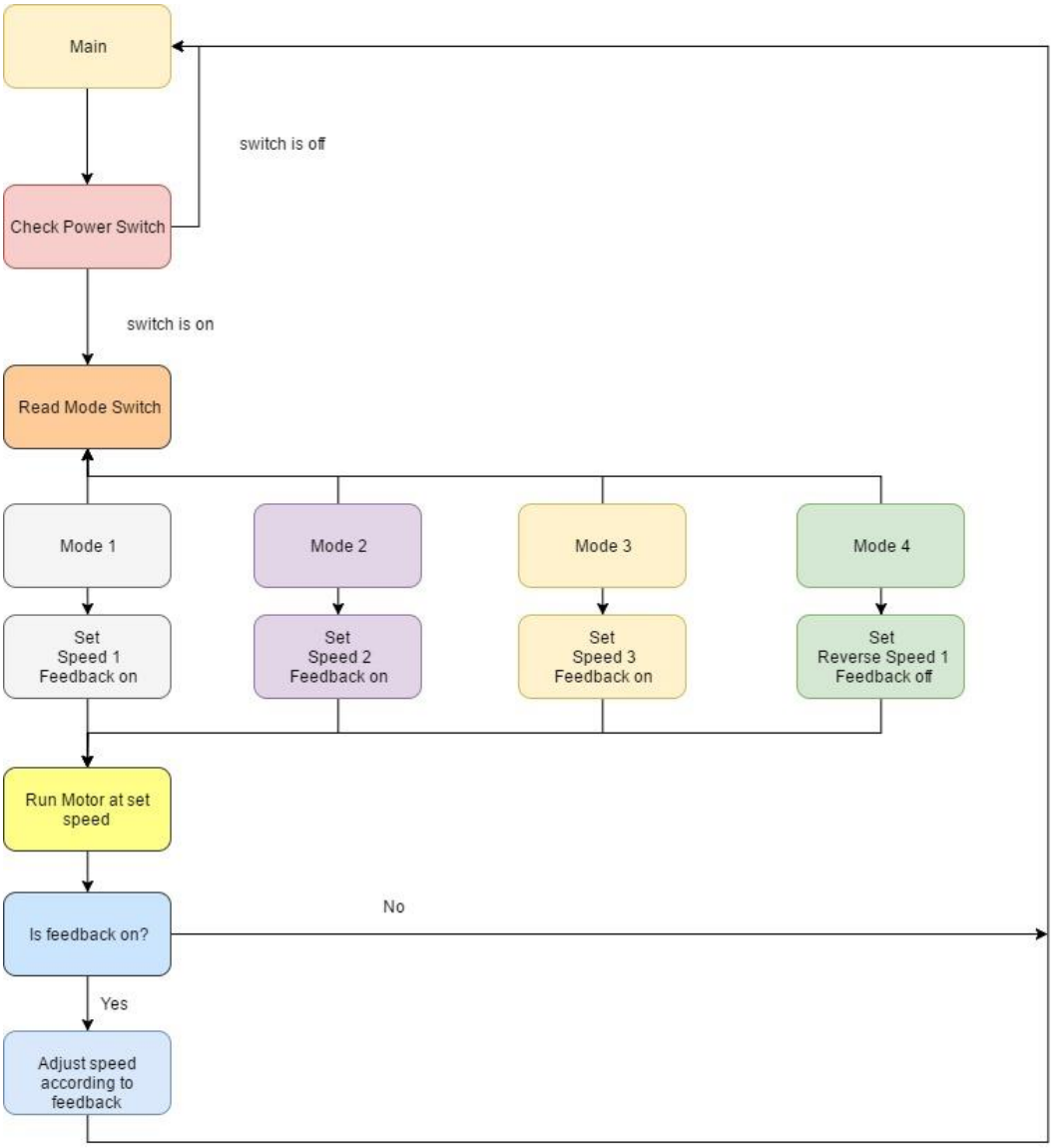
The main code is present in Appendix A, here we have included the base considerations and ideas which we made use of for writing the final code.

Considerations

1. **Noise**
   Since the feedback provided by the motor is extremely noisy even we have setup a noise threshold level under which we do not activate the feedback loop and assume that the motor is running at nearly the expected speed.

2. **Max/Min Speed**
   We do not want the feedback to reduce the duty cycle below a certain point if it thinks the speed is too much as that would make the motor stop or start running in the opposite direction. Also, we do not want the motor to run at a very high duty cycle as it may get damaged due to high voltage.
   Therefore, while increasing or decreasing the duty cycle in feedback we reduce or increase it only till a pre-specified limit.

Control Flow Diagram



Feedback Analysis

We expected the output voltage obtained by the second motor to be a function of the duty cycle set on the first motor

$$v_{out} = g(Duty)$$

And $Duty = g^{-1}(v_{out})$

The change in expected and obtained output voltage will relate to a change in duty cycle as

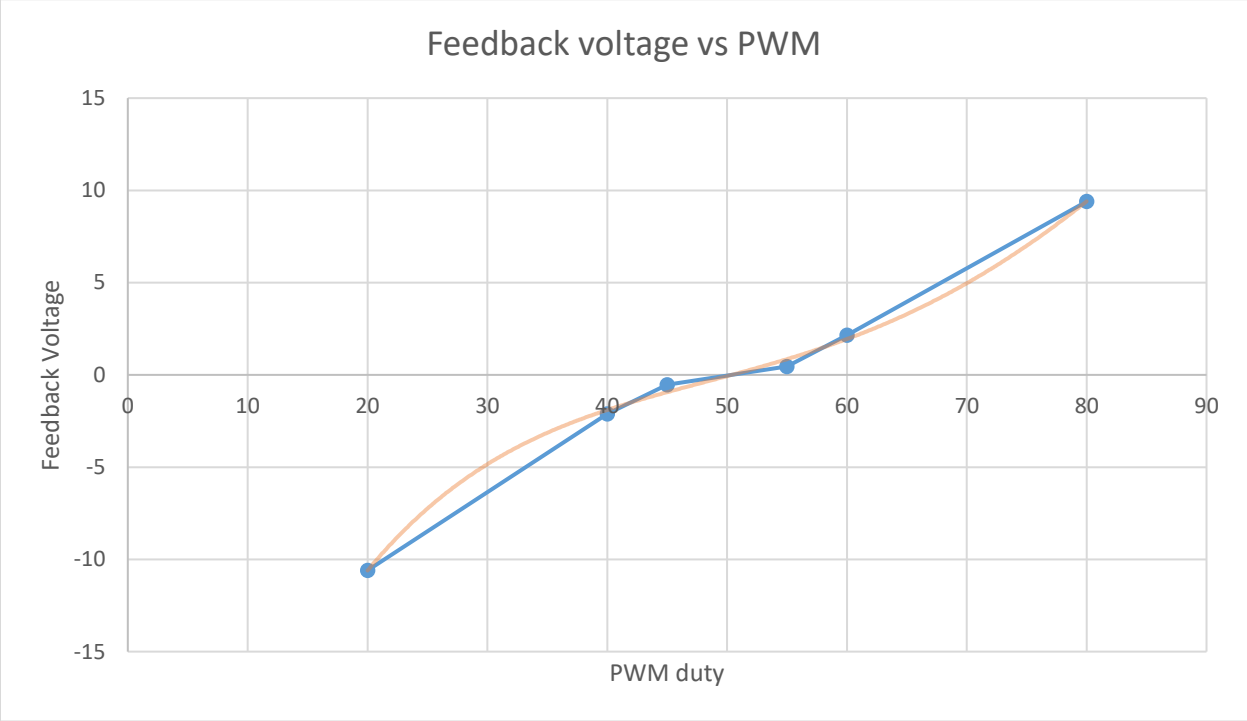
$$\Delta Duty = \Delta g^{-1}(v_{out})$$

So, we would set the duty cycle as

$$Duty_{final} = Duty_{desired} \pm \Delta Duty$$

The motor presented the following curve when we plotted the obtained feedback voltage with the supplied PWM duty cycle.

Even though the values followed the red curve more closely, we found that a piecewise linear model could be made use of for our specified speeds.



Following a piecewise linear model, we expect the output voltage to follow the following equation

$$v_{out} = m * Duty + c$$

This enables us to find the change in Duty required to be simply

$$\Delta Duty = \frac{1}{m} \Delta v_{out}$$

We chose the following speeds for our closed loop circuit:

1. 221 Duty Cycle
2. 211 Duty Cycle
3. 201 Duty Cycle

And for those values we found m as 0.24

Since PICAXE does not allow floating point arithmetic, we do the following

$$Duty_{final} = Duty_{desired} * 100$$

$$Duty_{final} = \frac{Duty_{final}}{24}$$

System performance

Minimum Speed

We observed that the motors can't run below a certain duty cycle for different loads, for that reason we had to modify the code so that the motors don't stop while demonstrating.

Feedback

The PWM output is altered by the PICAXE as expected from the feedback loop, under false feedback we observed that if the feedback is set to 0 the duty cycle is increased to the maximum permissible and if then the feedback is set to high (5V) then the duty cycle is decreased to minimum permissible.
In real cases, it tries to stabilize the output near the expected output and ends up oscillating around it increasing and decreasing the duty cycle as it tries to match expected and obtained results.

Noise

Noise is a major factor as even if the motors are running at the desired speed, they tend to send a spike of voltage which confuses the feedback loop.

Overall

Overall the system matches our expected outputs, it runs in both clockwise and counter clockwise directions in accordance with the modes. The feedback loop tries to match the real output with the expected output as closely as it can.

Appendix A (PICAXE Code)

```
#rem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Setting variables and properties
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#endrem
setfreq M32
symbol mainSwitch = b1
symbol modeSwitch = b2
symbol realVout = b3
symbol desiredFrequency = b4
symbol noiseLevel = b5
symbol feedbackFlag = b6
symbol expectedVout = b7
symbol diffVout = b8
symbol differenceFrequency = b9
symbol finalFrequency = b10
symbol mode = b11
symbol motorSetFlag = b12
symbol frequencyMin = b13
symbol frequencyMax = b14

mode=0
frequencyMin = 170
frequencyMax = 230

main:

	readadc B.1,b1
	readadc B.2,b2

	#rem
	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
	If switch is off
	set motor pwm at 50% => motor is off
	continue loop
	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
	#endrem
	if mainSwitch<50 then
	hpwm 1,0,0,79,159
	mode=0
	goto main
	endif

	gosub modeController

	gosub motor

goto main
```

```
#rem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
modeController
used for setting the mode to run the device in desired mode
sets feedbackflag to enable/disable feedback loopback mode
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#endrem
modeController:
      if modeSwitch < 60 then
            if mode != 1 then
                  sertxd("Mode 1",cr,lf)
                  desiredFrequency = 221
                  finalFrequency = desiredFrequency
                  mode = 1
                  feedbackFlag = 1
                  motorSetFlag = 0
            endif
      else if modeSwitch < 130 then
            if mode != 2 then
                  sertxd("Mode 2",cr,lf)
                  desiredFrequency = 211
                  finalFrequency = desiredFrequency
                  mode = 2
                  feedbackFlag = 1
                  motorSetFlag = 0
            endif
      else if modeSwitch < 220 then
            if mode != 3 then
                  sertxd("Mode 3",cr,lf)
                  desiredFrequency = 201
                  finalFrequency = desiredFrequency
                  mode = 3
                  feedbackFlag = 1
                  motorSetFlag = 0
            endif
      else
            if mode != 4 then
                  sertxd("Mode 4",cr,lf)
                  desiredFrequency = 120
                  finalFrequency = desiredFrequency
                  mode = 4
                  feedbackFlag = 0
                  motorSetFlag = 0
            endif
      endif

return




#rem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
motor
```

```
runs the motor at desired frequency
ensures speed if feedbackflag is set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#endrem
motor:

      if motorSetFlag = 0 then
            sertxd("final frequency for motor is ",finalFrequency,cr,lf)
            hpwm 1,0,0,79,finalFrequency
            motorSetFlag = 1
            if feedbackFlag=1 then
                        gosub feedback
                        motorSetFlag = 0
            endif
      endif
return


#rem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
feedback
makes sure that the motor is running
at desired speed within noise level
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#endrem
feedback:

      readadc B.3,b3

      noiseLevel = 3

      if mode = 1 then
            expectedVout = 20
      else if mode = 2 then
            expectedVout = 18
      else
            expectedVout = 15
      endif


      sertxd("realVout is ",realVout,cr,lf)


      if realVout > expectedVout then
            diffVout = realVout - expectedVout

            sertxd("diffVout is ",diffVout,cr,lf)
            if diffVout < noiseLevel then
                  return
            else
                  differenceFrequency = 100 * diffVout
                  differenceFrequency = differenceFrequency / 24
                  finalFrequency = finalFrequency - differenceFrequency
                  if finalFrequency < frequencyMin then
                        finalFrequency = frequencyMin
                  endif
            endif
```

```
	else
		sertxd("Diff vout is negative",cr,lf)
		diffVout = expectedVout - realVout
		sertxd("diffVout is ",diffVout,cr,lf)
		if diffVout < noiseLevel then
			return
		else
			differenceFrequency = 100 * diffVout
			differenceFrequency = differenceFrequency / 24
			finalFrequency = finalFrequency + differenceFrequency
			if finalFrequency > frequencyMax then
				finalFrequency = frequencyMax
			endif
		endif
	endif

	sertxd("desired frequency is ",desiredFrequency,cr,lf)
	sertxd("diffFrequency is ",differenceFrequency,cr,lf)
	sertxd("final frequency is ",finalFrequency,cr,lf)

return
```