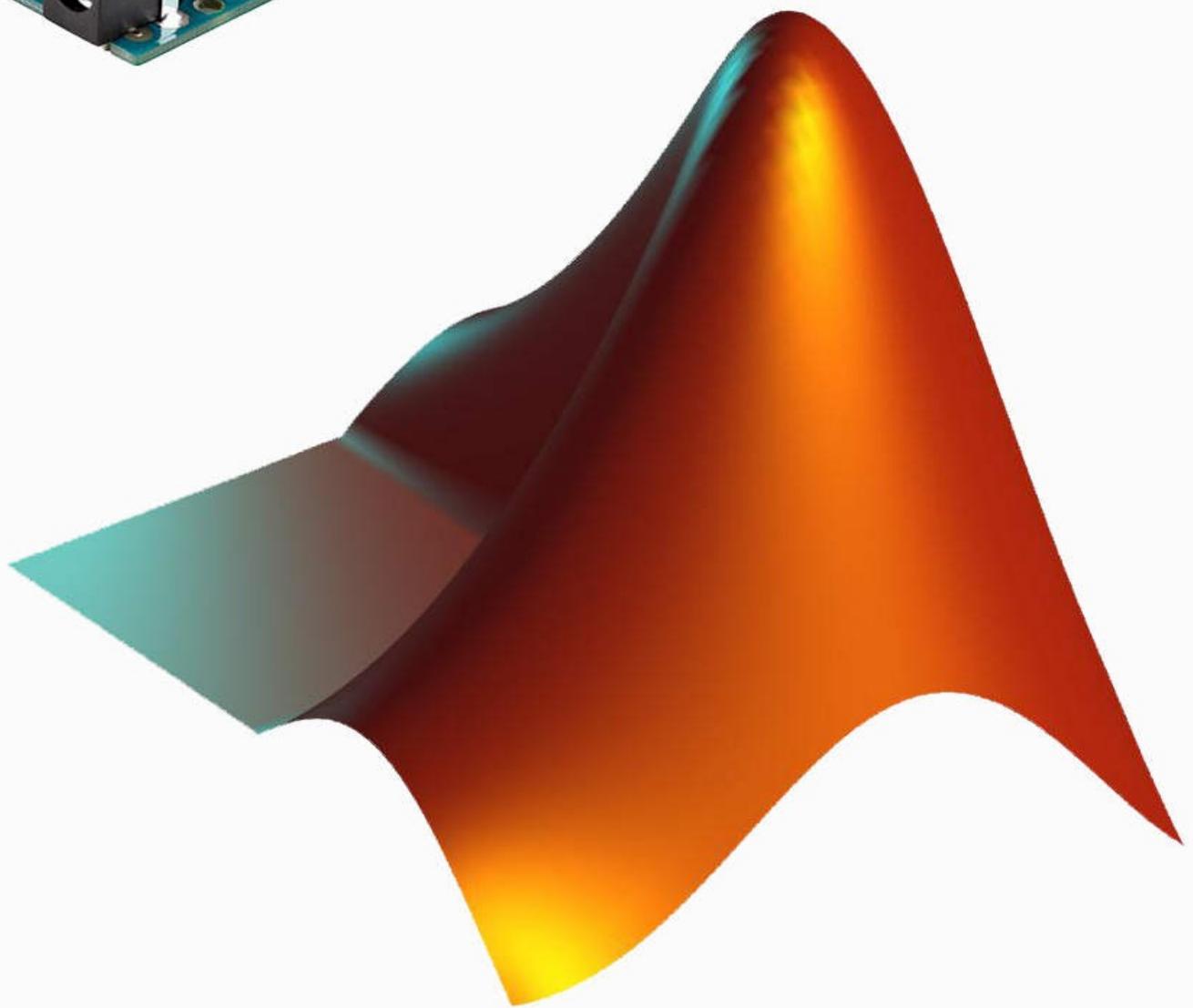


Arduino Programming using MATLAB



Agus Kurniawan

Copyright

Arduino Programming using MATLAB

Agus Kurniawan

1st Edition, 2015

Copyright © 2015 Agus Kurniawan

Table of Contents

[Copyright](#)

[Preface](#)

[1. Preparing Development Environment](#)

[1.1 Arduino](#)

[1.1.1 Arduino Uno](#)

[1.1.2 Arduino Leonardo](#)

[1.1.3 Arduino Mega 2560](#)

[1.1.4 Arduino Due](#)

[1.2 Electronic Components](#)

[1.2.1 Arduino Starter Kit](#)

[1.2.2 Fritzing](#)

[1.2.3 Cooking-Hacks: Arduino Starter Kit](#)

[1.2.4 Arduino Sidekick Basic kit](#)

[1.3 Matlab](#)

[1.4 Testing](#)

[2. Setting Arduino Development for MATLAB](#)

[2.1 Getting Started](#)

[2.2 Setting up Arduino Development for MATLAB](#)

[2.3 Connecting Arduino Board to Computer](#)

[2.4 Hello Arduino: Blinking LED](#)

[3. Working with Digital I/O](#)

3.1 Getting Started

3.2 Demo : LED and Pushbutton

3.2.1 Wiring

3.2.2 Writing a Program

3.2.3 Testing

4. Working with PWM and Analog Input

4.1 Getting Started

4.2 Demo Analog Output (PWM) : RGB LED

4.2.1 Wiring

4.2.2 Writing Program

4.2.3 Testing

4.3 Demo Analog Output Voltage: LED Brightness

4.3.1 Wiring

4.3.2 Writing a Program

4.3.3 Testing

4.4 Demo Analog Input: Working with Potentiometer

4.4.1 Wiring

4.4.2 Writing Program

4.4.3 Testing

5. Working with I2C

5.1 Getting Started

5.2 Writing Program

5.3 Demo 1: Scanning I2C

5.4 Demo 2: Reading Data from Sensor Based I2C

6. Working with SPI

6.1 Getting Started

6.2 Demo : SPI Loopback

7. Working with Servo Motor

7.1 Getting Started

7.2 Wiring

7.3 Writing a Matlab Program

7.4 Testing

8. Measuring and Plotting Sensor Data in Real-Time

8.1 Getting Started

8.2 Wiring

8.3 Writing a Program

8.4 Testing

Source Code

Contact

Preface

This book was written to help anyone want to develop Arduino board using MATLAB with Arduino supported. It describes the basic elements of Arduino development using MATLAB.

Agus Kurniawan

Depok, September 2015

1. Preparing Development Environment

1.1 Arduino

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. This board uses Atmel microcontroller series. There are many Arduino hardware models that you can use. Further information about Arduino products, you can visit on website <http://arduino.cc/en/> .

You must one Arduino hardware to follow practices in this book. I recommend to obtain one of the following Arduino hardware:

- Arduino Uno
- Arduino Leonardo
- Arduino Mega 2560
- Arduino Due

You can buy this product on your local electronic store. You also can order it by online. Find it on <http://arduino.cc/en/Main/Buy>. The following is the list of Arduino store you can buy

- Arduino store, <http://store.arduino.cc/>
- Amazon, <http://www.amazon.com>
- Cooking-hacks, <http://www.cooking-hacks.com/index.php/shop/arduino.html>
- RS Components, <http://www.rs-components.com>
- Element 14, <http://www.element14.com>
- EXP-Tech, <http://www.exp-tech.de>

Because Arduino is an open-source hardware, people can build it. It's called Arduino compatible. Generally it's sold in low prices.

1.1.1 Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328. You can download the datasheet file, http://www.atmel.com/dyn/resources/prod_documents/doc8161.pdf .

Further information about Arduino Uno, you can read it on <http://arduino.cc/en/Main/ArduinoBoardUno> .

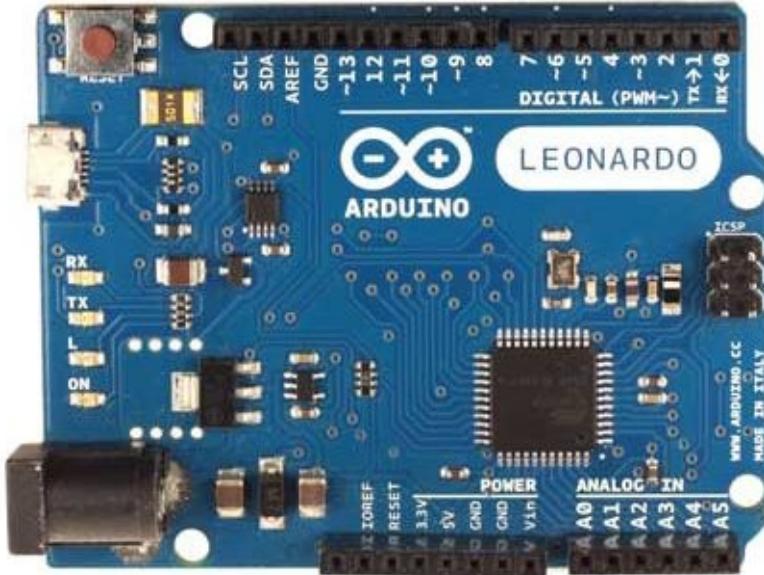


1.1.2 Arduino Leonardo

The Arduino Leonardo is a microcontroller board based on the ATmega32u4. Download datasheet for this product on

http://www.atmel.com/dyn/resources/prod_documents/7766S.pdf .

Visit this product to get the further information on
<http://arduino.cc/en/Main/ArduinoBoardLeonardo> .



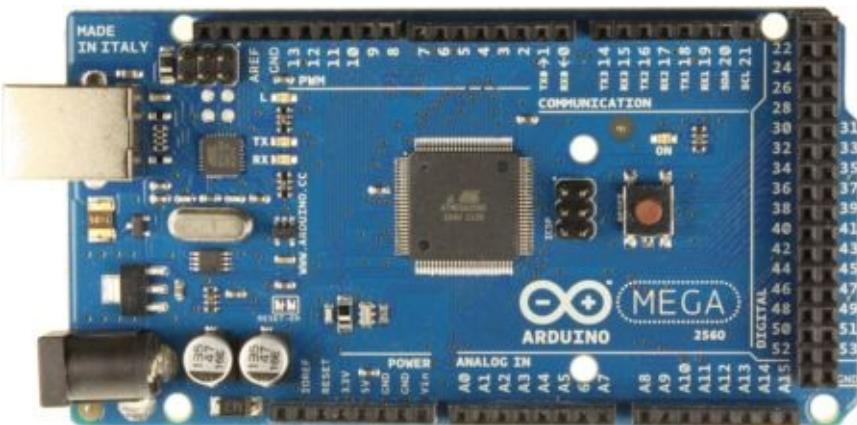
1.1.3 Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. You can download the datasheet file on

http://www.atmel.com/dyn/resources/prod_documents/doc2549.PDF.

Further information about Arduino Mega 2560, you can visit on

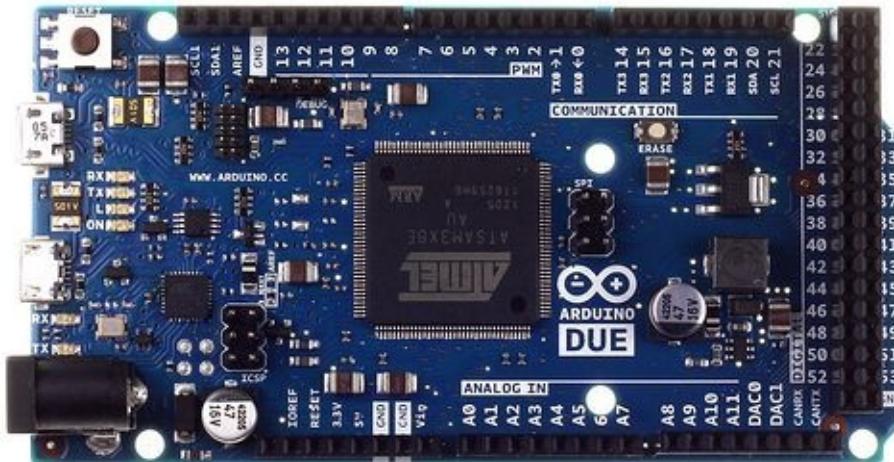
<http://arduino.cc/en/Main/ArduinoBoardMega2560> .



1.1.4 Arduino Due

The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. You can download the datasheet, <http://www.atmel.com/Images/doc11057.pdf>.

If you want to know about Arduino Due, I recommend to visit this website, <http://arduino.cc/en/Main/ArduinoBoardDue>.



1.2 Electronic Components

We need electronic components to build our testing, for instance, Resistor, LED, sensor devices and etc. I recommend you can buy electronic component kit.

1.2.1 Arduino Starter Kit

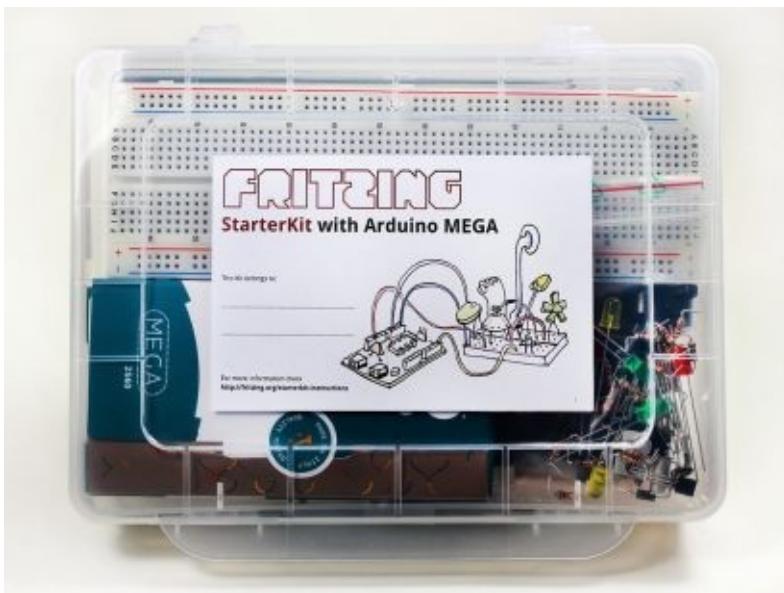
Store website: <http://arduino.cc/en/Main/ArduinoStarterKit>



1.2.2 Fritzing

Store website: <http://shop.fritzing.org/> .

You can buy Fritzing Starter Kit with Arduino UNO or Fritzing Starter Kit with Arduino Mega.



1.2.3 Cooking-Hacks: Arduino Starter Kit

Store website: <http://www.cooking-hacks.com/index.php/shop/arduino/starter-kits/arduino-starter-kit.html>



1.2.4 Arduino Sidekick Basic kit

Store website: <http://www.seeedstudio.com/depot/arduino-sidekick-basic-kit-p-775.html>

Alternative online store

<http://www.amazon.com/Arduino-Sidekick-Basic-Kit-Version/dp/B007B14HM8/>

<http://www.exp-tech.de/Zubehoer/Arduino-Sidekick-Basic-Kit.html>



1.3 Matlab

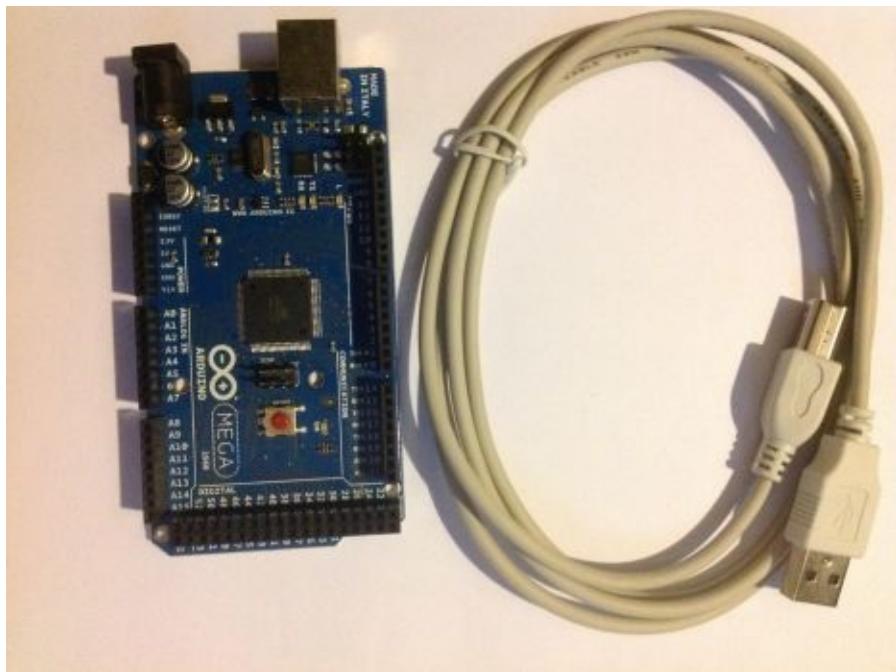
MATLAB Support Package for Arduino hardware enables you to use MATLAB® to communicate with the Arduino® board over a USB cable. This package is based on a server program running on the board, which listens to commands arriving via serial port, executes the commands, and, if needed, returns a result.

This support package is available for R2014a and later releases. It's available on 32-bit and 64-bit Microsoft® Windows®, 64-bit Mac OS, and 64-bit Linux®.

I will explain how to set up Matlab for Arduino development on chapter 2.

1.4 Testing

For testing, I used Arduino Uno R3 and Arduino Mega 2560 on OSX and Windows 10 platforms with Matlab 2015b.



I also used Arduino Sidekick Basic kit for electronic components.



2. Setting Arduino Development for MATLAB

This chapter explains how to work on setting up Arduino board on a computer and then, access it from MATLAB.

2.1 Getting Started

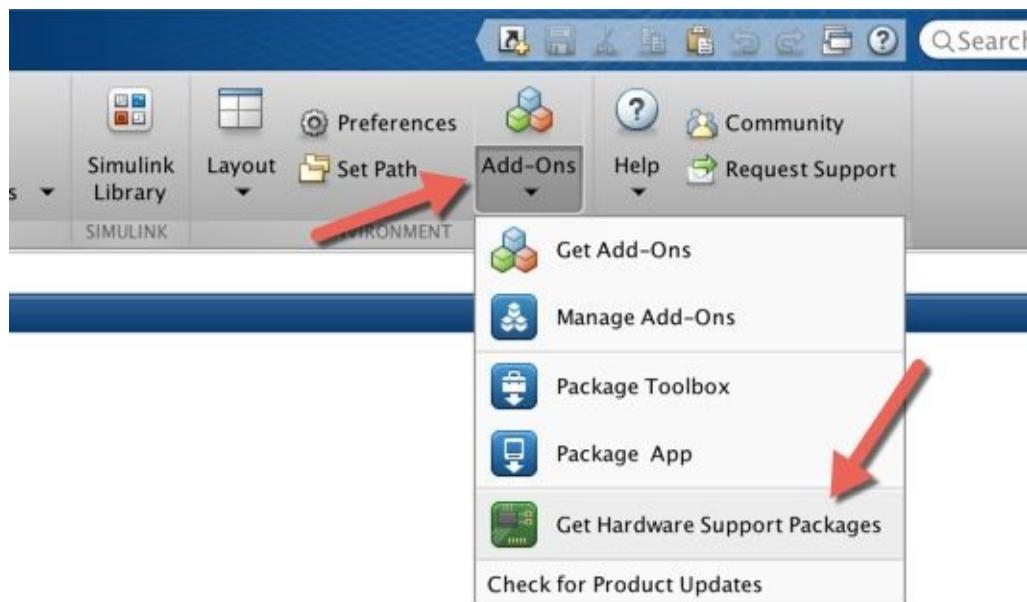
In this chapter, we set up Arduino board development using MATLAB support package for Arduino hardware. To set up this development, you must have MATLAB 2014a or later and MATLAB account to verify while installing.

2.2 Setting up Arduino Development for MATLAB

In this section, we try to set up Arduino development for MATLAB. You can configure MATLAB Support Package for Arduino hardware using MATLAB 2014a or later. We also need internet connection to download this package.

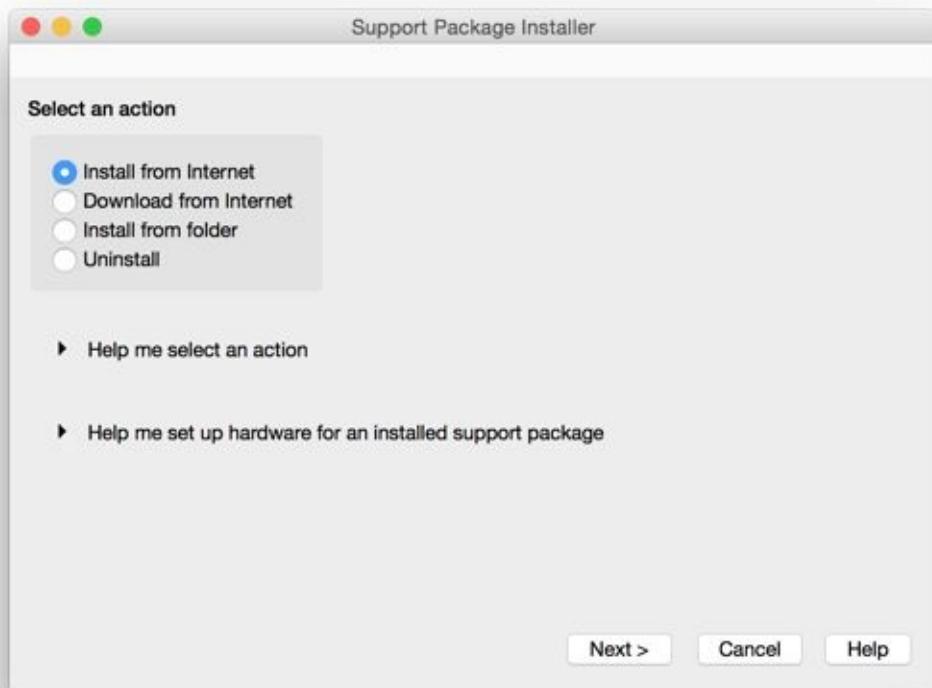
Let's start.

Run MATLAB application. Click **Get Hardware Support Packages** on Add-Ons icon on toolbox.



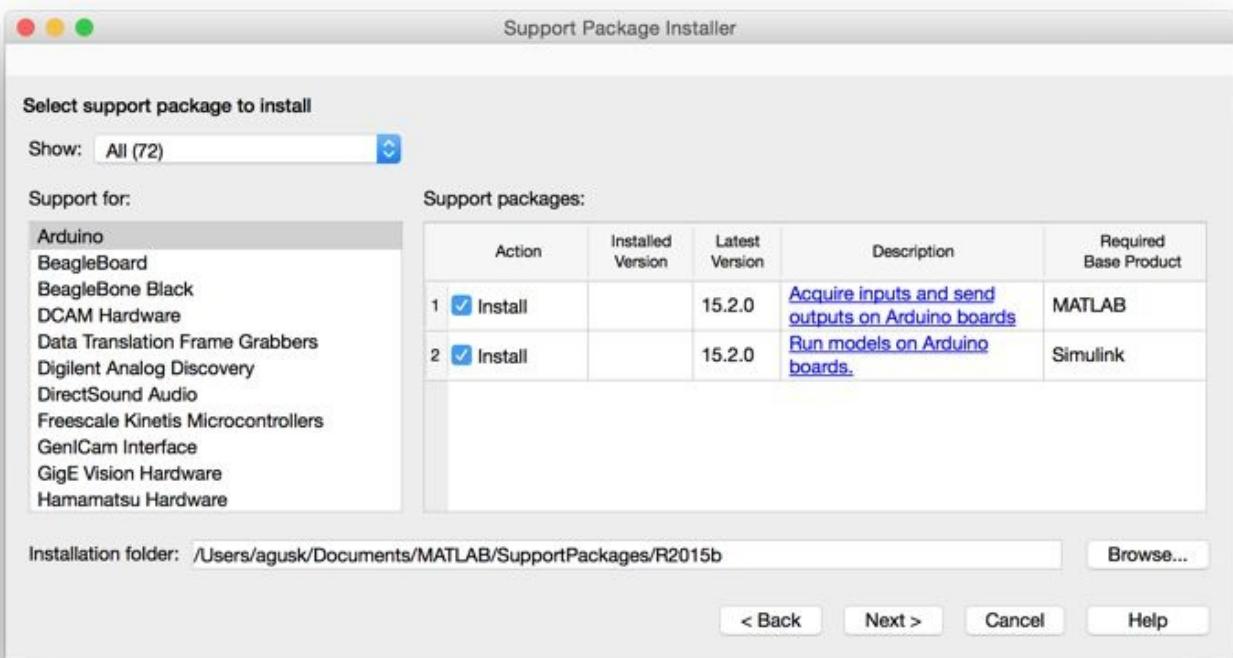
Then, you get a dialog. Select Install from Internet.

If done, click **Next>** button.



Select Arduino on Support for. You should see two Arduino support packages. You can select both.

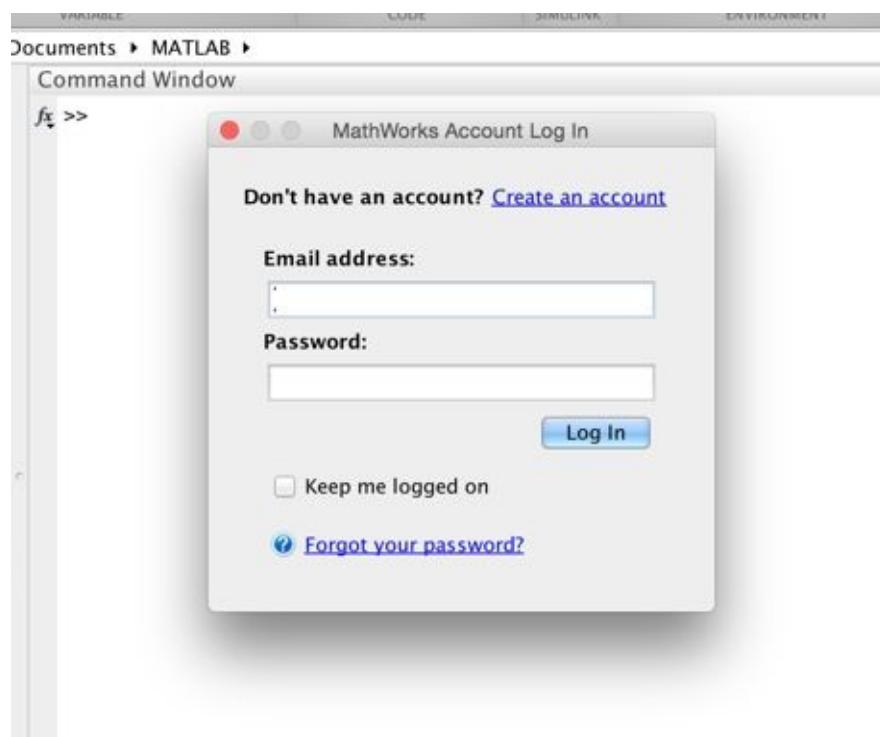
If done, click **Next >** button.



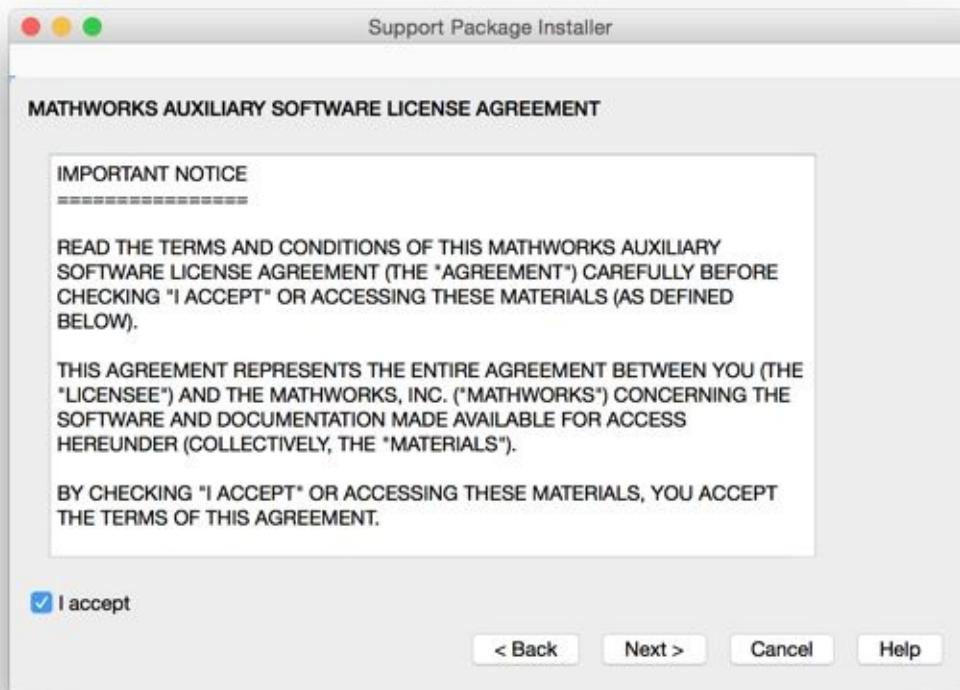
You will be asked to logon with your MATLAB account. You should have MATLAB license. Click **Log In** button.



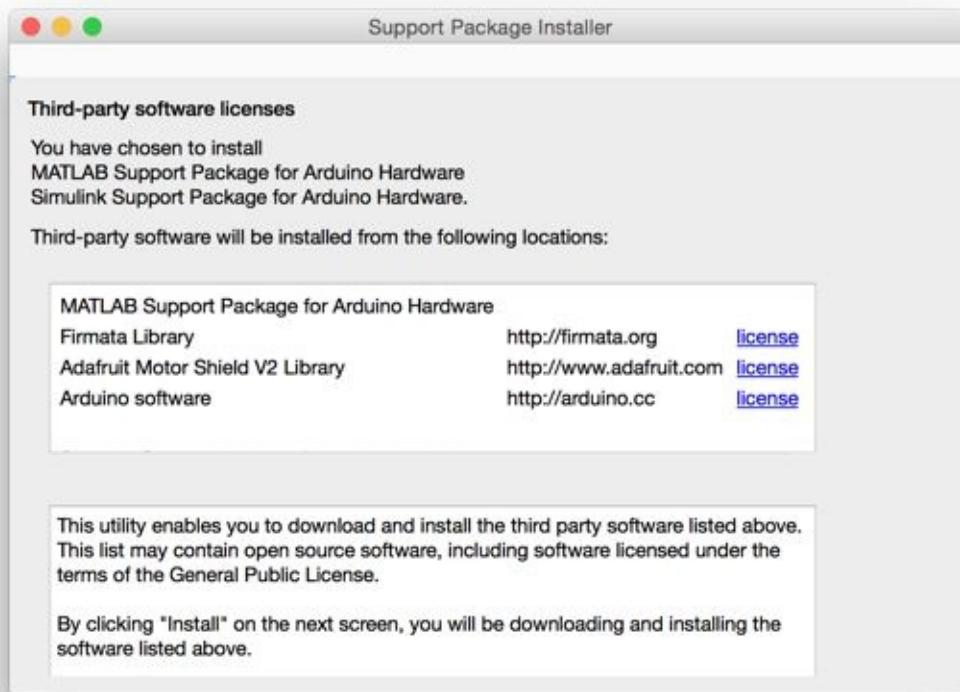
You should see the authentication dialog. Fill your account. After that, click **Log In** button.



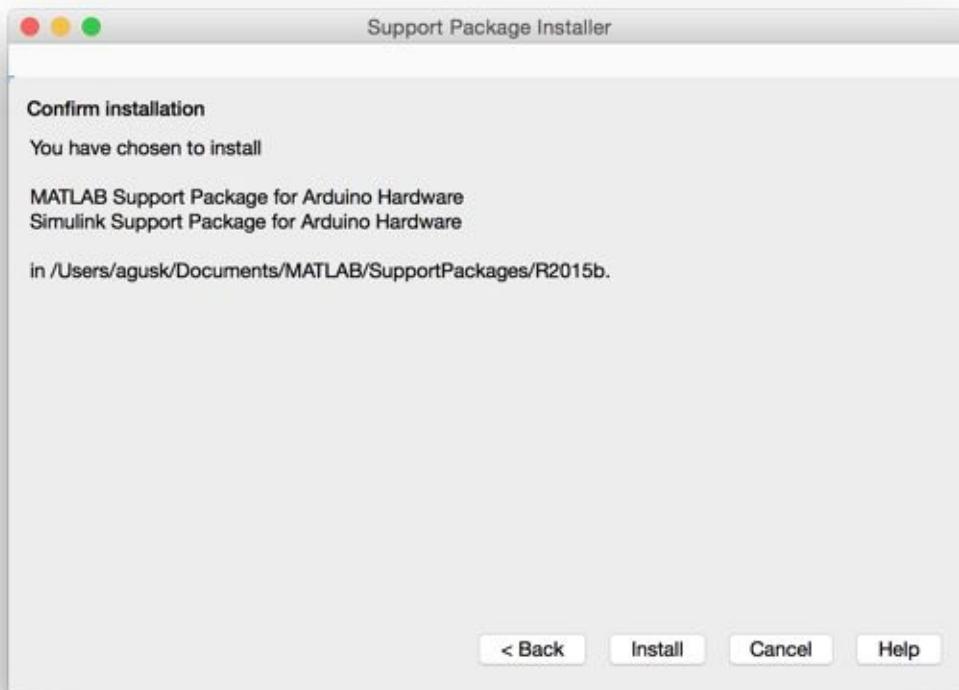
If success, you should get a software license agreement. Checked **I accept** and then click **Next>** button.



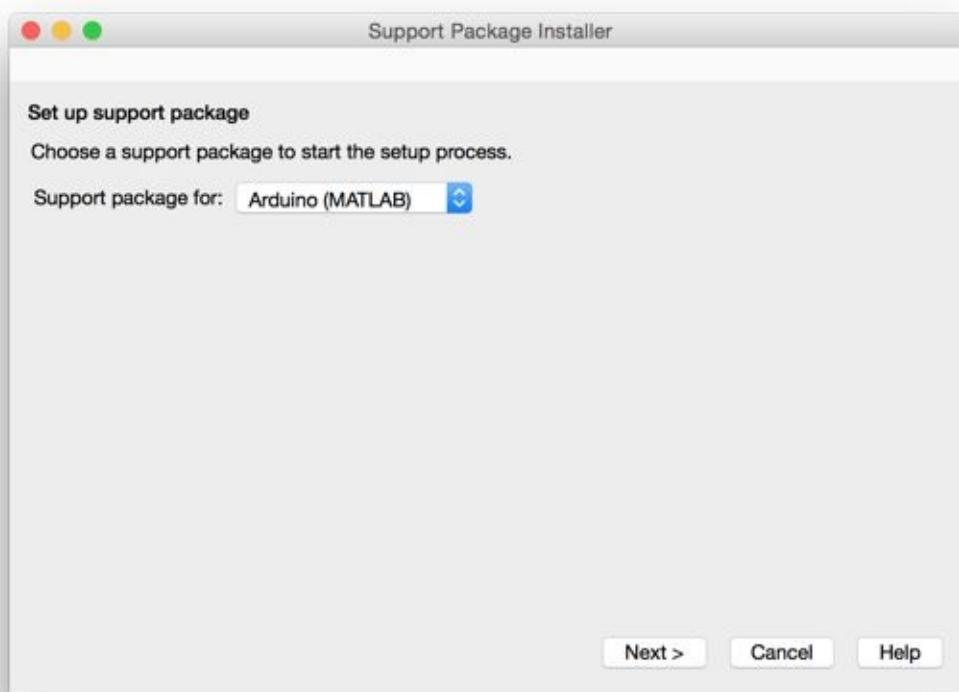
You will get confirmation. Click **Next >** button.



Click **Install** button to start installation.



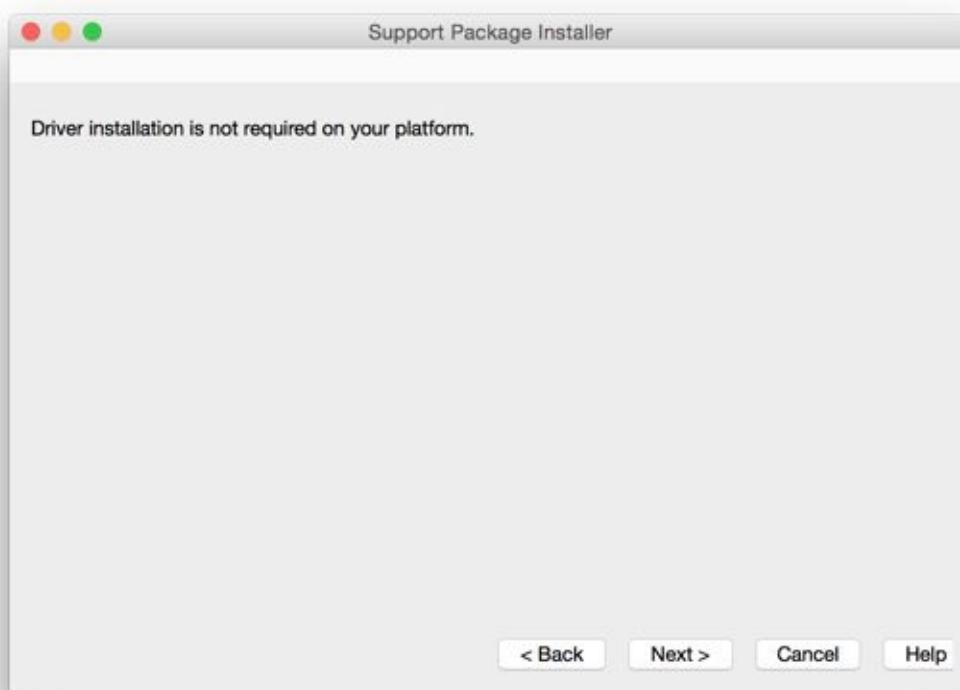
After done, you will be asked to configure Arduino board. Select Arduino and then, click **Next>** button.



Confirmation form will be shown. Click **Continue>** button.



The program will check if your platform needs Arduino driver or not. If you're working on Linux and Mac, you don't need a driver. You need to install Arduino driver if you're working on Windows platform. Click **Next>** button if done.



Installation is over. Click **Finish** button to close installation.



2.3 Connecting Arduino Board to Computer

Now you can connect Arduino board to computer. Then, verify which serial port is used for Arduino board. On Mac platform, you type this command.

```
$ ls /dev/cu*
```

On Linux platform, you type this command.

```
$ ls /dev/tty*
```

You can use Device Manager on Windows platform.

After that, you should see serial port of Arduino board which is attached on the computer.

My OSX detected my Arduino board used /dev/cu.usbmodem1421 serial port.



```
agusk$ ls /dev/cu*
/dev/cu.Bluetooth-Incoming-Port /dev/cu.usbmodem1421
/dev/cu.Bluetooth-Modem
agusk$
```

On MATLAB command Windows, type this command

```
>> a = arduino
```

MATLAB will detect your Arduino board. You should detected Arduino board information on Matlab Command Window.



documents ▶ MATLAB ▶

Command Window

```
>> a = arduino
Updating server code on board Uno (/dev/tty.usbmodem1421). Please wait.
```

```
a =
```

```
arduino with properties:
```

```
    Port: '/dev/tty.usbmodem1421'
    Board: 'Uno'
AvailablePins: {'D2-D13', 'A0-A5'}
    Libraries: {'I2C', 'SPI', 'Servo'}
```

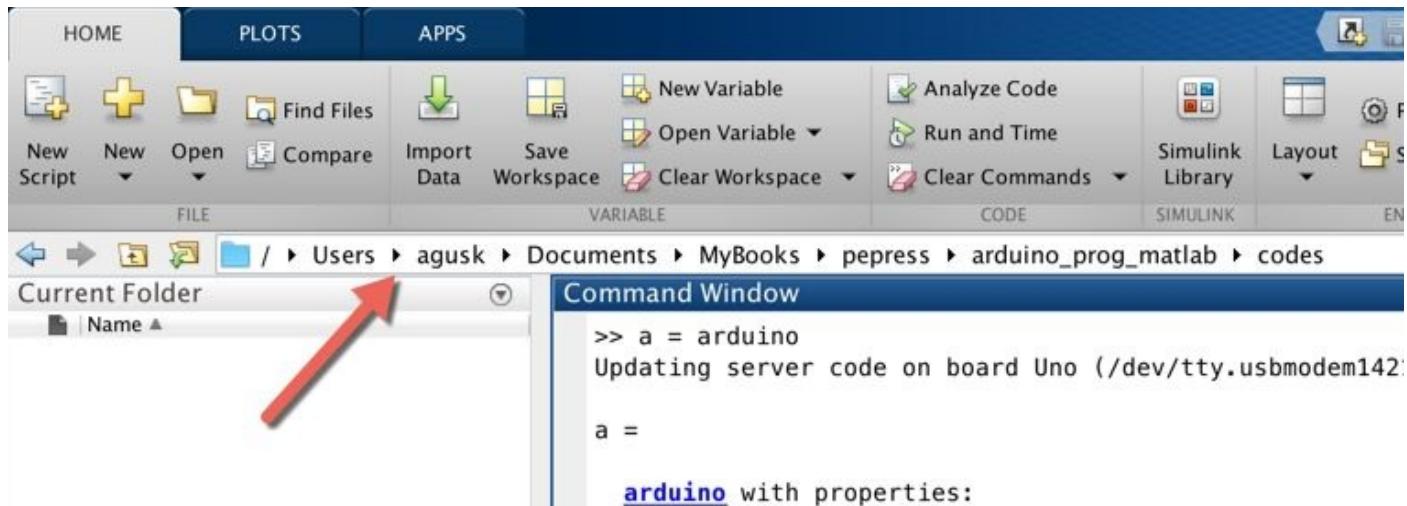
```
fx >> |
```

2.4 Hello Arduino: Blinking LED

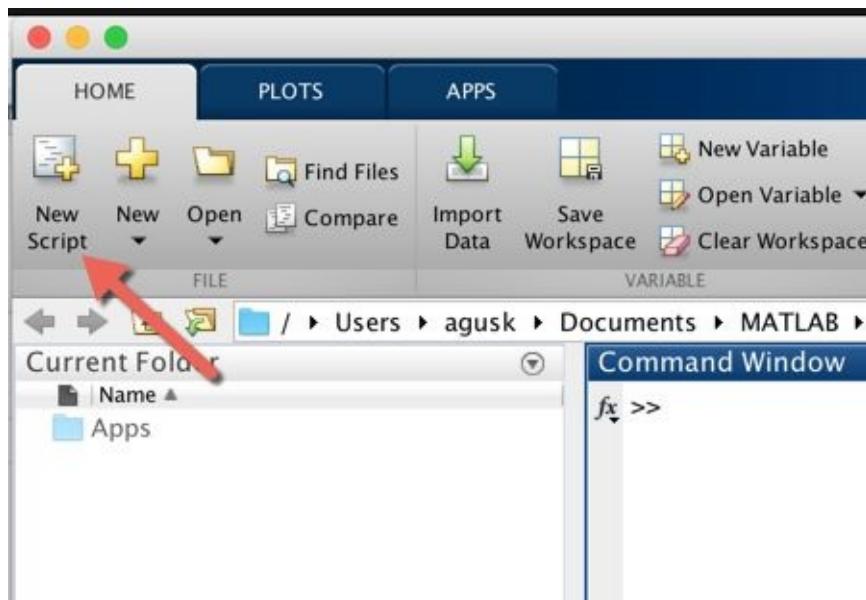
In this section, we build a blinking LED program using MATLAB. Arduino Uno/Mega/Leonardo boards provides onboard LED which is connected on pin 13.

Let's start to write our Blink program.

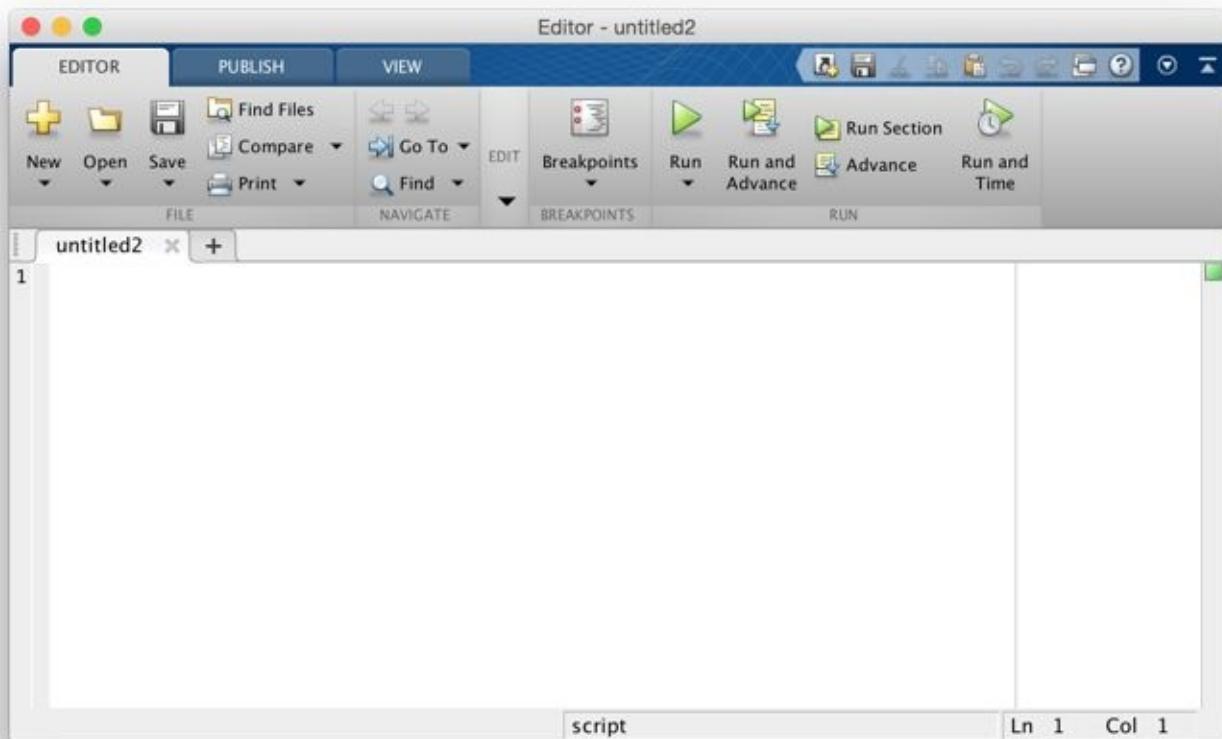
Firstly, you set working folder on MATLAB. You can change it on MATLAB IDE, see a red arrow.



Then, click **New Script** icon to create a new script file.



After that, you can get a script editor, shown in Figure below.



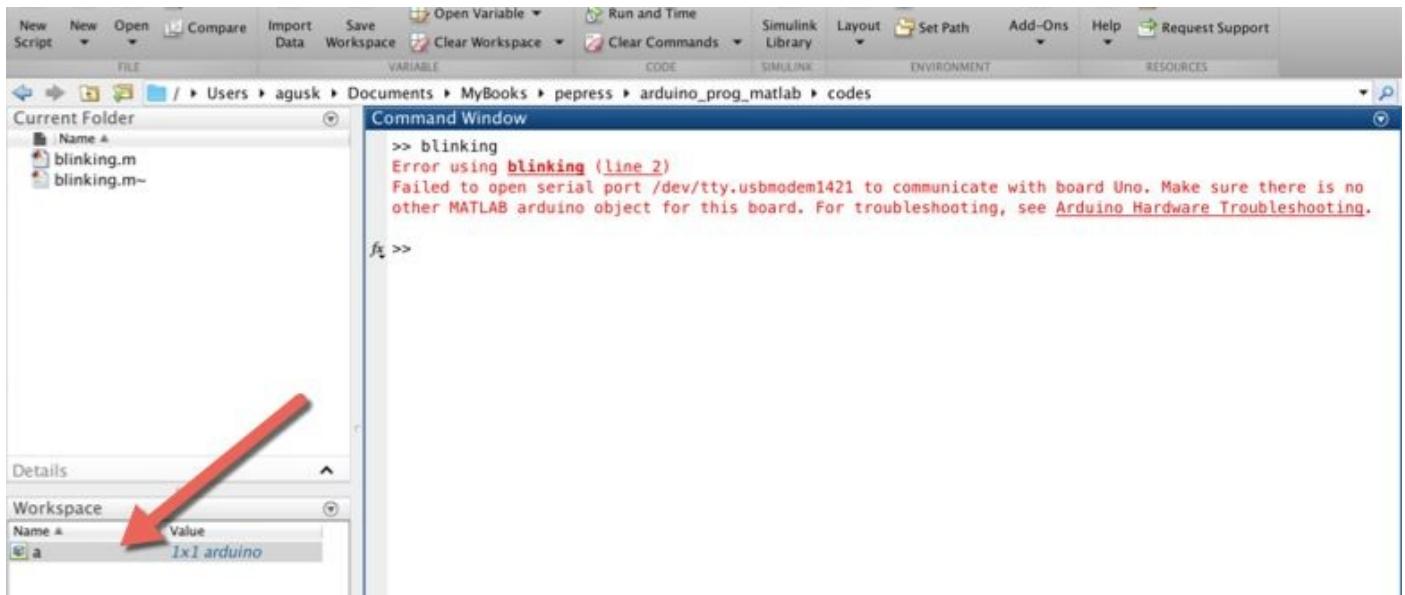
```
board = arduino();
led = 'D13';
for k=1:10
    disp('turn on LED');
    writeDigitalPin(board,led,1);
    pause(1);
    disp('turn off LED');
    writeDigitalPin(board,led,0);
    pause(1);
end

disp('close Arduino board');
clear board;
```

Save those scripts into a file, called `blinking.m`. Now you can run it.

```
>> blinking
```

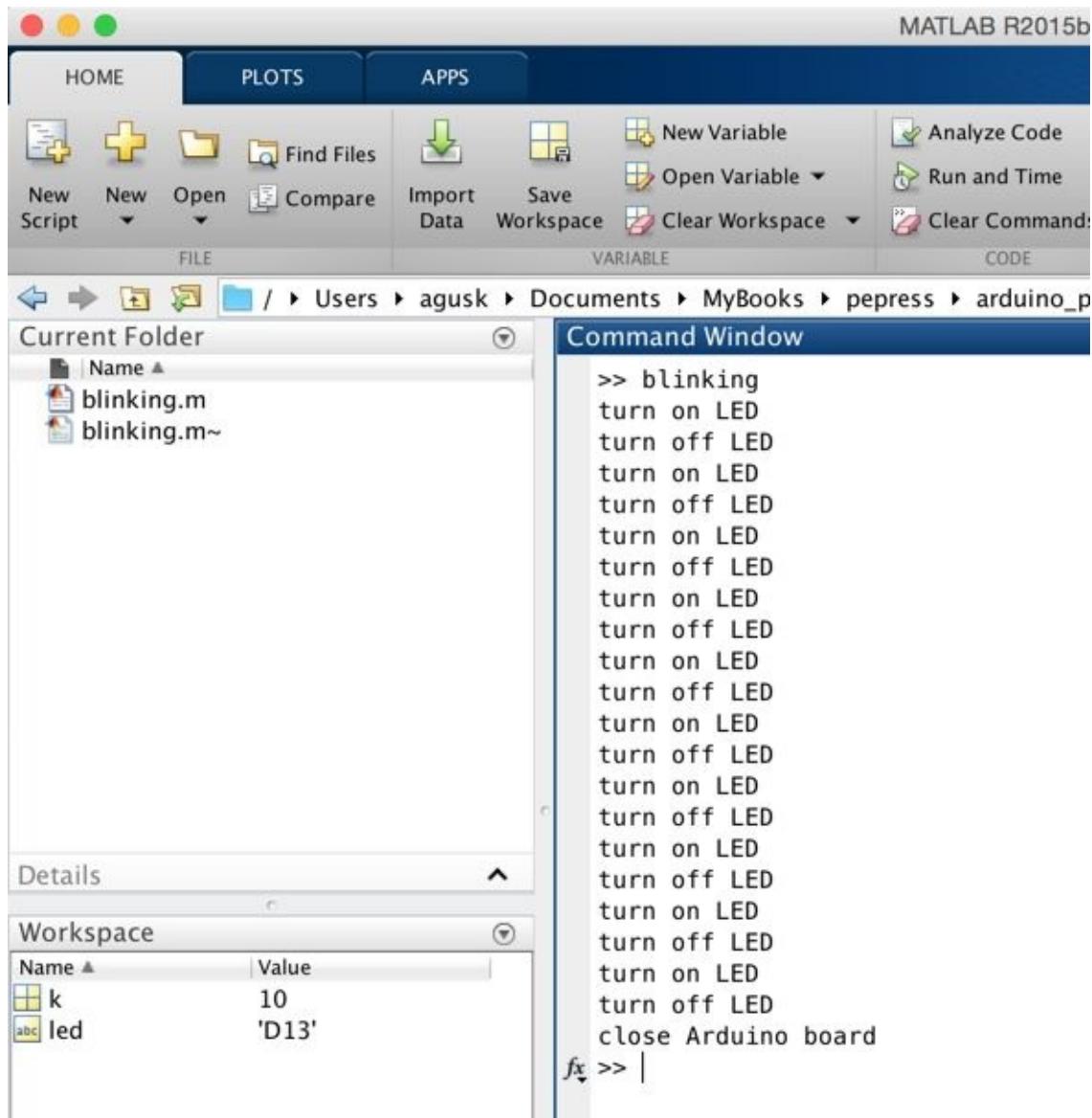
You may get error message, shown in Figure below.



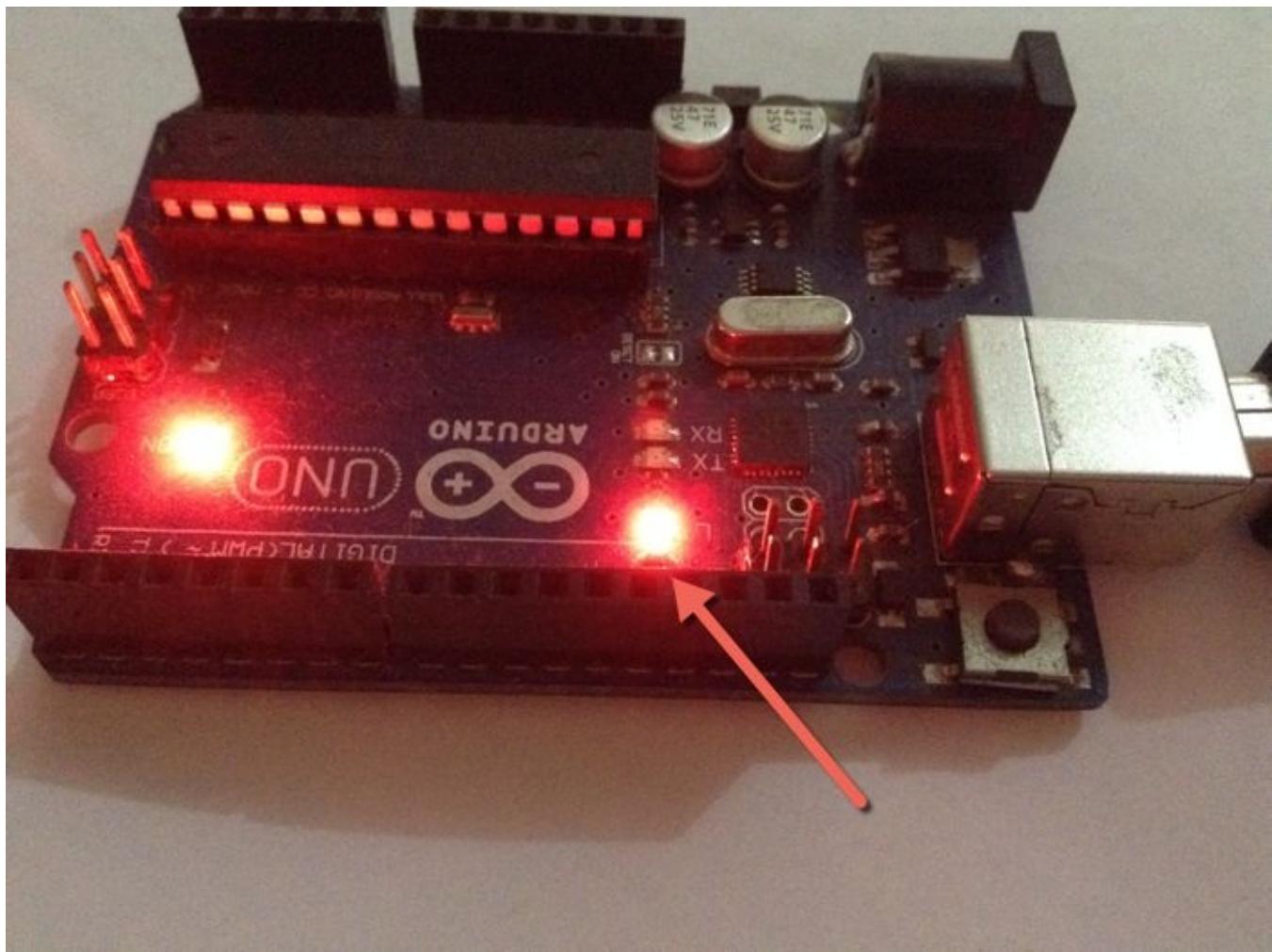
This error occurs because we create arduino object while there is existing arduino object. We can't use multiple arduino object. Delete existing arduino on Workspace Window. See a red arrow on above Figure.

We can clear our arduino object usage using clear syntax.

Now you can run the program again. The following is a sample output of blinking program.



You should see blinking LED on Arduino board.



3. Working with Digital I/O

In this chapter I'm going to explain how to work with digital I/O on Arduino board and write a program for demo.

3.1 Getting Started

MATLAB support for Arduino board provides three functions which we can use on digital I/O processing. The following is the functions:

- `configurePin()` is used to define pin mode either as input or output.
Reference: <http://www.mathworks.com/help/supportpkg/arduinoio/ref/configurepin.html>
- `writeDigitalPin()` is used to write digital data into a specific digital pin.
Reference: <http://www.mathworks.com/help/supportpkg/arduinoio/ref/writedigitalpin.html>
- `readDigitalPin()` is used to read digital input on specific digital pin.
Reference: <http://www.mathworks.com/help/supportpkg/arduinoio/ref/readdigitalpin.html>

To illustrate how to work with digital I/O, we build a simple program by utilizing LED and pushbutton.

3.2 Demo : LED and Pushbutton

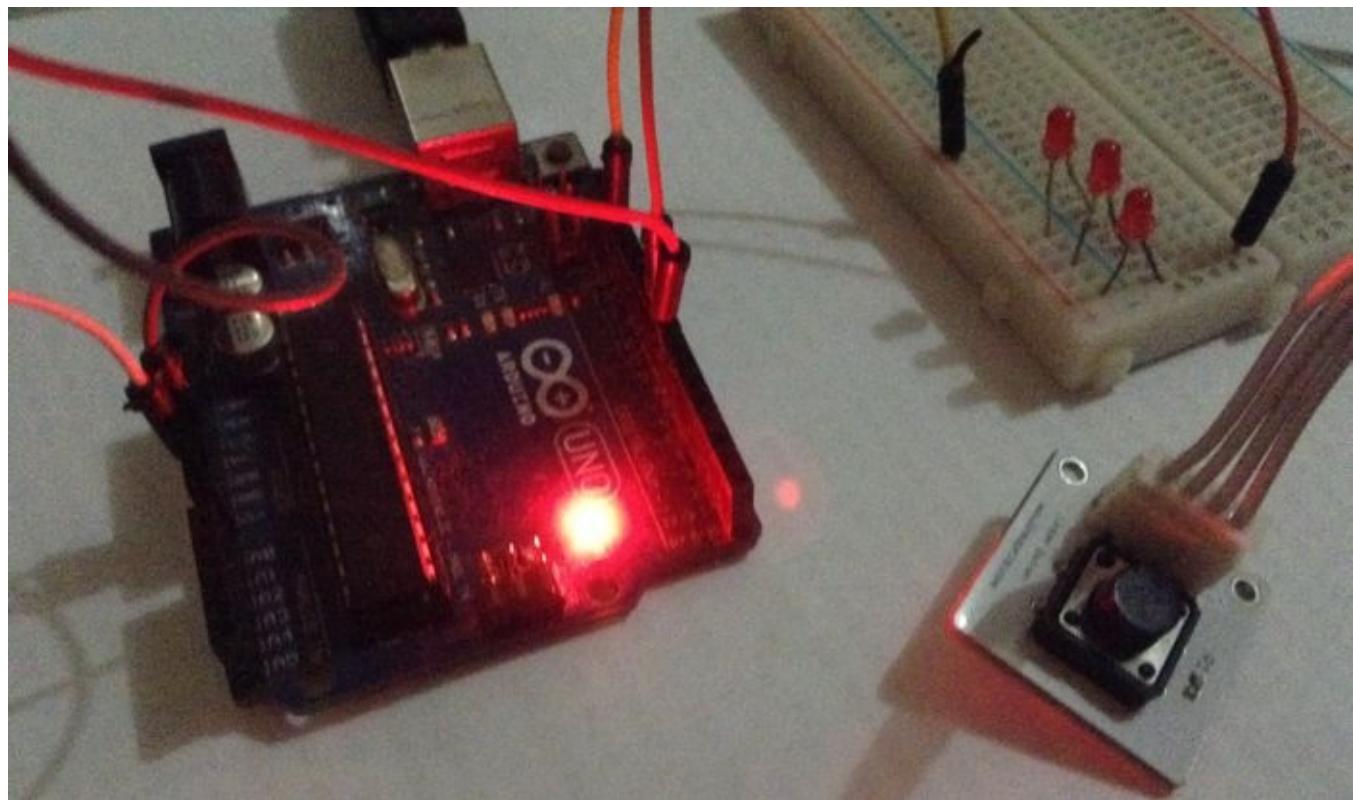
we build a program using LED and pushbutton. When we press a pushbutton, LED will lighting. It's a simple;).

3.2.1 Wiring

The following is hardware wiring:

- LED is connected to Arduino digital pin 9
- Pushbutton is connected to Arduino digital pin 8

A sample of hardware implementation is shown in Figure below.



3.2.2 Writing a Program

Now you can write these scripts.

```
function [] = led_pushbutton()
pushbutton = 'D8';
led = 'D9';
board = arduino();
```

```

finishup = onCleanup(@() exitprogram(board));

configurePin(board, pushbutton, 'DigitalInput');
disp('press Ctrl-C to exit');
while 1
    state = readDigitalPin(board, pushbutton);
    writeDigitalPin(board, led, state);
    disp(state);
    pause(0.5);
end
end

function exitprogram(b)
clear b;
disp('program has exit');
end

```

We use onCleanup(), <http://www.mathworks.com/help/matlab/ref/oncleanup.html> , to catch CTRL+C for quitting the program.

Save this program into file, called led_pushbutton.m.

3.2.3 Testing

Run this program by typing this command on Command Window on Matlab.

```
>> led_pushbutton
```

Press pushbutton. Then, you should see lighting LED. Press CTRL+C to exit program.

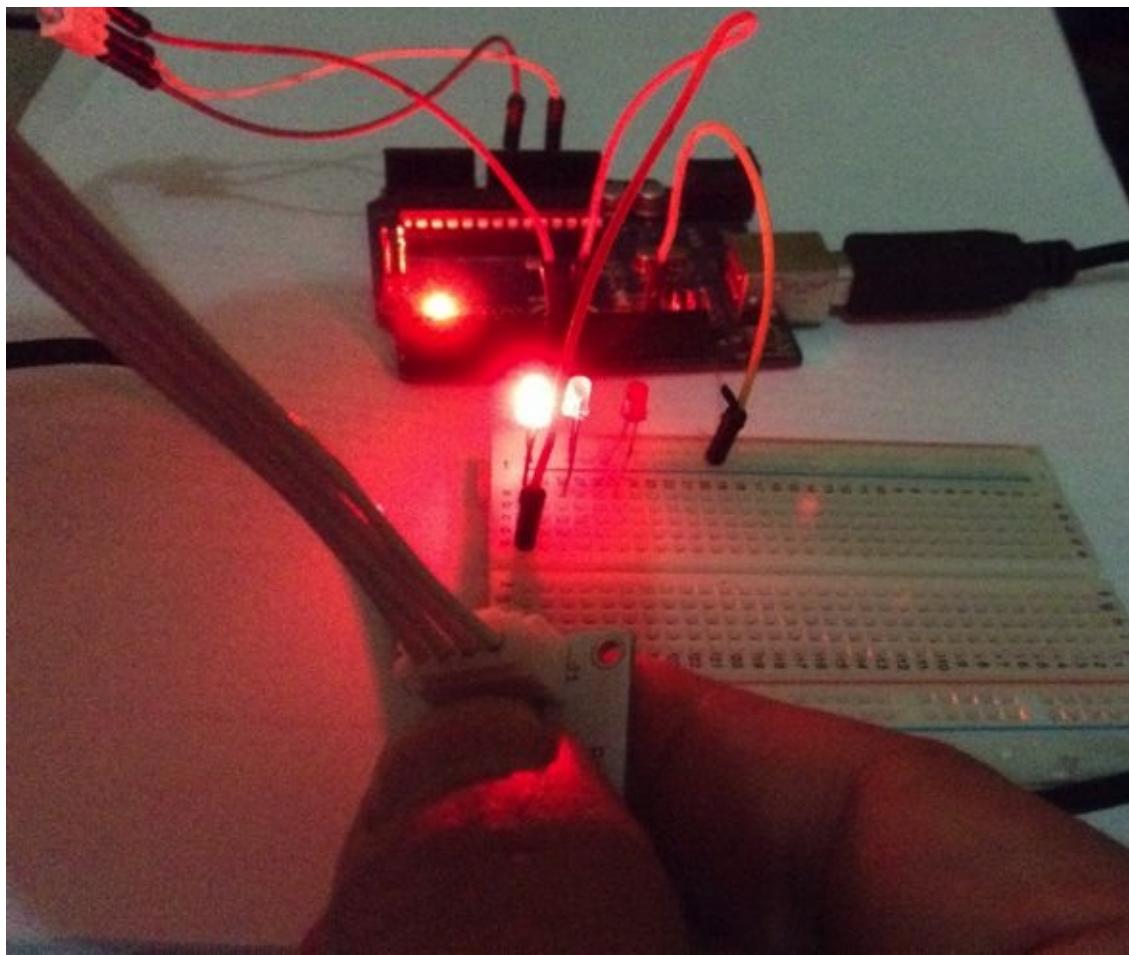
Program output:

Command Window

```
>> led_pushbutton
press Ctr-C to exit
0
0
0
1
1
1
0
1
1
0
1
1

program has exit
Operation terminated by user during led_pushbutton (line 13)
```

LED is lighting while a pushbutton is pressed.



4. Working with PWM and Analog Input

This chapter explains how to work with Arduino Analog I/O using MATLAB.

4.1 Getting Started

MATLAB support for Arduino board provides five functions which we can use on analog I/O processing. The following is the functions:

- `configurePin()` is used to define pin mode either as input or output.
Reference: <http://www.mathworks.com/help/supportpkg/arduinoio/ref/configurepin.html>
- `writePWMVoltage()` is used to write PWM voltage on digital pin.
Reference: <http://www.mathworks.com/help/supportpkg/arduinoio/ref/writepwmvoltage.html>
- `writePWMDutyCycle()` is used to set PWM duty cycle on digital pin.
Reference: <http://www.mathworks.com/help/supportpkg/arduinoio/ref/writepwmduycycle.html>
- `readVoltage()` to read analog input on Analog pin.
Reference: <http://www.mathworks.com/help/supportpkg/arduinoio/ref/readvoltage.html>

In this chapter, we try to access Arduino Analog I/O using MATLAB. There are three scenarios for our cases:

- Controlling RGB LED
- Controlling LED brightness
- Reading Analog input using Potentiometer

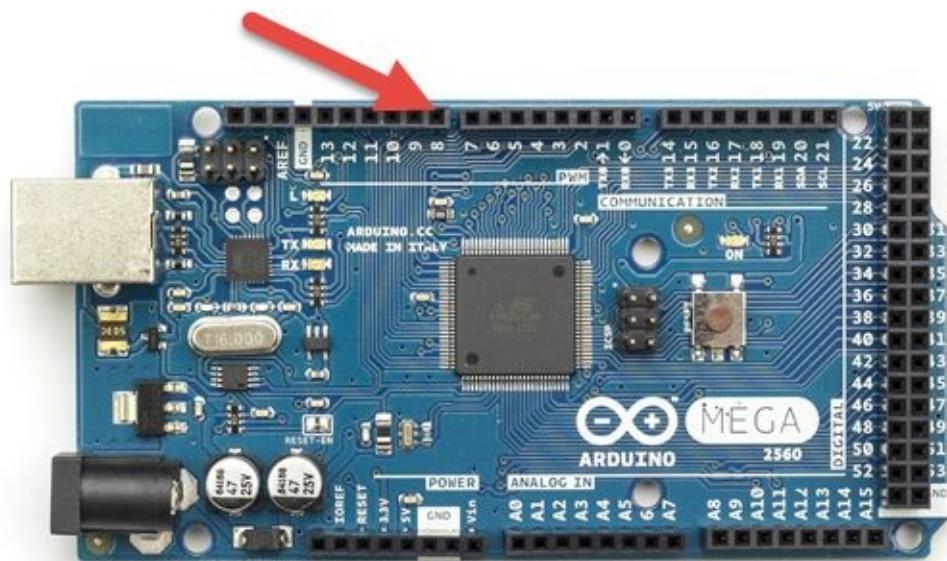
Let's start.

4.2 Demo Analog Output (PWM) : RGB LED

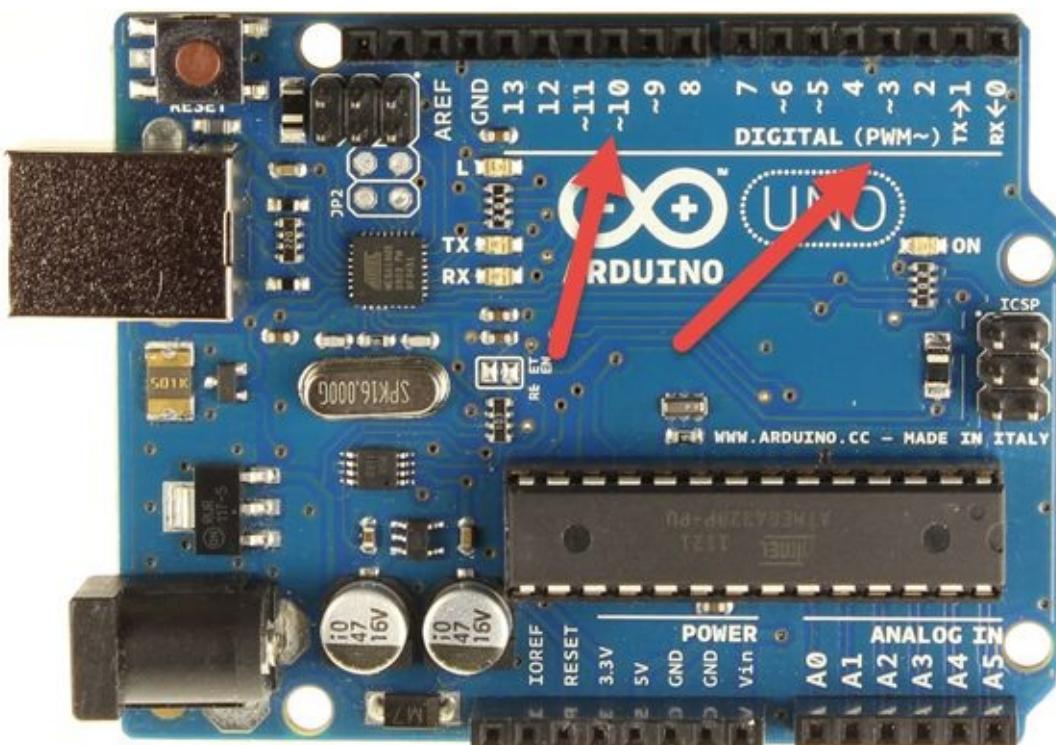
In this scenario we build a program to control RGB LED color using Arduino Analog output (PWM).

Please be careful if you want to work with Arduino PWM. If you have Arduino Mega, you will see PWM label so you obtain PWM pins easily but if you have Arduino Uno, it writes DIGITAL (PWM ~). It means your PWM pins can be found on DIGITAL pins which pin with ~, for instance, ~3,~5,~6,~9, ~10, ~11.

For Arduino Mega 2560, you can see PWM pins on picture below (see red arrow).



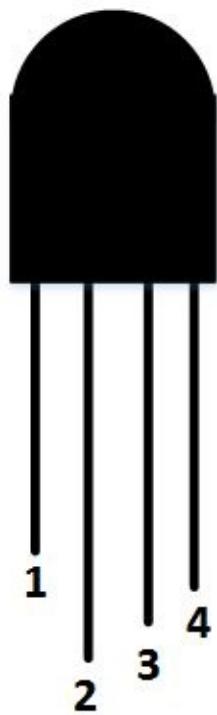
For Arduino Uno R3, you can see PWM pins as below.



RGB LED has 4 pins that you can see it on Figure below.



To understand these pins, you can see the following Figure.



Note:

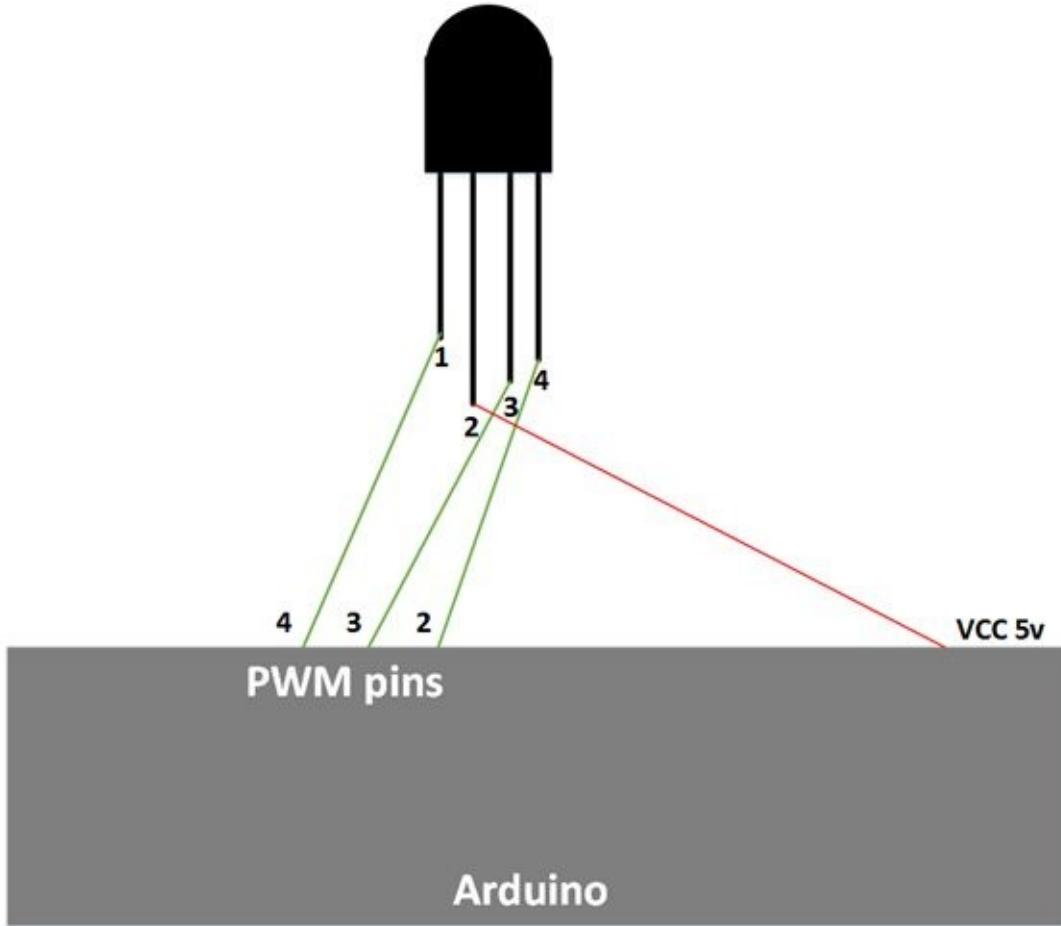
- Pin 1: Red
- Pin 2: Common pin
- Pin 3: Green

- Pin 4: Blue

Now we can start to build a program and hardware implementation.

4.2.1 Wiring

Firstly we implement RGB LED hardware. The following is a hardware schema.



For our testing, we configure the following PWM pins.

Arduino Mega 2560:

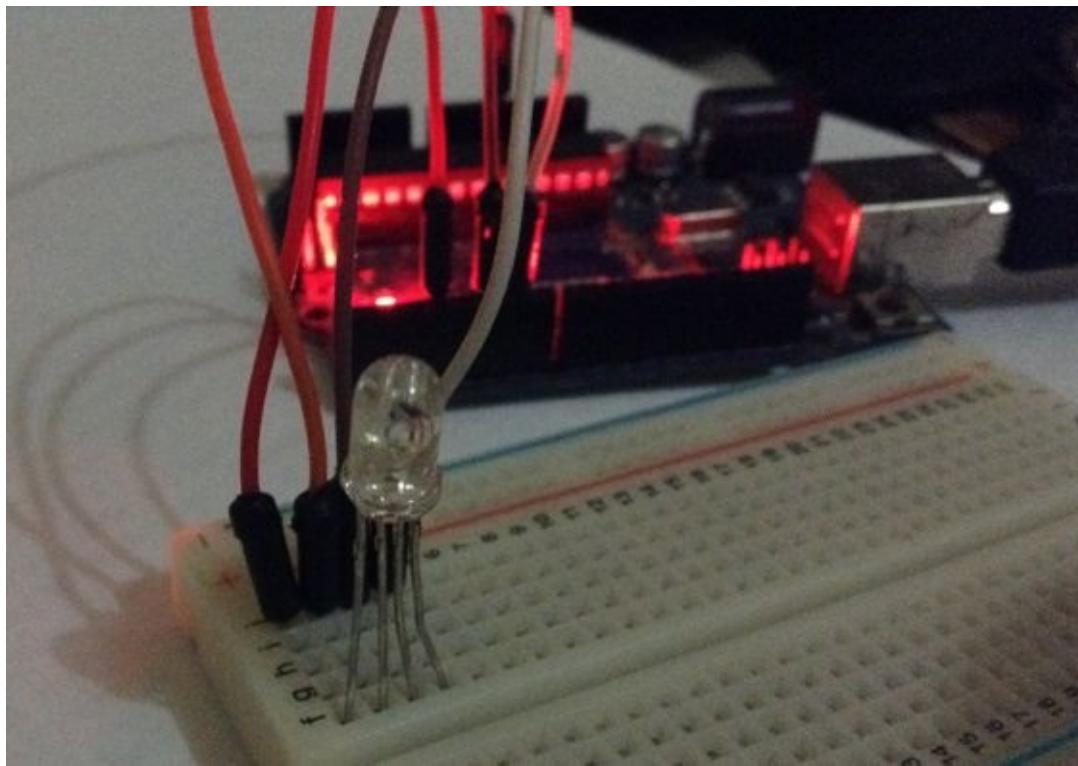
- RGB LED pin 1 (red) is connected to Arduino PWM pin 3
- RGB LED pin 2 is connected to Arduino VCC 5V
- RGB LED pin 3 (green) is connected to Arduino PWM pin 5
- RGB LED pin 4 (blue) is connected to Arduino PWM pin 6

Arduino Uno R3:

- RGB LED pin 1 (red) is connected to Arduino PWM pin 3

- RGB LED pin 2 is connected to Arduino VCC 5V
- RGB LED pin 3 (green) is connected to Arduino PWM pin 5
- RGB LED pin 4 (blue) is connected to Arduino PWM pin 6

Here is a sample implementation with Arduino Uno R3.



4.2.2 Writing Program

To display a certain color, we must combine colors from red, green, blue. MATLAB provides API for PWM like Arduino API such as `writePWMDutyCycle()` with analog value from 0 to 1.

Let's start to build a program. Firstly, open MATLAB. Then, write these scripts.

```
function [] = led_rgb()
board = arduino();
finishup = onCleanup(@() exitprogram(board));
configurePin(board, 'D3', 'PWM');
configurePin(board, 'D5', 'PWM');
configurePin(board, 'D6', 'PWM');
disp('press Ctrl-C to exit');
while 1
    disp('red');
    write_rgb(board, 0, 1, 1); % red
    pause(1);

    disp('green');
```

```

write_rgb(board,1,0,1); % green
pause(1);

disp('blue');
write_rgb(board,1,1,0); % blue
pause(1);

disp('yellow');
write_rgb(board,0,0,1); % yellow
pause(1);

disp('purple');
write_rgb(board,0.3,1,0.3); % purple
pause(1);

disp('aqua');
write_rgb(board,1,0,0); % aqua
pause(1);
end

end

% testing for Arduino Uno
function write_rgb(board,r,g,b)
writePWMDutyCycle(board,'D3',r);
writePWMDutyCycle(board,'D5',g);
writePWMDutyCycle(board,'D6',b);
end

function exitprogram(b)
clear b;
disp('program has exit');
end

```

Save this program as led_rgb.m.

This program will generate six colors: red, green, blue, yellow, purple, and aqua.

4.2.3 Testing

Upload and run the program. You should see several color on RGB LED.

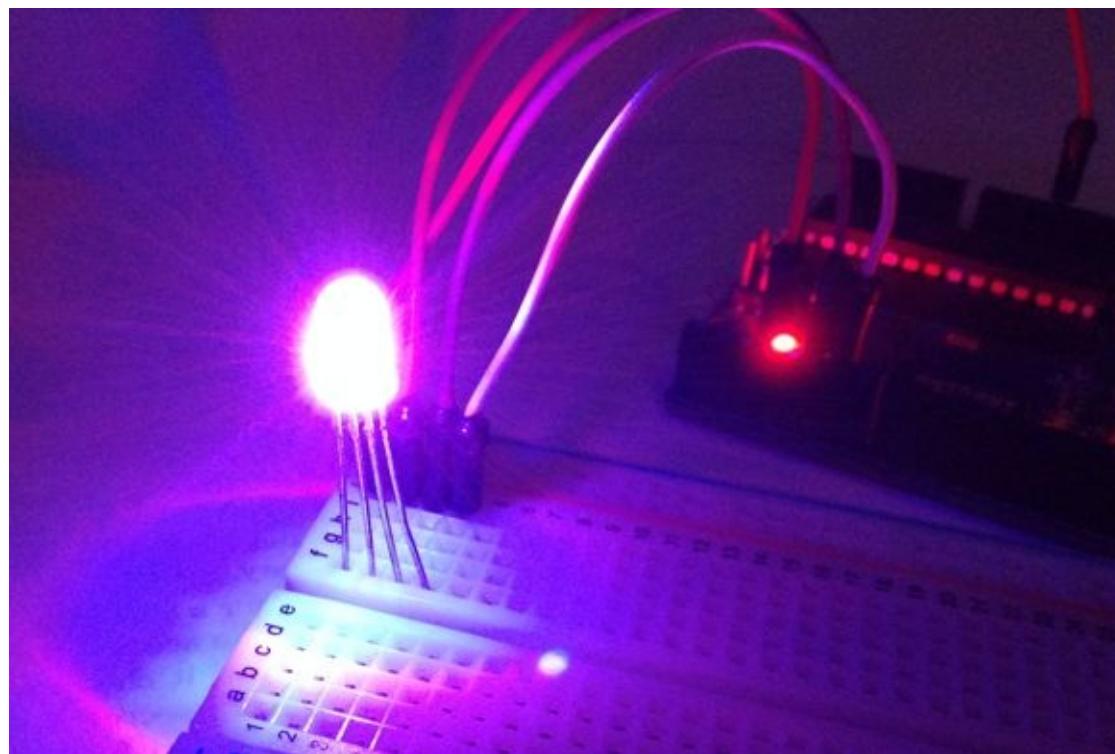
```
>> led_rgb
```

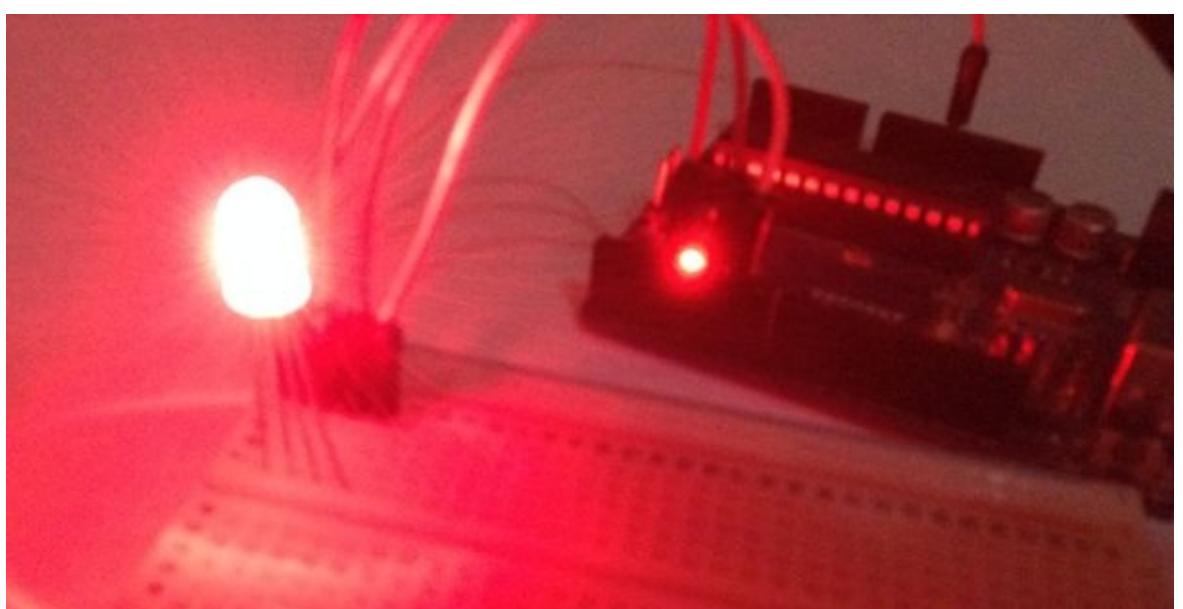
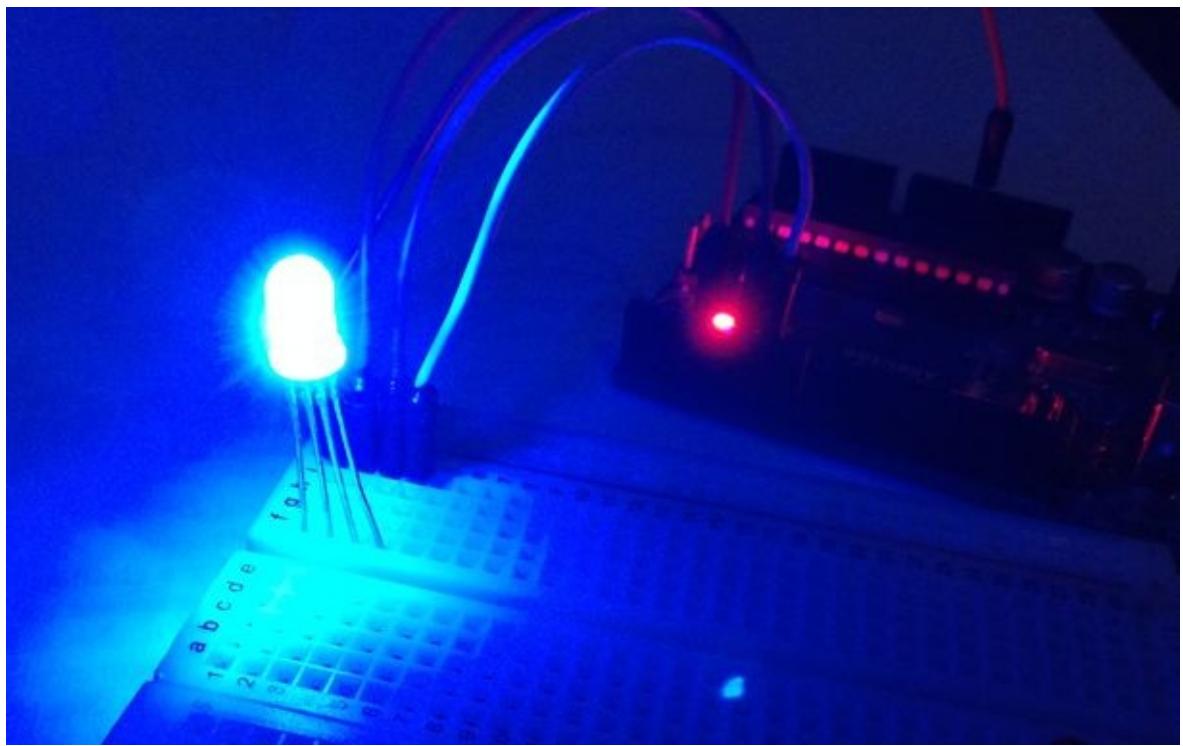
The following is a sample demo on Command Window on MATLAB.

Command Window

```
>> led_rgb
press Ctr-C to exit
red
green
blue
yellow
purple
aqua
red
green
blue
yellow
purple
aqua
red
green
blue
yellow
```

The following is a sample demo on hardware.





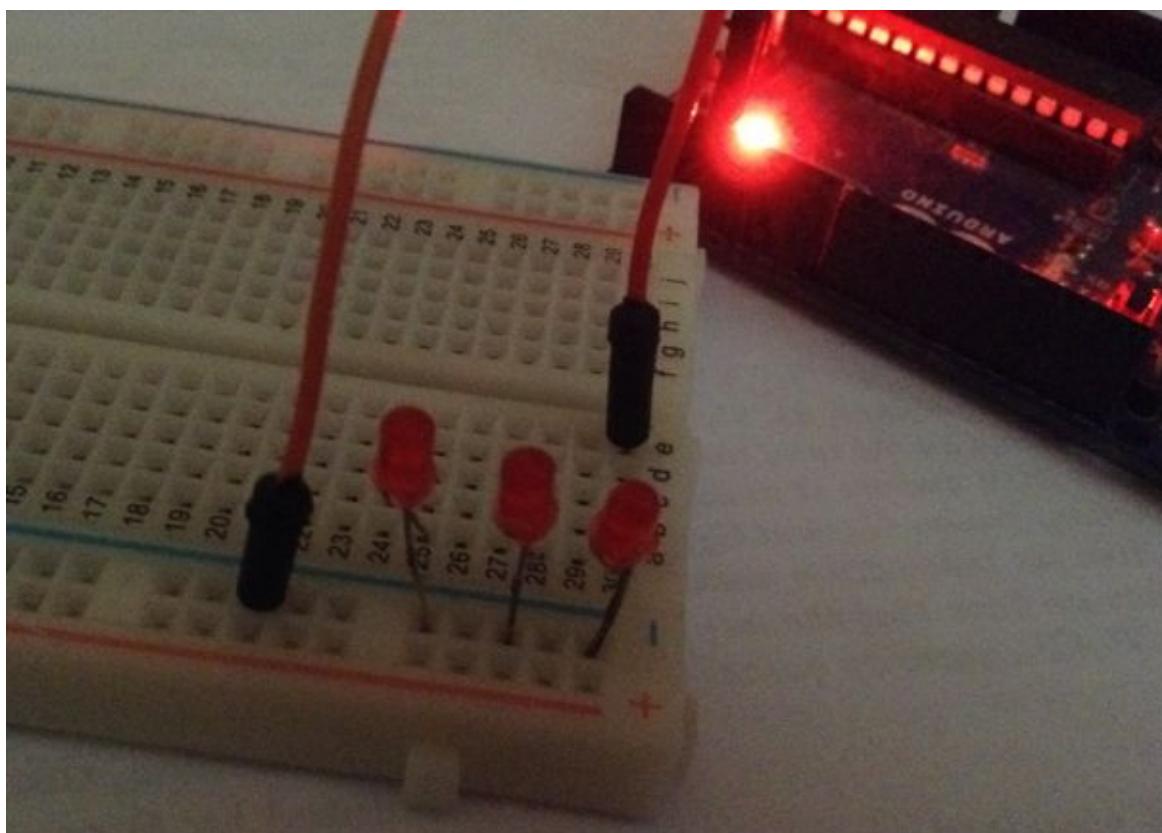
4.3 Demo Analog Output Voltage: LED Brightness

In this demo, we try to control LED brightness by controlling voltage on LED. MATLAB for Arduino support provides `writePWMVoltage()` function to set voltage on PWM pins. We can set a voltage value from 0 to 5 for Arduino Uno/Mega and 0 - 3.3 for Arduino Due.

Let's build.

4.3.1 Wiring

We connect a LED on PWM pin on digital pin 3. The following is my hardware wiring.



4.3.2 Writing a Program

Now you can open MATLAB and write these scripts.

```
function [] = led_brightness()
board = arduino();
finishup = onCleanup(@() exitprogram(board));
configurePin(board, 'D3', 'PWM');
disp('press Ctrl-C to exit');
```

```
while 1
  for k = 0:5
    writePWMMVoltage(board, 'D3', k);
    pause(1);
  end
  for k = 5:-1:0
    writePWMMVoltage(board, 'D3', k);
    pause(1);
  end
end

function exitprogram(b)
clear b;
disp('program has exit');
end
```

Save these scripts into a file, called led_brightness.m.

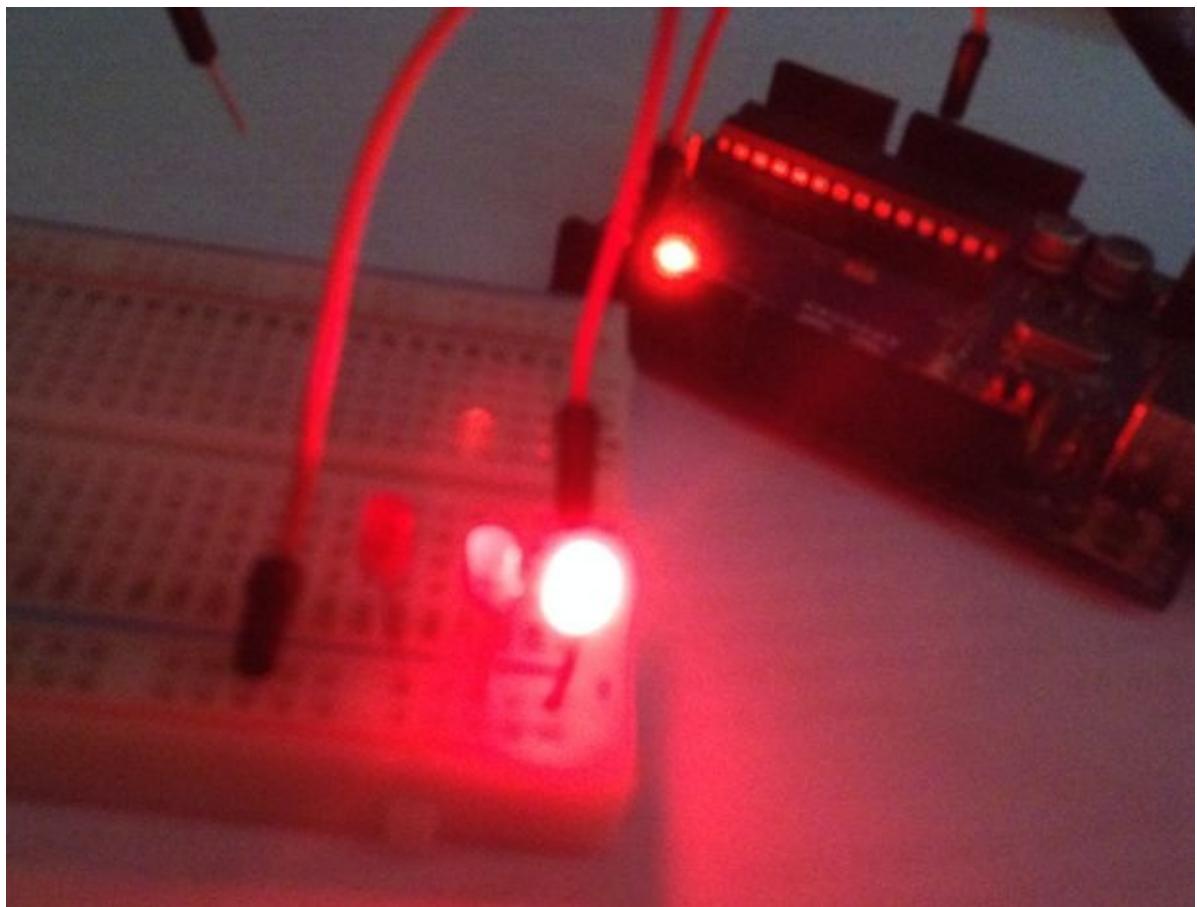
This program will set voltage value on PWM pin on digital pin 3 from 0 to 5 and then set a value from 5 to 0 too.

4.3.3 Testing

Make sure Arduino board already connected to your computer. You can run the program by typing this command.

```
>> led_brightness
```

On hardware side, you should see LED brightness changing gradually.



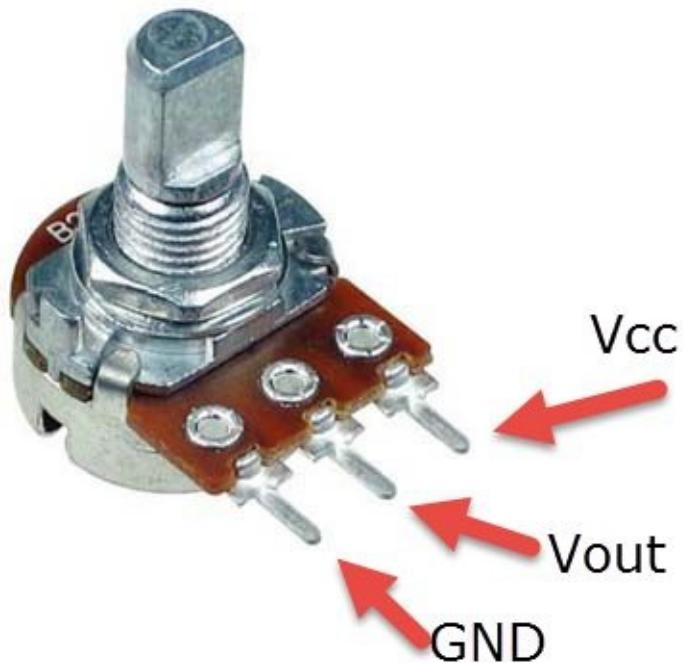
4.4 Demo Analog Input: Working with Potentiometer

In this section, we learn how to read analog input on Arduino board. For illustration, I use Potentiometer as analog input source. Our scenario is to read analog value from Potentiometer. Then, display it on console.

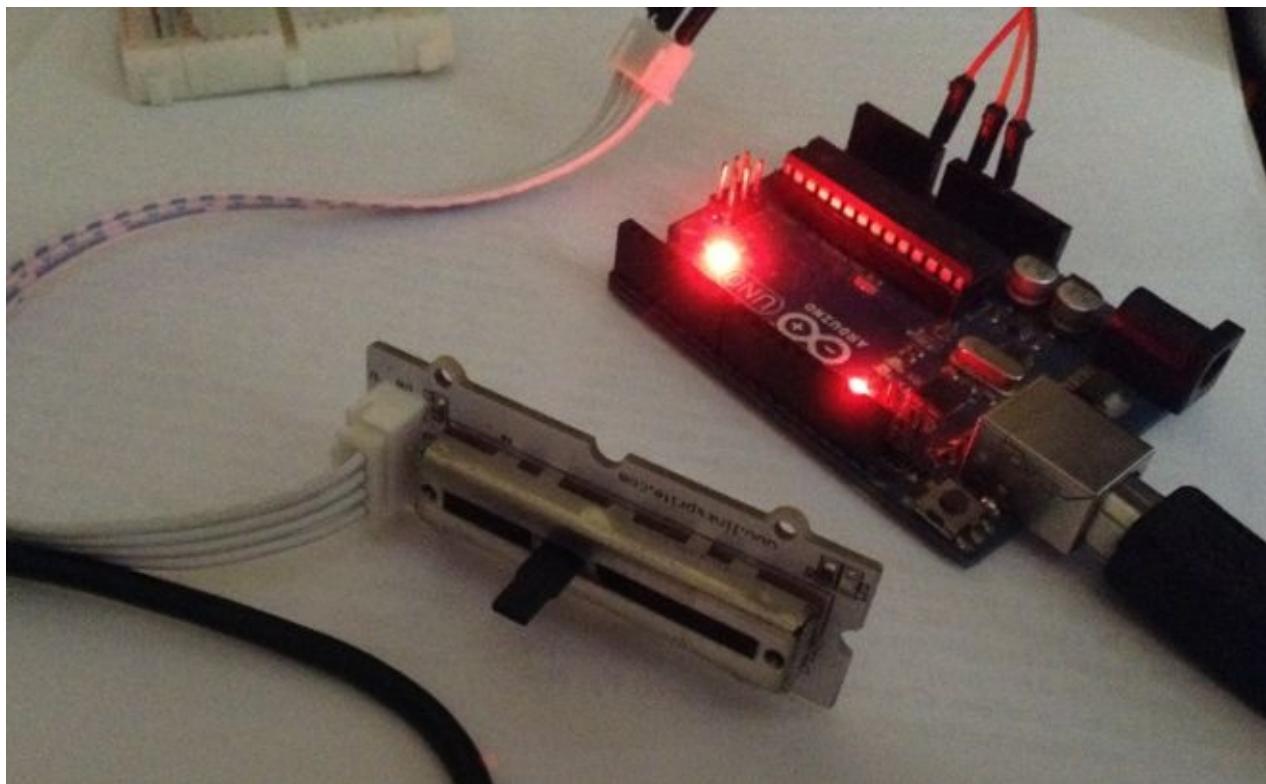
Let's start!.

4.4.1 Wiring

To understand Potentiometer, you see its scheme in Figure below.



You can connect VCC to Arduino board on VCC +5V pin. Vout to Arduino board Analog input A0. In addition, GND to Arduino board GND. The following is hardware implementation. I use slide potentiometer.



4.4.2 Writing Program

Firstly, create a program via MATLAB. To read analog input, we can use `readVoltage()` function. Ok, Let's write these scripts.

```
function [] = potentiometer()
board = arduino();
finishup = onCleanup(@() exitprogram(board));
disp('press Ctr-C to exit');
while 1
    analog = readVoltage(board, 'A0');
    disp(['analog= ', num2str(analog)]);
    pause(1);
end

function exitprogram(b)
clear b;
disp('program has exit');
end
```

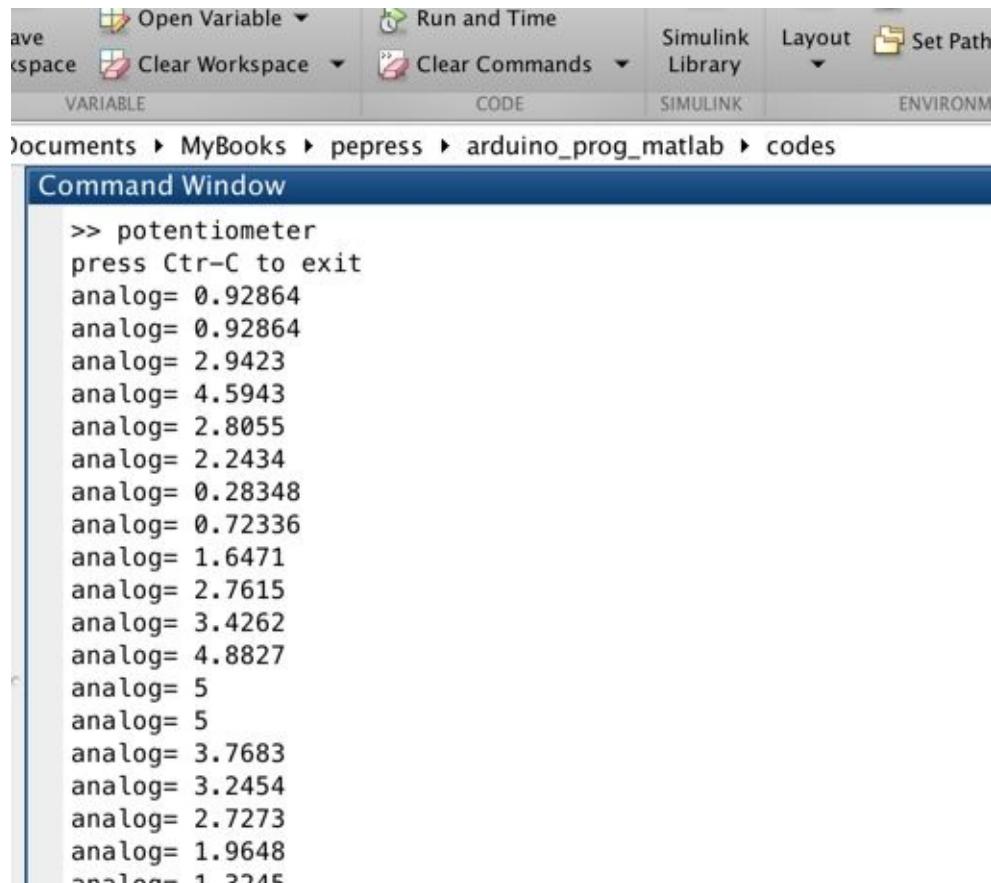
Save this code as `potentiometer.m`

4.4.3 Testing

To run the program, you can type this command.

```
>> potentiometer
```

You should see analog value on Command Window.



The screenshot shows the MATLAB interface with the Command Window active. The window title is 'Command Window'. The command entered is 'potentiometer'. The output shows a series of analog values ranging from 0.92864 to 1.3245, with a note to press Ctrl-C to exit. The MATLAB menu bar is visible at the top, showing options like 'File', 'Edit', 'View', 'Insert', 'Cell', 'Run', 'Time', 'Simulink', 'Layout', and 'Help'.

```
>> potentiometer
press Ctr-C to exit
analog= 0.92864
analog= 0.92864
analog= 2.9423
analog= 4.5943
analog= 2.8055
analog= 2.2434
analog= 0.28348
analog= 0.72336
analog= 1.6471
analog= 2.7615
analog= 3.4262
analog= 4.8827
analog= 5
analog= 5
analog= 3.7683
analog= 3.2454
analog= 2.7273
analog= 1.9648
analog= 1.3245
```

5. Working with I2C

In this chapter we learn how to work with I2C on Arduino board using MATLAB.

5.1 Getting Started

The I2C (Inter-Integrated Circuit) bus was designed by Philips in the early '80s to allow easy communication between components which reside on the same circuit board. TWI stands for Two Wire Interface and for most parts this bus is identical to I²C. The name TWI was introduced by Atmel and other companies to avoid conflicts with trademark issues related to I²C.

I2C bus consists of two wires, SDA (Serial Data Line) and SCL (Serial Clock Line). You can see I2C pins on Arduino board as follows:

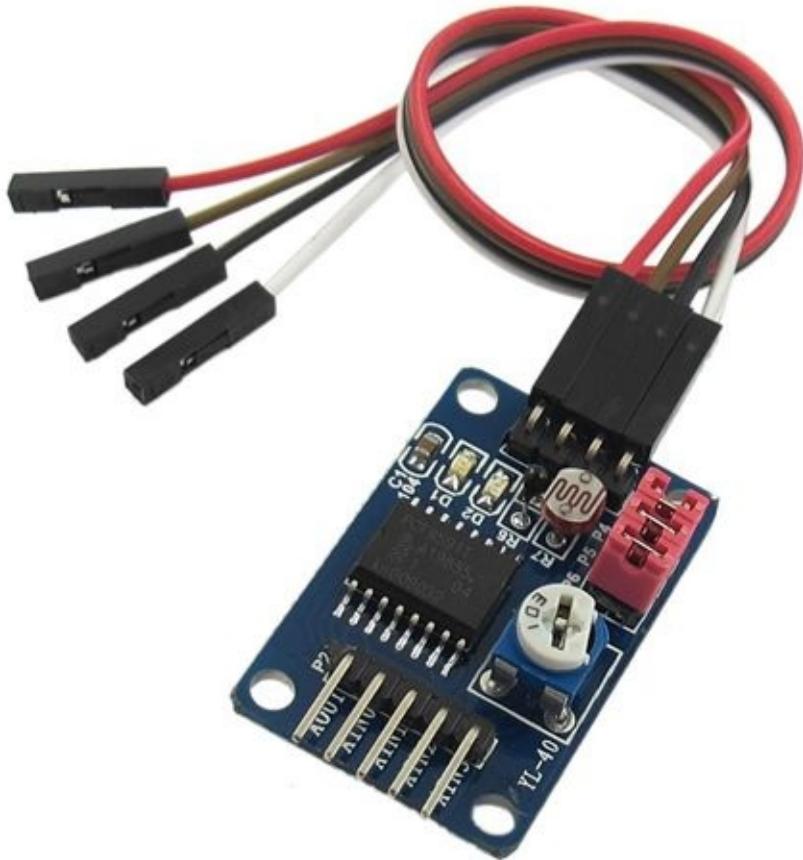
- Arduino Uno: A4 pin as SDA and A5 as SCL pin
- Arduino Mega 2560: Digital pin 20 as SDA and Digital pin 21 as SCL

MATLAB for Arduino support provides several functions to access I2C protocol. You can read it on <http://www.mathworks.com/help/supportpkg/arduinoio/i2c-sensors.html> .

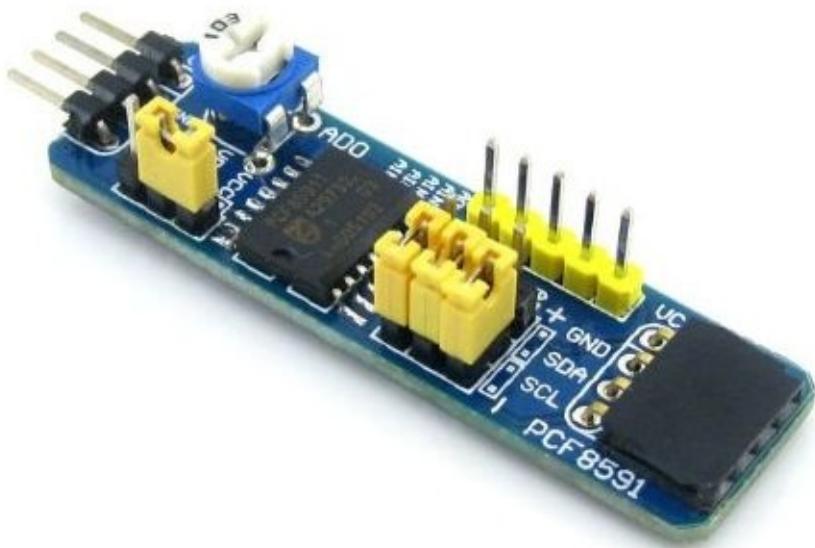
For testing, I used PCF8591 AD/DA Converter module with sensor and actuator devices. You can find it on the following online store:

- Amazon, <http://www.amazon.com/PCF8591-Converter-Module-Digital-Conversion/dp/B00BXX4UWC/>
- eBay, <http://www.ebay.com>
- Dealextreme, <http://www.dx.com/p/pcf8591-ad-da-analog-to-digital-digital-to-analog-converter-module-w-dupont-cable-deep-blue-336384>
- Aliexpress, <http://www.aliexpress.com/>

In addition, you can find this device on your local electronics store/online store.



This module has mini form model too, for instance, you can find it on Amazon, <http://www.amazon.com/WaveShare-PCF8591T-Converter-Evaluation-Development/dp/B00KM6X2OI/> .



This module use PCF8591 IC and you can read the datasheet on the following URLs.

- <http://www.electrodragon.com/w/images/e/ed/PCF8591.pdf>
- http://www.nxp.com/documents/data_sheet/PCF8591.pdf

In this chapter, we build a program to access sensor via I2C using Arduino software on Arduino board.

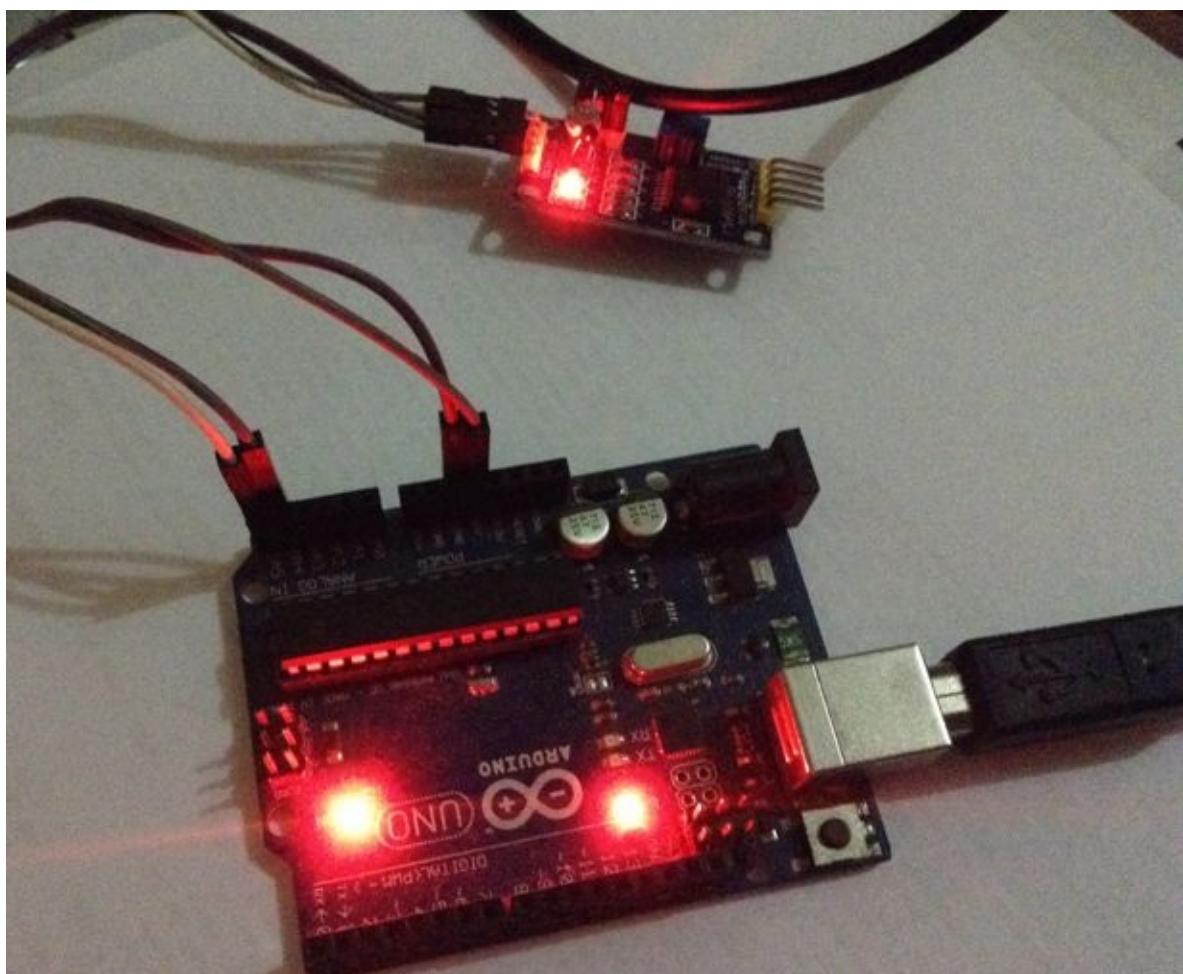
5.2 Writing Program

We use PCF8591 AD/DA Converter as I2C source. You can connect PCF8591 AD/DA Converter to Arduino board directly. In this demo, I use Arduino Uno.

The following is our wiring lab

- PCF8591 AD/DA Converter SDA —> Arduino SDA (A4)
- PCF8591 AD/DA Converter SCL —> Arduino CLK (A5)
- PCF8591 AD/DA Converter VCC —> Arduino VCC +5V
- PCF8591 AD/DA Converter GND —> Arduino GND

Hardware implementation can be shown in Figure below.



5.3 Demo 1: Scanning I2C

After attached our sensor to Arduino, we can scan our I2C address using `scanI2CBus()` function. On Arduino Uno, we use 0 for I2C.

Write these scripts.

```
board = arduino();
address = scanI2CBus(board, 0); % uno = 0
disp(['i2c address: ', address]);

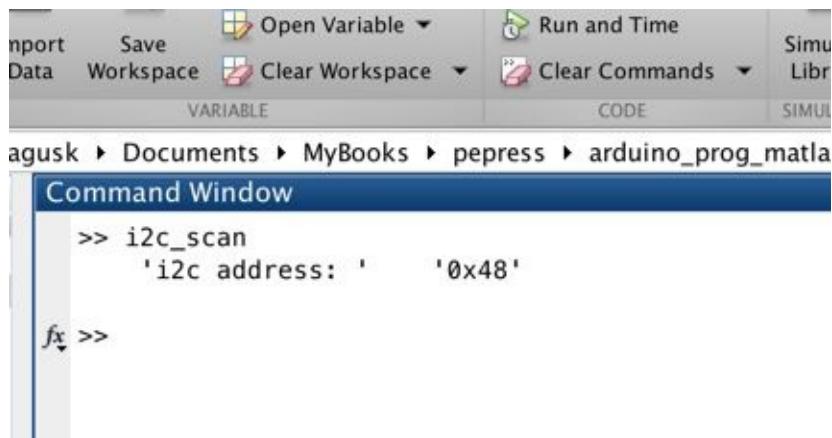
clear board;
```

Save the program into a file, called `i2c_scan.m`.

Run the program.

```
>> i2c_scan
```

On Command Window, you should see I2C address of sensor device. For instance, my sensor was detected on `0x48`.



I2C address will be used on the next demo.

5.4 Demo 2: Reading Data from Sensor Based I2C

We use I2C on Arduino board using i2c object, see

<http://www.mathworks.com/help/supportpkg/arduinoio/i2c-sensors.html>. PCF8591

AD/DA Converter module has three sensor devices: Thermistor, Photo-voltaic cell and Potentiometer. This module runs on I2C bus with address 0x48. In this case, we read all sensor data.

Write these scripts.

```
function [] = i2c_sensor()
board = arduino();
finishup = onCleanup(@() exitprogram(board));
disp('press Ctr-C to exit');
PCF8591 = '0x48';
PCF8591_ADC_CH0 = '40'; % thermistor
PCF8591_ADC_CH1 = '41'; % photo-voltaic
PCF8591_ADC_CH3 = '43'; % potentiometer
i2c = i2cdev(board,PCF8591);
disp(['thermistor ', 'photo ', 'potentiometer']);
while 1
    thermistor = read_adc(i2c,hex2dec(PCF8591_ADC_CH0));
    pause(0.5);
    photo = read_adc(i2c,hex2dec(PCF8591_ADC_CH1));
    pause(0.5);
    potentiometer = read_adc(i2c,hex2dec(PCF8591_ADC_CH3));
    pause(0.5);
    disp([thermistor, photo, potentiometer]);
end

end

function adc = read_adc(dev,config)
write(dev,config);
read(dev, 1);
out = read(dev, 1);
adc = out;
end

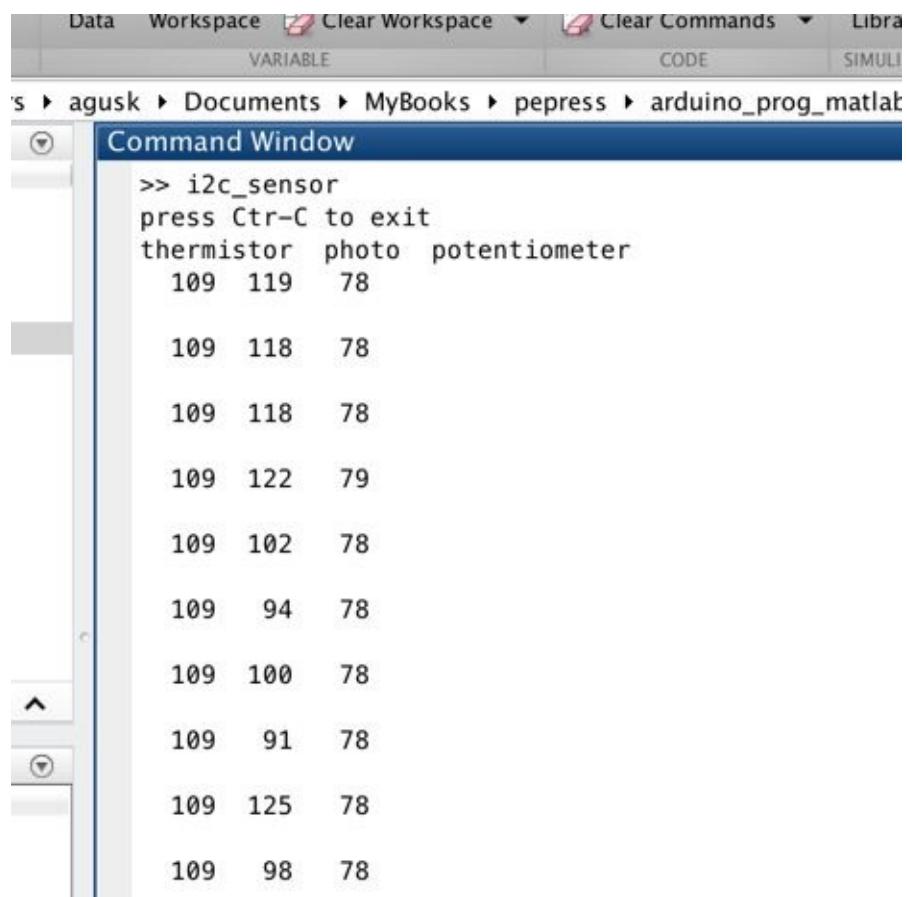
function exitprogram(b)
clear b;
disp('program has exit');
end
```

Save these scripts into a file, called i2c_sensor.m.

Run the program.

```
>> i2c_sensor
```

If success, you should see values for Thermistor, Photo-voltaic, and potentiometer.



The screenshot shows the MATLAB interface with the Command Window active. The window title is 'Command Window'. The command entered is 'i2c_sensor'. The output displays a series of three columns of data, labeled 'thermistor', 'photo', and 'potentiometer'. The values for each row are: 109, 119, 78; 109, 118, 78; 109, 122, 79; 109, 102, 78; 109, 94, 78; 109, 100, 78; 109, 91, 78; 109, 125, 78; and 109, 98, 78. A note in the output says 'press Ctr-C to exit'.

```
>> i2c_sensor
press Ctr-C to exit
thermistor    photo    potentiometer
  109    119    78
  109    118    78
  109    122    79
  109    102    78
  109    94     78
  109    100    78
  109    91     78
  109    125    78
  109    98     78
```

6. Working with SPI

In this chapter I'm going to explain how to work with SPI on Arduino board using MATLAB.

6.1 Getting Started

The Serial Peripheral Interface (SPI) is a communication bus that is used to interface one or more slave peripheral integrated circuits (ICs) to a single master SPI device; usually a microcontroller or microprocessor of some sort.

SPI in Arduino Uno board can be defined on the following pins:

- MOSI on Digital pin 11
- MISO on Digital pin 12
- SCK on Digital pin 13
- SS on Digital pin 10

If you have Arduino Mega, the following is its SPI pins:

- MOSI on Digital pin 51
- MISO on Digital pin 50
- SCK on Digital pin 52
- SS on Digital pin 53

In general, you learn SPI using MATLAB on this document, <http://www.mathworks.com/help/supportpkg/arduinoio/spi-sensors.html> . To access SPI on Arduino board from MATLAB, we can do the following steps:

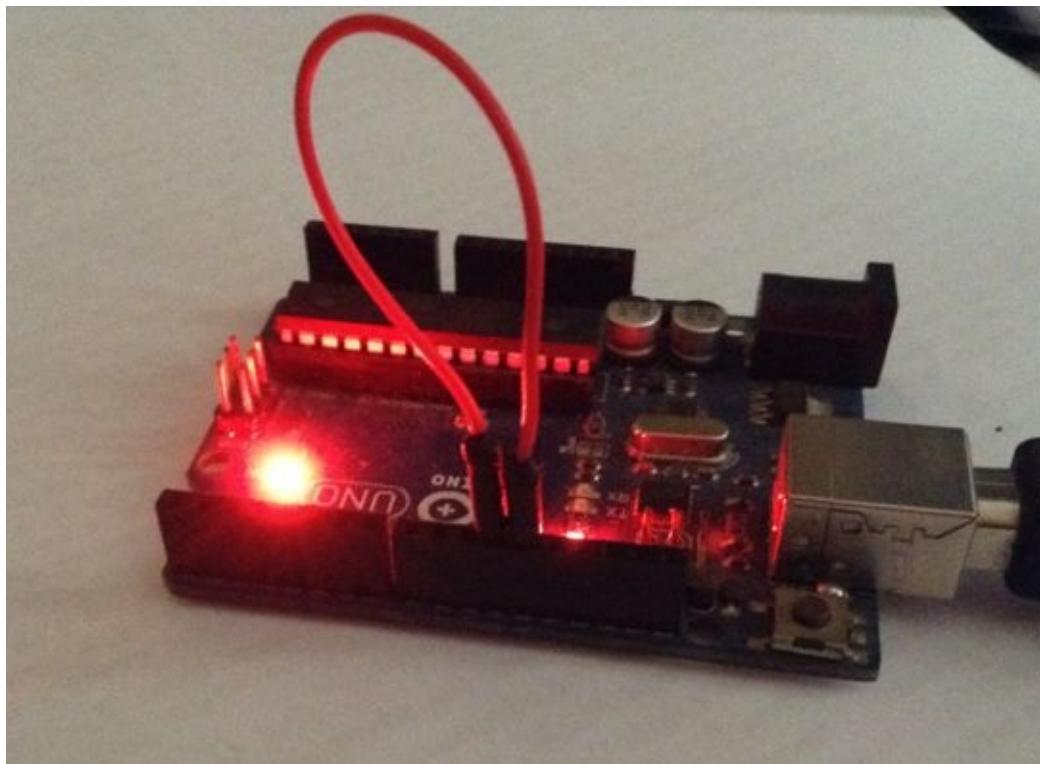
- Open Arduino communication by creating arduino object
- Open SPI connection by calling spidev with passing SS pin, <http://www.mathworks.com/help/supportpkg/arduinoio/ref/spidev.html>
- To write and read SPI data, we can use writeRead with passing spidev object, <http://www.mathworks.com/help/supportpkg/arduinoio/ref/writeread.html>

In this chapter, we build a SPI Loopback app. Let's start!.

6.2 Demo : SPI Loopback

To develop SPI loopback, we can connect MOSI pin to MISO pin. This means you connect pin 11 to pin 12 using cable.

The following is a sample of wiring.



On MATLAB, you can write thses scripts.

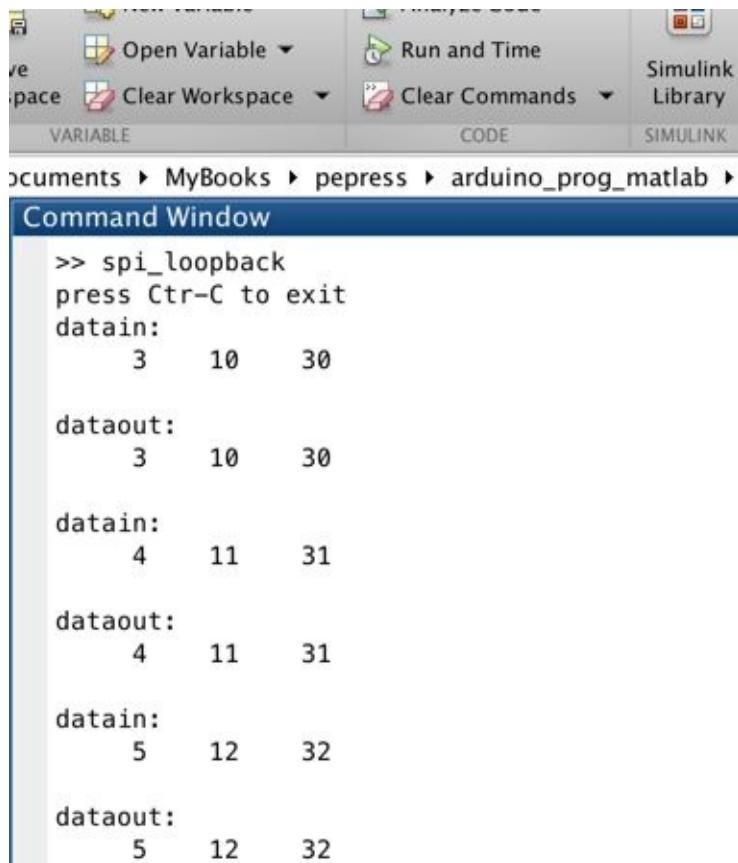
```
function [] = spi_loopback()
board = arduino();
finishup = onCleanup(@() exitprogram(board));
spi = spidev(board, 10);
k = 3;
m = 10;
n = 30;
disp('press Ctrl-C to exit');
while 1
    disp('datain: ');
    dataIn = [k m n];
    disp(dataIn);
    dataOut = writeRead(spi,dataIn);
    disp('dataout: ');
    disp(dataOut);
    pause(1.5);
    k = k + 1;
    m = m + 1;
    n = n + 1;
end
end
```

```
function exitprogram(b)
clear b;
disp('program has exit');
end
```

Save these scripts into a file, called spi_loopback.m. Then, run this program on Command Windows of MATLAB.

```
>> spi_loopback
```

A sample output program can be seen in Figure below.



The screenshot shows the MATLAB interface with the Command Window active. The window displays the output of the 'spi_loopback' script. The output shows data being transmitted in a loop between 'datain' and 'dataout' variables. The data values are 3, 10, 30, 4, 11, 31, 5, 12, and 32. The script includes a warning to press Ctrl-C to exit.

```
>> spi_loopback
press Ctr-C to exit
datain:
    3    10    30

dataout:
    3    10    30

datain:
    4    11    31

dataout:
    4    11    31

datain:
    5    12    32

dataout:
    5    12    32
```

7. Working with Servo Motor

In this chapter I'm going to explain how to work with servo motor on Arduino board using MATLAB.

7.1 Getting Started

Servo motor provides a shaft movement 360 degree. We can control this movement based on its degree. In this scenario, you can use any DC motor (servo) that will be connected to Arduino. I used a mini servo from Arduino Sidekick Basic kit.

The following is a picture of my mini servo motor.



To understand servo's cables, you can identify as follows:

- Red cable is be connected to 5V
- Black or brown cable is be connected to GND
- The rest (yellow or orange cable) is be connected to Arduino PWM pin

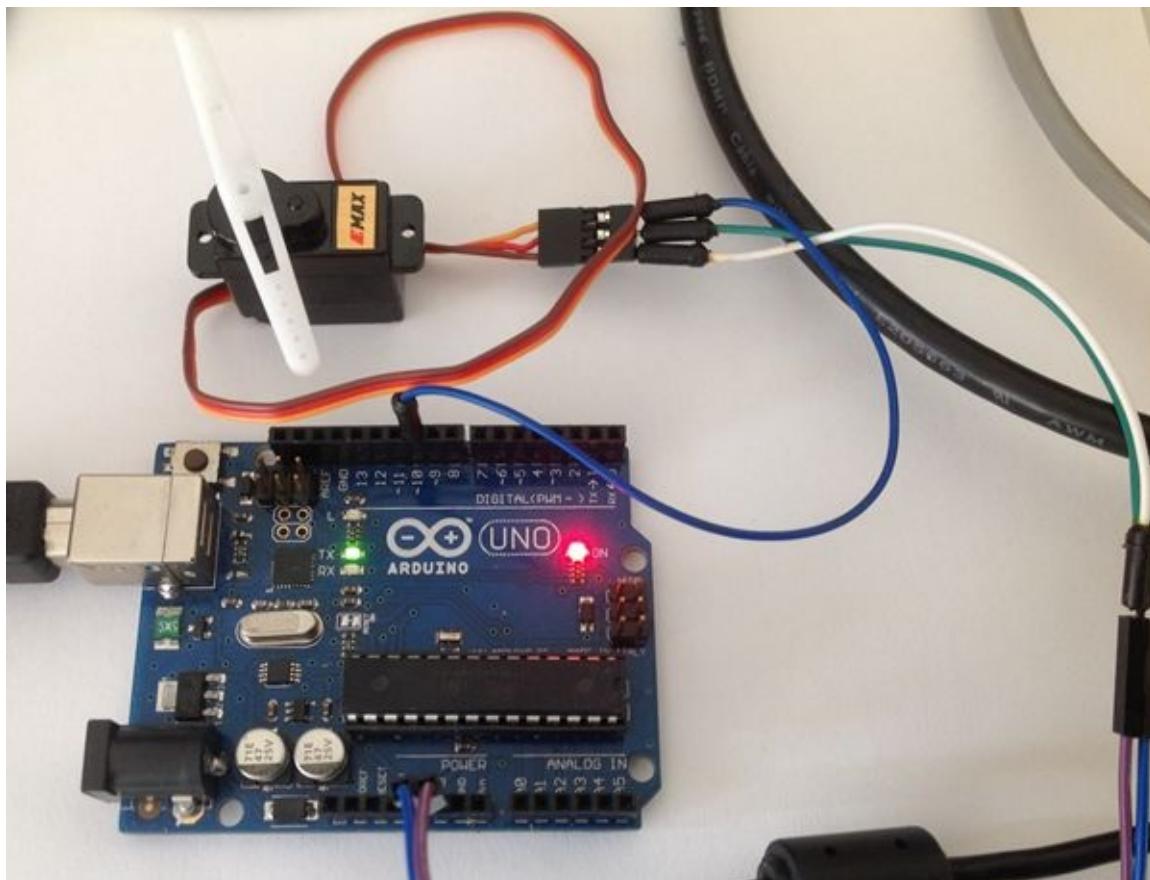
The next step we are going to build a MATLAB program with Arduino and servo motor.

7.2 Wiring

To build hardware implementation, you can connect servo motor to Arduino by following configuration:

- Red cable is be connected to 5V
- Black or brown cable is be connected to GND
- The rest (yellow or orange cable) is be connected to Arduino PWM pin. I used pin 10 for Arduino Uno R3 or Arduino Mega 2560

The following is a sample of hardware implementation.



7.3 Writing a Matlab Program

We can use servo object to control servos. You can read it on <http://www.mathworks.com/help/supportpkg/arduinoio/servo-motors.html> . We build a program to run a servo motor from position 0 to 1 using writePosition() function.

Write these scripts.

```
function [] = servo_motor()
board = arduino();
finishup = onCleanup(@() exitprogram(board));
motor = servo(board, 'D10');
disp('press Ctrl-C to exit');
while 1
    for pos = 0:0.25:1
        disp(['position: ', num2str(pos)]);
        writePosition(motor, pos);
        pause(1);
    end
    for pos = 1:-0.25:0
        disp(['position: ', num2str(pos)]);
        writePosition(motor, pos);
        pause(1);
    end
end

function exitprogram(b)
clear b;
disp('program has exit');
end
```

Save these scripts into a file, servo_motor.m.

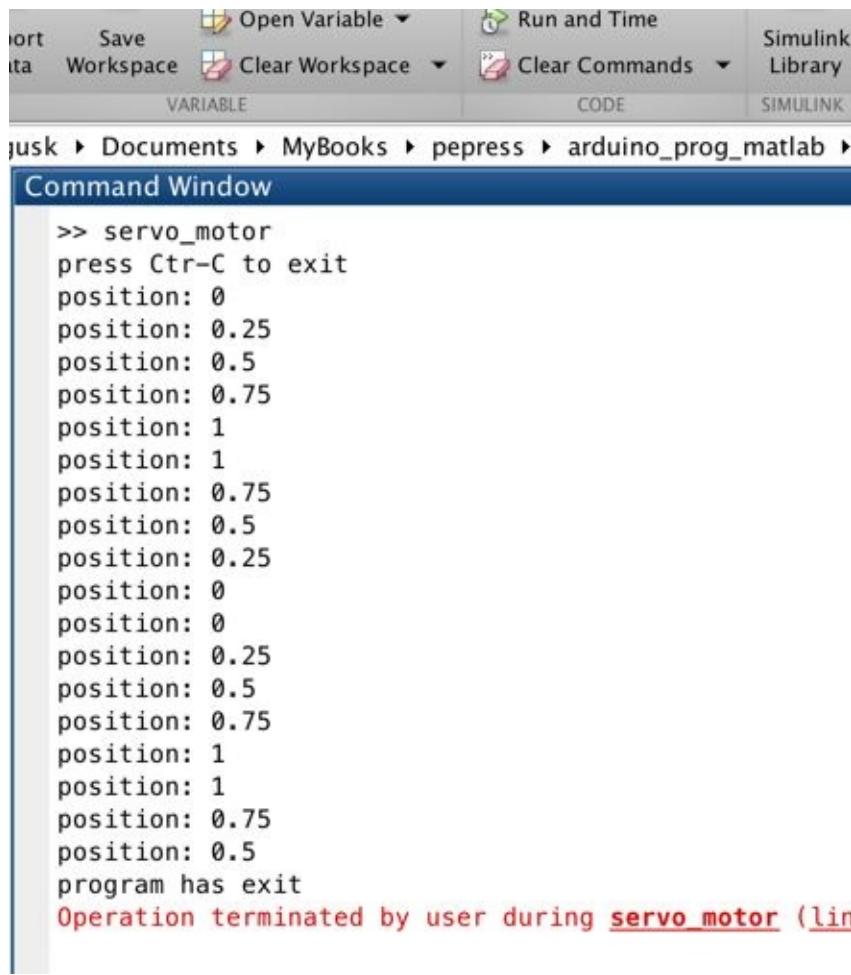
7.4 Testing

Now you can run the program on Command Window from MATLAB.

```
>> servo_motor
```

You should see servor motor is running from degree 0 to 180 and then back again from degree 180 to 0.

A sample output program can be seen in Figure below.



The screenshot shows the MATLAB Command Window interface. The menu bar includes 'File', 'Save', 'Open Variable', 'Run and Time', 'Simulink', 'Workspace', 'Clear Workspace', 'Clear Commands', 'VARIABLE', 'CODE', and 'SIMULINK'. The current directory is 'jusk > Documents > MyBooks > pepress > arduino_prog_matlab'. The Command Window displays the following text:

```
>> servo_motor
press Ctr-C to exit
position: 0
position: 0.25
position: 0.5
position: 0.75
position: 1
position: 1
position: 0.75
position: 0.5
position: 0.25
position: 0
position: 0
position: 0.25
position: 0.5
position: 0.75
position: 1
position: 1
position: 0.75
position: 0.5
program has exit
Operation terminated by user during servo_motor (lin
```

8. Measuring and Plotting Sensor Data in Real-Time

In this chapter I'm going to explain how to read data from sensor devices and plot it on graph in real-time.

8.1 Getting Started

This section has an objective to show how to work with a real-time on measurement. We read data from sensor devices and display it on graph.

Let's start!.

8.2 Wiring

We use the same wiring from section 5.2

8.3 Writing a Program

Now you run MATLAB and write these scripts.

```
function [] = sensing()
board = arduino();
disp('press Ctr-C to exit');
h = figure(1);
finishup = onCleanup(@() exitprogram(board,h));

PCF8591 = '0x48';
PCF8591_ADC_CH0 = '40'; % thermistor
PCF8591_ADC_CH1 = '41'; % photo-voltaic
PCF8591_ADC_CH3 = '43'; % potentiometer
i2c = i2cdev(board,PCF8591);

hLine1 = line(nan, nan, 'Color','red');
hLine2 = line(nan, nan, 'Color', 'blue');
hLine3 = line(nan, nan, 'Color', 'black');
i = 0;
while 1
    thermistor = read_adc(i2c,hex2dec(PCF8591_ADC_CH0));
    pause(0.5);
    photo = read_adc(i2c,hex2dec(PCF8591_ADC_CH1));
    pause(0.5);
    potentiometer = read_adc(i2c,hex2dec(PCF8591_ADC_CH3));
    pause(0.5);

    x1 = get(hLine1, 'XData');
    y1 = get(hLine1, 'YData');
    x2 = get(hLine2, 'XData');
    y2 = get(hLine2, 'YData');
    x3 = get(hLine3, 'XData');
    y3 = get(hLine3, 'YData');

    x1 = [x1 i];
    y1 = [y1 thermistor];
    x2 = [x2 i];
    y2 = [y2 photo];
    x3 = [x3 i];
    y3 = [y3 potentiometer];

    set(hLine1, 'XData', x1, 'YData', y1);
    set(hLine2, 'XData', x2, 'YData', y2);
    set(hLine3, 'XData', x3, 'YData', y3);
    i = i + 1;
    pause(1);
end
end

function adc = read_adc(dev,config)
```

```
write(dev,config);
read(dev, 1);
out = read(dev, 1);
adc = out;
end

function exitprogram(b,h)
clear b;
close(h);
disp('program has exit');
end
```

Save the program into a file, called sensing.m.

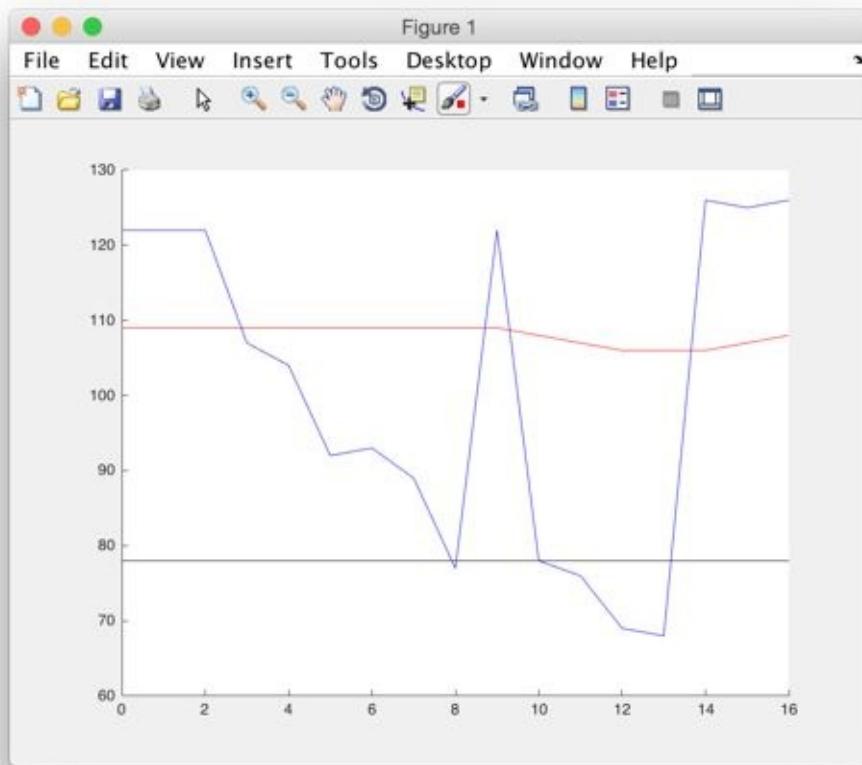
Basically, the program is similar to program from section 5.4. After read sensor data, we display it using line object from MATLAB.

8.4 Testing

You can run the program by typing this command.

```
>> sensing
```

You should see sensor data on Figure.



Source Code

You can download source code on

http://www.aguskurniawan.net/book/matprog_120x15.zip .

Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: <http://blog.aguskurniawan.net>