Serial Communication between Arduino and MATLAB

By-
Aman Mangal
IIT Bombay

June 6, 2012

Prerequisite

- Serial data transfer in Arduino. Follow the link to learn more-
  1. http://arduino.cc/en/Reference/serial
  2. http://www.ladyada.net/learn/arduino/lesson4.html
- MATLAB programming. To learn more, go to-
  1. http://users.ece.gatech.edu/bonnie/book/TUTORIAL/tut_1.html
  2. http://www.mathworks.in/help/techdoc/matlab_product_page.html
- You should be familiar with MATLAB structures, MATLAB objects etc.
- Basic programming concepts.

Serial Communication

- **Serial** means "One after another".
- Serial communication is when we transfer data **one bit at a time**, one right after the other.
- Information is passed back & forth between the computer and Arduino by, essentially, setting a pin high or low.
- Just like we turn an LED on and off, we can also send data. One side sets the pin and the other reads it.
- MATLAB can read/send the data from/to the serial port of the computer and process it.

Buffer

- It is most important to understand the nature of buffer to avoid errors later while writing codes.
- There exists a buffer between the two events of sending and reading the data.
- Say a sensor is streaming back data to your program, more frequently than your program reads it.
- Then the data is stored to a list which we call a buffer.
- One writes data into it and other reads it, may be with different speeds.
- Buffer are of finite length.

Buffer...in detail

- Initially the buffer is empty.
- As new data values come in they get added to the bottom of the list (most recent data).
- If your program reads a value from the buffer, it starts at the top of the list (oldest data).
- Once you read a byte of data, it is no longer in the buffer.
- The data in the second position on the list moves up to the top position
- As soon as the buffer is full and more data is sent, the oldest data gets discarded to make space for new data.
- You have to be smart enough not to loose data.

Buffer...example

- We set up the Serial communication between arduino and PC with a buffer length of 5.

1	
2	
3	
4	
5	

Buffer...example

- The value 10 and 6 are transferred respectively.

1	10
2	6
3	
4	
5	

Buffer...example

- One value is read. The value 10 is the oldest. So it will be read first.

1	6
2	
3	
4	
5	

- Note that 10 no longer exists in the buffer.

Buffer...example

- Now, assume that the buffer is full.

1	6
2	4
3	5
4	9
5	3

- What will happen if we write one more data value, say 12, to the buffer?

Buffer...example

- The buffer will look like as follows-

1	4
2	5
3	9
4	3
5	12

- Note that the oldest data "6" is discarded.
- All the data is shifted up and new value is added as the last entry to the buffer.

Before we continue...

- We have to connect the Arduino board to the PC.
- Each serial port on the PC is labeled COM1, COM2 etc.
- The Arduino will be given a COM Port Number. You should figure it out by following these steps-
  1. Right click on "My Computer icon" and select **Manage**.
  2. Select **Device Manager** in the tree view you will see on the left side in the new window opened.
  3. Find and select **Ports(COM& LPT)** in the center panel.
  4. There, you will find lists of all the ports attached to your computer.
  5. Figure out the one you are concerned with. Refresh the window if you don't find any!

Almost ready...

- Now we are ready for the MATLAB and Arduino serial communication.
- We will only focus upon the MATLAB. We will study how can we set up serial port objects, send and read data from the buffer in MATLAB.
- I assume the you already know how to send/read data in Arduino.
- Follow the link given in the beginning of the tutorial if you wish to learn Serial Communication in Arduino.

Styles used throughout the tutorial

- All the MATLAB commands are written in *italics* and preceded by >>
- The MATLAB output is written in blue color everywhere.
- Use

  >> help 'command'
  >> doc 'help text'

  to get help in MATLAB for any command used in the tutorial.

Setting up Serial Port Object

- We first need to create a serial port object.
- Serial port object is just a name given to that serial port so that we can use it in later commands.

```
>> s = serial ('COM1');
    Serial Port Object : Serial-COM1
        Communication Settings
            Port:               COM1
            BaudRate:           9600
            Terminator:         'LF'
        Communication State
            Status:             closed
            RecordStatus:       off
        Read/Write State
            TransferStatus:     idle
            BytesAvailable:     0
            ValuesReceived:     0
            ValuesSent:         0
```

Setting up Serial Port Object

- This line of command only constructs the object. It does **not check/setup/initialize** the communication.
- This command will still work even if the serial port is **not** connected to any device.
- Many objects can be created for a serial port but only one can be connected at a time.(we will see later)
- This shows all the property of the constructed serial port object.
- In MATLAB, **s** is a structure which has all the above properties. We can access/modify them using dot(.) operator.
- Note that the **Status** is **closed**. It implies that the serial port is not connected.

BaudRate

- It is the rate of transfer of the data in bits/sec.
- We can change the BaudRate using the **set** method as follows-

```
>> set(s, 'BaudRate', 4800);
>> s.BaudRate = 4800;
```

- You can also setup different BaudRate while making the serial port object as follows-

```
>> s = serial('COM1','BaudRate', 4800);
```

- You can verify the change using **get** method-

```
>> get(s, 'BaudRate')
    ans = 4800
```

- The following will also show the similar result-

```
>> s.BaudRate
    ans = 4800
```

BaudRate

- You can also do the following to verify the change-

```
>> s
Serial Port Object : Serial-COM1
    Communication Settings
        Port:               COM1
        BaudRate:           4800
        Terminator:         'LF'
    Communication State
        Status:             closed
        RecordStatus:       off
    Read/Write State
        TransferStatus:     idle
        BytesAvailable:     0
        ValuesReceived:     0
        ValuesSent:         0
```

- Note the new value of **BaudRate** shown.

Properties of Serial port object

- As we have already stated that **s** is of MATLAB datatype called **structure**(similar to structures in C++).
- There are lots of flexibility MATLAB provides to change the properties of the serial port object.
- The commands are similar to what we have used for BaudRate.
- Use following command to list down all these properties and their current value-

  >> *get(s)*
- To see the possible values of all these properties, use-

  >> *set(s)*

Setup the connection

- Before you actually write the data to the serial port, you must connect to device.
- This is like a JAVA lock. Only one entity can acquire the lock at a time.
- Use *fopen* to acquire the lock and setup the connection.

  $>>$ *fopen(s)*
- Notice the status property of the serial port object-

  $>>$ *s.Status*

  ans = open
- If the lock is already acquired, *fopen* will give an error.
- To avoid error, first check the Status property of the serial port object. If it is closed then try to setup the connection.

Writing to the Serial Port in MATLAB

- MATLAB can write any kind of data to the serial port binary, string, int, float etc. with specified precision.
- We use *fwrite* or *fprintf* to write data.
- Transfer an int/float array-

  >> *fwrite(s, vector_array, 'precision');*
- The precision specifies the datatype of the vector_array. It can be 'int8', 'int16', 'float32', 'float64', 'uint8', 'char' etc.
- String-

  >> *fwrite(s, 'string');*
- You can use *fprintf* for strings as well-

  >> *fprintf(s, 'string');*
- You can specify the format in *fprintf* for a string.

Reading from Serial Port in Arduino

- You can follow these steps to read data in Arduino-

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  if(Serial.available() > 0){
    byte b = Serial.read();
    //Process the data
  }
}
```

- You can choose the kind of data you are expecting, otherwise byte datatype can be used.

Writing to Serial Port in Arduino

- Use following function to write data in Arduino to the Serial Port.

```
Serial.println(data);
```

- Note the use of **println** not **print**.
- This will be helpful when we will read the data in MATLAB.

Reading from Serial Port in MATLAB

- **fscanf** is used to read the data-

  $>>$ *data* $=$ *fscanf(s);*
- This will read all the data until it finds a new line/Terminator.
- That is why we used **println** instead of **print**.
- By following this procedure you will be reading all the data sent in one command of **Serial.println** at a time.
- In this case, MATLAB automatically converts data into the correct format and stores into the MATLAB variable.
- If there is no data to read, a time out will occur.

Reading from Serial Port in MATLAB

- To avoid time out, you can first check the **BytesAvailable** property of serial port object just like we did in Arduino-

```
>> if s.BytesAvailable > 0
       data = fscanf(s);
   end
```

- You can specify the size of the data you want to read in case you use **Serial.print**.
- You have to specify the format of data together with the size of the data.

```
>> fscanf(s, 'format', size);
```

fread in MATLAB

- You can also use *fread* instead of *fscanf* to read the data from serial port.
- *fread* reads until the Terminator is seen.

  $>>$ *data* $=$ *fread(s)*
- *fread* doesn't convert the data in the correct format until you specify it.
- It is better to specify size while using *fread*.
- You also have to specify precision in *fread* if you want to specify size.

  $>>$ *data* $=$ *fread(s, size, 'precision')*
- Try using *fwrite* & *fscanf* for writing and reading the data.

Format

- Format is basically C conversion specification language.
- The following formats can be used-

Field Type	Specifier	Details
Integer, signed	`%d`	Base 10
	`%i`	Base determined from the values. Defaults to base 10. If initial digits are `0x` or `0X`, it is base 16. If initial digit is `0`, it is base 8.
Integer, unsigned	`%u`	Base 10
	`%o`	Base 8 (octal)
	`%x`	Base 16 (hexadecimal)
Floating-point number	`%f`	Floating-point fields can contain any of the following (not case sensitive): `Inf`, `-Inf`, `NaN`, or `-NaN`.
	`%e`	
	`%g`	
Character string	`%s`	Read series of characters, until find white space.
	`%c`	Read any single character, including white space. (To read multiple characters, specify field length.)
	`%[...]`	Read only characters in the brackets, until the first nonmatching character or white space.

Size

- There are 2 ways-

`n`	Read at most `n` values into a column vector.
`[m,n]`	Read at most m-by-n values filling an m-by-n matrix in column order.

- The filling in the matrix takes place columnwise. The first column is filled, then 2^{nd} column and so on.
- This will keep reading until the vector/matrix of the given size is created.
- Time out will occur if the available data is not enough to fill the whole vector/matrix.

BufferSize

- It is very important not to loose the data while the data gets transferred.
- As mentioned before, the data gets discarded if the **Buffer** is full to accommodate new data.
- To avoid loosing data, specify almost equal Baudrates in both the sytem MATLAB & Arduino.
- We have already seen how we can control the BaudRate.
- We can specify the BufferSize in MATLAB according to the need of program using following methods-
  - *set(s, 'BufferSize', 1024)* *%in bytes*
  - *s.BufferSize = 1024;* *%in bytes*

(Similar to BaudRate)

Read/Write State

- Each time you transfer the data, it is either receive or sent by MALTAB.
- The **ValuesSent** & **ValuesReceived** properties of serial port object shows these values.
- You can always check these values to ensure the correct amount to data transferred.
- You can also optimize your program to use the least possible size datatype for the data to be transferred.
- We have already seen the **BytesAvailable** property.
- **TransferStatus** shows whether the data transfer is complete.
- *fprintf* blocks the command line to execute other commands while the data is getting transferred while *fwrite* doesn't.

End the Connection

- This is very important step while Serial Communication.
- Make sure the execution of this step even in case the data is not transferred.
- Use *fclose* to end the connection-

  >> *fclose(s)*
- Delete the serial port object if you are done with communication.

  >> *delete(s)*

  >> *clear s*
- So if *fopen* is executed then *fclose* must be executed anyhow otherwise it will create problem when you use it next time.
- You should use try-catch statement to ensure that the serial port is closed before the program ends execution.

To remember

- You cannot simultaneously view the data in MATLAB and Arduino serial port because the data can only be read once.
- The data is removed as soon as it is read first time.
- The synchronisation of data transfer is very important while communication.
- Always check for the property before setting its new value.
- For example before you use *fopen*, check whether the serial port is already used.
- Check the **BytesAvailable** property before you read the data in MATLAB.

To remember

- Be selective while you choose the datatype of the data. It will affect the time taken in communication.
- Choose an optimum **BufferSize**.
- Always close the connection in the end so that we can use the port next time.
- Close the connection in MATLAB before you upload your code to Arduino.
- Whenever you setup a new connection, it flushes all the data sent/received earlier.

INSTRFIND

- The function *instrfind* finds all the possible, existing serial port objects.
- It returns an array of serial port object.

```
>> h = instrfind
```

Instrument Object Array

Index:	Type:	Status:	Name:
1	serial	closed	Serial-COM8
2	serial	closed	Serial-COM8
3	serial	closed	Serial-COM1

- You can access any of the object using index like **h(2)**

```
>> s = h(2);
```

- Now s is similar serial port object which we created in the beginning of the tutorial.

INSTRFIND....advanced

- You can specify the properties of the serial port object you want to search for-

  >> *out1 = instrfind('Port','COM1');*

  >> *out2 = instrfind('Port','BaudRate','COM2',4800);*

- Note that *instrfind* returns array/vector of serial port object.
- It is used when you want to retrieve a deleted serial port object.

More to come

- We have learnt all the basics of Serial Communication in MATLAB.
- We can extend this knowledge to make a real time serial communication system.
- You can make a system which reads the data from a digital pin of Arduino and transfers all the data to MATLAB in real time.
- Plot the received data on the graph in MATLAB.
- The pin number may be specified by the user in MATLAB.
- I will present the solution of the problem after a week.

Thank you

- Thanks for your patience while reading the tutorial.
- I hope you find it useful.
- Use MATLAB help to know more about anything we have learnt here.
- Contact me for any query: **mangalaman93@gmail.com**