

Application Note: MatLab data processing for Motion Capture

Zhichao Lu – Team 8 – Documents Prep.

Date: April 5, 2013

Executive Summary

This application note is a tutorial that guides users through how to use MatLab to process raw data, that's coming from a serial port connected devices, with examples of main functions involved. A MatLab based self-written comparison model, that's used to evaluate running forms, will be also illustrated with pros and cons of using MatLab compared to other real-time processing software like LabView. This application note is developed based on MathWorks Serial Port Devices manual (1). All related MatLab files and codes are attached in the end.

Keywords

MatLab, Data Process, Serial Communication, Data Conversion, Plotting

Introduction

The team is tasked to design a product to capture the motion of the runners. This product will analyze the form of runner and compare it to an elite runner under various conditions. In order to accomplish this ultimate goal, MatLab is chosen for data processing, comparing and plotting because of MatLab's various built-in functions for easy software development, great computation power for fast data conversion, compatibility for serial port communication devices and numerous supporting plot options.

The main functions used in the team's developed MatLab software are the following:

Serial () – Create serial port object;

Fopen () – Connect serial port object to device;

Readasync () – Read data asynchronously from device;

Fscanf (serial) – Read data from device, and format as text;

Fwrite () – Write binary data to device;

Fprintf () – Write text to device;

Fclose (serial) – Disconnect serial port object from device;

For – Execute statements specified number of times;

Stem – Plot discrete sequence of data;

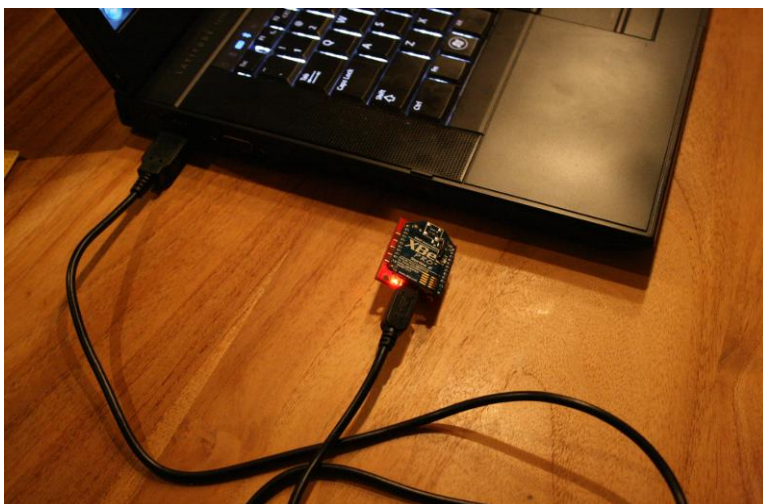
Examples will be provided for each of the above functions on how to use them in the ‘Hardware Connection and Software setting’ section.

Objectives

- 1) This MatLab based data processing software must be able to continuously read data from a serial port connected device, in the meanwhile, decode the raw data and correctly align them into matrices.
- 2) This MatLab based data processing software must also be able to continuously comparing processed data to elite runner and trigger feedback signal that sent back to a serial port connected device if designed conditions satisfied.
- 3) This MatLab based data processing software must be able to save data into files, plotting current user’s data against elite runner for post-analysis.

Hardware Connection and Software setting

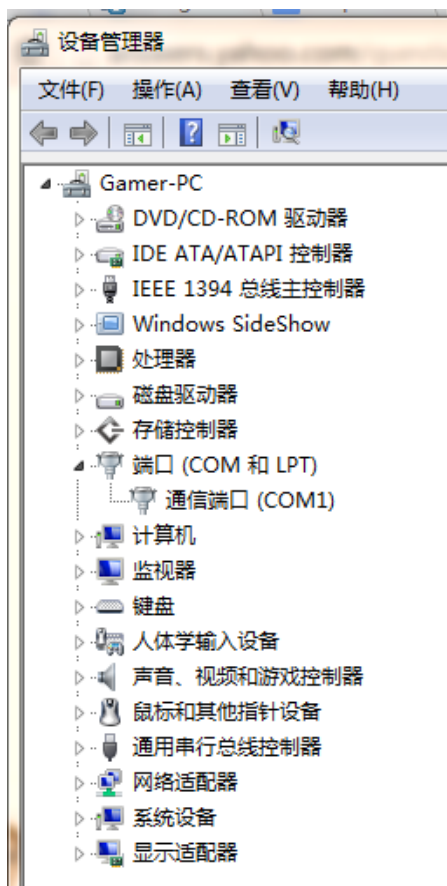
Hardware connection:



An Xbee transceiver is attached to an explorer board that's connected to a PC via USB cable as shown above. This will be the serial object that user need to create in MatLab environment to read data from.

Software Set-up

After proceeding to MatLab environment, a serial object needs to be created to initiate serial communication. First of all, the user needs to identify which COM Port the Xbee transceiver is connected to by clicking My Computer / Properties / Device Manager, choose 'Ports or any other device', the COM Port number should appears like below



In this case, the Xbee transceiver is connected to COM1, then user could initiate the serial communication in MatLab Environment by enter the command 's = serial ('COM1')', MatLab should response as following:

```
>> 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
```

fx >>

MatLab creates a new serial object named `s` and displays its current status. MatLab's default sending speed for serial port communication is 9600, thus, the BaudRate of this serial object could be changed by enter the command, `set(s, 'BaudRate', 115200)`, type '`s`' once more after setting the baudrate, MatLab will display the current status again. The new baudrate should be updated to the serial object.

```
>> set(s, 'BaudRate', 115200)
>> s

Serial Port Object : Serial-COM1

Communication Settings
  Port:          COM1
  BaudRate:      115200
  Terminator:    'LF'

Communication State
  Status:        closed
  RecordStatus:  off

Read/Write State
  TransferStatus: idle
  BytesAvailable: 0
  ValuesReceived: 0
  ValuesSent:     0
```

One last thing to do before MatLab could read incoming data from serial port is to turn on the connection by entering the command 'fopen(s)', until now, this new serial object is successfully initiated. There're several other options that user should specify, for instance, changing the reading mode to manually asynchronous by entering 's.ReadAsyncMode = 'manual' '. The serial object should appear as the picture shown below.

```
>> s

Serial Port Object : Serial-COM1

Communication Settings
  Port:          COM1
  BaudRate:      115200
  Terminator:    'LF'

Communication State
  Status:        open
  RecordStatus:  off

Read/Write State
  TransferStatus: idle
  BytesAvailable: 0
  ValuesReceived: 0
  ValuesSent:     0
```

Steps Examples and Related Issues

The Team's developed MatLab software converts IMU collected raw data, which contains nine values corresponding to accelerometer, gyroscope and magnetometer respectively, to usable data by converting the units and resolution. Then the software compares every single data entrée in the matrices to the baseline data matrices and triggers feedback signal if conditions satisfied. This software utilizes basic functions; three key functions will be explored further below with examples.

- Str2num – converts character string to numbers

The incoming raw data is in the format of character string consists of nine values. MatLab sees this data as no differences with a text sentences even it only contains numeric values.

MatLab would not be able to correctly exacts every value without converting the string to row vector. Here's an example:

```
>> data = '100 10 20 44 36 90'

data =

100 10 20 44 36 90

>> data(1)

ans =

1

>> data(2)

ans =

0
```

If the data is in correct format, command data (1) should returns 100 as the first entrée of the vector instead of 1; after using str2num function, data will be converted to a row vector.

```
>> data_new = str2num(data)

data_new =

    100     10     20     44     36     90

>> data_new(1)

ans =

    100

>> data_new(2)

ans =

     10
```

Now, data is in correct format and ready to be used for transformation.

- Fprintf - Write text to device.

This function is used to trigger the feedback signal to serial port. As compared to Fwrite function, Fprintf is chosen because of two reasons. First, the receiver on Arduino

microcontroller treats any incoming signal as string, thus, there'll be no meaning to use Fwrite which sends a binary signal to serial port device; In addition, descriptive words could be used as control signal using Fprintf function for better demonstration.

```
>> fprintf(s, 'Arm')
fx >>
```

This example shows the arm positioned IMU triggers the feedback signal, a video demo is available on ECE480 Team8 website for better understanding of this function.

- Stem - Plot discrete sequence of data

Regular Plot function MatLab provides wouldn't be able to plot discrete time sampled data; it requires at least the X-axis data set to be in continuous scale. In the team's case, IMU collected data is only valid at specific time slot, hence, Stem function is chosen.

Below is a complete preliminary data processing and plotting MatLab software

```
% Data Collection
for i = 1:dataNum
    out = fscanf(s);
    val = str2num(out)./ 16384;
    ax_raw(i) = val(1);
    ay_raw(i) = val(2);
    az_raw(i) = val(3);
    gx_raw(i) = val(4);
    gy_raw(i) = val(5);
    gz_raw(i) = val(6);
    mx_raw(i) = val(7);
    my_raw(i) = val(8);
    mz_raw(i) = val(9);

    ax_sort(i) = asind(val(1));
    ay_sort(i) = asind(val(2));
    az_sort(i) = asind(val(3));

    Sensor_Data = real([ax_raw.', ay_raw.', az_raw.', gx_raw.', gy_raw.', gz_raw.', mx_raw.', my_raw.', mz_raw.'])

    i = i+1;
end

%Comparison Development

%ploting operation
for i = 1:dataNum
    subplot(3,3,1)
    stem(i,ax_raw(i),'LineStyle','none',...
        'MarkerEdgeColor','k',...
        'MarkerFaceColor','g',...
        'MarkerSize',2);

    title('Accelerometer g value X-axis');
    xlabel('time(sec)');
    ylabel('g value');
    hold on;

    subplot(3,3,2)
    stem(i,ay_raw(i),'LineStyle','none',...
        'MarkerEdgeColor','k',...
        'MarkerFaceColor','g',...
        'MarkerSize',2);

    title('Accelerometer g value Y-axis');
```

Issues and Limitations

Due to the nature of MatLab software, the team's design comparison software is not truly a real-time system, it does take certain delay before the control signal feeds back to user body-worn controller, and the delay time is dependent on how long has this software been running.

Also, the running time of this software needs to be pre-determined before starting the program.

Unlike most of the microcontroller program, MatLab software is not designed to be running in an infinitely loop, the program has to leave the loop and returns the value to user at certain point.

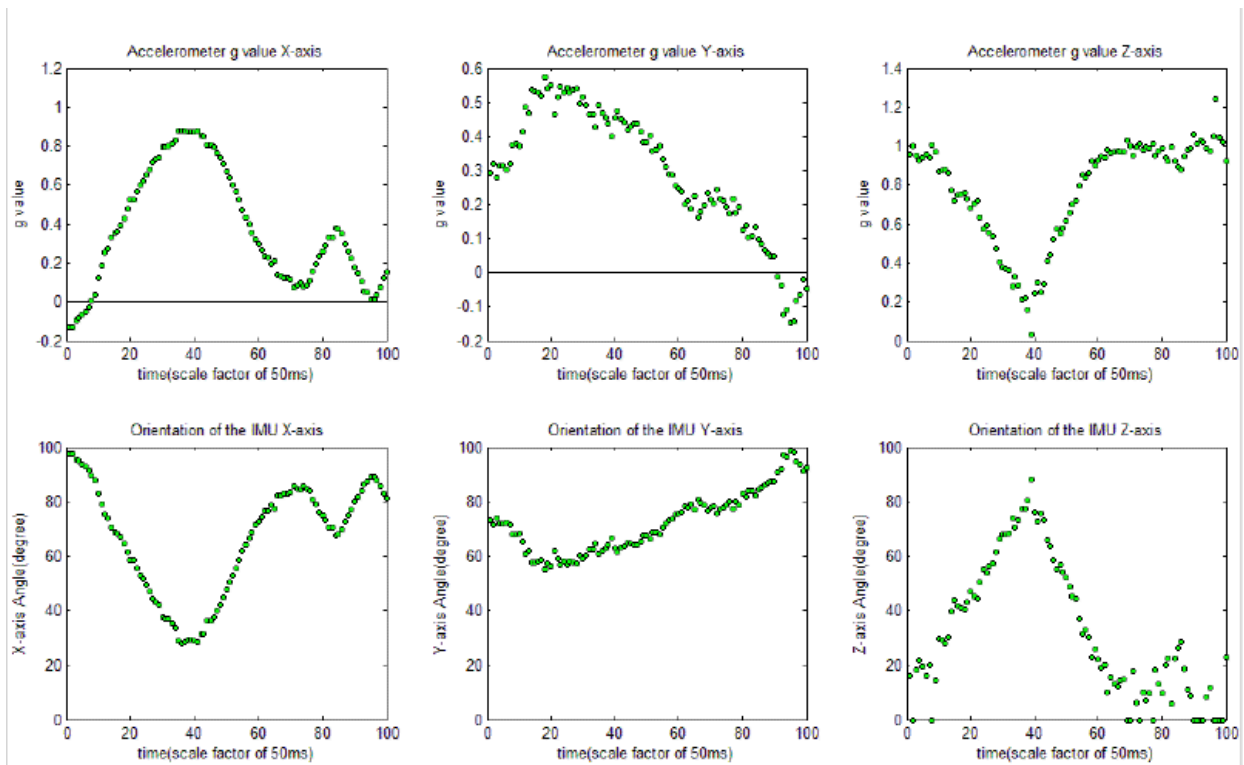
Results

This is a typical output data seen from command window after unit conversion. Both output data and plotting graph are obtained using MatLab file named, "Demonstration", which is attached in the end. As a quick check to verify output data, three numbers from accelerometer should add up to one, if IMU is moving at relatively low speed or placed still at certainly position. In this case, the total acceleration on IMU could be assumed to one gravity constant that's due to the Earth gravity.

Right below MatLab command window output data, it's a sample plot from accelerometer g value versus converted angles. Again, if IMU is moving at relatively low speed or placed still at certain position, one gravity constant should corresponding to zero degree, since the Earth gravity aligns with IMU's particular axis in this case, as a quick check.

Sensor_Data =

-0.0874	-0.0063	0.9988	0.0007	-0.0021	-0.0115	0.3125	1.9375	0.7500
-0.0945	-0.0071	0.9946	0.0028	-0.0028	-0.0092	0.2500	1.9063	0.7500
-0.0964	-0.0098	0.9985	0.0012	-0.0032	-0.0104	0.3438	1.9375	0.7500
-0.3870	-0.1055	0.8811	0.4000	0.5253	-0.7114	0.6250	-1.5625	0.1875
-0.7341	-0.3003	0.1619	0.0465	0.5929	-0.3829	0.8125	-1.4375	-0.7032
-0.8528	-0.4543	0.4436	-0.5394	-0.2488	-0.1910	0.5938	-1.3438	-0.3594
-0.3008	-0.2705	1.0210	-1.0753	-1.6768	0.2100	0.7344	1.7344	0.8594
-0.3896	-0.8264	0.4927	0.2888	-0.1282	0.2668	1.6406	1.0781	0.0781
-0.0068	-0.7773	-0.5664	0.8270	0.5158	0.8028	1.4844	1.1094	-1.9063
-0.0344	-0.4436	0.9749	0.4791	-0.1563	0.1074	1.0156	1.2969	1.2969
0.5940	-0.7371	0.6694	-0.0300	-0.1024	-0.2951	1.6094	-0.0313	0.3594
0.6707	-0.1829	0.8813	0.1022	0.3044	-0.2632	0.8594	0.1094	0.3594
0.0505	-0.4031	0.8042	-0.4780	-0.5838	-0.7383	0.9531	1.4219	0.4844
-0.1572	-0.4141	0.7622	-0.1511	-0.8360	0.0771	0.4844	1.9531	0.9219
-0.1074	0.0342	1.0605	-0.1814	-0.4185	0.0873	0.4844	1.4844	0.9219
0.1101	-0.0112	0.9985	0.0012	0.0073	-0.0084	0.4531	1.4844	0.7656
0.0967	-0.0071	0.9956	0.0002	0.0021	-0.0087	0.4688	1.5625	0.7188
-0.0103	0.0210	0.9973	-0.0030	0.1880	0.1072	0.4844	1.6719	0.8281
-0.7756	0.0879	0.2793	-0.2370	0.5374	-0.2102	-0.0469	-0.7500	-0.6719
-0.9470	-0.0215	-0.1638	0.0172	0.1395	0.0176	-0.0313	-0.8906	-1.5938



Conclusions and Recommendations

The team's designed MatLab software successfully obtains data from a serial port device, aligns raw data to organized matrices, convert raw data to usable information and perform points to points comparison with baseline data set as well as feeds back control signal in time if conditions applies. However, there're few aspects of the software could be modified for better functionality, namely,

1. A self-check function could be added to detect the size of the transformed data set, if the size is over certain limit, saves current data to a file in hard drive and deletes all data to prevent propagation delay keeps increasing as the size of data set growing.
2. The team's designed MatLab software could be divided into sub-scripts and sub-functions instead of using one main file to improve process speed and more user-friend. The data collecting part of the software should be running all the time, but not for comparison and plotting functions of the software. Thus, if reconstructing plotting section of the codes to a separate function that's only called when user specified, it'll improve the process speed of the main software as the software won't go through all the codes in every iteration. In addition, it will only provide user the ability to pass a control variable to the plotting function to specify what data to be plotted to improve the software more user-friendly.

Reference:

(1) MathWork Serial Port Devices

<<http://www.mathworks.com/help/matlab/serial-port-devices.html>>