Final Report – Odometry System

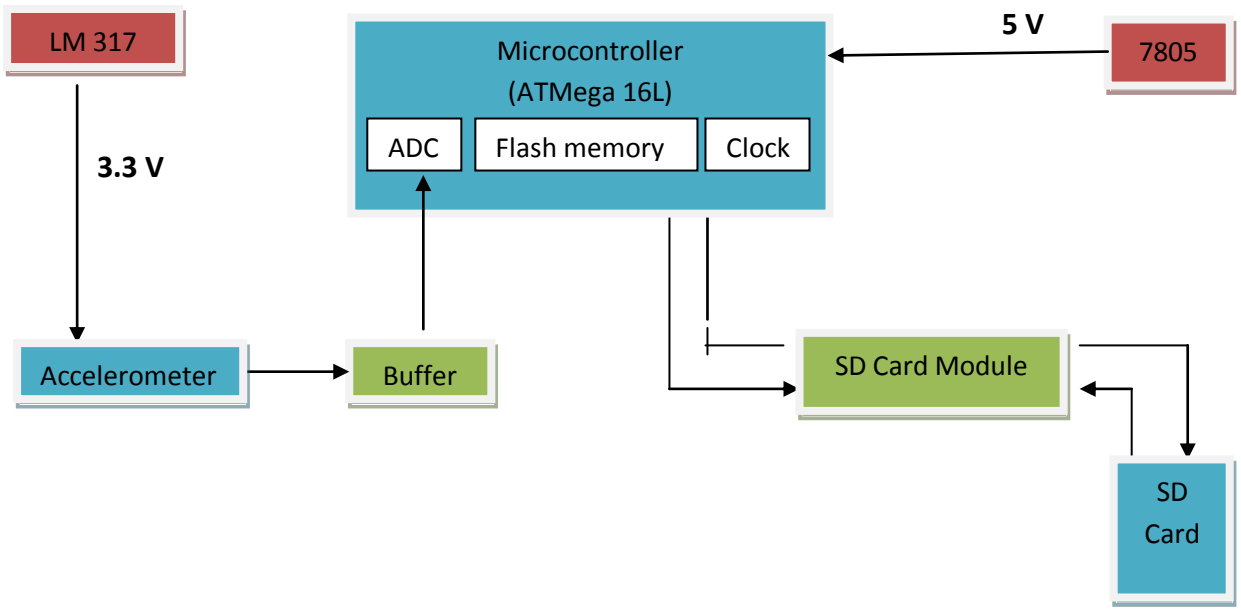
Electrical Design Lab, Spring'09

By: Arun Mukundan (06D07009), Sunny Mahajan(06D07025) and Himesh Joshi(06D07031)
Under the supervision of: Prof. Mukul Chandorkar
Group DD1

Problem Statement

To record the odometric data using a signal conditioned 3-axis accelerometer and log the data into a SD Card and subsequently, plot the trajectory on a computer with the help of the recorded data and MATLAB.

Block Diagram



Block Diagram for the Overall Circuit

An Overview of the functional parts of the circuit

MMA7331L – The Accelerometer

Usage and Application

The accelerometer used for the project is a 3-axis low g accelerometer acquired from Freescale.

The Basic Working

The device basically consists of a g-cell which does the sensing part.

The **g-cell** is a mechanical structure formed from semiconductor materials (polysilicon) using semiconductor processes (masking and etching). It can be modelled as a set of beams attached to a movable central mass that move between fixed beams. The movable beams can be deflected from their rest position by subjecting the system to acceleration.

As the beams attached to the central mass move, the distance from them to the fixed beams on one side will increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration. The g-cell beams form two back-to-back capacitors (Figure 3). As the center beam moves with acceleration, the distance between the beams changes and each capacitor's value will change, ($C = A\varepsilon/D$). Where A is the area of the beam, ε is the dielectric constant, and D is the distance between the beams.

So, we'll be having an equivalent circuit consisting of two capacitors in series.

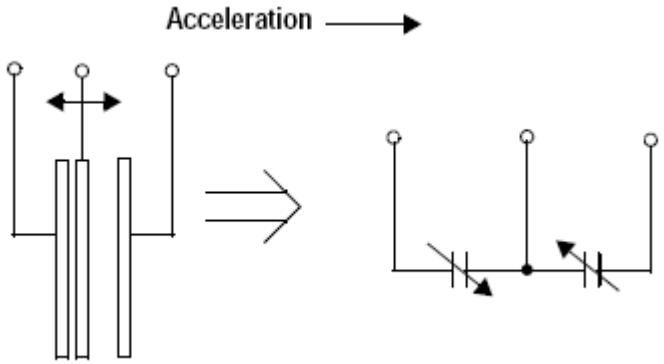
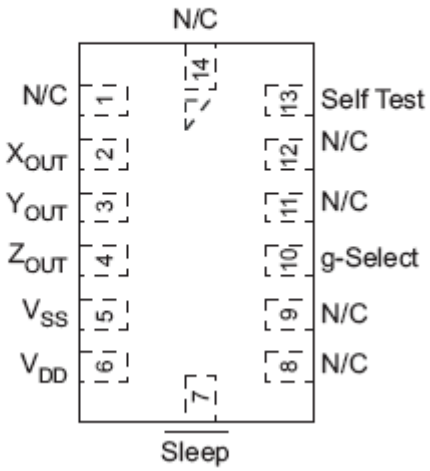


Figure : g-cell and the equivalent capacitor circuit

The Accelerometer samples the signal at a frequency of 11KHz. The A/D sampling rate and the external power supply switching frequency should be used such that they don't interfere with the acceleration sampling frequency.

Pin Descriptions

Top View



Pins 1, 2 and 3 give the direction output voltages which are fed to the ADC of the microcontroller (ATMega 32 in our case). To minimise the noise, capacitors (around 3.3 nF) were used at the outputs of these pins. These basically serve as low pass filters.

The **g-select** pin is used to select the g-mode. Basically, there are 2 modes available, the 4g mode and the 12g mode. In our case, we shall be using the **12g mode**.

g-Select	g-Range	Sensitivity
0	4g	308 mV/g
1	12g	83.6 mV/g

As we can see, the two modes have different sensitivities and according to the use, the sensitivity is selected.

The Sleep pin is used to select the sleep mode (if required). When in sleep mode, the device switches off all the functions and helps in saving power.

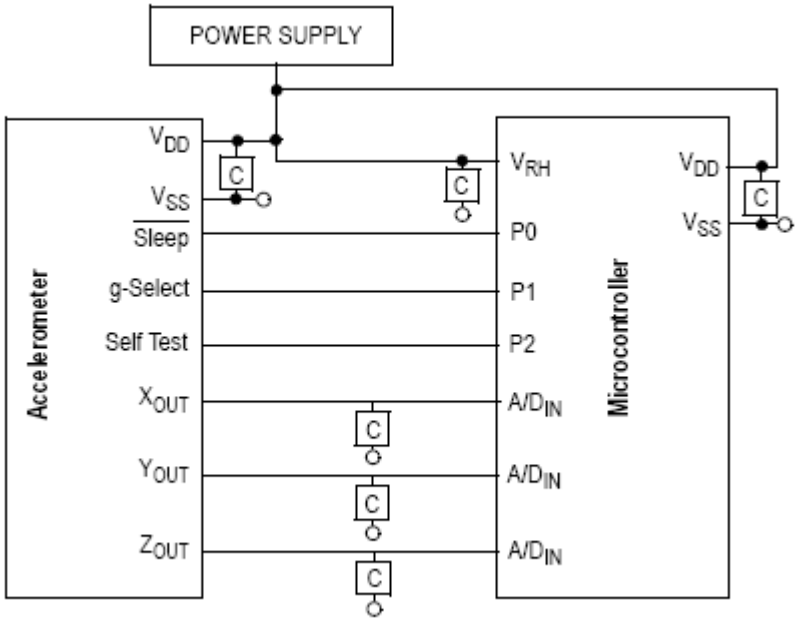
The pin 5 and 6 are the power pins, Vss being the ground pin and Vdd being the supply pin. Note that the device can take values of the voltage from 2.2V to 3.6 V. Also, a 0.1 uF capacitor should be used with the Vdd pin to decouple the power source.

PCB Layout for attaching the Accelerometer to the Microcontroller

The output pins are connected to the ADC input pins. Note the capacitors attached to achieve the reduction in noise.

The sleep, g-select and the self-test pins are attached to the pins P0, P1 and P2 of the microcontroller.

Also, an important task is attaching the capacitors at the supply to decouple the power source.

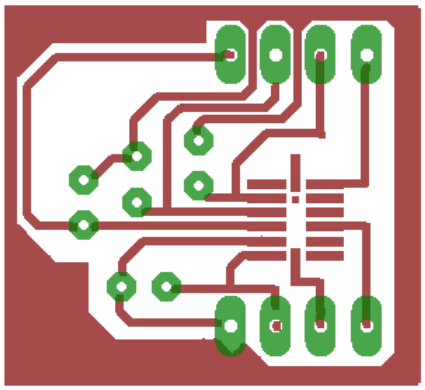


Development of Board for the device

The package of the accelerometer MMA7331L is a 14 pin LGA (Lead Grid Array). So in oder to use it, we had to create a footprint and also design a board for it. To solder it, we applied solder to both the footprint and the chip, and then let the solder melt in order to connect the pins.

For the sake of developing the board, we defined a new library file for the accelerometer.

The screenshot of the board is attached.



We also created a library file in order to use this board as one coherent unit while making the schematic of the final circuit.

ADC (Analog to Digital converter)

For the ADC, we chose a frequency of 12MHz and a prescalar of 64. These values were used because this allows for a ADC frequency of 187.5 KHz which lies between the specified range of 50-200kHz specified for 10 bit operation. Of all the possible combinations, this combination provided the maximum value of the ADC frequency which lies in the specified range.

We also used a voltage follower as an analog buffer before connecting the accelerometer to the ADC as the line to the accelerometer was supposed to be a high impedance one.

Also, with these values, it was possible to get a high number of clock cycles (12000 – 7680 = 4320) which is more than enough for our microcontroller operation.

The use of interrupts

In the initial stages, interrupts were not used and the basic ideology was just to check for the values in a given time and compute the position by integrating the acceleration over the given time. But this way of computation is highly erroneous and is not preferred because emulating floating point operations puts greater amount of load on fixed point only AVR processors.

Calculation of Position, Calibration and Movement-End Check for the Accelerometer

Calibration

This has been achieved by taking a constant number of values of the acceleration on each of the axis and taking the average to get the values which need to be subtracted from the acceleration values in order to obtain the real acceleration.

```
double getcalval(int channel,int iterations)                    //Calibration vlaues
{
setCh(channel);
double sum=0;
for (int i=0;i<iterations;i++) {sum+=(1/iterations)*read_ADC;}
return (0.01953125 * sum);
}
```

Movement End Check

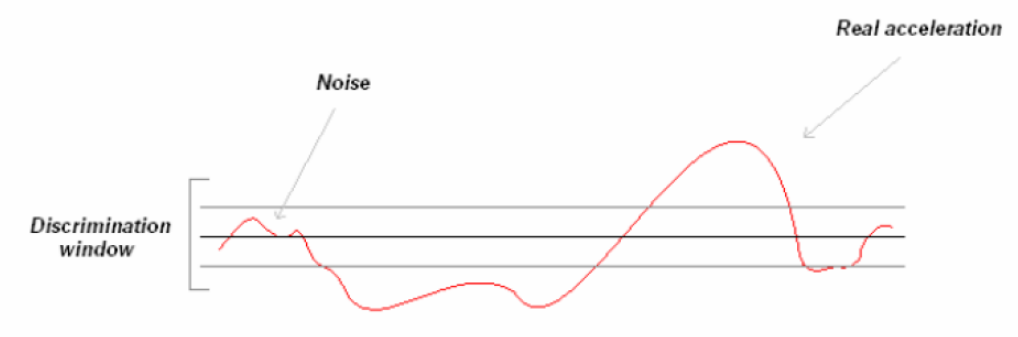
Even after the device has obtained zero acceleration, it is assumed that the velocity is still unchanged. But in our case, where the device is fit into the shoe of a walking person, such a situation is not possible for a prolonged period. So, to ensure that the device doesn't go on calculating the position assuming a constant velocity, we put in what is known as a movement end check.

Basically, we just check if the acceleration value remains unchanged for a certain time. If that is the case then the velocity is also set to zero.

```
 if (ax[1]==0)
 { countx++;}
else { countx =0;}
 if (countx>=25)
 {    vx[1]=0;    vx[0]=0;    }
```

Filtering

Under the no-movement condition, small values of accelerometer might arise because of noise. So, we need to implement a filtering mechanism which defines a noise margin and thus, helps in discriminating between real acceleration values and noise.



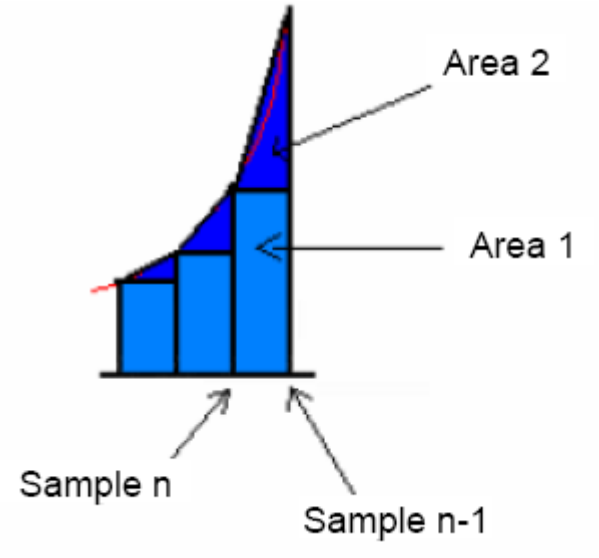
Defining of a discrimination window/noise margin to reduce mechanical noise effects

Implementation:

```
if ((ax[1] <=3)&&(ax[1] >= -3)) //Discrimination window applied to
{accelerationx[1] = 0;}          // the X axis acceleration variable
```

Calculation of Position

In order to obtain the position, we need to integrate the acceleration twice, first, to get the velocity and the second time, to get the position. The integration is done using the first principle, that is, by calculating the area under the curve. Since our sampling times are so small, the area under the curve can be approximated to be the some of the areas of a rectangle and a triangle.

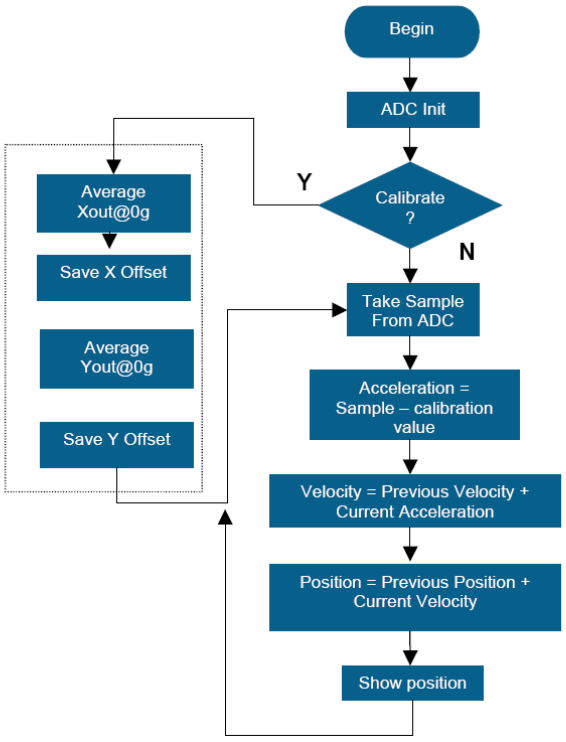


$$Area_n = Sample_n + \frac{|Sample_n - Sample_{n-1}|}{2} \times T$$

Thus, for obtaining the position, we use:

vx[1] = vx[0] + ax[0] + ((ax[1] - ax[0])>>1)

x[1] = x[0] + vx[0] + ((vx[1] - vx[0])>>1);



The basic scheme for implementing the position algorithm using accelerometer

SD Card

SD Card, or Secure Digital Card is an important part of the whole device as it is used for logging the data, i.e. the position values and is then, used to transfer the data onto a computer.

SD Card	DS80C400 SPI Header J21
CLK	Pin 1
DI	Pin 3
V_{CC}	Pin 4
DO	Pin 5
GND	Pin 6
CS	Pin 7

Fig. Pin Configuration for a SD Card

The SD Card usually interacts with a microcontroller in the SPI mode. For that purpose, it has 4 lines connecting it to the microcontroller, MOSI (Master Out Slave In), MISO (Master In Slave Out), Clock and SS (Slave Select). The card can either be directly interfaced with the microcontroller or an SD Card module can be used which has a microcontroller of its own interacting with the SD Card through SPI mode and the module itself uses the UART mode for interfacing with the main microcontroller.

Initialising the SD card

The SD card, by default is in the SD bus protocol. To switch the card to SPI mode, the host has to issue the command 0 (GO_IDLE_STATE). The SD card detects SPI mode selection by observing that the active-low card select (active low CS) pin is held low during the GO_IDLE_STATE command. The card responds with response format as shown:

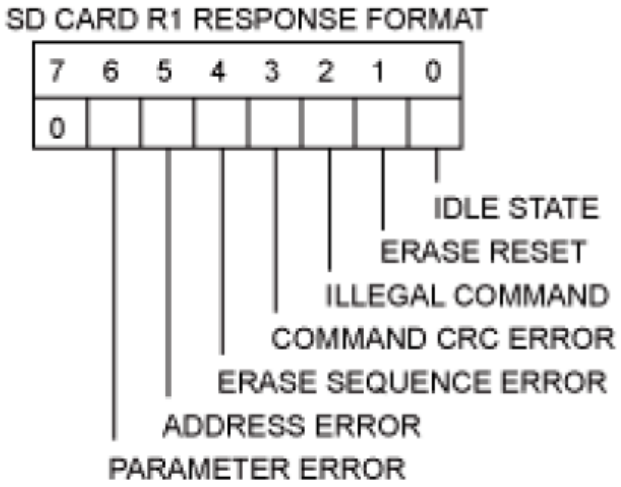


Fig. SD Card response on being given the command 0(GO_IDLE_STATE)

The Idle bit is set high to signify that the card has entered idle state.

After that, comes an important step, which is required to differentiate between an MMC Card and a SD Card. SD cards implement an alternative initialization command to which MMC cards do not respond to. Sending Command 55(APP_CMD) followed by Command 41(SEND_OP_COND) completes this step. As MMC cards do not respond to command 55, that command is used to reject MMC media. This command is repeated until all the bits in the response R1 are low, that is, the device goes out of its idle state.

Now, instead of connecting the SD Card directly to the microcontroller, we have used an SD Card Module which has an inbuilt microcontroller interacting with the SD Card in the SPI mode. The module is connected to the main microcontroller through the UART mode.

Precaution: The UART mode requires a very accurate clock and thus, when using the module, instead of using the internal clock, we should use an external clock which is far more accurate.

Testing and the techniques used

1. **Accelerometer testing:** To test the accelerometer, an LCD was used to display the real time values of the acceleration.

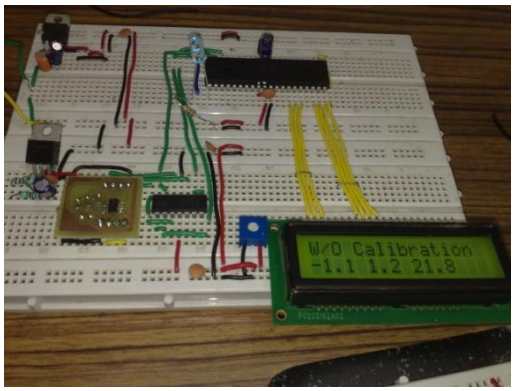


Fig. Testing of accelerometer with the LCD showing the uncalibrated values.

As shown in the figure, we can see the values of the acceleration being given by the accelerometer for the 3 axes without calibration.
To calibrate the accelerometer, thus, the initial readings of the device are taken as the 'zero' values and then the further readings are taken with the shifted origin. That is, every value given by the accelerometer gets subtracted by the average value of the acceleration obtained from the device initially when the device is just switched on.

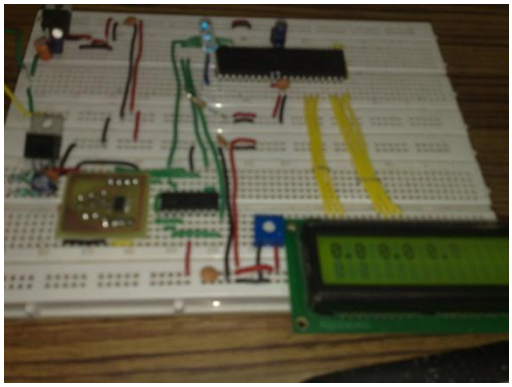


Fig. Zero values obtained for the acceleration in the 3 directions after calibration.

Simultaneously, the position was also calculated by integrating the acceleration values and displayed on the LCD as well (shown in the second line with the first line showing the acceleration values).

Fig. Acceleration and position values shown in the LCD.

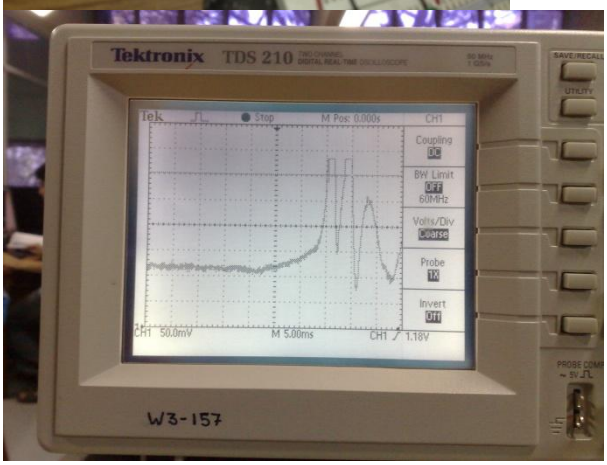


Fig. Impact testing of the accelerometer showing the range of acceleration values. Note that no saturation occurs.

Problems faced

Shorting of the internal transistor

While testing, one of the internal transistors got burnt(effectively got shorted). So, the impedance of one of the axis (z, in our case) was coming out to be lesser than the usual 1 MΩ, to a few hundered ohms. Thus, the output was effectively shorted to the ground with a lot of current flowing and therefore, the voltage was pinned at 0.30 V, thus giving erroneous readings. So, the accelerometer had to be changed and more care had to be exercised while using it.

Noise

The output is quite noisy and thus, a low pass filter is required to get a cleaner signal.

Fig. 10 mV peak to peak noise

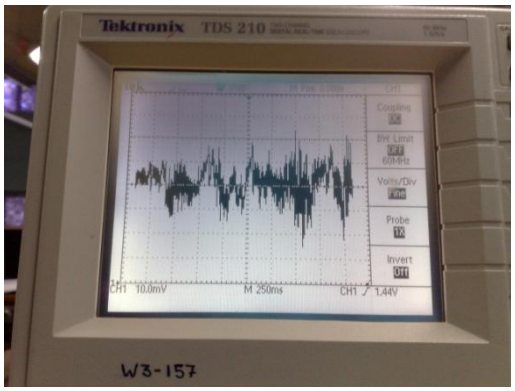


Fig. Extremely noisy signal obtained without a low pass filter

Fig. Much cleaner signal obtained after adding Low Pass Signal

As is clear from the given figures, the addition of a low pass filter helps improve the signal quality to a great extent.

2. **SD Card testing**

Firstly, it was tried to directly interface the SD Card with the microcontroller (ATMega16L) using the SPI mode. The following is the detailed analysis of the efforts put in to get the SD Card working.

Problems faced

But in our case, we weren't able to initialise the SD Card in spite of repeated efforts. One of the problems that we diagnosed was that all the input pins of the SD Card (that is, the pins having the data flow from master to slave) need to be pulled up. This was corrected but still, the initialisation didn't occur.

Uptil now, we had been working with a 3.3 V microcontroller. To address the problem, we even tried working with a 5V microcontroller and applied voltage translators on all the input lines to the card but to no avail.

Solution

Finally, we had to switch over to an SD card module which communicates with the microcontroller through a USART interface and writes data into the card using the SPI mode. It led to favourable results in terms of the initialisation and now, we proceeded on to see if reading/writing of data was occurring.

To check whether the data was actually being written to/read from the SD card, we used a software called **Winhex.** It scans the whole disc and gives the hex values of the data stored at different locations (read sectors). In this way, it can easily be verified if the data that we wanted to write has been written or not.

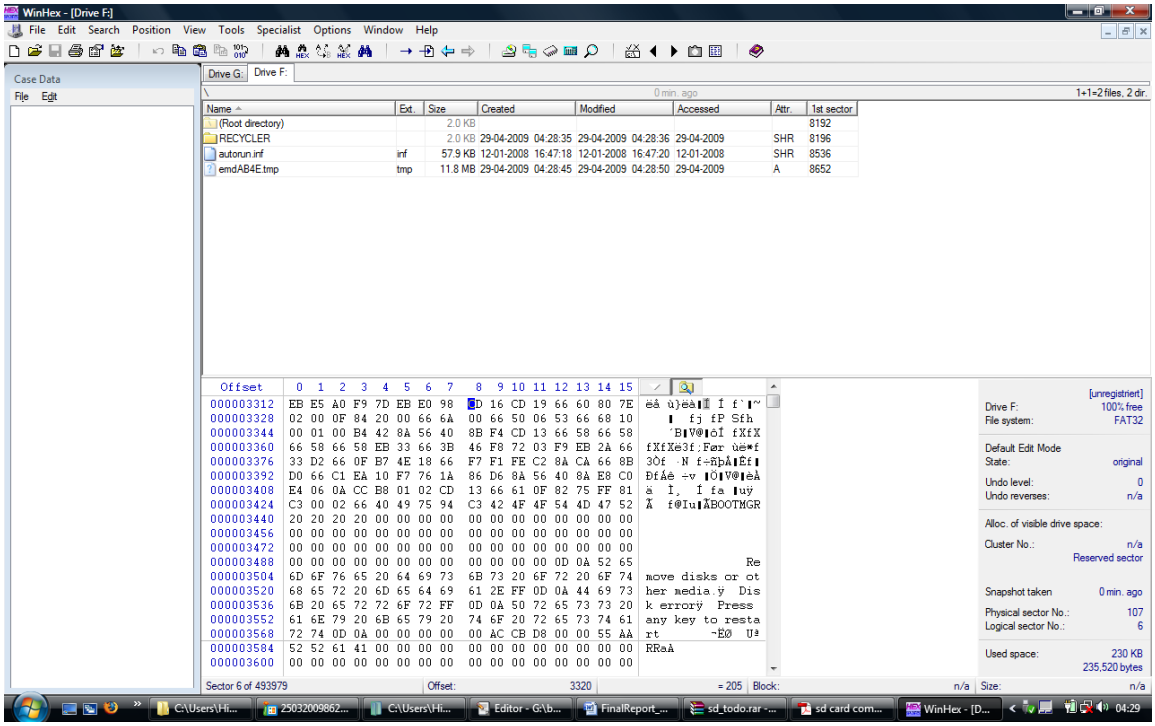


Fig. Screenshot showing SD Card analysis using Winhex. Note the bottom half of the figure where the hex values stored in various locations have been shown.

3. **Plotting the path using Matlab**
   Basic Technique used

The basic principle behind plotting the path using Matlab is that the data is made available through a data file which can be read using the 'fread' command. Now, the data, i.e. the acceleration values, is put into an array that automatically resizes to the number of entries in the file. The entries are in the form of 16 bit 2's complement numbers, so we had to first decipher if the number was positive or negative. Then

we plot the position by using MATLAB's 'plot3' command which takes 3 arrays as input and plots them interpreting corresponding entries in the array as (x,y,z) tuples.

Problems faced

The biggest problem faced was in getting the data file which can be read by Matlab. This was resolved by using two utilities, Winhex and Hexprobe. Winhex was first used to get a hex file of the data and then, Hexprobe was used to convert the hex file into a data file which can be accessed by Matlab.

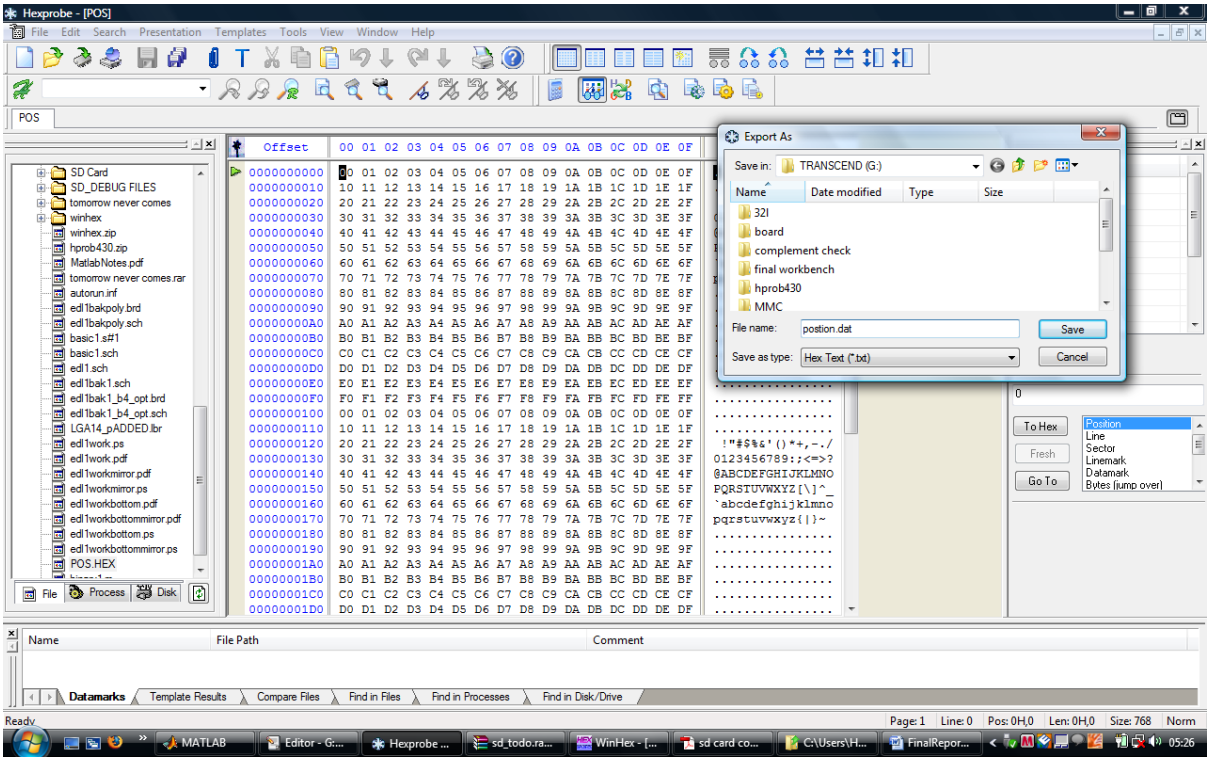


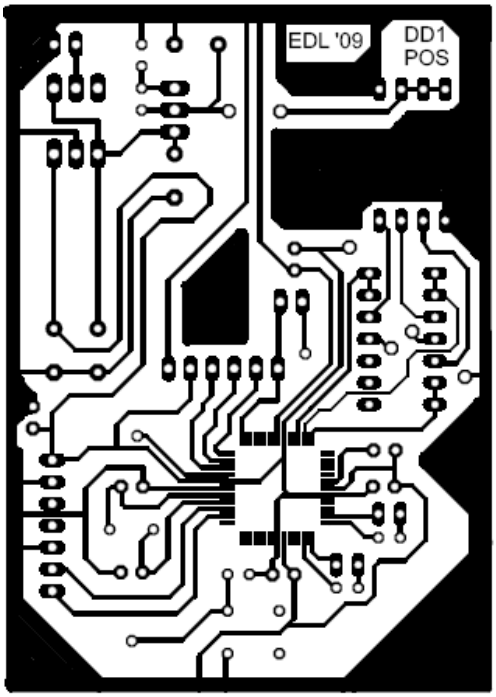
Fig. Screenshot showing the exporting of a hex file in the form of a text file (Matlab readable) using Hexprobe.

Thus, using Hexprobe we can obtain the data in a Matlab-readable format which can then be plotted by writing a simple code.

4. **Final Board**

The final board uses a Surface Mountable ATMega16L.

The layout of the board is as shown:



Problems

After connecting the USBasp In-System Programmer (ISP), we encountered a problem where the device was not being recognised. After checking the connections with the DMM, it was concluded that the circuit was error-free. However, the circuit was loading the USBasp power supply and hence, we decided to power our device using the USB bus and the problem was resolved.

Conclusion

All in all, the device looks promising and with the addition of gyroscopes, the scope of the device can be further enhanced.

Minor changes to improve device working and efficiency were incorporated throughout and they resulted in a better device finally. For example, the case of interrupts(discussed earlier on), where interrupts were implemented after it was found out that the idea of taking the values after fixed intervals of time and computing the position, was found to be highly erroneous.

Steps like upgrading the ADC to the 10-bit mode can improve the efficiency and accuracy of the device even further.
We could also try to have some onboard indication of the path or of displacement alone, but that would be at the cost of a bigger device.

The use of gyroscopes will further enhance the device although the plotting of the path through MATLAB will become a more daunting task. For this, the use of quaternions is suggested.

References

1. Freescale Semiconductor Application Note AN3107,"Measuring Tilt with log-g accelerometers"
2. Freescale Semiconductor Application Note AN3397, "Implementing Position Algorithms Using Accelerometers"
3. Freescale Semiconductor Application Note AN3461, "Tilt sensing using Linear Accelerometers"
4. Hylton B. Menz, Stephen R. Lord, Richard C. Fitzpatrick, "Acceleration Patterns of the head and pelvis when walking on level and irregular surfaces"
5. Experimental Results for Indoor Pedestrian Tracking with the Personal Dead Reckoning (PDR) System, University of Michigan
6. Win AVR-GCC Tutorials
7. Maxim Application Note 4068, "Interfacing SD Cards"
8. Application Note, Secure Digital Card Interface for the MSP430, F. Foust, Micigan State University
9. Application Note, Interfacing SD Cards, Texas Instruments
10. Datasheets of the accelerometer, ATMega16L, SD Card
11. BJ Furman, 2007, "ADC with the Atmega 128"
12. An Introduction to Matlab, Version 2.3, David F. Griffiths, Universit of Dundee

User Manual

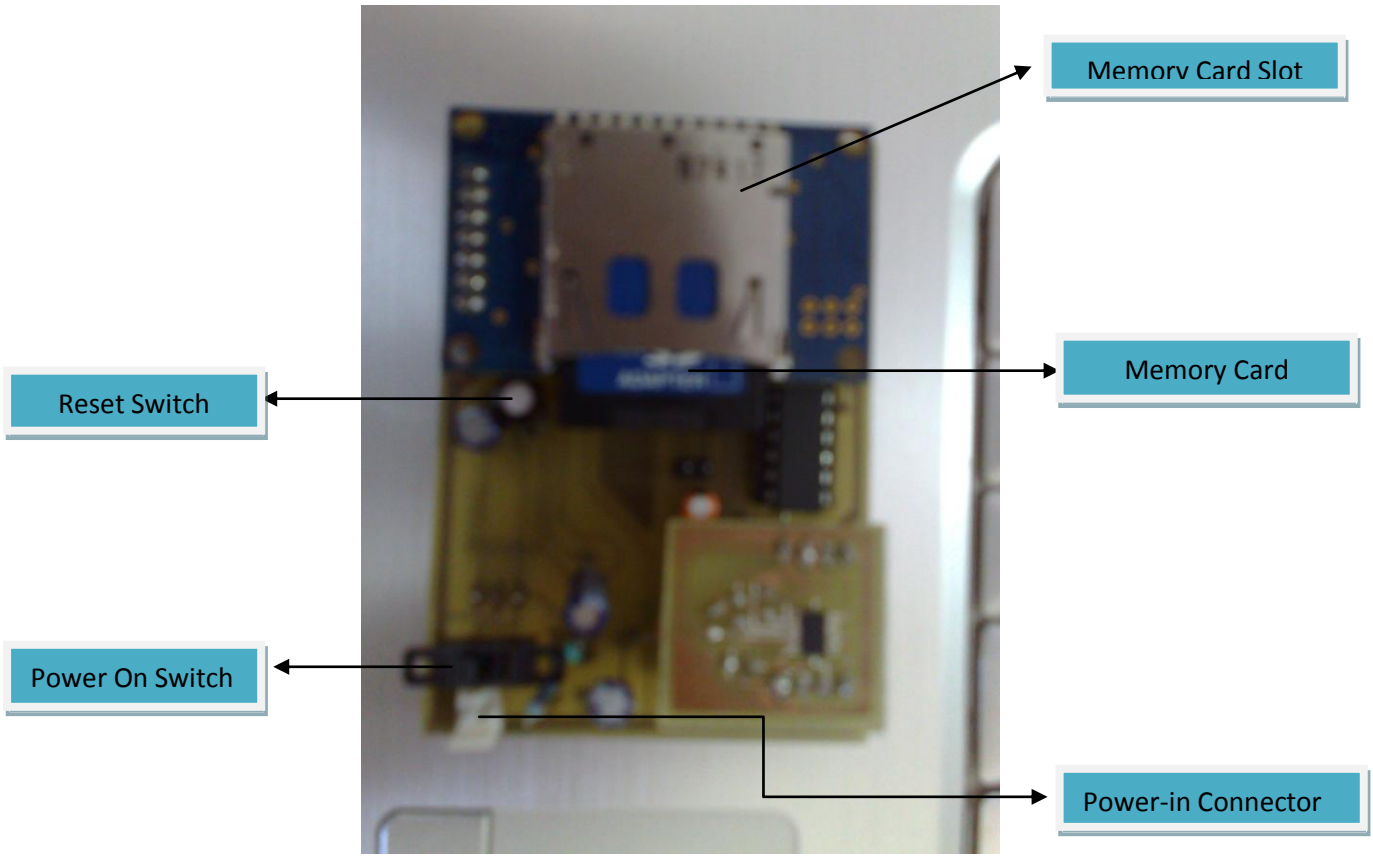


Fig. The device along with the labelled parts

In order to use the device,

A. Power it on using the `Power on' toggle switch
B. Wait for led to turn on
C. Press the `Reset' button
D. At the end of the journey, press the `Reset' button again
E. To plot path, run the file `binary1.m' in MATLAB

Batteries

This device runs on a 9V battery, which can be plugged into the `Battery' socket

Memory

This device supports Multimedia Cards(MMC) and Secure Digital(SD) Cards in MMC mode formatted either in FAT16 or FAT32.

The card can be changed by pulling it out from the `Card' socket and inserting a new one when the device is safely powered off.