CSP-315

Indoor navigation system for visually impaired

Mid-term evaluation report

Team:

- Anant Jain
- Himanshu Gupta
- Manas Paldhe
- Mridu Atray

Introduction

We are working upon the design and implementation for an indoor navigation system for the visually impaired. Our long-term goal is for a portable, self-contained system that will allow visually impaired individuals to travel through familiar and unfamiliar environments without the assistance of guides. The system shall consist of the following functional components:

1. assistance for determining the user's position and orientation in the building,
2. a detailed map of the interior of the building, and
3. the user interface.

By pressing keys on his/her mobile unit, directions concerning position, orientation and navigation can be obtained by the portable system that can prompt them acoustically over a text-to-speech engine. We aim to develop an industry deployable embedded solution to this problem.

Currently, we have divided the project into 4 major parts:

1. Maps
2. Mobile application
3. Algorithms
4. Hardware: user module and buzzer system.

5.

PART I: Creation of a .map file using Google Maps/Earth:

- Google Earth/maps allow the creation of a .kml file through a GUI available on maps.google.com or by downloading Google Earth from earth.google.com
- The **Keyhole Markup Language** (**KML**) is the format for storing maps in the form of Place marks, Lines, Ground Overlays etc. It is an XML-based language schema for expressing geographic annotation and visualization on Internet-based, two-dimensional maps and three-dimensional Earth browsers.
- The kml parser is written in Java with the help of freely available open source library Geokmlib: http://code.google.com/p/gekmllib/
- The .map file generated by the parser has the following format:

  Line 0: Orientation Angle (= Actual North - Map's 'North')

  Line 1: Global Coordinates of Node #1

  Line 2...N: <Listing of nodes>

  Where each line of <Listing of nodes> is as follows:

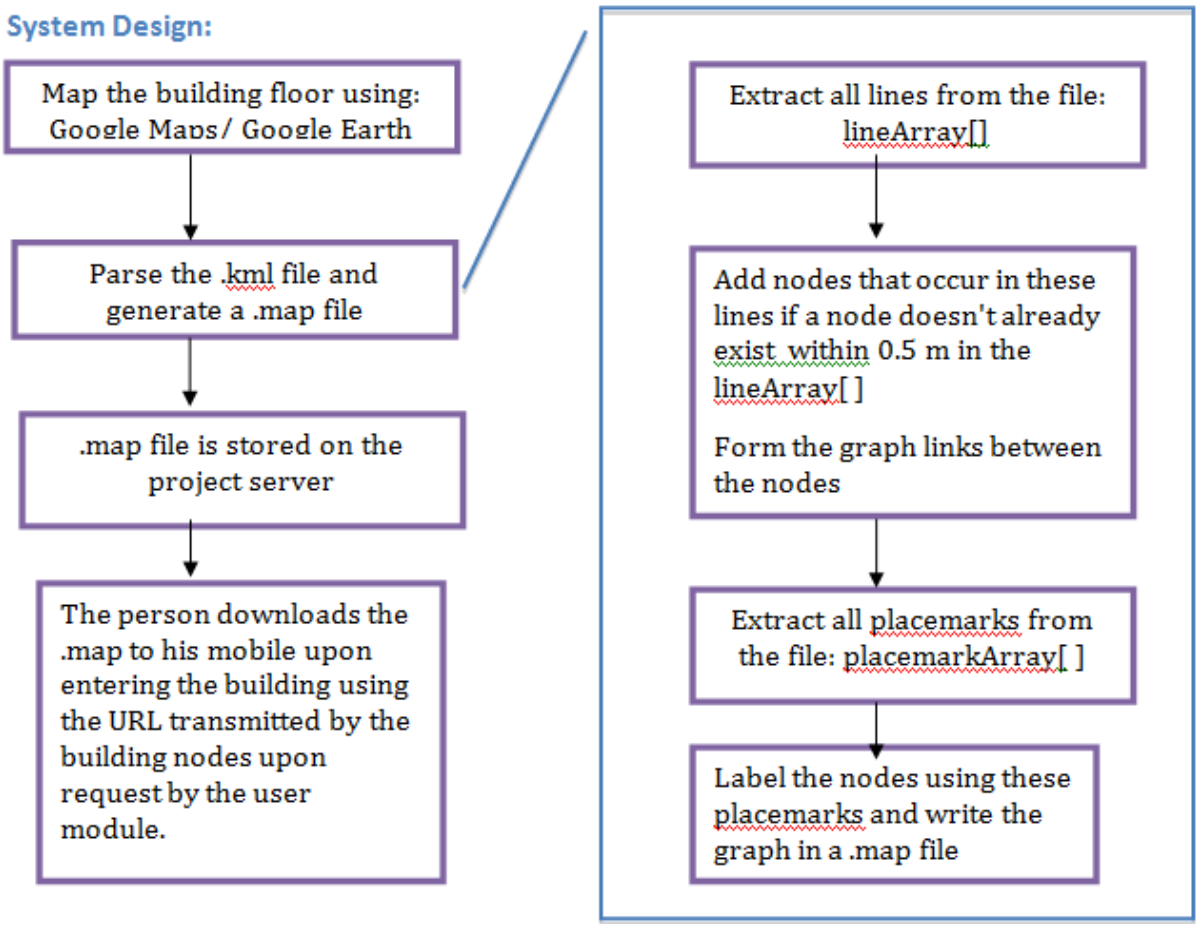
<Node No.>, <Global Coordinates>, <Properties>, <North Node #>, <NW Node #>, <West Node #>,…, <NE Node #>

The <Properties> is a class, with the following members:

- (bool) is PublicUtility
- (String) publicUtilityType
- (String) nodeName
- (int) priorityOrder; // 0 by default if not applicable

The .map file generated by the parser can then be downloaded from a web link provided to the mobile through user module upon entering the building. The nearest building node transmits the web-link over the CC2500 channel upon receiving request from the user module.

System Design:

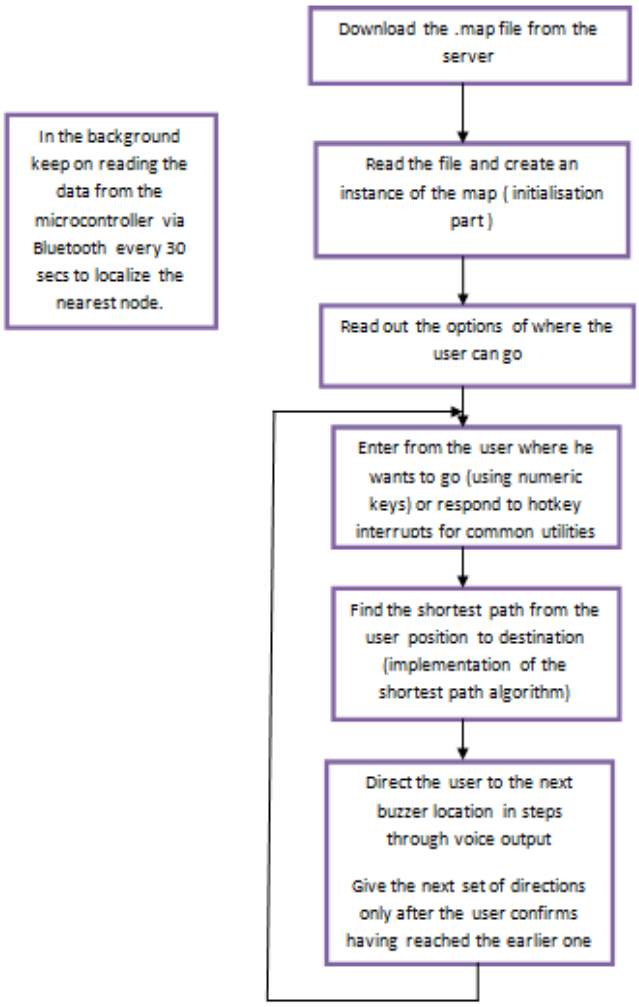


Progress report:

- The parser whose block diagram is given on the right is complete under the assumption that the user makes the map using Google Maps (i.e. it cannot handle floor/altitude information for now)
- The parser is able to parse the .kml file, and generates a .map file which contains a list of nodes; however the creation of graphs between the nodes is yet to be done.

PART II: Mobile Application:

- **Mobile uses the data collected from user sensor module to position the person on the map downloaded from the building. It also interacts with the user through keypad input and voice output. The steps of working are shown alongside in the block diagram.**

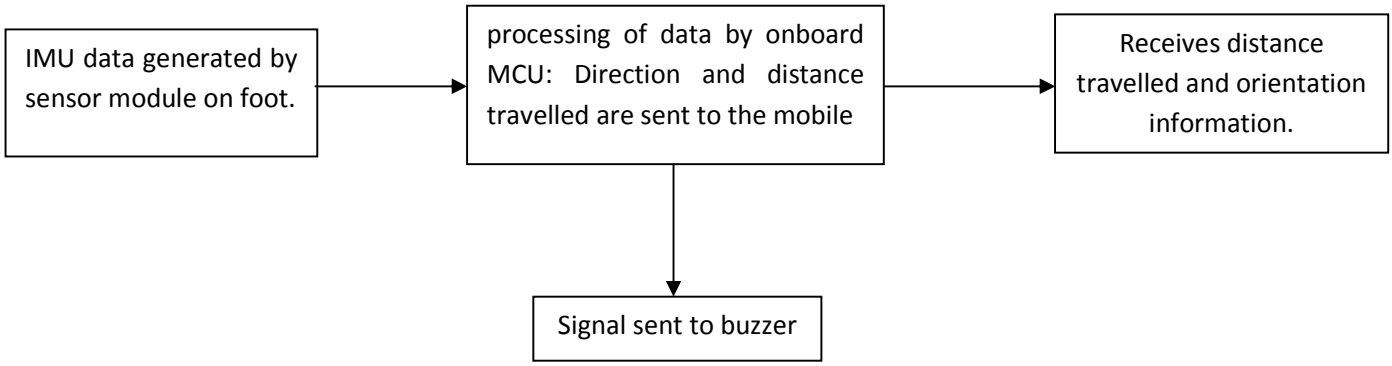


Progress report:

Specific codes for downloading map from a server, communicating with the BT module on the user device are complete. The DS for map is also complete. The algorithm in the block diagram remains to be implemented.

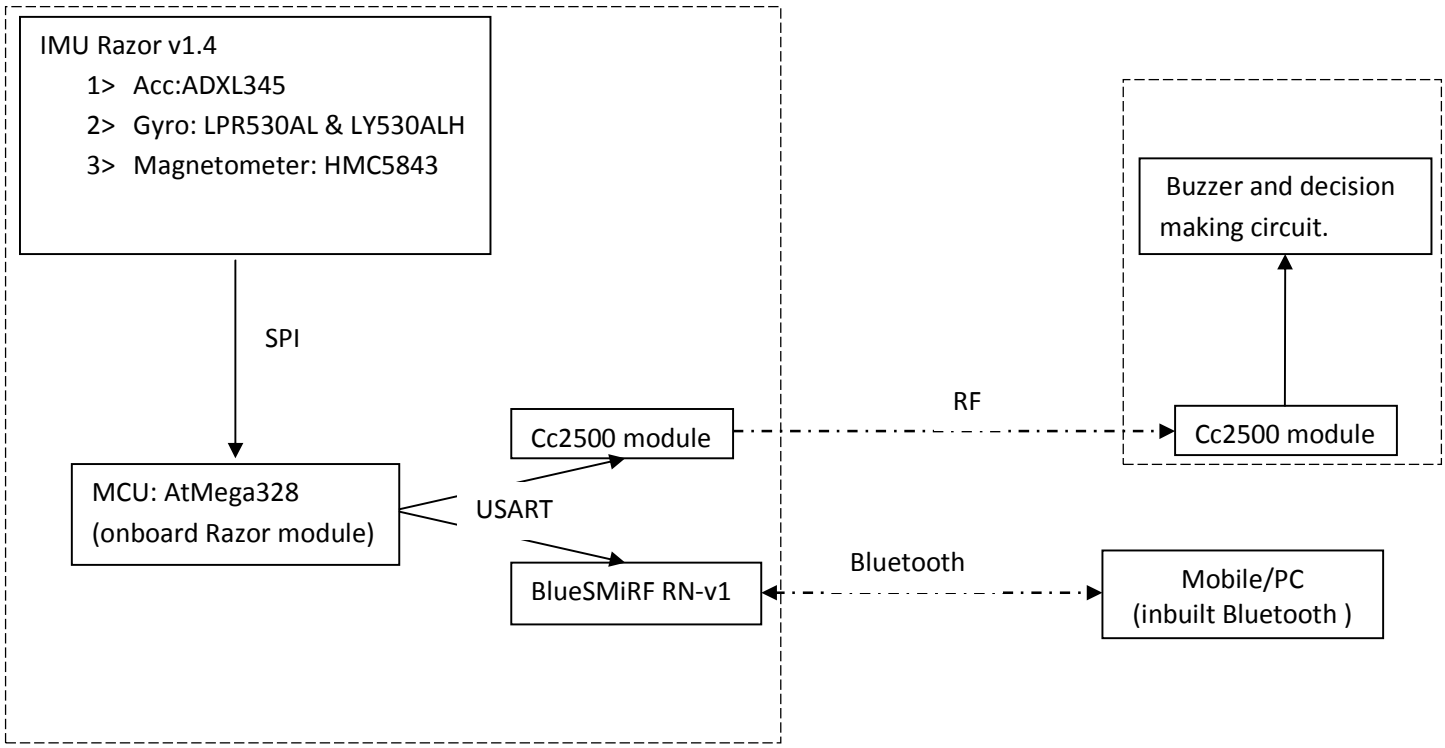
PART III: Hardware-User Module and Buzzer system

a> **Data/signal flow**: IMU sensor module generates acceleration, gyroscopic and magnetometer data. This is sent to the onboard MCU which processes this data to get heading and distance travelled. Heading and distance travelled are regularly sent to the mobile phone over Bluetooth. From this data mobile updates the user's current location. The onboard MCU also sends signal to buzzers in the building over RF. Once the user is in proximity with the destination buzzer, it triggers giving the exact location of the destination.



b> **Hardware Overview**: The hardware consists of off the shelf components.

- Razor-v1.4 is used as the sensor module. It has ATmega328 onboard MCU
- MCU receives and pushes data of sensors into Bluetooth module (blueSMiRF-v1).
- Data can be received by mobile phone (or a PC for testing algorithms)



Softwares/interfacing/testing tools

MATLAB:

- Used to write initial code and test them quickly. It's extremely large library and toolbox set helps in quick code development. Since, MATLAB works with matrices by default; it is very apt for this project where computations involve moving matrices around.
- If MATLAB is installed on a Bluetooth enabled PC, which is very common, then it can be used to receive data from sensors in real time, and the code tested for near real situation. Almost all the laptops today are Bluetooth enabled. The desktop PC can be Bluetooth enabled by using a BT dongle. The above used BT module appears as a virtual serial port.

Hyperterminal:

- Hyperterminal is a very handy tool that can be used to set up instant serial communication with any serial device. It is very helpful in testing and debugging modules. In the present project, it can be used for:
- Testing BT module
- Testing CC2500 module
- Receiving/testing sensor module data via Bluetooth.

Custom Hardware development:

Hardware forms an imp part of the project. User module consisting of IMU sensors, a powerful processor, two wireless communications (BT and RF) is fairly complex and has special requirements. It was decided to develop custom hardware for the project. However, looking at the overall complexity of the project it was suggested, to work with ready-made modules in the beginning. As a result, the hardware development was delayed and kept out of the semester target.

Specifications:

- Triple axis accelerometer: LSM303DLH
- Triple axis magnetometer: LSM303DLH
- Triple axis gyroscope: ITG-3200
- Bluetooth module RN14
- RF module: cc2500
- MCU: ARM7

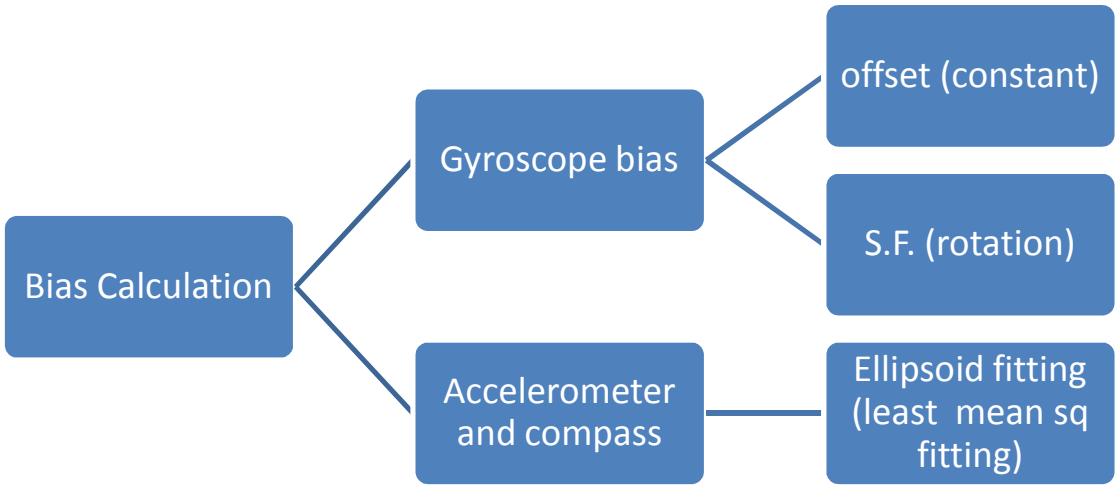
Progress report:

Sensor module, Bluetooth module, CC2500 modules and arduino board have been procured. The sensor board and Bluetooth module have been tested and are working fine. CC2500 modules have also been tested. The communication channels of Bluetooth and RF were successfully setup.

PART IV: Algorithms

1. **Bias Calculation:**

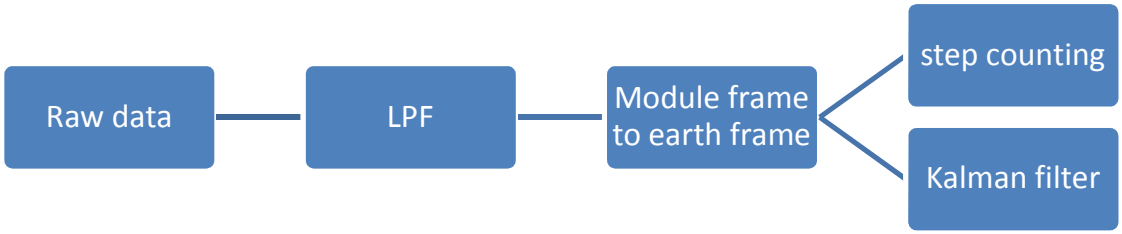
- Gyroscope:
    - Offset: Keep the module stationary for about ten minutes (The longer the better). Then take the average of the readings. The average values are the offset for the corresponding axes.
    - Scaling Factors: The module is rotated (say x rotations) about a particular axis of the gyroscope. The ratio of the "actual number of rotations" to the "number of rotations calculated by integrating the accelerometer readings" is the scaling factor for that particular axis.



- Accelerometer and Compass:

    The offset as well as the scaling factors of the accelerometer and the compass is calculated by using the "ellipsoid fitting method". The module is rotated in such a way to cover the "Euclidian Space" in the best possible way. The ordered readings are then ellipsoid fitted. The centre of the ellipsoid is the offset value and the radii are the corresponding scaling factors.

2. Algorithm and Programming:



- Low Pass Filter:
  For implementing the low pass filter in software the following algorithm was used:

```
x[];  // Raw input
y[];  // Filtered output
dt;   //Time interval
RC;   //Time Constant=1/omega
α=dt/(RC + dt);
y[0]=x[0];
For (i=1; i<n; i++)
{
      y[i]= α*x[i] + (1- α)*y[i-1];
}
```

  "x" is the raw input.
  "y" is the filtered data.

- Initial Orientation:
  - Assumption: The module is initially stationary.
  - We use Euler Angle and the rotation matrix to keep track of the orientation of the module. If the module is stationary then the Euler Angles can be calculates using the following formulae:

```
Pitch =   atan(x_acc/sqrt(y_acc^2+z_acc^2));
Roll  =   atan(y_acc/z_acc);
Yaw   =   atan(y_field/x_field);
```

- Module frame to Earth frame:
  - If the orientation and thus the Rotation Matrix is know, we can transform the vector values from the module frame to the inertial (Earth) frame. The equations used are:
  - For Gyroscope:

    RGyro= [**0 sin(phi)*sec(theta) cos(phi)*sec(theta);**
    0 cos(phi) sin(phi);
    1 sin(phi)*tan(theta) cos(phi)*tan(theta)];

    Rotation_Earth=RGyro*Rotation_module;
  - For accelerometer and compass:

    RAcc = **[cos(theta) sin(phi)*sin(theta) cos(phi)*sin(theta);**
    0 cos(phi) -sin(phi);
    -sin(theta) sin(phi)*cos(theta) cos(phi)*cos(theta)];

    RCmp=RAcc;
    EarthAcceleration=RAcc*ModuleAcceleration;

    The Rotation Matrix for accelerometer and compass are same, but it is different for the gyroscope transformation.

- New orientation:
  Even when not stationary, using the transformation matrix for gyroscope, and the angular velocity values, we can calculate the new rotation matrix, and thus get transform the vectors from module frame to the inertial frame.
  The equation to obtain the new transformation matrix is:
  R_new = dR * R_old;
  The dR matrix is the rotation matrix over a small rotation.

- Integrations:
  Having obtained the acceleration value in the inertial frame, obtaining the velocity and displacement is just integration, or the Reimann Sum.

- Kalman Filtering:
  Kalman filtering is generally used for data fusion.
  We observe that, here after every step (when the module is stationary), we have redundant data. The orientation is obtained using the accelerometer and compass (using the equations mentioned above). It can also be calculated using the last stationary state orientation and the gyroscope readings.
  As the gyroscope bias is not constant, so we can keep updating the gyroscope bias at every step.

3. Step Detection:

- Waist:
  For step detection when mounted on the waist we used the following algorithm:
  We detected the maxima and minima.
  Only the maximas above the threshold and minimas below a threshold are considered.
  Between two maximas (or minimas), there should be a minimum time difference, say slightly less than the walking frequency.
  The number of maximas (or minimas) satisfying the above conditions, is approximately the number of steps.

- Foot:
  We observe that the foot is stationary for about 0.25 seconds. Thus by checking if the readings are within g-delta and g+delta for about that period, we can conclude if the module is stationary or not.

- Positioning using step detection:
  Using step detection also we can determine the position of the user. The direction of motion can be found out using the compass at the beginning and end of every step. Step length is supposed to be known. Thus the displacement over each step is known. Summing the displacements, we can calculate the total displacement.

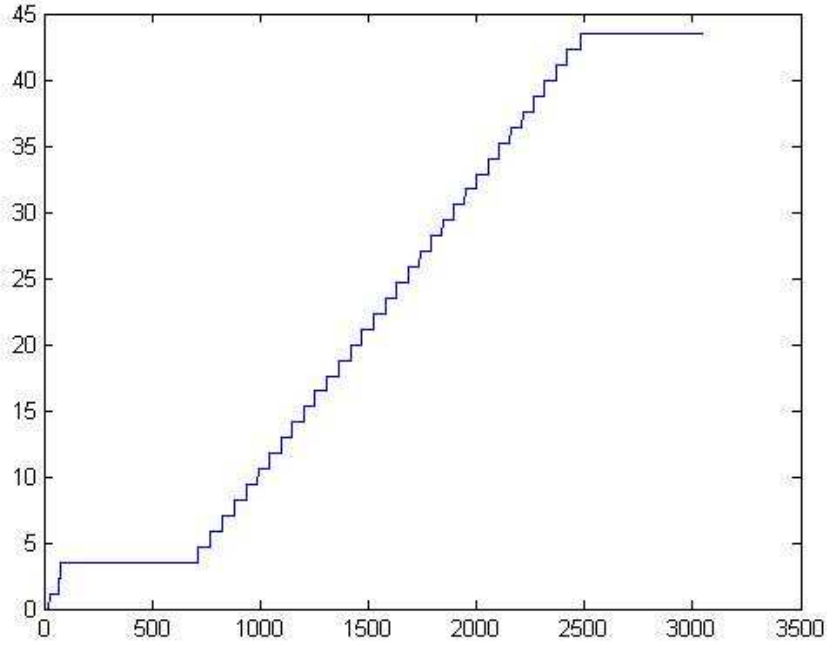
4. Foot mounted v/s Waist:

Foot mounted IMU	Waist mounted IMU
Better Step Detection	
Kalman Filter is not Required	
Computations are reduced significantly	
Can be used effectively for zero velocity updates	
	Is easy to mount.
	Easy for the user.
	No need of wireless communication between the module and the mobile.

5. Results:

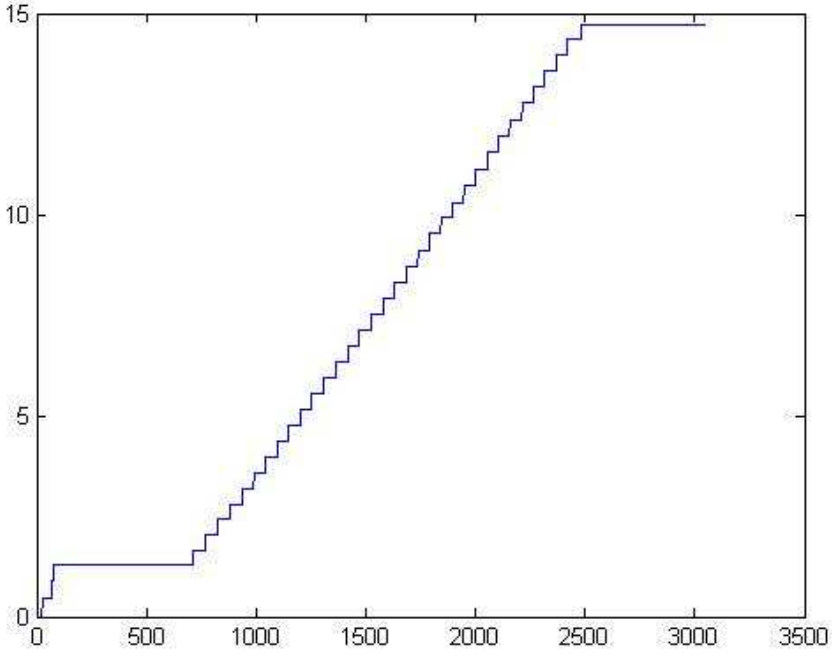
- Stationary module:
  We kept the module stationary, and then used the rotations and double integrations.
  We observed that after about 15 seconds, the displacement values shoot outside limits (0.5m).

- Step detection:
  Two-three samples of data were collected for “straight path” of about 50m. Using step detection, the number of steps and the displacement was calculated correctly (Error of about 1 meter). The result is yet to be tested on various other paths and samples.’
  For a walk at normal speed(Step length = 0.62m) from front of DHD to GCL (about 50m), the obtained results are:



East Displacement

North Displacement



6. Current Work and Progress Report:

The above mentioned functions are working satisfactorily.

The current work is on “Adaptive Step Length” to calculate the step lengths.