Expert System for Robust Navigation

Data fusion of latitude, longitude and yaw angle using GPS, IMU and Laser Speedometer
Course Project for EE5714 Introduction to Intelligent Systems

Professor: Dr. Butz

Author: Zexi Liu

Department of Electrical & Computer Engineer
College of Engineering
Temple University
Date of Issue: 11/27/2007

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

There are two major shortcomings of GPS navigation. First, GPS does not provide orientation (roll, yaw, pitch) information which is important for satellite navigation system. Second, GPS signals does not work in urban canyons, forests, and indoors. This limits the usage of GPS on the ground such as Personal Navigation System, Embedded Vehicle Navigation or Unmanned Vehicle Systems (UVs)

Our objective is augmenting GPS information to produce more robust navigation, position and orientation. For this purpose, other sensors such as inertial measure unit (IMU) should be used to aid GPS navigation signals.

In detail, we must obtain position (longitude, latitude, altitude) and orientation (roll, yaw, pitch) information. In this particular system, GPS gives position and IMU gives orientation (NED frame). If GPS signal is unavailable, IMU measures acceleration, angular rate and Euler angle. The position can be estimated based on these information.

Figure 1 NavBOX

For testing purpose, we designed a rolling cart mounted with the 'NavBOX'. It can be regarded as a sample satellite on the ground. Also, it can be a testing platform for Personal Navigation System and Vehicle Navigation System.

Figure 2 NavBOX with the rolling cart

2. System Configuration and Firmware design

2.1 System Configuration

Figure 1 shows the top level system hardware structure.

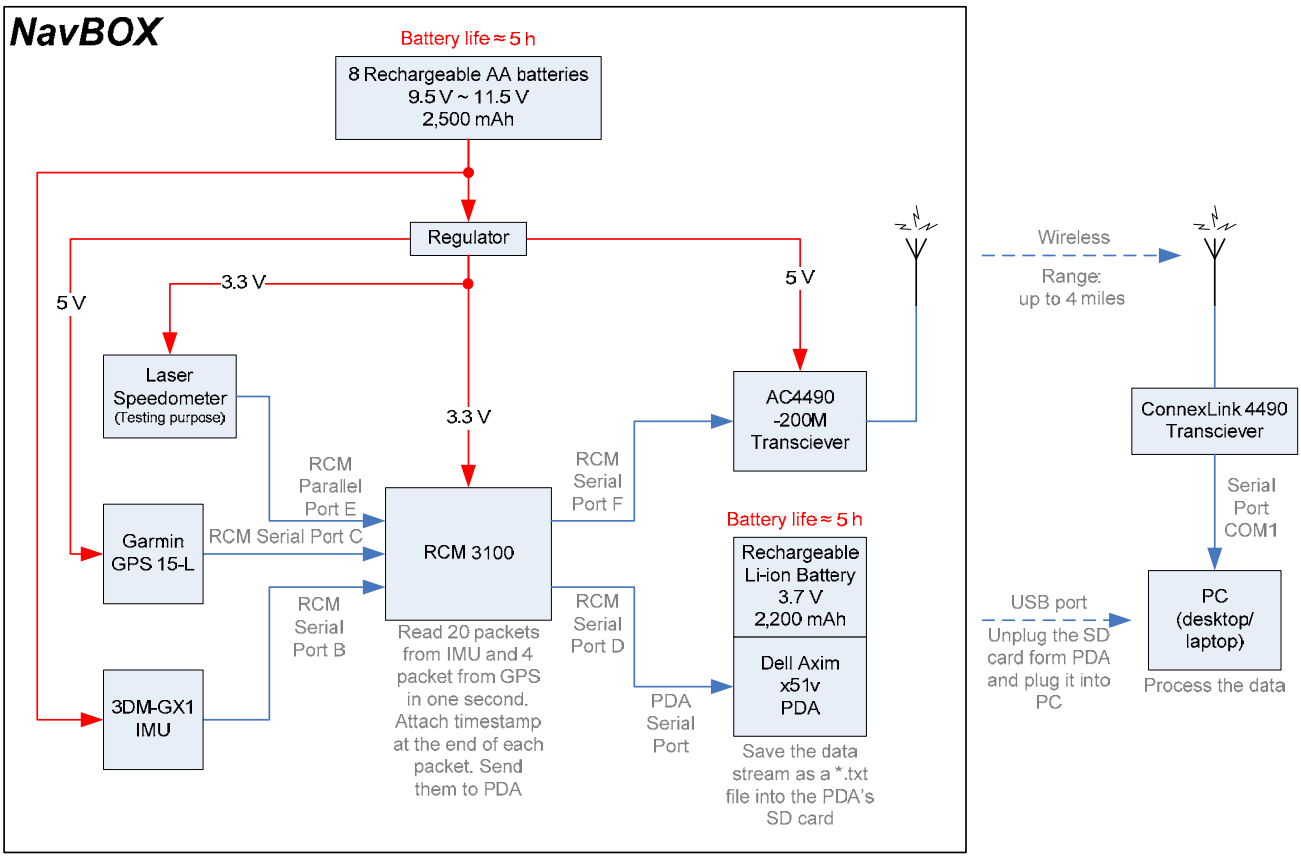

Figure 3 Top level hardware structure of the system

Table 1 Usage of Resources (pins)

System	User
RCM3100	IMU
PB1 - CLKA	PC2 - TXC
PC6 - TXA	PC3 - RXC
PC7 - RXA	GPS
Prototype Board	PC4 - TXB
PA0 - LCD	PC5 - RXB
PA1 - LCD	PDA
PA2 - LCD	PC0 - TXD
PA3 - LCD	PC1 - RXD
PA4 - LCD	RF
PA5 - LCD	PG2 - TXF
PA6 - LCD	PG3 - RXF
PA7 - LCD	PD0 - CTS
PB2 - LCD	Laser Alignment
PB3 - LCD	PB1 - Input
PB4 - LCD	PB3 - Input
PB5 - LCD	Laser Speedometer
PE3 - LCD	PE4 - Input
PE4 - FIR_SEL (IrDA)	PE5 - Input
PE6 - LCD	
PG0 - S3 (Button)	
PG1 - S2 (Button)	
PG2 - TXF (IrDA)	
PG3 - RXF (IrDA)	
PG4 - MD1 (IrDA)	
PG5 - MD0 (IrDA)	
PG6 - LED1	
PG7 - LED2	
PF0 - Motor/Encoder	
PF1 - Motor/Encoder	
PF2 - Motor/Encoder	
PF3 - Motor/Encoder	
PF4 - Motor/Encoder	
PF5 - Motor/Encoder	
PF6 - Motor/Encoder	
PF7 - Motor/Encoder	

2.2 Firmware Design

Figure 2 shows the top level flowchart of the firmware. (This program will be downloaded into RCM 3100.) The program is written in Dynamic C Development Environment version 9.21. Generally speaking, RCM 3100 will read the following data:

IMU: Roll, Pitch, Yaw, 3-axis Acceleration, 3-axis Angular Rate and Speed

GPS: Position, Heading, Speed, Error Estimation etc.

(Please see Appendix A Data Communication Protocol for more details)

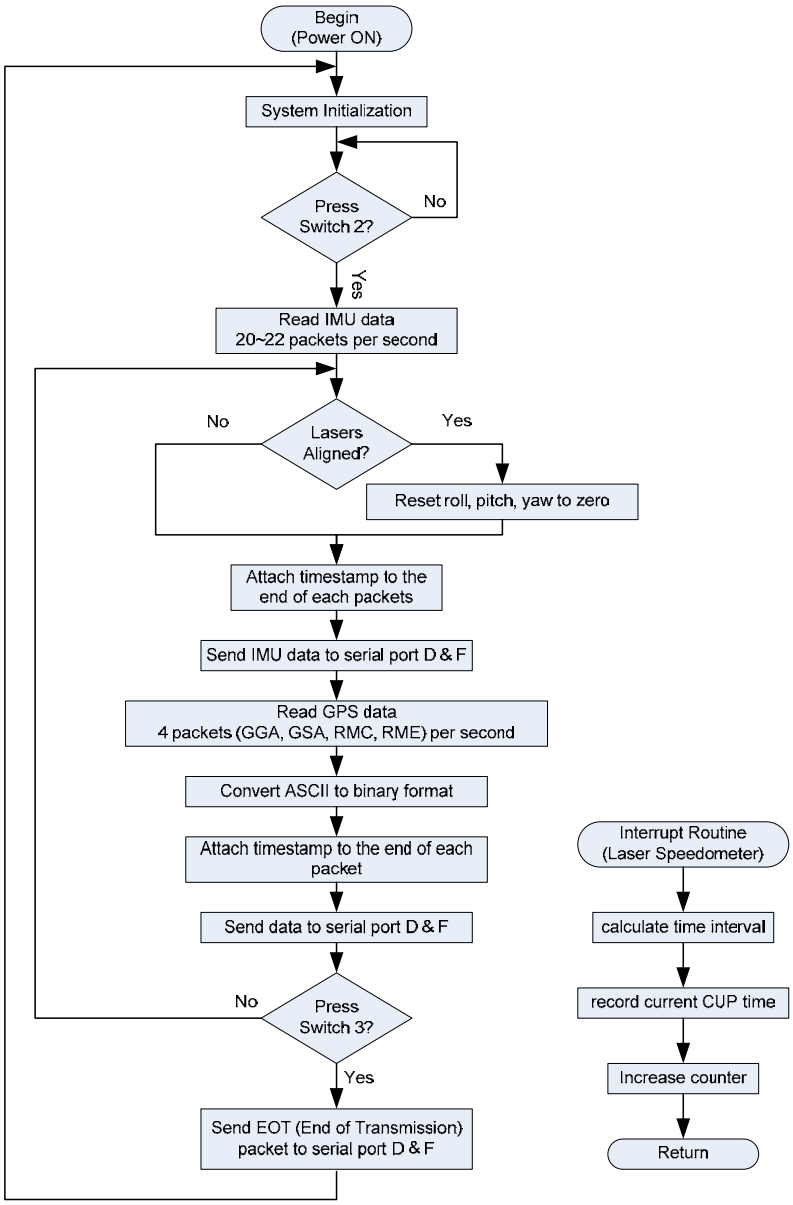

Figure 4 Firmware Flowchart

3. Data Fusion Software Design in Matlab

3.1 Top level flowchart

This Matlab routine is a post processing program with potential real-time process ability. It's because the program does not process all the data as a whole. In a 'standard' post processing Matlab program, all the data will be read into the workspace at the beginning. And during execution, the program could access all the data at any time.

Instead, this program is divided into three parts, the initialization stage, the process loop (program core) and the finial stage. Although all the data is loaded into the workspace during initialization stage, the access of the data is limited and

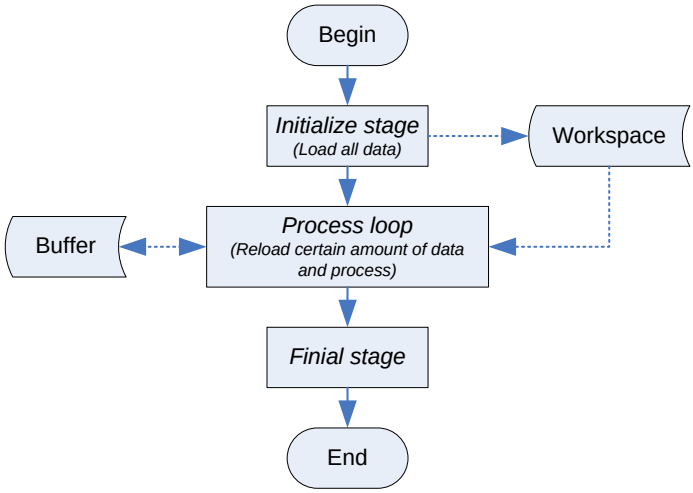

Figure 5 Program structure

controlled by the process loop. In each cycle the program could only reload one GPS packet and 20~22 IMU packets. (This is because in one second, the 'NavBOX' gives us one GPS packet and 20~22 IMU packets.) There is also a First In First Out (FIFO) buffer to store the current data and the results. The depth of the buffer is 10 seconds. This means, at any moments, the program can only access the current reloaded data and past 10 seconds data and results. Simply speaking, this program is trying to simulate a real-time environment (See Figure 5).

Figure 6 shows the details. Figure 7 is the top level flowchart of this program.

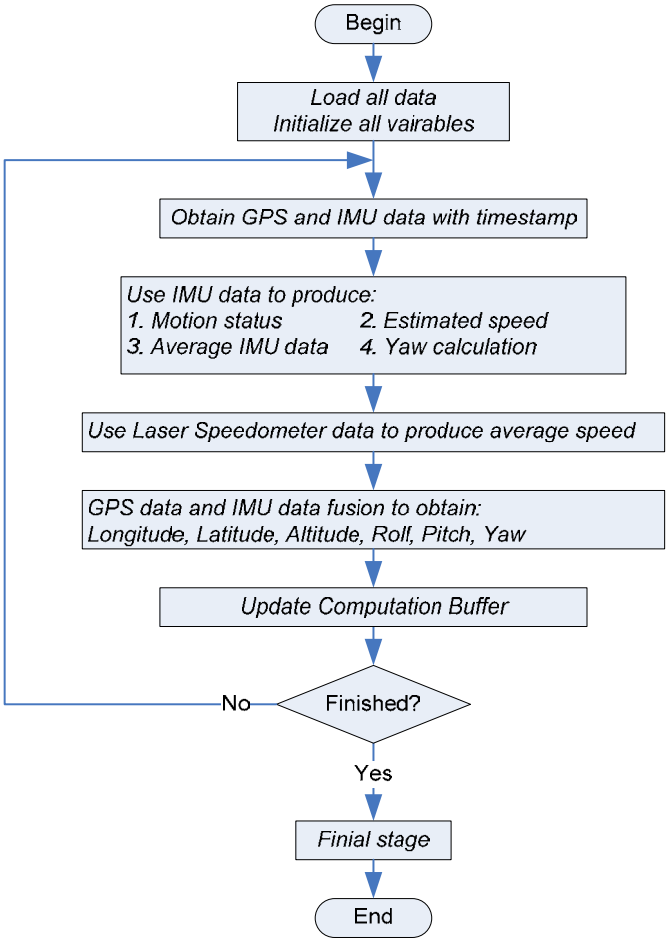

Figure 6 Program structure (details)

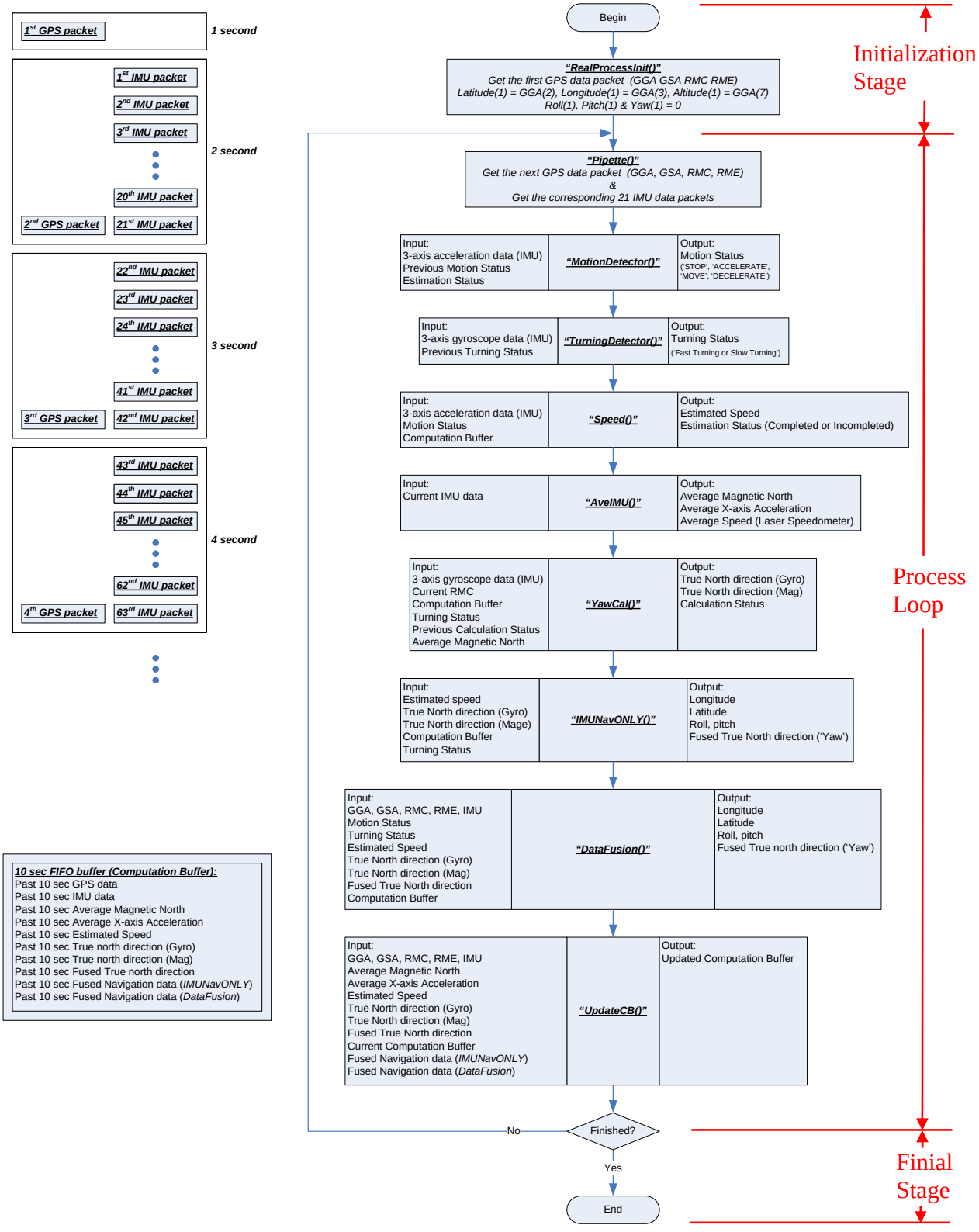

Figure 7 Top level flowchart

Figure 8 shows dependency report (tree view) of the all the functions/subfunctions.

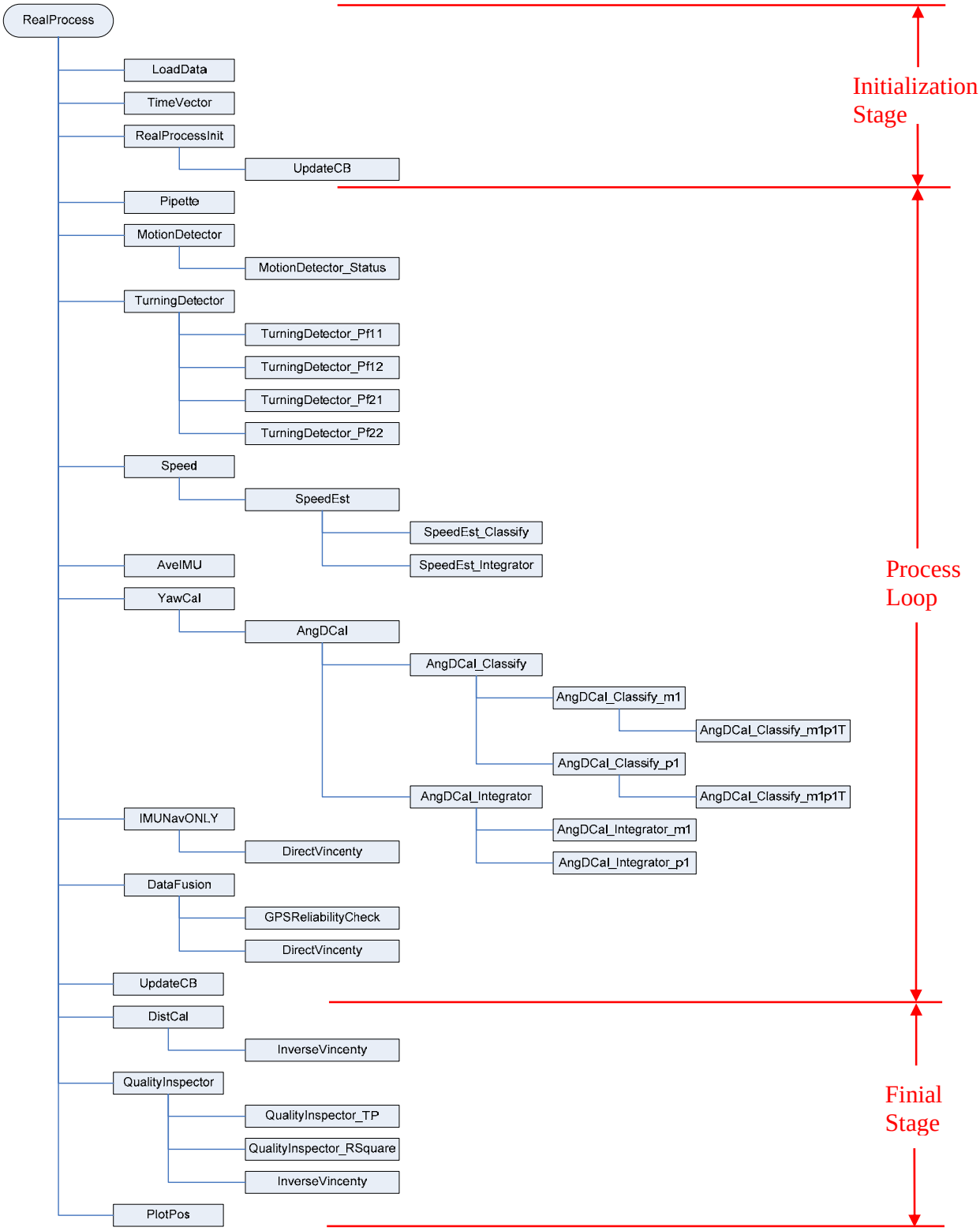

Figure 8 Functions Dependency Report (Tree View)

3.2 Sub function flowchart

3.2.1 Initialization Stage

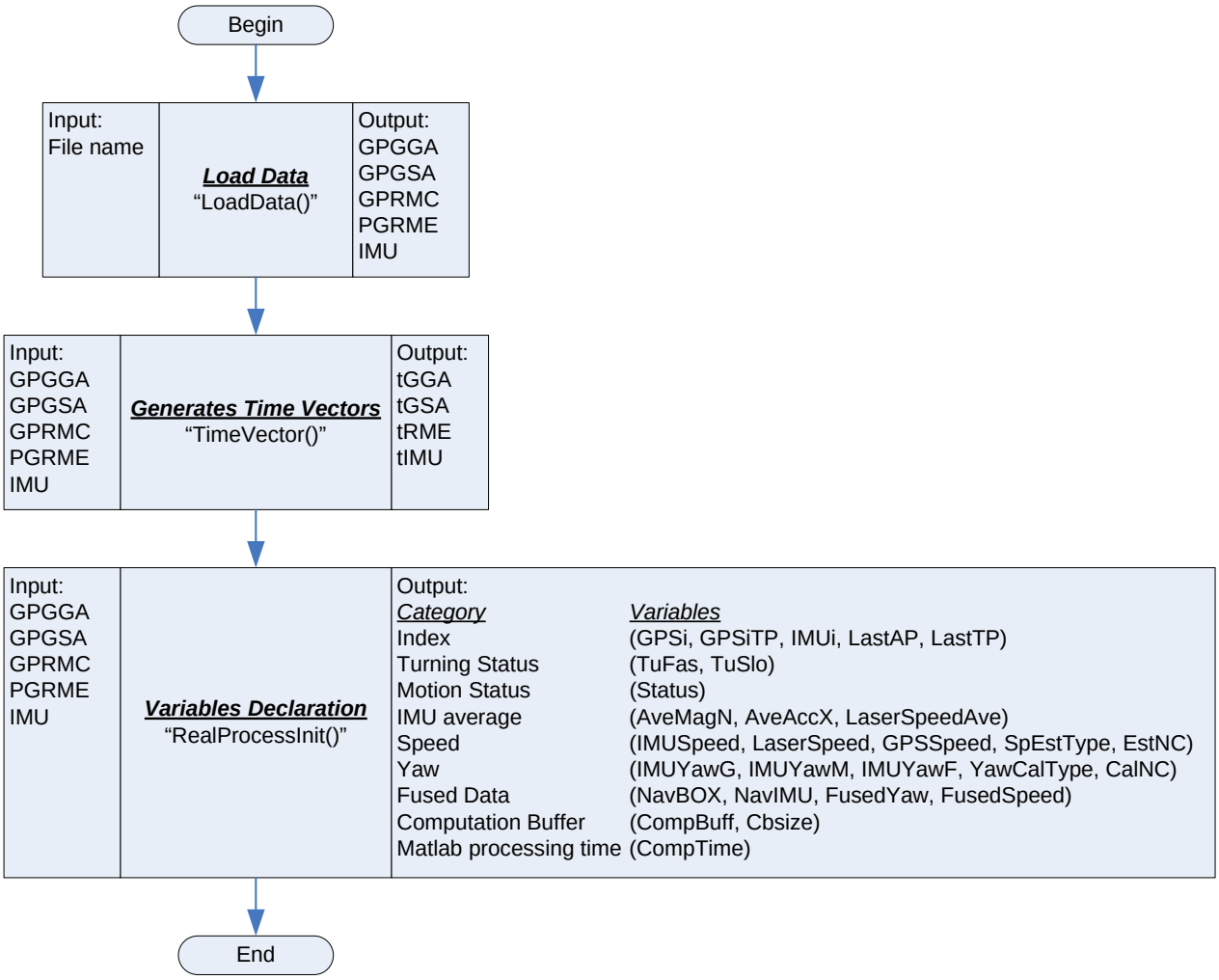

Figure 9 Flowchart of Initialization stage

At the end of each experiment, the PDA will save the data as a *.txt file into its SD card. After copying this *.txt file to our PC, the initialization stage can be executed. In this stage, all the data will be loaded in to the "workspace" of Matlab, waiting for further process. Of course, all the variables used in the following program will be declared here.

3.2.2 Process Loop

In figure 3, we could see that there are nine functions in the process loop. They are:

Pipette()

MotionDetector()

TurningDetector()

Speed()

AveIMU()

YawCal()

IMUNavONLY()

DataFusion()

UpdateCB()

Function Pipette() 'suction' certain amount of data (1 GPGGA, 1 GPGSA, 1 GPRMC, 1 PGRME, 21 IMUs) form Workspace every time when it is called. All the other eight functions are based on these data.

Function MotionDetector() gives out the Motion Status ('STOP', 'ACCELERATE', 'MOVE' or 'DECELERATE').

Function TurningDetector() gives out the Turing Status (fast turning or slow turning).

Function Speed() estimates the speed based on acceleration data.

Function AveIMU() calculates the average value of the certain IMU data.

Function YawCal() calculates the True North Direction based on magnetometers and gyroscopes.

Function IMUNavONLY() calculates the current position and orientation using IMU data only.

Function DataFusion() is designed to fuse the IMU and GPS data together to get the optimal position and orientation estimation.

Function UpdateCB() updates the Computation Buffer every time it is called. This is a First In First Out ('FIFO') buffer.

Since function Pipette(), AveIMU() and UpdateCB() are very easy to understand. We'll not go through these functions in detail.

Figure 6 ~ 11 shows the flowchart of function MotionDetector(), TurningDetector(), Speed(), YawCal(), IMUNavONLY(), DataFusion().

3.2.2.1 Motion detector

Figure 10 and figure 11 show the Z-axis acceleration data. Obviously, in figure 10, section 1, section 3 and section 5 are the part when the device is stopped. The key of the motion detector is to distinguish between section 1, 3, 5 and section 2, 4.

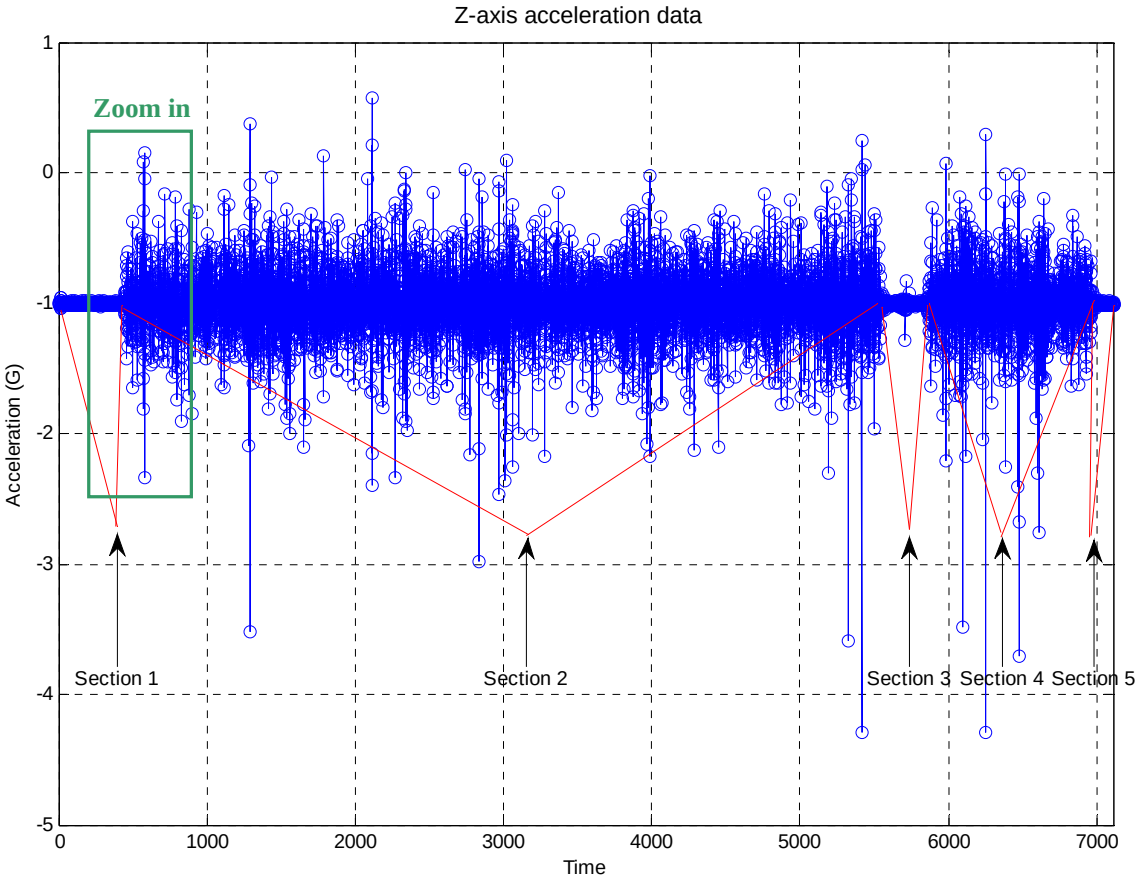

Figure 10 Z-axis acceleration data

In order to explain our algorithm, please look at figure 11 (it's a part of figure 10 in detail). The evident difference between 'motion' section and 'stop' section is that in section 1 almost all the samples are smaller than a certain value, in this case, 0.05G. Oppositely, in section 2, a large amount of samples are larger than 0.05G. Since in each cycle the program could only reload 20 ~ 22 samples, first we count the number of samples that are smaller than 0.05G, and then divided by the total number of samples to calculate the percentage. If 85% of the samples are smaller than 0.05G, then we are in section 1 or 3. Otherwise it's not.

Further more, if there are only two status 'MOVE' and 'STOP', we could not estimate the speed in the following program. In order to calculate the speed, we must have another two status 'ACCELERATE' and 'DECELERATE'. This is done by considering the previous status. The keys are:

- There is always an ‘ACCELERATE’ status between ‘STOP’ and ‘MOVE’.
- There is always a ‘DECELERATE’ status between ‘MOVE’ and ‘STOP’.
- The ‘ACCELERATE’ status is always followed by ‘MOVE’.
- The ‘DECELERATE’ status is always followed by ‘STOP’.

Figure 12 shows the detailed flowchart of the function MotionDetector().

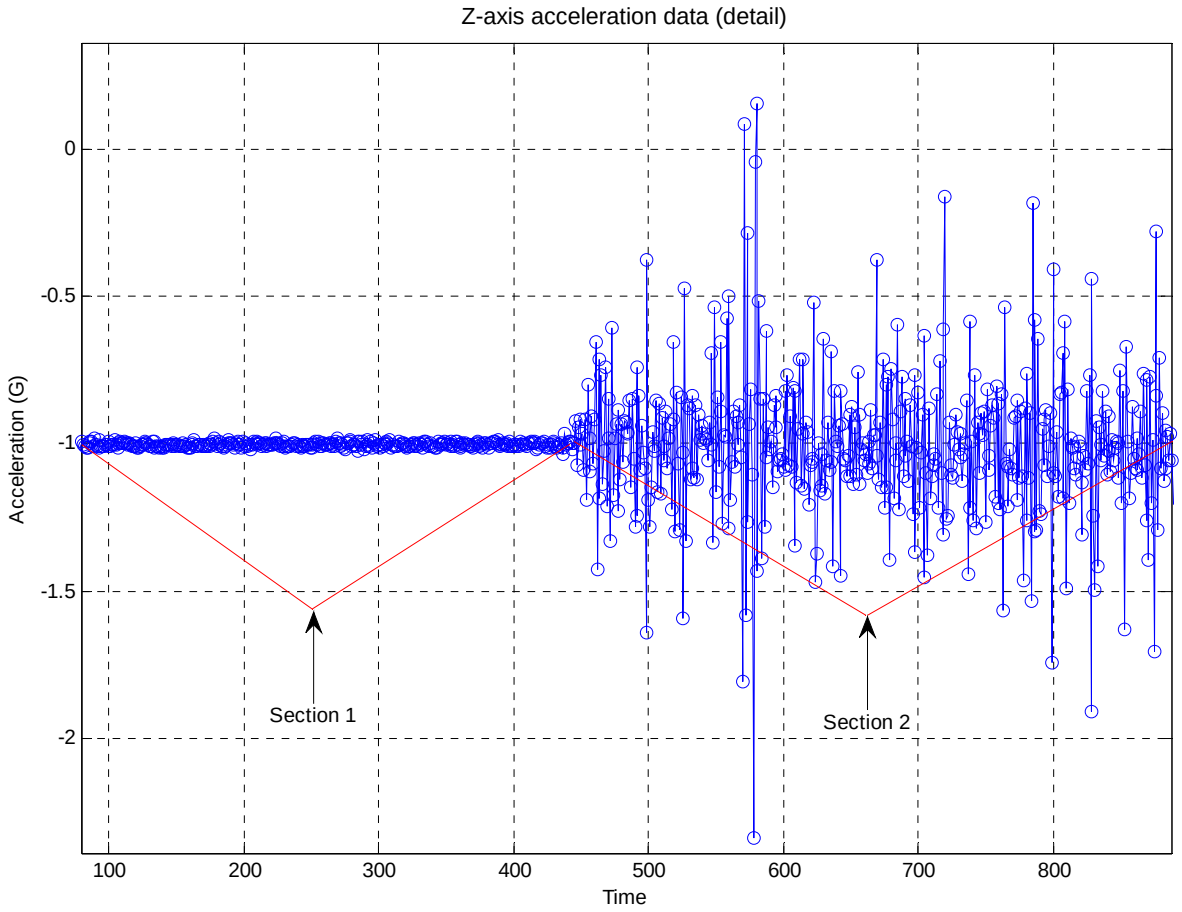

Figure 11 Z-axis acceleration data (Zoom in)

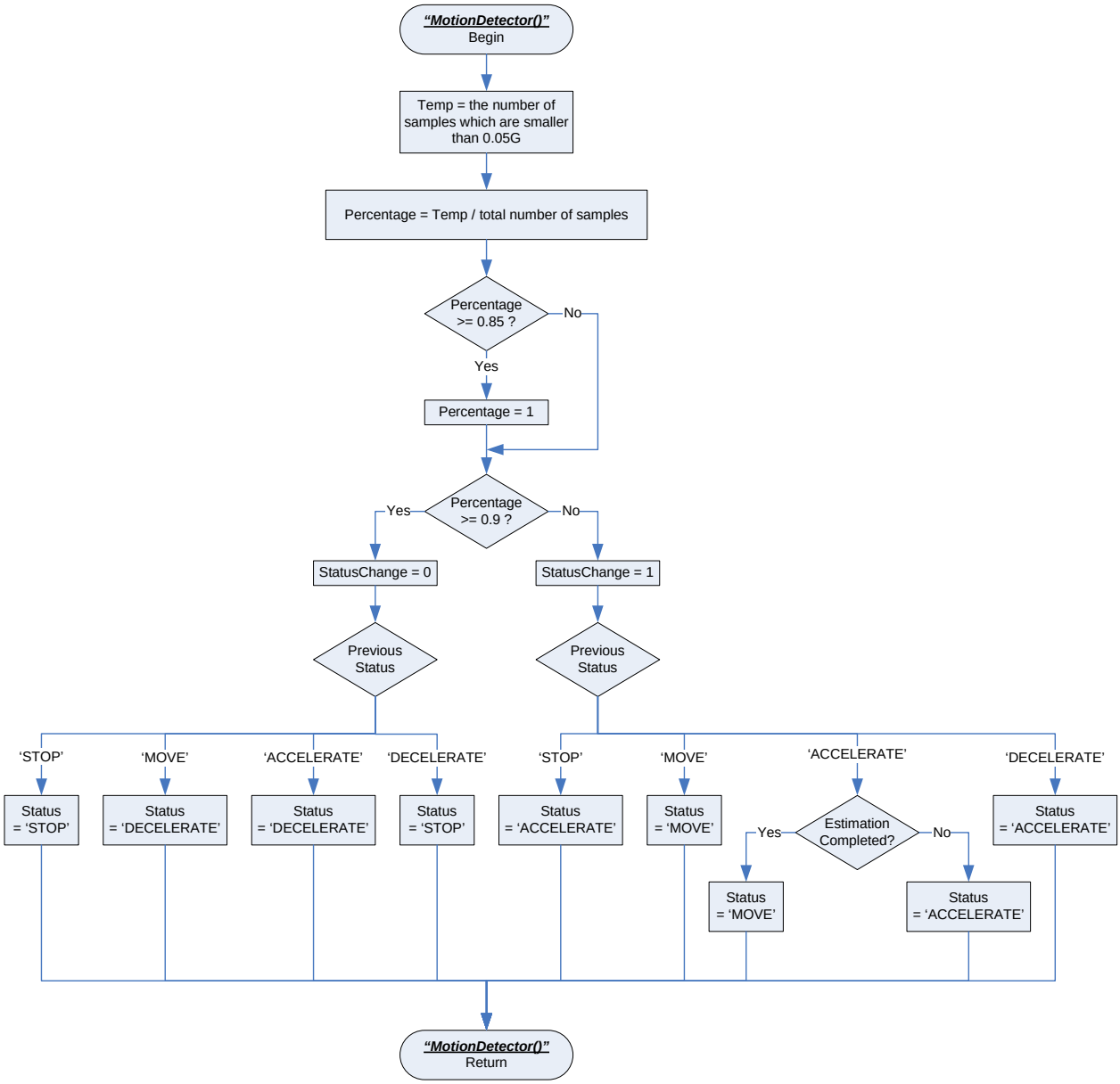

Figure 12 Flowchart of function MotionDetector()

3.2.2.2 Turning detector

Figure 13, figure 14 and figure 15 show the Z-axis angular rate data. Obviously, there are five peaks among these data. But it's not easy as it looks like. The turning detector is designed to distinguish 'a fast turning' and 'a slow turning'. Further more, it can filter the noise peaks.

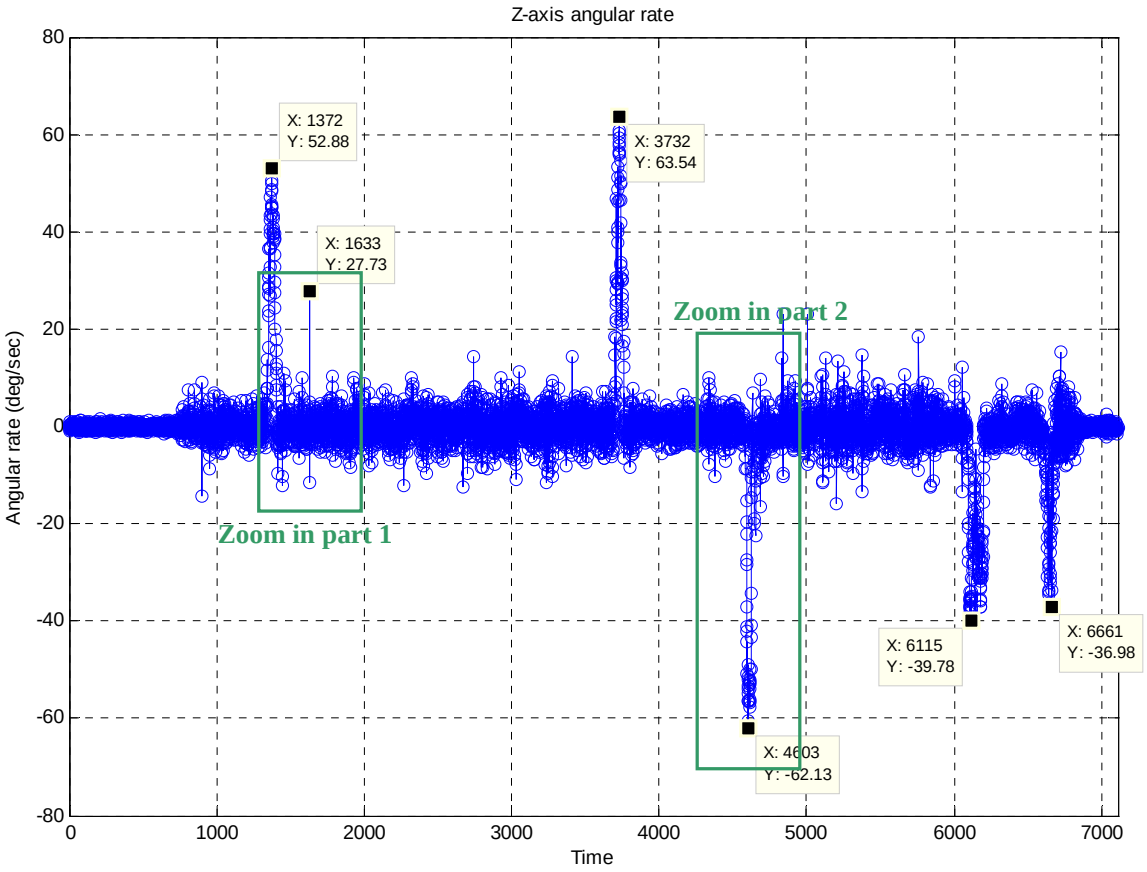

Figure 13 Z-axis gyroscope data

In our algorithm, we use 20 degree/second as the threshold. If there are more than 2 samples are larger than this threshold among 20~22 samples, it is a fast turning. If there is only one samples is larger than this threshold, the program will check the previous or the next sample. If one of them is larger than 10 degree/second, then it's a slow turning. If both of them are smaller than 10 degree/second, then it is just a noise sample. (see figure 14 and figure 15)

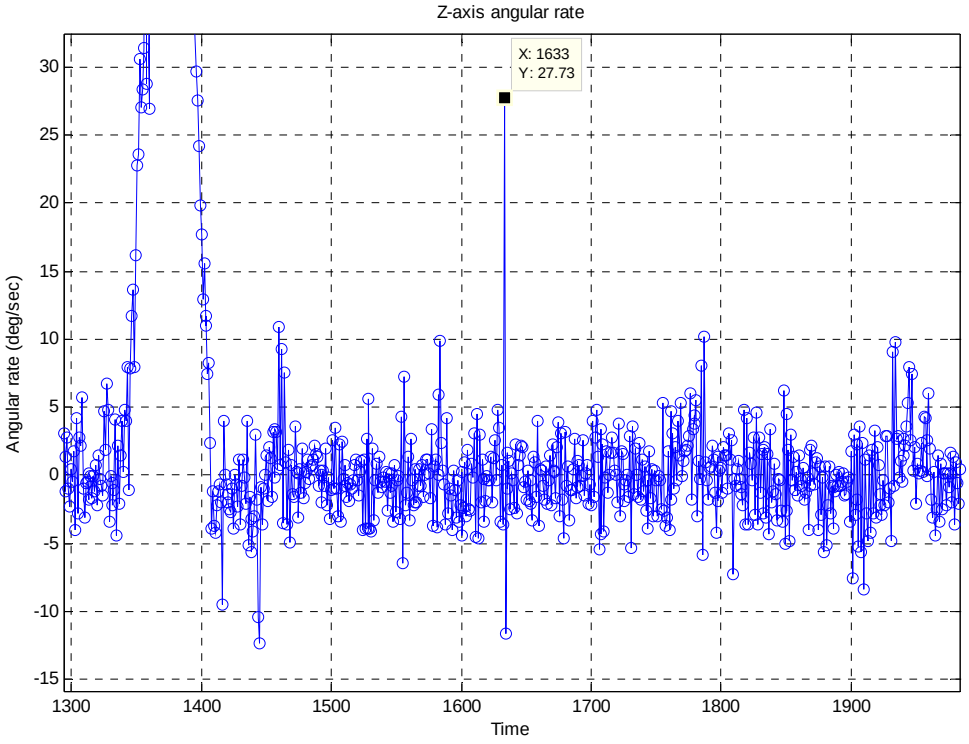

Figure 14 Z-axis gyroscope data (Zoom in part 1)

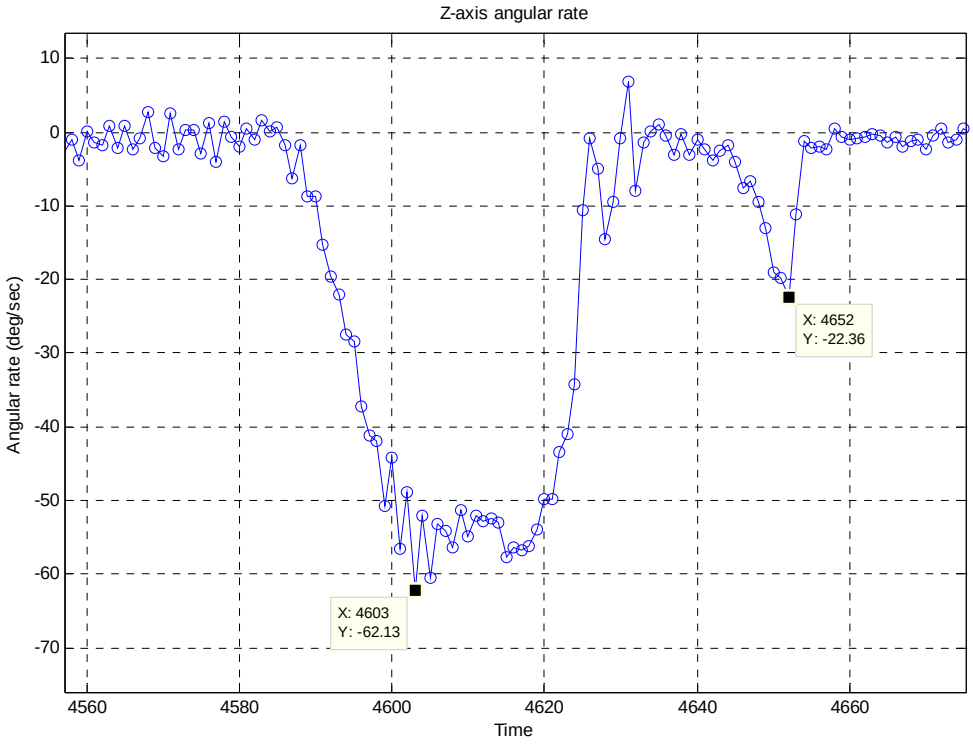

Figure 15 Z-axis gyroscope data (Zoom in part 2)

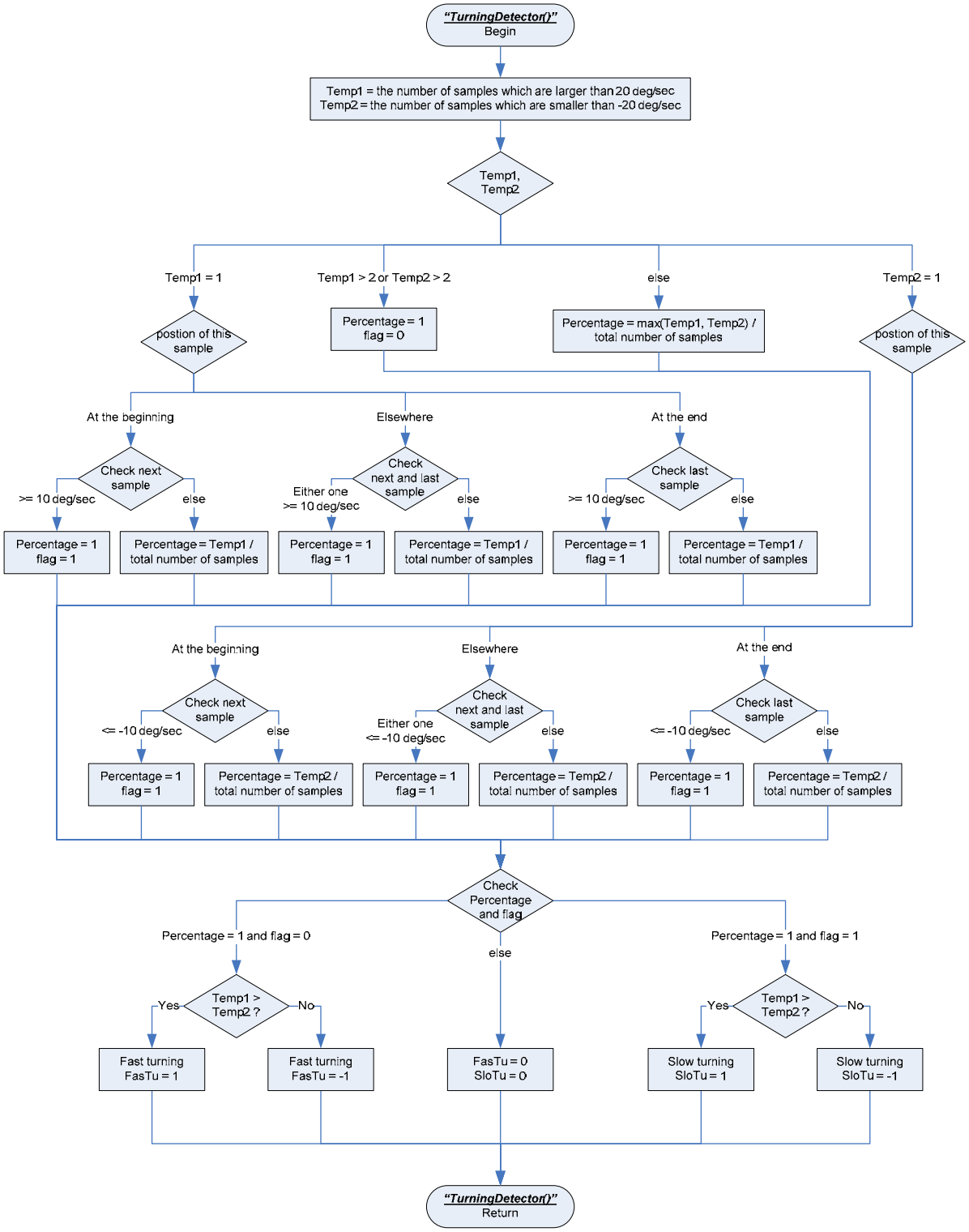

Figure 16 Flowchart of Function TurningDetector()

3.2.2.3 Speed Estimation

Figure 17, figure 18 and figure 19 show the X-axis acceleration rate data. Conventionally, speed can be obtained by calculating the integral of acceleration. However, this could introduce a large amount of cumulative error which will finally overcome the true value.

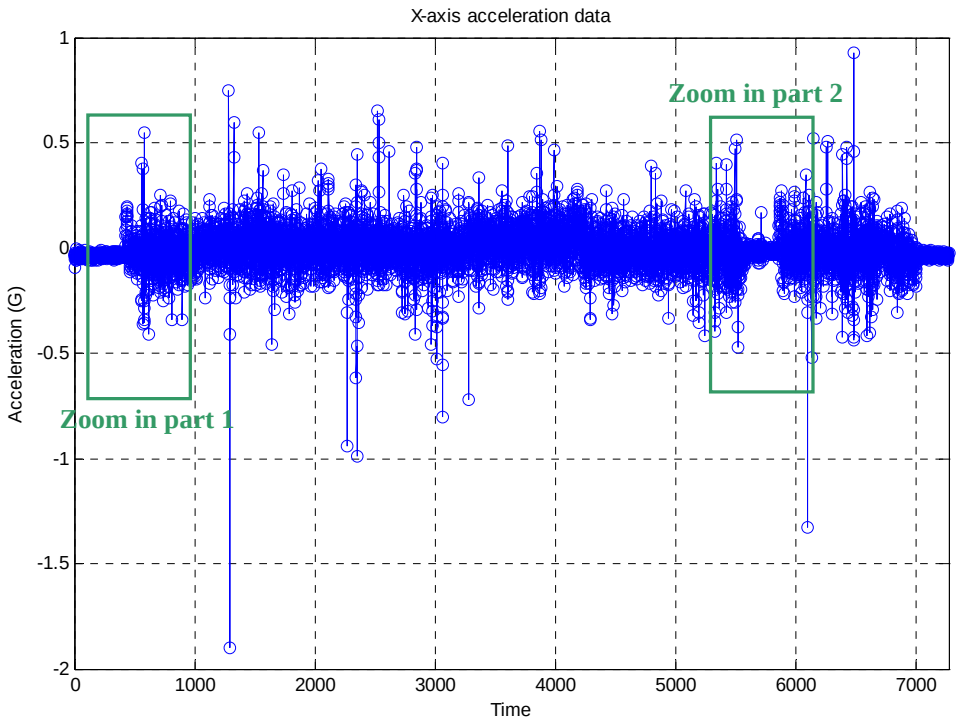

Figure 17 X-axis acceleration data

Our algorithm is to calculate the integral of acceleration ONLY in ‘ACCELERATE’ status. As a result, by discarding other useless data the bias could be minimized. (See figure 18 and 19)

Figure 20 shows the detailed flowchart of function Speed().

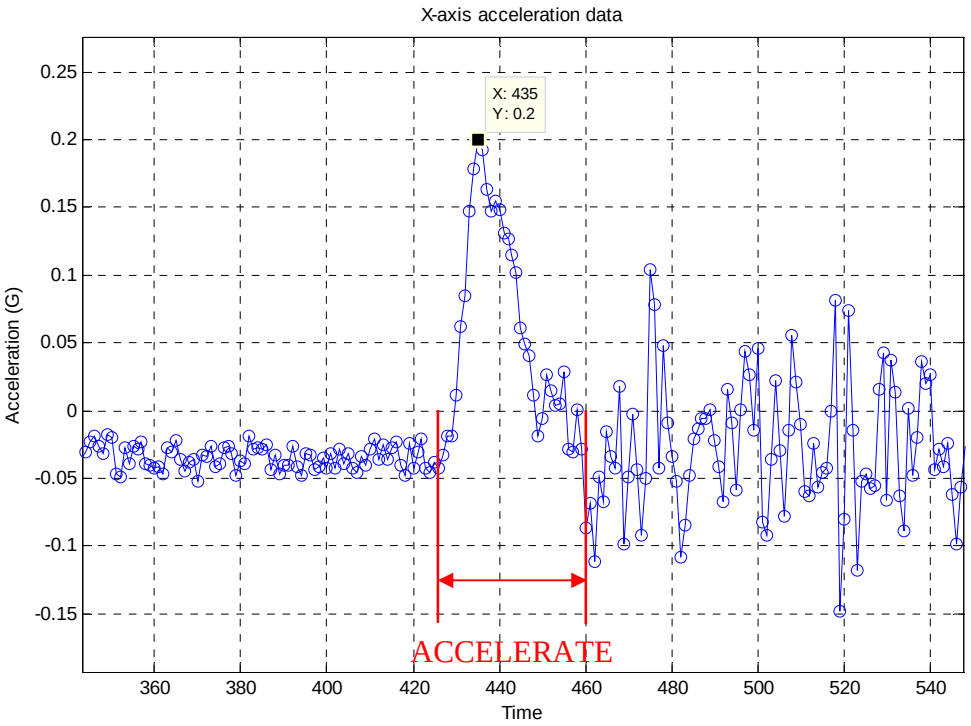

Figure 18 X-axis acceleration data (Zoom in part 1)

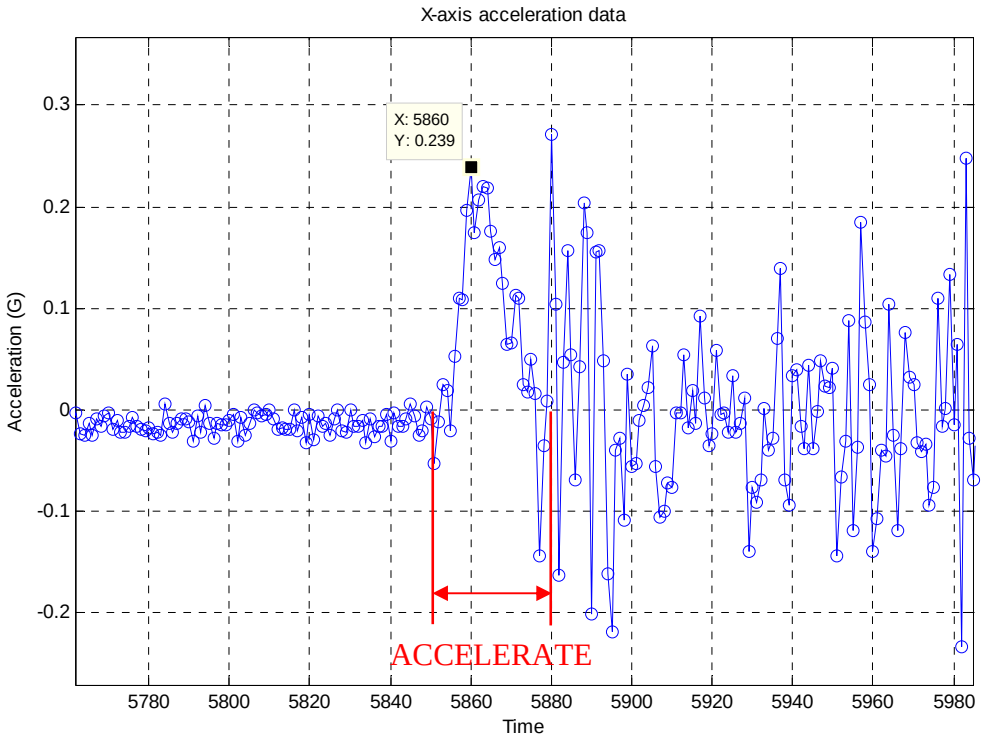

Figure 19 X-axis acceleration data (Zoom in part 2)

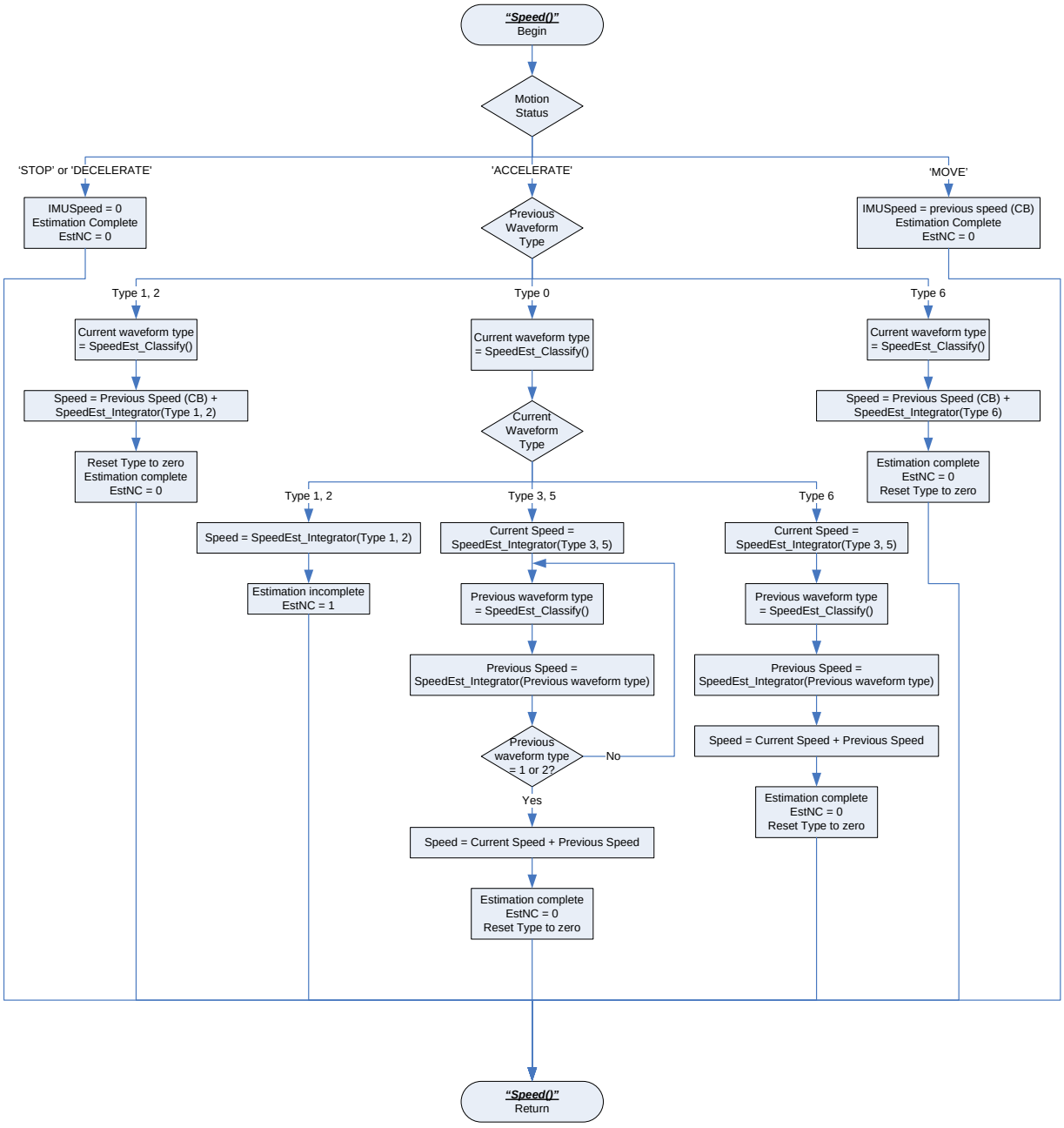

Figure 20 Flowchart of function Speed()

3.2.2.4 Yaw angle calculation

The basic idea is the same as speed calculation. It is to calculate the integral of angular rate ONLY in turning status to obtain the angular displacement. After this, by adding the initial value of magnetic north and the declination, we can get the true north direction.

The problem is if the 'NavBOX' was turning too slowly, the gyroscope may not sense the turning. In other words, if the samples can not reach the threshold, the turning will be overlooked. Even if we lower the threshold, the results after calculating the integral are much smaller than the true value.

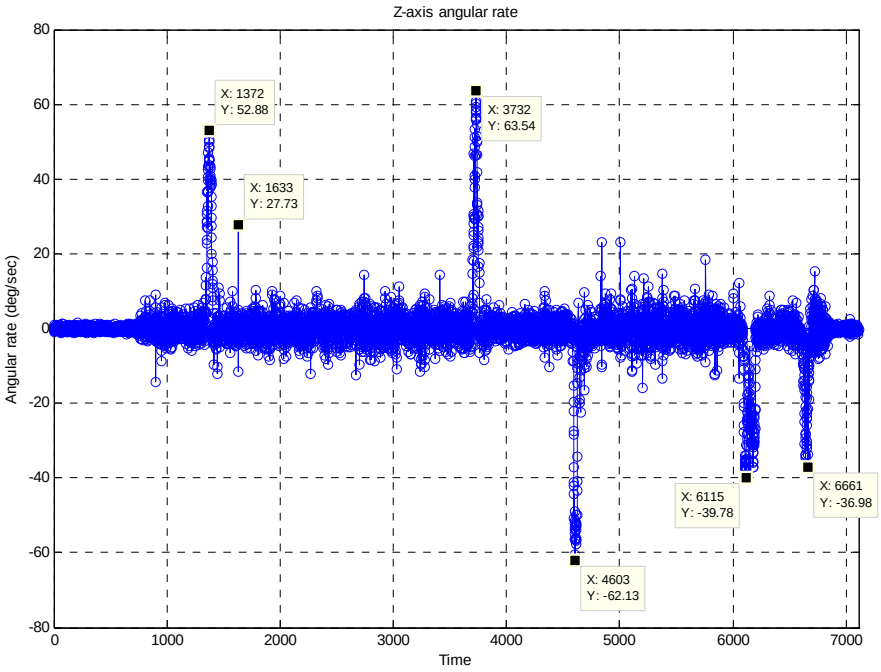

Figure 21 Z-axis gyroscope data

To solve this problem, we use the magnetic north data. Because of the interference from iron-made objects around (cars, buildings etc.), the absolute value of magnetic north data is not reliable. But based on our previous experiments, the relative value is reliable. Hence, the only thing we need to do is marking the beginning and the end of a turning and calculate the difference. (See figure 22)

Figure 23 shows the detailed flowchart of function YawCal().

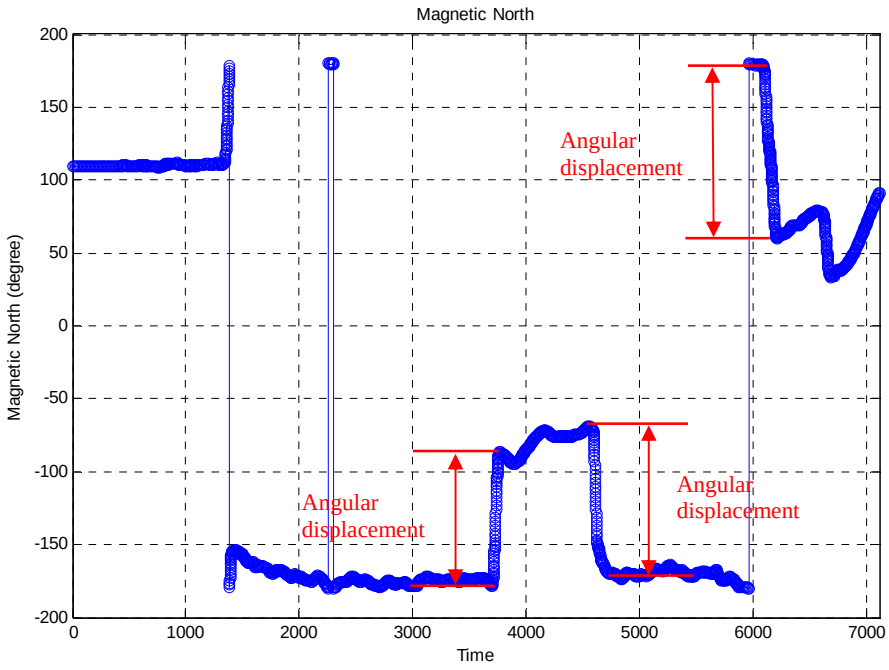

Figure 22 Magnetic north data

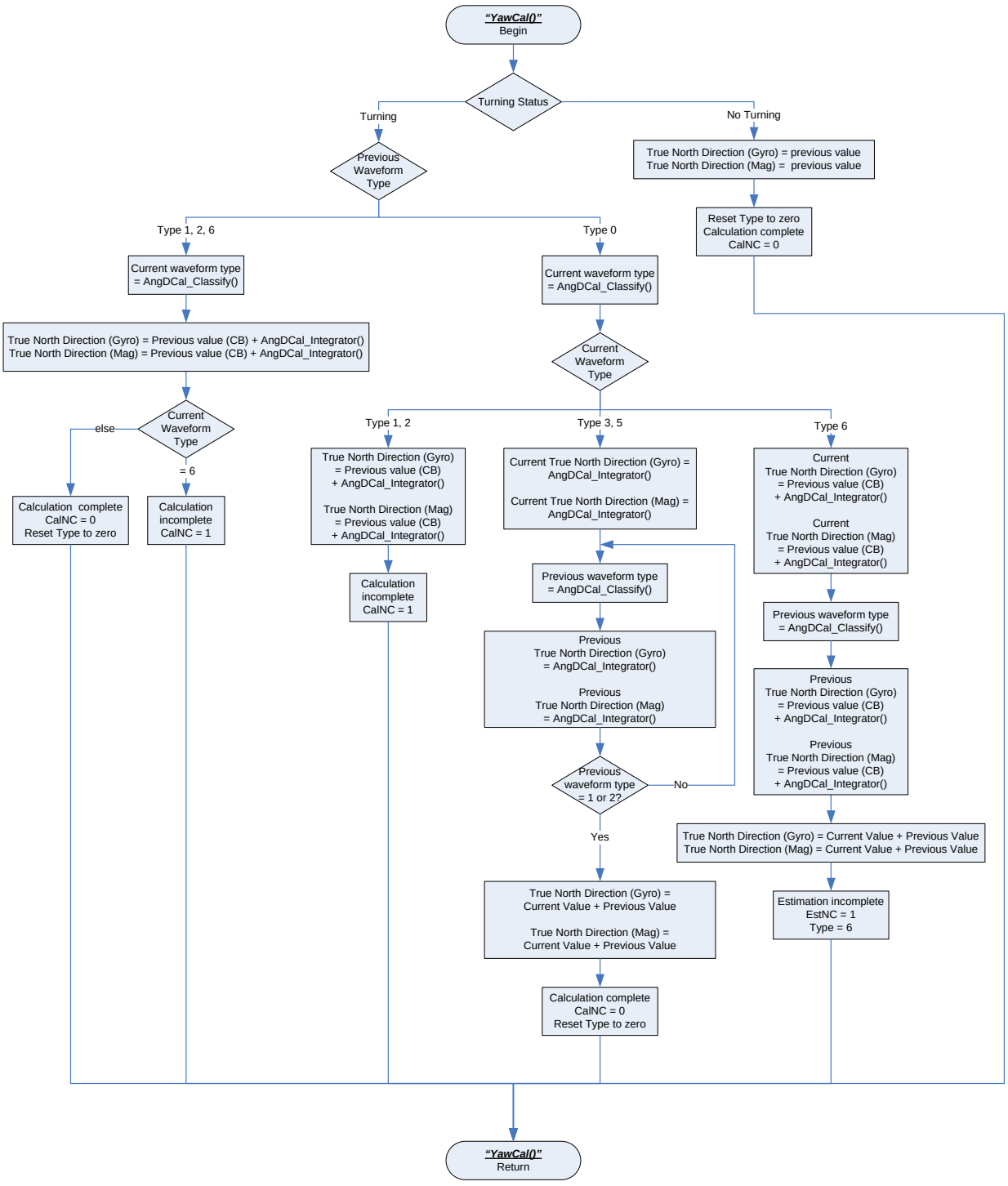

Figure 23 Flowchart of Function YawCal()

3.2.2.5 IMU navigation

This function calculates the current position based on IMU only. It fused the True North Direction from gyroscopes and the True North Direction from magnetometers together.

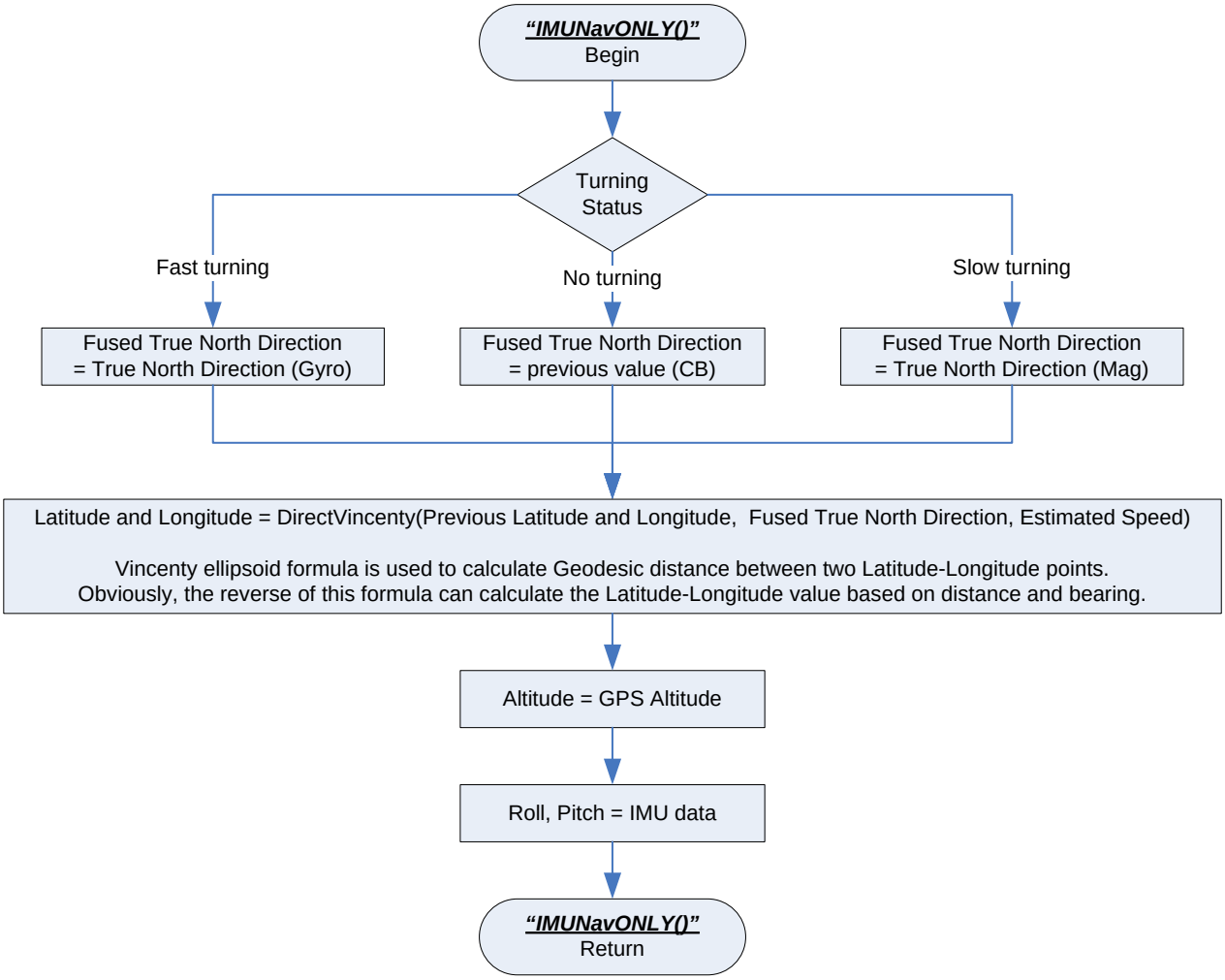

Figure 24 Flowchart of Function IMUNavONLY()

T. Vincenty described his algorithm in the paper 'Direct and Inverse solutions of Geodesics on the Ellipsoid with application of Nested Equations'. It is not difficult to apply these algorithms as Matlab codes.

3.2.2.6 Data fusion

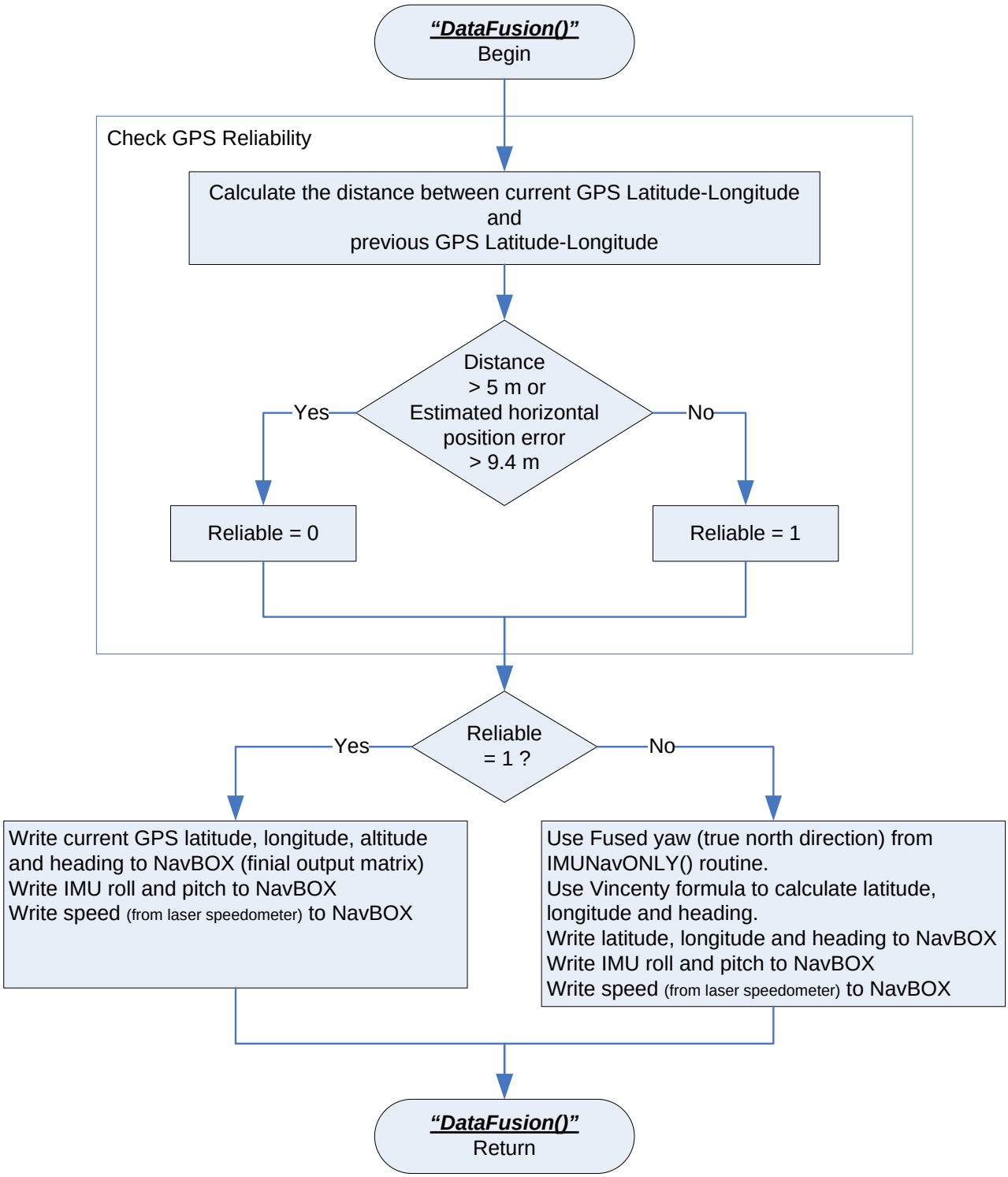

Figure 25 Flowchart of Function DataFusion()

3.3.3 Finial Stage

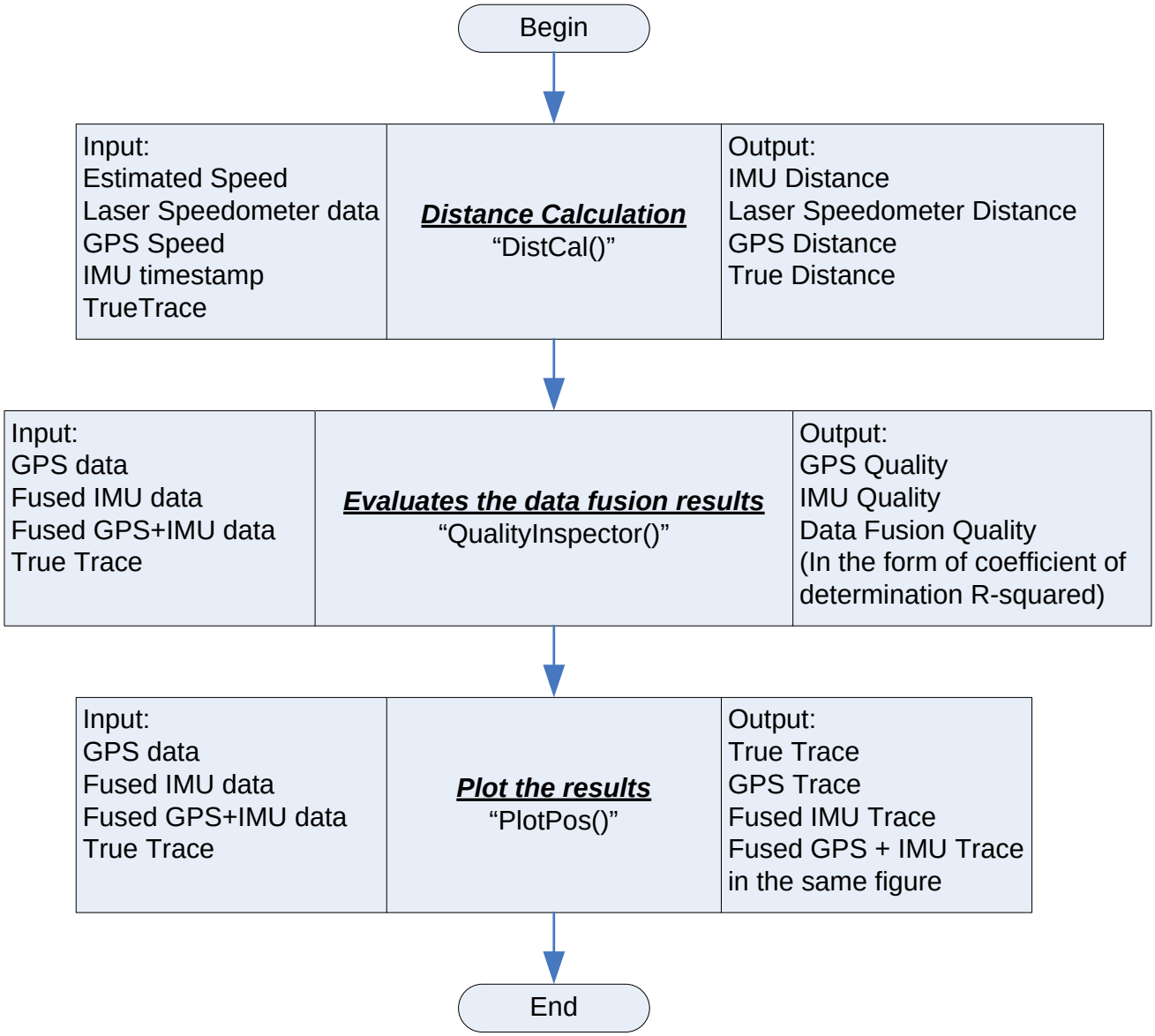

Figure 26 Flowchart of the Finial Stage

The Finial Stage is for comparing and results evaluation. The results from function DistCal() could show the differences among IMU Distance, Laser Speedometer Distance, GPS Distance and True Distance. The function QualityInspector() evaluates the data fusion results to see how well the GPS data and Fused data fit the true trace. The detailed algorithm of QualityInspector() will be discussed in the following chapter.

3.3 Evaluation Method for Data Fusion Results

3.3.1 Theory

In statistics, the coefficient of determination R^2 is the proportion of variability in a data set that is accounted for by a statistical model. In this definition, the term "variability" is defined as the sum of squares. There are equivalent expressions for R^2 based on analysis of variance decomposition. A general version, based on comparing the variability of the estimation errors with the variability of the original values, is

$$R^2 = 1 - \frac{SS_E}{SS_T}$$

Another version is common in statistics texts but holds only if the modelled values are obtained by ordinary least squares regression (which must include a fitted intercept or constant term): it is

$$R^2 = \frac{SS_R}{SS_T}$$

In the above definitions,

$$SS_T = \sum_i \left(y_i - \bar{y}\right)^2$$

$$SS_R = \sum_i \left(\hat{y}_i - \bar{y}\right)^2$$

$$SS_E = \sum_i \left(y_i - \hat{y}_i\right)^2$$

where $y_i, \hat{y}_i$ are the original data values and modelled values respectively. That is, SS_T is the total sum of squares, SS_R is the regression sum of squares, and SS_E is the sum of squared errors. In some texts, the abbreviations SS_R and SS_E have the opposite meaning: SS_R stands for the residual sum of squares (which then refers to the sum of squared errors in the upper example) and SS_E stands for the explained sum of squares (another name for the regression sum of squares).

In the second definition, R^2 is the ratio of the variability of the modelled values to the variability of the original data values. Another version of the definition, which again only holds if the modelled values are obtained by ordinary least squares regression, gives R^2 as the square of the correlation coefficient between the original and modelled data values.

R^2 is a statistic that will give some information about the goodness of fit of a model. In regression, the R^2 coefficient of determination is a statistical measure of how well the regression line approximates the real data points. An R^2 of 1.0 indicates that the regression line perfectly fits the data.

In some (but not all) instances where R^2 is used, the predictors are calculated by ordinary least-squares regression: that is, by minimising SS_E. In this case R-squared increases as we increase the number of variables in the model (R-squared will not decrease). This illustrates a drawback to one possible use of R^2, where one might try to include more variables in the model until "there is no

more improvement". This leads to the alternative approach of looking at the adjusted R^2. The explanation of this statistic is almost the same as R-squared but it penalizes the statistic as extra variables are included in the model. For cases other than fitting by ordinary least squares, the R^2 statistic can be calculated as above and may still be a useful measure. However, the conclusion that that R-squared increases with extra variables no longer holds, but downward variations are usually small. If fitting is by weighted least squares or generalized least squares, alternative versions of R^2 can be calculated appropriate to those statistical frameworks, while the "raw" R^2 may still be useful if it is more easily interpreted. Values for R^2 can be calculated for any type of predictive model, which need not have a statistical basis.

3.3.2 Examples

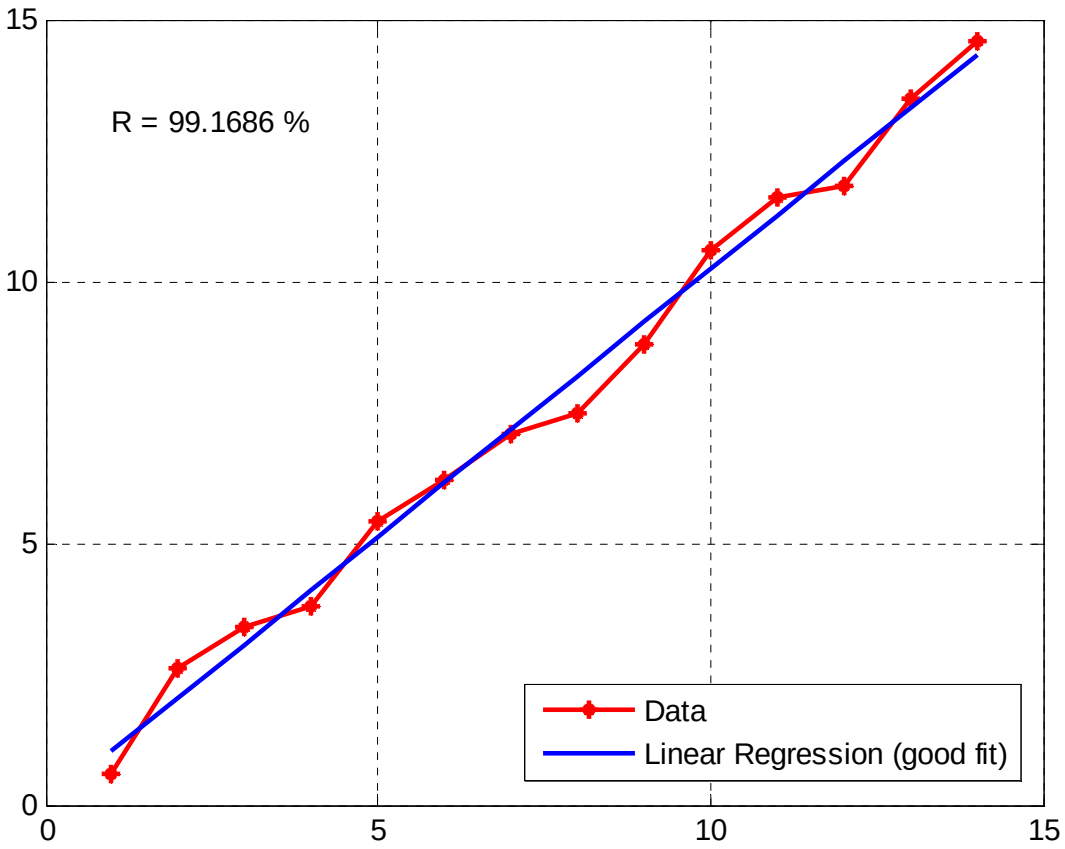

Figure 27 Goodness-of-fit, fit well

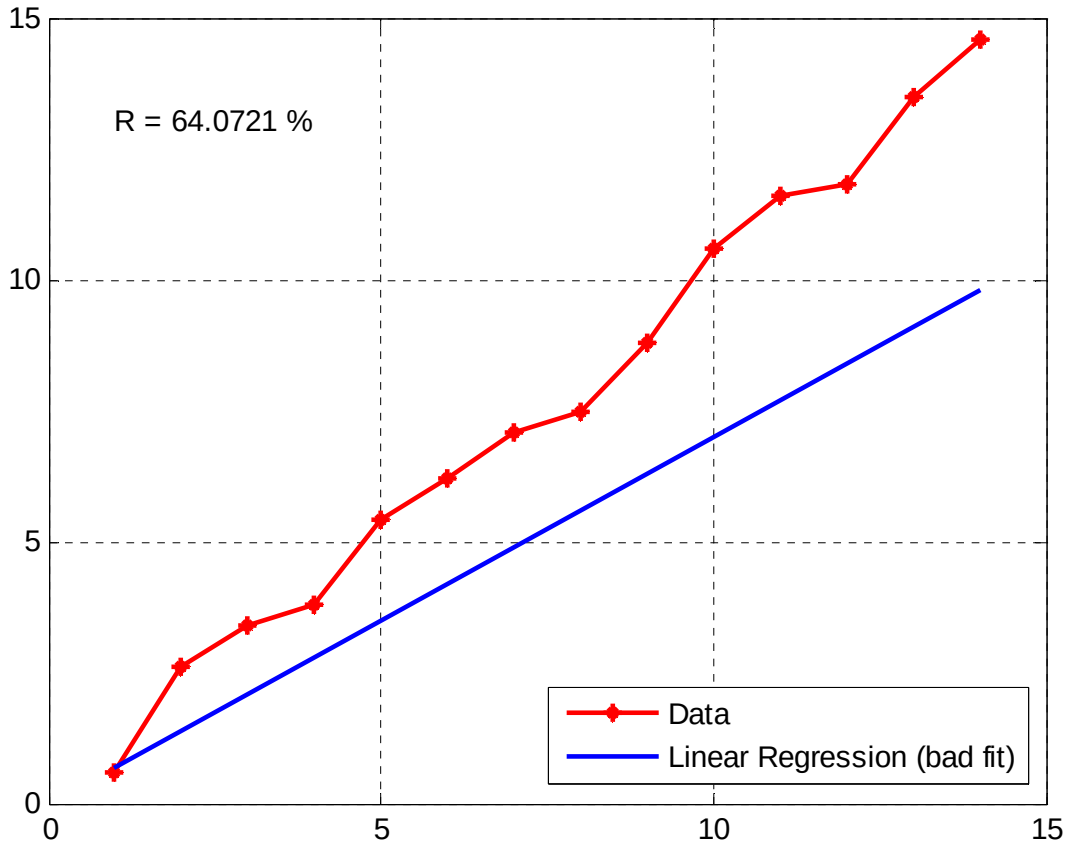

Figure 28 Goodness-of-fit, fit poorly

3.3.3 Application

The coefficient of determination, R-squared is used to measure the goodness-of-fit. If it's close to 1, the linear regression is very close to the raw data.

In our Matlab Program, the red line is the 'GPS data (trace)' or 'Fused data (trace)'; the blue line is the 'True data (trace)'. By using the above algorithms, we could see how close is the 'GPS data (trace)' or 'Fused data (trace)' to the 'True data (Trace)'.

4. Results

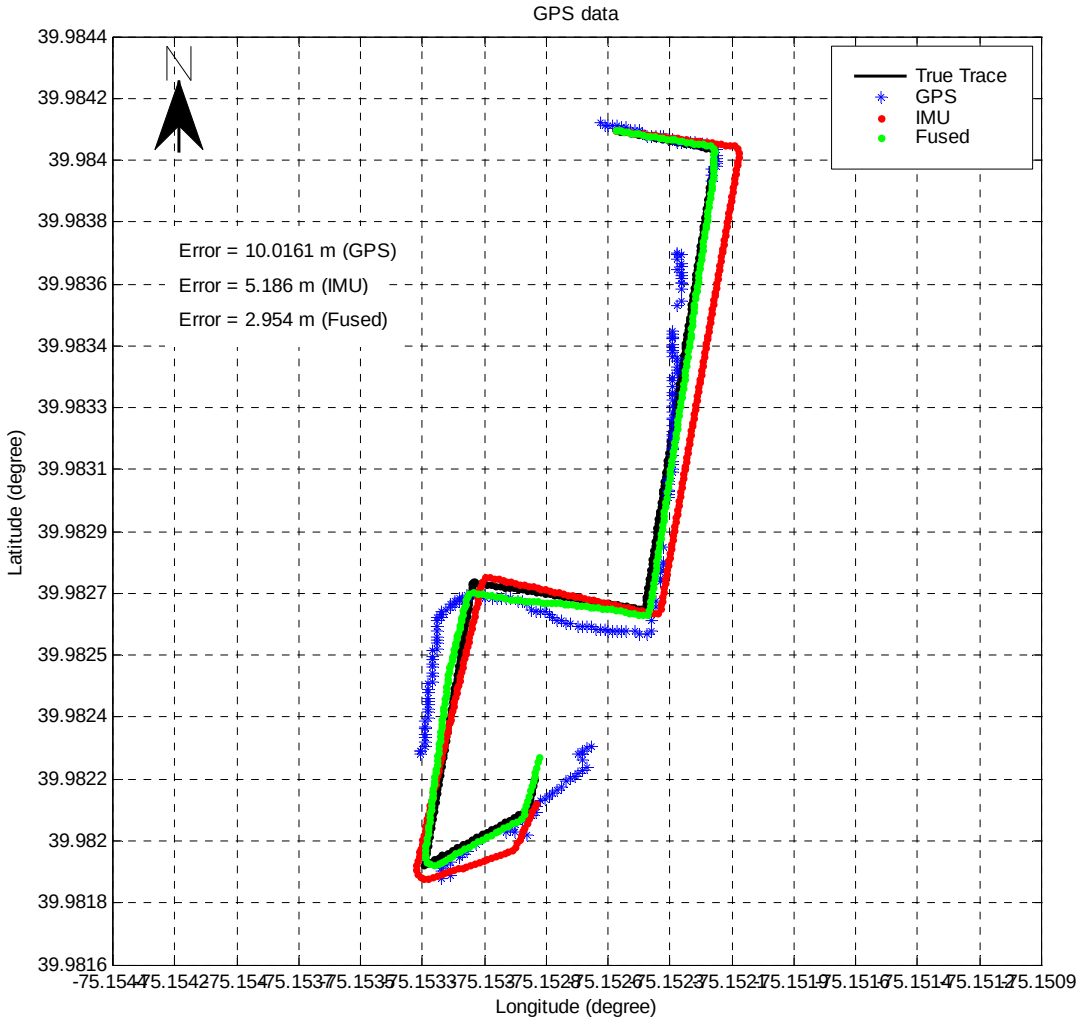

Figure 29 Data fusion results (data set 1)

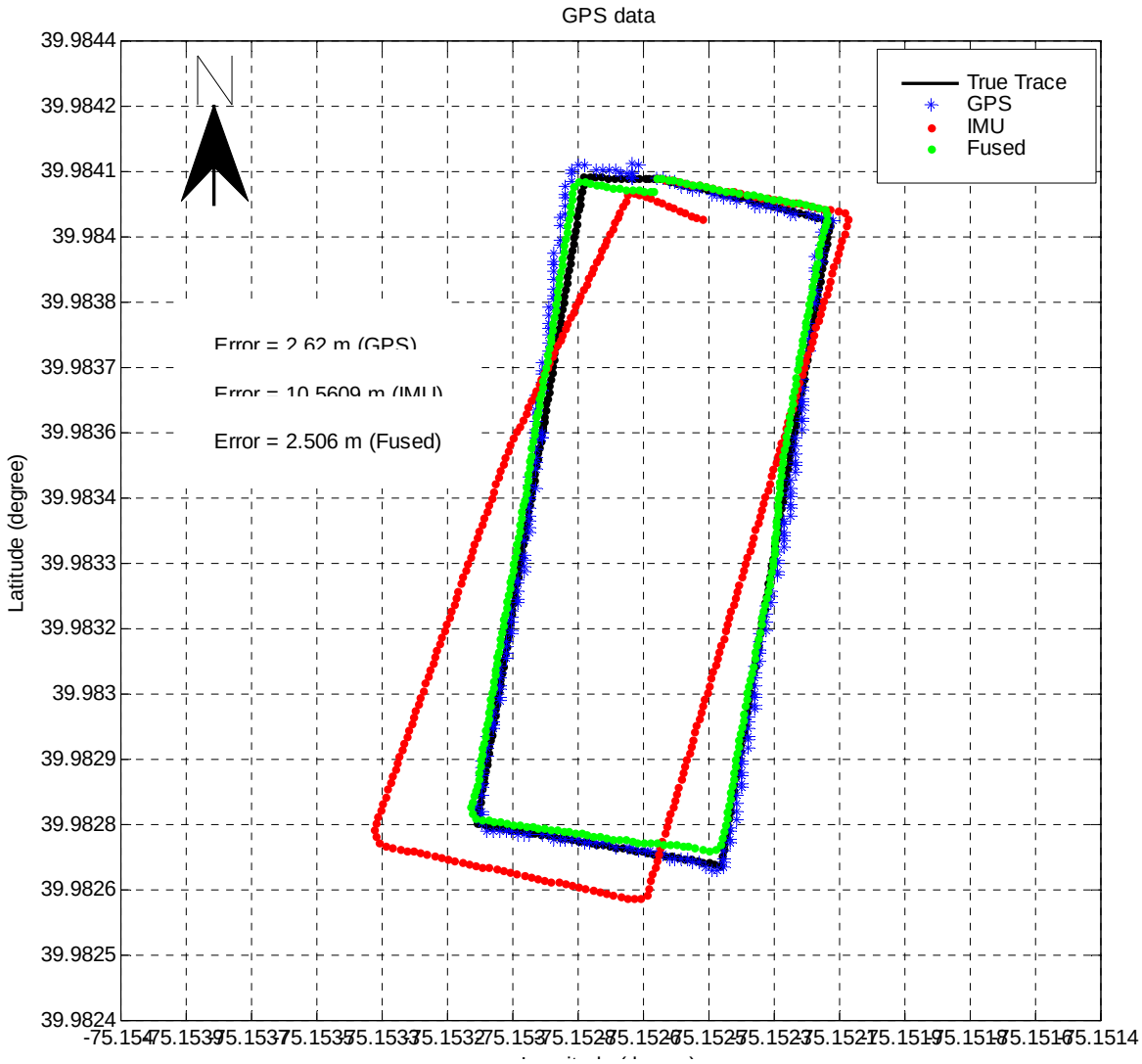

Figure 30 Data fusion results (data set 2)

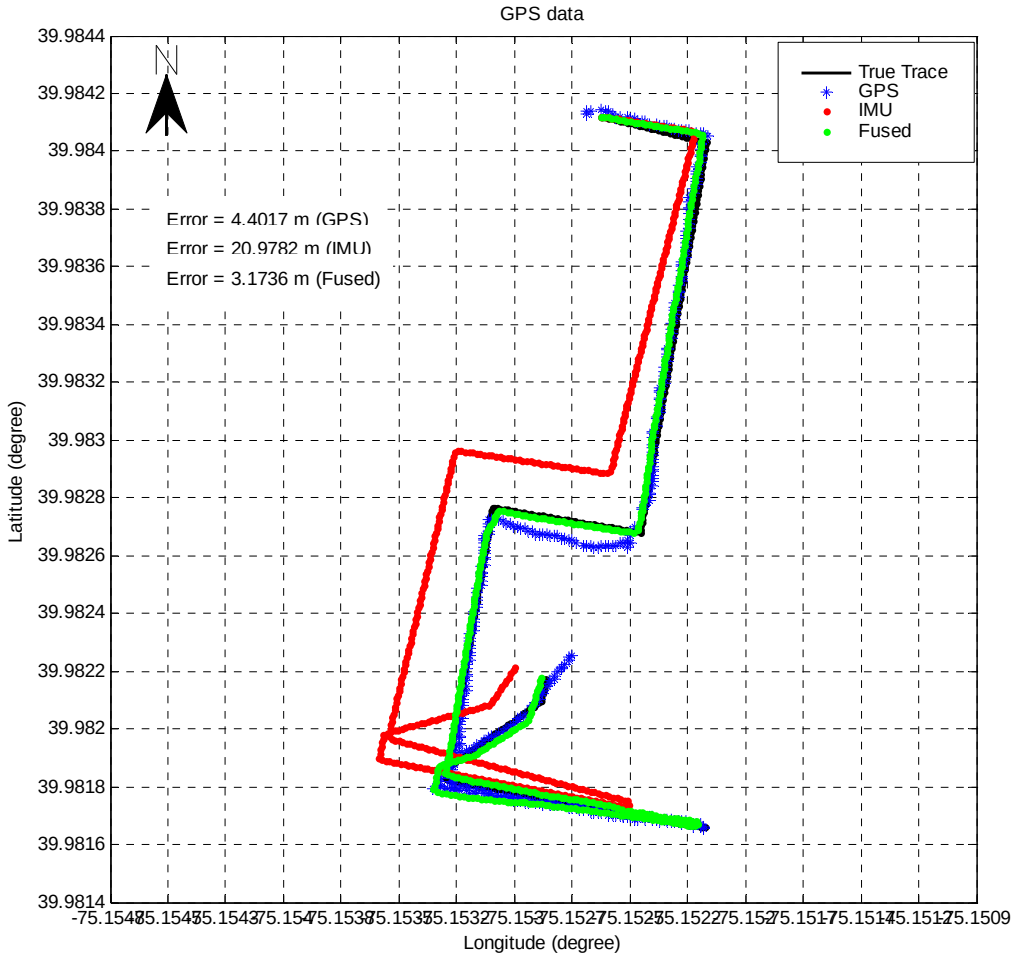

Figure 31 Data fusion results (data set 3)

5. References

[1]. T. Vincenty, “Direct and Inverse solutions of Geodesics on the Ellipsoid with application of Nested Equations”

[2]. Rabbit 3000 Microprocessor User's Manual

Appendix A – Data format

1. Converted GPGGA sentence with header and timestamp
Packet type: 0x04
Data length: 34 bytes
Continuous: 1 Hz

Description: Global Positioning System Fix Data (GGA)
Comment: This message outputs the geodetic position in the WGS84 Ellipsoid.

Byte	Format	Name	Unit	Comment
Byte 0	char	Header[0]	-	synchronization byte 1: 0xAA
Byte 1	char	Header[1]	-	synchronization byte 2: 0xAA
Byte 2	char	Header[2]	-	packet type: 0x04
Byte 3~6	unsigned long	Time	-	UTC time of position fix, hhmmss format
Byte 7~10	float	Lat	-	Latitude, dd.dddddd format
Byte 11~14	float	Lon	-	Longitude, ddd.dddddd format
Byte 15	char	GPSqua	-	GPS quality indication
Byte 16~17	int	NSat	-	Number of satellites in use, 0~12
Byte 18~21	float	HDOP	-	Horizontal dilution of precision, 0.5~99.9
Byte 22~25	float	HMSL	m	Height above mean sea level
Byte 26~29	float	GeoH	m	Geoidal height
Byte 30~33	unsigned long	TimeStamp	ms	timestamp attached by RCM3100

2. Converted GPGSA sentence with header and timestamp
Packet type: 0x05
Data length: 33 bytes
Continuous: 1 Hz

Description: GPS DOP and Active Satellites (GSA)
Comment: This message outputs the geodetic position in the WGS84 Ellipsoid.

Byte	Format	Name	Unit	Comment
Byte 0	char	Header[0]	-	synchronization byte 1: 0xAA
Byte 1	char	Header[1]	-	synchronization byte 2: 0xAA
Byte 2	char	Header[2]	-	packet type: 0x05
Byte 3	char	Mode	-	M = manual, A = automatic
Byte 4	char	Ftype	-	Fix type, 1 = not available, 2 = 3D, 3 = 3D
Byte 5~16	char	PRN	-	PRN number, of satellite used in solution
Byte 17~20	float	PDOP	-	Position dilution of precision, 0.5~99.9
Byte 21~24	float	HDOP	-	Horizontal dilution of precision, 0.5~99.9
Byte 25~28	float	VDOP	-	Vertical dilution of precision, 0.5~99.9
Byte 29~32	unsigned long	TimeStamp	ms	timestamp attached by RCM3100

3. Converted GPRMC sentence with header and timestamp
Packet type: 0x06
Data length: 38 bytes
Continuous: 1 Hz

Description: Recommended Minimum Specific GPS/TRANSIT Data (RMC)
Comment: This message outputs the geodetic position in the WGS84 Ellipsoid.

Byte	Format	Name	Unit	Comment
Byte 0	char	Header[0]	-	synchronization byte 1: 0xAA
Byte 1	char	Header[1]	-	synchronization byte 2: 0xAA
Byte 2	char	Header[2]	-	packet type: 0x06
Byte 3~6	unsigned long	Time	-	UTC time of position fix, hhmmss format
Byte 7	char	Status	-	A = Valid position, V = warning
Byte 8~11	float	Lat	-	Latitude, ddmm.mmmm format
Byte 12~15	float	Lon	-	Longitude, dddmm.mmmm format
Byte 16~19	float	GSpeed	km/h	Ground speed
Byte 20~23	float	Heading	deg	Course over ground, 0.0 ~ 359.9
Byte 24~27	unsigned long	Date	-	UTC date of position fix, ddmmyy format
Byte 28~31	float	MagVar	deg	Magnetic variation, 0.0 ~ 180.0
Byte 32	char	MagVarD	-	Magnetic variation direction, E or W
Byte 33	char	Mode	-	Mode indicator
Byte 34~37	unsigned long	TimeStamp	ms	timestamp attached by RCM3100

4. Converted PGRME sentence with header and timestamp
Packet type: 0x07
Data length: 19 bytes
Continuous: 1 Hz

Description: Estimated Error Information (PGRME)
Comment: This message outputs the geodetic position in the WGS84 Ellipsoid.

Byte	Format	Name	Unit	Comment
Byte 0	char	Header[0]	-	synchronization byte 1: 0xAA
Byte 1	char	Header[1]	-	synchronization byte 2: 0xAA
Byte 2	char	Header[2]	-	packet type: 0x07
Byte 3~6	float	EHPE	m	Estimated horizontal position error
Byte 7~10	float	EVPE	m	Estimated vertical position error
Byte 11~14	float	EPE	m	Estimated position error
Byte 15~18	unsigned long	TimeStamp	ms	timestamp attached by RCM3100

5. IMU packet with header and timestamp
Packet type: 0x31
Data length: 29 bytes
Continuous: 20 Hz

Description: Gyro-Stabilized Euler Angles & Accel & Rate Vector
Comment: The gyro-stabilized Euler Angles and the Instantaneous Acceleration Vector and the drift compensated angular rate vector will be transmitted.

Byte	Format	Name	Unit	Comment
Byte 0	char	Header[0]	-	synchronization byte 1: 0xAA
Byte 1	char	Header[1]	-	synchronization byte 2: 0xAA
Byte 2	char	Header[2]	-	packet type: 0x31
Byte 3~4	int	Roll	-	Scaled factor 360/65536
Byte 5~6	int	Pitch	-	Scaled factor 360/65536
Byte 7~8	int	Yaw	-	Scaled factor 360/65536
Byte 9~10	int	Accel_X	-	Scaled factor 32768000/7000
Byte 11~12	int	Accel_Y	-	Scaled factor 32768000/7000
Byte 13~14	int	Accel_Z	-	Scaled factor 32768000/7000
Byte 15~16	int	CompAngRate_X	-	Scaled factor 32768000/10000
Byte 17~18	int	CompAngRate_Y	-	Scaled factor 32768000/10000
Byte 19~20	int	CompAngRate_Z	-	Scaled factor 32768000/10000
Byte 21~22	int	TimerTicks	-	Time(sec) = TimerTicks * 0.0065536
Byte 23~24	int	Checksum	-	From byte 2 ~ byte 22
Byte 25~28	float	Speed	km/h	Walking speed
Byte 29~32	unsigned long	TimeStamp	ms	timestamp attached by RCM3100

6. EOT (End of Transmission) packet
Packet type: 0x00
Data length: 3 bytes

Description: This packet tells the receiver the end of this transmission.
Comment: once the PC receives EOT packet, the program terminate automatically

Byte	Format	Name	Unit	Comment
Byte 0	char	Header[0]	-	synchronization byte 1: 0xAA
Byte 1	char	Header[1]	-	synchronization byte 2: 0xAA
Byte 2	char	Header[2]	-	packet type: 0x00

Appendix B – Source code of firmware program

Appendix C – Source code of Matlab data process program