

Quadrocopter

Summer Semester 2012



Supervisor: Prof. Dr. Jörg Friedrich
Mr. Vikas Agrawal

Group members:

- Chandana Rokhade
- DongLiang Li
- GuangXun Nan
- Kah Ho Pau
- Nivas Devendran
- Yin Hao

Foreword

This report is divided mainly into two parts. One is the “real” report and the second part is the notes of implementation. The idea is that the future group could operate the hardware and software by just reading the notes of implementation. And if the group wants to know more about the theory behind, they could find it in the first part, the real report.

As group leader, I can’t thank my team members enough for their hard work during the semester to make this project successful. We would like to thank Mr Beltz for his assistance in physical integration of sensors. We would also like to thank Mr Agrawal, Mr Tribek and Mr Schöllig for their support during the whole project period. And most importantly, we would like to thank Prof. Friedrich for giving us the chance to work on this interesting project and providing us guidance throughout the semester.

Kah Ho Pau, 20.07.2012

Contents

Foreword	2
Contents	3
Table of Figures	6
List of Tables.....	8
1. Introduction	9
1.1. Project Definition.....	9
1.2. Physical Model of the quadrocopter	9
1.2.1. Earth frame and body frame	9
1.2.2. Motion of the quadrocopter	10
2. Communication.....	13
2.1. Description.....	13
2.2. XBee	13
2.3. Java implementation	14
2.3.1. Application class.....	14
2.3.2. Implementation.....	16
2.4. Real-time GUI.....	17
2.4.1. Requirement.....	17
2.4.2. ConnectSet class.....	18
2.4.3. TimeSeriesDemo class.....	18
2.4.4. Implementation.....	18
2.5. Future work	19
3. Hardware.....	20
3.1. GPS-Sensor.....	20
3.1.1. Description	20
3.1.2. Sensor Integration	20
3.1.3. Signal Conditioning.....	21
3.1.4. Simulink – Model.....	23
3.1.5. Conclusion	25
3.2. Pressure Sensor	26
3.2.1. Description	26

Quadrocopter

3.2.2.	Problems.....	26
3.2.3.	Suggestions and Conclusion	26
3.3.	Ultrasonic Sensor	27
3.3.1.	Description	27
3.3.2.	Sensor Integration	27
3.3.3.	Signal Conditioning.....	28
3.3.4.	Discussion and Conclusion	30
3.4.	Remote Control & Switches.....	31
3.4.1.	Description	31
3.4.2.	Reading states of switches	31
4.	Software	33
4.1.	Software Architecture	33
4.1.1.	Mode Switching Logic Design:.....	34
4.1.2.	Limitations and Drawbacks of the design:	35
4.2.	Implementations in Matlab/Simulink.....	36
4.2.1.	Introduction.....	36
4.2.2.	Hovering	37
4.2.3.	Soft landing.....	41
4.2.4.	Auto-Cruising.....	41
4.2.5.	Data and analysis.....	43
4.2.6.	Conclusion	44
4.3.	Suggestion for implementations in real quadrocopter	45
4.3.1.	Strategy for Soft Landing.....	45
5.	Summary	46
6.	Notes of Implementation.....	47
6.1.	Environment	47
6.2.	XBee	49
6.2.1.	Install X-CTU software	49
6.2.2.	Install USB Drivers	49
6.2.3.	Configure XBee	49
6.2.4.	Eclipse configuration	49

Quadrocopter

6.2.5.	Create project.....	49
6.2.6.	Implementation.....	50
6.3.	GPS-Sensor and related functions	51
6.3.1.	GPS – Sensor mounting	51
6.3.2.	GPS – Function	51
6.3.3.	GPS – Simulation model (GPS Model)	52
6.4.	Ultrasonic Sensor.....	53
6.4.1.	Sensor mounting	53
6.4.2.	Software for height sensor	55
6.5.	Remote Controller	57
6.5.1.	Set up switches to channels	57
6.5.2.	Calibration of Remote Controller (Mx-16s).....	59
6.6.	Simulation	60
7.	Appendix	62
7.1.	GPS-signals.....	62
7.2.	Ultrasonic Sensor.....	66
8.	Bibliography	68

Table of Figures

Figure 1-1 Quadrocopter.....	9
Figure 1-2 Earth frame (green) and body frame (red) (Source: [1]).....	10
Figure 1-3 Thrust (Source: [1]).....	11
Figure 1-4 Pitch (Source: [1]).....	11
Figure 1-5 Roll (Source: [1])	12
Figure 1-6 Yaw (Source: [1]).....	12
Figure 2-1 X-CTU Command Window.....	14
Figure 2-2 Process of communication between PC and Quadrocopter.....	15
Figure 2-3 Console Window of Eclipse	17
Figure 2-4 Graphical User Interface for monitoring sensor values.....	19
Figure 3-1 GPS sensor.....	20
Figure 3-2 Mounting of GPS receiver	21
Figure 3-3 Long signal 3 (Latitude and Longitude)	22
Figure 3-4 GPS Noise Model.....	24
Figure 3-5 GPS Noise Model (Detailed).....	24
Figure 3-6 Results of GPS noise Model	25
Figure 3-7 Ultrasonic Sensor	27
Figure 3-8 Mounting of Ultrasonic Sensor	27
Figure 3-9 Pitching movement (higher height, 1 st test)	28
Figure 3-10 Pitching movement (lower height, 1 st test)	29
Figure 3-11 Rolling movement (higher height, 1 st test)	29
Figure 3-12 Remote Controller.....	31
Figure 4-1 Flow-Chart for Mode Switching Logic	34
Figure 4-2 State space control for copter flight control with hovering, height and auto cruising function.....	36
Figure 4-3 Controller Block.....	37
Figure 4-4 Drifting due to inertia.....	37
Figure 4-5 Braking effect by adding speed controllers	38
Figure 4-6 Transformation Simulink Block	38
Figure 4-7 Hovering Controller.....	39
Figure 4-8 Hovering Height Setting Block	40
Figure 4-9 Winding effect of I-element.....	40
Figure 4-10 Cut off switch block.....	40
Figure 4-11 Soft-landing	41
Figure 4-12 Auto Cruising Control	42
Figure 4-13 Position controller for auto cruising mode	42
Figure 4-14 Analysis of different modes	43
Figure 4-15 Soft-Landing Strategy.....	45

Quadrocopter

Figure 6-1 CodeWarrior.....	47
Figure 6-2 Debugger Window	48
Figure 6-3 Pin-Connection of Ultrasonic sensor	53
Figure 6-4 Pin connection of board.....	53
Figure 6-5 Wires connection	54
Figure 6-6 Calibration.....	59
Figure 6-7 Simulation Model.....	60
Figure 7-1 Long Signal 1.....	62
Figure 7-2 Long Signal 2.....	63
Figure 7-3 Long Signal 3.....	64
Figure 7-4 Height 1&2	65
Figure 7-5 Pitching movement (higher height, 2 nd test)	66
Figure 7-6 Pitching movement (lower height, 2 nd test)	66
Figure 7-7 Rolling movement (higher height, 2 nd test)	67
Figure 7-8 Rolling movement (lower height, 1 st test)	67
Figure 7-9 Rolling movement (lower height, 2 nd test)	67

List of Tables

Table 2-1 Process of communication between PC and Quadrocopter	16
Table 3-1 Signal Analysis – Latitude, Longitude	21
Table 3-2 Signal Analysis - Altitude	22
Table 3-3 Signal Analysis	29
Table 6-1 Ultrasonic Sensor - Functions	56

1. Introduction



Figure 1-1 Quadrocopter

1.1. Project Definition

A quadrocopter is an autonomous flying drone with four rotors on its edges. It will be quite helpful in a lot of cases, thanks to its agility and size. It can scout out regions which are out of reach for humans or dangerous to humans, e.g. scouting in Nuclear Plant Fukushima. The ultimate goal of this project, Quadrocopter, would be to create an autonomous flying drone which is capable to perform scout missions in any landscape or some perform some simple instructions (like buying pizza). In this semester, the goals are:

- To establish communication between quadrocopter and PC through “xBee”.
- To add new functions: “Hovering and Soft Landing” and “Position Recording and Retaining Recorded Position”.
- To install additional sensors (GPS sensor and ultrasonic sensor) to support the functions.

1.2. Physical Model of the quadrocopter

As this is not the first group working on the quadrocopter, there are a few perfect introductions to the working principles of a quadrocopter in previous project reports (see [1], [2]). In this chapter, only those are important for understanding this project will be discussed.

1.2.1. Earth frame and body frame

To describe the behaviour of a quadrocopter, there are two coordination systems to be considered. The first one is earth frame, which is represented in green in Figure 1-2. It gives information about the geographical position of quadrocopter relative to the earth, but it does not give any information about the way quadrocopter is flying (upright or upside down). This information could be acquired easily using GPS-sensor. The second one is body frame,

which is fixed on quadrocopter and represented in red in same figure. It gives the information how the quadrocopter is flying (forward, sideward or backward). This frame is important for controlling the quadrocopter as the controller influences the motions in this frame directly (See chapter 1.2.2). This information could be acquired through the accelerometers and gyroscopes.

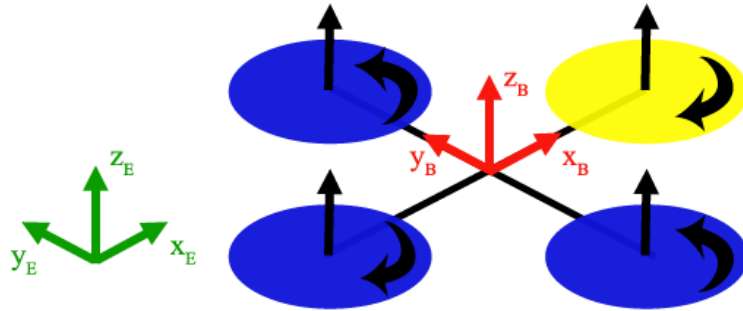


Figure 1-2 Earth frame (green) and body frame (red) (Source: [1])

To fully describe the behaviour of a quadrocopter, information of both frames is needed. The information of different frames can be transformed to each other using transformation matrix (See chapter 4.2.2).

1.2.2. Motion of the quadrocopter

In simple words, to control the motion of quadrocopter, each rotor can be controlled to rotate at different speeds and the faster it rotates, the more lift force is produced by that rotor. There are theoretically six degrees of freedom for quadrocopter's motions:

- Moving in x direction
- Moving in y direction
- Thrust (Moving in z direction)
- Rolling (Rotation around the x-axis)
- Pitching (Rotation around the y-axis)
- Yawing (Rotation around the z-axis)

But in effect, there is only four degree of freedoms, as the motion in x and y direction is the result of rolling and pitching.

Thrust

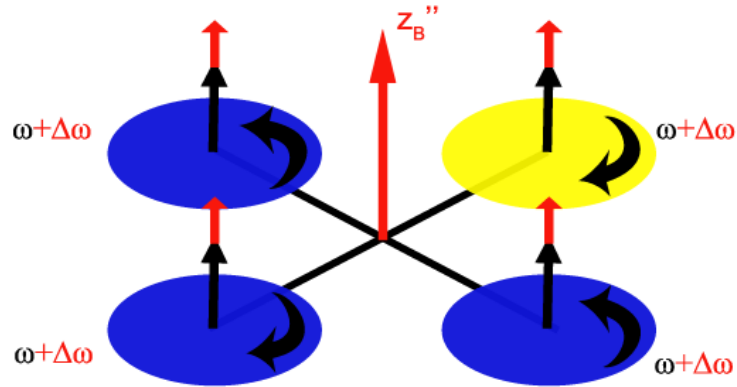


Figure 1-3 Thrust (Source: [1])

To increase the thrust, a same offset is added to all the rotors, thus increasing the same lift force at each rotor and resulting in a motion in z-direction.

- Pitch

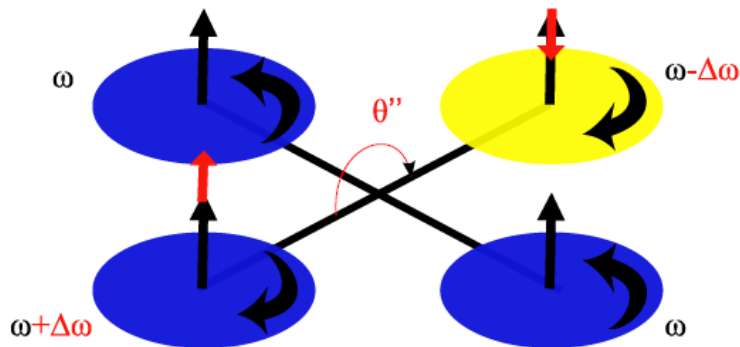


Figure 1-4 Pitch (Source: [1])

To pitch forward, front rotor is slowed down and rear rotor increases its speed. As a result, the quadrocopter rotate around the y-axis and the quadrocopter moves forward.

Quadrocopter

- **Roll**

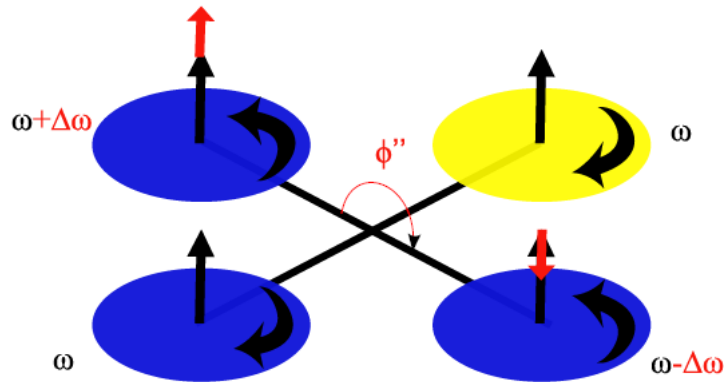


Figure 1-5 Roll (Source: [1])

To roll to the right, right rotor is slowed down and the left rotor increases its speed. As a result, the quadrocopter rotate around the x-axis and the quadrocopter moves to the left.

- **Yaw**

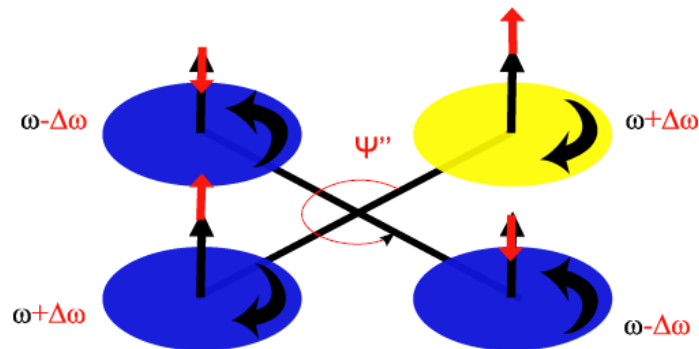


Figure 1-6 Yaw (Source: [1])

To rotate the quadrocopter around z-axis, one of the rotor-pairs (front-rear) is accelerated and the other pair (left-right) is decelerated. This results in an imbalance of resultant moment and thus produces a torque around z-axis.

2. Communication

This chapter describes and introduces the communication between quadrocopter and host PC. Beginning with a description of this subproject and the task it fulfills. Then it advances with short introduction into XBee. It also gives explanation of the Java implementation and the development of graphical user interface. Regarding the optimization and extension of the project, some suggestions for future work is also listed.

2.1. Description

As the quadrocopter is mounted with a number of sensors, it is necessary to monitor those sensor values in real time. To establish the communication between quadrocopter and PC, two XBee modules are needed, one is mounted on quadrocopter and the other is attached to PC through serial port. Besides, there is analysing software with protocol existing, which is accomplished in a study work called “Valiquad” in semester 2010/2011 by Mr.Stübler [3]. The analysing software is written in java and uses radio-frequency to communicate with the quadrocopter. It provides the possibility to read sensor values from quadrocopter over time, so as to monitor and analyse the flight behaviour.

Regarding the detail of the task in this part, there are two main sections. In the earlier period, it was required to store sensor values in files, like txt, to test the new sensors. For instance, for the GPS sensor, it's necessary to measure the position accuracy, which requires tracking of the corresponding sensor values for a certain period. Therefore it's quite helpful to store all those values during test into a file, which can be analysed later. Besides, as software part is written in Java, it is easy to export files.

The second task, which is also the main section in this part, is to create a real time graphical user interface to monitor the flight behaviour. To achieve this, Jfreechart is implemented; details would be introduced later in the chapter 2.4.

2.2. XBee

XBee is the brand name from Digi International family of form factor compatible radio modules [4], which uses the ZigBee communication standard, enabling the transmission of data over longer distances, passing data through intermediate devices to reach the more distant ones. Regarding the project, there are two XBee devices required, one is installed on the Quadrocopter, and the other is attached to PC.

The protocol used for the data transmission is already developed by previous group, which is based on packets transmitted from one node to another. Such packet has a maximum length of 64 Bytes, and normally the Xbee module can allow a transmission of packets with size up to 70 to 100 Bytes. Therefore a transmission of 64 Bytes is ensured with the current XBee module.

As to the baud-rate of XBee module, currently it's 9600 baud/sec, which could be set using X-CTU. X-CTU is a stand-alone tool for configuring XBee modules, as shown in Figure 2-1. Detail description of X-CTU installation and configuration is introduced in the chapter 6.2.

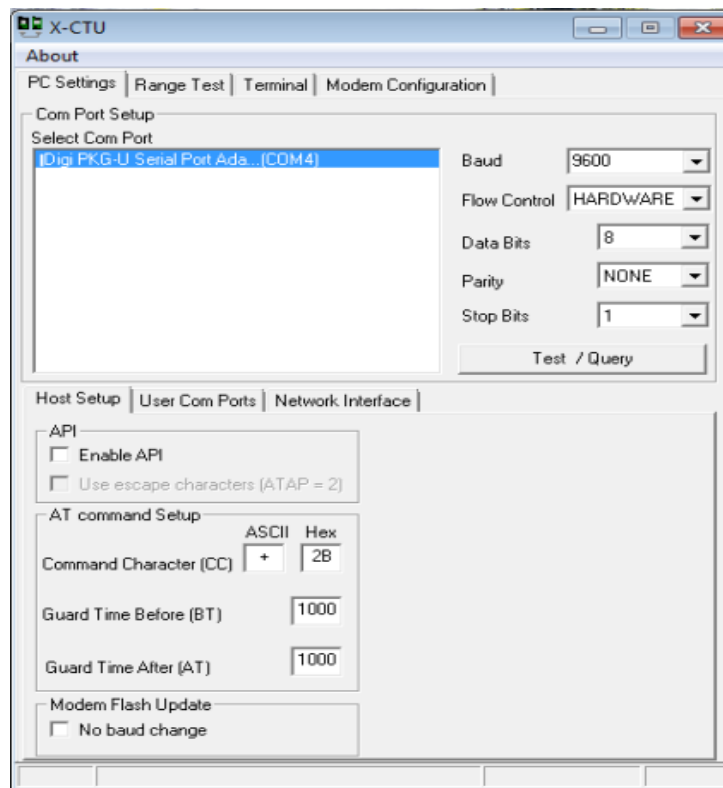


Figure 2-1 X-CTU Command Window

2.3. Java implementation

For the communication part, the programming language Java is used, with the tool Eclipse. It mainly develops the application class, with which the communication is achieved and received data can be stored in output file.

2.3.1. Application class

For communication between XBee and quadrocopter, the analysing software part has been developed. Therefore it's only necessary to create a new class called "Application" which implements all the corresponding functionalities.

As shown in Figure 2-2, the general procedure of the communication is divided into three steps, among of which, the first one is most important and complex. During configuration, class ConnectionAsync creates asynchronous serial connection; class Analyzer is to analyse and process received messages; class Messenger holds an object derived from ConnectionService interface that is responsible for handling the low level communication;

Quadrocopter

class Configuration holds itself an instance of a Communication object, which handles the low level part of the message and packet transmission.

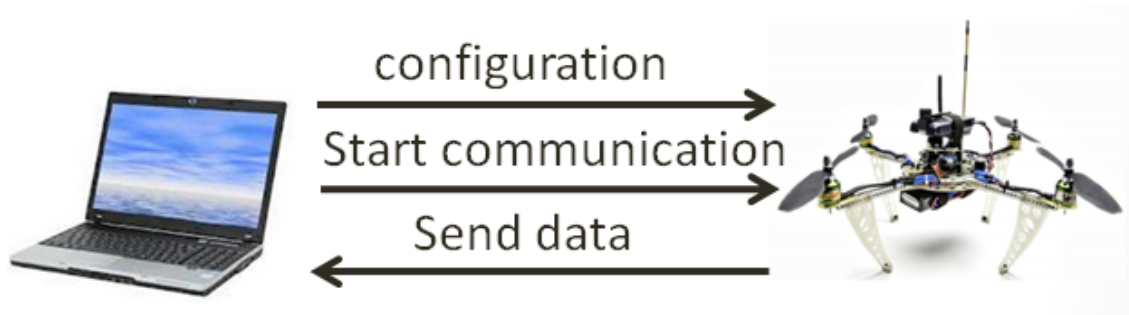


Figure 2-2 Process of communication between PC and Quadrocopter

Before sending configuration, it is necessary to define what kind of sensor value would be observed, which is implemented by the method `addObservable` in class `Configuration`. To implement this method, the so called parameter ID is needed, which individually represent certain kind of sensor value. For the new GPS sensor and ultrasonic sensor, new parameter IDs are also defined, which is done in source code `"valiquad.c"`. Table 2-1 lists all current sensor values with corresponding parameter IDs.

After adding the expected observable variables, configuration can be sent. Afterwards, it's expected to send a starting message to start the communication between those two nodes. Then corresponding sensor values are sent back to PC, which can be stored in files with Java `BufferWriter`. For the measuring duration, it's currently controlled by an integer variable in the while loop. Therefore it's possible to modify it to meet any required measuring duration. However, only after the whole loop runs out, the output file can be built up showing all stored data, any interrupt of the loop would result in an empty output file.

Quadrocopter

Parameter-ID	Data Type	Parameter		Parameter-ID	Data Type	Parameter
0xD000	int16	accXRaw		0xA016	uint8	setpointRear
0xD001	int16	accX		0xD017	int16	forceRight
0xD002	int16	accYRaw		0xA018	uint8	setpointRight
0xD003	int16	accY		0xD019	int16	forceLeft
0xD004	int16	accZRaw		0xA01A	uint8	setpointLeft
0xD005	int16	accZ		0xD01B	int16	forceTotal
0xD006	int16	angVelRRaw		0xA020	uint8	remoteConnected
0xD007	int16	angVelR		0xA021	uint8	remoteMotorsOn
0x8008	float32	angR		0xA022	uint8	remoteForceRaw
0xD009	int16	angVelPRaw		0xD023	int16	remoteForce
0xD00A	int16	angVelP		0xA024	uint8	remoteYawRaw
0x800B	float32	angP		0xD025	int16	remoteYaw
0xD00C	int16	angVelYRaw		0xA026	uint8	remotePitchRaw
0xD00D	int16	angVelY		0xD027	int16	remotePitch
0xD00E	int16	tempRaw		0xA028	uint8	remoteRollRaw
0xD00F	int16	Temp		0xD029	int16	remoteRoll
0xD010	int16	airPressureRaw		0xD02A	int16	altitude
0xD011	int16	batteryRaw		0xF02B	Int32	LatitudeRAW
0xD012	int16	battery		0xF02C	Int32	LongitudeRAW
0xD013	int16	forceFront		0xF02D	Int32	Latitude
0xA014	uint8	setpointFront		0xF02E	Int32	Longitude
0xD015	int16	forceRear		0xF02F	Int32	AltitudeRAW

Table 2-1 Process of communication between PC and Quadrocopter

2.3.2. Implementation

Before implementing the application class, the port number should be defined correctly, which can be checked with X-CTU. After that, run the application class. In the Console window, there will be some commands as shown below. First is the file name expected to be saved, second is the corresponding sensor ID. The saved file will be located in the same workspace with the project. In such way, sensor value would be read and stored in the

output file. The stored data can be exported to excel, for further analysing purposes, e.g. graphs.

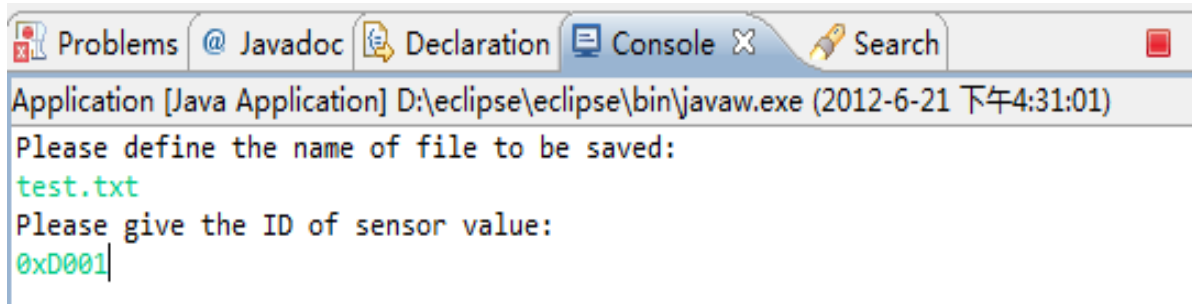


Figure 2-3 Console Window of Eclipse

2.4. Real-time GUI

To monitor and analyse the flight behaviour, a real-time graphical user interface is required to present different sensor values in real time manner on charts like an oscilloscope. For this purpose, Jfreechart is currently used, which is an open-source framework of Java, which allows the creation of a wide variety of both interactive and non-interactive charts [4]. As the sensor values are required to be present in real time, the TimeSeries is implemented, which can represent a sequence of data items in form of value and period, and such period can be defined, such as second or minute. Therefore, an “oscilloscope” is achieved with time as x-axis and value as y-axis.

2.4.1. Requirement

Regarding the features of such kind of real-time graphical user interface, there are several requirements as following:

- Multiple vertical axis

It's required to read several sensor values and display them in one graph, as different sensor values have different ranges, therefore each sensor needs its own vertical axis to represent its values.

- Automatic scaling

The display of sensor values should be flexible. It's not a good idea to have a predefined range of the vertical axis. Instead, axis scale should be automatically changed based on corresponding sensor values.

- Zoomable

Similar with many other kinds of display, the graph should be zoomable, enabling user to zoom in a certain part of graph for detail analysis.

- Pannable

It means user can scroll the graph back or forward along the time axis. As the screen only shows the graph in a limit range of time, “pannable” functionality would help user to view the graph in the past.

2.4.2. **ConnectSet class**

This class is to set up the connection between PC and quadrocopter, which is similar to the Application class, including configuration, starting message. Besides, the connectSet class defines a method called gettingdata, which would be implemented in TimeSeriesDemo class. This method is to read sensor values with two parts, that’s parameter ID and data value, both of them are stored in the array.

2.4.3. **TimeSeriesDemo class**

This is the main class for the graphical user interface. Currently it supports four vertical axes, which means four different sensor values can be present in a same chart, each with corresponding vertical axis scale.

The main method in this class is TimeSeriesDemo. First, it starts the thread, which is defined in the run method. The run method implements the method gettingdata that is defined in connectSet class, storing data value and corresponding parameter ID in the array. Based on the received parameter ID, object “timeseries” adds the data information together with time in the form [period, value]. After running the thread, “timeseriesscollection” starts to collect time series objects. The TimeSeriesCollection class implements the XYDataset interface, therefore the collected series objects are used as dataset, which can be plotted by implementing methods of class XYPlot. Therefore the received data information could be presented in the vertical axis, and the horizontal axis would display the corresponding time.

The method getScrollBar is implemented to pan the scope horizontally, which means user could scroll back the graph to check the history information. Meanwhile, the zoom functionality is default in Jfreechart, therefore it is not necessary to implement new method.

2.4.4. **Implementation**

Run the class *TimeSeriesDemo*. This class is implemented to plot the sensor values in real time graph as shown in Figure 2-4. To zoom in the graph, just move the mouse to the desired position and drag downward; to go back to normal view, drag upward; to move the graph, click on the bar on left side of screen to move back and click on the bar on right side to move forward, drag upward to go back to normal view.



Figure 2-4 Graphical User Interface for monitoring sensor values

2.5. Future work

Currently the crucial problem for communication between PC and quadrocopter is reading rate. In the moment, it can only read around four sensor values per second. There is currently one option available, which is to change the sub-period for the system, which in other words is to modify the sampling-rate, by the method `setSubPeriod` that is implemented in class `Configuration`. However, after increasing the sampling-rate, the system becomes unstable, which results in transmission termination in a random way. In this case, this problem is the first issue for the future work on the part of communication.

On the other hand, the “oscilloscope” is expected to further extend its functionalities, such as adding checkbox for selecting various sensor values, instead of predefining them in the class `TimeSeriesDemo` as used in current method.

3. Hardware

In this chapter, the sensors and any relevant hardware will be discussed in detail. At the beginning of this project, the quadrocopter had accelerometers and gyroscopes for three axes and pressure sensor integrated. During the project period, new GPS-sensor and ultrasonic sensor were integrated. In this chapter, detailed description about the functionalities of components will not be discussed, as it is available in [4] and data sheets, and it is not the major part of this project. This chapter deals mostly with what has been done throughout this semester.

3.1. GPS-Sensor

3.1.1. Description

GPS (Global positioning system), or officially NAVSTAR GPS, is a system developed by US military since 1970s and it is now widely used in civil and commercial applications. Its purposes, as stated like its name, are position, speed and time determining. In order to determine its position, a GPS receiver needs to access to at least four satellites.



Figure 3-1 GPS sensor

In this project, the GPS sensor, LS20032, from Locosys, is used. The communication between the sensor and microcontroller is done through serial connection RS232 and it supports several protocols (NMEA 0183 version 3.01, including RMC, GGA, GLL, GSA, GSV and VTG). The maximum update rate is 5 Hz. Time for signal acquisition is 36s for cold start and 2s for hot start. For more details of GPS sensor and the protocols, please refer to the [5] and [6]. The main tasks of this project are integrating the sensor and signal conditioning.

3.1.2. Sensor Integration

The driver for this sensor has been developed by previous group (see [6]). Therefore the first task was to integrate the driver into the microcontroller of quadrocopter and as well as the sensor itself physically. The latter task was done with help of Mr Beltz.



Figure 3-2 Mounting of GPS receiver

Integrating the driver was taking slightly more time than expected due to a small bug in the driver. This is due to the different development environment. The driver works perfectly with the evaluation board but not with the microcontroller of quadrocopter. After some extensive debugging, the error is identified and removed.

The driver allows the microcontroller to extract data included in RMC protocol: time, date, speed, latitude and longitude. In this project, the driver is extended to include additional parameter: altitude, which is not included in RMC protocol, but in GGA protocol. For more details on adding additional parameter, please refer to chapter 6.3.2.

3.1.3. Signal Conditioning

In order to control the quadrocopter with the signal from GPS, the reliability of the GPS signal must be guaranteed. In this subchapter, the three signals: latitude, longitude and altitude will be analysed. In this project, only signal under static state is analysed, i.e. the quadrocopter is left on ground for one to ten minutes.

Condition	Duration (min)	Deviation in Latitude (m)	Deviation in Longitude (m)	Resultant Deviation (m)
Long Signal 1	3	12.36	5.51	13.53
Long Signal 2 (C)	10	17.03	15.19	22.82
Long Signal 3 (H)	10	6.68	6.03	9.00

Table 3-1 Signal Analysis – Latitude, Longitude

Quadrocopter

Condition	Duration (min)	Deviation in Altitude (m)
Height 1 (C)	10	4
Height 2 (H)	10	9

Table 3-2 Signal Analysis - Altitude

(C): Cold start (H): Hot start

The deviation is defined as the maximum detected value minus the minimum detected value. Cold start means that readings are taken immediately after the sensor starts up and gets the first signal. While hot start means that readings are taken only after the sensor has been turn on for quite some time, e.g. after taking the first round of readings.

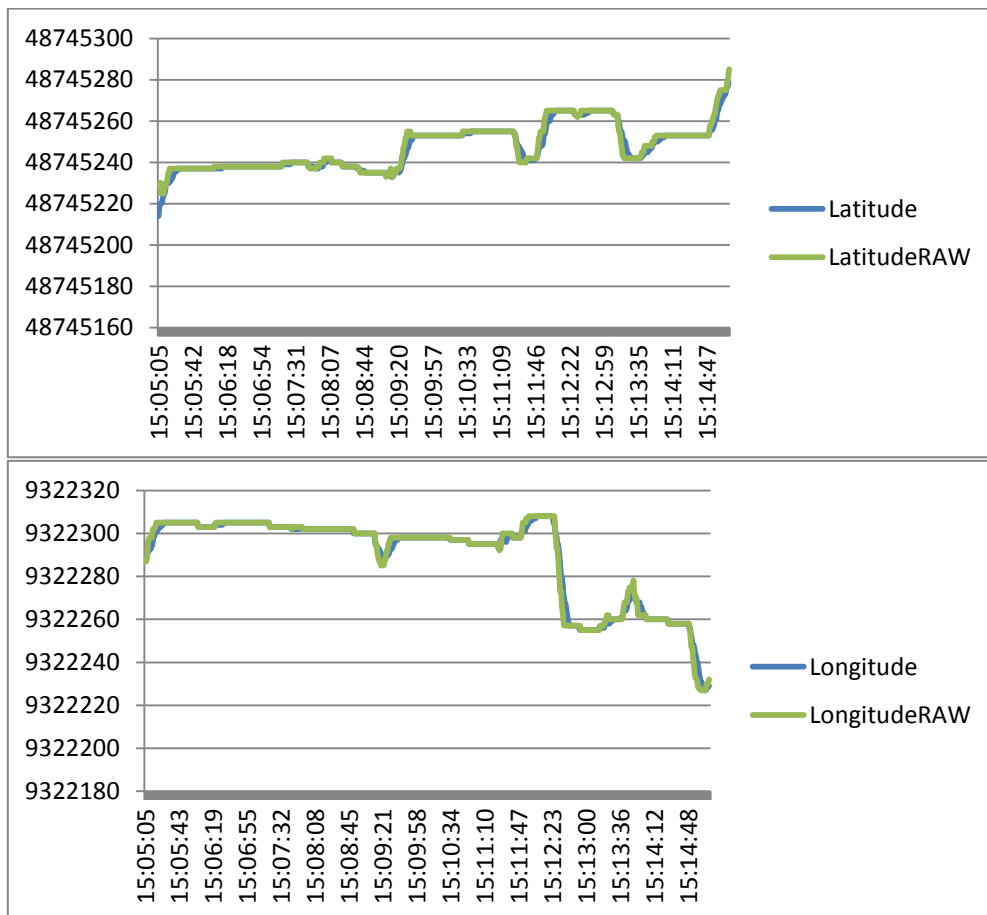


Figure 3-3 Long signal 3 (Latitude and Longitude)

Discussion

For discussion purpose, “long signal 3” is taken. The values shown in the graph are the same values received by microcontroller from GPS receiver. To get the real distance in Stuttgart region in meter, the latitude needed to be multiplied by 0.111307 and the longitude by 0.074479. For more details into getting these factors, see [6]. The LatitudeRAW and LongitudeRAW are the raw signal from sensor while Latitude and Longitude are filtered signal.

As shown in the two graphs, the signals are turbulent, showing a resultant deviation of 9m, even though the quadrocopter did not move. This behaviour could be explained by two main reasons. The first reason is the satellite. The jumps in the signals could be the consequence of changing in visible satellites. As mentioned in chapter 3.1.1, at least four satellites are needed to pinpoint the location of GPS receiver. The resultant value changes, when one of the satellites moves out of the range. Second reason is the general tolerance of GPS system which is in range of 3m. Even the best GPS receiver cannot have a better accuracy than that without further filtering.

As first attempt to improve the signal, a 10-states-moving-average is integrated, which is shown as blue line in Figure 3-3. Since the signal is taken every second, therefore this moving average is basically 10s-moving average. It does not help significantly to smooth out the curve. Increasing the number of states will reduce the responsiveness of sensor and will consume substantial amount of resources. A single longitude or latitude or altitude is 16 Bytes. So a 200-states-moving-average of all three parameters will take up 9600 Bytes, which is quite heavy for the microcontroller with limited resources.

Another suggestion to improve the signal is using Kalman’s filter. By fusing GPS signal together with acceleration signal, a better prediction of position could be achieved. But due to the complexity of the task and limitation of time, this filter is not implemented in this project. But this will be a good start for next group. For this purpose, a Simulink model for GPS sensor (see Chapter 3.1.4) is created so that the next group could work on further signal conditioning.

3.1.4. Simulink – Model

To study the noise of GPS signal, a Simulink model is created using the real signal from GPS receiver. With this model, future group could take the noise into consideration in future work on Kalman’s filter and further improvement of controller.

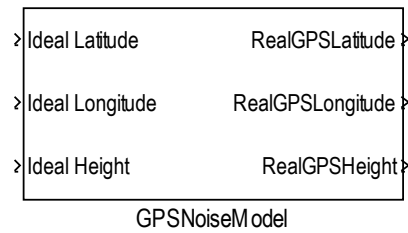


Figure 3-4 GPS Noise Model

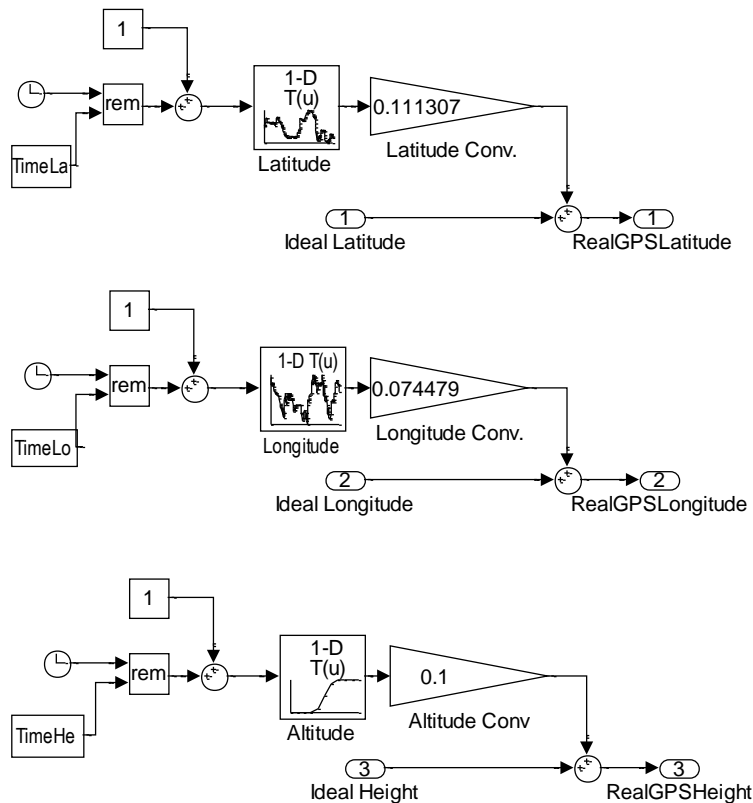


Figure 3-5 GPS Noise Model (Detailed)

The model is basically consist of a lookup table, which stores the deviations of signal to the average of the signals, a clock and a remainder block, which will repeat the signal if simulation time is longer than the signal period. By adding the deviation on the ideal parameter, a noise due to GPS sensor is created.

The following figure shows the result of noise created using "Long Signal 1" and "Height 1". For further instructions to include new signal into the model, please refer to Chapter 6.3.3.

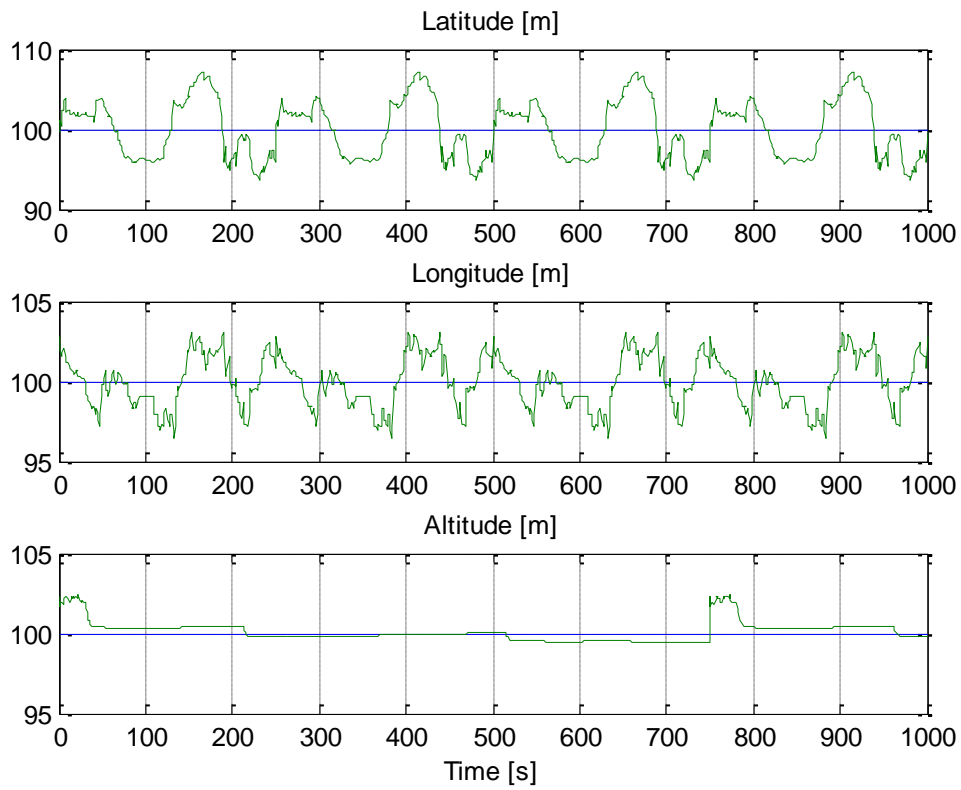


Figure 3-6 Results of GPS noise Model

3.1.5. Conclusion

A reliable GPS sensor is crucial for a flying drone like the quadrocopter. Any autonomous flying function is only feasible when the microcontroller is able to know its position accurately. Therefore, future group should continue to work on further signal conditioning for this sensor.

3.2. Pressure Sensor

3.2.1. Description

When cruising from one point to another, the quadrocopter should be able to navigate horizontally regardless of landscape. In order to do so, the barometric height should be known. Using a pressure sensor, the barometric height could be calculated using Equation 3-1.

$$\rho_h = \rho_0 \cdot e^{-\frac{mgh}{kT}}$$

Equation 3-1 Barometric Formula

On the quadrocopter, a pressure sensor, MPXAZ4115A from Motorola, is integrated. It gives reading in range of 15kPa to 115kPa by giving a voltage in range of 0.2V...4.8V. Through the 10-Bit AD-Converter of the microcontroller, the reading is then converted into value in range of 0...1024. For more details, see [7].

3.2.2. Problems

The sensitivity of the sensor is not suitable for the task, because the sensor cannot detect a height difference of 2m effectively. According to Equation 3-1, a difference of 1m gives a pressure difference of 0.01kPa. Therefore the change of pressure in this height difference is marginally. With the 10-Bit AD-Converter on microcontroller, the minimum detectable height difference is 5m, without consideration of converter's tolerance.

Another pressure sensor with working range in 90-110kPa is not a good solution either. The atmosphere's pressure is changing everyday significantly, depending on the weather condition.

3.2.3. Suggestions and Conclusion

Altitude can be extracted using GPS receiver and its driver has been modified for this purpose. But this signal, just like latitude and longitude, varies depending on numbers of visible satellites. Therefore further signal conditioning is required.

3.3. Ultrasonic Sensor

3.3.1. Description

The SRF 10 ultrasonic sensor is used to measure the height. It is small enough with a length of 32mm, width of 15mm and height of 10mm to fit under the quadrocopter.

Ultrasonic sensor measures a distance by using the principle of echo. The emitter emits an ultrasonic wave and the receiver will detect its echo. The time difference between emitting time and receiving time implies the distance to the first object in the direction of its wave propagation. Therefore, this sensor is prone to multiple reflections of wave due to obstacles or non-even surfaces. For more info, see [4] and [8].



Figure 3-7 Ultrasonic Sensor

The sensor has a maximum range of 6m and a minimum range of 0.04m, which is well within the requirement, and the resolution is 1cm. The sensor is connected to the I²C bus using four simple cables. This bus is present on the quadrocopter and it is already used by the motor controllers. There is no additional action or component needed to connect the sensor to the hardware. For more details of SRF 10 ultrasonic sensor and I²C bus, please refer to [8].

3.3.2. Sensor Integration

To physically integrate the sensor, the pins on the sensor should be connected to the corresponding pins on the board by four cables. This task was done with the help of Mr Beltz.

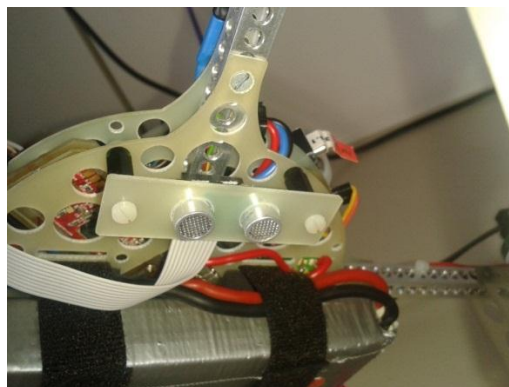


Figure 3-8 Mounting of Ultrasonic Sensor

To use the sensor, a set of codes were added in the software of quadrocopter which can configure the range and the gain of sensor and make the sensor able to measure in a certain frequency.

For more details about how to integrate, add codes and configure the sensor, please refer to chapter 6.4.

Caution: The reading rate of the sensor should not exceed 5Hz, or the bus on the board will get stuck. Failure in receiving signal from remote control was detected during this project.

3.3.3. Signal Conditioning

In order to control the height of the quadrocopter, a reliable height signal from the sensor should be guaranteed. Hence, several tests were done to check the maximum rolling and pitching angle of the sensor. The quadrocopter was held in the air and then rotated from 90° to -90° in rolling movement and pitching movement respectively. Both movements were done at two different heights.

Remark: All the tests were taken with gain = 0x08 and range = 0x3D. Only the important parts of the result are shown here. For more results, see Attachment 0.

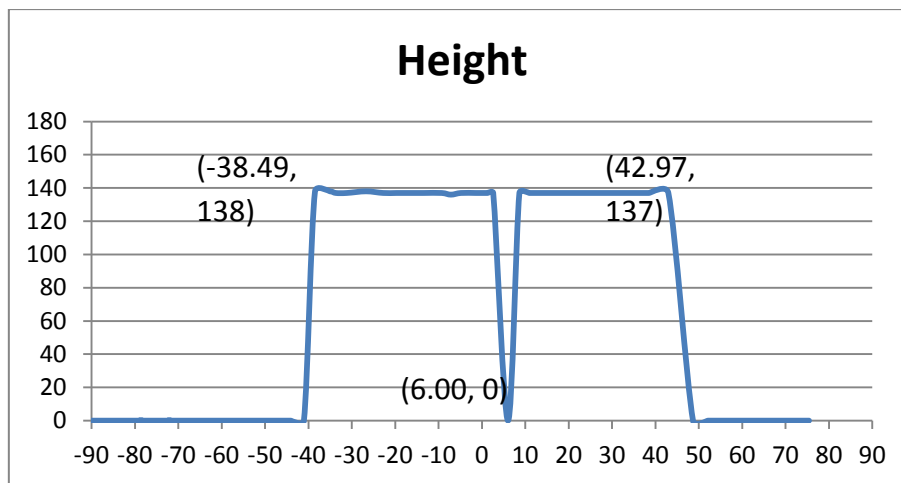


Figure 3-9 Pitching movement (higher height, 1st test)

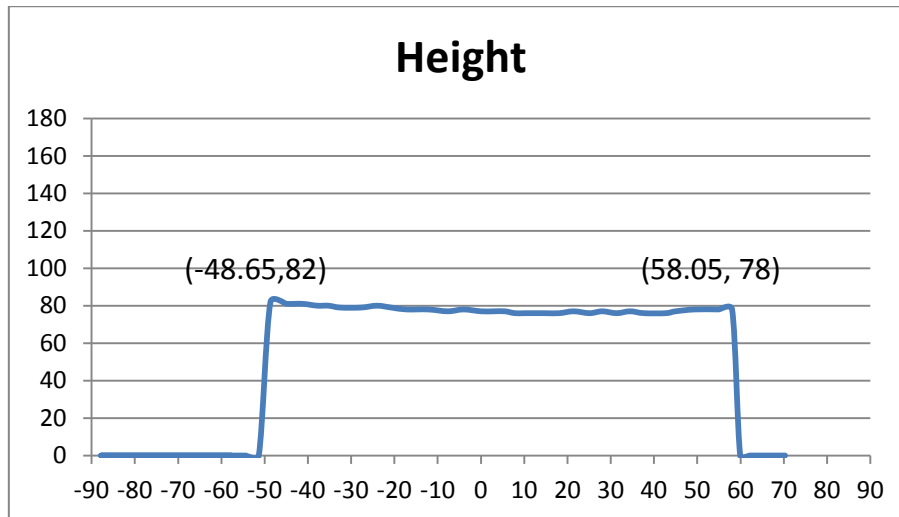


Figure 3-10 Pitching movement (lower height, 1st test)

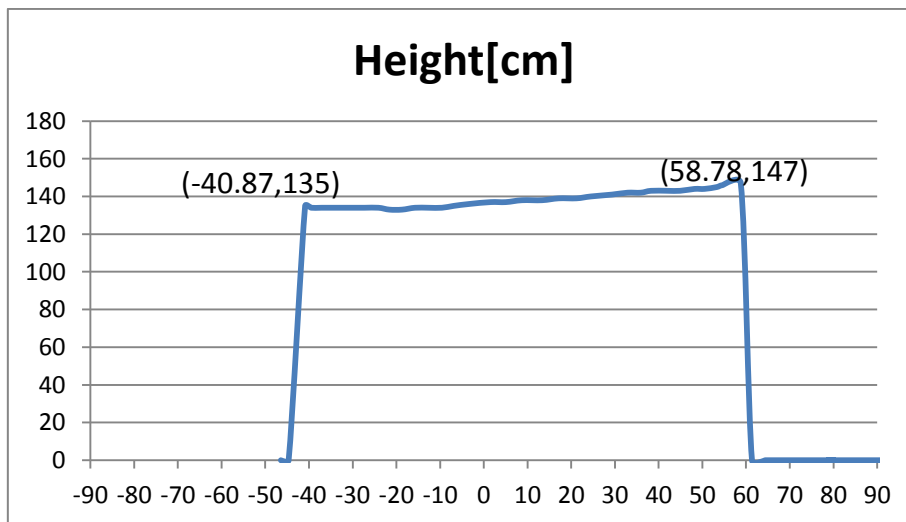


Figure 3-11 Rolling movement (higher height, 1st test)

Movement	Valid angle	Min. angle	Max. angle
Pitching(h≈140cm)		-32.52°	42.97°
Pitching(h≈80cm)		-48.65°	56.49°
Rolling(h≈140cm)		-40.87°	58.78°
Rolling(h≈80cm)		-21.22°	39.99°

Table 3-3 Signal Analysis

3.3.4. Discussion and Conclusion

During each test, the height could be regarded as constant at certain value. For pitching, the working range is quite symmetric on both direction, but an asymmetric pattern is detected for rolling. This is due to the mounting position of the sensor, which is not at the middle of quadrocopter.

A big error can be noticed in Figure 3-9 which shows that the height is 0cm around 0° . This is because there was a rubber band on the floor. Therefore it is safe to suggest that the ultrasonic reflection rate on different materials is quite different, especially the elastic materials can absorb the sound wave, i.e. no echo will be detected which results in 0 reading. This effect should be further investigated by future group.

From the data shown in Table 3-3, the confirmed valid detecting range is between -20° and $+20^\circ$ for any movement. It is acceptable because the sensor is used only in soft landing mode and soft landing mode comes only after hovering mode, so there will not be a large variation in angle.

As conclusion, this ultrasonic sensor is suitable for the purpose of soft-landing, which will only be called when the altitude of quadrocopter is within 2m to ground. This statement is only true provided the landing surface material is similar to those in experiments.

3.4. Remote Control & Switches

3.4.1. Description

In order to call different functions by turning on/off switches from the Remote Controller, the state of switches should be known by the quadrocopter. Here Mx-16 HoTT remote controller is used to control the quadrocopter.

The remote controller can transmit data in 8 channels in high frequency band (35...40MHz). Each channel is responsible for an assigned switch or joystick. For more details about the remote controller, please refer to the handbook [9].

3.4.2. Reading states of switches

As required by the flow chart logic (see chapter 4.1), different switches of remote controller are used to trigger different modes. For this purpose, some modifications to remote controller and function in microcontroller had been done:

Remote Controller: Assign switches to channels

Four switches are assigned to different channels in remote controller to call different modes of quadrocopter. For how to assign switches to channels, please refer to chapter 6.5.



Figure 3-12 Remote Controller

Source Code: Getting Switch State in copter

To update the microcontroller of the latest state of switches, a new function “ReadSwitch” is created in QH_remote.c and it is called in main.c at the beginning of loop to determine which mode the microcontroller should use.

QH_remote.c

The “ReadSwitch” function is added to allow the quadrocopter to get the signal value from channel 5 to 8 in remote controller which is saved in signalbuffer.

```
uint8 ReadSwitch(int SwitchID){
    uint8 signal;
    signal=readPtr[SwitchID+3];
    switch(SwitchID){
    case 1: return (signal==255? 1:0);break;
    case 2: return (signal==255? 1:0);break;
    case 3: return (signal==255? 1:0);break;
    case 4: return (signal==255? 2:(signal==128? 1:0));break;
    }
}
```

copter.h

This file contains Struct type describing the quadrocopter’s actual state. The states of 4 switches are added to struct “CopterState”.

```
/* Switch State*/
uint8 switch1;
uint8 switch2;
uint8 switch3;
uint8 switch4;
```

main.c

The function is called every 20ms to get the state of the switches and save the state value in CopterState.

```
if (timerIsFlagSet(TIMER_FLAG_20MS))
{
    copterGetStatePtr()->switch1=ReadSwitch(1);
    copterGetStatePtr()->switch2=ReadSwitch(2);
    copterGetStatePtr()->switch3=ReadSwitch(3);
    copterGetStatePtr()->switch4=ReadSwitch(4);
}
```


4. Software

4.1. Software Architecture

The two major functionalities that the quadrocopter is intended to perform are:

1. Hovering and Soft-Landing
2. Position Recording and Retaining Recorded Position

Hovering refers to the stationary position of the quadrocopter in space at a constant height from the ground level. Here the speeds in x and y directions are controlled to zero. In Soft-Landing, the quadrocopter should first become stable i.e. hover and then land by gradually decreasing the height of the quadrocopter from the ground level. The second functionality, Retaining Recorded Position refers to the quadrocopter returning to a pre-recorded geographical location.

In order to implement the above functionalities, the quadrocopter is designed to operate in different modes.

1. Manual Mode
2. Automatic Mode
 - a. Hovering Mode
 - b. Soft-Landing Mode
 - c. Record and Retain Recorded Position Mode

In Manual Mode of operation, the quadrocopter is completely controlled by the operator. And the Automatic Mode is realized using 4 switches in the remote control. Please refer to the Figure 4-1.

1. Switch #3 (SW3): Hovering
2. Switch #2 (SW2): Soft-Landing
3. Switch #1 (SW1): Recording current position
4. Switch #4 (SW4): Retain recorded position

Upon switching ON Switch #3, the quadrocopter hovers at the current position. Switch #2 corresponds to Soft-Landing Mode. If Switch #1 is ON, then the current position of the quadrocopter is recorded. With Switch #4 the quadrocopter returns back to the recorded position.

4.1.1. Mode Switching Logic Design:

Figure 4-1 shows the Flow-Chart for the mode switching logic.

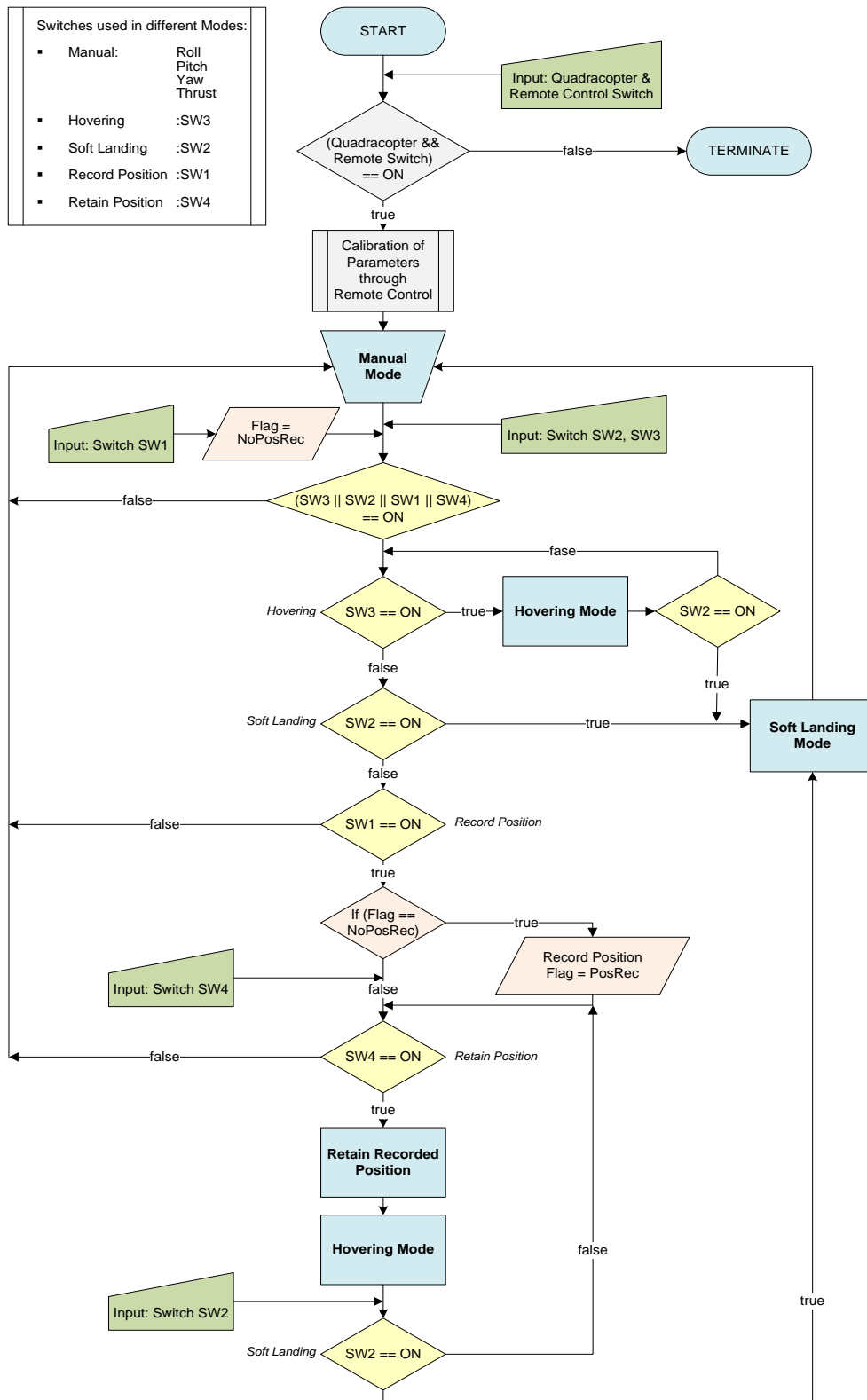


Figure 4-1 Flow-Chart for Mode Switching Logic

After the quadrocopter and the Remote Control are switched ON by turning the respective power switches ON, the quadrocopter remains in the Manual Mode. The operator can fly the quadrocopter and control it. The quadrocopter operates in this mode until one of the 4 switches is turned ON. Firstly, the control logic checks if the Hovering switch is ON. If so, it goes to Hovering Mode and the quadrocopter keeps hovering until this switch is switched OFF or Soft-Landing switch is turned ON in which case the quadrocopter performs soft-landing. When Record-Position switch is turned ON, the current geographical position of the quadrocopter is recorded and saved in memory. A Record Position flag is used to make sure that the position of the quadrocopter is recorded only once after the switch is turned ON. Any time after the position is recorded, when the Retain-Position switch is turned ON, the quadrocopter goes back to the recorded geographic location and keeps hovering. Now the quadrocopter goes back to Manual Mode if Retain-Position switch is turned OFF or does Soft-Landing if Soft-Landing switch is turned ON. Dummy C-Code is written in *main.c* file for this switching logic. The code for core functionalities needs to be yet developed.

4.1.2. Limitations and Drawbacks of the design:

1. Safety concerns: Safety measures should be considered when switching from one mode to another. The transitions between different modes should be safe and smooth.
2. Timing concerns: The system must be designed keeping the measure rates of the sensors and switch value change rates in mind. For example the measure rates of GPS and Ultrasonic sensors are approximately 200ms. So reading the sensor values more frequently than 200ms does not make sense.
3. Soft-Landing task is currently designed as uninterruptable.
4. Scenario where multiple switches are turned ON simultaneously should be considered.
5. Functionality to be robustly implemented.

4.2. Implementations in Matlab/Simulink

4.2.1. Introduction

Before the functions are implemented on quadrocopter, it is wiser to test them in simulation. A complete quadrocopter model with state space controller has been developed by previous group (see [1], [2]) which served as the basis for the simulation of the new functions. Figure 4-2 shows the complete Simulink model.

- Red blocks: To input the destination's coordination for auto cruising mode.
- Green block: The controller block that includes the hovering, height, and auto cruising control.
- Yellow block: The set point for soft landing.
- Blue block: Manual controller (Remote controller).
- White blocks: State space controller, quadrocopter block and sensors block.
- Switch 1: Switching between hovering and auto cruising/manual control.
- Switch 2: Switching between hovering and landing.
- Switch 4: Switching between manual control and auto cruising.

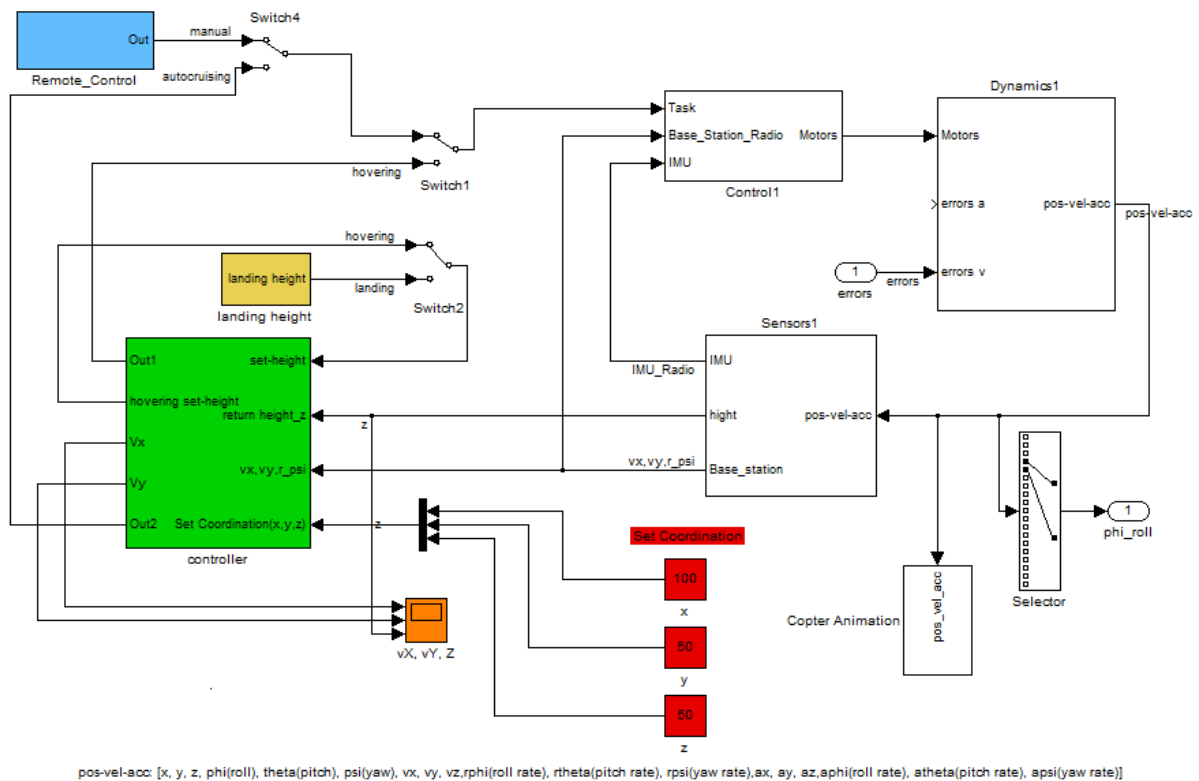


Figure 4-2 State space control for copter flight control with hovering, height and auto cruising function

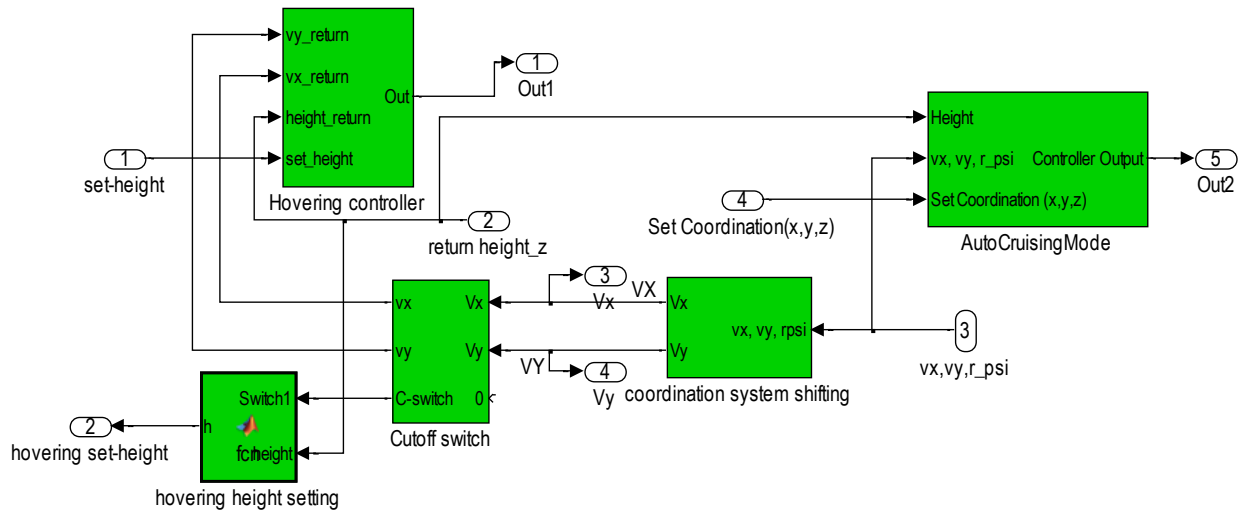


Figure 4-3 Controller Block

4.2.2. Hovering

Before starting with any flying function, it is important that the quadrocopter is able to stay afloat in the air stably. Therefore, the first function discussed here is the hovering mode.

Hovering is the state that the copter should be just hanging on the air, i.e. no speed and no acceleration in x, y, z directions and no pitching and no rolling. Yawing is not relevant in this case. This state is crucial for a stable soft-landing, as the ultrasonic sensor is most effective when the quadrocopter is aligned with z-direction.

Hovering could not be achieved by using only the state space controller. If the quadrocopter has movement before switching to hovering mode and even though the rolling, pitching and yawing angles are controlled to be zero, the copter will still be drifting due to the inertia. This is because of that air friction is not considered in the simulation model.

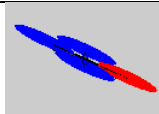


If, $V_x = \mathbf{V0}$, $V_y=V_z=0$	Set, $\theta=\psi=\phi=0^\circ$	$V_x = \mathbf{V0}$, $V_y=V_z=0$
		
Flying in X direction		Drifting X direction

Figure 4-4 Drifting due to inertia

Therefore, in order to achieve hovering, we need to brake the quadrocopter from moving by introducing additional speed controllers in x and y directions. These controllers will produce an opposite movement to counter the inertia.

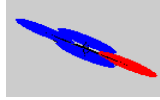
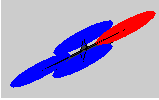
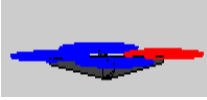
If, $V_x = \mathbf{V_0}$, $V_y=V_z=0$	$V_x=0$ (braking) and Set, $\theta=\psi=\phi=0^\circ$	$V_x = V_y=V_z=0$
		
Flying in X direction	Braking against V0	Hovering

Figure 4-5 Braking effect by adding speed controllers

Coordination system shifting

Before starting with designing the controllers, it is important to ensure that the coordination systems are matched. In the Simulink model, the feedback from “sensors” block is in global coordination system. But controlling the pitch and roll angle will produce movement in direction quadrocopter based on coordination system. Feeding the signals from different coordination systems does not make sense. In hovering control, to be controlled speeds are based on local coordination system. Therefore, as first step, the x and y signal from “sensors” block must be transformed. To simplify the transformation, movement in z-direction is ignored, which means the quadrocopter can only moving on a plane. Therefore only 2D transformation matrix is needed.

$$x = \cos \theta \cdot x_{in} + \sin \theta \cdot y_{in}$$

$$y = -\sin \theta \cdot x_{in} + \cos \theta \cdot y_{in}$$

Equation 4-1 Transformation equations

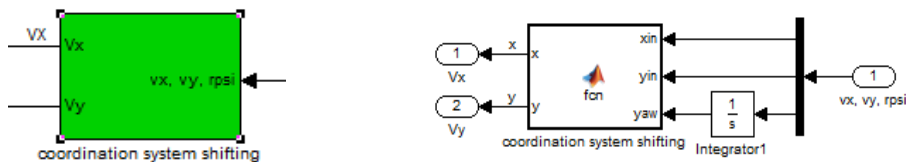


Figure 4-6 Transformation Simulink Block

PID Controller for speed in x and y direction

For hovering control, as mentioned before, speeds in x, y, and z should be controlled to zero. The position control for x and y direction is another way but it is not stable due to the lack of counter force, e.g. friction force or gravity (see chapter 4.2.4). But it is more efficient to control the height itself instead of speed in z-direction, as the ultrasonic sensor will feedback height directly.

To control the speed in x and y directions, the simple PID controller is chosen for this purpose. The controller will output the set pitch and roll angle for the state space controller.

As the inputs of state space controller are in range of 0...255, indicating $-180^\circ \dots 180^\circ$, a saturation block which limits the output to fit the range is added. A 128 is added as offset to the -180° . With set points set as zero, the quadrocopter achieves a stable state pretty well. As mentioned, yaw angle is non-relevant for hovering control, so no controller is added.

Height controller

Then a height controller is added to maintain the quadrocopter at certain height in the air and later to land in soft-landing mode. As mentioned before, it is more convenient to have a position controller than the speed controller as in x and y direction. It is easier to control the height in global coordination system, as the z-direction of quadrocopter is of no interest to simulation. Therefore no transformation is needed.

A PD-controller is used for height control. The equations of motion in z-direction are forming an I^2 -plant, which is critically stable with phase shift at -180° . In order to control this plant, a D-element is used to shift up the phase by 90° and I-element must be omitted, so that the effect of D-element will not be nullified. There was a suggestion from previous group (see [8]) to add another differentiator to get the speed in z-direction into the control loop. However this differentiator will result in numerical instability in Simulink environment, therefore a simple PD-controller is preferred. One disadvantage with this PD-controller is that there is always a steady state error. But it is remarkably small and almost constant for the control of input step difference below 200cm, which means the step between current height and set height should not be larger than 200. This is acceptable since the step for hovering control is marginal and the step for soft-landing is within this range. This could be compensated with a lookup table of predetermined offsets at different input step difference.

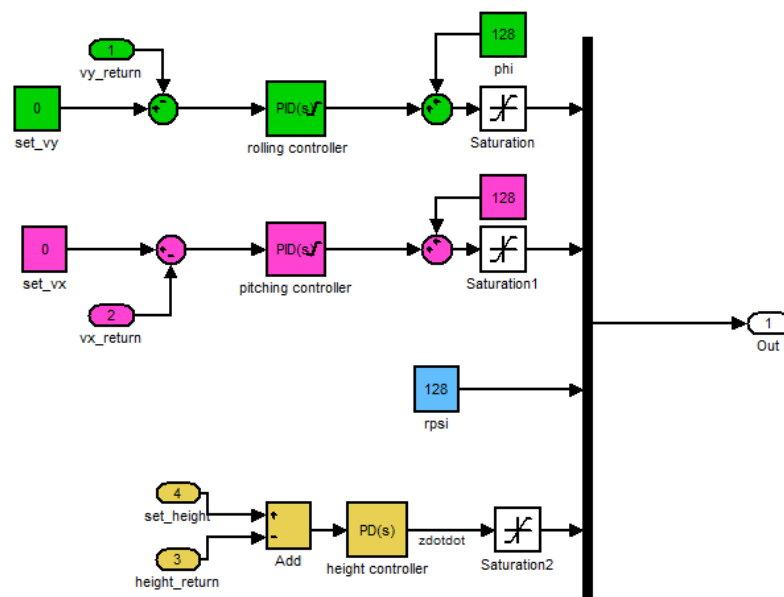


Figure 4-7 Hovering Controller

4.2.3. Soft landing

With hovering mode functions correctly, the soft-landing function is as good as done. The only difference is the set value for the height controller. Since the ultrasonic sensor of the quadrocopter is fixed on the copter body and copter's standing height is around 10cm, the set value is set to 10. As mentioned before, this set value is compensated with additional 5cm due to the steady-state error of PD-controller.

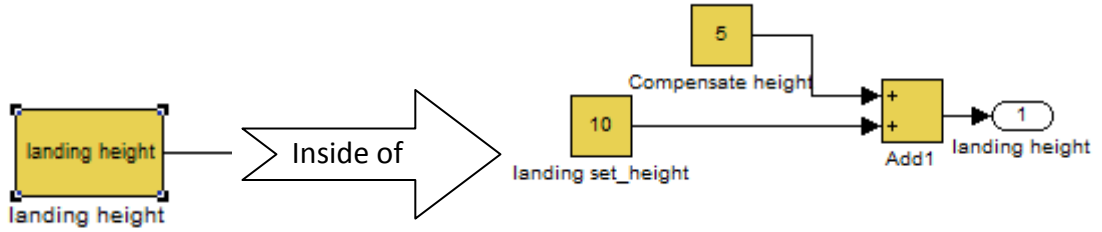


Figure 4-11 Soft-landing

4.2.4. Auto-Cruising

The second functionality as mentioned in chapter 4.1 is implemented in a slightly different function called auto-cruising. Instead of recording a position with a switch, user inputs directly the destination for the quadrocopter to fly to.

This function is done in a similar way to hovering mode. But instead of speed controller, position controller is used for x and y direction and the controlled position deviations are in global coordination system. These deviations will then be transformed to local coordination system before been input to controller.

It is the same height controller here as in hovering mode. Just like the height, x and y are I²-plant as well. Therefore, PID-controller cannot be used. PD-controller is chosen due to the same reason as explained in chapter 4.2.2. The only difference between height controller and the new position controller for x and y direction is that there is no counter force in the x and y direction to act as damping in the simulation. In z-direction, there is gravity.

$$m\ddot{z} = F - mg$$

$$m\ddot{y} = F$$

$$m\ddot{x} = F$$

Equation 4-2 Equations of Motion for x,y, and z

Quadrocopter

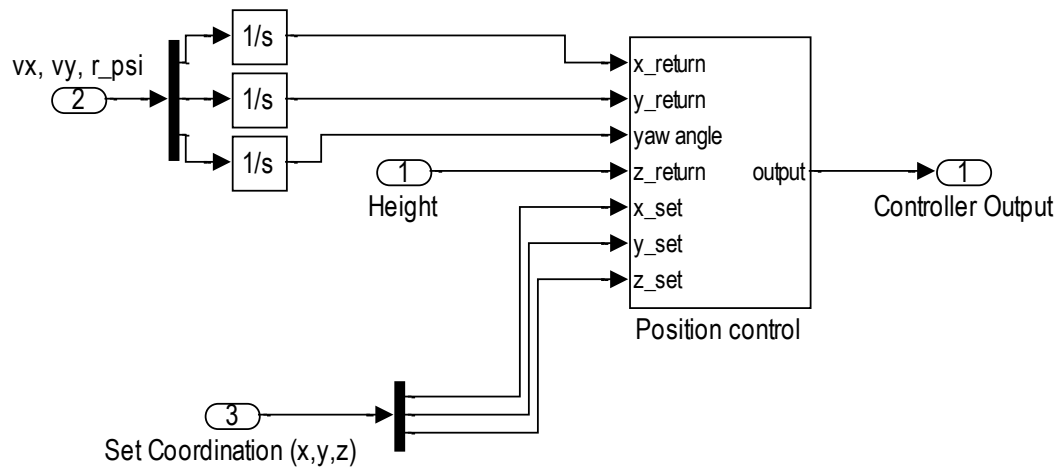


Figure 4-12 Auto Cruising Control

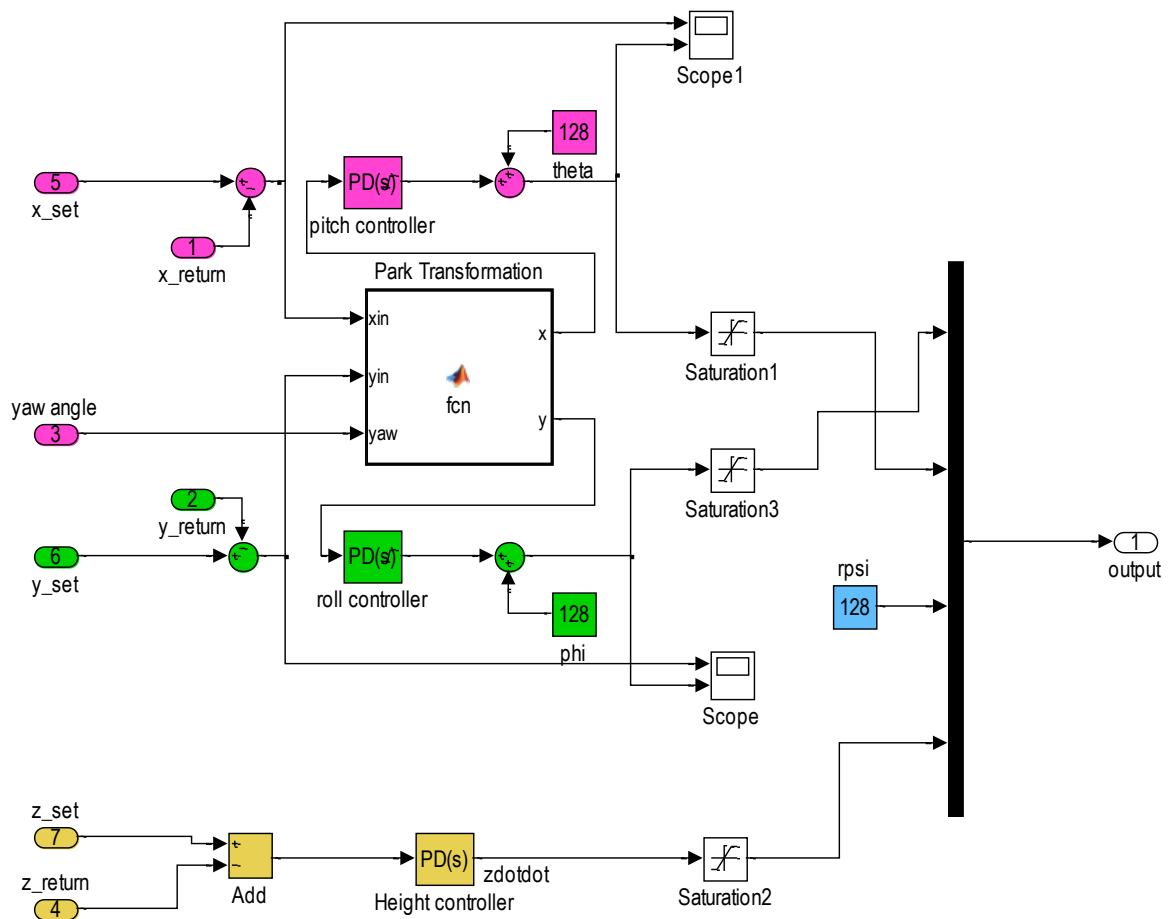


Figure 4-13 Position controller for auto cruising mode

Conclusion and Suggestions

As the result, quadrocopter will not stay static upon reaching destination as in hovering mode, but it is flying in circle around the destination for quite some time. Due to the limitation of time, further optimization of control parameters could not be performed. Another problem with this function is that it is still instable. Currently it is only stable if the quadrocopter is flied under this mode from the beginning. If it is switched from other mode, quadrocopter will go out of control. The reason for this is still unknown.

A suggestion to improve the controller would be a cascaded controller by integrating a speed controller between the position controller and quadrocopter block. Or an algorithm can be created to switch the mode to hovering after reaching the destination.

4.2.5. Data and analysis

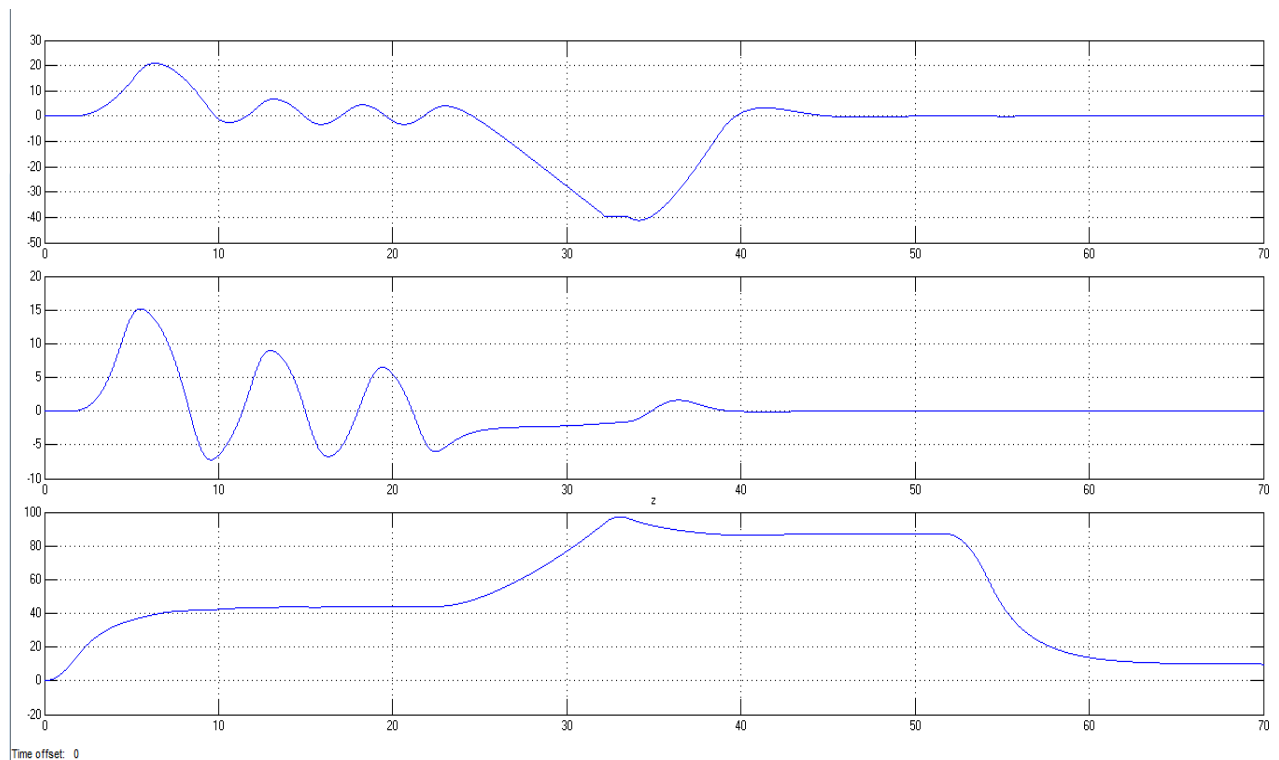


Figure 4-14 Analysis of different modes

The first graph is speed in x-direction, the second is speed in y-direction and the last one is height.

0-25s : auto cruising,

From the start, the quadrocopter flies to the set destination. As shown in the figure, it does not stay static, but it is circulating the point.

25- 32s: manual control,

At 25s, it is switched to manual control by giving a constant rolling, pitching, yawing angle and thrust.

32-52s : hovering,

At 32s, hovering is switched on. A drop in height could be observed, which is due to the steady state error. After a short while, it just stays stable and silent.

52-70s : soft landing.

At 52s, soft-landing is switched on. The quadrocopter lands without overshoot to 10cm, which is the height of quadrocopter stand.

4.2.6. Conclusion

The functions except auto-cruising are pretty stable under mentioned conditions, which are quite reasonable for the project's purpose. However these are just simulation, which has simplified the environment in a great deal, e.g. ignoring wind condition and air resistance, which cast dominant influences in reality. This is especially true for hovering function, as it is still unknown how the controller will react to disturbances like wind.

4.3. Suggestion for implementations in real quadrocopter

4.3.1. Strategy for Soft Landing

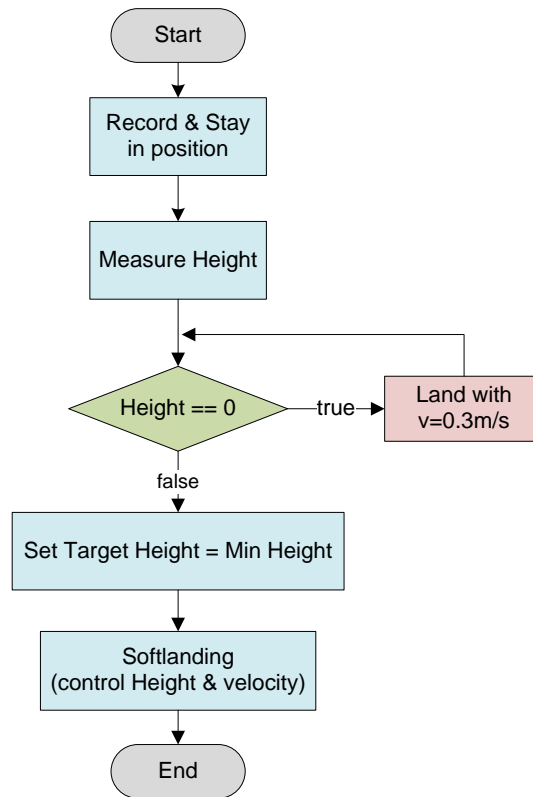


Figure 4-15 Soft-Landing Strategy

The value of height returned by the Ultrasonic sensor is always zero when the height of the Quadrocopter from the earth surface is not within the maximum measurement range of the sensor. In such a situation when the measured height by the Ultrasonic sensor is zero, the quadrocopter can be programmed to land with a small constant velocity of about 0.3 m/s. When the quadrocopter reaches the measurable range of the sensor, the sensor can measure correct values. By setting the minimum height of the quadrocopter as the target height of the PID controller, soft-landing can be achieved. With the help of a PID controller the height and velocity of the quadrocopter can be controlled and the quadrocopter can be safely landed.

5. Summary

Most of the chapters have its own conclusion and suggestions for future work. So, this chapter will conclude the project in this semester as a whole.

In this semester, despite the short time, the team managed to achieve a few main targets:

- Communication between quadrocopter and PC through xBee
- Integration of GPS and ultrasonic sensors
- Development of new control logic to adapt the new functions (in FlowChart)
- Base functions (Hovering and auto-cruising) in Simulink

These achievements should serve as good foundation for future development towards the ultimate goal of building a fully autonomous flying drone.

6. Notes of Implementation

For future groups, it is advisable to read through this chapter carefully before working on related tasks.

6.1. Environment

To install different tools needed for this project, like CodeWarrior, please refer to [10].

After installing the CodeWarrior, you can start the project source code by double clicking on “Quadrocopter_modified” under folder “Copter Software”.

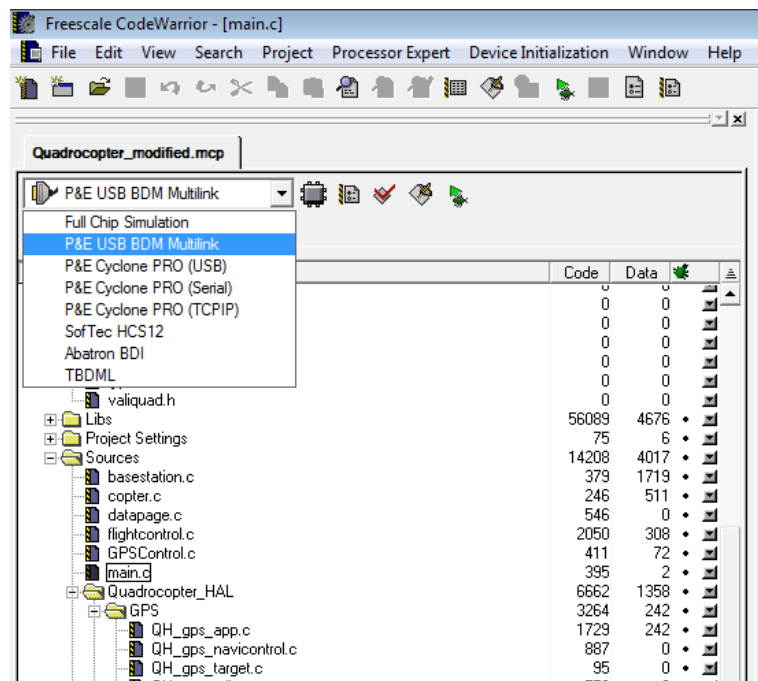


Figure 6-1 CodeWarrior

Connect the PE-BDM-Multilink module to the PC. The blue LED should light up.

Then connect the module to the first port on the right of the switch. The blue marked side of connector should be on left side. When it works, the yellow LED will light up. NOTE: the physical connection of the cable of this module is a bit loose in this semester. Consult your supervisor for the first time usage. Try to push the module around when it is not working if you are sure your connection is correct.

When the connection is established (both LEDs light up), choose the selected connection as shown in Figure 6-1. Then you can flash your source code into microcontroller by pressing “Debug” or F5.

Quadrocopter

A new debugger window will pop out. A dialog window may show an error due to the timing error. You may ignore that. If it asks to reload the application, click yes, and the whole source code will be flashed. If you end up nowhere, (after too many error messages), just close all the error messages. You can always load the application through File>>Load Application. Load the “project.abs” in “Bin” folder.

After flashing the code successfully, you can start the application by clicking “Start/Continue” or F5. Now, the programme on microcontroller is running.

You can check the state of different variable under “Data” window. To observe new variable, just find it in “Source” window and drag the variable into the “Data” window. You can load different source file by right click on “Source” window and select “Open Source file...”.

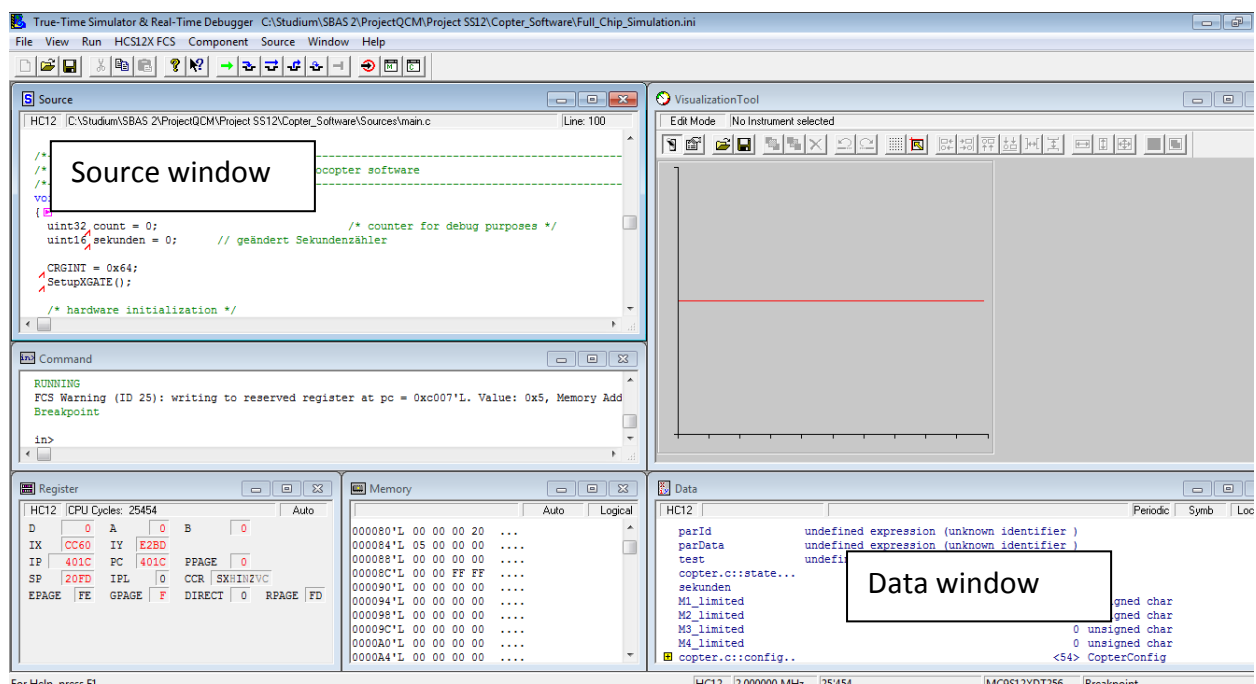


Figure 6-2 Debugger Window

Caution: Always keep a safe distance from quadrocopter. Any error in operating the debugger window will cause the quadrocopter to react differently, e.g. open the debugger window twice will cause a random initialization of parameters and drive the rotors.

6.2. XBee

6.2.1. Install X-CTU software

X-CTU is a stand-alone tool for configuring XBee modules. To install X-CTU:

1. Insert the Hardware and Software Setup CD in the PC's CD/DVD drive.
2. On the Home page, click "Gateway, Host, Enterprise Documentation/Software."
3. Click X-CTU.
4. Click Install X-CTU.

6.2.2. Install USB Drivers

1. Find the PKGU(USB) windows driver file named "usb-drivers_windows".
2. Plug in the USB cable between the PKG-U and the computers.
3. When the operating system asks for the drivers, browse to the temporary directory and complete the installation. Attention: there are two drivers needed to be installed, one is a USB driver and the other is a virtual COM port driver.

6.2.3. Configure XBee

This step is to make sure XBee is attached to PC correctly.

1. Launch the X-CTU Software.
2. Under the "PC setting" tab, find the selected com port as show. This port number is quite important as it is required to be predefined in java code before implementation.
3. Verify that the baud rate and data settings match the internal settings of the radios. The default settings for the radio are Baud Rate: 9600, Flow Control: HARDWARE, Data Bits: 8, Parity: None, and Stop Bits: 1.

6.2.4. Eclipse configuration

1. Find the file named "External file", which contains jar and dll files.
2. Place the corresponding files under those paths:
 - JAVA_HOME/lib/ext/RXTXcomm.jar
 - JAVA_HOME/bin/rxtxSerial.dll

6.2.5. Create project

1. Create new Java project in Eclipse.
2. Create new package under project, with name "valiquad.comm".
3. Import java files, which locates in file "Finalcopter"

4. Add external jar. Details: select project, click properties->Java Build Path->->Libraries->Add External JARs. There two jars needed, one is "jcommon-1.0.14.jar", the other is "jfreechart-1.0.13.jar". Both locate in the file "External file".

6.2.6. Implementation

1. Predefine the com port in class "Application" and "connectSet"
2. Power up the Quadrocopter.
3. Run the file "Application". This class is implemented to read sensor values and store data in files.

Attention: In case there is "access restriction" error, follow the next steps

1. Open project properties.
2. Select Java Build Path node.
3. Select Libraries tab.
4. Remove JRE System Library.
5. Add Library JRE System Library.

6.3. GPS-Sensor and related functions

There are three main parts in this chapter. First part is about the mounting of GPS-sensor. Second part is about using the sensor's driver and the last part is about the noise simulation model for GPS sensor.

6.3.1. GPS – Sensor mounting

To connect the pin:

1. Just connect the black socket to the pin. Black mark on the wire should be connected to the 1st pin on GPS sensor.

6.3.2. GPS – Function

In case of reintegrating the driver:

1. First copy the QH_GPS_*.c and QH_GPS_*.h files into the "Source" folder. Then include the QH_GPS_* files into the project.
2. Initialize the sensor by adding "gpsInit()" in main.c before the infinite for-loop.
3. To use the functions in the driver, include the following files:
 - a. "QH_GPS_app.h"
 - uint8 getCurrentGeogrPos(GeograficPosition * m)
 - bool getCurrentTime(GPSTime * m)
 - bool getCurrentDate(GPSDate * m)
 - b. "QH_GPS_navicontrol.h"
 - uint32 calcDistance(GeograficPosition pos1, GeograficPosition pos2);
 - c. "GPSControl.h"
 - void TestGPS(void) - This function is used for testing the GPS in this semester. It updates the current position (Latitude, Longitude, altitude and time, with and without filter) to copter states (included in copter.h) as global variables and could be used for control algorithm at any time.
4. If more data from GPS-sensor is needed, use following steps to access new data:
 - I. Check in data sheet (*Is20030-3_datasheet_v10*), in which protocol the data is included. (If it is included in RMC or GGA protocol, go to step VIII)
 - II. Then check in data sheet (*EB-230-Data-Sheet-V1.2*) page 9, Type 314, and find out the parameter to set the corresponding protocol's output frequency.
 - III. In "void setGPSOutputFreq()" under "QH_GPS_app.c", change the corresponding parameter.
 - IV. Define new GPS message type in "QH_GPS_app.h". (Use RMCmsg as template)
 - V. Add new static variable of the new GPS message type to "QH_GPS_app.c" to store the GPS data.

- VI. Define a new function, e.g. `processGLLmessage()` in `"QH_GPS_app.c"`. (use the current existing function `processRMCmessage()` as template)
 - VII. Put in this new function into `"void process_message(void)"` in `"QH_GPS_app.c"`.
 - VIII. If data is included in RMC or GGA protocol, just change the `processRMCmessage()` or `processGGAmessage()`. (Update the GPS message type, `RMCmsg` or `GGAmsg`, if necessary)
5. Tips: There are a few useful functions defined in `"QH_GPS_util.h"` which could be helpful in getting new data.

6.3.3. GPS – Simulation model (GPS Model)

To use this model in other Simulink model:

1. There are three files: `GPS.mat`, `GPS_database.m`, and `GPSModel.mdl`. All should be included in same working directory in Matlab.
2. You can copy the `GPSModel` to be used in any model, but remember to copy the `GPS.mat` and `GPS_database.m` into the same directory and run the `GPS_database.m` before you use it.
3. To add new signal into the model:
 - I. Get the GPS-signal from xBee. (you should know which reading rate the signals are recorded)
 - II. First load `GPS.mat` by double clicking on it. And delete the corresponding old data. (e.g. you are adding new latitude, then you should only delete the old latitude from workspace.)
 - III. Import the data into Matlab by right-click on the signal file and choose "Import Data...".
 - IV. Click "Next" and check only the "Data" (uncheck the other two. You can rename the variable here or afterwards in workspace). The import signal should be an array.
 - V. Rename the signal correspondingly. (either Longitude, Latitude or Height)
 - VI. Then type in `"save GPS Latitude Longitude Height"` in command window.
 - VII. Update the `data_rate_*` in `GPS_database.m` and run it.
 - VIII. Now the new signal should be shown in the lookup table in `GPSModel`.

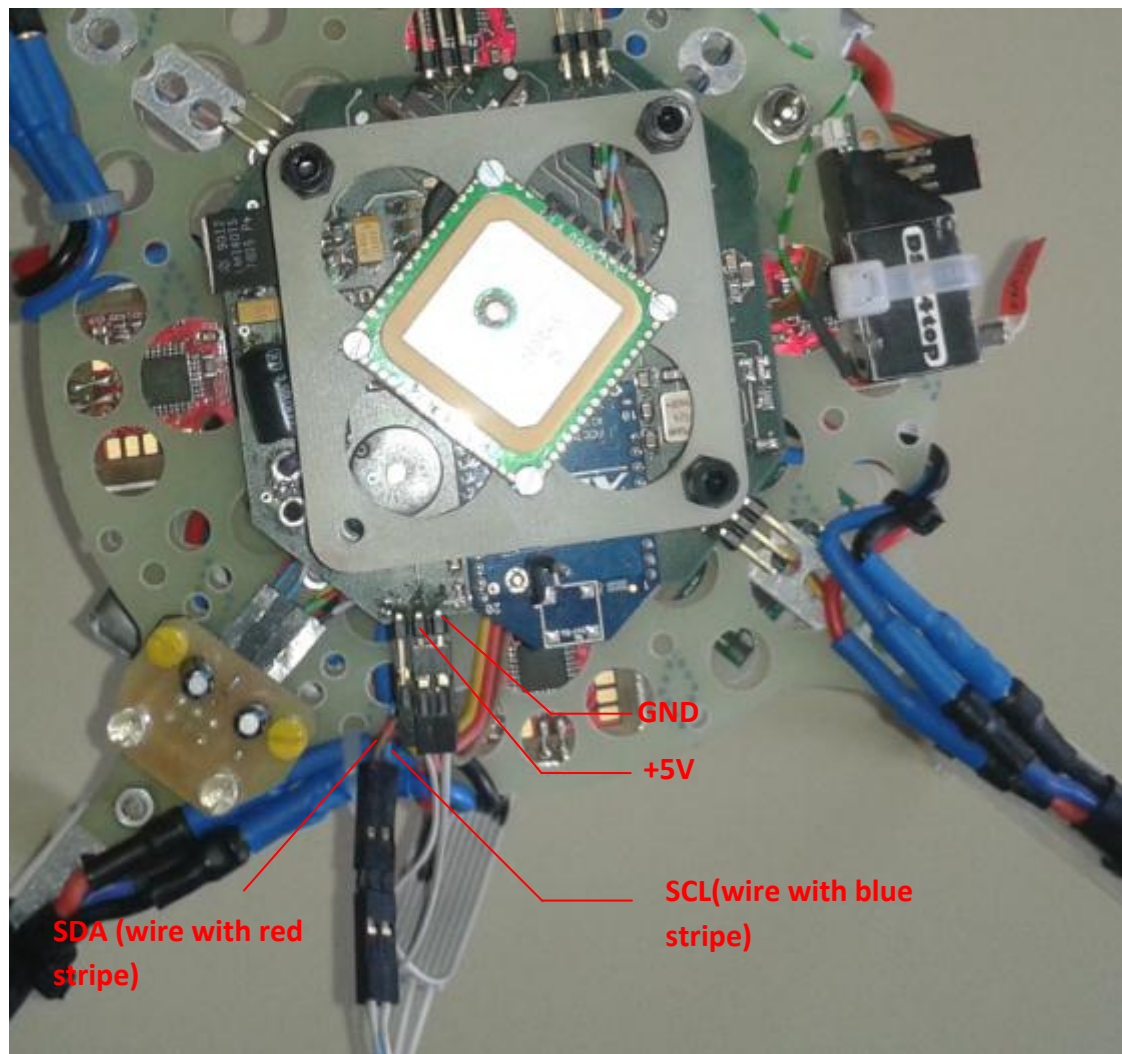


Figure 6-5 Wires connection

6.4.2. Software for height sensor

To measure a height using ultrasonic sensor:

In main.c, include the QH_heightsensor.h and the sensor should be initialized with heightsensorInit() before the for-loop.

```
#include "QH_heightsensor.h"
...
void main(void){
...
heightsensorInit();
...
for(;;){
...
}}
```

It is advisable to add the altitude as global variable in copter.h

```
/* Ultrasonic Sensor*/
uint16 altitude;
```

Then again in main.c, call function heightsensorGetHeight() every 200ms (or with other frequency) to get height and then save the measured height into the CopterState.

```
/* functions, called every 200 milliseconds */
if (timerIsFlagSet(TIMER_FLAG_200MS))
{
    copterGetStatePtr()->altitude = heightsensorGetHeight();
}
```

Caution: The reading rate of the sensor should not exceed 5Hz, or the receiver on the board will get stuck which means it will stop reading signal from the remote controller.

To configure the sensor and use other functions, see Table 6-1.

Quadrocopter

heightsensorSetRange(uint8 range)	Set the maximum range of the SRF10 sensor by adjusting the time the sensor waits for an echo. The higher the range, the longer the time needed for measuring. Maximum time is 65µs. - range: defines the maximum range in cm, theoretical maximum is 10m, max range of the sensor about 6m Return value: 0 for invalid values of range, 1 for valid values.
bool heightsensorSetGain(uint8 gain)	Set the gain for the SRF10 sensor. It is saved in the RAM of the sensor. This value must be set while initializing the sensor! - gain Return value: 0 for invalid values of gain, 1 for valid values.
uint16 heightsensorGetHeight(void)	Get the height of the Quadrocopter in cm. Return value: height in cm.
void heightsensorInit(void)	Initialize the gain and range of sensor.
void heightsensorI2CSend(uint8 slaveAddress, uint8 regName, uint8 data)	Send 1 byte to a slave on the I2C bus - slaveAddress: Address of the slave in the I2C bus, to which data shall be sent - regName: register on the slave to write to - data: data to be written
void heightsensorDelay(uint16 milisec)	Delay function - milisec: required time delay in milliseconds
uint8 heightsensorI2CRead(uint8 slaveAddress, uint8 regName)	Read byte from the i2c bus, sends no ACK, send STOP - slaveAddress: the address of the Slave from which you will read - slaveReg: the register to read from return value: value from the specified register in the specified slave
void heightsensorChangeID(uint8)	Set a new I2C ID for the SRF10 sensor. The standard value is 0xE0. For possible values see SRF10 data sheet. - newid is the new I2C ID for the sensor

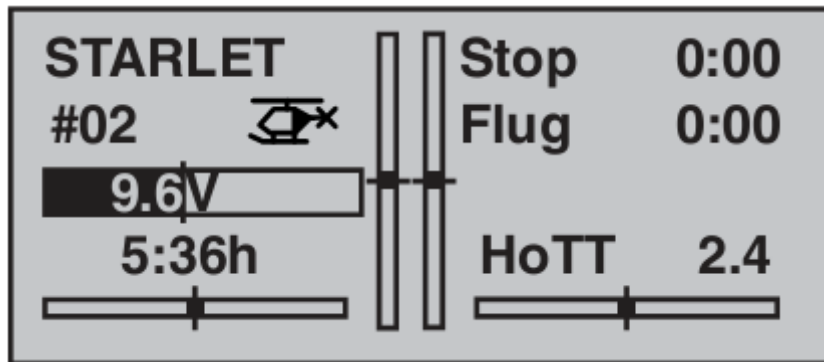
Table 6-1 Ultrasonic Sensor - Functions

6.5. Remote Controller

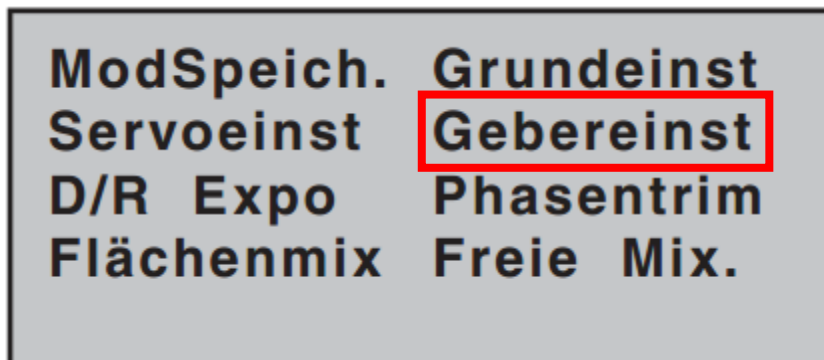
6.5.1. Set up switches to channels

Setting up the switches to channels can be simply realized by the following steps:

- Turn on the power and select “Ja”, then you can see the following main interface.



- Press “ENTER” button and move to the option “Gebereinst”.



- Enter the option, then all the available channels are shown blow.

►E5	frei	+100%	+100%
E6	frei	+100%	+100%
E7	frei	+100%	+100%
		– Weg	+
▼	SEL	SYM	ASY

Quadrocopter

- Choose a channel you want. For instance, here we want to set switch "SW3" to Channel 5. Press "SELECT" button, a prompting message appears which asks you to set the switch you want.

►E5	frei	+100%	+100%
Gewünschten Schalter oder Geber betätigen			
▼	SEL	SYM	ASY

- Turn on or off the switch to finish setting. Now the switch "SW3" is bound to channel 5.

►E5	3	+100%	+100%
E6	frei	+100%	+100%
E7	frei	+100%	+100%
- Weg +			
▼	SEL	SYM	ASY

6.5.2. Calibration of Remote Controller (Mx-16s)

The accelerometers and gyroscopes are initialized with random offsets each time after the code is flashed into the microcontroller. In order to have the sensors working properly, calibration should be done every time when the program is flashed and it should only be done when the quadrocopter is standing on even ground. For further info, see [2].

To do the calibration, turn on the quadrocopter and the remote controller, then move the left lever of the controller to the top right corner and move the right one to the lower right corner and then a beep will be heard from the Quadrocopter which indicates a successful calibration.



Figure 6-6 Calibration

6.6. Simulation

To run the Simulink model:

- All needed files are integrated in folder: >> Core >> MATLAB_State_Space_Controller >> BasicProject.
- To run these Matlab files, a C-compiler is needed on the computer. Configure it in Matlab Command Windows with command “mex –set up”. (skip this if it is already configured)
- Open Simulink file “System_Design_Quadrocopter_2012.mdl”.
- Then you can see the following picture:

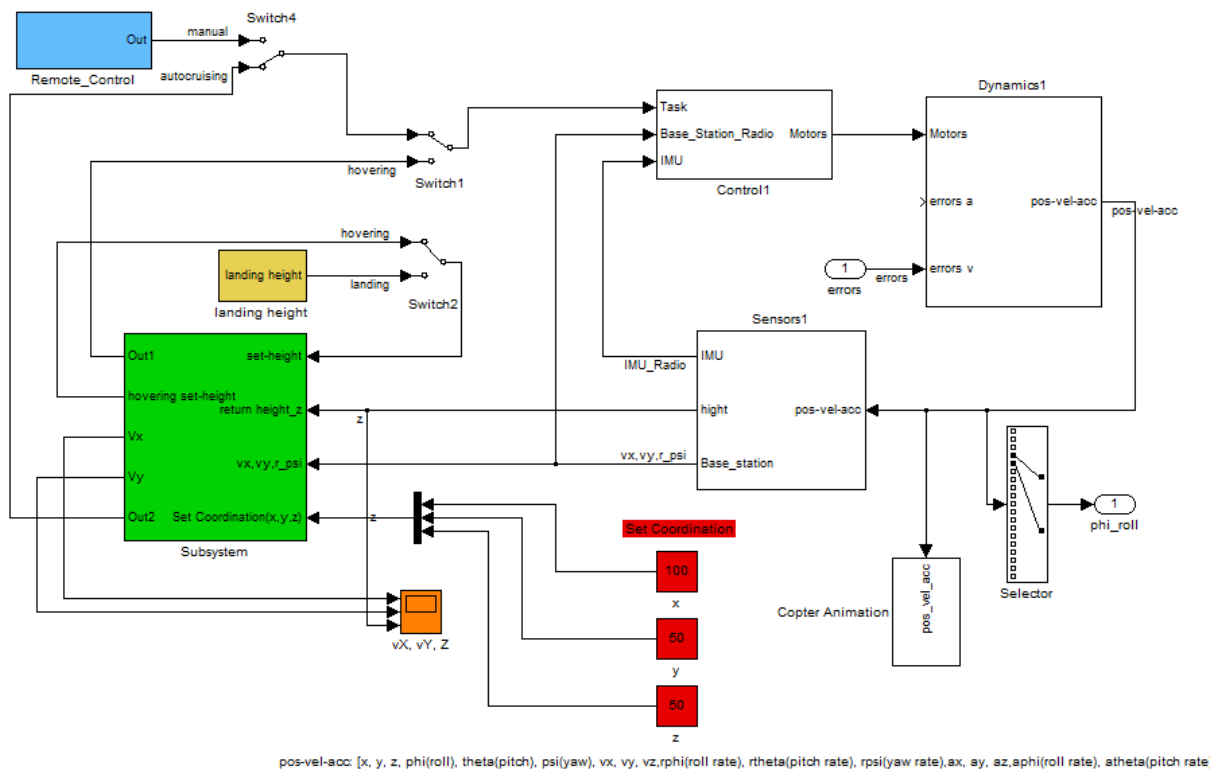


Figure 6-7 Simulation Model

Auto cruising:

- Use “set coordination” blocks (red blocks) to set destination coordination(x,y,z).
- Use switch4 to select manual control and auto cruising.
- Please note that Auto cruising should be performed in first step, because currently the auto cruising controller is not stable.

Manual controlling

- Set remote controller inputs in “Remote_Control”(light blue block).
- Set Pitching angle by modifying “theta_set”.
- Set rolling angle by modifying “phi_set”.
- Set yawing angular velocity by modifying “rpsi_set”.
- Set the thrust by modifying last block.
- Flip switch 4 to manual node to control copter with remote controller.

Hovering

- In green block, there are “Hovering controller”, “Cutoff switch”, “coordination system shifting”, “AutoCruising Mode” and “hovering height setting” blocks.
- Flip switch1 to hovering node, copter can change to hovering mode.

Landing

- Set landing height by modifying constant block “landing set_height” from “landing height” block (yellow block).
- Use “switch2” to switch into landing mode.

7. Appendix

7.1. GPS-signals

In this appendix, graphs of all recorded GPS signals are listed here for reference purpose.

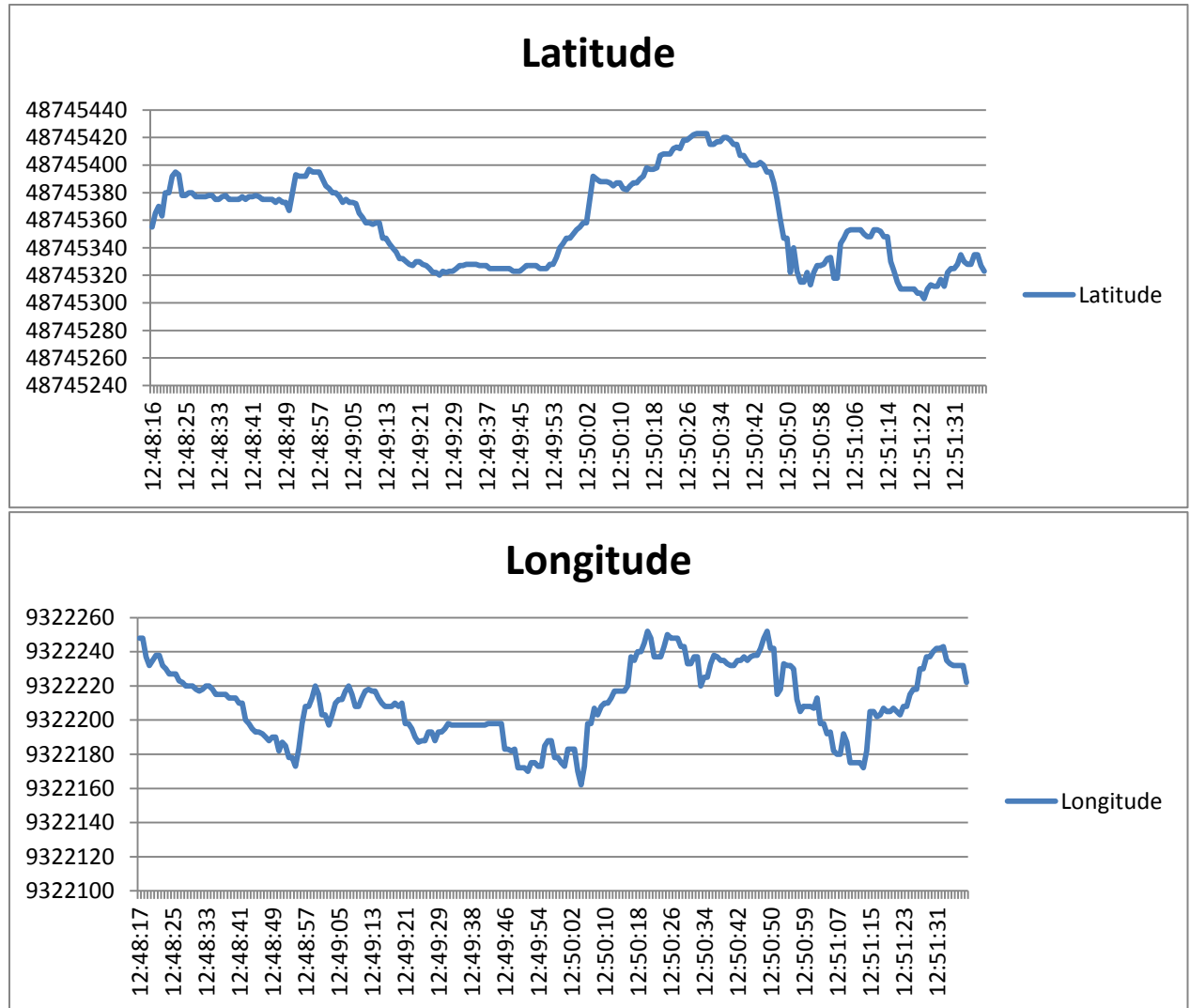


Figure 7-1 Long Signal 1

Quadrocopter

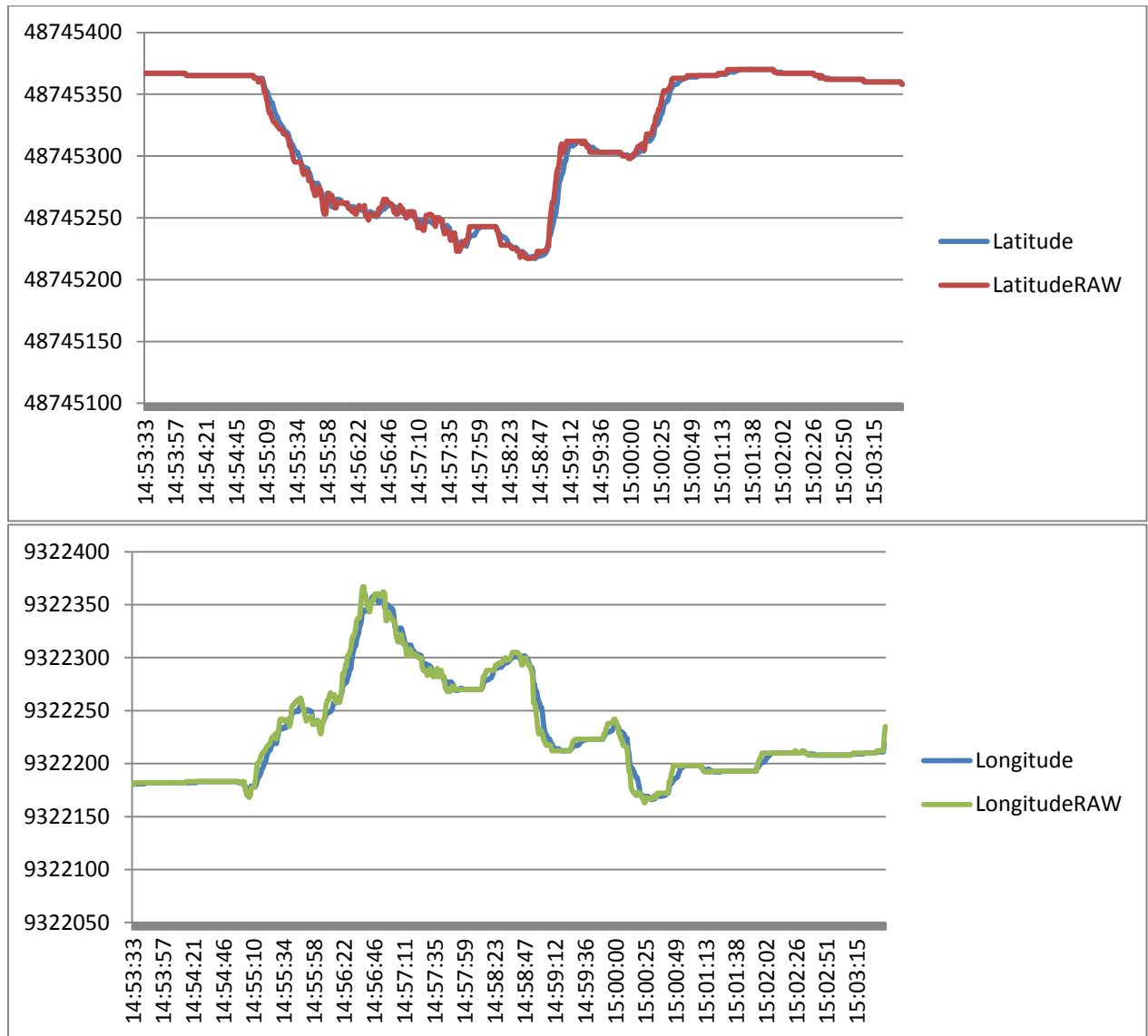


Figure 7-2 Long Signal 2

Quadrocopter

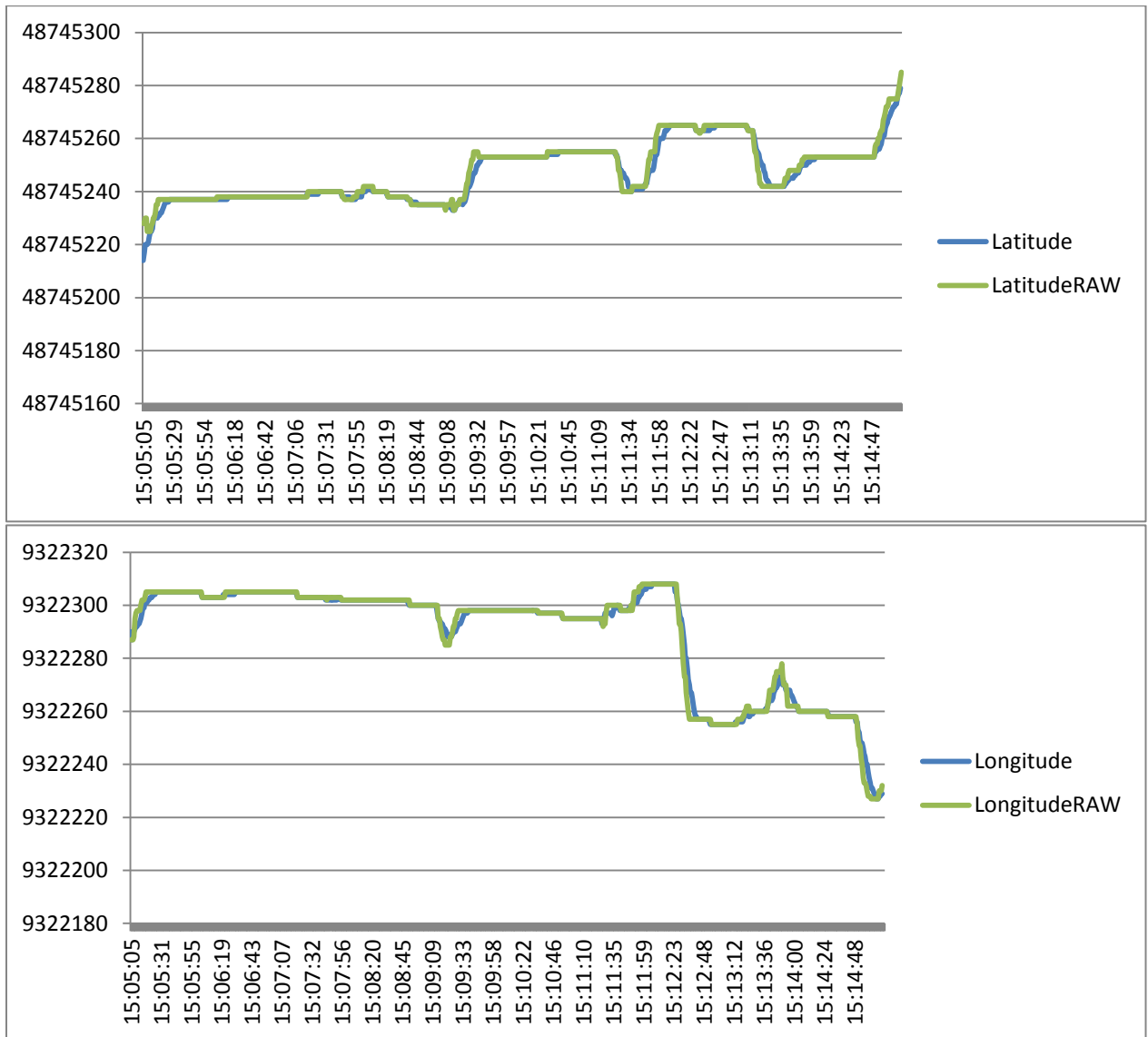


Figure 7-3 Long Signal 3

Quadrocopter

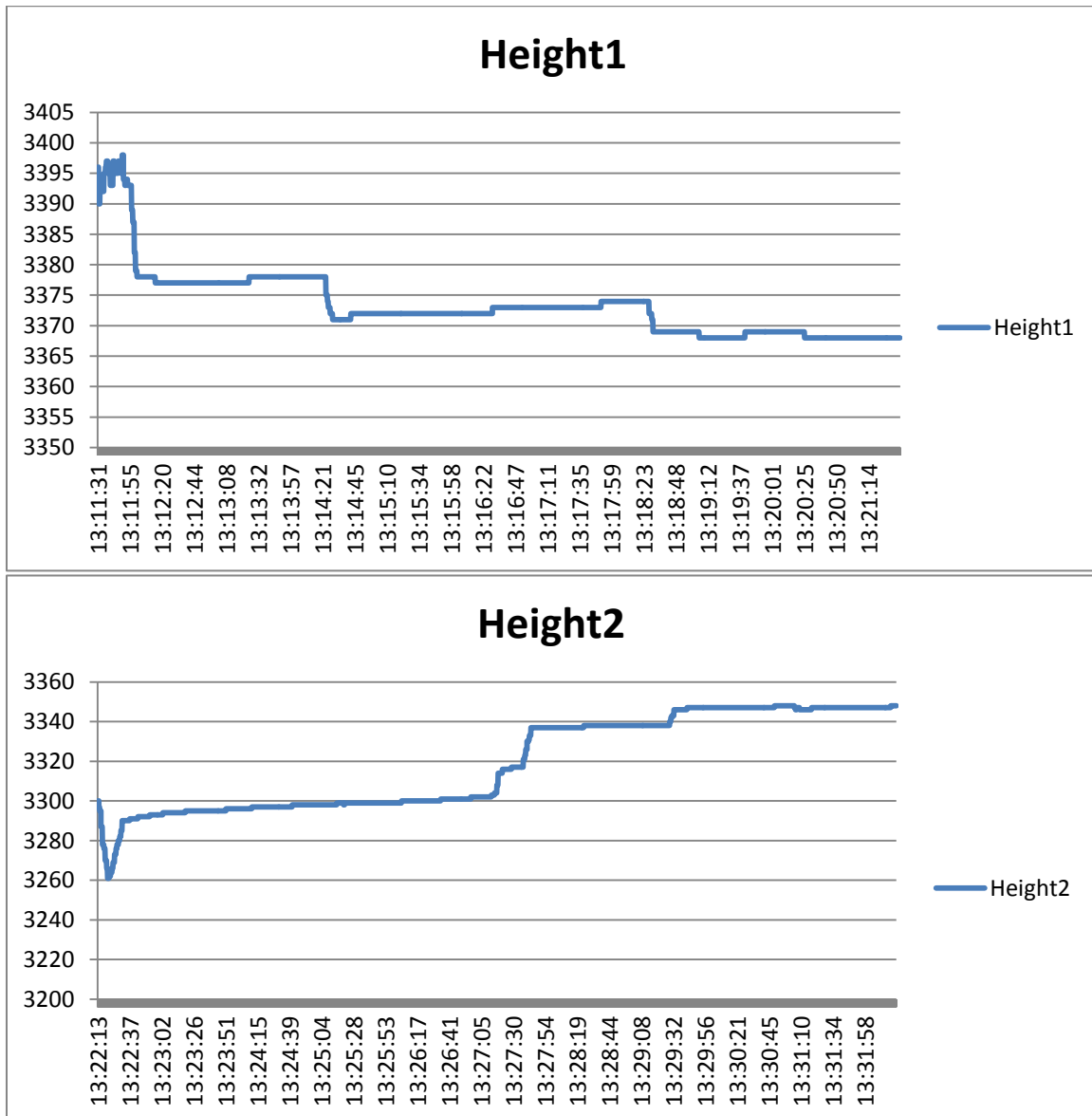


Figure 7-4 Height 1&2

7.2. Ultrasonic Sensor

Here are graphs of all experiments with ultrasonic sensor.

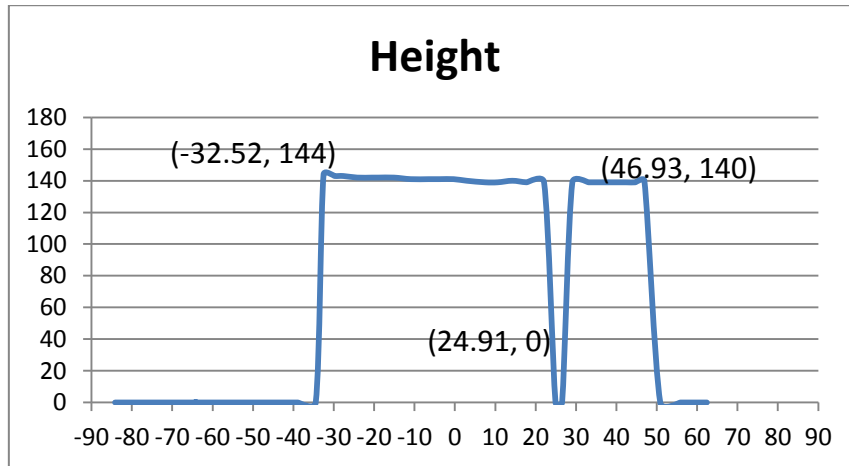


Figure 7-5 Pitching movement (higher height, 2nd test)

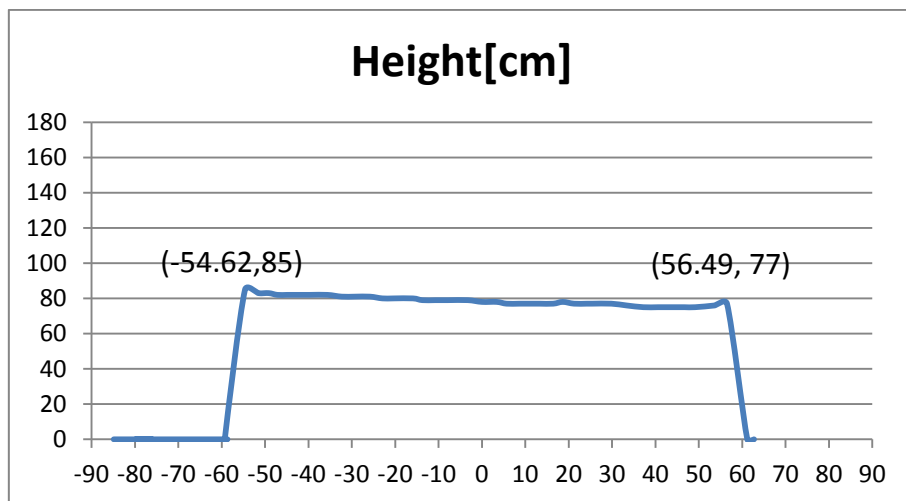


Figure 7-6 Pitching movement (lower height, 2nd test)

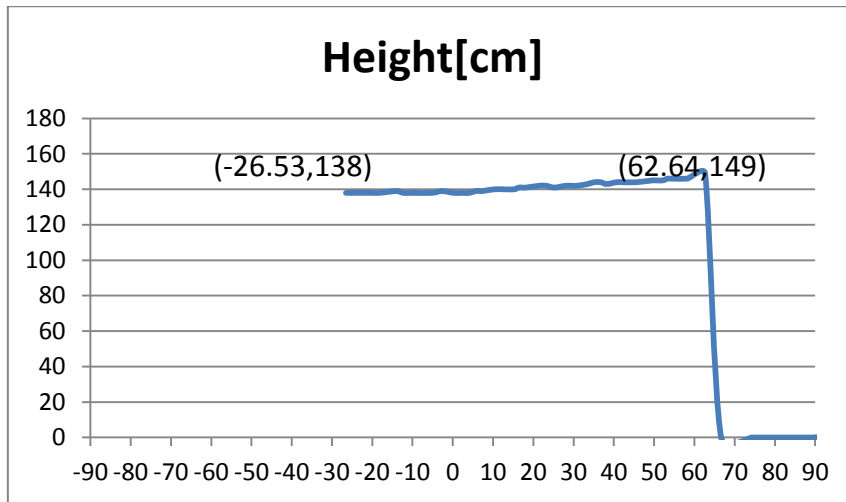


Figure 7-7 Rolling movement (higher height, 2nd test)

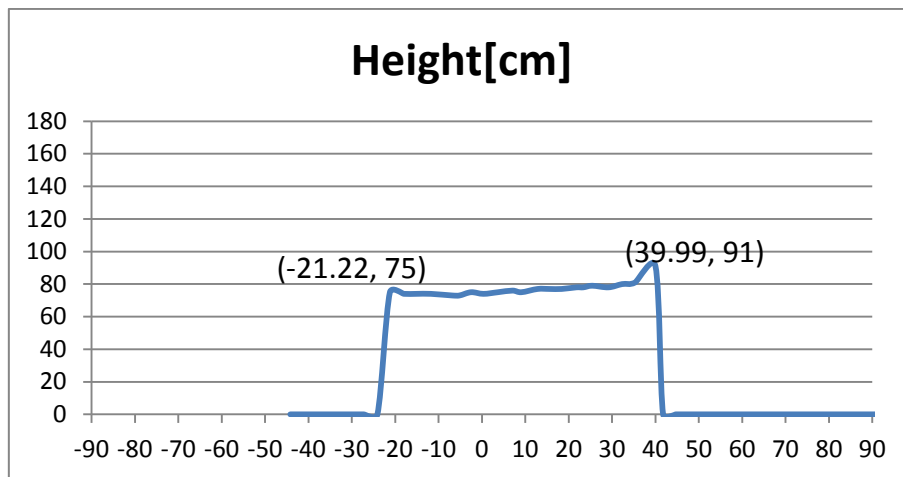


Figure 7-8 Rolling movement (lower height, 1st test)

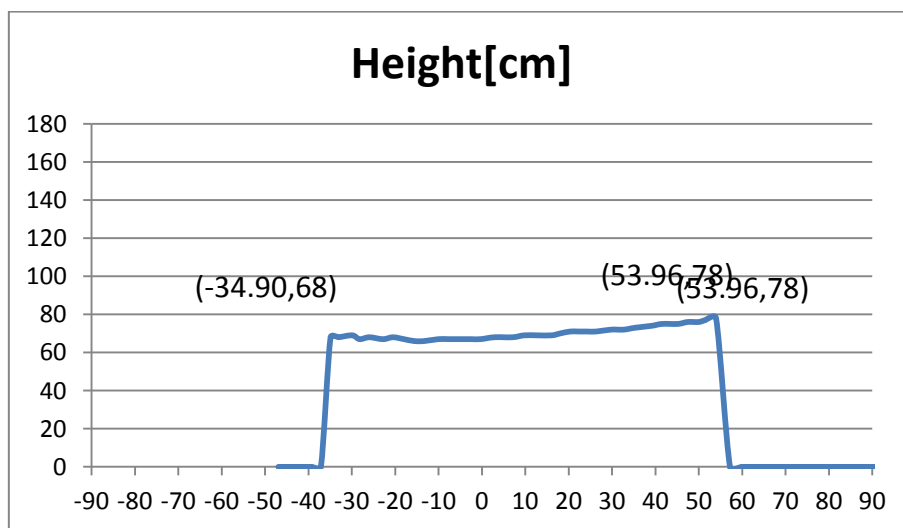


Figure 7-9 Rolling movement (lower height, 2nd test)

8. Bibliography

- [1] Alexander Stoltz, Benjamin Jaißle, "Development of a state space controller for a quadrocopter," Hochschule Esslingen, Esslingen, 2011.
- [2] M. Kapche, "Optimierung eines Zustandsreglers für einen Quadrocopter," Hochschule Esslingen, Esslingen, 2012.
- [3] Stübler, "Valiquad - The analyzing software for the Quadrocopter!," Hochschule Esslingen, Esslingen, 2010/2011.
- [4] "www.wikipedia.org," [Online].
- [5] L. T. Inc, *Datasheet of GPS smart antenna module, LS20030~3*, Taipei: LOCOSYS Technology Inc, 2006.
- [6] M. Pejs, "Anbindung eines GPS-Moduls und Implementierung eines Treibers," Hochschule Esslingen, Esslingen, 2011.
- [7] I. Freescale Semiconductor, *Data Sheet of MPXAZ4115A*, Freescale Semiconductor, Inc, 2001.
- [8] Martin Ehrle, Markus Schüler, "Analyse und Implementierung einer Höhenregelung für den Quadrocopter," Hochschule Esslingen, Esslingen, 2011.
- [9] Graupner, *Programmier-Handbuch MX-16s*.
- [10] P. D. J. Friedrich, *Introduction to the Tool Environment*, Esslingen: Hochschule Esslingen, 2012.