Stabilisation of a Pan-and-Tilt Unit holding a camera

Final Report for CS39440 Major Project

Author: Edgar Ivanov (edi@aber.ac.uk)
Supervisor: Fred Labrosse (ffl@aber.ac.uk)

May 5, 2014
Version: -999999999.045879 (Draft of the Draft of the Draft)

This report was submitted as partial fulfilment of a BSc degree in
Mobile And Wearable Computing (G421)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature .....

Date .....

Abstract

Computer Science department at Aberystwyth University has a large all-terrain rover that is equipped with a panoramic camera mounted on a Pan-and-Tilt Unit (PTU) that is stabilised using gyroscopes. One of the problems of the gyroscopes is that they tend to drift and therefore need to be regularly corrected to ensure the camera stays vertical. In order to fix this issue a small hardware module will be built and corresponding software developed. Complete system will interface tilt sensor (inclinometer) with the PTU to ensure camera "verticality". In this report root of the problem will be analysed, possible solutions and algorithms used to calculate PTU drift rate presented.

CONTENTS

Chapter 1

Introduction

1.1 Overview

The Pan-and-Tilt Unit (PTU) is a stabilised 2-axes platform mounted on the Idris electric vehicle that holds a panoramic camera. This vehicle is used in other research projects and drives in different places where it can go up and down the hills [4]. PTU has a built in functionality to stabilize at the chosen position: in case the rover drives up the hill camera will stay vertical. In order to achieve camera verticality stabilization command could only be invoked once, when the rover is standing on the flat surface(PTU would record current platform orientation and then when vehicles moves try to keep it in its original position). Its position could also be adjusted manually if vehicle is not on a flat surface and then stabilization command could be issued.

1.2 Problem analysis

To ensure stability of the camera PTU uses gyroscopes. This would be enough on its own in a perfect world. However, the well-known problem of the gyroscopes is that their readings are affected by the different air temperature, magnetic effects, jitter etc. [7]. Another problem with the gyroscopes is that they get current orientation by integrated twice rotational acceleration. This integration is error prone, and the error accumulates with time. I reality when PTU is issued with the stabilization command the platform slowly drifts on both axes and goes to the position which can be up to 20 degrees different from where it should be. This project will concentrate on fixing this particular issue as well as trying to enhance the overall system functionality.

1.3 Aim

The aim of this project is to modify system which is currently in place and implement functionality to stabilize Pan-Tilt Unit with drift compensation. The system should be fully automated and perform drift rate calculation and following calibration on its own, when the appropriate request comes in from the client.

1.4 Proposed Solution

One of the possible solutions could be the use of the additional PTU functionality. It allows to cancel drift by specifying the calculated in advance drift rate in radians per second. The problem

with this solution is that the drift rate can change if the surround environment changes (sun goes behind the clouds and air temperature drops affecting gyroscope readings). It would be impossible to calculate new drift rate manually every 10 minutes.

The proposed solution is to automate the drift rate calculation so that it can be adjusted while driving. However this approach requires additional information about the current vehicle orientation in space (its inclination angles with respect to gravity) to be able to predict required PTU position which will then be compared with the actual position and the drift calculated. Electronic inclinometer can be used for this purpose. Inclinometer response is an electric signal representing an angle between the internal axis and the gravity vector [7]. Accurate readings can only be obtained using the device if it is in a stable position and does not accelerate, otherwise it will provide erroneous data representing the sum of two vectors - earth gravity and acceleration. This imposes certain limitations. PTU drift rate calibration can only be performed when the vehicle does not move.

1.5 Objectives

The aims of this project are:

- To port current system from the Gumstix on to the Raspberry Pi.
- To implement commands for the PTU stabilization and drift rate.
- To develop drift rate calculation with following calibration.
- To develop PTU verticality functionality.
- To implement support for the new PTU functionality in the server side software.
- To extend client side API to reflect new server functionality.

1.6 Schedule and Project Management

The project will take around 15 weeks to complete. First week will be spent on the problem analysis, background research and Outline Project Specification production. In the middle of the project a poster will be produced and together with the Easter break it will take another week. Production of the final report, its review and proof reading should take no more than a week during the final stage. Twelve weeks in total will be left for the system development.

For the backup purposes and to keep track of the different code versions Version Control System will be used. I plan to use the free Github account to store all of my code and corresponding documentation. This way I will be able to restore work in a case of the system failure. VCS allows to see previous version of any file in the repository and what changes we made to it since it was added, this may help to resolve a bug by looking at what changes were introduced or it may help to restore piece of code that went missing.

Due to the tight time-frame imposed for the development of the system, an agile system development methodology will be followed. Agile development has a focus on generating working software prototypes quickly in cycles, rather than the classical approach of thorough documentation and following a strict plan. During the stage of software development there are likely to be unexpected technical difficulties that appear and need to be overcome, which can put the project behind the schedule.

The agile methodology attempts to provide the project with a clear direction by regularly assessing the development of the project at the end of each iteration and interacting with the client to ensure the project is coming closer by fulfilling its requirements. When the progress of the project is assessed regularly, it gives an opportunity to steer it back into the right direction if there is evidence suggesting that the course development is moving away from the target. The waterfall method which has a single fixed completion date at which the project finishes and should be ready to deploy runs the risk of delays if the system does not come together as planned. Thus the agile approach greatly reduces both development time and costs [9].

Following the initial evaluation of the different software development methodologies the feature-driven development methodology was chosen. At different stages of this project the work will focus on the different parts of the system. In each part there is a set of clearly identifiable features which are required for the final system to work. At the begging of the project the consultations with the client (project supervisor) were conducted to identify the overall model of the system. At the following meetings the overall model was shaped and the feature list built. The development of the system will be organised into iterations of relatively short 2 week cycles, with a set of working features produced at the end of each cycle.

Iteration 0 will focus on porting code to the new platform. It is not expected that the system will work from the very beginning. However, once minor code adjustment are done the basic features should start working.

Iteration 1 has an aim of implementing new commands in the PTU TASS library.

Iteration 2 will focus on drift rate calculation algorithm development and testing. This iteration builds on the functionality laid down in the previous cycle and incorporates the ability to calculate drift rate as well as stabilize platform with drift compensation.

Iteration 3 will depend on the work done during all previous cycles. Without successful completion of all previous steps this iteration will not be possible since at this stage all features are incorporated in to the final system. In this iteration server side software will be updated to make use of the new features in the PTU TASS library. Client side API will be changed to reflect new functionality of the server.

At the end of each cycle there will be a consultation with the client as to whether he feels that the project is continuing in the right direction.

The project schedule was produced to illustrates all the main tasks and planned deliverables (figure 1.1).

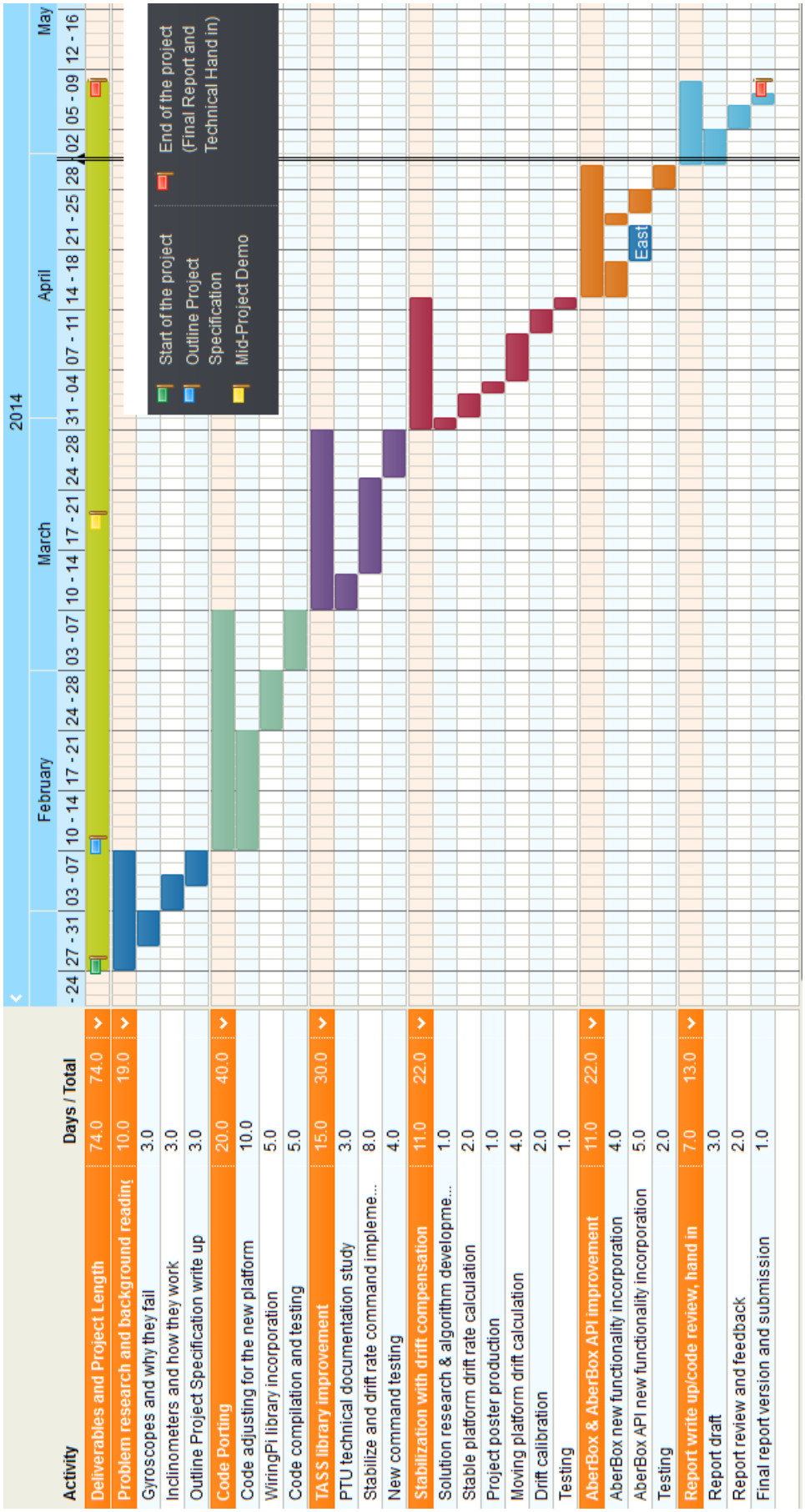

Figure 1.1: Project schedule

Chapter 2

System design

Since the hardware choice was dictated by the current system base and client choice there is not much to be discussed in this field. This chapter will give just a quick overview of the hardware design and then will concentrate on the software part and in particular drift rate algorithm development.

This project is based on the system that is currently in place, it has control over PTU can take readings from the inclinometer and control relays. It consists of the Gumstix minicomputer, GPIO14 chip, relays, inclinometer, PTU, server side code (running on Gumstix), client side code (provides API for the interaction with the server) and the library implementing PTU TASS communication protocol to interact with the PTU.

2.1 Changes to the system

There was a decision made by the client to replace the platform (Gumstix) that is currently used. The rationale behind the decision were client's concerns about the currently used Gumstix computer which is getting old. In case of a breakdown it would be difficult to find the replacement parts. One of the features requested was the ability to compile accompanying code on the platform itself (this is currently impossible on the Gumstix due to the hardware limitations) instead of cross-compiling code on the PC and then uploading executables to the Gumstix. Following a thorough discussion Raspberry Pi minicomputer was chosen as a replacement platform. It has enough power to compile the code as well as all the required interfaces for the connection of the other equipment. PTU TASS library will be reused and its functionality extended to implement the new features. Server and client side codes will be adjusted accordingly to provide access to the new functionality.

2.2 System architecture overview

This project will consist from the hardware and software parts, each of them will be discussed further in this chapter. The server side will have software running on the Raspberry PI minicomputer and handling all client requests. Connection will be initiated by the client over the TCP/IP protocol. Client side application will be using provided AberBox API to send/receive commands. Overall system archite cture is presented on the figure 2.1.

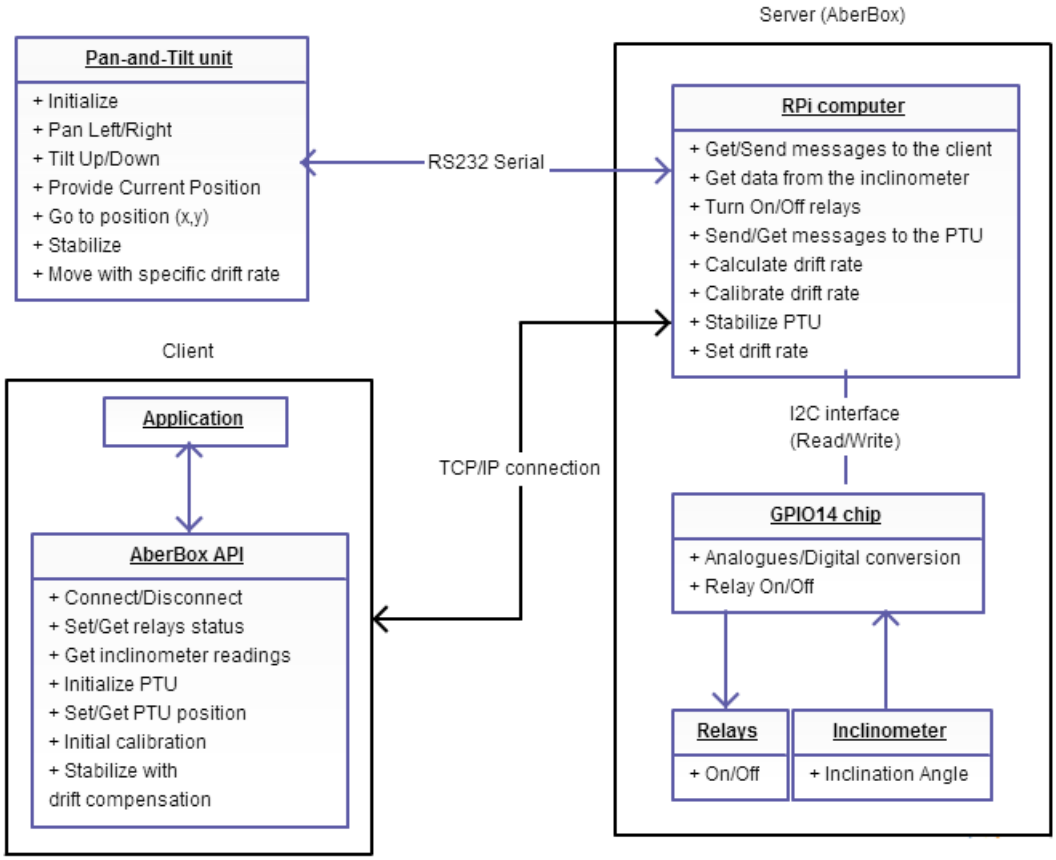

Figure 2.1: System Architecture Overview

2.3 Hardware Design

The proposed system will consist of the Raspberry Pi minicomputer, analogue inclinometer, GPIO14 chip, relays and the PTU.

2.3.1 Raspberry Pi

As a platform for the main control system Raspberry Pi minicomputer was chosen. It will replace the currently employed for this task Gumstix single board computer.

Raspberry Pi is a credit-card-sized single-board computer with a 512 MiB of RAM and 700 MHz ARM based CPU. It is powerful enough for the proposed tasks to be completed and have all the required interfaces to be connected to the other peripheral. It has GPIO pins, including SPI and I^2C interfaces, UART serial console, 5v and GND supply pins. Since there will be loads of power available on the vehicle power consumptions of the device were not taken in to account when choosing the platform. Such a powerful device may be an overkill for this task, but the decision was made by the client. Furthermore in the future this platform may be used to handle additional tasks.

2.3.2 Inclinometer

An inclinometer will be required to get data about the current chassis position in the space. The client suggested to use the analogue SCA121T-D05 dual axis inclinometer (figure 2.2) that provided instrumentation grade performance for levelling applications in harsh environment. It will be bolted to the chassis of the vehicle and will provide information about the inclination angles. This data will then be used during the PTU drift rate calibration. Since it is an analogue device a way to convert from analogue to digital format will be required.

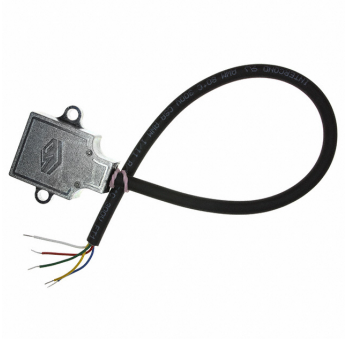

Figure 2.2: SCA121T-D05 Inclinometer

2.3.3 GPIO14 Chip

The GPIO14 chip is a pre-programmed PIC16F818 micro controller. It intends to provide general purpose I/O expansion on the I^2C bus. It has 14 general purpose I/O lines and 5 analogues input channels with 10-bit A/D conversion [2]. GPIO14 chip will be used to convert inclinometer signal from the analogue format to digital and to switch on/off relays. This chip will be connected over the I^{2}C bus to the Raspberry Pi.

2.3.4 PTU

Pan-and-Tilt Unit is a high-precision integrated motion control systems produced by the Sagebrush Technology (now part of the RIEtech Global, LLC [3]). It is designed for the <10Kg payloads and is often used to hold cameras, antennas or for instrument positioning. To connect it to the Raspberry Pi (which has TTL interface) RS-232 to TTL converter will be used. As a part of this project it is used to hold a panoramic camera.

2.4 Software Design

This study will be based on the code that is already used on the Gumstix. It is currently used to send control commands to the PTU, get readings from the inclinometer and switch on/off relays. The code base consists of three main parts: the client side software, the server side software (running on the Gumstix and responding to client calls) and the library that implements PTU TASS communication protocol. In this section different software design aspect will be discussed

2.4.1 PTU TASS library

PTU TASS library is written in the C++ and implements commands to communicate with the PTU. All communication is done over the serial interface. Currently the library has all the necessary commands implemented to perform basics tasks such as: pan left/right, tilt up/down, get/set position, get/set rotation limits. One of the objectives of this project is to implement 'stabilize' and 'drift rate' commands, as well as drift rate calculation.

The class diagram for the library is presented on the figure 2.3. PTinterface class provides functionality to interact with the PTU. It has two data structures defined. PTcoord data structure is used to hold values of the PTU position. DriftRate data structure is created at the beginning of the program and stores drift rate values for the both axes, there is a `StabilizationTime` attribute as well, it is required to store time of the last call to stabilization command. The drift rate data structure will have `private` access specifier since it should be accessed by the instantiated object only, no other code should be able to modify values inside since it will affect the PTU drift rate value used for the drift rate cancellation.

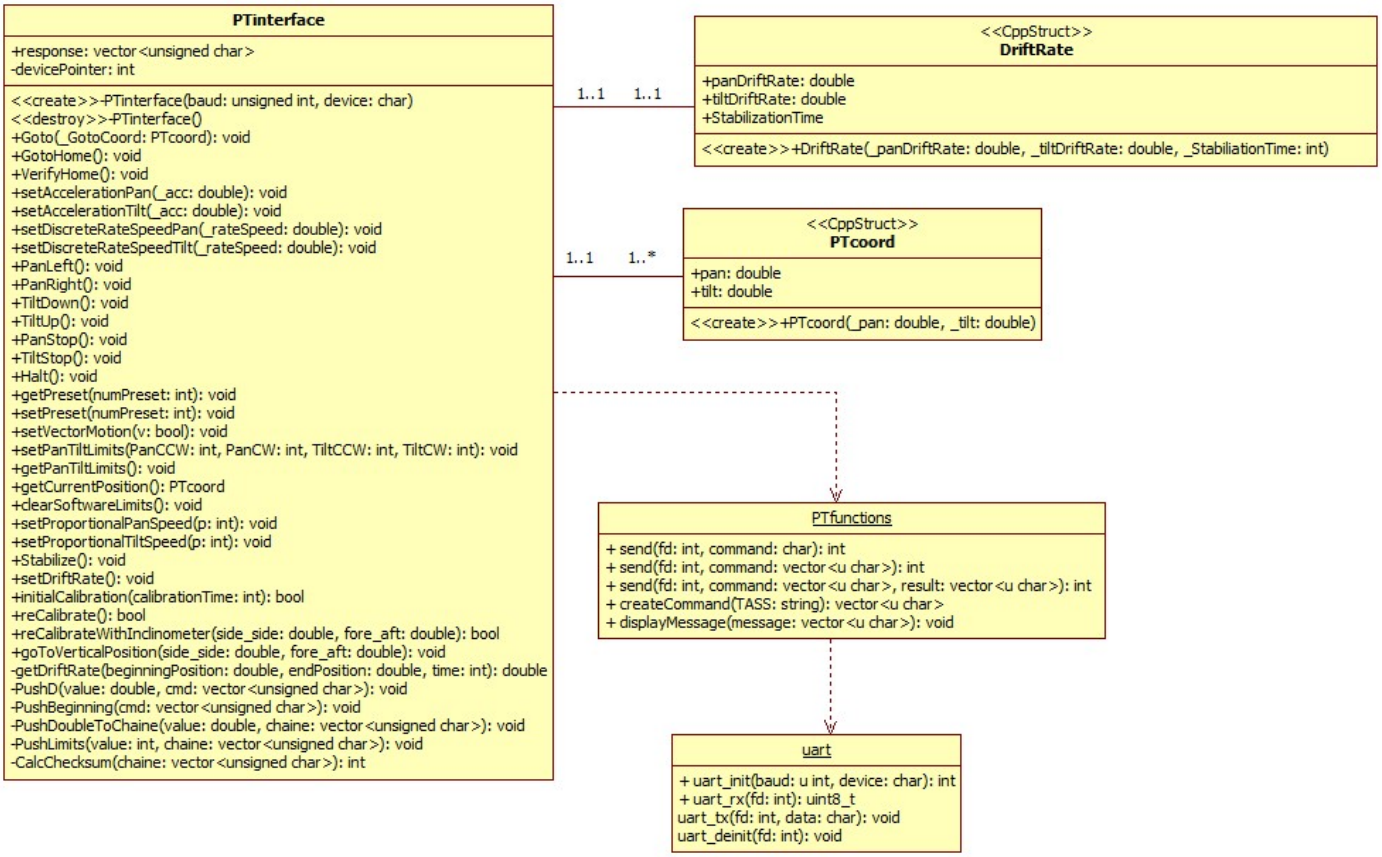

Figure 2.3: PTU TASS library

2.4.1.1 Stabilization command

The stabilization command tells PTU to maintain a desired platform orientation in the space independent of the vehicle motion. The stabilization command consists of the two letters "HI" which

needs to be sent to the PTU in ASCII encoding. The message format consists of six bytes of header plus one to 256 bytes of payload plus a one byte checksum [13]. This command will be implemented and added to the PTinterface class as the `stabilize()` function without any arguments and will not return anything. It will have a public access specifier to be accessible by the external code.

2.4.1.2 Drift rate command

There are two commands available to specify drift rate: "**mPf,f,f,f*" and "**mrf,f*". The first one moves the platform to a specific position and then command it to continue to move at a given inertial rate. The second one just command PTU to move at a given inertial rate without telling desired position. There were plans to use both commands, but after some testing was carried out it appeared that the first command doesn't work as it is described in the documentation. The problem in detail will be presented in the implementation chapter under the **??** section. This fact influenced further design decisions since there was just one command available to set the drift rate.

The star in the beginning of the **mrf,f* command means that this is experimental feature and may not work as described. Following two letters define command itself. The two *f* characters define floating-point number format and should be replaced with the two numbers representing the drift rate. It is necessary to supply fractional part of the number as well, even if it will consist from the zeroes only (e.g 10.00 not just 10) otherwise command will not be understood by the PTU.

As it was mentioned in previous subsection, command and following numbers should be in the ASCII encoding. The fact that floating-point numbers are unknown before the drift rate is calculated, means that they will have to be translated in to the ASCII encoding while the program is running. Special function will be developed just for this task. One possible approach to translate double in to the ASCII encoding in C++ is to use `std:cout` object. However with this approach alternative function to calculate the checksum would be required. The current function, that calculates the checksum of the message, takes as an argument vector of characters and returns integer.

I decided to use `sprintf` function to translate double values in to the ASCII encoding. With this approach the system will be able to calculate the checksum of the message with the function that is already available. I am also more comfortable using `sprintf` because it mimics behaviour of the `printf` which is already familiar to me. The only difference is that `sprintf` function directs all output to the supplied buffer instead of the console.

2.4.2 Server Side

The server side software is responsible for the overall control of the PTU and inclinometer. It will be running on Raspberry Pi minicomputer and will respond to the client commands. On of the main challenges is to port the current system from the Gumstix to the Raspberry Pi platform and make it work.

2.4.3 Client Side

The client side software provides an API to interact with the server side. The connection is made over the TCP/IP protocol. The plan is to extend the API upon the creation of the new stabilization functionality.

2.4.4 Operating System

Raspbian is a free operating system that will be running on the Raspberry Pi. It requires some initial configuration before the main software can be successfully run. Firstly, it requires configuration to prevent Linux from using the serial port; it also needs WiringPi library [6] installed to successfully communicate over the I^2C bus with the GPIO14 chip. All the necessary configuration is covered in the Appendix A.

2.5 Drift rate calculation

PTU drift rate in essence is a speed at which platform moves on the both axes. To calculate speed we use formula where distance travelled is divided by the time of travel. The same method can be applied to calculate the drift rate of the PTU platform. At the beginning we record current PTU platform position on both axes, tell it to stabilize at the current position (and it starts to drift), wait certain amount of time and record position again. Knowing start and end position we can find out the distance travelled. Dividing distance travelled by the time waited we get the speed or drift rate.

However this method will work only if the Pan-Tilt Unit orientation in the space remains the same during both reading of the platform position. If the unit would be tilted while it is stabilized it would try to adjust platform position to reflect its current orientation in the space. Then the position value recorded at the end would be made up of the drift rate plus the PTU orientation in the space, which is not what required. However this scenario needs to be accommodate as well since the rover will be continuously stopping at the different places (with the different chassis inclination angles) and performing drift rate calibration. In the following subsections algorithms to calculate drift rate will be presented for the both cases.

It is planned to perform initial drift rate calculation at the each start of the program. The first algorithm perfectly suits this task since it provides initial values which can be used to cancel drift at the early stage and avoid huge drift at the beginning. The second algorithm, that uses values from the inclinometer, is intended to be used to adjust or recalibrate the drift rate which is already applied to platform.

2.5.1 Fixed position drift rate

For the platform that is not moving during the calibration, drift rate can be calculated using the following formula:

$$DR = \frac{\Delta P_{1,2}}{t},$$

where DR - is a drift rate, $\Delta P_{1,2}$ - is a difference of the two position readings, taken at the beginning and the end of the stabilization, t - the time elapsed between two readings.

Since the platform reports its position in degrees, the result we get is in degrees per second as well. However the drift rate sent to the PTU must be specified in the radians per second. To convert it to the radians following formula can be used:

$$DR = (\frac{DR_{deg/s} * \pi}{180}) * -1,$$

where DR - is a drift rate in radians per second, $DR_{deg/s}$ - is a drift rate in degrees per second. To stabilize PTU drift rate sent needs to be of the same magnitude, but an opposite sign, for this purpose whole expression is multiplied by minus one.

The algorithm for the drift rate calculation is presented on the figure 2.4

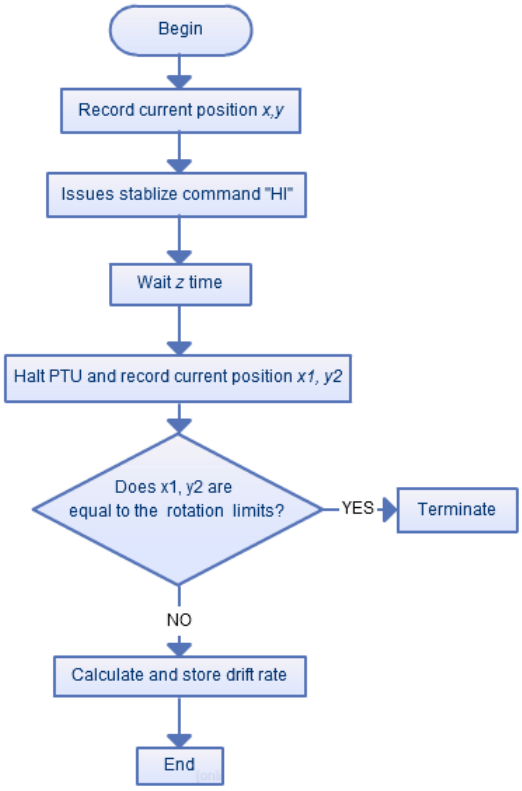

Figure 2.4: Drift rate calculation

Since the time frame for this task to take place is specified in advance, there is a check to make sure that during the initial drift rate calculation procedure PTU doesn't hit its physical rotation limits. The logic is that if it did hit rotation limits, platform would not be drafting any further, however the time would be still moving forwards. We can not rely on that data from about the second position since we don't know how quickly rotation limits where hit. Hence we can not accurately determine the drift rate based on this data.

2.5.2 Drift rate for the moving vehicle

The drift rate calculation for the moving vehicle is a bit more complicate and requires information about the platform position in the space. The rover will be stopping from time to time to do a drift rate calibration. However it is very unlikely that when it stops position will be the same as it was the last time when calibration took place. To be able to calculate drift rate, while PTU is being stabilized and rover moved to a different place, we need to know current PTU orientation in the space (inclination angles of the chassis that PTU is mounted on). For this purpose inclinometer, mounted on the chassis, will be used to obtain current vehicle position. Knowing the inclination of the chassis and current PTU platform position we can find out difference between these two values which will be the drift rate. The algorithm for the drift rate calculation is presented on the figure 2.5

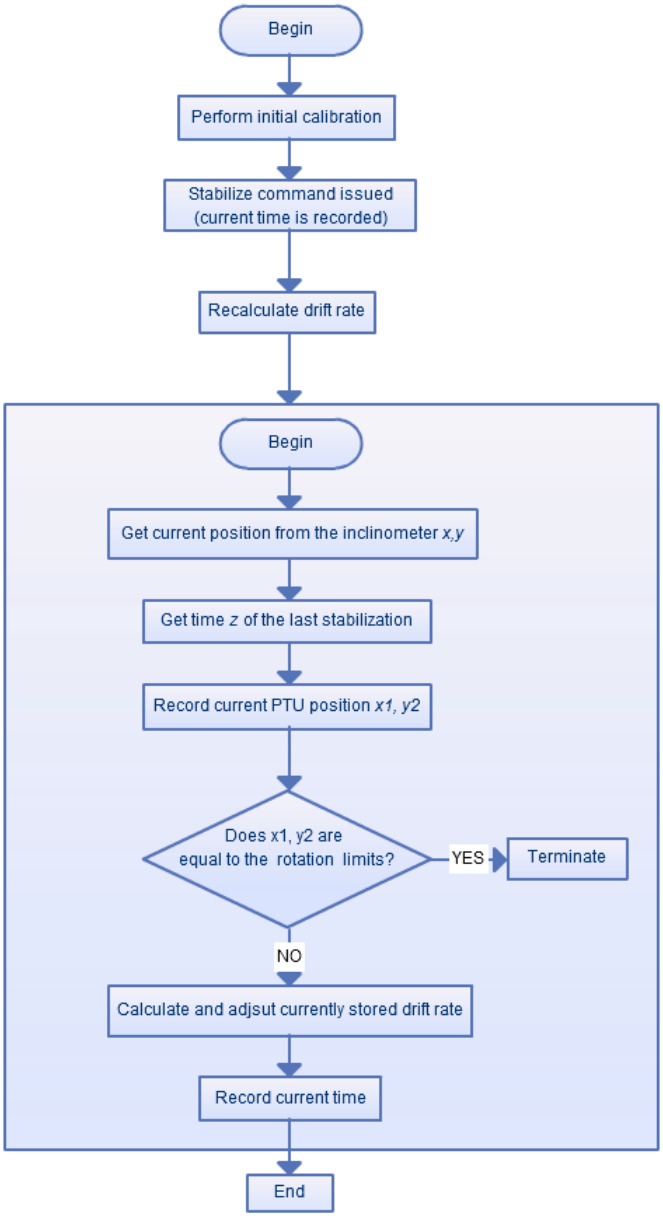

Figure 2.5: Drift rate calculation for the moving vehicle

Chapter 3

Implementation

The implementation of the project began on schedule at the beginning of week 2. The following chapter describes how the design was implemented in the development of a working system. Important code samples and functions have been included where appropriate. Using the chosen methodology, development ran in a series of iterations, at the end of each the different parts of the system were expected to work.

For the system development it was required to be able to connect Raspberry Pi to the PTU to test the code functionality. The PTU was unmounted from the rover, brought to the Digital Systems Laboratory , bolted on the stand and connected to the power supply.

Replacement of the Gumstix to the Raspberry Pi minicomputer and all the necessary wiring was done by the laboratory technician who I am very grateful to. Otherwise it would probably have taken a few more weeks of my time to find out how everything should be fitted in.

3.1 Raspberry Pi set up

Before any further work could be conducted Raspberry Pi minicomputer required operating system and some initial configuration. To get Raspberry Pi up and running quick start guide [11] was followed. Since the monitor with the HDMI input was not available, HDMI->VGA adapter was used, there were some problems in getting video output on the screen, however after tweaking configuration file problem was resolved. Once the system locale was changed and boot partition expanded initial setup was finished. To be able to work with the Pi remotely over the TCP/IP network secure shell protocol was enabled and static IP address configured (static IP address was used because RPi and computer used direct connection and as such no DHCP server was available). For the easier code editing and uploading the SAMBA suite was installed and SMB/CIFS share created. Now it was possible to access code directly from the Windows machine and edit it over the network.

3.2 Iteration 0

The iteration 0 focused on moving code from the Gumsitx on to the Raspberry Pi. This meant compiling source code on the new platform and testing its functionality. The following tasks were identified for this iteration:

1. Compile and run program on the new system.

2. Check that client on the different computer can make connection to the server.
3. Add platform specific I^2C library.
4. Check that data about inclination can be read from the inclinometer.
5. Check that relays can be switch on/off.

Since both platforms were running some sort of the Linux based OS it was not expected to see any OS specific compilation errors.

3.2.1 I^2C bus configuration

Due to the fact that two platforms were different in the architecture it was expected that implementation for the I^2C bus will be different on the Raspberry Pi and code that makes use of it may not work or even compile. It was decided to analyse current code and try to identify any platform specific code or libraries. After a quick analysis the piece of the code making calls to the I^2C library was found. It was a code in the file `gpio14.c` which provided certain level of abstraction when sending and receiving commands to/from the GPIO14 chip. This code was using a library which was specific to the Gumstix environment and would not work on the Raspberry Pi. Luckily the WiringPi project was found online. It is a GPIO access library written in C for the RaspberryPi and released under the LGPL license so it was available for commercial or non-commercial use. WiringPi includes a library which can make it easier to use the Raspberry Pi's on-board I^2C interface [6]. The installation of the WiringPi library is covered in the appendix A.

The I^2C library in WiringPi package provides the similar interface, to write and read data on the bus, as the library on the Gumstix. However there were a few differences. The file descriptor was required each time to read/write data on the bus. The function names in the new library were different as well.

The `gpio14.c` file required some changes to be able to interact with the GPIO14 chip over the I^2C bus. First of all function names to read and write data were change to the equivalent ones provided by the new library. Secondly after the I^2C system is initialised the returned file descriptor is stored in a variable and passed as an argument in future when reading/writing on the bus.

It looks like an easy fix, however a substantial amount of time was spent to understand the problem, look for the solution and put it in place. One of the problems was that a majority of the examples found on the web pages were describing interactions with the I^2C bus in the Python using incompatible with C libraries. At some point I even thought that I will have to develop my own library for the I^2C. Luckily WiringPi project was found.

Considerable amount of time was spent examining GPIO14 chip documentation and how it works, together with background reading of the I^2C bus specification and how data over it is sent to the GPIO14 chip. This knowledge was required to understand the code in the `gpio14.c` file and what it does, to be sure that when everything is compiled it will work.

After the issue with the library was resolved and minor issues fixed code was compiled and run. During testing application was running fine, it accepted client connection and would respond to the commands. However no sensible readings could be obtained from the inclinometer. Further investigation showed that device initialized in the default code was wrong. To obtain correct address of the GPIO14 chip on the I^2C bus I used `gpio i2cdetect` command. It showed up that chip appears under the 0x20 address, not the 0x21 as it was in the code. I suspect it was

specific to the previous platform or was changed for the testing. Another minor issue faced was the code, to initialize GPIO14 chip, placed in the file responsible for the relay control. The call to the `relay_init()` function in the `relay_control.c` file would initialize GPIO14 chip as well, although this is not obvious when looking at the function name and becomes apparent only after the implementation is examined. This lead to some confusion and slow down. After these issues were resolved inclinometer readings were successfully obtained.

The last minor problem was the issues with sending inclinometer readings back to the client. At client side they were not the same as on server, in fact most of the time it was showing just zeroes. Since this feature was not in the priority it was decided to leave it until the iteration 3, when new functionality would be incorporated to the current code.

3.3 Iteration 1

The iteration 1 focused on implementing new PTU control commands in the PTU TASS library. As it was mentioned in the 2.4.1 section there were two commands to implement:

1. The stabilize command "`HI`".
2. The drift rate command "`*mrf,f`"

The were no difficulties encountered with implementation of the stabilize command. All it required is to send "HI" letters in the ASCII encoding to the PTU. The man page for the ASCII character-encoding scheme was consulted to identify corresponding numbers to the letters "H" and "I". To hold the message was used a vector of unsigned characters where the header, payload (the message) and the checksum are pushed. Testing showed that the implemented command works fine and PTU starts to stabilize at the current position straight away.

The second command was a bit more complex. With the command(`*mr`) itself it required two floating-point numbers to be supplied as well. These numbers are unknown before the application performs drift rate calculation and hence can not be converted to the ASCII encoding in advance as it is done with the majority of the other commands. For this purpose was developed separate function that takes as arguments floating-point number and vector of characters. The supplied number is then converted to the ASCII encoding using `sprintf()` function and added to the vector.

Although these two commands presented a relatively simple task, the implementation took a bit longer that planned. Before staring this project I knew a very little about the ASCII encoding and how it works. There was some confusion since the documentation stated that all command information is to be represented by hexadecimal ASCII characters, however the current implementation used decimal notation and it worked fine. I decided to use decimal numbers as well and follow overall code notation laid out by the previous author.

3.4 Iteration 2

When the iteration 1 has finished with all the necessary features implemented and tested it was time to develop an algorithm for the drift rate calculation.

3.5 Stabilization with drift compensation

The algorithm presented below will work only if we assume that vertical position of the PTU platform is desired. It does not accommodate the case when the PTU platform is stabilized at the position other that vertical. This was found at the time of writing report and could be fixed given the more time.

3.6 AberBox & AberBox API improvements

Chapter 4

Testing

4.1 PTU TASS library testing

ID	Requirement	Description	Input	Expected output	Pass/ Fail	Comment
1	FR1	Test PTU connection and initialization	Connect to PTU and perform fast initialization	PTU platform should rotate to identify its limits	P	
2	FR2	Send tilt Up/Down commands to the PTU	"TU/TD" commands are sent	PTU platform should tilt UP/Down	P	
3	FR3	Send pan Left/Right commands to the PTU	"PL/PR" commands are sent	PTU platform should pan UP/Down	P	
4	FR4	Keep platform stabilized.	"HI" command is sent to the PTU.	PTU should adjust platform position to keep it vertical when whole unit is tilted.	P	Platform position changes when whole unit is tilted.
5	FR5	Send drift rate to the PTU	"*mr*f*,*f*" command is sent.	Platform is stabilized with the given drift rate	P	
6	FR6		Platform is stabilized with the given drift rate	P		

Chapter 5

Evaluation

Appendices

Appendix A

Libraries and Raspberry Pi configuration

This Appendix contains essential information on the Raspberry Pi configuration. This configuration is required for the proper server side software functioning.

1.1 Raspbian OS configuration

The installation procedure of the Raspbian OS is covered on this website: http://www.raspbian.org/

1.2 Preventing Linux using the serial port

Disabling the serial console is required to use the Raspberry Pi's serial port (UART) to talk to other devices. There is a script which allows to automate the whole process and is covered in detail on this page: https://github.com/lurch/rpi-serial-console .

1.3 WiringPi library

The WiringPi project has been used to read and write data on I^2C bus as part of the interaction with the gpio14 chip. The library is open source and it is available from the Gordons Projects website: http://wiringpi.com/ . Installation is covered on this web page: http://wiringpi.com/download-and-install/ . Before the I^2C interface can be used I^2C drivers need to be load into the kernel, guide is available at the http://wiringpi.com/reference/i2c-library/ .

Appendix B

Appendix 2

Annotated Bibliography

[1]

[2] "GPIO14 - General Purpose I2C I/O Expansion Chip." [Online]. Available: http://www.robot-electronics.co.uk/htm/gpio14tech.htm

This page gives an ovierview of the GPIO14 chip (which is essentialy a pre-programmed PIC16F818 8-bit microcontroller). Information on this page gave understand of how chip works and how it should be connected on I2C bus with other device. I used it heavily while migrating code from the Gumstix to RPI. In particular I was reffering to this page to understan what information is held in the different registers and where I need to write data to change chip behavior.

[3] "RIEtech Global." [Online]. Available: http://www.rietechmotioncontrol.com/aboutus.html

Includes description of the company: when it was established and what it produces.

[4] Dr Frédéric Labrosse, "Project: Idris." [Online]. Available: http://www.aber.ac.uk/en/cs/research/ir/robots/idris/

This web page gives an overview of the Idris vehicle and what it is used for. In particular I was interested in how PTU is mounted on the vehicle (its physical orientation).

[5] R. P. Foundation, "Raspbian Installer." [Online]. Available: http://www.raspbian.org/RaspbianInstaller

This page has extensive documntation describing how Raspbian OS should be installed on the RPi. It also contains suggestions for the tweaks

[6] Gordons Projects, "Wiring Pi - GPIO Interface library for the Raspberry Pi." [Online]. Available: http://wiringpi.com/

This website is a home for the WiringPi library. I was consulting it when porting code from the Gumstix to the RPi. In particular I was looking how to install WiringPi library and what functions are available to send commands over I2C to the gpio14 chip.

[7] Jacob Fraden, *Handbook of Modern Sensors*. Springer, 2010, p. 680. [Online]. Available: http://www.plentyofebooks.net/2011/05/download-handbook-of-modern-sensors-4th.html 1441964657.

This book gave me understanding of why different sensors a not perfect and may have problems with accuracy. On 339 p. it covers gyroscope architecture and in particular discusses factors that affect their accuracy.

[8] Y. Kreinin, "How to mix C and C++." [Online]. Available: http://yosefk.com/c++fqa/mixing.html

[9] Loads, "Principles behind the Agile Manifesto." [Online]. Available: http://agilemanifesto.org/principles.html

Explains ideas behing the agile manifesto and what they really mean. Gave me an understanding of how development should be done in agile environment.

[10] NXP Semiconductor, "I2C-bus specification and user manual," 2012. [Online]. Available: http://www.nxp.com/documents/other/UM10204_v5.pdf

This user manual gives in depth description of I2C bus; I was interested in analysing the fundamental architecture of this bus as well as reading about SDA and SCL signals

[11] Raspberry Pi Foundation, "Raspberry Pi Quick Start Guide." [Online]. Available: http://www.raspberrypi.org/wp-content/uploads/2012/12/quick-start-guide-v1.1.pdf

Describes steps to get RPi up and running in a short time.

[12] ——, "What is a Raspberry Pi?" [Online]. Available: http://www.raspberrypi.org

This website gives an overview of the Raspberry Pi computer and what it is capable of. It was a starting point for me in understanding of its hardware specifiacations and documentation about initial setup.

[13] Sagebrush Technology Inc, "Command Set Documentation, Pan-Tilt Gimbals,Servomotor Version," 2004.

This command set documentation defines a standard for the control of the PTU. I am interested in the contents of the tables with information about the error codes and general device control commands.

[14] Sagebrush Technology Inc., "Model 20 Pan-Tilt Gimbal User Manual," 2005.

This manual describes system set-up information and identifies the parameters necessary to operate a Model 20 Servo unit. In particular it describes available TASS stabilization commands.

[15] SensorWiki.org, "Gyroscope." [Online]. Available: http://sensorwiki.org/doku.php/sensors/gyroscope