CHALMERS

Development of Android Software for Logging of Engine Data for Shell Eco Marathon

Bachelor's Thesis in Computer Science and Engineering

JOANNA ERIKSSON
MIKAEL JOHANSSON
ANDERS NORDIN
EWA SIMPANEN
PHILIP STEINGRÜBER
JOHANNES WESCHKE

Department of Computer Science and Engineering (CSE)

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2013
Bachelor's Thesis/report no. 2013:023

Acknowledgements

This report is written as a partial fulfillment of the requirements for a bachelor's degree at Chalmers University of Technology at the Department of Computer Science and Engineering. The report describes the development of a new system for the Chalmers Vera Team car.

The work started in January and ended in June the same year, 2013.

We would like to thank Civinco AB for their help and support throughout the project.

Finally, a special thanks to Anders Johansson at the Department of Applied Mechanics and Roger Johansson at the Department of Computer Science and Engineering for their feedback and supervision.

Abstract

Teams taking part in the Shell Eco-Marathon challenge compete to create a vehicle that is as fuel-efficient as possible. To fulfill this goal the vehicles in general and especially their engines have to be as optimized as possible. To be able to do this as efficiently as possible the teams require a way to analyze how the engine performs and what it does over the course of a run. Most teams accomplish this by connecting the vehicle's ECU (Engine Control Unit) to a laptop in a development environment where a test-run is simulated. A much better solution would be for the team to be able to analyse the performance of the engine and the behaviour of the car in real-time under a real test-run on the track. This would allow the team to get as accurate data as possible to analyze and make qualified adjustments from. This report deals with the problems and result of an attempt to create a system that solve this issue. The system includes an Android application sensors and a desktop application.

Sammanfattning

Deltagarna i Shell Eco-Marathon tävlar om att bygga en så bränslesnål bil som möjligt. För att uppnå detta måste bilarna och i synnerhet dess motorer vara optimalt konfigurerade. Detta kräver att lagen har möjlighet att analysera hur motorn presterar och uppför sig under körning. De flesta deltagare gör detta genom att ansluta motorns styrenhet (ECU, Engine Control Unit) till en bärbar dator och analysera dess beteende och förbrukning. Detta görs dock med motorn och bilen stillastående i utvecklingsmiljön där en riktigt körning simuleras. En bättre lösning hade varit att låta laget kunna utläsa datan från styrenheten och hur bilen påverkas i realtid under en riktigt körning. Laget kan på så sätt se hur bilen och motorn beter sig och få så exakta testvärden som möjligt samtidigt som den kör. Det hade gett dem möjligheten att efter loppet kunna utföra justeringar på bilen och motorn utifrån dessa. Denna rapport behandlar problem och resultat som uppkommit vid gruppens försök att skapa ett system som löser denna uppgift. Detta görs med hjälp av en Android-applikation, sensorer och en dator-applikation.

List of abbreviations/terminology

3G Third generation telecommunication service.

Android Operating system for smartphones.

API Application Programming Interface, describes the available functions of a software component and how they should be used.

DAQ Data acquisition System. Is the process that samples signals which corresponds to physicals values and converts them into digital numerical values that can be read by a computer.

DOM/SAX Document Object Model, Simple API for XML: models for processing XML documents.

ECU Engine Control Unit.

GUI Graphical User Interface.

Java A platform independent programming language.

JFreeChart Java library used in Windows-application for drawing charts.

JMapViewer Java library used in Windows-application for drawing maps.

Kernel Central part of an operating system, acting as a communications layer between applications and hardware.

L2C Linear to Circular

MySQL My Structured Query Language, relational database management system typically used with PHP.

Paper prototyping Drawing crude prototypes on paper prior to development. Commonly used when designing GUIs.

Parser A method, or program, to do the parsing.

Parsing Processing of data for use in a software application.

PHP Hypertext Preprocessor, a server-side scripting language mainly used for web development.

RAD Requirements Analysis Document, a document containing the list of requirements set up for an application.

SDD System Design Document, a document set up by the developers on how the program is built.

SQL Structured Query Language, programming language used for relational database management systems.

SQLite Relational Database Management System used in Android.

Thread Lightweight process that allows for separate functions to be carried out in a manner that makes them seem simultaneous.

TPMS Tire Pressure Monitoring System

UI Thread User Interface Thread, the main thread for execution.

USB Universal Serial Bus, a industrial standard to define cables, contacts and communication protocol for connecting electronic devices.

WiFi Technology that allows wireless communication.

XML Extensible Markup Language, a language for defining rules on how to encode a document in a format that can be read by both humans and computers.

Contents

List of Figures

List of Tables

1 Introduction

The greenhouse effect has been a hot topic over the last years and one part of counteracting this development is to build more fuel efficient cars. To develop the most fuel efficient car in the world, the developers need to know as much about the car's behaviour as possible in order to find out where energy loss is taking place. This project will focus on developing a system consisting of an Android application, which receives data from both an ECU (engine control unit) and from sensors, displays some important values and sends all of the data to a computer via a web server.

1.1 Background

Shell Eco-marathon is a competition with the aim of creating the most fuel efficient vehicle. The competition consists of three races over the course of a year, where hundreds of universities are in contest over who can cover the longest distance with only one liter of petrol. One team has reached a distance of 2 485 kilometers which is equivalent to the distance between Paris och Moscow. Chalmers competes in the two main classes, which are Prototype and Urban Concept. In the Urban Concept class the goal is to emulate a real life car, whereas in the Prototype class the single goal is to construct the most fuel efficient vehicle. Chalmers currently holds the Swedish record in the Prototype class, with a distance of 1 243 kilometers.

The team participating in the Prototype class on behalf of Chalmers is the Chalmers Vera Team and they are in need of an Android application to log data from the Engine Control Unit in the car. The team has been using a Windows-based software for this task. However, due to the importance of keeping the weight of the vehicle to a minimum, they are not able to keep a laptop in the vehicle during races. This means that the team has no way of obtaining ECU-data from the actual runs. The ability to log and observe data in real time would enable them to make better adjustments.

Developing a system to perform this task will provide the Chalmers Vera Team a unique opportunity to process data that can result in improvements of the engine and the car itself.

1.2 Objective

The purpose of the project was to develop a software which could handle communication between the engine of the vehicle and an Android phone. The phone would save the engine data throughout a race and it would later be available to the Vera team.

An expansion of the project led to a system that handles the communication between the Vera team and the engine, in real time during a race. The software is used to log, save, display and send data from the engine. In addition to the data handled from the ECU, the vehicle's GPS position is logged and there are different types of data from sensors as well. The sensor data works as a complement to the engine data and enables optimization in other parts of the vehicle.

On the user end of the system, the Vera team can receive the data and examine it with the help of a computer software built for presentation. The decision to develop a new software instead of the one delivered along with the ECU by Civinco was made due to the fact that it did not support our needs on a couple of points, namely that it did not allow for real time updates as well as it not being able to support additional sensors.

In case all requirements are not met, the most important part is making sure the Android software is able to store data from the ECU and transfer it to a computer for presentation.

1.3 Limitations

In order to implement the system during a limited amount of time, while keeping costs low and avoiding unreasonable risks, some limitations were made to the project.

No investigations have been made to find the optimal Android phone. The aim has only been to find a phone which meets the requirements to fulfill the goals of this project.

The group has not had any influence over the type of ECU used in the vehicle and the system has therefore been customized for this particular ECU, which is a SA3000 from Civinco AB.

Based on the backgrounds of the group members, the programming in the desktop application is written in Java. No inquiry has been taken concerning which language would be the best to use.

The system is limited to read data from the ECU and will not write to it. This limitation is done based on the fact that the group has not been given the opportunity to take part of any documentation describing how this is done. Also, the group does not possess the proper knowledge of experimenting with engines, which could result in personal as well as material damage.

1.4 Task

The main function of the system and also the main goal of the project, is to establish communication between the Vera team and the ECU. The developed software, provides the team with information about the vehicle's properties while driving.

The communication with the engine is performed by the ECU and it provides data e.g. engine speed, temperature and petrol consumption. The data is then transferred by a USB-wire to an Android phone, where the data is saved. Additional data from sensors is sent to the phone via Bluetooth. The Android phone has a Graphical User Interface (GUI) which provides the driver with some of the most relevant data. Information about steering, elapsed time, lap times, average speed and current speed is displayed on the screen of the phone. The smartphone compresses and transfers the data (engine as well as external sensors) to a web server, where it is converted to physical values. Lastly, the data is presented to the user on a computer with the help of a desktop application which fetches the logged data from the server.

1.5 Method

The project will be split into multiple parts: a pre-study and analysis of requirements; the system development; and finally the testing, validation and integration into the vehicle.

1.5.1 Preliminary study and analysis

This stage consisted mainly of collection and analysis of requirements, which was done in collaboration with the Vera team. In practice, this was done by oral and written communication with team leader of the Vera team where the purpose of the application and its requested features were discussed. Based on this a document stating requirements were compiled which can be found in appendix A. A Requirements and Analysis document (RAD) has also been developed for the mobile application and desktop application which can be seen in appendix A and A. The documents were sent to the team for approval. To describe the functions of the application, a document stating use cases has been compiled and can be found in appendix A.

1.5.2 Development

Early on in the process of development the group was split into several parts; communication between ECU and smartphone; communication between smartphone and web server, including GUI for the app itself; communication between web server and computer as well as presentation of data; and development and integration of additional sensors to collect data from other parts of the vehicle than the ECU. Further reading on the development process can be found in section 4 through section 6.

1.5.3 Integration and validation

This phase of the project focuses on making sure that the applications developed meet the requirements set up by the Vera Team in a satisfactory way, as well as making any necessary changes to achieve this goal. This will be done in conjunction with the Vera Team in the process of integrating the finished product into the vehicle and the everyday work of the team.

2 Mobile Application

The mobile application is the central module throughout this project. The fundamental idea is that the smartphone itself is a substitute to a computer. Its main purpose is to collected logged data from the ECU (engine control unit), compress it and transfer it to the web server. The application also collects data from the sensors through a Bluetooth connection. Besides transferring data, the application works as a dashboard for the driver, presenting important values related to the run.

2.1 Layout and Design

First and foremost the application is designed to be practical. The application's main functions need to be easy to operate for the driver while driving, wearing gloves. To ensure that only the most important information is presented, a close communication has taken place between the software developers and the Vera Team. It should be noted that even though the application should be handy to use, a beautiful design is important to create confidence in the application and make it pleasant for the user to look at.

2.1.1 Start- and Run-view

The first thing that meets the user when starting the application is the start view illustrated in figure 1. When the user presses `Launch` on the screen, the run view appears. Thereafter, the search for a GPS signal starts, which can be seen in figure 2. While the application is searching for a GPS signal the `Start` and `Stop` buttons are inactivated. As soon as the signal is found the application displays a message which can be seen in figure 3. The `Start` button is at the same time activated. As the user presses start, when the race starts, the application can present the view as in figure 4. Also, the stop button is activated and the resume button inactivated.

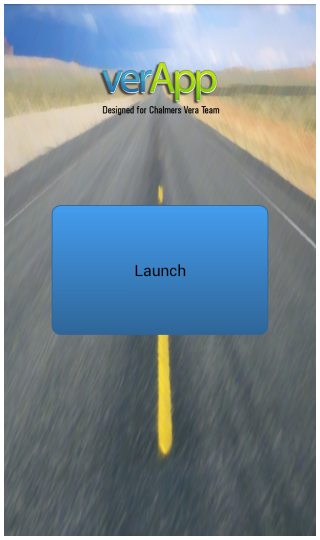


Figure 1: *Start view.*

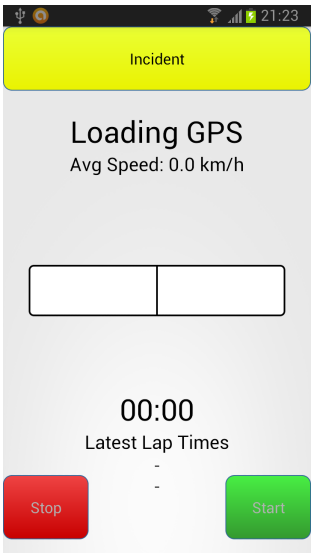


Figure 2: *Searching for GPS signal.*

Figure 3: *GPS signal found.*

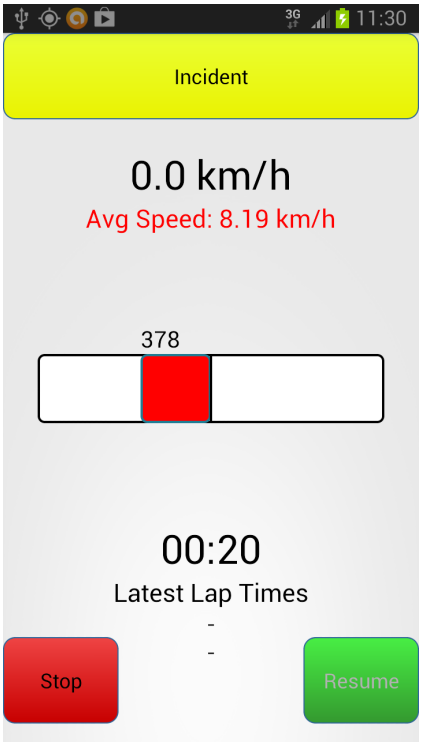


Figure 4: *Run view.*

The run-view contains three buttons. The buttons are placed far apart in order to avoid that the user presses the wrong button by accident. The `Incident` button is supposed to be pressed when something particular occurs and will be described more in detail in section 2.2.2. The `Stop` button pauses the timekeeping and the logging. The `Start/Resume` button starts/resumes the logging and timekeeping. When any of the buttons has been pressed the user is presented with a confirmation which can be seen in figure 5.

In the center of the screen the user is presented with the current speed of the car, the average speed throughout the whole session, the lap time which presents the time of the on-going lap and further down on the screen the driver can see the lap times of the previous laps. On the screen the total time of the race is also presented. In the center of the screen there is a bar with a steering indicator. The indicator shows the driver which way and how much the steering wheel was spun. The color of the indicator changes in accordance with the fact, if the steering was within accepted values.

When developing an interface it is important to take into consideration that

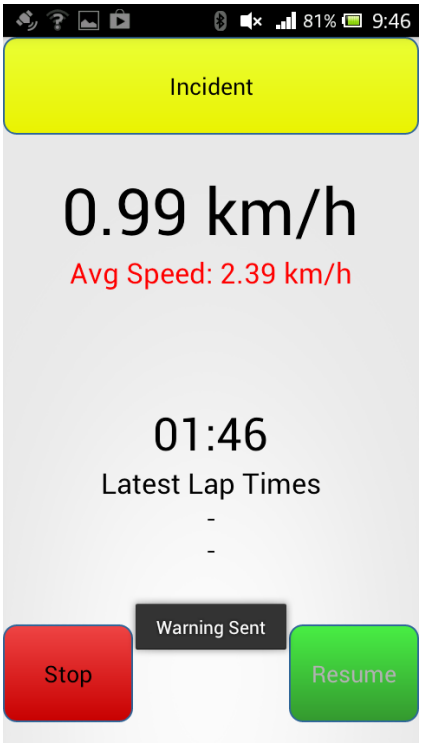


Figure 5: *Confirmation that a button has been pressed.*

the user might click on the screen by accident.[1] To ensure that the back-arrow has not been pressed by accident the user is presented a dialog box asking ”Are you sure you want to go back?”. This action is implemented in all views in the application.

2.1.2 Action bar and Settings

In accordance with Android standard the back arrow at the button of the screen is used to go back to previous screen. The options menu should contain actions which effects the app globally (Android Developer Guide, 2013d). In VerApp the options menu contains the actions `Reset`, `Calibrate Steering` and `Exit`. The first option, returns the user to the start screen and by doing that resets all calculations in progress. The second option, will be will be described in section 2.2.3. Finally, the third option, `Exit`, closes the application.

[1] Jonas Andersson, Ph.D. Student, Chalmers, lecture 2013-02-18

The settings menu is developed with the vision of making it expandable. It means that even though the settings menu today only contains a menu with one item it should be easy to add more items to the menu in the future.

Early on in the development process a feature was implemented which allowed the user to choose logging frequency through a dialogue box. Later on this was removed as the group did not see a need for being able to change this. The reasons behind not using this are described further in section 7.3.1.

2.2 Functionalities

The mobile application has two responsibilities: transferring data from the ECU(engine control unit) to the web server and work as a dashboard. As previously described, the application displays total race time, steering, speed, average speed and lap time. The three last mentioned functions are solved with the help of GPS. The GPS is also used together with the logged data for better understanding when reviewing the information. Another part of the application is the storage on the phone while the data has not yet been transferred to the web server.

2.2.1 Global positioning system

Global positioning system, or more commonly known GPS, is used to solve several functions in the app. GPS equipment uses satellites to determine its position. GPS is very reliable and works in all weather conditions anywhere in the world, at any time. With the help of GPS it is possible to determine the user's latitude, longitude and altitude. Once the user's position is set, the GPS can calculate things like bearing(direction), track, trip distance, distance to a given position, sunrise and more (Garmin, 2013).

Most new mobile phones, including the Sony Xperia Sola used for this project, supports GPS. Most use a technology similar to GPS called A-GPS (Assisted Global Positioning System) (Sony Mobile, 2013). The difference between A-GPS and GPS is that A-GPS uses the mobile network while GPS communicates directly with the satellites. A-GPS is more suitable for mobile devices since the amount of programming and CPU power required is reassigned to the assisting servers. Some of the downsides with this is that the user has

to pay the phone service provider for the amount of data usage. In practice A-GPS is often less accurate (Tech2, 2013).

Before the GPS can be used within the application, the GPS needs to allocate a signal. As soon as the smartphone has received its first signal the application is ready to use. Each signal update triggers a method inside run view with the latest position. To get the GPS working on Android the application needs to have permission to `ACCESS_FINE_LOCATION` which gives the most precise position from sources such as GPS, WiFi and cell towers. This connection is not always working, therefore, as a precaution, `ACCESS_COARSE_LOCATION` and `INTERNET` permission is also granted. The last mentioned gives access to a approximate location from network sources such as WiFi and cell towers (Developer Guide, 2013b). The app is designed so that it is possible to run without access to GPS.

2.2.2 Run view

The run view is the central activity in the application. It is present most of the time for the driver, calculating and displaying important values during the race.

As soon as the GPS has allocated a signal the `Start` button becomes available. When the race starts the driver pushes the start button which thereby starts a timer and ECU communication.

As mentioned in the above section 2.2.1 the application receives the latest position through a certain method. This position is later used to determine a new lap. All lap times are saved, even though, only the two latest lap times are present for the driver.

The new lap feature was originally estimated in a different way, namely by taking the start position and its course. By subtracting and adding 90 degrees to the initial course and finally adding 10 meters in each direction its possible to form a straight line. This line would represent the start/finish line. Then, for each new GPS position that was received, a method would check if the current direction of the car would intersect with the finish line. As a precaution, the method would also check so the car was in a radius of 10 meters to the finish line. Figure 6 illustrates the described approach.

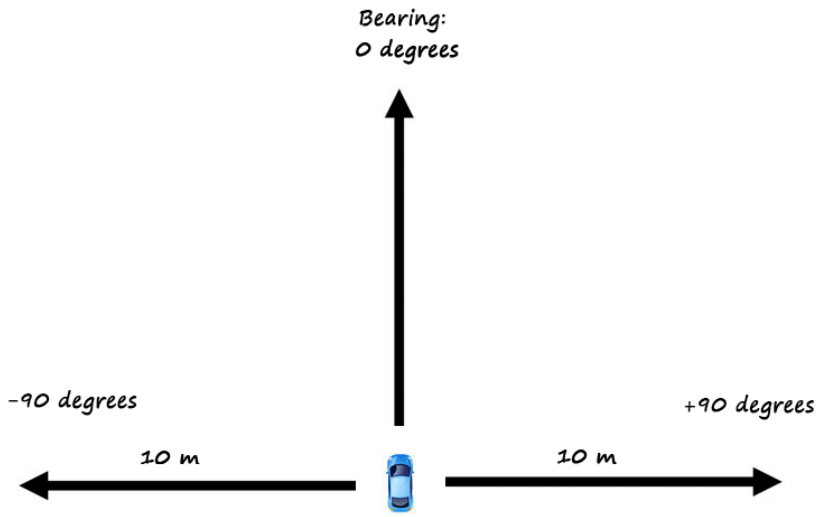


Figure 6: *First attempt to calculate new lap*

The first attempt did not work as planned, in fact, it did not work at all. A second attempt was made and included a much simpler solution. The heavy mathematical part was discarded and the new calculation used only the distance to the starting point. With help of testing and with respect to the GPS accuracy we found that a 10 meter radius would still do the job. Also, a thread, running concurrently was added, which would lock the lap counter when recently updated. This would ensure that a single lap only is counted once. The second attempt showed off well in tests and is the current solution that is being used in the app.

It is of great importance for the Vera Team to know the speed at all time. The speed is presented to the user with the help of the latest GPS position. Since the speed is given in meters per second the value has to be converted before printed to the `textview`(Android Developer Guide, 2013g). The average speed is estimated with the help of the formula (1).

$$\frac{total\ distance}{total\ elapsed\ time} \tag{1}$$

The total distance is estimated by the distance between the second latest point and the latest point. As mentioned before, the A-GPS is not always

completely accurate and might get the idea that the object is moving when it is not. The difference in coordinates may be very small but enough to trigger the whole system. A problem arose from the scenario above when determining the total distance. The solution is to put the calculation inside a separate running thread. Instead of logging for each triggering signal the thread forces it to execute every two seconds. In practice two seconds is a lot compared to the actual frequency of the GPS signal.

During the race unforeseen conditions might occur, the possibility to stop and resume run mode is available at all time. This means that if the driver would push `Stop`, all related functions are effected. To be even more precise, a flag is set to false which implies that average speed, lap time, total time, data logging and current speed is paused. However, the current value for each is stored.

The application is also logging data during run-time. If something occurs during the race it might effect the logged data. This could be hard to backtrack after the race. As a consequence of this, a button called `incident` has been implemented which gives the driver the opportunity to report something unexpected. This is done by setting a warning flag to true which is sent along with the data logged at the same time.

There are three more buttons which have not been covered yet, the reset, back and exit. Reset basically returns the user to the initial view. By doing this the run view is destroyed and all variable values are wiped away. Back button simply forces the current activity to finish. Which activity that should be displayed afterwards is handled by the operative system. As there are multiply threads running it is of great importance to the phones performance to kill all threads and that's exactly what the exit function does, causing the application to shutdown. Since these are core features that should be available in the whole app, all three are implemented in the base activity class which is a super class to all other activities in the app. The base class itself inherits from `Activity`, which is the class provided by the API.

There is is no indicator like the one used in this project in the Android API which meant it had to be built from the ground up. The indicator is implemented by putting rectangles upon each other. The size, color and placement of the front rectangle are then changed according to the steering. A line has also been put in the middle to help the driver distinguishing if the front rectangle is to the left or to the right. The background rectangle with the line in the middle can be seen in figure 2. To make the size of the

rectangle proportional to the steering equation (2). In the equation `steer` is the steering data received from the sensor, `nomSteer` is the nominal steering signal i.e. the signal then the car is going straight forward, `width back rect` refers to the width of the background rectangle and `maxSteer` and `minSteer` refers to the maximum and minimum steering signal possible.

$$\text{width front rect} = \frac{|\text{steer - nomSteer}| \cdot \text{width back rect}}{maxSteer - minSteer} \tag{2}$$

A listener has been implemented which listens if the steering variable in the program is changed. In the case that the steering variable is changed, the appearance of the front rectangle is updated according to its value and the value of the steering is put at top of the bar which can be seen in figure 4.

2.2.3 Calibration of sensor to measure steering

The steering is measured with a potentiometer which will be described in section 4.1.2. In order to ensure that the steering showed to the driver by the steering indicator is correct a calibration needs to be done. The calibration can be done anytime the driver wishes through the action bar. When the user chooses `Calibrate steering` in the menu a dialog is shown that instructs the user how to calibrate and then guides the user through the calibration. To calibrate the user turns the steering wheel back and forth a couple of times and the programs saves the most extreme values.

2.2.4 USB Connection

To connect the smartphone to the ECU, the Engine Control Unit of the car, see 3, we choose to use the mini-USB connection on the ECU box and connect this to the micro-usb on the smartphone. The USB, Universal Serial Bus, is an industrial standard developed to define a joint interface for cables, connectors and a communication protocols between electrical devices. The USB standard states that for a interaction between devices to be possible, one have to claim and act like a USB host and the other have to act like a USB device. In our case our destined phone, Xperia Sola, takes the part of

acting as the USB host device and thereby setting up and initialize the communication over the USB bus. The ECU, SA3000 from Civinco, connected to the engine in the Vera car will act as a USB device and active listen after instructions from the phone on the connected USB bus.

For the application to be able to setup and initialize the connection over the USB hardware, the Android operative system have included a `USB Host API` (for Android versions over 3.1) that provides support for communications with connected USB devices (Android Developer Guide, 2013i). The application sets up the communication to the ECU using the classes in the API, the VID, vendor id, and the PID, product id, that is the device identification number for the ECU. The connection is established at the beginning when the application is started and the communication between the phone and the ECU is open and idle for command. When the start button is pressed the program sends a start command, see section 2.2 Data transfer protocol, to the ECU and start a input stream of logged data from the USB bus to the application storage file.

2.2.5 Bluetooth Connection

The connection between the smartphone and the microcontroller, see section 4.3, is managed by a wireless Bluetooth connection. The Bluetooth connection is a wireless standard technology for exchanging data between devices over short distances and is supported in the Android OS. The application uses the standard Bluetooth API (Android Developer Guide, 2013j) to set up the framework and control transfers over the connection between the smartphone and the Bluetooth module connected to the microcontroller. The Bluetooth connection is established when the application is started and the serial input/output stream is open and idle for transmission. When the applications start button is pressed a signal is sent to the microcontroller and the data package flow from the microcontroller, see section

2.2.6 Storage

After the signal is received from the ECU, the application tries to transfer the signal to the web server as soon as possible. This is not always doable due to the fact that the connection may not be available or there are several

files waiting in queue to be transferred. Therefore it is important to store the file, at least, until it is possible.

The first implementation was done with a SQLite database which is the internal database in Android. By using a relational database system it is easy to organize all the data. Also there are significant advantages in speed and efficiency when retrieving large amounts of records from a SQL database(Illinois Institute of Technology, 2013).

Due to the amount of data being logged, the way of sending each signal value with related parameters one by one was dumped since it lacked of efficiency. Instead the logged data is written directly into a text file. The text file has been assigned a maximum size of 30 kB until the file is closed and the application starts to write to another text file. Once the file is complete, it is compressed into a zip file. By doing this reduces the file size from 30 kilobytes to approximately 900 bytes. Finally, the zip file is placed in a folder called `waiting`, more about the folder name in 2.2.7.

2.2.7 Transfer of logged data

When the logged data has been collected and packaged into zip files there is time to transfer the file to the web server where it will be unpacked and processed.

To transfer a zip file there is a class called `FileManager` which is responsible for only sending the files. The class itself is asynchronous. The advantage is that it is possible to run tasks, like the transfers, through the user interface thread (UI thread is the main thread for execution) in the background. This approach eliminates the need of using threads and also simplifies the implementation. When the `FileManager` class is invoked it first checks that all necessary directories exist and creates those which have not yet been created. As soon as the right file is located it is sent to the web server through a `POST(HTTP)` method.

When a race proceeds there might be hundreds of zip files in the memory which are waiting to be transferred. An important part is to keep track of which files have been sent and which have not. This was first implemented with a text file that kept all names of files that have been sent and for each new transfer the class would check whether the file had been sent before or

not. However, this caused an error which made the application crash. The problem will be discussed further in 7.3.3.

The solution that is currently active in the application is executed accordingly: instead of using the local database, the files are compressed and saved into a directory which is called `waiting`. The directory contains all files that have not been sent. Since the data is labeled with a time stamp it does not matter in which order they are sent. As soon as the application has received an OK message from the server the file is transferred to another directory called `sent`. With this method it is possible to lower total transfer time, amount of code, storage space and memory usage.

3 Engine Control Unit

The engine in a car today is controlled and managed by a small embedded computer system called the car's Engine Control Unit. The ECU controls all the critical parameters such as fuel injection, ignition and the boost-control of the car's engine. By optimizing these parameters you can enhance the performance and the efficiency of the engine.

All the parameter settings in the engine are stored in the memory of the ECU and can be changed during operation for optimization. The ECU controlling the Vera car's engine is from the tuning-device company called Civinco and their latest control unit, the SA3000.

3.1 Data transfer protocol

The data transfer between the ECU and the smartphone is based on a standard USB-serial connection, see section 2.2.4, where the phone initiates and controls the communication between the two units. In this system the phone acts as the USB host and is responsible for managing all the data transfers over the USB bus. The ECU acts as a USB device and only sends data onto the USB bus after an established connection and a request from the phone.

3.1.1 Initiate data transfer with the ECU

To initiate reading of log data from the ECU the protocol states that the host, the controller of the communication, sends a start command to the ECU, this sets the ECU in a binary logging state. In this state the ECU waits for an address to a register where to start reading from. For this project, we start to read from the first register in the ECU and reads the following 48 registers. This is where the main parameter of the engine is located, this includes all the data the Vera-team needs to be able to analyze and optimize the behavior of the engine.

When the ECU reads a full command input, in our case "bL 00 30", it starts to loop through the given registers and send data sequentially from the register (one data-package per loop) on to the USB bus.

3.1.2 Data-package from ECU

After a complete initialization and the transfer-command has been executed, the ECU starts to send data-packages on to the USB bus. The packages are specified by the protocol to contain three parts, the head, the payload and the tail, see figure 10. The header contains of three bytes of data, where the first 2 bytes are “PS” (ASCII-value) to declare the beginning of a new package. The third byte in the header is the number of registers that have been read and added in to the payload. The payload carries the data from the registers that have been read in the ECU. The payloads can have a size ranging from 0 to 200 bytes depending on the number of register read in the ECU. Each register consists of 16 bits and is therefore divided into two bytes before being added to the payload. The tail consists of four bytes, two bytes of control-bits and two bytes, ”DS” (ASCII-value), to declare end of package.

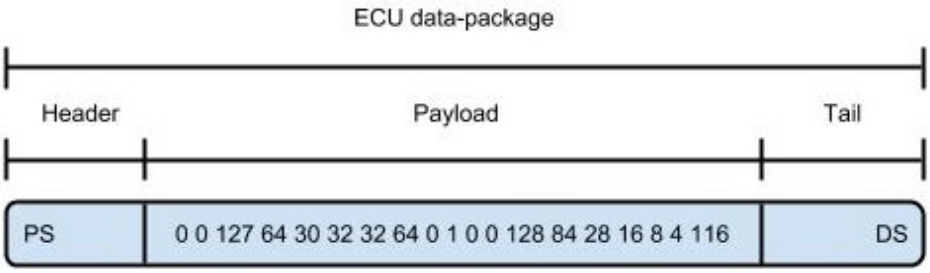


Figure 7: *ECU data package.*

4 Sensors

Much needed data is conveniently accessible through the ECU. However, there are still some parameters that are of interest, which are not connected to the engine in the same manner. Some of these are a helpful support to the driver, who is constantly coping with the task of keeping a steady pace and a straight course, momentarily lowering the impacts of friction and getting the most out of as little fuel as possible. Other data is valuable for the team as they evaluate the runs and attempt to optimize performance in a longer time perspective, which enables continuous development.

Accessing new data in the vehicle, requires a selection process to find useful transducers and sensible ways of installing them. These systems need to be robust enough to be able to perform in this unique environment, taking to account the disturbances caused by ignition, moisture, high or low temperatures, judder etc..

The data from the ECU is sent to the Android phone which presents some of the data and transmits all of it to a server. Therefore, the data collected by the various sensors is preferably sent to the phone and merged with the data from the ECU, before transmitting it any further.

There are mainly two alternative routes to choose from. Either the sensors can be connected to the ECU's available input ports and get translated from there or they can be connected to a micro processor which gathers the data and then pushes it forward to VerApp. Which path is used is not at all much significant, a combination of the two could be an alternative. Some of the signals could travel through the ECU and some through a micro processor.

An Arduino microcontroller was chosen to manage all of the sensor data and send it through to the mobile unit by using a bluetooth module. This proved to be the most practical and convenient method to use when installing the sensors. Six different types of sensors were installed in the vehicle, which added up to 11 sensors in total.

The system built to gather data external to the ECU, is designed to be flexible, relatively stable and easy to use. The Vera Team should be able to maintain and make use of the systems without the need of assistance.

4.1 Critical data

The data which is of help when driving the vehicle should appear on the display of the mobile phone, through VerApp, which is easily accessible to the driver. One concern is that the screen may get too cluttered. This must be taken into account when choosing which of the signals to be displayed.

4.1.1 Speed

Knowing the speed of the vehicle is important for many reasons. One is that the driver obtains increased control, which is good for safety. Another reason is that the official rules for Shell Eco-Marathon, require both a certain speed and lap time. In the definition of competition it is stated, that teams must complete ten laps with an average speed of approximately 25 km/h, for their attempt to be validated (Shell Eco-Marathon, 2013).

One possible way to estimate the speed of the vehicle is through measuring wheel speed. Being able to measure wheel speeds using sensors, allows a much better measurement of speed than calculating it by using GPS signals. This method also provides additional information, as it can reveal how the wheels move in relation to each other. Measuring wheel speeds on all three wheels, provides the possibility to see if the car rears or if any of the tires skid, both of which can occur when accelerating quickly. Wheel speed is a great indication when optimizing the timing of the beginning and end of an acceleration.

To measure the wheel speed, small magnets are attached on the rims of the vehicle and for each wheel there is a proximity sensor attached to the frame, see figure 8. The sensor chosen for this system was the Hamlin 59025-1-S-02-A, see appendix B.2 for the data sheet. The Hamlin sensor is easy to install, it can handle variations in temperature, shock and vibration.

Whenever one of the magnets come within a distance of 11.5 mm of the sensor, an induced current triggers it. This will result in a series of indications for each wheel. The frequency of which the indications are made is combined with the circumference of the wheel, to calculate the speed. Six magnets are attached on each wheel and if the measurements are to be made at least every 0.1 seconds, they allow a speed as low as 9.01 km/h.

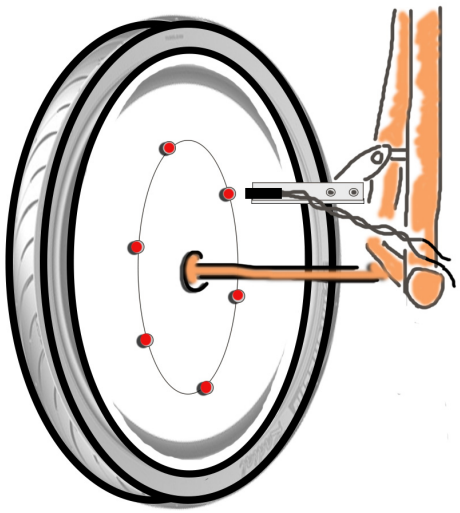


Figure 8: *Installation of the six magnets and the Hamlin sensor.*

In the case that a magnet would fail to trigger the proximity sensor for whatever reason, every sixth value would be lost. This is solved by processing the data in a way that the resulting speed is a combination of a calculated average value and the instantaneous speed. The speed and average lap time is presented on the display of the mobile unit.

4.1.2 Steering

A visual presentation of the steering angle helps the driver to keep the vehicle in line and minimize speed losses. It is valuable to have full control in turns on the track. Being able to get data of the steering angle allows repetitive testing and try outs in a manner that was not possible before.

The potentiometer, Wabash 971 RPS, is attached on the steering wheel, for more information see appendix B.3. It provides values which have a linear relation to the amount of degrees it is being twisted in relation to its center position. It is also possible to use sensors that measure angles or distances between a fixed point at the frame and the front or back of the wheel. This method would have been more complicated to install and maintain. The Vera team is only interested in relative values and not in absolute angles, witch makes the potentiometer accurate enough. It can also handle temperature extremes, harsh environments and mechanical shock and vibration.

The linear rotation can be translated into a relative steering angle by conducting tests. The values will be divided into steps that are of an appropriate size. As the vehicle is being maintained and the potentiometers position is altered, its settings will change. There is a function available in VerApp which allows calibration of the steering angle by specifying a new center value. The steering angle is presented on the display of the mobile unit.

4.1.3 Tire pressure

Measuring tire pressure can give the driver an indication of when a tire is loosing pressure. The vehicle has custom made carbon fiber rims, which are too expensive to run down. This would happen if the tires were to loose too much pressure.

As the wheels are in constant rotation, it is appropriate to use a pressure sensor capable of wireless transmission. It is common for car owners to use TPMS, *Tire Pressure Monitoring System*, which is wireless. There is a wide range of models of TPMS available for cars and motorbikes (TPMS Sweden, 2013a). A sensor that would fit the requirements of the vehicle is the TPMS 10.02.011 from Impaqed Products BV, see appendix B.4 for more information. The sensor was not implemented in the car during this project, although it could be a part of the future developments of the vehicle.

Unlike other TPMS sensors, this one is fitted straight onto the valve rather than having to dismount the tire and replace the entire valve. Moreover, they are very small in size and weigh only 3 g each. These sensors are capable of measuring up to 13 bar. The Michelin tires perform best while having a tire pressure of just above 6 bar.

Each of the sensors are marked with a tag to keep the different tire pressures apart, and they send data associated to this tag. A radio receiver which operates at the same frequency, 433.92 MHz, is needed to pick up the signals. The receiver Radiotronix RCR-433-MPR is designed for remote communication using radio waves of this frequency, see figure 9 and appendix B.5. There are many receivers of this kind to choose from and the one from Radiotronix is as good as any of them.

The TPMS sensors available on the market have good resolution and accuracy. They are designed to withstand temperature fluctuations, moisture,

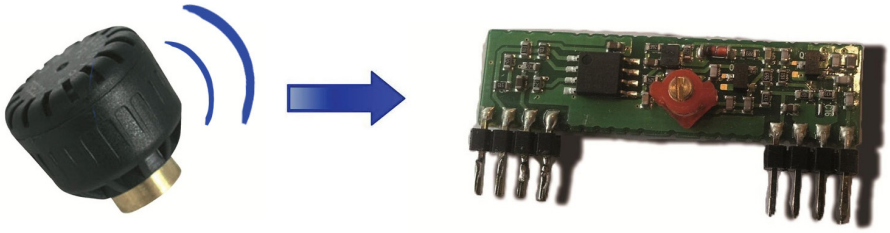


Figure 9: *The tire pressure sensor TPMS 10.02.011 from Impaqed Products BV can communicate with the receiver from Radiotronix.*

dirt and judder. The data sheet in appendix B.4 specifies that the sensor has an accuracy of pressure measurement of ±0.15 bar. TPChecker motorcycle TPMS monitor tire pressure and temperature every 3 seconds (TPMS Sweden, 2013b), it can be assumed that the sensors from Impaqed Products use a similar rate. The receiver has a data rate of 4 800 bytes per second and won't be hindering the transmission. However, there is need for a script that coordinates the data from the three sensors so that there will not be any losses.

4.2 Post evaluation data

There is sensor data which is not necessary for the driver to see while in the middle of a run. This data will not be presented before it is sent to the database. The values are used for the purpose of enhancing performance and is a basis for coming up with suggestions for long term improvements.

4.2.1 Strains in the framework

Vera's frame formerly consisted of carbon fiber, which initially is a very sturdy material but when it ages it loses its rigidness. The framework begun to shear. Therefore, it is interesting to measure unwanted displacements in the new steel frame to see if this is a good alternative material.

The deflections can be measured with the help of the SpectraSymbol Flex Sensor, see figure 10 and appendix B.6. Whenever the flex sensor is bent, its resistance changes. The flex sensor can easily be glued or otherwise mounted onto the frame where measurements are needed. The sensor can handle

variations in temperature, although it might be important to keep it dry. If there is too much moisture the resistance might be affected. The active length of the sensor is 9.5 cm and while flat, its resistance is 10 kΩ.

Figure 10: *The SpectraSymbol Flex Sensor. Some of the most common uses for this sensor is in robotics, gaming and medical devices.*

The Vera Team has two flex sensors which have been prepared with sockets that are easy to connect and disconnect from the microcontroller. The team is interested in relative values which means that the data will not have to be translated from resistance into an absolute degree of displacement.

4.2.2 Camber, Toe in & Toe out

The wheels of the vehicle can sometimes be bent in different directions, in addition to the steering angle. When the tires form an angle where the top of the two front wheels are slightly tilted inward, is called camber, see figure 11. Toe in is when the front of the wheels are tilted inward and toe out is when the front of the two wheels are tilted outward, see figure 12. The deflections are inevitable, but if they become apparent enough, they will cause increased friction and other disruptions.

The measurements are made by using the distance sensor GP2Y0A41SK0F from Sharp, see appendix B.7 for the data sheet. This sensor is as the previous ones, resistant to variations in temperature, moisture and mechanical shock. It is easily installed by mounting it on the frame and it measures the distance to where the light emitter and detector are directed. This Sharp sensor measures distances within the range of 4 cm up to 30 cm. As the data sheet implies, the sensor's accuracy is at its best when 7 - 9 cm from its target, where it is intended to be installed.

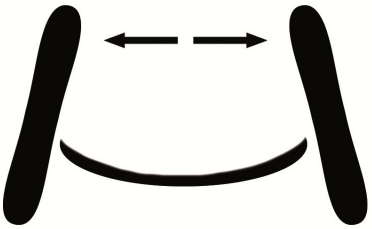


Figure 11: *Camber.*

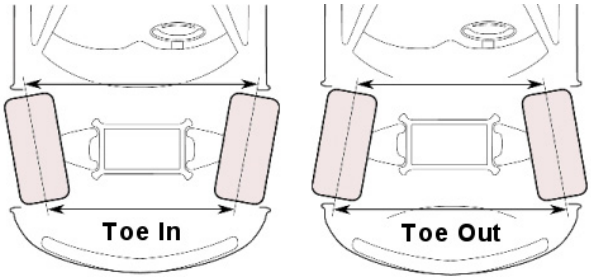


Figure 12: *Toe in and toe out.*

It is possible to install as many distance sensors as one would like. During this project two sensors have been installed and it is intended that they measure the camber. Since the suspension can not be changed during a race, the Vera team could move the sensors around and optimize the wheel settings during try outs. The data from the sensors only result in relative displacement.

4.2.3 G-force acceleration

To save fuel during the race, Vera is repeatedly accelerated up to a certain speed and is then left to make as much use of the momentum as possible until the next acceleration phase. This requires the acceleration to be monitored closely and adds pressure to make it as efficient as it can be.

There Memsic 2125 Dual-Axis Accelerometer #28017, see appendix B.8, is installed in the center of the vehicle to measure g-force acceleration. This sensor is capable of measuring tilt, collision, static and dynamic acceleration, rotation, and vibration. To obtain measurements in three axes, two Memsic accelerometers are solded together perpendicularly to each other, see figure

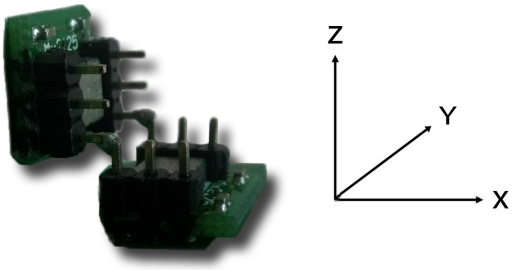


Figure 13: *Two Memsic 2125 Dual-Axis Accelerometer #28017, solded together.*

13. The sensors give the Vera team information about the movements of the vehicle and the values can be used to observe Vera's behaviour in curves or when an obstacle occurs on the track.

4.2.4 Engine chain tension

The engine chain tensioner makes sure the connection between the parts of the engine remains intact, without them being overstressed. If the chain is pulled too tightly, the engine can be damaged and if it is too loose, the performance is lowered.

To register and measure the movements in the chain tensioner, the linear image sensor IC Haus LFL 1402, see figure 14 and appendix B.9, is used. There is room for this small sensor on a surface right next to the lever, which is fixed between, and moved by, both a spring and the engine chain, see figure 15. The sensor has an array of 256 active photo pixels with a high resolution and it is capable of receiving and processing an image of what is in front of it. The lever is marked with a bar code pattern which is used to compute in what degree the chain tensioner is moving.

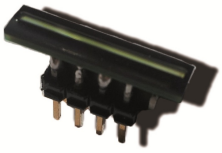


Figure 14: *The linear image sensor IC Haus LFL 1402.*

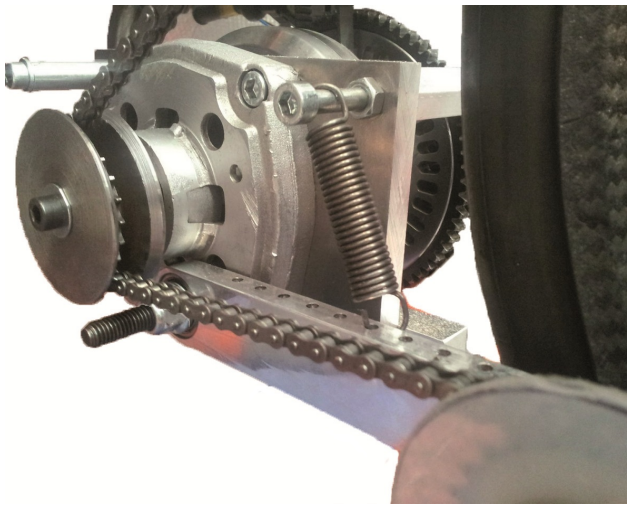


Figure 15: *The engine chain tensioner.*

4.3 Microcontroller

To manage the collected data from the 11 sensors we choose to use the Arduino Micro microcontroller that includes 20 digital inputs/output pins where 7 can be used as a PWM, Pulse Width Modulation, outputs and 12 as analog inputs/outputs. This suits the task of be able to read all the sensors data and connecting the Bluetooth-module, see appendix B.8, used to transmit the data from the microcontroller to the mobile phone, and also be small and light-weight, only 6.5 g, further minimizing the weight and space addition to the car.

The sensors measuring wheel speed, steering and acceleration, are permanently installed and connected to the microcontroller while the sensors for camber/Toe in Toe out, strains in the framework and engine chain tension are used mostly during testing and not while competing and therefore connected to the microcontroller with sockets that can be removed easily.

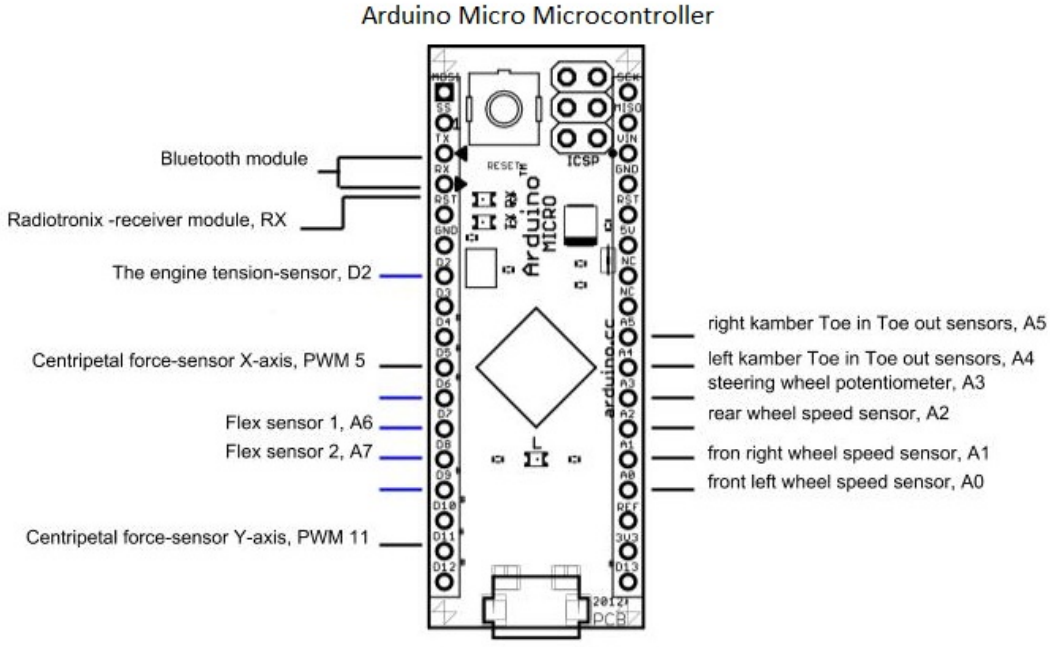


Figure 16: *Microcontroller connection scheme.*

The sensors are connected to the microcontroller as shown in figure 16, the three wheel speed sensors are connected to first three analog pins, A0- A2. A0 is connected to the front left wheel-, A1 to the front right wheel- and

A2 to the rear wheel-sensors. The steering wheel potentiometer is connected to the analog input pin A3 and the two camber/Toe in Toe out sensors are connected to the A4, left front wheel, and A5, right front wheel. The g-force acceleration sensors have three data values, the change in X-axis, the change in Y-axis and the change in Z-axis, and are connected to the PWM pins PWM5, PWM11 and PWM12. The three wireless tire pressure sensors are connected to the microcontoller via the receiver module, Radiotronix RCR-433-MPR see appendix B.5, and are connected to the RX-pin. The external sensors are only used for testing are marked with blue and are easy to connect and remove. The sensor which measures engine chain tension is connected to the digital pin D2 for digital input and output reading. The two flex sensors used to measure the strains in the framework are connected to the two analog input pins A6 and A7.

4.3.1 Data transfer from the Arduino to the Android-phone

The data transfer from the Arduino microcontroller to the mobile phone is based on the wireless Bluetooth technology, which is a standard for transmitting data over short distances. The Bluetooth protocol is split in to two parts where one part acts as the host-device, so called “host stack”, and is responsible for setting up and managing the connection of the two parts. The other part is the device that listens for the host-device to setup the connection, the so called ”controller stack”-part. A Bluetooth module, free2move F2M03GLA see appendix B.10, are connected to the Arduino receiver and transmitter pins and acts as the “controller stack” part in the system. The smartphone acts as the “host stack” part and manages the connection between the devices.

4.3.2 Data transfer protocol

To manage the 17 different data values read from the sensors a protocol has been set up for the data packages transmitted between the Arduino and the smartphone. The protocol states that the data-package from the Arduino’s bluetooth-module will contain three parts. It begins with a header part that states the beginning of the data package, followed by the payload holding the sensor data and lastly the tail part that states the end of the data-package, see figure 17.

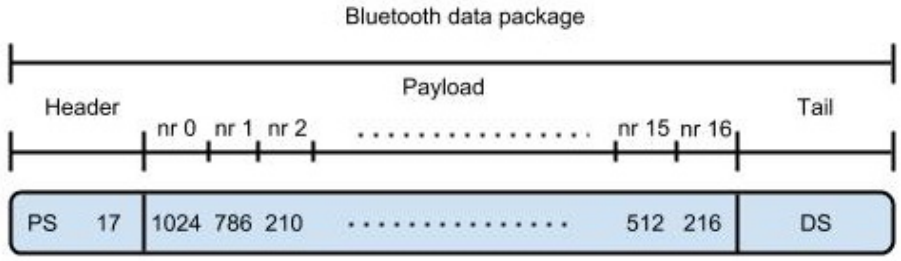


Figure 17: *Bluetooth data package.*

The head of the package starts by sending a 2 bytes "PS" declaration to declare the beginning of the package and then followed by 1 byte number that give the number of sensor-data included in the payload. The payload carries the read integer data from the sensors in the order from 0 to 16, see table 1. The tail announces the finish of the package by sending a 2 byte "DS" end declaration.

Table 1: *Payload data table.*

Payload nr	Sensor	Payload nr	Sensor
0	Left front wheel speed sensor	9	Wireless front right wheel pressure receiver
1	Right front wheel speed sensor	10	Wireless rare wheel pressure receiver
2	Rare wheel speed sensor	11	Front left wheel house pressure sensor
3	Steering wheel sensor	12	Front right wheel house pressure sensor
4	Left front wheel kamber sensor	13	Rare wheel house pressure sensor
5	Right front wheel kamber sensor	14	Engine chain tension sensor
6	Centripetal force sensor X-axis	15	Flex sensor 1
7	Centripetal force sensor Y-axis	16	Flex sensor 2
8	Wireless front left wheel pressure receiver		

4.3.3 Arduino program

The Arduino Micro microcontroller programming language is based on the C/C++ language and links to the AVR Libc, an API package that provides a subset of the standard functions from C for 8-bit microcontrollers. The program starts by declaring and setting the input and output pins to each sensor connected to the microcontroller according to figure X followed by the initialization of the bluetooth serial input-/output stream. It then connects and sets up the receiver to the wireless wheel tire pressure sensors. The program then starts to loop through the connected inputs and stores the value from each sensor every 20 ms. When the program has read the values from one loop it compiles the data to a data package according to the protocol and sends the package over the bluetooth connection.

5 Database

The initial plan for storage of log data was to only use a local database on the smartphone. The idea was to use a SQLite database since Android has native support for it. Originally the project only concerned local logging of data, but as the project expanded a decision was made to utilize an external MySQL database stored on an external web host.

5.1 Database modeling

The model was initially drawn on paper and translated into SQL statements for creating the MySQL database. After creating the database some arbitrary data points were inserted to allow for initial testing of the desktop application.

5.2 Data transfer to database

Data is sent from the smartphone in the form of a compressed .txt document containing logged data from both the ECU (Engine Control Unit) and external sensors. One such document is created every 5 ms. The data is transferred using a PHP script which decompresses, parses and converts the data (see 5.3) into physical values. The converted data is then inserted into the database using SQL statements.

5.3 Conversion to physical values

When the data is logged in the ECU no consideration is taken to the fact that different parameters have different units which works differently. e.g. ignition is in degrees, frequency in hertz and fuel consumption in liters per 10 km.

In order to do these conversions Civinco (the manufacturers of the ECU) provided a Matlab script which describes how this conversion is be done. This script was rewritten in PHP as it was to be used on a web server. Since all the data in the log file is not useful it is of interest to delete useless data

as early in the transfer process to minimize the amount of data being sent. The reason the calculations are not made on the Smartphone is that the web server has a lot more capacity. Civinco also provided the group with ROWDEF-,SENSDEF and TEMPDEF files which will be explained later on.

The script begins with the ROWDEF-, SENSDEF and TEMPDEF files being read and their content put into matrices. The ROWDEF file contains row definitions, SENSDEF contains sensor definitions and TEMPDEF contains temperature definitions. In order to find a certain parameter in the SENSDEF and TEMPDEF file the program uses the ROWDEF file to find at which row the relevant parameter can be found in the actual file.

Secondly the file containing logged data sent from the smartphone is read and put into another matrix called it DATA matrix. Before putting the data into the matrix, the program finds the beginning and the end of the logged data and erases the rest.

Then the ECU is logging it assumes that all parameters are linear which is not the case. If a parameters is defined as L2C (linear to circular) in the SENSDEF file the physical value might actually be negative. The physical value of all parameters (except temperatures which has a nonlinear behavior) is linear and are therefore calculated as equation (3).

$$\text{physical data} = \text{logged data} \cdot \text{constant} \cdot \text{factor} + \text{offset} \tag{3}$$

The factor and offset can be found through different columns in the the SENSDEF matrix. The temperatures are calculated using the TEMPDEF matrix.

5.4 Transfer from a web server

As stated in section 5 the system uses a MySQL database hosted on a web hosting service. The initial plan was to have the desktop application communicating directly with the database and code was written to fulfill this purpose. During the first phase of testing, a local MySQL server was used and the developed implementation was fully functional. However, due to the web host not allowing external connections directly to their database server a workaround was created in the form of a PHP script which creates an XML document using the logged data in the database.

6 Desktop application

The purpose of the developed desktop application is to retrieve logged data from a web server, process it and present it in a clear and concise manner.

The application was developed using the Java programming language and specifically the Swing framework as well as a few external libraries for drawing maps and charts, something that Java has no native support for. The choice of language and framework was made based on previous experience of the developers to facilitate a smooth development process.

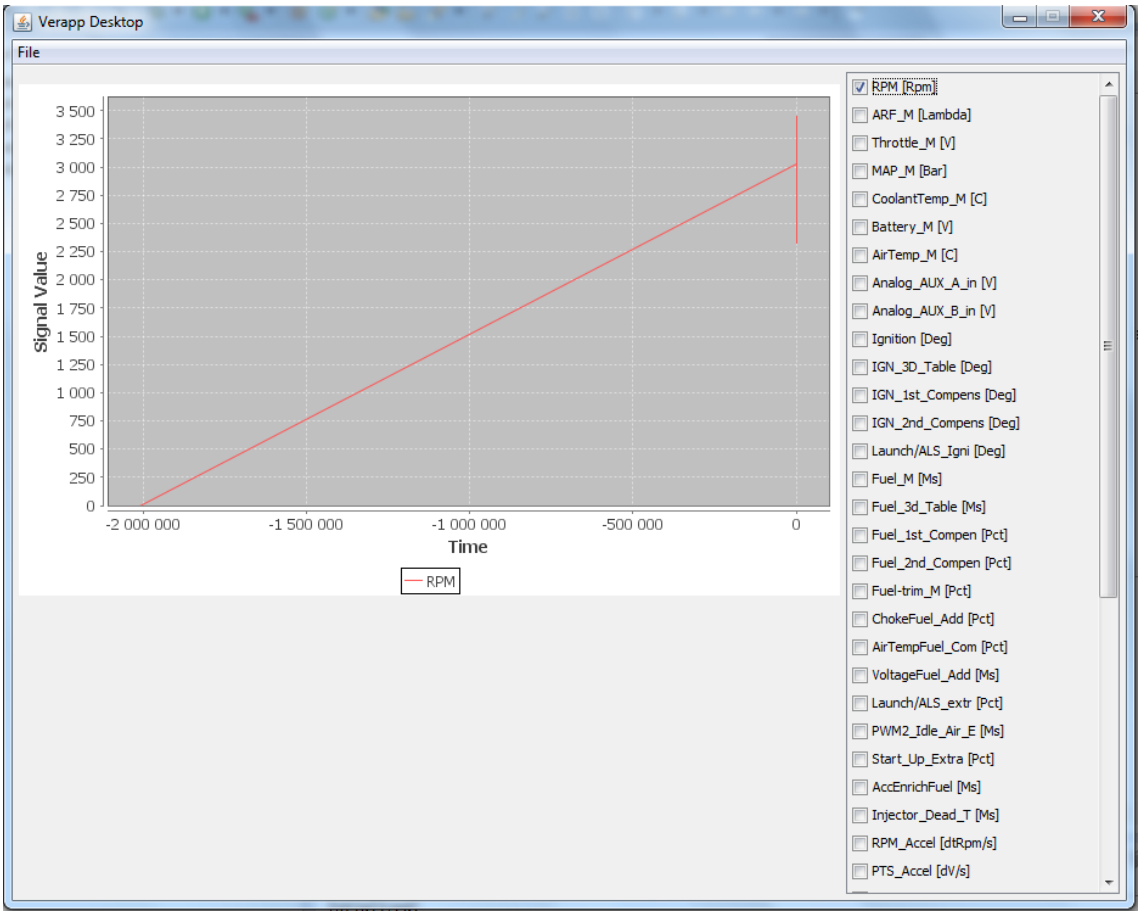


Figure 18: *Screenshot of desktop application. Note that this is from an early version, before logging was implemented correctly.*

6.1 GUI design process

While designing the GUI, paper prototyping was used fairly extensively. The goal was to create a very simplistic interface with minimal clutter. Civinco's old software was used as a model for the design with the aim of creating a familiar design that the Vera team wouldn't have any problems getting used to. The GUI is to be seen in figure 18.

6.2 Implementation

The implementation was split up into a few parts: connection to web server and collection of data, parsing of data and presentation of data using a graphical interface. The actual code was written with these fairly distinct parts in mind and has been split up into packages reflecting this fairly well.

6.2.1 Web connection and collection of data

Logged data is initially stored in a MySQL database located under our web host, however, as the host does not allow external queries to be made to their databases a PHP script was created to convert the database entries to an XML file for the desktop application to access.

Connection to the webserver is established using Java's native URL class which provides an easy-to-implement interface to web resources using the HTTP protocol. After a connection has been made a continuous stream of data is received and processed.

As soon as a connection has been established the application retrieves data at an interval specified by the user. A drawback of allowing the user to pick the interval is that the application is not able to find out how often data is uploaded to the server which could lead to the application retrieving the same data multiple times.

6.2.2 Reading existing logs

As a request from the Vera Team's the application also lets the user view data from previously saved logs. These are stored and parsed in the same manner as live recorded logs which allows the feature to be implemented with mainly existing code.

6.2.3 Parsing

The data, which is stored in XML format, is processed using the DOM model instead of the perhaps more commonly used SAX. The main difference between DOM and SAX is that DOM reads the entire document and loads it into the memory while SAX reads the document sequentially. Due to this SAX potentially has higher performance at a tradeoff of having a larger overhead. The decision to use DOM over SAX was made mainly due to it being easier to use and implement as well as the relatively small size of the documents in question.

Using the previously mentioned DOM the document is processed and split into individual data points which contain data about signal value, GPS coordinates as well as a timestamp of when the data was recorded.

6.2.4 Threading

The application was implemented using multiple threads to allow for real-time updates of the model and GUI as new data is logged. Not using threads would lock the application in terms of user interaction which is obviously not acceptable.

6.2.5 GUI

The GUI was implemented using the previously mentioned Swing framework as well as the external libraries JMapViewer as well as JFreeChart. Swing was used for creating the overall structure of the GUI as well as some of the graphical elements featured in the interface. In addition to this, JFreeChart is

used to create charts and JMapViewer is used to create visual maps showing the route taken by the vehicle.

The user is able to choose from a list of parameters corresponding to those logged in the ECU. When one is selected a chart and map is drawn containing all data points associated with the chosen parameter. The chart and map is dynamically updated as parameters are selected and deselected.

7 Discussion

The project has had a lot of things that have caused problems but most of them have been solved. Solving these issues has given new knowledge, but things that were not problems could in retrospect have been solved in a better way. In this section some of the problems encountered and lessons learned will be discussed in further detail.

7.1 ECU

When we first got the task of reading data from the ECU (engine control unit) we started to research information about how to use the Android operating system to set up a USB connection between the smartphone and ECU interface. The research resulted in information from the Android Developer Guide stated that USB host mode, a requirement to be able to establish a connection between two or more devices, was directly supported in the Android versions from 3.1 and above throughout the USB API library. This resulted in a decision to choose the reasonably cheap, small and light weight smartphone Sony Xperia Tipo which supported Android 3.1 with support of the host mode API.

When we received the phone and started the development of the USB connection communication, we soon realized that even though the Android operating system on the smartphone supported the USB host mode the smartphone's hardware configurations and kernel did not.

A long process of trying to reconfigure the kernel to support the USB host-mode was started. This was almost accomplished but not reliable since the configurations stopped working and refused the connections at seemingly random occasions while running. In the fourth week of the project we gave up the reprogramming of the kernel approach and started looking in to other smartphones, now at a higher price level, that supported both the Android version 3.1 and the hardware kernel configurations needed. This resulted in the Xperia Sola smartphone from Sony at twice the price.

This taking a lot of time to gain insight into resulted in a big delay of the whole project. As this was the key part and the main task of the project this affected all parts and group members in the project. When receiving the

Xperia Sola smartphone we were able to set up and initialize a communication with the ECU and start extracting log data. This log data have been generated from a test bench and is not from the actual Vera car's engine since the team have not yet installed the ECU box type we been working with, the Civinco SA3000 in the car.

When conquered the first big problem of the ECU communication the next problem occurred. The ECU boxes are designed to use the Civinco software program to collect data and request commands to and from the microprocessor inside the box. This resulting in very little information on how the communication protocol was formed and how to talk and interpret the data return from the ECU. We approached this problem by listen and analyze the traffic over the USB-bus with a USB protocol Analyzer box and tried to backward engineer the communication while using the Civinco software program and sending requests to the ECU.

This approach was abandoned due to the amount of data and control transfers sent at a high rate over the USB-bus where making it impossible to solve the problem this way. After consulting Markus Ekstrom from Civinco several times on the phone and getting good advices and help we managed to understand the communication protocol and the framework of the data-packages sent to and from the ECU. This resulting in a fully established communication between the smartphone and the Civinco SA3000 Engine Control Unit via the USB interface.

7.2 Sensors

The Chalmers Vera Team were interested in a new solution for collecting data from sensors. The data they had in mind fit well with VerApp and was appropriate to use in the GUI. It also meant that we were not as dependent on the use of GPS-signals to measure speed and lap time. The new sensor system that is installed does not weight much and is significantly smaller in size when comparing it to the system the team occasionally borrows when its time to compete. Another benefit is that the new system is installed to be flexible and adjustable to the team. The sensors are easy to move or detach from the microcontroller when not needed, and it is also possible to add new sensors.

Overall, it is important to have good communication with the customers at

the earliest possible stage and to be prepared either for them to change their preferences or that there may be misunderstandings about what is desired. It can be challenging to communicate when you have different technical background. Sometimes the purpose of a sensor was unclear, and you had to figure out what the idea was behind it all in order to translate it into a technical solution.

The sensor part of the project initially demanded a lot of time for research. There were difficulties in finding the sensors that were required. Some of these difficulties were due to that the right type of retailers are hard to find, their websites are tricky to navigate and it is difficult to distinguish which of the sensors' characteristics that are important for the context. At the same you have to keep in mind that the customer has to be satisfied and that the system may not become too expensive or difficult to install.

It was hard to find TPMS sensors that would fit smaller, tubeless tires which were sold separately and measure up to 7 bar. The sensors usually measure up to no more than 3 – 5 bar, which is suitable for an ordinary car. They are sold in packages of two or four and are virtually always supplied with a huge display, which make the systems both heavy and expensive. This would not be an ideal solution for the vehicle, as Vera has three tires and the data is presented through VerApp. An additional display would be too distracting for the driver and would contribute to unnecessary extra weight in the car. Although they are difficult to find, separate tire pressure sensors can be purchased in order to replace lost sensors. The model we found measured up to 13 bar. Unfortunately the sensors were never delivered by the retailer.

If we would have had an unlimited budget, we would not have had the need to spend as much time on research but had been able to buy the more available and expensive systems. We could have bought a costly TPMS system and disassembled it, we could also have bought more distance sensors, we would have acquired a triaxial accelerometer instead of soldering together the two two-axis accelerometers, we would have found encapsulations to all components, and we would probably have found a sensor to measure elongations in the chain tension spring instead of using the line sensor, since it is not its actual use.

7.3 Mobile application

The group members in general had no or very little experience of working with developing smartphone applications and Android programming. As result of this the group ran into many unforeseen problems. The following section will contain a description of the most significant problems encountered.

7.3.1 Graphical User Interface

When developing the application's graphical user interface it was important that the application would be easy to read while driving as well as easy to operate, even while wearing gloves. The phone to be placed in the car was a Sony Xperia Sola but since this phone was the only phone allowing host mode it was being used to study the communication with ECU and the development of the GUI therefore had to take place on other phones. The fact that other phones with a different display size were being used to develop the graphical design resulted in things that looked good on the phone used to develop the graphical design could sometimes end up looking bad on the Xperia Sola.

Initially a settings menu was added which was previously described. However, at the end of the project we came to the conclusion that choosing logging frequency was not a necessary feature. This decision was taken due to the fact that even when logging at the highest frequency there was such a small amount of data being sent that it did not cause any problems. For the user it is preferable to have as high logging frequency as possible and since it did not cause us any problem offering the user the highest logging frequency available we did not see any reason to give the user the possibility of choosing a lower logging frequency.

The steering indicator provides a graphical illustration of the steering value. In order to do that it needs to get a value from the sensor. Since it has taken some time to get the sensor in place and the data from it into the cell phone it has not yet been possible to test the indicator under its normal conditions.

7.3.2 Calculation of a new lap

As mentioned in the section 2.2.2 there have been two attempts to calculate the lap time whereas the first one did not work at all. Since the earth is not flat there is the mathematical part in which much time were spent. We believe that the first attempt required some very precise data from the GPS unit. and we have come to the conclusion that this could be the part that caused failure.

In the second attempt we narrowed down the solution to only watch if the car would be in range for the starting position. If that was the case, increase lap counter, grab lap time and lock the counter for 5 minutes. The margin of error might sound much with 10 meters to spare but for this app it is acceptable. Let's say that the car is travelling in 25 km/h, which is the minimum average speed through the race and the new lap code is triggered on the edge of its maximum (10 meters from starting point). This scenario would give an incorrect value of 1,44 seconds. Since Shell Eco Marathon is not about getting the best time, the faulty of 1,44 seconds is acceptable.

7.3.3 Heap size error

Problem was encountered when trying to send the data from the application to the web server. As mentioned in section 2.2.7 several attempts were made. The first one was implemented with a text file that kept all names of files that have been sent and for each new transfer the class would check whether the file had been sent before or not. This caused the heap to run out of memory. The `execute` method was run from an infinite thread. As the application was running, hundreds of zip-files were created and the class would list the whole folder repeatedly. At last the memory did not stand a chance, which caused this error. The current solution can be read from the same section as the second attempt.

7.4 Database transfer

The database, and all transfers to and from it has caused a lot of problems during the entire project. To begin with, we chose a hosting service that did not allow an external connection to their database server. That is something

that we should have confirmed before choosing one.com as the service to go with. We spent a lot of time developing an SQL-parser in Java that ended up being discarded.

Another big issue concerning the database transfer was that no clear decision on how it was supposed to be done was made. It was not up for discussion using a zip file and a zip parser during the first half of the project time but at the end we came to the conclusion that this would be the best option. Some more research should have been performed at the beginning of the project and a clear path should have been chosen. If that would have been the case, problems such as zip-parsing and data-conversion, could have been solved earlier.

7.5 Desktop application

While the development of the desktop application was perhaps the smoothest, it did have its share of problems. These did, however mainly come from other parts of the project being held up for various reason which meant that the desktop application was not able to be sufficiently tested early enough.

7.5.1 Lack of testdata

One major problem faced during development was a lack of real test data. This occurred due to data collection from the ECU becoming a bottleneck. This problem propagated to other parts of the project as testing was severely delayed. Throughout the project the desktop application was solely tested using dummy data which was manually inserted into the database using the web host's own tools. While this was sufficient for making sure the XML parsing worked correctly it did not allow for testing of real-time logging. This lack of test data pushed the testing into the very end of the development process which was problematic as this could be seen as one of the very core features of the system.

7.5.2 GPS Plotting

Restrictions in the API used for drawing maps (JMapViewer) led to the plotting of GPS coordinates ultimately being cut from the application. The plan was to use the GPS coordinates to allow the team to compare signal values not only to specific points in time, but also to geographical location. In the end, as stated previously, the API used did not support this in a satisfactory way leading to the feature not being fully implemented. There is however quite a bit of code written which means that it would probably be fairly easy to either implement without an external API or if API is found that better supports the needs.

7.5.3 Warning flags

Since GPS Plotting was cut, we decided to cut the warning flags as well. The flags were supposed to be connected to each data point but since we could not figure out a smart way to plot the points there was no point for this. Another reason for this idea being cut is the fact that we chose to plot the data points in a chart. If we would have done some kind of list instead, this could have easily been implemented. Although, we chose to send the warning information all the way to the desktop application. This means that no other changes, except from GUI changes, have to be done to implement it in the future. Both we and the Vera team wanted this feature so it is a shame that we did not find any good ways to implement it.

7.6 System in total

The project has consisted of many parts which have been developed rather independently but at the same time strongly depend on each other: the ECU (engine control unit), the DAQ (Data Acquisition System), the mobile application, the web server and the desktop application. The system can be illustrated with figure 19. To the far left in the picture are the engine and the sensors. The data goes into the ECU and the data from the sensors goes into an Arduino micro controller. This far the data from the engine and the DAQ has been transmitted in two parallel processes but thereafter the information from both systems converges in the smartphone. The ECU communicates

with the smartphone via a USB wire and the Arduino communicates via Bluetooth.

In the phone, the data from the ECU, the data from the Arduino and the data from the smartphone including e.g. GPS coordinates are written to a text file which is compressed and sent to the web server. On the web server the file gets uncompressed and the data from the ECU and is recalculated to physical values before the data is once again written to a text file. Finally the new text file can be fetched by the desktop application which presents the data in relevant graphs where the user can choose which parameters to display.

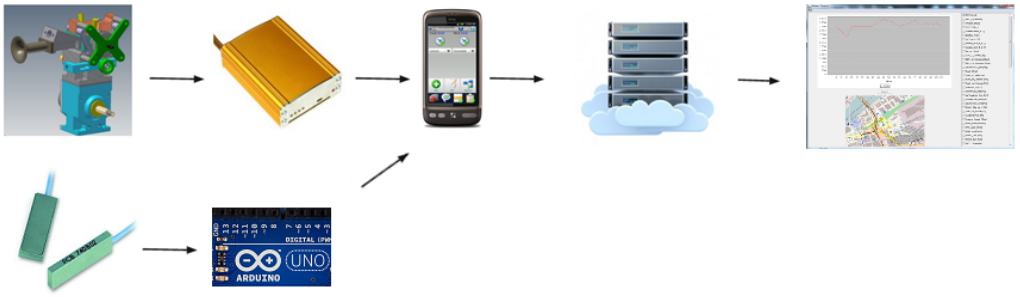


Figure 19: *Overview of the entire system*

Since the goal of the system is to transfer data through the system, all parts of the system are depending on a continuous flow of data to be thoroughly tested. However, all parts of the system have been developed in parallel which means the testing has been done using test data without continuous updating.

Under optimal circumstances the work with ECU would have been finished first so that it could provide the system with data to do proper testing. In reality this has not been the case, there have been a lot of difficulties extracting data from the ECU and this has been a bottleneck for the entire system. The system also has a weakness in the fact that both intermediate parts between the ECU and desktop application are transforming the data in some way: the mobile application writes it to a text file which is being compressed while the web server decompressed the file and converts the data into physical values. This means that shortcuts can never be used while developing.

It might seem like having two parallel processes in the data transfer process

makes it more complicated but in fact it is a strength considering the robustness of the system. Since the systems are parallel, if the sensors in the engine or the logging in the ECU were to stop working, the team would still get information from the DAQ and vice verse in case the DAQ fails.

In retrospect if we would have started developing the DAQ at the same time as the rest of the system, then having problem with the acquiring data from the ECU it could have been possible to let the DAQ supply the system with data and then have done proper testing earlier on the rest of the system.

8 Conclusion

This section will provide a conclusion of how well the task was performed and to what degree the task was fulfilled. We will look back to investigate whether the objective and task of the project has been solved. We will also look forward at how this work continued.

To begin with, this project has grown immensely. From the beginning all the customer (Chalmers Vera Team) requested was the possibility to log data in the engine and send it wireless to a computer which displays it in real-time. Today we are also giving the customer the opportunity of logging data from other sensors in the car and we provide the driver with relevant information while driving.

As mentioned in section 1.2 the purpose of the project is to develop a system that handles the communication between the Vera-team and the engine. The data shall also be presented in real-time. The objective of the project in currently partly fulfilled. Data from the engine can be logged and sent. The problem is that the real-time update on the computer does not work properly. The reason for this is unknown and needs to be investigated further.

In the objective it is also stated that in case we do not fulfill our purpose entirely the most important thing is that the software on the smartphone can store data and then transfer it to the computer using the already existing software. The data logged in the cellphone can be displayed in the existing program by manually moving the files to the computer after the race.

It might seem strange that the project was extended when the original task could not be fulfilled. The reason that the objective was not fulfilled is that the ECU(engine control unit) was causing a lot of problems. It is doubtful that we could have been more persons working on the ECU if we would have known from the beginning that it was going to be so difficult getting the communication between the ECU and smartphone to work. But if we would have known it would take such a long time getting the communication to work we would probably have developed a simulator or some other way to test the system. That way we could have found earlier that things did not work properly at the end of the system.

8.1 Possible future developments

During development, several features that were originally planned to be included in the system had to be cut. This section will discuss a few in terms of their feasibility and how they could be implemented in the future.

8.1.1 Plotting GPS coordinates

As previously stated, this was meant to have been implemented in the final product, but due to API limitations it was cut. The plan was to plot GPS coordinates on a map and allow the user to read recorded signal values at a specific geographical point. One thought was to draw the coordinates on a map and allow the user to read signal values by mousing over the points, but this would require the structure of the XML documents to be adjusted as there would otherwise be multiple points at the same coordinate, one for each signal. This is due to the fact that data points are stored by their signal, not by time stamp, meaning that for each time unit there would be a number of data points equal to the number of available signals.

8.1.2 Remotely modifying engine parameters

This feature was raised as an extra extension early in the project by the Vera team. This would however possibly introduce several new problems into the system. For one, additional attention would have to be paid to thread safety in the system, as sending values outside of the recommended range could cause damage to the engine. If the threading is not sufficiently secure (values are read and written in incorrect order), this could cause the application to send other values than those input by the user.

8.1.3 Additional sensors

Wireless tire pressure sensors would be of great value to the Vera team. If the tires were to loose too much pressure it would affect the performance of the car and the expensive carbon fibre rims would be run down. A tire pressure system could alert the driver when the pressure drops.

It is also of interest to know in what way the air flows around the vehicle and inside the wheel houses. It would be valuable if the team could lower the impact of air resistance. If sensors inside the wheel houses could expose where the pressure is at its highest, the team could come up with solutions of how to distribute the pressure evenly. This would get rid of unwanted suction between the vehicle and the ground.

8.1.4 Warning flag

Much like the GPS plotting this was supposed to be implemented in the final product. Due to the fact that we could not implement GPS coordinates this left out. Without the coordinates to point out on map where the car was when the flag was set we did not see any point in implementing this. Also, we did not figure out any smart way to connect each error with a data point in our data output. Mostly because of the chosen diagram API. Although, if a smart way is figured out in the future, we implemented it all the way until the desktop application. If an error is sent, it is connected to a data point all the way until the data is presented to the user.

Reference list

Android Developer Guide (2013a) *Design Principles*.
http://developer.android.com/design/get-started/principles.html.
(9 May, 2013)

Android Developer Guide (2013b) *LocationManager*.
http://developer.android.com/reference/android/location/
LocationManager.html.
(23 April, 2013)

Android Developer Guide (2013d) *Settings*.
http://developer.android.com/design/patterns/settings.html.
(19 April, 2013)

Android Developer Guide (2013e) *Menus*.
http://developer.android.com/guide/topics/ui/menus.html.
(19 April, 2013)

Android Developer Guide (2013e) *Iconography*.
http://developer.android.com/design/style/iconography.html.
(21 April, 2013)

Android Developer Guide (2013f) *Launcher Icons*.
http://developer.android.com/guide/practices/ui_guidelines/
icon_design_launcher.html.
(21 April, 2013)

Android Developer Guide (2013g) *Location*.
http://developer.android.com/reference/android/location/Location.html.
(24 April, 2013)

Android Developer Guide (2013h) *AsyncTask*.
http://developer.android.com/reference/android/os/AsyncTask.html.
(16 Maj, 2013)

Android Developer Guide (2013i) *USB Host and Accessory*.
http://developer.android.com/guide/topics/connectivity/usb/index.html.
(16 Maj, 2013)

Android Developer Guide (2013j) *Bluetooth*.
http://developer.android.com/guide/topics/connectivity/bluetooth.html.
(16 Maj, 2013)

Boghard et al. (2008) *Arbete och teknik på människans villkor*. 1:1.
Stockholm: Prevent.

Garmin (2013) *What is GPS?*.
http://www8.garmin.com/aboutGPS.
(22 April, 2013)

Illinois Institute of Technology (2013) *dvantages of SQL*.
http://www.cs.iit.edu/ cs561/cs425/VenkatashSQLIntro/
Advantages%20&%20Disadvantages.html.
(18 April, 2013)

Mark A. Weiss (2005) *(*Data Structures and Algorithm Analysis in Java) 2.
Pearson.

Shell Eco-Marathon (2013) *Shell Eco-Maraton Europe 2013, Official Rules Chapter 2*. http://s09.static-shell.com/content/dam/shell-new/local/
corporate/ecomarathon/downloads/pdf/europe/
sem-europe-official-rules-chapter-2-180413.pdf. p.11.
(12 May, 2013)

Sony Mobile (2013) *Xperia sola*.
http://www.sonymobile.com/global-en/products/phones/xperia-sola/
specifications.
(22 April, 2013)

Tech2 (2013) *What is A-GPS? How Does it Work?*.
http://tech2.in.com/features/all/what-is-agps-how-does-it-work/115142.
(22 April, 2013)

TPMS Sweden (2013a) *Vad är TPMS?*.
http://www.tpms-swe.se/tpms/what-is-tpms.
(17 April, 2013)

TPMS Sweden (2013a) *Motorcykel TPMS (M202)*.
http://www.tpms-swe.se/eftermarknadsprodkter/motorcykel-tpms.
(18 April, 2013)

A Documentation of software

The following documentation has been done:

1 Requirements VerApp

2 RAD (Requirements and Analysis Document) VerApp

3 RAD (Requirements and Analysis Document) VerApp Desktop

4 Use Cases VerApp

Visible requirements:

ID 1.01: Start application
ID 1.02: Present data to the driver (speed, lap time, total time)
ID 1:03: The driver can report to the team if something occurs.
ID 1.04: Exit the application

System requirements:

ID 2.01: Logging the car position
ID 2.02: Connecting the position data to the engine data
ID 2.03: Present engine data and GPS position on the computer
ID 2.04: Logging of engine data from ECU
ID 2.05: Data from the current run saved onto the cellphone
ID 2.06: Possible logging data from DAQ
ID 2.07: Stop logging data
ID 2.08: Sending engine data and position data through 3G
ID 2.09: Sending engine data and position data through Wi-Fi

Visible requirements:

Start application

ID 1.01
Use case / scenario:
Start the application.
Trigger:
The user launches the app.
Precondition:
The application is installed.
Basic path:
The main screen is shown to the user.
Exception path:
N/A
Post condition:
The main screen is shown to the user.

Present data to the driver (speed, lap time, total time)

ID 1.02
Use case / scenario: Present speed, average speed, lap time, total time of run to driver.
Trigger: Startbutton on application
Precondition: Timer-, GPS-function is started.
Basic path: The application is using GPS-data and timer to calculate speed, average speed, lap time and total time. Present data on main-screen of application.
Exception path: Restart and try again.
Post condition: Saving data and calculating speed while running.

The driver can report to the team if something occurs.

ID 1.03

Use case / scenario:

By a button the driver can notify the team that something has occurred.

Trigger:

Action button

Precondition:

Application is started.

Basic path:

Stores location and time when the event occurred.

Exception path:

Saved to phone storage. Sent when connection is available.

Post condition:

Data is either sent or stored. Confirmation message should be displayed for a couple of seconds.

Exit the application

ID 1.04

Use case / scenario:

Exit application

Trigger:

Exit button on application.

Precondition:

Application is running and is not logging data.

Basic path:

The precondition is fulfilled and exit button is pressed.

Exception path:

Application is facing an unexpected shutdown.

Post condition:

The logging of data is stopped and the application shut down

System requirements:

Logging the car position

ID 2.01

Use case / scenario:

The application is saving the GPS-data given by the GPS.

Trigger:

The logging of GPS data starts then the start-/resume botton is pressed

Precondition:

The application is started. The GPS is activated.

Basic path:

The GPS-data is saved onto the cell phone.

Exception path:
N/A
Post condition:
There is GPS data available on the cell phone.

Connecting the position data to the engine data

ID 2.02
Use case / scenario:
The position data och engine data sent to the application are combined.
Trigger:
Continuosly when data are availiable
Precondition:
Application is started and data are availiable
Basic path:
The data are associated with each other..
Exception path:
N/A
Post condition:
The position data and motor control data are connected

Present engine data and GPS position on the computer

ID 2.03
Use case / scenario:
Trigger:
Precondition:
Data has been acquired from the web server and is stored on the computer.
Basic path:
Exception path:
N/A
Post condition:

Logging of engine data from ECU

ID 2.04
Use case / scenario:
Read and save bit-data from USB-bus.
Trigger:
Then the start- /resume button is pressed
Precondition:
USB connected, ECU-bus initialized.
Basic path:
Setup and initialize start register in ECU, send start command to ECU.
Exception path:
Restart setup.
Post condition:

Bit data is read from the bus.

Data from the current run saved onto the cellphone

ID 2.05
Use case / scenario: Log-data is saved to cellphone while data logging is running.
Trigger: Start when data logging is started.
Precondition: Data logging is started.
Basic path: Read input stream from USB-bus and write to a textfile
Exception path: Log exception and restart write-process.
Post condition: Data is written to text file which is saved on the phone

Logging data from DAQ

ID 2.06
Use case / scenario: Receive data from DAQ-sensors to ECU and logg the data in the application on the phone.
Trigger: Start-/ resume button is being pressed
Precondition: DAQ-sensors are started and connection established to the ECU by an arduino.
Basic path: Data sent from DAQ to the application through the arduino and the ECU.
Exception path: N/A
Post condition: The data from the DAQ is saved onto the phone

Stop logging data

ID 2.07
Use case / scenario:
The application stops to log data.
Trigger:
User clicks the stop button.
Precondition:
Application is logging data.
Basic path:
Application is logging data and the user clicks stop button. The app stops the data logging.
Exception path:
N/A
Post condition:
Data logging is turned off.

Sending logged data and position data through 3G

ID 2.08
Use case / scenario:
Logged data with its position is sent to web server through 3G.
Trigger:

Sent regularly with predefined interval when the logging is started.
Precondition:
Application is started and logging is activated.
Basic path:
Data is sent to web server.
Exception path:
If no 3G connection is available, the data is stored and sent when connection is available.
Post condition:
Data is sent.

Sending logged data and position data through Wi-Fi

ID 2.09
Use case / scenario:
Logged data with it's position is sent to web server through Wi-Fi.
Trigger:
Sent regularly with predefined interval when the logging is started.
Precondition:
Application is started and logging is activated.
Basic path:
Data is sent to web server.
Exception path:
If no Wi-Fi connection is available, the data is stored and sent through 3G.
Post condition:
Data is sent.

RAD VerApp

Requirements and Analysis Document for "VerApp"

Table of Contents

Version: 1.0.1

Date: 2013-05-19

This version overrides all previous versions.

1 Introduction

Since the ECU can not communicate directly with a computer wirelessly this project is going to have a smartphone which is connected to the ECU and then handles further communication. To make this possible an application is going to be developed. The phone will also work as a dashboard for the driver.

1.1 Purpose of application

The purpose of the application is to make it possible for the ECU to communicate wirelessly with a computer. The application shall acquire data from the ECU and forward it. The application shall also present important information to the driver.

1.2 General characteristics of application

Our aim is an application that will run smoothly on the phones chosen for this project. The finished product will at least contain the possibility to log data from the ECU and a dashboard.

1.3 Scope of application

The final goal is that we can provide real time logging of data and present this to the team, but in case of lack of time and/or possibility to present data in a simple and smooth way, the most important thing is the possibility of logging data.

1.4 Objectives and success criteria of the project

The objective of the application is to provide the driver with information about total run time, average speed, lap time, steering and speed. The application shall also handle all communication with the ECU and send the data received from the ECU to a web server.

The minimum criteria for success is that the application is able to retrieve data from the ECU and then either present it on the cell phone or upload it on the web server.

1.5 Definitions, acronyms and abbreviations

ECU - Engine Controller Unit, placed inside the car.
App - Android Application
GUI - Graphical user interface
Google Play - Google's official platform for selling Android-applications
DAQ - Data Acquisition system

2 Proposed application

Based on the information provided above. There will be presented an application which is providing a solution.

2.1 Overview

The goal of VerApp is to provide a graphical user interface, to initiate the communication with the ECU and to save and forward data received from the ECU.

2.2 Functional requirements

- Logging of engine data from ECU
- Sending engine data and position data through 3G
- Sending engine data and position data through Wi-Fi
- Logging of car's position

- Associate the position data with the engine data
- Data from the current run saved onto the smartphone
- The driver can report to the team if something occurs during the run. (graphic interface for the vehicle's position in real time)
- Exit the app
- Reset session
- Possible logging data from DAQ
- Present data to the driver (speed, average speed, lap time, total time, steering)

2.3 Non-functional requirements

- Data is transferred with expected time resolution
- Data can be saved onto the cell phone until it can be transferred onto a computer
- Data shall be transferred in a cost efficient manner
- Data shall be transferred in a way that minimizes the risk of data loss

2.3.1 Usability

The application shall be used by a person in Chalmers Vera Team. It is important that the application can be handled by a person wearing gloves which means that there can not be any button that are small and any keyboard interaction.

2.3.2 Reliability

As the application relies on having either a 3G- or Wifi-network available some care will have to be put into a solution for when neither of these is available. This will be done be saving all data on the smartphone until it can be uploaded to the web server.

Apart from this, there aren't many reliability issues at hand as the application is rather simple in addition to the fact that the phone used will likely not be required to handle any other tasks while running the application.

2.3.3 Performance

Since the application will be pretty light, the only performance issues could be with data transfer. Our goal is that data should be transferred in a safe and controlled way with as little data loss as possible.

2.3.4 Supportability

The application will be developed for Android 4.0 and newer. Due to this, phones having older OS will not be able to use the app. The application is also somewhat customized to the specific ECU-model SA3000 from Civinco.

2.3.5 Implementation

The application will be implemented with Java, using version 4.0 of the Android OS, Ice Cream Sandwich (ICS).

2.3.6 Verification

The application will be verified in collaboration with the Chalmers Vera team which will, at least early on, be the application's primary user.

2.3.7 Packaging and installation

Due to the tight relationship with the ECU-provider Civinco the application may not be available on Google

Play but rather as a .apk-file which Android-phones are able to install with some simple configuration.

2.3.8 Legal

Our goal is to own everything by ourselves and not use any third party applications during the development of this app.

RAD VerApp Desktop

Requirements and Analysis Document for "VerApp Desktop"

Table of Contents

Version: 1.0.0

Date: 2013-02-05

This version overrides all previous versions.

1 Introduction

Since Android does not support handling an external database, the mobile application sends the data to a web server and from there the data will need to be fetched. Since the desktop application developed by Civinco (the manufacturers of the ECU) does not support real time update of data a desktop application needed to be developed.

1.1 Purpose of application
The purpose of the application is to acquire data from the web server and then present it in a manner more suitable for a group of people.

1.2 General characteristics of application
The application runs smoothly on an arbitrarily chosen PC. It provides a graphical user interface where data is presented to the user in the form of a chart.

1.3 Scope of application
The application fetches the data from the web server and presents it to the user. The current version does not support changing of engine parameters in the ECU. Neither does it provide the user with the possibility of seeing on a map where the data was logged.

1.4 Objectives and success criteria of the project
The objective of the application is to provide the team with data from the ECU and DAQ and present the data to the race team in a relevant manner through a chart.

The minimum criteria is that the data is presented to the user. Displaying a map is a desirable feature.

1.5 Definitions, acronyms and abbreviations
ECU - Engine Controller Unit
App - Android Application
GUI - Graphical user interface
DAQ - Data Acquisition system

2 Proposed application
Based on the information provided above. There will now be presented an application which is providing a solution.

2.1 Overview
The desktop application shall retrieve data from web server and present it to the user in a relevant way.

2.2 Functional requirements

- Retrieving data from web server via 3G
- Retrieving data from web server via WiFi
- Present data to user
- Present relevant graphs

2.3 Non-functional requirements

- Data shall be transferred with expected time resolution
- Data shall be transferred in a cost efficient manner
- Data shall be transferred in a way that minimizes the risk of data loss
- It is possible to call while data continues to be logged

2.3.1 Usability

When designing the GUI care was taken to make it sufficiently similar to the one currently used as to ease the transition as well as creating a very easy-to-use interface with minimal clutter.

2.3.2 Reliability

As the application relies on having either a 3G- or Wifi-network available some care will have to be put into a solution for when neither of these is available. The data will be saved onto the web server until there is 3G- or WiFi-connection available to enable the desktop application can fetch it.

2.3.3 Performance

Since the application will be pretty light, the only performance issues could be with data transfer. Our goal is that data should be transferred in a safe and controlled way with as little data loss as possible.

2.3.4 Supportability

The application will be developed for Android 4.0 and newer. Due to this, older phones will not be able to use the app.

2.3.5 Implementation

The application will be implemented in Java, using a few external libraries. In addition, some PHP will be used for supporting functions.

2.3.6 Verification

The application will be verified in collaboration with the Chalmers Vera team which will, at least early on, be the application's primary user.

2.3.8 Legal

The application will not use any non-open source code.

2.4 Test Cases

Use Cases VerApp

ID: 1
Name: Install application
Includes: -
Actor: User
Goal: To install the application.
Description: Installation file is transferred to the phone and is initiated.

ID: 2
Name: Start application
Includes: Install application
Actor: User
Goal: Start application
Description: Prepare the application for using

ID: 3
Name: Enable GPS
Includes: Start application
Actor: User
Goal: Enable GPS
Description: Enable GPS before starting run mode.

ID: 4
Name: Start run mode
Includes: Start application, Enable GPS
Actor: User
Goal: To start the run mode.
Description: User presses the start button when GPS signal has been allocated.

ID: 5
Name: View values
Includes: Start run mode
Actor: -
Goal: Present speed, average speed, steering, total time and lap times.
Description: Display values on the screen in run mode.

ID: 6
Name: Report incident
Includes: Start run mode.
Actor: User
Goal: Set warning flag.
Description: Flag later viewed together with logged data

ID: 7
Name: Press stop button
Includes: Start run mode.
Actor: User

Goal: Pause logging and race values. Show possibility to end session.
Description: Pause logging and race values. Shows a summarization of the race together with the option of ending the session.

ID: 8
Name: End session
Includes: Press stop button
Actor: User
Goal: Go back to the start screen.
Description: Go back to the start screen and end the current session.

ID: 9
Name: Transfer data
Includes: -
Actor: -
Goal: To transfer loggdata from phone to computer through HTTP protocol.
Description: Transfer collected data from the phone to computer.

ID: 10
Name: Reset application
Includes: -
Actor: User
Goal: Go back to initial screen
Description: Close current session and send user to initial start view. All values are reset.

ID: 11
Name: Exit application
Includes: -
Actor: User
Goal: To exit the app.
Description: User clicks exit button. App with all related threads are shutdown.

B Data sheets

The following data sheets are included:

1 Michelin 45-75R16 RADIAL 2013

2 Hamlin 59025-1-S-02-A

3 Wabash 971 RPS

4 Impaqed Products BV TPMS 10.02.011

5 Radiotronix RCR-433-MPR

6 SpectraSymbol Flex Sensor Rev A1

7 Sharp GP2Y0A41SK0F

8 Memsic 2125 28017

9 IC Haus LFL 1402

10 Free2Move F2M03GLA

MICHELIN TIRES AND RIMS CARACTERISTICS
Prototype vehicles

— TECHNICAL SPECIFICATIONS —

Maximum pressure:	**7 bars (700 kPa)**
Load capacity:	**100 kg**
Speed limit:	**70 km/h**
Recommended rim size:	**1.20 J16 - 1.35 J16**
Electrical resistance:	**> 3 E+10 Ω**

— TIRE IDENTIFICATION (SIDEWALL MARKINGS) —

Manufacturing week/year : **WW13** **FOR COMPETITION PURPOSE ONLY**

Only tires with above markings, manufactured in 2013, are certified conform to this datasheet.

— TIRE DIMENSIONS —

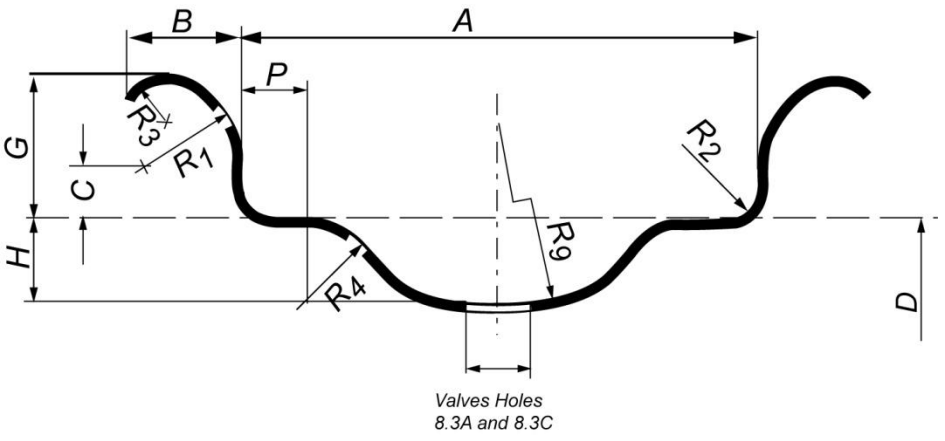
TIRE SIZE DIMENSION	Section width mm	Overall Diameter mm
45/75 R16	**45**	**478**

Theoretical dimensions depending on pressure and rim

— RIM DIMENSIONS —

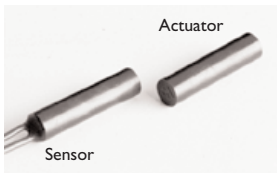
"drop center rim with cylindrical bead seats"

WIDTH CODE	DIMENSIONS (mm)													
	A	B		G	P	H	C	R1	R2	R3	R4	R9	D	πd
	+0,15 -0,5	min.	max.	±0,5	+0,2 -0	+1,0 -0,5	ref.	ref.	max.	min.	min.	min.		+2,0 -0,5
1,20	30,5	5,5	7,5	9	3	7	3,5	6	1,5	1,5	5	7,0	405,6	1274,2
1,35	***34,0***	***6,5***	***8,5***	***10***	***3,5***	***7,5***	***4***	***6,5***	***1,5***	***2***	***5***	***7,0***	***405,6***	***1274,2***



File E61760(N)

59025 Firecracker Features and Benefits



Features

- 2 part magnetically operated proximity sensor
- UL recognised
- Choice of normally open, normally closed or change over contacts
- Customer defined sensitivity
- Choice of cable length and connector

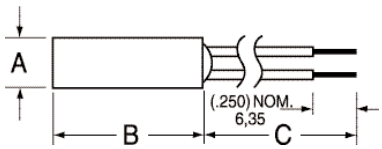
Benefits

- Quick and easy to install
- No standby power requirement
- Operates through non-ferrous materials such as wood, plastic or aluminium
- Hermetically sealed, magnetically operated contacts continue to operate long after optical and other technologies fail due to contamination

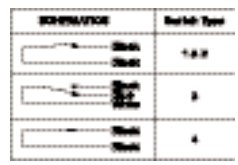
Applications

- Position and limit sensing
- Security
- Level sensing
- Linear actuators

DIMENSIONS (in) mm



	A Max	B Max	C NOM.
57025	(.245) 6,22	(1.000) 25,40	-
59025	(.245) 6,22	(1.000) 25,40	Table 3 ± (.393) 10,00



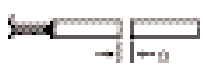
CUSTOMER OPTIONS - Switching Specifications

TABLE 1 Contact Type			Normally Open	Normally Open High Voltage	Change Over	Normally Closed
Switch Type			1	2	3	4
	Power	Watt - max.	10	10	5	5
Voltage	Switching	Vdc - max.	200	300	175	175
	Breakdown	Vdc - min.	250	450	200	200
Current	Switching	A - max.	0.5	0.5	0.25	0.25
	Carry	A - max.	1.2	1.5	1.5	1.5
Resistance	Contact, Initial	Ω - max.	0.2	0.2	0.2	0.2
	Insulation	Ω - min.	10^{10}	10^{10}	10^{9}	10^{9}
Capacitance	Contact	pF - typ.	0.3	0.2	0.3	0.3
Temperature	Operating	°C	-40 to +105	-20 to +105	-40 to +105	-40 to +105
	Storage	°C	-65 to +105	-65 to +105	-65 to +105	-65 to +105
Time	Operate	ms - max.	1.0	1.0	3.0	3.0
	Release	ms - max.	1.0	1.0	3.0	3.0
Shock	11ms 1/2 sine	G - max.	100	100	50	50
Vibration	50-2000 Hz	G - max.	30	30	30	30

CUSTOMER OPTIONS - Sensitivity, Cable Length and Termination Specification

TABLE 2

Sensitivity Options:-
Activate Distances are approximate using Hamlin 57025 actuator as illustrated
Switch AT before modification



Select Option		S		T		U		V	
Switch Type		Pull In AT Range	Activate Distance d (in) mm	Pull In AT Range	Activate Distance d (in) mm	Pull In AT Range	Activate Distance d (in) mm	Pull In AT Range	Activate Distance d (in) mm
1	Normally Open	12-18	(.453) 11.5	17-23	(.374) 9.5	22-28	(.315) 8.0	27-33	(.295) 7.5
2	High Voltage								
3	Change Over	15-20	(4.13) 10.5	20-25	(.354) 9.0	25-30	(.295) 7.5		
4	Normally Closed								

TABLE 3

Cable Type:-
24 AWG 7/32 PVC 105°C UL1430/UL1569

Standard Lengths

SELECT OPTION	CABLE LENGTH (in) mm
01	(3,94) 100
02	(11,81) 300
03	(19,69) 500
04	(29,53) 750
05	(39,37) 1000

TABLE 4

Termination Options:-

SELECT OPTION	DESCRIPTION (2 WIRE VERSIONS ILLUSTRATED)	
A	Tinned leads	
B	Crimped terminals	
C	6.35mm fastons	
D	AMP MTE 2.54mm pitch	
E	JST XHP 2.5mm pitch	

ORDERING INFORMATION

59025 – X – X – XX – X

- Series 59025
- Switch Type — Table 1
- Sensitivity — Table 2
- Cable Length — Table 3
- Termination — Table 4

Hamlin USA Tel: +1 920 648 3000 • Fax: +1 920 648 3001 • Email: sales.us@hamlin.com
Hamlin UK Tel: +44 (0)1379 649700 • Fax: +44 (0)1379 649702 • Email: sales.uk@hamlin.com
Hamlin Germany Tel: +49 (0) 6181 953660 • Fax: +49 (0) 6181 9536666 • Email: sales.de@hamlin.com
Hamectrol France Tel: +33 (0) 1 4687 0202 • Fax: +33 (0) 1 4686 6786 • Email: sales.fr@hamlin.com

ISSUE No: 4 DATE: 1/5/3

Wabash 971 RPS

Rotary Position Sensor

Designed to operate in demanding environments where long life and high performance is required.

The Wabash 971 Rotary Position Sensor (RPS) uses high performance conductive polymer tracks and contact designs to achieve 2 percent independent linearity. Reliable and versatile, it is ideal for applications such as:

- Electric industrial vehicles
- Off-road steering and transmission
- Engine management and controls
- Recreational vehicles
- Agricultural equipment

The RPS can be supplied with options for electrical track length, resistance, flying leads, integral connector and actuator drive direction. Its rugged design can withstand:

- Temperature extremes
- Harsh environments
- Mechanical shock and vibration

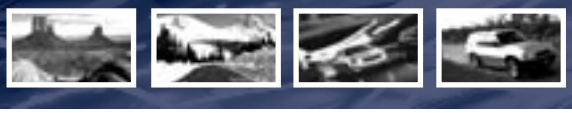
Wabash generic sensors offer customers low cost options with minimal or little tooling investment.

Count on Wabash Technologies for sensing solutions that add performance and value to products. We serve customers with advanced design and engineering capabilities, flawless quality performance, flexible manufacturing and on-time delivery.

To learn more about how our products can help you, contact us at 260-355-4100 or visit www.wabashtech.com

Wabash TECHNOLOGIES

Committed to sensor advancement.



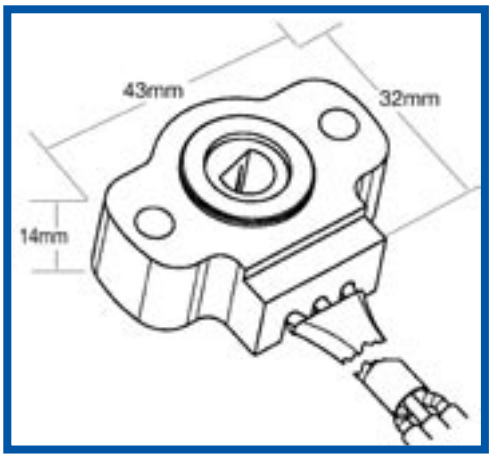
The Wabash 971 RPS

Rotary Position Sensor

Technical Specifications

PHYSICAL

- Fully "sealed" robust package suitable for automotive, agricultural, marine and industrial environments
- High performance, compact potentiometric sensor
- Suitable for arduous engine management and closed loop control system feedback applications
- Through-hole actuation capability
- Integral connector or flying lead versions
- Actuator and mounting configuration options



ELECTRICAL

	Specifications
Track Resistance (R†)	5KΩ ±20% @ 20°C ±10°C
Linearity (Independent)	±2%
Index Point	3% ±2% @ -Low End Stop
Output Gradient	0.973%/° Max 0.873%/° Min
Power Rating	I Watt @ 40°C Derated to Zero @ 135°C
Temperature Coefficient	±600ppm/°C
Insulation Resistance	1000MΩ Min 500V DC
Maximum Voltage	13.5V DC

MECHANICAL

	Specifications
Rotation	128° ±2°
Spring Torque	Min Return - 6 Nmm Max Wind up - 120 Nmm
Mechanical End Stop Strength	680Nmm Min
Fixing Torque (M4 and Washer)	2Nm Maximum
Lead Wire Version 971 - 0002	16/0.2, 0.5 mm CSA 1.8 mm OD Pull Strength: 10 kg Max (all 3 wires)
Connector Version 971 - 0001	Connector to M47/2 AMP P/N 828748 - 3

PERFORMANCE & ENVIRONMENTAL

	Specifications
Rotation Life (- 40° - +130°C)	5, 000, 000 Full Cycles 10, 800, 000 Dither Cycles (2°)
Functional Temperature Range	-40°C to +85°C Wire Version -40°C to +155°C Connector Version
Mechanical Shock (Handling)	Im Drop onto Concrete Floor
Mechanical Shock (Bump)	1000 40 g 11 ms Shocks (3 axis)
Vibration (Sinusoidal)	10 - 57 Hz @ 1 mm Displacement 57 - 100 Hz @ 10 g 100 - 500 Hz @ 27 g
Sealing	IP 5X (dust)
Pressure Wash	90 Bar 0.5-0.6 m 5-6 Seconds
Humidity	40°C 96% RH 504 Hours
Chemical Resistance	Screen Wash, Gearbox Oil Brake Fluid Dot 4 Isoctane/Toluene (70/30) + 15% Methanol Engine Cleaning Agent, Engine Coolant Antigel Fluid, Electrolyte Density 1285 Kg/m3 (Sulphuric Acid H2S04)

Wabash TECHNOLOGIES

Committed to sensor advancement.

Wabash Technologies, Inc.
1375 Swan Street, P.O. Box 829, Huntington, IN 46750-0829 USA Phone: 260-355-4100 Fax: 260-355-4265

Wabash Technologies, Ltd.
5 Faraday Park, Dorcan, Swindon, Wiltshire, SN3 5JF, England UK Phone: +44 1793 600500 Fax: +44 1793 600600

VIII. Specifications

Monitor

Operating Temperature:	-30□ ~ +70□
Modulation Type:	FSK
Mid-frequency:	433.9 MHz
Receiving Sensitivity:	-105 dbm
Input Voltage:	5V (Cigarette lighter plug)
	1.5V×2 (AA batteries)
Weight:	72±2 g

Transmitter

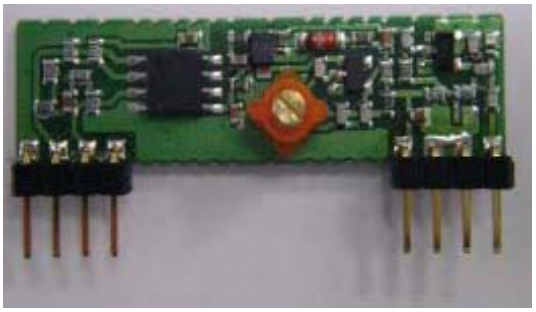
Operating Temperature:	-30□ ~ +85□
Pressure Range:	0~13 bar / 0~188 psi
Accuracy of Pressure Measurement:	± 0.15 bar / ± 2 psi
Modulation Type:	FSK
Mid-frequency:	433.9 Mhz
Transmitting Power:	-10 dbm
Weight:	8±1 g

Typical Applications

- Home Security Systems
- Remote Lighting Controls
- Remote Gate Controls
- Asset Tracking
- Sensor Monitoring

Features

- The Module's Frequency is from UHF (ASK) 280~433.92 MHz
- High Sensitivity Passive Design
- 4800 B/S Baseboard Data Rate.
- Simple to apply with Low External Parts count
- Low Supply Voltage: VCC= 5 Vdc
- ASK Data Shaping Comparator Included



Description

The RCR-433-MPR is a miniature receiver module that receives On-off keyed (OOK) modulation signal and demodulated to digital signal for the next decoder stage. Local Oscillator is made of L/C structure. The result is excellent performance in simple-to-use. The RCR-433-MPR is designed specifically for unlicensed remote control and wireless security receiver operating at 433 MHz in the USA under FCC Part 15 regulation.

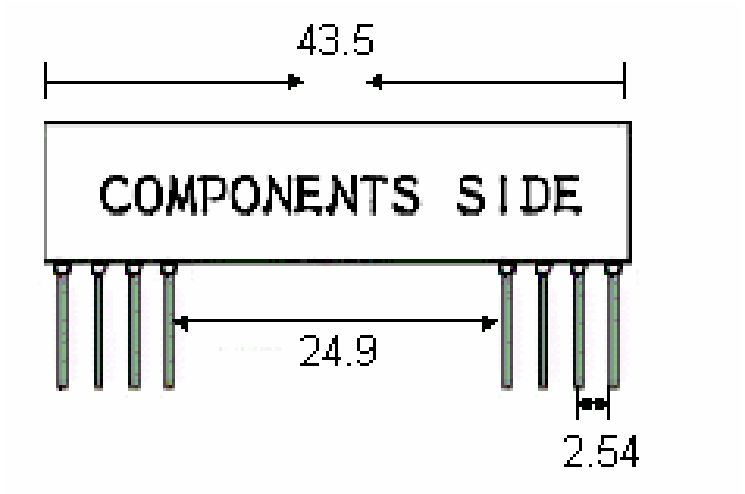
Item No.	Parameter	Description	Value			
			MIN	TYP	MAX	UNIT
1.	Operating Radio Frequency		280`433.92			MHz
2.	Sensitivity	Vcc 5.0, AT25C BER= 3/100		-105		dBm
3.	Modulation			ASK		
4.	Power Supply		4.75	5	5.25	v
5.	Supply Count			4	5	mA
6.	Data Rate			3		
7.	Operating Temperature		-40		+85	^{o}C
8.	RF Bandwidth – 3dB			4		MHz

Document Control

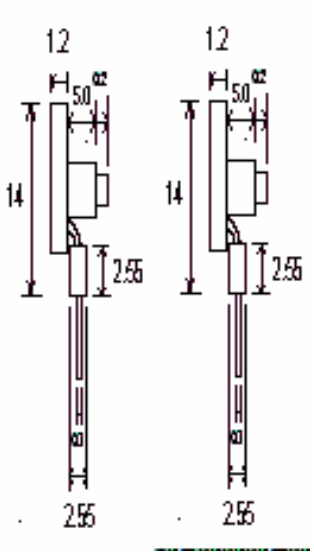
Created By	TJ Everett	1/20/07
Engineering Review		
Marketing Review		
Approved - Engineering		
Approved - Marketing		

Revision History

Revision	Author	Date	Description
1.0	TJE	1/20/07	Document Created



Mechanical Drawing



Pinout Diagram

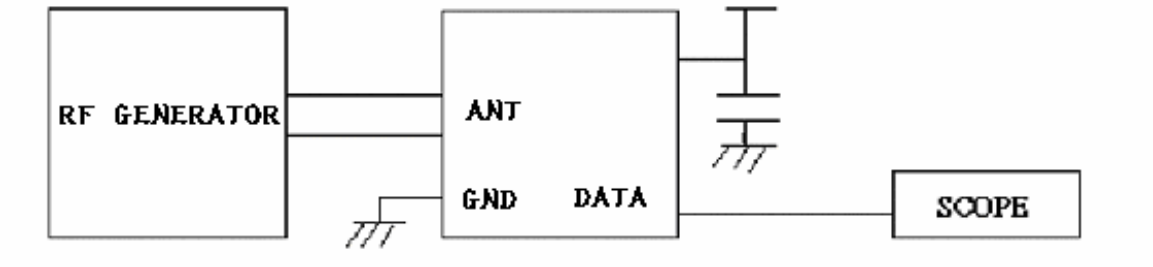
Absolute Maximum Ratings

Rating	Value	Units
Power Supply and All Input Pins	-0.3 to +12	VDC
Storage Temperature	-50 to +100	°C
Soldering Temperature (10 sec)	350	°C
Industrial Temperature	-40 to +85	°C

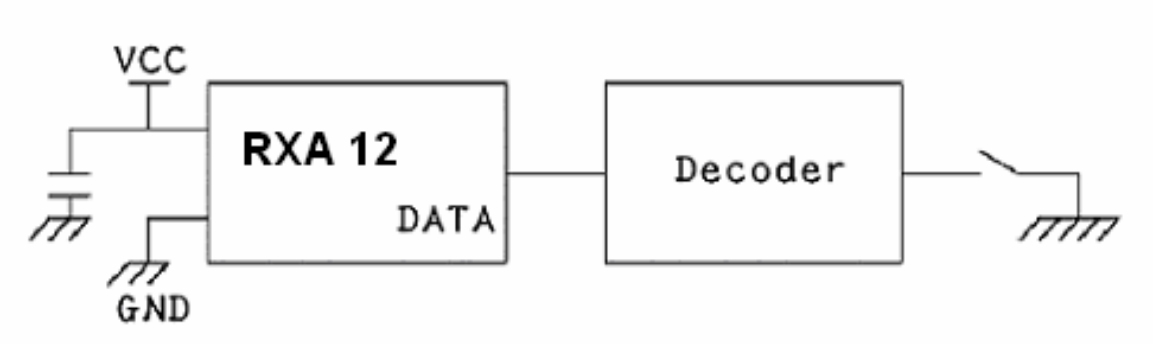
Pin Description

Pin	Name
1	GND
2	DIGITAL OUTPUT
3	LINEAR OUTPUT
4	VCC
5	VCC
6	GND
7	GND
8	ANT

Typing Test Circuit



Typical Receiver Application



Notes:
1. Encoder : HT12D/F , PTC (2262)
2. Antenna : Length = 22.6cm for 315MHz

Ordering Information

PRODUCT	ORDER CODE
RCR-433-MPR	RCR-433-MPR

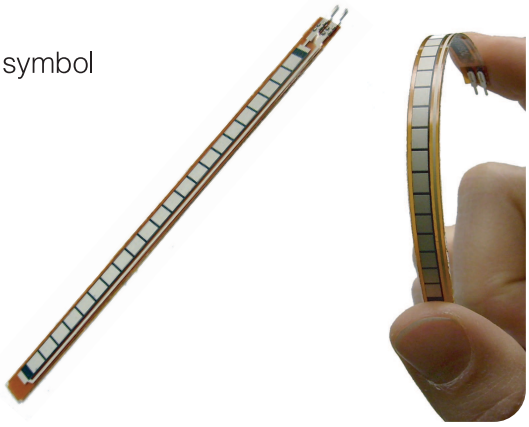
Radiotronix Inc.

905 Messenger Lane
Moore, OK 73160

(405) 794-7730
(405) 794-7477 (Fax)

www.radiotronix.com
sales@radiotronix.com

spectrasymbol



Flex Sensor FS

Features

- Angle Displacement Measurement
- Bends and Flexes physically with motion device
- Possible Uses
  - Robotics
  - Gaming (Virtual Motion)
  - Medical Devices
  - Computer Peripherals
  - Musical Instruments
  - Physical Therapy
- Simple Construction
- Low Profile

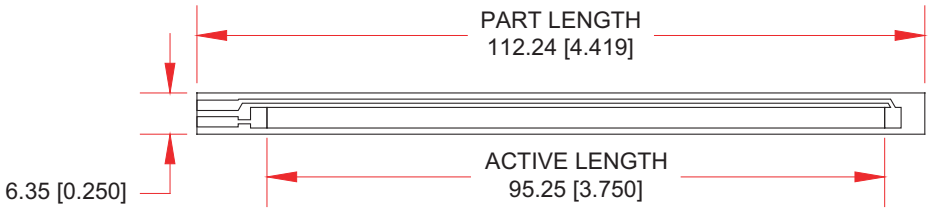
Mechanical Specifications

-Life Cycle: >1 million
-Height: ≤0.43mm (0.017")
-Temperature Range: -35°C to +80°C

Electrical Specifications

-Flat Resistance: 10K Ohms
-Resistance Tolerance: ±30%
-Bend Resistance Range: 60K to 110K Ohms
-Power Rating : 0.50 Watts continuous. 1 Watt Peak

Dimensional Diagram - Stock Flex Sensor

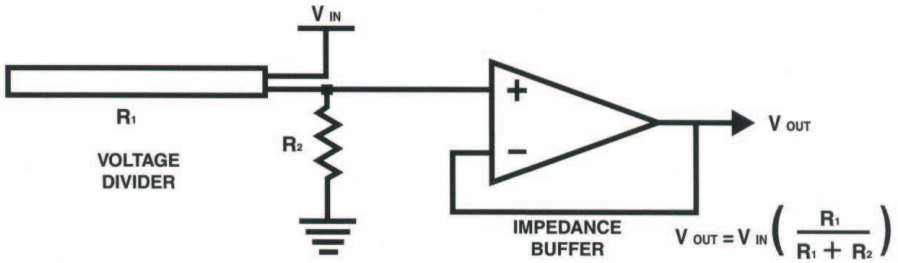


How to Order - Stock Flex Sensor

FS	L	0095	103	ST
Series	Model	Active Length	Resistance	Connectors
FS = Flex Sensor	L = Linear	0095 = 95.25mm	103 = 10 KOhms	ST = Solder Tab

How It Works

BASIC FLEX SENSOR CIRCUIT:

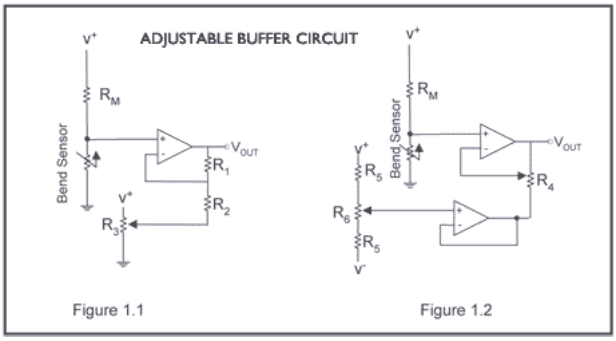


Following are notes from the ITP Flex Sensor Workshop

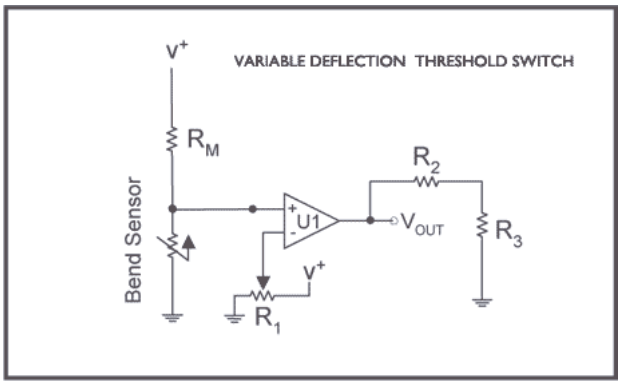
"The impedance buffer in the [Basic Flex Sensor Circuit] (above) is a single sided operational amplifier, used with these sensors because the low bias current of the op amp reduces errer due to source impedance of the flex sensor as voltage divider. Suggested op amps are the LM358 or LM324."

"You can also test your flex sensor using the simplest circut, and skip the op amp."

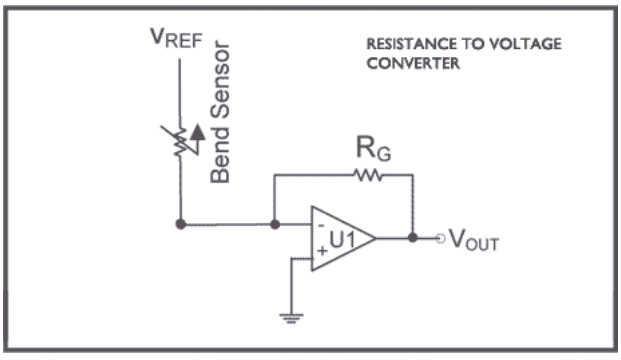
"Adjustable Buffer - a potentiometer can be added to the circuit to adjust the sensitivity range."



"Variable Deflection Threshold Switch - an op amp is used and outputs either high or low depending on the voltage of the inverting input. In this way you can use the flex sensor as a switch without going through a microcontroller."



"Resistance to Voltage Converter - use the sensor as the input of a resistance to voltage converter using a dual sided supply op-amp. A negative reference voltage will give a positive output. Should be used in situations when you want output at a low degree of bending."



SPEC. No. ED-05G101A
ISSUE March 5, 2007

SHARP

OPTO-ANALOG DEVICES DIVISION
ELECTRONIC COMPONENTS GROUP
SHARP CORPORATION

SPECIFICATION

DEVICE SPECIFICATION FOR

Analog Output Type Distance Measuring Sensor

MODEL No.

GP2Y0A41SK0F

Enclosed please find copies of the Specifications which consists of 10 pages including cover.
After confirmation of the contents, please be sure to send back ☐ copies of the Specifications with approving signature on each.

CUSTOMER'S APPROVAL	PRESENTED
DATE	DATE
BY	BY H. O

H. Ogura,
Department General Manager of
Engineering Dept.,III
Opto-Analog Devices Division.
ELECOM Group
SHARP CORPORATION

Product name : Analog output type distance measuring Sensor

Model No. : GP2Y0A41SK0F

1. These specification sheets include materials protected under copyright of Sharp Corporation ("Sharp"). Please do not reproduce or cause anyone to reproduce them without Sharp's consent.

2. When using this product, please observe the absolute maximum ratings and the instructions for use outlined in these specification sheets, as well as the precautions mentioned below. Sharp assumes no responsibility for any damage resulting from use of the product which does not comply with the absolute maximum ratings and the instructions included in these specification sheets, and the precautions mentioned below.

(Precautions)

(1) This product is designed for use in the following application areas ;

(・Computers ・OA equipment ・Telecommunication equipment (Terminal)
・Measuring equipment ・Tooling machines ・Audio visual equipment
・Home appliances)

If the use of the product in the above application areas is for equipment listed in paragraphs (2) or (3), please be sure to observe the precautions given in those respective paragraphs.

(2) Appropriate measures, such as fail-safe design and redundant design considering the safety design of the overall system and equipment, should be taken to ensure reliability and safety when this product is used for equipment which demands high reliability and safety in function and precision, such as ;

(・Transportation control and safety equipment (aircraft, train, automobile etc.)
・Traffic signals ・Gas leakage sensor breakers ・Rescue and security equipment
・Other safety equipment)

(3) Please do not use this product for equipment which require extremely high reliability and safety in function and precision, such as ;

(・Space equipment ・Telecommunication equipment (for trunk lines)
・Nuclear power control equipment ・Medical equipment)

(4) Please contact and consult with a Sharp sales representative if there are any questions regarding interpretation of the above three paragraphs.

3. Please contact and consult with a Sharp sales representative for any questions about this product.

1. Application

This specification applies to the outline and the characteristics of the analog output distance measuring sensor ; Model No. GP2Y0A41SK0F

2. Outline

Refer to the attached drawing No. CY13036j02.

3. Ratings and characteristics

Refer to the attached sheet, page 4, 5.

4. Reliability

Refer to the attached sheet, Page 6.

5. Outgoing inspection

Refer to the attached sheet, Page 6.

6. Supplements

6-1 GP2Y0A41SK0F Example of output distance characteristics

Refer to the attached sheet, page 8.

6-2 GP2Y0A41SK0F Example of output characteristics with the inverse of distance

Refer to the attached sheet, page 9.

6-3 This product shall not contain the following materials.

Also, the following materials shall not be used in the production process for this product.

Materials for ODS : CFC_S, Halon, Carbon tetrachloride 1.1.1-Trichloroethane (Methyl chloroform)

6-4 Product mass : Approx. 3.6g (TYP)

6-5 This product does not contain the chemical materials regulated by RoHS directive.

(Except for the NOT regulated by RoHS directive.)

6-6 Compliance with each regulation

6-6-1 The RoHS directive(2002/95/EC)

This product complies with the RoHS directive(2002/95/EC).

Object substances: mercury, lead (except for lead in high melting temperature type solders*1 and glass of electronic components), cadmium, hexavalent chromium, polybrominated biphenyls (PBB) and polybrominated diphenyl ethers (PBDE)

*1 : i.e. tin-lead solder alloys containing more than 85% lead

6-6-2 Content of six substances specified in Management Methods for Control of Pollution Caused by Electronic Information Products Regulation (Chinese : 电子信息产品污染控制管理办法).

Category	Toxic and hazardous substances					
	Lead (Pb)	Mercury (Hg)	Cadmium (Cd)	Hexavalent chromium (Cr(VI))	Polybrominated biphenyls (PBB)	Polybrominated diphenyl ethers (PBDE)
Voltage regulator	*	✓	✓	✓	✓	✓

✓ : indicates that the content of the toxic and hazardous substance in all the homogeneous materials of the part is below the concentration limit requirement as described in SJ/T 11363-2006 standard.

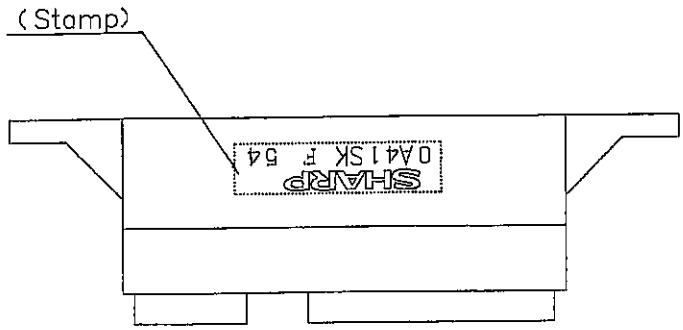
* : indicates that the content of the toxic and hazardous substance in at least one homogeneous material of the part exceeds the concentration limit requirement as described in SJ/T 11363-2006 standard.

Lead in high melting temperature type solders (i.e. tin-lead solder alloys containing more than 85% lead) and glass of electronic components (designated by "*" in the above table) are exempt from the RoHS directive (2002/95/EC), because there is no effective way to eliminate or substitute them by present scientific technology.

7. Notes

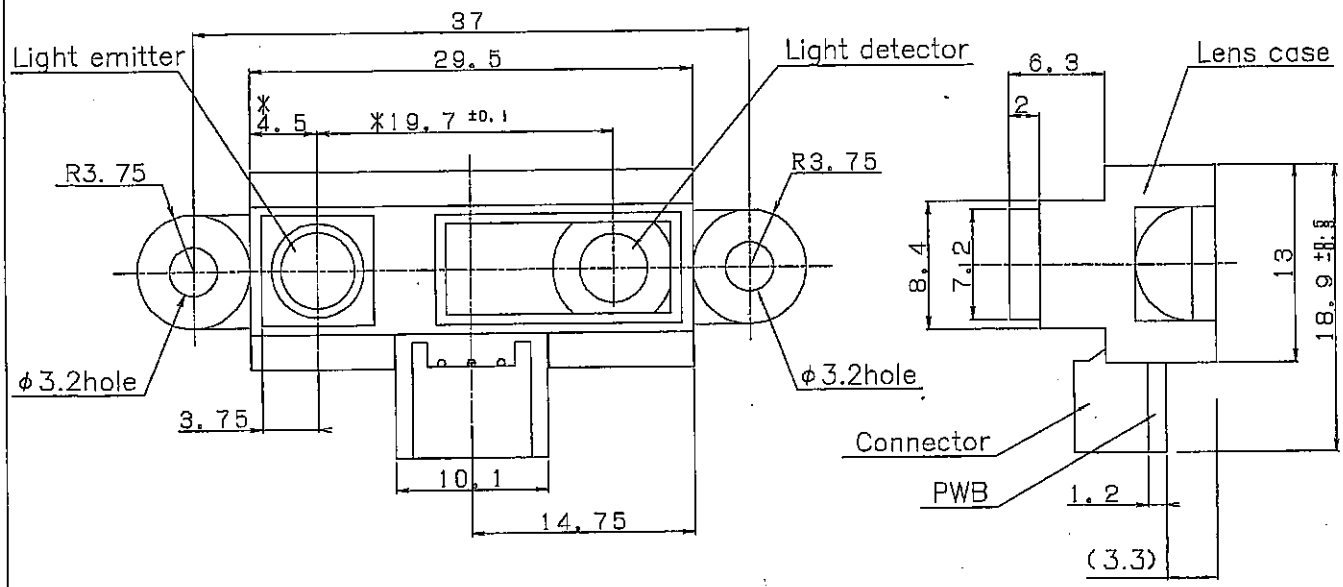
Refer to the attached sheet, page 7.

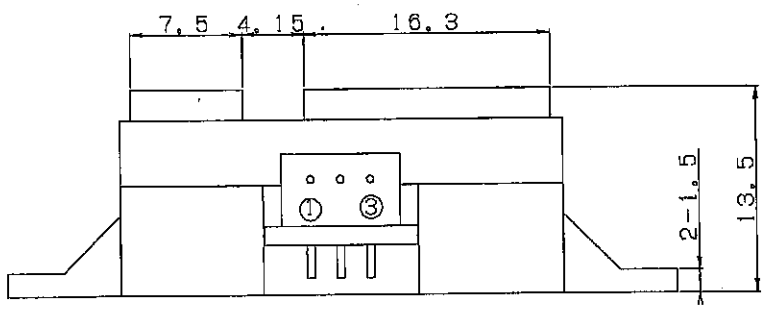
2. Outline



Stamp (Example)

SHARP
0A41SK F 5 4
Model name
Month (1 to 9, X, Y, Z)
Year (2005:5)





Connector signal

	Signal name
①	Vo
②	GND
③	Vcc

Connector :
J.S.T.TRADING COMPANY,LTD.
S3B-PH

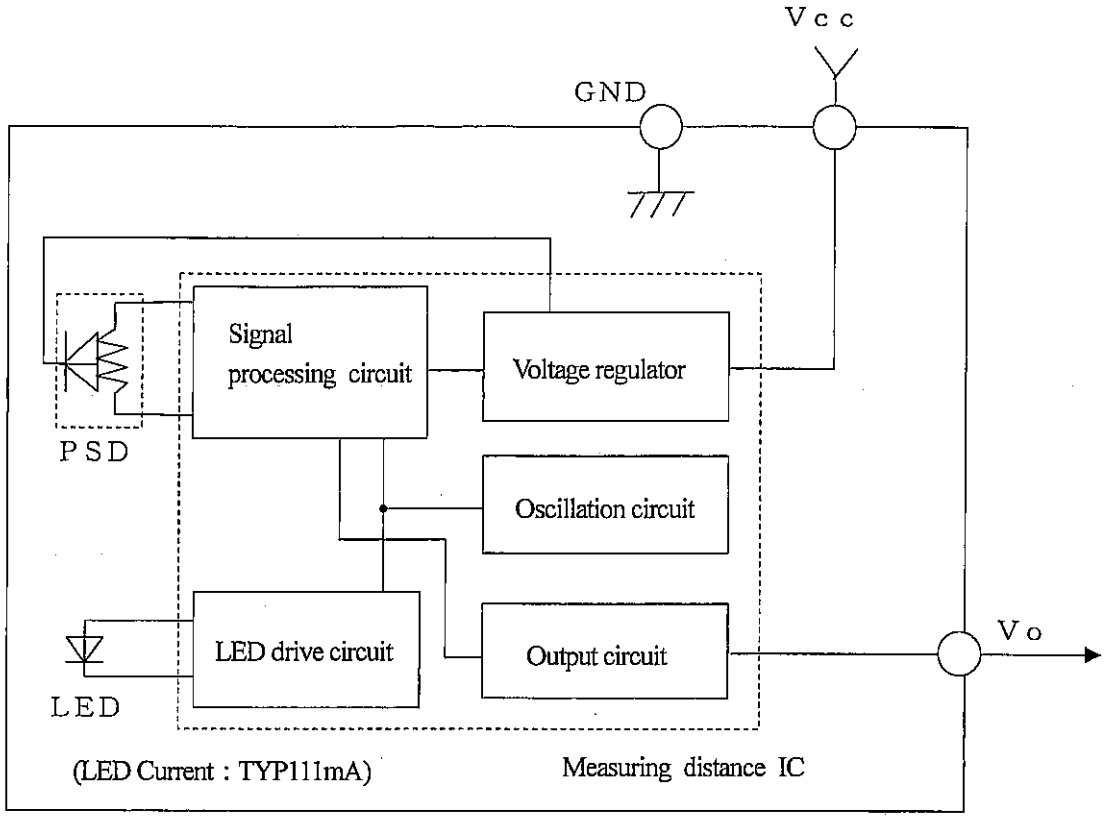
Materials
Lens: Acrylic acid resin
(Visible light cut-off resin)
Case: Carbonic ABS
(Conducttive resin)
PCB: Paper phenol

Unit: mm Scale: 2/1

Note 1: The dimensions marked * are described the dimensions of lens center position.
Note 2: Unspecified tolerance shall be ±0.3mm.
Note 3: The dimensions in parenthesis are shown for reference.

Name	GP2Y0A41SK0F Outline Dimensions
Drawing No.	CY13036j02

3-1 Schematic



3-2 Absolute maximum ratings (Ta=25℃, Vcc=5V)

Parameter	Symbol	Ratings	Unit	Remark
Supply voltage	Vcc	-0.3 to +7	V	-
Output terminal voltage	Vo	-0.3 to Vcc+0.3	V	-
Operating temperature	Topr	-10 to +60	℃	-
Storage temperature	Tstg	-40 to +70	℃	-

Operating supply voltage

Symbol	Rating	Unit	Remark
Vcc	4.5 to 5.5	V	-

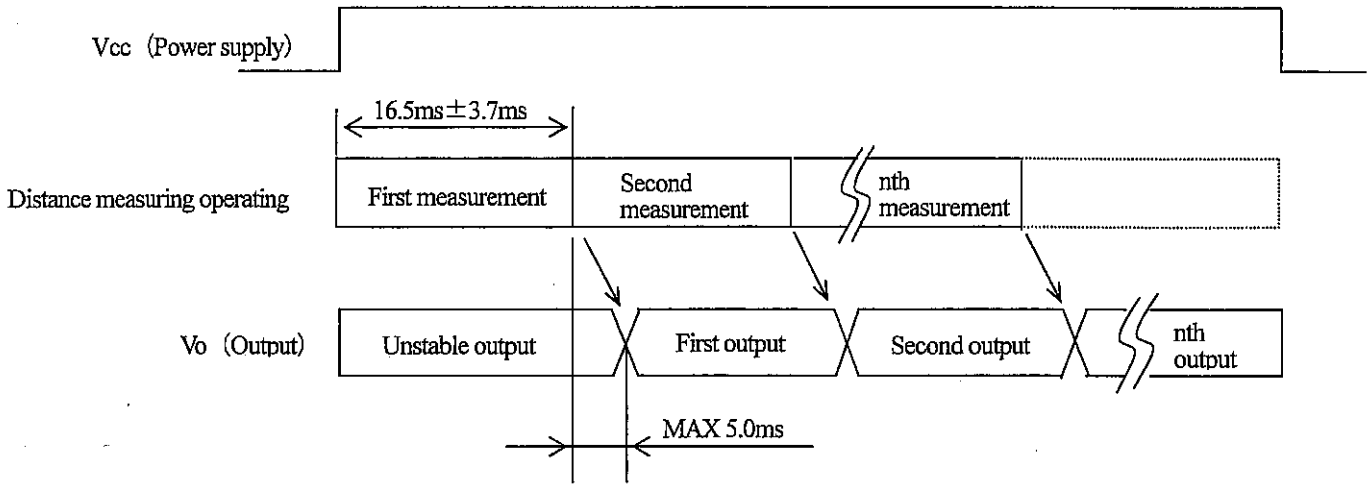
3-3 Electro-optical Characteristics

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Measuring distance range	ΔL	(Note 1)	4	-	30	cm
Output terminal voltage	Vo	L=30cm (Note 1)	0.25	0.4	0.55	V
Output voltage difference	ΔVo	Output change at L change (30cm → 4cm) (Note 1)	1.95	2.25	2.55	V
Average supply current	Icc	L=30cm (Note 1)	-	12	22	mA

※ L: Distance to reflective object

(Note 1) Using reflective object: White paper (Made by Kodak Co., Ltd. gray cards R-27・white face, reflective ratio ; 90%)

3-4 Timing chart



4. Reliability

The reliability of products shall be satisfied with items listed below. Confidence level : 90%

LTPD : 20 or 30

No.	Test Items	Test Conditions	Failure Judgement Criteria	Samples (n) / Defective (c)
1	Temperature cycling	1 cycle -40℃ to +70℃ (30min.) (30min.) 25 cycle test	Initial × 0.8 > Vo Vo > Initial × 1.2 (Note 1)	n=11, c=0
2	High temp. and high humidity storage	+40℃, 90%RH, 500h		n=11, c=0
3	High temp. storage	+70℃, 500h		n=11, c=0
4	Low temp. storage	-40℃, 500h		n=11, c=0
5	Operation life (High temp.)	+60℃, Vcc=5V, 500h		n=11, c=0
6	Mechanical shock	1000m/s^2, 6.0ms 3times/±X, ±Y, ±Z direction		n=8, c=0
7	Variable frequency vibration	10 to 55 to 10Hz/1min. 2h/X, Y, Z direction overall amplitude : 1.5mm		n=8, c=0

(Note 1) Test conditions are according to 3-3 Electro-optical characteristics.

(Note 2) After test, measurement shall be measured after leaving under the normal temperature and the normal humidity for two hours. But no dew point.

5. Outgoing inspection

(1) Inspection lot

Inspection shall be carried out per each delivery lot.

(2) Inspection method

A single sampling plan, normal inspection level II based on ISO 2859 is applied.

The AQL according to the inspection items are shown below.

Defect	Inspection item	AQL (%)
Major defect	Electro-optical characteristics defect (In para. 3-3)	0.4
Minor defect	Defect on appearance and dimension ※ Crack, chip, scratch, stain	1.0

※ Crack, chip, scratch, stain

One which affects the characteristics of para. 3-3 shall be defect.

7. Notes

[Advice for the optics]

7-1 Lens of this device shall be kept cleanly. There are cases that dust, water or oil and so on deteriorate the characteristics of this device. Please consider in actual application.

7-2 In case that protection is set in front of the emitter and detector portion, the protection cover which has the most efficient transmittance at the emitting wavelength range of LED for this product (λ=870nm±70nm), shall be recommended to use. The face and back of protection cover should be mirror polishing. Also, as there are cases that the characteristics may not be satisfied with according to the distance between the protection cover and this product or the thickness of the protection cover, please use this product after confirming the operation sufficiently in actual application.

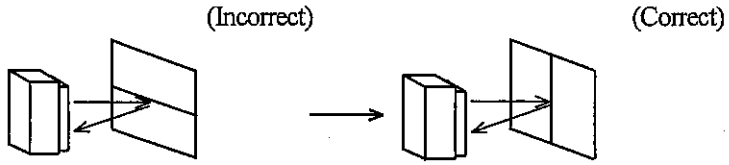
[Advice for the characteristics]

7-3 In case that there is an object near to light exits of the sensor between the sensor and the detected object, please use this device after confirming sufficiently what the characteristics of this sensor do not change by the object.

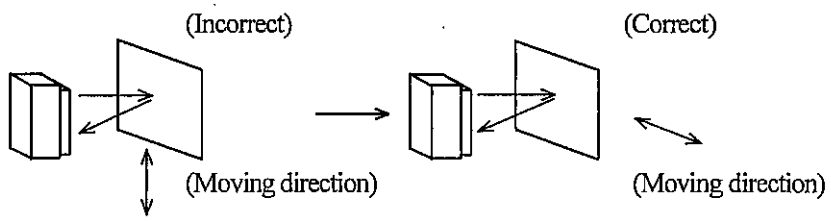
7-4 When the detector surface receive direct light from the sun, tungsten lamp and so on, there are cases that it can not measure the distance exactly. Please consider the design that the detector does not receive direct light from such light source.

7.5 Distance between sensor and mirror reflector can not sometimes measure exactly.
In case of changing the mounting angle of this product, it may measure the distance exactly.

7.6 In case that reflective object has boundary line clearly, there is cases that distance can not measure exactly.
At that time, if direction of boundary line and the line between emitter center and detector center parallels, it is possible to decrease deviation of measuring distance.



7-7 In order to decrease measuring error by moving direction of object, we recommend to mount the sensor like below drawing.



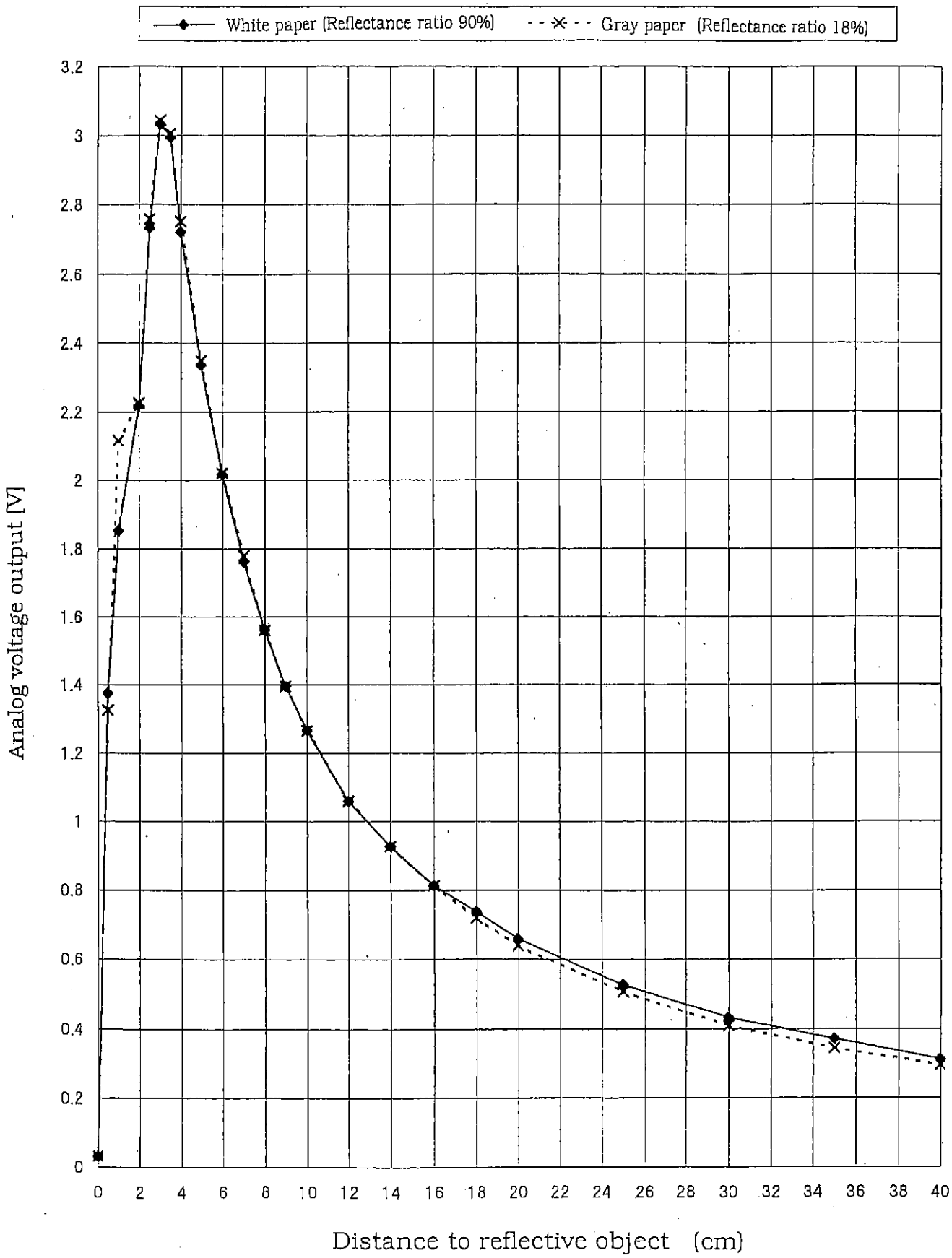
7-8 In order to stabilize power supply line, we recommend to connect a by-pass capacitor of 10μF or more between Vcc and GND near this product.

[Notes on handling]

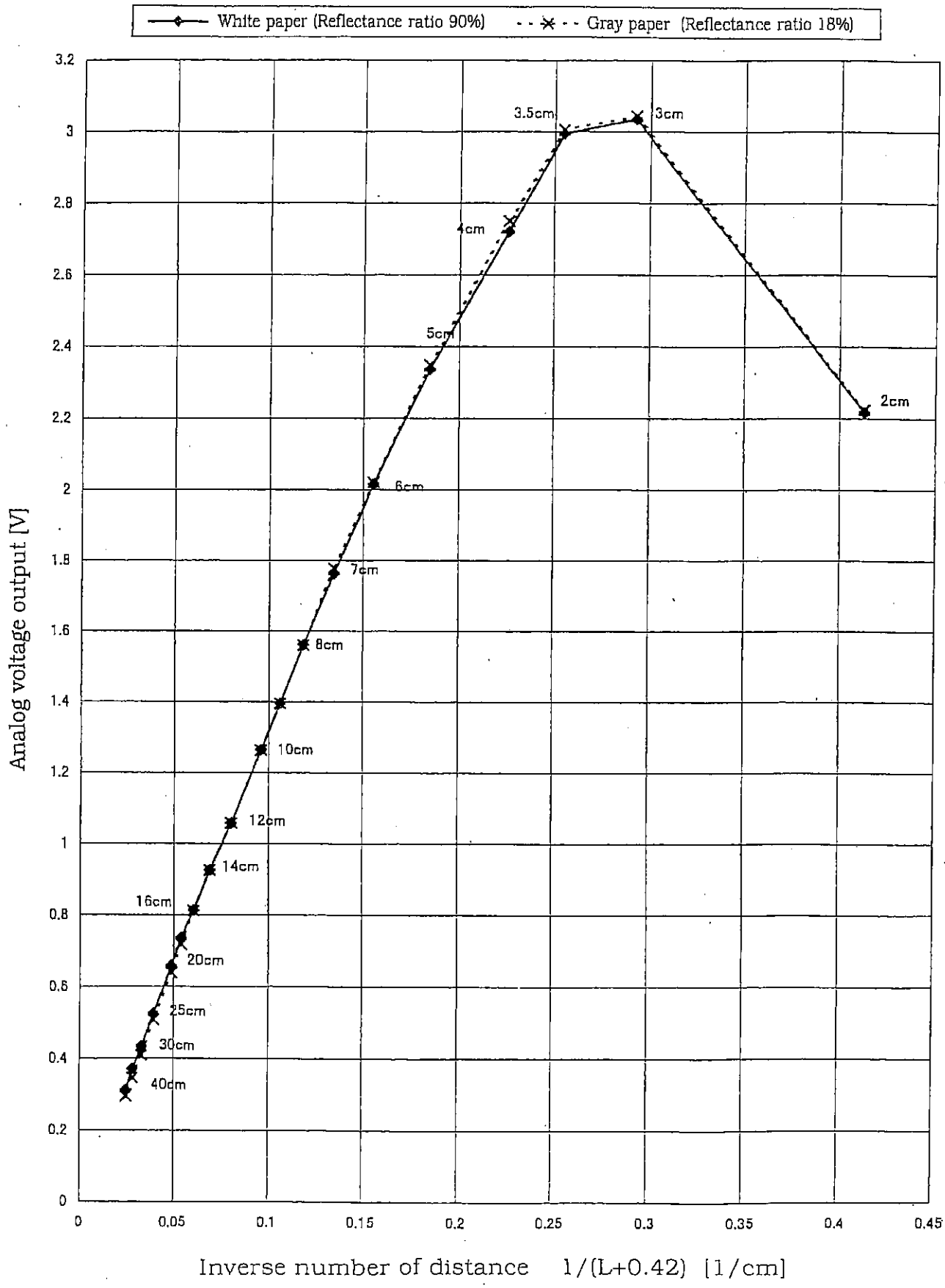
7-9 Please don't do washing. Washing may deteriorate the characteristics of optical system and so on.
Please confirm resistance to chemicals under the actual usage since this product has not been designed against for washing.

7-10 There are some possibilities that the sensor inside the case package with lens may be exposed to the excessive mechanical stress. Please be careful not to cause any excessive pressure on the case package with lens and also on the PCB at the assembly and inserting of the set.

6-1 GP2Y0A41SK0F Example of output distance characteristics



6-2 GP2Y0A41SK0F Example of output distance characteristics with the inverse of distance



Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

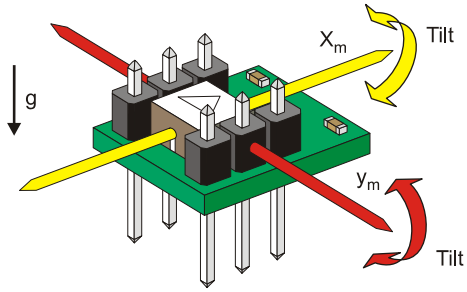
Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

Memsic 2125 Dual-Axis Accelerometer (#28017)

The Memsic 2125 is a low-cost thermal accelerometer capable of measuring tilt, collision, static and dynamic acceleration, rotation, and vibration with a range of ±3 g on two axes. Memsic provides the 2125 IC in a surface-mount format. Parallax mounts the circuit on a tiny PCB providing all I/O connections so it can easily be inserted on a breadboard or through-hole prototype area.

Features

- Measures ±3 g on each axis
- Simple pulse output of g-force for each axis
- Convenient 6-pin 0.1" spacing DIP module
- Analog output of temperature (TOut pin)
- Fully temperature compensated over 0 to 70 °C operating temperature range



Key Specifications

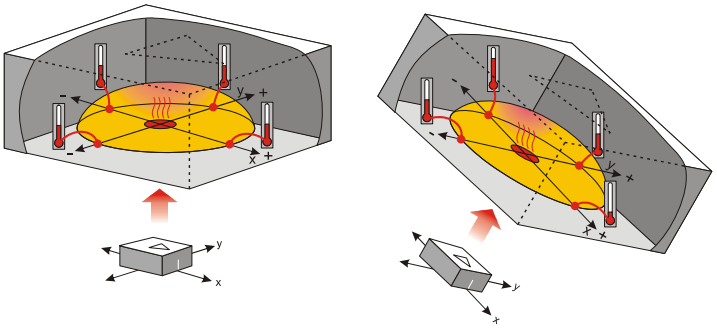
- Power Requirements: 3.3 to 5 VDC; < 5 mA supply current
- Communication: TTL/CMOS compatible 100 Hz PWM output signal with duty cycle proportional to acceleration
- Dimensions: 0.42 x 0.42 x 0.45 in (10.7 x 10.7 x 11.8 mm)
- Operating temperature: 32 to 158 °F (0 to 70 °C)

Application Ideas

- Dual-axis tilt and acceleration sensing for autonomous robot navigation
- R/C tilt controller or autopilot
- Tilt-sensing Human Interface Device
- Motion/lack-of-motion sensor for alarm system
- Single-axis rotational angle and position sensing

Theory of Operation

The MX2125 has a chamber of gas with a heating element in the center and four temperature sensors around its edge. When the accelerometer is level, the hot gas pocket rises to the top-center of the chamber, and all the sensors will measure the same temperature.

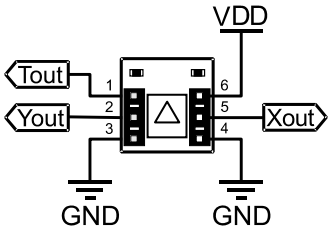


By tilting the accelerometer, the hot gas will collect closer to some of temperature sensors. By comparing the sensor temperatures, both static acceleration (gravity and tilt) and dynamic acceleration (like taking a ride in a car) can be detected. The MX2125 converts the temperature measurements into signals (pulse durations) that are easy for microcontrollers to measure and decipher.

Pin Definitions

For Memsic MXD2125GL pin ratings, see the manufacturer's datasheet posted on the 28017 product page at www.parallax.com.

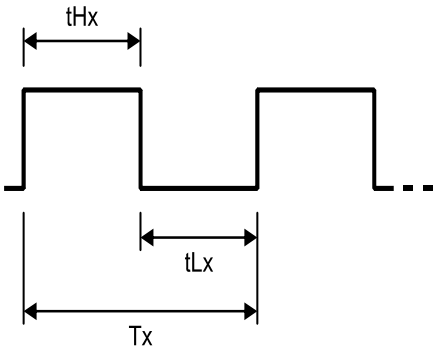
Pin	Name	Function
1	Tout	Temperature Out
2	Yout	Y-axis PWM output
3	GND	Ground -> 0 V
4	GND	Ground -> 0 V
5	Xout	X-axis PWM output
6	VDD	Input voltage: +3.3 to +5 VDC



Communication Protocol

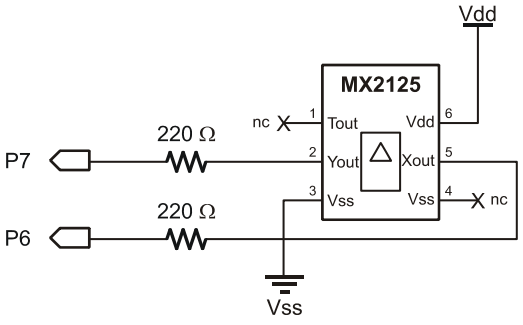
Each axis has a 100 Hz PWM duty cycle output in which acceleration is proportional to the ratio tHx/Tx. In practice, we have found that Tx is consistent so reliable results can be achieved by measuring only the duration of tHx. This is easy to accomplish with the BASIC Stamp PULSIN command or with the Propeller chip's counter modules.

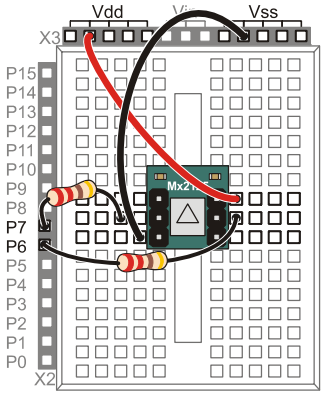
With Vdd = 5V, 50% duty cycle corresponds to 0 g, but this will vary with each individual unit within a range of 48.7% to 51.3%. This zero offset may be different when using Vdd = 3.3 V. See the manufacturer's datasheet for details.



Example Circuit

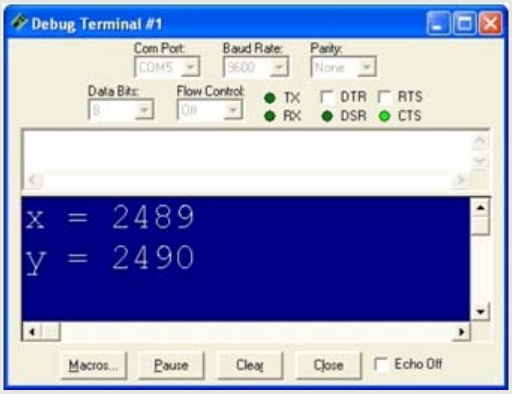
The example schematic and wiring diagram below are for the BASIC Stamp and Board of Education.





The program below, SimpleTilt.bs2, simply measures the pulse width, that is, the duration of tHx, for each axis. The raw values are displayed in the BASIC Stamp Editor's Debug Terminal. If you run the program, then tilt the accelerometer, you should see the values for each axis change.

```
' Smart Sensors and Applications - SimpleTilt.bs2
' Measure room temperature tilt.

'{$STAMP BS2}
'{$PBASIC 2.5}

x               VAR     Word
y               VAR     Word

DEBUG CLS

DO

  PULSIN 6, 1, x
  PULSIN 7, 1, y

  DEBUG HOME, DEC4 ? X, DEC4 ? Y

  PAUSE 100

LOOP
```



Programming Resources and Downloads

BASIC Stamp

- **Smart Sensors and Applications** — The BASIC Stamp example above is taken from the Stamps in Class text *Smart Sensors and Applications*, which features several chapters specific to the Memsic Dual-Axis Accelerometer. Topics include output scaling and offset, measuring vertical rotation, tilt-controlled video gaming basics, data logging g-force during a skateboard trick, and data logging acceleration on an RC car. The book and sample code can be downloaded from the 28029 product page at http://www.parallax.com

- **Boe-Bot Robot Projects with the Memsic 2125 Accelerometer** — The following projects with source code are posted under the Stamps in Class Mini-Projects sticky-thread in the Stamps in Class Forum at http://forums.parallax.com:
  - Boe-Bot Robot Navigation with Accelerometer Incline Sensing
  - A Tilt Radio Controller for Your Boe-Bot
- **The Memsic 2125 Demo Kit BASIC Stamp Source Code** — this source code contains conditional compile directives that allow it to be used with the BS2, BS2e, BS2sx, BS2p, and BS2pe.

Propeller Objects

Several Memsic 2125 Accelerometer code objects and applications for the Propeller chip are available in the Propeller Object Exchange (http://obex.parallax.com).

Below is a photograph of the high-speed Memsic MXD2125 Accelerometer Demo in action. This application "provides a high speed assembly driver, and separate-cog and same-cog Spin versions of the MXD2125 Dual Axis Accelerometer. The high speed version displays the data on a television as a 3D wireframe plane with normal vector.

iC-LFL1402

256x1 LINEAR IMAGE SENSOR

preliminary

FEATURES

- 256 active photo pixels of 56 µm at a gap and distortion free pitch of 63.5 µm (400 DPI)
- Integrating L-V conversion followed by a sample & hold circuit
- High sensitivity and uniformity over wavelength
- High clockrates of up to 5 MHz
- Only 256 clocks required for readout
- Shutter function enables flexible integration times
- Glitch-free analogue output
- Push-pull output amplifier
- 5 V single supply operation
- Can run off external bias to reduce power consumption
- Function equivalent to TSL1402 (serial mode)

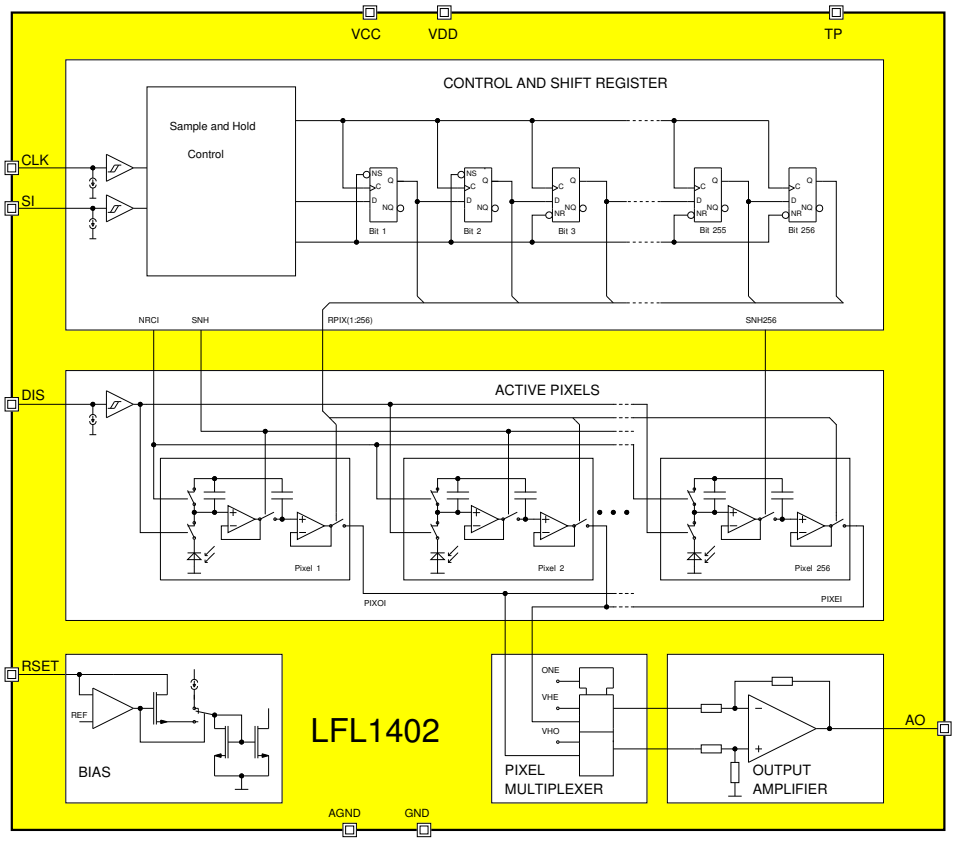
APPLICATIONS

- Optical line sensors
- CCD substitute

PACKAGES

OBGA™ LFL1C

BLOCK DIAGRAM



DESCRIPTION

iC-LFL1402 is an integrating light-to-voltage converter with a single line of 256 pixels pitched at 63.5 µm (center-to-center distance). Due to the monolithical integration there is no pixel-gap or pitch distortion whatsoever. Each pixel consists of a 56.4 µm x 200 µm photodiode, an integration capacitor and a sample and hold circuit.

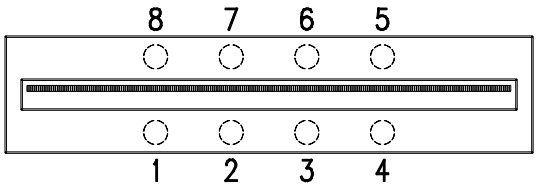
The integrated control logic makes operation very simple, with only a start and clock signal necessary. A third control input enables the integration period to be prematurely terminated at any time (electronic shutter).

When the start signal is given the hold mode is activated for all pixels simultaneously with the next rising clock edge; starting with pixel 1 the hold voltages are switched in sequence to the push-pull output amplifier. The second clock pulse deletes all integration capacitors and the integration period starts again in the background during the output phase. A run is complete after 256 clock pulses.

iC-LFL1402 is suitable for high clock rates of up to 5 MHz. If this is not required the supply current can be reduced via the external bias setting.

PACKAGES OBGA™ LFL1C

PIN CONFIGURATION OBGA™ LFL1C (top view)

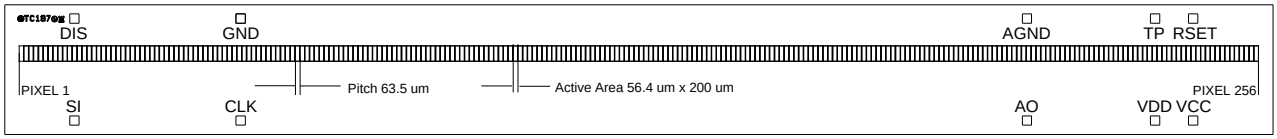


PIN FUNCTIONS

No.	Name	Function
1	SI	Start Integration Input
2	CLK	Clock Input
3	AO	Analogue Output
4	VCC	+5 V Supply Voltage
5	RSET	Bias Current (resistor from VCC to RSET; when connected to GND the internal bias setting is activated)
6	AGND	Analogue Ground
7	GND	Digital Ground
8	DIS	Disable Integration Input

CHIP-LAYOUT

iC-LFL1402
Chip size: 16.6 mm x 1.7 mm



ABSOLUTE MAXIMUM RATINGS

Beyond these values damage may occur; device operation is not guaranteed.

Item No.	Symbol	Parameter	Conditions	Fig.	Min.	Max.	Unit
G001	VDD	Digital Supply Voltage			-0.3	6	V
G002	VCC	Analogue Supply Voltage			-0.3	6	V
G003	V()	Voltage at SI, CLK, DIS, RSET, TP, AO			-0.3	VCC + 0.3	V
G004	I()	Current in RSET, TP, AO			-10	10	mA
G005	Vd()	ESD Susceptibility at all pins	MIL-STD-883, Method 3015, HBM 100 pF/1.5 kΩ			2	kV
G006	Tj	Operating Junction Temperature			-40	125	°C
G007	Ts	Storage Temperature Range	see package specification OBGA™ LFL1C				

THERMAL DATA

Operating Conditions: VCC = VDD = 5 V ±10%

Item No.	Symbol	Parameter	Conditions	Fig.	Min.	Typ.	Max.	Unit
T01	Ta	Operating Ambient Temperature Range	see package specification OBGA™ LFL1C					

All voltages are referenced to ground unless otherwise stated.
All currents into the device pins are positive; all currents out of the device pins are negative.

ELECTRICAL CHARACTERISTICS

Operating Conditions: VCC = VDD = 5 V ±10%, RSET = GND, Tj = -25...85 °C unless otherwise noted

Item No.	Symbol	Parameter	Conditions	Tj °C	Fig.	Min.	Typ.	Max.	Unit
Total Device									
001	VDD	Digital Supply Voltage Range				4.5		5.5	V
002	VCC	Analogue Supply Voltage Range				4.5		5.5	V
003	I(VDD)	Supply Current in VDD	f(CLK) = 1 MHz f(CLK) = 5 MHz				0.39 1.85		mA mA
004	I(VCC)	Supply Current in VCC					11.5		mA
005	Vc()hi	Clamp Voltage hi at SI, CLK, DIS, TP, RSET	Vc()hi = V() − V(VCC); I() = 1 mA			0.3		1.8	V
006	Vc()lo	Clamp Voltage lo at SI, CLK, DIS, TP, RSET	Vc()hi = V() − V(AGND); I() = -1 mA			-1.5		0.3	V
007	Vc()hi	Clamp Voltage hi at AO	Vc()hi = V(AO) − V(VCC); I(AO) = 1 mA			0.3		1.5	V
008	Vc()lo	Clamp Voltage lo at AO, VCC, VDD, GND	Vc()lo = V() − V(AGND); I() = -1 mA			-1.5		-0.3	V
Photodiode Array									
201	A()	Radiant Sensitive Area	200 µm x 56.40 µm per Pixel				0.01128		mm²
202	S(λ)max	Spectral Sensitivity	λ = 680 nm		1		0.5		A/W
203	λ_{ar}	Spectral Application Range	S(λ_{ar}) = 0.25 x S(λ)max		1	400		980	nm
Analogue Output AO									
301	Vs()lo	Saturation Voltage lo	I() = 1 mA					0.5	V
302	Vs()hi	Saturation Voltage hi	Vs()hi = VCC - V(), I() = -1 mA					1	V
303	K	Sensitivity	λ = 680 nm, package OBGA™ LFL1C				2.88		V/pWs
304	V0()	Offset Voltage	integration time 1 ms, no illumination				400	800	mV
305	ΔV0()	Offset Voltage Deviation during integration mode	ΔV0() = V(AO)t1 − V(AO)t2, Δt = t2 − t1 = 1 ms			-250		50	mV
306	ΔV()	Signal Deviation during hold mode	ΔV0() = V(AO)t1 − V(AO)t2, Δt = t2 − t1 = 1 ms			-150		150	mV
307	tp(CLK-AO)	Settling Time	Cl(AO) = 10 pF, CLK lo → hi until V(AO) = 0.98 x V(VCC)					200	ns
Power-On-Reset									
801	VCCon	Power-On Release by VCC						4.4	V
802	VCCoff	Power-Down Reset by VCC				1			V
803	VCChys	Hysteresis	VCChys = VCCon − VCCoff			0.4	1	2	V
Bias Current Adjust RSET									
901	Ibias()	Permissible External Bias Current				20		100	µA
902	Vref	Reference Voltage	I(RSET) = Ibias			2.5	3	3.5	V
Input Interface SI, CLK, DIS									
B01	Vt()hi	Threshold Voltage hi				1.4		1.8	V
B02	Vt()lo	Threshold Voltage lo				0.9		1.2	V
B03	Vt()hys	Hysteresis	Vt()hys = Vt()hi − Vt()lo			300		800	mV
B04	I()	Pull-Down Current				10	30	50	µA
B05	fclk	Permissible Clock Frequency						5	MHz

OPTICAL CHARACTERISTICS: Diagrams

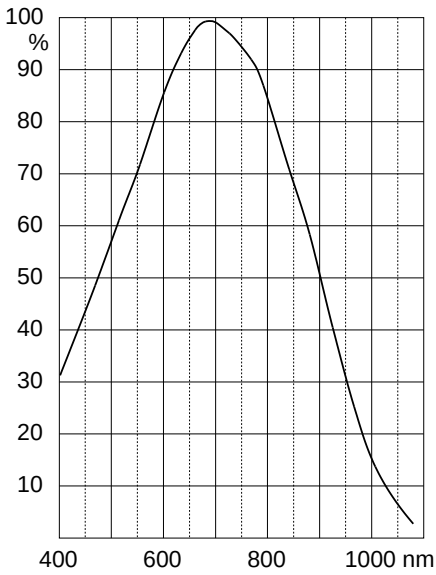


Figure 1: Relative spectral sensitivity

OPERATING REQUIREMENTS: Logic

Operating Conditions: VCC = VDD = 5 V ±10%, Tj = -25...85 °C
input levels lo = 0...0.45 V, hi = 2.4 V...VCC, see Fig. 2 for reference levels

Item No.	Symbol	Parameter	Conditions	Fig.	Min.	Max.	Unit
I001	tset	Setup Time: SI stable before CLK lo → hi		3	50		ns
I002	thold	Hold Time: SI stable after CLK lo → hi		3	50		ns

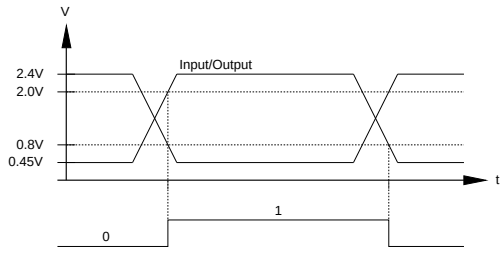


Figure 2: Reference levels

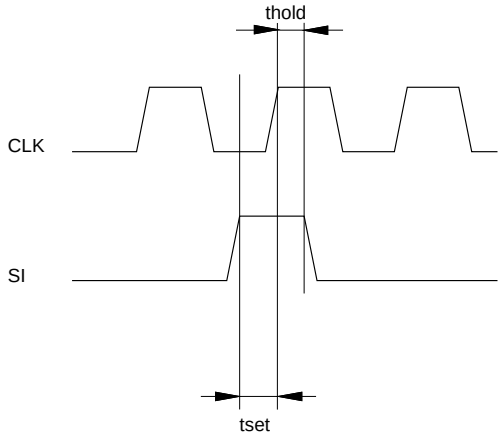


Figure 3: Timing diagram

DESCRIPTION OF FUNCTIONS

Normal operation
Following an internal power-on reset the integration and hold capacitors are discharged and the sample and hold circuit is set to sample mode. A high signal at SI and a rising edge at CLK triggers a readout cycle and with it a new integration cycle.

In this process the hold capacitors of pixels 1 to 255 are switched to hold mode immediately (SNH = 1), with pixel 256 (SNH256 = 1) following suit one clock pulse later. This special procedure allows all pixels to be read out with just 256 clock pulses. The integration capacitors are discharged by a one clock long reset signal (NRCI = 0) which occurs between the 2nd and 3rd falling edge of the readout clock pulse (cf. Figure 4). After the 255 pixels have been read out these are again set to sample mode (SNH = 0), likewise for pixel 256 one clock pulse later (SNH256 = 0).

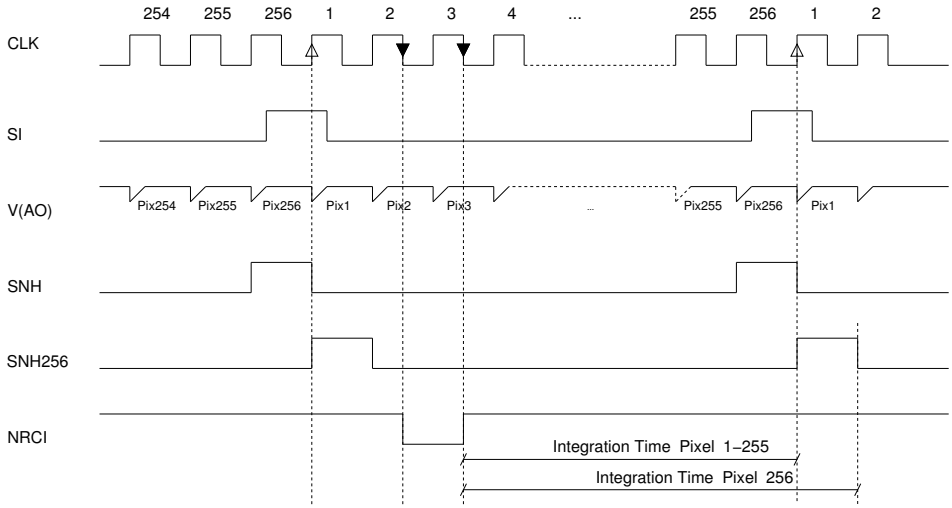


Figure 4: Readout cycle and integration sequence

If prior to the 256th clock pulse a high signal occurs at SI the present readout is halted and immediately reinitiated with pixel 1. In this instance the hold capacitors retain their old value i.e. hold mode prevails (SNH/SNH256 = 0).

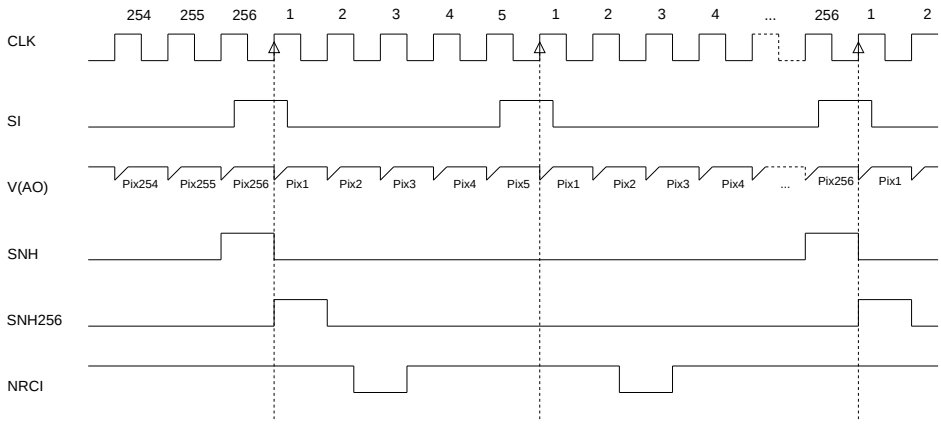


Figure 5: Restarting a readout cycle

With more than 256 clock pulses until the next SI signal, pixel 1 is output without entering hold mode; the output voltage tracks the voltage of the pixel 1 integration capacitor.

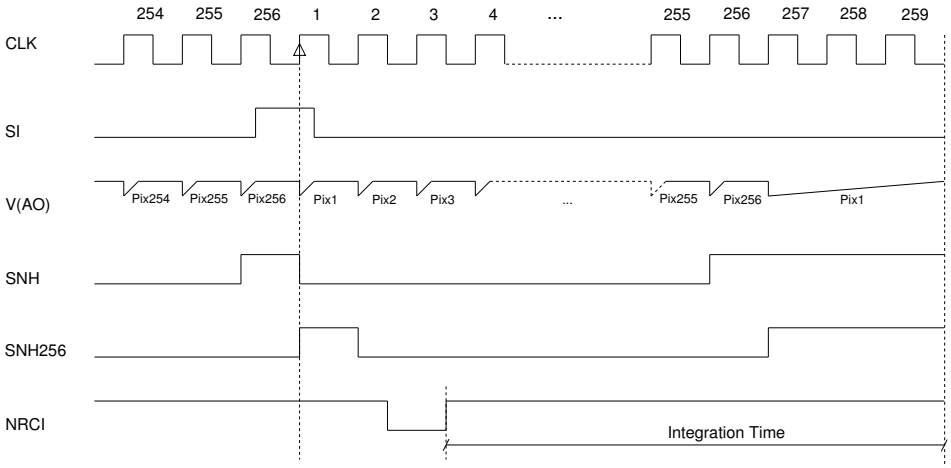


Figure 6: Clock pulse continued without giving a new integration start signal

Operation with the shutter function

Integration can be stopped at any time via pin DIS, i.e. the photodiodes are disconnected from their corresponding integration capacitor when DIS is high and the current integration capacitor voltages are maintained. If this pin is open or switched to GND the pixel photocurrents are summed up by the integration capacitors until the next successive SI signal follows.

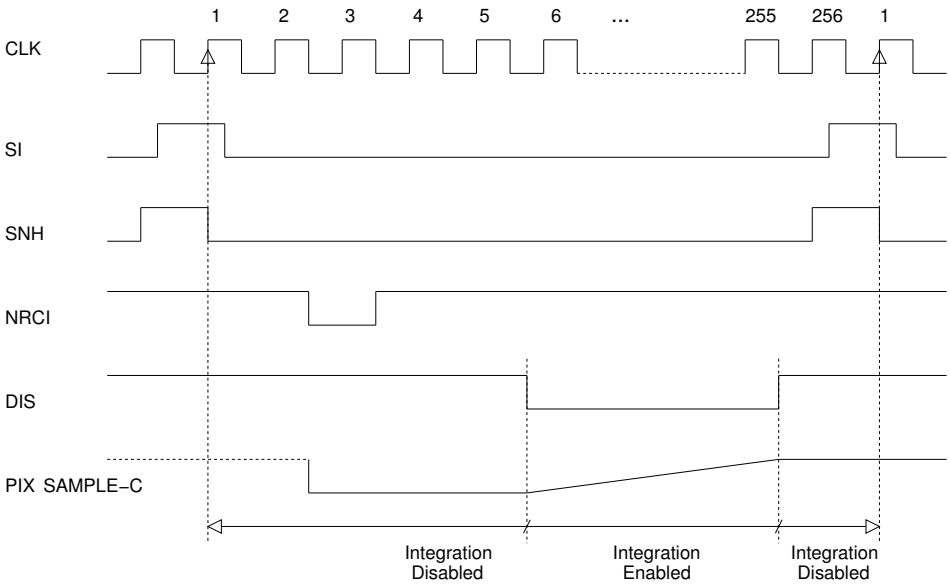


Figure 7: Defining the integration time via shutter input DIS

External bias current setting

In order to reduce the power consumption of the device an external reference current can be supplied to pin RSET which reduces the maximum readout frequency, however. To this end a resistor must be connected from VCC to RSET. If this pin is not used, it should be connected to GND.

ORDERING INFORMATION

Type	Package	Order Designation
iC-LFL1402	OBGA™ LFL1C -	iC-LFL OBGA LFL1C iC-LFL Chip

For information about prices, terms of delivery, other packaging options etc. please contact:

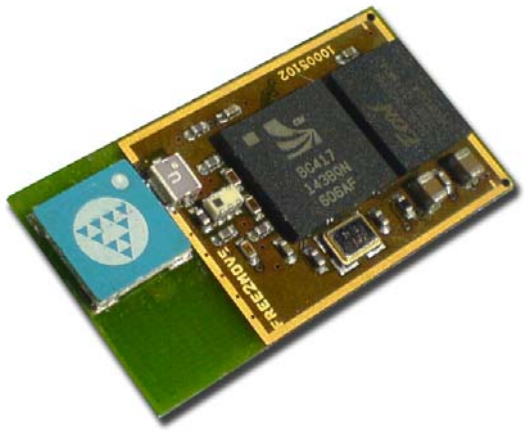
iC-Haus GmbH
Am Kuemmerling 18
D-55294 Bodenheim
GERMANY

Tel.: +49 (61 35) 92 92-0
Fax: +49 (61 35) 92 92-192
Web: http://www.ichaus.com
E-Mail: sales@ichaus.com

Low power Bluetooth™ Module with antenna- F2M03GLA

Datasheet

Features



- Fully qualified end product with
- Bluetooth™ v2.0+EDR, CE and FCC
- Low power consumption
- Integrated high output antenna
- Transmit power up to +8dBm
- Class1/ 2/ 3 Configurable
- Range up to 350m (line of sight)
- Piconet and Scatternet capability, support for up to 7 slaves
- Require only few external components
- Industrial temperature range -40°C to +85°C
- USB v2.0 compliant
- Extensive digital and analog I/O interface
- PCM interface for up to 3 simultaneous voice channels
- Large external memory for custom applications
- Support for 802.11b/g Co-Existence
- RoHS compliant

Applications

- Industrial and domestic appliances
- Cable replacement
- Medical systems
- Automotive applications
- Stand-alone sensors
- Embedded systems
- Cordless headsets
- Computer peripherals (Mice, Keyboard, USB dongles, etc.)
- Handheld, laptop and desktop computers
- Mobile phones

Pb-Free

RoHS COMPLIANT 2002/96/EC

Bluetooth™

General Description

F2M03GLA is a Low power embedded Bluetooth™ v2.0+EDR module with built-in high output antenna. The module is a fully Bluetooth™ compliant device for data and voice communication. With a transmit power of up to +8dBm and receiver sensibility of down to –83dBm combined with low power consumption the F2M03GLA is suitable for the most demanding applications. Developers can easily implement a wireless solution into their product even with limited knowledge in Bluetooth™ and RF. The module is fully Bluetooth™ v2.0+EDR qualified and it is certified according to CE and FCC, which give fast and easy Plug-and-Go implementation and short time to market.

The F2M03GLA comes with an on board highly efficient omni-directional antenna that simplifies the integration for a developers Bluetooth™ solution. The high output power combined with the low power consumption makes this module ideal for handheld applications and other battery powered devices.

F2M03GLA can be delivered with the exceedingly reliable and powerful easy-to-use Wireless UART firmware implementing the Bluetooth™ Serial Port Profile (SPP).

Bidragsrapport

I början av projektet delade vi in gruppen i mindre grupper. Eftersom vi är sex personer kändes det naturligt att dela in sig i par. Vi hade regelbundna möten minst två gånger i veckan med en mötesagenda. Nedan följer gruppindelningarna och ansvarsområdena för respektive grupp.

Tolkning av ECU-boxen och USB-gränssnitt
Joanna
Johannes

Databas and Android-app
Mikael
Anders

Desktop Applikation
Philip
Ewa

I mitten av projekttiden utökade vi projektet med sensorer och givare. Detta fick till följd att vi blev tvungna att göra en ny gruppindelning. Den nya gruppindelningen och dess ansvarsområden finns beskrivna nedan.

Tolkning av ECU-boxen och USB-gränssnitt
Johannes

Android-app
Joanna
Anders(Projektledare)

Webbserver
Mikael

Desktop Applikation
Philip

Sensorer
Ewa

Mikrokontroller
Johannes

Arbetet har genomförts i smågrupperna medan mötena har ägnats åt problem, diskussion och status uppdateringar.

Utveckling av individuella bidrag följer nedan.

Joanna Eriksson - joannae@student.chalmers.se - 891122-5905

- Möte (~80 h)
- Litteraturstudier (~60 h)
  Då jag aldrig har programmerat i Android innan har en viss tid fått läggas på att lära mig detta samt att finna material för de specifika programmeringsuppgifterna som jag har haft hand om.
  Även MatLab: Tolkning av MatLab Script (av Civinco)
- Skriva rapport (~77 h)
  I denna tid ingår förutom ren rapportskrivning även översättning till engelska av delar av planeringsrapporten samt att jobba med formateringen av LaTex-dokumentet innehållandes projektrapporten
    - Introduction
        - Intro
        - Limitations
    - Mobile Application
        - Layout and Design,
            - Start- and Run-view
            - Action bar and Settings,
        - Functionalities
            - Run View
            - Calibration of sensor to measure steering
    - PHP-skript
        - Conversion to physical values
    - Discussion
        - Intro
        - Mobile Application
            - Graphical User Interface
        - System in Total
    - Conclusion
        - Intro
- Programmering(~57 h)
  Mobilapp: del av Settings-menyn (som senare inte användes), styrutslagsindikator, kalibrering styrutslagssensor
- Meka (~10 h)
- Presentation (~25 h)
  Ansvarig för halvtidsredovisning
- CTK's Bachelor's Challenge: Moment 1 & 2 (~20 h)
- Mjukvarudokumentation (~6 h) Med detta avser jag att jag gick igenom dokumentation och skrev om inför inlämning av slutrapport. Dokumentation gjordes även gemensamt i gruppen i början av projektet men den tiden har jag inte räknat in här utan på möte.
- Opponering(~17 h)
- Fackspråk (~5 h)
- Informationsteknik (~6 h)
- Projektföreläsning (~5 h)
- Övrigt (~10 h)

Total tid: 378 h

Mikael Johansson - Mikaejo@student.chalmers.se - 901106-0754

- Dokumentation ~5 timmar
- Programmering
    - Desktop applikation ~29 timmar
    - PHP ~77 timmar
    - Databasutveckling ~11 timmar
- Testning ~15 timmar
- Rapportskrivning ~27 timmar
    - Kap 5 Database
    - Kap 7 Discussion
        - Database transfer
        - Warning flags
    - Kap 8 Conclusion
        - Warning flags
- Inläsning ~39 timmar
    - Fräscha upp kunskaper om androidprogrammering
    - Fräscha upp kunskaper om javaprogrammering
    - Fräscha upp kunskaper om PHP-programmering
    - Fräscha upp kunskaper om SQL (MySQL)
    - Söka efter hjälp vid programmeringsfrågor
- Möten ~73 timmar
- Övrigt ~32 timmar
    - Planering
    - Diskutera koddesign
    - Boka mötessalar
    - Utvecklingsmiljö-problem
- Presentationer ~22 timmar
- Fackspråk ~5 h
- Informationsteknik ~6 h
    - 2 tillfällen
    - 1 hemuppgift
- Föreläsningar ~5 timmar

Totalt ~346 timmar

Anders Nordin - anordin@student.chalmers.se - 910122-5036

- Projektledning och administration(~19h): Ansvarig för styrdokument, kontakt med handledare , mötesbokningar, ansvarsfördelning, övrigtrelaterat.
- Inlärning och söka material(~48h): Lärde mig bl.a. Android och andra mindre delar.
- Dokumentation(~8h): kommentering, projekt-dokument
- Testning(~18h): Praktiska tester av funktionalitet i appen
- Databasmodellering(~2h): ER-diagram, SQL Queries
- Design(~27h)
    - Skapa affisch(~8h)
    - App-ikon och App-logo(~9h)
    - App-design(~10h)
- CTK Bachlor's Challenge: Moment 1(~8h)
- Programmering(~65h): GPS, varvtid, medelhastighet, start och stopp, total tid, nuvarande hastighet, överföring av zip-filer, reset, exit, felsökning.
- Skriva rapport(~58h)
    - Acknowledgements
    - Introduction
        - Background
        - Limitations
    - Mobile Application
        - Intro
        - Functionalities(except: Calibration of sensor to measure steering, USB Connection and Bluetooth Connection)
    - Kap 7 Discussion
        - Mobile application
            - Intro
            - Calculation of a new lap
            - Heap size error
    - Editorial Work
- Möten(~75h)
- Föreläsningar(~7h)
- Fackspråk(~5h)
- Felsökning av styrbox till telefon(~24h)
- Presentationer(~22h)
    - 2 tillfällen(~7h+6h)
    - Halvtidsredovisning-förberedelser(~9h)
- Biblioteksövning(~6h)
- Opponering(~14h): Opponeringsrapport, inläsning, muntligt framträdande.

Total tid: ~406 h

Ewa Simpanen - ewas@student.chalmers.se - 871017-4882

- Möte: 77 h
  På mötena har vi allihopa planerat, diskuterat, bokat handledartider, skrivit på rapporter, designat, kodat och jobbat med presentationer bland annat.
- Litteraturstudier: 88 h
  Urvalsprocess och söka material om sensorer, arduino, installationsguider, bilens egenskaper och allt som hör därtill. Läsa in sig på materialet samt även undersöka och lära sig vilka sensordata som är relevanta och viktiga för projektet.
- Skriva rapport: 96 h
  Ansvarig för LaTex-dokumentet innehållandes projektrapporten. Jag har lagt upp all grundkod och sett till så att rapportens struktur hålls riktig. Dessutom har jag tagit hand om och varit en hjälp vid diverse editering av dokumentet.
  - Introduction
    Omarbetning av intro, 1.1, 1.2, 1.3 efter opposition.
  - Sensors
    Intro, 4.1, 4.2 och alla underliggande stycken.
  - Discussion
    Sensors 7.2.
  - Conclusion
    Additional sensors 8.1.3.
- Installation av sensorer: 47 h
  Löda, testa och installera sensorer i bilen.
- Presentation: 38 h
  Ansvarig för bibliotekspresentationen och projektets slutpresentation.
- Programmering: 3 h
- Projektföreläsningar: 6 h
- Övrigt: 43 h
  Övrigt innefattar administrativa göromål så som tidsbokning, mail, tidrapportering m.m. samt ärenden hos handledare, opponering och bidragsrapport.
- Fackspråk: 5 h
- Informationsteknik: 6 h

Total tid: 409 h

Philip Steingrüber - 891022-4834 - phiste@student.chalmers.se

- Dokumentation (Desktop) ~25 timmar
- Programmering
    - Desktop applikation ~105 timmar
    - PHP ~1 timme
- Testning ~5 timmar
- Rapportskrivning ~65 timmar
    - Abstract/Sammanfattning
    - Kap 1 Introduction
        - Method
        - Task
    - Kap 5 Database
    - Kap 6 Desktop application
    - Kap 7 Discussion
        - Database transfer
        - Desktop application
    - Kap 8 Conclusion
        - Plotting GPS Coordinates
        - Remotely modifying engine parameters
    - Stora mängder korrekturläsning och redaktionella ändringar
- Inläsning ~35 timmar
    - Främst leta lösningar på programmeringsproblem
    - En del inläsning av androidprogrammering
- Möten ~65 timmar
- Övrigt ~25 timmar
    - Planering
    - Diskutera koddesign
- Slutpresentation ~15 timmar
- Fackspråk ~4 timmar handledning (2 tillfällen)
    - ~2 timmar kompletteringsuppgift
- Föreläsningar ~5 timmar

Totalt ~350 timmar

Johannes Weschke 840829-5015 johwesc@student.chalmers.se

- Möte (79h)
- Föreläsningar (5h)
- Söka Material (15h)
  Letat efter information att läsa på om de olika delarna jag jobbat på.
- Inläsning(13h)
  - Läst material om Android programmering (Android Tutorials och Android skola från internet), hade endast gjort en liten app innan.
  - Läst material om Arduino programmering, inte använt mig av innan.
- Skriva rapport (45h)
  - Mobile application
  - USB Connection
  - Bluetooth Connection
  - ECU
  - Sensors
  - Microcontroller
  - Discussion - ECU
  - Korrekturläsning
- Koda(79h)
  - Tagit hand om all programering till och från ECU:n.
  - Skrivit Android-applikations delarna som tar hand om kommunikation till och från styrboxen och som omvandlar datan till värden.
  - Skrivit Android-applikationens delar som ansluter med blåtand till Arduinon och som tolkar datan därifrån.
  - Skrivit Android-applikations delen som samlar ihop datan från Microcontrollern och ECU:n och lägger ihop till ett data paket för avläsning.
  - Skrivt Android-applikations delen som zippar datafilerna och gör dom redo för att skickas.
  - Hjälpt till med Android-applikations delen som skickar datan till servern.
  - Tagit hand om all programmering till Microcontrollern (Arduinon) och sensorerna.

- Meka(92h)
  - Kopplat ihop sensorer och Arduino.
  - Mycket testning med hårdvaror och felsökningar.
  - Rootat Android telefon för omprogrammering av kerneln så att det skulle fungera att använda telefonen som en USB-host.
  - Hämta testbänk av Civinco.
- Testning(6h)
  - Testning av appliaktionens olika delar
- Övrigt(44h)
  - Kontakt med Arne
  - Kontakt med Markus på Civinco
  - Material letande (hårdvara) så som Arduino, sensorer, telefon m.m.
- Diskussioner om problemlösning och nya idéer.
- Köpa hårdvara som behövts i Farnell butiken.
- Fackspråk(8h)

- Presentation(23h)
- Informationsteknik(6h)

Totalt tid: 415 h