

Development of a Remote Sensor Network to Monitor a Solar Power System

Zenville Erasmus

Project presented in fulfilment
of the requirements for the degree of
B.Sc. (Honours) of Computer Science
at the University of the Western Cape

Supervisor: Professor Antoine Bagula

October 2014

Declaration

I, ZENVILLE ERASMUS, declare that this project “*Development of a Remote Sensor Network to Monitor a Solar Power System*” is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signature:

Date: 24th October 2014.

ZENVILLE ERASMUS.

Abstract

The purpose of this project is to discuss the development of a cost effective remote sensing network (RSNET) for the continuous acquisition of remote energy yields and performance measures of a network of installed solar power systems. As a new innovative solution that demonstrates a low cost mechanism using the existing mobile network infrastructure, the *remote sensor system* to be developed presents the following key benefits:

- Access to Photovoltaic (PV) system from anywhere through the use of the Internet.
- Reports of power output and energy production trends.
- Verification of system operation.
- Collection of data for service and maintenance planning.
- Use of least cost devices to enable replicability of the solution.

The main features of the *remote sensor network* include:

1. solar power consumption monitoring using sensors which measure panel voltage and current capture
2. information dissemination using SMS and GPRS protocols
3. data publishing using Web services based on PHP and associated graphing tools
4. and situation recognition (awareness and prediction) using machine learning techniques and/or statistical analysis methods.

Key words

Remote Sensor Network

Solar Power System

Photovoltaic

Remote sensor system

Information dissemination

Situation recognition

Machine learning techniques

Statistical analysis methods

Acknowledgements

This project is a compilation of the efforts of many people that helped me through the year. I would first like to thank my supervisor, Prof. Antoine Bagula and Prof. G. Swart for supporting and encouraging me during my Honours year. Without our weekly meetings, this work would not have been possible. Without their help, this project would not have reached a developmental milestone. I would also like to thank The Namibian Student Financial Assistance Fund (NSFAF) and the UWC Computer Science department for their unwavering financial assistance without which our efforts would not have been possible.

Contents

Declaration	iii
Abstract	v
Key words	vi
Acknowledgements	vii
List of Figures	xii
Glossary	xiv
1. Introduction	1
1.1 Motivation	1
1.2 Contribution and Outline	1
2. User Requirements	4
2.1 Introduction	4
2.2 User's view of the problem	4
2.2.1 The storage of renewable energy	4
2.2.2 Measuring readings	4
2.2.3 Technology used to take measurements	5
2.2.4 Traffic	5
2.3 Description of the problem	5
2.3.1 The storage of renewable energy	5
2.3.2 Measuring readings	5
2.3.3 Technology used to take measurements	5
2.3.4 Traffic	6
2.4 Expectations from a Software Solution	6
2.4.1 Interaction with Users	6
2.4.2 Measuring readings and Technology used	6
2.4.3 Traffic	6
2.5 Conclusion	7
3. Requirements Analysis	8
3.1 Introduction	8
3.2 Designer's interpretation of the problem	8
3.3 Software and programming tools needed to develop this system	9

3.4	Existing solutions	10
3.4.1	An evaluation of the Solar Monitoring System in Malawi	10
3.4.2	SIMbaLink: Towards a Sustainable and Feasible Solar Rural Electrification System	11
3.5	Alternative technical solutions	12
3.5.1	Indoor solar energy harvesting for sensor network router nodes	12
3.6	Suggested Solution	13
3.7	Conclusion	13
4.	User Interface Specification	15
4.1	Introduction	15
4.2	Interface appearance	15
4.2.1	Web Portal	15
4.2.2	Android Application	20
4.3	Conclusion	20
5.	Object Oriented Analysis or High Level Design	21
5.1	Introduction	21
5.2	Data dictionary	21
5.3	Relationships between objects	22
5.4	Full solution components	23
5.5	Conclusion	23
6.	Object Oriented Design or Low Level Design	24
6.1	Introduction	24
6.2	Low Level data dictionary	24
6.3	Low Level Design	26
6.4	Conclusion	26
7.	Documentation	27
7.1	Introduction	27
7.2	Project Construction	27
7.3	Sensors and their Metrics	30
7.3.1	Precision Voltage Sensor	30
7.3.2	30 Amp Current Sensor AC/DC	30
7.3.3	Humidity/Temperature Sensor	31
7.3.4	Light Sensor 70 000 lux : Luminosity sensor	31
7.3.5	Phidget Interface Kit 8/8/8	32

7.4	Voltage Sensor Setup	34
7.5	Current Sensor Setup	35
7.6	Raspberry Pi	36
7.7	Raspbian installation	37
7.8	State of Charge	39
7.9	Code Documentation	40
7.9.1	Gathering data from sensors	41
7.9.2	Sensing with the Raspberry Pi	42
7.9.3	Android function to retrieve readings from MySQL database	42
7.9.4	Android Application Execution	44
7.10	Conclusion	45
8.	Testing and Results	46
8.1	Introduction	46
8.2	Testing procedure	46
8.3	White-Box testing - Static Testing	46
8.4	Stress Testing	47
8.5	Performance Testing	47
8.6	Testing results	47
8.6.1	Environment - Lab vs outside	47
8.6.2	Environment - Phidget Sensor vs Lab Thermometer vs Air-conditioner Remote	48
8.6.3	Phidget Sensors vs Accuweather	49
8.6.4	Panel voltage - Lab vs outside	49
8.6.5	Phidget Sensors vs Multimeter	49
8.6.6	Charge Controller Terminals	50
8.6.7	SMS Notification	51
8.6.8	Luminosity results	52
8.6.9	Averages of last 2 days	53
8.7	Conclusion	53
9.	User Guide	54
9.1	Introduction	54
9.2	Sensors and Phidget Interface Kit	54
9.3	Raspberry Pi	54
9.4	Web Portal	55
9.5	Android Application	56
9.6	Conclusion	56

10. Conclusion	57
Bibliography	58

List of Figures

3.1	System Architecture: Solar PV with Wireless Sensor and Central Management Servers (Source: SM ² . Image by Nkoloma)	10
3.2	4 data readings per day based on the SIMbaLink System (Source: SM ² . Image by Nkoloma)	11
3.3	Integration of SIMbaLink with the other components of a solar home system (Source: SIMbaLink. Image by Schelling)	12
4.1	Login screen for accessing the Web Portal	16
4.2	Remote sensor system details on Web Portal	16
4.3	Remote sensor system state of charge	17
4.4	Remote sensor system live bar graph	17
4.5	Remote sensor system averages of last 3 days	18
4.6	Remote sensor system averages of daily luminosity	18
4.7	Remote sensor system averages of sensor readings plotted against each other	19
4.8	Remote sensor system details on Android App	20
5.1	Input to output	22
5.2	System processing	23
6.1	Sensing state diagram	26
7.1	Remote Sensor System / Network	27
7.2	Circuit diagram / Schematic / Layout of prototype	28
7.3	Pocket Power	29
7.4	Precision Voltage Sensor	30
7.5	30 Amp Current Sensor AC/DC	30
7.6	Humidity/Temperature Sensor	31
7.7	Light Sensor 70 000 lux: Luminosity sensor	32
7.8	Phidget Interface Kit 8/8/8	32
7.9	Voltage sensor in parallel	34

7.10	Voltage sensor schematics	34
7.11	Current sensor connected in series	35
7.12	Current sensor schematics	35
7.13	Raspberry Pi Model B	36
8.1	Incremental approach	46
8.2	Battery leaching vs AC Power	48
8.3	Ambient Temperature (°C) and Relative Humidity (%)	49
8.4	SMS Notification sent out for an approximal 0% SOC reading	51
8.5	Average lab luminosity readings on 2 Oct 2014	52
8.6	Averages of last 2 days	53

Glossary

RSNET	Remote Sensing Network
PV	Photovoltaic
SMS	Short Message Service
GPRS	General Packet Radio Services
PHP	PHP Hypertext Preprocessor
NSFAF	Namibia Student Financial Assistance Fund
URD	User Requirements Document
RAD	Requirements Analysis Document
GSM	Global System for Mobile communications
SAIAMC	South African Institute of Advanced Materials Chemistry
SHS	Solar Home System
WSN	Wireless sensor networks
ECGs	Electrocardiograms
OOA	Object Oriented Analysis
HLD	High Level Design
OOD	Object Oriented Design
LLD	Low Level Design
SBC	Single Board Computer

Chapter 1

Introduction

The goal of this project is to discuss the development of a remote sensing network (RSNET) which will continuously acquire energy yields and performance measures of renewable energy solar power systems.

1.1 Motivation

Since sensor technology is a new field that is receiving global research and since renewable energy technologies are on the rise, the combination of the two would allow the monitoring of the latter mentioned system from remote locations. This would most certainly eliminate the need for an engineer to travel to far off locations, connect a PC or laptop to present monitoring devices and download the data manually.

Since the Photovoltaic system could be accessed from any location due to the proliferation of web technologies, an engineer would instead have a detailed data flow of power outputs and energy production trends. The system's operation could also be verified to ensure data integrity and this would also lead to the scheduling of service and maintenance planning. In order to release such a system for market purposes, a least cost solution has to be considered in order to replicate the solution to multiple sites.

1.2 Contribution and Outline

The project discusses the conception of a *remote sensing network* for the SA-IAMC¹ Li-ion Battery Product Development program. Their objectives state that they want to develop affordable batteries technology based on South African raw materials to meet the energy storage demands for renewable energy, smart grid and cleaner transportation markets. It is within this market, that the project focuses its attention.

¹South African Institute of Advanced Materials Chemistry

In this project we design and implement a RSNET which will gather sensor information from a battery system and relay the data for data publishing purposes, situation recognition, using machine learning techniques and/or statistical analysis methods.

The project has ten chapters. This first chapter provides an introduction to the project. The second chapter discusses the User Requirements Document (URD). Chapter 3 discusses the Requirements Analysis Document (RAD) that has been used in the RSNET and describes the technologies and the reasons why they have been chosen. The fourth chapter discusses the user interface specification, which namely focuses on the web portal and android application. Chapter 5 illustrates the high level design of the project and chapter 6 follows up with the low level design. The implementation documentation is discussed in chapter 7 followed by testing and results in chapter 8. Chapter 9 will include a user guide to indicate how the system is to be setup and used. The final chapter concludes and gives an overview of the project and suggests possible future work which may be undertaken.

Chapter 2

User Requirements

2.1 Introduction

In this chapter we focus on the user's view of the problem and give a brief description of the problem domain. We also list similar systems in developing countries implementing similar renewable energy solutions.

2.2 User's view of the problem

2.2.1 The storage of renewable energy

In order for a renewable energy system to be efficient, the energy it generates has to be stored in a manner so that it can be harnessed at times when there exist no present generation environment. Such as when a Solar Panel can capture no feasible light during the night or when a wind turbine has no supply of wind in order to serve its purpose. Then systems connected to it rely solely on the battery for sustainability. Knowing when the battery is under performing could serve as a means to use less devices on it to have it last longer until the next generation phase.

2.2.2 Measuring readings

A user of the system would typically want to know what a renewable energy system is generating, the amount of power produced, voltage and current readings at specific times of a 24 hour day. This would give users an indication of when their system is most effective and least effective. They would also be able to determine which locations serve as a more viable source for generation capabilities. Having a system that could log such parameters would serve for establishing renewable energy systems at best known locations.

2.2.3 Technology used to take measurements

In order to implement a successful monitoring system, devices known as sensors need to be used. Using a sensor technology that is ready made, plug-'n-play-able, easy to transport and cost efficient would grant such a system a performance gap as data can be collected autonomously.

2.2.4 Traffic

Concerning data gathering which needs to happen continuously, sensors in such a remote system can relay their data to another device which can broadcast the data with the aid of different technologies. This would serve much faster than a human would if on-site with a portable computer, having to plug his machine in and download the data.

2.3 Description of the problem

2.3.1 The storage of renewable energy

Whilst being charged by a photovoltaic panel, a battery will be able to only hold a specific amount of energy. The health of the battery employed will determine how much energy it can store. And this is where the project comes into play as its purpose will serve to monitor the battery's health and storage capability as time progresses.

2.3.2 Measuring readings

When readings are taken by the monitoring system, that will indicate whether the panel is producing energy as it should, whether environmental factors are influencing the readings and whether the battery being used is in need of a service or replacement. These readings could also be monitored over a certain time period and predictions made based on past data.

2.3.3 Technology used to take measurements

The sensors to be utilized need to draw low power, need to be easily replaceable and they need to be accurate.

2.3.4 Traffic

Traffic needs to be saved to a file or database that could be presented on an online portal and then further displayed on hand held devices. If critical alerts could be sent to mobile phones that are not smart phones, all users could benefit and the system can be inspected to correct critical issues in its operation.

2.4 Expectations from a Software Solution

2.4.1 Interaction with Users

The end-user requires the remote monitoring system to be integrated into an online portal to graphically display what the system is gathering. Proper graphing tools with various graphs and charts would satisfy performance expectations. A further enhancement would serve to port the readings to personal hand-held devices, smart phones and tablets based on the Android operating system. The system should also relay critical alerts via Short Message Service (SMS) technology based on the Global System for Mobile communications (GSM) protocol. Further expansions would involve the usage of faster/better network protocols.

2.4.2 Measuring readings and Technology used

Users are not interested in mathematical notations and what rather just await the final results of system performance. Thus only the most critical information is of importance to them. The more detailed parameters would only be of use to the engineering team in charge of the monitoring system.

2.4.3 Traffic

Users do not want to be irritated by constant SMS alerts or Android notifications. They would rather only receive time critical alerts/notifications and would rather prefer to view the online portal when it suits them.

2.5 Conclusion

In this chapter we had a look at the user's point of view concerning a battery system for a photovoltaic panel. We also delved into how measurements are to be taken and the technologies used to do so. Lastly, we focused on how traffic should be handled.

The above topics were further described and expectations for a solution were determined. In Chapter 3, we next discuss the designer's interpretation of the user's requirements. We basically identify the "real" problem(s).

Chapter 3

Requirements Analysis

3.1 Introduction

This chapter takes the user's requirements and evaluates them from a designer's perspective. Since a software designer is considered to be astute in his/her knowledge of the types of systems and solutions that perform better in various environments, his/her perspective is important in the projects design phase.

This chapter will identify the software systems and paradigms that will best fit the user's requirements. Please note that a detailed design solution will not be given, yet the most beneficial means for an implementation will be identified.

3.2 Designer's interpretation of the problem

Renewable energy systems are a key ingredient to reducing green house emissions and turning the tide on global warming. The current initiative of linking a battery or battery pack to a photovoltaic system would allow the storage of produced energy for times when that energy may be needed. Such as during the evenings when the moon's shadow is overcast of a region and no usable light is available for the solar system.

With the battery concept, devices dependent on the solar system will be able to function during the course of an entire day and will grant a service that does not contribute to global warming. Thus, the initiative by SAIAMC is very innovative as a first by South Africa in developing a new form of a lithium-ion battery.

The project will focus on determining the health of such a battery, its performance when charging and when discharging. This would serve as a valuable tool in knowing if the battery can be released to markets and also

the expected life span that it could have.

Readings that are of great interest in such a solar panel, battery implementation is the the monitoring module's current, the module's voltage, charging current, discharging current, battery voltage, load current and load voltage. These readings would give an indication of whether the battery used is working as expected. They would also give us information pertaining to the environment in which the solar panel and battery system is located. Information that could be critical is the temperature of the region wherein which the battery is seated, the relative humidity which could indicate a possible oncoming shower, the lux - the amount of light reaching the light sensor at a particular time of the day(indicating whether it is night or day), and the altitude if necessary by a barometric sensor.

3.3 Software and programming tools needed to develop this system

- libusb development libraries - A C library that gives applications easy access to USB devices on many different operating systems. libusb is an open source project [1].
- Phidget libraries - A library for a user-friendly system available for controlling and sensing the environment [2].
- Phidget Python library - The python library for communicating with phidgets.
- Python - A widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability and its syntax allows programmers to express concepts in fewer lines of code than would be possible in other languages such as C.

3.4 Existing solutions

3.4.1 An evaluation of the Solar Monitoring System in Malawi

The project output gives direct access to generated electric power at the rural site through the use of wireless sensor boards and text message (SMS) transmission over a cellular network. The SMS recipient at the central site houses an intelligent management system based on FrontlineSMS for hosting SMSs and publishing remote measurement trends over the Internet. Figure 3.1, gives an indication of the system implemented in Malawi.

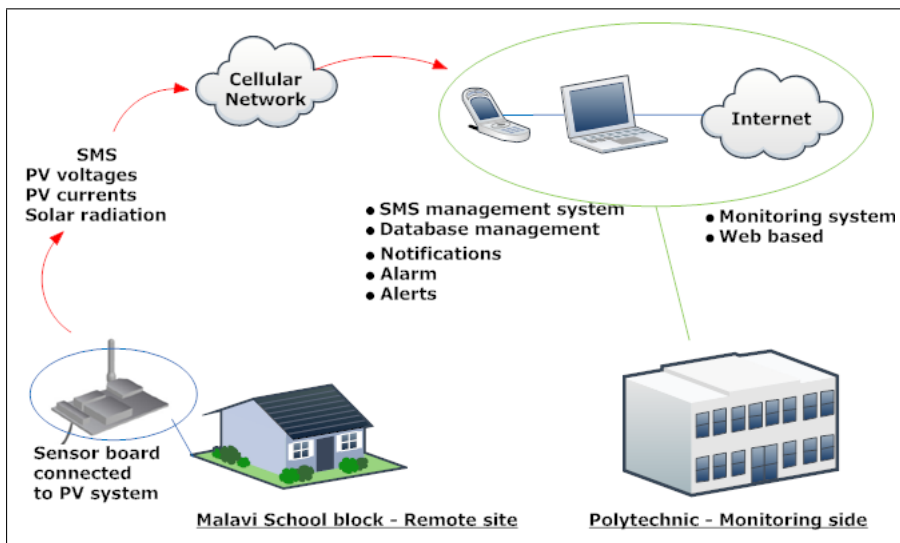


Figure 3.1: System Architecture: Solar PV with Wireless Sensor and Central Management Servers (Source: SM². Image by Nkoloma)

The SIMbaLink project aims to provide sustainable electrification solutions for rural areas. SIMbaLink is based on an extremely low cost real time solar monitoring system that reduces the maintenance costs and the time to repair. The system reveals important information about the battery's state of charge and daily energy usage. Data is transmitted over GSM cellular networks to a regional technician to allow remote system diagnostics. However, data readings are only taken 4 times per day, as depicted by Figure 3.2, and this does not enable real time trends to enable critical performance analysis and timely detection of solar plant problems [3].

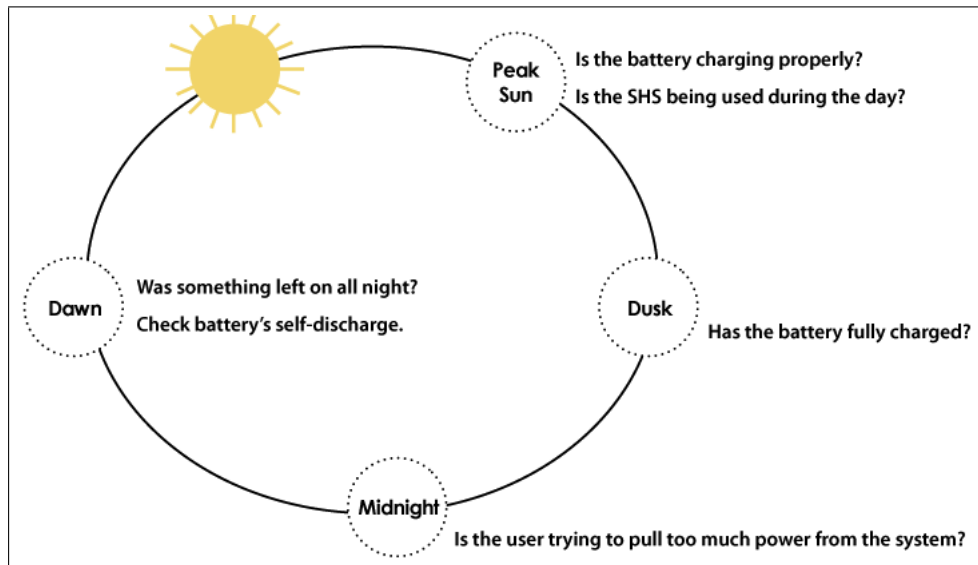


Figure 3.2: 4 data readings per day based on the SIMbaLink System (Source: SM². Image by Nkoloma)

3.4.2 SIMbaLink: Towards a Sustainable and Feasible Solar Rural Electrification System

The project makes use of a website that acts as an easily accessible platform containing in-depth information regarding each individual Solar Home System (SHS). An SMS system is used to increase the frequency of data gatherings or to run an intensive 24-hour diagnostic on that system. Software is used to group the SHS geographically to plan technicians maintenance visits in a way that maximizes time, materials needed and operational costs. Availability of GSM serves as a key to reach rural consumers that were previously inaccessible without high operational costs. Sending an SMS over the GSM network is currently the best option for retrieving data in rural areas [4]. Figure 3.3, shows a sketch of the SIMbaLink module used in rural electrification systems together with the charge controller, PV panels, the connected components to the system and the rechargeable battery in use.

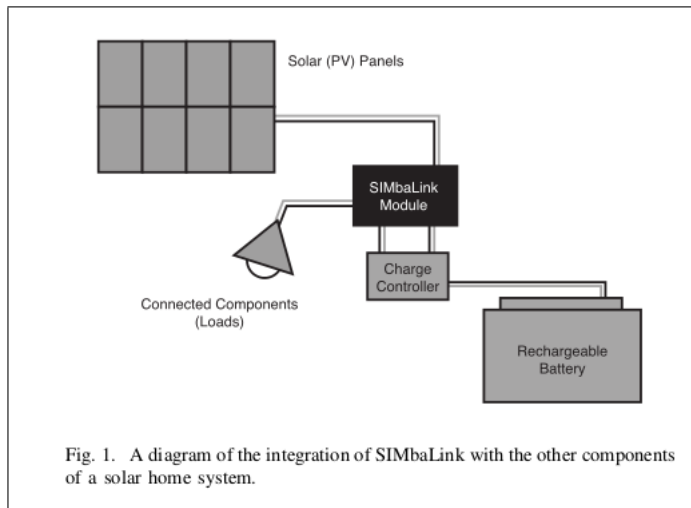


Figure 3.3: Integration of SIMbaLink with the other components of a solar home system (Source: SIMbaLink. Image by Schelling)

3.5 Alternative technical solutions

3.5.1 Indoor solar energy harvesting for sensor network router nodes

The development of a unique method to scavenge energy from monocrystalline solar cells to power wireless router nodes used in indoor applications. The system's energy harvesting module consists of solar cells connected in series-parallel combination to scavenge energy from 34W fluorescent lights. A set of ultra capacitors were used as the energy storage device. Two router nodes were used as a router pair at each route point to minimize power consumption.

Test results show that the harvesting circuit which acted as a plug-in to the router nodes manages energy harvesting and storage, and enables near-perpetual, harvesting aware operation of the router node. In-hospital WSN can be used to monitor patient vital sign data from instruments such as electrocardiograms (ECGs), pulse oximeters and blood pressure (BP) monitors. These units can be interfaced to WSN nodes that are programmed as sensor nodes. The sensor nodes are required to perform the function of sensing vital sign data from the patient and are typically required to be ambulatory in nature.

Therefore, it is more convenient to allow them to run on stable sources of energy such as batteries. The maximum range of data transmission for such nodes is approximately 10 meters. Thus additional nodes may be required to

pass the data back to the central monitoring location. These router nodes need to be continuously on so that data can be promptly transferred. Since hospitals have fluorescent lights in the hallways that are always on, it is advantages to operate these router nodes by scavenging light energy. This would result in huge cost savings over time (this arrangement eliminates the need to monitor and replace batteries) [5].

3.6 Suggested Solution

The suggested solution proposes connecting sensor devices to a board that can accept multiple sensors. This board will then be further connected to a device with a Linux operating system on it. The operating system together with a python script will collect the readings from the sensors and categorize them into a neat fashion. The latter mentioned board will also be responsible for broadcasting the data via SMS and Wifi. The Wifi implementation for uploading the data to an online web portal and the SMS implementation for sending critical and time dependent messages to a user for monitoring purposes.

The web portal will act as a monitoring and graphing solution to aid in visually displaying the data, meeting the user's requirements. The SMS implementation will serve for instant notifications of system errors and 4-hourly categorized system performance measures.

3.7 Conclusion

In this chapter we had a look at the designers view of the problem at hand. We also listed software and programming tools that are needed to develop the system. A few existing solutions and their implementation we highlighted. Since alternative solutions are almost always available, one was listed to enhance our knowledge base. Lastly we rounded the chapter off with a suggested solution for the development/implementation of a remote sensor network to monitor a solar power system. In the next chapter, we will discuss the specifications of a user interface for the project.

Chapter 4

User Interface Specification

4.1 Introduction


This chapter describes what the user interface is going to do. Its look and feel will be mentioned as well as how the user interacts with it. Since our user, SAIAMC, places emphasis on a well designed web interface and mobile application, the look of a web portal and android application will be discussed. How our user interacts with the remote sensor system will also be mentioned.

4.2 Interface appearance

4.2.1 Web Portal


The web portal will serve as a platform whereby our client can connect to via the internet from any personal computer, laptop, smart phone or tablet. It will have a log in screen for administrative purposes whereby a user name and password will be needed for security reasons. Figure 4.1 shows the login screen.

Login to Remote Monitoring System



Username:

Password:



UNIVERSITY of the
WESTERN CAPE

Figure 4.1: Login screen for accessing the Web Portal

After logging in the following details will be viewable as presented by Figure 4.2.

Battery and Environmental Readings						
Date/Time	Ambient Temperature	Relative Humidity	Luminosity	DC Volt(s)	DC Current	Power
yyyy-mm-dd hh-mm-ss	degrees Celcius	%	Lux	Volt(s)	Amp(s)	Watt(s)
2014-08-08 17:48:00	23.56	38.14	422.24	11.753	0.530	6.229090
2014-08-08 17:47:00	23.56	37.95	412.25	11.878	0.530	6.295340
2014-08-08 17:46:00	23.78	37.95	422.24	11.789	0.530	6.248170
2014-08-08 17:45:02	23.56	38.14	422.24	12.093	0.530	6.409290
2014-08-08 17:43:28	23.56	38.14	402.50	11.735	0.530	6.219550
2014-08-08 17:42:28	23.56	38.14	412.25	11.860	0.606	7.187160
2014-08-08 17:41:27	23.56	38.33	402.50	11.717	0.530	6.210010
2014-08-08 17:40:27	23.56	38.14	412.25	11.771	0.606	7.133226
2014-08-08 17:39:28	23.56	38.33	412.25	11.753	0.530	6.229090
2014-08-08 17:38:28	23.56	38.14	402.50	11.753	0.606	7.122318

Figure 4.2: Remote sensor system details on Web Portal

The information on the web portal is fetched from a database which is updated every second if an internet connection is available. The database is fed data over Wifi if the connection is feasible to broadcast the readings obtained from the remote sensor system.

Along with these readings, a state of charge is displayed and updated every 10 seconds as illustrated by Figure 4.3.

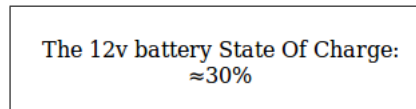


Figure 4.3: Remote sensor system state of charge

Figure 4.4 illustrates a live bar graph of the readings as they are retrieved from a database every second. These visual readings give an easy on the eye indication of the system's performance.

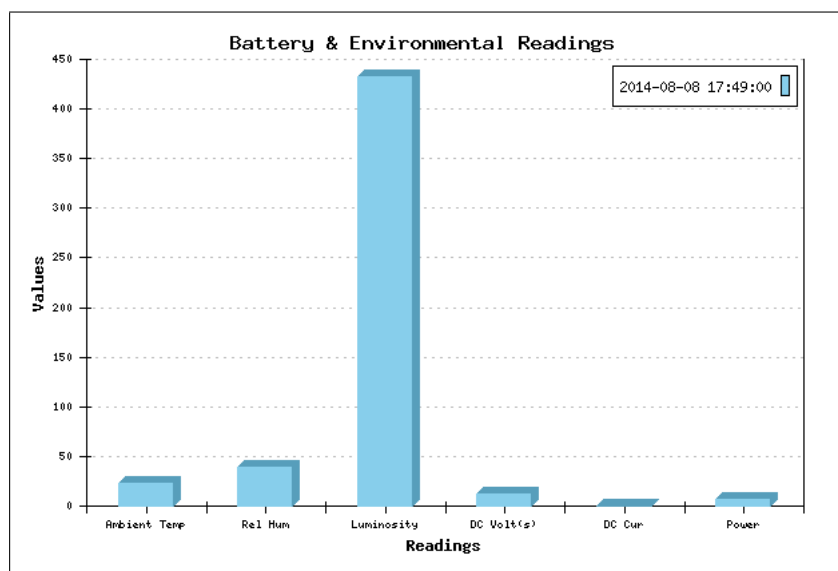


Figure 4.4: Remote sensor system live bar graph

The web portal further supports displays of averages of the last 3 days, averages of daily luminosity and averages of the sensor readings plotted against each other. These are graphically depicted by Figures 4.5, 4.6 and 4.7.

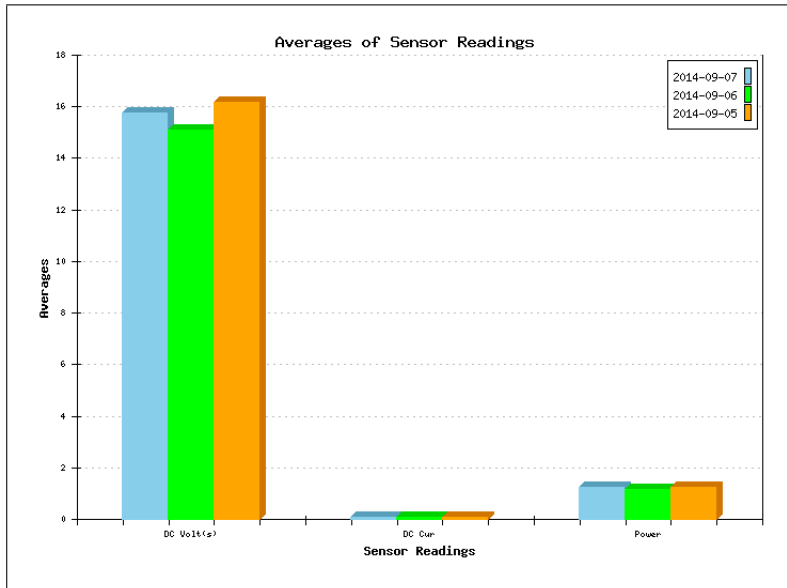


Figure 4.5: Remote sensor system averages of last 3 days

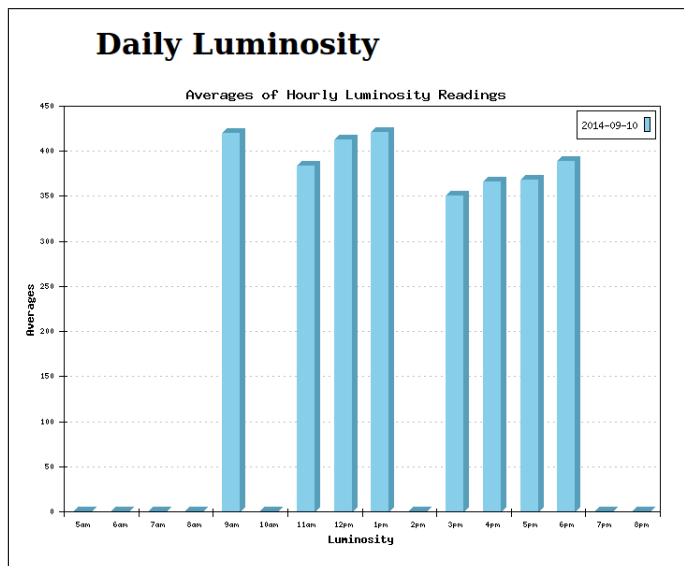


Figure 4.6: Remote sensor system averages of daily luminosity

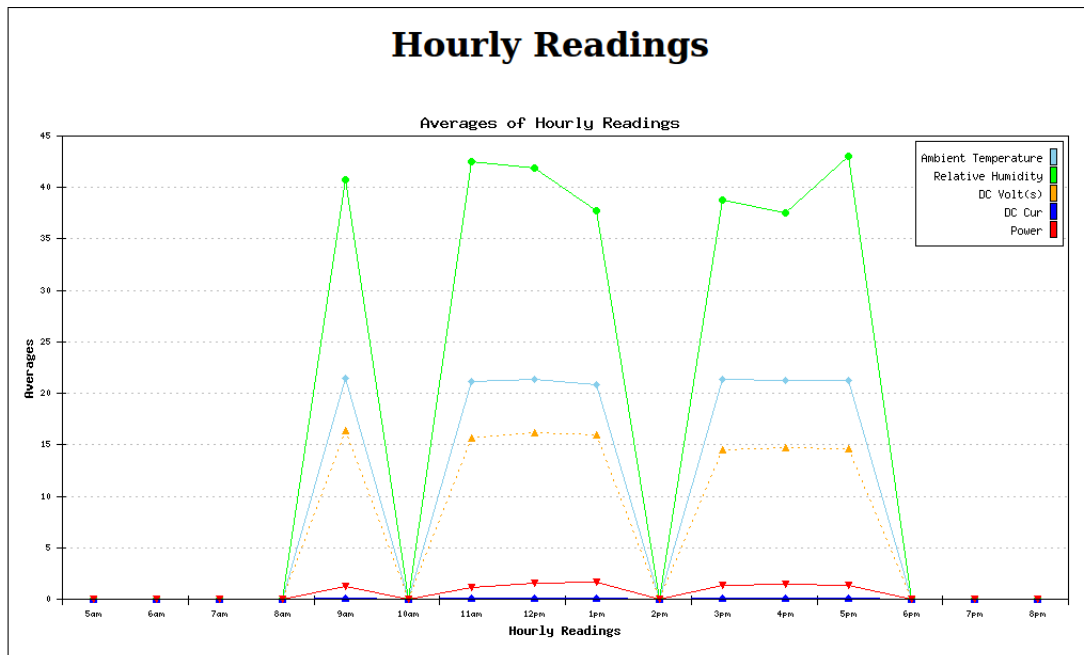


Figure 4.7: Remote sensor system averages of sensor readings plotted against each other

Readings from Figures 4.6 and 4.7 are shown from 5 a.m. to 8 p.m. in increments of one hour.

4.2.2 Android Application

The android application will serve to allow users to view the status of their remote solar powered system. It will also give detailed specifications as to the battery's parameters as well as external conditions. Figure 4.8 lists these details.

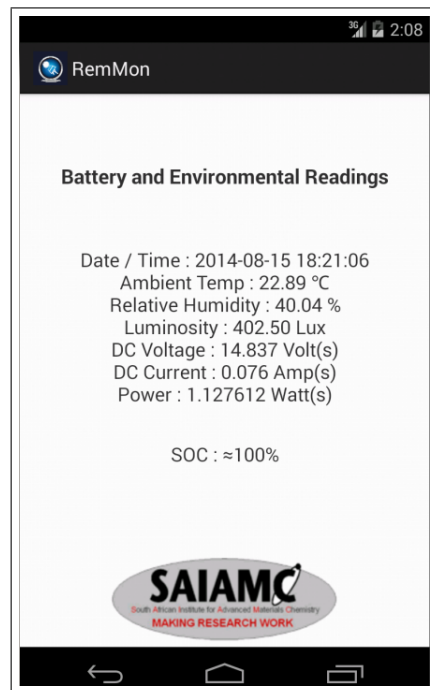


Figure 4.8: Remote sensor system details on Android App

As with the web portal, the android application will also fetch data from a remote database. The application, however, will also allow users to receive notifications when low performance measures are to be reported. Users can therefore receive the readings of their solar powered system from any internet enabled means by simply opening the application on their phones and viewing the details.

4.3 Conclusion

This chapter detailed the look and feel of the user interfaces for the remote solar monitoring system. The interfaces' behaviour was also mentioned and how users would interact with them. The next chapter will detail the Object Oriented Analysis (OOA) or High Level Design (HLD).

Chapter 5

Object Oriented Analysis or High Level Design

5.1 Introduction

This chapter focuses on Object Oriented Analysis or High Level Design of the remote monitoring system. The OOA will show case a data dictionary defining what each object/noun represents. The HLD will give a detailed breakdown of the technical solution involved in each subsystem. It will also show case detailed interactions between interface subsystems.

5.2 Data dictionary

The following contains objects and their descriptions. These objects were used during the development of this research project.

Object	Description
libusb development libraries	A C library that gives applications easy access to USB devices on many different operating systems. libusb is an open source project.
Phidget libraries	Libraries for a user-friendly system available for controlling and sensing the environment.
Phidget Python library	The Python library for communicating with phidgets.
Python	A widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability and its syntax allows programmers to express concepts in fewer lines of code than would be possible in other languages such as C.

Table 5.1: Objects/Nouns and their descriptions

5.3 Relationships between objects

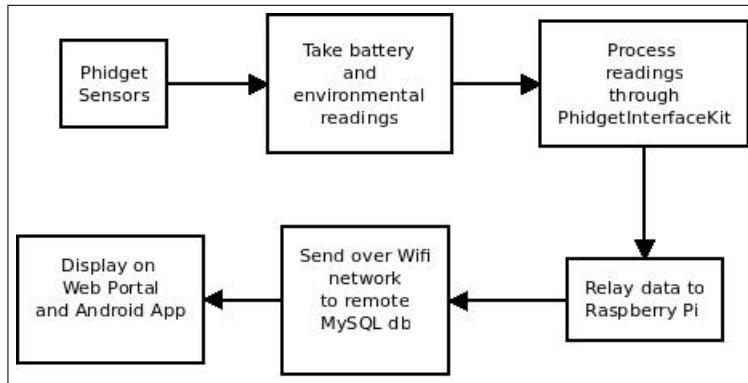


Figure 5.1: Input to output

Figure 5.1 gives a clear indication concerning how the battery's readings will be processed for display. The DC current and voltage are of importance to determine the power output. External parameters such as ambient temperature, relative humidity and luminosity will give an indication of the environment in which the solar system is operating in. The figure also lists how the data will be populated in a MySQL database for display purposes. This database will reside on a remote server and will be used to populate the web portal. The same database will also be used to send information to the android application. Note that the language PHP will be used to communicate with the database and display the data on the web portal. Java/Android like-syntax will be used to communicate with the database from mobile enabled android devices.

5.4 Full solution components

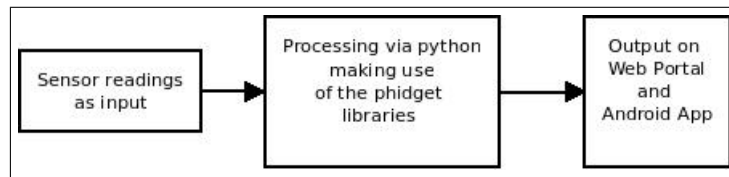


Figure 5.2: System processing

Figure 5.2 highlights how the system will take in sensor readings and process those readings via the python programming language. The said language makes use of the phidget libraries to complete the processing operations. The data is then further relayed for output purposes by intermediate steps in the systems' cycle.

5.5 Conclusion

This chapter concludes the Object Oriented Analysis and High Level Design. A data dictionary was defined with objects/nouns of substance to the project. Objects and the relationships between them were shown as well as solutions to the system's full components. The next chapter will delve into the Object Oriented Design (OOD) or Low Level Design (LLD).

Chapter 6

Object Oriented Design or Low Level Design

6.1 Introduction

In this chapter a low level design will be applied to the problem. Details will be given about the data types and functions. Pseudo code will be shown as well as algorithmic descriptions.

6.2 Low Level data dictionary

The following table contains the system components and concepts that are going to be used.

Class/Module	Attributes
datetime	A python module that supplies classes for manipulating dates and times in both simple and complex ways.
time	time provides various time-related functions.
InterfaceKit	This class represents a Phidget Interface Kit. All methods to read and write data to and from an Interface Kit are implemented in this class.

Table 6.1: Classes/Modules and their attributes

Table 6.2 lists the methods/functions to be used and their details.

Methods/Functions	Details
<code>sleep(t)</code>	Suspends execution for the given number of seconds, <i>t</i> . The argument may be a floating point number to indicate a more precise sleep time.
<code>openPhidget()</code>	Open a Phidget with or without a serial number. Open is pervasive. You can call open on a device before it is plugged in and keep the device opened across device dis-connections and re-connections.
<code>getSensorValue()</code>	Returns the value of an analog input. The analog inputs are where analog sensors are attached on the InterfaceKit 8/8/8. On the Linear and Circular touch sensor Phidgets, analog input 0 represents position on the slider. The valid range is 0 - 1000. In the case of a sensor, this value can be converted to an actual sensor value using the formulas provided in the sensor product manual [6]. Returns the sensor value in (int).
<code>getSensorRawValue()</code>	Returns the raw value of an analog input. This is a more accurate version of <code>getSensorValue</code> . The valid range is 0 - 4095. Note however that the analog outputs on the InterfaceKit 8/8/8 are only 10-bit values and this value represents an oversampling to 12-bit. Returns the sensor value in (int).
<code>closePhidget()</code>	Closes this Phidget. This will shut down all threads dealing with this Phidget and you won't receive any more events.

Table 6.2: Methods/Functions and their details

6.3 Low Level Design

On the low level of design, sensing with the Phidgets will be depicted by Figure 6.1 since it represents the first part of the project.

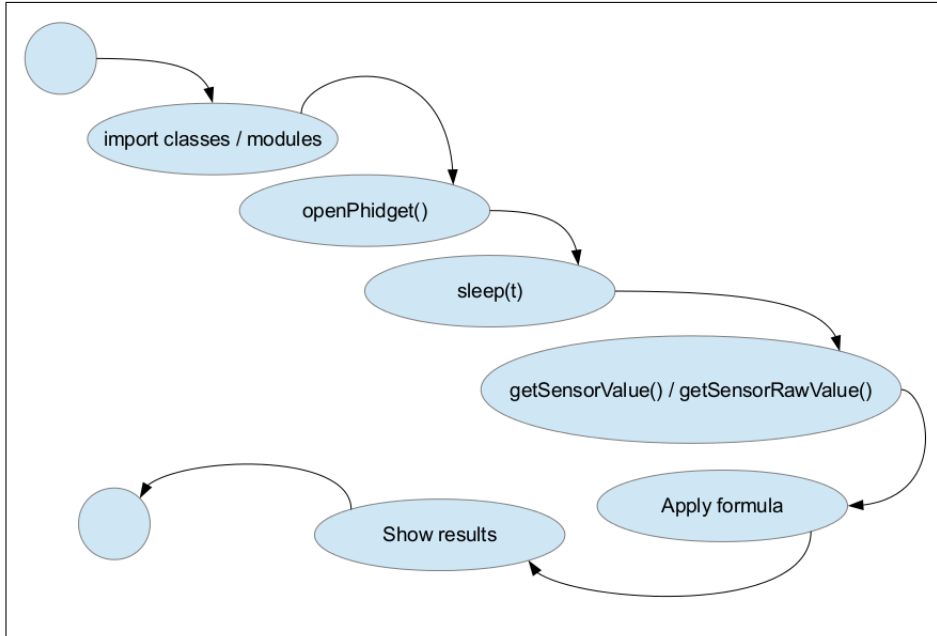


Figure 6.1: Sensing state diagram

The above figure can also be expressed algorithmically. See below.

Algorithm 6.1 Sense battery and environment readings

```

import or load the required classes or modules for usage
open the Phidget InterfaceKit
sleep - give the Phidget InterfaceKit time to open
while true do
  retrieve the sensors values
  apply the necessary formulas to each sensors' values
  retrieve and display the results
  repeat the procedure
end while
close the Phidget InterfaceKit

```

6.4 Conclusion

This chapter examined the project from an Object Oriented Design (OOD) and a High Level Design (HLD). The most interesting details revolve around the state diagram shown and its accompanying pseudo code/algorithmic description. The next chapter will focus on the code documentation.

Chapter 7

Documentation

7.1 Introduction

This chapter will feature detailed documentation relating to the project. The projects construction will be shown followed by documentation of the code that runs the actual construction.

7.2 Project Construction

The following figure summarizes the system components that are going to be implemented.

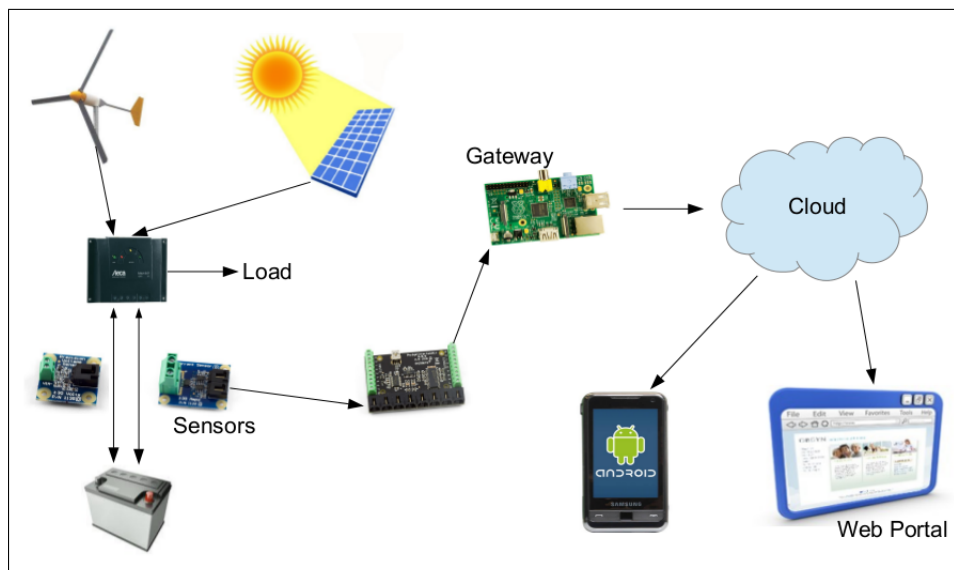


Figure 7.1: Remote Sensor System / Network

Figure 7.1 depicts 5 components of the project. Sensing, communication, storage, processing/data mining and display. Phidget sensors will be responsible for collecting voltage and current supplied by the battery/battery bank. In this setup, a 12v 7.2Ah lead acid battery has been used. The analog sensors then communicate with the phidget interface kit that is responsible for processing the readings from the sensors. The kit passes those readings on to a script running on a raspberry pi that handles the readings by applying the necessary calibration formulas. The pi also serves as a gateway to pass the readings collected to cloud storage for processing, data mining and display on Android devices and for viewing on various web browsers via a web portal.

A more schematic layout or circuit diagram is depicted by figure 7.2.

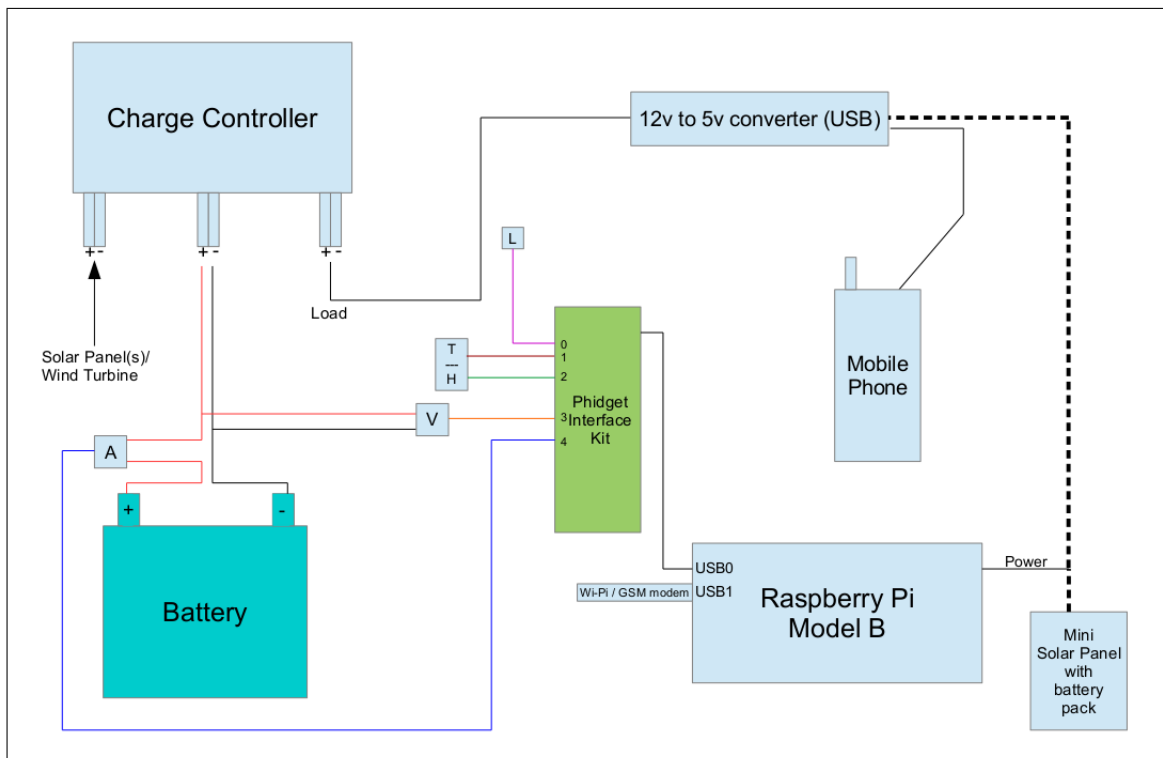


Figure 7.2: Circuit diagram / Schematic / Layout of prototype

The Mini Solar Panel with battery pack could be a Pocket Power. The Pocket Power has the following features and benefits:

- USB output
- Powerful with long operation time
- Portable and convenient
- High efficiency of power consumption
- Electricity-saving function
- User friendly
- Long cycle life
- Rapid recharge
- Environmental friendliness and Economic Efficiency
- Output capacity : 18.5Wh (5000mA/h)
- USB output : 5V, 1.5A
- Charging time : about 8.5 hours
- Input : 5V, 500mA
- Max output current : 1.5A
- Dimensions : 125x79x14mm
- Weight : about 170g

Figure 7.2 shows a Pocket Power mini solar panel with built-in battery pack for powering the Raspberry Pi, Phidget Interface Kit and Phidget Sensors.



Figure 7.3: Pocket Power

7.3 Sensors and their Metrics

Of importance to the development of the remote sensor network, is the accuracy with which the sensors used will be able to capture the data needed. The following figures will detail each sensor used in the project along with their metrics.

7.3.1 Precision Voltage Sensor

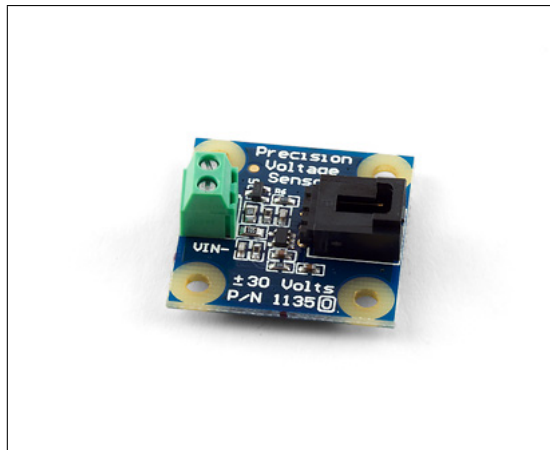


Figure 7.4: Precision Voltage Sensor

The voltage sensor measures DC voltages from -30 to +30 Volts with a typical error of ± 100 mVolts. The Precision Voltage Sensor is not ratiometric.

7.3.2 30 Amp Current Sensor AC/DC

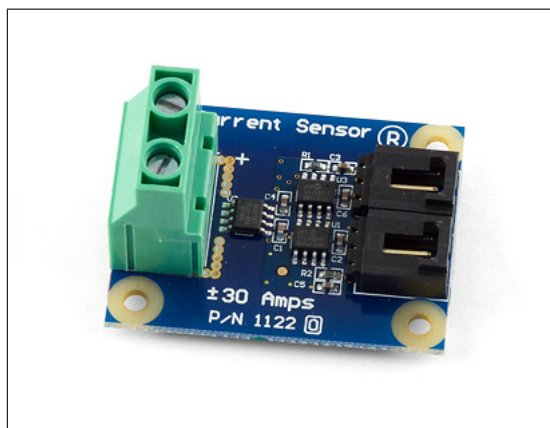


Figure 7.5: 30 Amp Current Sensor AC/DC

The current sensor measures alternating current up to 30 Amps and direct current from -30 Amps to +30 Amps. Dual outputs allow the user to measure both the AC and DC components of complex waveforms separately. The Measurement Error Max for this sensor is 5%.

7.3.3 Humidity/Temperature Sensor

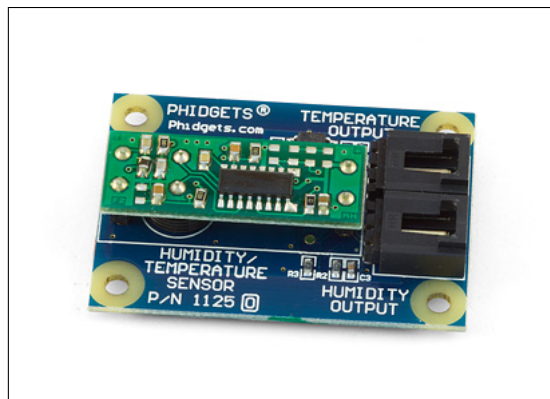


Figure 7.6: Humidity/Temperature Sensor

The humidity sensor measures relative humidity from 10% to 95% with a typical error of $\pm 2\%RH$ at 55% RH. The ambient temperature sensor measures ambient temperature in the range of $-30^{\circ}C$ to $+80^{\circ}C$ with a typical error of $\pm 0.75^{\circ}C$ in the $0^{\circ}C$ to $80^{\circ}C$ range.

The temperature sensor component is rated at $-40^{\circ}C$ to $+100^{\circ}C$, but the other components on the board, the connector and the cable are rated at $-30^{\circ}C$ to $+80^{\circ}C$. In a fast prototyping environment, the temperature sensor board can be pushed to the ratings of the sensor component, but you should use the lower temperature ratings if you plan to use it in a commercial application. It is a ratiometric sensor with a Temperature Typical Error (At $25^{\circ}C$). The ambient temperature Error Max is $\pm 2^{\circ}C$.

7.3.4 Light Sensor 70 000 lux : Luminosity sensor

Figure 7.7 indicates the luminosity sensor that can measure ambient light up to 70 kilolux (roughly equivalent to the brightness of direct sunlight). Each sensor has been individually calibrated and a label has been applied to the back of the board with a calibration value. This value can be used in calculations



Figure 7.7: Light Sensor 70 000 lux: Luminosity sensor

to increase measurement accuracy. The sensor's output is logarithmic, so it will be more accurate at low light levels.

7.3.5 Phidget Interface Kit 8/8/8

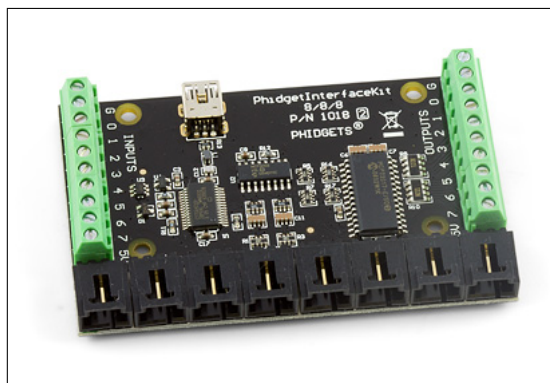


Figure 7.8: Phidget Interface Kit 8/8/8

Figure 7.8 shows the Phidget Interface Kit 8/8/8 which is responsible for processing the readings obtained from analog and digital inputs. Sampling rates can be set at 1ms, 2ms, 4ms, 8ms and multiples of 8ms up to 1000ms.

The kit provides:

- 8 Analog Inputs
- 8 Digital Inputs
- 8 Digital Outputs

The kit operates between a minimum temperature of 0°C and a maximum temperature of 70°C.

The Analog Inputs are used to measure continuous quantities, such as temperature, humidity, position, pressure, etc. Phidgets offer a wide variety of sensors that can be plugged directly into the board using the cable included with the sensor. The Analog Input can measure a voltage between 0V and 5V. The analog measurement is represented in the software as a value between 0 and 1000, so a sensor value of 1 unit represents a voltage of approximately 5mV.

The Digital Inputs can be used to convey the state of devices such as push buttons, limit switches, relays and logic levels. The Digital Outputs can be used to drive LEDs, solid state relays (e.g. 3052 SSR Relay Board), transistors; in fact, anything that will accept a CMOS signal. The kit comes packaged with a 6-foot USB cable, a Getting Started Manual, a mounting hardware kit and a sheet of labels. It has outside dimensions: 3.20" x 2.10" and mounting holes: 2.20" x 1.25".

USB Voltage Min	4.6 V DC
USB Voltage Max	5.5 V DC
Current Consumption Min	13 mA
Current Consumption Max	500 mA
Available External Circuit	487 mA
Recommended Wire Size	16 - 26 AWG
USB Speed	Full Speed

Table 7.1: Kit specifications/attributes

Table 7.1 details the specifications of the kit and are helpful in determining the power source needed to power the kit and the sensors connected to it.

7.4 Voltage Sensor Setup

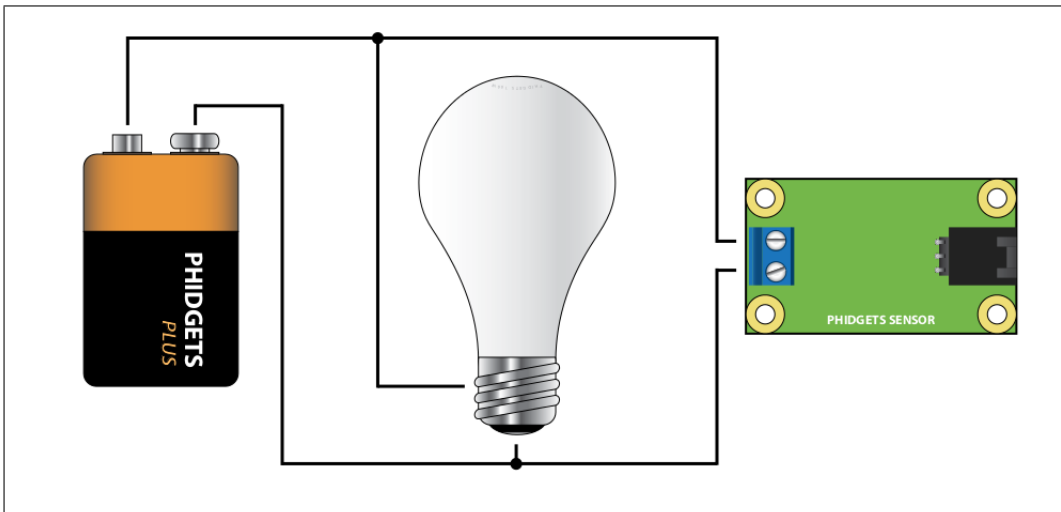


Figure 7.9: Voltage sensor in parallel

The voltage sensor measures the differential voltage across the load.

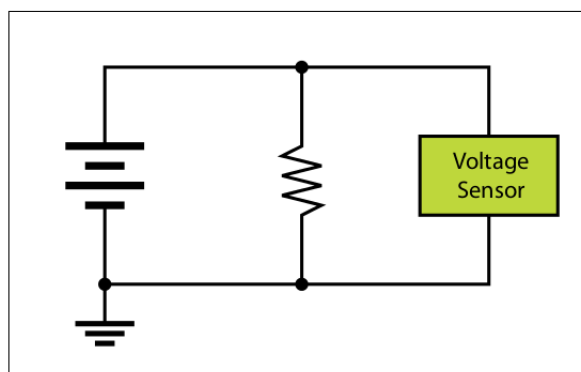


Figure 7.10: Voltage sensor schematics

Figures 7.9 and 7.10 illustrate that the voltage sensor in use has to be connected in parallel on the circuit. The voltage source is represented by a battery symbol and the load is represented by a light bulb or a schematic resistor symbol.

7.5 Current Sensor Setup

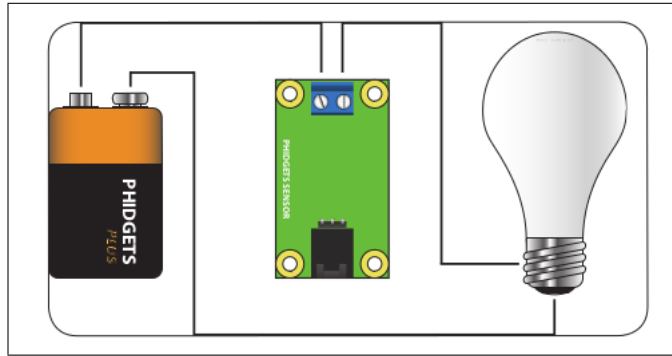


Figure 7.11: Current sensor connected in series

The current flowing through the battery to the load is measured through the current sensor.

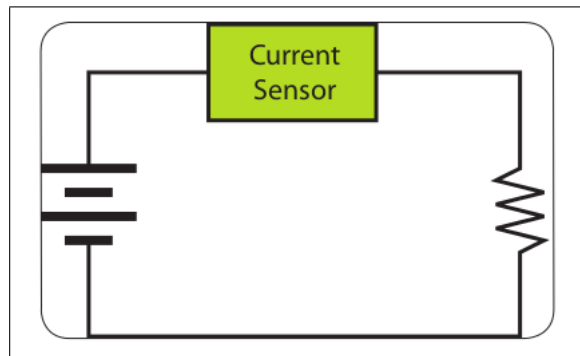


Figure 7.12: Current sensor schematics

Figures 7.11 and 7.12 show that the current sensor should be wired in series with the circuit under test.

7.6 Raspberry Pi

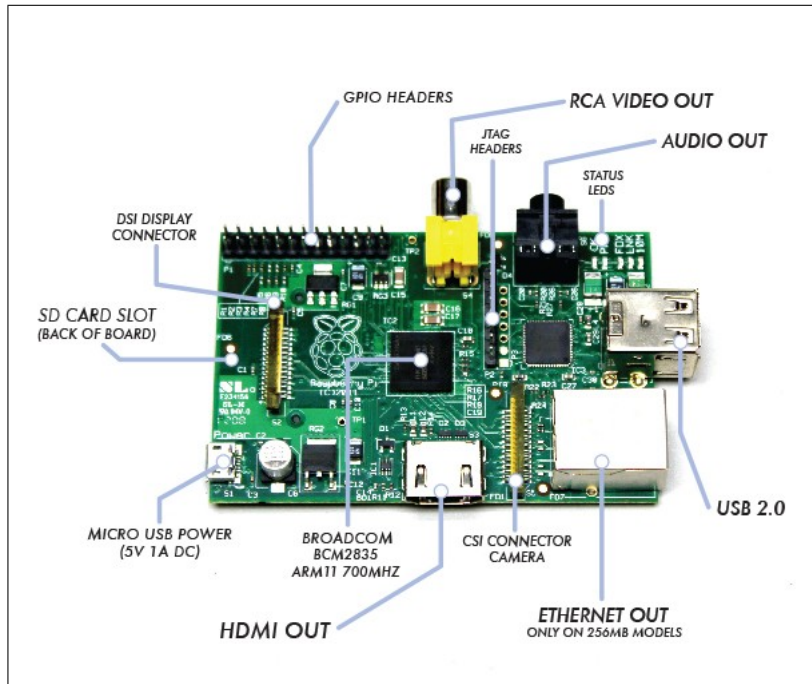


Figure 7.13: Raspberry Pi Model B

Figure 7.13 shows a Raspberry Pi. It is a credit-card sized computer that can plug into a TV and be equipped with a keyboard and mouse. It is a capable little computer which can be used in electronics projects and for desktop projects such as spreadsheets, word-processing and even games. It also plays high definition video. For the purposes of this project, the device will serve as a gateway to process the actual calibrations of sensor readings and to relay those readings on to a MySQL database for storage.

It has the following specifications [7]:

- Memory : 512 MB SDRAM
- Ethernet : onboard 10/100 Ethernet
- USB 2.0 : Dual USB connector
- Video Output : HDMI / Composite RCA
- Audio Output : 3.5mm jack, HDMI
- Operating System : Linux
- Dimensions : 8.6cm x 5.4cm x 1.7cm
- Onboard Storage : SD, MMC, SDIO card slot

The Raspberry Pi in this implementation houses a 16GB SD card and has been loaded with the Raspbian operating system. It is a free operating system based on Debian and optimized for the Raspberry Pi hardware. Raspbian provides more than just a pure OS. It comes with over 35,000 packages, a pre-compiled software bundle in a nice format for easy installation on a Raspberry Pi. The device will make use of a Wi-Fi dongle for relaying data over WiFi. Another option is to fit it with a USB GSM Modem for relaying data over 3G / 2G / SMS. Thus, we can say that the Pi will serve as a gateway.

7.7 Raspbian installation

The steps below detail how to install Raspbian on a SD card via Linux (I used Ubuntu).

Please note that the use of the **dd** tool can overwrite any partition of your machine. If you specify the wrong device in the instructions below you could delete your primary Linux partition. Please be careful.

- Run **df -h** to see what devices are currently mounted.
- If your computer has a slot for SD cards, insert the card. If not, insert the card into a SD card reader, then connect the reader to your computer.
- Run **df -h** again. The new device that has appeared is your SD card. The left column gives the device name of your SD card; it will be listed as something like **/dev/mmcblk0p1** or **/dev/sdd1**. The last part (**p1** or **1** respectively) is the partition number but you want to write to the whole SD card, not just one partition. Therefore you need to remove that part from the name (getting, for example, **/dev/mmcblk0** or **/dev/sdd**) as the device for the whole SD card. Note that the SD card can show up more than once in the output of **df**; it will do this if you have previously written a Raspberry Pi image to this SD card, because the Raspberry Pi SD images have more than one partition.
- Now that you've noted what the device name is, you need to unmount it so that files can't be read or written to the SD card while you are copying over the SD image.

- Run **umount /dev/sdd1**, replacing **sdd1** with whatever your SD card's device name is (including the partition number).
- If your SD card shows up more than once in the output of **df** due to having multiple partitions on the SD card, you should unmount all of these partitions.
- In the terminal, write the image to the card with the command below, making sure you replace the input file **if=** argument with the path to your **.img** file, and the **/dev/sdd** in the output file **of=** argument with the right device name. This is very important, as you will lose all data on the hard drive if you provide the wrong device name. Make sure the device name is the name of the whole SD card as described above, not just a partition of it; for example **sdd**, not **sdds1** or **sddp1**; or **mmcblk0**, not **mmcblk0p1**.

```
dd bs=4M if=2014-06-20-wheezy-raspbian.img of=/dev/sdd
```

- Please note that block size set to **4M** will work most of the time; if not, please try **1M**, although this will take considerably longer.
- Also note that if you are not logged in as root you will need to prefix this with **sudo**.
- The **dd** command does not give any information of its progress and so may appear to have frozen; it could take more than five minutes to finish writing to the card. If your card reader has an LED it may blink during the write process. To see the progress of the copy operation you can run **pskill -USR1 -n -x dd** in another terminal, prefixed with **sudo** if you are not logged in as root. The progress will be displayed in the original window and not the window with the **pskill** command; it may not display immediately, due to buffering.
- Instead of **dd** you can use **dcfldd**; it will give a progress report about how much has been written.
- You can check what's written to the SD card by **dd**-ing from the card back to another image on your hard disk, and then running **diff** (or **md5sum**) on those two images. There should be no difference.

- Run **sync**; this will ensure the write cache is flushed and that it is safe to unmount your SD card.
- Remove the SD card from the card reader [8].

7.8 State of Charge

A sealed lead-acid battery of 12 volt provides different voltages depending on its state of charge. When the battery is fully charged in an open circuit, the output voltage is about 12.8 V. The output voltage lowers quickly to 12.6 V when loads are attached. As the battery is providing constant current during operation, the battery voltage reduces linearly from 12.6 V to 11.6 V depending on the state of charge. A sealed lead-acid battery provides 95% of its energy within this voltage range. If we make the broad assumption that a fully loaded battery has a voltage of 12.6 V when full and 11.6 V when empty, we can estimate that a battery has discharged 70% when it reaches a voltage of 11.9 V. These values are only a rough approximation since they depend on the life and quality of the battery, the temperature, etc [9].

State of Charge	12 V Battery Voltage	Volts per Cell
100%	12.7	2.12
90%	12.5	2.08
80%	12.42	2.07
70%	12.32	2.05
60%	12.2	2.03
50%	12.06	2.01
40%	11.9	1.98
30%	11.75	1.96
20%	11.58	1.93
10%	11.31	1.89
0%	10.5	1.75

Table 7.2: State of Charge of 12V battery

According to Table 7.2 and keeping in consideration that a truck battery should not be discharged more than 20% to 30%, we can determine that the useful capacity of a truck 170Ah battery is 34Ah (20%) to 51Ah (30%). Using the same table, an observation can be made to prevent a battery from discharging below 12.3V.

7.9 Code Documentation

The code will be documented in a fashion that illustrates the sensing and communication aspects of the project. Before we dive into the code, please note that sensor calibration formulas are used to translate sensor values (SensorValue) into readings that fit our understanding. For maximum accuracy, the RawSensorValue property can be used. Each sensor calibration is listed below:

Precision Voltage Sensor

- Voltage (in volts) = $(\frac{(\text{SensorValue}) - 2.5}{0.0681})$
- Substitute (SensorValue) with (RawSensorValue / 4.095) [10]

30 Amp Current Sensor AC / DC

- AC RMS Amps = SensorValue x 0.04204
- DC Amps = $(\frac{\text{SensorValue}}{13.2}) - 37.8787$
- Substitute (SensorValue) with $(\frac{\text{RawSensorValue}}{4.095})$ for maximum accuracy

Humidity / Temperature Sensor

- RH (%) = (SensorValue x 0.1906) - 40.2
- Temperature (C) = (SensorValue x 0.22222) - 61.11

Light Sensor 70 000 lux : Luminosity sensor

- Luminosity (lux) = $e^{m * \text{SensorValue} + b}$, where m = 0.02394 and b = -1.1843
- 'm' and 'b' are calibration values found on the label on the underside of the 1143. If for some reason you cannot use the calibration values that come with the sensor, you can use the generalized values of **m = 0.02385** and **b = -0.56905** to get a rough approximation.

7.9.1 Gathering data from sensors

As mentioned before, every sensor will have its own calibration formula for computing the sensor's value into human readable form. Please note that each sensor value is retrieved from a different port on the phidget interface kit. The luminosity sensor is plugged into port 0, the ambient temperature sensor into port 1, the relative humidity sensor into port 2, the precision voltage sensor into port 3 and the current sensor into port 4. The code below details the retrieval of sensor values and the calibrations applied.

```
# get Luminosity – Wide Range Light Sensor P/N: 1143_0
# Luminosity (lux) = e**(m*Sensor Value + b)
lum = float(round(e**(m * (interfaceKit.getSensorValue(0)) + b), 2))

# get Ambient Temperature in degrees Celsius – P/N: 1125
# Temperature (C) = (Sensor Value x 0.22222) – 61.11
temp = float(round(((interfaceKit.getSensorValue(1) * 0.22222) – 61.11, 2))

# get Relative Humidity – P/N: 1125
# RH (%) = (Sensor Value x 0.1906) – 40.2
hum = float(round(((interfaceKit.getSensorValue(2) * 0.1906) – 40.2, 2))

# get the precise voltage – Precision Voltage Sensor P/N: 1135
# Differential voltage = (((Sensor Value/200) – 2.5) / 0.0681)
# where V diff is defined as V positive – V negative

#voltage = float(round((((interfaceKit.getSensorValue(3) / 200) – 2.5)
    / 0.0681), 3))
v = float(interfaceKit.getSensorRawValue(3)) / 4.095 # maximum accuracy
voltage = round(((v / 200) – 2.5) / 0.0681), 3)

# get the current – 30 Amp Current Sensor P/N: 1122_0
# DC Current (A) = (Sensor Value / 13.2) – 37.8787
c = interfaceKit.getSensorValue(4)
current = round(((c / 13.2) – 37.8787), 3)
```

```
# Calculate the Power
# Power = Voltage x Current
power = volt * cur
```

7.9.2 Sensing with the Raspberry Pi

The raspberry pi runs the sensing python script via a crontab/cronjob. Cron is a system daemon that is used to execute tasks in the background at designated times. A crontab is a simple text file with a list of commands meant to be run at specified times. It can be edited with a command-line utility. These commands and their run times are controlled by the cron daemon, which executes them in the system's background. Cronjobs run regardless of whether the user who created them are logged into the system [11].

The sensing python script pi_wifi_1min.py is run with the following cronjob:

```
# m h dom mon dow command
* * * * * sudo python /home/pi/pi_wifi_1min.py
```

Cron runs the script every minute of the hour for every hour of the day for every day of the month for every month of the year for every day of the week.

7.9.3 Android function to retrieve readings from MySQL database

The following Android function is responsible for displaying the sensor readings which are sent to the MySQL database. The same readings are displayed on the web portal.

```
public void getData(){
    String result = "";
    InputStream isr = null;
    try{
        HttpClient httpClient = new DefaultHttpClient();
        HttpPost httpPost = new HttpPost("http://www.cs.uwc.ac.za/~zerasmus/
        getLatestReadings.php");
        HttpResponse response = httpClient.execute(httpPost);
        HttpEntity entity = response.getEntity();
```



```

        isr = entity.getContent();
    }
    catch(Exception e){
        Log.e("log_tag", "Error in http connection " + e.toString());
        resultView.setText("Couldn't connect to database");
    }
    // convert response to string
    try{
        BufferedReader reader = new BufferedReader(new InputStreamReader(
            isr, "iso-8859-1"), 8);
        //BufferedReader reader = new BufferedReader(new InputStreamReader(
            isr, "UTF-8"));
        StringBuilder sb = new StringBuilder();
        String line = null;
        while((line = reader.readLine()) != null){
            sb.append(line + "\n");
        }
        isr.close();

        result = sb.toString();
    }
    catch(Exception e){
        Log.e("log_tag", "Error converting result " + e.toString());
    }

    // parse json data
    try{
        String s = "";
        JSONArray jArray = new JSONArray(result);

        for(int i = 0; i < jArray.length(); i++){
            JSONObject json = jArray.getJSONObject(i);
            s = s +
                "Date / Time : " + json.getString("date_time") + "\n" +
                "Ambient Temp : " + json.getString("ambient_temp") + " \u2103\n"
                "Relative Humidity : " + json.getString("rel_hum") + " %\n" +

```

```

        "Luminosity : " + json.getString("lum") + " Lux\n" +
        "DC Voltage : " + json.getString("dc_volt") + " Volt(s)\n" +
        "DC Current : " + json.getString("dc_cur") + " Amp(s)\n" +
        "Power : " + json.getString("power") + " Watt(s)";
    }

    //System.out.println(s);
    resultView.setText(s);
}
catch(Exception e){
    // TODO: handle exception
    Log.e("log_tag", "Error Parsing Data " + e.toString());
}
}

```

7.9.4 Android Application Execution

Upon launching the android application, it opens up and initiates a connection to the database. This initiation is handled at certain time frames. Essentially, timing is handled via `android.os.Handler`;. The following handlers have been utilized:

- Sensor readings handler

```
TimeUnit.SECONDS.sleep(1)
```

- State of Charge handler

```
TimeUnit.SECONDS.sleep(10)
```

The sensor readings are refreshed every second and the state of charge (SOC) is refreshed every ten seconds, since the SOC value does not change frequently.

HttpPost is responsible for making a POST Request to the server for the readings. The response from the server is converted to a string which contains the data. The data is parsed to extract the sought after readings. The application then displays these readings on its view.

7.10 Conclusion

This chapter examined the project's documentation, namely the project's construction, sensors and their metrics, voltage and current sensor setups, the credit card sized computer responsible for running the calibrations, Raspbian installation, state of charge and the code documentation. The most interesting details revolve around sensing and communication and the circuit schematics and setup. The next chapter will focus on testing.

Chapter 8

Testing and Results

8.1 Introduction

This chapter will feature detailed testing documentation relating to the project. Testing results will be illustrated and discussed.

8.2 Testing procedure

Throughout the project's development, an incremental approach was used. Thus before adding a component to the project, testing was implemented to ensure that component was working and merged with the rest of the project successfully. Figure 8.1 illustrates the approach mentioned.

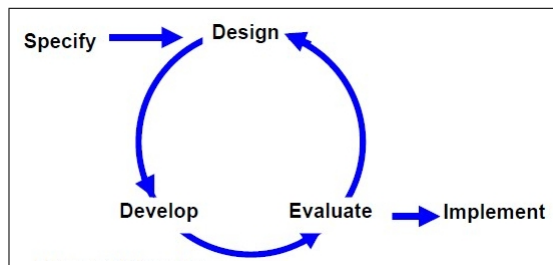


Figure 8.1: Incremental approach

8.3 White-Box testing - Static Testing

Throughout development increments, checks were performed for code and algorithm sanity. The code of the sensing script, web portal, android application and SMS notification system was manually reviewed to find errors. In isolation, these walkthroughs, inspections and reviews revealed bugs that were easy to fix. An example is a calibration error on the sensing script that produced a slightly higher voltage reading.

8.4 Stress Testing

Stress testing involved seeing whether the database could handle queries sent to it every second as per the client's specifications. This involved readings sent to the database and pulled from the database every second. Both scenarios performed successfully. The android application was also tested in this manner and did not hang or produce any performance lags. It relayed per second updates of readings to its view as was desired.

8.5 Performance Testing

The Raspberry Pi yielded a successful sensing and relaying operation from boot-up. The Raspberry Pi could boot-up, connect to a wifi network and start broadcasting readings under 1 min. Network performance was ultimately a deciding factor towards performance testing as a poorly performing wifi network would lead the Raspberry Pi to miss interval broadcasts as it had to reconnect to other available networks.

8.6 Testing results

8.6.1 Environment - Lab vs outside

Parameters	Lab	Outside
Ambient Temperature (°C)	23.33	15.33
Relative Humidity (%)	35.28	53.77
Luminosity (lux)	232.08	1.11
DC Volt(s)	12.685	14.478
DC Current (I)	-0.227	0.076
Power (W)	-2.879495	1.100328

Table 8.1: Testing - Environment

The readings of Table 8.1 were taken on 16 October 2014. Lab readings were taken at 21:05:44 and outside readings at 20:46:38. During Lab testing the Raspberry Pi was powered via AC power, thus the negative current reading is representative of no load on the system. Outside the lab reveals a current reading of 0.076 where the Raspberry Pi drew power from the system. The voltage differences are a key component in determining the effectiveness of the voltage sensor. At this stage, it is unknown why the battery's voltage

reading differs immensely as the Raspberry Pi is not directly connected to the precision voltage sensor. This could indicate a voltage leakage.

Figure 8.2 graphically illustrates a voltage difference taken on 22 October 2014. From the figure it can be seen that leaching presents a higher voltage reading than AC power does.

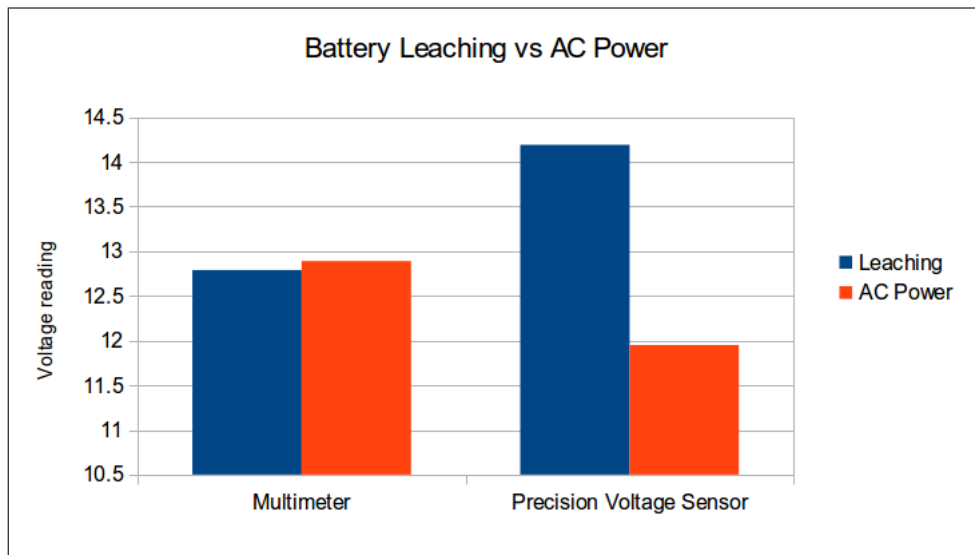


Figure 8.2: Battery leaching vs AC Power

8.6.2 Environment - Phidget Sensor vs Lab Thermometer vs Air-conditioner Remote

Parameters	Phidget Sensor	Lab Thermometer	Aircon. Remote
Ambient Temperature (°C)	21.78	21.70	22.00

Table 8.2: Testing - Lab Temperature

The temperature readings of Table 8.2 were measured against twin air-conditioner temperatures set to 22 degree Celsius. The table highlights how the lab thermometer and phidget ambient temperature sensor were able to measure the lab's temperature. The air-conditioners were allowed to blow for 15 min. before readings were taken.

8.6.3 Phidget Sensors vs Accuweather

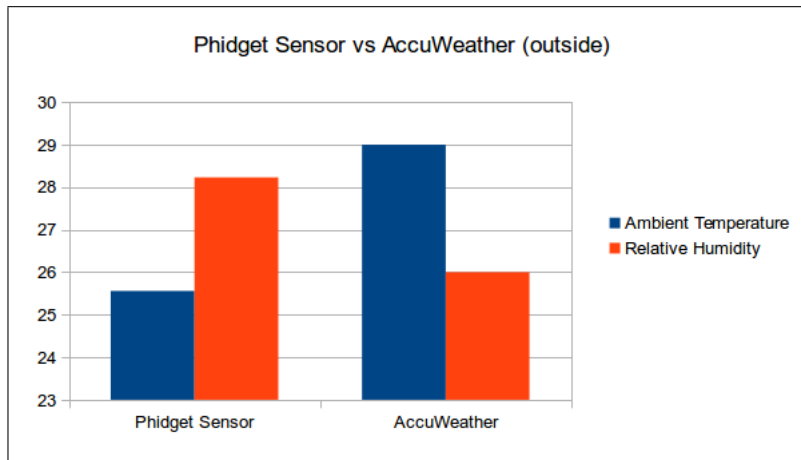


Figure 8.3: Ambient Temperature (°C) and Relative Humidity (%)

The readings of Figure 8.3 were taken on 19 October 2014 at 15:21:13. The readings from AccuWeather were computed for 15:00 in the Bellville, Cape Town area.

8.6.4 Panel voltage - Lab vs outside

Parameter	Lab	Outside
Panel Voltage	12.71 - 14.67 V	20.6 - 21.0 V

Table 8.3: Testing - Panel Voltage

The readings of Table 8.3 were taken on 21 August 2014 at 14:00. The main emphasis of the above table is to illustrate the power generation capabilities of the panel in use. It is also able to generate a voltage indoors by scavenging from secondary power sources (lab lighting).

8.6.5 Phidget Sensors vs Multimeter

Parameters	Phidget Sensor	Multimeter
DC Volt(s)	11.036	12.07
DC Current	0.152	0.440
Power	1.677472	5.3108

Table 8.4: Testing - Phidget Sensors vs Multimeter

Table 8.4 presents voltage and current readings of the phidget sensors versus a multimeter. As per the results, it can be seen that the phidget

sensors sense low values for voltage as well as current. The phidget voltage sensor correlates with the charge controller, indicating that the battery is empty with a low voltage disconnection prewarning and that loads are still on. The load is a cellphone which is being charged by the system.

8.6.6 Charge Controller Terminals

Parameters	Date	Time	Panel	Battery	Load
DC Volt(s) at 226.59 lux	2 Oct. '14	14:15	12.99	12.99	12.99
DC Volt(s) at 232.08 lux	16 Oct. '14	21:05:44	12.68	12.68	12.68
DC Volt(s) at 1.04 lux	16 Oct. '14	21:09:44	6.89	12.76	12.76

Table 8.5: Testing - Charge Controller Terminals (LAB)

Parameters	Date	Time	Panel	Battery	Load
DC Volt(s) at 1.09 lux	16 Oct. '14	20:46:38	6.88	12.06	12.06
DC Volt(s) at 4853.52 lux	19 Oct. '14	15:19:13	12.07	12.06	12.07

Table 8.6: Testing - Charge Controller Terminals (Outside)

Tables 8.5 and 8.6 are included to illustrate voltages across the three terminals of the charge controller, namely, the panel terminal (left), battery terminal (center) and the load terminal (right). The maximum terminal voltage determined during lab testing revealed a voltage reading of 12.99 volts across all three terminals. The lowest panel voltage terminal reading was obtained at 1.04 lux when the lab lights had been turned off at 21:09:44. At a similar lux reading outside, an almost equal panel terminal voltage was detected at 6.88 volts at 20:46:38.

8.6.7 SMS Notification

Voltage Range	SOC (Approx.)
$11.58 \leq \text{voltage} < 11.75$	20%
$11.31 \leq \text{voltage} < 11.58$	10%
$10.50 \leq \text{voltage} < 11.31$	0%

Table 8.7: Testing - Voltage ranges below accepted thresholds

Field test - A voltage of 11.018 V resulted in the SMS notification as illustrated by Figure 8.4 below.

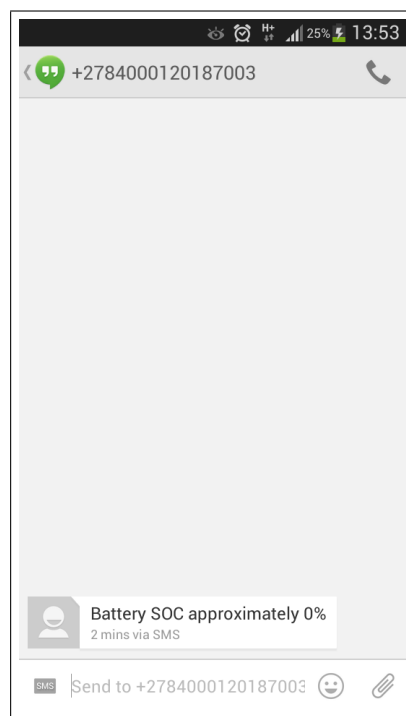


Figure 8.4: SMS Notification sent out for an approximal 0% SOC reading

A cronjob upon the server housing the web portal code would initiate a SOC check at a preset interval. When a low SOC is detected based on the voltage ranges of Table 8.7, a SMS will be sent out immediately. Testing reveals the delivery of a SMS within a three second window from SOC detection to notification. This result meets the user requirements as it serves as an effective means of battery SOC notification on a remote basis.

8.6.8 Luminosity results

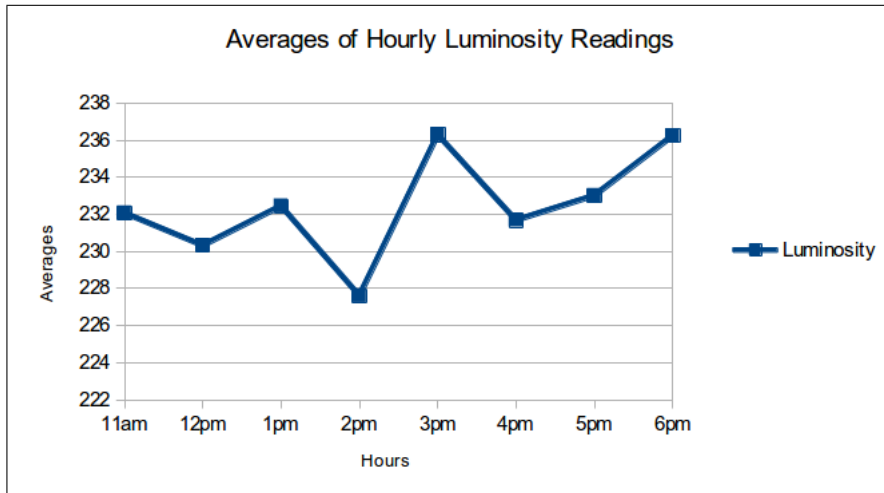


Figure 8.5: Average lab luminosity readings on 2 Oct 2014

Figure 8.5 illustrates the average hourly lab luminosity readings from 11am to 6pm on 2 October 2014. From the figure it can be seen that from the highest (3pm) to the lowest (2pm) reading, there is a difference of 8.68 lux during the time frame of 8 hours. Being able to track the light levels in the lab throughout a 24 hour day can give an indication of their energy patterns.

8.6.9 Averages of last 2 days

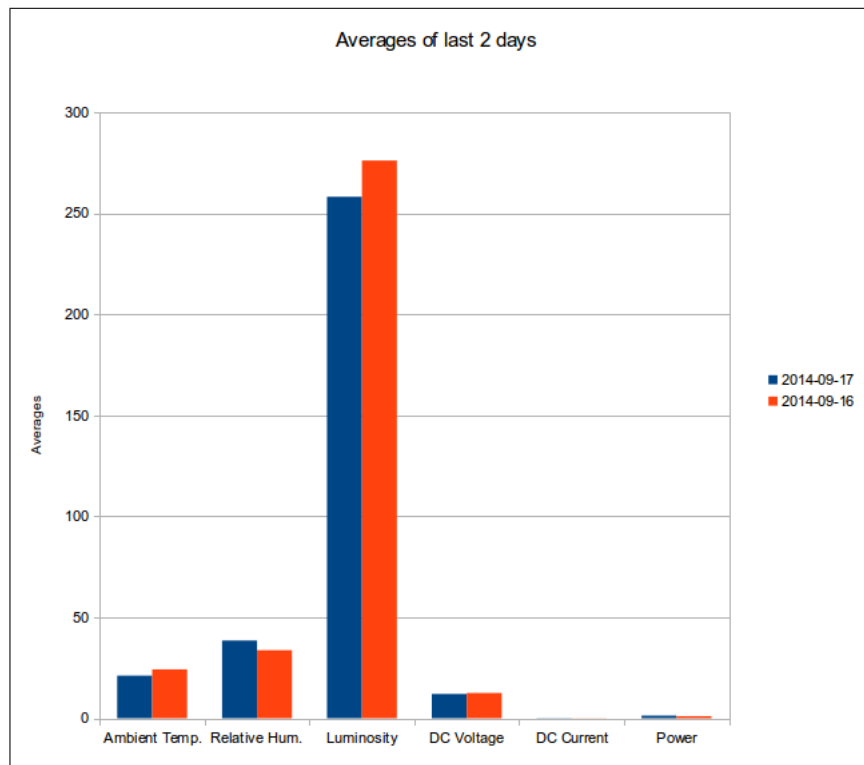


Figure 8.6: Averages of last 2 days

Figure 8.6 illustrates the averages of the last 2 days recorded by the system. These averages are compared against one another to gain an understanding on how a day relates to the one before it.

8.7 Conclusion

This chapter examined the project's testing documentation and the results gathered. The results include differences and similarities between a lab environment as well as a location outside the lab. The next chapter will focus on the user guide.

Chapter 9

User Guide

9.1 Introduction

This chapter will feature a detailed user guide relating to the project. It will mainly focus on setting up the sensors, Phidget Interface Kit and Raspberry Pi and also give brief instructions for using the web portal and Android application.

9.2 Sensors and Phidget Interface Kit

The following sensors have to be connected to the Phidget Interface Kit in the specified ports:

- Luminosity / Light Sensor 70 000 lux **0**
- Temperature **1**
- Humidity **2**
- Precision Voltage Sensor **3**
- 30 Amp Current Current Sensor AC/DC **4**

9.3 Raspberry Pi

House cleaning:

- `sudo apt-get update`
- `sudo apt-get upgrade`

Software tools / packages to install before setting up the sensor system:

- `sudo apt-get install vim`
- `sudo apt-get install vim-gtk`
- `sudo apt-get install chromium`

The following ports are used/connected to by these devices:

- **USB0** Phidget Interface Kit
- **USB1** Wi-Pi / GSM Modem

See pages 37-39 for the Raspbian installation.

Also make sure the following libraries are installed on a Raspberry Pi before loading and running the sensing script.

- `libusb-1.0-0-dev`
- Phidget libraries - `libphidget.tar.gz`
- Phidget python library

Place `pi-wifi_1min.py` in the directory `/home/pi/`. Set a cronjob and direct it with the following parameters:

```
# m h dom mon dow command
* * * * * sudo python /home/pi/pi_wifi_1min.py
```

9.4 Web Portal

The web portal will serve as the main means of observing the remote sensor system. Upon login, a user will be directed to the home page. This page will illustrate a table of sensor readings and the date and time they were logged with the server. Below this table is a SOC calculation that is computed from the battery's voltage reading. Below the SOC div is a live graph that models the present readings that are being sent from the Raspberry Pi to the database.

The links on the left side of the page are detailed below:

- Logout - Exiting / Signing off from the web portal
- Averages - Displays averages of the last 3 days textually and graphically
- Luminosity - Daily Luminosity / Averages of Hourly luminosity readings
- Readings - Averages of Hourly Readings

All readings displayed are from 5am in the morning to 20pm in the evening.

9.5 Android Application

Upon clicking on the Android application, it opens up and connects to the database if an internet connection is present. If no connection is present, an error message will be displayed. Assuming internet is available, the readings will be requested from the server and displayed by the application. Please note that the rate that the sensor readings are refreshed are different from the SOC. The sensor readings are refreshed every second, where as the SOC is refreshed every ten seconds. Upon wanting to exit the application, a user will just need to press the Android back button.

9.6 Conclusion

This chapter examined the project's user guide. It illustrated how to use the sensors, phidget interface kit and raspberry pi. The next chapter will conclude the project and highlight possible future work.

Chapter 10

Conclusion

The entire system can be improved or edited and compared to the current prototype by using different sensing equipment and gateway devices. Of great interest is the newly development Phidget Single Board Computer (SBC) that would replace the Phidget Interface Kit and Raspberry Pi as it would serve the purposes of both devices. The Android application can be expanded to cater for graphing of vital system readings. A different graphing library can be implemented to make the web portal appear more feature rich and appealing on the eye.

Packaging of the sensors, interface kit and Raspberry Pi would be important before deploying the hardware side of the system for outside usage. This would also prevent rust, corrosion and damage from environmental factors. Situation recognition using machine learning techniques or statistical analysis methods will need to be run on a large dataset gathered from running multiple sensor systems at various locations. This would lead to prediction capabilities.

The project successively partook in the development of a remote sensor network (RSNET) which can continuously acquire energy yields and performance measures of renewable energy solar power systems. Based on the motivation that sensor technology is still a relatively new field, the project was able to research and construct a means for engineers to avoid traveling to far off locations to gather data, but rather receive the readings by means of a remote monitoring system. Engineers are able to receive the readings by means of a low cost solution and the sensing devices could be deployed at multiple locations.

Bibliography

- [1] Edgwall Software, “libusb,” 2014. <http://www.libusb.org/>.
- [2] Anonymous, “The phidgets manual,” 2014. <http://rs.cs.iastate.edu/smarthome/documents/ManualsandTutorials/Phidgets/PhidgetsManual.pdf>.
- [3] M. Nkoloma, M. Zennaro, and A. Bagula, “SM² : Solar monitoring system in malawi,” 2011 ITU-T Kaleidoscope Academic Conference, 978-92-61-13651-2/CFP1138-E-CDR, 2011.
- [4] N. Schelling, M. J. Hasson, S. L. Huong, A. Nevarez, P. W.-C. Lu, M. Tierney, L. Subramanian, and H. Schützeichel, “Simbalink: Towards a sustainable and feasible solar rural electrification system,” ICTD '10 Proceedings of the 4th ACM/IEEE International Conference on Information and Communication Technologies and Development Article No. 42, 2010, ISBN: 978-1-4503-0787-1 doi>10.1145/2369220.2369260.
- [5] A. Hande, T. Polk, W. Walker, and D. Bhatia, “Indoor solar energy harvesting for sensor network router nodes,” (Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas, P.O. Box 830688, EC33, Richardson, TX 75083, USA), Elsevier B.V., Microprocess. Microsys. (2007), doi:10.1016/j.micpro.2007.02.006.
- [6] A. Stelmack, “Python:module phidgets.devices.interfacekit,” May 17, 2010. <http://www.phidgets.com/documentation/web/PythonDoc/Phidgets.Devices.InterfaceKit.html>.
- [7] Anonymous, “Comparison between model a and model b,” 2014. <http://downloads.element14.com/raspberryPi1.html>.
- [8] Anonymous, “Rpi easy sd card setup,” 2014. http://elinux.org/RPi_Easy_SD_Card_Setup.
- [9] R. Flickenger, C. Aichele, S. Büttrich, and L. M. Drewett, “Wireless

networking in the developing world second edition,” Hacker Friendly LLC, December 2007.

- [10] Anonymous, “1135 user guide,” 2014. http://www.phidgets.com/docs/1135_User_Guide.
- [11] d-admin a, “Cronhowto,” 2014. <https://help.ubuntu.com/community/CronHowto>.