Solar Panel Anomaly Detection and Classification

by

Bo Hu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2012

Abstract

The number of solar panels deployed worldwide has rapidly increased. Solar panels are often placed in areas not easily accessible. It is also difficult for panel owners to be aware of their operating condition. Many environmental factors have negative effects on the efficiency of solar panels. To reduce the power lost caused by environmental factors, it is necessary to detect and classify the anomalous events occurring on the surface of solar panels.

This thesis designs and studies a device to continuously measure the voltage output of solar panels and to transmit the time series data back to a personal computer using wireless communication. A program was developed to store and model this time series data. It also detected the existence of anomalies and classified the anomalies by modeling the data. In total, ten types of anomalies were considered. These anomaly types include temporary shading, permanent shading, fallen leaves, accumulating snow and melting snow among others. Previous time series anomaly detection algorithms do not perform well for real-life situations and are only capable of dealing with at most four different types of anomalies. In this work, a general mathematical model is proposed to give better performance in real-life test cases and to cover more than four types of anomalies. We note that the models can be generalized to detect and to classify anomalies for general time series data which is not necessarily generated from solar panels. We compared several techniques to detect and to classify anomalies including the auto-regressive integrated moving average model (ARIMA), neural networks, support vector machines and k-nearest-neighbors classification. We found that anomaly classification using the k-nearest-neighbors classification was able to accurately detect and classify 97% of the anomalies in our test set. The devices and algorithms have been tested with two small 12-volt solar panels.

Acknowledgements

I would like to thank Professor S. Keshav for supervising this research and Nicole Keshav for helping me to correct grammatical errors.
I would also like to thank Alan Kaplan, who designed the sensor board and helped me debug it.
I would like to thank Professor Johnny Wong and Professor Bernard Wong to be readers of this thesis.

Dedication

This is dedicated to my parents for supporting me and giving me courage all the time.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The goal of this thesis project is to assist solar panel owners to be aware of the status of their panels. The project applies computer science and statistics techniques to design a system that provides feedback to solar panel owners when anomalous events take place. After anomalies take place on the surface of solar panels, if panel owners are informed of the types of the anomalies in time, then they can remove the anomalies to avoid further energy loss. Previous solar anomaly detection research has depended on designing specific hardware devices, such as micro-inverters, that are integrated with solar panels. Their generality is limited because they can only be used with solar panels supporting that specific hardware. The system can be used for most existing solar panels as long as their voltage outputs are measurable; therefore, it results in a more general approach. It is also cost efficient, because it does not modify the solar panel and all computation is done using low-cost microprocessors and personal computers (PC).

1.1 Solar energy

Solar energy technologies have long been used in the areas of solar heating, solar photovoltaics, solar thermal electricity, and solar architecture. Energy shortages and increasing energy prices are two of the most urgent problems we face today. One desirable solution to the energy shortage problem is renewable energy, and solar energy is one of the cleanest and most efficient energy sources. Solar panels are among the most common methods of harvesting solar energy from solar radiation, which accounts for a large portion of available renewable energy.

At the end of 2011, the photovoltaics (PV) capacity world-wide was 67.4 GW, with a yearly growth of 69.8% in 2011[32][4]. Solar panels are widely implemented in solar lighting, solar thermal and solar power generation areas. Currently, most solar panel products available cannot detect or classify solar anomaly events.

1.2 Environmental influence on solar energy systems

The basic unit of a solar energy system is a solar cell which converts solar radiance energy to electric energy using photovoltaic effect. Connecting to each other via hidden wires, solar cells are assembled together to form a solar panel. Solar panels in a solar energy system are connected via connectors, among which the MC4 connector is most common.

The efficiency of solar panels are affected by many environmental factors like clouds, dust, shadow, and temperature. The solar panel performance can also be influenced by artificially environmental factors like evaporation of pollution from different factories. Edge shadow, which is a special type of shadow intensively occurs in the edge of tilted panels, causes an energy loss of up to $10 - 20\%$[24]. Dust on the panels can lead to as much as 40% degradation in the peak power output of panels[5].

1.3 Challenges

Solar panels can be classified according to their mounting systems. Household solar panels are mounted mostly on roofs and it is difficult to obtain access to roof-based solar panels. Commercial solar generation systems include thousands of solar panels, making it difficult to identify which solar panel is not working well among thousands of panels. Solar panel owners want to make sure that solar panels are working well without continuous manual checking. Instead of the sensing approach studied in this thesis, an alternative simple approach would be to install cameras to monitor solar panels. However, monitoring these cameras still requires manual effort. Therefore, the design goal of our work is to assist solar panel owners to be aware of whether the solar panels are functioning properly. It acts as a virtual assistant to provide feedback to them.

Chapter 2

Measuring System

An overview of our system is shown in Figure 2.1. It obtains continuous voltage measurements from solar panels, and then transmits the data to the personal computers in a format of time series. The anomaly detection and classification algorithm running on the personal computer is trained by the time series data. Afterwards, the personal computer provides panel owners with feedback about the time and types of detected anomalies. For most solar panels, changes in external environmental factors are immediately reflected in the solar panel voltage output. This chapter describes the measuring system we built to obtain and transmit sensed data from a solar energy system.

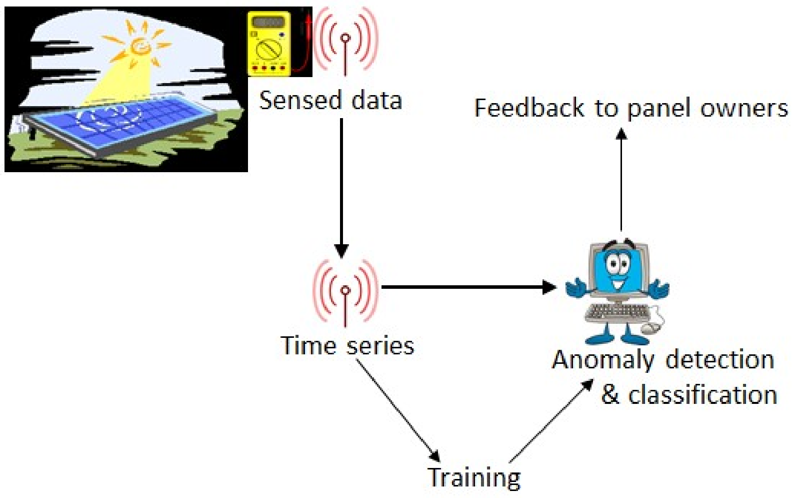

Figure 2.1: Model overview

2.1 Sensor placement

We discuss how we place our sensors in an existing solar energy system. Solar energy generation systems are formed by solar panel arrays, in which each solar panel has one male connector and one female connector. The male connector of one solar panel is connected to the female connector of the next solar panel. We have designed our sensor to easily integrate into this system. Eventually, we replace junctions with one-to-two adaptors so that the sensor can be placed in parallel to the panel. Each sensor has only male connectors and we need to introduce two M-to-2F adaptors per panel. Figure 2.2 shows the schematic view of the system connections.

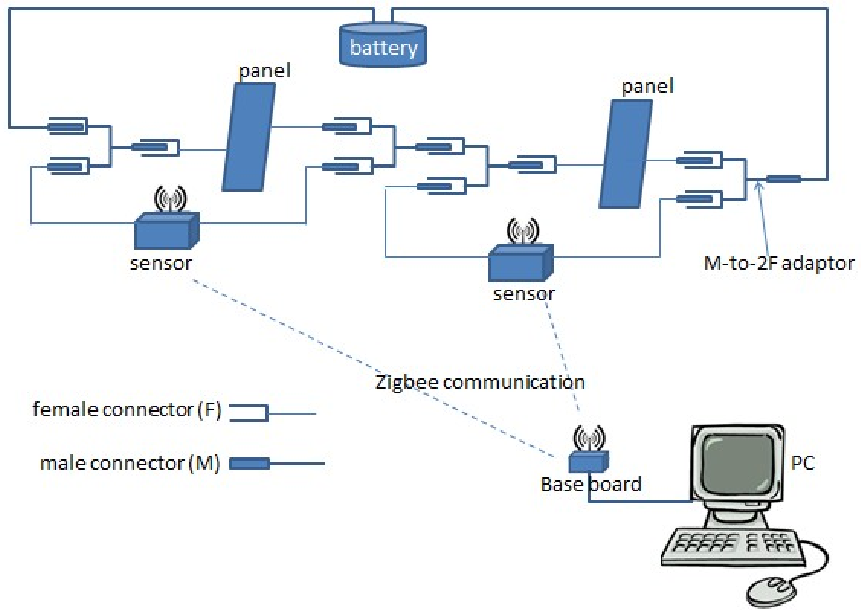

Figure 2.2: Panel connection layout

2.2 Sensor board

2.2.1 Sensor board design overview

The sensor board and the microprocessor program were designed by Alan Kaplan, an external consultant of this project. My contribution is debugging the periodic sleep mode of the board and fixing two hardware bugs. The major parts of the sensor board include a 8-bit flip switch, an ampere measuring amplifier, a voltage measuring amplifier, a radio-frequency module, an MSP430 microprocessor, and other auxiliary circuitry. The microprocessor controls the entire sensor board; it reads measured values from the amplifiers and returns the values to the base board through the radio-frequency module. For a solar array which is composed of n solar panels, n sensor boards are needed and at least one of them should be able to measure current. In order to reduce the cost of manufacture, the ampere measuring amplifiers are removed on boards measuring only voltages.

Figure 2.3 shows a real picture of the designed board.

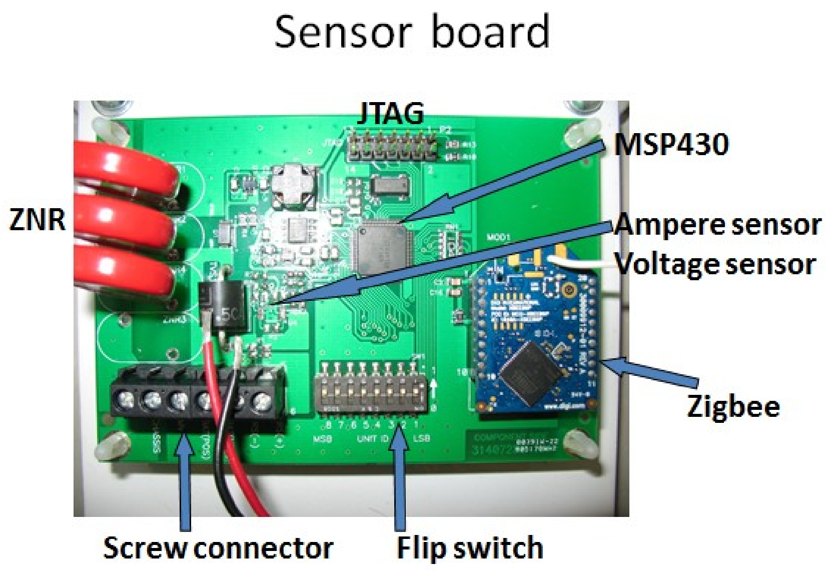

Figure 2.3: Detailed board

2.2.2 Connection between sensor boards and solar panels

Figure 2.4 shows the detailed connections between sensor boards and the solar panels:

- BAT+ and BAT- are used for powering the board by connecting to an external 6-volt battery
- MV is connected to the positive side of the solar panel
- For the board which measures both current and voltages, RS+ and RS- are connected to the main line of solar panel in series. RS+ is connected to the return side (low voltage) of the solar cell load, which is also the low side of the inverter that is connected across the string of solar cells. For the board which measures voltages only, the RS+ pin and RS- pin are not used

- BAT- and RS- on all boards both connects via a separate wire to the negative side of the battery. The negative side of the solar panel, together with BAT- and RS-, is connected to the battery negative, which also acts as the analog ground and the digital ground of the board.

The working power current in a solar panel can be high, varying from zero to 500mA, so a one-million-ohm power resistor is needed to reduce the ampere value at the sensor board. With a clock frequency of about 902 − 928 MHz, the board can afford relatively complex computation.

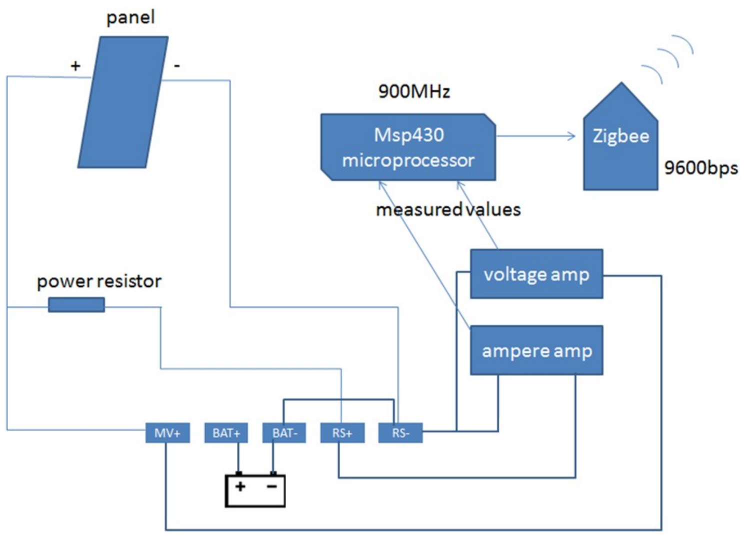

Figure 2.4: Circuitry layout

2.2.3 Auxiliary circuitry

Protection

The board is intended to resist power surges caused by lightning strikes because solar panels are deployed on the roofs of high buildings. The Zinc Oxide Nonlinear (ZNR) module resists lightning strikes. Moreover, a thick plastic box holds the entire circuit board for protecting the circuits from rain, wind and other types of damage.

Board identification (I.D.) switch

The 8-bit flip switch identifies each board. The measuring system can support up to 256 sensors that work concurrently. Each board has a serial number which is read from the eight position switch and the serial numbers are used for determining which board is configured to measure the current.

2.2.4 Sensing circuits

The voltage amplifier is chosen to be the AD8615 instrumentation amplifier on the board. It converts analog voltage values into digital values that can be used by the microprocessor. The voltage computation process is identical for the two types of boards (boards measuring voltage only and boards measuring voltage plus current), because they have identical voltage measuring circuitry and software, and they only differ at the ampere instrumentation amplifier and its associated components.

The connections and functionalities of the pins on AD8615 are listed as follows:

- Pin1 is the output pin of the amp, which should be 1/31 of the measured voltage
- Pin2 is analog ground and the negative supply for AD8615; it connects to BAT- via a resistor of around $0.2 \sim 0.3$ ohms
- Pin3 is a positive supply which is approximately 3.3 volts
- Pin5 is an power supply pin (VCC), which is 3.3 volts
- Pin6 is a reference voltage which should be 2.50 volts with respect to analog ground
- Pin7, labeled as "Vref", is the offset voltage

The resistor divider applied to the measured voltage consists of two resistors of $10k$ ohms and $300k$ ohms. The dividing ratio is computed as $10k/(10k+300k) = 1/31$. For example, if the measured value is 10 volts, then Pin1 on AD8615 is 10/31 volts or approximately 0.32 volts. The voltages of the resistor divider should be 10 volts at one end and 0.32 volts at the other end with respect to the analog ground. R7 and R8 forms another pair of resistor divider with R7 being $100k$ ohms and R8 being $33k$ ohms. The offset voltage for both voltage instrumentation and ampere instrumentation should be quite close to 0.8175 volts nominally. The value of 0.8175 comes from the pair of resistor divider and Vref, is computed as $3.3 * 33k/(100k + 33k) = 0.8187 \approx 0.8175$ volts.

Similar to the voltage amplifier, the ampere amplifier is designed to convert the analog ampere values into digital values used by the microprocessor program. The offset voltage pin of the instrumentation amp should also be guaranteed to be close to 0.8175 volts nominally; otherwise, the R7/R8 voltage divider has an incorrect resistor and the ampere measurement will be off.

2.2.5 Microprocessor

The MSP430 microprocessor is designed and programmed to receive synchronization signals from the personal computer and to return measured values periodically. The microprocessor is in charge of switching the antenna off when no data is to be transmitted in the measuring network. To program the microprocessor, the source code is first compiled in the PC, and then into the microprocessor via an “MSP430 USB debug interface” and a J-tag connector. The program is stored into the main memory. The total RAM size on the chip is 2048 bytes, of which 340 bytes are used for global variables, which comprises 16% of the total. The main flash ROM has $61,184$ bytes, 22% of which are used for program codes.

Microprocessor programming

The processors are programmed in C. The voltage level of the microprocessor is mapped to the values of C variables. The time between each clock tick is fixed as 1/32 seconds and the corresponding variable of “timer” in the main program can be referred directly to record the current timing counter. The timer is mainly used for turning the antenna on or off.

C-style pseudo code running in the microprocessor

Please refer to Appendix A

2.2.6 Wireless module

Zigbee radio-frequency (RF) modules provide the communication part of the sensor board. A Zigbee RF module, embedded in the sensor board, has 20 pins on it and five of them are used in the board design; they are a data out pin, a data in pin, a reset pin and two power pins.

The Zigbee modules are pre-programmed to communicate with others in a Zigbee protocol, which belongs to a family of high-level communication protocols. Zigbee is designed to achieve low data rate, long battery life, and secure networking. The protocol is designed basing on an IEEE 802 standard for personal area network with a design goal to provide small, low-cost and low-power digital radio communication. The maximum rate for the ZigBee protocol is 250 kbps, which makes Zigbee a good choice for periodic or intermittent data transfer[35]. The measuring frequency in this work is once per minute and it is far lower than 250 kbps.

Radio frequency module specification

The valid communication range for the Zigbee RF modules on sensor boards and base boards is about 45 meters indoors and $1,500$ meters outdoors. The communication range is sufficiently long for most solar panel installations. The modules on the boards are XBee-PRO 900 RF Modules and they are designed to support the unique needs of low-cost sensor networks running at low-power levels. The key features for the RF modules are as follows[2]:

- Maximum RF data rate of 156 kbps
- Low power consumption; the typical voltage is 3.3 volts, and the sleep current is 60 uA
- The module configuration can be modified in real time
- Software provided to test and configure the RF module; each module can be configured by modifying 100 parameters

Data interface

In each module, the functionalities are divided into different layers: the lowest is baseband layer, then the MAC layer, the security layer, data transfer layer, and finally the serial interface layer. The microprocessor program and the PC program that controls the base board both run at the highest serial interface layer.

Data that is represented as the format of an asynchronous serial signal enters the module through the DIN pin. Received data is detected at the DOUT pin of the radio module. The signal line should be idle high when no data is being transmitted. Each data byte consists of 10 bits in the serial data line. Each data byte is formed in order by a combination of a start bit, eight data bits, and a stop bit. The eight data bits, representing numbers from 0 to 255, are in the order of the least significant bit coming first and the most significant bit coming last. The start bit is low while the stop bit is high. For example, to send the data “0x0a”, the bits sent are as follows: 0 (the start bit) 0,1,0,1,0,0,0,0,1 (the stop bit).

According to our testing, the board supports a maximum of 300 data segments transmitted every second. Each data segment is composed of about 2 to 10 data bytes. When the gap between two consecutive data segments is too small, there is a high probability that the consecutive segments are mixed. If the gap between two pieces of data segments is less than 10 milliseconds, then the success ratio for transmitting the data is greatly reduced. Therefore, to make sure that the intermittent data are successfully transmitted, a large enough gap should exist between two consecutive segments of data.

2.3 Base board

The base board, attached to the PC, communicates with the sensor. The base board and the sensor board each contain a peer Zigbee RF module. The RF module on the base board is mounted on a development kit and the development kit communicates with PC via a RS-232 wire or a USB wire. The PC runs a C++ software program that controls the Zigbee RF module on the base board by sending and receiving commands or data via COM ports. The main task of the software is to periodically send out commands to synchronize all the sensor boards, then receive and gather data from distributed sensor boards. The personal computer runs the Microsoft Windows operating system.

Both the base board and the sensor need to carefully control the timing with which data is sent to or received from the communication modules. Timing control is relatively easier to implement on the sensor board than on the base board because no other process or thread is running on the sensor board. On the PC that controls the base board, there are other tasks or processes that are running concurrently and disturb the accuracy of the timing control. The timing control on the PC uses the Windows API called "QueryPerformanceFrequency". This API call is built on a high-resolution performance counter that counts in milliseconds. We considered two other methods to control the timing in Windows system API. The first method was using the "sleep()" function, the second was to inquire about the system time frequently. "QueryPerformanceFrequency" performs better than the other two methods in terms of timing accuracy. The required error tolerance of the timing control is set to about 2 milliseconds and only "QueryPerformanceFrequency" can satisfy this requirement. Note that if the timing error tolerance were to be increased, then the antenna would have to be on for longer in each cycle, and thus results in greater battery consumption. For example, if the error tolerance were increased to 3 milliseconds from 2 milliseconds, then the power consumption would increase by 1%.

Software was developed on the PC to convert the data, whose format cannot be used by the anomaly detection and classification program directly, into "csv" format. The csv file is read by the Matlab program.

2.4 Communication Protocol

2.4.1 Point-to-multipoint topology

To avoid data interference, the sensing systems deployed for two different solar energy system should be running at two different channels. Each sensing system with a different channel from nearby sensing systems forms a measuring subnetwork. Modules within each subnetwork share a unique communication tunnel and modules across different subnetworks do not interfere with each other. In each measuring subnetwork, there are multiple sensor boards, only one base board and one personal computer.

In current solar panel sensor deployment, all RF modules form a point-to-multipoint communication network. The module on the base board broadcasts its command to the entire network while modules on the sensor boards send their messages only to the base board. In fact, the system contains several subnetworks and each subnetwork is labeled with a unique network I.D..

2.4.2 Controlling the Zigbee module

The XBeePro Zigbee modules support two types of interface: a transparent interface and an API interface. We selected the transparent interface because it is easier to work with. The communication module can be in one of four modes of operation when being controlled using the transparent interface: transmit mode, receive mode, command mode, and sleep mode.

The module only transfers data when the module is not in the command mode; if it is in the command mode, then data sent to it are regarded as commands. To enter the command mode, the program sends a particular string pattern to the module following a predefined strict timing pattern. Please refer to Appendix C for details on how to switch between command mode and data mode.

Sleep mode forces the radio frequency (RF) module to be in a state of low power consumption. There are different ways in which the module enters and leaves the sleep mode. The sleeping and waking pattern is controlled by the variable “sleep mode (SM)” in the module configuration. When SM=0, the module will never fall asleep. SM= 0 forces the module to remain awake all the time. The second choice is asynchronous pin sleep mode (SM=1), in which the module sleeps and wakes according to the state of the sleep pin RQ. Third, in the asynchronous cyclic sleep mode (SM=2), the module wakes up after a preset time and does not check the sleep pin. Fourth, in the asynchronous cyclic sleep mode (SM=3), a pin wake up choice is included. The module does not process any data when sleeping.

Please refer to appendix D for verifying the sleep mode.

2.4.3 Low-power synchronization

As the battery capacity is limited, one of the most important design principles is to reduce the electrical energy consumption as much as possible. There are two major power consumption resources, of which the first is the microprocessor that should be powered on continuously and the second resource is the antenna. We now discuss how the antenna resource can be conserved.

Recall that the data transfer rate is one data byte transferred every minute. Therefore, the power consumption can be reduced by turning on the antenna only when there is data ready. To save more battery, the design goal is to set the length of interval of the window during which the antenna is on to be as small as possible, yet not small enough to destabilize the data transmission. The board is running at a baud rate of 9600 bps, making the time to send each bit 1/9600 second. Therefore, it takes $10/9600 * 1000 = 1.04167$ milliseconds to send each byte, which consists of 10 bits.

We found in our experiment that the time taken to send a data segment was between 2 and 10 milliseconds and the time taken to stabilize the antenna before it was ready to transmit was at least 10 milliseconds. It takes approximately another 15 ms for the radio to turn on and stabilize. Therefore, the microprocessor has to control the board antenna so that it would wake up at least 10 milliseconds before the data transmission. The board antenna must keep awake until the board finishes transmitting the last bit of a data segment. After data are transmitted, the sensor will wait for another 40 milliseconds before it enters the sleep mode. Different sizes of sending and receiving windows have been tested to find the possibly minimal window interval length.

Note that, the program on the sensor will not attempt to sleep until the board finishes establishing synchronization, which takes at least three minutes under no transmission errors. The synchronization protocol requires that sensor board receives two synchronization tokens whose interval is exactly 60 seconds before the sensor board starts to send measured values back to the base board. One-token synchronization protocol is an alternative protocol while two-token synchronization provides higher synchronized timing accuracy after synchronization has been established.

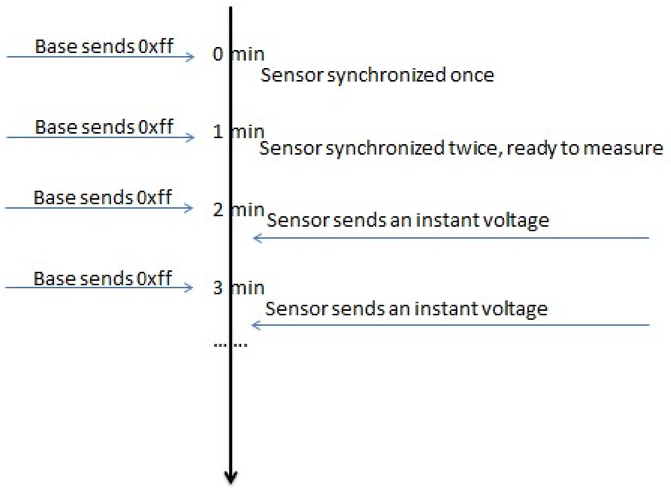

Figure 2.5: Time lines of synchronization

Figure 2.5 illustrates the controlling time table. An example of a successful transmission is shown as follows:

before 0 ms	: The sensor board and the base board are both powered on and stay idle; SM=0 on the sensor board
0ms	: The base board starts to send the first command token of '0xff\n' to the sensor board
2ms	: The sensor board finishes receiving the first token
60, 000ms	: The base board starts to send the second command token of '0xff\n' to the sensor board
60, 002ms	: The sensor board finishes receiving the second token
60, 003ms	: The sensor board starts to enter the command mode
60, 013ms	: The sensor board enters the command mode

119,970ms : The sensor board enters asynchronous cyclic sleep mode (SM=2) and enters the periodical sleep mode

119,975ms : The sensor board exits the command mode and enters the data mode

120,000ms : The base board sends another command token of '0xff\n' to the sensor board and the sensor board starts to receive it

120,002ms : The sensor board finishes receiving the token

120,003ms : The sensor board sends a data segment containing the measured value

120,013ms : The sensor board finishes sending and starts to wait for the next data transmission task; the base board finishes receiving the measured data

120,053ms : No data has been transferred for 40 milliseconds, therefore, the sensor board falls asleep by turning down the antenna

179,970ms : The module on the sensor board wakes up after a predefined amount of time and starts to wait for the next command token

180,000ms : The base board sends a command token to the sensor board

180,013ms : The sensor board receives the token, then sends another measured value back to the base board

180,053ms : The sensor board enters the sleep status again

··· : ···

A synchronization failure occurs when one node fails to receive the anticipated synchronization signal within the desired interval. A failure may occur if the sensor board is in the sleep mode while the base board is sending command token to the sensor board. In case of a synchronization error or a timeout event, it is necessary to re-establish synchronization. To do so, the sensor board pauses the measuring process, then continuously attempts to enter the command mode and set SM=0. After the board sets SM=0, it is programmed to restart the measuring process until its receives another two command tokens from the base board. When SM=0, the antenna remains on all the time, and the sensor will not miss any following command token from the base board, thus reestablishing synchronization.

2.4.4 Battery life analysis

The battery chosen has a capacity of 6 volts and 4.5 Ah. The device has a working current of 2 mA when the antenna is off for 99.83% of one cycle and 60 mA when the antenna is on for 0.16% of one cycle. As the antenna accounts for most of the energy consumed, the energy consumed every minute is $6 * 60 * 10^{-3} * 0.16\% + 6 * 2 * 10^{-3} * 99.83\%$ while the total energy stored in one battery is $6 * 4.5$VAh. Therefore, the total number of charged days for the battery is represented as follows:

$$\frac{6 * 4.5 * 60}{6 * 60 * 10^{-3} * 0.16\% + 6 * 2 * 10^{-3} * 99.83\%} = 129026.1(mins) = 89.6(days)$$

Then, one device can work for three months without recharging or replacing its battery. Another possible approach is to charge the battery using the power from solar panels. However, the approach is not feasible for some solar panels, because it requires modifying the panel structures, and the censor board can only run with a 6-volt power supply.

2.4.5 Measured value

Rather than constituting an instant measurement, the measured voltage value sent by the sensor board is an average value of many measured voltages during a cycle. The average value can be computed without initializing a large buffer to store all past measurements. Supposing the average of all past measurement is $\bar{x'}$ and the number of past measurement is n, then given a new measurement of x, the new average value can be computed as $\bar{x} = \frac{\bar{x'} * n + x}{n+1}$. To reduce the rounding error, the formula is revised to be:

$$\bar{x} = \frac{n}{n+1}\bar{x'} + \frac{x}{n+1}$$

The above method is also used to measure the variance of the measured voltage by maintaining a running sum of the squared measured values.

2.5 Debugging and refit

2.5.1 Debugging methods

The method for testing data transmission is to use an oscilloscope to monitor the output pins of the communication module. Instead of a normal oscilloscope, which is not sufficient to accomplish the testing, a digital storage oscilloscope that supports storing waveforms is required. The waveforms are first recorded and stored, then the stored waveforms are analyzed and compared with the desired waveforms. The storage oscilloscope, set to be triggered by "low", is used to also measure the accurate timing for data transmission.

When the main procedure was tested, a binary search approach was employed to determine the position where the potential error occurs. In the binary search approach, the suspected error zone is reduced by half after each stage. An ordinary program can be debugged by printing information onto the screen or onto the terminal. The microprocessor program can also be debugged by "printing", where "printing" means sending messages from the sensor board back to the base board. When testing synchronization problems, the base board will not get any message back for more than 1 and less than 3 minutes. Therefore, a shorter cycle was tested at first to save debugging time.

Hardware debugging is difficult even for hardware specialists because of the difficulty of obtaining debugging information. We gained some debugging experience during the long process of finding the two critical bugs in the hardware. The first was a version error and the second was a resistor error. If the versions of two modules do not match, then information transmitted between the two boards is incorrect. The version error bug is found by doubting about and investigating every step of data transmission. The "printing" method could not be used for the first bug because the transmission between the sensor board and the base board could not be established due to the bug. We could only use oscilloscopes to monitor the DIN and DOUT pins on the RF modules and modify the RF configuration. Therefore, it took nearly one month to fix the first bug. The second bug was found by using a multimeter to measure the voltages of the resistors and pins on the board.

2.5.2 Increase the measuring capacity of the boards

The measuring capacity of the boards can be increased with slight modifications in terms of maximum measured voltage tolerance, measuring frequency, and data transmission frequency.

To obtain the ability to measure greater voltages, two device configurations must be changed. The conversion ratio of the resistor divider must be further reduced. The original ratio of the divider R3/R11 is 1/31, which makes an input voltage of 70 volts reduced to about 2.3 volts. Supposing that the maximum voltage to be measured is increased to 300 volts, then 300 volts needs to be reduced to around 2.5 volts, requiring the new dividing ratio to be about 1/120. Therefore the $33k$ resistor of R3 should be changed to 1.25 million ohms and R11 can remain $10k$ ohms, making the new dividing ratio $10k/(1250k + 10k) = 1/126$. The software should be changed to reflect the different division as well.

At a higher voltage, a higher power resistor should be maintained to allow larger resistor heat dissipation. The power consumption for 35 volts is computed as follows: $U^2/R = 35 \times 35/310,000 \approx 1/30$ watts. The regular small resistors can safely dissipate approximately 1/4 watts. If the system volt is 600 volts, then the divider resistor should be changed to 2.6 million ohms and the power would be $360k/2.6m \geq 1/8$ watt. Then a resistor that can dissipate at least 1/2 watt, 1 watt preferably, should be employed instead. Even though a 1/4 watt resistor will not explode, it will suffer some accuracy loss due to different temperatures on the 2.6m-ohm resistor and the 10k-ohm resistor. To maximize the accuracy, the resistor family for both should be the same, or similar, so that their temperature coefficients match.

The second type of modification is to increase the board sampling frequency. The master clock of the board is formed by crystals and capacitors; for example, the current configuration is a 4MHZ parallel resonant crystal of 20pf, plus two capacitors of 39pf. A modification in the crystal or capacitor configuration results in a different master clock rate.

2.5.3 Extensibility of boards and measuring capacity

Even though the sensor is designed for normal solar panels, the boards can be used for other purposes. We found that the sensor board can be used for other measurements.

In particular, we used the board for a project in the Civil Engineering department at University of Waterloo to measure the acceleration of a bridge when people walked over it. The board could be modified to measure the acceleration by converting the acceleration sensor reading into voltage values. The voltage meter measured the voltage value and then transmitted the voltage values back to the base board.

Chapter 3

Anomaly detection

A time series is a sequence of data points that are typically measured at successive time instants. We study time series that are generated with uniformly equal time intervals. Time series data, unlike other data structures, is naturally represented using the temporal ordering of measured time. Resembling spatial data analysis, data entries close to each other along the time series are more relevant to each other than elements that are further apart.

Although there are many ways to investigate time series data, this work targets anomalous time series analysis. We attempt to determine the existence of anomalous events and classify them based on the assumption that time series affected by sudden or gradual events will appear differently from unaffected time series. One of the goals of the analysis is to find out the occurrence time of the anomalies and how anomalies affect the solar panels. Intuitively, the effects of the anomalies can be removed to obtain the original time series if the anomaly time, the amplitude, the type of the anomaly is known.

3.1 Algorithm preference

3.1.1 Markov model

A Markov model is a potential approach for anomaly detection. There are two types of Markov models: the general Markov model and the hidden Markov model. For either type, a Markov model assumes that the system can be modeled as a Markov process. A hidden Markov process contains discrete hidden states while a general Markov process contains discrete observable states. Unfortunately, neither model seems appropriate.

Suppose there are two Markov states, which are “anomalous” and “normal”. A solar panel will be in the state of “anomalous” most of time due to persistent anomalies such as dust, and two states will not be sufficient to represent the number of anomaly types. On the other hand, supposing there is one state for each anomaly type, then it is difficult to compute the transition probability of transferring from one state to another. In this problem configuration, it is difficult to find a suitable set of state variables. Therefore, a state-free approach is preferred. Moreover, anomaly detection algorithms in the areas of time series analysis has been more intensively studied than that of the Markov model.

3.1.2 Time series analysis

Time series data are often affected by unexpected intervention factors that are regarded as anomaly formats. The anomaly format detection problem is to identify the locations and the category of the anomaly formats in the time series.

The anomaly format detection problem was first proposed in 1972[20]. The foundation for time series anomaly detection is the intervention model, which was first introduced by Box and Tiao in 1975[8] and was designed to accommodate anomaly effects. Many improved algorithms followed in the next 20 years. Significant proposals include Hillmer and Steven in 1984[22], Tsay in 1986[36], Tsay in 1988[37], Chang, Tiao and Chen in 1988[14], Chen and Liu in 1993[15], Sanchez et al. in 2003[25] . Later algorithms performed better in the model parameter estimation efficiency under anomalies and in anomaly detection accuracy under masking effects.

Two types of anomalies, as well as the first iterative algorithm for detecting anomalies, are presented in “Estimation of time series parameters in the presense of outliers” by Chang, Tiao and Chen in 1988: innovational anomalies and additive anomalies. An algorithm to distinguish an additive anomaly from an innovational anomaly was also proposed. The overall detection algorithm consists of two stages; the first is anomaly-detection stage and the second is parameter estimation stage[14].

In the paper “Joint Estimation of Model Parameters and Outlier Effects in Time Series” by Chen and Liu in 1993[15], a more advanced iterative algorithm for detecting anomalies and two more anomaly types were introduced. The model parameters were estimated based on less contaminated model parameter estimations; the anomaly effects and affected model parameters were estimated jointly and multiple regression is used for anomaly effect estimations[15].

3.1.3 Drawback with previous algorithms

Previous researchers proposed four anomaly formats and previous algorithms only detect at most four types of anomalies, as far as we are able to determine. However, in practice, the number of anomaly types is greater than four. For example, if an anomaly is of an exponentially dropping anomaly, then previous algorithms would detect the algorithm with very high accuracy while if the anomaly is actually a gradually dropping anomaly, then they would not offer sufficient detection and classification accuracy. We address these issues in our work. Previous research is summarized in Chapter 3.2, 3.3 and 3.6; our modifications are introduced in Chapter 3.5 and 3.7.

The terminology of “anomaly format” introduced in Chapter 3 is different from “anomaly type” in Chapter 4. “Anomaly format” is equivalent to “outlier” that is studied in previous research and there are exactly four different anomaly formats in total. On the contrary, the number of anomaly types can be much greater than four. Anomaly formats, which are determined by time series analysis in Chapter 3, are the bases of the computation of anomaly type classification. A time series that is classified as one anomaly type may contain several anomaly formats.

3.2 Time series model fitting

The mathematical basis for our work is the autoregressive integrated moving average model (ARIMA), defined by[9]:

$$\varphi(B)z_t = \phi(B)\nabla^d z_t = \theta(B)a_t$$

In the above equation,

B^q: the letter of B denotes a backward operator commonly used in time series, meaning that $Bz_t = z_{t-1}$; B^q means the B operator applied q times

$\phi(B)$: the autoregressive operator where p denotes the order; $\phi(B) = 1 - \phi_1 B - \phi_2 B - \cdots - \phi_p B^p$

∇^d: $\nabla^d = (1-B)^d$, the differentiating factor where d is the order

$\varphi(B) = \phi(B)\nabla^d$: the generalized autoregressive operator where $p + d$ is the order

$\theta(B)$: moving average operator that is assumed to be invertible and where q is the order. $\theta(B) = 1 - \theta_1 B - \theta_2 B - \cdots - \theta_q B^q$. The proposed algorithm assumes that this moving average operator is invertible

a_t: a white noise process with zero mean and unit variance. Entries in the process are independent from each other

z_t: the originally observed values

3.3 Anomaly major formats and estimation

The following model is considered to describe a time series subject to the influence of an anomaly:

$$Y_t = A_t + z_t \tag{3.1}$$

As the formula is derived from the intervention model, an anomaly is represented as an intervention. In the above formula, z_t describes an ARIMA time series without any intervention; A_t is used to describe the intervention effect due to anomalies; Y_t is the actually observed time series. A_t is the core part to be studied in the intervention model because A_t contains the information about the anomaly[8].

The combined equation for the intervention term is $A_t = \omega R_t P_t^{(T)}$ assuming T is the time when the anomaly happens. As stated in the formula, A_t is composed of three parts: first, the amplitude denoted as ω, second, the regression part denoted as R_t, third, the occurrence function which indicates the start and the end of the anomaly. The occurrence function can be either a step function whose formula is

$$S_t^{(T)} = \begin{cases} 0 & \text{if } t < T \\ 1 & \text{if } t \geq T \end{cases}$$

or a pulse function whose formula is

$$P_t^{(T)} = \begin{cases} 0 & \text{if } t \neq T \\ 1 & \text{if } t = T \end{cases}$$

Only the second occurrence function type is considered in this work because it is easier to manipulate and the two types are equivalent if the integrated order is adjusted. It can be proved that[10]

$$(1 - B)S_t^{(T)} = P_t^{(T)}$$

The intervention models can be expressed in four different major formats according to different representations of R_t[14][15]. These are

- $R_t = 1$, additive anomaly (AA), such as a recording typo at a particular time
- $R_t = \frac{\theta(B)}{\nabla^d \phi(B)}$, innovational anomaly (IA), such as an anomalous event that has effects on all the observations after the anomaly time
- $R_t = \frac{1}{(1-B)}$, level shift anomaly (LS)
- $R_t = \frac{1}{1-\delta B}$, temporary change (TC)

Assuming that the accumulated effects of multiple anomalies can be achieved by adding all individual effects together and that the anomalies are taking place at time points of $T_1, T_2, \cdots, T_k$, the multiple-anomaly model can be written as the following form,

$$Y_t = \sum_{j=1}^{k} \omega_j R_j(B) P_t^{(T_j)} + z_t$$

$k = 1$ when there is only one anomaly. Assuming that anomaly type is an innovational anomaly (IA) and the anomaly time is T, the equation reduces to

$$Y_t^{IA} = \omega \frac{\theta(B)}{\phi(B)\nabla^d} P_t^{(T)} + z_t = \omega \frac{\theta(B)}{\varphi(B)} P_t^{(T)} + \frac{\theta(B)}{\varphi(B)} a_t \tag{3.2}$$

Then, z_t is defined by $z_t = \frac{\theta(B)}{\phi(B)(1-B)^d} = \frac{\theta(B)}{\varphi(B)} a_t$ under the assumption that z_t fits the ARIMA model of order (p, d, q) where p is the order of $\phi(B)$, d is the order of $(1-B)^d$, $p+d$ is the order of $\varphi(B)$, q is the order of $\theta(B)$.

The rest three formats can be defined similarly as

- the additive anomaly (AA),

$$Y_t^{AA} = \omega P_t^{(T)} + z_t = \omega P_t^{(T)} + \frac{\theta(B)}{\varphi(B)} a_t$$

- the level shift anomaly (LS),

$$Y_t^{LS} = \omega \frac{1}{1-B} P_t^{(T)} + \frac{\theta(B)}{\varphi(B)} a_t$$

- the temporary change (TC),

$$Y_t^{TC} = \omega \frac{1}{1-\delta B} P_t^{(T)} + \frac{\theta(B)}{\varphi(B)} a_t$$

When $\delta = 0$, the temporary change anomaly Y_t^{TC} reduces to the additive anomaly Y_t^{AA}; and when $\delta = 1$, Y_t^{TC} reduces to the level shift anomaly Y_t^{LS}. Therefore, an extra constraint of $0 < \delta < 1$ should be applied to the definition formula for the temporary change anomaly.

The additive anomaly only affects the observation at the time T while in the innovational anomaly model, the pulse at time T affects all the succeeding time points. Suppose that $\psi(B) = \frac{\theta(B)}{\varphi(B)}$, then for all the time points later than T, $Y_t = \omega\psi_i + z_t$ where $t = T + i \geq T$. The effect of an innovational anomaly can either decay gradually to 0 or remain at a certain level for a long time.

For convenience of analysis, $\pi(B)$ is introduced as the inverse of $\psi(B)$[9], then

$$\pi(B) = \frac{\phi(B)\nabla^d(B)}{\theta(B)} = 1 - \Sigma_{i=1}^{\infty} \pi_i B^i = 1 - \pi_1 B - \pi_2 B - \pi_3 B - \cdots \quad (3.3)$$

$\pi(B)$ is valid only if $\theta^{-1}(B)$ is invertible; otherwise, the coefficients of $\pi(B)$ would be unbounded and $\pi(B)$ would be insignificant. If $\pi(B)$ is multiplied to both sides of the anomaly equation(3.1), then the original time series z_t reduces to a_t, a white noise process. In summary, the original anomaly function $Y_t = \omega R(B)P_t^{(T)} + z_t$ is updated to

$$\pi(B)Y_t = \omega\pi(B)R(B)P_t^{(T)} + \pi(B)z_t = \omega\pi(B)R(B)P_t^{(T)} + a_t$$

$e_t = \pi(B)Y_t$ is called the residual. Given that an anomaly takes place, the effects will spread to the residual as they spread to the original series, the residual is employed as the core factor to detect or classify anomalies as it contains significant information. Different residual equations for the four different major types are as follows,

- IA: $e_t = \omega P_t^{(T)} + a_t$
- AA: $e_t = \omega\pi(B)P_t^{(T)} + a_t$
- LS: $e_t = \omega\frac{\pi(B)}{1-B}P_t^{(T)} + a_t$
- TC: $e_t = \omega\frac{\pi(B)}{1-\delta B}P_t^{(T)} + a_t$

which equation can be simplified to $e_t = \omega x_t + a_t$ if x_t denotes $\pi(B)R(B)P_t^{(T)}$[15]. Commonly for four major formats, $x_t = 0$ when $t < T$ and $x_t = 1$ when $t = T$. They differ in the time points later than T, therefore for $k \geq 1$,

- IA: $x_{T+k} = 0$
- AA: $x_{T+k} = -\pi_k$
- LS: $x_{T+k} = \delta^k - \sum_{j=1}^{k-1}\delta^{k-j}\pi_j - \pi_k$
- TC: $x_{T+k} = 1 - \sum_{j=1}^{k}\pi_j$

x_t contains the features of the anomaly, thus by checking $x_1, x_2, \cdots, x_n$, it is determined which anomaly format occurs, when it starts, whether it will end and how much the anomaly affects the actually observed time series[15]. In the equation of

$$e_t = \omega x_t + a_t, \tag{3.4}$$

if e_t is taken as the output, x_t as the input, ω as the gradient, and a_t as the bias, then this equation fits into a standard linear regression function. Therefore the common maximum likelihood estimation of the four formats for ω is

$$\hat{w} = \frac{\sum_{t=1}^{n} e_t x_t}{\sum_{t=1}^{n} x_t^2}$$

For innovational anomalies, the least square estimator of the anomaly magnitude of ω is e_T and the variance of the estimator is σ_a^2. The estimator for the additive anomaly can also be written as

$$\hat{\omega_{AA}} = \frac{e_T - \sum_{i=1}^{n-T} \pi_i e_{T+i}}{\sum_{i=1}^{n-T} \pi_i^2}$$

The amplitude estimations for $\hat{w}$ at the last time point of the time series are equal for four different major anomaly formats. As a result, it is impossible to determine the format of an anomaly taking place at the end of a series[15].

From [14], a common approach to determine the anomaly time and anomaly format is to consider the likelihood ratio of four different anomaly types at different time points. The likelihood ratio is obtained by comparing the likelihood of the anomaly's occurrence and that of nothing taking place. Logically, the condition of $\omega \neq 0$ is equivalent to that an anomaly happens while the condition of $\omega = 0$ is equivalent to situations that nothing happens.

In the following, $\lambda(T)$ denotes the likelihood ratio of anomaly happening at time T, *norm* is defined by $norm(x_1 \cdots x_n, 2) = (\sum_{t=1}^{n} x_t)^{1/2}$, and the standard deviation is estimated by $\sigma_a = \frac{\sum_{t=1}^{n} e_t^2}{n}$[14][37]. From standard statistics,

- IA:

$$\lambda(T) = \frac{\omega(T)}{\sigma_a} = \frac{e_T}{\sigma_a}$$

- AA:

$$\lambda(T) = \frac{norm(x_1 \cdots x_n, 2) \cdot \omega(T)}{\sigma_a} = \frac{\hat{\omega_{AA}}}{\sigma_a}$$

- LS:

$$\lambda(T) = \frac{norm(x_1 \cdots x_n, 2) \cdot \omega(T)}{\sigma_a}$$

- TC:
$$\lambda(T) = \frac{norm(x_1 \cdots x_n, 2) \cdot \omega(T)}{\sigma_a}$$

The entire anomaly estimation procedure is based on the subproblem of how to determine the likelihood ratio of an anomaly given fixed time and anomaly format. For a fixed time point, the anomaly format providing highest likelihood ratio is chosen as the dominating format at that time point. Afterwards, the time point with highest likelihood ratio provided by its dominating format is chosen as the final anomaly happening time.

3.4 Overview of the jointly iterative procedure for anomaly detection

The goal of the iterative procedure is to detect one anomaly in each iteration and the iteration is terminated when an obvious anomaly can no longer be found. In each iteration, the algorithm identifies an anomaly which is most likely to have occurred and associates it with one category.

The pseudo code for single anomaly detection is:

1. At the first stage of the procedure, the ARIMA model is estimated to obtain $\theta(B)$, $\phi(B)$, $(1-\nabla)^d$ and $\varphi(B)$ assuming no anomaly

2. Compute $\pi(B)$ as $\pi(B) = \frac{\phi(B)(1-\nabla)^d}{\theta(B)}$

3. For each anomaly time t in the interval of $[1, n]$, the residual e_t is computed for each major anomaly format with the estimated ARIMA model parameters and the actually observed time series of Y_t. The residuals are then employed to estimate the amplitude of ω for each major format. The estimated ω, the estimated σ and the estimated x_t are then used to estimate the likelihood ratio λ of each major anomaly format at t[37][14]

4. The algorithm continues to select a time T with the greatest β_T as a potential anomaly time point. β_T, introduced in Chapter 3.5, is defined by incorporating a support vector machine classifier

5. Use low order approximation (introduced in Chapter 3.7) to approximate $\psi(B)$, which is the inverse of $\pi(B)$, and compute the effect of the anomaly using $\psi(B)$ and the residuals. The effect of the detected anomaly is removed to obtain an updated time series

The iterative multiple anomaly detection algorithm is based on the single anomaly detection. Its pseudo code is:

1. Run the single anomaly detection algorithm for multiple times to obtain a set of detected anomaly formats. Each detected anomaly format has a determined format category (IA, AA, LS or TC), a determined character vector of x from Chapter 3.3 and a determined anomaly time step

2. Apply multiple regression (introduced in Chapter 3.6) to determine the amplitude for each detected anomaly[15]

For the joint estimation of time series anomalies, please refer to Appendix B for the Matlab-style code in more detail. The code in the appendix involves model parameter re-estimation for reducing the influence of detected anomalies on the ARIMA parameters.

3.5 Anomaly format identification

We use a support vector machine (SVM), which is intensively used in the classification and regression model, for anomaly format identification. In a support vector machine, each item is located uniquely in R^n. The support vector machine finds a hyperplane that separates clusters in the input data as much as possible[6]. The distance between each item and the separating hyperplane shows how far it lies from the edge of the category.

The simplest support vector machine can be made by drawing a line on a piece of paper and declaring that all the points above the line to be in the first category and all the points below the line to be in the second category. The points forming the border of two categories are called support vectors. In a two-class support vector machine classification problem, each data item is classified according to the value of $y(x) = w^T\phi(x)$. Let the training data set comprise N input vectors $x_1, x_2, \cdots, x_N$ and N target values $t_1, t_2, \cdots, t_N$. $t_i \in [-1, 1]$.

Assuming that the training data is linearly separated in space and a hyperplane is defined by $y(x) = 0$, there will be some choices of parameters w and b such that the function values $y(x)$ are greater than zero for points having $t_n = 1$ and less than zero for points having $t_n = -1$. The perpendicular distance of a point x from the hyperplane is $\frac{|y(x)|}{||w||}$. The *margin* is defined as the perpendicular distance to the closest point x_n from the data set. The maximum margin solution is found by solving the following equations[11]:

$$arg\ max_{w,b}\{1/||w||\ min_n[t_n(w^T\phi(x_n) + b]\}$$

For the kernel version support vector machine, the classification equation for the test data can be denoted as follows:

$$c = \sum_i w_i k(s_i, x) + b \tag{3.5}$$

where all w_i are weights of support vectors, k is the kernel function, s_i are a set of support vectors, x is the input vector whose category we wish to determine, and b is the bias[17]. If $c \geq 0$, then the data belongs to the first category; otherwise $c < 0$, and the data belongs to the second category. In the case of $-1 < c < 1$, even though c is categorized, the absolute value of c is so small that the categorization is unconvincing.

The simplest linear kernel functions were employed at first but we found that these failed to categorize the data well. Therefore, the radial basis function, given by $r = \|x_1 - x_2\|$ and $\phi(r) = e^{-(\epsilon r)^2}$ was later used instead. $\phi(x)$ transforms the original data into a new feature-space. This function has much better performance.

Due to the design goal of detecting multiple simultaneous or concurrent anomalies, a multiple-class support vector machine, instead of a two-class support vector machine, is needed. However, the design of multiple-class support vector machines still remains an open problem. An alternative approach is to apply several two-class support vector machines one after the other to simulate a multiple-class support vector machine. A set of two-class support vector machines with all the possible pairs of categories is needed during the computation. The category which wins most of the competition rounds is selected to be the final category[18]. Alternatively, all the absolute values of c in all pairs of support vector machines are summed up.

Specifically, we define a metric β_t value as:

$$\beta_t = \lambda_t + C_t \tag{3.6}$$

where λ_t is the likelihood ratio and $C_t = \sum_{1\leq i,j\leq 4} |c_t^{(i,j)}|$. $c_t^{(i,j)}$ is from formula (3.5) where (i,j) denotes the support vector machine that operates format i and j. The larger the value of $c^{(i,j)}$, the more easily the anomaly format can be classified. β_t hence describes how easily an anomaly at time t can be classified or how close an anomaly at time t is to a known anomaly format. If β_t is great, then it can judged that the anomaly at time t is nearly a known anomaly, while if β_t is small, then it can be judged that the anomaly at time t cannot be easily classified. Suppose there are two anomalies of format A and format B happening with the same likelihood ratio, whereas the first anomaly format is closer to a standard anomaly than to the second anomaly. The algorithm will then prefer the first anomaly format in the computation process.

The key idea is that anomaly detection and anomaly identification are merged and conducted simultaneously. Fortuitously, the support vector machine functions and related classification functions are already supported by the Bioinformatics Toolbox in Matlab.

3.6 Multiple Regression

To solve the problem of the interference between neighboring anomalies, multiple regression is used to jointly estimate the amplitudes of multiple anomalies. Suppose that there are k concurrent anomalies so that every observation is affected by k anomalies simultaneously; therefore, every single time point in the residual series is also affected by the k anomalies. Each anomaly format has a characteristic vector determined by the anomaly type and by the anomaly time point. x^i is the characteristic vector for the i-th anomaly. The residual at time t can be written as $\pi(B)Y_t = e_t = \omega_1 x_t^1 + \omega_2 x_t^2 + \cdots + \omega_k x_t^k$ and $x^i = [x_1^i, x_2^i, x_3^i, \cdots, x_n^i]$, where n is the length of the time series. In summary, an equation set can be obtained as follows:

$$e_1 = \omega_1 x_1^1 + \omega_2 x_1^2 + \omega_3 x_1^3 + \omega_4 x_1^4 + \cdots + \omega_k x_1^k + a_1 \quad (3.7)$$

$$e_2 = \omega_1 x_2^1 + \omega_2 x_2^2 + \omega_3 x_2^3 + \omega_4 x_2^4 + \cdots + \omega_k x_2^k + a_2 \quad (3.8)$$

$$\cdots \quad (3.9)$$

$$e_n = \omega_1 x_n^1 + \omega_2 x_n^2 + \omega_3 x_n^3 + \omega_4 x_n^4 + \cdots + \omega_k x_n^k + a_n \quad (3.10)$$

The equation set above has two dimensions of n and k; n is also the dimension of training data. The effect of the anomaly at every time point in the time series is obtained by linearly accumulating all effects of individual anomalies.

We assume that the residual arises from the time series as well as a white noise term. The assumption of Gaussian noise is that, given x, the distribution of e is unimodal, where e is the factor of the residuals from the time series. As shown by $e_t = \omega^T x + a_t$, where a_t is a Gaussian white noise process with zero mean and variance of σ, the residual value is given by a linear deterministic function plus a noise. Consequently, it leads to a Bayesian equations of $p(e|x,\omega,\sigma) = N(e|\omega^T x, \sigma^{-1})$ where e is a general residual point. In case of a squared loss function, the optimal fit for a multiple regression is given by the conditional mean of the target variable. If a Gaussian conditional distribution is assumed, then the conditional mean would be $E[e|x] = \int ep(e|x)dt = y(x,\omega)$. In this multiple regression model, the data points are drawn independently from the distribution. As they are independent, the likelihood function of the residual is obtained as follows:

$$p(e|X,\omega,\sigma) = \prod_{j=1}^{n} N(r_j|\omega^T)$$

Since univariate Gaussian has been thoroughly studied, it is straightforward to take the logarithm of the previous equations to obtain the likelihood function:

$$\ln p(e|\omega,\sigma) = \sum_{j=1}^{n} \ln N(e_j|\omega x,\sigma) = \frac{n \cdot \ln \sigma}{2} - \frac{n \cdot \ln(2\pi)}{2} - \sigma E(\omega)$$

The sum of squares loss error function is defined as $E_D(\omega) = \frac{\sum_{j=1}^{n} e_j - \omega x^2}{2}$. Our goal is to compute ω and σ to maximize the likelihood function. The gradient of the log likelihood function is:

$$\Delta \ln p(e|\omega,\sigma) = \sum_{j=1}^{n} e_n - \omega x x^T$$

ω can be estimated by setting the above gradient to zero:

$$0 = \sum_{j=1}^{n} e_j x_j^T - \omega(\sum_{j=1}^{n} x x^T)$$

As a consequence, the maximum likelihood estimation of the ω is $\omega_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{e}$[6], in which the matrix of $(\Phi^T\Phi)^{-1}\Phi^T$ is called the Moore-Penrose pseudo-inverse matrix[33][21].

Moreover, the maximum likelihood estimation of the noise is calculated as[6]:

$$\frac{1}{\sigma} = \frac{\sum_{j=1}^{n} (e_j - \omega_{ML} x)^2}{n}$$

Once the list of the anomaly formats and the ARIMA model parameters have been determined, the residual vector $[e_1, e_2, \cdots, e_n]$ and the matrix of $[x_j^i (1 \leq i \leq k, 1 \leq j \leq n)]$ are both determined as well. Therefore, the amplitude of these anomaly formats can be calculated using the multiple regression technique directly.

One of the assumptions of linear multiple regression is that each data entry is independent of the others. However, the assumption does not necessarily hold in our work, where neighboring residuals are highly correlated. The alternative approach is to attempt a multiple non-linear regression model. The previous kernel is $y(x, w) = w_0 + w_1 x_1 + \cdots + w_D x_D$ while the revised kernel is $y(x, w) = w_0 \phi_0(x) + w_1 \phi_1(x) + \cdots + w_D \phi_D(x)$, where $\phi_i(x) = (x_i + x_{i+1})/2 (1 \leq i < n)$ and $\phi_n(x) = x_n$. The new kernel applies the average value between two consecutive entries and thus correlates the neighboring values more deeply than the previous one. The multiple regression is later used for approximating the higher order polynomials using lower order polynomials.

3.7 Low order approximation

A model with higher order better fits the original time series. However, as indicated by stability testing, the use of a higher order model introduces a greater chance of instability in Chapter 3.7.1, which means a lesser chance for the inverse to be convergent. Low order approximation is needed for removing the effect of one anomaly at the end of each anomaly detection iteration. The effect of one anomaly is computed by multiplying $\psi(B)$ to the residual e. When the ARIMA model order is greater than two, $\psi(B)$ is always instable and needs to be approximated by a lagging polynomial of lower order.

It is time-consuming to compute $\psi(B)$, the inverse of $\pi(B)$. Consequently, a lower order approximation algorithm is recommended to approximate the lagging polynomials with higher order. The challenge is using $1 + \psi_1 B + \psi_2 B + \psi_3 B + \cdots + \psi_m B$ to approximate $1 + \psi_1 B + \psi_2 B + \cdots + \psi_k B$, where $m < k$, when the higher order equations and the training data for lower order equations are known:

$$\begin{aligned}
(1+\psi_1 B+\psi_2 B+\cdots+\psi_k B)z_1 &= (1+\psi'_1 B+\psi'_2 B+\psi'_3 B+\cdots+\psi'_m B)z_1 \quad (3.11)\\
(1+\psi_1 B+\psi_2 B+\cdots+\psi_k B)z_2 &= (1+\psi'_1 B+\psi'_2 B+\psi'_3 B+\cdots+\psi'_m B)z_2 \quad (3.12)\\
&\cdots \quad (3.13)\\
(1+\psi_1 B+\psi_2 B+\cdots+\psi_k B)z_n &= (1+\psi'_1 B+\psi'_2 B+\psi'_3 B+\cdots+\psi'_m B)z_n \quad (3.14)
\end{aligned}$$

The left side in the equation set is of higher order while the right side is of lower order. The values on the left are known numbers and $[z_1, z_2, \cdots, z_n]$ is also known. The problem can be solved by approximating $[\psi'_1, \psi'_2, \cdots, \psi'_m]$ using multiple regression. In brief, the multiple regression model is the key to solve the problem of an instable polynomial.

3.7.1 Stability testing

Stability testing incorporates a small numerical tolerance and the stability condition requires that the absolute values of all roots of the characteristic polynomials are less than 1 minus the small numerical tolerance. Polynomials with a degree of zero are always stable. Table 3.1 shows the relationship between polynomial orders and their probabilities of being stable. The lagging polynomials are in the format of $1 - \phi_1 B - \phi_2 B - \cdots - \phi_n B$ where n is the polynomial order and $\|\phi_i\| < 1$ and ϕ_i are randomly generated between 0 and 1 excluding 1. The polynomials are guaranteed to be stable with orders less than 3 and remain unstable with higher probability when the orders are increasing.

Table 3.1: Stability statistics

order	percentage (%)
1	100
2	100
3	70.4
4	62.5
5	42.2
6	31.9
7	19.2
8	15.5
9	7.7
10	5.6
11	3.3
12	1.7
13	0.6
14	0.3
15	0.8

Chapter 4

Anomaly classification

The time series z_t generated from the solar panels are pre-processed using the techniques in Chapter 4.2 at first. The anomaly detection algorithm proposed in Chapter 3 is then used to detect all anomaly formats contained in the pre-processed time series. One piece of classification data is assembled by concatenating the pre-processed time series, the characteristic vector for z_t, and the anomaly effect for z_t. The characteristic vector for the entire time series z_t is computed as the sum of characteristic vectors for all detected anomaly formats and the anomaly effect for z_t is computed as the sum of anomaly effects of all detected anomaly formats. Afterwards, the classification algorithms described in Chapter 4.3 are used to classify the assembled data.

4.1 Anomaly types

Figure 4.1 to Figure 4.10 illustrates the graphs of the ten types of anomalies considered in the project in order. Each type of anomaly is shown in one row, where the left graph illustrates the originally measured values and the right graph illustrates the revised graph after the data pre-processing operation described in Chapter 4.2. The ten types of anomalies are described as follows:

1. Permanent shadow: Shadow occurs on the solar panel surface and lasts until the end of the observation

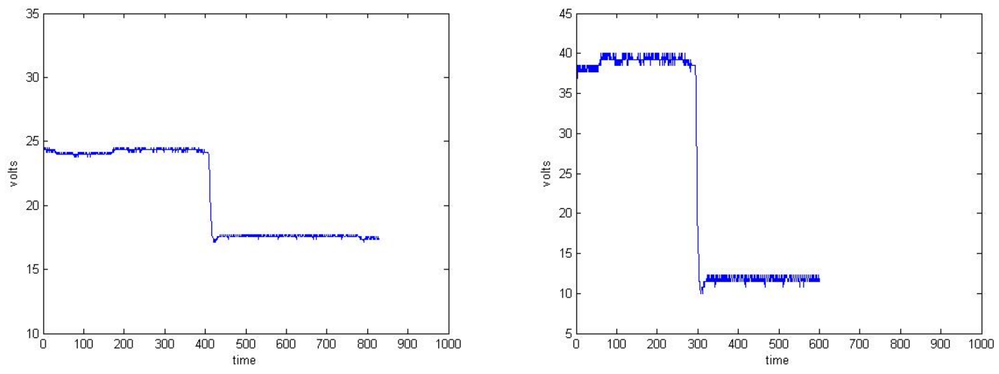

Figure 4.1: Anomaly Type 1

2. Brief shadow: Shadow occurs on the solar panel surface and disappears before the end of the observation

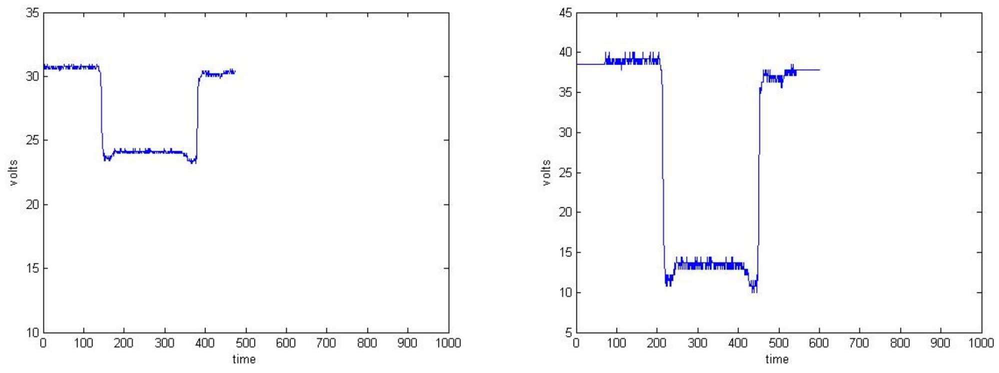

Figure 4.2: Anomaly Type 2

3. Leaves accumulating: Large leaves falls on the panel surface one by one and they remain on the surface to the end.

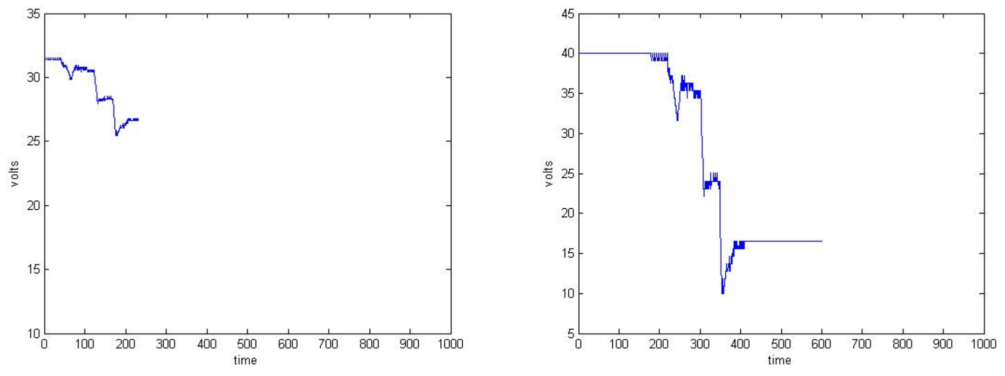

Figure 4.3: Anomaly Type 3

4. Clouds: Moving clouds shade the panel surface, causing a maximum of 20% voltage drop. The effects of the shadow increase gradually and then decrease gradually.

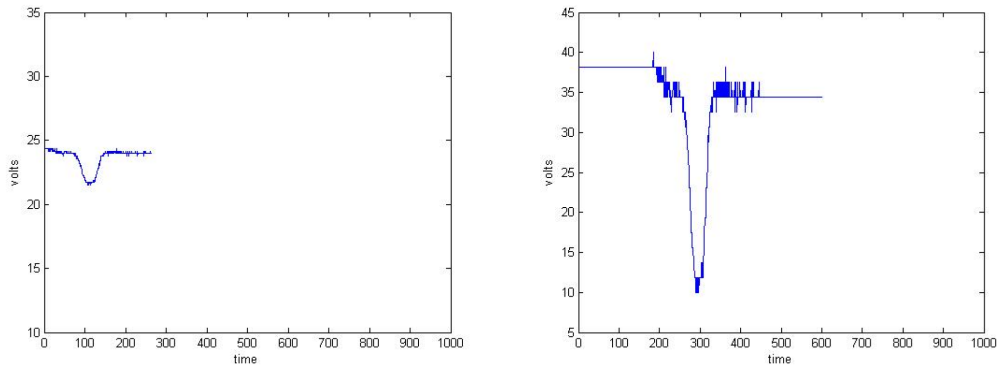

Figure 4.4: Anomaly Type 4

5. Dust or snow: Dust or snow accumulates on the panel surface gradually, causing a steady and slow drop of the solar panel energy output

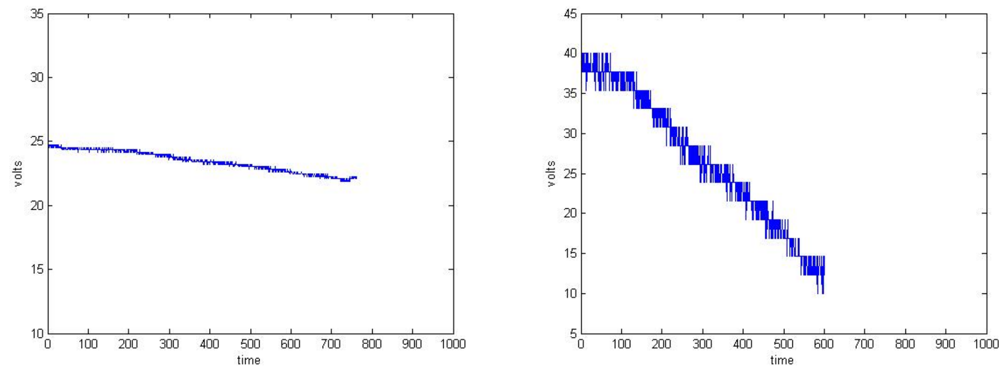

Figure 4.5: Anomaly Type 5

6. Leaves falling down and going away: Large leaves fall upon the panel surface, and are then blown away after staying on the surface for a certain amount of time. There are more than one batch of leaves falling down and flying away

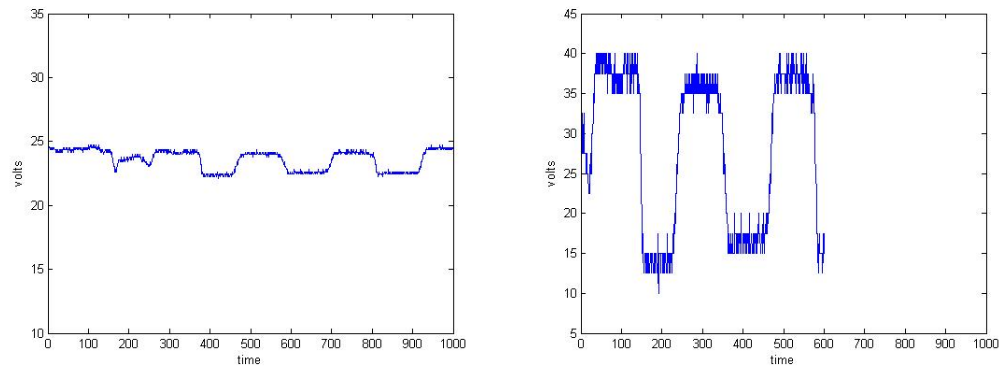

Figure 4.6: Anomaly Type 6

7. Long and gradual shadow: Shadow occurs on the panel surface and last for a sufficient amount of time before it disappears

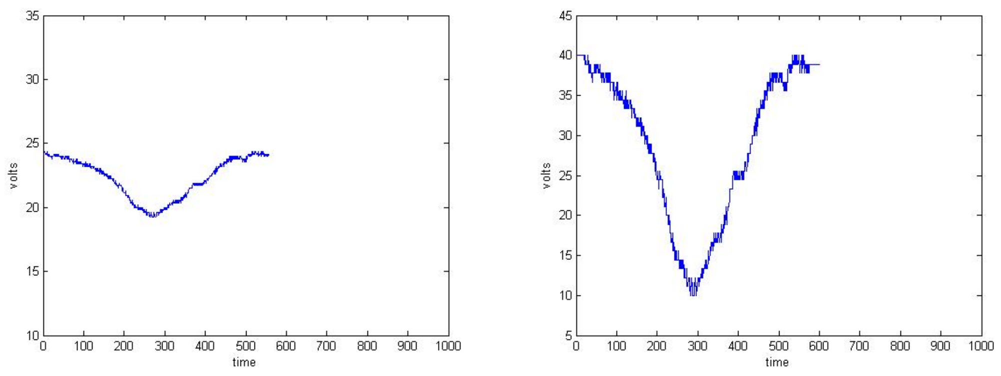

Figure 4.7: Anomaly Type 7

8. Birds: This type of anomaly assumes a large bird suddenly lands on the solar panel, stands for a while and then flies away. It does not necessarily have to be a bird because the program can only be aware of the shadow effects rather than the appearance of the object

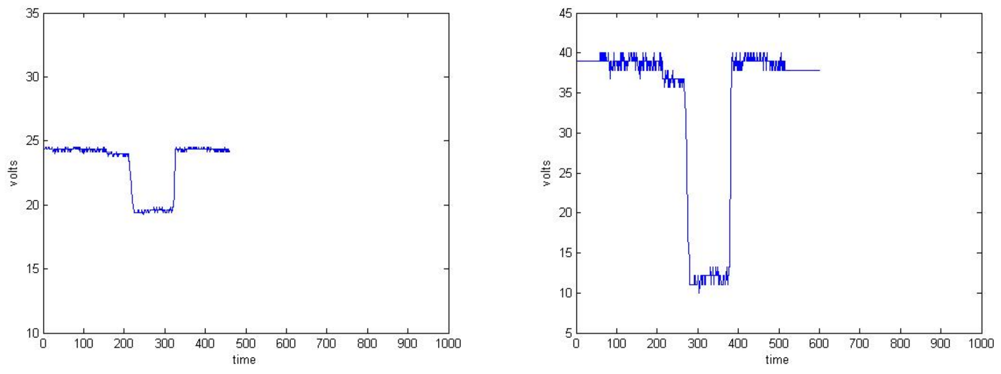

Figure 4.8: Anomaly Type 8

9. Shadow disappears: Opposed to the permanent shadow, the shadow exists from the beginning and disappear suddenly before the end

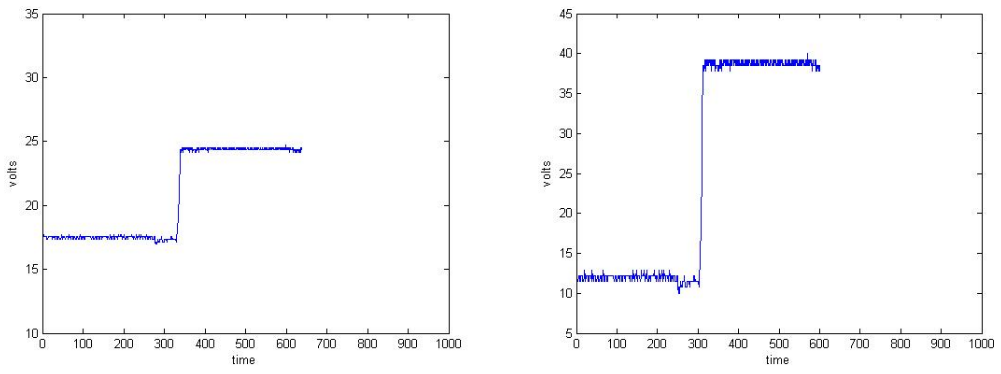

Figure 4.9: Anomaly Type 9

10. Snow melts down: Snow has been accumulated by the beginning and melts down gradually since then

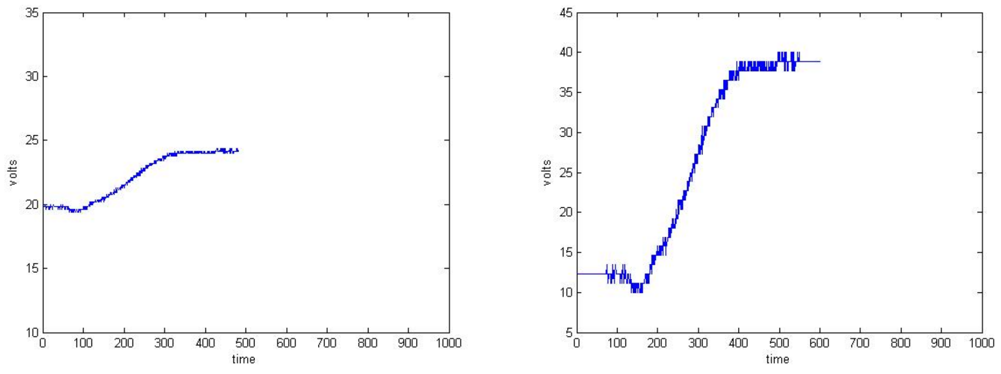

Figure 4.10: Anomaly Type 10

4.2 Data pre-processing

Data pre-processing is the first step to remove shifting factors and scaling factors before data is ready for anomaly detection and anomaly classification. An anomaly can take place at any time point along the time series. The data pre-processing involves reconstructing the time series fragment so that the anomaly time point is placed in the center of the new fragment.

4.2.1 Range scaling

The measured voltage from a sensor can be from a large range of values. It is difficult to create a database which covers all these scenarios. For example, the small solar panels tested in Chapter 5 generate voltages as low as 2.5 volts while the large solar panels on the roof of a building generate voltages as high as 45 volts. A linear regression approach is employed to rescale the input time series by calculating the center point of the time series and then shifting or scaling the time series to a standard form.

To create a standard time series anomaly database, similar to a canonical signature for an anomaly type, all the anomalies are rescaled to be from a standard solar panel whose average voltage output is 25 volts, whose low point energy output is 10 volts and whose high point energy output is 40 volts. All the original time series are transformed as needed so that the time series appears to be generated from a standard solar panel. The simplest approach is to first compute the maximum value and minimum value in a time series segment and then scale the time series according to the ratio between its range of the high point to the low point and the close interval of $[10, 40]$. However, in the presence of some flawed observations in the time series segment, the approach does not perform well enough because the minimum or maximum would be influenced by the flawed observation. Therefore, when spikes and measuring errors are present in the time series, the improved approach is to apply linear regression to fit the time series.

After the linear approximation equation is estimated, the low and high point values in the time series can be approximated by the low and high points in the linear equation. The high point and the low point estimated from linear regression can be later revised by the maximum and minimum of the time series using weights.

4.2.2 Centroid point computation

For ease of classification, we would like to have the anomaly centered in the time series. It is easy to compute the time step that corresponds to the centroid of the time series values.

If z_i is the time series values, the centroid is given by

$$\frac{\sum_i i \cdot z_i}{\sum_i z_i} \tag{4.1}$$

If the time series contains only one anomaly, then the centroid point is assumed to be the anomaly time point.

4.3 Classification Algorithms

Given a rescaled and centered anomaly, we now wish to classify it according to the underlying physical phenomena. Many different classification algorithms are well known in the literature. We implemented several of these classifying algorithms and compared in terms of their performance. There are some common characteristics among different classification metrics. The input for these training examples are vectors in a multidimensional feature space. For every classification problem, the dimensionality of the input and the output is also fixed because the dimensionality must fit the problem specification.

4.3.1 k-nearest-neighbors algorithm

The k-nearest-neighbors algorithm, considered to be the simplest algorithms of all machine learning algorithms, classifies an object according to its closest neighbors in the training example space. The class assigned to the object is the majority class among the k nearest neighbors of the class. In the simplest case where $k = 1$, the class assigned to the object is the class of the nearest neighbor of the object. The k-nearest neighbor algorithm can also be used for regression and for computing the decision boundary of different classes. The distance metrics of different vectors can be of Euclidean distance or of Hamming distance. With different choices of the value k, the k-nearest neighbor algorithm offers different performance. As the value of k becomes larger, the model can tolerate more noise while as the value of k gets smaller, the model becomes more sensitive to noise.

4.3.2 Neural networks

A neural network, as an efficient pattern classification algorithm, is a mathematical method which simulates a network or circuitry of biological neurons[23]. The basic components of a neural network are artificial neurons and nodes. In a real biological system, neurons are connected and related in functionalities. A group of neurons can act together to implement some specific functionality. An artificial neural network, which is used for abstraction of large networks, tries to represent a highly complex real biological nervous system using a simple linear system. In terms of training neural networks, error back-propagation is an efficient training technique[6].

4.3.3 Convolutional neural network

Convolutional neural networks (CNN)have been widely used for handwritten digit recognition[28]. Convolutional neural networks involve a basic unit called a feature map plane that is used for analyzing the connectivity patterns between different layers of neurons. A feature map first applies a linear filter to the original input, then adds a bias term and a non-linear function to the input. In the feature map, each unit takes points only from a specific subregion of the original input. Each feature map has a matrix of adjustable weight parameters plus one adjustable bias parameter that forms a linear transformation with other parameters. The training procedure for convolutional neural networks is also back-propagation[6].

A feature of convolutional neural networks is that it is robust to simple transformations and it meets the requirement of classifying input correctly regardless of translations, scaling, and shifting[31]. Normally shifting should be small; however, if the shifting is great, a better algorithm to deal with a great shifting would be needed. The simplest approach to solve this problem would be to train the learning machine with sufficient data. Nevertheless, it is difficult to generate one test case in this problem configuration considering that each case takes several minutes to create. Another characteristic of this problem configuration is that the measured voltage values are represented in time series format, and the order of data entries is important; therefore, information is hidden in the ordering of time series.

There are some techniques which attempt to detect common local connections between nearby data points. Most of them belong to the category of local feature extraction, a computer vision technique that exploits local properties in small subregions of the images. A hierarchical neural network can be built based on local feature extraction. In the hierarchical network model, local features processed in the beginning are used for later processing. A neural network is always composed of an input layer, a hidden layer, and an output layer, while the more complex convolution neural network employed in this work is designed to detect variabilities in the input vector.

Chapter 5

Evaluation

We manually created training data and stored the data in a database described in Chapter 5.1. The algorithm from Chapter 3 is evaluated in Chapter 5.2 in terms of its ability to detect the existence of anomalies. The classification algorithm proposed in Chapter 4 is evaluated in Chapter 5.3 about its performance of classifying the detected anomalies.

5.1 Test database

A database that contains 260 artificial anomaly cases has been constructed and all the anomalies are manually created. In reality, an anomaly event may last for several hours, costing many manual efforts and much waiting time to create. Moreover, some anomalous events are not easy to observe, for example, snow only occurs in the winter and leaves only fall in the autumn. As the designed sampling rate is once every minute, a real anomaly event of five hours may cover 300 measured values. The anomalies in the database are created with a modified sampling rate of 60 Hz; therefore, a five-minute simulated case in the database can represent a five-hour real case. In the database, each base anomaly test case has been extended to create more anomaly cases by shifting, scaling and transformation.

5.2 Anomaly detection evaluation

The anomaly detection algorithm is evaluated in terms of Type I errors and Type II errors that are usually used in statistics.

A hypotheses is an assertion which can be proven false under a certain set of observed data and the null hypothesis is typically a general or default assertion. A Type I error is counted when a true null hypothesis is incorrectly rejected, while a Type II error is counted when a false null hypothesis is not rejected.

In this work, a default null hypothesis is defined when no anomaly is occurring. Therefore, a Type II error occurs when an anomaly is actually taking place and it is not detected, because at that time the null hypothesis is false and the algorithm fails to reject it. Type I errors, which are also false positives, occur when there are no anomalies, but the algorithm returns anomalies. If there were no anomalies and the algorithm did not generate any, then it was called true negative; if there were anomalies and they were detected, then it was called true positive.

Tabularised relations between truth and falseness of the null hypothesis and the outcome of the test is shown at 5.1.

Table 5.1: Error types

	null hypothesis is true no anomaly in fact	null hypothesis is false there are anomalies in fact
anomaly detected	Type I error, false positive	Correct outcome, true positive
no anomaly detected	Correct outcome, true negative	Type II error, false negative

5.2.1 Anomalous input

Figure 5.1 shows a time series that contains two anomalies obtained from a solar panel deployed outdoors. The variation on the residual time series is obvious after the first round of anomaly detection, while the residuals become more smooth after ten rounds.

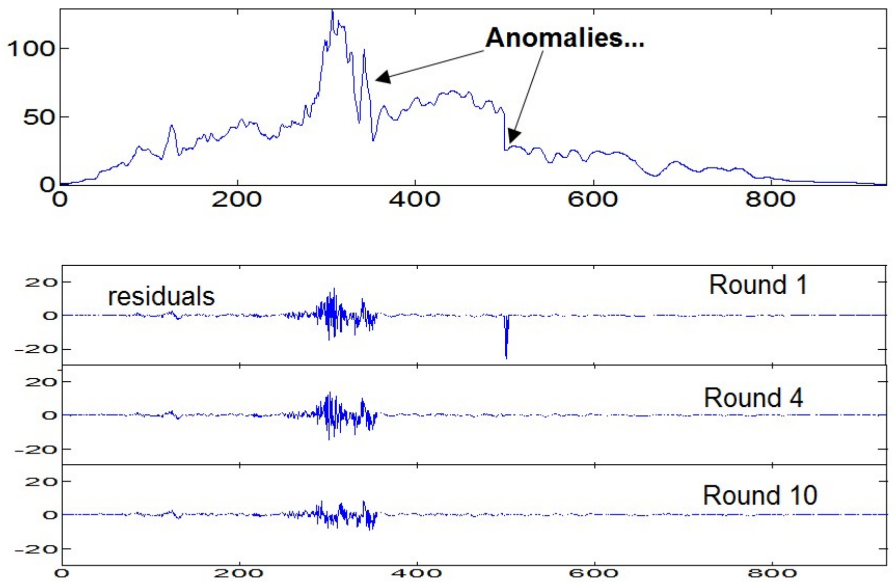

Figure 5.1: Anomalous residuals

The anomaly detection algorithm is evaluated by the time series input which contains anomalies from the database. A true positive is counted if the algorithm successfully detects the anomaly; otherwise, a false negative is counted. The anomaly contained in the time series is randomly selected among ten types of anomalies and the anomaly time is uniformly distributed between the beginning and the end of the time series.

Table 5.2 shows the evaluation results.

Table 5.2: Anomalous input detection

threshold	1 round	5 rounds	10 rounds
0.1	65.3 / 1.5	71.0 / 3.8	84.8 / 2.5
0.5	61.9 / 1.2	73.7 / 2.4	92.3 / 2.8
1.5	58.1 / 2.7	85.2 / 2.1	95.4 / 3.1
3	47.3 / 0.9	84.1 / 1.8	89.3 / 2.8
5	35.2 / 1.2	79.9 / 1.0	91.2 / 2.1

Every entry in the Table 5.2 shows a pair of values in the form of a/b, where a denotes the probability in percentage that the algorithm successfully detects the existence of the anomaly and b denotes the number of wrong detections. "Threshold" in the table represents the threshold value of β in the detection algorithm while "round" denotes the number of maximum rounds run in the iterative anomaly detection algorithm. There is a tradeoff between a and b because seen from the results, b increases as a increases most of times. When the threshold decreases, the amount of anomalies detected by the algorithm also tends to decrease; however, the amount of wrong detections, as seen from the value of b decreases as well.

5.2.2 Non-anomalous input

Given a non-anomalous input time series, if the algorithm detects something, then a false positive (Type I error) is counted, otherwise, a positive negative is counted. Table 5.3 shows the evaluation results of the algorithm with different parameters of "round" and "threshold".

Table 5.3: Non-anomalous input detection

threshold	1 round	5 rounds	10 rounds
0.1	30.1	34.6	35.0
0.5	10.2	14.8	17.1
1.5	4.7	5.6	7.3
3	2.3	4.1	5.6
5	1.9	2.3	2.5

The entries in Table 5.3 denote the probability in percentage that the algorithm returns any anomaly. Since there is no anomaly created in the original time series, the anticipated value in the table is 0, and the less, the better. As seen from the data, the number of rounds does not significantly influence the detection performance. The reason is that the first round of detection result dominates the final detection result, and if the algorithm does not detect any anomaly in the first round, then the algorithm will probably not detect more anomalies in next rounds. When the threshold is set under 0.5, the algorithm is regarded so sensitive to the anomalies that with a probability of more than 10%, it returns an anomaly for an anomaly-free time series.

5.3 Comparison between different classification algorithms

A comparison between different classification algorithms is presented. Different algorithms are evaluated in terms of their training errors and testing errors.

The anomaly database is formed by ten sub-databases, the i-th of which contains all cases with the anomaly type of i. In each sub-database all cases are sequentially numbered starting from 0. The evaluation of each classification algorithm includes ten test rounds, and in each round, about 90 percent cases from the database are chosen as the training data and the remaining 10 percent cases as the testing data. Particularly, in the i-th round, all the cases with a label number of $x\%10 = i$ are chosen as the testing data and the remaining is chosen as the training data. The training errors and test errors are the averages from the ten test rounds.

Referring to the anomaly detection evaluation results above, the "round" parameter is set to 5 and the "threshold" parameter is set to 1.5 to achieve the relatively best detection results for the later anomaly classification stage. The algorithms considered are k-nearest-neighbors ($k = 1, 3, 5$), support vector machines (Gaussian kernel and degree-4 polynomials), linear regressions, neural networks and convolutional neural networks. Table 5.4 shows the evaluation results of all classification algorithms considered.

Table 5.4: Anomaly classification

algorithms	training error (%)	test error (%)
k-nearest neighbors ($k = 1$)	2.1	2.7
k-nearest neighbors ($k = 3$)	5.7	4.1
k-nearest neighbors ($k = 5$)	6.0	5.8
support vector machine (Guassian)	3.2	15.3
support vector machine (deg-4 poly)	1.6	35.8
linear regression	7.5	22.9
neural network	4.8	17.2
convolutional neural network	5.9	18.3

The best testing correctness ratio obtained is 97.3% from k-nearest-neighbors with $k = 1$. The classification performance decreases slightly when k increases; a reasonable explanation is that the number of cases is relatively small, being less than 1000; therefore, when k increases, the probability that a test case reaches to other types also increases. The support vector machine with a polynomial kernel and the linear regression does not perform well because the dimension of the test cases is large that the polynomial kernel cannot capture the complexity. It is believed that if the number of cases in the database increases, all the classification algorithms will perform better and converge to a correctness ratio of at least 95%. However, as it is not easy to create one case, all the classification algorithms are only considered in terms of their performance when the number of test cases is relatively small.

Chapter 6

Conclusion

A general and extensible framework to detect and classify anomalies is built in this work. Previous researchers focuses on cases of detecting anomalies in theory. Anomalies are unfortunately not obvious to detect in real applications; thus, an extensible approach is proposed to solve the problem. The performance of the algorithm is relatively more robust under different problem configurations. The framework can be generalized to objects which can be abstracted in a time series format; the stock price series is a possible real application case to be analyzed. A general anomaly detection framework is summarized as follows,

1. Organize the training data and the test data, then store the training data and the test data numerically
2. Represent the observed values in the time series format.
3. Apply the time series anomaly detection algorithm and append the detected anomaly format data to the end of the original data segment
4. Train the classifier with all the training data and assign each test data with a category label
5. Classify detailed anomalies

6.1 Similar problems

The proposed algorithm can be applied to the handwritten digit recognition problem whose core principle is to obtain the features from the bitmap of digits and then to classify the digit according to the relevance between its feature and the features in the database. The handwritten digit recognition problem has been thoroughly analyzed and an open public test database called MNIST which contains $60,000$ test cases has been published[29]. The digits from the MNIST database are all size-normalized and centered in a fixed-size image. The database is known for its generality and convenience for testing machine learning techniques and pattern recognition methods as the data has already been well preprocessed and formatted.

At least 99 different classification methods, ranging from the simplest linear classifier to the relatively more complex neural network with multiple layers, are posted. Except for the simplest method of one-layer linear classifiers whose test error rate is 12%, most classification methods achieve a test error rate of under 5% and around half of them achieved a test error of under 1.5%. However, to achieve a test correctness rate of 99%, a sufficient amount of training data should be applied to train the classification model. The proposed classification algorithm can be extended to this problem by classifying original digits and their detected anomaly format data together.

6.2 Future work

6.2.1 Multivariate time series analysis

Detection and classification of anomalies for single time series has been described and the research scope can be extended to multivariate time series that are inversely based on univariate time series analysis. Multivariate time series analysis, representing a process where multiple time series are observed simultaneously over time, involves univariate time series as components of vectors. In multivariate time series analysis, two aspects should be considered; first, the properties of individual time series, and second, the relationships and correlations between different time series. A multivariate time series is assembled under the assumption that different time series would interfere with each other both simultaneously and across the time axis. The general equation for multivariate time series analysis is[10]:

$$\Phi(B)Z_t = \Psi(B)X_t,$$

where the vector of Z_t is obtained by assembling observations from multiple streaming data source with each data source contributing to one dimension in the multivariate data set.

6.2.2 Streaming databases

The data obtained from solar panel sensors is currently stored in a format of plain text and in a regular database. Streaming databases that are more complex and offers better performance are planned to store the data. Examples of streaming database include Auroa[3], TelegraphCQ[27][12], PSoup[12][13] and Datacell[30]. As designed for processing constantly changing workloads, streaming databases support data retrieval more efficiently and should be considered when the number of data sensors deployed on solar panels reaches a certain level. The streaming database technology to be applied is intended for data security, data integrity, convenience, and standardization of data management.

6.2.3 CUDA

The compute unified device architecture (CUDA), developed by Nvidia, is designed for parallel computing and graphics processing and it provides a Matlab programming interface to the computing engine in Nvidia graphics processing units (GPU)[1]. The parallel throughput architecture in GPU can be used for executing concurrent float-point number computation, for example, in the Hessian computation of convolutional neural networks. The algorithm efficiency revised by the CUDA technique is expected to increase around 10 times[34].

6.2.4 Autoregressive conditional heteroskedasticity model

Autoregressive conditional heteroskedasticity model (ARCH) is a model proposed in the financial industry[19]. Unlike the ARIMA model, the basic assumption of the ARCH model is that current variance is a function of the actual sizes of the error terms of previous time periods[7]. The ARCH models originate from the financial time series market where the volatility of the financial time series is always changing. Judging from the definition, the ARCH model should cover more complexity than the ARIMA model does.

6.2.5 Seasonal model

Seasonal fluctuation in data is taken under consideration, as well as daily solar voltage observations. Weather conditions are reliant on time of day, as is solar voltage output. The solar voltage output in the time series analysis is seasonal. The model for the seasonal time series is divided into four parts, which are the trend component, the cyclical component, the seasonal component, and an irregular component[26]. The seasonal model is supposed to better model seasonal anomalies.

APPENDICES

Appendix A

Pseudo code running in microprocessor

```
main()
{
    enable interrupt;
    sn = sensor board id;
    set up the internal hardware;

    sync:
    disable interrupt;
    //rcvline==1 when there is data to read in the receiving buffer
    //rcvline==0 otherwise
    rcvline = 0;
    enable interrupt;
    while (!rcvline);
    read from receiving buffer;
    store the received bytes in 'rcvbuf';
    if (rcvbuf[0] == 0xff)
        timer = 0; //timer is a clock counter of 32HZ
    else
    {
        timer = 0;
        while(timer < 65 * HZ) //HZ is defined to be 32
        {
            pause(1250); //guard time before '+++'
            putstring('+++');
            pause(1250); //guard time after '+++'
            //set the parameter of SM as 0
            putstring('ATSM0\r');
            //exit command mode
            putstring('ATCN\r');
        }
        timer = 0;
        while(timer < HZ);
        goto sync;
    }

    pause(1250);
    putstring('+++');
    pause(1250);

    while(timer < 60 * HZ - 1)
    putstring('ATSM2\r'); //set SM=2
    putstring('ATCN\r'); //exit command mode

    //wait for a minute for the next command token
    while(timer <= (HZ * 60) - 2);
    disable interrupt;
    rcvline = 0;
    enable interrupt;

    while (!rcvline);
    if (buf[0] == 0xff && timer < (HZ * 60 + 2))
        synchronization finished;
    else
        goto sync;

    loop:
    getvoltage(); // to measure the solar panel voltage
    measure the battery level;
    determine whether to measure the current according to sn;
    format a string which includes sn, measured voltage value,
    ampere value and battery value;
    while(timer < (60 * HZ) - 2));
    disable interrupt;
    rcvline = 0;
    enable interrupt;

    while(!rcvline && timer <= 60 * HZ + 2)
    //to receive the command token from the base board
    getline();

    if(timer > ((60 * HZ) + 2) || buf[0] != 0xff)
    {
        timer = 0;
        while(timer < 65 * HZ)
        {
            pause(1250);
            putstring('+++');
            pause(1250);
            putstring('ATSM0\r');
            putstring('ATCN\r');
        }
        timer = 0;
        while(timer < 5 * HZ);
        goto sync;
    }
    timer = 0;
    goto loop;
}
```

Appendix B

Matlab pseudo code for jointly anomaly detection

The pseudo code is listed as follows[15]:

```
series = original series;

//Stage 1:
//Initial parameter estimation and anomaly detection
set<anomaly> anomaly_sets
for ( stage_1_iteration = 1; ; stage_1_iteration ++)
{
 //series == original series in the first iteration,
 compute the maximum likelihood estimates of series;
 get the residuals of series;
 //Inner loop of detection for fixed model parameter estimation
 while(true)
 {
  for t = 1:n
  compute tau(t);
  if(max(tau(1:n)) > C)
  {
   there is an anomaly of type tp at time t;
   anomaly_sets.insert( anomaly(tp,t) );
   remove the anomaly effect from the residual;
  }
  else
   break;        //no anomaly
  }
  if count_of_anomalies == 0
  {
   if stage_1_iteration == 1
    stop, the observed series is free from anomaly effects;
   else
   {
    //We obtained anomalies in previous loop,
    //no additional anomaly in current loop
    break; // go to the next stage
   }
  }
  else
  {
   //we get an anomaly now,
   //adjust the series and re-estimate the series again
   series =
   series - anomaly_effects_detected_in_current_round;
  }
 }

 //Stage 2:
 //Joint estimation of anomaly effects and model parameters
m = anomaly_sets.size();
 apply the most recent model parameters upon the original series;
 while(true)
 {
  while(true)
  {
   use multiple regression model
   to estimate the amplitudes w1,w2,...,wm;
   compute tau1, tau2, ..., taum
   if(min(tau(1:m)) < C)
    delete m; m --;
   else
    break;
  }
  //we get m anomalies
  obtain the adjusted series using anomalies 1,2,3,...,m;
  estimate the adjusted series with maximum likelihood;
  if (the distance of standard error from previous estimate is
   less than \epsilon)
   break;
 }

//Stage 3:
//Detection of anomalies based on the final parameter estimates
anomaly_sets.clear();
parameter_model = most recent parameter model estimation;
filter the original series;
for ( stage_1_iteration = 1; ; stage_1_iteration ++)
{
 //series == original series in the first iteration,
 //compute the maximum likelihood estimates of series;
 get the residuals of series;
 //Inner loop of anomaly detection
 //for fixed model parameter estimation
 while(true)
 {
  for t = 1:n
   compute tau(t);
  if(max(tau(1:n)) > C)
  {
   there is an anomaly of type tp at time t;
   anomaly_sets.insert( anomaly(tp,t) );
   remove the anomaly effect from the residual;
  }
  else
```

```
    break;        //no anomaly
   }
   bool skip = false;
   if count_of_anomalies == 0
   {
    if stage_1_iteration == 1
    stop, the observed series is free from anomaly effects;
    else
    {
     //get anomalies in previous loop
     //no additional anomaly in current loop
     skip = true;
     break; // go to the next stage
    }
   }
   else
   {
    //we obtained an anomaly now,
    //so we have to adjust the series
    //and re-estimate the series again.
    series =
    series - anomaly_effects_detected_in_current_round;
   }
   if skip
    break;
}
while(true)
{
   use multiple regression model
   to estimate the amplitudes w1,w2,...,wm;
   compute tau1, tau2, ..., taum;
   if(min(tau(1:m)) < C)
    delete m; m --;
   else
    break;
}
```

Appendix C

Switching between command mode and data mode

To enter the command mode, the microprocessor has to send to the module a string of "+++". Before and after sending the data, the microprocessor must not send any other data to the module. The requirement of the length of the silent period before and after the "+++" string can be configured separately. If no data has been sent to the module since the module enters the command mode for a certain period of time, the module will exit the command mode automatically. If the microprocessor wants to force the module out of the command mode, then the microprocessor can send a string of "ATCN" directly to the module.

Appendix D

Verifying the sleep mode

The current sleep mode of the module can be verified in two ways. First, detect Pin 13 on the Xbee module, which sees low when the module is trying to sleep. 3.3 volts is high while 2.88 volts might be due to a floating level. If a resistor of between $10k$ and $100k$ ohms is attached from this pin to the ground, and the signal drops to near 0 volt then it is a floating level; on the other hand, if it remains high then it is a real output. The second way is to monitor the consumption current. If the module is set up correctly, then the battery feed to the board is between 2 and 3 mA when sleeping and around 60 mA when not sleeping.

References

[1] *Nvidia CUDA Programming Guide Version 1.0.*

[2] *XBee-PRO 900 RF Modules Manual.*

[3] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, August 2003.

[4] European Photovoltaic Industry Association. Market report 2011, 2011.

[5] Nimmo B and Seid Sam. Effects of dust on the performance of thermal and photovoltaic flat plate collectors in saudi arabia: preliminary results. *Proceedings of the 2nd Miami international conference on alternative energy sources*, pages 223–5, 1979.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, April 1986.

[8] G. E. P. Box and G. C. Tiao. Intervention Analysis with Applications to Economic and Environmental Problems. *Journal of the American Statistical Association*, 70(349):70–79, 1975.

[9] George Edward Pelham Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edition, 1994.

[10] George Edward Pelham Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, 4nd edition, 2008.

[11] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[12] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[13] Sirish Chandrasekaran and Michael J. Franklin. Psoup: a system for streaming queries over streaming data. *The Vldb Journal*, 12:140–156, 2003.

[14] Ih Chang, George C. Tiao, and Chung Chen. Estimation of time series parameters in the presence of outliers. *Technometrics*, 30(2):pp. 193–204, 1988.

[15] Chung Chen and Lon-Mu Liu. Joint estimation of model parameters and outlier effects in time series. *Journal of the American Statistical Association*, 88(421):pp. 284–297, 1993.

[16] Alexandra Constantin. Solar power modelling, anomaly detection and short-term forecasting. Technical report.

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[18] Kai-Bo Duan and Sathiya S. Keerthi. Which Is the Best Multiclass SVM Method? An Empirical Study. pages 278–285. 2005.

[19] Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, July 1982.

[20] A. J. Fox. Outliers in time series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(3):pp. 350–363, 1972.

[21] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[22] Steven Hillmer. Monitoring and adjusting forecasts in the presence of additive outliers. *Journal of Forecasting*, 3(2):205–215, 1984.

[23] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, April 1982.

[24] A. Ibrahim. Effect of shadow and dust on the performance of silicon solar cell. *Journal of Basic and Applied Scientific Research*, 2011.

[25] Maria Jesus Sanchez and Daniel Pena. The identification of multiple outliers in arima models. *Communications in Statistics - Theory and Methods*, 32(6):1265–1287, 2003.

[26] Siem Jan Koopman and Kai Ming Lee. Seasonality with trend and cycle interactions in unobserved components models. Technical report, 2008.

[27] Sailesh Krishnamurthy, Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Frederick Reiss, and Mehul A. Shah. Telegraphcq: An architectural status report. *IEEE Data Eng. Bull.*, 26(1):11–18, 2003.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[29] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits.

[30] Erietta Liarou and Martin L. Kersten. Datacell: Building a data stream engine on top of a relational database kernel. In *VLDB PhD Workshop*, 2009.

[31] Christopher Malon, Matthew Miller, Harold Christopher Burger, Eric Cosatto, and Hans Peter Graf. Identifying histological elements with convolutional neural networks. *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology CSTST 08*, page 450, 2008.

[32] PVinsights. Pvinsights announces worldwide 2010 top 10 ranking of pv module makers, May 2011.

[33] C. R. Rao and S. K. Mitra. *Generalized Inverse of Matrices and Its Applications.* John Wiley, New York, 1971.

[34] Mihail Sirotenko. Cnn - convolutional neural network class, 2011.

[35] Oyvind Strom. Smart energy apps making the move to zigbee, 2011.

[36] Ruey S. Tsay. Time series model specification in the presence of outliers. *Journal of the American Statistical Association*, 81(393):132–141, March 1986.

[37] Ruey S. Tsay. Outliers, level shifts, and variance changes in time series. *Journal of Forecasting*, 7(1):1–20, 1988.