SunDown: Model-driven Per-Panel Solar Anomaly Detection for Residential Arrays

Submitted March 2020, in partial fulfillment of
the conditions for the award of the degree of **MS in Mechanical Engineering.**

Menghong Feng
UMass ID: 27536282

Prashant Shenoy, Dragoljub (Beka) Kosanovic, James F. Manwell

College of Engineering - Mechanical and Industrial Engineering
University of Massachusetts Amherst

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date _____ / _____ / _____

Abstract

There has been significant growth in both utility-scale and residential-scale solar installations in recent years, driven by rapid technology improvements and falling prices. Unlike utility-scale solar farms that are professionally managed and maintained, smaller residential-scale installations often lack sensing and instrumentation for performance monitoring and fault detection. As a result, faults may go undetected for long periods of time, resulting in generation and revenue losses for the homeowner. In this paper, we present SunDown, a sensorless approach designed to detect per-panel faults in residential solar arrays. SunDown does not require any new sensors for its fault detection and instead uses a model-driven approach that leverages correlations between the power produced by adjacent panels to detect deviations from expected behavior. SunDown can handle concurrent faults in multiple panels and perform anomaly classification to determine probable causes. Using two years of solar generation data from a real home and a manually generated dataset of multiple solar faults, we show that our approach has a MAPE of 2.98% when predicting per-panel output. Our results also show that SunDown is able to detect and classify faults, including from snow cover, leaves and debris, and electrical failures with 99.13% accuracy, and can detect multiple concurrent faults with 97.2% accuracy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent technological advances and falling hardware price have led to significant growth in the deployment of renewable solar within the electric grid. The cost of solar deployments have dropped to less than $2.75 per watt in recent years [2] and have become competitive with traditional energy sources. As a result, utility-scale and residential-scale solar deployments have experienced sustained growth across the world, with more than 2.6GW of deployments in 2019 Q3 in the US alone [2].

Typically, larger utility-scale solar farms are professionally monitored and maintained for optimal performance—they are instrumented for monitoring real-time generation to identify production issues, and also cleaned frequently to reduce dust or pollen. Researchers have also suggested using drones carrying thermal cameras to identify and locate faults in large solar arrays [6]. However, the majority of solar installations today are small-scale installations, often on residential rooftops, with capacities of less than 10 kW in 2018 [1]. Due to cost reasons, such systems lack sensing and instrumentation that may be present in larger utility-scale solar farms. Further, monitoring of these systems is left to homeowners, who lack the technical expertise for this task. At best, system performance may be monitored at a coarse-grain system-wide basis to determine system-level issues. As a result, it is not uncommon for residential solar arrays to encounter power anomalies or other local faults that go undetected for long periods of time, resulting in a loss of generation and revenue for the owner. While it is possible to add sensors and instrumentation for real-time monitoring, do-

ing so for small-scale installations increases their cost, and is challenging to do for millions of installations that are already operational without such capabilities.

To address these challenges, in this paper, we present *SunDown*, a sensor-less approach for detecting generation faults in small-scale solar arrays on a per-panel basis (the terms fault and anomaly are used interchangeably in this paper). Our approach assumes that per-panel generation information is available from the array—an assumption that holds true for any installation that uses micro-inverters or DC power optimizers—and uses a model-driven approach to detect when the panel output deviates in an anomalous manner from the model-predicted output. Our approach is based on machine learning and can detect physical anomalies such as snow obstructions, leaves, and electric faults at panels. Our approach seeks to identify and alert solar owners of such issues in a timely manner so that they can be rectified to avoid production losses.

In designing, implementing, and evaluating our SunDown system we make the following contributions.

- We present a model-driven approach, based on machine learning, that leverages correlations in the generated output between adjacent panels to predict the expected output of a particular panel and flags anomalies when the model predictions deviate from the expected values. Unlike prior work that has performed system-level fault detection, our approach is designed to perform more fine-grain fault detection at a per-panel level. Further, our approach can handle and detect multiple concurrent faults in the system, a key challenge that has not been addressed by prior work.

- We present a random forest-based classification technique to classify the probable cause of the observed fault. To validate our approach, we construct two labelled datasets of solar anomalies: a two year dataset from a real-home with real snow cover anomalies that we hand label using ground truth information, and a solar anomaly dataset that we construct with a twenty-panel array by injecting synthetic faults such as dust, leaves, and open circuit faults. Since there is a dearth of solar anomaly datasets, we release both datasets and our code as open-source tools to the community.

- We conduct a detailed experimental evaluation of our methods. We show that our approach has a MAPE of 2.98% when predicting per-panel output, which shows the efficacy of using neighboring panels to perform model-driven predictions. Our results also show that SunDown is able to detect and classify faults such as snow cover, leaves, and electrical failures with 99.13% accuracy for single faults and is able to handle concurrent faults in multiple panels with 97.2% accuracy.

Chapter 2

Background

In this section, we present background on residential solar arrays and solar anomaly detection.

2.0.1 Residential Solar Arrays

Our work primarily focuses on residential solar arrays, such as ones often found on residential rooftops. Such installations are typically small-scale installations with capacities of 10kW or less and comprise a few to a few dozen solar panels (see Figure 2.1). Since we are interested in monitoring anomalies and faults at a per-panel level, we assume that the power generation of the array can be monitored at a per panel level.

This is a reasonable assumption in practice since many residential arrays are equipped with micro-inverters (e.g. Enphase micro-inverters [3]) or DC power optimizers [4] on each panel that are designed to track and independently optimize the power generation of each individual panel. Such installations, which are now commonplace, are advantageous since they maximize the total system output even for deployments that span multiple roof surfaces and under partial shading-effects. As shown in Figure 2.1, such systems provide real-time per-panel generation data, which is essential for our approach. Other than knowledge of per-panel output, we do not assume any other sensors or instrumentation on the residential solar installation. Thus, we seek to develop a sensor-less approach for per-panel solar anomaly detection.

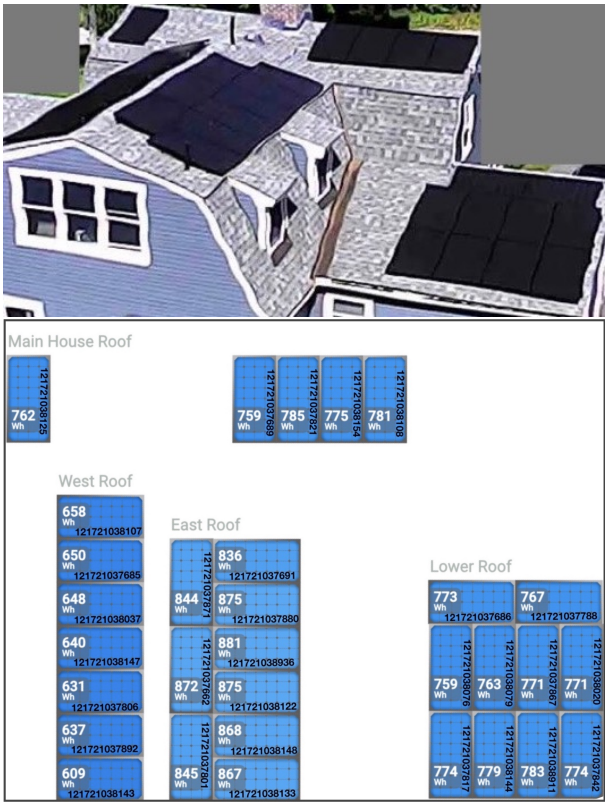


Figure 2.1: A residential solar array (top) with 31 panels deployed on four roof planes, and real-time panel-level generation data from the array (bottom)

2.0.2 Solar Generation

It is well-known that solar generation at any site depends directly on the amount of sunlight – solar irradiance – received at that location. The solar irradiance is a function of the latitude and longitude of that location and the season of the year [18]. Of course, the weather—specifically cloud cover—can reduce the solar irradiance at a particular site.

For the purpose of this work, we assume that per-panel solar generation on any given day can be reduced to two factors: *transient*, which comprises of factors that temporarily impact power output, and *faults* which comprise of factors that have a prolonged negative impact on output.

Transient factors include weather conditions such as cloud cover, wet panels caused by rain or dew, as well as site specific factors such as shading caused by nearby trees or other structures. We can classify transient factors into two classes—common or local. Common transient factors are those that impact all panels of a particular site such as overcast condition or rain. Local factors are those that impact a particular panel, or a group of panels, but not

all of the panels at that site. For example, many shading effects may impact a portion of the site, depending on the foliage and the location of the sun.

2.0.3 Solar Faults

Anomalies (also referred to as faults) in our case are defined to be factors that cause a persistent drop in production but can be rectified by the owner of the site. We are particularly interested in the following three types of faults (1) snow cover on one or more panels, (2) partial occlusions such as bird droppings, dust or leaves on a panel, (3) electric faults such as module failure, short circuits or open circuits. These faults cause either a reduction in output or zero output for a particular panel or a subset of panels.

Due to their close proximity to one another, multiple panels in a residential array may experience the same fault—for example, snow may cover multiple adjacent panels (or even the entire system), resulting in concurrent faults. Of course, a site may also suffer a full system outage, which is also a fault but is easier to detect than those that cause partial outages or partial output reduction.

2.0.4 Problem Statement

Consider a solar array with N solar panels. We assume that the panels are mounted on a residential roof and may be mounted on one or multiple roof planes. Note that in the latter case, panels will have different tilts and orientations. We assume that the power generated by each panel can be monitored in real-time and that the weather at the site is also known (e.g. from a weather service). Given such a setup, our problem is to design a technique that monitors the power output of each panel and the entire system, and labels the observed output in each time interval (e.g. a day) as normal or abnormal. Further, our technique should identify specific solar panels in the system that are experiencing faults and also determine possible cause of the fault (e.g. snow, partial occlusion, or electric fault).

Chapter 3

PER-PANEL SOLAR ANOMALY DETECTION

In this section, we describe our model-driven approach for per-panel solar fault detection and how we can build on this approach to perform multiple fault detection. We first describe the basic idea, followed by the details of our models and algorithms.

3.0.1 Basic Idea

Consider a solar installation with N panels. Suppose that k panels are experiencing an anomaly that result in a reduction, or loss, of output from those panels. Initially, let us assume $k = 1$ (only one panel out of N is faulty). Later on, we show how our approach can be extended to handle multiple, concurrent faults where $k > 1$.

Since all N panels are mounted on the same roof in close proximity of each other, it follows that they experience highly correlated weather conditions, and produce similar output. Thus, our "sensorless" approach first constructs a model to predict the expected output of a panel from n neighboring panels ($n \geq 1$). For example, a simple predictor is one that uses the mean output of n neighboring panels to estimate a particular panel's output. Under normal conditions, since adjacent panel outputs are highly correlated, the model prediction will match the observed output of that panel with high accuracy. Note that any n out of the available N panels can be chosen to model the output of a particular panel. A useful

heuristic is to use the “closest” n panels to the one being predicted or to use the n panels on the same roof plane since they will have higher correlations than those on a different roof surface of the same house. In our evaluation, we experimentally evaluate the accuracy of these heuristics and also evaluate the value of n that yields sufficient accuracy.

When a panel experiences an anomaly, however, the model predictions will continue to estimate the "normal case" output of that panel, while the observed output will deviate from this normal case. If the deviation is "large" and persists over an extended period of time, it is indicative of a fault, rather than an error in the model prediction. The cause of the fault can be separately determined by analyzing amount of loss or the power pattern exhibited by the panel. Such a model-driven approach only uses the observed output of panels to detect anomalies—no other instruments or sensors are needed for anomaly detection unlike some other approaches [6].

3.0.2 Model-Based Predictions

Based on the above intuition, we now present two model-driven techniques for predicting the power output of an individual panel using neighboring panels. Our first model is based on linear regression and uses only power output of panels as input parameters to make predictions. Our second model is based an a probabilistic graphical model and half-sibling regression.

Linear Regression-Based Model

Since the power generated by solar panels in close proximity of one another are highly correlated, we can use regression to predict the output of a panel given the observed output of neighboring panels.

Let P_i denote the observed power output of panel i at time instant i. Let us assume we wish to predict the output of panel i using n other panels. Typically we can choose n nearest panels, or n panels on the same roof plane, out of the N total panels on the roof. A linear regression model allows us to estimate the output of desired panel as a linear function of the others:

$$P_i = w_i P_{i1} + w_2 P_{i2} + w_3 P_{i3} + ... + w_n P_{in} + \epsilon_i \tag{3.1}$$

where $X = \{i_1, i_2, ..., i_n\}$ is the set of n panels used to model the output of the i^{th} panel. We can use linear regression to estimate the weight w_i that minimize the error term ϵ_i.
Such an approach yields N distinct regression models, one for each panel in the system, where each model makes prediction using the observed output of n other panels. To determine if a panel has a fault, we compare the model predictions at time t, $P_i(t)$ with the observed value $\hat{P}$. If the difference between the model predictions and observed values is large and persists over a period of time (e.g., a day or multiple days), the approach flags that panel as faulty.

Graphical Model and Half-Sibling Regression

Our second model is based on a recently proposed machine learning technique called half-sibling regression that uses a Bayesian approach to remove the effects of confounding variables [28]. This approach has been used by astronomers to remove noise from measurements of multiple telescopes observing the same phenomena. The main intuition behind the approach can be understood from the astronomy use-case. Suppose that $n+1$ telescopes are observing the same object such as star. The observations will have some "common" noise introduced by factors such as air pollution or haze that impact visibility of the object. Furthermore, each telescope will have local factors such as instrument calibration error that introduce additional local errors. If we use observations of n telescopes to estimate the expected observation of the $(n+1)$-st instrument, and take the difference between the observed and predicted values, we are left with the local errors ("anomalies") at that instrument. In our case, we have $n+1$ solar panels "observing" the sun—their power output represent their observations of the sun. All panels see common factors such as clouds that introduce similar output reductions in the power values. Further, each panel has local factors such as shade (transient factors) or faults that can result in additional reductions in the power output. If we use n panels to predict the output of the $n+1$-st panel using a Bayesian

	SolarClique	SunDown
Per-Panel faults	No	Yes
System-wide faults	Yes	Yes
Multiple faults	No	Yes
Anomalies Detected	System-wide electrical	Snow, electrical, occlusion

Table 3.1: A comparison of the state-of-the-art SolarClique system and our SunDown approach

model, the difference between the predictions and observed output should isolate local factors including the effect of faults. This is the intuition behind using the Bayesian approach of [28].

More recently, this approach was used in a system called SolarClique[17] to predict the output of an entire array using nearby solar arrays. We draw inspiration from the half-sibling regression paper [28] and SolarClique [17] for SunDown's anomaly detection, but point out important differences between the SolarClique method and our approach as shown in table 3.1. First, SolarClique is designed for system-level predictions (predicting the total generation of an entire array) and does not have the capability of making fine-grain per-panel predictions, which is the focus of our method. Second, a key technical limitation of SolarClique is that it assumes a single fault can occur at a time, and that the system is not capable of scenarios where multiple arrays are faulty. This is a reasonable assumption for SolarClique since it uses n arrays from n different homes to predict the output of a specific home, and faults across arrays and homes can be assumed to occur independently. In our case, since panels are in close proximity to one another, the same fault (e.g., snow) can impact multiple panels, and faults therefore no longer occur independently. Since the independence assumption of SolarClique does not hold in our case, a key technical improvement over prior work is our ability to handle multiple faults (as discussed in the next section). For simplicity, we first assume a single fault in the entire system and present our approach. We then relax the assumption in the next section and show how the basic model can be extended to handle multiple concurrent faults. A final difference is that SolarClique did not focus on fault classification (and only detects large system-level electrical failures) while SunDown can identify multiple types of faults, including snow cover, occlusion faults and electrical faults.

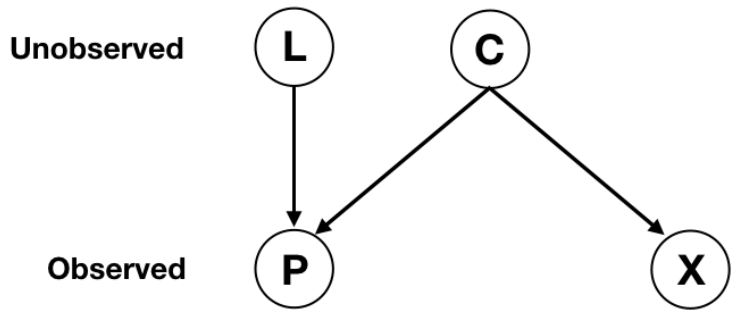


Figure 3.1: Graphical model representation

To describe our Bayesian model, let P be a random variable denoting the power output of a particular panel. Let X denote a random variable representing the power output of n other panels in the system. Hence, X is a vector of size n. Let C denote the confounding variables that impact both X and P. In our case, C denotes common confounding variables such as cloud cover that have the "same" impact on panels. Let L denote the local factors that impact the output of a panel. L will include transient factors, including partial shading, as anomalies that locally impact P. The relationship between P, X, L, and C can be captured using a (causal) graphical model as shown in Figure 3.1. Since the output of each panel can be directly monitored, P and X are observed variables, while C and L are latent unobserved variables.

As can be seen, P depends on both L and C while X depends only on C (and is independent of L). C impacts X, and when conditioned on P, P becomes a "collider", making X and L dependent. To reconstruct L using half-sibling regression, we assume the following additive model

$$P = L + f(C) \tag{3.2}$$

Since C is unobserved, we can use X (which is observed) to approximate $f(C)$. If X exactly approximate the function $f(C)$, we can then compute $f(C)$ on $E[f(C)|X]$. Even otherwise, if X is a sufficiently large vector, it can yield a ground approximation. Thus, we can use X to predict P and recover L from Equation 3.2 as

$$\hat{L} = P - E[P|X] \tag{3.3}$$

Note that $\hat{L}$ estimates both anomalies and transient factors, and the impact of transient factors must be removed from L to estimate the anomaly.

Given these concepts, our algorithm to estimate the amount of production loss due to anomalies is as follows:

We first use regression to estimate P using X. This is similar to the linear regression method from the prior section. The regression yields $E[P|X]$ - an estimate of P given the observed output of n neighboring panels that constitute X. Since P itself is observed, subtracting $E[P|X]$ from P yields an estimate of the output loss $\hat{L}$ due to transient factor and anomalies as shown in Equation 3.3. A key difference between linear regression model of section 3.0.2 and here is that we use bootstrapping to construct multiple regression model by subsampling the data (instead of a single regression model) and use an ensemble method based on Random Forest that uses the mean of multiple models to estimate $E[P|X]$.

Next, since $\hat{L}$ contains effects of transient factors such as shade on panels as well anomalies, we must remove the impact of transient factors to obtain the "true" anomalies. We can use time series decomposition to extract the seasonal component that represents shading effect that occur daily at set time periods and remove it from $\hat{L}$ [17]. The remainder of $\hat{L}$ represents production loss at that panel due to any anomalies.

Under normal operation $\hat{L}$ will be close to zero (no anomalies and no loss of output). When $\hat{L}$ is significant and persistent over a period of time, our model-driven approach flags an anomaly in the panel.

3.0.3 Handling Multiple Concurrent Faults

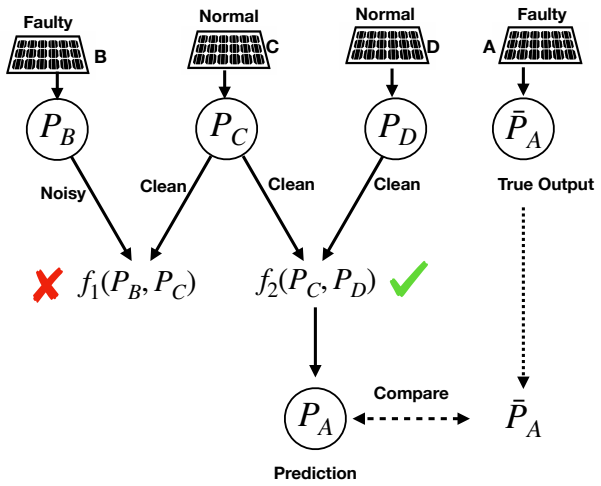


Figure 3.2: A forecasting model is used to ensure non-noisy inputs to our Bayesian model.

Both our regression and Bayesian models use the power output of n panels to predict the expected output of another panel. A very important assumption is that the n panels being used as inputs to the model are non-faulty and hence be used to predict the normal case output of another panel. An anomaly is flagged when the model prediction of normal case output deviates from the observed output, indicating the presence of an anomaly.

This approach works well when there is only one faulty panel in the system - which implicitly implies that all remaining panels are non-faulty and any model that uses some of these remaining panels to make predictions will have “clean” non-faulty inputs. However, due to the close proximity of panels, anomalies such as snow cover, dust, leaves, are likely to impact multiple panels. In this case, some of the inputs to the model may come from faulty panels, causing model prediction to have high errors.

Of course, if n is made large and only a small number of panels are faulty, the model may be able to tolerate the "noise" in a small number of inputs and still produce reasonable accurate prediction. However, many residential rooftops may have a small number of panels, which means n can not always be large. Hence, we need an explicit method to tolerate the impact of multiple concurrent faults in the system.

Observe that our models use *any* n out of N total panels to predict the output of panel i. Thus, it is possible to construct *multiple models for each panel* by choosing different subsets

of n panels out of N, and then using them as inputs to predict the output of panel i. In the normal case (no faults), all of these models show similar predictions for panel i's output. However, when multiple panels are faulty, any model that uses faulty panels as input will have higher errors while a model that uses all non-faulty inputs will continue to provide good predictions. Our goal then is to construct multiple models for each panel using our Bayesian or regression method, and then choose one of these models at each instant that uses non-faulty inputs.

To do so, we need to distinguish between faulty and non-faulty inputs. However, since the models are themselves being used to detect faults, we need a different method to determine which inputs are possibly faulty. To do so, we use a solar forecasting approach that predicts the output of the solar panel based on weather forecasts. There is extensive work on solar forecasting using weather forecasts and any such model can serve our purpose. We use a machine learning forecasting-based model that uses the location of the system (longitude and latitude), time of day, past power observations and near-term weather forecasts (e.g., sunny, cloudy) to estimate the output of a panel [18]. This model, and many others, have been implemented into the Solar-TK open-source library [7], which we leverage to design a custom forecasting model for each panel in the system using near-term future weather forecasts.

Suppose that $P_i(t)$ is the estimate of power output of a panel i based on this forecasting model. If $P_i(t) - \hat{P}_i$ is large, it implies that expected output differs from the prediction and the panel is possibly a "noisy" input. Our per-panel forecasting models perform these prediction for each panels and labels it as "noisy input" or "normal input". Any model that uses one or more noisy panel as an input should be eliminated from consideration for anomaly detection purposes.

That is, SunDown chooses any regression or Bayesian model (out of multiple models for a panel constructed from different subsets comprising n panels) such that all inputs to that model are labelled normal.

Consider the following example to illustrate the process (figure 3.2). Suppose that a solar rooftop install has 4 panels: A, B, C, D. We wish to predict the output of panel A using two

other panels. Suppose both A and B are faulty. Let us assume we have the following two half-sibling regression-based Bayesian models, f_1 and f_2 to predict P_A, the power output of panel A

$$P_A = f_1(P_B, P_C) \tag{3.4}$$

$$P_A = f_2(P_C, P_D) \tag{3.5}$$

where model f_1 predicts A using panels B and C as inputs, while f_2 predicts A using C and D. Our approach first predicts P_A, P_B, P_C, and P_D using per-panel machine learning solar forecasting models for each of the four panels [7]. Since A and B are faulty, they get labeled noisy inputs. Hence, f_1 is eliminated from consideration since one of its inputs, P_B, is a noisy input and f_2 is chosen for prediction since both its inputs, P_C and P_D, are labelled "normal". Using model f_2 yields a better estimate for P_A than model f_2. Note that, doing so enables us to handle concurrent faults–we can avoid using faulty panels as model inputs, and at the same time, use our Bayesian method to identify the presence of multiple faults.
Note that although our solar forecasting models also provide an estimate of the panel's output, they are not suitable for anomaly detection. This is because they use weather forecasts of cloud cover, along with other parameters, to estimate a panel's output. Forecasts of future weather are inherently error-prone, which means the the forecasting model will also have higher errors. Using the solar forecasting model directly for anomaly detection will have higher false positive (due to model errors). In contrast, the Bayesian approach uses actual power output observations to estimate a panel's output for purposes of anomaly detection, which yields a more accurate model and reduces changes of false positives. This is the reason we use forecasting models to only identify noisy inputs; incorrectly labeling a panel as noisy due to forecasting error only causes some of the models to suppressed for anomaly detection, and does not impact accuracy of the remaining models for finding faulty panels.

3.1 CLASSIFYING SOLAR ANOMALIES

While the previous section presented model-driven approaches to detect the presence of anomalies in one or more panels, in this section, we present a classification approach to determine the possible causes of the output loss seen at the panel(s).

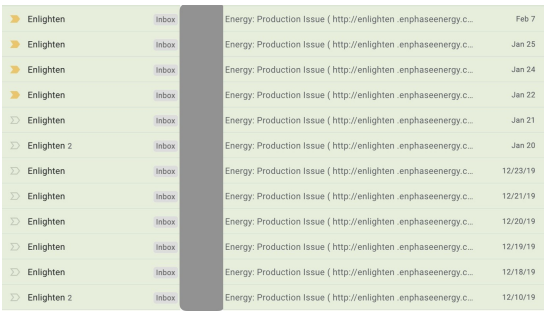


Figure 3.3: Snow event email alert

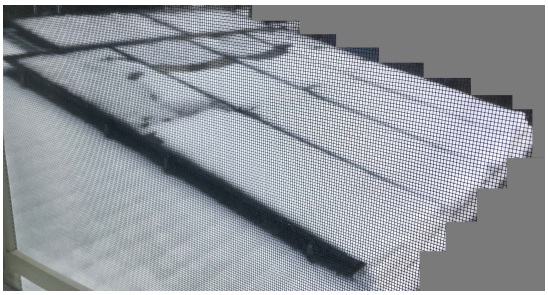


Figure 3.4: Lower roof under snow

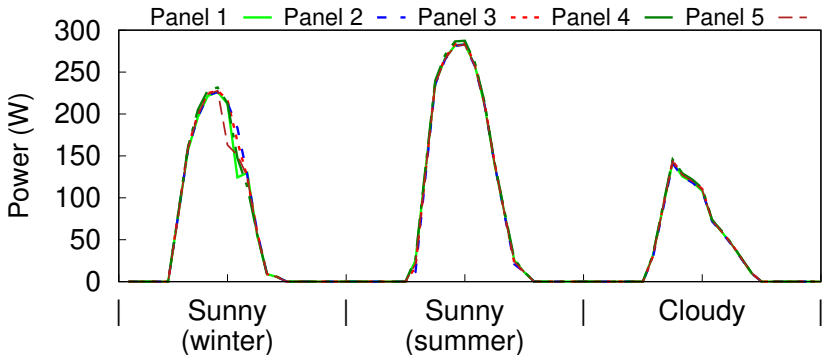
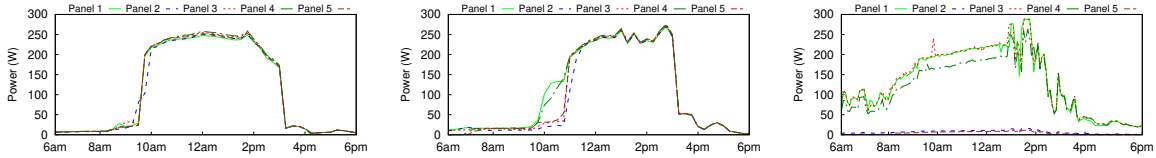


Figure 3.5: Panel output on sunny day in summer and winter and a cloud day.

3.1.1 Solar Anomaly Open Dataset

To assign a possible cause to an observed output loss, we must analyze the observed power pattern and match it to the "power signature" exhibited by different type of solar faults. However, this requires that we have ground truth data for various type of faults, which is challenging since there are no open datasets of solar faults available for research use (solar farm operators likely have such data but have not released it to others). Consequently, we need to gather our own data with ground truth information on solar faults.

Our anomaly dataset contains data from two residential scale solar installations:



(a) Normal (b) Partial Shading (c) Snow Fault

Figure 3.6: Residential home power output on an example day under (a) normal condition, (b) partial shading on some panels on east side, (c) snow covering on some of the panels.

1. a 31-panel, 9kW solar installation (Figure 2.1 top) that experienced multiple snow cover anomalies (Figure 2.1 bottom) over its two year lifetime

2. a 20-panel ground mounted solar installation (Figure 8.1) where we systematically introduce anomalies such as dust, leaves, electrical faults, etc., to mimic real-world faults and measure its impact on the output.

We discuss each dataset in more detail before describing our classification method.

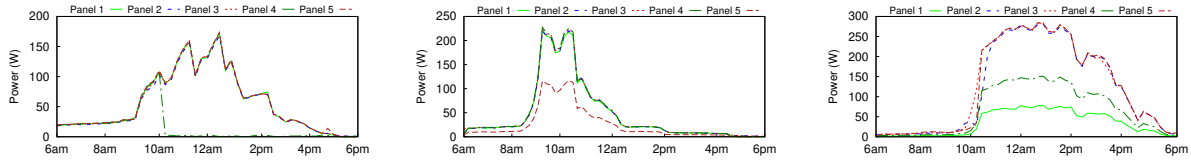
Snow Anomaly Dataset

This dataset comes from a residential solar array deployed on a home in Northern America (location details removed for double blind renewing). The house contains 31 rooftop panels, mounted on four different roof planes, as shown in figure 2.1(bottom). Each panel is a 320W LG panel with an Enphase micro-inverter that can optimize the panel's output independently of the rest. As noted earlier, micro-inverters optimize and report panel-level generation data, which is a prerequisite for our models.

We have been gathering data from this system for over two years and have per panel generation information at 5 minute granularity from September 2017 to February 2020. We have also gathered weather data for the location from Darksky and NOAA weather service.

The only real anomaly encountered by this system over the two year period is snow cover, following a snow storm (the area receives frequent snowfall in the winter). Depending on how long the snow sticks on the panels following a snow event, snow-covered panels may produce little or no output. As snow melts, some panels generate output, while others stay covered with snow (Figure 3.6(c)).

We have two sources of ground truth to label snow faults. First, the Enphase system sends



(a) Synthetic Open Circuit (b) Synthetic Object Covering (c) Synthetic Multiple Faults

Figure 3.7: Synthetic fault injection with (a) open circuit fault, (b) leaves covering fault, (c) multiple leaves covering faults

an email to the homeowner when it observes near zero output for an entire day, as shown in figure 3.3. The email indicates a "possible production" issue at the system. Second, Darksky and NOAA provide past weather data, such as snow events and the extent of the snowfall at an location.

We use both sources of information (which match closely with each other) to manually inspect the per panel generation data on a snow day and the following several days. We then hand label each panel's output as normal (if it produces any output) or as a snow anomaly (if the panel output is near zero). This yields a hand-labelled dataset of snow anomalies.

Solar Anomaly Dataset

Using our 20-panel ground mounted experimental array and sensors to measure its power output, we carefully introduced several types of anomalies onto specific panels, and measured its impact on the power output. We conduct several data gathering experiments over a period of several weeks under different conditions (sunny, partially overcast, overcast etc) and gathered data for the following anomalies.

1. Leaf occlusion: We introduced different number leaves on panels (partial occlusion anomaly) and measured its impact

2. Dust occlusion: We added different amounts of dust on the panels and measured its impact

3. Water drops occlusion: We add varying amount of water drops on the panel and measure its impact. This is designed to mimic morning dew on panels, which is not a true anomaly but a weather effect

4. Open circuit fault: We used a variable potentiometer to introduce a high resistance seen by the panel to mimic an open circuit fault and mesured its impact.

This hand-crafted anomaly dataset, along with photographs and labels, provides an additional source of data for our experiments. For example, Figure **??** shows leaves on the panel that emulate a partial occlusion fault. Figure 3.6(a) and (c) depicts the output of the panels in normal conditions and under a snow fault, respectively. Figure 3.7 (a) and (b) illustrate the power output under synthetically-generated open circuit fault and a partial occlusion fault. We have released both datasets to the reseearch community.

3.1.2 Classifying Anomalies

Given anomalies detected by our Bayesian model we use a random forest classifier to label the possible cause of the fault for each panel that is faulty. The classifier needs to distinguish between three types of faults: snow, partial occlusion and open circuit. Note that partial snow over on a panel and partial occlusion faults both result in diminished, but non-zero output. Full snow cover on a panel and open circuit faults both yield zero output. To distinguish between these cases, we first sample 40 randomly chosen points over an entire day and compute the percentage reduction in power output when compared to the model predictions for each of these points. This power loss vector is a key feature to our classifier. We also use two other features: month of the year and snow depth values from NOAA weather service. We train our random forest classifier using a training dataset of real snow and synthetic anomalies. Depending on the season (winter versus other seasons) and the observed power loss over a period of time, our classifier can label the probable cause of fault for each panel. Our approach can also label system-wide faults, caused either by a system-wide electrical failure or full snow cover on the entire system, both of which cause near total loss of power output.

Chapter 4

EXPERIMENTAL EVALUATION

We evaluate SunDown by quantifying (1) the accuracy of model-based power inference where we infer the output of a single panel using nearby panels, (2) the impact of parameters such as number of panels, roof geometry, and weather, and (3) the accuracy of our anomaly classification. We quantify the accuracy of predicting a panel's output using Mean Absolute Percentage Error (MAPE) between the inferred output and the actual solar generation, as below.

$$MAPE = \frac{1}{m}\sum_{t=1}^{m}\left|\frac{P_O(t) - P_I(t)}{\bar{P_O}}\right| \tag{4.1}$$

where m is the number of samples, $P_O(t)$ is the observed solar power at time t, $P_I(t)$ is the inferred power at time t, and $\bar{P_O}$ is the mean of observed power generation. Equation 4.1 is an alternative form of standard MAPE where we replace the denominator comprising a single observed value by the mean of all observed values. The alternative form avoid divide by zero issues when the denominator (and observed value) are zero.

For the anomaly detection and classification tasks, our goal is to correctly classify all the different anomalies. We use three different metrics to quantify different aspects of the classification task: accuracy, sensitivity, and specificity. The accuracy is computed by dividing the number of correctly classified anomalies by the total number of anomalies. Sensitivity and specificity metrics are used for the unbalanced data case where the number of one category is smaller than other. The different metrics are computed as below.

$$Accuracy = \frac{TP + TN}{N} \tag{4.2}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.3}$$

$$Specificity = \frac{TN}{TN + FP} \tag{4.4}$$

where N is the total number of instances, TP is the number of anomalies correctly classified, TN is the number of normal days correctly classified, FP is the number of normal days classified as anomalies, and FN is the number of anomalies misclassified as normal days. Accuracy is used to evaluate the overall model's performance, while sensitivity and specificity are used to test how accurate the model is to correctly detect the anomalies and normal cases.

4.0.1 Prediction Model Accuracy

We begin by evaluating the accuracy of predicting the power output of an individual panel using neighboring panels.

Machine Learning Model

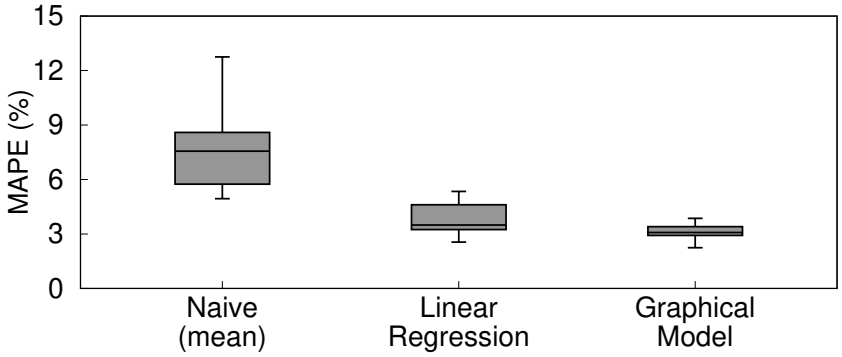


Figure 4.1: Machine Learning Model

To evaluate the accuracy of model inference, we choose a test data only from the days where the site experiences no anomaly. We then use the normal days of the home dataset to train our linear regression and graphical model. We also compare their performance with a naive approach that infers the power output of a panel as the mean output of n other panels. We then compare the model predictions using a test dataset and compute the MAPE values for each approach. As shown in Figure 4.1, the MAPE values for Bayesian model, linear regression, and naive approach are 3%, 4%, and 8.6%, respectively. The naive approach has the worst accuracy since it all panels produce similar output, which is not true in many cases due to panel level variations. Linear regression works well when the output of different panels are highly correlated and have a linear relation between them, which is not true when some of the panels experience partially shading. Our graphical ensemble learning approach is able to model non-linear relationships and yields highest accuracy and a tight confidence interval. We use the graphical model for the subsequent experiments, unless stated otherwise.

Impact of Training Data Size

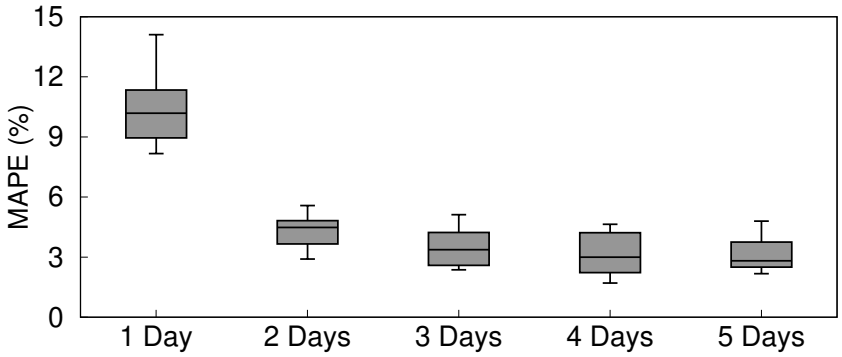


Figure 4.2: Size of training data required

Next, we evaluate model accuracy for different amounts of training data. If a model requires a lot of training data for good accuracy, it can hinder its use for solar sites that have been recently deployed or for the sites where long-term panel level data is not available. We vary the training data size (by randomly choosing a certain number of days) and evaluate its accuracy for predicting output using a test dataset. Figure 4.2 demonstrates that our model can achieve a decent accuracy and a 10% MAPE with only one day of per panel data. If the number of days is increased to 4, the MAPE drops to 3.5% and stays almost constant beyond four days.

Results: Our graphical model can predict per-panel output with 2.98% MAPE and outperforms linear regression and a naive averaging approach. The random forest-based ensemble graphical model does a better job of capturing non-linear relationships among less correlated data than linear regression. While model accuracy increases with training data size, even only four days of training data yield good accuracy.

4.0.2 Impact of parameters

We next investigate various factors that impact the inference accuracy, including number of panels, geometry of the solar deployment and weather.

Impact of Number of Panels

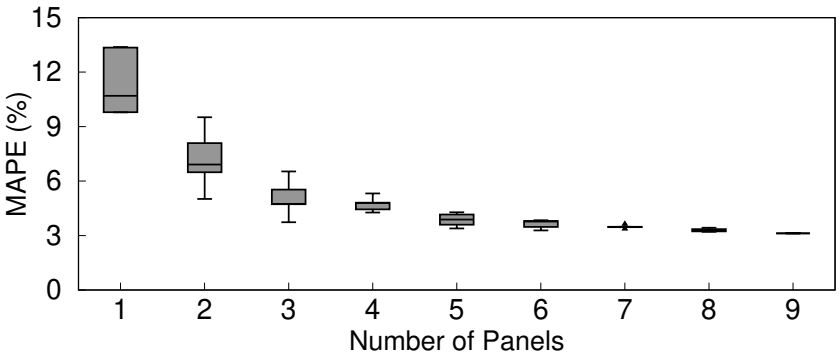


Figure 4.3: Numbers of Panels' Impact

The individual solar panels at a site can demonstrate subtle variations in their solar output, despite their close proximity, due to panel-level dust, different tilt and orientation angles, and panel level physical faults such as cracked glass. To evaluate how many panels are need by a model to provide adequate accuracy, we vary n (the number of panels used by the model as input) and compute MAPE for different n. Figure 4.3 shows inaccuracy is high when using less than 3 panels for inference. The accuracy improves as number of panels is increased to 5 and shows diminishing gains beyond that. The model has an average MAPE value of only 3-4% and a very tight bound, when using 5 panels, as compared to 9% MAPE with single panel. This result suggests that SunDown requires as little as 5 panels to be highly accurate.

Roof Geometry Impact

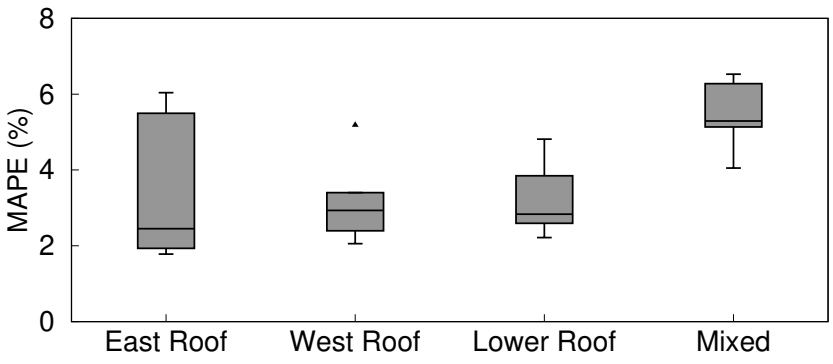


Figure 4.4: Rooftop geometry impact

The output of a solar panel depends upon its tilt and orientation, among other factors [9]. Since a residential array may be installed on multiple roof planes, it is preferable to use panels on the same roof plane to predict others (since they will have similar tilt and oritentation and will exhibit higher correlations).

To evalaute the effect of roof geometry, we split the home dataset into four sub-dataset based on the four roof planes where panels are deployed. We create four graphical models to predict the power output of i_{th} panel by using $n = 7$ panels as inputs. For east roof, west roof, and lower roof cases, all 7 input panels are mounted side by side on the same roof plane facing the same direction. In the forth scenarios, a mixed dataset is created by combined 2 panels from each east roof and west roof datasets, and 3 panels from lower roof dataset. Figure 4.4 illustrates the inference accuracy as the geometry of panels used for inference is varied. For the same roof plane, the model is highly accurate and the MAPE value is between 3% to 3.2%. The large variation for the east roof is due to the partial shading on some of the panels on the roof, leading to inaccurate inferences. The average MAPE of 5.5% for the mixed dataset demonstrates that our model produces a decent accuracy even when input panels are chosen from different roof planes. Thus, when knowledge of the roof geometry is available, it should be exploited, but the model works well even for systems where the roof geometry may be unknown causing the model to use panels from different roof planes for inference.

Impact of Weather

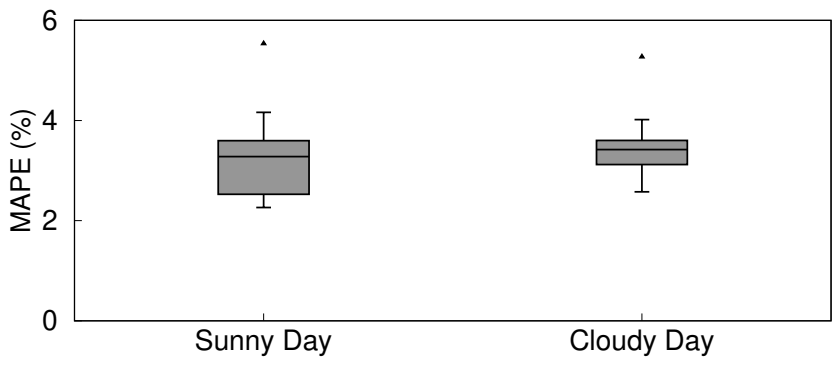


Figure 4.5: Weather impact

The weather at a solar site, primarily cloud cover, impacts the power generation of a site. On a sunny day, all the solar panels produce similar amount of power. However, on a cloud day, scattered clouds may only cover one or few of the panels leading to power variation across panels, which can complicate inference. Figure 4.5 illustrates the effect of weather on the accuracy of the inference task. Our model achieves similar mean accuracy on both sunny and cloudy days, indicating it performs well regardless of weather. The higher variance in MAPE on a sunny day is due to shading from nearby structures, that has a more prominent impact on a sunny day over a cloudy one.

Results: Our experiments show that the number of panels used for prediction as well as the roof geometry play an important role in the model's performance. We find that model yields higher accuracy when five or more panels are used for predictions and when these panels are co-located on the same roof plane. The weather conditions, however, do not impact model accuracy.

4.0.3 Anomaly Classification Accuracy

Snow Fault Detection

The previous section evaluated the accuracy of our model in predicting the output of a panel using nearby panels. We next evaluate the accuracy of model-drives approach and the classifier in detecting anomalies and classifying anomalies, respectively. The common anomalies we consider include snow fault, open circuit, and partial occlusions due to leaves. Although, others factors such as partial shading also results in the loss of energy, we do not consider shade to be an anomaly since it it is a transient phenomena and does not need corrective action.

Our home dataset already includes real snow faults that are labelled and we evaluate the accuracy of our classifier on identifying these snow faults. We then use the synthetic faults from our solar anomaly datatset and synthetically inject them into the home data set by introducing synthetic single panel faults as well as concurrent fault and evaluate the accuracy of our classifier. Figure 3.7 presents per-panel data for a typical day when electric fault or object covering anomaly has been injected into one or many panels.

We first evaluate the ability of our classifier in detecting snow faults in the home dataset (recall that the data set is labelled as normal or snow for each panel). We extract the features from daily power output, which include Pearson's correlation coefficient, ratio of maximum observed power and the nominal panel capacity, and weather data such as snow and cloud cover and use them as inputs to our random forest classifier. Figure 4.6(a) shows the confusion matrix of our classifier and shows high accuracy. Table 4.1 shows that our approach is able identify system-level snow faults an accuracy of 99.13%, sensitivity of 100%, and specificity of 95.12%. We note that snow faults seen in our dataset tends to be system-wide faults, where all panels get covered with snow after a snow event and exhibit a snow fault concurrently. While it is certainly possible for only some panels to have snow cover (e.g., if snow melts unevenly across panels), our dataset presently does not have such faults.

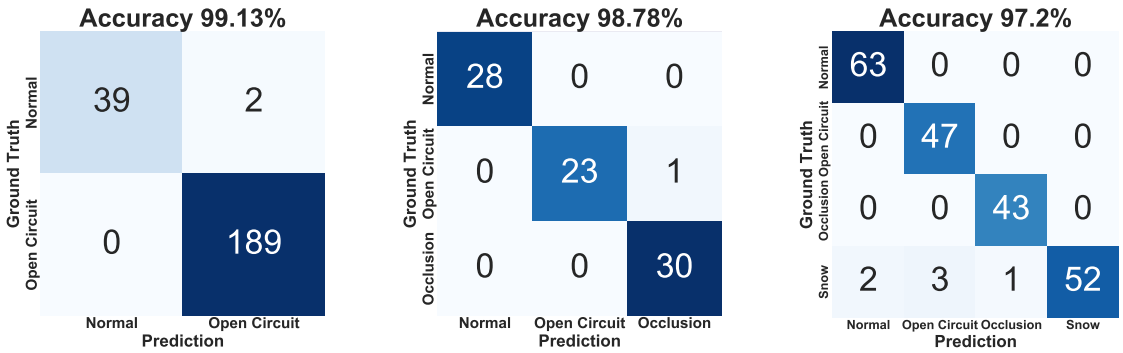
Single and Concurrent Fault Classification

Since all observed snow faults in our dataset were system-faults, we next show that our approach is still capable of fine-grain anomaly detection and classification of a single fault and it is also capable of detecting concurrent faults in a subset of the panels.

To do so, we use our solar anomaly dataset and choose the partial occlusion and open circuit anomaly from the dataset and inject these faults into a single, randomly chosen, panel of the array; different panels have faults injected into them on different days. We use our model to detect the presence of the fault and our random forest classifier to identify the type of fault. We next inject multiple concurrent faults of all types (snow, occlusion, open circuit) into the array using a similar methodology and attempt to detect and classify each fault using our model and classifier (note that we need to use our concurrent fault detection approach in this case).

Figure 4.6b and 4.6c show the confusion matrix of classifying single and concurrent faults in the array. Table 4.1 shows that our model can classify single fault with accuracy of 98.78%, specificity of 97%, and sensitivity of 100%. For multiple concurrent faults, the model obtains accuracy of 97.2%, specificity of 97.06%, and sensitivity of 97.26%.

Results Our experiments demonstrate the efficacy of our fault detection and classification



(a) System level fault (b) Single panel fault (c) Concurrent faults

Figure 4.6: Classification accuracy for (a) system-wide snow faults, (b) single panel faults, (c) multiple panel faults.

Classification	Accuracy	Specificity	Sensitivity
System level	98.13%	95.12%	100%
Single, panel-level	98.78%	97%	100%
Multiple panel-level	97.2%	97.06%	97.26%

Table 4.1: Classification Metrics

methods for real snow faults as well as synthetically injected single and concurrent panel-level faults. Our results show that the random forest classifier is an effective approach for identifying both system-wide faults as well as faults that occur on a subset of panels. Our approach is able to classify snow, partial occlusion and open circuit faults with accuracy of more than 97% in terms of overall accuracy, specificity, and sensitivity.

Chapter 5

Related Work

There has been significant work on predicting power output for solar sites [5, 7, 12, 23, 24, 27, 29]. All of these studies predict only system level output by using long term historical data for model training [27, 29], small amount of historical data for estimating system parameters [7], system configuration details [5, 23, 24], or output from a nearby site [12]. None of the studies predict the individual panel level output, their prediction for all of the panels would be the same. Furthermore, while the anomaly detection and classification is not the key goal, some of these studies can be used to detect panels whose output significantly varies from the system level output. However, a 20-30% error reported by these approaches in system level output prediction will limit their anomaly detection and classification accuracy.

There is also significant prior work on anomaly detection and classification in solar photovoltaic systems, that can be broadly classified into model-based approaches [19, 16, 20, 11, 14] and machine learning based [8, 13, 15, 26, 31, 25, 10, 21, 22, 32] approaches. Model based approaches produce accurate analytical results, but require PV module's specifications and cannot adapt to complex PV systems if the pre-defined parameters change with dynamic environment [21]. Some of the studies use power output data from nearby solar sites [30, 17] to detect and classify anomalies. In [30], authors compare the performance of different solar arrays at the same site, but do not do anomaly classification.

To the best of our knowledge, there is no prior work on classifying panel-level anomalies. All of the aforementioned approaches target system-level anomaly detection and are not

suitable for panel-level anomaly classification tasks. We extend the anomaly detection and classification capability to panel level, where we are able to classify various types of faults, i.e. snow, object covering, and electrical faults, on a single or multiple panels.

Chapter 6

Conclusion

In this paper, we proposed SunDown, a sensorless approach to detecting per-panel anomalies in residential solar arrays. Our approach uses a model-driven approach that leverages correlations between the power produced by adjacent panels to detect deviations from expected behavior. SundDown can handle faults in multiple panels and determine the probable cause of anomalies. We evaluated SunDown using two year panel-level generation data from the from a real site and a manually gathered dataset of various faults. Our approach requires data from only 5 panels for accurate prediction, is agnostic to weather characteristics, and yields high accuracy even when panels from different roof geometries are used. We show that our approach is accurate in predicting panel level output with a MAPE of 2.98% and can correctly classify anomalies with >97% accuracy. We released the per-panel dataset from the real site and the manually generated dataset of various faults for research use.

Chapter 7

Appendix A - Data Access

In this section we will discuss the details for accessing per panel level data from the Enlighten Systems API. The example script is written in Python language and several data analysis packages are required. Please make sure you have installed Jupyter Notebook and packages in your system, as described in introduction section.

7.0.1 Per Panel Level Data

Customers who install the solar panels provided by Enphase Energy and active the service will have the access to per panel level data. Thanks to Prof. David Irwin's permission, we are able to access the data generated from his house roof. There are 31 panels in total as shown in figure **??**. Enphase Energy installs micro inverter device to each of the 31 panels and provide system level and per panel level data in its user interface. To obtain the permission to access the data please contact Prof. David Irwin (irwin@ecs.umass.edu)
You can manually download daily system level data and per panel level data from past 7-day output from user interface. but in order to access the data longer than past 7 days you will have to request data from the server via Enlighten System API. Students are recommended to read API *user manual* for more information. You will need to request API key from Enphase Energy. To obtain the API key, please login into Enlighten account provided by Prof. David Irwin and click *account*. Copy and save the API key from *API Access* section. The current API key is:

```
API Key: 4d54677a4d7a6b794d413d3d0a
```

7.0.2 Request Data from API

As you may realize after reading the API user manual, the Enlighten Systems API does not provide per panel level data. In this subsection we will show an alternative way to to download the per panel level data for longer than past 7 days.

Step 1: Save Login Information as Cookie

First of all, you need to install Firefox browser in your system. We are going to download the user name and password and save it as cookie. Cookie will be used to open the session whenever you are going to access the data from the server. To download the cookie please run the following code:

```
## Import package
from selenium import webdriver

## Open the virtual browser
driver = webdriver.Firefox()

## Open the website
website = "https://enlighten.enphaseenergy.com/systems/1302574"\
"/inverters/28563594/time_series_x?&date=2019-05-04"\
"&stat=POWR%2CDCV%2CDCA%2CACV%2CACHZ%2CTMPI"

driver.get(website)

## Username and password
user="mfeng@umass.edu"
password="UmassCS!"

## Login the website
driver.find_element_by_id('user_email').click()
```

```
driver.find_element_by_id("user_email").send_keys(user)
driver.find_element_by_id('user_password').click()
driver.find_element_by_id("user_password").send_keys(password)
driver.find_element_by_id('submit').click()

## Store the login info in cookie
driver.get(website)
cookie_items = driver.get_cookies()

post = {}

for cookie_item in cookie_items:
    post[cookie_item['name']] = cookie_item['value']

cookie_str = json.dumps(post)
with open('cookie.txt', 'w', encoding='utf-8') as f:
    f.write(cookie_str)
f.close
```

After successfully running this code, you should be able to see a file named "cookie.txt" been created in local folder. To load the login information from the saved "cookie.txt", please run the following code:

```
# Use the stored cookie to login and keep session open
with open('cookie.txt', 'r',encoding='utf-8') as f:
    cookie = f.read()
cookies = json.loads(cookie)
```

Step 2: Download Per Panel Level Data

With the login information saved as cookie, we can access the data from Enphase Energy server and download per panel level data as *json* data file format. Before we proceed this step, we need to relate the micro-inverters' serial number to their ID number. Every panel has its unique serial number and ID number for its micro-inverter. Serial number is a man-

ufacturing sequence number that is physically labeled on each inverter when it comes out from factory. ID number is the registration number on server that is used to identify the device. In order to access the data for specific panel, we need to first link the serial number with its corresponding ID number. These information are provided in table 7.1 below.

With serial number aligned with ID number for each panel, we can use ID number to request daily output data for specific panel defined by its ID number. The website from which you request the data has following formula:

```
system_id = '1302574'
link_1 = 'https://enlighten.enphaseenergy.com/systems/'
link_2 = '/inverters/'
link_3 = '/time_series_x?&date='
link_4 = '&stat=POWR%2CDCV%2CDCA%2CACV%2CACHZ%2CTMPI'

website = link_1 + system_id + link_2 + inverter_id + link_3\
          + date + link_4
```

Where *link 1* is the Enphase Energy server address, *link 2* and *inverter id* define the specific panel you want to access the data from, *link 3* and *date* defines the specific day, *link 4* is the website suffix. The format for *date* has to be *yyyy-mm-dd*, such as *2019-11-13* (year-month-day). After the website is defined, you can request the data from API by running following code:

```
res = requests.get(url=website, cookies=cookies)
data = res.json()
```

Roof Direction	Serial Number	ID Number
(South) Lower Roof	121721037686	28563601
	121721037788	28563607
	121721038076	28563608
	121721038079	28563610
	121721037867	28563611
	121721038020	28563612
	121721037817	28563613
	121721038144	28563614
	121721038911	28563615
	121721037842	28563621
West Roof	121721037871	28563594
	121721037662	28563595
	121721037801	28563596
	121721037691	28563597
	121721037880	28563602
	121721038936	28563603
	121721038122	28563604
	121721038148	28563606
	121721038133	28563622
East Roof	121721038107	28563593
	121721037685	28563598
	121721038037	28563600
	121721038147	28563609
	121721037806	28563618
	121721037892	28563619
	121721038143	28563623
Other Roof	121721038125	28563599
	121721037689	28563605
	121721037821	28563616
	121721038154	28563617
	121721038108	28563620

The data you request has granularity of 5 minutes in ideal case. The date and time for each requested data are represented by both unix and standard format (year-month-day). Each requested data includes the parameters of:

- Date and time
- Power (W)
- DC voltage (V)
- DC current (A)
- AC voltage (V)
- AC frequency (Hz)
- In device temperature ($C°$)
- Unknown parameter

Step 3: Join the Dataset by Timestamp

The downloaded data will be saved as individual file for each panel. We need to join the data of all panels by timestamp. It is recommended to use or convert timestamp format as year-month-day, such as 20190910 (September 10th in 2019). Fortunately the *pandas* library provides very convenient function to join the data, as shown in following code:

```
# Import pandas library
import pandas as pd

# Set timestamp as index
data_1 = data_1.set_index('date_time')
data_2 = data_2.set_index('date_time')

# Join the data 1 and data 2 based on timestamp index
joined_data = data_1.join(data_2, lsuffix='_caller', rsuffix='_other')
```

Step 4: Missing Data and Linear Interpolation

Although the micro-inverter records the data at every 5 minutes, such granularity may be affected by weather condition such as heavy snow and device malfunction. Additionally, each device records the data asynchronously and results in the inability to find existing data

for all panels at certain timestamp. Therefore, students may often face the missing data situation when they try to join the data for all panels based on the timestamp. In this study we use linear interpolation method to fill the missing section where there is no data existed for all panels at that timestamp. In mathematics, linear interpolation is a method of curve

fitting using linear polynomials to construct new data points within the range of a discrete set of known data points. The equation is presented as:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - y_0} \tag{7.1}$$

where x is the timestamp where the data is missing, y is the missing data. (x_0, y_0) and (x_1, y_1) represents two known data points before and after missing data (x, y). Therefore, we can use this equation to find the missing data y. You don't need to program this equation in your code since *pandas* library has already provided such function as

```
data = data.interpolate(method ='linear', limit_direction ='both',\
                        limit = 3)
```

The *data* is in DataFrame format defined by *pandas* library. You can select either *linear* or *polynomial* followed by order parameter to define interpolation method. *limit* defines the maximum missing space. In this case only the missing section equal or less than 3 will be considered to fill the data by linear interpolation. The *limit direction* can be selected as *forward*, *backward*, or *both* to specify the direction where the *limit* will be applied. For any missing section that outside the limit, the data will be filled as NaN and entire row will be dropped out before feeding into the model. To learn more about this function please refer to its *documentation*.

7.0.3 Weather Data

We can request weather data from *Dark Sky API*. Dark Sky is a free API where you can download the hourly parameters including:

- Apparent (feels-like) temperature
- Atmospheric pressure
- Cloud cover
- Dew point
- Humidity
- Liquid precipitation rate
- Moon phase
- Nearest storm distance
- Nearest storm direction
- Ozone
- Precipitation type
- Snowfall
- Sun rise/set
- Temperature
- Text summaries
- UV index
- Wind gust
- Wind speed
- Wind direction

The data above can be used to collaborate as features with per panel level data. You can run the code below to request weather data from Dark Sky API. To download the data, you need to first request the API key and define the longitude and latitude of the location from which you'd like to access the weather data. You can easily find the longitude and latitude information from Google Map. In the *info list* you can define the parameters and extract the features you are interested.

```
# darksky website
darksky = "https://api.darksky.net/forecast/"
# API key
key = "69fcefff98d6c6a0f9b0ebfec1e6f2cc"
# Position: Amherst
lon_lat = "/42.3601, -71.0589,"
# Offset
offset = "?exclude=minutely,flags"

# define parameter you want to collect
info_list = ['time','icon','temperature','humidity','pressure',\
```

```
            'cloudCover','windSpeed','uvIndex','visibility']
info = {para:[] for para in info_list}

# API calling
for day in day_list:

    epoch_time = convert_utc_to_epoch(day)
    link = darksky + key + lon_lat + str(epoch_time) + offset

    res = requests.get(link)
    data = res.json()
```

Please keep in mind that you need to convert timestamp to unix format. *day* variable has the standard format of *yyyy-mm-dd hh:mm:ss* and *epoch time* has the unix time format. For example, by converting 2019-01-29 00:00:00 (January 29th, 2019, at 00 hour 00 minute and 00 second) we obtain unix timestamp of 1548720000. You may find the following functions that are convenient to convert the timestampe:

```
# convert epoch time to YYYYMMDDHHMM format (year-month-day-hour-minute
    )
def date_time_convert(item):

    date_time = time.strftime('%Y-%m-%d %H:%M:%S',
    time.localtime(item)).split(' ')
    date = date_time[0].split('-')
    time_ = date_time[1].split(':')
    date = 10000*int(date[0])+100*int(date[1])+1*int(date[2])
    time_ = 100*int(time_[0])+1*int(time_[1])
    date_time = date*10000+time_

    return date_time

# convert utc time (year-month-day-hour-minute) to unix(epoch) time
def convert_utc_to_epoch(timestamp_string):
    '''Use this function to convert utc to epoch'''
    timestamp = datetime.strptime(timestamp_string,\
```

```
18                                      '%Y-%m-%d %H:%M:%S')
19     epoch = int(calendar.timegm(timestamp.utctimetuple()))
20
21     return epoch
```

Chapter 8

Appendix B - Field Experiment Design

In this section we will discuss the details for implementing field experiment setup. We have designed and built two photovoltaic system setups that are currently maintained by Prof. Prashant's research group. Each setup consists of ten 12V-25W solar panels that are manufactured by ECO-WORTHY. Aluminum frame is designed for outdoors and pitching angle can be adjusted from 10° to 80°. The setup is shown in figure 8.1.

Figure 8.1: Photovoltaic System Setup

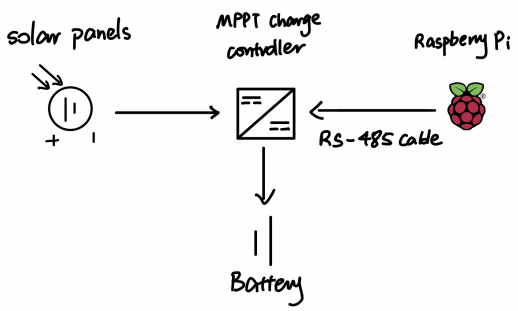


Figure 8.2: Setup Wiring Diagram

Students should decide where is the best location to put setup so that it can harvest maximum amount of energy. The location should be safe and ideally not surrounded by trees that could project shadow on it. It is recommended to put setup toward south so that sun light is able to cover on panels during most of the day time.

After location is decided, students need to connect setup into the system for collecting data, as shown in figure 8.2. The minimum devices required to complete the system include:

- Solar panel setup
- Lead-acid battery
- MPPT charge controller
- Raspberry Pi
- Programmable DC electronic load
- Cables and wires

There are other devices such as programmable relay, sensors, and etc can be also connected into system for extra functions. We will discuss the details to implement these devices in the following sections.

8.0.1 MPPT Charge Controller

MPPT stands for maximum power point tracking. It is a technique to adjust the current and voltage of setup to make sure optimal amount of power can be generated. MPPT charge controller is often connected with solar panel setup to achieve this goal. Additionally, the charge controller could prevent over-charging when battery is full. The controller using in this study is 20A-150V MPPT Charge Controller (2215BN) manufactured by EPEVER, as shown in figure 8.3.

Figure 8.4 is diagram to show how to connect charge controller with other devices. Two critical indicator LED lights are PV and BATT. The PV and BATT LED lights tell the current condition of solar panels and battery. Students are recommended to read the user manual or refer to table 8.1 before use it. Slowly flashing PV LED and solid green BATT LED indicate the connection is normal and setup is currently generating electricity to charge the battery. In order to extract the data from MPPT charge controller, students need to connect Raspberry Pi with charge controller by RS-485 communication cable.

Figure 8.3: MPPT Charge Controller

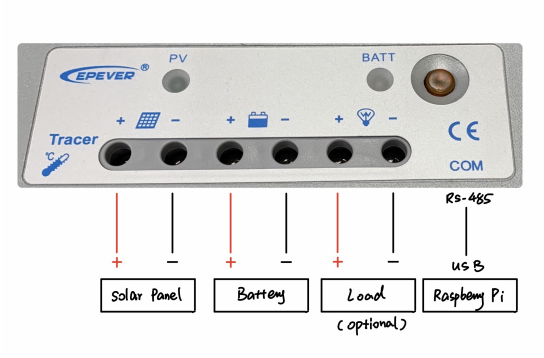


Figure 8.4: Charge Controller Connection

Figure 8.5: RS-485 Communication Cable

Figure 8.6: MT50 Remote Meter

Figure 8.5 is the RS-485 communication cable applicable for standard MODBUS interface. RS-485 cable is used to connected MPPT charge controller with Raspberry Pi to pull out the data. It can also be used to connected with MT50 monitor (figure 8.6) for a quick power check. The MT50 monitor shows the MPPT regulated current (A), voltage (V), power (W), and battery charging condition.

LED Indication	Color	Indicator	Status
PV	Green	On Solid	PV connection is normal but low voltage (irradiance), no charging
	Green	Slowly Flashing (1Hz)	In charging
	Green	OFF	No PV voltage (night time) or PV connection is broken
BATT (battery)	Green	On Solid	Normal, in charging
	Green	Slowly Flashing (1Hz)	Battery is fully charged
	Green	Fast Flashing (4Hz)	Over voltage
	Orange	On Solid	Under voltage
	Red	On Solid	Over discharge
	Red	Slowly Flashing (1Hz)	Battery Overheating

Table 8.1: LED Indication

8.0.2 Raspberry Pi Setup

In this study we use Raspberry Pi 3 model B to collect and store the data. Raspberry Pi is a single-board computer that has been widely used in research and industry. Students will need a SD card with a minimum of 8GB to install operation system. For more information please refer this *link* for the detail instruction to setup Raspberry Pi. After Raspberry Pi is setup, we need to install Pymodbus package to communicate with charge controller. The process has been well documented in this *link*. Students are highly recommended to follow the instruction from this website.

With Raspberry Pi been setup and Pymodbus packaged been installed, download this Github *repository* in Raspberry Pi. In order to establish the communication between two devices, you need to compile and install the common usb serial driver module. Open the Raspberry Pi's terminal and navigate to the Github repository you just downloaded. Compile and install the module by running following code:

```
Installation
------------
* Compile and install the common usb serial driver module

   # make
   # insmod ./xr_usb_serial_common.ko

* Alternativley install via DKMS

   # cp -a ../xr_usb_serial_common-1a /usr/src/
   # dkms add -m xr_usb_serial_common -v 1a
   # dkms build -m xr_usb_serial_common -v 1a
   # dkms install -m xr_usb_serial_common -v 1a

* Ensure that thecdc-acm module is not loaded (assuming that
it is not needed)

   # echo blacklist cdc-acm > /etc/modprobe.d/blacklist-cdc-acm.conf
   # update-initramfs -u
```

After usb serial driver module been compiled and installed, connect Raspberry Pi to MPPT charge controller by RS-485 cable. The ethernet head connects to charge controller and USB head connects to Raspberry Pi. If the module is successfully compiled and installed, you should find a device called ***ttyXRUSB[0-3]*** by running following code:

```
# To check that the USB UART is detected by the system
    # lsusb
    # ls /dev/tty*
```

It is observed that for sometimes you may need to remove the CDC-AMC driver first and then reinstall it at the every time when you boost device. Otherwise you wouldn't be able to see ***ttyXRUSB[0-3]*** been created when you connect Raspberry Pi with charge controller. To

do this manually, you need to run the following code:

```
* To remove the CDC-ACM driver and install the driver:
    # rmmod cdc-acm
    # modprobe -r usbserial
    # modprobe usbserial
    # insmod ./xr_usb_serial_common.ko
```

It is recommended to automatically launch this script whenever you boost Raspberry Pi so that you don't need to do this manually every time. You can open the modules file in Raspberry Pi by running following code in terminal, and paste above script in there:

```
# sudo nano /etc/modules
```

8.0.3 Access Data from MPPT Charge Controller

The info.py file located in Github repository has all the code you need to extract the data read from charger controller. In this subsection we will present an example to show the minimum code required for reading current, voltage, and power data from MPPT charge controller and saving them in local SD card.

We assume you have managed to complete the Raspberry Pi setup, install the *Pymodbus* package, and establish the communication between Raspberry Pi and MPPT charge controller, the minimum code you need to read the data from charge controller is presented below. If you'd like to explore more function that *Pymodbus* could offer, we recommend you to read *LS-B series protocol* as well as *Pymodbus documentation*.

```
# Import Pymodbus package
from pymodbus.client.sync import ModbusSerialClient as ModbusClient
import time

client = ModbusClient(method = 'rtu', port='/dev/ttyXRUSB0',\
```

```
                        baudrate=115200)
client.connect()

result = client.read_input_registers(0x3100, 6, unit=1)
solar_voltage = float(result.registers[0]/100.0)
solar_current = float(result.registers[1]/100.0)
solar_power = float(result.registers[2]/100.0)
record_time = time.asctime(time.localtime(time.time()))

# Read data at every 1s
time.sleep(1)
client.close()
```

The above code will return the power (W), DC voltage (V), and DC current (A) read from MPPT charge controller. You can build a loop function to continuously request data from charge controller. Also you can modify the sleep time to adjust the access frequency and create data structure to store the data.

8.0.4 Program a Relay to Simulate Wiring Defect

In field experiment sometimes we want to inject wiring defects, such as open circuit, contact resistance, short circuit, and etc. The wiring defects can be simulated by altering the path in circuit where current flows through. For example, figure 8.7 shows an example where relay is series connected with solar panel. By switching the relay we can break the circuit so that no current flows through to simulate open circuit defect. In this study we use a 4-channel relay module (figure 8.8) to inject wiring defect. Relay module has to be connected to Raspberry Pi through its GPIO pins and can control maximum of 4 circuit paths simultaneously.

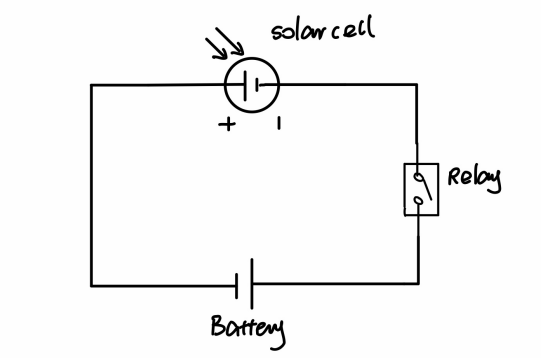


Figure 8.7: Open Circuit Defect

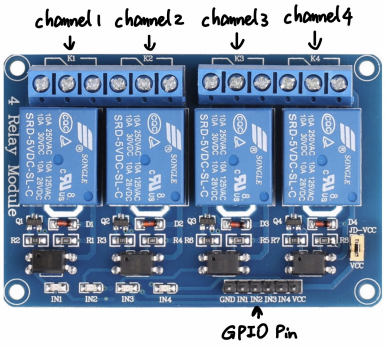


Figure 8.8: 4 Channel Relay Module

You need to program Raspberry Pi to switch relay automatically. You can find a good tutorial in *here*. Refer to table 8.2 when you connect relay module to Raspberry Pi. Figure 8.38 in Appendix section shows a detail GPIO layout for Raspberry Pi Model 3 B+. Please be aware that GPIO layout varies for different Raspberry Pi model. In this study we are using Model 3 B+. The minimum code to program your relay is shown below:

Relay Module Pin	Raspberry Pi Pin
VCC (power)	Pin 2 - 5V Power
IN1 (channel 1)	Pin 3 - GPIO 2
IN2 (channel 2)	Pin 5 - GPIO 3
IN3 (channel 3)	Pin 7 - GPIO 4
IN4 (channel 4)	Pin 11 - GPIO 17
GND (ground)	Pin 6 - Ground

Table 8.2: Serial-ID Number Table

```
# Import GPIO package
import RPi.GPIO as GPIO

# Set GPIO mode
GPIO.setmode(GPIO.BCM)

# The channels of relay connect to Raspberry Pi on its
# GPIO pin number 2, 3, 4, and 17 respectively
```

```
pin_num = [2,3,4,17]

# Initialize relay to make sure all ports are connected
for i in pin_num:
    GPIO.setup(i, GPIO.OUT)
    GPIO.output(i, GPIO.HIGH)
    # Change GPIO to LOW if you wish to disconnect all ports

# GPIO.output(pin number, break condition)
# Set break condition to HIGH to connect circuit
# Set break condition to LOW to break circuit

# Chanel 1 connects
GPIO.output(2, GPIO.HIGH)
# Chanel 2 disconnects
GPIO.output(3, GPIO.LOW)
# Chanel 3 disconnects
GPIO.output(4, GPIO.LOW)
# Chanel 4 disconnects
GPIO.output(17, GPIO.LOW)
```

8.0.5 Battery and Programmable Load

Battery (figure 8.9) is required to connect into MPPT charge controller so that the energy generated by solar panels can be stored in it. Students must be very careful when working with lead acid battery to prevent electric shock. One thing worthy to document in this subsection is the way to discharge the battery when it is full.

Figure 8.9: Lead Acid Battery

Figure 8.10: Programmable DC Load

After battery is fully charged, the BATT light on MPPT charge controller will begin to slowly flash (1Hz). Under such circumstance charge controller will basically "shut down" the solar panels so no energy will be further charged into battery. Also, all the voltage, current, and power read from Raspberry Pi will become or close to 0. In order to resume the experiment you will have to switch a battery or discharge it by plugging into a programmable DC load (figure 8.10). The procedures are listed below:

1. Plug battery to DC load by connecting positive and negative nodes of battery to their corresponding nodes of DC load, respectively
2. Carefully read the specification of your battery and select appropriate current to discharge battery
3. Push the "switch" button to turn on the DC load and make sure the "load" button is in off position
4. Push the "menu (8)" button to set the maximum current and power during the discharging to prevent any potential damage to battery and device
5. Click the "ESC" button and set the discharging current by turning the knob
6. Push the "load" button in on position and discharging will begin

As mentioned in step 2, students must carefully read the specification of the battery to choose appropriate discharging current. The most important parameters they need to pay

attention is the ampere hour (*Ah*). Ampere hour is the amount of energy charged in the battery will allow one ampere of current to flow for one hour. For example, in this study we use the battery with 100Ah. It means the battery is able to supply 100A in one hour. Students can use ampere hour to estimate how long it will take to discharge the battery by following the equation below:

$$T \times I = Ah \tag{8.1}$$

where T is the time in hour that will take to fully discharge the battery, I is the discharging current in Ampere that you set in DC load, and Ah is ampere hour parameter of the battery. Please be aware that it is not recommended to fully discharge the battery.

8.0.6 Field Experiment Dataset

In field experiment various defects are manually induced to generate corresponding dataset. The goal for collecting the dataset under defects is to observe how power output would be changed and affected under those conditions. The feature patterns extracted from observation are helpful to label the per panel level data as well as to generate synthetic dataset for classification experiment. In this study, the conditions that will be induced to create dataset includes: partial shading, leaf covering, dust covering, open circuit, and contact resistance. In each of the experiment there are two solar panel setups used to report data. The defective condition will be induced on one of the setup while the other is used as ground truth.

Partial Shading

Partial shading is a phenomenon where the shadow is projected on part of the solar panels. It has significant impact to decrease overall power level even when only one panel is covered by shadow. In this experiment, two solar panel setups are put side by side to capture the moment when one setup is covered by shadow but the other is not. The experiment is taken place between 11:00 am to 12:10 am. Figure 8.12 to figure 8.15 show four different stages

during the experiment. Figure 8.11 is the output of two setups. Panel A and panel B refer to the right and left setup in the figures, respectively. The cloud cover level at the top axis in figure 8.11 indicates the level of shadow coverage. The higher the level, the more the panel is covered by shadow.

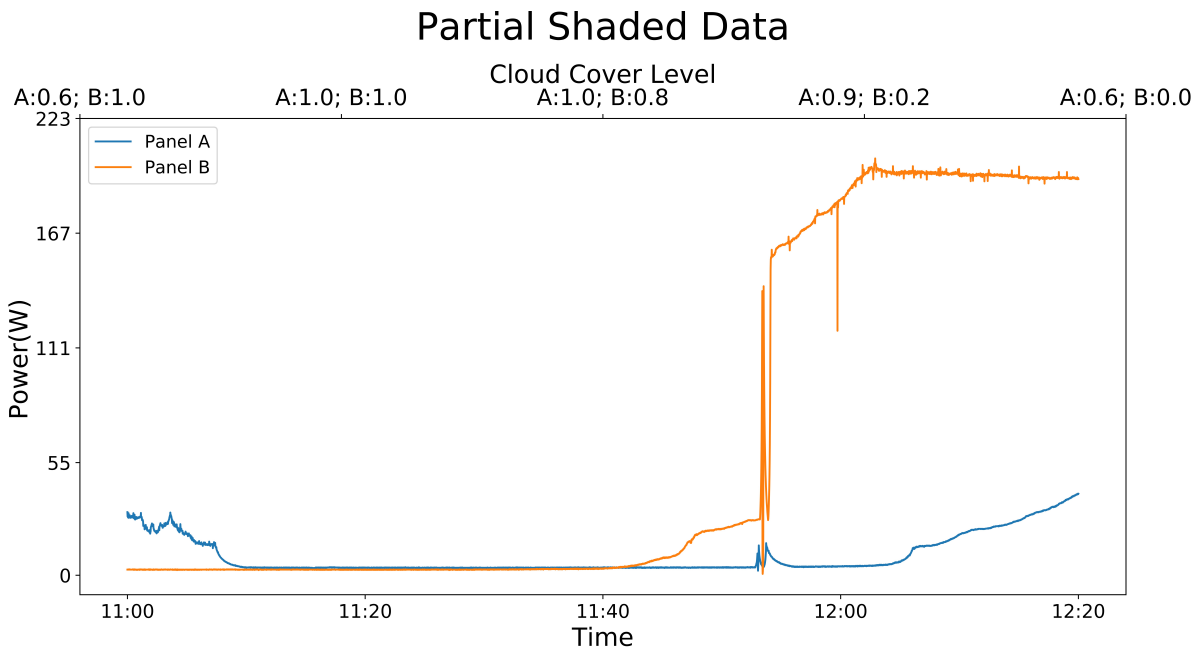


Figure 8.11: Partial Shaded Data

Figure 8.12: 11:00 am

Figure 8.13: 11:20 am

Figure 8.14: 11:50 am

Figure 8.15: 12:10 pm

Initially, panel A is partially covered while panel B is fully covered by shadow between 11:00 am to 11:10 am, as shown in figure 8.12. The power output of panel A is around 25W while panel B does not generate any power. Between 11:10 am to 11:40 am, both panels are fully covered by shadow (figure 8.13). The power output in fully shaded condition is 0 for both panels. Starting from 11:40 am to 12:05 pm, the shadow covering on panel B begins to move away while panel A is still fully covered (figure 8.14). Panel B starts to generate power during in partially shaded condition and continue to increase as shadow disappears, while the power generated by panel A is less than 10 W. Finally, from 12:00 pm to the rest of time, no shadow is projected on panel B and the shadow on panel A begins to decrease, leading to significant power increase in panel B and moderate power increase in panel A.

Overall, partial shading can cause significant energy loss for solar panels. The amount of energy loss is direct proportional to the coverage area. Partial shading is a temporary effect and will disappear eventually. It is not a defect that requires immediate action by owners, but it is recommended to put solar panels in an open area where no shadow can be projected by surrounding objects, such as tree or buildings.

Object Covering

Object covering is a condition in which one or more external objects, such as leaves, bird drops, dust, and etc, are covering on the surface of solar panels. Covering a part of solar panel reduces the amount of solar irradiance received by panel. The level of power reduction depends on the covering object and the period of time it covers. In this field experiment, an

entire solar panel setup is separate into two systems. As shown in figure 8.17, the top five individual solar panels are connected in series as one system, and the bottom five individual solar panels are connected in series as second system. The first system, labeled as panel A in figure 8.16, is manually covered by leaves for a period of time, while the second system labeled as panel B is not covered as a ground truth. Additionally, the level of coverage is enhanced by increasing the number of leaves as shown from figure 8.17 to figure 8.22. The power output is shown in figure 8.16.

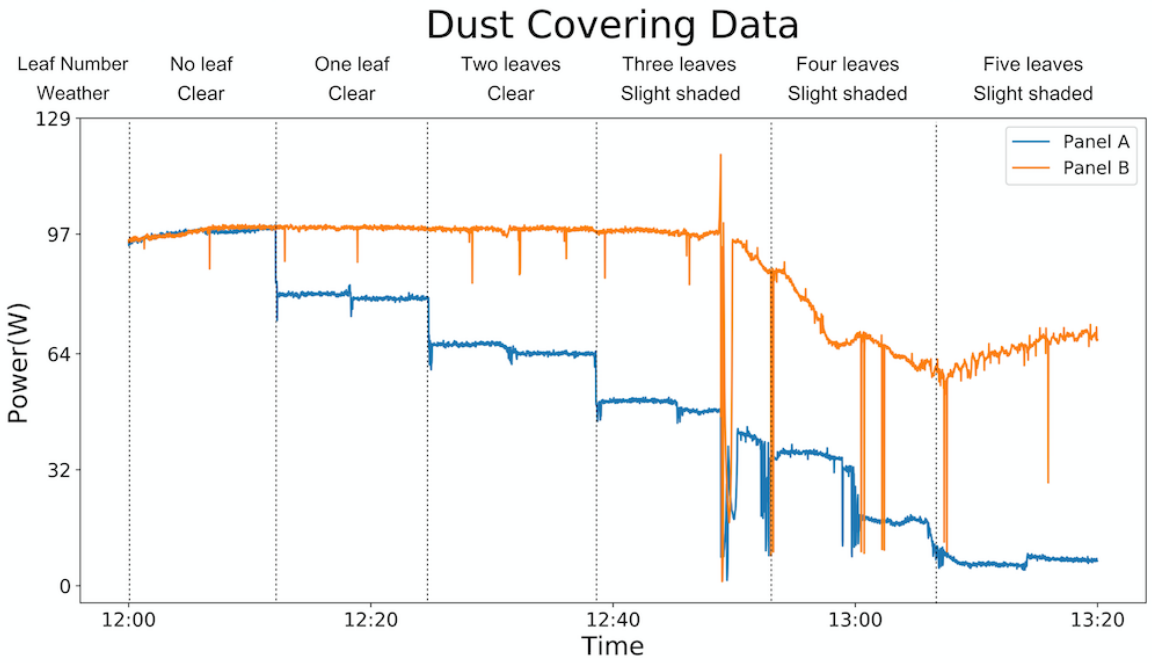


Figure 8.16: Leaves Covering Data

Figure 8.17: Leaf = 0

Figure 8.18: Leaf = 1

Figure 8.19: Leaves = 2

Figure 8.20: Leaves = 3

Figure 8.21: Leaves = 4

Figure 8.22: Leaves = 5

The experiment is taken place between 12:00 to 13:20 on sunny day. As shown in figure 8.16, the bottom x-axis is the timestamp and top x-axis indicates the section where various numbers of leaves are covering on solar panel setup. The entire process is demonstrated from figure 8.17 to figure 8.22. When there is no leaf covered on panels, the power output from panel A and panel B are roughly same. As the number of leaves covered on panel A increases, the panel A's power continues to decrease while panel B's power keeps same. Starting around 12:50 a slight shadow is projected by tree onto the panels and causes the overall power decreasing for panel A and panel B. The experiment result shows that object covering can cause energy loss. The amount of energy loss is direct proportional to the number of covering objects.

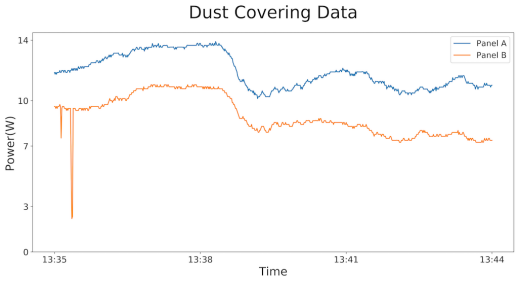


Figure 8.23: Dust Covering Data

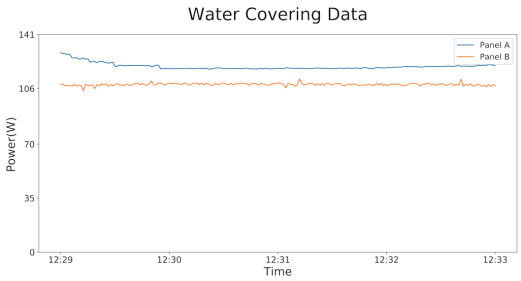


Figure 8.24: Water Covering Data

There are many objects other than leaves could fall onto solar panels and cause energy loss. Those common objects include bird drops, dust, water, and etc. Figure 8.23 and figure 8.24 show the energy loss due to the dust and water coverage on panel B.

Open Circuit and Contact Resistance

The open circuit is a defect that the wire is broken and no current is flowing through the circuit. There are many reasons that could cause open circuit, including worn wire, animal bites, external damage, and etc. It is relatively easy to realize the defect since the solar panel output is 0, but it is hard to identify the exact reason for no power output. For example, zero power output could not only be caused by open circuit, but also could be led by large contact resistance or complete panel coverage by snows or other object. In order to differentiate reasons behind the zero output, more information or parameters are required. It is observed that large contact resistance could complete block the current to flow through the circuit, but it also lead to the temperature increase and may even burn the wire. Therefore the measure of in-device temperature may help to differentiate between open circuit and contact resistance. On the other hand, image or weather parameter can be used to narrow down the reasons between open circuit and snow coverage.

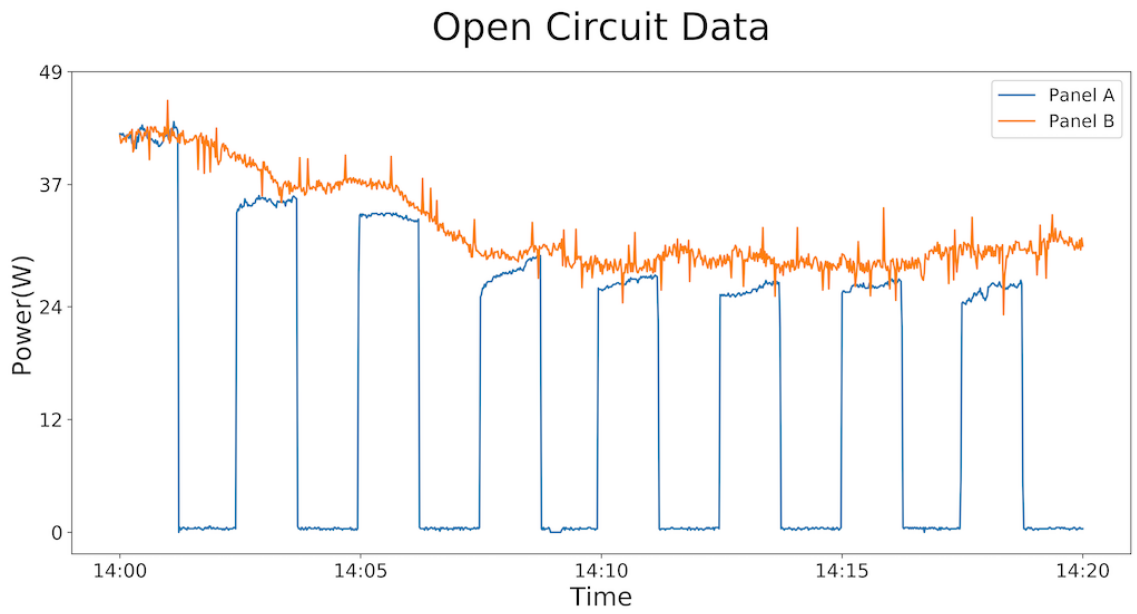


Figure 8.25: Open Circuit Data

Figure 8.25 shows the experiment where the circuit is broken periodically for panel A while keeping panel B as a normal ground truth. Whenever there is the circuit is in open condition, there will be no current flowing through the wire and no energy could be generated by solar

panel.

8.0.7 Synthetic Data

Section 8.0.6 shows the power output pattern under partial shading, object covering, and open circuit condition. The features of output pattern can be used to generate synthetic data. It is expensive to collect the data under certain condition and thus limiting the size of dataset to prevent further analysis, such as anomaly classification. The synthetic data created upon the observed features under above conditions can greatly augment data size.

Partial Shaded Synthetic Data

Partial shading is a temporary phenomenon when the solar panel setup is partially covered by shadow projected by tree or building. The solar panel setup under partial shaded produce much less power than in normal condition, but power output may recover to normal capacity when shadow disappear. Power output under partial shaded condition can be simulated by multiplying original data with shading factor α in sequence as shown in equation 8.2:

$$y_{shaded} = [x_1\alpha_1, x_2\alpha_2, x_3\alpha_3, ..., x_n\alpha_n] \tag{8.2}$$

Where y_{shaded} is the synthetic data under partial shading effect, x_n is the original data at n sequence, and α_n is the shading factor corresponding to x_n. The period during which the partial shading effect applied is randomly induced throughout a day. Shading factor is a random variable drawn from uniform distribution between 0.1 to 0.6. The range of shading factor is based on the observation that partial shading cause significant amount of energy loss. When partial shading effect is not applied, the shading factor α is set to 1 so that the synthetic data is same as normal data.

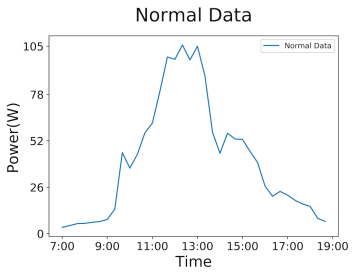


Figure 8.26: Normal Data

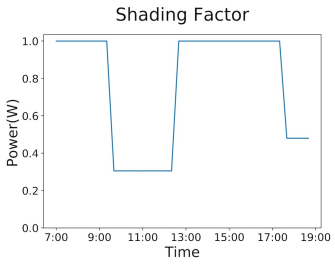


Figure 8.27: Shading Factor

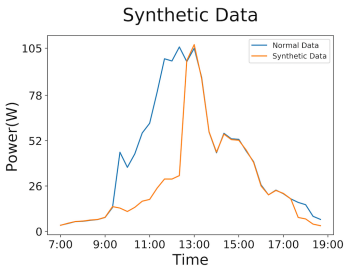


Figure 8.28: Synthetic Data

Above is an example shown how synthetic data is created. Figure 8.26 is a normal data from per-panel level dataset and figure 8.28 is the synthetic data plot. Figure 8.27 is the sequence of shading factor corresponding to each point of normal data. From 7:00 to 9:20 the partial shading effect is not applied and shading factor is set to 1. Between 9:20 to 12:50, partial shading is induced and shading factor is randomly generated as 0.3. From 12:50 to 17:40 the partial shading effect disappears and it comes back at 17:40 with shading factor of 0.5. As shown in figure 8.28, the synthetic data is same as normal data when no partial shading effect is applied between 7:00 to 9:20 and 12:50 to 17:40. However, the power output drops between 9:20 to 12:50 and from 17:40 to the end when partial shading effect is applied.

Object Covering Synthetic Data

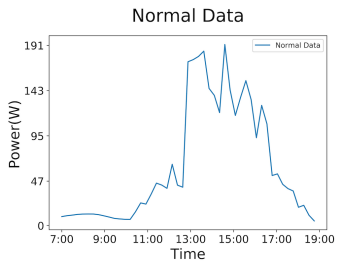


Figure 8.29: Normal Data

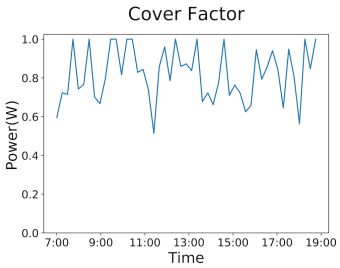


Figure 8.30: Cover Factor

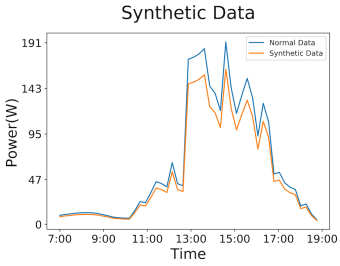


Figure 8.31: Synthetic Data

The observation from section 8.0.6 shows that the solar panel covering by object could lead to the energy loss. The output of affected panel is less than the output of uncovered panel but still follows similar output trend with normal panel throughout the day. Also, the energy loss due to small object covering such as leaf, dust, and water, is less than the energy loss due to partial shading effect. Therefore, we can simulate such effect by equation 8.3:

$$y_{covered} = [x_1 P(\alpha + \beta_1 | \gamma_1), x_2 P(\alpha + \beta_2 | \gamma_2), ..., x_n P(\alpha + \beta_n | \gamma_n)] \tag{8.3}$$

$$P(\alpha + \beta_n | \gamma_n) = \begin{cases} \alpha + \beta_n & \gamma_n = 0 \\ 1 & \gamma_n \neq 0 \end{cases} \tag{8.4}$$

Where $y_{covered}$ is the synthetic data under object covering effect, and x_n is the original data at n sequence. Every synthetic data point is the multiplication of original data point x_n and a conditional probability based on γ_n. As sated in equation 8.4, α is the cover factor, β_n is the noise added with α for each corresponding to x_n, and γ is the reset factor. The cover factor α is randomly generated from a uniform distribution between 0.6 to 0.9. The upper and lower limits of cover factor distribution are higher than those of shading factor introduced in section 8.0.7, due to the observation that partial shading can cause much more energy loss than object covering does. The noise β added with cover factor α is randomly generated from a normal distribution between -0.1 and 0.1 to mimic the dynamic change of covering object. For example, the leaf or dust may be blown away by wind and water or rain may vaporize. Reset factor γ is random integer drawn from 0 to 5. Reset factor is used to simulate the situation when the object is completely removed from solar panel and no energy loss. Figure 8.29 to 8.31 demonstrate this simulation process.

Open Circuit Synthetic Data

When solar panel undergoes open circuit issue, there will be no current flows through the broken path and thus lead to entire or partial energy loss. The amount of energy loss depends on how the setup is wired and the defect location. To simulate the power output under the open circuit condition, equations 8.5 and 8.6 are established to create synthetic data:

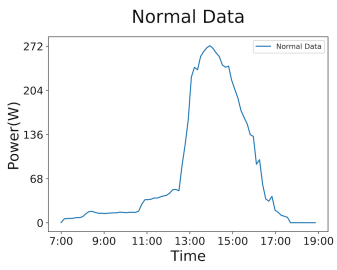


Figure 8.32: Normal Data

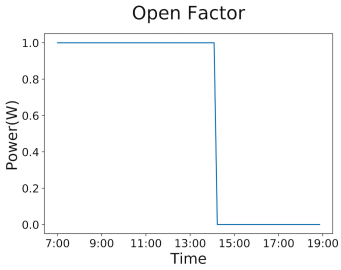


Figure 8.33: Open Factor

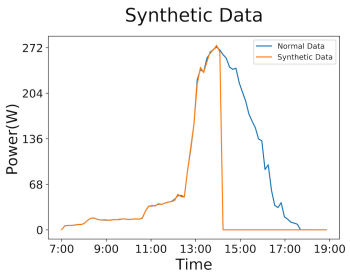


Figure 8.34: Synthetic Data

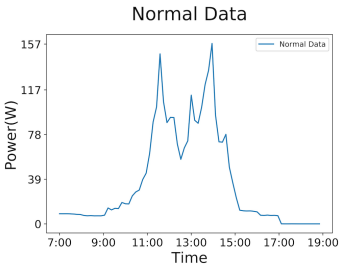


Figure 8.35: Normal Data

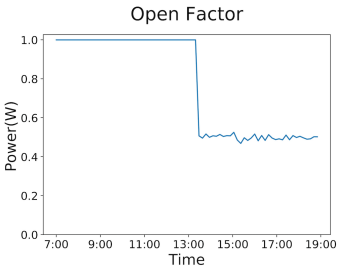


Figure 8.36: Open Factor

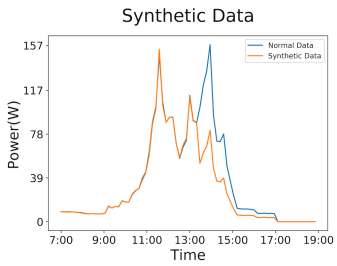


Figure 8.37: Synthetic Data

$$y_{open} = [x_1 P(\alpha_1 | t = 1), x_2 P(\alpha_2 | t = 2), ..., x_n P(\alpha_n | t = n)] \tag{8.5}$$

$$P(\alpha | t) = \begin{cases} 1 & t < T \\ \alpha \in [0, 0.5, 0.33] & t >= T \end{cases} \tag{8.6}$$

Where y_{open} is the synthetic data under open circuit defect, and x_n is the original data at n sequence. Every synthetic data point is the multiplication of original data point x_n and a conditional probability based on time t. As stated in equation 8.6, a random variable T is generated from a uniform distribution ranging between 9:00 to 16:00 throughout a day. When t is greater than T, the circuit is in normal condition and synthetic data is same as original data. When t is equal or less than T, the conditional probability equals to open factor α. To simulate the situation where the wire is broken at subbranch path of parallel-connected circuit, the open factor α is a randomly picked from a list containing 0, 0,5, and 0.33.

The examples above show two cases with different open factor α. As shown from figure 8.32 to figure 8.34, the breaking time T is around 14:20. When $t < T$, the open factor α equals to 1 and synthetic data is same as original data in normal condition. When $t >= T$, α equals to 0 and the power drops to 0. It simulates the situation when the wire is broken at a series-connect circuit or at the main path of parallel circuit. In second example as from figure 8.35 to figure 8.37, the breaking time T is around 13:50. When $t >= T$, the synthetic power output drops to one-half of the power in original data, showing that the one of the path is broken at a two-parallel connected circuit.

8.0.8 PGIO Layout

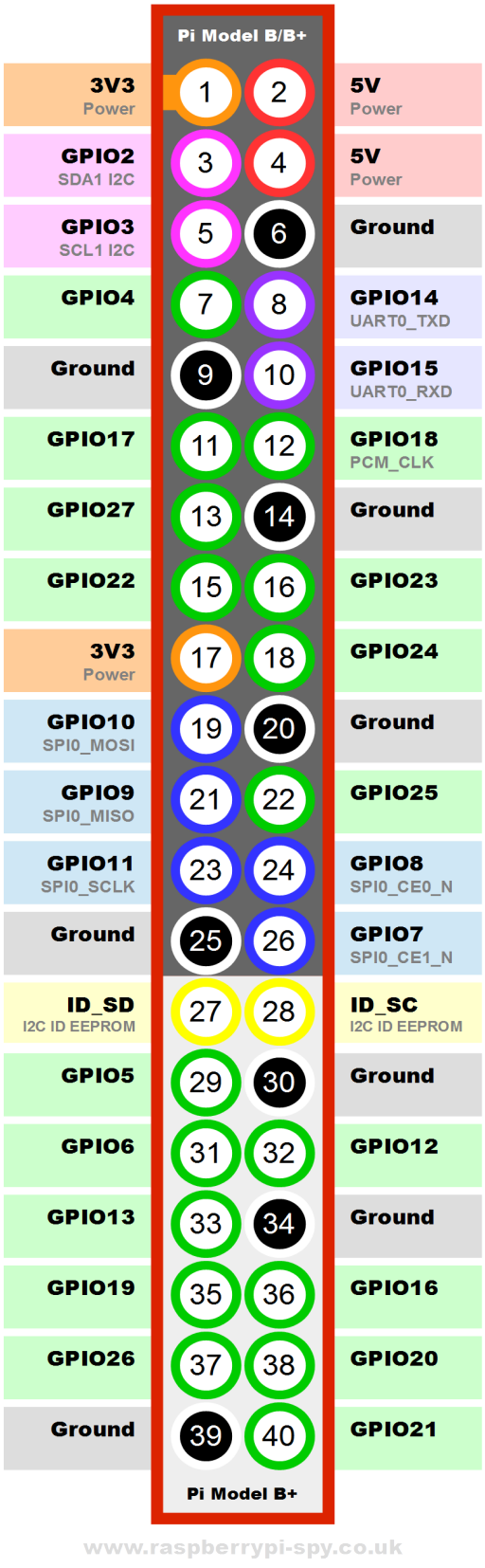


Figure 8.38: Raspberry Pi Model 3 B+ GPIO Layout

Bibliography

[1] 2018. Solar Market Insight Report 2018 Year In Review. https://www.seia.org/research-resources/solar-market-insight-report-2018-year-review. (2018). Accessed March, 2020.

[2] 2019. Solar Market Insight Report 2019 Q4. https://www.seia.org/research-resources/solar-market-insight-report-2019-q4. (2019). Accessed March, 2020.

[3] 2020. Enphase Microinverters. https://enphase.com/en-us/products-and-services/microinverters. (2020).

[4] 2020. SolarEdge Power Optimizer. https://www.solaredge.com/products/power-optimizer. (2020).

[5] R.W. Andrews, J.S. Stein, C. Hansen, and D. Riley. 2014. Introduction to the Open Source PVlib for Python Photovoltaic System Modelling Package. In *IEEE Photovoltaic Specialist Conference*.

[6] A. Arenella, A. Greco, A. Saggese, and M. Vento. 2017. Real Time Fault Detection in Photovoltaic Cells by Cameras on Drones. In *International Conference on Image Analysis and Recognition*. Springer.

[7] N. Bashir, D. Chen, D. Irwin, and P. Shenoy. 2019. Solar-TK: A Data-driven Toolkit for Solar PV Performance Modeling and Forecasting. In *IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*.

[8] M.D. Benedetti, F. Leonardi, F. Messina, C. Santoro, and A. Vasilakos. 2018. Anomaly Detection and Predictive Maintenance for Photovoltaic Systems. *Neurocomputing* (2018).

[9] D. Chen and D. Irwin. 2017. Sundance: Black-box Behind-the-meter Solar Disaggregation. In *ACM International Conference on Future Energy Systems (e-Energy)*.

[10] W. Chine, A. Mellit, V. Lughi, A. Malek, G. Sulligoi, and A.M. Pavan. 2016. A Novel Fault Diagnosis Technique for Photovoltaic Systems Based on Artificial Neural Networks. *Renewable Energy* (2016).

[11] M. Dhimish, V. Holmes, and M. Dales. 2017. Parallel Fault Detection Algorithm for Grid-connected Photovoltaic Plants. *Renewable Energy* (2017).

[12] N.A. Engerer and F.P. Mills. 2014. Kpv: A Clear-sky Index for Photovoltaics. *Solar Energy* 105 (July 2014).

[13] P.X. Gao, L. Golab, and S. Keshav. 2015. What's Wrong with my Solar Panels: a Data-Driven Approach. In *EDBT/ICDT Workshops*.

[14] E. Garoudja, F. Harrou, Y. Sun, K. Kara, A. Chouder, and S. Silvestre. 2017. Statistical Fault Detection in Photovoltaic Systems. *Solar Energy* (2017).

[15] F. Harrou, A. Dairi, B. Taghezouit, and Y. Sun. 2019. An Unsupervised Monitoring Procedure for Detecting Anomalies in Photovoltaic Systems using a One-class Support Vector Machine. *Solar Energy* (2019).

[16] Y. Hu, B. Gao, X. Song, G.Y. Tian, K. Li, and X. He. 2013. Photovoltaic Fault Detection using a Parameter Based Model. *Solar Energy* (2013).

[17] S. Iyengar, S. Lee, D. Sheldon, and P. Shenoy. 2018. Solarclique: Detecting Anomalies in Residential Solar Arrays. In *ACM SIGCAS Conference on Computing and Sustainable Societies*. 1–10.

[18] S. Iyengar, N. Sharma, D. Irwin, P. Shenoy, and K. Ramamritham. 2017. A Cloud-Based Black-Box Solar Predictor for Smart Homes. *ACM Transactions on Cyber-Physical Systems* (2017).

[19] B.K. Kang, S.T. Kim, S.H. Bae, and J.W. Park. 2012. Diagnosis of Output Power Lowering in a PV Array by Using the Kalman-filter Algorithm. *IEEE Transactions on Energy Conversion* (2012).

[20] K.A. Kim, G.S. Seo, B.H. Cho, and P.T. Krein. 2015. Photovoltaic Hot-spot Detection for Solar Panel Substrings using AC Parameter Characterization. *IEEE Transactions on Power Electronics* (2015).

[21] G. Liu and W. Yu. 2017. A Fault Detection and Diagnosis Technique for Solar System Based on Elman Neural Network. In *IEEE Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE.

[22] G. Liu, W. Yu, and L. Zhu. 2018. Condition Classification and Performance of Mismatched Photovoltaic Arrays via a Pre-filtered Elman Neural Network Decision Making Tool. *Solar Energy* (2018).

[23] E. Lorenz, D. Heinemann, H. Wickramarathne, H.G. Beyer, and S. Bofinger. 2007. Forecast of Ensemble Power Production by Grid-connected PV Systems. In *20th European PV Conference*. Milano.

[24] E. Lorenz, J. Hurka, D. Heinemann, and H.G. Beyer. 2009. Irradiance Forecasting for the Power Prediction of Grid-connected Photovoltaic Ssystems. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* (2009).

[25] H. Mekki, A. Mellit, and H. Salhi. 2016. Artificial Neural Network-based Modelling and Fault Detection of Partial Shaded Photovoltaic Modules. *Simulation Modelling Practice and Theory* (2016).

[26] J. Pereira and M. Silveira. 2018. Unsupervised Anomaly Detection in Energy Time Series Data using Variational Recurrent Autoencoders with Attention. In *IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE.

[27] R. Perez, J. Schlemmer, S. Kivalov, J. Dise, P. Keelin, M. Grammatico, T. Hoff, and A. Tuohy. 2018. A New Version of the SUNY Solar Forecast Model: A Scalable Approach

to Site-Specific Model Training. In *2017 IEEE 45th Photovoltaic Specialists Conference (PVSC)*.

[28] B. Schölkopf, D.W. Hogg, D. Wang, D. Foreman-Mackey, D. Janzing, C.J. Simon-Gabriel, and J. Peters. 2016. Modeling Confounding by Half-sibling Regression. *Proceedings of the National Academy of Sciences* (2016).

[29] N. Sharma, P. Sharma, D. Irwin, and P. Shenoy. 2011. Predicting Solar Generation from Weather Forecasts using Machine Learning. In *IEEE International Conference on Smart Grid Communications (SmartGridComm)*.

[30] S. Vergura. 2018. Hypothesis Tests-based Analysis for Anomaly Detection in Photovoltaic Systems in the Absence of Environmental Parameters. *Energies* (2018).

[31] Y. Zhao, Q. Liu, D. Li, D. Kang, Q. Lv, and L. Shang. 2018. Hierarchical Anomaly Detection and Multimodal Classification in Large-Scale Photovoltaic Systems. *IEEE Transactions on Sustainable Energy* (2018).

[32] H. Zhu, L. Lu, J. Yao, S. Dai, and Y. Hu. 2018. Fault Diagnosis Approach for Photovoltaic Arrays Based on Unsupervised Sample Clustering and Probabilistic Neural Network Model. *Solar Energy* (2018).