Ambient Source and Energy Prediction for Energy-Aware Task Scheduling in IoT

by

Mohamad Azzam

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

The deployment of the Internet of things (IoT) is lagging when compared to the forecasted data. This is due to the battery-limited IoT devices. One possible solution is to deploy energy harvesters and use energy management schemes. However, due to the time-varying availability of environmental factors and their effect on harvested power, a structured solution from ambient source and state of charge (SoC) prediction, to the utilization of energy management schemes needs to be presented. In this dissertation, we propose a cost-effective ambient source prediction model, which we then feed into an energy harvesting management unit to predict the batteries' SoC. Lastly, we feed the predicted SoC into our scheduling algorithm to fulfill the application and balance the energy across the IoT network, by distributing the tasks. This solution reduces the deviation of the available energy of the nodes, whilst completing the application and abiding by its quality of service.

Acknowledgments

I initiate this dissertation by acknowledging every individual who contributed to this work in any form.

Firstly, I would like to sincerely thank Professor Mohamed Ibnkahla. He is such a brilliant supervisor who leads and motivates his students to excel in their fields. I would like to thank him for his advice, patience, and work ethic. Most Importantly, I thank him for providing me with this memorable opportunity to join his spectacular team and allowing me to grow both as a researcher and a person. I have learned a lot from him.

Secondly, I would like to express my gratitude to my Co-Supervisor Dr. Zied Bouida, who closely guided me in every step of my work, from organization and weekly meetings to setting up short-term and long-term milestones. His assistance throughout this journey made it feel seamless.

Thirdly, I truly thank my colleagues in the Internet of Things Lab. Your kindness and insight got me through the rough times. Also, a special thanks to Abdallah Jarwan who, without hesitation allocated his valuable time to assist me in both my personal and research problems.

Lastly, I am genuinely grateful to my parents and my sister Nadine Azzam. They were always there to provide me with the needed moral support and unconditional love.

Table of Contents

List of Tables

List of Figures

Nomenclature

Notation	Description
AC	Alternating current
ACF	Autocorrelation function
ADC	Analog-to-digital converter
ADF	Augmented Dicky-Fuller
BMS	Battery management system
CPM	Critical path method
DAG	Direct acyclic graph
DC	Direct current
DVFS	Dynamic voltage and frequency scaling
EF	Early finish
EH	Energy harvesters
EATA	Energy aware task allocation
EHATA	Energy harvesting aware task allocation
EHMU	Energy harvesting management unit
EHU	Energy harvesting unit
ES	Early start
ETCA	Energy task and critical aware
EWMA	Exponential weighted moving average
IC	Integrated circuitry
IoT	Internet of things
LDR	Light-dependent resistor

LF	Late finish
Li-Ion	Lithium-ion
LS	Late start
MAD	Mean absolute deviation
MAE	Mean absolute error
MASE	Mean absolute scaled error
MILP	Mixed-integer linear programming
MOSFET	Metal-oxide semiconductor field effect transistor
MPPT	Maximum power point tracking
P&O	Perturbation and observation
PACF	Partial autocorrelation function
PV	Photo-voltaic
PWM	Pulse width modulation
QoS	Quality of service
RMSE	Root mean squared error
RPI	Raspberry Pi
SAR	Seasonal autoregressive
SARIMA	Seasonal autoregressive integrated moving average
SMA	Seasonal moving average
SoC	State of charge
STC	Standard test condition
WSN	Wireless sensor network

Chapter 1: Introduction and Related Works

1.1 Introduction

The Internet of Things (IoT) is a recent information and communication technology paradigm for distributed embedded computing and communication systems [1]. IoT represents an intelligent network infrastructure where many uniquely identifiable things (microprocessors, wireless sensors) are often wirelessly interconnected to perform complex tasks in cooperative manners [2]. These wireless sensor networks (WSNs) are designed to provide autoconfiguration of the nodes and to operate without any human intervention for years. WSNs are easy to deploy and more flexible. This wide acceptance of WSNs originated from its ability to provide real-time data collection from remote areas, given low production costs, regardless of the underlying networking technologies.

Recently, IoT applications tailored towards WSN architecture became of interest for both academic and industrial entities. By enabling easy access and interaction with things, the WSN-based IoT paradigm has been finding applications in many domains, such as smart homes, smart healthcare, intelligent transportation systems, smart cities and smart grid [3]. It is envisioned that the WSN-based IoT will revolutionize the way we live, work, and interact with the physical world. Nevertheless, deploying a cost-effective and fully functioning WSN-based IoT infrastructure in the real world comes with its challenges. One of the most critical challenges is to manage the resources of IoT devices, especially with their limited computational and power resources. Given that most IoT devices are battery-operated, managing the power consumption and optimizing battery usage is of utmost importance as they represent the most valuable source of providing WSNs with energy.

In various WSN applications, IoT nodes powered by batteries have a limited lifetime, leading to finite runtime. This is a major restriction to the execution of the intended applications and a barrier to the success of the business model of such deployments. Replacing the batteries of dead nodes is a tedious process and is not always viable due to the large quantities of objects in the network, locations of the objects (e.g., harsh environments), and the additional cost. As a result, the IoT network may suffer from disintegration, low QoS, high delays, and loss of important information. Thus, sustainable and efficient solutions need to be developed to tackle this problem. In our case, we propose the utilization of energy harvesting units (EHU).

EHUs are emerging as one of the key enabling technologies for field-deployable WSNs and IoT applications [4]. Using energy harvesting (EH) techniques, the deployed sensor nodes can extract energy from renewable sources (e.g., solar, wind, thermal, etc.) and recharge their batteries during operation. EH networks may also allow fixed battery-less operation, making it an important component of a sustainable "near-perpetual" WSN architecture [5]. Although using EHUs sounds like an ultimate solution, it still holds various obstacles of its own such as the time-varying availability of the ambient sources, and the inconsistent power output of energy harvesting management units (EHMUs) due to the inconsistent presence of solar rays and losses of electrical circuitry. In Chapters 2 and 3, we examine the mentioned limitations and integrate them as a part of the overall proposed solution.

Another concern is regarding the energy and acceptable QoS (e.g., scheduling length, task dependencies, and criticality) associated with the activities executed on the sensor nodes. Typical node activities include sensing, processing, and communication. In traditional

WSN applications, the energy consumption from the loads distributed across the units is very modest compared to the consumption of wireless communication, as single-bit transmission requires 1000 times the energy cost of a 32-bit computation in a classical architecture [6]. However, with the recent introduction of IoT and in network processing, WSNs are expected to carry out more sophisticated and computationally expensive jobs. The energy cost associated with such computation is similar to or greater than the communication cost [7], which results in a lower network lifetime if the workload is not efficiently balanced across the nodes.

In light of the above, active resource and task allocation is vital. To this end, in this work, we explore the research area of energy harvesting aware task allocation (EHATA) for a solution. Energy aware task allocation (EATA) plays a crucial role in extending the network lifetime and reducing applications' execution times, by efficiently distributing/balancing the different tasks among sensor nodes. Although task allocation has been widely studied in wired systems, the resulting approaches are insufficient for IoT-based WSNs due to the limited power and computational resources. In addition, the introduction of EHUs has led to the uprise of energy harvesting aware task allocation. This transition from energy aware allocation to energy harvesting aware allocation considers the inconsistent presence of harvesting capability due to the scarce availability of ambient sources. Chapter 4 addresses this problem.

In this dissertation we plan to cover the following:

In Chapter 2, we address the availability of the renewable sources, that influences the state of charge (SoC) of EH-IoT nodes and the proactive EHATA, by proposing a novel prediction model, composed of a time statistical component (SARIMA) and a control

theory component (Kalman filter). Supported by a dataset of Canada's Capital, our model predicts Ottawa's future irradiance and temperature values for the next time step i.e., *t+1*. In Sections 2.2 and 2.4, the SARIMA formulation and model are presented, respectively. In Section 2.6 and 2.7, the Kalman filter is presented. Finally, the forecasting results and conclusion are presented in Section 2.8 and 2.9, respectively.

In Chapter 3, we propose and simulate (using SIMULINK) an efficient and optimized model for IoT-based Solar EHMU. This model constitutes of a solar module, DC-DC converter, charge controller, and a battery. With the forecasted irradiance and temperature from Chapter 2 acting as inputs to our model, we can estimate the SoC of the battery in *t+1*. In Sections 3.2 to 3.6, the theory behind each component and the system block that represents it in SIMULINK is discussed. In Section 3.6, we discuss the overall system built in SIMULINK. In Sections 3.7, we present our simulation results which include system efficiency, battery SoC (gained from the forecasted irradiance and temperature values in Chapter 2), and power values for different irradiance inputs. Finally, in Section 3.8, we present our conclusion.

In Chapter 4, we propose an energy, task dependent and critical aware scheduling algorithm (ETCA). This is a form of the mentioned EHATA. With the SoC as inputs from Chapter 3, the algorithm using the feature of task distribution minimizes the scheduling duration of the application, and maximize the available energy of the nodes while, abiding to task dependency, criticality, and energy awareness of the application. In Section 4.2, we present the system model, formulation, and heuristic for our proposed scheduling algorithm. Finally, in Sections 4.3 and 4.4, we present the attained results and conclusion.

In Chapter 5, we present the implementation of the ambient source prediction (Chapter 2), energy prediction (Chapter 3), and ETCA scheduling (Chapter 4). The goal is to demonstrate to the reader that, the previously presented theory can be implemented via readily available IoT sensor nodes, the likes of the Raspberry Pi (RPI). In addition, we present a methodology that can be used to split real world applications to suit our ETCA. The chapter is broken down into Sections 5.1, 5.2, and 5.3, where we discuss the implementation of ambient source prediction, energy prediction and ETCA scheduling, respectively. Each section contains four subsections implementation setup, challenges and solutions, implementation results and a conclusion.

1.2 Motivation

With the noticeable increase in population and consumer demands, large corporations ranging from e-commerce such as Amazon to agriculture (indoor vertical farming) such as Aerofarms, had to expand their operations. These corporations increased their yield by expanding their facilities and employing a larger workforce. However, these measures only tackle part of the solution. In order to meet the demand and increase revenue, automation and effective deployment of the workforce are needed. To achieve these goals, IoT devices not only provide up to date data but also offer data analysis thanks to their increased computational capabilities [7]. However, most previous works focus on reducing transmission costs, at the expense of computation [8][9]. Therefore, research as [10] and [11] does not take into account the renewable energy aspect, battery awareness, and sustainability of IoT devices in their energy saving scheme.

Taking the renewable energy aspect into consideration, our aim in this thesis is first, proposing a prediction model for the ambient source availability. Second, addressing the available energy in the node given the ambient source, and the harvesting circuitry. Lastly, proposing an energy management solution using tasks scheduling.

1.3 Main Contributions

In this thesis, we propose a structured solution to the EH in IoT-based WSNs, by maximizing and balancing the available energy across nodes, while satisfying the application QoS (scheduling length, task dependencies, and criticality). This is done by addressing the solar time variability, managing the energy circuitry (which dictates battery's SoC), and proposing the energy management schemes. The main thesis contributions can be summarized as follows:

- Propose a novel prediction model, composed of a time statistical component (SARIMA) and a control theory component (Kalman filter). Supported by a dataset of Canada's Capital, our model predicts Ottawa's future irradiance and temperature values. We then input these values to the EHMU, to observe the impact on the charging state of the battery, as detailed in Chapter 3. The energy, tasks dependent, and critical aware scheduling (ETCA) scheme is then proposed in Chapter 4 allowing for proactive scheduling measures.
- Modeling the electrical components in SIMULINK, we propose an efficient and optimized model for IoT-based Solar EHMU. The goal is to increase the overall system harvesting efficiency by selecting the most suitable solar array and battery according to their power specifications. With inputs from Chapter 2, we can

forecast the batteries' state of charge (SoC). The predicted SoC is then fed into the ETCA algorithm in order to adjust and schedule the IoT applications, accordingly, as detailed in Chapter 4.

- Propose an algorithm that solves the EH-aware-scheduling problem. The goal is to minimize the scheduling duration and to maximize the available energy of the nodes while fulfilling the QoS targeted the application.

1.4 EHATA Literature Review

The harvested energy from EH-IoT sensors is insufficient to power the IoT nodes constantly without any disturbance. Therefore, task scheduling based energy management schemes are required in EH-IoT systems to guarantee the efficient use of harvested energy. Task scheduling algorithms guarantee optimal utilization of harvested energy to prolong the system lifetime, while fulfilling the assigned tasks in an application. Task scheduling schemes are created to distribute the tasks across IoT-based WSNs, to maximize the systems performance given the limited available energy budget. This means that the objective of task scheduling schemes is to minimize the energy consumption and deadline misses, maximize the harvested energy, and ensure the operation of sensor nodes for extended periods with higher application fulfillment [12]. According to the type of tasks in an application and their dependencies, the tasks are scheduled by the task scheduler. The task scheduler takes numerous parameters into account, while deciding the execution of upcoming tasks on the node [12], as depicted in Figure 1.4-1. As seen in the figure, the scheduler uses energy required (consumption dependent), deadline (criticality and start time dependent), other tasks (task dependent) and predicted energy (forecasted SoC from Chapter 3) as parameters, to decide the distribution of the tasks across the nodes in the

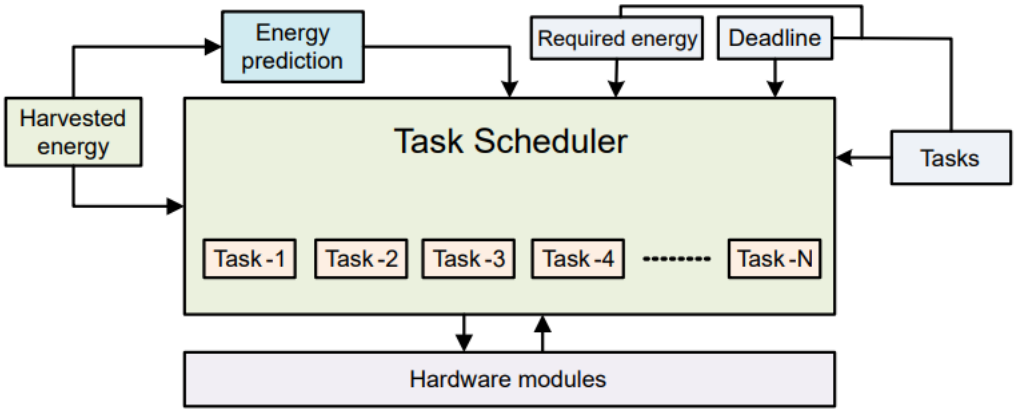


Figure 1.4-1: Scheduler schedule tasks based on energy, other tasks (dependencies) and deadline

system. Tasks with high energy consumption are assigned to the nodes with most anticipated energy and vice versa [12].

Now that we know the goal of task scheduling, the next step is to discuss the methods used to solve it. Various methods were used in the literature to try to tackle the task scheduling problem. Most works were split into two categories low-level task allocation and high-level task allocation. We discuss both in Subsections 4.2.1 and 4.2.2.

In both high and low-level task scheduling most of the solutions, if not all, followed some of the key aspects presented in [12]. They are as follows:

Priority and deadline techniques

- Queuing tasks according to their priority (e.g., high energy consumption tasks first or highly critical tasks) where time delay is of highest priority.
- Executing high priority tasks first, if sufficient energy is present.
- Executing the second task in the queue if insufficient harvested energy is present.
- All high priority tasks are executed first, followed by the low priority tasks if sufficient energy is available.
- Pre-empt the execution of low priority tasks in the presence of a high priority ones.

Predicted energy and deadline

- Scheduling of tasks based on deadline by considering the anticipated harvested energy.

Application breakdown and similarities

- Similar tasks of pipelined applications are executed together to reduce time and energy consumption (e.g., all sensing tasks for different application are carried out together). This lowers the consumed energy by reducing the frequent switching from idle to active state.

1.4.1 Low-level Task Scheduling

Low-level task scheduling algorithms, tend to focus more on reducing energy consumption by varying hardware parameters in the EH-IoT nodes itself, or prolonging the active state in certain portions of the node, to prevent frequent switching. These strategies can be grouped into three categories:

- Dynamic voltage and frequency scaling
- Decomposition and combining of tasks
- Duty cycling

Dynamic voltage and frequency scaling (DVFS) adjusts both voltage and frequency of the EH-IoT node to reduce power consumption. Since the power consumption of the IoT nodes is affected by the input voltage, current flow, and the operating frequency, a scheduling algorithm can be used to solve an optimization problem where the objective is to optimize the power consumption by varying the mentioned parameters. This allows us to minimize

the energy consumption under a performance bound, or maximize the performance given the limited energy budget. In [13], they studied the task scheduling problem in rechargeable systems with battery and deadline constraints. They showed that variable speed processors allowed them to reduce the length of the frame (applications are executed in this time frame) by slowing down the tasks. In [14], the paper introduced a DVFS based scheduling algorithm which depends on the summation of stored energy, predicted harvested energy and deadline of tasks. If the available energy is above a certain threshold, the clock is run at full speed, and if it is below the threshold, the speed is reduced. In [15], the authors used both static and dynamic DVFS based schemes with timing and energy constraints. The algorithm schedules the most demanding tasks when energy overcharge is expected. With this study they were able to reduce the deadline miss rate while using minimum energy storage capacity. Lastly, in [16], the authors model an energy harvesting embedded system with an energy, task and resource model, and present a task scheduling algorithm based on DVFS. The method combines the free dispersed time slots together to harvest maximum energy for running the tasks.

Most works display the effectiveness of DVFS by reducing the task miss ratio while optimizing the energy used. However, DVFS are not always a viable solution since it is hardware restricted. Most IoT sensors are simple in design to avoid energy losses therefore, it does not have the capability to dynamically alter voltage and clock frequency. In addition, various IoT sensors function at a fixed voltage while varying the current. This variability in current is not controlled by DVFS based task scheduling algorithms.

Decomposition and recombining of tasks. As mentioned, EH-IoT nodes have simplistic designs due to size and energy constraints therefore, running such tasks puts a toll on the

sensing node leaving it with little to no power for latter functionalities. One solution to this problem is to minimize the switching of the hardware from idle to active by breaking down highly complex and energy intensive tasks into smaller and simpler subtasks, which requires less energy; then grouping identical tasks together for execution (e.g., the task of transmitting the sensed data is decomposed into two separate subtasks: sensing and data transmission. Likewise, the transmission of saved information in the memory is decomposed into two subtasks: reading the attribute from memory and data transmission). To minimize energy consumption, the two transmission subtasks are combined together by grouping the data from these subtasks and transmitting it in one data packet [12]. Looking through the literature it was evident that research and contribution in this field was lacking. The only source that discussed this low-level task scheduling scheme was [17]. In this work they decomposed larger tasks into subtasks based on their grouping compatibility with other smaller tasks. They summarized the decomposition and recombining technique into the following four phases:

- *Decomposition*: complex tasks are broken down to simpler subtasks by the energy-oriented scheduler. Then similar tasks are grouped to save energy.
- *Combination*: the grouping of subtasks is based on two rules, the elimination of redundant hardware activation (multiple subtasks are performed on the same processor), and allowing concurrent activities on different components (sensing and reading from memory tasks are executed simultaneously, since they utilize different hardware segments of the node).
- *Admission control*: in this phase, filtration of the grouped subtasks takes place with respect to the priority of the task, available harvested energy, and energy

consumption of the tasks. This is because, insufficient harvested energy prevents the combined subtasks to execute all at once. In this work the priority (deadline) is split into soft deadline and hard deadline. Where soft deadline is the earliest the task could finish executing, and hard deadline is the latest the task could finish executing without incurring a penalty.

- *Optimization*: further refinement takes place in this phase where the predicted energy, required number of slots per task execution, and energy consumption of each task are considered. This phase uses predicted harvested energy to prevent deadline violations or future execution of time critical tasks.

The results in [17] show that the proposed decomposition and recombing technique executes more tasks with fewer deadline misses.

Duty cycling is the most common task scheduling technique used in various studies. This scheme minimizes the energy consumption of the energy harvesting IoT nodes by using the sleep/awake mechanism [18]. Whenever a node is not required to execute a task, it is turned to sleep/idle mode. Once a task is assigned, the node is reactivated allowing for task execution. Most traditional duty cycle mechanisms for IoT devices depend on the number of tasks, their corresponding energy consumption, and the residual energy in the battery. However, these protocols are ineffective when it comes to energy harvesting IoT nodes, due to the variability of the ambient source used to recharge it [19]. To account for this variability, the duty cycle-based task scheduling algorithm should consider both, the amount of harvested energy at the current time slot (replenishment), and the next time slot (predicted replenishment). Considering both variables, the algorithm will proactively allow for inactivity (sleep mode) to be scheduled in time slots where the predicted harvesting rate

is high, resulting in a higher recharging rate. It will also command the scheduler to delay energy extensive tasks until sufficient energy is anticipated while maintaining QoS. The authors in [20] propose an adaptive slotting scheme for energy forecasting, where they identify and learn the harvesters energy pattern from the previous time slots, and integrate it into their novel solar forecasting scheme. They were able to minimize the memory footprint while increasing the forecasting accuracy. This work lacks the SoC prediction and the task scheduling algorithm. In [21], the authors propose an accurate method for SoC estimation during runtime, by using the current flow in the battery and the battery voltage as a measure of absolute SoC. They also implement a rule-based tasks scheduler. However, this work lacks the ambient prediction and the novelty in the task scheduling algorithm. In [22], the authors demonstrated an energy balancing method, via a duty cycle process to minimize the energy consumption of sensor nodes equipped with solar harvesters. Although this work achieves energy balancing, and maximizes the throughput of the network, it does not consider task-related computational costs, task dependency constraints, deadline costs, and EH prediction techniques. In [23], the authors implemented a scheduling strategy for task transmission using finite battery capacity for wireless EH sensor nodes. The nodes utilize the harvested energy to transmit data packets while accounting for battery capacity, data deadline, and finishing time of the node. Although the presented algorithm improved data transmission and QoS of the network, it does not account for the criticality and dependency of the transmitted packets. Also, it does not propose any prediction techniques nor EH-awareness.

In [24], the authors demonstrated a global energy management scheme that uses a joint duty cycle approach and transmission power control. The goal is to maximize the number

of packets being transmitted while respecting the variability of the harvested energy. The method successfully increases the efficiency of the system leading to energy balancing. However, the paper does not address the assignment of tasks based on their computational costs and dependencies. In addition, the ambient prediction technique used is the EWMA (exponential weighted moving average). Which is efficient in uncontrolled and unpredictable environments but, it only provides an approximation since it disregards the intensity of light, temperature, and seasonality of weather patterns. Lastly in [25], the authors propose a hybrid energy management approach which uses battery states and operating conditions of awake, sleep and low duty cycle modes. In addition, they utilize a decentralized controller with low computational load to reduce the number of switching. This was the only work that was implemented on a real testbed. Nevertheless, this work lacks the deployment of energy harvesters, their prediction and accurate estimation of SoC.

1.4.2 High-Level Task Scheduling

High-level task scheduling algorithms tend to focus more on reducing energy consumption by studying tasks at an abstract level, where multiple versions of the tasks that make up the application are scheduled across various IoT sensor nodes in the WSNs. The goal is to fulfill an application and a greater objective like energy neutrality (harvested energy is equivalent to consumed energy), deadlines, reliability etc., by assigning multi-versions of the tasks with varying QoS. This method was discussed in [26] where the authors tackle a certain application (monitoring problem) but with different performances, thus execute tasks with different QoS. This variation in QoS is either in terms of quality of the data produced (e.g., tasks may use different transducers with different specifications in terms of

resolution, accuracy etc.), or the quantity of the data (e.g., modulating the sampling rate [27]). An example would be the task of sensing and sending. We can have two versions of this task either send and transmit, or sense, store the sensed data (improve reliability) and then transmit. It is apparent that the second version is more reliable but more energy expensive whereas, the first version is less energy consuming but less reliable. These variations in tasks can include factors like different resolutions, using different transducers, varying frequency and voltage of the processor etc.

In [27], the authors proposed a dynamic programming algorithm for high-level task scheduling in energy harvesting IoT. The study reduces the NP-hard problem to a knapsack problem and solves it using dynamic programming. The goal is to maximize the utility function by executing tasks with the highest quality values. In [28], the authors propose a Mixed Integer Linear Programming (MILP) task scheduling approach to maximize the rewards whilst considering the available energy. However, the works [26-28] differ from our problem as they schedule the tasks on a single node and disregard the criticality and dependency of the tasks. In addition, to the best of our knowledge, the previous works in the literature do not take into account the environmental factors that affect EH and the effect of EHMU with respect to the charging.

To summarize, each application is made up of tasks, and for each task we have different versions. Each version is made up of different editions of a defined execution. All the different versions fulfill the goal of the task but at varying performances. Therefore, two attributes are assigned for each task edition, the first being energy consumption and the second being the reward or quality attained if this version is executed. The goal of the high-

level task scheduling algorithm is to select the optimum versions by using the help of the mentioned attributes, to complete the application, and fulfill the greater objective.

Chapter 2: Ambient Source Forecasting

2.1 Introduction

Solar EH-IoT-based WSNs operate using the battery SoC gained by solar panels (PV). Thus, the availability of the renewable source (in our case solar irradiance and temperature) affects the functional capabilities and lifetime of the nodes in these WSNs. The option of forecasting these renewable sources will provide us with a valuable time window to allow for proactive actions (in our case scheduling the tasks across different IoT nodes).

In this chapter, the SARIMA formulation and model are presented in Sections 2.2 and 2.4, respectively. In Section 2.6 and 2.7, the Kalman filter is presented. Finally, the forecasting results and conclusion are presented in Section 2.8 and 2.9, respectively.

Both irradiance and temperature randomly vary at fixed periods of time. Such sources are considered as Time Series signals, which are stochastic processes that are time dependent. This provides them with the following underlying features that need to be considered during their study:

- **Trend**: ascending or descending trend in the signal. For example, in summer from morning to afternoon, the irradiance and temperature usually increase with time.
- **Seasonality**: repeating patterns that occur at a specific period. For example, highest increase temperature and irradiance usually occur in the summer months.
- **Residual**: resulting random error after both trend and seasonality are removed from the signal. It has a constant mean and variance.

Given our source is a time series signal, and we need to account for trend, seasonality, and residuals, we now need to choose a prediction technique that can be deployed on resource limited IoT devices. After looking into the literature, we can summarize the available prediction methods as follows:

Stochastic forecasting methods: capture the weight of historical data to predict the future values:

- **Error based moving averages:** estimate the trend-cycle by averaging error values of the time series within a certain previous period.
- **Weighted moving averages:** apply weighting factors to historical data points to account for their relational significance in our prediction. For example, exponential weighted moving averages (EWMA) apply a constantly decreasing weighted factor for the previous values. This results in lower importance for values that are further from the current timestamp.
- **Regression models**: establish a relationship between the forecast variable (irradiance) and a single predictor (time). For example, a simple linear regression model allows for a linear relationship between the forecast variable and the predictor variable.

State space methods: use of observed and unobserved variables that describe the evolution in the state of the underlying system [29]. They are heavily used in control theory. State variables (irradiance and temperature in our case) can be estimated with some of the commonly used iterative filters such as Bayes filter, alpha-beta-gamma filter, and Kalman filter. These filters produce successive one step ahead predictions depending on the past and concurrent observations.

Neural network prediction: utilizes networks made up of numerous neurons, whose connections are opportunistically weighted by coefficients that link the neurons and form the input-output relationships [30]. This is beneficial for realizing intricate relationships that cannot be captured by analytical functions (e.g., weather forecasting).

For our predictions, we need an energy efficient system that can be deployed on low computational nodes. Plus, we need a model that can be easily extended to other energy sources i.e., does not require large datasets for training like neural networks. Therefore, in this work, we present an autoregressive integrated moving average (stochastic) with Kalman filtering (state space) to predict both temperature and irradiance.

2.2 SARIMA Theory and Formulation

In each season of the year, we have certain weather patterns that tend to repeat. For example, June, July, and August tend to be hot months i.e., high temperature and radiation. In addition to monthly patterns, we have daily patterns, as the temperature and irradiance increase and decrease when the sun rises and shines, respectively. Also, irradiance and temperature depend on the previous values (for instance, 12PM is usually sunnier than 9AM). Environmental features also depend on the error from previous hours (e.g., due to

shade in the previous hours, a deviation in the values from the normal can be observed). Join all of the three together, and we get seasonal autoregressive integrated moving average (SARIMA). The following is a mathematical representation of the SARIMA model [31]:

$$SARIMA(p,d,q)(P,D,Q)_S =$$

$$\left(1-\phi_1 B\ -\cdots\ -\ \phi_p B^p\right)(1-\Phi_1 B^s-\cdots\ -\ \Phi_P B^{PS})(1-B)^d$$

$$(1-B^s)^d\,.\,y_t=\left(1+\theta_1 B+\cdots+\theta_q B^q\right)\left(1+\Theta_1 B^s+\cdots+\Theta_Q B^{QS}\right)\varepsilon_t, \quad (2.2.1)$$

where y_t is the considered signal (irradiance or temperature), ε_t is a random error term, ϕ and θ are non-seasonal autoregressive (AR) coefficients and non-seasonal moving average (MA) coefficients, respectively. Φ and Θ are the seasonal AR and seasonal MA coefficients. Lastly, B is the back shift operator [14].

Next to account for the historical data of the considered signal:

$$y_{t-m}=B^m y_t, \quad (2.2.2)$$

where B is the back shift factor that shifts the considered signal by m lags.

Seasonal (S) in SARIMA, is a component that accounts for a recurring fluctuation that occurs within a year. This pattern is usually caused by the presence of seasons and daily patterns as explained previously. The span of the most predominant seasonal pattern in the time series is selected by the S term in Equation (2.2.1).

The autoregressive (AR) component of SARIMA relies on the fact that the current value of the output y_t is expressed as a linear combination of the previous y_t observations, i.e., y_{t-1}, y_{t-2} ...., with a random error term. [32], as seen in Equation (2.2.3):

$$\left(1 - \phi_1 B \ - \cdots \ - \ \phi_p B^p\right) y_t = \ \varepsilon_t, \tag{2.2.3}$$

where p is the non-seasonal AR order dictating the number of previous observations of y_t that need to be considered to obtain y_t.

The seasonal autoregressive (SAR) component of SARIMA predicts the signal y_t by utilizing prior observations with lags that are multiples of S (span of Seasonality), as shown in Equation (2.2.4):

$$(1 - \Phi_1 B^s - \cdots \ - \ \Phi_P B^{PS}) y_t = \ \varepsilon_t, \tag{2.2.4}$$

where P is the seasonal AR order, that dictates the number of previous observations y_{t-PS} that needs to be considered to obtain the signal y_t. For example, with monthly data (S = 12), a seasonal first order ($P = 1$) autoregressive model would use y_{t-12} to predict y_t.

The moving average (MA) component of SARIMA is different than the moving average smoothing, where we just find the average for a specified previous period. The moving average in a time-series makes up for the fact that AR models ignore the unobservable correlation noise structures in the time series [33]. With that in mind, the MA is a combination of imperfectly predictable terms in the current error component of the time series ε_t with respect to y_t and previous error lags i.e., ε_{t-1} , ε_{t-2} , …… , ε_{t-q}, as seen in Equation (2.2.5):

$$\left(1 + \theta_1 B + \cdots + \theta_q B^q\right) \varepsilon_t = \ y_t, \tag{2.2.5}$$

where q is the non-seasonal MA order, that dictates the number of previous error observations of y_t i.e., ε_t , that need to be considered to obtain y_t.

The seasonal moving average (SMA) component of SARIMA predicts the signal y_t by utilizing previous error observations of y_t with lags that are multiples of S (span of Seasonality), as seen in Equation (2.2.6):

$$\left(1 + \Theta_1 B^s + \cdots + \Theta_Q B^{QS}\right)\varepsilon_t = \ y_t, \tag{2.2.6}$$

where Q is the seasonal MA order, that dictates the number of previous observations ε_{t-QS} that need to be considered to obtain the signal y_t. For example, with monthly data ($S =$ 12), a seasonal first order ($Q = 2$) moving average model would use ε_{t-12} & ε_{t-24} to predict y_t.

Finally, the integrated component (I) of the SARIMA model. In time series, the signals are probabilistic in nature, meaning the output values are non-deterministic which makes it harder to predict their future values. To overcome this, the time series which is often non-stationary, is converted into a stationary signal by applying certain mathematical transformations. Stationarity implies that the statistical properties of the signal in the future are equivalent to that of the past. In general, there are two types of stationarity that are: weakly stationary and strictly stationary. Strictly stationary implies that the probability of a joint distribution of a time series process is unchanged overtime [34]. This condition is often too strong and cannot be fulfilled. Therefore, we assume weakly stationary in our case. A weakly stationary signal features constant statistical properties i.e., constant mean, variance, and covariance.

With our signal being non-stationary, we use a mathematical transformation tool known as differencing to achieve stationarity. This is the integrated component (I).

A time series signal is a combination of mean, trend, seasonality, and random noise as mentioned previously, and as shown in Equation (2.2.7):

$$\boldsymbol{y_t} = \mu + T_t + S_t + \varepsilon_t, \tag{2.2.7}$$

where μ is the mean, T_t is the trend, S_t is seasonality, and ε_t is the random error component. Equation (2.2.7) presents an additive decomposition since the magnitude of the seasonal component does not vary with the level of the time series.

To achieve weakly stationarity, we need to remove both trend and seasonality components, resulting in just a constant mean and random noise. This is done using the mentioned differencing method. In this method, we just difference the signal y_t with its lagged counterpart y_{t-d}. The proof to this method is shown in Equation (2.2.8), which verifies the removal of the trend component, and in Equation (2.2.9), which verifies the removal of the seasonal component:

$$\nabla y_t = y_t - y_{t-d} = \beta_1 \cdot t - \beta_1 \cdot (t-1) + \varepsilon_t - \varepsilon_{t-1} = \beta_1 + \nabla\varepsilon_t, \tag{2.2.8}$$

$$\nabla_D y_t = s_t - s_{t-D} + \varepsilon_t - \varepsilon_{t-D} = \nabla_D \varepsilon_t, \quad s_t = s_{t-D}, \tag{2.2.9}$$

where d is the degree of differencing applied to a non-stationary signal, in the above $d = 1$. D equals the period of seasonality i.e., $D = 1$. β is the slope of the trend. s resembles the repeating pattern, which replicates every S periods.

It is not always easy to spot trend and seasonal patterns in the signal therefore, we need the help of certain tools to identify their presence.

The first tool is the autocorrelation function (ACF), which is a correlation plot in which point on the ACF plot stands for the correlation between the signal and its lagged counterpart. As seen in Equation (2.2.10), we find the variance between the signal y_t and its *k* lagged version y_{t-k}, divided by the standard deviation:

$$\mathrm{ACF}_{x=k} = \frac{\mathrm{Cov}(y_t, y_{t-k})}{\sqrt{\mathrm{Var}(y_t)\,\mathrm{Var}(y_{t-k})}}. \tag{2.2.10}$$

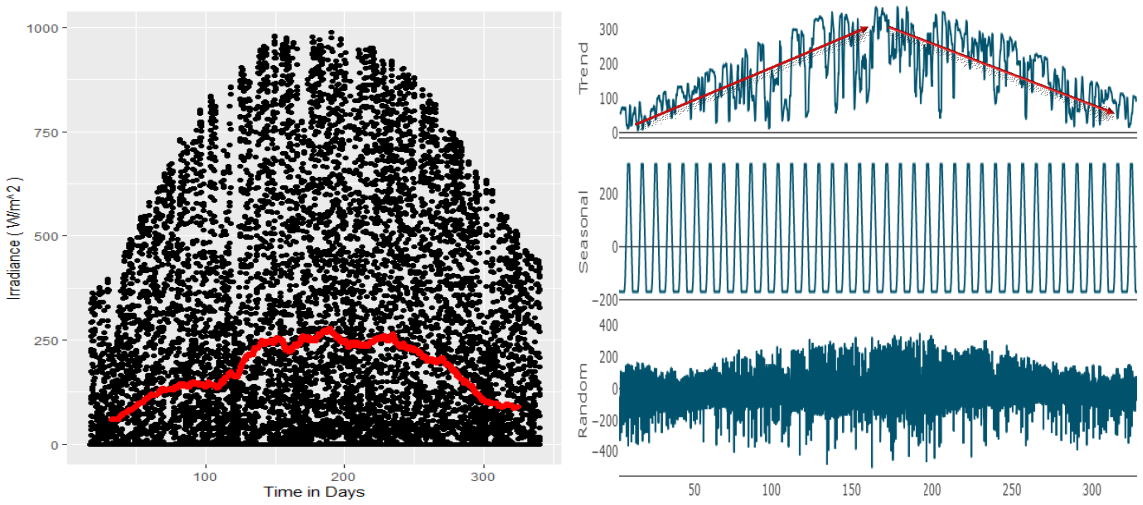
The second tool is the partial auto correlation function (PACF). It is similar to the ACF, but the lags between the signal and the *k* lagged signal is removed. This helps us identify the sole effect of the *k* lagged signal. The PACF is given as:

$$\mathrm{PACF}_{x=k} = \mathrm{ACF}_{x=k}(y_t, y_{t-k} | y_{t-1}, y_{t-2}, \dots, y_{t-k+1}). \tag{2.2.11}$$

Lastly, most statistical libraries support a *Decomposition* visualization tool. This method can be used to visually examine both trend and seasonal attributes of a time series signal.

2.3 Building SARIMA Model for Irradiance – City of Ottawa

The graph in Figure 2.3-1 (a) shows the suns' irradiance for 2019, for the City of Ottawa, Ontario, Canada [35]. Each day consists of 48 points representing the irradiance every 30 min. We can see an increasing trend in irradiance until day 200, and then a decreasing trend afterwards. The red line is the 1000-point smoothing moving average used to indicate the presence of a trend. Figure 2.3-1 (b) is obtained from the decomposition tool.



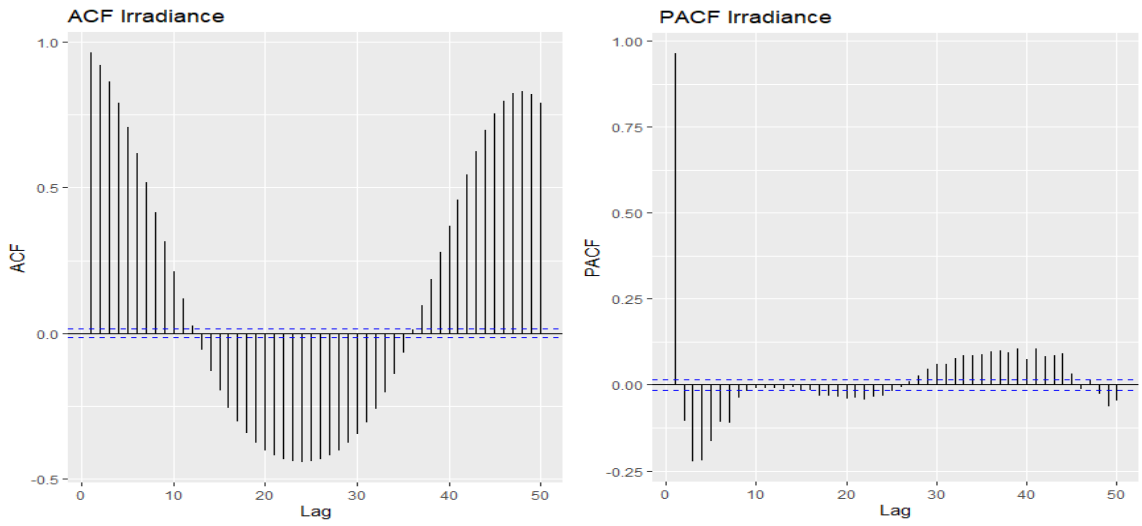
(a) Actual irradiance vs Time in days (b) Trend, seasonal and residual decomposition

Fig. 2.3-1: 2019 Irradiance values for the City of Ottawa

As indicated by the red arrows in the trend graph, we can confirm both upward and downward trends. In the seasonal graph, we can also spot seasonality, which is not so clearly seen by Figure 2.3-1 (a). This seasonality occurs every 48 points and resembles a daily seasonal pattern. Lastly, the random graph presents the noise component.

Figure 2.3-2 (a) shows the discussed ACF plot for irradiance. We can see a decaying pattern from 0 to10 for the lags above the dashed blue line. This line stands for the 95% confidence interval. Trend in the irradiance graph is one of the things that this decay represents. We can also see clear oscillations, confirming the presence of seasonality. ACF of the stationary signal will be used later to identify the number of lags used in both MA (q) and SMA (Q) terms.

Finally, Figure 2.3-2 (b) is the PACF graph that will be used later to identify the number of lags for AR (p) and SAR (P) terms after we achieve stationarity.



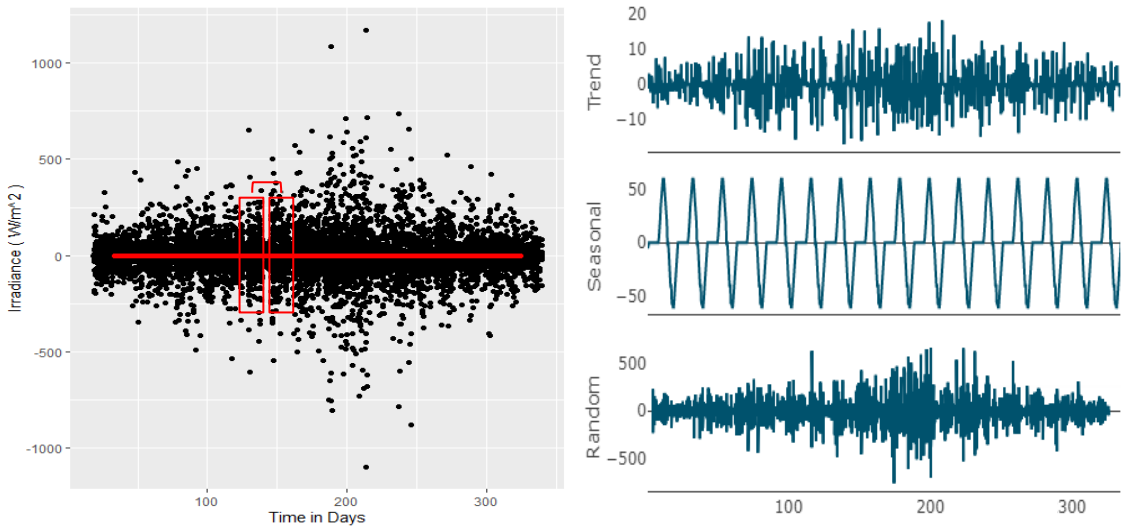
(a) Auto-correlation function plot (b) Partial auto-correlation function plot

Fig. 2.3-2: 2019 Irradiance correlation functions

After concluding that the signal is nonstationary, we now attempt to achieve stationarity by differencing both trend and seasonal components once, ensuing $d = 1$ and $D = 1$ respectively. We then use the augmented Dicky-Fuller (ADF) test to test for stationarity.

Figure 2.3-3 (a) shows the irradiance data after it was trend differenced. It is clear that we neutralized the trend, as the red moving average is at 0. Nevertheless, the seasonality is not clear enough. We can barely notice its presence with the help of the spacing between the accumulated data points as indicated by the red-vertical rectangles.

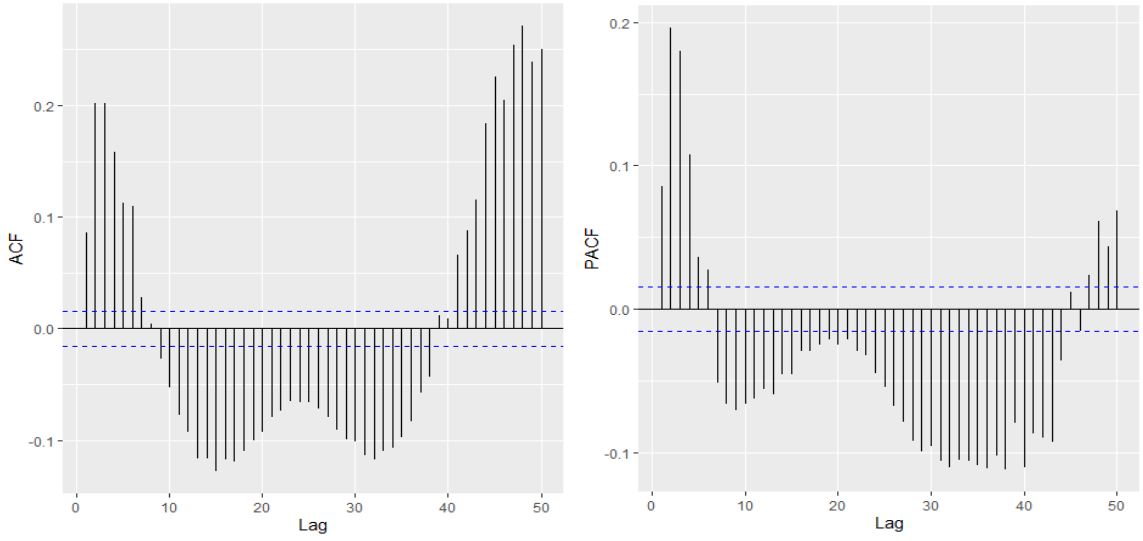
On the other hand, with the help of the decompose graph in Figure 2.3-3 (b), it is easy to spot the daily seasonality. Moreover, we notice that the trend graph turned into random noise which indicates the absence of a trend.



(a) Irradiance versus Days (b) Decomposition plot

Fig. 2.3-3: Trend differenced irradiance plots

Figure 2.3-4 (a) shows the ACF plot of the trend differenced series. Oscillation might have been distorted, but a defined oscillation is still present. Furthermore, differencing is required since oscillation still exists in both ACF and PACF.



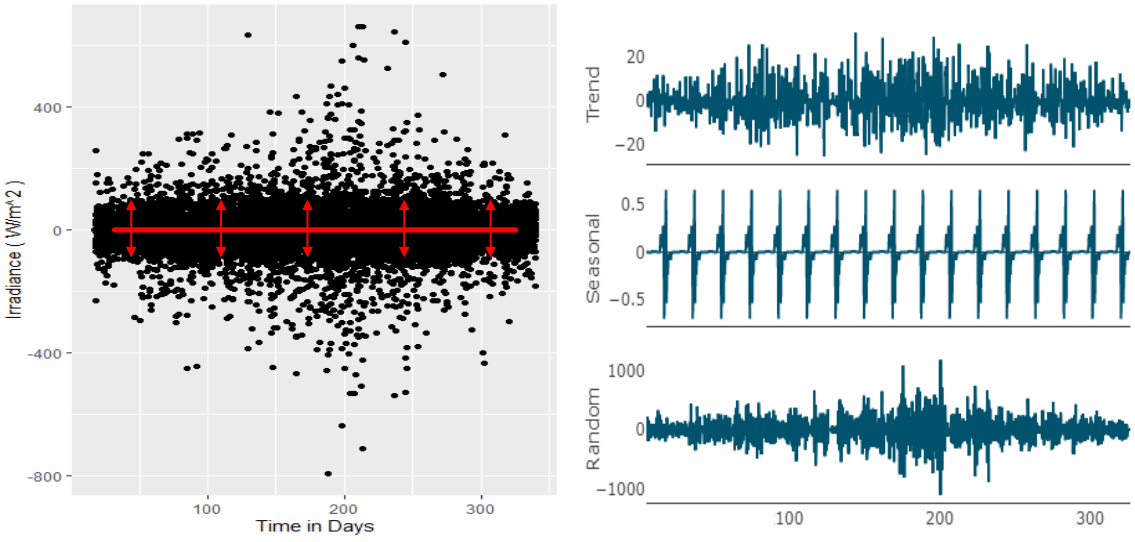
(a) Auto-correlation function plot (b) Partial auto-correlation function plot

Fig. 2.3-4: Trend differenced irradiance correlation functions

The graph in Figure 2.3-5 (a) shows the irradiance data after it was both trend and seasonally differenced. The slight spacing between the data points is now absent, and the data points are now more aligned towards the mean, this shows a constant mean. On the other hand, constant variance is indicated by the red vertical arrows of equal length.

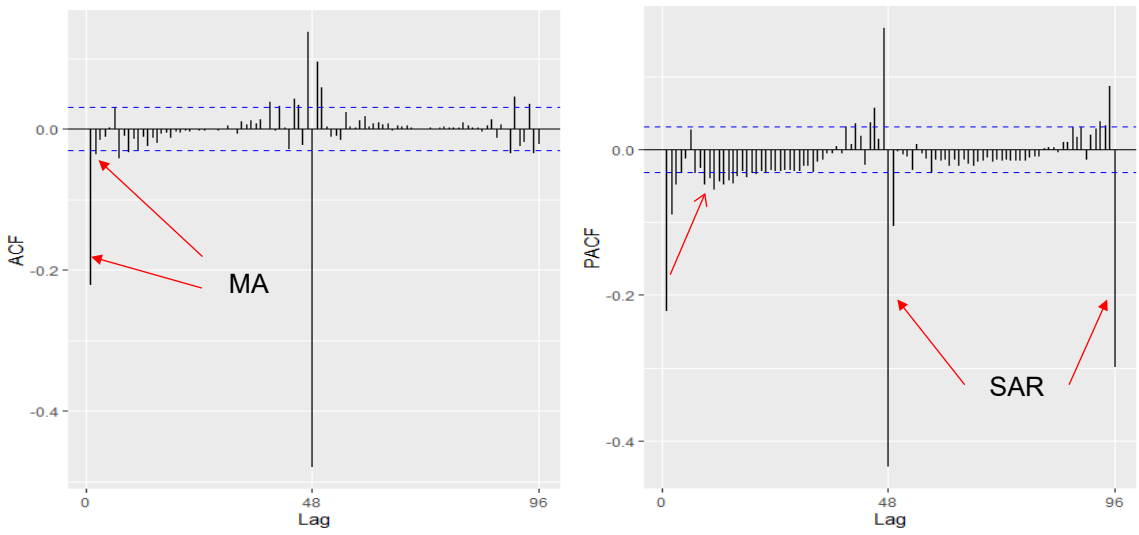
Moreover, looking at the decomposition graph in Figure 2.3-5 (b), it is obvious that the oscillation in the seasonal sub-graph is completely distorted, signaling the removal of seasonality. Furthermore, the trend plot portrays the shape of a random signal with inconsistent fluctuations. We can further confirm stationarity by plotting the ACF and PACF plots of the trend and seasonally differenced signal.

In Figure 2.3-6, both ACF and PACF plots show more lags inside the 95% confidence band visually indicating that we reached stationarity.



(a) Irradiance versus Days (b) Decomposition plot

Fig. 2.3-5: Trend and seasonally differenced irradiance plots



(a) Auto-correlation function plot (b) Partial auto-correlation function plot

Fig. 2.3-6 : Trend and seasonally differenced irradiance correlation function plots

Although, the trend and seasonally differenced signal is visually stationary, it is still required to pass the ADF test to proceed with the SARIMA model selection. The ADF tests the AR coefficient ϕ (as seen in Equation (2.2.3)) for the existence of a unit root, if absent then the time series signal is stationary. It uses a Student's t-test and compares it to a Dicky Fuller distribution. Assuming a confidence level of 95% (p = 0.05), If the p value of the test is less than 0.05 then the null hypothesis is rejected, and the signal is stationary. Otherwise, if the p value is greater than 0.05 the null hypothesis is accepted, and the signal is nonstationary. In Figure 2.3-7, the p value is 0.01 indicating a stationary signal.

$$Null\ hypothesis \rightarrow\ implies\ \ Non-Stationarity$$

$$Alternative\ \ hypothesis \rightarrow implies\ Stationary$$

```
        Augmented Dickey-Fuller Test

data:  Irradiance
Dickey-Fuller = -32.024, Lag order = 24, p-value = 0.01
alternative hypothesis: stationary
```

Fig. 2.3-7 : Null hypothesis assignment and ADF results

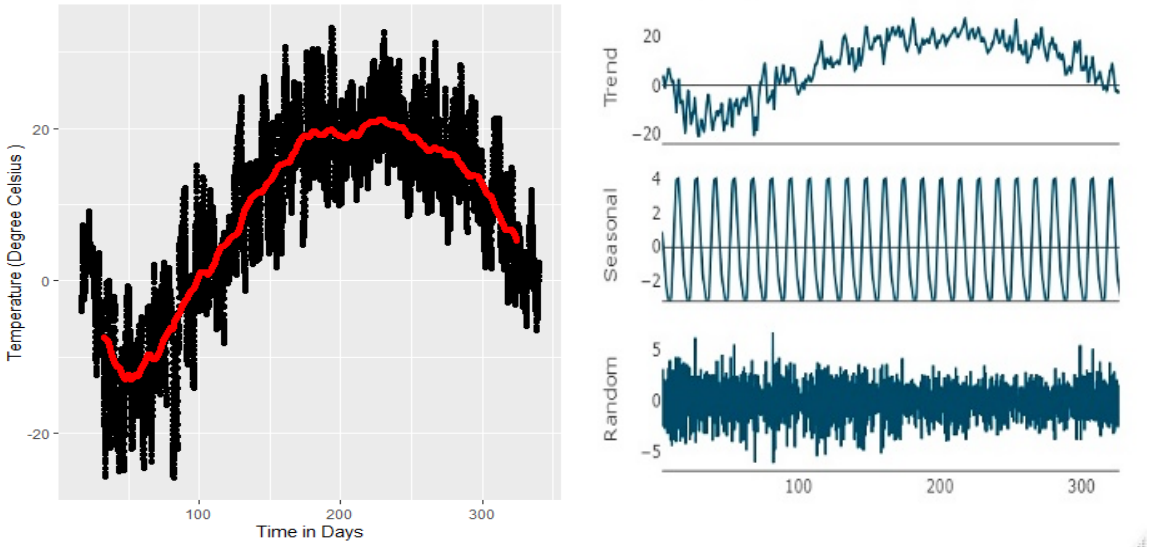
With our signal being stationary, we need to estimate a SARIMA model that fits. This can be achieved by looking at both the ACF and PACF plots. If the ACF plot decays while the PACF cuts-off after a few lags then this signal possesses an AR (p) component. If the ACF plot cuts-off after a few lags while, the PACF lags decay, then this signal possesses an MA (q) component.

As seen in Figure 2.3-6, the PACF illustrates a decaying trait whereas, the ACF cuts-off abruptly after the first lag. This states the presence of the MA terms. We can find the number of these terms by looking at the lags that are above the confidence bands, before the cutoff takes place. It is 2 in this case therefore, $q = 2$ and $p = 0$. Both the trend difference d and the seasonal difference D are 1 since we only differenced the signals once with respect to each other. Lastly, to assign the SMA term, we need to look at the periodic lags in the PACF graph. In our case, we have two lags, one is 48 and the other is 96. So, the SAR term is 2. On the other hand, the SMA term is 0 since multiples of the 48^{th} lag are absent. This gives us a model of SARIMA (0,1,2)(2,1,0).

The other environmental factor that we account for is the temperature. Similar, procedures were carried out to achieve stationarity and then deduce the most suitable SARIMA model for forecasting. In the next section, we walk through the steps taken to deduce the SARIMA model for the temperature of the City of Ottawa.

2.4 Building SARIMA Model for Temperature – City of Ottawa

The graph in Figure 2.4-1 (a) shows the temperature given the number of days in the year. As stated earlier, each day consists of 48 points, which represents 30 min intervals. We can



(a) Actual irradiance vs Time in days (b) Trend, seasonal and residual decomposition

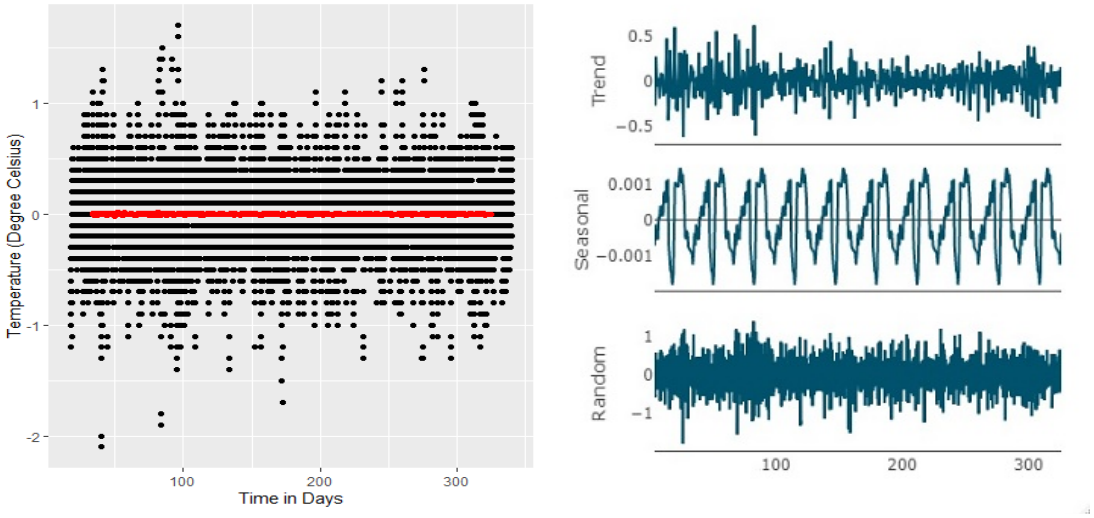
Fig. 2.4-1 : 2019 Temperature values for the City of Ottawa

clearly see an increasing trend in Temperature until day 200, and then a decreasing trend afterwards.

The graph in Figure 2.4-1 (b), indicates both upward and downward trends. In the seasonal graph, we can also spot seasonality, which is not so clearly seen by Figure 2.4-1 (a). This seasonality occurs every 48 points as well, this imitates a daily seasonal pattern.

The graph in Figure 2.4-2 (a) shows the temperature data after it was both trended and seasonally differenced. Again, the data points are now more aligned towards the mean. This re-emphasizes the fact that it possesses a constant mean and variance. Now looking at the decomposition graph in Figure 2.4-2 (b), it is obvious that the oscillation in the seasonal graph is distorted signaling the removal of seasonality as well.

With the temperature signal being visually stationary, we run the ADF test. The obtained *p* value is 0.01, which rejects the null-hypothesis and confirms stationarity.

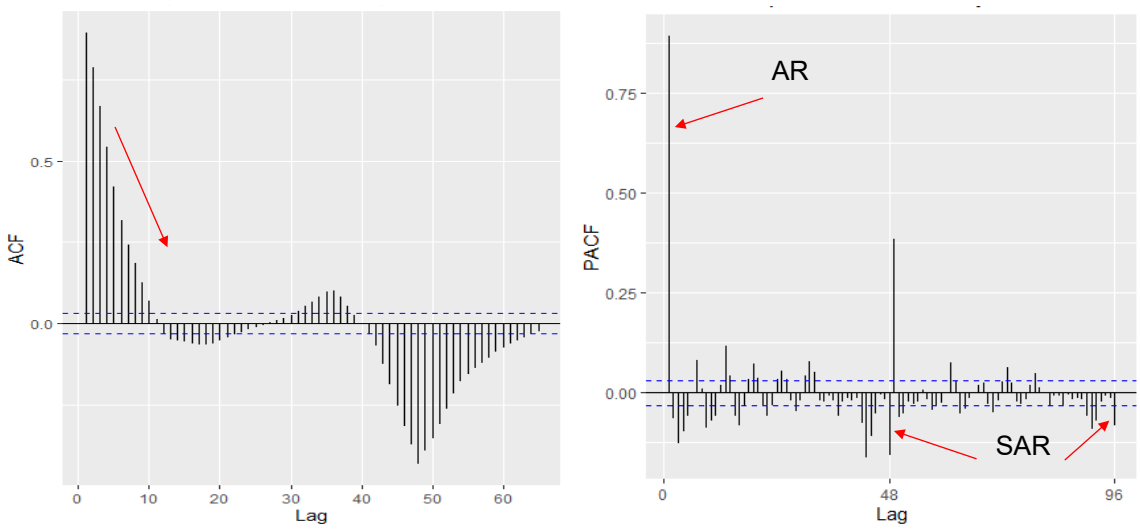


(a) Temperature versus Days (b) Decomposition plot

Fig. 2.4-2: Trend and seasonally differenced temperature plots

Therefore, now we estimate a SARIMA model by looking at the ACF and PACF plots in Figure 2.4-3.

As seen in the Figure, the ACF illustrates a decaying trait whereas, the PACF cuts-off abruptly after the first lag. This states the presence of an AR term. Looking at the lags that are above the confidence bands, before the cutoff takes place, we find the number of AR terms which equals 1 (i.e., $q = 0$ and $p = 1$). Both the trend difference d and the seasonal difference D are equal to 1, since we only differenced the signals once with respect to each other. Lastly, to assign the SAR term we need to look at the periodic lags in the PACF graph. In our case, we have two lags one is 48 and the other is 96. So, the SAR term is 2. On the other hand, the SMA term is 0 since multiples of the 48^{th} lag are absent. This gives us a model of SARIMA (1,1,0)(2,1,0).



(a) Auto-correlation function plot (b) Partial auto-correlation function plot

Fig. 2.4-3: Trend and seasonally differenced correlation function plots

2.5 Model Selection while accounting for Training Period

Due to spatial constraints of IoT devices, we need to select the smallest historic dataset, whilst limiting accuracy degradation. To do that, we plotted the average RMSE and MAE for different randomly predicted days with respect to their SARIMA model and historic periods. The plan is to choose the best model and historic data period by choosing the lowest RMSE and MAE values.

In Table I, the top model for the irradiance, i.e., (0,1,2)(2,1,0), and temperature, i.e., (1,1,0)(2,1,0), is our estimated model. The rest of the models were chosen using the auto function in the R programming language. Our irradiance model is the best model with one week of historic data. Whereas, when it comes to temperature, the (0,1,2)(2,1,0) model slightly outperformed our estimated model. Compared to irradiance, the ACF and the PACF plots were cluttered and not visually telling therefore, it was hard to distinguish the

actual SARIMA parameters. In conclusion, we end up selecting the SARIMA(0,1,2)(2,1,0) for both Irradiance and temperature forecasting with one week of historic data.

Table I: Training period, model parameters, and forecasting error

Period of Past Data – Irradiance	*Model*	*RMSE*	*MAE*
5 Days	(0,1,1)(2,1,0)	185	126
	(0,1,2)(2,1,0)	184	126
	(0,1,2)(1,1,1)	175	121
1 Week	**(0,1,2)(2,1,0)**	**159**	**104**
	(0,1,1)(2,1,0)	167	108
	(1,1,0)(1,1,0)	201	126
1 Month	(0,1,1)(2,1,0)	167	108
	(0,1,2)(2,1,0)	165	110
	(3,1,0)(1,1,0)	197	132
4 Months	(0,1,1)(2,1,0)	169	106
	(0,1,2)(2,1,0)	169	106
	(1,0,1)(1,1,0)	204	127
6 Months	(0,1,1)(2,1,0)	169	106
	(0,1,2)(2,1,0)	168	106
	(2,0,3)(2,1,0)	171	114
Period of Past Data -Temperature	*Model*	*RMSE*	*MAE*
5 Days	(1,1,0)(2,1,0)	2.93	2.58
	(2,1,1)(1,1,0)	2.83	2.45
	(0,1,2)(2,1,0)	1.20	1.02
	(1,1,0)(2,1,0)	2.33	2.06

1 Week	(2,1,1)(1,1,0)	1.72	1.48
	(0,1,2)(2,1,0)	**1.02**	**0.87**
1 Month	(1,1,0)(2,1,0)	1.66	1.44
	(2,1,1)(1,1,0)	1.46	1.24
	(0,1,2)(2,1,0)	1.14	0.94
4 Months	(1,1,0)(2,1,0)	1.20	0.99
	(2,1,1)(1,1,0)	1.28	1.07
	(0,1,2)(2,1,0)	1.06	0.90
6 Months	(1,1,0)(2,1,0)	1.34	1.13
	(2,1,1)(1,1,0)	1.23	1.02
	(0,1,2)(2,1,0)	1.06	0.88

RMSE – Root Mean Squared Error
MAE – Mean Average Error

2.6 Kalman Filter Theory and Formulation

Kalman filter is a stochastic and recursive data filtering algorithm, which uses the estimated state from the previous time step and the current measurements to compute improved estimations [19]. The goal is to minimize the mean squared error of the estimation towards the true value when specific uncertainties are available [1].

Before we proceed to the formulation, which is relevant to our application, we provide a quick walkthrough of how the filter operates with the help of Figure 2.6-1.

- First, initialize the process (red block in Figure 2.6-1) by inputting an initial guess and its uncertainty. This will act as our previous state *t* [37].

- Second, use the dynamic model (SARIMA) to predict the next state *t+1* and extrapolate the estimate uncertainty. This is the prediction phase (yellow block) [37].
- In the update phase, use the estimation error and measurement error from the sensor to calculate the filter gain *K*. This gain sets more weight towards the estimate or the measurement values, depending on their uncertainty. After calculating the gain, the estimate is updated with the help of the measured values and the estimate uncertainty. The entire process then repeats [37].

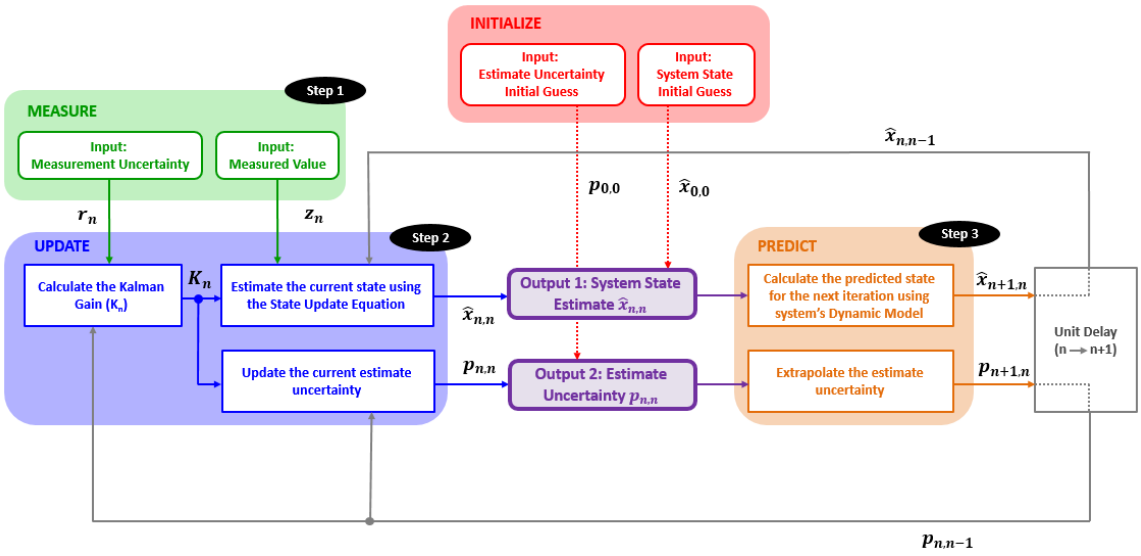


Fig 2.6-1: Kalman filter block diagram

In our application, the state variables we are trying to predict are both irradiance and temperature. The dynamic model uses the estimated SARIMA for both irradiance and temperature. Lastly, the measurements from the sensors are used to update the current state. The following elaborates on Figure 2.6-2:

Initial Guess:

$$\hat{x}_{1,0} \ , P_{1,0} \ , \tag{2.6.1}$$

initial irradiance and temperature guess in a state vector $\hat{x}_{1,0}$, with its uncertainty matrix $P_{1,0}$.

Prediction Phase:

$$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + G\hat{u}_{n+1,n}, \tag{2.6.2}$$

$$P_{n+1,n} = FP_{n,n}F^T + Q, \tag{2.6.3}$$

predict the new state $\hat{x}_{n+1,n}$ by applying the transition matrix *F* and the control matrix *G* . Both *F* and *G* are identity matrices in our case. We also extrapolate the estimate uncertainty *P* of the initial guess by applying the covariance matrix *Q*. This matrix adds the dynamic model uncertainty into account. $\hat{u}_{n+1,n}$ is the SARIMA model for both irradiance and temperature.

Refining Phase:

$$K_n = P_{n,n-1}H^T\left(HP_{n,n-1}H^T + R_n\right)^{-1}, \tag{2.6.4}$$

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n\left(z_n - H\hat{x}_{n,n-1}\right), \tag{2.6.5}$$

$$P_{n,n} = (I - K_nH)\,P_{n,\mathrm{n}-1}(I - K_n\mathrm{H})^{\mathrm{T}} + K_n\mathrm{R}\,K^T, \tag{2.6.6}$$

in the Kalman Gain K_n, matrices *R* and *H* which represent the measurement covariance matrix and the observation matrix receptivity, are used. *H* (external input transformation matrix) in our case is an identity matrix since the measured values represent the state variable themselves. As seen in (2.6.5), in addition to the SARIMA estimate, the estimate for the current time slot $\hat{x}_{n,n}$ is updated using the measured value from the sensor z_n.

The result of the forecasted irradiance and temperature for *t+1* is the product of the above process. This concludes the summary of the general Kalman equations. Now we procced

to the step-by-step formulation, and demonstrate the tailored equations used in our simulation.

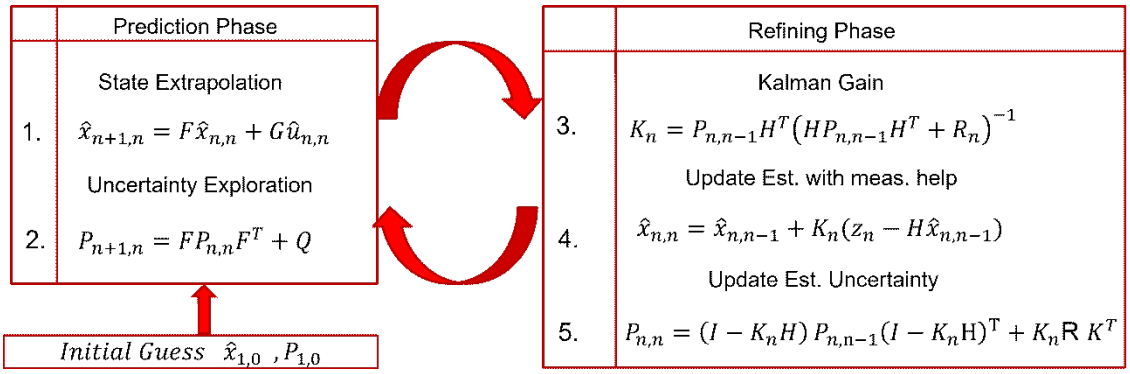


Fig 2.6-2: Kalman filter general equations

2.7 Kalman Filter Model Building for Irradiance and Temperature

Our goal is to predict both irradiance and temperature values of time slot t+1. Then we feed them into our Solar-EHMU, to anticipate the expected state of charge of the battery in time slot t+1, as discussed later in Chapter 3. With this in mind, solar irradiance and ambient temperature will act as our state variables, and with the help of the dynamic model i.e., SARIMA (0,1,2)(2,10) for both irradiance and temperature, we are able to extrapolate the variables at state n to state n+1 as seen in Figure 2.7-1.

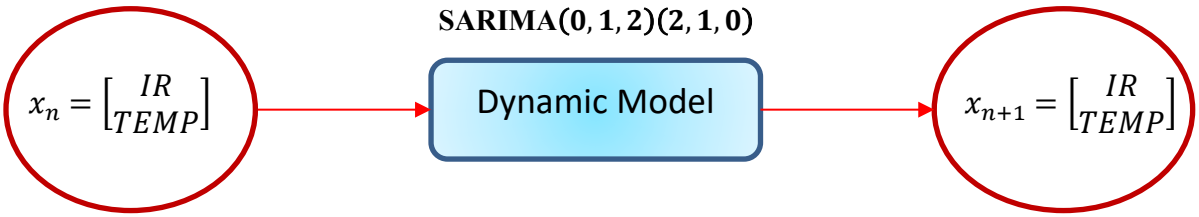


Fig 2.7-1: State space representation of irradiance and temperature

As shown in Figure 2.7-1, the main variable x is represented in a 2D vector, where the irradiance (*IR*) is in the first dimension and is represented by x_1, and the temperature (*Temp*) is the second dimension and is represented by x_2.

The main goal of this state space model is to capture the temporal relationship between the state of both irradiance and temperature at the current time slot n and their states in the upcoming time slot n+1, given the rate of change provided by the dynamic model SARIMA(0,1,2)(2,1,0) as seen in Equation (2.7.1):

$$x_{n+1} = x_n + \Delta x_{n+1}, \tag{2.7.1}$$

where the rate of change Δx_{n+1} represents the natural change in both irradiance and temperature between time n and n+1. This change depends on various known and unknown factors. The known factors include the time of day, as reflected in Figure 2.7-2 and the season. The unknown factors, include shade from unprecedent objects and sudden changes of cloud opacity during the day. These factors usually lead to prediction errors and are hard to account for since they tend to be random in nature. To capture as much of the consistent patterns of the known factors, we introduced SARIMA and settled on SARIMA(0,1,2)(2,1,0) for both irradiance and temperature, this is represented by $\hat{u}_{n+1,n}$ in Equations (2.6.2) and (2.7.2). Now, the prediction and estimation errors are represented by $P_{n+1,n}$ which depends on the previous estimation error and the dynamic prediction error i.e., average deviation from the true value x_{n+1}, in our case this follows a normal distribution between 0 and variance Q.

$$\hat{x}_{n+1,n} = \hat{x}_{n,n} + \hat{u}_{n+1,n} + P_{n+1,n}. \tag{2.7.2}$$

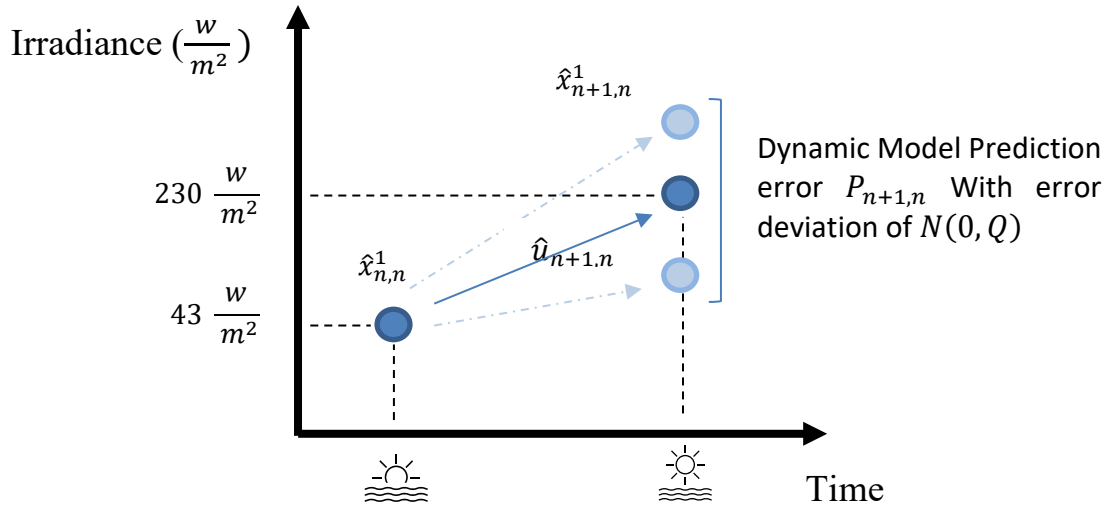


Fig. 2.7-2: Sunrise and future state of irradiance with prediction phase representation

Now, given the prediction Equation (2.7.2) for the next state of both irradiance and temperature, we can build the prediction phase equations of the Kalman filter. We first start with the state estimate equations:

$$\text{Future State of Irradiance:} \quad \hat{x}^1_{n+1,n} = \hat{x}^1_{n,n} + \hat{u}^1_{n+1,1} + P^1_{n+1,n}. \tag{2.7.3}$$

$$\text{Future State of Temperature:} \quad \hat{x}^2_{n+1,n} = \hat{x}^2_{n,n} + \hat{u}^2_{n+1,1} + P^2_{n+1,n}. \tag{2.7.4}$$

We represent both variables in vector form as follows:

$$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + G\hat{u}_{n+1,n} + P_{n+1,n}, \tag{2.7.5}$$

$$\hat{x}_{n+1,n} = \begin{bmatrix} \hat{x}^1_{n+1,n} \\ \hat{x}^2_{n+1,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}^1_{n,n} \\ \hat{x}^2_{n,n} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{u}^1_{n+1,1} \\ \hat{u}^2_{n+1,1} \end{bmatrix} + \begin{bmatrix} P^1_{n+1,n} \\ P^2_{n+1,n} \end{bmatrix}. \tag{2.7.6}$$

With the help of Equations (2.7.5) and (2.7.6) we are able to deduce the values of both matrices *F* and *G*. In our case, both are identity matrices.

For our application, we can deduce the second prediction equation of the Kalman filter by substituting the values of *F* into Equation (2.6.3). We get:

$$P_{n+1,n} = P_{n,n} + Q, \quad where\ Q = \begin{bmatrix} Q^1_{n+1,n} & 0 \\ 0 & Q^2_{n+1,n} \end{bmatrix}. \tag{2.7.7}$$

Since F is an identity matrix and $IP_{n,n}I^T = P_{n,n,}$ we end up with Equation (2.7.7), where matrix Q consists of uncorrelated irradiance variance $Q^1_{n+1,n}$ and temperature variance $Q^2_{n+1,n}$. In this application, we utilize a time varying Kalman filter where both $P_{n,n}$ and K_n vary with time.

Figure 2.7-3 summarizes the initial guess and prediction phase of the Kalman filter. To initiate the prediction phase of the Kalman filter, we need to input an educated initial guess i.e., initial state $\hat{x}_{1,0}$, and an estimate uncertainty i.e., the covariance matrix $P_{1,0}$, since the actual value (green point in Figure 2.7-3) is unknown to us. The best initial estimate used in most cases is the mean (blue point in Figure 2.7-3) which is the center of the covariance blob $P_{1,0}$. In our case, the chosen mean value for irradiance is $0\,\frac{w}{m^2}$ since there is negligible irradiance at dawn accompanied by small variance ($P^1_{1,0}= 6.08^2$). For temperature, $\hat{x}^2_{0,0}$ was chosen i.e., the previous time slot before dawn, with a variance of $P^2_{1,0}= 5.422^2$ and $P^2_{1,0} = 2.948^2$ for winter and summer, respectively. We attained all variances by averaging the RMSE for various forecasts and squaring it. Using $\hat{x}_{1,0}$ and $P_{1,0}$, the estimate in the next time slot $\hat{x}_{2,1}$ and its uncertainty $P_{2,1}$ can be attained using Equations (2.7.6) and (2.7.7), respectively. Also, it is quite evident that $P_{2,1}$ is a combination of $Q_{2,1}$ and $P_{1,0}$ forming a bigger covariance blob (red dashed circle in Figure 2.7-3). Most importantly our estimate (as seen in Figure 2.7-3), is not close to the true value as is often the case. Therefore, we apply the refining phase feature in the filter, which uses the measurements from irradiance and temperature sensors to enhance the accuracy of our estimate.

Since the used sensors measure temperature and irradiance with some uncertainty, and the outputs of our system are the state variables themselves, there is no need for an observation (transformation) matrix *H,* leaving us with the system output Equation (2.7.8):

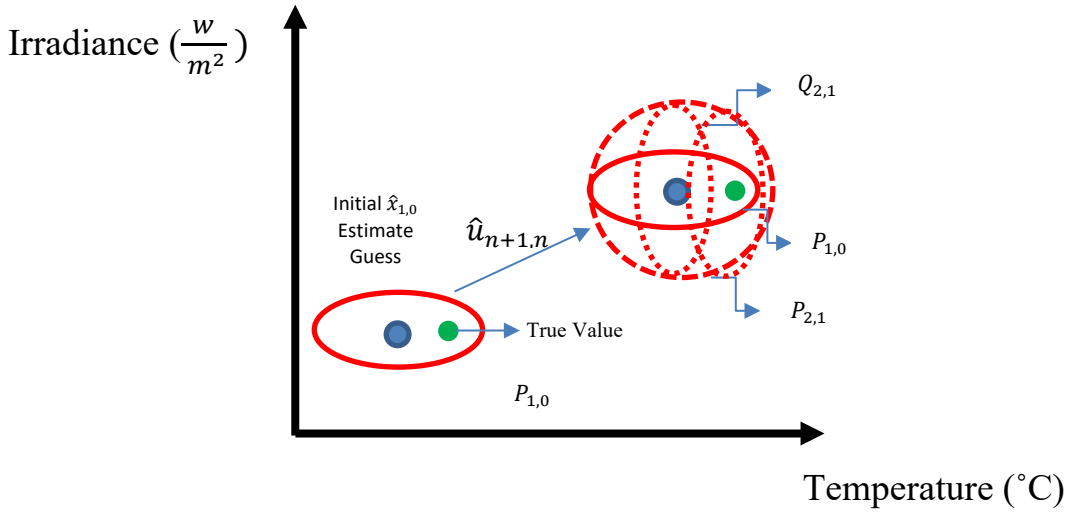


Fig. 2.7-3: Kalman filter initial guess and prediction phase summary

$$Ouptut\ Variable\ = y_{n,n} = \begin{bmatrix} y^1_{n,n} \\ y^2_{n,n} \end{bmatrix} = H \cdot \begin{bmatrix} x^1_{n,n} \\ x^2_{n,n} \end{bmatrix},\ where\ H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$y_{n,n} = x_{n,n}. \tag{2.7.8}$$

The uncertainty of both irradiance and temperature are represented using a 2 by 2 matrix R_n, shown in Equation (2.7.9):

$$R_n = \begin{bmatrix} R^1_{n,n} & 0 \\ 0 & R^2_{n,n} \end{bmatrix}. \tag{2.7.9}$$

The values of R_n can be easily obtained from the specification sheets provided by the supplier of these units. In our case, the temperature sensor under consideration is the commonly used DHT22 with $R^2_{n,n}$= 0.5^2 and the irradiance sensor under consideration is a Si-V-010 with an $R^1_{n,n}$ = 5^2. Most available sensors have comparable uncertainty values. Knowing H, $P_{n,n-1}$ and R_n, we can use Equation (2.6.4) to attain the Kalman gain K_n:

$$K_n = P_{n,n-1}\left(P_{n,n-1} + R_n\right),^{-1}\ \ since\ HP_{n,n-1}H^T = P_{n,n-1}, \tag{2.7.10}$$

where $P_{n,n-1}$ is the estimate error from the prediction phase. With K_n being defined, we can input both K_n and H in Equation (2.6.5), to refine our predicted estimate.

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n\left(z_n - \hat{x}_{n,n-1}\right) = (1 - K_n)\hat{x}_{n,n-1} + K_n z_n, \qquad (2.7.11)$$

where z_n is the measured readings from the sensors at state n. It is clearly seen from Equation (2.7.11) that $(1 - K_n)$ and K_n are the weights given to the predicted estimate of the previous state ($\hat{x}_{n,n-1}$) and the measurements of the sensors from the current state z_n. When the measurement uncertainty is larger than the estimate uncertainty, K_n is smaller than 0.5 since more weight is put towards the estimate compared to the measured values. On the other hand, when the measurement uncertainty is smaller compared to the estimate uncertainty, more weight is given to the measured values by making the value of K_n greater than 0.5.

Lastly, we need to calculate the uncertainty associated with the refined estimate by inputting the values of R_n, K_n and $P_{n,n-1}$ into Equation (2.7.12), which is a tailored form of Equation (2.6.6).

$$P_{n,n} = (I - K_n)P_{n,n-1}(I - K_n)^T + K_n R_n K^T. \qquad (2.7.12)$$

The prediction with the help of the dynamic model (SARIMA(0,1,2)(2,1,0)), and the refining phase with the help of the measured values, is repeated until no further measurement values are present to refine the estimate, or until K_n and $P_{n,n}$ converge and reach steady state.

2.8 Irradiance and Temperature Forecasting Results

In this section, we demonstrate the results attained from running both SARIMA and SARIMA-KF in MATLAB, where we try to forecast both irradiance and temperature readings for the City of Ottawa. The dataset [35] is used, which is for the year 2019. It was obtained from Solcast which provides both live and historical data for solar forecasting. The dataset includes solar irradiance, temperature, and cloud opacity measurements with 30 min resolution. We start from state n=0 at 12:01 AM; the prediction is done for the next state, i.e., n=1 at 12:31 AM. Once the current state n equals the predicted state n + 1, actual irradiance and temperature data is assigned to z_n with the sensor covariance, to implement the refining phase. This process is repeated until we reach the end of the daily data at 11:31 PM.

The forecasted results of both SARIMA-KF and SARIMA are evaluated using the common evaluation metrics. We introduce the metrics first and then present the results.

Mean absolute error (MAE) calculates the absolute difference between the actual values and the predicted values from our model. MAE is good to use as a baseline since it looks at the absolute error. The closer the MAE value to 0, the better the performance [38].

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|. \tag{2.8.1}$$

Root mean squared error (RMSE) calculates the error, squares the difference, and then square roots the mean. RMSE is another fundamental evaluation metric based on the squaring of residuals, therefore it penalizes large errors [38].

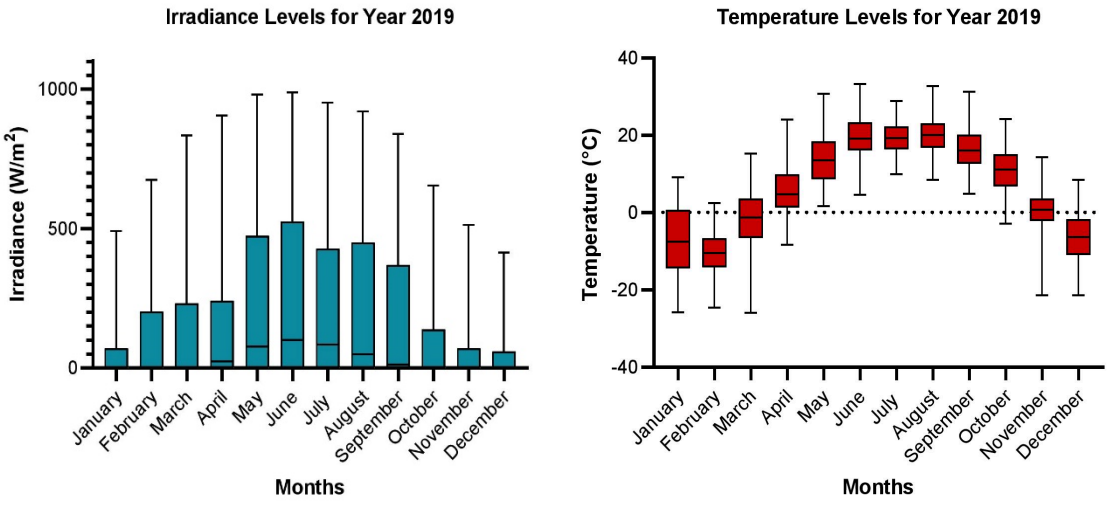
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}. \tag{2.8.2}$$

Mean absolute scaled error (MASE) is a metric that allows to compare two models. Using the MAE for each model, one can put the MAE value for the new model in the numerator and the MAE value for the original model in the denominator. If the MASE value is less than 1, then the new model performs better. If the MASE value equals 1, then the models perform the same. If the MASE value is greater than 1, then the original model performs better than the new model [20].

$$MASE = \frac{MAE_i}{MAE_j}. \tag{2.8.3}$$

Following is the results obtained from our forecasting. First, we present the monthly irradiance and temperature curves for the entire year of 2019. Secondly, per season, three random days are selected to present irradiance curves of actual, SARIMA and SARIMA-KF; accompanied by a temperature plot of one of the selected days. Lastly, the daily errors are averaged and presented for selected months per season.

The graph in Figure 2.8-1 (a) shows the monthly irradiance of 2019 in a box and whisker format. It is intuitive to see the irradiance and temperature increase leading to the summer months and drop towards the winter months. The top of the wicks represents the maximum points, and the top of the blue bar represents the upper quartile of the data. It is important to note that three-quarters of the data is below 500 $\frac{w}{m^2}$. This emphasizes the importance of the EHMU simulation to try to estimate the power going to the battery. The irradiance levels in Ottawa are not as high as the cities close to the sunbelt so, room for error in gained SoC is less tolerable. Also, we need to model the temperature effect on the solar power output given the low temperatures shown to the right.



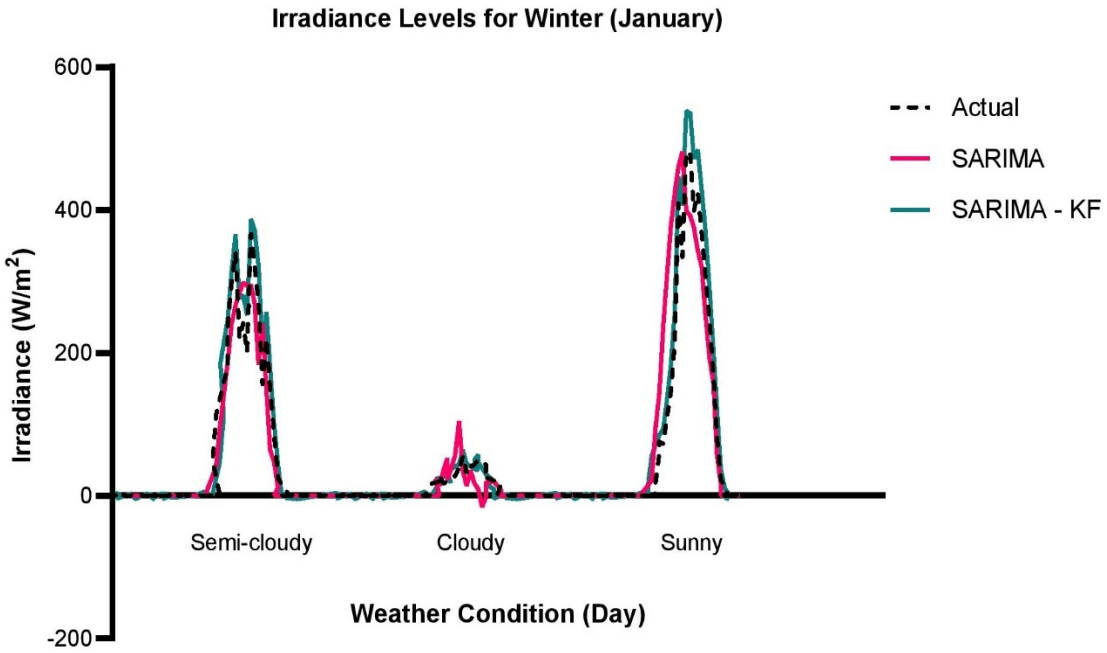
(a) Irradiance vs Months (b) Temperature vs Months

Fig. 2.8-1 Irradiance and temperature curves – Ottawa year 2019

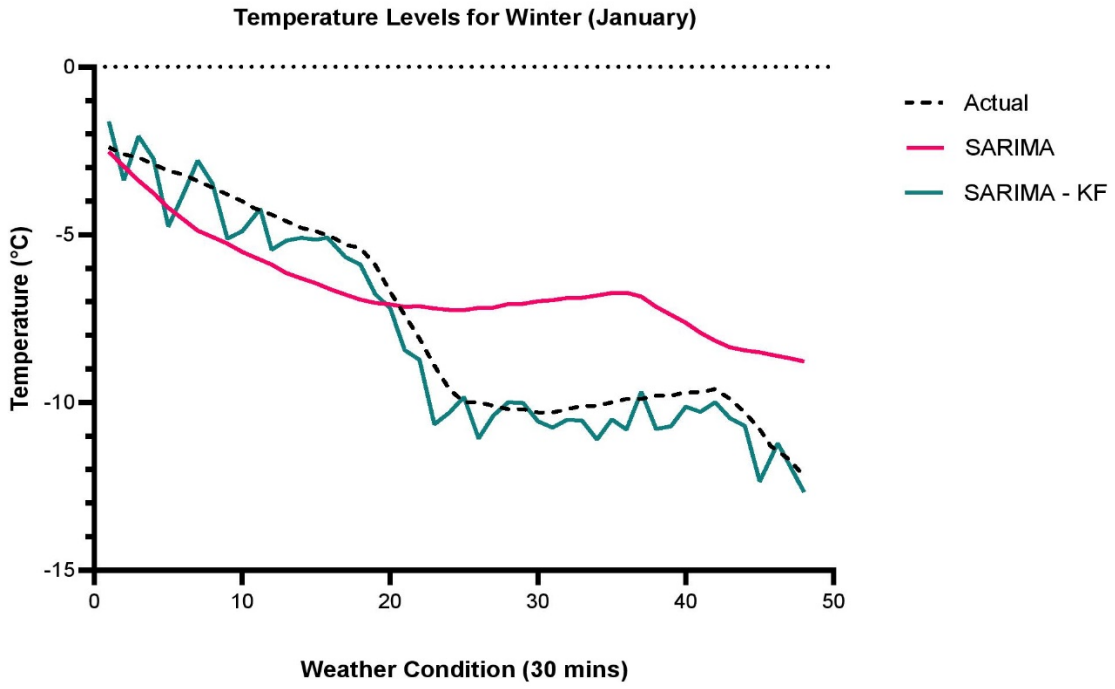
Figure 2.8-2 (a) shows the irradiance of three randomly selected days in January 2019, depending on the cloud opacity. We can see that the SARIMA-KF outperforms the SARIMA in all scenarios. In the semi cloudy day, the SARIMA-KF and the SARIMA have an RMSE of 20.1 and 37.0, respectively. Meaning, the SARIMA-KF reduced the error by 45%. It is quite visible that the SARIMA-KF tends to capture the abrupt spikes better than the SARIMA model. In the cloudy day, the SARIMA-KF model outperformed the SARIMA by an error reduction of 76%. In the sunny day, the SARIMA-KF outperformed by an error reduction of 57%. Although, both models tend to capture the trend, in this instant the SARIMA-KF overshoots on strong inflection points. But past the inflection point, due to its refining phase, the prediction follows up with the actual signal. To reemphasize the superiority of the SARIMA-KF, the MASE for all the presented days is calculated. The MASE values for the semi cloudy, cloudy, and sunny days are 0.57, 0.44

and 0.53, respectively. As seen from these values, SARIMA-KF outperforms SARIMA by an average of 50%, as the MASE is less than 1.

Figure 2.8-2 (b) represents the temperature of a randomly selected day in January 2019. The SARIMAKF tries to converge towards the actual temperature, outperforming the SARIMA model. The waveform seen in the SARIMA-KF is due to the high inaccuracy of the SARIMA model, as the KF uses input from both SARIMA and the sensor. The RMSE values of the SARIMA and SARIMA-KF are 2.9 and 0.7 respectively, this is an error reduction of 75%. The MASE is 0.30, indicating an outperformance of 70%.



(a) Irradiance Actual, SARIMA and SARIMA-KF for January 2019



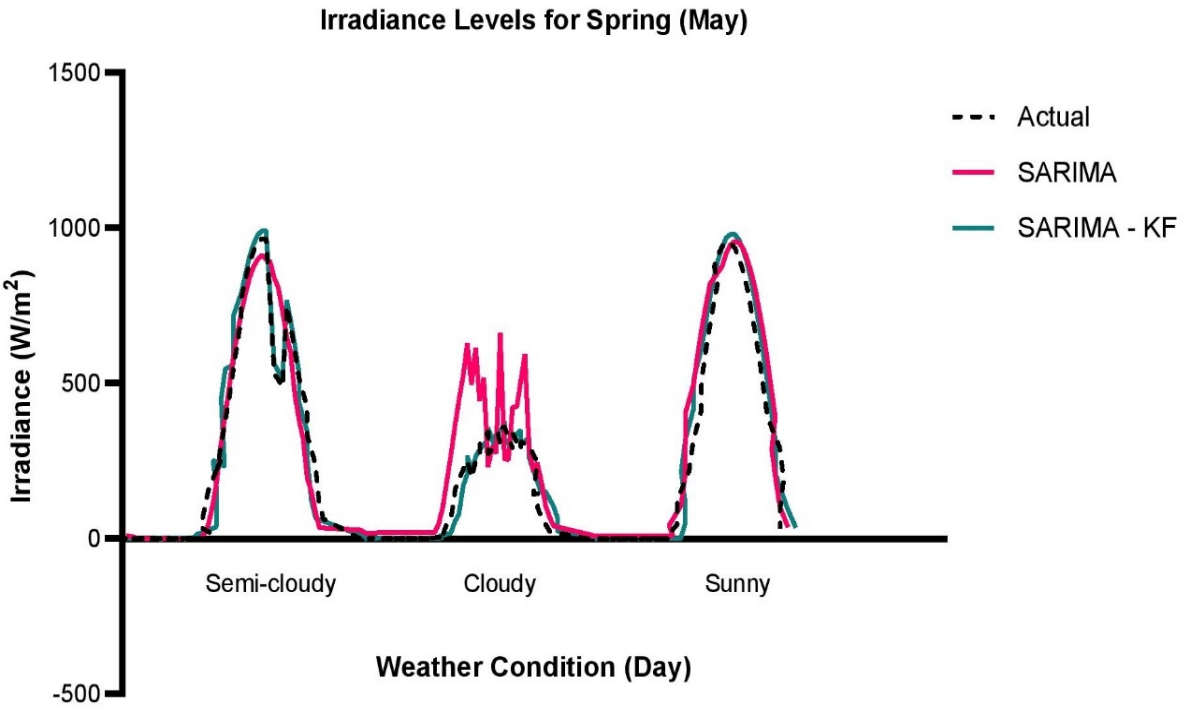
(b) Temperature Actual, SARIMA and SARIM-KF for January 2019

Fig. 2.8-2 January (winter) irradiance and temperature plots

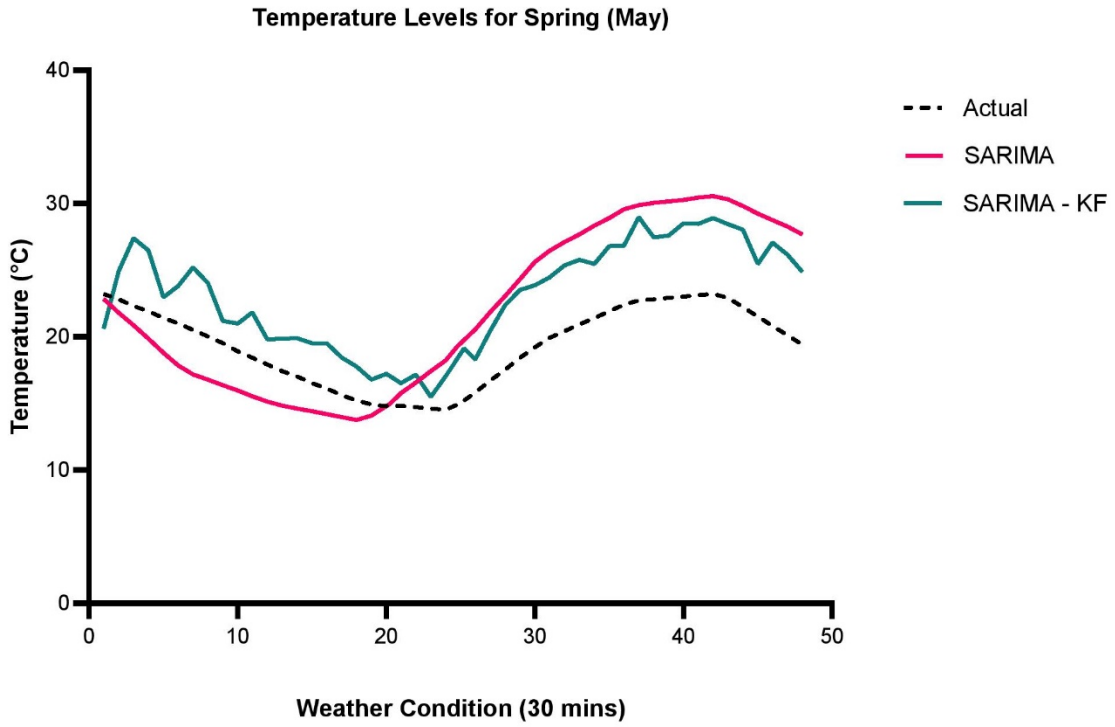
In the semi cloudy day of Figure 2.8-3 (a), SARIMA-KF and SARIMA have an RMSE of 13.8 and 90.9, respectively, this is 84% lower. With SARIMA acting as the original model and SARIMA-KF as the new model an MASE of 0.17 was calculated. This demonstrates the superiority of the SARMIA-KF. Again, abrupt changes do not affect the SARIMA-KF as much. In the cloudy day, the SARIMA-KF model outperformed the SARIMA by an error reduction of 90% and an MASE of 0.06. In the sunny day, the SARIMA outperforms with an MASE of 1.63 and an RMSE of 7.2, whereas the RMSE of the SARIMA-KF is 14.2. This is an outlier event, as in this case, the error from the sensor caused the SARIMA-KF to underperform. In addition, SARIMA models perform better when the historic days have similar patterns. In this case, four consecutive clear sunny days with smooth bell

curves for irradiance preceded the presented sunny day, leading to a more accurate model compared to the SARIMA-KF.

In the temperature plot, Figure 2.8-3 (b), SARIMA-KF converges towards the actual temperature. However, due an inaccurate initial guess, the Kalman filter overshoots for the first 20 samples and then converges between the actual and SARIMA readings. The RMSE values of the SARIMA and SARIMA-KF are 5.1 and 4.1 respectively; MASE is 0.85 showing the outperformance of the SARIMA-KF.



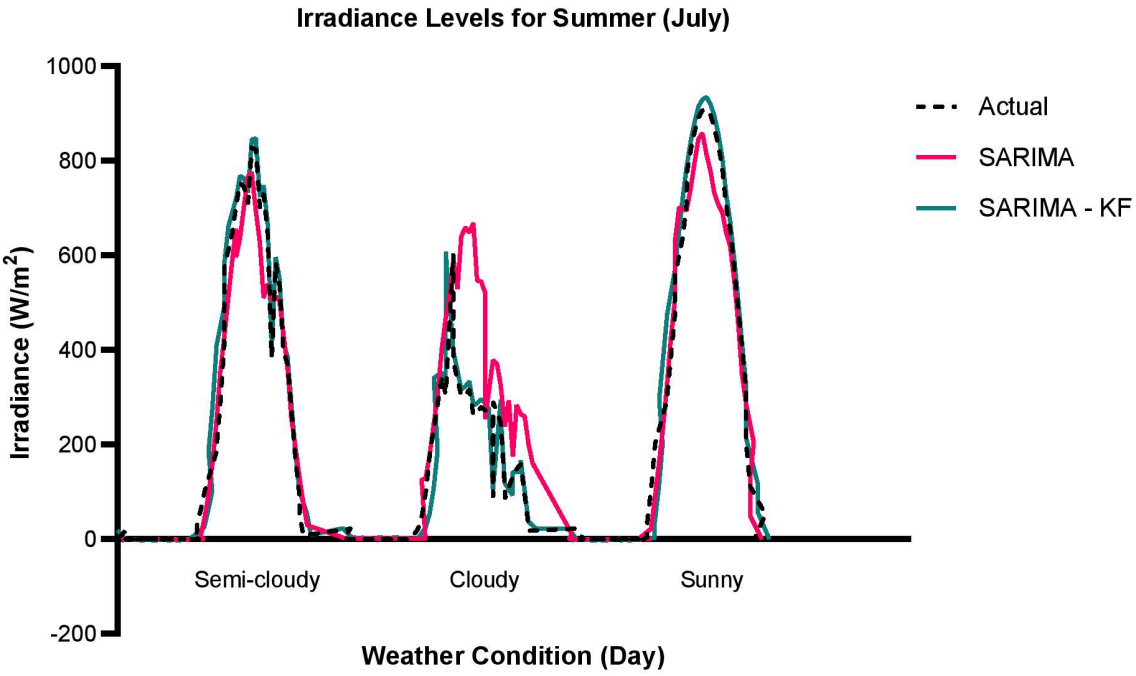
(a) Irradiance Actual, SARIMA and SARIM-KF for May 2019



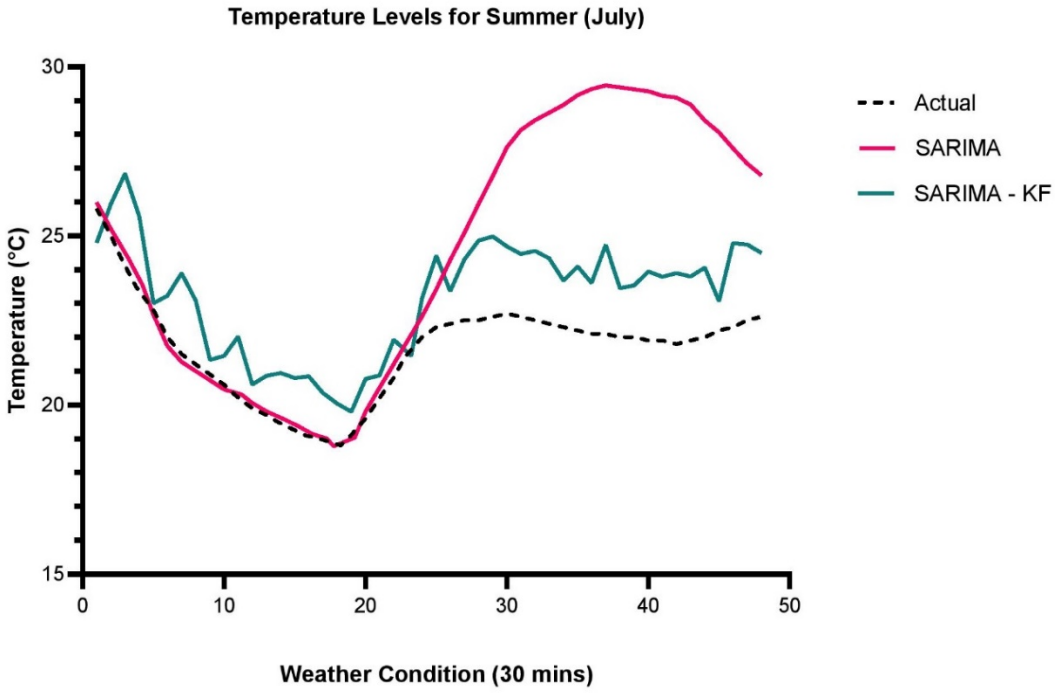
(b) Temperature Actual, SARIMA and SARIM-KF for May 2019

Fig. 2.8-3: May (spring) irradiance and temperature plots

In the semi cloudy day of Figure 2.8-4 (a), SARIMA-KF and the SARIMA have an RMSE of 10.8 and 57.7, respectively. Meaning, the SARIMA-KF reduced the error by 81%; calculated MASE is 0.18, this is close to 0 showing the superior performance of the SARFMA-KF, and its ability to follow sharp changes. In the cloudy day, the RMSE of the SARIMA and SARIMA-KF is 153.9 and 9.8 respectively; MASE of 0.07. This significant deviation illustrates the significance of SARIMA-KF, as it uses the sensor's feedback to overcome random patterns not accounted for in historical data. In the sunny day, The RMSE of the SARIMA-KF is 74% lower than the SARIMA model; MASE is 0.24. In the temperature plot, Figure 2.8-4 (b), the MASE value is 0.54 demonstrating the outperformance of the SARIMA-KF.



(a) Irradiance Actual, SARIMA and SARIM-KF for July 2019



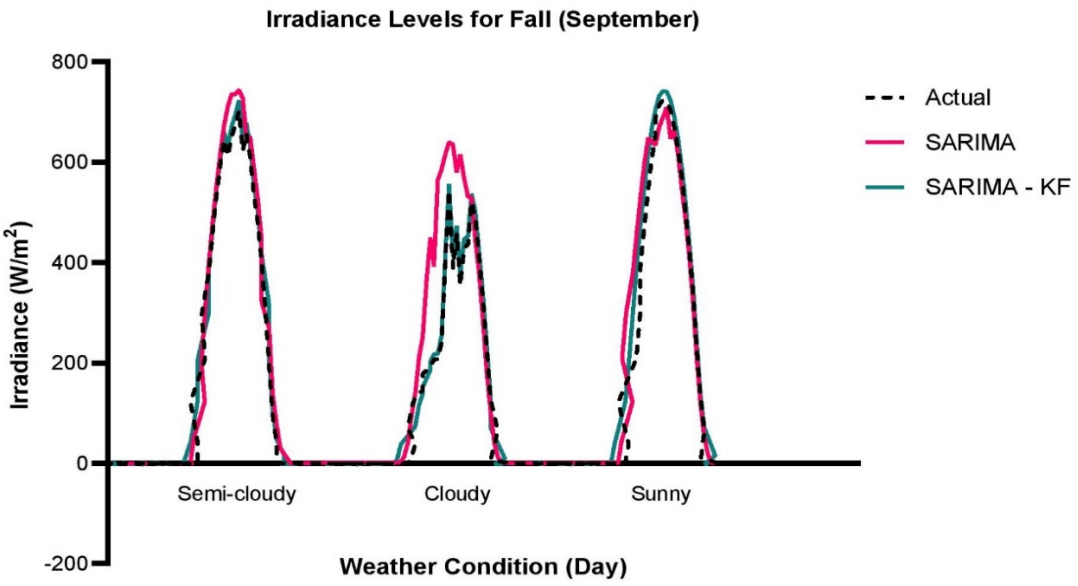
(b) Temperature Actual, SARIMA and SARIM-KF for July 2019

Fig, 2.8-4: July (summer) irradiance and temperature plots

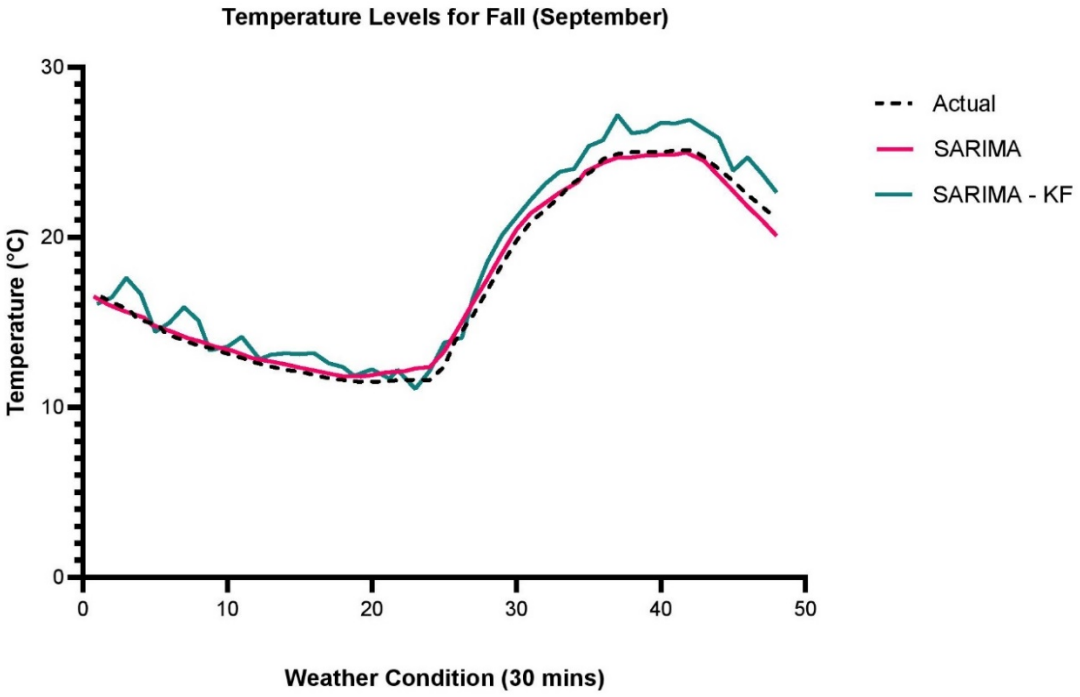
In the semi cloudy day of Figure 2.8-5 (a), SARIMA-KF and SARIMA have an RMSE of 10.7 and 33.6, respectively. This is 68% lower; MASE is 0.37, which demonstrates the outperformance of the SARIMA-KF. In the cloudy day, the RMSE of the SARIMA and SARIMA-KF is 114.7 and 9.02 respectively, with an MASE of 0.1, again this significance of sensor feedback proves its efficacy in tackling the high randomness in cloudy days. In the sunny day, the RMSE of the SARIMA-KF is 34% lower than the SARIMA model, with an MASE of 0.88.

In the temperature plot, Figure 2.8-5 (b), the RMSE of SARIMA and SARIMA-KF is 0.4 and 1.2, respectively. In this rare case, the SARIMA outperforms.

To this end, we conclude that regardless of the sudden changes in irradiance graphs and their deviation from the predicted dynamic model, the SARIMA-KF was able to converge back to the actual values providing us with a better forecasting tool compared to SARIMA. Regarding temperature, the same conclusion stands in most of the cases, regardless of the inaccurate initial guess and the large relative $P_{1,0}^2$.



(a) Irradiance Actual, SARIMA and SARIM-KF for September 2019



(b) Temperature Actual, SARIMA and SARIM-KF for September 2019

Fig. 2.8-5: September (fall) irradiance and temperature plots

To further validate our conclusion, we demonstrate the error metric values, RMSE and MAE for the selected months. The error values were attained by forecasting random points in each day of the month, calculating the error metric values, and then calculating the monthly average (sum of daily errors divided by the number of days in the month) . It is evident from Table II that for irradiance, the SARIMA-KF outperformed the SARIMA model in all instances. The minimum error reduction of 56.6% took place in the month of January whereas, the maximum error reduction of 87.2% took place in July. We also notice that the highest SARIMA RMSE values are in the months of May and July, this is

Table II: Monthly average errors for both forecasting methods

Months	*Model*	*RMSE*	*MAE*
January	Irradiance - SARIMA	40.75	20.15
	Irradiance - SARIMA-KF	17.68	10.72
	Temperature - SARIMA	3.59	3.24
	Temperature-SARIMA-KF	0.90	0.76
May	Irradiance - SARIMA	83.05	53.14
	Irradiance - SARIMA-KF	12.34	8.64
	Temperature - SARIMA	3.76	3.28
	Temperature-SARIMA-KF	3.04	2.79
July	Irradiance - SARIMA	87.11	60.43
	Irradiance - SARIMA-KF	11.14	8.05
	Temperature - SARIMA	2.12	1.56
	Temperature-SARIMA-KF	2.43	1.37
September	Irradiance - SARIMA	54.60	29.68
	Irradiance - SARIMA-KF	9.95	6.66
	Temperature - SARIMA	1.67	1.49
	Temperature-SARIMA-KF	1.26	1.14

because the irradiance pattern of infrequent cloudy days is not captured by the AR and MA historical components, since less cloudy days are present.

For Temperature, it is evident that the SARIMA-KF outperformed the SARIMA model in all instances, except slightly underperforming in the month of July. Both models seem to differ slightly, with an average RMSE deviation of 1.0. The slight deviations in temperature

values compared to irradiance affect the performance of the KF, since the relative variance of the initial guess and the sensor have a greater impact. This can be reduced by using a highly accurate temperature measuring device and a less uncertain initial guess.

2.9 Conclusion

In this chapter, we discussed the different time series prediction models available in the literature. Given the energy, computation, and memeory-limited IoT devices, we concluded that having a combined stochastic (SARIMA) and a state space model (Kalman filter) suits our application since this requires less historic data and no training. We ran the prediction model for semi-cloudy, cloudy, and sunny days for each season of the year, for the city of Ottawa, Canada, and we concluded the superiority of the SARIMA-KF models over the SARIMA. SARIMA-KF outperformed the SARIMA models by 53% and 76% reduced MAE for irradiance and temperature, respectively, in the month of January. It has also reduced MAE by 84% and 12% for irradiance and temperature, respectively, in the month of July.

Chapter 3: Energy Prediction – Solar Energy Harvesting Unit

3.1 Introduction

In this chapter, we utilize both the forecasted irradiance and temperature from Chapter 2, in order to calculate the generated electrical energy that will charge IoT nodes' batteries. This is done with the help of a solar-based EHMU.

In Sections 3.2 to 3.6, the theory behind each component and the system block that represents it in SIMULINK is discussed. In Section 3.6, we discuss the overall system built in SIMULINK. In Sections 3.7, we present our simulation results which include system efficiency, battery SoC (gained from irradiance and temperature values in Chapter 2), and power values for different irradiance inputs. Finally, in Section 3.8, we present our conclusion.

As seen in Figure 3.1-1, these EHMUs consist of photo-voltaic (PV) arrays, DC-DC converters, rechargeable batteries, charge controllers [39], and battery management

systems (BMS). In solar dependent energy harvesting IoT-based WSNs, ambient solar energy is harvested via PV arrays attached to the nodes. Their goal is to convert solar into electrical energy. Then, with the help of a DC-DC converter (Buck), the high output voltage of the PV array is stepped down and regulated to suit the charging requirements of the rechargeable battery.

The perturbation and observation (P&O) based maximum power point tracker (MPPT) tracks both voltage and current outputs of the PV array, to try to match it to the maximum voltage and current possible, given the present irradiance and temperature values. This is done by varying the duty cycle being fed into the metal-oxide semiconductor field-effect transistor (MOSFET) of the buck converter. Lastly, the adjusted output voltage and current

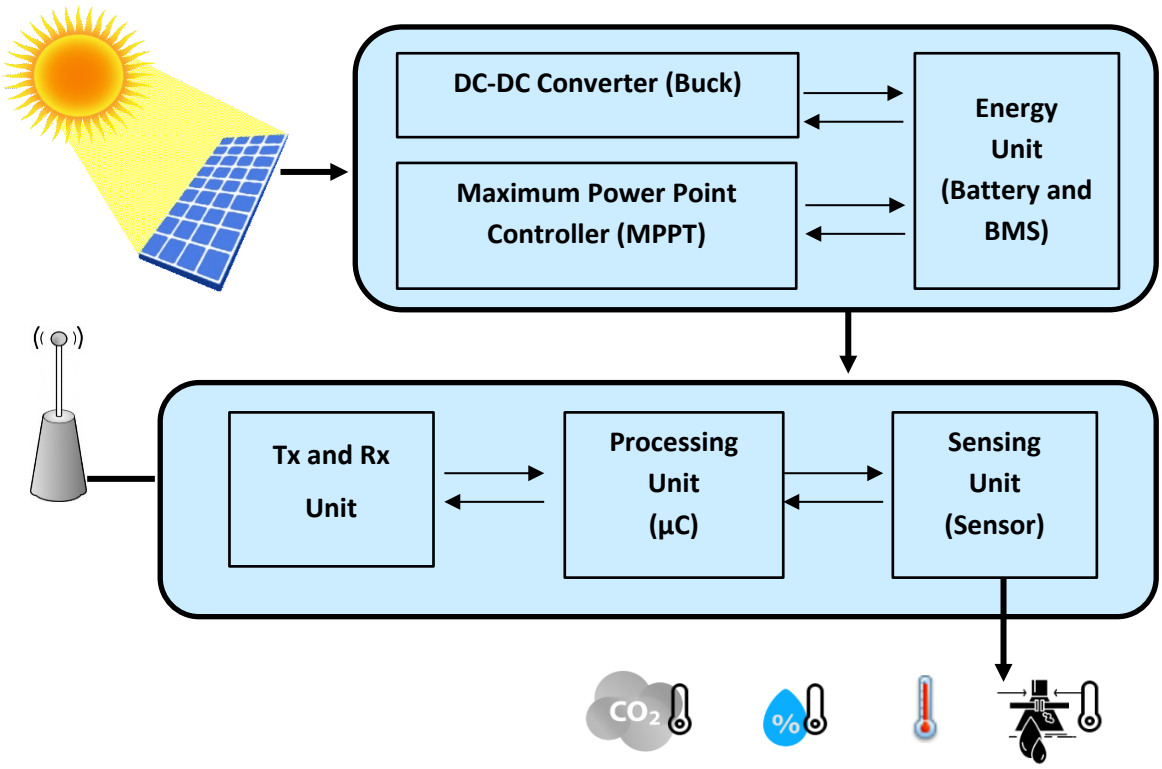


Fig. 3.1-1: Solar dependent energy harvesting IoT nodes

is feed into the battery for storage and use by the sensor node. The functionalities of the sensor node include transmitting (Tx), receiving (Rx), processing, and measuring physical quantities like air quality, moisture, temperature, leaks, etc. That said, the efficiency of the overall system is crucial to optimize, and, thus we model and simulate the EHMU.

3.2 Solar Panel Modeling

A PV module, or array, is made up of multiple solar cells. Solar cells are made up of N and P type semi-conductors with N-P junctions, similar to a diode. Silicon is by far the most common semiconductor material used in solar cells, since it the second most abundant material available on earth, and it is widely used in chips [40]. The crystal lattice formed by the crystalline silicon cells provides an efficient conversion from light to electricity. As the energy of photons E from sunlight ($E = h\frac{c}{\lambda}$ where h is plank constant, c is the speed of light and λ is the wavelength of incident light rays) is greater than excitation energy threshold E_g, the excitation of an electron from the valence band to the conduction band occurs. This excitation leaves a hole in the valence band which behaves as a positive charge, creating an electron-hole pair. This pair acts as a building block of current I_L, and the NP feature of the cell is modeled as a diode with current I_d. This provides us with the generated output current from a solar cell ($I_{sh} + I_s$) as seen in Figure 3.2-1 (b) and Equation (3.2.1) [41].

$$I = I_L - I_d, \text{ where } I_d = I_0\left[exp\left(\frac{V_D}{nV_T}\right) - 1\right], \tag{3.2.1}$$

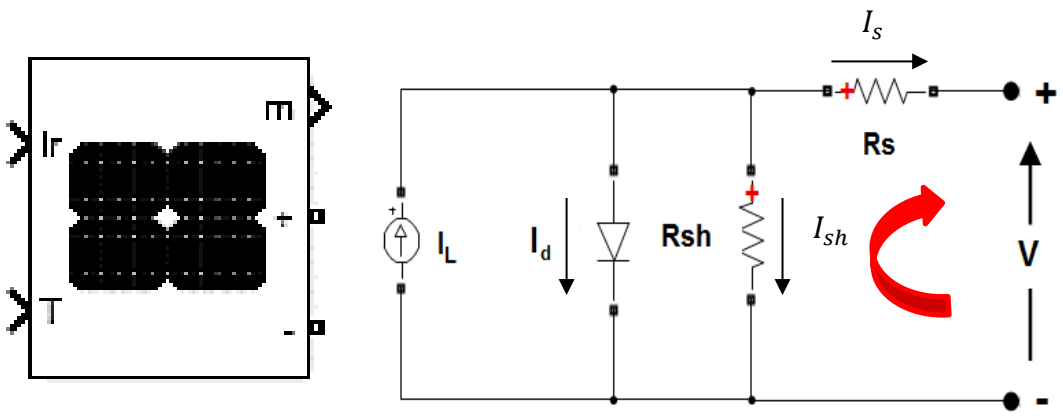


Fig. 3.2-1: Irradiance and temperature curves – Ottawa year 2019

where I is the total output current of the solar cell, I_L is current from the electron hole pair i.e., light generated current source, I_o is the reverse saturation current from recombination, n is the non-ideality factor, and V_T is the thermal voltage V_D, is the voltage across the diode. The actual output (I_{sc}) of the cell after considering losses is I_s, as written in Equation (3.2.2):

$$I_s = I_L - I_d - I_{sh}, \tag{3.2.2}$$

where I_d and I_L are defined in Equation (3.2.1) and the specification sheet provided by the manufacturer, respectively. We just need to identify I_{sh} to obtain the open circuit voltage (V) and the short circuit current (I_s) of the PV module (as seen in Figure 3.2-1 (b)). This is done using Kirchhoff's voltage law in Equation (3.2.3) [42]:

$$I_{sh} = \frac{V_D}{Rsh} = \frac{V+IR_s}{Rsh}. \tag{3.2.3}$$

This provides us with the output values needed for the solar module. As seen in Fig 3.2-1 (a), we use an already built PV Array block which implements an array of PV modules. This block is present in the Simscape library for MATLAB. The array is built of strings of modules connected in parallel. Each string consists of modules connected in series. This

block allows us to model preset PV modules from the National renewable energy laboratory (NREL) System Advisor, as well as user defined PV modules [43].

3.3 Buck Converter (DC-DC) Modeling

The Buck Converter is used in circuits where the direct current (DC) output voltage needs to be lower than the DC input voltage. The DC input can be derived from a rectified alternating current (AC) source or from any DC supply. It is useful where electrical isolation is not needed between the switching circuit and the output [44]. In our case, the buck regulates the DC input voltage from the solar panel to a lower DC voltage to maintain idle power delivery to the battery. The Buck converter consists of a switching transistor (MOSFET), current flow control (D1 – diode), voltage surge smoother (L1 – inductor), and a voltage ripple smoother (C1- capacitor) as shown in Figure 3.3-2.

When the MOSFET is switched on, the output of the PV module is fed into the battery. Initially, the flow of the current is restricted by L1 by building up a magnetic field. This leads to a gradual buildup of current and charge in the battery and C1, respectively. In addition, due to high positive charge at the negative terminal of D1, the diode is in the reversed bias phase stopping the flow of current through it.

When the MOSFET is switched off, the energy stored in L1 in the form of magnetic fields is released gradually as electrical current into the circuit and towards the load. This is because inductors oppose changes by releasing current when the magnetic fields start collapsing. This current flows for at least part of the time the MOSFET is switched off. The diode in this instance is forward biased causing a closed loop between D1, L1, and C1.

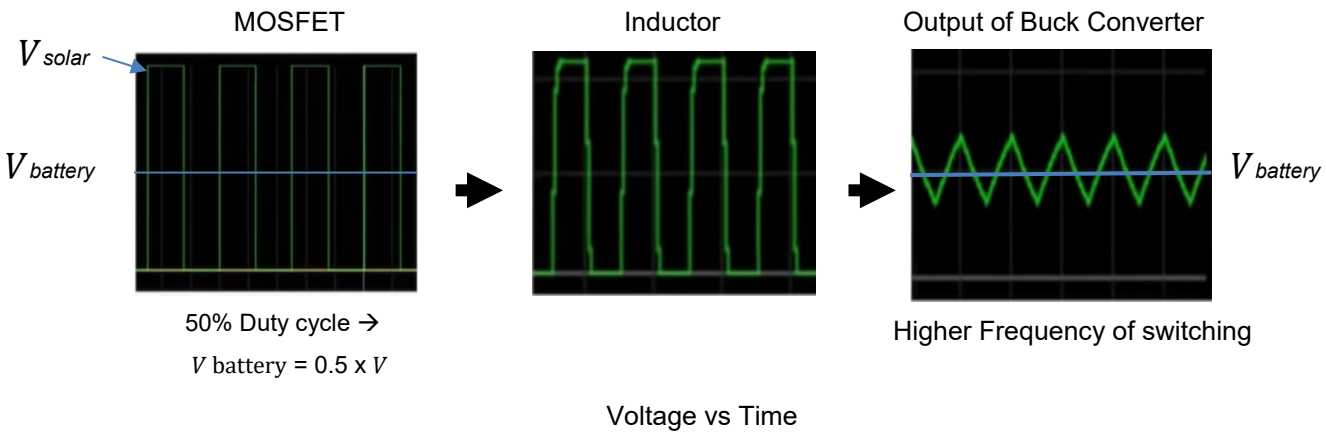


Fig. 3.3-1: Waveforms at MOSFET, Inductor and output of Buck converter

Once L1 releases, a significant part of it stored energy, C1 will start to supply the battery until the MOSFET is switched back on. This ON and OFF action results in a small amplitude, smoothened high frequency triangular wave (DC + ripple in Figure 3.3-1), with an average DC voltage of V_{out} as seen in Equation (3.3.1) [45]:

$$V_{out} = V_{in} \cdot \frac{t_{on}}{T}, \tag{3.3.1}$$

where V_{out} is the output voltage of the buck converter being fed into the battery, V_{in} is the input voltage of the buck converter from the PV panel, t_{on} is the time the MOSFET is ON, and T is the period of the switching waveform. This duty cycle ($\frac{t_{on}}{T}$) of the square switching waveform is dictated by the MPPT controller. It is used to adjust the power output given temperature and irradiance being fed into our solar module.

In our case, we utilize the system blocks provided by the Electrical Simscape Library to build the Buck Converter. As seen in Figure 3.3-2, the blocks in use are a MOSFET, Schottky diode, Capacitor, and an Inductor. For our simulation we set the switching frequency to 5 KHz, the initial duty cycle to 0.4 and the forward voltage to 0.5 V.

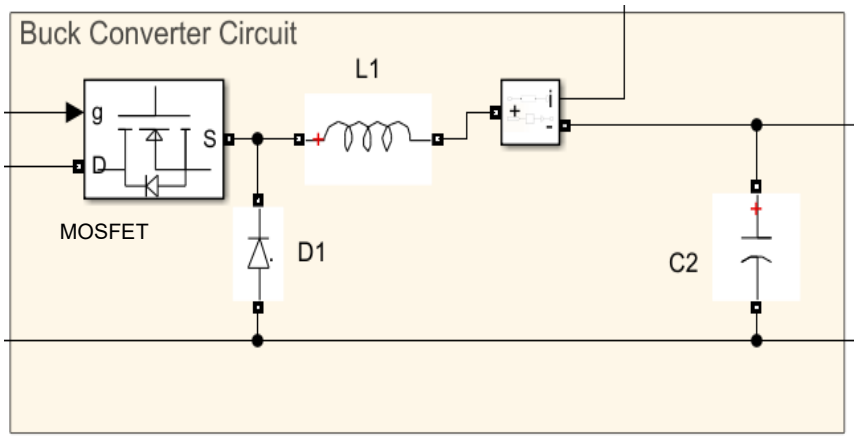


Fig. 3.3-2: Waveforms at MOSFET, Inductor and output of Buck converter

With the given parameters, we can find the values for both the inductor and the capacitor using Equations (3.3.2) and (3.3.3) respectively.

As previously mentioned, there is a ripple component in the output of the Buck converter. The goal is to reduce that ripple by choosing proper capacitance and inductance values. The value of inductance is obtained via Equation (3.3.2) [46]:

$$L = \frac{v_{in}D(1-D)}{f_{switching}\Delta I_L}, \tag{3.3.2}$$

the inductance is a function of v_{in}, the input voltage of the buck converter (20V PV module), D duty cycle (0.4), $f_{switching}$ MOSFET switching frequency (5 KHz), and I_L inductor ripple current (10 mA). The calculated inductance is 100 mH.

Similarly, the output voltage ripple of the Buck converter is a function of capacitance C2 as seen in Equation (3.3.3) [46]:

$$C = \frac{v_{in}D(1-D)}{8Lf_{switching}^2\Delta v_C}. \tag{3.3.3}$$

With the above parameters and the inductance value L of 100 mH the capacitance of C2 is found. The capacitance is 24 µF however, a 22 µF capacitor is chosen given its availability in the market.

3.4 MPPT Modeling

P&O MPPT charge controllers are commonly used in commercial applications. The goal of the MPPT is to maximize the output power of the PV array given varying temperature and irradiance values [47]. This can be achieved by constantly measuring the output voltage and current of the PV array via port *m* in Figure 3.2-1 (a). This allows to calculate the ideal duty cycle *D* that is being fed into the *g* port of the MOSFET. The P&O algorithm is presented in Figure 3.4-1.

This algorithm uses an iterative and trial and error approach. First, both voltage and current readings of the current time step are measured. Second, the output power is calculated by multiplying $V(t)$ and $I(t)$ and is, then, compared to the power output of the previous time

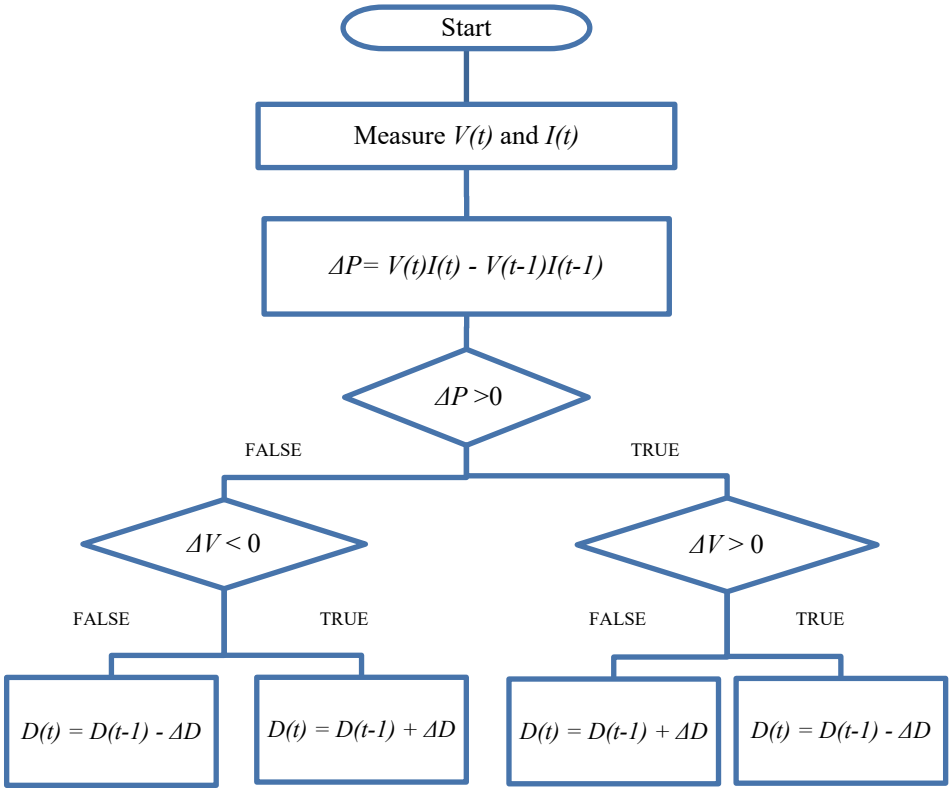


Fig. 3.4-1: P&O flow chart

step. Third, a power comparison is done, to sense the proximity of the maximum achievable power point given the P-V curve of the PV module (Figure 3.4-3 (b)). If the former power value is greater, then the condition is false and vice versa. Fourth, a voltage comparison is performed to locate on which side the of the P-V characteristic graph do we reside. Lastly, the duty cycle being fed into the MOSFET is altered, which in turn changes the input resistance of the Buck converter, resulting in an increased power output. This process repeats until the maximum tracked power output is attained.

For our simulations, we utilized an MPPT solar charger controller model built by Rodeny Tan [47]. This model, shown in Figure 3.4-2, is open source and is available in the MathWorks file exchange library. The logic behind the block diagram is depicted from the MPPT algorithm in Figure 3.4-1. The MPPT component takes the output voltage and current of the PV module block and outputs the power of the PV panel via port “PV Power”, and the duty cycle of the MOSFET via port “PWM”. An abstract view of the MPPT block is seen in Figure 3.4-3 (a). The Battery input port is neglected in our case.

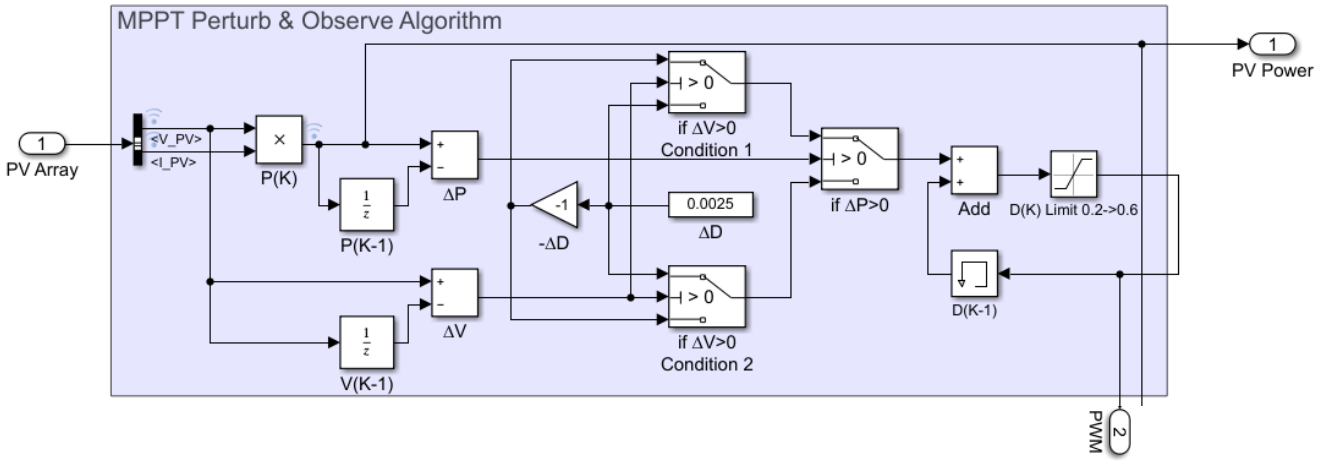
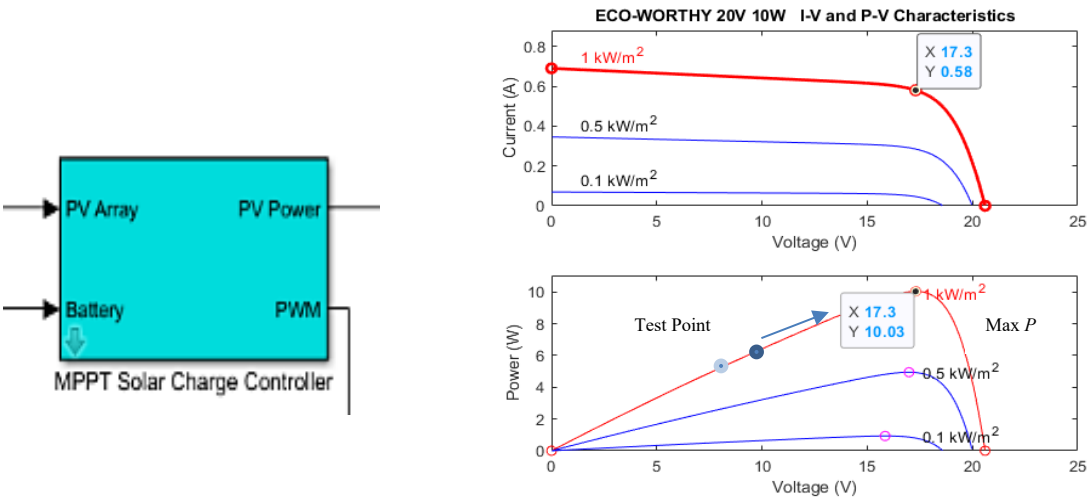


Fig. 3.4-2: P&O block diagram



(a) MPPT Simulink block (b) I-V (top) and P-V (bottom) curves for PV used in simulation

Fig. 3.4-3: MPPT block and Eco-Worthy solar characteristics curves

Here, we plan to walkthrough an example to show the MPPT logic (Figure 3.4-1) and how the implemented block logic (Figure 3.4-2) achieves maximum power, given specific irradiance and temperature values. For the sake of this example, we will use 1000 $\frac{w}{m^2}$ and 25 ˚C (Standard Test Conditions - STC) for irradiance and temperature values, respectively (red curve in Figure 3.4-3 (b)). We begin with the dark blue test point at 6 W i.e., 10 V and 0.6 A, this point lies on the left of the 1000 $\frac{w}{m^2}$ P-V curve and is far from the Max power point at 10.03 W i.e., 17.3 V and 0.56 A. It is intuitive that the MPPT needs to shift our test point to the right to get closer to the maximum point. As seen in Figure 3.4-2, first the voltage and current values of the test point are fed into the *P(K)* block where the product of both variables (power) is obtained. Second, as seen in Figure 3.4-1, we need to attain *ΔP*, this is done by passing the power output through the unit delay block, where the signal is held with one sample period delay. Both delayed and current power signals are then inputted into a sum block for differencing. Third, as seen in Figure 3.4-1, we need to realize if the difference in power is less than or greater than zero, this is done by passing the *ΔP*

signal into a switch block. If ΔP is greater than 0, the signal from the top port/input 1 is passed and vice versa. In our case, as seen in Figure 3.4-3 (b), the new power value is greater, since the power of the dark blue point (current P) is greater than the power of the light blue point (previous P), therefore input 1 is passed. Fourth, as seen in Figure 3.4-1, we need to realize if the difference in voltage is less than or greater than zero, this is done by passing the ΔV into the switch block for condition 1 i.e., when $\Delta P > 0$ is true. If ΔV is greater than 0 the $-\Delta D$ signal is passed, whereas if ΔV is less than 0 the ΔD signal is passed. In our case the voltage of the current signal is greater than the delayed one i.e., $\Delta V > 0$ is true, therefore port 1 is allowed through. Lastly, the ΔD is inputted into the sum block where the previous duty value cycle is reduced by ΔD i.e., 0.0025. This reduction of the duty cycle results in a greater PV power output V_{in}, since D is inversely proportional to V_{in} as seen in Equation (3.3.1). This process then continuously reiterates until the value of V_{in} equals the maximum point.

3.5 Battery Modeling

Nowadays various types of batteries are manufactured to fit a wide range of applications. This diversity of manufactured batteries is expanded to meet the demand of electrical vehicles, portable electronics and devices with renewable energy harvesting capabilities [48]. In our battery module, electrical energy from the output of the Buck converter is converted into chemical energy for storage.

In Figure 3.5-1 [49], The first segment presents the exponential voltage drop when the

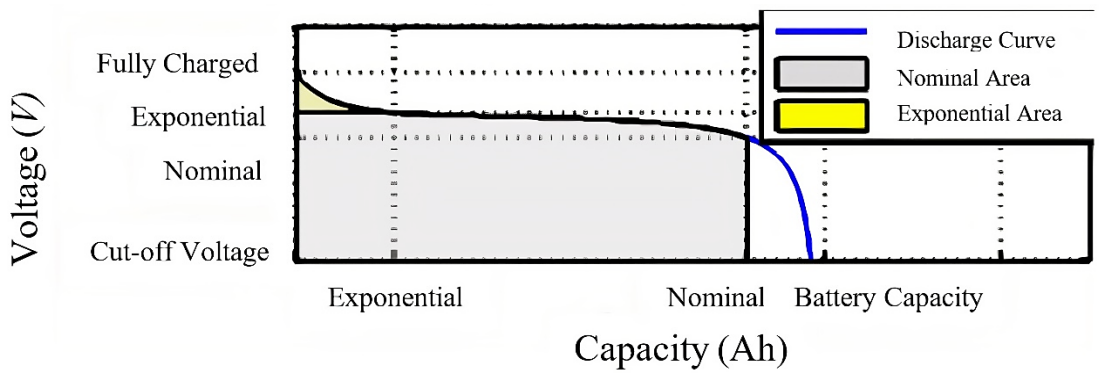


Fig. 3.5-1: Discharge pattern of a generic battery

battery is charged. The width of the drop depends on the battery type. The second segment presents the charge that can be extracted from the battery until the voltage drops below the battery nominal voltage. Finally, the third segment presents the total discharge of the battery, when the voltage drops rapidly [49]. It is visible that the bulk of the discharge is in segment two of the graph, this makes it easier to model the battery with reasonably large accuracy. Depending on the type of batteries, we can leverage price for battery capacities. The two most known types of battery are Lead acid and Lithium-ion (Li-Ion) batteries. Lead acid batteries are the oldest and most mature technology used in energy storage for power electronics. The main advantage of lead-acid battery is the low cost compared to other types. Li-Ion batteries, on the other hand, are light weight and eight times more energy dense compared to Lead acid batteries. This makes them more appropriate for our application. Their only downfall is their high price tag and the lifetime degradation due to deep discharges [50].

The most used battery model is seen in Figure 3.5-2 [51]. This model consists of a DC voltage source and a fixed internal resistance. This model is independent of the SoC of the battery therefore, it just captures the middle section in Figure 3.5-1 i.e., 80% of the curve.

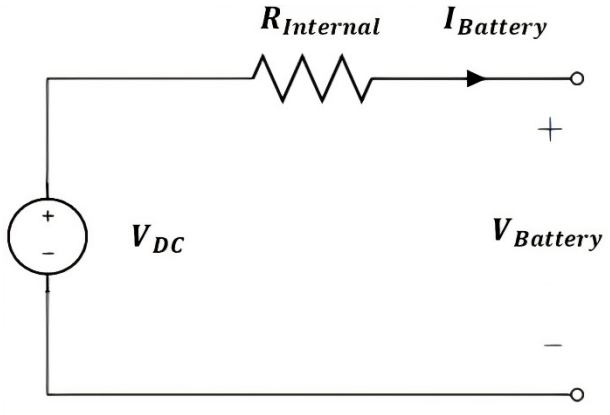


Fig 3.5-2: Simple linear battery model

Since the model in Figure 3.5-2 does not capture the exponential drop areas, a better model must be presented. In our case, a generic battery model available in MathWorks was used [49]. This model as seen in Figure 3.5-3 [49], uses Equation (3.5.1) to model the discharge curve. This equation accounts for the varying voltage curves in section one and three of Figure 3.5-1.

$$f_1(i_t, i^*) = E_0 - k\frac{Q}{Q-i_t}\cdot i^* - k\frac{Q}{Q-i_t}i_t + A\cdot exp(-B\cdot i_t), \qquad (3.5.1)$$

where E_0 is the constant voltage component (V), k is the polarization resistance (Ω) i.e., the resistance that occurs due to oxidation when voltage potential is applied, i^* is the low frequency current dynamics (A), i_t is the extracted capacity (Ah), Q is the maximum battery capacity (Ah), A is the exponential voltage (V) and B is the exponential capacity (Ah^{-1}). This equation is used as a control signal to the voltage source. Whenever $i^* > 0$, the battery is in the discharge stage, since the current is flowing away from controlled DC voltage source [52] as seen in Figure 3.5-3. The second and third terms in Equation (3.5.1) account for the polarization voltage from the low frequency current and the total charge from time 0 to t (extracted capacity in Equation (3.5.2)). Both the second and third terms

$$i_t = Q_{Start} - \int_0^t I_{Batt}\,dt, \quad \text{where } Q_{Start} \text{ is intial charge,} \qquad (3.5.2)$$

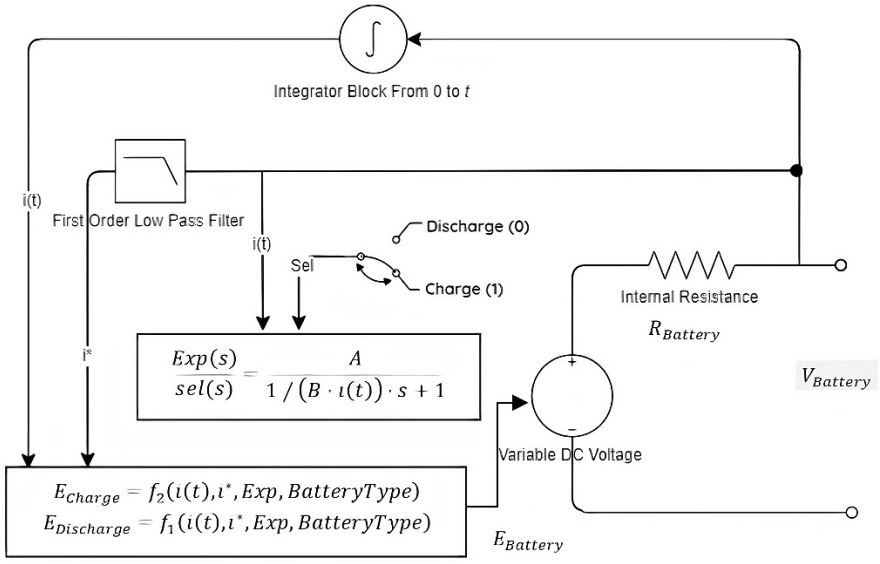


Fig. 3.5-3: Equivalent block diagram of the generic battery model

affect severity of exponential drop in segments 1 and 3. The fourth term in Equation (3.5.1) models the exponential drop in segment one in Figure 3.5-1. The charging pattern of the generic battery model is the inverse of the discharge curve, initially from the empty battery state, through a period of rapid establishment of the nominal voltage, followed by a period of charging at nominal voltage, and finally the exponential area when the no-load voltage is restored. The charging stage is modeled by Equation (3.5.3):

$$f_1(i_t, i^*) = E_0 - k\frac{Q}{i_t + 0.1Q} \cdot i^* - k\frac{Q}{Q - i_t} i_t + A \cdot exp(-B \cdot i_t), \qquad (3.5.3)$$

since the charging current has an opposite sign, i.e., $i^* < 0$, the resistance of polarization is changing, so the function of charging voltage is slightly different [52].

Now we need to approximate the variables *A, B, k* and E_0 to substitute them into Equations (3.5.2) and (3.5.3). This is done by utilizing certain parameters in the specification sheet provided by the manufacturer. All battery datasheets include discharge curves similar to Figure 3.5-1. We use the nominal, fully charged, exponential and max points to derive the

mentioned variable. First we start with A the exponential voltage (V). It can be approximated using Equation (3.5.4) [51]:

$$A = E_{full} - E_{exp}, \tag{3.5.4}$$

where E_{full} and E_{exp} are equivalent to the fully charged voltage value and exponential voltage value respectively, as seen in Figure 3.5-1. B on the other hand can be deduced using Equation (3.5.5) [51], where $Qexp$ is the capacity in (Ah) at the exponential value.

$$B = \frac{3}{Qexp}. \tag{3.5.5}$$

The polarization resistance k is calculated using Equation (3.5.6) [51], where Q_{Nom} is equivalent to the capacity in (Ah) at the Nom value in Figure 3.2-1. Q_{Rated} is the rated capacity of the battery (e.g. 5V 10Ah battery where 10Ah is the rated capacity).

$$k = \frac{\left(E_{full} + A(exp - BQ_{Nom})\right)(Q_{Rated} - Q_{Nom})}{Q_{Nom}}. \tag{3.5.6}$$

Finally, the constant voltage component E_0 is calculated using Equation (3.5.7) [51], where $R_{internal}$ is the internal resistance provided by the manufacturer.

$$E_0 = E_{full} + k + R_{internal}I_{battery} - A. \tag{3.5.7}$$

3.6 Complete Simulation Model

After modeling the main components of the EHMU, we have built a complete model on SIMULINK, as shown in Figure 3.6-1. Both dataset irradiance and temperature values are imported from the MATLAB workspace and are inputted into the PV module. The output of the PV module is then passed to the Buck converter which in turn is connected to a battery model. The top blue block is the MPPT controller, it takes the input from the PV

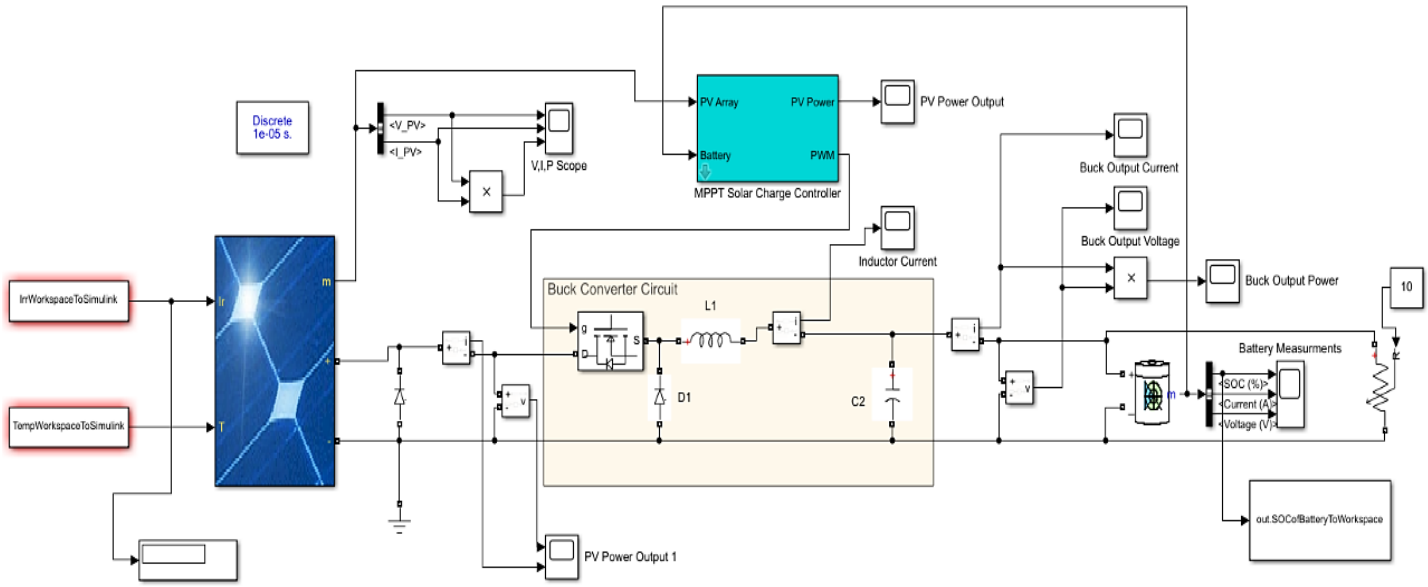


Figure 3.6-1: Energy harvesting management unit SIMULINK model

module and outputs the duty cycle to the MOSFET. Lastly, the charging efficiency and output variables are recorded and sent back to the workspace for use in Chapter 3.

3.7 EHMU Simulation Parameters and Results

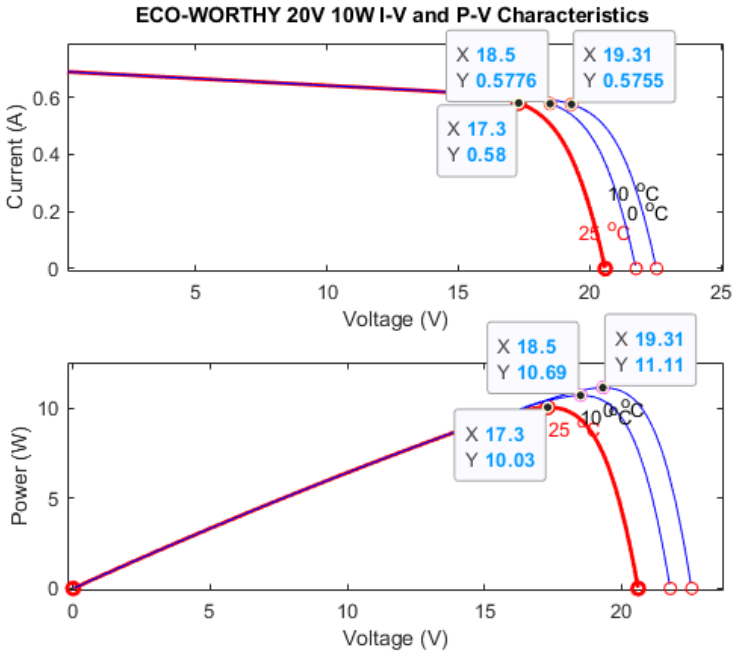
We initialize this section by presenting the simulation parameters in Table III. As seen in the table, the parameters for the three main units, PV Panel, Buck converter and the battery are given. We begin with the PV panel. Our goal is to mimic a real-world implementation as best as we can therefore, we chose a viable product that is sold by various sellers including Amazon. The product chosen is the 800 cm^2 20V 10W ECO-WORTHY solar panel. The area of this panel complies with the size restriction of IoT sensor nodes. Under standard test conditions i.e., 1000 $\frac{w}{m^2}$ and 25 ˚C, it has a maximum power of 10.03 W and a DC voltage of 17.3 V. The Buck Converter consists of a capacitor with 22 µF capacitance, an inductor with 100 mH inductance and a MOSFET with an initial duty cycle and

switching frequency of 0.4 and 5 KHz, respectively. Again, these components are readily available. Lastly, the Li-ion battery's specifications were deduced from a 5V 10Ah Li-Ion battery pack available on Sparkfun and various other providers. The fully charged, exponential, nominal and cut off voltages are 5.44 V, 5.09 V, 5 V, and 4.2 V, respectively.

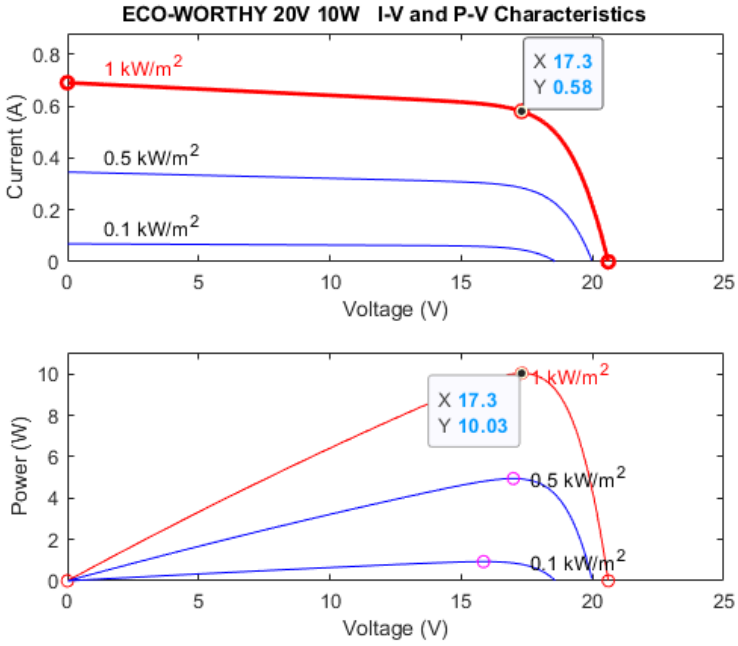
Table III: Simulation parameters

Unit	Parameter	Value	Unit	Parameter	Value
PV Panel	Maximum Power	10.03 W	**Battery**	Nominal Voltage	5 V
	Open Circuit Voltage	20.6 V		Rated Capacity	10 Ah
	Voltage at Max. Power Point	17.3 V		Maximum Capacity	10.42 Ah
	Short Circuit Current	0.69 A		Cut-off Voltage	4.2 V
	Current at Max. Power Point	0.58 A		Fully Charged Voltage	5.44 V
Buck Converter	Capacitance of C2	22 µF		Nominal Discharge Current	2 A
	Inductance of L1	100 mH		Internal Resistance	0.005 Ω
	Initial Duty Cycle	0.4		Capacity at Nom	3.2 Ah
	MOSFET $f_{switching}$	5 KHz		Exponential Zone	5.09 V and 0.49 Ah

In what follows, we present the graphical illustrations attained after the simulation was carried out. We first start with Figure 3.7-1 which demonstrates the current versus Voltage curves (I-V) and the Power versus Voltage curves (P-V) for the ECO WORHTY PV panel. Figure 3.7-1 (a) indicates the I-V and P-V curves for three different temperatures, 25 ˚C, 10 ˚C and 0 ˚C. It is clear that as the temperature decreases the P-V curves shift to the right and up indicating higher attainable output power and voltages with steady current outputs. Note, the absence of both P-V and I-V curves for negative temperatures. This is due to the plotting limitations of the SIMULINK model. However, it is quite intuitive to deduce that



(a) P-V and I-V Curves for 25, 10 and 0 ˚C



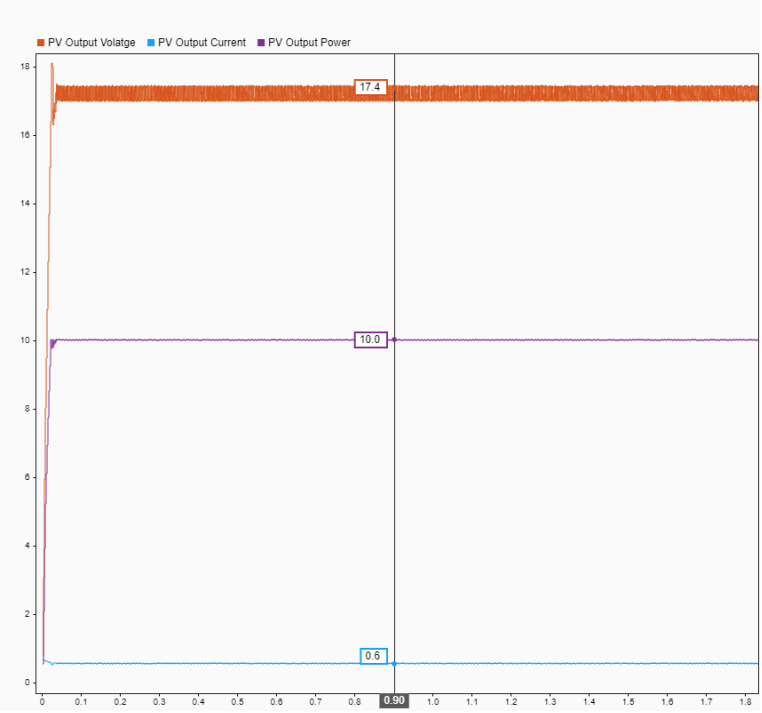
(b) P-V and I-V Curves for 1000, 500 and 100 $\frac{w}{m^2}$

Fig. 3.7-1: ECO WORHTY 10W 20V I-V and PV characterization curves

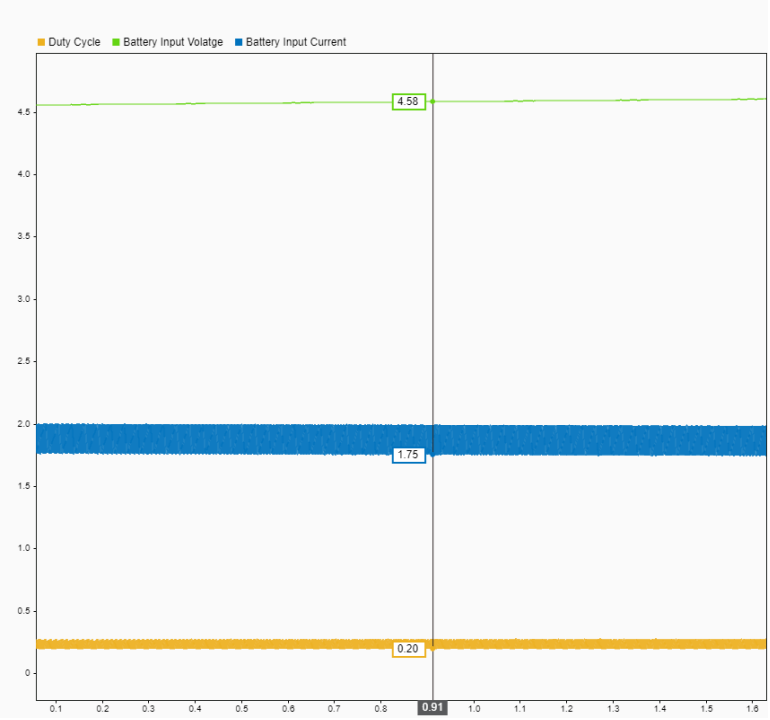
the lower the temperature, the higher the output power and voltage achieved. Furthermore, the power values, as shown by the labels, reveal that the maximum output power provided in Table III (10.03 W) is exceeded as the temperature decreases. This is due to the fact that the maximum power provided by the supplier is at 25 ˚C. In conclusion, we can see that, temperature reduces linearly to the output power. Figure 3.7-1 (b) indicates the I-V and P-V curves for three different irradiances 1000 $\frac{w}{m^2}$, 500 $\frac{w}{m^2}$ and 100 $\frac{w}{m^2}$. It is clear and intuitive that as the irradiance increases the output power and the current increases as well, since more electron-hole pairs are created. The maximum power point at 1000 $\frac{w}{m^2}$ is 10.03 W and its corresponding voltage is 17.3 V, as presented in Table III . With the help of Fig. 3.7-1, we can conclude that, with lower temperatures and higher irradiance values, the efficiency of the PV panel increases.

3.7.1 Varying Irradiance at 25 ˚C

Figure 3.7-2 shows the electrical metric values of the utilized PV panel. We used the standard test conditions (STC) i.e., 1000 $\frac{w}{m^2}$ and 25 ˚C, as used by the manufacturers to test our EHMU circuit. From the PV electrical metrics (Figure 3.7-2 (a)), it is evident that the panel module was able to output the full rated capacity of 10 W as shown by the purple curve. We can also see that an output current of 0.6 A was drawn from the panel. This is between the working rated current and the maximum attainable current of 0.58 A and 0.69 A, respectively. Finally, the measured output voltage of the panel ranges from 17 to 17.4 V, this is 0.1 to 0.3 V away from the maximum voltage attainable at STC, which is 17.3 V.



(a) Solar module output under STC



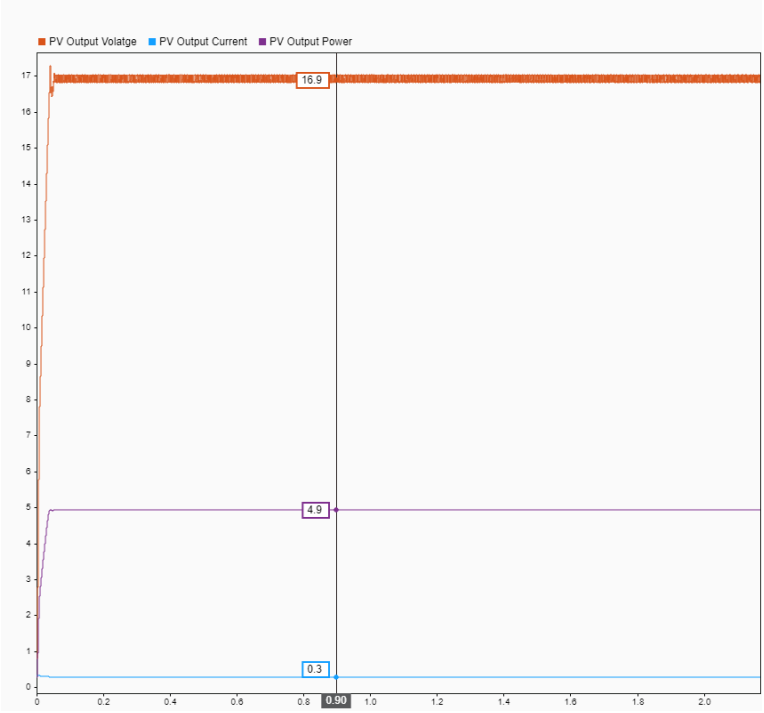
(b) Battery module input under STC

Fig. 3.7-2: Electrical metrics for both PV panel and battery at 1000 $\frac{w}{m^2}$ and 25 ˚C

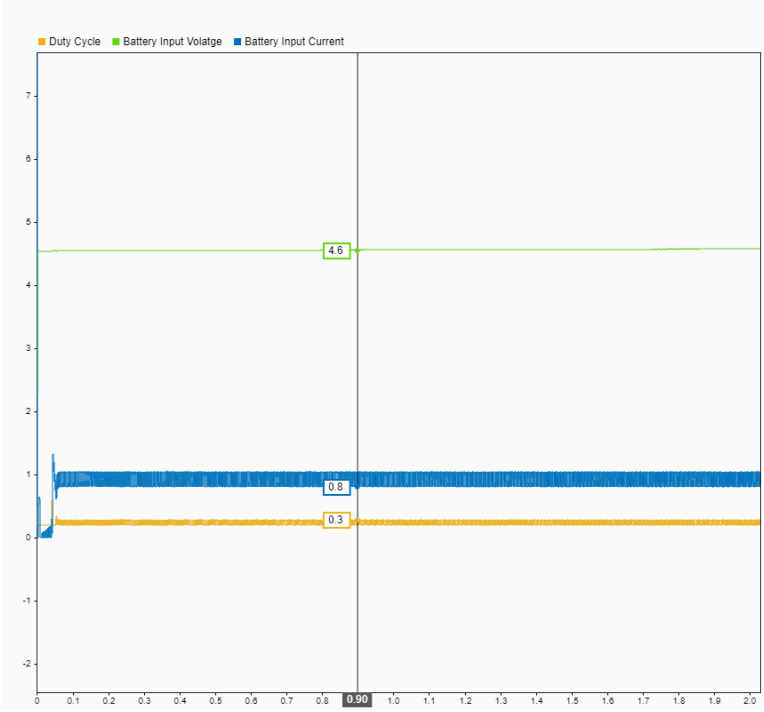
Figure 3.7-2 (b) shows the electrical metric values of the load. It is evident that the output voltage of the buck converter being fed into the battery is 4.58 V, this aligns with the 4.2 V cutoff voltage since the battery's initial state charge was 0% . We can also see that the input current of the battery ranges from 1.75 to 2 A. This lies between the standard and rapid nominal charging current of 1 A and 2 A, respectively. Finally, we can calculate the charging efficiency of the system by dividing the output power of the PV panel and the input power of the battery. Under STC, our charging efficiency ranges from 82 to 91%. This shows that a maximum of 18% and 9% of the input power was lost due to electrical conversion.

Figure 3.7-3 shows the electrical metric values of the utilized PV panel. Since 75% of the data lies below 500 $\frac{w}{m^2}$ and it acts as a threshold line, as seen in Figure 2.8-1, we used 500 $\frac{w}{m^2}$ and 25 ˚C as our next testing conditions. From the PV electrical metrics in Figure 3.7-3 (a), the output power of the PV module shown by the purple line is 4.9 W. This corresponds to the peak power point of the 0.5 $\frac{kw}{m^2}$ curve in Figure 3.7-1 (b). We can also see that an output current of 0.3 A was drawn from the panel. Under the mentioned conditions, this is in between the working rated current and maximum attainable current of 0.3 A and 0.34 A, respectively. Finally, the measured output voltage of the panel ranges from 16.9 to 17 V, this is only 0 to 0.1 V shy of the voltage at the maximum point, which is 16.98 V.

Figure 3.7-3 (b) shows the electrical metric values of the load. It is evident that the output voltage of the buck converter being fed into the battery is 4.6 V, again this aligns with the 4.2 V cutoff voltage since we started with a 0% state of charge. We can also see that the input current of the battery ranges from 0.8 to 1 A, this is 0 to 0.2 A away from the nominal



(a) Solar module output measurments



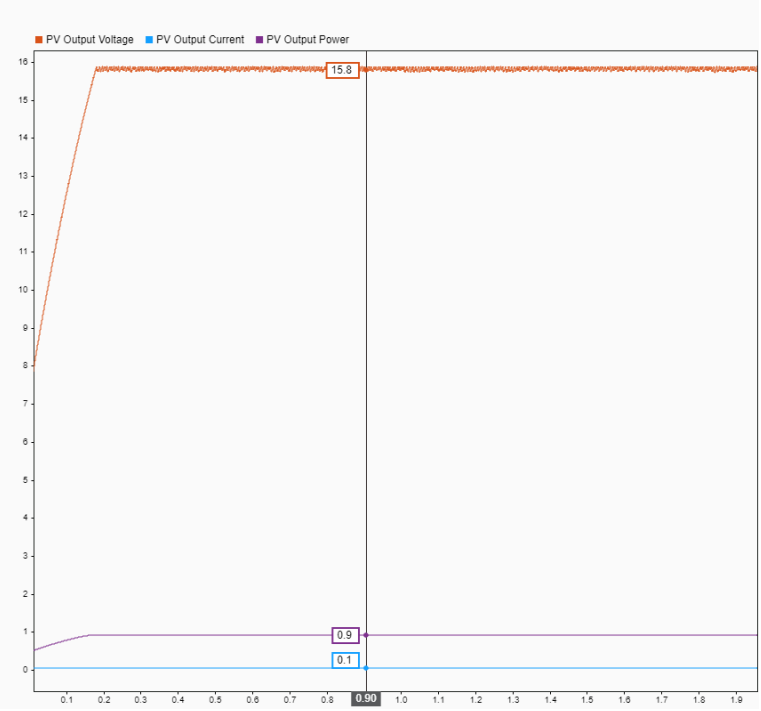
(b) Battery module input measurments

Fig. 3.7-3: Electrical metrics for both PV panel and battery at 500 $\frac{w}{m^2}$ and 25 ˚C

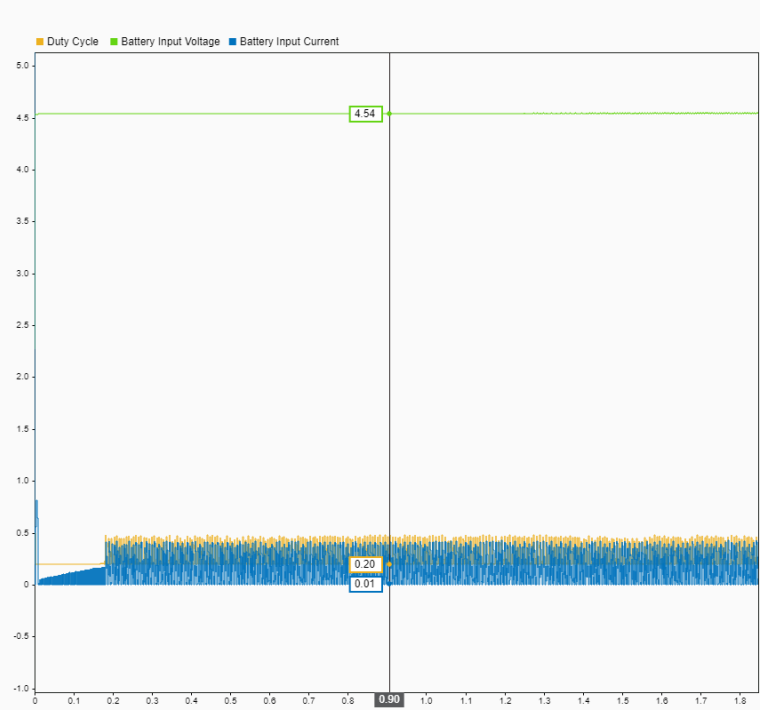
charging current of 1 A. This deviation is due to the reduced amount of irradiance captured by the PV panel. Finally, the charging efficiency under these conditions is between 75 and 94%. This shows that a maximum of 25% and 6% of the input power was lost due to electrical conversion.

Figure 3.7-4 shows the electrical metric values of the utilized PV panel. Since three quarters of the data lie below 500 $\frac{w}{m^2}$ as previously mentioned, we need to simulate the functionality of the system under low irradiance values so, we chose to test our model at 100 $\frac{w}{m^2}$ and 25 ˚C. From the PV electrical metrics in Figure 3.7-4 (a), the output power of the PV module shown by the purple line is 0.9 W. This corresponds to the peak power point of the 0.1 $\frac{kw}{m^2}$ curve in Figure 3.7-1 (b). We can also see that an output current of 0.059 A was drawn from the panel. This is between the working rated current and the maximum attainable current of 0.058 A and 0.070 A, respectively. Finally, the measured output voltage of the panel ranges from 15.8 to 15.9 V, this is only 0.05 V away from the voltage at the maximum point, which is 15.85 V.

Figure 3.7-4 (b) shows the electrical metric values at the battery side. It is evident that the output voltage of the buck converter is 4.54 V. This aligns with the 4.2 V cutoff voltage since we started with a 0% state of charge. We can also see the input current of the battery ranging from 0.01 to 0.45 A, this is 0.55 to 0.99 A away from the nominal charging current of 1 A. This deviation is due to the minimal irradiance captured by the PV panel. Finally, the charging efficiency between 5% and 225%. The values are illogical. This is caused by the huge current fluctuation on the battery side. One reason could be the low output power of the PV attained from irradiance of 100 $\frac{w}{m^2}$.



(a) Solar module output measurments



(b) Battery module input measurments

Fig. 3.7-4: Electrical metrics for both PV panel and battery at 100 $\frac{w}{m^2}$ and 25 ˚C

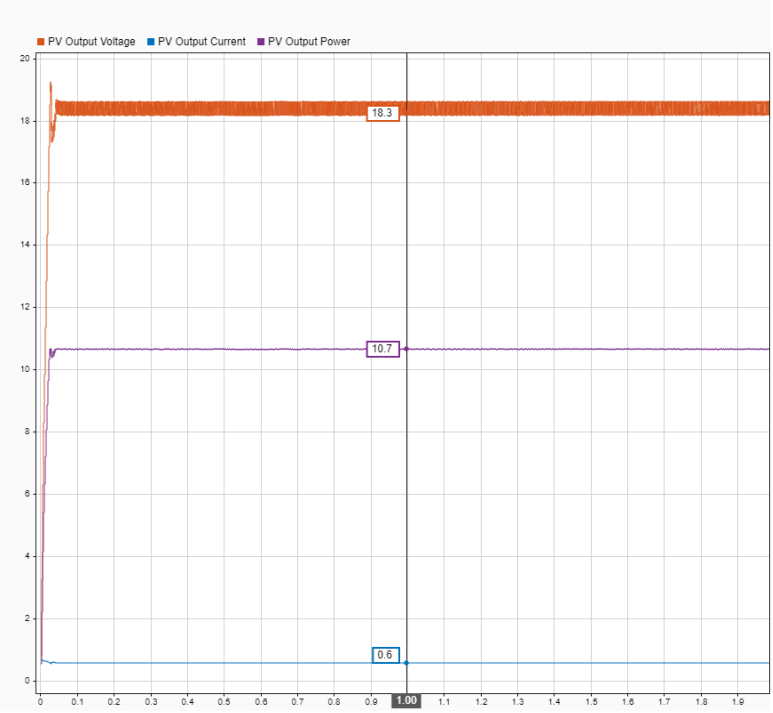
We can conclude from the previous simulations that our system performs as expected. From capturing the maximum power points with the help of the MPPT, to maintaining the correct charging voltage of the battery by manipulating the duty cycle of the Buck converter. Now, it is crucial to test the system under varying temperature, to see how it performs.

3.7.2 Varying Temperature at 1000 $\frac{w}{m^2}$

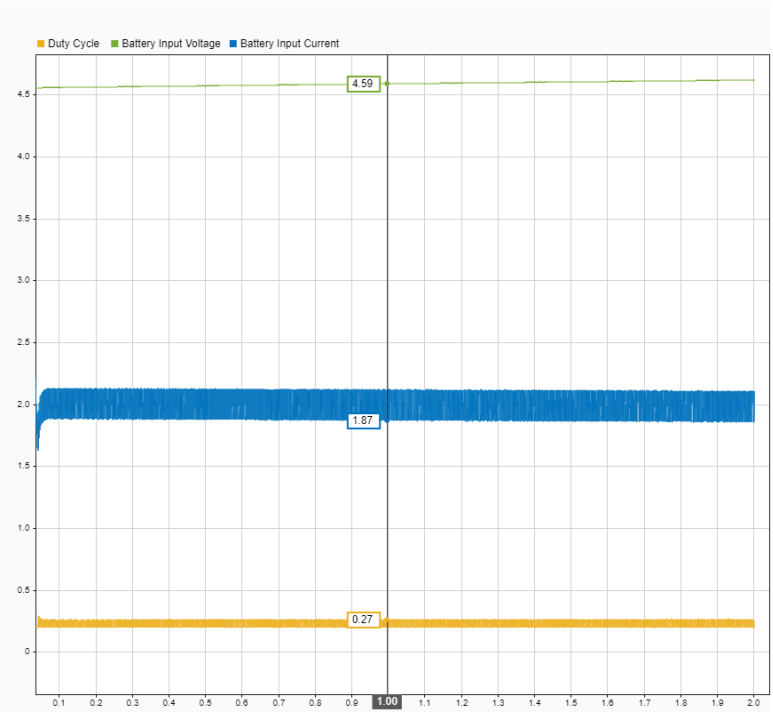
As discussed earlier, as the temperature decreases the P-V curve shifts to the right and upwards resulting in a higher maximum power point. We previously presented the results for the STC. In this section, we plan to present two cases where first, the input temperature is the yearly average temperature (10 ˚C), and second, the input temperature is 0 ˚C.

Figure 3.7-5 shows the electrical metric values of the utilized PV panel at 1000 $\frac{w}{m^2}$ and 10 ˚C. From the PV electrical metrics in Figure 3.7-5 (a), the output power of the PV module shown by the purple line is 10.7 W. This corresponds to the peak power point of the 10 ˚C P-V curve in Figure 3.7-1 (a). We can also see that an output current of 0.6 A was drawn from the panel. This value is between the working rated current and maximum attainable current of 0.58 A and 0.69 A, respectively. Finally, the measured output voltage of the panel ranges from 18.3 to 18.5V, this is only 0 to 0.2V away from the voltage at the maximum point, which is 18.5 V.

Figure 3.7-5 (b) shows the electrical metric values of the battery. The output voltage of the Buck converter is 4.54 V, which aligns with the 4.2 V cutoff voltage of the battery. We can



(a) Solar module output measurments



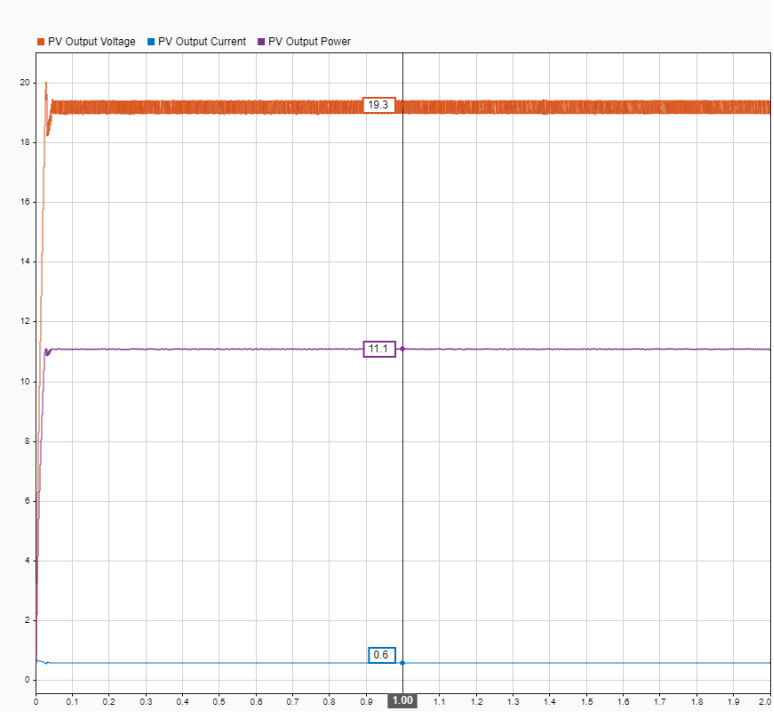
(b) Battery module input measurments

Fig. 3.7-5: Electrical metrics for both PV panel and battery at 1000 $\frac{w}{m^2}$ and 10 ˚C

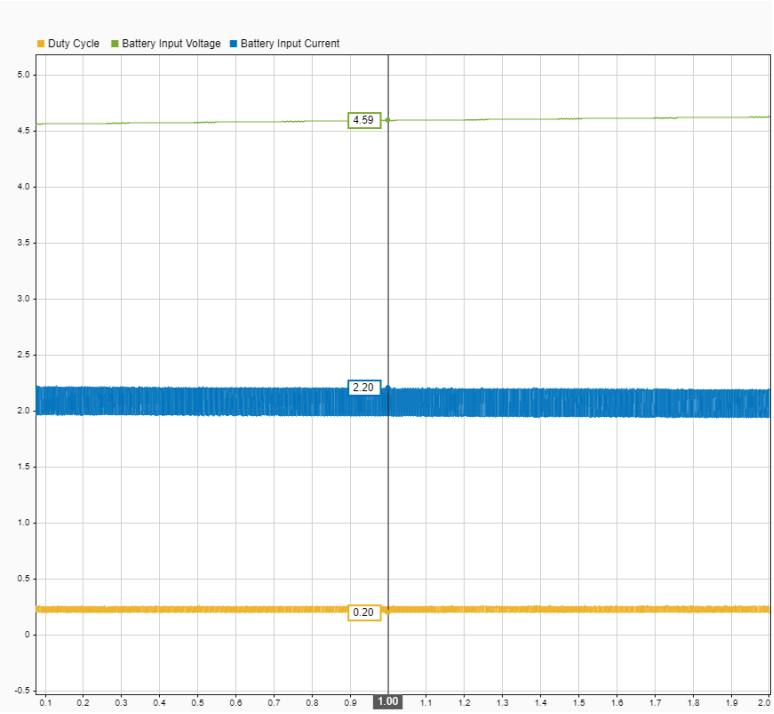
also see that the input current of the battery ranges from 1.87 to 2.1 A, this lies between and exceeds the standard and rapid nominal charging current of 1 A and 2 A, respectively. Finally, the charging efficiency under these conditions is between 78% and 89%. This shows that a maximum of 22% and 11% of the input power was lost due to electrical conversion.

Figure 3.7-6 shows the electrical metric values of the utilized PV panel under 1000 $\frac{w}{m^2}$ and 0 ˚C . From the PV electrical metrics in Figure 3.7-6 (a), the output power of the PV module shown by the purple line is 11.1 W. This corresponds to the peak power point of the 0˚C P-V curve in Figure 3.7-1 (a). We can also see that an output current of 0.6 A was drawn from the panel. Under the mentioned conditions, this is between the working rated current and the maximum attainable current of 0.58 A and 0.69 A, respectively. Finally, the measured output voltage of the panel ranges from 19 to 19.3 V, this is only 0 to 0.3V shy of the voltage at the maximum point, which is 19.31 V.

Figure 3.7-6 (b) shows the electrical metric values of the battery. It is evident that the output voltage of the buck converter is 4.59 V, again this aligns with the 4.2 V cutoff voltage since we started with a 0% state of charge. We can also see that the input current of the battery ranges from 2 to 2.2 A, which exceeds the standard and rapid nominal charging current range of 1 A and 2 A, respectively. Finally, the charging efficiency under these conditions is between 79% and 87%. This shows that a maximum of 21% and 3% of the input power was lost due to electrical conversion.



(a) Solar module output measurments



(b) Battery module input measurments

Fig. 3.7-6: Electrical metrics for both PV panel and battery at 1000$\frac{w}{m^2}$ and 0 ˚C

3.7.3 SoC Results During Different Months

Up to this point, under changing irradiance and temperature, we simulated the output and input of both PV panels and battery, respectively. The next step is to run the simulation for different days in both the months of January 2019 (low irradiance and temperature) and July 2019 (high Irradiance and temperature). In this simulation, we aim to anticipate the SoC of the 5V 10AH battery and study its variability given the changing cloud opacity of the selected days. Figure 3.7-7, the SoC graph shows the SoC in percentage against the time in minutes after the sunrise takes place. We ran the simulation past the peak sun hours (9:00 am till 3 pm) until no further irradiance readings were detected in the dataset from Chapter 2. The plot confirms that the sunny days outperform both the semi-cloudy and cloudy days in both January and July. We also note that the linear gains in the SoC take place during the peak sun hours i.e., between 110 and 470 minutes for the month of July

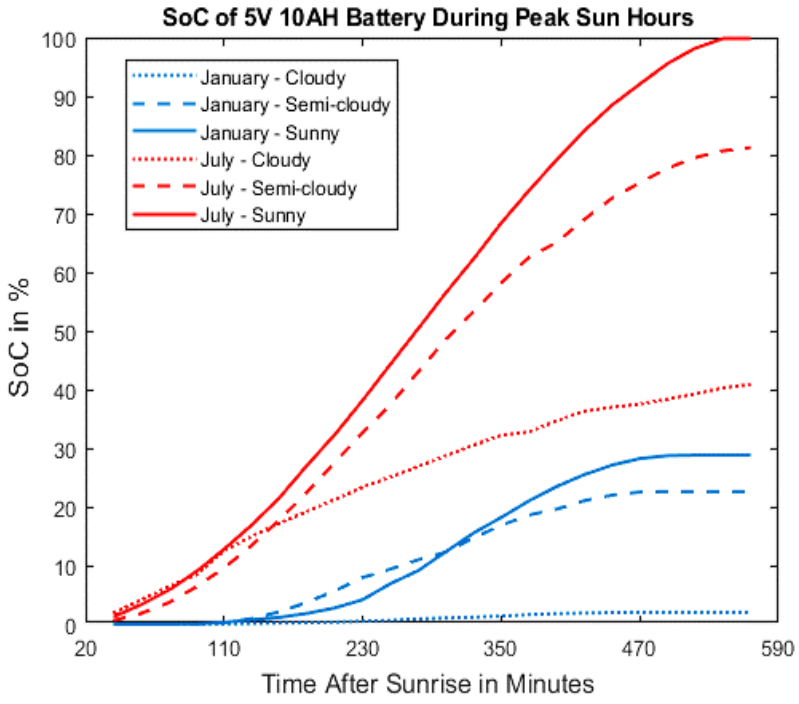


Fig. 3.7-7: SoC of 5V 10AH battery during different months

and between 230 and 450 for January. In July, the irradiance from the sunny day was enough to fully charge a depleted battery in just 570 minutes (9.5 hours) from sunrise, whereas in the semi cloudy and cloudy day, the maximum charge level reached was 80% and 40%, respectively. In January, the maximum reached charge level was 29% and the lowest was 2%. This indicates the importance of another energy source during the off-peak, winter and cloudy days for node operation.

3.8 Conclusion

In this chapter, we proposed an efficient and optimized model for IoT-based solar-EHMU. We have achieved a maximum charging efficiency of 91% by selecting the most suitable solar array and battery according to their power specifications, and by reducing the energy losses. All of these measures allowed us to reach 97-99% of the maximum achievable power. We also deduced that high irradiance (≥1000 $\frac{w}{m^2}$) with low temperatures (<25 ˚C) lead to higher output power compared to the results obtained under standard test conditions. Nevertheless, very low temperatures will negatively affect the health of the battery while charging, and thus reduce the charging efficiency. Lastly, with the help of the simulation, we were able to anticipate the variability in the SoC of the battery under different weather conditions within different seasons.

Chapter 4: Energy, Task Dependent and Critical Aware Scheduling

4.1 Introduction

With the help of the forecasted irradiance and temperature values from Chapter 2, and the anticipated SoC from Chapter 3, we apply a proactive energy, task dependent and critical aware scheduling algorithm (ETCA) for IoT nodes. The algorithm seeks to minimize the scheduling duration of the application, and balance the available energy of the nodes while, abiding to task dependency, criticality, and energy awareness of the application.

In Section 4.2, we present the system model, formulation, and heuristic for our proposed scheduling algorithm (ETCA). Finally, in Sections 4.3 and 4.4, we present the attained results and conclusion.

4.2 ETCA Scheduling

In this dissertation, we propose a different version of the high-level task scheduling algorithm where we use multi-version nodes instead of tasks. The multi-version nodes have harvesting and available energy attributes. In this scheduling algorithm, an application

(made up of unique tasks with start and finish deadlines) is scheduled across an IoT-based WSN. The goal is to maximize the available energy of the nodes and minimize the scheduling duration while, fulfilling the task dependency, criticality, completion, and energy awareness of the application.

We first present the system model, followed by the mathematical formulation of the optimization problem, followed by the proposed heuristic to solve this problem. Finally, we present and discuss the attained results.

4.2.1 System Model

The system model is presented in Figure 4.2-1. This system consists of various solar-EH sensors deployed in a WSN. Sensors are allocated tasks based on their present conditions of availability, energy status, and computational cost. The goal is to fulfill the required application while trying to reduce the scheduling duration of the application, maximize available energy, and abide by the dependencies and the sequence of which the tasks should be satisfied.

At the user end, with the help of the input from the data collection and analysis unit, users decide about the applications that they want to run. These applications are stacked based on either the users' preferences or their deadline. Once the application is ready to be executed, the direct acyclic graph (DAG) representation of the chosen application is fetched, and both the time's stamps of the tasks (nodes in DAG), and their followers are defined in the de-stacked segment of the system model. After the priorities and execution costs are defined, the tasks are then allocated to the deployed energy harvesting sensor nodes that result in the least energy cost and minimum scheduling length. After the

application has been fully executed, the output is then fed back to the user end, to satisfy the users' query and provide data for further processing.

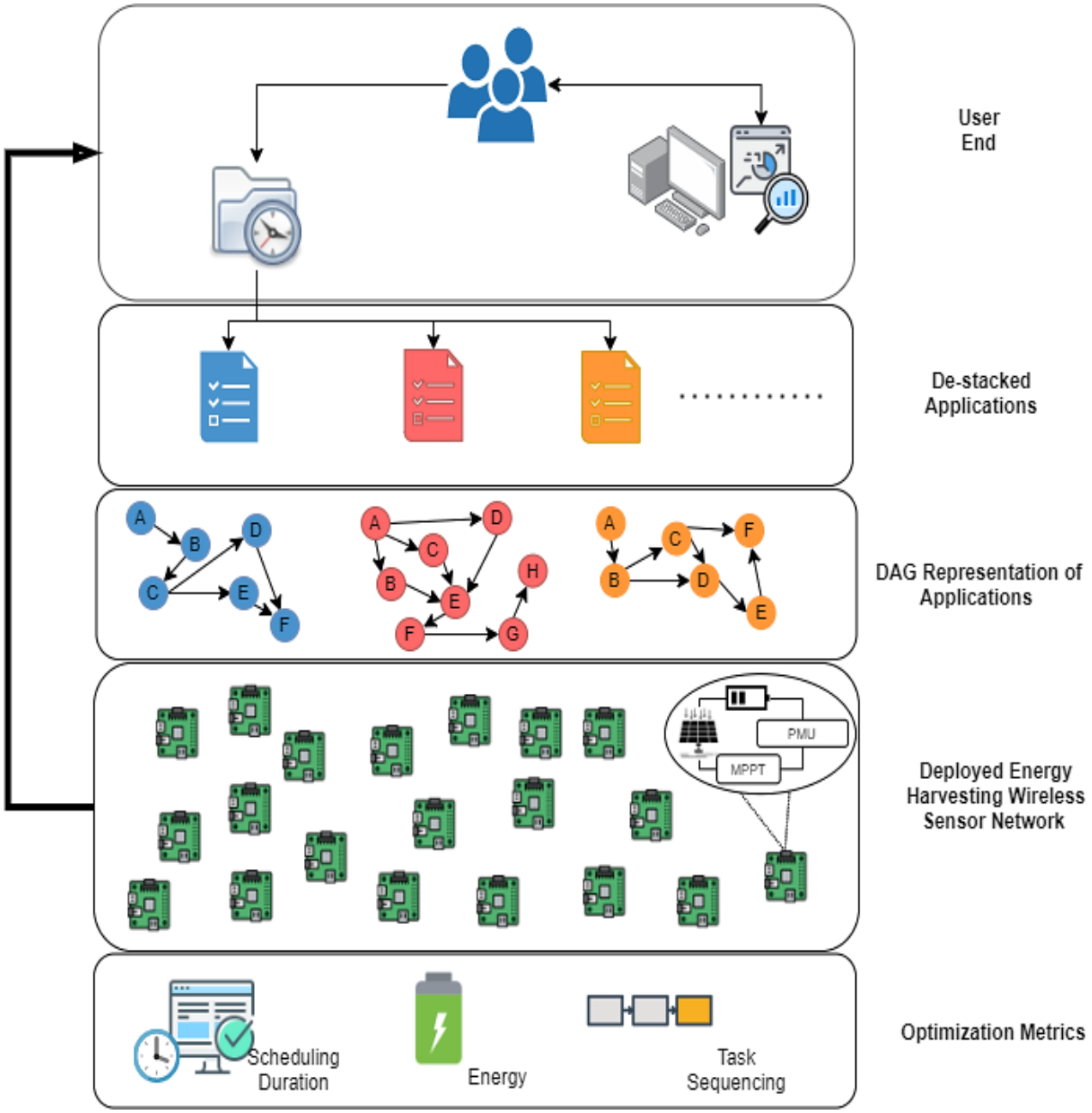


Fig. 4.2-1: System model of the ETCA scheduling algorithm

4.2.2 ETCA Formulation

In this section, we use the information provided in the system model, to build a mathematical representation of our ETCA problem. To this end, we aim to minimize the scheduling duration and balance the available energy across the WSN, while satisfying the task dependencies, energy cost, and priority of critical tasks.

Task: *m* = A or B or C

Set of all Tasks: *M* = {A,B,C,D,E,F}

A → ***B*** = Task B is dependent on A

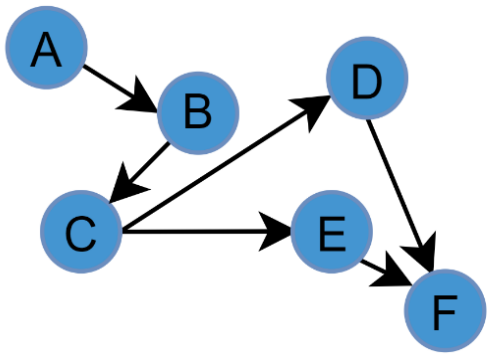


Fig. 4.2-2: DAG representation for an application

Consider a set of multiple solar powered sensor nodes n where $n \in N$, and $N \in Z^+$ is the set of all energy harvesting sensor nodes deployed in the WSN. Let m be the task under consideration where $m \in M$, and $M \in Z^+$ is the set of all tasks that correspond to the application under consideration. Each considered application (objects in the De-stacked segment of the system model) translates into a DAG, where the nodes are the tasks that need to be completed to accomplish the application; the arrows are the dependencies between them, as seen in Figure 4.2-2. It is worth mentioning that tasks in this work represent blocks of instructions that once executed achieve an action like sensing, communication, storing, actuation etc. Each application consists of a set of tasks.

This work deals with a discrete time model. Both the harvesting and scheduling periods consist of evenly sized time slots *t*, where $t \in T$, and $T \in Z^+$ is the considered timeframe. In addition, the duration of a task *m* is denoted by D_m where $D_m \in Z^+$; the harvested energy of node *n* at time *t* is denoted by $E_{h\,n,t}$. Given the above information, we can formulate the energy model for this system in Equation (4.2.1).

$$E_{n,t+1} = E_{n,t} + E_{h\,n,t} - (\eta_d \,.\, E_{C\,n,m} \,.\, B_{m,n,t}), \quad 0 \le E_{n,t} \le E_{max}, \qquad (4.2.1)$$

where $E_{n,t+1}$ and $E_{n,t}$ are the energy at node *n* at time *t+1* and *t,* repetitively. η_d is the discharging efficiency. $E_{C\,n,m}$ is the energy consumption of executed task *m* at node *n*. E_{max} is the maximum battery capacity, and $B_{m,n,t}$ is the Boolean decision variable such that it is defined as follows:

$$B_{m,n,t} = \begin{cases} 1, \text{ if task } m \text{ scheduled to sensor } n \text{ at time slot } t \\ 0, \text{ otherwise} \end{cases} . \qquad (4.2.2)$$

Since *t* represents the base unit of time, each task in the set *M* can occupy more than one time slot. Therefore, we need to find the starting time slot $t_{start,m}$ of each task and their corresponding nodes. We then add the duration of the corresponding task D_m to $t_{start,m}$ to guarantee occupancy of the designated node. In this work, tasks cannot be stopped in the middle of the execution, and they must proceed until completion. The following Equations (4.2.3) and (4.2.4) show the evaluation of $t_{start\,,m}$ and $\chi_{m,n,t}$, respectively.

$$t_{start,m} = \; argmax_t \left(max_n \left[\chi_{m,n,t} \,.\, {}^1/_t \right] \right), \qquad (4.2.3)$$

$$\chi_{m,n,t} = \begin{cases} 1, & t = \; t_{start,m} \;,\; t_{start,m} + 1 \; \ldots\ldots\ldots \; (t_{start,m} + D_m) \\ 0, & \text{otherwise} \end{cases} . \qquad (4.2.4)$$

In light of the above, we are able to utilize the definitions to create the mathematical representation of our optimization problem. Then, we propose the ETCA heuristic and test application used, to solve and test our conditions.

Optimization Problem and Constraint

$$min_{\chi}\left\{ c_1\, max_m\left(t_{start,m} + \ Dm\right) \ + c_2 \sum_{t,n,m} C_{m,n,t}.\chi_{m,n,t} \right\}, \qquad (4.2.5)$$

$$s.t. \qquad \chi_{m,n,t} \in [0,1], \quad \text{for all m, n and t,} \qquad (4.2.6)$$

$$\sum_t \chi_{m,n,t} \ = \begin{cases} 0 \\ L_m \end{cases}, \ \text{for all m and n,} \qquad (4.2.7)$$

$$\sum_m \chi_{m,n,t} = \ 1, \qquad \text{for all n and t,} \qquad (4.2.8)$$

$$t_{start\,,m=1} \ \geq \ \ t_{start,m=2} \ + \ D_{m=2}\,, \qquad \text{for all m,} \qquad (4.2.9)$$

$$E_{n,\,t+1} = E_{n\,,\,t} \ + E_{h\,n,t} - \left(\eta_d \,.\ E_{C\,n,m} \,.\ \chi_{m,n,t}\right), \quad 0 \leq E_{n,\,t} \ \leq \ E_{max}, \qquad (4.2.10)$$

This multi-objective integer linear problem aims at minimizing the scheduling duration and maximizing the available energy, while satisfying the integer constraints of task dependencies, energy, and availability of sensor nodes. In Equation (4.2.5) the first part of the weighted multi-objective function via weight c_1, minimizes the length of the schedule by minimizing the maximum of the finish times of tasks. The second part with c_2 minimizes the assignment cost of tasks to achieve maximum energy across the network. Where $(C_{m,n,t} = Dm \times \left(E_{n,t}\right)^{-1})$ is a cost function that resembles the cost of assigning tasks *m* to node *n* at time slot *t*. Constraint (4.2.6) assures a Boolean assignment of tasks to a given node at a certain time. Constraint (4.2.7) assures all tasks are executed. Constraint (4.2.8) assures that a maximum of one task assigned per node in a single time slot. Constraint

(4.2.9) assures tasks' dependency. Constraint (4.2.10) assures energy limits between 0% and E_{max} (100% in our case).

4.2.3 ETCA Heuristic

The ETCA algorithm assigns the tasks to the available nodes by selecting the lowest $C_{m,n,t}$ for a given combination (*m,n,t*), while giving priority to critical tasks, fulfilling dependencies, and avoiding execution conflicts and redundancies.

We first initialize the system requirements such as the battery capacity, current available energy, number of nodes, tasks (their name, duration, and follower tasks), and the considered time span with a user defined unit of time (e.g., milliseconds or seconds). With the duration and follower combination of all the tasks, we then pass these inputs into a function deduced from the work in [53]. The function is a computerized critical path analysis tool that outputs the critical path, early start (ES), and late start (LS) of the tasks, without the use of the traditional networking diagrams. This is discussed in the next section. Third, we initialize $E_{n,t+1}$ and then sort the tasks given their LS time values, since that is the latest a task *m* can start without incurring a penalty or rejecting the applications' criticality. Fourth, we iteratively proceed through all (*m,n*) for a given *t* to assign the combination costs, while avoiding execution conflicts and redundancies, giving priority to critical tasks, and respecting the dependency of all tasks. Lastly, the least $C_{m,n,t}$ for all *m* and *n* is selected, and $E_{n,t+1}$ is then updated for a given *t.*

ALGORITHM 1 - THE PROPOSED ETCA ALGORITHM

Initialization: Battery Capacity E_{max}, Available Energy $E_{n,1}$, Total number of nodes *N*, time slots *T* and tasks *M,* their respective durations D_m, their follower dependencies (e.g., *m=1 is followed by m = 2, 3, 5*) and their expected energy consumptions
CPM Function: Pass D_m and their respective follower dependencies to return *ES* and *LS* of their corresponding tasks *m*
Assign $E_{n,t+1}$ and **Sort** LS of tasks in an array ascending order
Find Critical Tasks i.e., tasks where *ES* is equal to *LS* and sort them
For *T*
  For *M*
    If in iteration *t* there is no tasks *m* to assign for the current application skip to the next application
    If task *m* was assigned to any *n* skip it
    For *N*
      If node *n* in use skip it
      If ES $\leq t \leq$ LS of Critical task *m*
        Assign *m* to *n*
      else
        If $E_{n,\,t}$ $\geq$ User defined threshold
          If (*m,n,t*) $= 0$
            Assign *m* to *n*
          else
            Calculate the cost function for (*m,n,t*)
        else
          Calculate the cost function for (*m,n,t*)
    end
  end
Find Optimal selection for all (*m,n*) by selecting the least cost pair
Update $E_{n,\,t+1}$
end

Critical Path Analysis

The first part of the heuristic i.e., function in Algorithm 1, is responsible for finding the critical tasks and their ES and LS times, while abiding to the task dependencies. We base our function on a traditional method used in project management and scheduling known as Critical Path Method (CPM). CPM is a powerful, basic, and widely used technique for analyzing, planning, and scheduling large and complex projects (applications in our case).

The goal of CPM is to determine which jobs or activities (tasks in our case) that comprise a project, are "critical" in their effect on the total project time, and what is the best way to schedule these tasks to meet target deadlines and costs [54]. The CPM method is usually illustrated and solved using a project schedule network diagram. A project graph is valuable as a means of depicting, visually and clearly, the complexity of the jobs in a project, and their dependencies. We introduce the network diagram-based CPM method by walking through one a test applications used in the results Section 4.4. We then walk through the logic used in our computer program based CPM that is depicted from the work in [53].

Figure 4.2-3 presents the network diagram-based representation of our test application. First, each block in the network diagram represents a task in the test application; the set of all blocks make up the application. The arrows in the network are used to represent the task dependencies (as mentioned in Section 4.2.2) by connecting the tasks with their immediate successors (i.e., the successor tasks with their followers). Blocks/Tasks with no predecessors are starting tasks. Likewise, all blocks with no successors are finishing tasks [54]. Each block constitutes of six attributes ES, EF, LS, LF, Dur. and task, as shown in the index block. The task entity is a unique identifying symbol used to differentiate tasks from each other. The entity Dur. stands for Duration, this is the estimated time of execution. The duration of each task can be found by using built in features such as the "*tic tok"* command line in MATLAB.

The entities ES and EF resemble the early start and early finish time of the tasks. ES time is the earliest possible time a task can begin, if all its predecessors started at their ES. Likewise, EF time is the latest possible time that a task can conclude, if all its predecessors

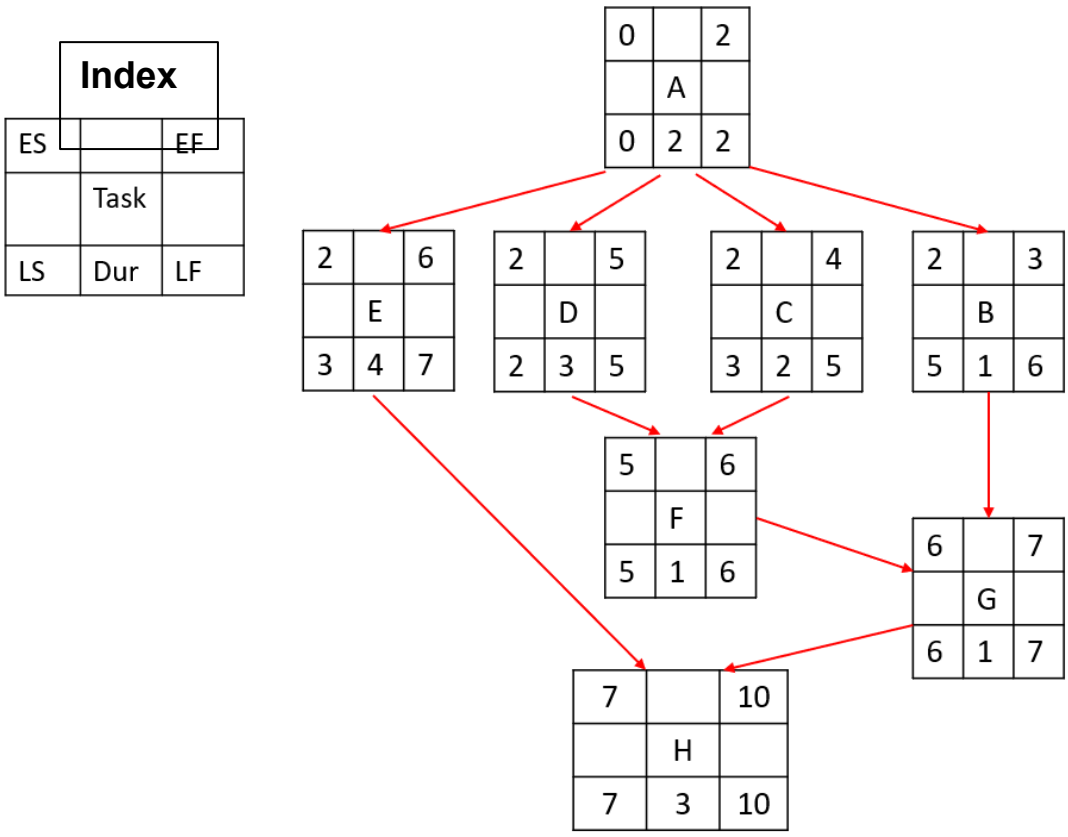


Fig. 4.2-3: Network diagram-based CPM for our test application

started at their EF [54]. The values of ES and EF are calculated using a method known as forward pass. This method begins by placing a unit of time 0 or 1 for ES of the first task (task A in Figure 4.2-3). The rest of the process continues by following the arrows and using the following rules:

- ES = highest EF value from immediate predecessors.
- EF = ES + Dur.

For understanding purposes, we walk through the test application in Figure 4.2-3. We first initiate the ES variable for task A as 0 since it acts as our first activity. The duration of task A is 2 units of time, this means task A is expected to conclude at unit of time 2 (ES (0) + Dur. (2) = 2), or in other words EF = 2. We then proceed to the follower tasks by advancing

to the tasks connected via the dependency arrows. The follower tasks are B, C, D and E. The ES time assigned to these tasks is 2, this is because follower tasks can only initiate once their predecessors conclude. As done previously, the EF values for B, C, D and E equate to 3, 4, 5, and 6 respectively. Now, since task F is the follower task for immediate predecessors D and C, we need to select the highest EF value and assign it to ES of task F (as seen the first bullet point). Task D has the highest EF of 5, therefore the ES of task F is 5. The same process is done for tasks G and H respectively. The EF of F i.e., 6, is assigned to task G since it is larger than task B; The EF of G i.e., 7, is assigned to task H, since it is larger than task E.

The entities LS and LF resemble the late start and late finish time of the tasks. LF time is the latest possible time an activity can conclude without incurring a delay. LS time is the latest possible time an activity can initiate without incurring an error in application execution. Both LS and LF values are calculated using the backward pass method. The method begins by starting from the last task (H in Figure 4.2-3). The EF value of the last task is assigned to the LF value. The rest of the process continues by following the arrows in the opposite direction, and using the following rules:

- LS = LF - Dur.
- LF = lowest LS value from immediate successors.

Again, for understanding purposes, we walk through the test application in Figure 4.2-3. We first initiate the LF variable of task H as 10 since EF equals LF for the last task. The duration of task H is 3, this means the LS of task H is 7 (LF (10) + Dur. (3) = 7). We then proceed to the predecessor tasks by retracing to the immediate tasks connected via the

dependency arrows. The predecessor tasks are E and G. As shown in the second bullet point, The LF value for both tasks E and G is 7 since, task H is the only task available therefore, making it the lowest LS. Now taking task G into consideration, the LS value is 6, this is calculated using the equation in the first bullet point. The same process is repeated for tasks F, D, C and B resulting in LF values of 6, 5, 4 and 3, respectively; LS values of 5, 2, 2 and 2, respectively. Finally, for task A the LF value is the lowest value from the immediate successors B, C, D and E. This value is 2; the LS value is 0 since LS is LF minus the Dur. (2 – 2 = 0).

Looking at the CPM method without the use of a network diagram. We utilize the following heuristic i.e., Algorithm 2 CPM function.

ALGORITHM 2 - CPM FUNCTION

Inputs: Vector of durations D_m, List *followers* that contains followers for each task *m*
Initializations: struct *path,* int *start* = 0, Vector *follower* tasks , Vector *nonfollower* tasks, struct *way*, struct *new_way,* int *initial = 0*
%% Find all possible paths
For *ii* in length *nonfollower*
    Store in struct *way* % both *way* and *new_way* stand for incomplete paths
end
while *initial* not equal to 0
    initial = 0 % exits once the last task in all paths have no followers
    For *ii* in length *way*
        e = way(ii).end; f = follower(e) % last task in follower and their followers are stored
        If *f is empty* % if the last task in the *way* has no followers then path is complete, store it
            Increment *start; path(start) = way(ii)*
        else
            For *jj in length f* % if *way* ≠ *path* get the follower of each task and add it to *new_way*
                Increment *initial; new_way(initial)* = [*way(ii) f(jj)*]
            end
        end
        way = new_way; new_way = []
end

%% Find ES and EF
Create a Zero Vector *EF* of length *follower*

For *ii* in length *follower*
    Create empty array *M* and *N* % *M* will store location of task *ii* in all paths
    For *jj* in length *path* % *N* stores the index of the paths that contain *ii*
        If task *ii* is in *path(jj)*
            N = [*N ; jj*]; *M* = [*M ;* location of *ii* in *path(jj)*];
    end
    Create a Vector *T* of length *N*
    t = 0 % Arbitrary value of iteration
    For *kk* in length *N*
        For *mm* in *M(kk)*
            $t = t + D_m$ *(path(N(kk)).mm)* % Sum duration of all predecessors of *ii* in *path(kk)*
        end
        T(kk) = t
    end
    EF(ii) = max(T); T=[] % Find maximum sum of Duration time to get to task *ii* this is *EF*
end
ES = EF – D_m % Find the ES of all tasks by subtracting D_m from *EF*

%% Find LS and LF
Create a Zero Vector *LF* of length *follower* % Stores LF of all tasks
Create a Zero Vector *path_Dur* of length *path* % Stores path time of all paths
For *ii* in length *path*
    path_Dur(ii) = sum(*time(path(ii))*) % Sum the duration of all tasks in *path(ii)*
end
max_path_Dur = max(*path_Dur)* % We find the path with maximum Duration for later use
For *ii* in length *follower*
    Create empty array *M* and *N* % *M* will store location of task *ii* in all paths.
    For *jj* in length *path* % *N* stores the index of the paths that contain *ii*
        If task *ii* is in *path(jj)*
            N = [*N ; jj*]; *M* = [*M ;* location of *ii* in *path(jj)*];
    end
    Create a Vector *T1* of length *N*
    t1 = 0 % Arbitrary value of iteration
    For *kk* in length *N*
        For *nn* from *M(kk)* to length(*path(N(kk))*)
            $t1 = t1 + D_m$ *(path(N(kk)).nn)* % sum duration of all predecessors of *ii* in *path(kk)*
        end
        T1(kk) = t1 – time(path(N(kk)).M(kk))
    end
    EF(ii) = max_path_Dur – max(*T2*)
end
LS = LF – D_m % Find the LS of all tasks by subtracting D_m from *LF*

As shown In the CPM function, we find the ES, EF, LS and LF values of all the tasks, which we then feed into our ETCA heuristic. The CPM function finds the mentioned values without the use of the network diagram in Figure 4.2-3.

In the first section of Algorithm 2, we find all the possible paths by considering the starting tasks that are not followers of any other task. So, in our test application, task A is the starting task for all paths. We then consider each follower of task A as a different initial path (variable *way)*. In our case we have initial paths A-B, A-C, A-D and A-E. We then take the end tasks of each path in the variable *way* and consider their followers, to build on the initial paths given. In our case, the new paths are A-B-G, A-C-F, A-D-F and A-E-H. This method repeats until all the paths are completed, and then stored in the variable *path*.

In the second section, we first find EF using the forward pass. We then find ES by subtracting Dur. from EF. From all paths, we store the location and path index of each task in *M* and *N,* respectively. We then sum the duration of all predecessors leading to selected task. For example, in our test application we consider task F. There are two paths leading to F, they are A-C and A-D. We sum their durations and store it in Vector *T*. Out of the two values stored in *T*, we then find the maximum duration using *EF(ii) = max(T)*, this is EF. The same is done for all tasks.

In the last section, the backward pass method is used to find LF. LS is then found by subtracting the Duration from LF. This method is similar EF however, the only difference is that we difference the sum of the maximum path that contains the task (max(*T2*)) from the path with maximum sum of durations (*max_path_Dur*).

4.3 Results

The results obtained after running the CPM function in Algorithm 1 is illustrated in Table IV. Follower tasks and Dur. acted as the inputs of the CPM function. Whereas, ES, EF, LS, LF, and Slack are its outputs. Slack is used to dictate the critical tasks in the application, if a task has equivalent ES and LS values then its slack is 0 (ES - LS = 0). Tasks with 0 slack are considered critical since we cannot extend the task past its ES time. From Table IV, the critical tasks of the test application are A, D, F, G and H. The ES, LS and critical tasks from the CPM function are then used in Algorithm 1.

Tasks A to H make up the test application. The followers of the tasks imply the dependencies. For example, task G is dependent on both B and F, since G is seen as their follower task. This test application with a maximum of two dependencies, was randomly chosen to test the effectiveness of the ETCA algorithm

Table IV: Results after running CPM function for test application

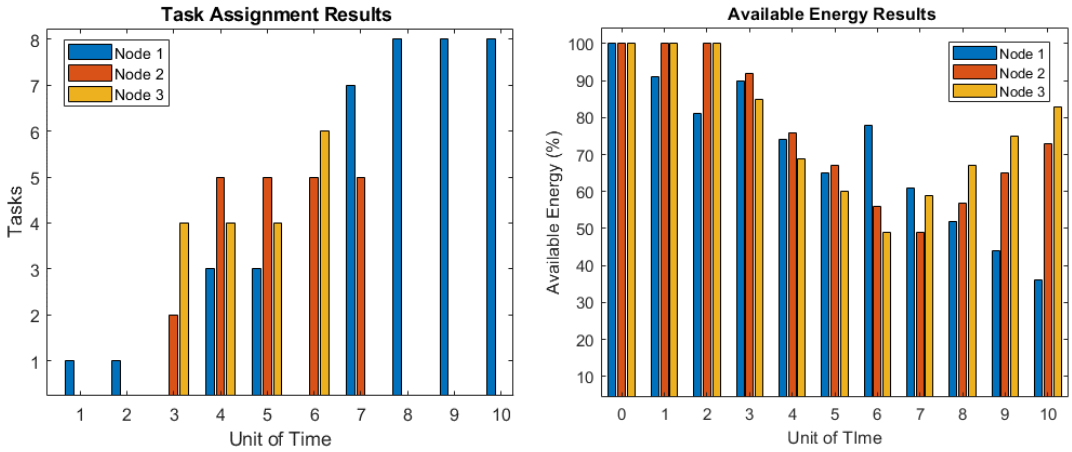
Tasks	Follower Tasks	Dur.	ES	EF	LS	LF	Slack
A or 1	B C D E	2	0	2	0	2	0
B or 2	G	1	2	3	5	6	3
C or 3	F	2	2	4	3	5	1
D or 4	F	3	2	5	2	5	0
E or 5	H	4	2	6	3	7	1
F or 6	G	1	5	6	5	6	0
G or 7	H	1	6	7	6	7	0
H or 8	-	3	7	10	7	10	0

Dur. – Time Duration of the tasks
ES – Early Start Time
EF – Early Finish Time
LS – Late Start Time
LF – Late Finish Time

Figure 4.3-1 presents the output of the ETCA algorithm after running the test application. From these results we are looking to assess the completion time of the application (should equal LF of the last task in the application), critical task priority (critical tasks are not postponed), task dependencies, and low available energy variance or mean absolute deviation between all nodes.

The graph in Figure 4.3-1 (a) shows the assignment of the tasks in the y-axis to the color-coded nodes. We ran the application in Table IV using a practical and arbitrary number of 3 nodes. First, we can see that the time criticality has been fulfilled since the last task was executed at unit of time 10 i.e., LF of Task H. Second, Task dependencies. If we observe the tasks starting and finishing times, we will see that dependent tasks do not execute before their predecessors (e.g., G or 7 does not execute before B and F). This fulfills the dependency condition. Third, each task was fulfilled to completion. Lastly, the critical tasks were completed in their designated time slots to prevent a miss (e.g., Task A and H at Unit time 1 and 8, respectively).

The graph in Figure 4.3-1 (b) shows the available energy in the y-axis, versus the color-coded nodes and unit of time *t* in the x-axis. Whenever the energy deviation between the nodes increases due to task assignment to one of the nodes, the energy of the nodes later tend to converge towards one another. For example, moving from time slots 2 to 3 the mean absolute deviation (MAD) is 8.46 and 2.6, respectively. The ETCA algorithm chose to execute tasks 2 and 4 on nodes 2 and 3, respectively resulting in a MAD lower by 5.86 points. The same goes for time slots 6 to 7, the MAD is 11.3 and 4.9, respectively. The ETCA algorithm lowered the available energy spread by assigning task 7 to node 1, this resulted in a MAD lower by 6.4 points. This demonstrates the energy balancing capabilities



(a) Task Assignment Results (b) Available Energy Results

Figure 4.3-1: Task assignment and available energy for the test application

of the algorithm. It is important to note that the energy divergence in time slot 10 is reduced afterwards, since the application completed.

The next step is to assess the variance in available energy as we increase the number of nodes. Figure 4.3-2, shows the variance in available energy after running the test application across an ascending number of nodes. It is evident that the variance asymptotically decays and saturates between nodes 11 and 12. This is due to the fact that there are more nodes available to choose from resulting in more harvesting time. In

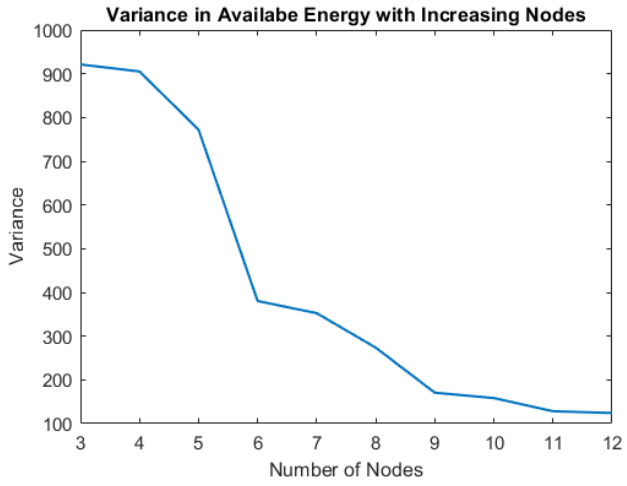
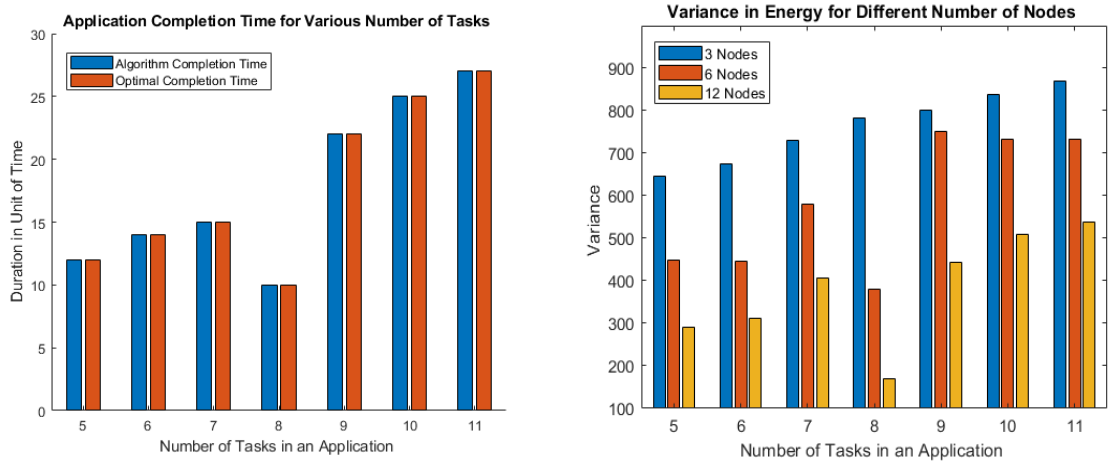


Figure 4.3-2: Variance of available energy as the number of nodes increase

addition, as we increase the number of nodes i.e., sample size, the variance is expected to decrease. The steepest decay of 50% occurs at node 6, we presume this is due to the structure of dependencies, the harvested energy of each node, and the duration of the tasks. This changes for different applications since they constitute of different nomenclature. For different applications, users can use this method to find the number of nodes that result in the most significant drop in energy variance, instead of just increasing the number of nodes randomly.

Next, we test the functionality of our algorithm by fully executing applications with a varying number of tasks, while reducing energy variance with an increasing number of nodes. Figure 4.3-3 presents the results of testing various applications with an increasing number of tasks, and their respective energy variance for three configurations 3, 6 and 12 nodes. Figure 4.3-3 (a) illustrates an increasing completion time as the number of tasks in an application increase (except for 8 tasks). This is expected since applications made up of more tasks take longer to complete. However, that is not the case with our test application.



(a) Application duration compared to Graph Method (b) Variance in Energy for 3, 6 and 12 nodes

Figure 4.3-3: Application completion time and energy variance

This is due to the number of dependencies in an application. Our test application consists of a maximum of two dependencies (2 tasks precede) for 3 tasks 6 (F), 7 (G) and 8 (H). Whereas applications with 5, 6 and 7 tasks constitute of a maximum of three dependencies for tasks 1, 2 and 3 respectively. This means higher dependencies followed by increasing number of dependent tasks, will lead to longer completion times. The application with 10 tasks has a maximum dependency of five confirming our logic. The optimal completion time refers to the LF results attained from the traditional CPM using a network diagram. It is seen that our CPM function acts as expected by coinciding with the optimal durations.

Figure 4.3-3 (b) illustrates an increasing variance in available energy as the number of tasks in an application increases. In all instances, as the number of nodes increase the variance in energy decreases. Again, this is due to the greater sample size resulting in a distributed consumption scheme. In 8 tasks, we notice the most significant variance drop compared to the rest of the applications. The variance drop is 48% from 3 to 6 nodes and 45% from 6 to 12 nodes. This is due to the low number of dependencies available. With less dependencies more tasks are executed simultaneously across the nodes therefore, tasks are more evenly distributed as the nodes increase.

The variance values in the previously presented results were calculated using the remaining energy of the nodes after the application was fully executed. First, the remaining energy of the nodes were selected, then the " *=VAR.S* " function was used in excel.

Before we conclude, it is worth mentioning the scalability of the algorithm in larger settings. As seen in our system model, the algorithm runs on a central node. $E_{n,\,t+1}$ values are sent from the IoT nodes to the central every time window *T*. Nodes then receive the

tasks for execution. This method is feasible for small networks however, once the scale increases the cost of transmission is significant. Therefore, we propose the following:

- Since are algorithm runs on computationally restricted IoT nodes, we can distribute the algorithm by splitting it into subtasks, as seen in Fig 4.2-3. This distribution reduces the centrality aspect.
- Add a scalability aware attribute to the algorithm. This allows for the dynamic adaptation to the changing network.

4.4 Conclusion

In this chapter, we proposed an ETCA Scheduling algorithm. With the number of nodes, tasks (their name, duration, and energy consumption), and the forecasted available energy as inputs, the algorithm uses a programmed CPM function instead of a network diagram to minimize the scheduling duration and balance the available energy of the nodes, while fulfilling the task dependency, criticality, completion, and energy awareness of the application. We can conclude that the results of the CPM algorithm coincide with the traditional method as shown in Figure 4.3-3 (a). In addition, the ETCA algorithm reduces the scheduling duration to the maximum allowed LF of the last task in the application, while executing all the tasks and respecting their deadlines as shown in Figure 4.3-1 (a). Finally, the algorithm reduces the nodes available energy deviation by distributing the tasks across them as shown in Figure 4.3-1 (b).

Chapter 5: Implementation of Prediction and Scheduling

5.1 Introduction

In this chapter, we present the implementation of the previously discussed chapters, ambient source prediction, energy prediction and ETCA scheduling. The goal is to demonstrate to the reader that, the previously presented theory can be implemented via readily available IoT sensor nodes, the likes of the Raspberry Pi (RPI).

The chapter is broken down into Sections 5.1, 5.2, and 5.3, where we discuss the implementation of ambient source prediction, energy prediction and ETCA scheduling,

respectively. Each section contains four subsections implementation setup, challenges and solutions, implementation results and a conclusion.

5.2 Ambient Source Forecasting

As shown in Chapter 2, our forecasting method takes one week of historical data for both irradiance and temperature, then fits it with the best (minimum squared error) SARIMA model. This acts as the dynamic model of the Kalman filter. Finally, with the help of the Kalman filter we can smoothen our prediction. In this regard first, we build up the training set by attaining actual irradiance and temperature readings from sensors connected to a microcontroller; we then compare it with the predicted values. Second, we run the Kalman filter on the sensor node to allow for instant location specific prediction, which proves that our prediction method runs on constrained IoT devices. Finally, we use the predicted values for energy prediction. Our $t+1$ time window is 15 min. This was selected to accommodate for the low rate of charge caused by the PV module and the 15 min application window

5.2.1 Implementation Setup

The considered IoT node is presented in Figure 5.2-1. The main components of the sensor node include the temperature sensor, light dependent sensor, microcontroller (RPI), and integrated circuitry (IC). We define these components further as follows:

- **Temperature Sensor**: in this setup a AM2302 temperature-humidity sensor was used. It is a low-cost digital temperature sensor used in small setups, such as an IoT sensor node. With this sensor attached to the back of the PV panel, we can measure the temperature of the PV panel throughout the day, to account for varying power outputs caused by the varying temperature (as discussed in Chapter 3).

- **Light Dependent Resistor (LDR)**: a PDV-P8001 photocell sensor was used. It is a low-cost and readily available sensor used in various projects. With this sensor attached to the side of the PV panel, we are able to detect the intensity of the incident sunlight throughout the day. As the intensity of the light increases, the resistance decreases, and vice versa. The resistance of the sensor varies from 300 Ω to 10 MΩ.
- **IC**: consists of various components such as an INA291 (current monitor), voltage divider, and analog to digital converter (ADC). The INA291 is a power and current monitor used to measure the consumption of the RPI. The voltage divider is used to drop the voltage of 5 V to 3.3 V used by other components (e.g., ADC). Finally, the ADC used is a MCP 3008. It is a low-cost analog to digital converter with a 10-bit resolution.
- **Microcontroller**: A RPI 3B+ was used. This is the latest and final product of the RPI 3 range. It features a 64-bit quad core processor running at 1.4GHz and 1GB

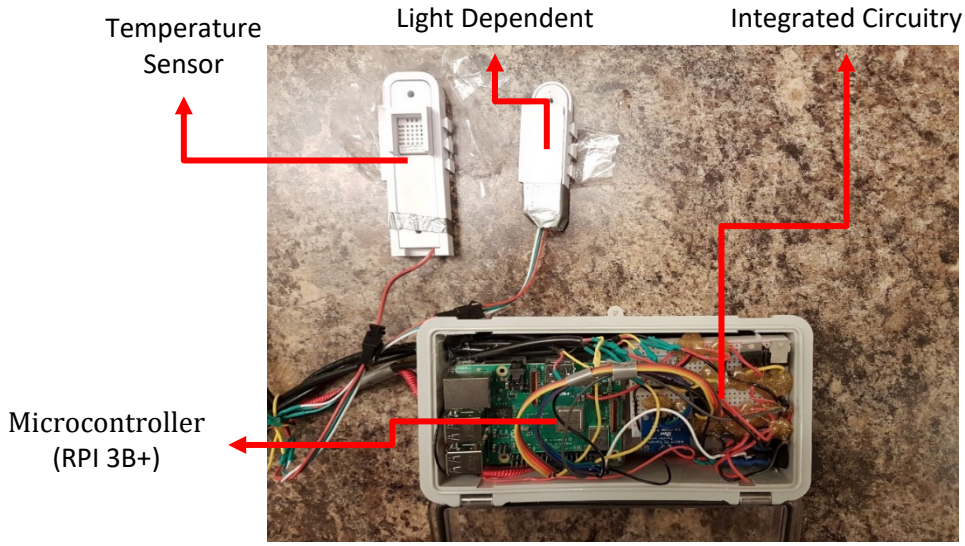


Figure 5.2-1: IoT sensor node unit

RAM. The RPI is a low-cost readily available microcontroller used to explore various projects in the realm of IoT. With the help of a RPI we are able to read values from both temperature and LDR sensors, and run our prediction model.

In light of the mentioned components in the sensor node, we were able to measure both the temperature and incident light of the PV panel, and apply our prediction model. In Figure 5.2-2, we present the logic flow of our prediction model that was deployed on the RPI. As seen, we first create a file that will host both predicted and measured data for both irradiance and temperature. We then define the function for both sensor readings (returns temperature and irradiance) and Kalman filter (returns the refined prediction). We then create the SARIMA model for both irradiance and temperature, given the historical data from the file. Finally, in an infinite while loop, new measurements are appended, and the prediction for the next time slot is executed.

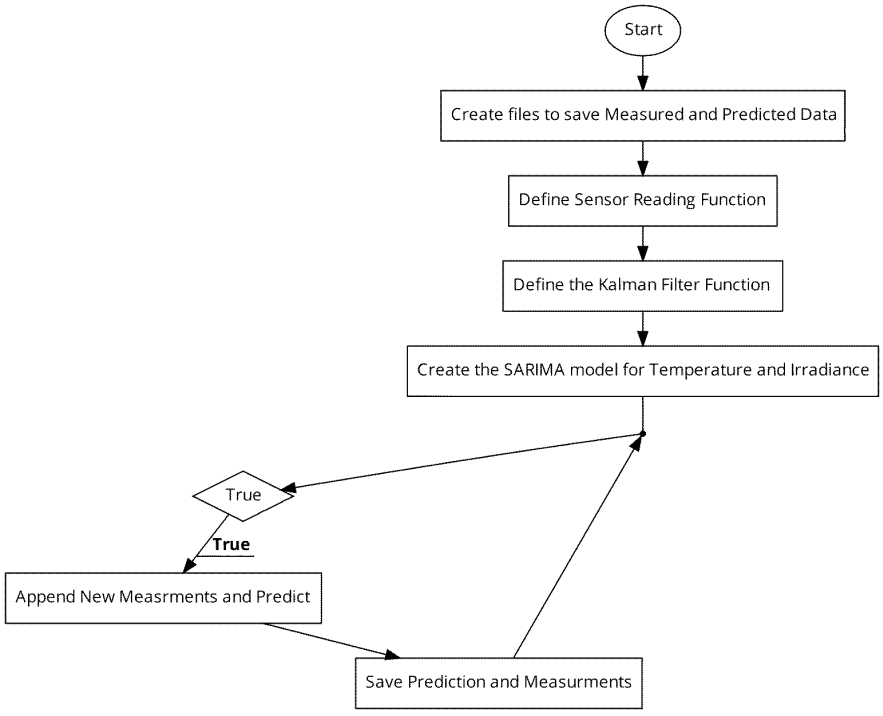


Figure 5.2-2: Flow of logic for Prediction model

5.2.2 Challenges and Solutions

Various challenges were faced during the hardware implementation. In this section, we outline some of the significant challenges and their solutions.

1. ***Challenge***: LDR measures the light intensity (lux) and not irradiance ($\frac{w}{m^2}$). lux is a measure of power from the visible light. This does not include the rest of the electromagnetic spectrum. Irradiance on the other hand, measures the power received from the entire electromagnetic spectrum in ($\frac{w}{m^2}$). This is what the output power of the solar panel depends on.

   Solution: buy a pyranometer to measure irradiance. However, they are expensive and not easily bought. Another solution was to use a conversion factor deduced by [55], which finds a strong correlation between irradiance and luminosity. The study carried out outdoor experiments with both LDR and pyranometer sensors, and found that 1 $\frac{w}{m^2}$ = 122 ±1 lux.

2. ***Challenge***: since our experiment was carried out in the winter season, the sensor node stopped working due to low temperatures. The RPI starts to continuously reboot and then turn off. It is also worth mentioning that, adding an additional enclosure was of no use.

   Solution: carry out the experiment indoors, and attach the sensors to a clear window that receives prolonged periods of light.

3. ***Challenge***: faced setbacks in downloading and importing the required libraries to run the prediction on the RPI.

   Solution: in the literature, the go-to Linux command line to install packages is *apt-get*. However, this command installs packages from the ubuntu repository, which

lacks some of the libraries, and is not always up to date. On the other hand, the command line *pip install* downloads the libraries from the broader python package index.

4. ***Challenge***: space limitations of the RPI 3B+ 1GB RAM. Since the dynamic model of the Kalman filter uses previous data points to predict both irradiance and temperature, appending new data points while maintaining the historical one's results in out of memory.

   Solution: utilize a lighter function from the imported python library (Statsmodel), that does not refit unnecessary old data points, and deletes them after a certain threshold. In our case, datapoints preceding seven days are deleted.

5.2.3 Implementation Results

The graph in Figure 5.2-3, shows the measured irradiance for December 1^{st,} using the LDR sensor in orange, and the irradiance from a weather station here in Stitsville, ON, Canada [56] in blue. We see that both curves follow the same trend, with the measured being lower than the weather station curve. This is because, the derating factor is higher since the sensor is indoors and behind a window. Nevertheless, given the low price of the LDR sensor and the conversion factor used, the loss of accuracy is not an obstacle in our case.

The graph in Figure 5.2-4 shows both the measured irradiance and the predicted irradiance in red and blue, respectively. At midnight, when the irradiance is 0 $\frac{w}{m^2}$ the Kalman filter takes five data points to converge, this is due to the refinement process. We can also see that at the maximum, minimum and inflection points, the SARIMA-KF tends to overshoot or undershoot. This is due to abrupt changes in the measured data. The RMSE and MAE

of the SARIMA-KF with respect to the measured values are 10.52 and 6.17, respectively. Finally, if we compared the prediction error of both SARIMA and SARIMA-KF models,

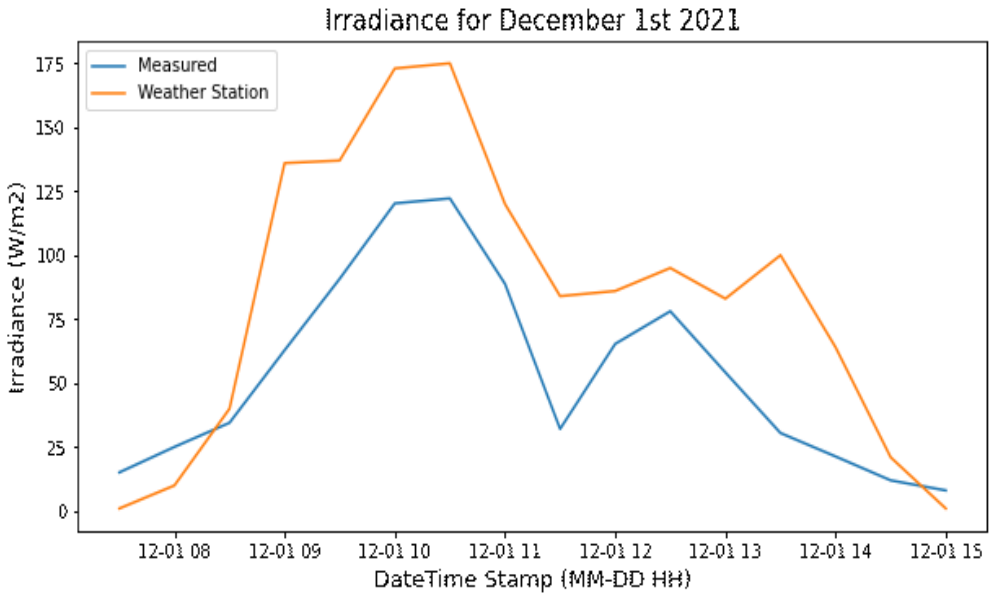


Figure 5.2-3: Sensor and weather station irradiance comparison

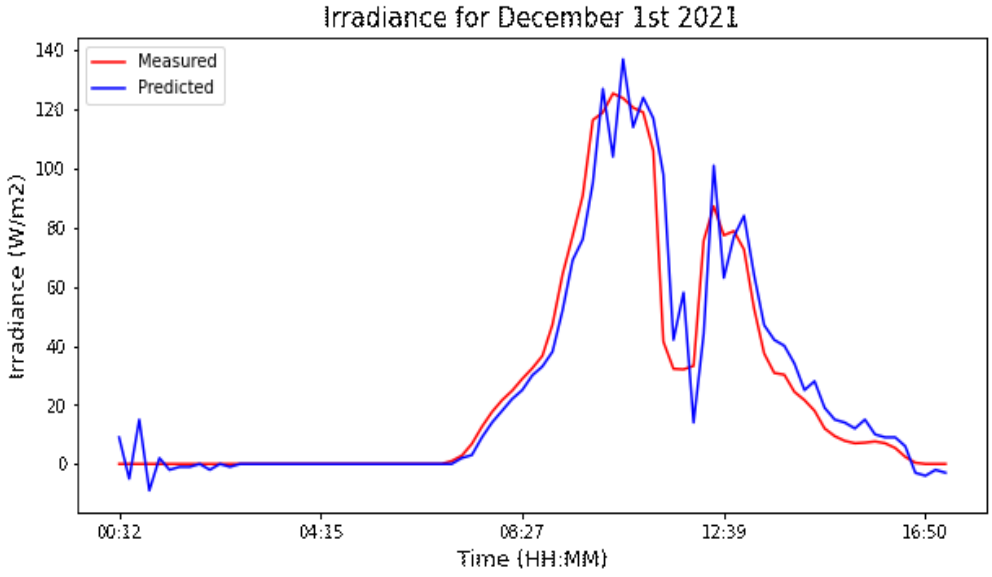


Figure 5.2-4: Measured and predicted irradiance comparison

we can conclude that the addition of the Kalman filter lowers the mean average errors by a factor of 10 and 8 respectively.

The graph in Figure 5.2-5, shows both the measured temperature and the predicted temperature in red and blue, respectively. At midnight, when the temperature is around 16 ˚C the Kalman filter takes only three data points to converge. Compared to irradiance, the temperature has a gradual incline and decline, resulting in the absence of undershooting at inflection points. It is worth noting that the presence of high temperatures, ranging from 16 ˚C to 45 ˚C, is due to the PV panel being indoors, and the temperature sensor attached to the PV panel. The RMSE and MAE of the SARIMA-KF with respect to the measured values are 0.74 and 0.23, respectively. Finally, if we compared the prediction error of both SARIMA and SARIMA-KF models, we can conclude the superiority of the SARIMA-KF model, as it reduces the mean average errors by a factor of 6 and 8 respectively.

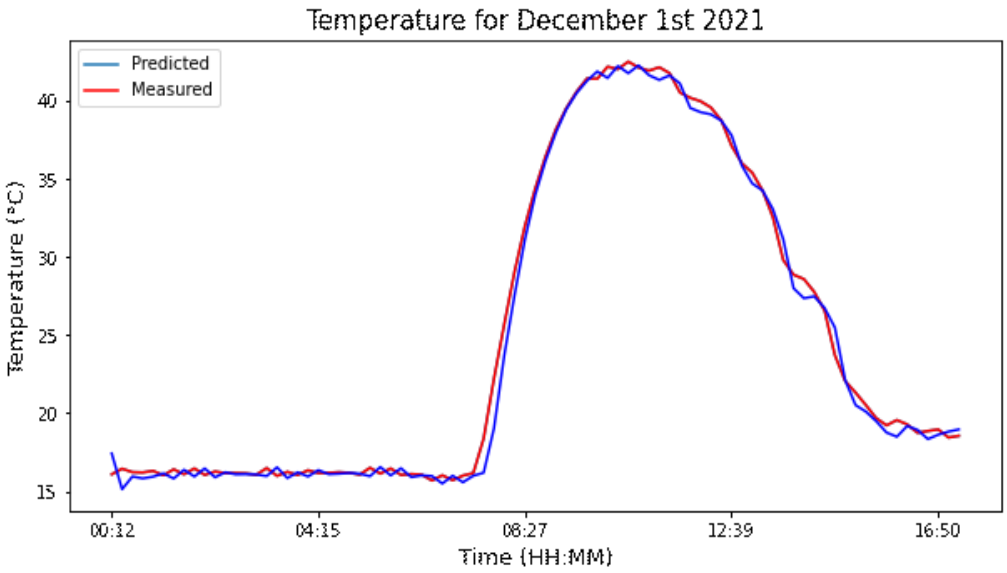


Figure 5.2-5: Measured and predicted temperature comparison

5.2.4 Conclusion

In Section 5.2, we presented the implementation setup and results for Chapter 2. With the help of cheap and readily available components like a LDR sensor, temperature sensor and a RPI, we were able to demonstrate that our theoretical prediction model can perform active predictions using measured values from the sensors, and not just predictions on a static dataset as presented in most of the literature. We also proved that our prediction model suits computational and memory constrained IoT devices by running it on a soon to be obsolete microcontroller (RPI 3B+), where the newer version (RPI4) supports a larger RAM (4 GB and 8 GB) and faster CPU clocks (1.5 KHz).

5.3 Energy Prediction

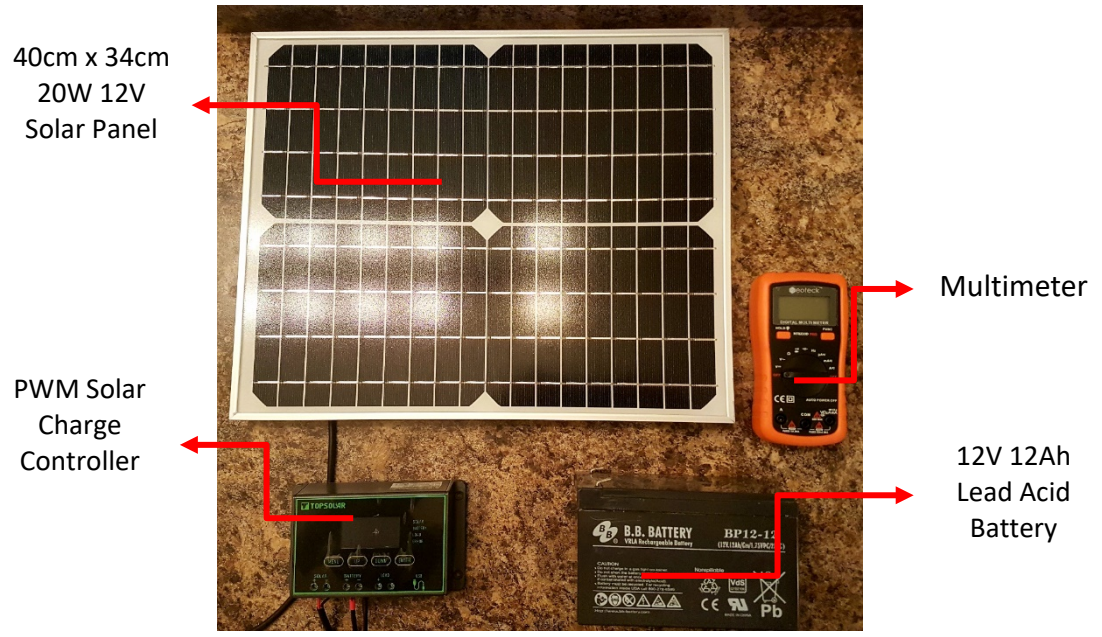
As shown in Chapter 3, our EHU model takes in both irradiance and temperature values as inputs, and with the help of the battery model, outputs the SoC gained. Since our prediction is every 15 minutes and the sampling time is discrete, the SIMULINK model in Figure 3.6-1 was executed for 900 seconds per time slot. In this regard first, we need to attain the predicted irradiance and temperature readings from the microcontroller of the sensor node, and import them to the MATLAB workspace. Second, we run the simulation of the MPPT based model in Figure 3.6-1. Finally, we export the SoC of the battery to the MATLAB's workspace and compare it to the actual SoC gained during our implementation.

5.3.1 Implementation Setup

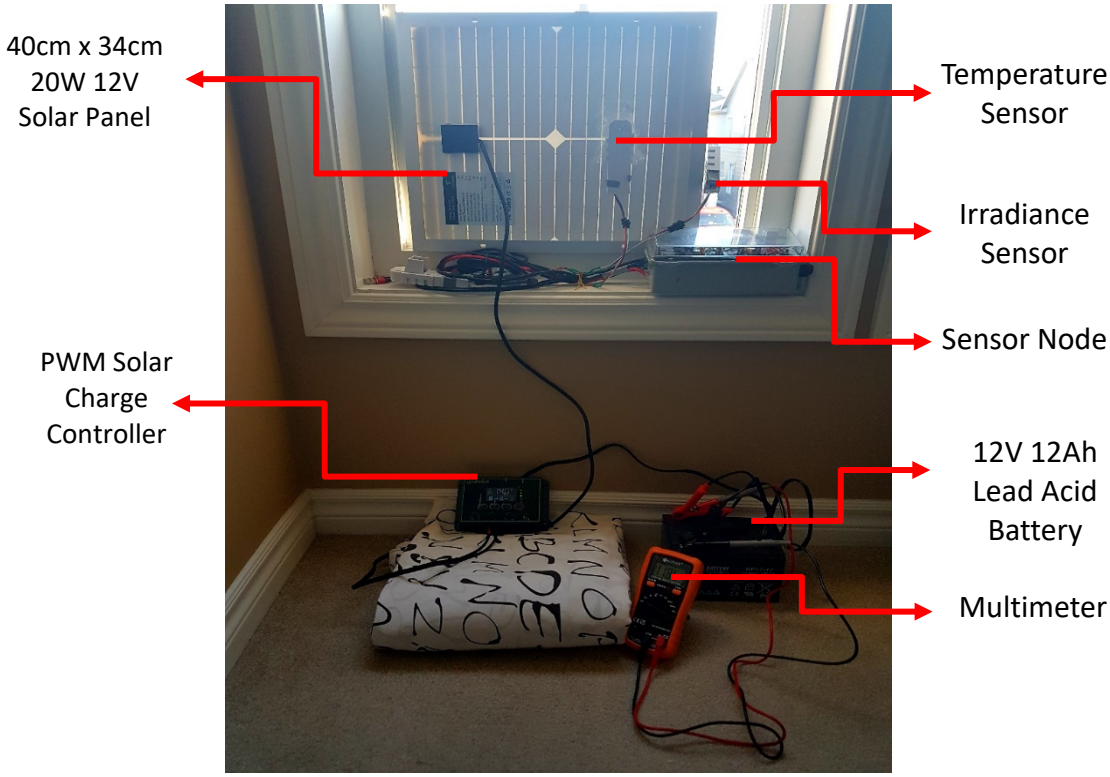
The considered experimental setup is presented in Figure 5.3-1. The main components of the setup are shown in Figure 5.3-1 (a). They are the solar panel, pulse with modulation

(PWM) solar charge controller, multimeter, sensor node, and lead acid battery. We define these components further as follows:

- **Solar panel**: in this setup, a 20W 12V monocrystalline solar panel from TOPSOLAR was used. It is a low-cost, readily available, and affordable option for energy harvesting projects. Given STC, we can achieve a maximum output power of 20 W, at a maximum point voltage of 18 V, and a maximum point current of 1.11 A. The small dimensions (40 cm x 34 cm) of this panel allows for scaling up and use in various remote applications.
- **Solar charge controller**: a TOPSOLAR 10A charge controller was used. Given the inexpensive price of these solar kits, the manufacturers tend to include the less expensive option of the solar charge controller, which is a PWM (not an MPPT). The provided solar charge controller has a maximum input power of 130 W for 12 V setups and 260 W for 24 V setups. The maximum output current is 10 A.
- **Multimeter**: a digital multimeter from Neoteck was used. The purpose of the multimeter is to measure the PV connected voltage, the battery connected voltage and the open-circuit voltage of the battery after 2 to 12 hours from charging. This will provide us with a reliable reading for the current SoC of the battery.
- **Battery**: a 12V 12Ah lead acid battery was used in this setup. These batteries are used in motorized and electrical vehicles since they withstand cold weather. This type of battery was also chosen for two reasons. One, it was already available and two, it has a linear nominal discharge area. This linearity provides a stronger correlation between the open-circuit voltage and SoC, in contrast to the lithium batteries.



(a) Components used in the implementation



(b) System connection

Figure 5.3-1: Complete implementation setup

With the components now defined, we explain the experimental approach and connection setup in steps. In step 1, we measured the initial SoC of the battery by measuring the open circuit voltage and associating it with the SoC, via the SoC vs open-circuit voltage graph provided by the supplier. We then connect the battery to the battery inlets of the charge controller, where the battery-connected voltage is measured. In step 2, we connect the solar panel to the solar inlets of the charge controller. We use these ports to measure the PV connected. Finally , we attach both temperature and irradiance sensors of the sensor node.

5.3.2 Challenges and Solutions

Various challenges were faced during the hardware implementation. In this section we outline some of the significant challenges and their solutions:

1. ***Challenge***: inconsistent irradiance, resulting in fluctuating output power from the charge controller, prevented the use of a USB power meter (more reliable measuring device in contrast to a multimeter). Since the power meter requires constant power to function, the fluctuating output power from the charge controller failed to meet it.

   Solution: limit the system by neglecting any loads, this will provide us with the true gained SoC, and use a multimeter instead.
2. ***Challenge***: the use of a lithium power bank is not viable since, the proper power measurement equipment (USB power meter) cannot be used.

   Solution: Use a lead acid battery instead.
3. ***Challenge***: The simulation model takes a lot of time to execute. The Simulink model uses a sample time of 10^{-5} seconds to allow for 100KHz switching in the

buck converter. This leads to long execution times, defeating the point of prediction since, the execution time will extend into the prediction time slot.

Solution: come up with a simplified model that runs on a RPI, and provides us with instant SoC results given both irradiance and temperature as inputs. The logic of the model is discussed in the following subsection.

4. ***Challenge***: the charge controller provided by the supplier is a PWM and not an MPPT. The model presented in Chapter 3 was for a MPPT system. Both these systems differ in functionality. We will discuss their differences and the new model in the following sub section.

   Solution: alter the MPPT model to mimic the behavior of the PWM based system; compare the performance of both systems and their achieved SoC with respect to the actual (measured) SoC.

PWM versus MPPT Charge Controllers

The charge controller is an essential component in a battery-based system. Their primary role is to manage the charging of a battery bank. During light hours, it prevents battery overcharge from the PV into the battery. At night, it prevents battery discharge from the battery to the PV module. The two types of controllers are PWM and MPPT.

PWM stands for pulse width modulation. In summary, If the voltage of the battery is below floating voltage, the PV panel is directly connected to the battery. This lowers the PV voltage to that of the battery, resulting in power loss. If the voltage of the battery is higher, the PWM controller pulsates the MOSFET with a certain duty cycle, to maintain floating voltage.

On the other hand, MPPT tracks the maximum power. Instead of lowering the PV voltage to that of the battery, the output power of the PV panel at the maximum point equals the input power of the battery. This is done with the help of impedance matching via the buck converter. Since the input power is the same and the voltage of the battery is stable, the input current is increased.

Figure 5.3-2 presents the PWM model whereas, Figure 3.6-1 presents the MPPT module. We skip over the MPPT model since we already discussed it in Section 3.6. The PWM model in Figure 5.3-2 only supports below floating voltage state. As shown, the panel is directly connected to the battery. The resistor is used to mimic power loss due to transmission and other available circuitry. The PV panel takes both irradiance and temperature as constants, then outputs the power which is directly fed into the battery.

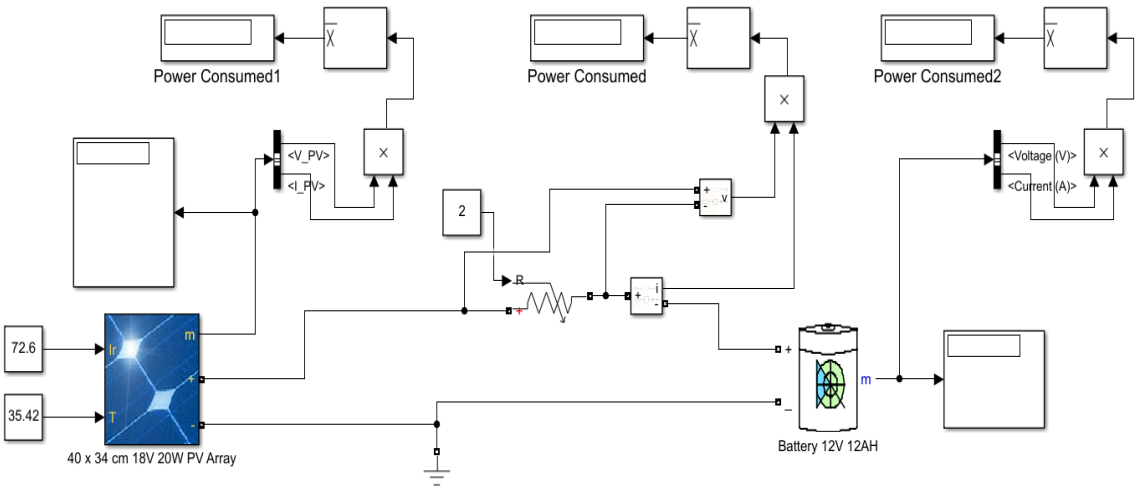


Figure 5.3-2: PWM simulation model

PWM and MPPT Alternative Models

As explained earlier in Section 5.3.2, the simulations take a lot of time to complete and cannot be implemented on a RPI. Therefore, an alternative model is needed. The flow of logic used for the PWM model is shown in Figure 5.3-3.

First, with the help of Equation (5.3.1) provided buy HOMER Solutions [57], we can calculate the PV output power of a MPPT system given both irradiance and temperature.

$$P_{pv} = Y_{pv} f_{p_v} \left(\frac{\bar{G}_T}{\bar{G}_{T,STC}}\right) \left[1 + \alpha_p \left(T_C - T_{C,STC}\right)\right], \tag{5.3.1}$$

where P_{pv} stands for output power in (W), Y_{pv} stands for rated power output at standard test condition in (kW), f_{p_v} is the PV derating factor in (%), $\bar{G}_T$ and $\bar{G}_{T,STC}$ are the measured incident solar irradiance and standard test condition incident solar irradiance, respectively, both have units of $(\frac{w}{m^2})$. α_p being temperature coefficient power in $(\frac{\%}{^\circ C})$. T_C and $T_{C,STC}$ are measured PV temperature and standard test condition PV temperature, respectively, both have units of $(^\circ C)$.

Since Equation (5.3.1) provides the PV output of a MPPT based system and not a PWM, certain losses in PV output power are expected. We compared the PV output simulation results for both systems and deduced that the efficiency of the PWM system is 74% of that of the MPPT. This reduction of power is due to the direct connection of the PWM. We also account for the power lost due to transmission and PWM circuitry. This is 80% according to [58]. Now with the power at the battery being known, we need to convert it to Watt-Hours (Wh). This is done by multiplying the power in (W) with the simulation period i.e., 0.2 in our case (15 min / 60min = 0.2 hours). We then find the Amp-hours being fed into the battery by dividing (Wh) by battery voltage (12 V). Next, we find the difference in SoC (%) using Equation (5.3.2):

$$\Delta SoC\ (\%) = \frac{Battery\ Input\ Current\ (Ah)}{Current\ Capcity\ (Ah)}\ .100. \quad (5.3.2)$$

Finally, with ΔSoC being known, the final SoC is found using Equation (5.3.3):

$$SoC_{final} = SoC_{initial} + \sum_{t=start}^{t=finish} \Delta SoC(t). \quad (5.3.3)$$

The same flow of logic follows for the MPPT based system however, we just substitute direct connection and transmission loss, with the Buck converter and MPPT efficiency. Their corresponding values are 85% and 99% respectively.

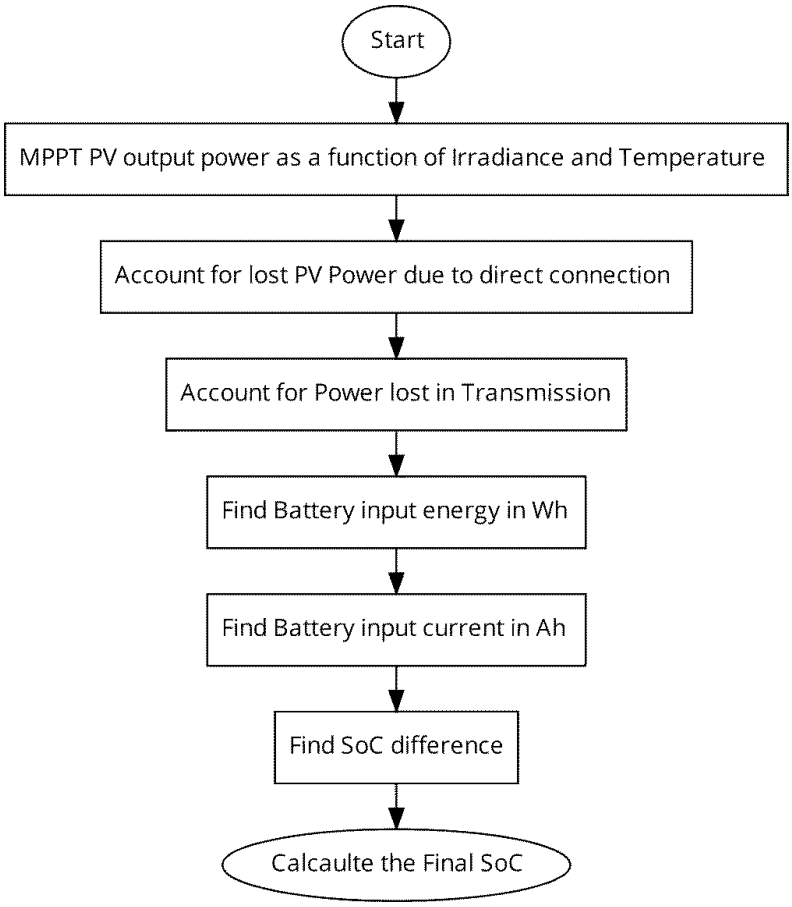
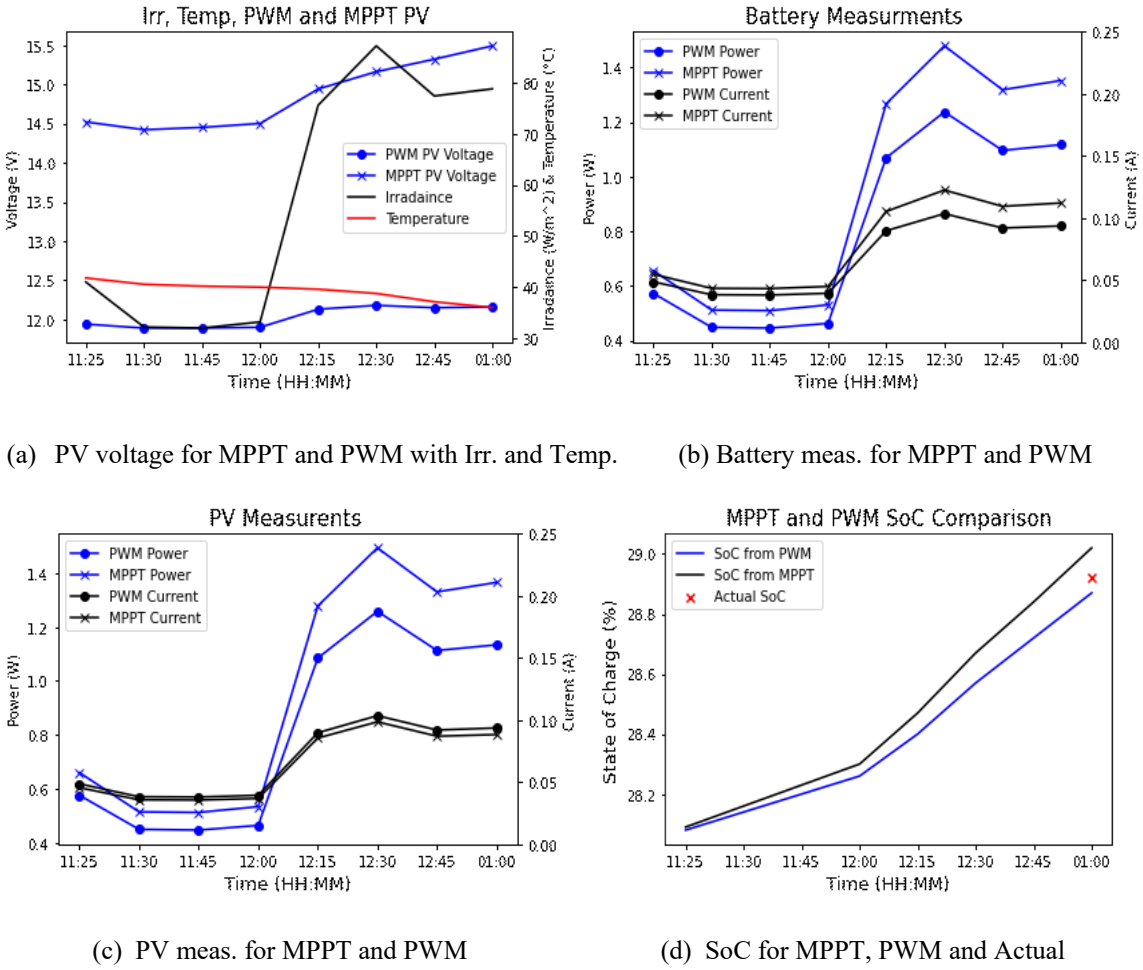


Figure 5.3-3: Flow of logic for PWM model

5.3.3 Implementation Results

We start the implementation results by looking at Figure 5.4-3. These illustrations present the simulation results (Simulink-MATLAB) for both PWM and MPPT systems. Figure 5.3-4 (a) presents the relationship between irradiance, temperature and the PV output voltage. As shown after 12:00, the irradiance starts to increase while the temperature slowly decreases. An increase in irradiance results in an increase in both outputs of the PV panel however, as mentioned earlier, the PV output in a PWM system is restricted due to the direct connection to the battery whereas, in a MPPT system it is not, leading to a greater power output.



(a) PV voltage for MPPT and PWM with Irr. and Temp.

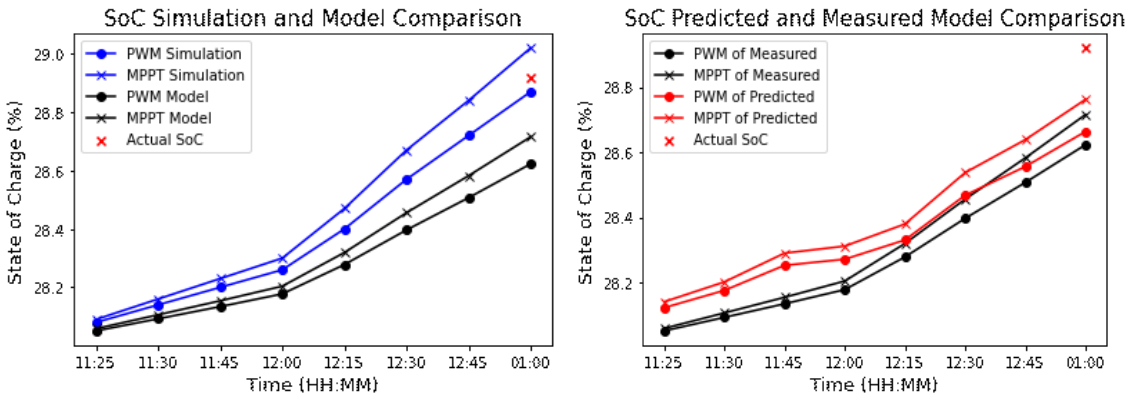
(b) Battery meas. for MPPT and PWM

(c) PV meas. for MPPT and PWM

(d) SoC for MPPT, PWM and Actual

Figure 5.3-4: Results of the PWM and MPPT simulation models

Another fact to point out, between 12:30 and 12:45, although the irradiance decreased the PV voltage still rose, this is due to the decrease in temperature. We can deduce that in this case that the effect of temperature outweighs the effect of irradiance. Figure 5.3-4 (b) shows the measurements (power and current) on the battery side. It is clear that the MPPT provides more power than the PWM system, leading to a greater flow of current. In addition, the rise in power and current is detected after 12:00. This is expected since irradiance increases beyond that timestamp. Figure 5.3-4 (c) shows the measurements on the PV side. Again, the power output of the panel is higher in the MPPT system, compared to the PWM. In addition, the Power outputs of the MPPT system matched the maximum achievable power outputs with a 99% efficiency, this is also achieved by the MPPT products in the market. Finally, and most importantly, Figure 5.3-4 (d) shows the state of charge achieved by the PWM and MPPT system throughout the duration of the experiment; compares it to the final actual SoC attained by the implementation. As expected the PWM simulation results are closer to the final SoC compared to the MPPT. The absolute measurement percentage errors are 0.17% and 0.35% for PWM and MPPT respectively.



(a) SoC simulation and alternative model comparison (b) SoC comparison using predicted values

Figure 5.3-5: SoC comparison for PWM and MPPT given measured and predicted parameters

Given that we ran and compared the simulation models of both MPPT and PWM systems, we proceed to attain the results for the alternative PWM and MPPT models deployed on the RPI. This is portrayed in Figure 5.3-5, where we present the comparison between the SoC from the simulation, the alternative model and the actual SoC from the implementation. Figure 5.3-5 (a) shows that the final SoC from the PWM simulation is the closest to the actual SoC whereas, the SoC form the alternative (deployed on RPI) PWM model is the furthest. The absolute percentage error with respect to the actual SoC for the PWM and MPPT Simulations, and the PWM and MPPT Alternative Models are 0.172%, 0.345%, 1.03% and 0.7% respectively. Note, these results were attained using the measured irradiance and temperature from the deployed sensor node.

Now we use the predicted irradiance and temperature values attained in Section 5.2 to examine the SoC charge attained as shown in Figure 5.3-5 (b). The graph compares the MPPT and PWM alternative models for both measured and predicted irradiance and temperature. It is seen that the SoC from the predicted and measured values are really close to each other, with a mean absolute percentage error of 0.15% from their counterparts. The absolute average error for all the alternative models from the actual SoC is 0.79%. The individual measurement errors for PWM and MPPT using sensor measurements, and PWM and MPPT using SARIMA-KF predicted values are 1.03, 0.70, 0.89 and 0.55 respectively.

5.3.4 Conclusion

In Section 5.3, we presented the implementation setup and results for Chapter 3. With the help of the measured and predicted values from the sensor node, the measured values from the PV panel, the PWM charge controller and the lead acid battery, we were able to charge

the lead acid battery for a specific duration and measure its final SoC. We then compared it to the SoC attained from both the simulation models (Simulink) and the Alternative model (built to suit the deployment on a sensor node). We also compared the performance of a MPPT and PWM charge controller. We concluded that that the simulation models outperformed the lighter alternative models since, they account for more modeling features resulting in more accurate results. We also showed that a MPPT based system is more rewarding than a PWM since, it provides us with 26% more power. Finally, we showed that the resulting SoC from both measured and predicted irradiance and temperature are similar with only an error of 0.15%.

5.4 Task Scheduling

As shown in Chapter 4, with the help of the ETCA scheduling algorithm we are able to reduce the scheduling duration of the application, and maximize the available energy (distribute the consumption) while, fulfilling the required application and abiding by the dependencies. The inputs of the algorithm are the tasks, their dependents, their duration, and their energy consumption. In this regard, we first need an application. This application should constitute of tasks i.e., blocks of code. These tasks should be interconnected with dependencies. Second, we need to find the duration of these tasks. Finally, we need to attain the energy consumption of these tasks.

Before proceeding to the implementation section, we propose a real-world use for this algorithm. The most relevant use case would be vertical farming. Vertical farming is the practice of generating food on vertically installed surfaces in enclosed warehouses. Using this method of controlled environment architecture, we can artificially control the

temperature, light intensity, humidity etc., by acting on the measured data from various sensor nodes [59]. However, one of the main drawbacks, other than the energy consumed to run the facility, is the infrastructure cost and energy consumption of the sensor nodes used for data collection. Each layer in a vertical farm requires large amounts of sensor nodes to sense the variables that affect health of the plants. These include water quality, temperature, nutrients etc. Therefore, running electrical wiring to power up these small nodes is costly. One method that is currently under study is the use of energy harvesting capabilities however, the setbacks as discussed previously, is the unpredictability of ambient source. With the help of ETCA scheduling, we can proactively distribute the tasks of an application across these nodes, to distribute the consumption and prolong their lifetime.

5.4.1 Implementation Setup

Our goal is to run our algorithm using an actual application. To do that we utilized the code used in the current IoT testbed for smart gardening [60]. With the help of the setup in Figure 5.4-1, we were able to virtually connect to the RPI sensor node and access the executable python files. First, we create an application by depicting blocks of code from the files (tasks). We then link them to other tasks to create the dependencies. For example, the sensor node checks the temperature and humidity of the enclosed environment. Both temperature and humidity tasks constitute of blocks of code which include the sensor and pin initialization, function to read values, ADC conversion, and data transmission. In this case, both temperature and humidity tasks are independent therefore, they can be executed simultaneously. After the execution of both tasks, we decide whether to turn on the fan to

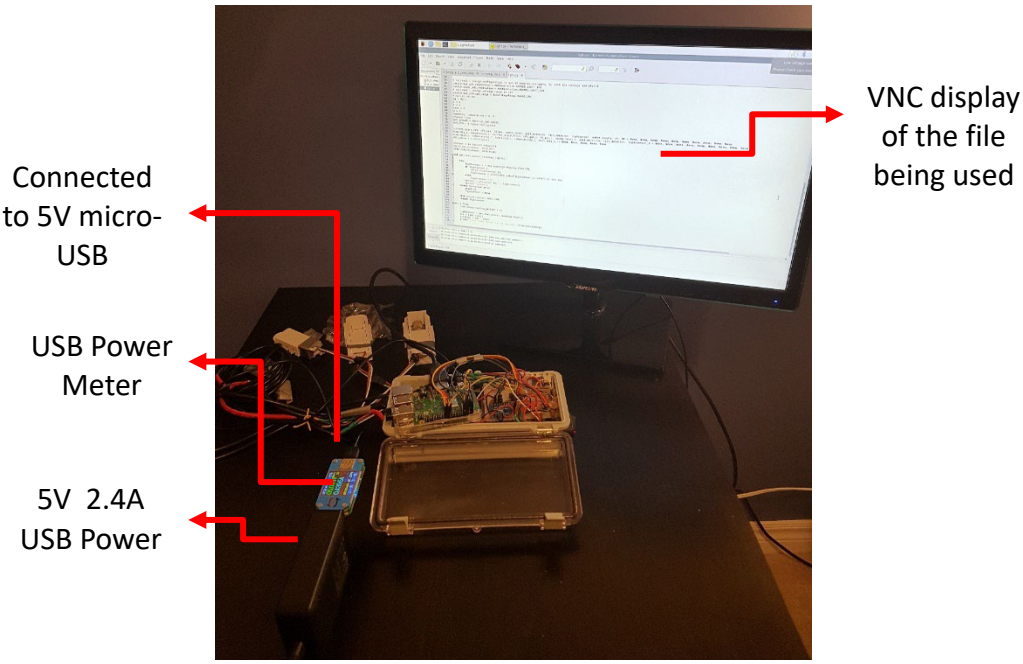


Figure 5.4-1: Task scheduling setup

reduce temperature and humidity. This action is the fan task. We compare the temperature and humidity values to a threshold, and decide whether to actuate the fan or not. We can conclude that the task fan is dependent on both temperature and humidity.

When it comes to the tasks' duration (as shown in Figure 5.4-2), the code of the task is encapsulated between the start and end time reading stamps. This returns the duration of

```
#>>>>> Start time reading
start = time.time()
time.sleep(1)
#
#
  Code of the task
#
#
end = time.time()
duration = end - start
#>>>>> End time reading
print("I ran light sensor for {} seconds".format(duration))
```

```
I ran light sensor for 206.54288244247437 seconds
```

Figure 5.4-2: Task scheduling setup

the task as shown in the print statement. In this case the light detection task ran for 206.5 seconds. The same method is repeated for the other tasks.

Finally, we measure the power consumption of the tasks with the help of the power meter. The USB power meter has the ability to measure the cumulative energy consumption starting from a specific timestamp. As soon as a task is executed the starting timestamp is recorded and the power reading is initiated. Once the task is completed, the end timestamp is recorded and the cumulative power reading is noted. The same process is repeated for all tasks. We now have the required inputs to run our algorithm.

Figure 5.4-3 shows the tasks that make up the application, their respective duration in seconds and energy consumption in (mWh). Description of each task is as follows:

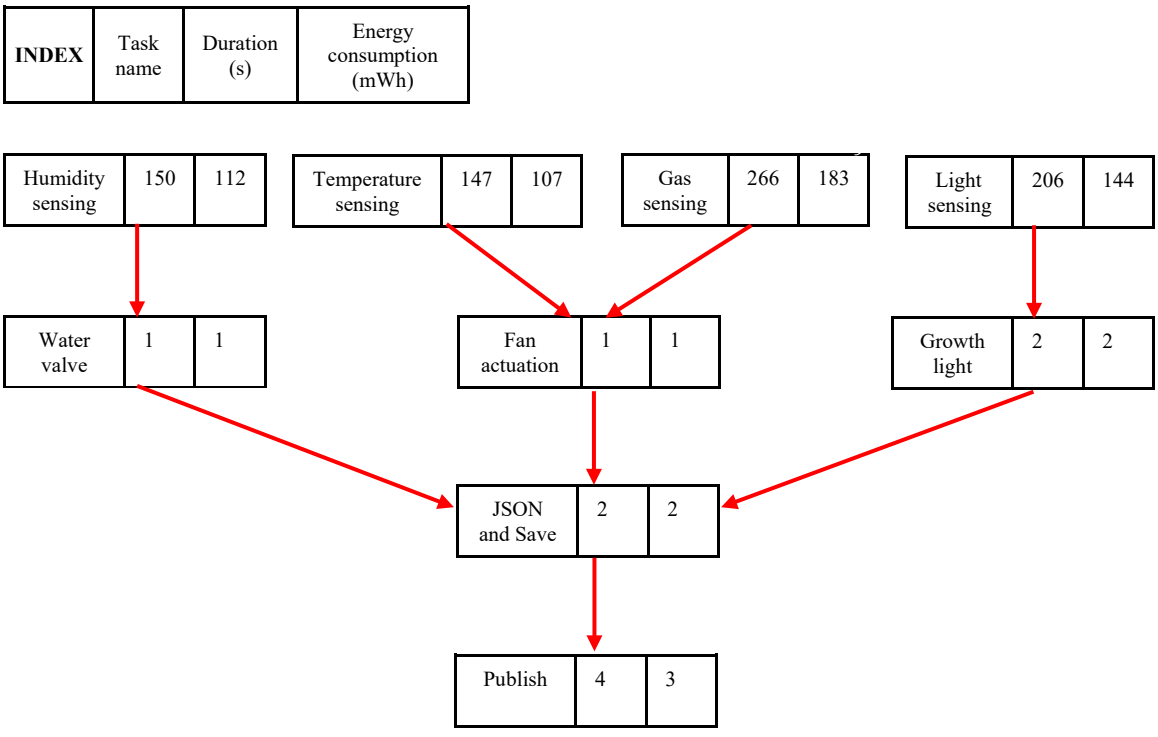


Figure 5.4-3: Smart gardening application

Humidity sensing: this task is responsible for measuring the relative humidity of the soil in (%). This block of code constitutes of sensor and pin initialization, function to read values, ADC conversion and data transmission. The task takes 150 seconds to execute and consumes 112 mWh.

Temperature sensing: this task is responsible for measuring the temperature in ˚C. The code is a duplicate of the humidity sensing. The task takes 147 seconds to execute and consumes 107 mWh.

Gas sensing: detects the concentration of methane and other gases in particles per million (ppm). The code is a duplicate of the humidity sensing. It takes 266 seconds to execute and consumes 183 mWh.

Light sensing: measures intensity of visible light in lux. The code is a duplicate of the humidity sensing . It takes 206 seconds to execute and consumes 144 mWh.

Water valve: This task compares the humidity values of the soil to a user threshold. If higher it turns on the valve and vice versa. It takes 1 second and 1 mWh.

Fan actuation: This task compares the temperature and gas values to a user threshold. If values are higher, it turns on the fan. It takes 1 second and consumes 1 mWh.

Growth light: this task compares the light intensity values to a user threshold. If lower it turns on the growth lights and vice versa. It takes 2 seconds and consumes 2 mWh.

JSON and Save: this task imports the sensed readings in a JSON format for publishing and stores it in a text file on the local machine. It takes 2 seconds and consumes 3 mWh.

Publish: sends the sensed data to ThingsBoard (open-source platform for data collection and analysis) via the MQTT protocol. It takes 4 seconds and consumes 3 mWh.

5.4.2 Challenges and Solutions

Only a few challenges were faced during the hardware implementation. In this section we outline some of the significant challenges and their solutions

1. ***Challenge***: difficulty in finding any real world open-source application, and the ones present are incompatible with the sensor node we have. This incompatibility stems from the need for different types of sensors or actuating devices.

   Solution: create our own application that suits our IoT node with the help of the in- house IoT smart gardening use case, to prove the functionality of our algorithm.

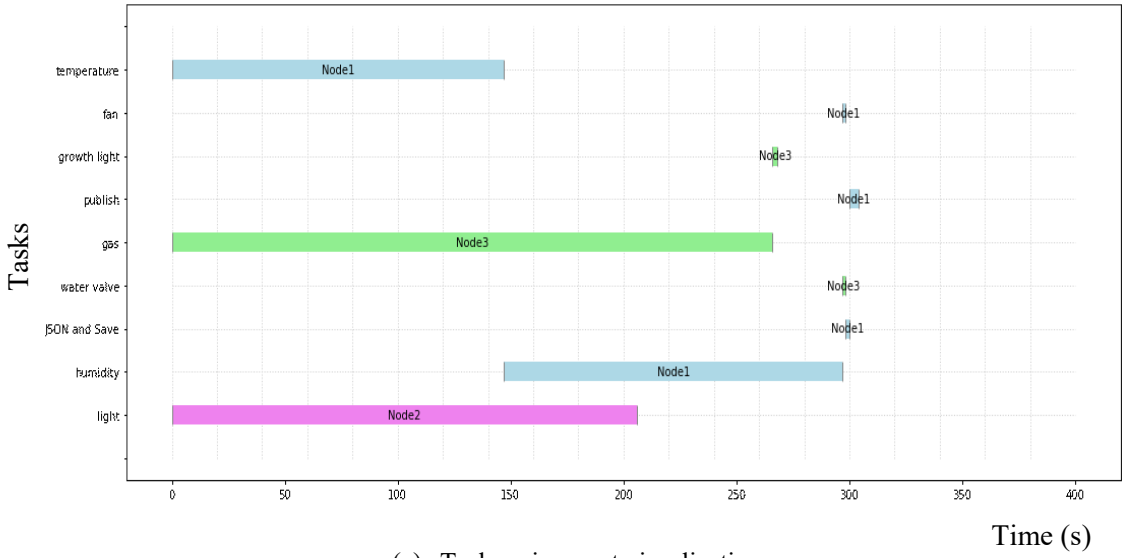
2. ***Challenge***: unable to measure the direct energy consumption of the tasks. Although the cumulative energy between the start and stop timestamps was recorded, the tasks' sole energy consumption was not. This is because other essential background applications were running during that time (e.g., virtual network computing, python 3 and python 2 executable files).

   Solution: for our setup, it is to our advantage to include these background consumptions in our readings since, they are consistent and are always running.

5.4.3 Implementation Results

After inputting the attained parameters from our implementation setup and running the algorithm, we attained the task assignment in Figure 5.4-4. For displaying purposes and consistency, three nodes were used as well. As seen in Figure 5.4-4 (a) the tasks are displayed in an increasing starting order. The duration of this application is 304 seconds. If we look at the tasks temperature, gas, and light, they were all distributed and executed simultaneously on the available nodes, this is possible due to their independencies. This

also shows that the algorithm is trying to reduce the duration given the constrained number of devices. If we consider the fan actuation task which is dependent on temperature and gas, we notice that it started at 297 seconds while, both temperature and gas ended at 147 and 266 seconds, respectively. This confirms that the algorithm respects the tasks dependencies. This is also seen in task JSON and Save, where it executes at 298 seconds after the completion of water valve (298 s), fan actuation (298 s) and Growth light (268 s). Finally, with the help of Figure 5.4-4 (b) we validate the one task per machine per time slot constraint, where 1 machine takes on 1 task at a specific time slot avoiding conflicts.



(a) Task assignment visualization

```
List of tasks in increasing start order:
From 0 to 147, temperature in Node1 in App 1
From 0 to 266, gas in Node3 in App 1
From 0 to 206, light in Node2 in App 1
From 147 to 297, humidity in Node1 in App 1
From 266 to 268, growth light in Node3 in App 1
From 297 to 298, fan in Node1 in App 1
From 297 to 298, water valve in Node3 in App 1
From 298 to 300, JSON and Save in Node1 in App 1
From 300 to 304, publish in Node1 in App 1
```

(b) Assignment of tasks in increasing order

Figure 5.4-4: Task assignment using three nodes

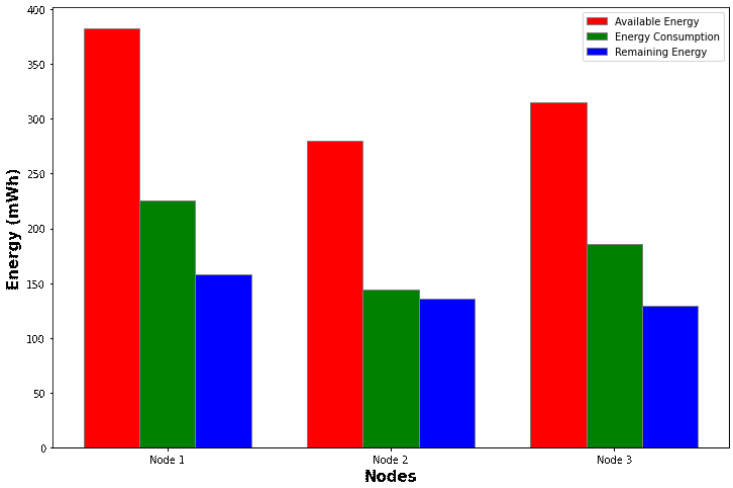


Figure 5.4-5: Energy profile of nodes

Figure 5.4-5 shows the energy profile of the nodes after the assignment. Each node constitutes of the available energy before the assignment (residual plus predicted) in the red bar, energy consumption due to task assignment in the green bar, and the remaining energy in the blue bar. It is seen that the node with the highest available energy (node 1) is assigned the most consumption, and the node with the least available energy is assigned the lowest consumption. It is also evident that the remaining energy of all nodes are close to each other with a MAD of 11.3.

5.4.4 Conclusion

In Section 5.4, we presented the implementation setup and results for Chapter 4. With the help of Section 5.3 and the implementation setup, we found the inputs required (predicted energy, duration, energy consumption and dependencies) by the algorithm to reduce the makespan and distribute the consumption, while respecting available energy and task dependencies. We were able reduce the makespan and distribute the energy consumption across nodes given their available energy and time availability.

Conclusion and Future Work

Conclusion

The field of incorporating energy harvesting units into IoT nodes is a prosperous one. With the continuous development and increased accessibility of these units that promise sustainability, it is only a matter of time until market adoption starts expanding and industrial entities gain interest. In this dissertation we proposed a comprehensive solution for incorporating energy harvesting units to IoT nodes in WSNs.

We first addressed the future time variability of irradiance and temperature by proposing a novel prediction model SARIMA-KF. We were able to forecast both parameters on IoT constrained devices using one week of historical data. On average, for the month of January, we were able to predict with an RMSE and MAE of 17.68 and 0.9, respectively. For the month of July, we predicted with a RMSE and MAE of 11.14 and 2.43, respectively.

Second, we addressed the anticipated SoC of these nodes by using a comprehensive (more accurate) and computationally light model (deployed on constrained nodes) of IoT based solar-EHMUs. We were able to achieve maximizing charging efficiency of 91%; assess the anticipated SoC of charge under different weather conditions for different seasons. We found that the maximum charge level reached in January and July was 100% and 29%, respectively.

Third, we tackled the energy management challenge by introducing ETCA scheduling. The algorithm was able to fulfil the applications within their deadlines and dependencies while, reducing the variance of energy across the WSN.

Finally, we conducted a real experiment to justify our work and display a proof of concept.

Future Work

This dissertation provided us with encouraging results and foundation to build upon. In this context, the following steps can be considered to further this study:

- Experiment with new prediction tools that address other independent variables such as geolocation and forecast from close by weather stations.
- Purchase the designated sensors (e.g., pyranometer) to enhance the prediction.
- Introduce floating voltage to the PWM model and acquire a MPPT to compare it to the MPPT simulation results.
- Incorporate low level scheduling such as DVFS since, newer version of RPI's support them.
- Implement the work in an actual small scale vertical farm to test its reliability and effectiveness.

List of Publications

1. **M. Azzam**, Z. Bouida, and M. Ibnkahla, “Irradiance and Temperature Forecasting for Energy Harvesting Units in IoT Sensors using SARIMA-KF”, IEEE Wireless Communication and Networking Conference (WCNC), Austin, TX, USA, pp. 1-6, April 2022.
2. **M. Azzam**, Z. Bouida, and M. Ibnkahla, “Ambient Source and Energy Prediction for Energy-Aware Task Scheduling in IoT”, in IEEE Sensors Journal, May 2022.

References

[1] T. D. Nguyen, J. Y. Khan, and D. T. Ngo, "A Distributed Energy-Harvesting-Aware Routing Algorithm for Heterogeneous IoT Networks," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 4, pp. 1115–1127, Dec. 2018, doi: 10.1109/TGCN.2018.2839593.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010, doi: 10.1016/j.comnet.2010.05.010.

[3] G. Kamalinejad *et al.*, "Wireless energy harvesting for Internet of Things," *IEEE Commun. Mag.*, no. 6, pp. 102–108, 2015, Accessed: Jun. 22, 2021. [Online]. Available: http://sro.sussex.ac.uk.

[4] T. D. Nguyen, J. Y. Khan, and D. T. Ngo, "Energy harvested roadside IEEE 802.15.4 wireless sensor networks for IoT applications," *Ad Hoc Networks*, vol. 56, pp. 109–121, Mar. 2017, doi: 10.1016/j.adhoc.2016.12.003.

[5] H. Chen, Y. Li, J. L. Rebelatto, B. F. Uchôa-Filho, and B. Vucetic, "Harvest-Then-Cooperate: Wireless-Powered Cooperative Communications," *IEEE Trans. Signal Process.*, vol. 63, no. 7, pp. 1700–1711, Apr. 2015, doi: 10.1109/TSP.2015.2396009.

[6] K. C. Barr and K. Asanović, "Energy-aware lossless data compression," *ACM Trans. Comput. Syst.*, vol. 24, no. 3, pp. 250–291, Aug. 2006, doi: 10.1145/1151690.1151692.

[7] G. De Meulenaer, F. Gosset, F.-X. Standaert, and O. Pereira, "On the Energy Cost of Communication and Cryptography in Wireless Sensor Networks."

[8] H. Rajab, T. Cinkler, and T. Bouguera, “IoT scheduling for higher throughput and lower transmission power,” *Wirel. Networks*, vol. 27, no. 3, pp. 1701–1714, Apr. 2021, doi: 10.1007/S11276-020-02307-1/TABLES/2.

[9] A. K. M. Al-Qurabat, C. A. Jaoude, and A. K. Idrees, “Two tier data reduction technique for reducing data transmission in IoT sensors,” *2019 15th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2019*, pp. 168–173, Jun. 2019, doi: 10.1109/IWCMC.2019.8766590.

[10] X. Shang, A. Liu, Y. Wang, Q. Xie, Y. Wang, and D. Chen, “Energy-Efficient Transmission Based on Direct Links: Toward Secure Cooperative Internet of Things,” *Wirel. Commun. Mob. Comput.*, vol. 2018, 2018, doi: 10.1155/2018/5012096.

[11] J. Li, B. N. Silva, M. Diyan, Z. Cao, and K. Han, “A clustering based routing algorithm in IoT aware Wireless Mesh Networks,” *Sustain. Cities Soc.*, vol. 40, pp. 657–666, Jul. 2018, doi: 10.1016/J.SCS.2018.02.017.

[12] M. M. Sandhu, S. Khalifa, R. Jurdak, and M. Portmann, “Task Scheduling for Simultaneous IoT Sensing and Energy Harvesting: A Survey and Critical Analysis,” Apr. 2020, Accessed: Aug. 17, 2021. [Online]. Available: https://arxiv.org/abs/2004.05728v1.

[13] A. Allavena and D. Mossé, “Scheduling of Frame-based Embedded Systems with Rechargeable Batteries,” 2001.

[14] S. Liu, Q. Qiu, and Q. Wu, “Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting,” *Proc. -Design, Autom. Test Eur. DATE*, pp. 236–241, 2008, doi: 10.1109/DATE.2008.4484692.

[15] S. Liu, J. Lu, Q. Wu, and Q. Qiu, “Harvesting-aware power management for real-time systems with renewable energy,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 8, pp. 1473–1486, 2012, doi: 10.1109/TVLSI.2011.2159820.

[16] Y. Tan and X. Yin, “A dynamic scheduling algorithm for energy harvesting embedded systems,” *EURASIP J. Wirel. Commun. Netw. 2016 20161*, vol. 2016, no. 1, pp. 1–8, Apr. 2016, doi: 10.1186/S13638-016-0602-8.

[17] T. Zhu, A. Mohaisen, Y. Ping, and D. Towsley, “DEOS: Dynamic energy-oriented scheduling for sustainable wireless sensor networks,” in *Proceedings - IEEE INFOCOM*, 2012, pp. 2363–2371, doi: 10.1109/INFCOM.2012.6195625.

[18] L. Guntupalli, J. Martinez-Bauset, F. Y. Li, and M. A. Weitnauer, “Aggregated packet transmission in duty-cycled WSNs: Modeling and performance evaluation,” *IEEE Trans. Veh. Technol.*, vol. 66, no. 1, pp. 563–579, Jan. 2017, doi: 10.1109/TVT.2016.2536686.

[19] K. S. Adu-Manu, N. Adam, C. Tapparello, H. Ayatollahi, and W. Heinzelman, “Energy-harvesting wireless sensor networks (EH-WSNs): A review,” *ACM Trans. Sens. Networks*, vol. 14, no. 2, Mar. 2018, doi: 10.1145/3183338.

[20] C. Renner and V. Turau, “Adaptive energy-harvest profiling to enhance depletion-safe operation and efficient task scheduling,” *Sustain. Comput. Informatics Syst.*, vol. 2, no. 1, pp. 43–56, Mar. 2012, doi: 10.1016/J.SUSCOM.2012.02.001.

[21] P. Sommer, B. Kusy, and R. Jurdak, “Power management for long-term sensing applications with energy harvesting,” *ENSSys 2013 - Proc. 1st Int. Work. Energy Neutral Sens. Syst.*, 2013, doi: 10.1145/2534208.2534213.

[22] P. Lee, M. D. Han, H. P. Tan, and A. Valera, “An empirical study of harvesting-

aware duty cycling in environmentally- powered wireless sensor networks," in *12th IEEE International Conference on Communication Systems 2010, ICCS 2010*, 2010, pp. 306–310, doi: 10.1109/ICCS.2010.5686442.

[23] M. Gregori and M. Payaro, "Energy-efficient transmission for wireless energy harvesting nodes," *IEEE Trans. Wirel. Commun.*, vol. 12, no. 3, pp. 1244–1254, 2013, doi: 10.1109/TWC.2013.011713.120621.

[24] A. Castagnetti, A. Pegatoquet, T. N. Le, and M. Auguin, "A joint duty-cycle and transmission power management for energy harvesting WSN," *IEEE Trans. Ind. Informatics*, vol. 10, no. 2, pp. 928–936, 2014, doi: 10.1109/TII.2014.2306327.

[25] C. A. Sanchez, O. Mokrenko, L. Zaccarian, and S. Lesecq, "A Hybrid Control Law for Energy-Oriented Tasks Scheduling in Wireless Sensor Networks," *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 6, pp. 1995–2007, Nov. 2018, doi: 10.1109/TCST.2017.2750999.

[26] C. Rusu, R. Melhem, and D. Mossé, "Multiversion scheduling in rechargeable energy-aware real-time systems," *Proc. - Euromicro Conf. Real-Time Syst.*, pp. 95–104, 2003, doi: 10.1109/EMRTS.2003.1212732.

[27] A. Caruso, S. Chessa, S. Escolar, X. Del Toro, and J. C. López, "A dynamic programming algorithm for high-level task scheduling in energy harvesting IoT," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2234–2248, Apr. 2018, doi: 10.1109/JIOT.2018.2828943.

[28] J. Leithon, L. A. Suarez, M. M. Anis, and D. N. K. Jayakody, "Task Scheduling Strategies for Utility Maximization in a Renewable-Powered IoT Node," in *IEEE Transactions on Green Communications and Networking*, Jun. 2020, vol. 4, no. 2,

pp. 542–555, doi: 10.1109/TGCN.2019.2959730.

[29] “State-Space Modelling.” https://kevinkotze.github.io/ts-4-state-space/ (accessed Jan. 16, 2022).

[30] C. Bergonzini, D. Brunelli, and L. Benini, “Algorithms for harvested energy prediction in batteryless wireless sensor networks,” *3rd Int. Work. Adv. Sensors Interfaces, IWASI 2009*, pp. 144–149, 2009, doi: 10.1109/IWASI.2009.5184785.

[31] S. Atique, S. Noureen, V. Roy, V. Subburaj, S. Bayne, and J. MacFie, “Forecasting of total daily solar energy generation using ARIMA: A case study,” in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019*, Mar. 2019, pp. 114–119, doi: 10.1109/CCWC.2019.8666481.

[32] M. Zhang, “Time Series: Autoregressive models AR, MA, ARMA, ARIMA Overview 1 Introduction of Time Series Categories and Terminologies White Noise and Random Walk Time Series Analysis,” 2018.

[33] A. E. Yahaya, E. H. Etuk, N. Kingdom, and N. W. Chimee, “ARIMA Model for Gross Domestic Product (GDP): Evidence from Nigeria,” *Arch. Curr. Res. Int.*, pp. 49–61, Dec. 2020, doi: 10.9734/acri/2020/v20i730213.

[34] “Strict Stationarity .” http://www.maths.qmul.ac.uk/~bb/TimeSeries/TS_Chapter4_2.pdf (accessed Jul. 01, 2021).

[35] “Validated historical solar irradiance and meteo data.” https://solcast.com/historical-and-tmy/ (accessed Feb. 14, 2022).

[36] “Kalman Filtering: Theory and Practice with MATLAB, 4e - MATLAB &

Simulink Books.” https://www.mathworks.com/academia/books/kalman-filtering-theory-and-practice-using-matlab-grewal.html (accessed Jul. 01, 2021).

[37] “Kalman Filter Tutorial.” https://www.kalmanfilter.net/default.aspx (accessed Aug. 01, 2021).

[38] “Evaluation Metrics for Machine Learning | by Frankie Cancino | Aug, 2021 | Towards Data Science.” https://towardsdatascience.com/evaluation-metrics-for-machine-learning-2167fca1a291 (accessed Aug. 06, 2021).

[39] H. Elahi, K. Munir, M. Eugeni, S. Atek, and P. Gaudenzi, “Energy harvesting towards self-powered iot devices,” *Energies*, vol. 13, no. 21, p. 5528, Oct. 2020, doi: 10.3390/en13215528.

[40] “Silicon - Element information, properties and uses | Periodic Table.” https://www.rsc.org/periodic-table/element/14/silicon (accessed Aug. 09, 2021).

[41] Vinod, R. Kumar, and S. K. Singh, “Solar photovoltaic modeling and simulation: As a renewable energy solution,” *Energy Reports*, vol. 4, pp. 701–712, Nov. 2018, doi: 10.1016/J.EGYR.2018.09.008.

[42] E. M. Salilih and Y. T. Birhane, “Modeling and Analysis of Photo-Voltaic Solar Panel under Constant Electric Load,” *J. Renew. Energy*, vol. 2019, pp. 1–10, Aug. 2019, doi: 10.1155/2019/9639480.

[43] “Implement PV array modules - Simulink.” https://www.mathworks.com/help/physmod/sps/powersys/ref/pvarray.html (accessed Aug. 09, 2021).

[44] “Buck Converters.” https://learnabout-electronics.org/PSU/psu31.php (accessed Jul. 04, 2021).

[45] S. Chander, P. Agarwal, and I. Gupta, “Design, modeling and simulation of DC-DC converter,” *2010 9th Int. Power Energy Conf. IPEC 2010*, pp. 456–461, 2010, doi: 10.1109/IPECON.2010.5697039.

[46] N. Mohan, T. M. Undeland, and W. P. Robbins, “DC/DC Converters ,” in *Power Electronics Convertors, Applictions, and Design*, 3rd ed., New York: Wiley., 2002, p. 824.

[47] “MPPT Solar Charge Controller Model - File Exchange - MATLAB Central.” https://www.mathworks.com/matlabcentral/fileexchange/73115-mppt-solar-charge-controller-model (accessed Aug. 09, 2021).

[48] A. Joseph and M. Shahidehpour, “BATTERY STORAGE SYSTEMS IN ELECTRIC POWER SYSTEMS.”

[49] “Generic battery model - Simulink.” https://www.mathworks.com/help/physmod/sps/powersys/ref/battery.html (accessed Aug. 11, 2021).

[50] K. C. Divya and J. Østergaard, “Battery energy storage technology for power systems-An overview,” *Electr. Power Syst. Res.*, vol. 79, no. 4, pp. 511–520, Apr. 2009, doi: 10.1016/J.EPSR.2008.09.017.

[51] O. Tremblay, L. A. Dessaint, and A. I. Dekkiche, “A generic battery model for the dynamic simulation of hybrid electric vehicles,” *VPPC 2007 - Proc. 2007 IEEE Veh. Power Propuls. Conf.*, pp. 284–289, 2007, doi: 10.1109/VPPC.2007.4544139.

[52] I. Baboselac, Z. Hederić, and T. Benšić, “MatLab simulation model for dynamic mode of the Lithium-Ion batteries to power the EV,” *undefined*, 2017.

[53] P. H. Soe and T. M. Htike, “Critical path analysis programming method without network diagram,” *MATEC Web Conf.*, vol. 192, p. 01027, Aug. 2018, doi: 10.1051/MATECCONF/201819201027.

[54] “The ABCs of the Critical Path Method.” https://hbr.org/1963/09/the-abcs-of-the-critical-path-method (accessed Aug. 29, 2021).

[55] P. Michael, “A Conversion Guide: Solar Irradiance and Lux Illuminance | IEEE DataPort,” 2019. https://ieee-dataport.org/open-access/conversion-guide-solar-irradiance-and-lux-illuminance (accessed Dec. 27, 2021).

[56] “Solar Radiation - Hourly data for Ottawa (Kanata - Orléans).” https://ottawa.weatherstats.ca/charts/solar_radiation-hourly.html (accessed Feb. 26, 2022).

[57] “How HOMER Calculates the PV Array Power Output.” https://www.homerenergy.com/products/pro/docs/latest/how_homer_calculates_the_pv_array_power_output.html (accessed Jan. 02, 2022).

[58] “Review of DC to DC buck converter based on LM2596 - Joe’s Hobby Electronics.” https://www.hobbyelectronics.net/review_dctodcmodule.html (accessed Jan. 02, 2022).

[59] “What You Should Know About Vertical Farming.” https://www.thebalancesmb.com/what-you-should-know-about-vertical-farming-4144786 (accessed Jan. 04, 2022).

[60] Z. Bouida *et al.*, “Carleton-Cisco IoT Testbed: Architecture, Features, and Applications,” pp. 1–6, Jan. 2022, doi:10.1109/GCWKSHPS52748.2021.9681940.