

**ARTIFICIAL INTELLIGENT INTEGRATED
SUN-TRACKING SYSTEM WITH SUN AND
CLOUD POSITIONS PREDICTION**

HUANG DICK SHEN

UNIVERSITI TUNKU ABDUL RAHMAN

**Artificial Intelligent Integrated Sun-Tracking
System with Sun and Cloud Positions Prediction**

Huang Dick Shen

**A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Electrical and Electronics
Engineering with Honours**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

May 2024

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :  _____

Name : Huang Dick Shen _____

ID No. : 1903824 _____

Date : 20/05/2024 _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**Artificial intelligent integrated sun-tracking system with sun and cloud positions prediction**” was prepared by **HUANG DICK SHEN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Electrical and Electronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature : **K.K.Chong**

Supervisor : _____
Date : _____

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2024, HUANG DICK SHEN. All right reserved.

ACKNOWLEDGEMENTS

First of all, I would like to express my greatest gratitude and appreciation to my supervisor, Prof. Dr. Chong Kok Keong for providing me the opportunity to research on the tittle “Artificial Intelligent Integrated Sun-Tracking System with Sun and Cloud Positions Prediction”. In this research, I have gained a lot of experience and knowledge. I am grateful to all the guidance and advice provided throughout this research development.

I would also like to extend my gratitude to my senior, Tiow Yik Hong of Universiti Tunku Abdul Rahman for the discussion and support throughout the research. Lastly, I would like to express my gratitude to my parents and friends for all their supports and advice throughout the project study.

ABSTRACT

The purpose of this study is to develop an artificial intelligent (AI) -integrated sun-tracking system and cloud position prediction system. These systems are to be applied on concentrated photovoltaic (CPV) to improve the efficiency in solar power generation. In this study, YOLOv8 is chosen as object detection model to recognize and locate the sun position. After successfully tracked the sun's coordinate, Q-learning will be applied to control the motor in order to follow the sun. In addition, YOLOv8 also be applied to recognize the position of clouds. The purpose of cloud tracking is to encounter the problem of lost tracking of sun when it is shaded by clouds during cloudy weather. YOLOv8 will first obtain the position of the cloud, then calculation will be made according to the cloud movement and speed to predict the cloud shading time. The cloud shading time will then be applied to calculate the predicted sun reappear position. After that, the CPV will turns toward and stand by at the predicted reappear sun position. This is important to shorten the response time after lost track of sun to increase the efficiency of power generated by CPV system. Furthermore, this project also using 180° fisheye lens for a wider view, so that the larger cloud image can be captured. In result, the YOLOv8 model trained have an accuracy of 0.69% on cloud detection, and 100% on sun detection. The Q-learning training result also shown that the agent is able to move towards the target in the end of 1,000,000 episodes. The fish-eye lens had improved the cloud detection by widening the field of view of the camera module. Furthermore, the solar irradiance results also proved the accuracy of the sun object detection model. While the sun position prediction result had shown the percentage error ranging from 11% to 25%, and 43% during rapid change of weather. Lastly, the implementation of artificial intelligence had improved the efficiency of concentrated photovoltaic system during cloudy day. For future improvement, wind speed sensor, real-time satellite forecast system can be implemented for higher accuracy in prediction.

TABLE OF CONTENTS

DECLARATION		i
APPROVAL FOR SUBMISSION		ii
ACKNOWLEDGEMENTS		iv
ABSTRACT		v
TABLE OF CONTENTS		i
LIST OF TABLES		iii
LIST OF FIGURES		iv
LIST OF SYMBOLS / ABBREVIATIONS		vi
LIST OF APPENDICES		vii
CHAPTER		
1	INTRODUCTION	1
1.1	General Introduction	1
1.2	Importance of the Study	2
1.3	Problem Statement	2
1.4	Aim and Objectives	3
1.5	Scope and Limitation of the Study	3
1.6	Contribution of the Study	4
1.7	Outline of the Report	5
2	LITERATURE REVIEW	6
2.1	Introduction	6
2.2	Artificial Intelligence	6
2.2.1	Object Detection Model	7
2.2.2	Versions and Evolution of YOLO	9
2.3	Sun-Tracking System	11
2.3.1	Recent Development of Sun Tracking System	13
2.4	Cloud Prediction System	15
2.4.1	Current Existing Cloud Prediction System	15

2.5	Summary	16
3	METHODOLOGY AND WORK PLAN	17
3.1	Introduction	17
3.2	AI Sun–Tracking with Sun and Cloud Position Prediction	17
3.2.1	Selection of Hardware and Software	17
3.2.2	Design Architecture and Practical Process	19
3.2.3	Custom Dataset of Sun and Clouds	23
3.2.4	Model Training for Object Detection	23
3.2.5	Q-learning Training	24
3.2.6	Calibration on Sun Moving Speed and Direction	25
3.2.7	Cloud Movement Predictions	26
3.2.8	Sun-Tracking and Position Predictions	29
3.3	Summary	30
4	RESULT AND DISCUSSION	31
4.1	Introduction	31
4.2	Result of Machine Learning Training	31
4.2.1	YOLOv8 Training Result	31
4.2.2	Q-learning Training Result	34
4.3	Fisheye Lens Assisted in Object Detection	35
4.4	Solar Irradiance Analysis with Solar Tracking System	37
4.5	Sun and Cloud Position Prediction	39
4.6	Summary	41
5	CONCLUSION AND RECOMMENDATIONS	43
5.1	Conclusions	43
5.2	Recommendation for Future Work	44
	REFERENCES	45
	APPENDICES	49

LIST OF TABLES

Table 2.1: Table of Comparison Between One-Stage and Two-Stage Detector	8
Table 2.2: Versions of YOLO with Corresponding Improvement	10
Table 3.1: List of Hardware Components	18

LIST OF FIGURES

Figure 1.1: Graph of PV Power Output During Different Weather.	3
Figure 2.1: Structure of YOLOv4	9
Figure 2.2: Comparison of Accuracy (Augmented Startups, 2023)	11
Figure 2.3: Comparison of Speed (Augmented Startups, 2023)	11
Figure 2.4: The Structure Design of a Dual-Axis Tracker (Jamroen et al., 2020)	13
Figure 2.5: Four Identical Time Intervals During Daylight	13
Figure 2.6: Circuit Design of Photosensors (Al-Mohammad, 2004)	14
Figure 2.7: Neural Network Identification Scheme (EI Shenawy et al., 2012)	15
Figure 2.8: Illustration of Motion Vector Prediction (Hu et al., 2018)	16
Figure 2.9: Prediction of Cloud Trajectory (Hu et al., 2018)	16
Figure 3.1: The Comparison of Field of View Before and	18
Figure 3.2: Block Diagram of the Design	21
Figure 3.3: Flow Chart of the Design	22
Figure 3.4: Classification of Weather (Dreamstime, n.d.)	23
Figure 3.5: Reward and Penalty Points for Q-learning.	24
Figure 3.6: Reward and Penalty system for Q-learning	25
Figure 3.7: Illustration of the Calculation for Sun Speed Calibration.	26
Figure 3.8: An Example Situation of a Moving Cloud Detection and Sun Tracking Stream from Camera	28
Figure 3.9: Situation when Cloud Covered the Sun	28
Figure 3.10: One Minute After Cloud Shading	28
Figure 3.11: Calculation from the Two Frames of Same Cloud	29
Figure 3.12: Calculation to Obtain Cloud Shading Time	29

Figure 3.13: Calculation of Predicted Sun Reappear Position	30
Figure 4.1: Training of YOLOv8 Model on Google Colab with 150 Epochs.	32
Figure 4.2: Result of Sun and Cloud YOLOv8 Detection.	33
Figure 4.3: Confusion Matrix of Sun and Cloud YOLOv8 Detection.	33
Figure 4.4: Sun and Cloud Detection on Sky Images.	34
Figure 4.5: The Reward Obtained for Every 50,000 Episodes	35
Figure 4.6: Image Captured by the Webcam Before Fisheye Lens is Installed	36
Figure 4.7: Image Captured by the Webcam After Fisheye Lens is Installed	36
Figure 4.8: Cloud Detection of Yolov8 Model Before Installation of Fisheye Lens	37
Figure 4.9: Cloud Detection of Yolov8 Model After Installation of Fisheye Lens	37
Figure 4.10: Graph of Solar Irradiance and YOLOv8 Sun Detection vs Time	38
Figure 4.11: The Trajectory Motion Prediction of Sun After Covered by Cloud	39
Figure 4.12: Percentage Error vs Time Graph During Cloudy Day (27 April 2024)	41

LIST OF SYMBOLS / ABBREVIATIONS

S_{sun}	Displacement of sun during calibration, unit
S_1	Displacement of cloud in 1 minute, unit
S_2	Displacement from sun to (X_3, Y_3) , unit
S_3	Predicted displacement, unit
V_s	Velocity of sun, unit/min
V_c	Velocity of cloud, unit/min
X_{sun1}	Coordinate X before calibration
X_{sun2}	Coordinate X after calibration
Y_{sun1}	Coordinate Y before calibration
Y_{sun2}	Coordinate Y after calibration
X_1	Coordinate X before 1 minute
X_2	Coordinate X after 1 minute
Y_1	Coordinate Y before 1 minute
Y_2	Coordinate Y after 1 minute
X_{sun}	Coordinate X of sun before lost track
Y_{sun}	Coordinate Y of sun before lost track
X_3	Coordinate X aligned on box
Y_3	Coordinate Y aligned on box
θ_{sun}	Sun direction, degree ^o
t	Cloud shading time

LIST OF APPENDICES

Appendix A: Solar Irradiance and Prediction on 27 April 2024	49
Appendix B: Code of Q-learning Training	50
Appendix C: Code of CPV Control System	58

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Energy is the lifeblood of modern society, transportation, communications systems and fuelling industries. It is the force that drives progress and underpins economic growth. However, most of the source of energy in this century are non-renewable, such as fossil fuels, coal, and natural gasses. The extremely drop of sources in 21's century had warned the worldwide that energy landscape must undergo a transformative change. The quest for energy sources that power our life while preserving the planet had become limited. In such, renewable energy such as sunlight, wind and water had become a hot topic for study and research. Among all those sources, solar energy is the easiest to obtain as sunlight is always available.

Solar power can be easily achieved by harnessed through photovoltaic (PV) panel, which converting sunlight into electricity. Although this technology had been found in the year 1839 by Edmond Becquerel, it is still not the major source for most country (Office, 2023). This is happened due to the mis-estimated and unpredictable weather change which significantly cause the low efficiency of solar power generator. When the world still heads aching with this century problem, the rapid growth of artificial intelligence (AI) had become beacon of hope. In this era of advanced technology, a new hope arises with the fusion of two fields: artificial intelligence and solar energy. The introduce of AI with powerful calculation had unlocked the potential of solar power with its high accuracy prediction through data learning.

By integrating AI into photovoltaic (PV) panel, the solar panel is upgraded such that providing an 'eye' to the system. An object detection AI model can be planted to track the position of sun. However, achieving sun tracking alone is not yet solving the problem. The main causes of low efficiency in PV system are due to the unpredictable weather change. The sudden cover of sun by clouds may causing loss of sun tracking. In order to enhance the performance of PV, cloud prediction can be integrated into the system. This can be done by using the AI object detection model to detect the cloud, then predict

the cloud moving direction, moving speed, and estimated covering time through calculations. By integrating cloud tracking and sun positioning system, the efficiency of PV system can be improved.

1.2 Importance of the Study

As the efficiency of photovoltaic system is a major factor affecting the future of solar power renewable energy, it is important to have high and stable performance system. In this regard, AI-integrated sun tracking system plays an important role in tracking the sun position during daytime. This function can cooperate with Q-learning AI model to achieve automatic adjustment of CPV panels to always facing the sun. Additionally, cloud tracking and prediction AI are also crucial to estimate the covering of cloud. By integrating a cloud prediction system, the CPV system can accurately calculate the position of sun reappear using the cloud's shading time and moving direction obtain from the prediction. Besides, the cloud shading time, sun position and solar irradiance are also an important data in solar generation study.

1.3 Problem Statement

Current efficiency of PV system has faced a critical bottleneck as a primary source of power generation. Although there are current existing AI-integrated sun tracking PV system, the efficiency of the PV is still below the required demand. The main reason causing the problem is the loss track of sun during unstable weather such as cloudy and windy day where the sun is possibly shaded by clouds. This problem is further illustrated by the graph shown in Figure 1.1, which display the PV power output during different weather conditions. From Figure 1.1, it is clearly seen that the performance of PV is highly unstable during cloudy and rainy day.

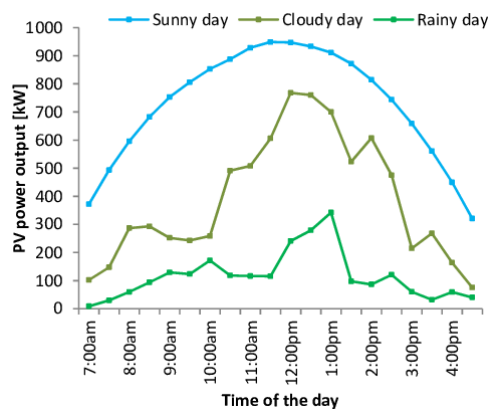


Figure 1.1: Graph of PV Power Output During Different Weather.
(ResearchGate, 2016)

The same problem is encountered by the AI sun tracking integrated CPV system. Once the sun is lost from tracking, the AI-integrated CPV system may stop turning and remains in the same position. When the cloud goes and the sun reappears, the CPV system will need time to react and reorient towards the sun. This time-consuming issue had caused the CPV system unable to receive the sunlight immediately, leading to a drop in efficiency. This issue could be even more critical when coming to a larger and heavier CPV system where lots of time is wasted in aligning the CPV panels with the sun.

1.4 Aim and Objectives

The aim of this study is to address the performance issue of CPV system by integrating an AI object detection model. The goal of this project is to increase the efficiency of CPV system during bad weather. The objective of this project is as follows:

1. To integrate AI sun detection model into CPV system to achieve sun tracking feature.
2. To develop a cloud prediction algorithm using AI cloud detection model
3. To develop a CPV control system with cloud prediction and sun tracking algorithm cooperating with each other.

1.5 Scope and Limitation of the Study

In this project with the title ‘Artificial Intelligent Integrated Sun-Tracking System with Sun and Cloud Positions Prediction’, the scope of the project is to

develop a system to predict sun and cloud's position using AI object detection model. The scope including calculations to obtain predicted cloud shading time and predicted position of sun reappearance. These efforts are aimed to enhance the efficiency of CPV system.

However, for the limitation of the tittle, the wind speed was assumed to be constant throughout the entire process, from cloud shading until the sun reappears. To incorporate the wind factor, wind speed sensors would be required, or a forecast system would need to be integrated into the system, where this part will not be covered in this project. Besides, as long as the sun cannot be detected by the AI object detection model, it will be considered as no sun detected. This limitation arises from the working algorithm of AI model.

1.6 Contribution of the Study

This study had contributed to solve the limitation of the solar-tracking CPV system. During cloudy day, CPV system may lost track the sun when the sun is covered by clouds. This causes the CPV system consume time to retracking the sun when the sun is reappeared. In order to solve this problem, this project had integrated sun and cloud position prediction system to predict the sun reappear position. This could help the CPV system to quickly recapture the sun once the sun is reappeared.

Besides, this project had also improved the performance of the AI sun and cloud detection model, by implementing the latest detection model released in 2023, the YOLOv8. The implementing of YOLOv8 will improve the accuracy of the solar tracking system and prediction system.

Moreover, this project will also implement new reward and penalty system to the Q-learning algorithm. The dual-axis CPV motor with previously only able to turn 4 directions in each step is aimed to be upgraded to 8 directions available in each step. This could improve the efficiency of the CPV motor to find to shortest path towards the sun.

Apart from that, solar irradiance throughout a cloudy day is also measured and discussed to study the performance of the CPV system. Lastly, the study on the solar irradiance during cloudy day can contribute important data to the industry for solar irradiance prediction and solar energy research.

1.7 Outline of the Report

This report describes the study on title ‘Artificial Intelligent Integrated Sun-Tracking System with Sun and Cloud Positions Prediction’ which composed the following chapters:

Chapter 1 describes the introduction of the title which consist of background, problem statement, objective, limitation and scope of study.

Chapter 2 describes the literature review of artificial intelligence (AI), sun tracking system, and cloud prediction system. The study consists of historical background, algorithm of the AI model and recent development of PV system.

Chapter 3 describes the methodology of the project which includes AI object detection model training, design architecture, sun and cloud movement calculations, and hardware setup.

Chapter 4 describes the result and discussion of the project which consist of YOLOv8 training result, Q-learning training, fisheye lens assisted in object detection, solar irradiance analysis, and sun and cloud position prediction result.

Chapter 5 describes the conclusion of the project and the recommendation for future work is discussed.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this era with rapid growth in artificial intelligence (AI), people are striving to apply it on automation system. This is because automation system with processing capabilities can achieve fully automatic control without human's help. This same applies to solar power field, where efforts are being made to achieve the automatic control PV system for sun tracking. In recent years, many research has been conducted on integrating AI with solar power plant. In this topic, the background study of AI, sun tracking, and cloud prediction system will be discussed.

2.2 Artificial Intelligence

Artificial intelligence (AI) refers to the development of computer system which can perform tasks that typically require human intelligence. In recent years, AI has gain popularity for integration into automation system, due to its abilities in problem-solving, learning and decision making. There are few categories and types of AI which currently in use including machine learning, deep learning, reinforcement learning, and robotics. With these capabilities, people are striving to integrate AI into automation system. In solar field, AI has been implemented to achieve sun tracking with automatic control of motors. The introduction of AI into photovoltaic (PV) cells has significantly increased the overall efficiency of solar power generation. However, training a good AI model could be challenging.

The AI model training process includes data collection, data preprocessing, model building, training validation and model testing (Javaid.S.,2023). For data collections, AI required a large set of labelled datasets in order to achieve high accuracy performance. For example, an AI object detection model would need lots of labelled images of the targeted object. The accuracy of the trained model depends on the quality and quantity of data, where the targeted object must clarify clearly in the dataset. While data preprocessing

process involves resizing image, enhancing and cleaning data to prepare for model training.

When comes to model selection, several types of algorithms can be chosen based on the complexity of problem, size and structure of data, and accuracy of the task. One of the most commonly used is machine learning, which is a subset of AI that focuses on statistical models. While deep learning is a subset of machine learning which uses artificial neural networks (ANN) to analyse and learn from data. Deep learning is popular for applications in object detection model, such as YOLO (You Only Look Once), which able to perform real-time object recognition (Chablani.M., 2017).

Lastly, training validation take place after the initial training phase. This step is important to check the performance of the trained AI model. A new set of data, also known as validation dataset, will be run through to check the accuracy of the model. In this stage, adjustment of dataset will be performed in order to meet the requirement of the task. In summary, an AI model training consist of five crucial steps. The rapid improvement and evolution of AI technology, especially in machine learning and deep learning, will greatly benefits data processing task.

2.2.1 Object Detection Model

An AI object detection model is a fundamental task in computer vision, which enabling to identify and localize the objects within an image or video stream. In recent years, object detection model has been integrated into sun tracking system to develop high-performance solar power generator.

Early object detection models such as Viola-Jones' face detection algorithm are developed with slow, inaccurate and low performance on unfamiliar data (Zaidi et al., 2022). These issues are then solved with the introduction of convolutional neural network (CNNs), which leads to the speed improvement in deep learning. CNNs is a neural network that designed to process grid-like data such as images and videos (Peng et al., 2017). It is a fundamental of deep learning which can increase the effectiveness for works related to computer vision. This invention had leads to the exponential growth in object detection model by improving the detection accuracy. Over time, real-

time deep learning-based object detection model, YOLO (You Only Look Once) were introduced with accurate positioning and speed performance.

Generally, the object detection model classified into one-stage and two-stage object detection algorithm. One-stage detector contains single feed-forward fully convolutional network which enables bounding boxes and object classifications directly predicted on image (Carranza-Garcia et al., 2020). This feature provides high efficiency by reducing the computational time, making it suitable for real-time applications. The most common model with one-stage detector is YOLO, which also the first proposed single unified architecture.

While two-stage detector algorithm have a higher accuracy with two-step process. In the first step, preliminary test is carried out to generate a set of region of interest (RoIs) where all positive samples are removed. Then, the RoIs will pass over to second stage, to undergo regional classification and location refinement. This detector algorithm had given benefits of more accurate localization but required more processing time due to the complexity. The most common example for two stage object detection algorithm is Faster R-CNN. In summary, the comparison for two type of algorithm is shown in Table 2.1.

Table 2.1: Table of Comparison Between One-Stage and Two-Stage Detector

Aspect	One-stage detector (YOLO)	Two-stage detector (Faster R-CNN)
Localization Accuracy	Lower accuracy for small objects.	Higher accuracy due to two steps of refinement.
Run time required	Shorter time for one step object localization.	Longer period of time for the additional RoIs step.
Model complexity	Less complexity which leads to higher efficiency.	More complex due to two-stage process.
Real-Time Applications	Well-suited due to its fast performance on videos and images.	Not suitable due to the complexity of the algorithm.

2.2.2 Versions and Evolution of YOLO

You Only Look Once (YOLO) is a deep learning-based object detection algorithm which first published in the year 2015 (Jiang et al., 2022). The invention of YOLO is to target a small size model yet fast calculation speed in object detection. It is applied on object tracking system in recent years due to fast image processing capability. YOLO achieves the objective by directly output bounding of box through neural network. This advantage brings YOLO with ability to suit in real-time applications. Apart from that, YOLO reduces the detecting error on background by using global image for detection. However, this also bring negative effect where the accuracy of YOLO will be limited.

Over the years, YOLO had undergone several evolutions. The original version of YOLO architecture contains 24 convolution layers and two connected layers. This version of YOLO has a major issue which is low accuracy in positioning and lower recall rate. This led to the development of YOLOv2 which have slightly improve the defects by using new classification model, Darknet-19. While the development of YOLOv3 uses Darknet-53, with multi-scale features. In the year 2020, YOLOv4 has developed with greater focus on data comparison. YOLOv4 changed the overall detector framework into Input, Backbone, Neck and Head. When images are input to YOLOv4 with pre-processed size, Backbone will capture hierarchical features from multiple scales. Different from YOLOv3, version 4 is using enhanced version of Darknet-53, which is CSP Darknet53. The Neck will be act as an intermediate between Backbone and Head, while Head will be responsible to make predictions through bounding box and classifications. The structure of YOLOv4 is shown in Figure 2.1.

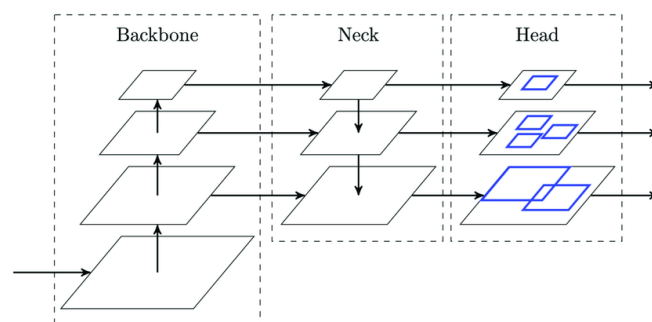


Figure 2.1: Structure of YOLOv4

The released of YOLOv5 improved the performance of YOLOv4 with faster model training and user-friendly coding environment. Overall, the versions from YOLO to YOLOv5 can be summarise in Table 2.2.

Table 2.2: Versions of YOLO with Corresponding Improvement

Versions of YOLO	Improvements
YOLO	Achieve object detection with small model and faster speed
YOLOv2	Improve in accuracy and overall performance
YOLOv3	Provide multi scale detection
YOLOv4	Improved feature extraction and enhance performance
YOLOv5	Flexible control of model size and user friendly

In the year 2022, the introduction of YOLOv7 had surpassed all known object detection model with high processing speed and high accuracy (Gasparovic et al., 2023). YOLOv7 are trained using MS COCO dataset and the architecture of the algorithm had upgraded. Similar to YOLOv4, Bag of Freebies are used as a general framework of training strategies to obtain high accuracy detection yet not affecting the overall processing speed. However, the training speed will be slightly lower compared to other existing model.

While for YOLOv8, anchor-free architecture, enhancement of backbone and multi scale prediction ability had leads YOLOv8 to perform task faster with more accurate detection than YOLOv7. A study is done by Augmented Startups., (2023) in order to test the object detection accuracy and speed between YOLOv7 and YOLOv8 as shown in Figure 2.2 and Figure 2.3. From Figure 2.2, it shows that YOLOv8 has the highest accuracy with the limited amount of parameter while Figure 2.3 shows YOLOv8 can achieve the same accuracy as other models in a shorter period of time. Unlike YOLOv7, YOLOv8 able to detect smaller object with more irregular shapes, such as clouds.

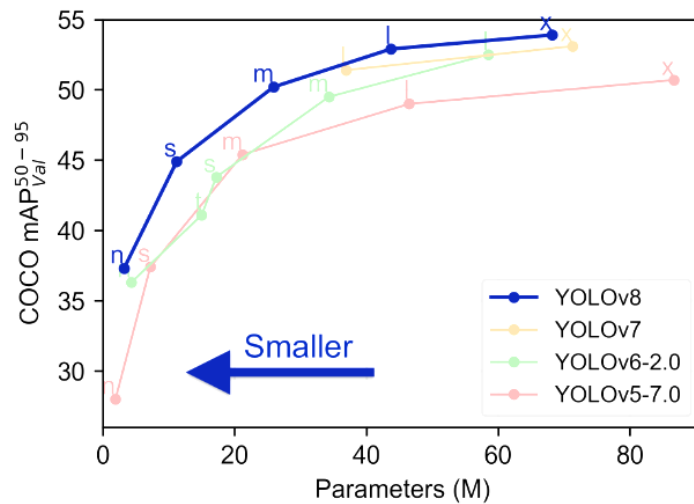


Figure 2.2: Comparison of Accuracy (Augmented Startups, 2023)

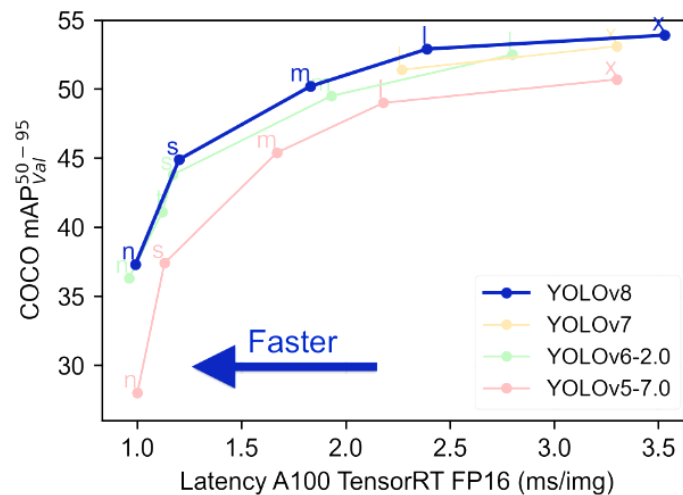


Figure 2.3: Comparison of Speed (Augmented Startups, 2023)

In summary, YOLOv7 and YOLOv8 had significantly improved in object detection accuracy and processing time. The simple structure of the model had become reason of two models being highly applied on daily life, especially real time object detection. This is also a reason these two models are mostly chosen to apply in solar tracking system.

2.3 Sun-Tracking System

Sun tracking system, also known as solar tracking system, is a mechanism to orient a solar photovoltaic (PV) or sun reflector towards the sun. The purpose

of this mechanism is to maximise the energy output from the solar device by ensuring the PV panels receive direct sunlight throughout the day. As the position of sun changes throughout the day, a solar tracking system is important to adjust the angle of the PV cells so that it aims precisely at the sun (Abdallah & Nijmeh, 2003). This idea is came out in the mid-20th century, with the expansion of solar energy and advanced electronics that can leads to accurate tracking mechanism.

Solar tracking is commonly implemented in photovoltaic (PV) system and concentrated photovoltaic (CPV) system. In PV system, this generator converts sunlight directly into electricity using solar cells. This type of mechanism can result in high efficiency of PV panels. However, this method only applicable on areas with high solar insolation. While for CPV system, mirrors or lenses are used to reflect and focus the sun image on a small spot where multi-junction solar cells installed. This type of system requires sun tracking algorithm to tilt the entire structure with mirrors towards the sun so that maximum sunlight can be reflected.

In mechanism design, solar tracking system is classified into single-axis and dual-axis. For single axis, the trackers consist of one degree freedom which act as an axis of rotation, aligned along the true north meridian (Ray & Tripathi, 2016). While dual-axis tracker has two degrees of freedom for rotation, where the primary axis is fixed to the ground, and it will be the reference for secondary axis. This mechanism allowed the tracker to follow the sun vertically and horizontally. Compared to single-axis, dual-axis are far more flexible with extra degree of movement to parallel the normal of sun to normal of CPV. A sample design of dual-axis tracker is as shown in Figure 2.4.

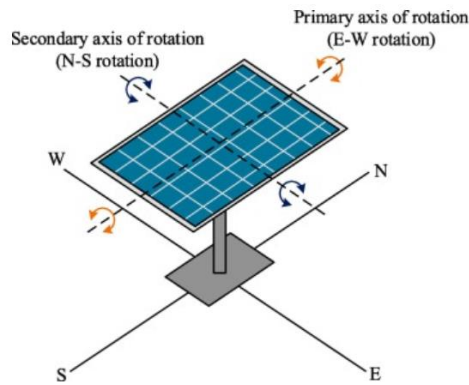


Figure 2.4: The Structure Design of a Dual-Axis Tracker (Jamroen et al., 2020)

2.3.1 Recent Development of Sun Tracking System

Since 20th century, different type of sun tracking system is developed to improve the performance of PV system. In traditional way, the tracking of the sun can be achieved by using photosensors. From the research paper by Abdallah & Nijmeh, (2003) the sun tracking system is achieved using the photosensors to obtain the sun position and send electrical signals to the PLC control unit. A set of mathematical calculations using azimuth and zenith angle of sun is done using a computer program. Besides, this design also considers the motor speed calculation by dividing the daylight hours into four identical time intervals as shown in Figure 2.5. Although this kind of mechanism is able to achieve solar tracking, however, the cost implementation is slightly higher as it involves lots of hardware such as PLC control unit and photosensors.

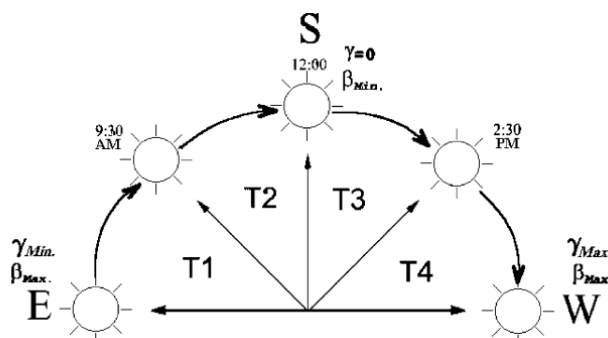


Figure 2.5: Four Identical Time Intervals During Daylight Hours (Abdallah & Nijmeh, 2003)

This theory is almost similar to Al-Mohamad, (2004) solar tracking system design where photo resistor is used to send command to control the PLC. The design includes two symmetric photo resistors installed on the same PV module with solid barrier in between. The increase and decrease of the solar-radiation intensity will produce different voltage drop, which directly connected to analogue inputs of PLC for control purpose. The design of the photosensor is shown in Figure 2.6. This design is quite similar with the previous one however the accuracy will be slightly lower as it does not have detail control on motors.

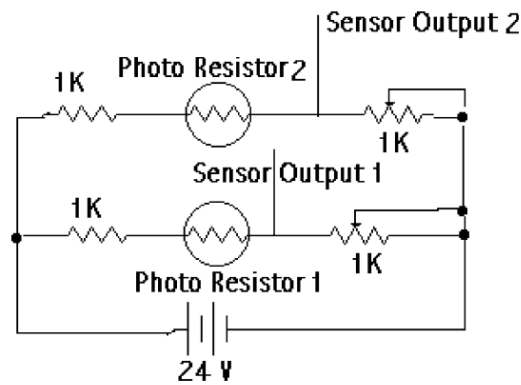


Figure 2.6: Circuit Design of Photosensors (Al-Mohammad, 2004)

In recent years, as the introduce of AI, there are several developments of sun tracking system which optimize the advantage of it. Such example is the research from EI Shenawy et al., (2012) where artificial neural network (ANN) is used to control the solar tracking system. By using ANN, a series parallel feed-forward neural network was designed to construct a two-axis solar tracker as shown in Figure 2.7. The training of the ANN involves 1054 epochs to learn the error between output from ANN and output from training pair. By using the AI algorithm with large datasets provided, this design could highly improve the performance of sun tracking system.

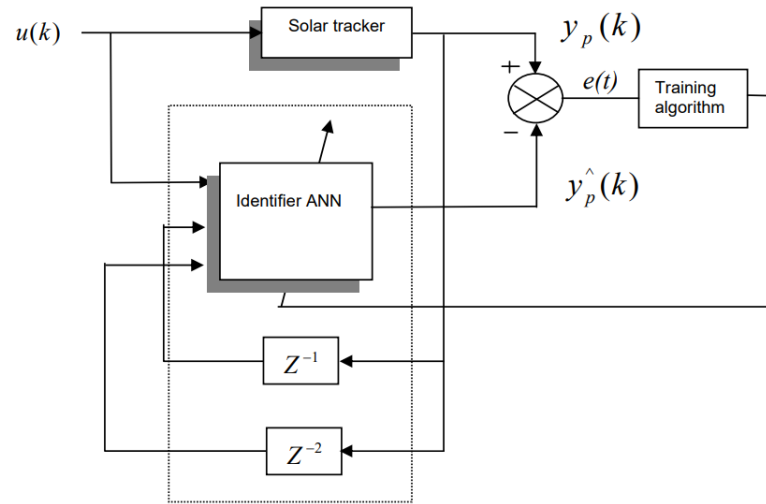


Figure 2.7: Neural Network Identification Scheme (EI Shenawy et al., 2012)

2.4 Cloud Prediction System

Cloud prediction system is a technology with advance algorithm integrated to predict the movement of the cloud. This technology is crucial in 21st century as it plays an important role in forecast to predict the weather. In the past, the observation of the cloud movement can only be done using satellite platforms. With the rapid growth of machine learning and deep learning, the cloud movement prediction had marked a significant turning point, where the cloud prediction from ground image had getting more accurate. The commonly used algorithm includes motion vector, and trajectory prediction system such as Kalman Filter.

2.4.1 Current Existing Cloud Prediction System

According to Hu et al., (2018) research paper, a cloud prediction system is planted in a PV system to solve the efficiency issue cause by cloud shading. The cloud prediction system is achieved by using motion vector to predict the moving trajectory. The motion vector achieved the assignment by calculating the position deviation, which also known as two-dimensional motion vector. By calculating the motion vector, the future position of the moving cloud can be obtained as illustrated in Figure 2.8.

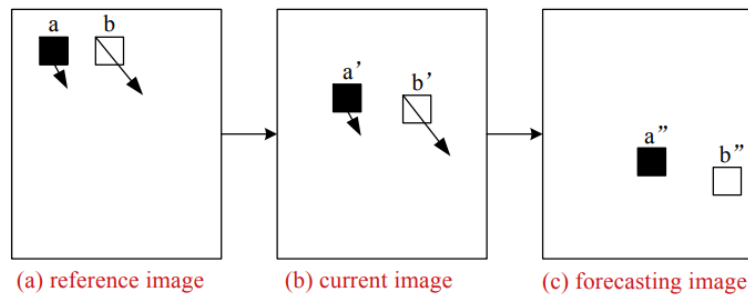


Figure 2.8: Illustration of Motion Vector Prediction (Hu et al., 2018)

From the research, the advantage of using motion vector includes high accuracy cloud trajectory prediction. This can be shown from the result obtained in Figure 2.9 where the time of the target clouds shades the sun is predicted accurately.

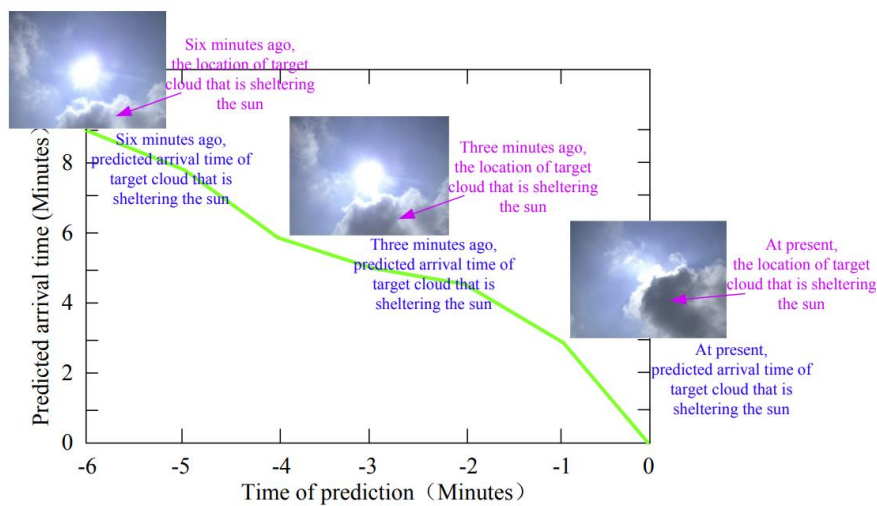


Figure 2.9: Prediction of Cloud Trajectory (Hu et al., 2018)

2.5 Summary

There have been several developments in sun and cloud tracking system, with different method and algorithm being used. The main purpose of these research is to improve the efficiency of PV system. Overall, AI has been widely used in solar power field in recent years to enhance the performance of PV system. The advancement of AI has provided the technological foundation to power the sun and cloud prediction system.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter outlines the details of work to achieve the objectives in chapter 1. Selection of components and software will be discussed in this chapter. Moreover, calculation and theory to achieve the cloud prediction and sun tracking will be included. While the connection of the hardware will be visualized in block diagram and the process flow of the system will be illustrated in flow chart.

3.2 AI Sun–Tracking with Sun and Cloud Position Prediction

In this project, to achieve the sun tracking and cloud prediction system, AI object detection model, YOLOv8 will be used to identify the cloud and sun directly stream from the webcam module. Calculation of sun position and cloud movement prediction will be done using Pythagoras and Trigonometry theorem. This algorithm will be integrated in an existing CPV model consist of non-imaging dish concentrator (NIDC) prototype on rooftop of Universiti Tunku Abdul Rahman (UTAR). This CPV contains 60 mirrors where each mirror has a dimension of 120mm × 120mm. Furthermore, the CPV is driven by a motor control unit which consist of two stepper motor and controlled using Arduino UNO.

3.2.1 Selection of Hardware and Software

For the hardware, Xiaomi Xiaovv webcam will be used to stream the sky image for sun and cloud detection. This webcam has resolution of 1024×768 with 200w pixels. It provides high quality undistorted image which benefits for object detection. However, this camera module does not have waterproof feature. Thus, a waterproof electrical junction box will be used to cover the module.

Besides, this webcam only provides view angle of 150° which is not wide enough to capture the whole sky image. As wide view of camera is important to prevent lost track of sun and cloud detection, a fisheye camera lens

are attached to the webcam to enhance the field of view. By the aids of fisheye lens, the field of view is upgraded to nearly 180° of vertical and 180° of horizontal view. The comparison of the field of view before and after fish-eye lens attached is shown in Figure 3.1. Furthermore, in order to protect the lens from exposing to bright sun, black film will be stick on the lens to ensure the long-term use of device. In summary, the hardware components used are listed in Table 3.1.

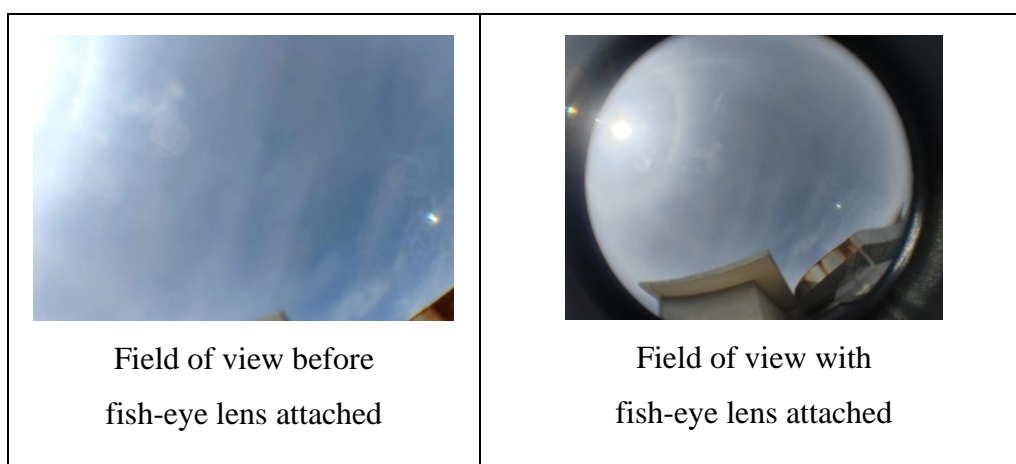



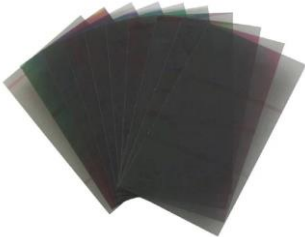


Figure 3.1: The Comparison of Field of View Before and After Fisheye Lens Attached.

Table 3.1: List of Hardware Components

Hardware Components	Statistics
 <p>Xiaomi Xiaovv Webcam</p>	<p>Resolution: 1024 × 768 Field of view: 150° Connection: USB2.0 cable</p>

 <p>Waterproof Electrical Junction Box</p>	<p>Size: 4cm × 6cm × 3cm Material: PVC Features: Waterproof enclosure</p>
 <p>Fish-Eye Lens</p>	<p>Field of view: 180° vertical and 180°horizontal</p>
 <p>Black Film</p>	<p>-</p>

For software part, YOLOv8 will be used as AI object detection model due to the higher accuracy in detection and fast processing speed. The high accuracy detection of the model had benefits in detection of clouds with various irregular shape. Besides, the fast-processing speed allow the algorithm to work in real-time application. The training process of the sun and cloud detection model will be carried out in Google Collab, which is a workplace with programming language of Python. Then, the trained model will be export and run in Python IDLE. Lastly, Arduino IDE will be involved in control system part with the assistant of Q-learning.

3.2.2 Design Architecture and Practical Process

To achieve the sun tracking and cloud prediction system, AI object detection model YOLOv8 will be trained using datasets of sky images taken from UTAR

rooftop. A camera with fisheye lens will be installed to stream the real time sky image.

Before starts operating, a calibration will be done to obtain the sun moving speed of the day. Calibration is important as the speed of sun will be various throughout the year. While the speed of sun throughout the day should be nearly constant as the rotation of earth is in constant speed (Sawal.I., n.d.). In such, the calibration on sun moving speed only needed once every day.

This system will starts operating after calibrations. AI model will detect sun and clouds, and the CPV will be turned towards the sun using Q-learning that planted in the control system. Once the sun is shaded where the AI model could not detect the sun, the centre coordination of the shading cloud will be collected and wait for 1 minute buffer. The purpose of this time buffer is to filter out small clouds that covered for a moment only, where no adjustment needed as it does not affect much in efficiency. For clouds shading 1 minute and above, the cloud centre point before and after 1 minute will be compared to obtain the displacement of cloud. By using this information, speed of cloud can be calculated. Besides, by comparing the coordinate of the two frames, the moving direction of the cloud in degree angle can be found using Trigonometry theory. Further calculations will be shown in the cloud prediction section.

After obtaining the predicted cloud movement and shading time, the predicted sun reappear position will be calculated using formula and the angle of adjustment will be sent to the control system of the CPV, to turn towards the predicted sun reappear position. The block diagram of the overall design will be shown in Figure 3.2. While the flow chart of the procedures will be shown in Figure 3.3.

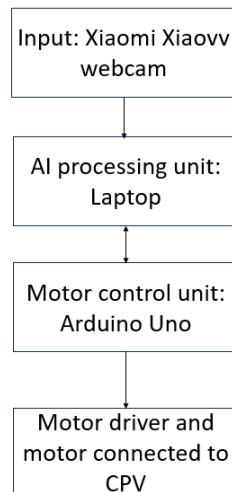


Figure 3.2: Block Diagram of the Design

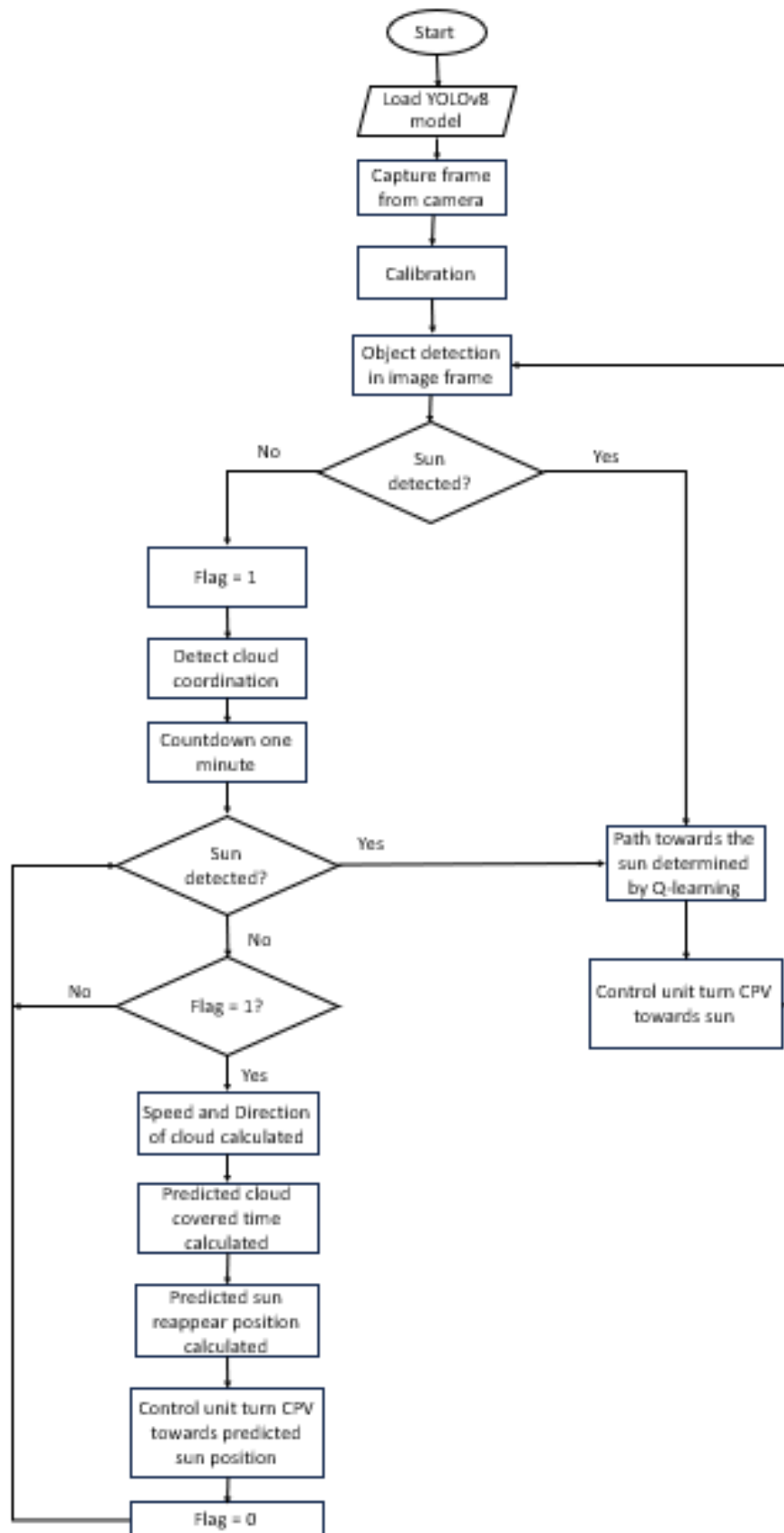


Figure 3.3: Flow Chart of the Design

3.2.3 Custom Dataset of Sun and Clouds

For training the sun and clouds AI detection model, sun and clouds image is collected on UTAR rooftop. The sky image will be collected during daytime from morning to evening involving different types of weather such as Altostratus cloud, Cirrostratus cloud and no cloud condition as shown in Figure 3.4. The image taken should be in 180 degrees as the lens is attached to the camera.

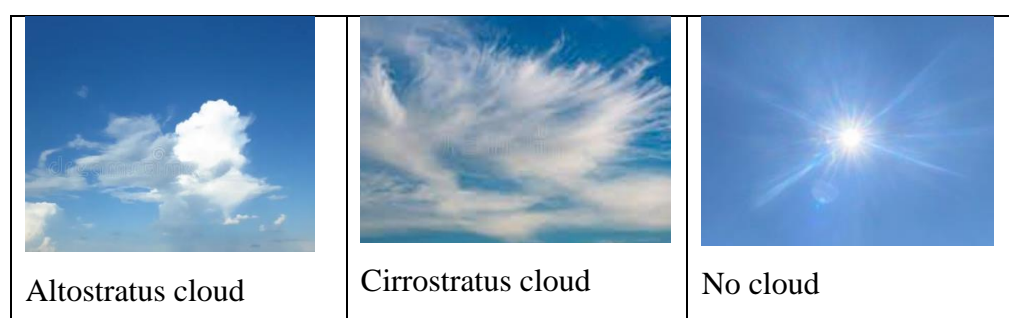


Figure 3.4: Classification of Weather (Dreamstime, n.d.)

Once the sky images are collected, the sets of images will be uploaded to Roboflow for preprocessing step. Roboflow is a platform which provides tools for managing and preprocessing dataset. It is often used for YOLO detection model training because it can export formats that compatible with YOLO. The labelling of the image is done in Roboflow to label each cloud and sun presence in the image. The object labelling is ensured to be accurate especially for irregular shaped clouds. This is because the quality of the datasets will directly affect the result accuracy. Lastly, the dataset is exported into YOLOv8 in Google Colab for model training.

3.2.4 Model Training for Object Detection

The training of AI object detection model YOLOv8, is done in Google Colab which is a workspace specially designed for AI training using GPU accelerator. The model is trained using the sun and cloud labelled dataset, which previously prepared in Roboflow. The datasets can be easily imported to the training through Roboflow link.

In the training of the model, transfer learning is applied where the YOLOv8 model is initially trained on a large dataset, such as general object

detection on a large dataset MS COCO. The purpose of transfer learning is to improve the learning effective of the model where it will learn new task based on the learning method done previously. This become more important when limited datasets are used for training. For the sun and cloud detection, 403 images with 100 epochs will be run to get a model with mean average precision (mAP50) of 0.7 and above for clouds and sun detection.

3.2.5 Q-learning Training

The Q-learning training is done in python IDLE. This Q-learning model is trained to find the shortest path from the ‘MOVINGPOINT’ towards the ‘CENTRETARGET’. The ‘MOVINGPOINT’ will represent the sun while the ‘CENTRETARGET’ will represent the centre of the camera. In this case, as the Q-learning is trained for 800×800-unit camera view, the ‘CENTRETARGET’ will be located at coordinate (400,400).

During the training process, the model used is not a pretrained model. In such, the moving point or also known as agent is initially moving in random direction as there are no Q-table for the reference. To ensure the moving point reach the target, 1,000,000 episode is done with maximum step of 1150 per episode. Besides, the epsilon is adjusted to 1 and epsilon decay set to 0.9998.

Furthermore, reward system and penalty system is programmed to limit the moving direction of the agent. This is to ensure the agent can find the shortest path towards the centre with 8 choices of moving direction provided in each step. The reward and penalty points are shown in Figure 3.5 while the algorithm of the reward and penalty system is shown in Figure 3.6.

```

HM_EPISODES = 1000000
maximum_step= 1150 #define maximum step per episode
MOVE_PENALTY = 30
MOVE_REWARD= 0.9 #Can be 0
MOVE_SLOPE_REWARD= 2
MOVE_SLOPE_PENALTY= 15
SAME_AXIS_PENALTY= 25
ON TARGET REWARD = 1000

```

Figure 3.5: Reward and Penalty Points for Q-learning.

```

for i in range(maximum_step):
    obs = moving_point - moving_target
    old_x = moving_point.x
    old_y = moving_point.y
    old_distance = np.sqrt(obs[0]**2+obs[1]**2)
    if np.random.random() > epsilon:
        # GET THE ACTION
        action = np.argmax(q_table[obs])
    else:
        action = np.random.randint(0, 8)

    moving_point.action(action)
    new_obs = (moving_point-moving_target)
    new_distance = np.sqrt(new_obs[0]**2+ new_obs[1]**2)
    If_distance_horizontal= np.sqrt((obs[0]-1)**2+ obs[1]**2)
    If_distance_vertical= np.sqrt(obs[0]**2+ (obs[1]-1)**2)

    if moving_point.x == moving_target.x and moving_point.y == moving_target.y:
        reward = ON_TARGET_REWARD
    elif new_distance < old_distance:
        reward= MOVE_REWARD
        if action == 4 or action == 5 or action == 6 or action == 7:
            if new_distance> If_distance_horizontal or new_distance>If_distance_vertical:
                reward= -MOVE_SLOPE_PENALTY
            elif old_x == moving_target.x or old_y == moving_target.y:
                reward = -SAME_AXIS_PENALTY

        else:
            reward= MOVE_SLOPE_REWARD

    else:
        reward = -MOVE_PENALTY

```

Figure 3.6: Reward and Penalty system for Q-learning

During the training, Q-value is constantly updated to the Q-table by using the formula (3.1). Lastly, the agent is expected to find the shortest path towards the target after 1,000,000 episodes of training.

$$\begin{aligned}
 new_q = (1 - LEARNING_{RATE}) * current_q + LEARNING_{rate} \\
 * (reward + DISCOUNT * max_future_q)
 \end{aligned} \tag{3.1}$$

3.2.6 Calibration on Sun Moving Speed and Direction

Calibration process can be done by pausing the CPV for 30 minutes. Then, the displacement of the sun, S_{sun} in 30 minutes will be collected, and the sun moving speed, V_s can be calculated by using the formula (3.2) and (3.3). While, the sun moving direction in angle, Θ_{sun} can be calculated using formula (3.4). Figure 3.7 had illustrated the calculations for sun speed calibration. Throughout the calibration process, the moving speed and moving direction of the sun will be obtained.

$$S_{sun} = \sqrt{(Y_{sun2} - Y_{sun1})^2 + (X_{sun2} - X_{sun1})^2} \text{ (unit)} \quad (3.2)$$

$$V_s = \frac{S_{sun}}{30} \text{ (unit/min)} \quad (3.3)$$

$$\theta_{sun} = \tan^{-1} \left(\frac{y_{sun2} - y_{sun1}}{x_{sun2} - x_{sun1}} \right) \text{ (degree)} \quad (3.4)$$

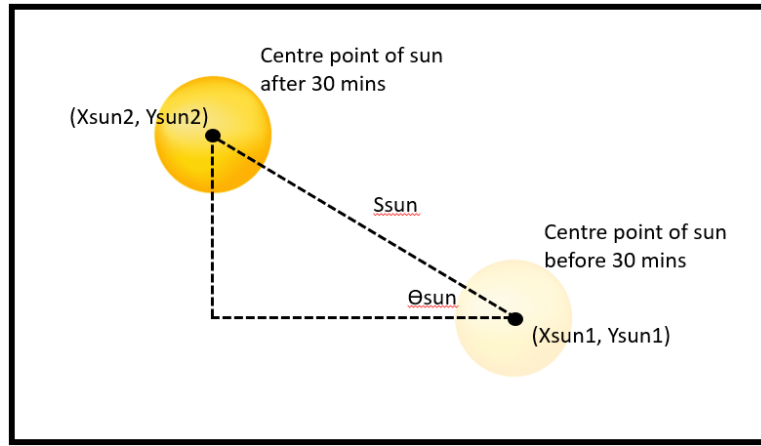


Figure 3.7: Illustration of the Calculation for Sun Speed Calibration.

3.2.7 Cloud Movement Predictions

To achieve the cloud movement prediction, a trained YOLOv8 model will be run in Python. Once the system starts operating, the sun and clouds will be detected, and control system will control CPV to follow the sun as shown in Figure 3.8. When one of the clouds had covered the sun where the AI model is unable to detect and track the sun position, Python will record the centre point (X_1, Y_1) of the shading clouds as shown in Figure 3.9. Then, the system will stop operating for 1 minute buffer, to filter off the small cloud covering situation. As small clouds did not cover for a long time, no adjustment of CPV is needed.

After 1 minutes, the new centre point of the same cloud (X_2, Y_2) will be recorded as shown in Figure 3.10. By using the information obtain, the cloud's moving speed (or velocity), V_c can be easily obtained using the formula (3.5) and (3.6). Besides, the cloud moving direction, Θ_{cloud} can also be obtained

using Trigonometry theory with the formula (3.7). The illustration of the calculation is shown in Figure 3.11.

$$\text{Displacement, } S_1 = \sqrt{(Y_2 - Y_1)^2 + (X_2 - X_1)^2} \text{ (unit)} \quad (3.5)$$

$$V_c = \frac{S_1}{1} \text{ (units/min)} \quad (3.6)$$

$$\theta_{cloud} = \tan^{-1} \left(\frac{y_2 - y_1}{x_2 - x_1} \right) \text{ (degree)} \quad (3.7)$$

Then, the point (X_3 , Y_3) which lie on the side of the box of cloud detection is found using the formula (3.8) and (3.9). The displacement, S_2 from the last appear sun centre point to (X_3 , Y_3) will be calculated using the Pythagoras theory with the formula (3.10) as illustrated in Figure 3.12.

$$X_3 = X_2 + \frac{\text{Cloud bounding box length, } X}{2} \text{ (unit)} \quad (3.8)$$

$$Y_3 = \left(\tan \theta \times \frac{X}{2} \right) + Y_2 \quad (3.9)$$

$$S_2 = \sqrt{(Y_3 - Y_{sun})^2 + (X_3 - X_{sun})^2} \quad (3.10)$$

Lastly, the predicted cloud shading time, t can be calculated by using the cloud moving speed, V_c and displacement, S_2 obtained as shown in formula (3.11).

$$t = \frac{S_2}{V_c} \quad (3.11)$$

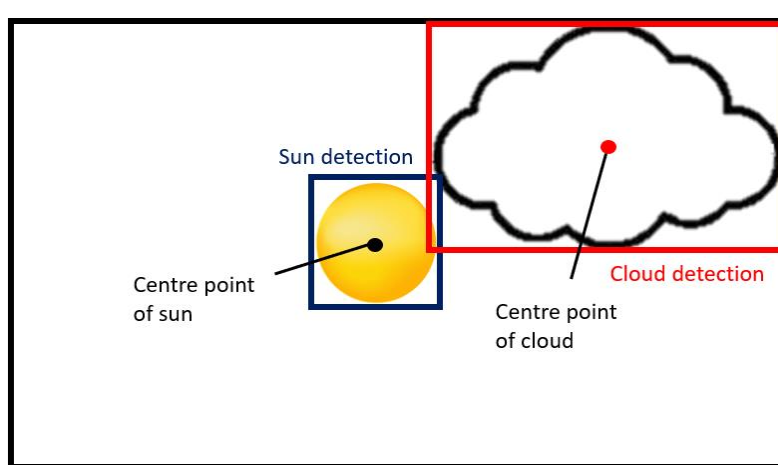


Figure 3.8: An Example Situation of a Moving Cloud Detection and Sun Tracking Stream from Camera

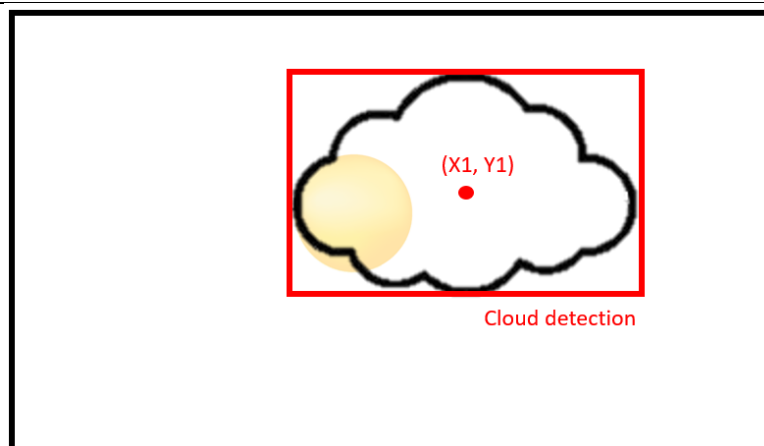


Figure 3.9: Situation when Cloud Covered the Sun

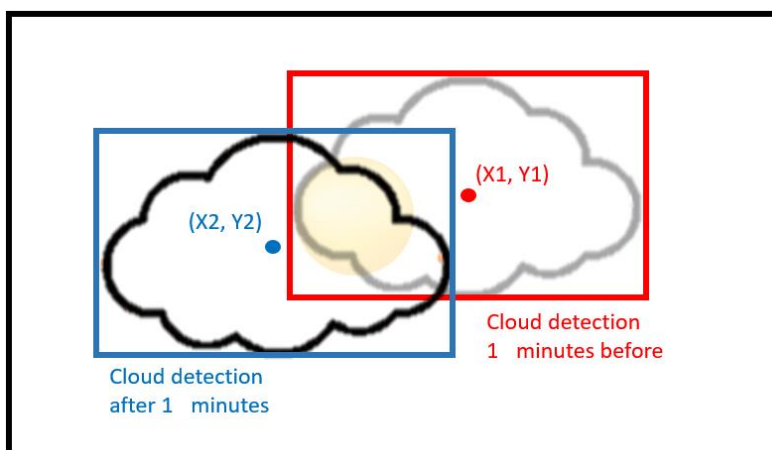


Figure 3.10: One Minute After Cloud Shading

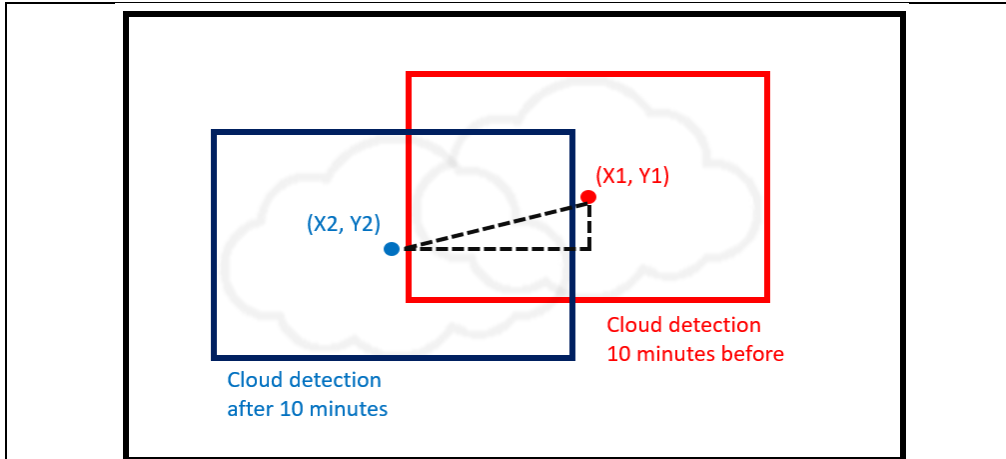


Figure 3.11: Calculation from the Two Frames of Same Cloud

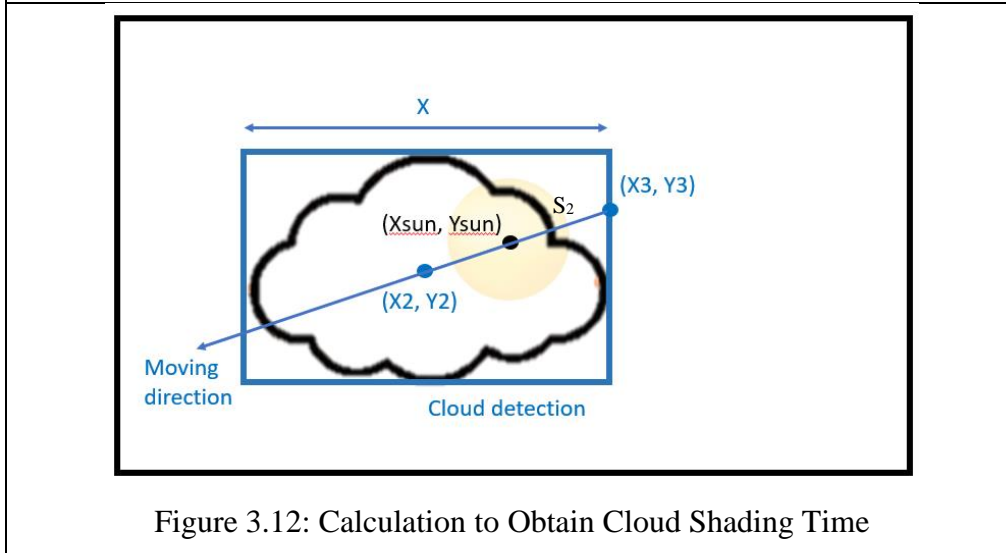


Figure 3.12: Calculation to Obtain Cloud Shading Time

3.2.8 Sun-Tracking and Position Predictions

To achieve sun tracking, a trained YOLOv8 sun detection model will be run in Python. As the camera streamed the sky image into Python, the sun will be detected by the AI algorithm and the centre point of the sun will be obtained. Then, Q-learning cooperate with control system will tilt the CPV in horizontal and vertical axis towards the position of sun.

When the sun is shaded by a cloud, the position of sun will be lost tracked and cloud prediction calculations will come in to obtain the predicted cloud shading time. By using the cloud shading time, t and speed of sun, V_s , the predicted displacement of sun, S_3 throughout the cloud shading can be calculated using formula (3.12).

$$S_3 = V_S \times t \text{ (unit)} \quad (3.12)$$

Lastly, the CPV will turn to the location of the predicted sun reappear by using the precalculated sun moving direction during calibration to find the x-axis displacement and y-axis displacement using the formula (3.13) and (3.14) as illustrated in Figure 3.13. When the sun reappears, the sun tracking system with the aids of Q-learning will take over the control to carry out fine tune and continues sun tracking.

$$X \text{ displacement} = \cos \theta_{sun} \times S_3 \quad (3.13)$$

$$Y \text{ displacement} = \sin \theta_{sun} \times S_3 \quad (3.14)$$

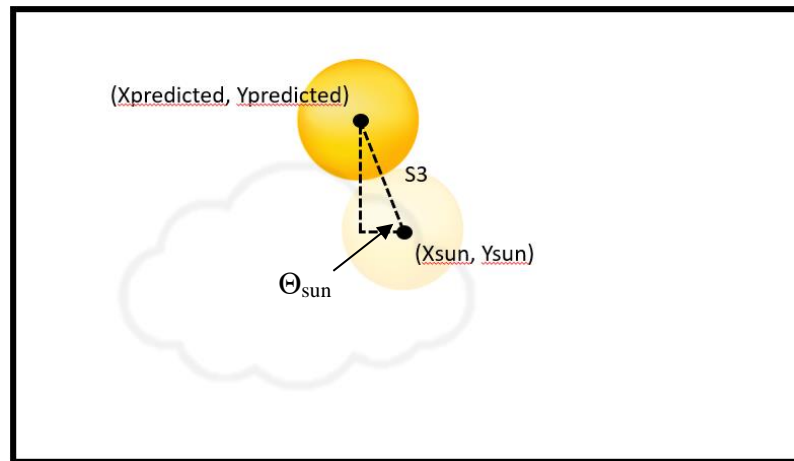


Figure 3.13: Calculation of Predicted Sun Reappear Position

3.3 Summary

In summary, this AI sun-tracking and cloud prediction system involved object detection model, YOLOv8 to detect the coordination of sun and clouds. Q-learning is integrated to command the movement of motor. When sun is lost tracked, prediction calculations will be performed in Python to predict the sun position and cloud movement. Pythagoras and Trigonometry theories will be involved in the calculations. Lastly, command to Arduino Uno to control the CPV towards targeted position.

CHAPTER 4

RESULT AND DISCUSSION

4.1 Introduction

In this chapter, the result of machine learning training including YOLOv8 detection model and Q-learning is discussed. Besides, the result on sun tracking and prediction algorithm with the assistance of AI will be present in graph and diagram. In addition, the performance of the CPV system will be tested during cloudy day and solar irradiance value will be analysed and discussed.

4.2 Result of Machine Learning Training

In this project, two machine learning model have been used to achieve the sun tracking system and prediction system. For object detection model, YOLOv8 have been trained to detect the sun and clouds presence in sky images. While Q-learning algorithm have been trained to find the shortest path from the agent 'MOVINGPOINT' towards the 'CENTRETARGET'. Both machine learning training result will be discussed.

4.2.1 YOLOv8 Training Result

YOLOv8 is a deep learning model which can be trained to make prediction by providing sufficient amount of data. In this training, Google Colab had been used to train the model with dataset of 403 sky images prepared in Roboflow. For the training process, 150 epochs have been done with the size of image resized to 800×800 pixels. The completion of the training is shown in Figure 4.1: Training of YOLOv8 Model on Google Colab with 150 Epochs.

```

Epoch   GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
148/150  13G      0.6475   0.5023   1.141     2          800: 100% 14/14 [00:12<00:00, 1.12it/s]
Class    Images  Instances  Box(P)    R          mAP50  mAP50-95): 0% 0/2 [00:00<?, ?it/s]/usr/lib
self.pid = os.fork()
Class    Images  Instances  Box(P)    R          mAP50  mAP50-95): 100% 2/2 [00:02<00:00, 1.10s/it]
all      60      186       0.827     0.818     0.832  0.548
clouds   60      170       0.695     0.635     0.669  0.408
sun      60      16        0.959     1         0.995  0.689
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with
self.pid = os.fork()

Epoch   GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
149/150  13G      0.5846   0.4445   1.062     2          800: 100% 14/14 [00:13<00:00, 1.07it/s]
Class    Images  Instances  Box(P)    R          mAP50  mAP50-95): 0% 0/2 [00:00<?, ?it/s]/usr/lib
self.pid = os.fork()
Class    Images  Instances  Box(P)    R          mAP50  mAP50-95): 100% 2/2 [00:02<00:00, 1.18s/it]
all      60      186       0.882     0.779     0.832  0.549
clouds   60      170       0.778     0.557     0.668  0.403
sun      60      16        0.987     1         0.995  0.696
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with
self.pid = os.fork()

Epoch   GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
150/150  13G      0.5859   0.4354   1.058     4          800: 100% 14/14 [00:12<00:00, 1.13it/s]
Class    Images  Instances  Box(P)    R          mAP50  mAP50-95): 0% 0/2 [00:00<?, ?it/s]/usr/lib
self.pid = os.fork()
Class    Images  Instances  Box(P)    R          mAP50  mAP50-95): 100% 2/2 [00:02<00:00, 1.49s/it]
all      60      186       0.836     0.824     0.837  0.569
clouds   60      170       0.714     0.647     0.68   0.414
sun      60      16        0.959     1         0.995  0.724

```

Figure 4.1: Training of YOLOv8 Model on Google Colab with 150 Epochs.

From the result of the training as shown in Figure 4.2, the 150 epochs is completed in 0.786 hours by using GPU Tesla T4 provided by Google Colab. In object detection model training, the accuracy and performance of the trained model is measure in mAP50, which stands for mean average precision at IoU threshold of 0.50. The mAP value can be range from 0.0 to 1.0 where the higher the mAP value, the better the performance of the detection model. As a well-known performance metric for machine learning model, it is frequently be referred when improving the model detection accuracy (Ahmed, N.N., n.d.).

From Figure 4.2, the result of sun detection training had shown that the performance of sun detection in mAP is 0.995, while for the result of cloud detection, the mAP is 0.685. The 0.995 mAP value in sun detection had explained that the accuracy of sun being detected is nearly 100%. This is because the framework of the YOLOv8 detection is based on the shape of the object. As the sun will only appear in round shape, the YOLOv8 could easily reach high mAP value by only required less amount of sun image dataset. However, the result of cloud detection shows a lower mAP value. This is due to the irregular shape of clouds appearance, where the model have more confusion in the prediction between clouds and sky background. Therefore, with the same number of datasets provided, the cloud detection has a significant lower accuracy than the sun detection. Besides, the mAP of each object detection can

also be visualized from the confusion matrix as shown in Figure 4.3. This confusion matrix had represented the true positive, false positive, true negative and false negative in each object detection. From the result obtain, 69% of clouds is predicted with true positive while 100% of sun is predicted with true positive.

```

150 epochs completed in 0.786 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 52.1MB
Optimizer stripped from runs/detect/train/weights/best.pt, 52.1MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.20 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 218 layers, 25840918 parameters, 0 gradients, 78.7 GFLOPs
Class      Images  Instances  Box(P)      R      mAP50  mAP50-95):  0% 0/2
self.pid = os.fork()
Class      Images  Instances  Box(P)      R      mAP50  mAP50-95): 100% 2/2
  all         60      186      0.88      0.803  0.84   0.607
  clouds      60      170      0.776     0.606  0.685  0.415
  sun         60      16       0.983     1      0.995  0.799
Speed: 0.3ms pre-process, 17.6ms inference, 0.0ms loss, 2.0ms post-process per image
Results saved to runs/detect/train

```

Figure 4.2: Result of Sun and Cloud YOLOv8 Detection.

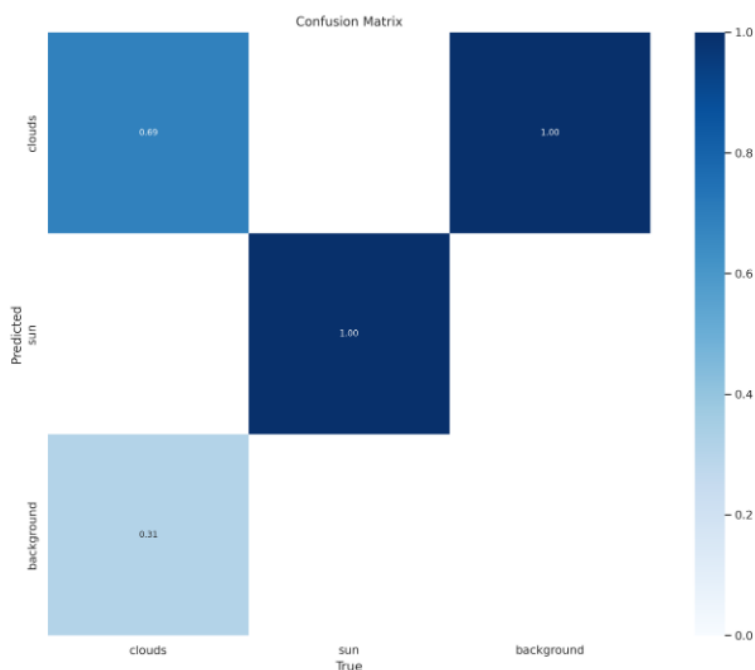


Figure 4.3: Confusion Matrix of Sun and Cloud YOLOv8 Detection.

In addition, Figure 4.4 had shown some of the detection on sky images using the trained model. From the result obtain, the sun detection has an obviously higher confident level above 0.8, which had proved the high accuracy performance of YOLOv8 on sun recognition. While the cloud detection has a slightly lower confident level due to the irregular shape of clouds presence.

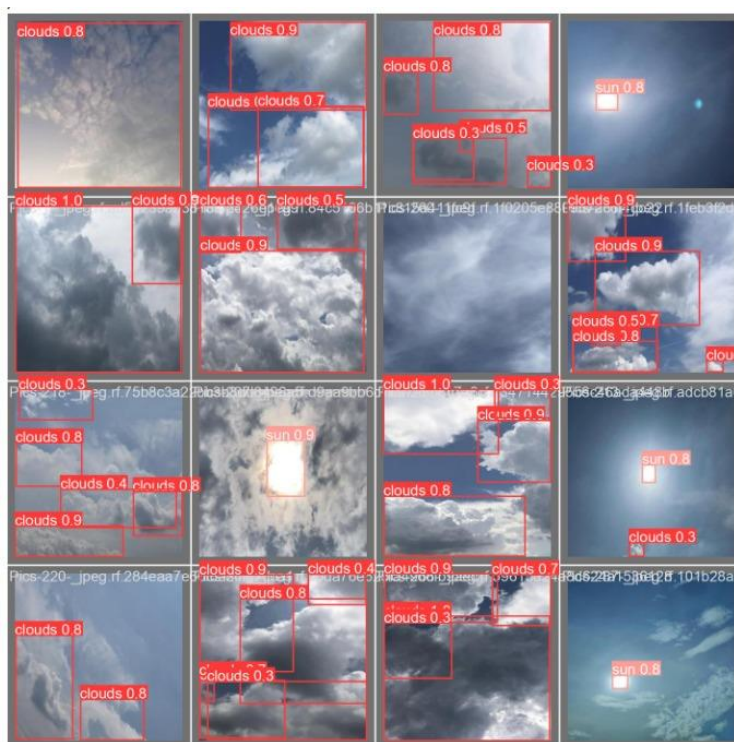


Figure 4.4: Sun and Cloud Detection on Sky Images.

In short, YOLOv8 is trained to detect sun and clouds present in the sky image. The sun detection has a higher detection accuracy with 0.996 mAP while the cloud detection has a lower mAP value which is 0.685. From the results obtained, the accuracy of both sun and cloud detection had meets the requirement for CPV sun tracking system.

4.2.2 Q-learning Training Result

For the Q-learning, training is done in python IDLE with 1,000,000 episodes. In this training, the result of the reward is recorded for every 50,000 episodes. The rewards are plotted into graph of reward over episodes, and the part from 0 to 150,000 episode is cropped out for discussion, as shown in Figure 4.5.

From the result, the reward is increasing from 0 episode to 50,000 episode. During this stage, the agent is still in the learning process and the moving direction is in random. When the agent reaches saturated in 100,000 episodes, the Q-learning model is able to reach the 'CENTRETARGET' by referring the Q-table. The reward in saturated phase is around 1400 as shown in

the graph. However, in order to make sure the agent be able to find the shortest path from any direction, the training is still continued until 1,000,000 episodes is complete.

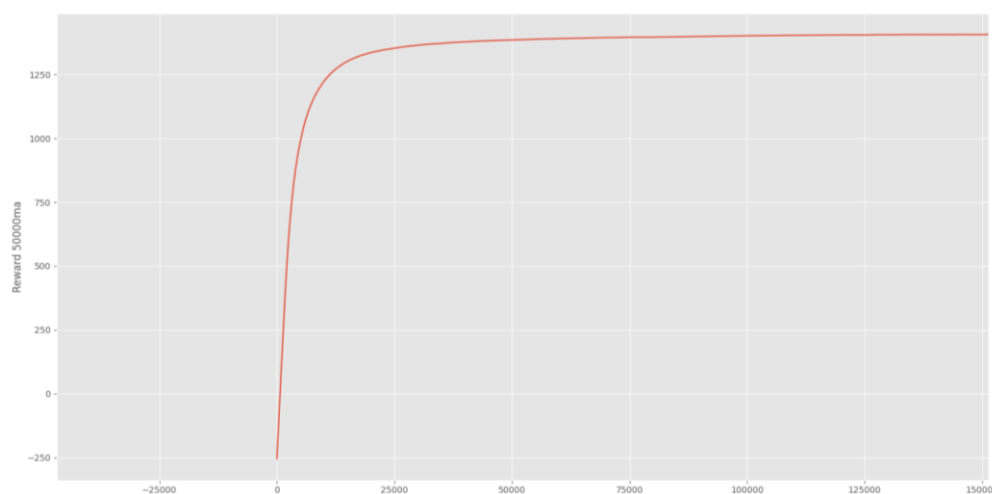
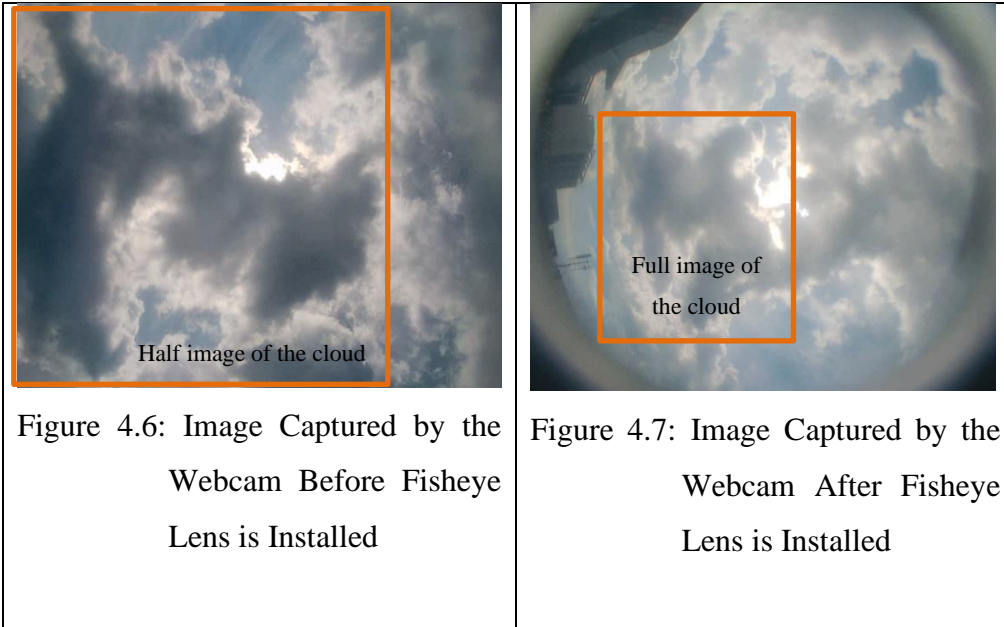


Figure 4.5: The Reward Obtained for Every 50,000 Episodes

In short, the result of Q-learning training had shown that the model is successfully trained to find the shortest path towards the target. The Q-table obtain can be directly used in the CPV system where the agent represents the sun and the 'CENTRETARGET' will be the centre point of camera view.

4.3 Fisheye Lens Assisted in Object Detection

In this project, fisheye lens had been attached to the camera module to provide a wider view for YOLOv8 detection. As the detection model is trained using complete full image of clouds, it is important to provide a wide vision that able to capture the whole clouds figure during the cloud detection. A comparison of sky image capture with and without assistant of fisheye lens is shown in Figure 4.6 and Figure 4.7. From the comparison, it is clearly seen that the sky image captured without fisheye lens have a narrow view and it is only able to obtain half image of the large cloud. However, when fisheye lens is attached, the view is obviously wider and the whole cloud image can be captured.



Furthermore, the comparison shown in Figure 4.8 and Figure 4.9 also proved that the installation of fisheye lens had improved the accuracy of cloud detection. From Figure 4.8, the cloud pointed by red arrow is initially not detected due to the incomplete cloud image provided to cloud detection model. However, when the fisheye lens is installed, the cloud has been detected as shown in Figure 4.9. The detection of cloud is important as it will affect the prediction algorithm when comes to cloud and sun position prediction. If clouds are not detected, the prediction algorithm may not run and cause the loss tracking of sun.

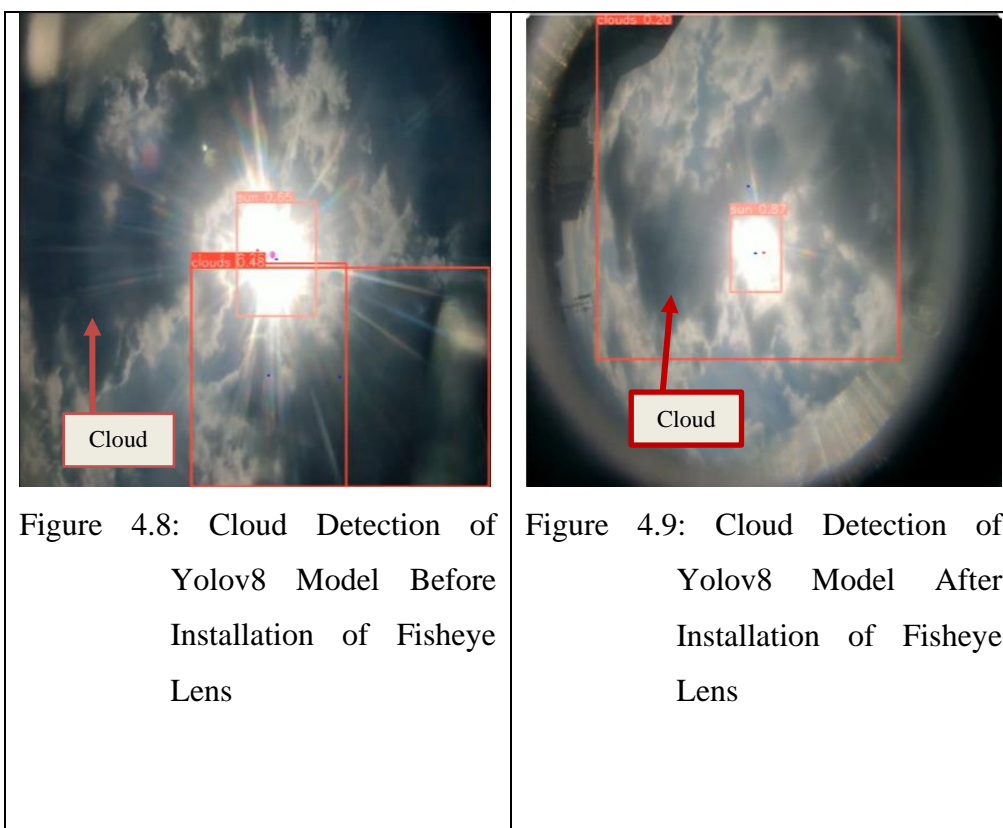


Figure 4.8: Cloud Detection of Yolov8 Model Before Installation of Fisheye Lens

Figure 4.9: Cloud Detection of Yolov8 Model After Installation of Fisheye Lens

Therefore, the installation of fisheye lens is necessary, and it has improved the detection model performance by providing a wide view of sky. The comparison had proved the importance of fisheye lens in assisting the detection model.

4.4 Solar Irradiance Analysis with Solar Tracking System

In order to measure the performance of CPV system with solar-tracking feature, solar irradiance value is recorded from 10:00am to 3:40pm on 27th April 2024. The weather is cloudy before 3:40pm and turns to rain after that. In this CPV performance test, a photometer is fixed parallelly to the camera, and it will be always 90 degrees facing the sun. This photometer will measure the analogue value of solar irradiance with the unit W/m^2 . While the YOLOv8 sun appearance detection is also collected by using Excel file. The sun appearance will be present in 0%, 50% and 100% sun appearance based on the area of sun detected. The 0% represent a totally loss of sun appearance; 50% represent a partially appearance of sun; and 100% will represent a full image of sun appearance. The

data collection of both solar irradiance and YOLOv8 sun detection is done with 10 minutes interval.

The Figure 4.10 had shown the solar irradiance and yolov8 sun detection vs time graph plotted from the data collected. In this graph, the solar irradiance had shown the instable value throughout the day. This is due to the shading of clouds during the cloudy day. When the sun is covered by clouds, the solar irradiance value will drop significantly and remains until the sun reappear. In the other hand, the graph had shown that the YOLOv8 sun appearance detection follows and exactly match to the fluctuation of solar irradiance graph. This had proved the accuracy of the sun detection model, where the solar irradiance value increase when sun is detected.

Moreover, according to ENERGY.GOV (n.d.), the solar irradiance will reach the peak value in the afternoon and drops in morning and late afternoon. This is due to the rays' long-distance travel through the atmosphere when the sun altitude is low during both periods. With the information provided, the trend of the solar irradiance increasing from 10:00am to the peak at 12:30pm and starts to drop after the peak had proved the accuracy of the solar tracking feature. The solar irradiance result had proved the dual-axis CPV system facing 90 degrees toward the sun throughout the day.

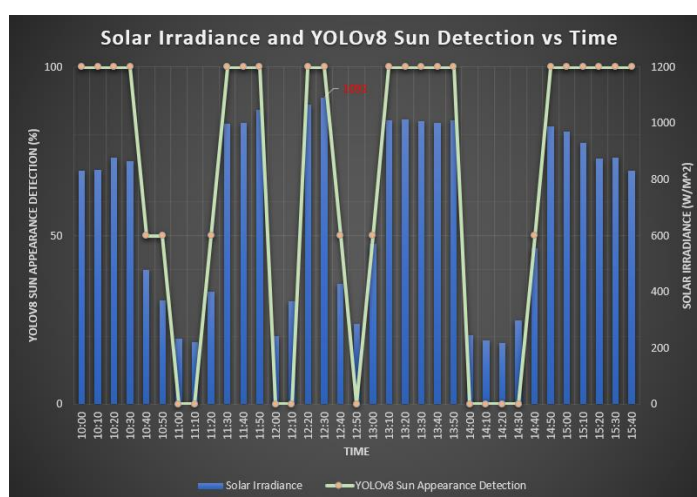


Figure 4.10: Graph of Solar Irradiance and YOLOv8 Sun Detection vs Time

In short, the matching of the detection with the fluctuation of solar irradiance graph had proved the accuracy and performance of YOLOv8 sun

detection model. While the trend of the solar irradiance throughout the day had verified the performance of CPV solar tracking system.

4.5 Sun and Cloud Position Prediction

To identify the accuracy of cloud and sun position prediction, the actual and predicted sun reappear coordinate is recorded during each shading of sun. This data is collected during a cloudy day on 27th April 2024. The percentage error will then be calculated to illustrate the prediction algorithm. To calculate the percentage error, formula (4.1) will be used, with predicted value represent the predicted displacement between sun last appear coordinate with sun reappear coordinate; while actual value represents the actual displacement between the sun last appear position with sun reappear position. In Figure 4.11, the diagram had illustrated the position of predicted and actual sun, with percentage error calculated from 11:00am to 11:20am. This diagram is drawn with 5 minutes interval to further analyse the percentage error and position of sun in each time stamp.

$$\text{Percentage Error} = \left| \frac{\text{Predicted} - \text{Actual}}{\text{Predicted}} \right| \times 100\% \quad (4.1)$$

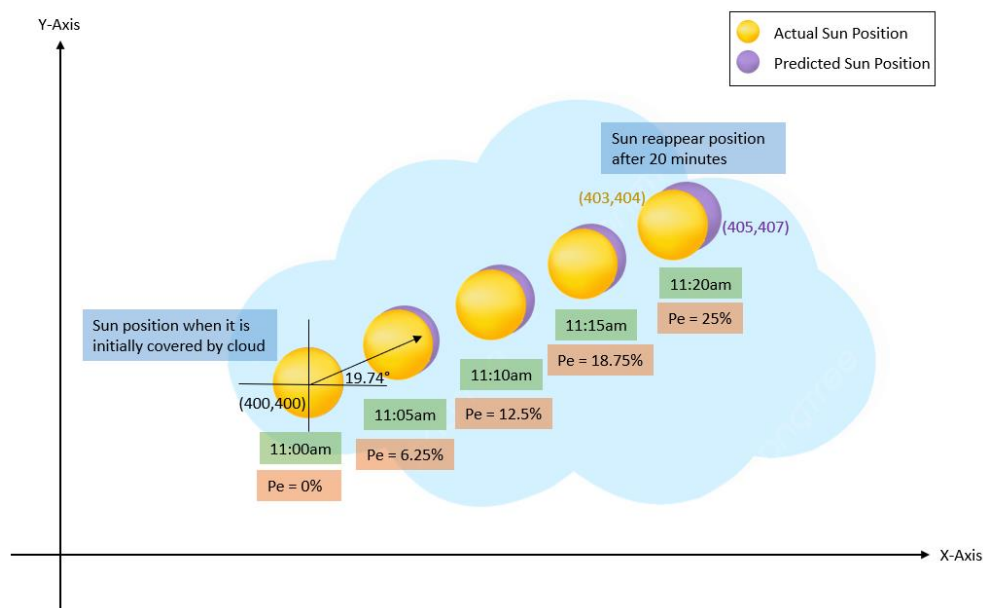


Figure 4.11: The Trajectory Motion Prediction of Sun After Covered by Cloud

From the prediction diagram in Figure 4.11, the sun moving angle is 19.74° , which is calculated during the calibration. The result shows that the predicted sun reappear position is slightly further than the actual sun reappear position. The prediction results an increment in percentage error of 6.25% for each 5 minutes. While the sun reappearance results a percentage error of 25%.

Apart from that, a percentage error vs time graph is plotted for each prediction throughout the day as shown in Figure 4.12. From the graph, the sun is covered by the clouds in four periods. In the first three periods, the percentage error is range from 11% to 25%. This percentage error is due to the change of wind speed which causing the inconstant speed of moving clouds. As the prediction system is assuming the sun and clouds have a constant speed, any changes in speed may affect the result. This can be clearly seen during 1:50pm to 2:50pm where the percentage error reached 43%. This is due to the sudden change of weather where it starts to rain after 3:40pm. The changes in weather and speed of wind causing the cloud moving in inconstant speed and leads to the prediction error. In addition, as the cloud will changes its shape over time, the irregular shape and difference in cloud coverage area will also affect the prediction accuracy.

Furthermore, the graph during the last period also shows the predicted sun shaded time is longer than the actual sun shaded time. Although there is prediction error, the sun-tracking of the CPV will not be affected much as long the predicted direction is accurate. This is because the CPV sun tracking system is programmed to keep detect the appearance of sun. As long as the sun does not appear outside the view of camera, the tracking system will recapture the sun once it reappears. When sun is detected, the Q-learning will command the CPV to turn towards it, even though the sun is reappeared earlier than predicted. This can be proved from solar irradiance graph in Figure 4.10, where the sun tracking system had recaptured the sunlight once the sun is reappeared again.

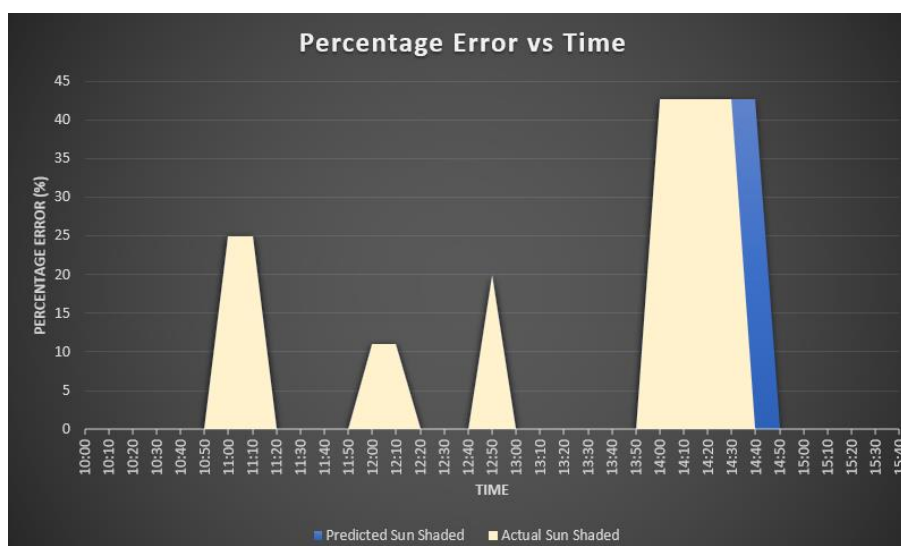


Figure 4.12: Percentage Error vs Time Graph During Cloudy Day (27 April 2024)

In summary, the results had shown that the prediction system able to predict the movement of sun when covered by clouds. However, there are percentage error where the predicted sun reappears position is further than the actual sun reappear position. The percentage error is getting higher when the wind speed changes rapidly during weather change. To solve the limitation, there are some future improvements can be done. The first solution is implementing a wind sensor into the prediction system. This could help to improve the accuracy of cloud movement prediction by monitoring the wind speed changes. Besides, real time satellite forecast system can also be integrated so that the changes in cloud's shape and area can be monitored.

4.6 Summary

In summary, YOLOv8 is trained to detect the sun and clouds from sky images. The accuracy of the YOLOv8 on sun detection is 0.996 mAP while cloud detection is 0.685 mAP. For Q-learning training, the reward is saturated at 1400 during 100,000 episodes of training. While from the solar irradiance recorded in cloudy day, the matching of sun detection with the fluctuation of YOLOv8 detection had proved the performance of the sun detection model. Besides, the trend of the solar irradiance also proved the accuracy of the CPV on sun tracking feature. Furthermore, percentage error of prediction system is calculated and

plotted in graph. From the graph, the predicted sun reappears position is slightly further from the actual sun position. However, the small difference in prediction error does not affect much in solar tracking as the CPV is programmed to recapture the sun as long as the sun is still in the view of webcam.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusions

This project had integrated two types of machine learning to automate the CPV sun tracking system. For sun and cloud detection model, YOLOv8 is chosen due to its high accuracy and less computational time. The advantage of YOLOv8 detection model allowed the detection to run in stream, avoiding the delay issue. For YOLOv8 model training, 403 sky images is used and the training is conduct with 100 epochs. The trained model results a sun detection accuracy of 0.996 mAP50 while the cloud detection results 0.685 mAP50.

For Q-learning training, 1,000,000 episodes is done with the reward obtain for every 50,000 episodes. When episodes of 100,000 is run, the reward obtained has starts to saturate at 1400. The result has showed that the Q-learning model are able to find the shortest path towards the target point.

Furthermore, the performance and accuracy of the CPV system sun-tracking feature is proven by the result of solar irradiance throughout a day. From the solar irradiance graph, the detection of sun in matching the fluctuation of the solar irradiance had proved the accuracy of the sun detection model.

Apart from that, the percentage error of prediction vs time graph had shown that the predicted sun reappear position is slightly further than the actual sun reappear position. The percentage error can be range from 11% for nearly constant cloud moving speed to 43% for large area clouds with unstable weather change. The error in the prediction is due to the changes of speed and inconstant shape of clouds over time. However, this limitation can be further improved by integrating wind speed sensor and satellite forecast system, so that the clouds changes can be monitored.

In addition, the installation of fisheye lens on the webcam had improved the vision of sky image streaming. When fisheye lens is attached, a wider view is provided to sun and cloud detection model. This had improved the accuracy of the detection as more information is input to the model.

In summary, the integration of YOLOv8 model with sun and cloud position prediction system had improved the overall efficiency of the CPV system by solving the sun lost tracking issue during cloudy day. By integrating the sun position prediction system, the CPV is able to turned and standby at the sun reappear position. This could capture the sun once it is reappeared and reduce the time-consuming problem during retracking of sun. Lastly, the AI-based CPV system had introduce a new trend of technology to solar industry and this could improve the efficiency of solar energy generation in the future.

5.2 Recommendation for Future Work

For future improvement, the cloud detection model can be retrained with more datasets. This is because the clouds are irregular shape, which it is a challenge to the YOLOv8 detection model. Thus, more sky images can be collected in the future and the epoch of the training can be increase based on the datasets available.

Besides, wind speed sensor can be integrated in the sun and cloud prediction algorithm. By applying wind speed sensor, the speed of the moving clouds can be monitored to improve the accuracy of prediction. Furthermore, a real time satellite forecast system can be integrated to monitor the changes in cloud coverage area. This could help the prediction system to predict the cloud shading time through the changes of cloud.

Lastly, trajectory machine learning can be implemented in the future to improve the prediction algorithm. This could achieve higher accuracy in prediction as machine learning model have more powerful calculation tools and it can be trained using past data.

REFERENCES

- Abdallah, S. and Nijmeh, S. (2004) “Two axes sun tracking system with PLC control,” *Energy Conversion and Management*, 45(11–12), pp. 1931–1939. Available at: <https://doi.org/10.1016/j.enconman.2003.10.007>.
- Admin, A. (2022) “Do solar panels work on cloudy days?,” *AESOLAR* [Preprint]. Available at: <https://ae-solar.com/do-solar-panels-work-on-cloudy-days/>.
- Al-Mohamad, A. (2004) “Efficiency improvements of photo-voltaic panels using a Sun-tracking system,” *Applied Energy*, 79(3), pp. 345–354. Available at: <https://doi.org/10.1016/j.apenergy.2003.12.004>.
- Anka, A. (2021) “YOLO v4: Optimal Speed & Accuracy for object detection,” *Medium*, 14 December. Available at: <https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>.
- “Artificial Intelligent Control of a Solar Tracking System” (2011) *Journal of Applied Sciences Research*, 8(8)((3971–3984)).
- Author.fullName (no date) *How fast does Earth spin?* Available at: <https://www.newscientist.com/question/fast-earth-spin/#:~:text=At%20the%20equator%2C%20its%20circumference,about%201670%20kilometres%20per%20hour>.
- Carranza-García, M. *et al.* (2020) “On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data,” *Remote Sensing*, 13(1), p. 89. Available at: <https://doi.org/10.3390/rs13010089>.

Chablani, M. (2023) “YOLO — You only look once, real time object detection explained,” *Medium*, 28 August. Available at: <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>.

Du, L., Zhang, R. and Wang, X. (2020) “Overview of two-stage object detection algorithms,” *Journal of Physics*, 1544(1), p. 012033. Available at: <https://doi.org/10.1088/1742-6596/1544/1/012033>.

Fig. 1. Solar power output for different weather conditions: a sunny... (no date). Available at: https://www.researchgate.net/figure/Solar-power-output-for-different-weather-conditions-a-sunny-day-20-April-2013-cloudy_fig1_301680317.

Figure 2. The model architecture of YOLO, where the backbone extracts... (no date). Available at: https://www.researchgate.net/figure/The-model-architecture-of-YOLO-where-the-backbone-extracts-features-from-an-image-the_fig1_351889219.

Gašparović, B. *et al.* (2023) “Evaluating YOLOV5, YOLOV6, YOLOV7, and YOLOV8 in Underwater Environment: Is There Real Improvement?,” *Is There Real Improvement?* [Preprint]. Available at: <https://doi.org/10.23919/splitech58164.2023.10193505>.

Hu, K. *et al.* (2018) “A new ultra-short-term photovoltaic power prediction model based on ground-based cloud images,” *Journal of Cleaner Production*, 200, pp. 731–745. Available at: <https://doi.org/10.1016/j.jclepro.2018.07.311>.

Jamroen, C. *et al.* (2020) “A low-cost dual-axis solar tracking system based on digital logic design: Design and implementation,” *Sustainable Energy*

- Technologies and Assessments*, 37, p. 100618. Available at: <https://doi.org/10.1016/j.seta.2019.100618>.
- Javaid, S. (2023) "5 AI Training Steps & Best Practices in 2023," *AIMultiple* [Preprint]. Available at: <https://research.aimultiple.com/ai-training/>.
- Jiang, P. *et al.* (2022) "A review of Yolo algorithm developments," *Procedia Computer Science*, 199, pp. 1066–1073. Available at: <https://doi.org/10.1016/j.procs.2022.01.135>.
- Laman web rasmi Jabatan Meteorologi Malaysia* (no date). Available at: <https://www.met.gov.my/en/pendidikan/iklim-malaysia/#Evaporation>.
- Mean Average Precision (MAP): A complete guide* (no date). <https://kili-technology.com/data-labeling/machine-learning/mean-average-precision-map-a-complete-guide>.
- Office, E.C. and D.L.T.U.S.P. and T. (2023) 'A brief history of solar panels,' *Smithsonian Magazine*, 15 November. <https://www.smithsonianmag.com/sponsored/brief-history-solar-panels-180972006/#:~:text=It%20all%20began%20with%20Edmond,to%20light%20or%20radiant%20energy>.
- Peng, S. *et al.* (2017) "Modulation classification using convolutional Neural Network based deep learning model," *IEEE* [Preprint]. Available at: <https://doi.org/10.1109/wocc.2017.7929000>.
- Ray, S. and Tripathi, A.K. (2016) "Design and development of Tilted Single Axis and Azimuth-Altitude Dual Axis Solar Tracking systems," *IEEE* [Preprint]. Available at: <https://doi.org/10.1109/icpeices.2016.7853190>.

Solar Radiation Basics (no date). <https://www.energy.gov/eere/solar/solar-radiation-basics#:~:text=In%20the%20early%20morning%20and,solar%20collector%20around%20solar%20noon.>

Unlock the Full Potential of Object Detection with YOLOv8 (no date). Available at: <https://www.augmentedstartups.com/blog/unlock-the-full-potential-of-object-detection-with-yolov8-faster-and-more-accurate-than-yolov7-2#:~:text=Faster%3A%20YOLOv8%20is%20faster%20than,YOLOv7%20in%20detecting%20small%20objects.>

Zaidi, S.S.A. *et al.* (2022) “A survey of modern deep learning based object detection models,” *Digital Signal Processing*, 126, p. 103514. Available at: <https://doi.org/10.1016/j.dsp.2022.103514>.

APPENDICES

Appendix A: Solar Irradiance and Prediction on 27 April 2024

Time	W/ m ²	Sun Appeara nce (%)	Predict d Sun Coordin ate	Actual Sun Coordin ate	Predict ed Sun Distan ce Move (unit)	Actual Sun Distan ce Move (unit)	Percenta ge Error (%)
10:0 0	831. 4	100	0	0	0	0	0
10:1 0	834. 3	100	0	0	0	0	0
10:2 0	879	100	0	0	0	0	0
10:3 0	866. 7	100	0	0	0	0	0
10:4 0	478. 9	50	0	0	0	0	0
10:5 0	371. 3	50	0	0	0	0	0
11:0 0	234. 5	0	405,407	403,404	8	6	25
11:1 0	222. 7	0	405,407	403,404	8	6	25
11:2 0	399. 8	50	0	0	0	0	0
11:3 0	997. 4	100	0	0	0	0	0
11:4 0	100 3	100	0	0	0	0	0
11:5 0	104 7	100	0	0	0	0	0
12:0 0	242. 4	0	405,408	403,407	9	8	11
12:1 0	367	0	405,408	403,407	9	8	11
12:2 0	106 7	100	0	0	0	0	0
12:3 0	109 2	100	0	0	0	0	0
12:4 0	429. 8	50	0	0	0	0	0
12:5 0	286. 4	0	404,406	403,407	5	4	20
13:0 0	571. 2	50	0	0	0	0	0

13:1 0	101 1	100	0	0	0	0	0
13:2 0	101 3	100	0	0	0	0	0
13:3 0	100 9	100	0	0	0	0	0
13:4 0	100 1	100	0	0	0	0	0
13:5 0	101 0	100	0	0	0	0	0
14:0 0	244. 7	0	434,460	422,434	68	39	43
14:1 0	227. 8	0	434,460	422,434	68	39	43
14:2 0	217. 5	0	434,460	422,434	68	39	43
14:3 0	299. 9	0	434,460	422,434	68	39	43
14:4 0	555. 6	50	0	0	0	0	0
14:5 0	988. 2	100	0	0	0	0	0
15:0 0	972. 1	100	0	0	0	0	0
15:1 0	929. 9	100	0	0	0	0	0
15:2 0	874. 5	100	0	0	0	0	0
15:3 0	877. 8	100	0	0	0	0	0
15:4 0	830. 3	100	0	0	0	0	0

Appendix B: Code of Q-learning Training

```

import numpy as np
from PIL import Image
import cv2
import matplotlib.pyplot as plt
import pickle
from matplotlib import style
import time

style.use("ggplot")

```



```

#functions required
def SaveQtable(start_q_table,q_table):
    if start_q_table != None:
        print('oktest')
        with open(start_q_table, "wb") as f:
            print('ok2')
            pickle.dump(q_table, f)
            print("q_table is updated.")
    else:
        with open(f"qtable-{{int(time.time())}}.pickle", "wb") as f:
            print("ok1")
            pickle.dump(q_table, f)
            print("New q_table is created.")

# classes required
# for environment and agent
class MOVINGPOINT:
    def __init__(self):
        self.x = np.random.randint(0, XSIZE)
        self.y = np.random.randint(0, YSIZE)
    def __str__(self):
        return f"{{self.x}}, {{self.y}}"

    def __sub__(self, other):
        return (self.x-other.x, self.y-other.y)

    def action(self, choice):
        """
        Gives us 4 total movement options. (0,1,2,3)
        """
        if choice == 0:
            #move right

```

```
self.move(x=1, y=0)
```

```
elif choice == 1:
```

```
    #move left
```

```
    self.move(x=-1, y=0)
```

```
elif choice == 2:
```

```
    #move up
```

```
    self.move(x=0, y=1)
```

```
elif choice == 3:
```

```
    #move down
```

```
    self.move(x=0, y=-1)
```

```
elif choice == 4:
```

```
    #move right up
```

```
    self.move(x=1, y=1)
```

```
elif choice == 5:
```

```
    #move left up
```

```
    self.move(x=-1, y=1)
```

```
elif choice == 6:
```

```
    #move right down
```

```
    self.move(x=1, y=-1)
```

```
elif choice == 7:
```

```
    #move left down
```

```
    self.move(x=-1, y=-1)
```

```
def move(self, x=False, y=False):
```

```
##### put signal code here for motor to move

# If no value for x, move randomly
if not x:
    self.x += np.random.randint(-1, 2)
else:
    self.x += x

# If no value for y, move randomly
if not y:
    self.y += np.random.randint(-1, 2)
else:
    self.y += y

# If we are out of bounds, fix!
if self.x < 0:
    self.x = 0
elif self.x > XSIZE-1:
    self.x = XSIZE-1
if self.y < 0:
    self.y = 0
elif self.y > YSIZE-1:
    self.y = YSIZE-1

class MOVINGTARGET():
    def __init__(self):
        self.x = np.random.randint(0, XSIZE)
        self.y = np.random.randint(0, YSIZE)
    def __str__(self):
        return f"{self.x}, {self.y}"

    def __sub__(self, other):
```

```
        return (self.x-other.x, self.y-other.y)

    def action(self):
        pass

class CENTERTARGET():
    def __init__(self):
        self.x = int (XSIZE/2)
        self.y = int (YSIZE/2)
    def __str__(self):
        return f"{self.x}, {self.y}"

    def __sub__(self, other):
        return (self.x-other.x, self.y-other.y)

    def action(self):
        pass

# define parameter
XSIZE = 800
YSIZE = 800

HM_EPISODES = 1000000
maximum_step= 1150 #define maximum step per episode
MOVE_PENALTY = 30
MOVE_REWARD= 0.9 #Can be 0
MOVE_SLOPE_REWARD= 2
MOVE_SLOPE_PENALTY= 15
SAME_AXIS_PENALTY= 25
ON_TARGET_REWARD = 1000
epsilon = 1 #0 for retraining the models
EPS_DECAY = 0.9998 # Every episode will be epsilon*EPS_DECAY
```

```
SHOW EVERY = 50000 # how often to play through env visually.
```

```
start_q_table = None # None or Filename
```

```
LEARNING_RATE = 0.1
```

```
DISCOUNT = 0.95
```

```
MOVING_POINT_N = 1 # moving point key in dict
```

```
MOVING_TARGET_N = 2 # target key in dict
```

```
CENTER_TARGET_N = 3 #center target key in dict
```

```
# the dict for colour
```

```
d = {1: (255, 175, 0),
```

```
      2: (0, 255, 0),
```

```
      3: (0, 0,255)}
```

```
if start_q_table is None:
```

```
    q_table = {}
```

```
    for i in range(-XSIZE+1, XSIZE):
```

```
        for ii in range(-YSIZE+1, YSIZE):
```

```
            q_table[(i, ii)] = [np.random.uniform(-5, 0) for i in range(8)]
```

```
else:
```

```
    with open(start_q_table, "rb") as f:
```

```
        print('ok9')
```

```
        q_table = pickle.load(f)
```

```
episode_rewards = []
```

```

for episode in range(HM_EPISODES):
    moving_target = CENTERTARGET()
    moving_point = MOVINGPOINT()
    if episode % SHOW_EVERY == 0:
        print(f"on #{episode}, epsilon is {epsilon}")
        print(f"{SHOW_EVERY} ep mean: {np.mean(episode_rewards[-
SHOW_EVERY:])}")
        show = True
    else:
        show = False

    episode_reward = 0
    for i in range(maximum_step):
        obs = moving_point - moving_target
        old_x = moving_point.x
        old_y = moving_point.y
        old_distance = np.sqrt(obs[0]**2+obs[1]**2)
        if np.random.random() > epsilon:
            # GET THE ACTION
            action = np.argmax(q_table[obs])
        else:
            action = np.random.randint(0, 8)

        moving_point.action(action)
        new_obs = (moving_point-moving_target)
        new_distance = np.sqrt(new_obs[0]**2+ new_obs[1]**2)
        If_distance_horizontal= np.sqrt((obs[0]-1)**2+ obs[1]**2)
        If_distance_vertical= np.sqrt(obs[0]**2+ (obs[1]-1)**2)

        if moving_point.x == moving_target.x and moving_point.y ==
moving_target.y:
            reward = ON_TARGET_REWARD
        elif new_distance < old_distance:

```

```

reward= MOVE_REWARD
if action == 4 or action == 5 or action == 6 or action == 7:
    if      new_distance>      If_distance_horizontal      or
new_distance>If_distance_vertical:
        reward= -MOVE_SLOPE_PENALTY
        elif old_x == moving_target.x or old_y == moving_target.y:
            reward = -SAME_AXIS_PENALTY

    else:
        reward= MOVE_SLOPE_REWARD

else:
    reward = -MOVE_PENALTY

max_future_q = np.max(q_table[new_obs])
current_q = q_table[obs][action]

if reward == ON_TARGET_REWARD:
    new_q = ON_TARGET_REWARD
else:
    new_q = (1 - LEARNING_RATE) * current_q + LEARNING_RATE *
(reward + DISCOUNT * max_future_q)
    q_table[obs][action] = new_q

if show:
    #to show visually
    env = np.zeros((XSIZE, YSIZE, 3), dtype=np.uint8) # starts an rgb of
our size
    env[moving_point.x][moving_point.y] = d[MOVING_POINT_N]
    env[moving_target.x][moving_target.y] = d[MOVING_TARGET_N]
    img = Image.fromarray(env, 'RGB')
    img = img.resize((600, 600)) # resizing
    cv2.imshow("image", np.array(img))

```

```

        if reward == ON_TARGET_REWARD: # crummy code to hang at the
end if we reach abrupt end for good reasons or not.
            if cv2.waitKey(500) & 0xFF == ord('q'): #hit q key it will break
                break
            else:
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
            episode_reward += reward
            if reward == ON_TARGET_REWARD:
                break

episode_rewards.append(episode_reward)
epsilon *= EPS_DECAY

moving_avg = np.convolve(episode_rewards,
np.ones((SHOW_EVERY,))/SHOW_EVERY, mode='valid')
plt.plot([i for i in range(len(moving_avg))], moving_avg)
plt.ylabel(f"Reward {SHOW_EVERY}ma")
plt.xlabel("episode #")
plt.show()
print("ok")
SaveQtable(start_q_table,q_table)

```

Appendix C: Code of CPV Control System

```

import time
from ultralytics import YOLO
import cv2
import math
import pickle
from datetime import datetime,date
from openpyxl import Workbook, load_workbook
import signal
import numpy as np

```



```

import matplotlib.pyplot as plt
import pathlib
import serial

model_path =
r"C:\Users\USER\Documents\FYP\Sun0995Cloud0724Detection.pt"

model = YOLO(model_path)

cap = cv2.VideoCapture(1)
fourcc = cv2.VideoWriter_fourcc(*'MP4V')
out = cv2.VideoWriter('output.mp4', fourcc, 20.0, (640,640))

cloud_midpoints = [] # Array to store cloud midpoints
twocloud_coord = [] #to save the coordinate of the two clouds before and after
60sec
cloud_corner1_frame = [] #to save cloud corner coordination for first block
cloud_corner2_frame = [] #to save cloud corner coordination for second block
loop = 0
Shad_time = 0

#define parameter for Q-learning
suntarget = []
XSIZE = 800
YSIZE = 800
sun_threshold_distance_range = (1,3) # define threshold distance in pixel for
the motor to run when sun is detected
threshold_distance = sun_threshold_distance_range [0]
MOVE_PENALTY = 30
MOVE_REWARD= 0.9 #Can be 0
MOVE_SLOPE_REWARD= 2
MOVE_SLOPE_PENALTY= 15
SAME_AXIS_PENALTY= 25

```

```
ON_TARGET_REWARD = 1000
epsilon = 0 #0 for retraining and run the model
EPS_DECAY = 0.9998 # Every episode will be epsilon*EPS_DECAY
LEARNING_RATE = 0.1
DISCOUNT = 0.95
start_q_table = "qtable-1710127871.pickle" # None or Filename

movex = 0
movey =0
predictedxmove = 0
predictedymove = 0
rollpermovex = 0.21380
tiltpermovex = 0.41895
starttime = float(0)
endtime = float(23)

#to connect arduino
arduino = serial.Serial(port='COM4', baudrate=9600, timeout=.1)#Create Serial
port object called arduinoSerialData
time.sleep(2)

def sendsignal(desiredmove):
    byte_command = bytes(desiredmove,'utf-8')
    arduino.write(byte_command)

def recorddata(array,number):
    return array.append(number)

def currenttimedecimal():
    currentDateAndTime = datetime.now()
    currenttime = currentDateAndTime.hour + currentDateAndTime.minute/60
+ currentDateAndTime.second/3600
    return (currenttime)
```

```

def SaveQtable(start_q_table,q_table):
    if start_q_table != None:
        with open(start_q_table, "wb") as f:
            pickle.dump(q_table, f)
            print("q_table is updated.")
    else:
        with open(f"qtable-{{int(time.time())}}.pickle", "wb") as f:
            pickle.dump(q_table, f)
            print("New q_table is created.")

```

```

class SUNMOVINGPOINT():
    def __init__(self,Xcoordinate,Ycoordinate):
        self.x = Xcoordinate
        self.y = Ycoordinate
    def __str__(self):
        return f"{{self.x}}, {{self.y}}"

    def __sub__(self, other):
        return (self.x-other.x, self.y-other.y)

    def action(self, choice):
        """
        Gives us 4 total movement options. (0,1,2,3)
        """
        if choice == 0:
            #move right
            self.move(x=1, y=0)

        elif choice == 1:
            #move left
            self.move(x=-1, y=0)

```

```
elif choice == 2:
    #move up
    self.move(x=0, y=1)

elif choice == 3:
    #move down
    self.move(x=0, y=-1)

elif choice == 4:
    #move right up
    self.move(x=1, y=1)

elif choice == 5:
    #move left up
    self.move(x=-1, y=1)

elif choice == 6:
    #move right down
    self.move(x=1, y=-1)

elif choice == 7:
    #move left down
    self.move(x=-1, y=-1)

def move(self, x=False, y=False):
    global status
    if not x:
        self.x += x
    elif x == 1:
        desiredmove = '4' #frame move left, target move right

        status = 'Left(1)'
        sendsignal(desiredmove)
```

```

        self.x += x
    elif x == -1:
        desiredmove = '6' #frame move right, target move left

        status = 'Right(1)'
        sendsignal(desiredmove)
        self.x += x

    if not y:
        self.y += y
    elif y == 1:
        desiredmove = '8' #frame move down, target move up (remember
        generalise coordinate system)

        status = 'Up(1)'
        sendsignal(desiredmove)
        self.y += y
    elif y == -1:
        desiredmove = '2' #frame move up, target move down (remember
        generalise coordinate system)

        status = 'Down(1)'
        sendsignal(desiredmove)
        self.y += y
class TARGET():
    def __init__(self,Xcoordinate,Ycoordinate):
        self.x = Xcoordinate
        self.y = Ycoordinate
    def __str__(self):
        return f"{self.x}, {self.y}"

    def __sub__(self, other):
        return (self.x-other.x, self.y-other.y)

```

```

def action(self):
    pass

if start_q_table is None:
    # initialize the q-table#
    q_table = {}
    for i in range(-XSIZE+1, XSIZE):
        for ii in range(-YSIZE+1, YSIZE):
            q_table[(i, ii)] = [np.random.uniform(-5, 0) for i in range(4)]

else:
    with open(start_q_table, "rb") as f:
        q_table = pickle.load(f)

def predictmotormove(predictedxmove,predictedymove):
    global status
    x=0
    y=0
    while predictedxmove != 0:
        if predictedxmove > 0:
            desiredmove = '6'
            status = 'Predict Right'
            x = x + 1
            predictedxmove = predictedxmove - 1
        elif predictedxmove < 0:
            desiredmove = '4'
            status = 'Predict Left'
            x = x - 1
            predictedxmove = predictedxmove + 1
        sendsignal(desiredmove)
        time.sleep(2)
    while predictedymove != 0:

```

```

if predictedymove >0:
    desiredmove = '2'
    status = 'Predict down'
    y = y +1
    predictedymove = predictedymove -1
elif predictedymove <0:
    desiredmove = '8'
    status = 'Predict Up'
    y = y - 1
    predictedymove = predictedymove + 1
sendsignal(desiredmove)
time.sleep(2)
return x,y

```

```
def
```

```
motormove(label,moving_point,moving_target,threshold_distance,sun_thresho
ld_distance_range):
```

```

    global status
    obs = moving_point - moving_target
    old_x = moving_point.x
    old_y = moving_point.y
    old_distance = np.sqrt(obs[0]**2+obs[1]**2)
    if old_distance > threshold_distance:
        threshold_distance= sun_threshold_distance_range[0]
        if np.random.random() > epsilon:
            # GET THE ACTION
            action = np.argmax(q_table[obs])
        else:
            action = np.random.randint(0, 8)

    if action == 0:
        if label == 'sun':
            x = 1

```

```
        y = 0
elif action == 1:
    if label == 'sun':
        x = -1
        y = 0
elif action == 2:
    if label == 'sun':
        x = 0
        y = 1
elif action == 3:
    if label == 'sun':
        x = 0
        y = -1
elif action == 4:
    if label == 'sun':
        x = 1
        y = 1
elif action == 5:
    if label == 'sun':
        x = -1
        y = 1
elif action == 6:
    if label == 'sun':
        x = 1
        y = -1
elif action == 7:
    if label == 'sun':
        x = -1
        y = -1

moving_point.action(action)
new_obs = (moving_point-moving_target)
new_distance = np.sqrt(new_obs[0]**2+ new_obs[1]**2)
If_distance_horizontal= np.sqrt((obs[0]-1)**2+ obs[1]**2)
```



```

If_distance_vertical= np.sqrt(obs[0]**2+ (obs[1]-1)**2)

if moving_point.x == moving_target.x and moving_point.y ==
moving_target.y:
    reward = ON_TARGET_REWARD
elif new_distance < old_distance:
    reward= MOVE_REWARD
    if action == 4 or action == 5 or action == 6 or action == 7:
        if new_distance> If_distance_horizontal or
new_distance>If_distance_vertical:
            reward= -MOVE_SLOPE_PENALTY
            elif old_x == moving_target.x or old_y == moving_target.y:
                reward = -SAME_AXIS_PENALTY

        else:
            reward= MOVE_SLOPE_REWARD

else:
    reward = -MOVE_PENALTY

max_future_q = np.max(q_table[new_obs])
current_q = q_table[obs][action]

if reward == ON_TARGET_REWARD:
    new_q = ON_TARGET_REWARD
else:
    new_q = (1 - LEARNING_RATE) * current_q + LEARNING_RATE *
(reward + DISCOUNT * max_future_q)
    q_table[obs][action] = new_q
elif old_distance <= threshold_distance:
    threshold_distance = sun_threshold_distance_range[1]
    print('reach target')
    status = 'Reach'

```

```

#Flag = 1
x = 0
y = 0
else:
    status = 'Stay'
    x = 0
    y = 0
return x,y,threshold_distance

def reversemotormove(xmove,ymove,motorresttime): #redo
    while xmove != 0:
        if xmove > 0:
            desiredmove = '4'
            xmove = xmove - 1
        elif xmove < 0:
            desiredmove = '6'
            xmove = xmove + 1
        sendsignal(desiredmove)
        time.sleep(motorresttime)
    while ymove != 0:
        if ymove > 0:
            desiredmove = '8'
            ymove = ymove - 1
        elif ymove <0:
            desiredmove = '2'
            ymove = ymove + 1
        sendsignal(desiredmove)
        time.sleep(motorresttime)
    print('Reverse movement done.')

#Define

def calculate_distance(x1, y1, x2, y2):

```

```
return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

def calc_Ssun(coord1, coord2):
    xsun1, ysun1 = coord1
    xsun2, ysun2 = coord2
    distance = math.sqrt((xsun2 - xsun1)**2 + (ysun2 - ysun1)**2)
    return distance

def calc_Anglesun(coord1, coord2):
    xsun1, ysun1 = coord1
    xsun2, ysun2 = coord2
    angle_rad = math.atan2(ysun2 - ysun1, xsun2 - xsun1)
    angle_deg = math.degrees(angle_rad)
    return angle_deg

def calc_S1(coord3, coord4):
    x1, y1 = coord3
    x2, y2 = coord4
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return distance

def calc_S2(coordsun, coordpre):
    x1, y1 = coordsun
    x2, y2 = coordpre
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    if distance < 0:
        distance = distance * (-1)
        return distance
    else:
        return distance

def calc_Anglecloud(coord3, coord4):
    x1, y1 = coord3
```

```
x2, y2 = coord4
angle_rad = math.atan2(y2 - y1, x2 - x1)
angle_deg = math.degrees(angle_rad)
return angle_deg

#Fixed variable
xcam = 800
ycam = 800
cammid = (xcam/2, ycam/2)
time_cali = 900
time_cloud= 60 #cloud prediction time or known as buffer

#Main

sun_detected = False # Flag to check if sun is detected

print("Tracking Started.")
today = date.today()
currentdate = datetime.now().strftime('%Y-%m-%d')
workbook = Workbook()
file_name = f"data_{currentdate}.xlsx"
workbook.save(file_name) #
wb = load_workbook(file_name)
ws = wb.active
ws2 = wb.create_sheet(title="Sheet 2")
ws3 = wb.create_sheet(title="Sheet 3")

def save_excel_and_exit(sig,frame):
    print('Ctrl + C detected! Saving Excel file...')
    wb.save(file_name)
    print('Excel file saved. Exiting gracefully.')
    cap.release()
```

```

out.release()
cv2.destroyAllWindows()
exit(0)

signal.signal(signal.SIGINT, save_excel_and_exit)

suntarget = TARGET(xcam/2, ycam/2)
ws.append(['Time','MoveX','MoveY','ObjectX','ObjectY','Label'])
ws2.append(['Time','Predicted      X      displacement','Predicted      Y
displacement','PredictedMoveX','PredictedMoveY'])
calibration = 0 #calibration needed if coord1 and 2 empty.
coord1=[379,398]
coord2=[340,384]
unitpermovex = 0
unitpermovey = 0
sun_xmid = 0
sun_ymid = 0
global status
status = 'Stay'
calibrationonce = True
nextstoptime = None
n = True

while cap.isOpened ():
    # Read a frame from the video
    success, frame = cap.read()
    print(status)

    if success:
        # Run YOLOv8 inference on the frame
        frame = cv2.resize(frame,(800,800))
        results = model.predict(frame,imgsz=(800,800), conf=0.1)
        if calibration == 1:

```

```

#print("0")
for result in results: # iterate results
    boxes = result.boxes.cpu().numpy() # get boxes on CPU in numpy
    cloud_midpoints_frame = [] # Temporary array for cloud midpoints
in the current frame
    for box in boxes: # iterate boxes
        x1box, y1box, x2box, y2box = box.xyxy[0].astype(int)
        x_mid = (x1box + x2box) // 2 # calculate midpoint x-coordinate
        y_mid = (y1box + y2box) // 2 # calculate midpoint y-coordinate

        name = box.cls[0]

        if name == 1: # Sun is detected
            sun_xmid = x_mid
            sun_ymid = y_mid
            if n == True:
                coord1 = (sun_xmid, sun_ymid)
                print(f"Midpoint of sun: ({sun_xmid}, {sun_ymid})")
                nextstoptime = currenttimedecimal()+60/3600
                t1 = currenttimedecimal()
                #print("1")
                n = False
            elif n ==False and currenttimedecimal()>=nextstoptime:
                print('1')
                coord2 = (sun_xmid, sun_ymid)
                print(f"Midpoint of sun: ({sun_xmid}, {sun_ymid})")
                calibration = 0
                t2=currenttimedecimal()
                time_cali = (t2-t1)*3600#sun calibration time 10 mins
                print(f"calibration duration = {time_cali}")
                #print("2")
                break

```

```

elif calibration == 0:
    if calibrationonce == True:

        print(f"Coordinate of sun before 10 mins: {coord1}")
        print(f"Coordinate of sun after 10 mins: {coord2}")
        calcunitpermove = True
        firstsun = True
        ws3.append(['coordinate of sun before 10 mins',coord1[0],coord1[1]])
        ws3.append(['coordinate of sun after 10 mins',coord2[0],coord2[1]])
        ws3.append(['calibration time',time_cali])
        calibrationonce = False

for result in results: # iterate results
    buffer_time = 0
    x = 0
    y = 0
    xdisplacement = 0 #unit
    ydisplacement = 0
    predictedxmove = 0
    predictedymove = 0
    label = None
    #global status
    #status = 'Stay'
    Flag = 0

    if calcunitpermove == True and firstsun == True:
        sunx1,suny1 = sun_xmid, sun_ymid
        firstsun=False
        print('ok')
    elif status == 'Reach'and calcunitpermove == True:
    #elif Flag == 1 and calcunitpermove == True:
        sunx2, suny2 = sun_xmid, sun_ymid

```

```

#sunx2,suny2 = coord2[0], coord2[1]
unitpermovex = (sunx2-sunx1)/movex
unitpermovey = (suny2-suny1)/movey
print(unitpermovex)
print(unitpermovey)
calcunitpermovex = False

```

```

if(currenttimedecimal())>starttime and
currenttimedecimal()<endtime):

```

```

boxes = result.boxes.cpu().numpy() # get boxes on CPU in numpy

```

```

sun_detected = False # Reset the flag for each frame

```

```

cloud_midpoints_frame = [] # Temporary array for cloud
midpoints in the current frame

```

```

for box in boxes: # iterate boxes

```

```

    x1box, y1box, x2box, y2box = box.xyxy[0].astype(int)

```

```

    x_mid = (x1box + x2box) // 2 # calculate midpoint x-coordinate

```

```

    y_mid = (y1box + y2box) // 2 # calculate midpoint y-coordinate

```

```

    name = box.cls[0]

```

```

    if name == 1: # Sun is detected

```

```

        label = 'Sun'

```

```

        sun_detected = True

```

```

        print(f"Midpoint of sun: ({x_mid}, {y_mid})")

```

```

        sun_area = (x2box - x1box)*(y2box - y1box)

```

```

        sun_area_threshold = 400

```

```

        if sun_area < sun_area_threshold:

```

```

            print('half sun detected')

```

```

        sun_xmid = x_mid #for q learning

```



```

sun_yamid = y_mid #for q learning
twocloud_coord = [] #to save the coordinate of the two clouds
before and after 60sec
loop = 0
sunposition = SUNMOVINGPOINT(sun_xmid,sun_yamid)
x,y,threshold_distance=
motormove('sun',sunposition,suntarget,threshold_distance,sun_threshold_dista
nce_range)

xdisplacement = 0 #unit
ydisplacement = 0
predictedxmove = 0
predictedymove = 0

elif name == 0 :      #sun not detected
    label = 'Cloud'
    print(f"Midpoint of cloud: ({x_mid}, {y_mid})")
    cloud_midpoints_frame.append((x_mid, y_mid)) # Store
cloud midpoint in the current frame
    cloud_corner1_frame.extend([x1box, y1box])
    cloud_corner2_frame.extend([x2box, y2box])
    buffer_time = 60

movex = movex + x
movey = movey + y

ws.append([currenttimedecimal(),movex,movey,x_mid,y_mid,label])

# If the sun is not detected, save all cloud midpoints from the last
frame
if not sun_detected:

```

```

        cloud_midpoints.extend(cloud_midpoints_frame) # Append
cloud midpoints from the current frame
        print("Sun not detected. Saving cloud midpoints...")
        print("Cloud Midpoints when sun dissappear:", cloud_midpoints)

        loop += 1

        min_distance = float('inf')
        blockcloud = None

        for coord in cloud_midpoints:
            distance = calculate_distance(coord[0], coord[1], cammid[0],
cammid[1])
            if distance < min_distance:
                min_distance = distance
                blockcloud = coord
            if blockcloud:
                print(f"The cloud blocking is: {blockcloud}")
                twocloud_coord.extend(coord)
                cloud_midpoints = []
            if loop > 1:
                loop = 0
                #Shad_time = True
                print(f"The coordinate of blocking cloud b4 and after is:
{twocloud_coord}")
                coord3 = (twocloud_coord[0],twocloud_coord[1])
                coord4 = (twocloud_coord[-2],twocloud_coord[-1])

                #the two corner coordinate of the shading cloud
                x1cloud = cloud_corner1_frame[2]
                x2cloud = cloud_corner2_frame[2]
                y1cloud = cloud_corner1_frame[3]

```

```

y2cloud = cloud_corner2_frame[3]

#xlength and ylength of the shading cloud
xlength = x2cloud - x1cloud
ylength = y2cloud - y1cloud

#vector
Sx = (coord2[0] - coord1[0])/time_cali
Sy = (coord2[1] - coord1[1])/time_cali
Cx = (coord4[0] - coord3[0])/time_cloud
Cy = (coord4[1] - coord3[1])/time_cloud

#Cloud movement
Anglecloud = calc_Anglecloud(coord3, coord4)

#Cloud prediction
Check =
(math.tan(math.radians(Anglecloud)))*(xlength/2)
Treshold = ylength/2

if Check >= Treshold:
    Ycloud = coord4[1]
    Shad_time = abs(((ycam/2)- Ycloud +
(ylength/2))/(Cy - Sy))

else:
    Xcloud = coord4[0]
    Shad_time = abs(((xcam/2)- Xcloud +
(xlength/2))/(Cx - Sx))

x3 = (xcam/2)+ (Shad_time*Sx)
y3 = (ycam/2)+ (Shad_time*Sy)

```

```

xdisplacement = x3 - (xcam/2)
ydisplacement = y3 - (ycam/2)
print(f"x displacement: {xdisplacement}unit")
print(f"y displacement: {ydisplacement}unit")
predictedxmove = round(xdisplacement / unitpermovex)
predictedymove = round(ydisplacement / unitpermovey)

x,y = predictmotormove(predictedxmove,predictedymove)
buffer_time = 0

ws2.append([currenttimedecimal(),xdisplacement,ydisplacement,predictedxmove,
predictedymove])

else:
    print("No cloud midpoints to calculate the nearest.")

    print(f"Pls wait {buffer_time+ Shad_time}seconds")

    time.sleep(buffer_time+ Shad_time)

else:
    break

# Visualize the results on the frame
annotated_frame = results[0].plot()
for box in results[0].boxes.xywh:
    x, y, w, h = box

```

```
annotated_frame = cv2.circle(annotated_frame, (int(x), int(y)), radius=0,  
color=(255, 0, 0), thickness=4) #BGR
```

```
annotated_frame = cv2.circle(annotated_frame, (xcam//2, ycam//2),  
radius=0, color=(0, 0, 255), thickness=4) #BGR
```

```
annotated_frame = cv2.resize(annotated_frame, (640,640))
```

```
# Display the annotated frame
```

```
cv2.imshow("YOLOv8 Inference", annotated_frame)
```

```
out.write(annotated_frame)
```

```
# Break the loop if 'q' is pressed
```

```
if cv2.waitKey(1) & 0xFF == ord("q"):
```

```
    break
```

```
else:
```

```
    # Break the loop if the end of the video is reached
```

```
    break
```