ARTIFICIAL INTELLIGENT INTEGRATED INTO SUN-TRACKING SYSTEM TO ENHANCE THE ACCURACY, RELIABILITY AND LONG-TERM PERFORMANCE IN SOLAR ENERGY HARNESSING

TAN JUN YOU

UNIVERSITI TUNKU ABDUL RAHMAN

ARTIFICIAL INTELLIGENT INTEGRATED INTO SUN-TRACKING SYSTEM TO ENHANCE THE ACCURACY, RELIABILITY AND LONG-TERM PERFORMANCE IN SOLAR ENERGY HARNESSING

TAN JUN YOU

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Physics

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

September 2022

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	:	[illegible]
Name	:	TAN JUN YOU
ID No.	:	19UEB06464
Date	:	11/09/2022

APPROVAL FOR SUBMISSION

I certify that this project report entitled **“ARTIFICIAL INTELLIGENT INTEGRATED INTO SUN-TRACKING SYSTEM TO ENHANCE THE ACCURACY, RELIABILITY AND LONG-TERM PERFORMANCE IN SOLAR ENERGY HARNESSING”** was prepared by **TAN JUN YOU** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Physics at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	K.K.Chong
Supervisor	:	Prof. Dr. Chong Kok Keong
Date	:	11/09/2022

ACKNOWLEDGEMENTS

I would like to express my greatest gratitude and appreciation to my supervisor, Prof. Dr. Chong Kok Keong for providing me the opportunity to take part in this research on the title “Artificial Intelligent Integrated into Sun-Tracking System to Enhance the Accuracy, Reliability and Long-Term Performance in Solar Energy Harnessing”. I have gained much experience and knowledge throughout the research and I’m grateful to all the guidance and advice provided throughout this research development.

I would also like to extend my gratitude to my project partner, Tiow Yik Hong and Alvin Koo Xian Ming of Universiti Tunku Abdul Rahman for the discussion and support throughout this project. In addition, I would want to express my gratitude to my parents and friends for their support in helping me getting through this dissertation.

ABSTRACT

The solar energy collected by the sun-tracking system depends on the accuracy of the sun-tracking algorithm, which is different for each type of sun-tracking system. Furthermore, the solar image formed on the on-axis target can also be easily affected by gear backlash and wind load, and an absolute encoder is used to reduce the errors produced by the sun-tracking system. Therefore, a fully artificial intelligent (AI)-integrated sun-tracking algorithm is proposed and can be used in any type of sun-tracking systems such as concentrated photovoltaic (CPV), flat photovoltaic (PV) or heliostat systems. The proposed AI algorithm integrates two deep learning models which are object detection algorithm and reinforcement learning. YOLOv7 is chosen as the object detection algorithm to detect sun, while Q-learning is chosen as the algorithm for reinforcement learning to control the motors. A custom dataset of 300 images of sun and clouds is prepared for the training process of YOLOv7. The YOLOv7 is trained for 100 epochs at different batch sizes, where the batch size of 4 shows the highest mean average precision (mAP) of 0.768 and the lowest loss. A python script is created to integrate both YOLOv7 and Q-learning to execute the tasks simultaneously to adjust the position of the sun tracker based on the detected sun. Once the sun coordinate is obtained, Q-learning will determine the minimum steps taken for the centre point of camera to reach the midpoint of the Sun. The algorithm is also tested under different conditions such as during sunny day, sunset and also when the sun is partially blocked. It shows that the sun is detected under each condition and has a confidence level of over 90%. The Q-learning provides the minimum steps and the movement options of the agent to move from the centre point to the sun coordinate. It shows that the agent is successful in reaching its goal which is the coordinate of the Sun. The proposed AI algorithm also eliminates the use of encoders as the algorithm can constantly feedback the errors due to the misalignment of the sun tracker. Thus, the sun tracker is able to adjust and align itself with the sun at the central receiver.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE LISTINGS

LIST OF SYMBOLS / ABBREVIATIONS

$M_{i,j}$	matrix of central point of each facet mirror
$\gamma_{i,j}$	tilted angle of facet mirror about X axis
$\sigma_{i,j}$	tilted angle of facet mirror about Y axis
A	Azimuth angle
h	solar elevation angle
T	matrix of coordinates of target of NIDC
δ	declination angle
$Q(St, At)$	Q-value at current state and action
$Rt(s, a)$	reward an agent get for its current state and action
α	learning rate of update rule
γ	discount rate of update rule
ε	epsilon of Q-learning

AI	artificial intelligence
ANN	artificial neural networks
AP	average precision
BIPV	building integrated photovoltaic
BN	Bayesian network
CNN	convolutional neural network
CNN	convolutional neural network
CPV	concentrator photovoltaic
DACPV	dense-array concentrator photovoltaic
DNI	direct normal irradiance, W/m^2
E-ELAN	Extended Efficient Layer Aggregation Network
FPS	frames per second
mAP	mean average precision
ML	machine learning
NIDC	non-imaging dish concentrator
NIPC	non-imaging planar concentrator
PV	photovoltaic
SCR	solar cell concentration ratio

YOLO	You Only Look Once
YOLOR	You Only Look One Representation

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Global energy consumption is increasing due to the growth in population, and therefore the energy production is required to increase to meet the demand. However, this leads to environmental issues such as global warming and air pollution due to the increase usage of non-renewable energy sources such as oil, coal and natural gas. As a result, renewable energy sources such as solar energy, wind energy and biomass are considered as an alternative to the current energy framework. Solar energy is abundant as it can be harnessed directly from sunlight, which is one of the best renewable energy sources. Solar energy is the world’s fastest growing source of power and according to the World Economic Forum, the price for solar energy has dropped and is much cheaper than fossil fuels since 2021 (Masterson, 2021).

Solar technologies such as photovoltaic (PV) modules and concentrated solar power are used to collect and harnessed solar energy. Among PV technologies, flat PV systems are often installed as rooftop-mounted or building-integrated systems as flat PV has a competitive production cost. On top of that, flat PV has a comparatively lower maintenance cost and is easier to install. However, flat PV requires a large land area to generate a large amount of electricity which is inefficient. Therefore, concentrator photovoltaics (CPV) systems are proposed as an alternative to flat PV systems. CPV is also a PV technology which utilizes lens or mirrors to focus the incident solar radiation onto a solar cell, and would not cause any air or water pollution. In comparison with flat PV systems, CPV drastically reduced the area of solar cells as only a large area of mirrors or lenses are required to concentrate sunlight onto a small area of solar cells. Furthermore, the effect of temperature on PV modules can be reduced as the area of solar cells is smaller than in flat PV systems.

A solar tracking system can be used to orientate the CPV system towards the sun at all times, which allows more sunlight to be collected, thus increasing the electricity generation of the system. In recent years, artificial intelligence (AI) has been applied to solar concentrator systems to increase the accuracy of predicting the sun position, and thus reducing tracking errors in the sun tracking algorithm. In a CPV system, external factors such as mechanical structure of the concentrator and immoderate high solar concentration ratio affects the sensitivity of the concentrator. Therefore, an AI integrated sun-tracking system is proposed to reduce the tracking errors by detecting the Sun using object detection algorithm. Furthermore, reinforcement learning is used to determine the movement options of the motor and orientate the solar concentrator to the correct position.

1.2 Importance of the Study

An AI integrated sun tracking algorithm is important to increase the accuracy in tracking the position of the sun. The normal of the surface of the CPV has to always be parallel to the normal of the sun to collect maximum amount of incident solar radiation. The tracking errors can be reduced through computer vision algorithms such as object detection, and provide corrections to the tilt and roll angles of the CPV system based on the difference in coordinates of the actual sun position and the centre of the camera using reinforcement learning. The AI integrated sun-tracking system is proposed to achieve long term performance and reliability, and able to achieve high accuracy in long periods of time.

1.3 Problem Statement

The solar energy collected by the sun-tracking system depends on the accuracy of the sun-tracking algorithm, which is different for each type of sun-tracking system. Furthermore, the solar image formed on the on-axis target can also be easily affected by gear backlash and wind load, and an absolute encoder is used to reduce the errors produced by the sun-tracking system. Therefore, this project proposed an Artificial Intelligent (AI) sun-tracking algorithm to constantly detect the position and midpoint of the sun with a low-cost webcam without applying different sun-tracking algorithms. Besides that, the AI algorithm is capable of constantly adjusting the sun-tracking system to ensure the orientation angle of the sun-tracker is always parallel to the sunlight. Thus, the tracking errors of the sun-tracking system can be reduced with the AI sun-tracking algorithm and without using an absolute encoder.

1.4 Aim and Objectives

The aim of this study is to tackle the long-term sun-tracking issue of a non-imaging dish concentrator (NIDC) by integrating AI into the sun-tracking control algorithm to enhance the accuracy, reliability and long-term performance. The objectives of this project are described as follows:

1. To develop an AI algorithm with the capability of solar image recognition at the central receiver.
2. To design a sun-tracking system by integrating AI into the control system of the sun-tracking algorithm.
3. To evaluate the sun tracking system based on the new AI algorithm.

1.5 Scope and Limitation of the Study

The project title for this project is “Artificial intelligent integrated into sun-tracking system to enhance the accuracy, reliability and long-term performance in solar energy harnessing”. The scope of the project is to evaluate the performance of AI integrated sun-tracking algorithm for the CPV system that will ensure long term performance and reliability. The AI integrated sun tracking system must also produce high accuracy for long periods of time. The AI algorithm will predict the sun position by considering various weather conditions during detection stage.

1.6 Report Overview

This report describes the development of this study and is composed of the following chapters.

Chapter 1 is the introduction which consists of the background and objectives of the project. The importance of the study and problem statement has been described.

Chapter 2 is the literature review which introduces the literature review of CPV systems and AI integrated solar trackers. This chapter also discuss on problems faced in CPV design and AI integrated sun-tracking algorithm in past literature.

Chapter 3 is the methodology which introduces the methodology and design of the prototype. The flowcharts for the motor control unit and sun tracking algorithm are introduced.

Chapter 4 is the results and discussion which describes the evaluation of the performance of the AI sun-tracking algorithm. The algorithm is tested under different conditions and the results are presented.

Chapter 5 is the conclusions and recommendations of the study of the proposed AI integrated sun-tracking algorithm. The recommendations of improving the project and future work are also discussed.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter introduces the background of CPV systems, sun tracking systems and also various AI models. An AI sun-tracking algorithm which integrates object detection algorithm and reinforcement learning is proposed to detect the position of the sun which can be used in CPV system such as shown in Figure 2.1. Past literature on the computer vision algorithms and also weather classification in sun tracking algorithms is also discussed. Furthermore, deep learning models such as object detection algorithm and reinforcement learning are discussed in this chapter.

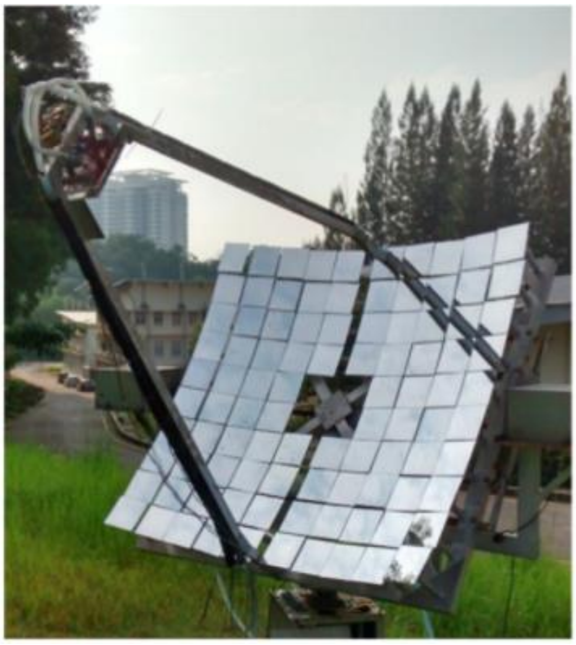


Figure 2.1: The previously constructed prototype of dense array concentrator photovoltaic (DACPV) at TARUC, Setapak (Chong *et al*., 2017). The DACPV prototype consisted of a non-imaging dish concentrator (NIDC) and a DACPV receiver.

2.2 Concentrator Photovoltaics (CPV)

Concentrator photovoltaic (CPV) system is a PV technology that converts solar energy into electrical energy. CPV uses a large area of lenses or mirrors to focus sunlight onto a small area of photovoltaic cells, which reduces the area of solar cells. The effect of temperature on CPV systems are minimal as the area of solar cells are lesser than in normal flat PV. The solar cell concentration ratio (SCR) is defined as the ratio of optical area to the solar cell area, and is divided into three classes: low concentration photovoltaic system (LCPV), high concentration photovoltaic system (HCPV) and ultra-high concentration photovoltaic system (UHCPV). The description of the three CPV classes are shown in Table 2.1, which describes the typical concentration ratio, tracking system and the suitable type of converter of each class.

Table 2.1: Description of CPV classes.

CPV class	Typical concentration ratio	Tracking	Type of converter
LCPV	<100	One or two-axis	c-Si or other cells
HCPV	300-1000	Two-axis	III-V multi-junction solar cells
UHCPV	>1000	Two-axis	Multi-junction solar cell

2.2.1 Type of Concentrators

Various type of solar concentrators is available in the market, and can be divided into either reflective optics or refractive optics. Solar concentrators that have a central receiver (heliostat tower), or in the shape of parabolic troughs or dishes are examples of solar concentrators with reflective optics; while concentrators that uses Fresnel lens are examples of refractive optics.

A concentrator that is shaped like a parabolic trough is constructed as a long parabolic mirror with a Dewar tube at the focal point along the concentrator axis. The parabolic trough concentrator uses line focusing optics, which allows the incident solar radiation to reflect onto the focal light toward a receiver. This concentrator system is usually aligned on a north-south axis, and solar trackers can be installed to track the position of the sun throughout the day.

A parabolic dish concentrator is constructed by an array of parabolic dish-shaped mirrors. This concentrator utilizes the optical properties of a parabolic curved surface to concentrate incident solar radiation towards the focal point. A dual-axis solar tracker is installed on the parabolic dish concentrator to increase the collected solar concentration.

A heliostat is a solar power facility which consists of a central receiver (power tower) and uses a field of dual-axis tracking mirrors. Each tracking mirrors is individually controlled by a computer control system to reflect the incident solar radiation towards the central receiver. The solar concentration ratio of the combined effect of heliostats is theoretically possible to achieve ultra-high concentration photovoltaic that exceeds 1000 suns.

A refractive optics concentrator that uses Fresnel lens refracts the light to concentrate the incident solar radiation on the lens surface to a solar cell. A Fresnel lens solar concentrator is cheaper compared to other concentrators as the lens are moulded out of inexpensive plastic. This is due to the acrylic materials which are cheaper per square meter than mirrored glass, and the optical efficiency is greater as it only passes through one weathered or dirty surface. The point-focused Fresnel concentrator also uses a dual-tracking system to increase collected solar concentration.

2.2.2 Non-Imaging Dish Concentrator (NIDC)

Non-imaging concentrators are concentrators that do not form any optical image of a source, as opposed to an imaging concentrator which forms an image of the sun by reflecting it on the receiver. Various design of non-imaging concentrators is proposed in the past such as the non-imaging planar concentrator (NIPC) by Chong *et al.* (2009) which consists of numerous flat facet mirrors arranged in a square array and spaced evenly at the same height. The proposed NIPC produces high SCR and uniform solar irradiance, however is affected by the increasing shading effect as the ratio of the focal distance to width of the NIPC decreases. Furthermore, the facet mirrors that are further from the centre of the NIPC requires higher tilting angles to reflect the incident solar radiation towards the focal point, while causing more incident solar radiation from an adjacent facet mirror to be blocked.

In order to eliminate the blocking and shading effect from adjacent facet mirrors, Tan *et al*. (2014) proposed the NIDC with single-stage focusing and developed a computational algorithm to determine the initial configuration of facet mirrors. The NIDC consists of a $2m \times 2n$ array of identical facet mirrors which are arranged at different heights. The locality of each facet mirror in the NIDC is indexed as (i, j) where i and j represents the position of facet mirror at the ith column and jth row of the NIDC respectively. The origin of the coordinate system is located at the centre of the NIDC with the coordinates assigned as $(0,0,0)$. The coordinate of the central point of each facet mirror is represented by the matrix, $M_{i,j}$ and is written as

$$M_{i,j} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix}_{i,j} \tag{2.1}$$

The initial configuration of each facet mirrors in a NIDC is shown in Figure 2.2, where the configuration is divided into four quadrants.

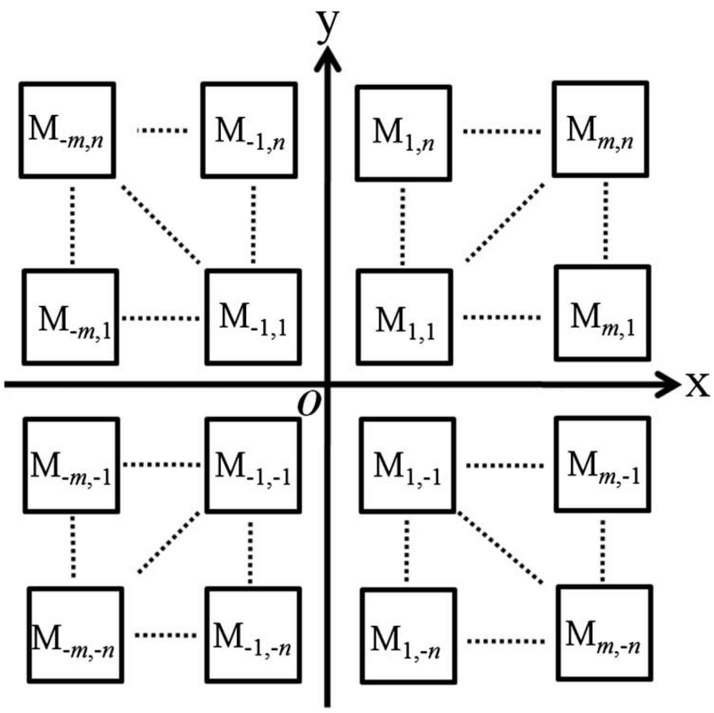


Figure 2.2: The initial configuration of the facet mirrors of the NIDC. The concentrator can be divided into four quadrants, which are the top left, top right, bottom left and bottom right (Tan *et al.*, 2014).

The target is located at the focal point of the NIDC, which can be regarded as the height of the receiver from the centre point of the NIDC. The coordinate of the receiver, T is defined as

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ F \end{bmatrix} \tag{2.2}$$

where F is the focal point of the NIDC and is also the shortest distance from the origin to the receiver. The tilting angle of each facet mirrors is given by tilted angle about Y axis, $\sigma_{i,j}$ and tilted angle about X axis, $\gamma_{i,j}$ which are given by

$$\sigma_{i,j} = \sin^{-1}\left[\frac{M_x}{2\cos\theta_{i,j}\sqrt{M_x^2+M_y^2+(F-M_z)^2}}\right] \tag{2.3}$$

$$\gamma_{i,j} = tan^{-1}\left[\frac{M_y}{(F-M_z)+\sqrt{M_x^2+M_y^2+(F-M_z)^2}}\right] \tag{2.4}$$

respectively. Figure 2.3 shows the comparison of the initial configuration of facet mirrors between NIPC and NIDC, where the facet mirrors of the NIDC is shifted upwards along the Z axis, and has a smaller gap, G between each facet mirror compared to NIPC. The configuration of facet mirrors of NIDC greatly reduces the shading effect caused by adjacent facet mirror and also reduces the area of the concentrator as the gap between facet mirrors is smaller.

2.3 Sun Tracking System

Sun tracking system can be categorized into single-axis and dual-axis sun tracking system. Single-axis sun tracking system moves only in one direction, while dual-axis sun tracking system moves in two directions. In dual-axis sun tracking system, it can be divided into tilt-roll tracking and azimuth-elevation tracking. Dual-axis sun tracking system is often preferable compared to single-axis sun tracking system as it has an extra degree of movement, and faces the sun more accurately. As the position of the sun changes throughout the year, solar harnessing is maximized using dual-axis sun tracking system as it allows the normal of the sun to be parallel to the normal of the solar concentrator.

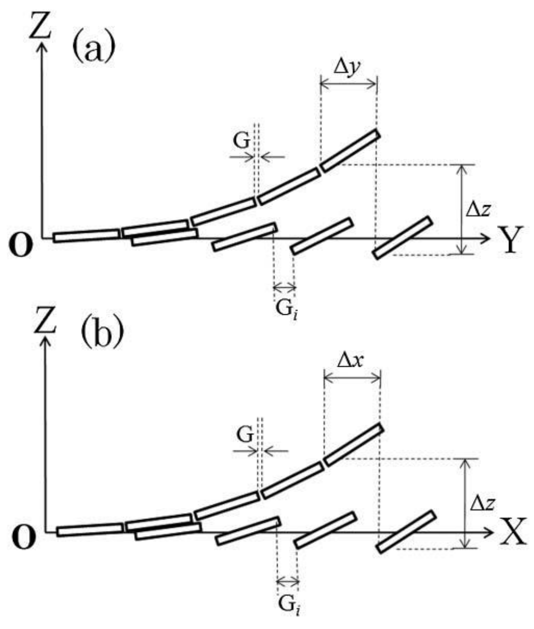


Figure 2.3: The conceptual drawing which defines the initial positions of each facet mirrors (Tan *et al*., 2014).

The position of the sun changes throughout the day and varies with seasons, which affects the amount of sunlight collected by a fixed CPV system. Thus, a solar tracker can maximise the amount of sunlight collected by facing the CPV system towards the sun at all times. Prediction of the sun position relative to the concentrator based on the given input is important to collect the maximum solar energy from the sun. The position of the sun with reference to the frame of concentrator is governed by a few parameters, such as declination angle, solar time, hour angle, equation of time and local clock time, which are also the input parameters of the solar tracking algorithm.

2.3.1 Non-Imaging Dish Concentrator (NIDC)

The sun tracking algorithm requires several input parameters to accurately determine the position of the sun. The declination angle, δ refers to the angle between the equatorial plane and a line drawn from the centre of the Earth to the centre of the sun. The declination angle varies with the seasons as the Earth orbits around the sun on the ecliptic plane, and can be approximated as

$$\delta = \sin^{-1}\{0.39795 \cos[0.98563(N - 173)]\} \qquad (2.5)$$

where N is the day number starting at 1st January which is defined as $N = 1$.

In an azimuth-elevation solar tracking system, the movement of the solar concentrator depends on the azimuth and elevation angles. The solar azimuth angle, A is the angle between the projection of the centre of the sun onto the horizontal plane that is towards the south direction. The azimuth angle can be approximated as

$$A = \cos^{-1}\left\{\frac{\sin(\delta)\cos(\phi) - \cos(\delta)\cos(\omega)\sin(\phi)}{\cos\phi}\right\} \tag{2.6}$$

where ϕ is the local latitude and $A = 360° - A$ if $\sin(\omega) > 0$.

The solar elevation angle is the elevation angle of the sun, which is between the direction of the sun and the horizon. The solar elevation angle can be approximated as

$$h = \sin^{-1}\{\sin(\delta)\sin(\phi) + \cos(\delta)\cos(\phi)\cos(\omega)\} \tag{2.7}$$

2.4 Artificial Intelligence

In recent years, AI has been applied to sun-tracking system to enhance the accuracy and reliability in long-term solar harnessing. The AI integrated sun-tracking system is improved through signal processing, image recognition and prediction of the sun path. The motor control of the sun-tracking system is aided by AI to predict the elevation and azimuth angle based on given input such as latitude, longitude, time and date. Solar image may not be formed on the on-axis target of the solar concentrator due to imperfection of the mirror, refraction of the sun or other relevant factor, and can be corrected through AI. Machine learning (ML) can be divided into supervised and unsupervised learning, where labels known as the target variable is predicted from a set of features in supervised learning while the data used in unsupervised learning is unlabeled. ML algorithms are used to solve either a classification or regression problem, where the former is the task of predicting a discrete class label and the latter is the task of predicting a continuous value. The common algorithms that can be used to model a regression problem include linear regression, decision tree, random forest, polynomial regression and support vector machines.

2.4.1 Recent Developments in AI Integrated Sun-Tracking System

In recent years, many studies on the AI integrated sun tracking systems have been published. Various AI models are used to control the sun tracking systems such as Logistic Regression, Fuzzy Logic, Genetic Algorithms and many more. Fuzzy logic principle is widely applied in control systems as it is flexible and allow modifications in the rules. A sun tracking control system that adapts Adaptive Neural Fuzzy Inference System (ANFIS) principle is proposed for single-axis and dual-axis sun tracking system (Nadia et al., 2020). The proposed ANFIS algorithms only uses three input variables which are the month of the year, day of the month and solar time of the day to predict the tilt and orientation angles for a dual-axis sun tracking system. The algorithms predicted the optimum tilt and orientation angles with high accuracy and low mean square error. The study shows that the system with three membership functions achieved a 100 % accuracy and mean square error of 6.580×10^(-6) for a dual-axis sun tracking system. However, the proposed algorithm may not be suitable for more complicated systems such as NIDC systems, since the algorithm consider lesser factors. In a more complicated system, more external factors that will affect the accuracy in sun tracking have to be considered, such as wind load, pedestal tilt or apparent sun position.

Aside from the prediction of sun position, the performance of a CPV system is affected by weather conditions. In sunny conditions, the direct normal irradiance (DNI) is high and the position of the sun can be measured directly using most image processing methods. However, in overcast day, the presence of clouds affects the brightness of sun which makes the detection of sun position harder. Therefore, AI is applied to sun-tracking algorithms that considers weather conditions, thereby improving the accuracy of sun position prediction. The weather conditions are dynamic and uncertain, and will change throughout the day, where the DNI is affected by several factors such as the blocking of sun by clouds, presence of cloud and rain.

Bayesian network (BN) algorithm is a supervised learning algorithm to handle problems with uncertainties and is often used in solving classification problems. A CPV tracking system with BN that considers various weather conditions is constructed (Kim & Cho, 2019). The proposed CPV system is applied with optimal selections of sun-tracking algorithms, which has a higher performance compared to those with only a single algorithm. The proposed system has a 93.9% accuracy and 16.58% higher energy generation was obtained from 1630 collected data. In the classification of weather conditions, the shape and type of the cloud was considered as it primarily affects the amount and position of sunlight. The nine weather conditions that are recognized in the proposed system are shown in Figure 2.4, which are classified according to the amount of sunlight and various physical features of clouds. However, the deviation is large for a complicated case such as in case 2 that has a complex cloud shape. The paper mentioned that in future works, the classification of weather of the proposed system will be improved by using additional data such as weather service information and more deliberate domain knowledge on visual features of various kind of weathers.

Although various AI algorithms have been proposed, most systems have system limitations due to cost and operational problems. Thus, a sun tracking system that applies computer vision techniques with low-cost hardware have been proposed (Carballo et al., 2018). The system uses object detection algorithms with region proposal techniques and convolutional neural network (CNN) to track the position of the sun. The control strategies of the system have been improved through key variables provided such as cloud movement prediction, block and shadow detection, atmospheric attenuation and measures of concentrated solar radiation, which improves the system performance. A low-cost camera is placed at the centre of the heliostat to determine the coordinate of the centre of the sun and centre of the target using object detection algorithms. The midpoint of the coordinate is calculated and the motor in the heliostat will rotate in such a way that the centre is pointing to the that midpoint. This approach is advantageous as the algorithm is independent of the solar technologies used, the size of the system, location and also time, and is also not affected by external factors such as pedestal tilt, wind load and apparent sun position. However, the solar image might not be formed on the central target even though the heliostat is facing to midpoint between centre of the sun and the central target due to imperfection of mirror.

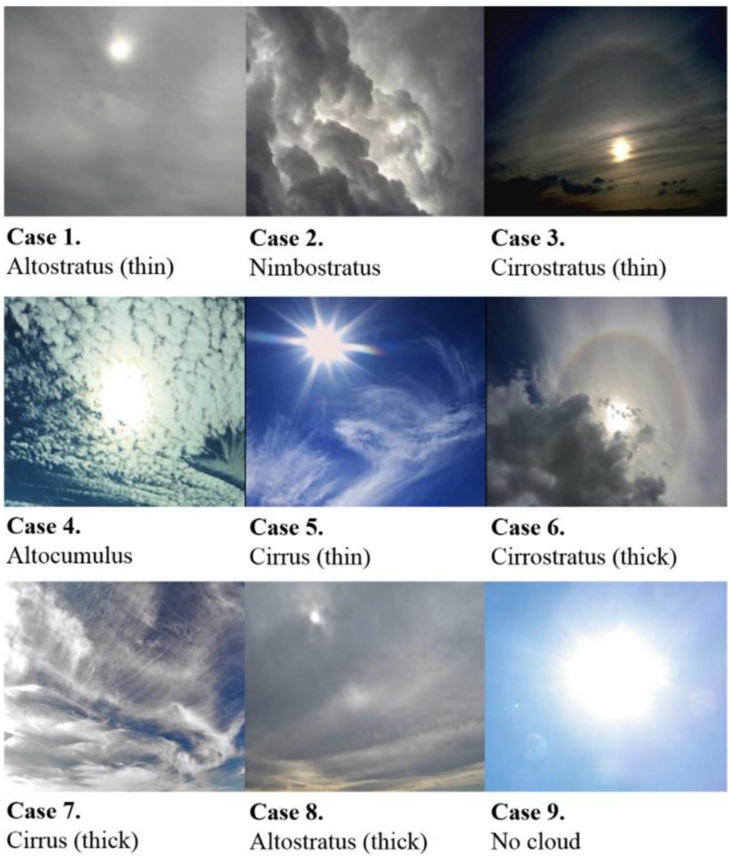


Figure 2.4: Weather classification according to cloud and sky conditions (Kim & Cho, 2019).

2.4.2 Artificial Neural Networks (ANN)

An artificial neural network (ANN) can recognize patterns through neural networks which are composed of multiple neurons to perform predictions based on the input. ANNs are comprised of an input layer, one or more hidden layers and an output layer, each containing various number of neurons or also known as nodes. Each node connects to another and has an associated weight and threshold value. The node is activated when the output of any individual node is above the specified threshold value and the data is sent to the next layer of the network. The accuracy of the neural network is improved as more training data is input to the neural network.

Backpropagation method is used in ANN to adjust the connection weights to compensate for each error found in the learning process by using the loss function. The partial derivative of the loss function respective to each individual weights in the ANN is calculated, and is followed by applying chain rule to each path and the corresponding error is found. In backpropagation method, gradient descent is applied in the network to minimize the significant errors in the output.

Convolutional neural networks (CNN) are a type of neural networks which has greater performance in handling image, speech and audio signal inputs. CNNs are usually utilized in image and pattern recognition and also in computer vision. A CNN contains 3 layers, which are the convolutional layer, pooling layer and the fully-connected layer. In the convolutional layer, it uses convolutional filters which has adjustable weights to convolve the input image. The dot product between each subgroup of pixels encapsulated by it is performed when the input image is passed to the filter and new pixel values are calculated.

The pooling layers reduced the number of parameters by conducting dimensionality reduction. The pooling layer are similar to the convolutional layers; however, they do not contain weights as it sweeps a filter across the entire output. Although some information is lost in the pooling layer, it helps to reduce the complexity of the model and limit risk of overfitting.

The fully-connected layer is the last layer in the CNN, and perform classification based on the extracted features through the previous layers and filters. The fully-connected layers applied the softmax activation function to classify inputs as compared to ReLu functions used in convolutional and pooling layers.

In machine learning, a hyperparameter is a constant parameter that are specified before the learning process begins, and the value of the parameters are derived via the learning process. The hyperparameters include the learning rate, number of hidden layers and batch size which has an effect on the learning process. However, there are values of some hyperparameters which are dependent of its hyperparameters such as the size of some layers are dependent on the overall number of layers. The learning rate in an ANN defines the size of the corrective steps that are required by the model to adjust for each error in an observation. A shorter training time is required for a high learning rate but is accompanied with a lower accuracy, where the reverse is also possible.

2.4.3 Background of Computer Vision

Computer vision is an important field of this study, which is a subset of AI that allows computers and systems to extract information from digital media such as images, videos or other visual inputs. Based on the collected information, computer vision allows the system to take action or make recommendations using deep learning or convolutional neural network (CNN). This would allow the systems to recognize the images and classify them accordingly, and this image recognition process can be divided into four categories, which are classification, tagging, detection and segmentation.

In object classification, a specific object can be identified and a bounding box is drawn around the object. The system is able to produce the class of the identified object, however it is not able to locate the position of the object. Thus, the coordinates of the identified object cannot be obtained, and is only able to identify the class that the object belongs to. For example, in an image where a dog is present, object classification algorithms are able to identify the class of the object, which is a dog, but is unable to give the location of the dog.

In object tagging, multiple objects can be identified by drawing bounding box around each specific object. It is similar to object classification; however, object tagging is able to produce the class of different objects and not just a specific class. Same as object tagging, the coordinates of the object cannot be obtained. For example, in an image where a dog and a cat are present, object tagging algorithms can identify the class of both objects, which is a dog and cat respectively, but is unable to locate the position of both objects.

In object detection, it is the same as in object tagging. However, object detection algorithms are able to obtain the coordinates of the specified objects. The output of the algorithm are the coordinates of the bounding box, width, height and class of the object.

In object segmentation, the image is divided into different regions based on the characteristics of the pixels. Therefore, bounding boxes are not required to identify the objects as every pixel is related with an object type. Object segmentation is further divided into semantic segmentation and instance segmentation. Semantic segmentation algorithms mark all objects of the similar object type with one class label while instance segmentation algorithms mark similar objects with separate class labels.

2.4.4 Object Detection

In this study, an AI algorithm is required to predict the position of the sun while also considering weather conditions. Therefore, object detection algorithms are chosen to obtain the coordinates of the sun, the solar image formed, facet mirrors, on-axis target and also clouds. Each of the detected objects will be given a class label and also its coordinates, which allows the correction of the azimuth and elevation angles based on the coordinates of the detected objects. In object detection algorithms, one-stage and two stage object detection algorithms are available. Object detection algorithms extract features from the input image and solves two subsequent tasks, which are to first identify an arbitrary number of objects and then classify each object and estimate its size with a bounding box.

In two-stage detectors, deep networks are used to provide an approximate object region, followed by object classifications based on the extracted features. Although this kind of detectors achieve the highest detection accuracy, it consumes large amount of computational time as many inference steps per image are required. In one-stage detectors however, the object approximated region is not proposed, and bounding boxes are directly predicted over the images. This would drastically reduce the computational time needed, but is not efficient in recognizing irregularly shaped objects. Various one-stage detectors algorithms are proposed, including the newest algorithm that was proposed in 2022, which is the You Only Look Once (YOLO) object detection algorithm (Wang et al., 2022).

2.4.5 YOLOv7

The YOLO framework is made up of three main components, which are the Backbone, Head and Neck. The Backbone extracts the essential features of an image, and the Neck will feed the extracted features to the Head. Feature maps are extracted by the Backbone and feature pyramids are created once the feature maps are collected by the Neck. Finally, the feature maps are collected by the Head which consists of output layers to execute the final detections. In summary, the image frames are featured through the Backbone, and the features are combined in the Neck and is finally passed to the Head where the YOLO algorithm predicts the locations of bounding boxes, classes of the bounding boxes and also the objectness of the bounding boxes.

In YOLOv7, the speed and accuracy of the algorithm was improved through the modification of the architectural structure of the YOLO framework. The YOLOv7 model is trained using the COCO dataset entirely instead of the pretrained backbones by ImageNet, similar to Scaled YOLOv4. The E-ELAN (Extended Efficient Layer Aggregation Network) architecture is the computational block in the YOLOv7 backbone, where it is designed based on factors such as memory access cost, activations and gradient path that will impact the speed and accuracy of the algorithm. The gradient to back-propagate through the layers affects the amount of memory it takes to keep the layers in memory; therefore, the structure is better in learning with a shorter gradient. Thus, the YOLOv7 framework learn better as compared to other models through the E-ELAN architecture (Wang, Bochkovskiy & Liao, 2022).

As compared to the previous YOLO models, YOLOv7 surpasses all the previous object detection algorithm in speed and also accuracy as shown in Figure 2.5. The YOLO models are compared using two metrics which are mAP (mean average precision) and FPS (frames per second) respectively. The YOLOv7 algorithm has a higher mAP compared to models such as YOLOv4 and YOLOR due to a smaller parameter size. The YOLOv7 normal model has a validation AP of 51.2% and has only 37 million parameters, where the parameter size of other YOLO models exceeded 46 million. YOLOv7 also achieved the highest FPS, and runs at 161 FPS and has a test AP of 51.4%.

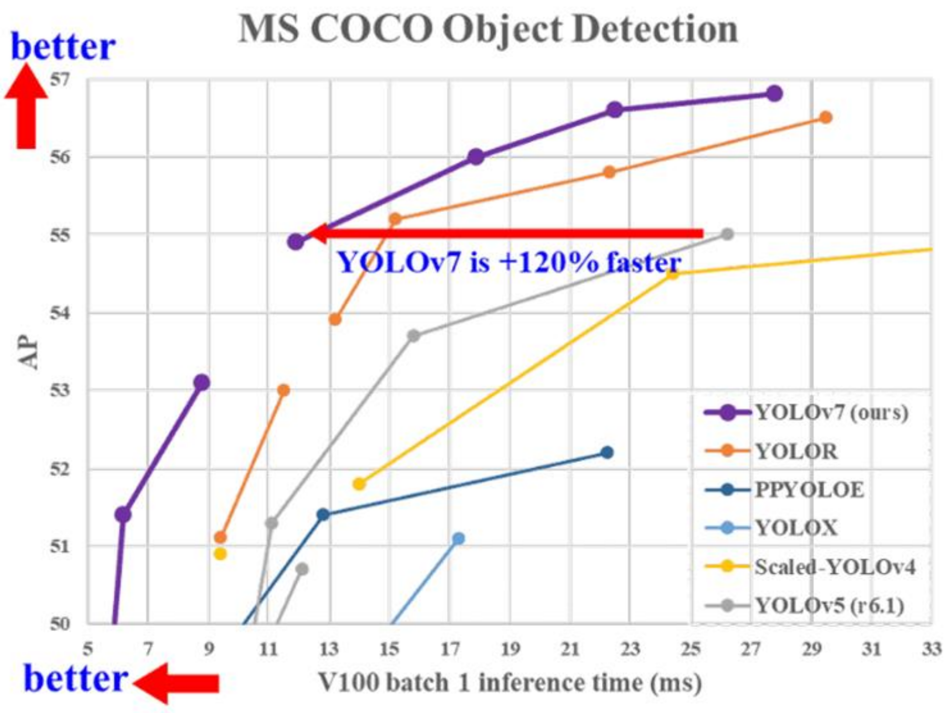


Figure 2.5: The Average Precision (AP) and speed of YOLOv7 model compared to previous YOLO model (Wang, Bochkovskiy & Liao, 2022).

2.4.6 Reinforcement Learning

Reinforcement learning is employed in many applications to find the best possible behaviour or path it should take by maximizing the reward for a given action. Reinforcement learning differs from supervised learning where the reinforcement agent decides what to do to perform the given task. Supervised learning however trains its model by using a training dataset with labeled data, which provides the correct answer. Therefore, reinforcement learning has to learn from its experience in the absence of a training dataset. The difference between reinforcement learning and supervised learning is shown in Table 2.2.

In reinforcement learning, an intelligent agent is used to perceive and interpret its environment and achieve a certain goal. The reinforcement learning is a training method that is based on rewarding desired behaviours and punished undesired ones. This method assigned desired behaviours with positive values to encourage the agent and assigned undesired behaviours with negative values to penalize the agent. Thus, the agent will seek long-term and maximum overall reward to achieve an optimal solution.

Table 2.2: Difference between reinforcement learning and supervised learning.

Reinforcement Learning	Supervised Learning
Decision is made depending on the state of the current input and the next input depends on the output of the previous input	Decision is made on the initial input given at the start
Decision is dependent of each other	Decision are independent of each other
Example : Chess game	Example : Object recognition

The input of the reinforcement learning is where the initial state from which the model will start, and there are many possible outputs due to the variety of solutions to a particular problem. The training will proceed and is based on the input, where the model will return a state and the model is either rewarded or punished based on its output. The model of the reinforcement learning will continue to learn and the best solution is decided based on the maximum reward.

Reinforcement learning can be categorised into two types, which are positive reinforcement and negative reinforcement. Positive reinforcement will have a positive effect on the behaviour of an event and will increase its strength and frequency of that behaviour. This will maximize the performance of the algorithm and allowed it to sustain change for a longer period of time. On the other hand, negative reinforcement will strengthen the behaviour of the event due to a negative condition is stopped or avoided. The negative reinforcement will increase the behaviour and provide resistance to a minimum standard of performance.

In reinforcement learning, a state space defines all the possible state that the agent can be in. The state space could be discrete or continuous, and contains all the information that will determine the next state such as the coordinates of the points, the maximum steps taken and the velocity of the points.

As compared to state space, an actions space only defines all possible action the agent could take. For instance, the action space describes the direction or action the point can take such as moving left, right, up or down. Thus, action space is usually finite and contains lesser elements as compared to the state space.

The transition probability is the probability of the agent to get from its current state, St to the next state, $St + 1$ and executed an action, At. The transition probability indicates that the behaviour of the agent is probabilistic, and the agent might not behave as instructed. Thus, a reward or penalty system is applied to train the agent to behave in a certain way, and is an important component in Q-Learning.

The reward in reinforcement learning is a function of state and action such that the reward function is $R(St, At)$. The reward is an incentive to the agent to encourage it to reach its desired goal more quickly and efficiently. A positive reward is set for the desired state and a negative reward or penalty is set for each extra step taken by the agent. In practical situations, the reward system is not well defined, and is normally defines by gathering the information to estimate the rewards through running a large sample of random actions executed by the agent.

The policy takes the current state as an input and the action state as the output, and maps a state into an action. The agent will constantly interact with the environment and gather information about the reward it gets to maximize its total rewards. The agent will execute an action as specific states and search for the most optimal action to take for each state.

2.4.7 Q-Learning

In reinforcement learning, Q-Learning is a type of value-based learning algorithms, where the objective of the agent is to optimize the value function suitable for each problem. The value function is similar to the reward function, but only access a particular action in any particular state for a given policy. The value function takes account of all future rewards the agent gets from executing a particular action and not just its current reward. A Q-table is created in a Q-Learning process which will store all the Q value for each state and each possible action.

The learning process in Q-Learning has four steps, which are the initialization stage, space exploration stage, reward observing stage and the value function update stage. All the Q-values in the Q-table are first initialized to 0, where the agent has no knowledge about the environment around it. The agent will then continue exploring the environment through executing different actions at different states. There are various considerations to set how the agent should behave in this stage, such as if the agent is set to execute an action which always return the highest value function, the problem will definitely converge to a global minimum, but this process is slow. Therefore, for a large state action space, the agent is set to occasionally choose its action randomly, and there is a chance that this process will return the optimal value much faster. The epsilon greedy approach is one such process, where the epsilon has a probability to choose a random action than the action to maximize the value function. With epsilon greedy approach, the parameter epsilon, ε controls the amount of exploration and determines the randomness in action selections (Watkins, 1989).

During the exploration stage, the reward obtained from executing an action at state, St to go to the next stage, $St + 1$ is observed by the agent. It will then update the value function for that particular state and action pair by using the update rule as described by Equation 2.8,

$$Q(St, At) = Q(St, At) + \alpha * [Rt(s, a) + \gamma * \left(maxQ(St + 1, \alpha)\right) - Q(St, At)] \tag{2.8}$$

where the $Q(St, At)$ at the left-hand side is the updated Q-value, and the one at the right-hand side is the original Q-value, α is the learning rate, $Rt(s, a)$ is the reward an agent get for its current state and action, γ is the discount rate and

$maxQ(St+1, \alpha)$ is the maximum future value at state, $s+1$ if the agent take the best action, a. In the update rule, the learning rate and the discount rate are two hyperparameters that requires tuning to improve the Q-Learning. The discount rate is a parameter to incentivize the agent to achieve its objective faster, as it discounts its future value at the next state, $St+1$. The learning rate determines the weights for the current value as compared to its future value.

2.5 Summary

NIDC is an efficient solar concentrator design which minimizes the shading effect caused by adjacent facet mirrors. Thus, the AI integrated sun tracking algorithm can greatly increase the accuracy of predicting the sun position and reduce the tracking errors. Based on the literature review, a CPV system with AI integrated sun tracking algorithm can provide maximum incident solar radiation and increase the electricity generation. Furthermore, the AI sun-tracking algorithm can reduce tracking errors by constantly adjusting the sun-tracking system using reinforcement learning.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter provides the details regarding the proposed artificial intelligent (AI) sun-tracking system. The deep learning model used in the sun-tracking system consists of YOLOv7 for object detection and Q-learning for motor control. The training process of the deep learning model is discussed. Unfortunately, the AI sun-tracking algorithm is not able to be tested in real-time as the CPV system was not able to be built in time.

3.2 System Design of the AI Sun-Tracking System

The proposed AI sun-tracking system is used in the CPV system consisting of a NIDC prototype installed at the rooftop at KB Block, Universiti Tunku Abdul Rahman (UTAR), Sungai Long Campus. The NIDC prototype consists of an array of 8×8 mirrors with the centre 4 removed to avoid being blocked by the target, and requires 60 mirrors in total. The dimension of one facet mirror is $120\ mm \times 120\ mm$ and the focal distance is $1000\ mm$. Each mirror has a gap of $5\ mm$ between each other to avoid shading effect by adjacent facet mirrors. The NIDC prototype is adjustable and uses springs to adjust the height of each facet mirror, the same design used by Chong et al. (2017). A motor control unit that controls the two stepper motors are installed and is controlled by Arduino UNO.

3.2.1 Design Architecture

The design architecture for the AI sun-tracking system is as shown in Figure 3.1. In the AI sun-tracking system, a webcam with extension wire is used to capture the image frame and provides input to the embedded system for AI processing. A laptop will receive the input from the webcam and process the object detection algorithm to obtain the coordinates of the Sun. After the sun coordinate is obtained, the Q-learning algorithm is run to provide the minimum steps taken to reach the sun coordinate. A signal will then be sent to Arduino UNO to align the CPV system with the sun coordinate. The CPV position will constantly feedback to the system throughout the day and sent a signal to the hardware control of the motor driver to adjust the tilt axis and roll axis.

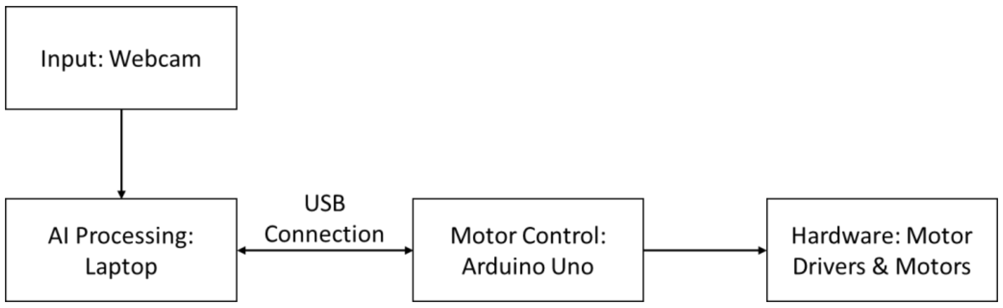


Figure 3.1: Design architecture for the AI sun-tracking system.

3.2.2 Motor Control Unit

The motor control unit is designed to control the tilting angle and rolling angle of the CPV system. This unit consists of a microcontroller unit, stepper motor drivers, absolute encoder and stepper motors. An Arduino UNO will be used as the microcontroller unit to produce pulse width modulation (PWM) pulse to control the speed of the stepper motors. A PWM is a technique to obtain analogue values by using digital control to create a square wave and the signal is switched between on and off. Thus, the speed of the stepper motor can be controlled by simulating this on-off pattern which affects the signal speed of the microcontroller.

Two CSD2120-P VEXTA stepper motor drivers will be used to control the rotating direction of each motor individually in the clockwise or anticlockwise direction. The stepper motor driver is connected to one PK296A2-SG36 Oriental Motor stepper motor each, which provides 12 Nm torque. The stepper motor will then be connected to a gearbox which provides tilt-roll rotation. One of the stepper motors is installed at the tilt shaft to rotate the tilt rotation, while the other stepper motor is installed at the roll shaft to rotate the roll rotation.

3.2.3 Arduino

An Arduino code is written to receive the signal generated from the AI sun-tracking algorithm. The step size of the motor is 0.05° per revolution which provides 7200 number of steps in one revolution. The Stepper library from Arduino is imported to the code, which provides the function ‘Stepper’, ‘setSpeed’ and ‘step’. The minimum steps per move is defined as 200 steps, which allow precise movement for the sun-tracking system. The motor at the tilt shaft is defined as motorA and the motor at the roll shaft is defined as motorB. The number of rotations per minute is set as constant using the function setSpeed and is set at 20 rotations per minute.

The PIN connections of the Arduino UNO is as shown in Figure 3.2, where the PIN 9-12 is assigned to motor A, PIN 4-7 to motor B, and PIN 2, A0, A1, GND and 5V is assigned to the joystick. Two types of mode of tracking are created for the AI sun-tracking system, which are (i) Manual Tracking and (ii) Auto Tracking. In Manual Tracking, the CPV is controlled using the joystick connected to the Arduino UNO. The Manual Tracking is used to pre-set the initial position of the CPV to orientate towards the Sun or manually adjust the position of the CPV.

In Auto Tracking, the CPV is controlled by the AI system, which consists of the YOLOv7 object detection algorithm and the Q-Learning reinforcement learning algorithm. The YOLOv7 will obtain the coordinates of the Sun and passed the coordinates to Q-Learning. This will allow Q-Learning to get the minimum steps required for the CPV to align with the coordinates of the Sun. The Arduino UNO will receive the minimum steps required and orientate the CPV to the correct position.

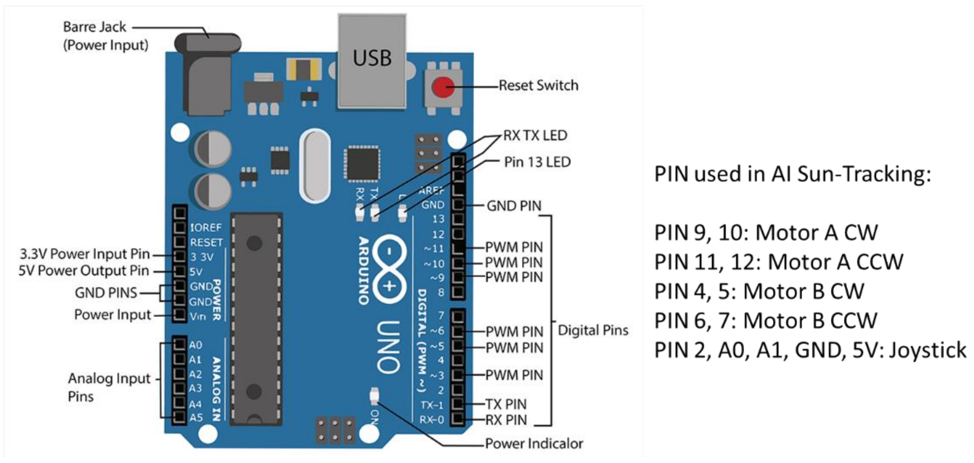


Figure 3.2: PIN connections for Arduino Uno in the AI sun-tracking system.

3.3 AI Sun-Tracking System

The sun-tracking system of the CPV consists of 2 cameras which are use to capture images for computer vision. The 2 cameras are located at the center of the NIDC and at the target holder facing the sky. Once the image frame is captured, YOLOv7 algorithm will obtain the coordinate of the Sun for sun-tracking. The sun coordinate is then processed using Q-learning to determine the minimum steps required to reach the sun coordinate from its current position. This chapter outlines the deep learning process for the AI sun-tracking system while Figure 3.3 shows the flowchart of the AI sun-tracking system.

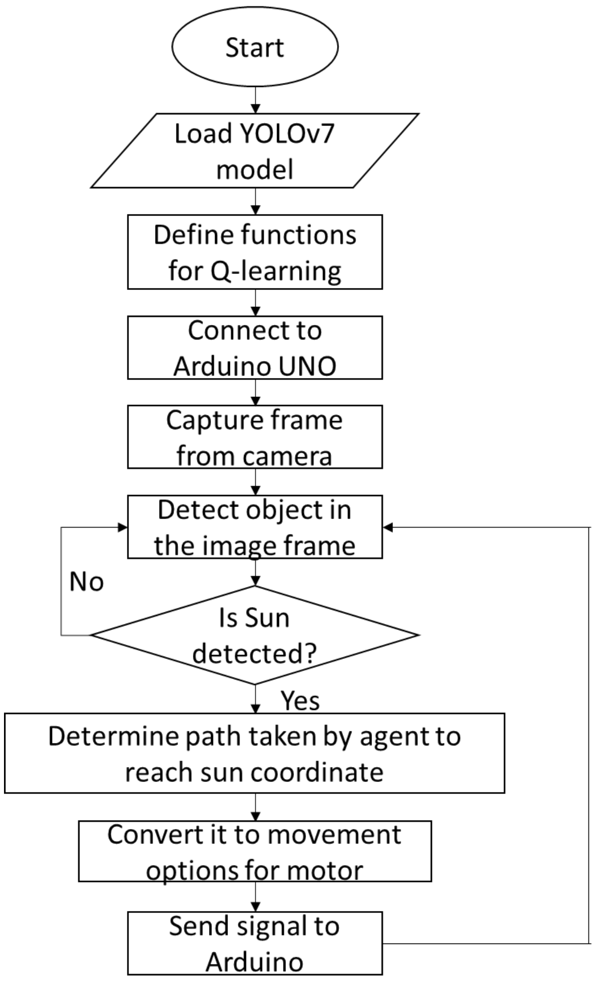

Start
Load YOLOv7 model
Define functions for Q-learning
Connect to Arduino UNO
Capture frame from camera
Detect object in the image frame
No
Is Sun detected?
Yes
Determine path taken by agent to reach sun coordinate
Convert it to movement options for motor
Send signal to Arduino

Figure 3.3: Flow chart of the AI sun-tracking algorithm.

3.3.1 Custom Dataset of Sun and Clouds

A custom dataset consisting of the class ‘sun’ and ‘clouds’ are created using the labelImg software. The labelImg is a graphical image annotation tool and is used to label classes of objects by drawing bounding boxes over the target class. A total of 300 images of sun and clouds are taken in various orientations and also at different location and time to construct a model that can resists noise and environmental changes. The dataset is divided with a 70:30 ratio, where 210 images are used for training and the remaining 90 images are used for validation. Figure 3.4 shows the examples of images of sun and clouds within the dataset.

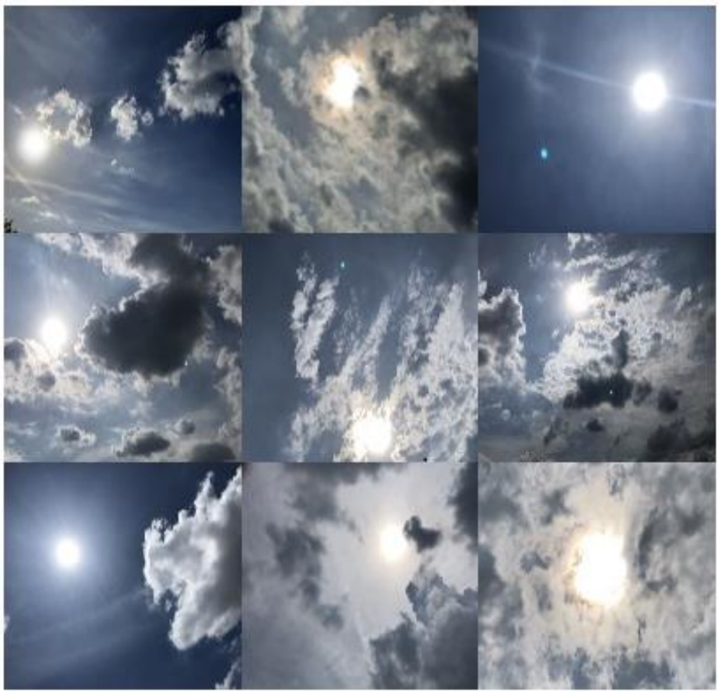


Figure 3.4: Nine samples of dataset which contains the class ‘sun’ and ‘clouds’.

The bounding box is drawn over the object for labelling according to their classes as shown in Figure 3.5. The dataset is then saved in a txt file in YOLO format in the same directory as defined for the images. A classes.txt file is saved to the same directory, and the file defines the list of class names defined for the YOLO labels. The predefined classes which are the ‘sun’ and ‘clouds’ for this custom dataset is added in data/predefined_classes.txt. An example of the txt file of a labelled image is shown in Figure 3.6 and contains information of the class labelled and also the coordinates of the four corners of a bounding box.

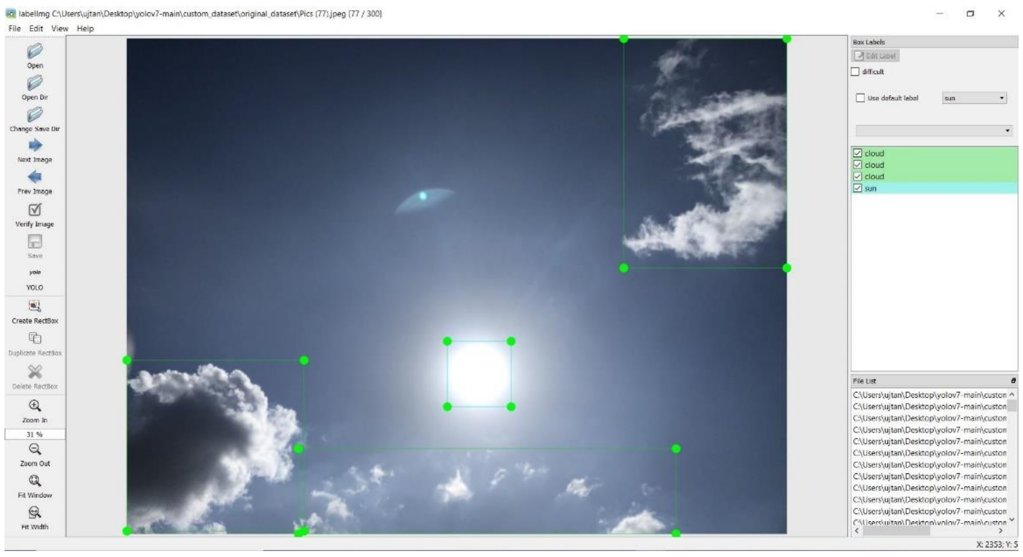
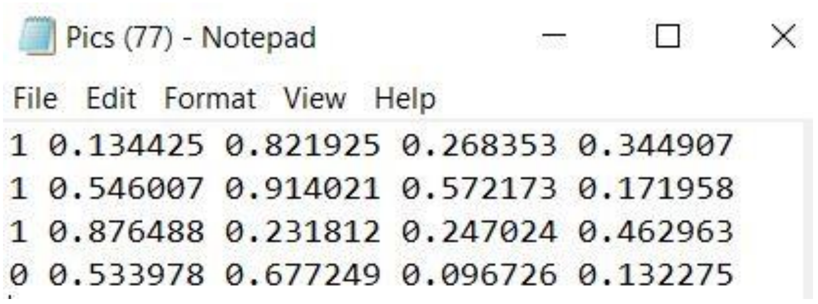


Figure 3.5: Preparation of dataset in labelImg by drawing bounding boxes over the target classes ‘sun’ and ‘clouds.



```
1 0.134425 0.821925 0.268353 0.344907
1 0.546007 0.914021 0.572173 0.171958
1 0.876488 0.231812 0.247024 0.462963
0 0.533978 0.677249 0.096726 0.132275
```

Figure 3.6: Text file containing the coordinates of the bounding box of the target class. ‘0’ represents the class ‘sun’ and ‘1’ represents the class ‘cloud’.

3.3.2 Model Training for Object Detection

The YOLOv7 model is trained in Google Colab with GPU accelerator, which provides training speed that is 10 times faster for each step compared to a model trained using CPU only. Once the dataset is prepared as discussed in the previous subsection, the path of the txt file of each labelled image is written into a new txt file which contains the path of all the txt file of the labelled image.

Transfer learning is the most efficient way to be implemented to train the AI model. It can improve the performance of the neural network as it transfers the knowledge gained from previous trainings to its current task. To start the model training, the command as shown in Code Listing 3.1 is run in

Google Colab. The yolov7_training.pt weights file is used for transfer learning as it can reduce the time and size of dataset of the training process. To get a good AI model for object detection, the training process is run for 100 epochs with different batch sizes. The batch size that gives us the highest mean average precision (mAP) and lowest loss function will be chosen for the object detection model.

```
!python train.py --workers 8 --device 0 --batch-size 4 --data data/custom.yaml --
img 640 640 --cfg cfg/training/yolov7_custom.yaml --weights 'yolov7_training.pt'
--hyp data/hyp.scratch.custom.yaml --epochs 100
```

Code Listing 3.1: Command to run training in Google Colab.

3.3.3 Model Training for Reinforcement Learning

Q-learning is chosen as the reinforcement learning for motor control in the proposed AI sun-tracking system. The code for Q-learning is written in Jupyter Notebook for both training process and motor control process. Before the Q-learning starts the training process, some function class and dependencies are defined. The class 'MOVINGPOINT', 'MOVINGTARGET' and 'CENTERTARGET' is defined for the environment and the agent. The class 'MOVINGPOINT' is defined for the function to extract the sun coordinate from the YOLOv7 algorithm. The function 'move' is also assigned to the class 'MOVINGPOINT' to send signal to Arduino to move the motors clockwise or anticlockwise. The class 'CENTERTARGET' is defined to provide the midpoint of the camera for the agent to move from the midpoint to the sun coordinate. The class 'MOVINGTARGET' is defined to be a randomised point, which defined the path taken for the agent to move from the CENTERTARGET to the MOVINGPOINT.

Before the training for Q-learning, the training parameters such as the episodes, rewards, maximum steps, learning rate and discount is set. The pixel size of the Q-learning process is set to match the pixel size frame of the webcam, where the XSIZE and YSIZE are 640 and 480 respectively. The number of episodes for the training set at 500000 and the maximum step per episode is 460 which is the distance between the corners of the screen size and the midpoint.

The MOVE_REWARD and MOVE_PENALTY is set as 10 and 5 respectively, and the ON_TARGET_REWARD is set as 1000. The learning rate is 0.1 and discount is 0.95.

In the function ‘motormove’, the agent is trained to move from the centre point to the target point through a series of rewards. A Q-table is used to determine the behaviour of the agent, where the agent will be rewarded if the new distance is smaller than the old distance such that it will always move closer to the target point. If the agent reaches the target point, a larger reward called ‘ON_TARGET_REWARD’ is assigned to the agent to encourage the agent to stay at the target point and discouraged from leaving. The new distance is constantly added into the Q-table, and if the new Q-value is not the ON_TARGET_REWARD, the new Q-value is calculated by:

$$new_q = (1 - LEARNING_{RATE}) * current_q + LEARNING_{RATE} * (reward + DISCOUNT * \max \_future\_q) \tag{3.1}$$

The loop will only break when the reward is equal to the ON_TARGET_REWARD.

The training process of Q-learning starts by creating a new Q table to save the Q values of the training process for the motor control process. An empty array is used to save the rewards for each episode, where the initial reward given to the agent is zero. The epsilon for each episode is given by

$$epsilon_{future} = epsilon_{current} * decay_{episode} \tag{3.2}$$

where the future episode is the product of the current epsilon and the episode decay. To get a better model, the greedy epsilon approach is chosen where the behaviour of the agent is determined by a random generated number. If the generated number is larger than the epsilon, the agent will use the Q values to determine its next move. However, if the generated number is lesser than the epsilon, the next move of the agent is determined randomly. The whole training process is shown visually as shown in Figure 3.7 where the blue point is the CENTERTARGET and the green point is the MOVINGTARGET.

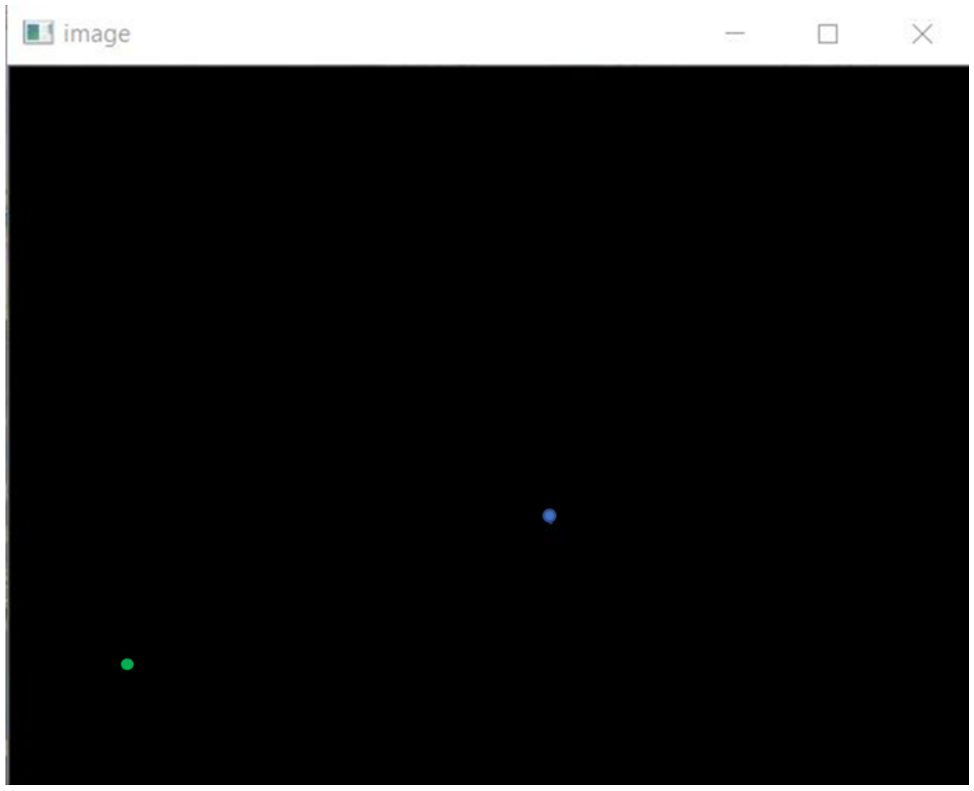


Figure 3.7: Visualisation of the MOVINGTARGET to move from the CENTERTARGET to the MOVINGTARGET. The blue point is the CENTERTARGET while the green point is the MOVINGTARGET.

3.3.4 AI Sun-Tracking Algorithm

There are two machine learning algorithm that is used in the AI sun-tracking system which are YOLOv7 and Q-learning. The two algorithms are not connected to each other and can only perform their own tasks separately. Therefore, a python script is required to integrate both of the machine learning algorithm and form a complete AI sun-tracking algorithm. This AI sun-tracking algorithm can provide real-time sun tracking and will constantly adjust the position of the CPV and always face it towards the sun. The AI sun-tracking algorithm is connected to the Arduino UNO by using the serial library, the port and baudrate is set as ‘COM3’ and ‘115200’ respectively which is the same in the Arduino code. The baudrate is set at 115200 which transfers at a speed of 11520 bytes/s to allow for faster connection between laptop and Arduino.

The algorithm will first load the class functions and dependencies for both YOLOv7 and Q-learning as explained in the previous subsections. In order to create a connection between Jupyter Notebook and Arduino, functions 'write_read' and 'sendsignal' is created to send the signal created by the AI sun-tracking algorithm to the Arduino for motor control. This signal is defined as 0, 1, 2 and 3 which are the movement options for the CPV system, which translates to the clockwise and anticlockwise movement of the tilt and roll motors.

The detect.py file of the official YOLOv7 is modified to include both object detection and Q-learning. The function 'plot_box_point' is created to plot the midpoint of the bounding box. This midpoint is useful as it extracts the coordinate of the centre point of the target, which is the sun in our case. Once the classes and dependencies are set up, a pipeline for the object detection of sun and clouds is created. The video frames are capture and is processed by the object detection to obtain the sun coordinate. Once the sun coordinate is obtained, the loop for the Q-learning will start to determine the minimum steps taken to reach the target point. The Arduino function will then send the signal which are the movement of the CPV from Jupyter Notebook to Arduino.

The Python script is saved to a file called detect.py and the command in Code Listing 3.2 is run in Jupyter Notebook for real-time object detection and automatic motor control.

```
!python finaldetect.py --weights best.pt --img 640 --conf 0.30 --source 0 --classes 0 --
no-trace
```

Code Listing 3.2: Command to run AI sun-tracking algorithm in Jupyter Notebook.

3.4 Summary

The whole process of the AI sun-tracking system is outline in this chapter, which consisted of the object detection and reinforcement learning algorithm.

YOLOv7 is chosen as the object detection algorithm while Q-learning is chosen for motor control in this proposed AI sun-tracking system. The process for the deep learning model training is introduce for both YOLOv7 and Q-learning. A new python script is created for the AI sun-tracking algorithm is created to integrate both the YOLOv7 and Q-learning algorithms.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, the results of the deep learning training for the sun-tracking system will be discussed, which includes the deep learning model training for YOLOv7 and Q-learning. The model performance of the integrated AI sun-tracking system is discussed in detail. The model will be tested for different situations such as during a clear and sunny days, during sunsets and also during cloudy days.

4.2 Performance of the Model Training

The deep learning models used in this AI sun-tracking system are YOLOv7 for object detection and Q-learning for the motor control. As discussed in Methodology section, the training process for YOLOv7 is conducted in Google Colab for utilising its free GPU option to reduce training time. The training process for Q-learning is conducted in Jupyter Notebook, while the visualisation of the agent moving from the CENTERTARGET to the MOVINGPOINT is shown using the OpenCV module. The training of both of the deep learning models will be discussed in details, and compared it to different situations such as different batch sizes.

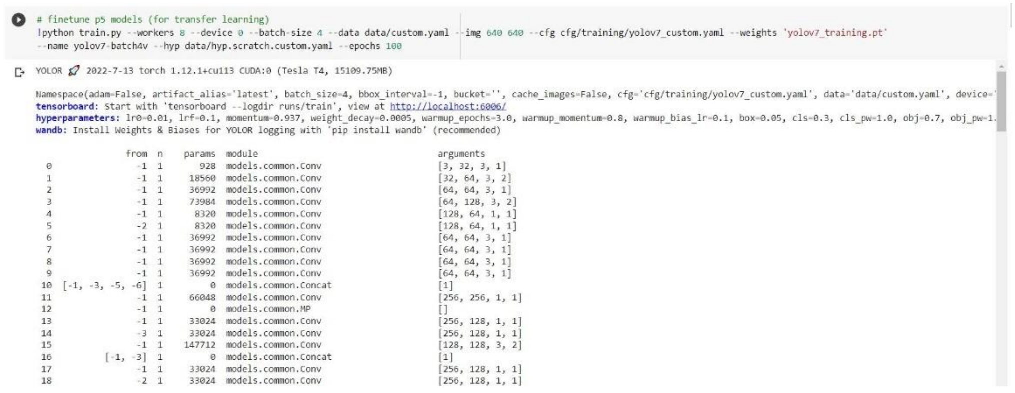
4.2.1 YOLOv7

In machine learning, one of the difficulties is to create a generalized model that is good in predicting new data that are not available in the training process. However, through performance visualization, it can help to create a better generalized model by visualizing the learning curve of the deep learning model. The learning curve is visualised by plotting the learning performance against its experience. The model evaluated the training and validation dataset after each update and shows the learning curves.

The performance of a deep learning model can be evaluated by observing the model loss plot where the loss is plotted against number of epochs.

However, a model can have poor performance due to overfitting and underfitting, where overfitting refers to a model that is trained too well and underfitting refers to a model that can neither train data or generalize new data. Therefore, the model loss plot is important to evaluate the model and changes can be made to hyperparameters such as learning rate and number of layers in a network to improve the deep learning performance.

Figure 4.1 shows the transfer learning progress for YOLOv7 in Google Colab by utilising its GPU option to speed up the training process. The YOLOv7 model is trained with a batch size of 4 and is trained for 100 epochs. After the completion of the model training, the mAP for each class is given as shown in Figure 4.2. The model training is repeated with different batch sizes and is trained for 100 epochs.



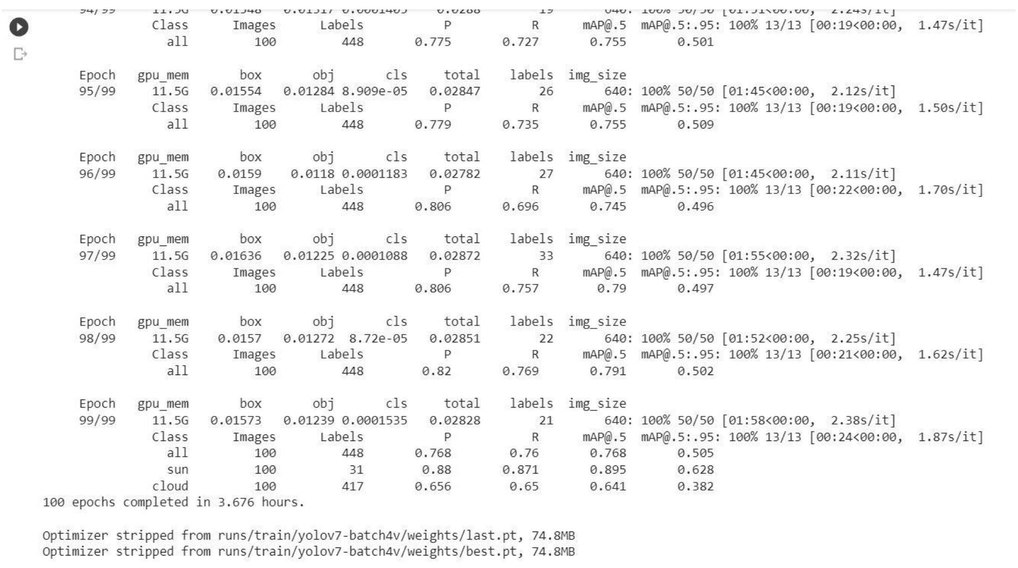
```
# finetune p5 models (for transfer learning)
!python train.py --workers 8 --device 0 --batch-size 4 --data data/custom.yaml --img 640 640 --cfg cfg/training/yolov7_custom.yaml --weights 'yolov7_training.pt'
--name yolov7-batch4v --hyp data/hyp.scratch.custom.yaml --epochs 100
```

```
YOLOR 🚀 2022-7-13 torch 1.12.1+cu113 CUDA:0 (Tesla T4, 15109.75MB)

Namespace(adam=False, artifact_alias='latest', batch_size=4, bbox_interval=-1, bucket='', cache_images=False, cfg='cfg/training/yolov7_custom.yaml', data='data/custom.yaml', device='
tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.3, cls_pw=1.0, obj=0.7, obj_pw=1.
wandb: Install Weights & Biases for YOLOR logging with 'pip install wandb' (recommended)

                 from  n    params  module                                  arguments
  0                -1  1       928  models.common.Conv                      [3, 32, 3, 1]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1      8320  models.common.Conv                      [128, 64, 1, 1]
  5                -2  1      8320  models.common.Conv                      [128, 64, 1, 1]
  6                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  7                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  8                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  9                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
 10  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
 11                -1  1     66048  models.common.Conv                      [256, 256, 1, 1]
 12                -1  1         0  models.common.MP                        []
 13                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 14                -3  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 16          [-1, -3]  1         0  models.common.Concat                    [1]
 17                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 18                -2  1     33024  models.common.Conv                      [256, 128, 1, 1]
```

Figure 4.1: Transfer learning for YOLOv7 model on Google Colab with batch size of 4 and 100 epochs.



```
  94/99     11.5G   0.01548   0.01317 0.0001405    0.0288        19       640: 100% 50/50 [01:51<00:00,  2.24s/it]
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:19<00:00,  1.47s/it]
                all         100         448       0.775       0.727       0.755       0.501

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      95/99     11.5G   0.01554   0.01284 8.909e-05   0.02847        26       640: 100% 50/50 [01:45<00:00,  2.12s/it]
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:19<00:00,  1.50s/it]
                all         100         448       0.779       0.735       0.755       0.509

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      96/99     11.5G    0.0159    0.0118 0.0001183   0.02782        27       640: 100% 50/50 [01:45<00:00,  2.11s/it]
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:22<00:00,  1.70s/it]
                all         100         448       0.806       0.696       0.745       0.496

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      97/99     11.5G   0.01636   0.01225 0.0001088   0.02872        33       640: 100% 50/50 [01:55<00:00,  2.32s/it]
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:19<00:00,  1.47s/it]
                all         100         448       0.806       0.757        0.79       0.497

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      98/99     11.5G    0.0157   0.01272  8.72e-05   0.02851        22       640: 100% 50/50 [01:52<00:00,  2.25s/it]
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:21<00:00,  1.62s/it]
                all         100         448        0.82       0.769       0.791       0.502

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      99/99     11.5G   0.01573   0.01239 0.0001535   0.02828        21       640: 100% 50/50 [01:58<00:00,  2.38s/it]
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:24<00:00,  1.87s/it]
                all         100         448       0.768        0.76       0.768       0.505
                sun         100          31        0.88       0.871       0.895       0.628
              cloud         100         417       0.656        0.65       0.641       0.382
100 epochs completed in 3.676 hours.

Optimizer stripped from runs/train/yolov7-batch4v/weights/last.pt, 74.8MB
Optimizer stripped from runs/train/yolov7-batch4v/weights/best.pt, 74.8MB
```

Figure 4.2: Completion of the training for YOLOv7 model on Google Colab with batch size of 4 and 100 epochs.

One of the parameters that can affect the performance of a convolutional neural network is the batch size of the deep learning model. Therefore, the mAP of the transfer learning for YOLOv7 is plotted against the epochs for different batch sizes as shown in Figure 4.3. Generally, it is shown that the accuracy of a model increases with the number of batch size (Radiuk, 2017). However, in the model training for the custom dataset of sun and clouds, the mAP is not in an increasing trend, where the lowest batch size which is 4 shows the highest mAP. This situation may be due to the low number of sample size, as the largest batch size the laptop used can handle is batch size of 16 and the training process is very long compared to other models. The higher mAP for batch size of 4 may also be due to the fact that the mAP for batch size 4 is at a local maximum, thus more batch size is required to obtain an increasing trend. Thus, the increasing trend may be visible when a large amount of batch size is present. It is also shown that the effect of batch size on the accuracy of the model may varied with the dataset used (Masters & Luschi, 2018). Table 4.1 also shows the mAP for all classes and the time taken to complete the training.

Table 4.1: Comparison of YOLOv7 model on custom training dataset with different batch size.

Batch Size	mAP	Average time per epoch/ s	Total time used to train data/ h
4	0.768	132.48	3.68
8	0.757	130.93	3.63
16	0.764	136.94	3.80

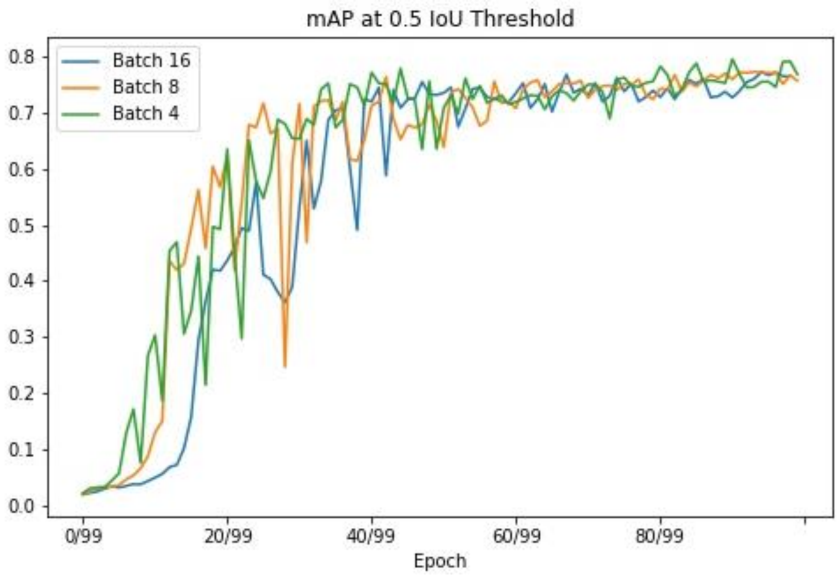


Figure 4.3: Mean average precision (mAP) at 50% Intersection of Union (IoU) threshold for different batch sizes of 4, 8 and 16. Batch size of 4 shows the best mAP score.

Another performance metrics that can evaluate the performance of a neural network is the loss curve graph. This graph gives us the details of the training process and the direction in which the network learns. As shown in Figure 4.4, the loss curve can show the learning rate of the model training, where the red curve shows a good learning rate. A good learning rate needs to have a steep or short learning curve, where the green curve shows the steepest curve indicating a high learning rate. The red curve is comparable to the loss curve as shown in Figure 4.5 which is the validation class plot of the training process for the custom dataset. This shows that the model training has a good learning curve, and indicates that a large amount of learning had taken place in a short period of time. Loss curves that have very high learning rate or learning rate are not desirable as the deep learning model might face issues in oversampling and undersampling. Therefore, batch size of 4 is chosen for the model as it has the highest mAP and lowest loss.

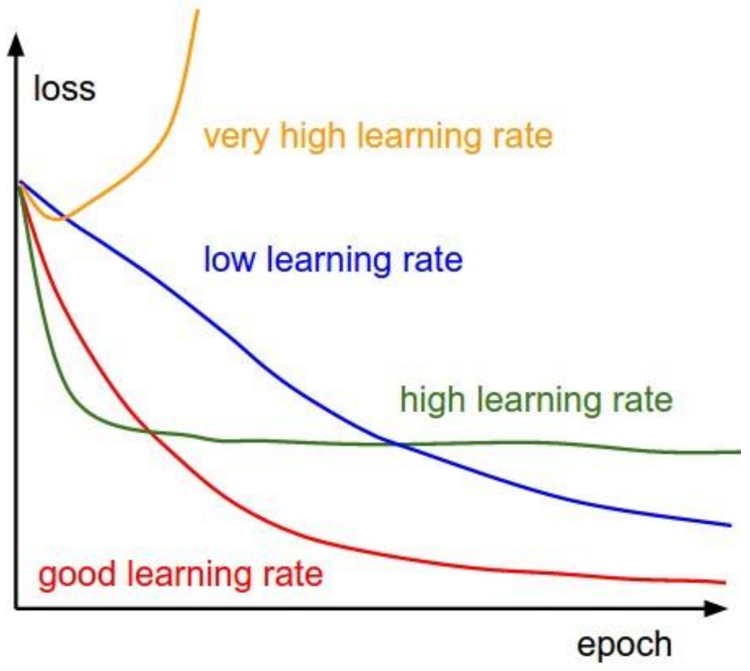


Figure 4.4: The effect of the learning rate on the loss of the deep learning model.

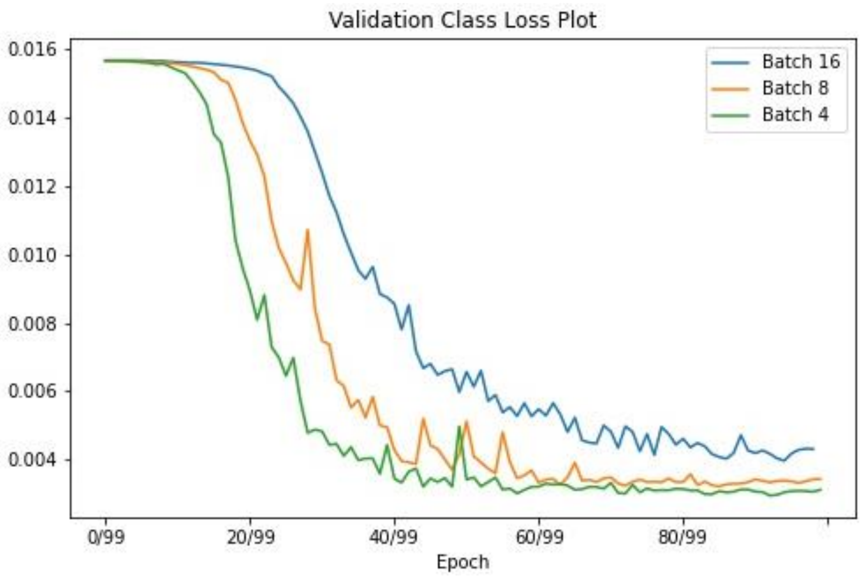
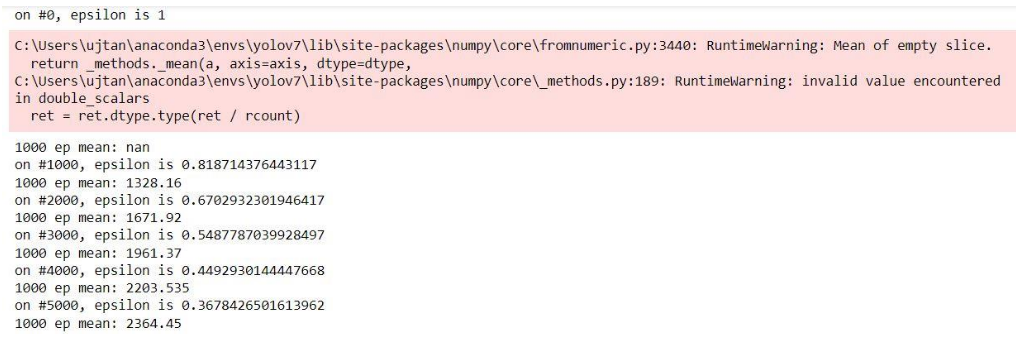


Figure 4.5: The validation class loss for different batch sizes of 4, 8 and 16. Batch size of 4 shows the lowest validation class loss.

4.2.2 Q-Learning

The Q-learning model training is run on Jupyter Notebook and the training process of the Q-learning is shown in Figure 4.6. Since the Q-learning has no previous Q-table, a new Q-table have to be created to store the Q-values from the training process. Therefore, the initial epsilon is set as 1 to train the Q-learning model, where the future epsilon is the product of the current epsilon and epsilon decay as shown in Equation 3.2. In Figure 4.6, the mean episode rewards for every 1000 episodes are shown, where it is initially 1328 and gradually increase. This shows that during the initial training process, the agent has more penalty than rewards, which means that the agent is not able to reach its goal successfully. As there are insufficient Q-values, the agent tends to choose random values for the movement of the agent rather than using the values from the Q-table. However, as more Q-values are collected, the agent tends to use the Q-values, and therefore the rewards gained is increasing as the agent is able to reach its goal more effectively.



```
on #0, epsilon is 1

C:\Users\ujtan\anaconda3\envs\yolov7\lib\site-packages\numpy\core\fromnumeric.py:3440: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
C:\Users\ujtan\anaconda3\envs\yolov7\lib\site-packages\numpy\core\_methods.py:189: RuntimeWarning: invalid value encountered
in double_scalars
  ret = ret.dtype.type(ret / rcount)

1000 ep mean: nan
on #1000, epsilon is 0.818714376443117
1000 ep mean: 1328.16
on #2000, epsilon is 0.6702932301946417
1000 ep mean: 1671.92
on #3000, epsilon is 0.5487787039928497
1000 ep mean: 1961.37
on #4000, epsilon is 0.4492930144447668
1000 ep mean: 2203.535
on #5000, epsilon is 0.3678426501613962
1000 ep mean: 2364.45
```

Figure 4.6: Training process of the Q-learning in Jupyter Notebook. The epsilon is shown for every 1000 episodes, and the episode mean is shown.

The episode rewards obtained for every 1000 episodes is plotted as shown in Figure 4.7. The model is trained for 500000 episodes and the rewards obtained by the agent increases initially and fluctuates between 3000 and 3250. This shows that the agent will initially tend to use random values to achieve the target, and store values in the Q-table. Once enough episodes are reached, the agent tends to use the values in the Q-table, and the rewards achieved start to fluctuate between 3000 and 3250.

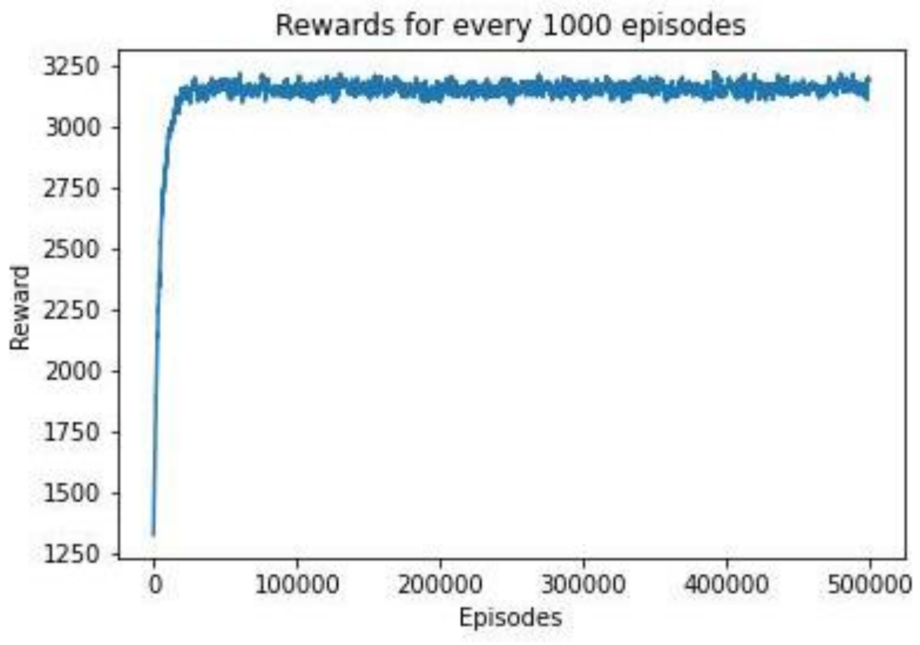


Figure 4.7: Rewards obtained by the agent for every 1000 episodes for a training of 500000 episodes.

4.3 Model Performance

Once the model training of the AI sun-tracking algorithm is completed, the algorithm is evaluated based on several conditions. The effect of the confidence value on the detection of the AI sun-tracking algorithm will be evaluated. The algorithm is also tested on different weather conditions such as sunny, cloudy and sunset. The algorithm is considered successful if it is able to detect the Sun with high accuracy when the image is not available in the dataset used.

4.3.1 Confidence Value for AI Sun-Tracking Algorithm

It is shown in Figure 4.8 that the confidence level will affect the object detection algorithm where the Sun is detected in one but not in another. The confidence value reflects the probability of the bounding box containing the object of interest and how confident the classifier detects the object. The confidence value is zero when no object exists inside the bounding box. Therefore, the confidence level that is set will determine if the object is detected or not, and will only shows the detected object if the confidence value of the detected object is higher than the confidence value that is set for the detection algorithm.

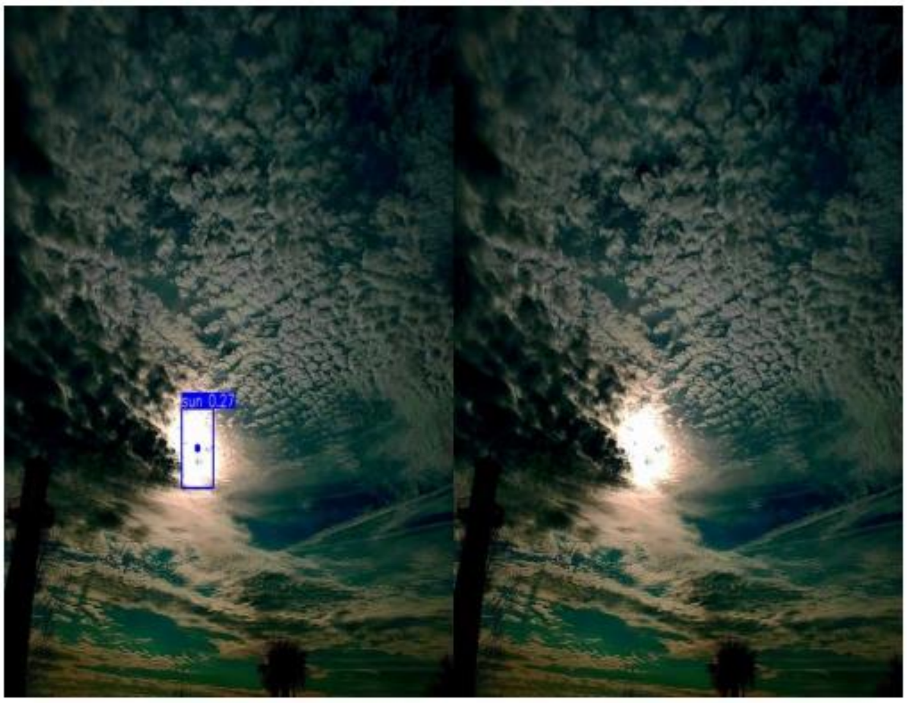


Figure 4.8: Comparison between two images that is partially blocked by the cloud with different confidence level. The left image is at confidence level of 0.10 while the right image is at confidence level of 0.30.

However, there are cases when if the confidence value is set too low, wrong objects will be detected as shown in Figure 4.9. When the confidence value is set at 0.10, objects that are not the actual targeted object is detected. This might be due to the insufficient amount of training images, which may cause the algorithm to detect the wrong object due to it having similar shape, colour or brightness. However, this issue is settled when the confidence value is set at 0.30, where only the actual 'Sun' is detected. Therefore, the confidence value for the AI sun-tracking system will be set at 0.30.

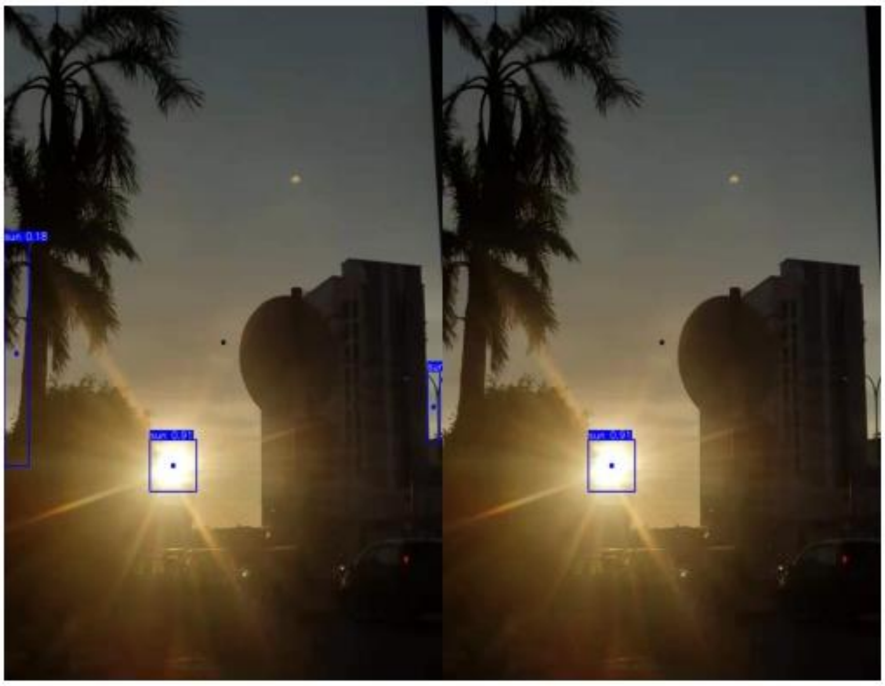


Figure 4.9: Example of the wrongly detected object, where the left image wrongly detected the object while the right image only shows the correct detected object. The left image is at confidence level of 0.10 while the right image is at confidence level of 0.30.

4.3.2 Evaluating the Model at Different Conditions

To evaluate the AI sun-tracking system for it to be able to detect the sun at all times, the AI algorithm is tested on a few images at different situations which are during sunny day, sunset or when the sun is partially blocked by the clouds as shown in Figure 4.10. It is shown that the sun is detected with high confidence level and is over 90% for each case. This means that the training of the object detection is successful and is able to detect objects in images that are not available in the dataset. The custom dataset for this project does not include too many images of the object at different conditions due to the difficulty in obtaining or capturing the image at that particular situation. However, a large amount of sample is needed to test the overall performance of the sun-tracking algorithm at different conditions. Once the sun is detected, the midpoint of the bounding box is obtained and is used for the Q-learning process for motor control.

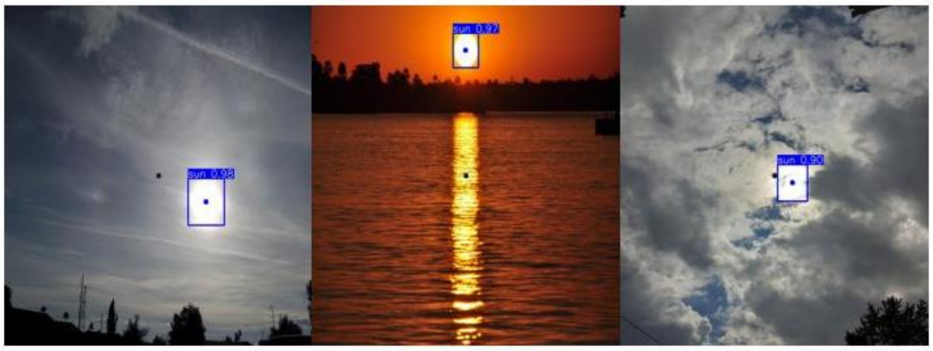


Figure 4.10: Detection of the Sun at different situations. From left to right: (Left) Detection of Sun during sunny day, (Middle) Detection of Sun during sunset, (Right) Detection of Sun when partially blocked. The value beside the class ‘Sun’ is the confidence level of the detection algorithm, which is over 90% for each case.

The agent in the Q-learning has 4 movement options which are left, right, up and down. Once the midpoint of the sun is obtained, the agent will determine the path and minimum steps taken for the centre to reach the midpoint of the Sun which is shown in Figure 4.11. The movement of the agent will determine the signal sent to Arduino, which will control the clockwise and anticlockwise movement of the tilt and roll motors. With Q-learning, the same AI sun-tracking algorithm can be used in any type of sun-tracking system no matter if it’s a heliostat, flat PV or CPV system as the Q-learning can automatically adjust the position of the sun-tracking system to always have the sun coordinate at the centre point of the webcam. Encoders are also no longer required as the Q-learning can automatically and constantly feedback the errors and adjust the sun-tracker accordingly.

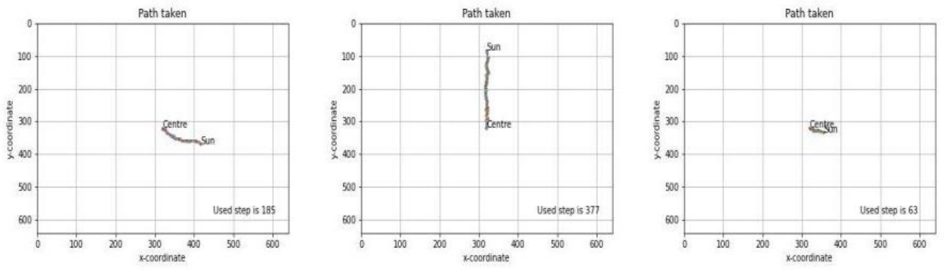


Figure 4.11: The path taken by the centre (centre point of camera) to reach the midpoint of the Sun and its corresponding steps taken. The path taken and minimum steps is determined for each sun detection as shown in Figure 4 from left to right.

4.4 Summary

The results for the training of the AI sun-tracking algorithm which consists of YOLOv7 and Q-learning are presented in this chapter. An in-depth analysis had been performed to evaluate the AI sun-tracking algorithm at different conditions. The AI sun-tracking system shows good performance even at different conditions such as when the sun is partially blocked. The AI sun-tracking algorithm can be used in any type of sun-tracking system and also eliminates the need of encoders for error feedback for motor control. Due to the absence of a physical setup of sun-tracking system, the model is only evaluated based on the performance of the deep learning model and various images at different conditions.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The AI sun-tracking system integrated object detection algorithm and reinforcement learning to detect the Sun and adjust the position of the sun-tracking system. The YOLOv7 is chosen for object detection while Q-learning is chosen for reinforcement learning to control the motor. A custom dataset for the object detection is prepared which consists of a total of 300 images of sun and clouds. The deep learning training model for YOLOv7 is trained for 100 epochs with different batch sizes, where the batch size of 4 is chosen due to its high mAP and low loss. After the training process for YOLOv7 in Google Colab, the Q-learning training process is trained in Jupyter Notebook for 500000 episodes.

Once the training process for the AI sun-tracking algorithm is complete, the deep learning is evaluated under different conditions. It is shown that the confidence level will affect the object that is detected, where the detected object must have a confidence level higher than the confidence level that is set for the sun-tracking algorithm. The algorithm is also tested under different conditions such as during sunny day, sunset and also when the sun is partially blocked. It shows that the sun is detected under each condition and has a confidence level of over 90%. The Q-learning provides the minimum steps and the movement options of the agent to move from the centre point to the sun coordinate. It shows that the agent is successful in reaching its goal which is the coordinate of the Sun.

Through the evaluation of the AI sun-tracking algorithm, it shows that the AI integrated sun-tracking algorithm is suitable for any type of sun-tracking systems as the AI algorithm will learn by itself. Thus, it is no longer needed to create different formulas for each type of sun-tracking system. Furthermore, the external effects such as wind load does not affect the AI algorithm as it will provide counter feedback to the Q-learning which will auto-correct the position of the CPV system.

5.2 Recommendations for Future Work

This project can be further improved to ensure the scenario where two suns will never appear, and only one sun coordinate is obtained. This can be done by first comparing the coordinate of the sun when only one sun is being detected with the coordinate of the suns when two sun is detected. The coordinate of the sun that is closest to the sun at previous time is chosen, where the coordinate of the other sun is ignored. This is due to the fact that the sun will not move too far apart within short time duration, therefore it is able to discard the other coordinate of the sun that is wrongly detected.

Once there is enough of data collected during the real-time operation of the AI sun-tracking system, a pattern of the movement of the sun throughout the day can be form. This allows us to predict the movement of the sun using another machine learning algorithm which can be the future work done for this study.

REFERENCES

Carballo, J. A., Bonilla, J., Berengual, M., Fernandez-Reche, J. and Garcia, G., 2018. New approach for solar tracking systems based on computer vision, low cost hardware and deep learning. Renewable Energy, 133, pp. 1158-1166.

Chong, K. K., Siaw, F. L., Wong, C. W. and Wong, G. S., 2009. Design and construction of non-imaging planar concentrator for concentrator photovoltaic system. Renewable Energy, 34, pp. 1364-1370.

Chong, K. K., Yew, T. K., Wong, C. W., Tan, M. H., Tan, W. C. and Lim, B. H., 2017. Dense-array concentrator photovoltaic prototype using non-imaging dish concentrator and an array of cross compound parabolic concentrators. Applied Energy, 204, pp. 898-911.

Kim, K. H. and Cho, S. B., 2019. An efficient concentrative photovoltaic solar system with Bayesian selection of optimal solar tracking algorithms. Applied Soft Computing Journal, 83, 105618.

Masters, D. and Luschi, C., 2018. Revisiting small batch training for deep neural networks. *arXiv preprint*, arXiv: 1804.07612v1.

Masterson, V., 2021. Renewables were the world's cheapest source of energy in 2020, new report shows. World Economic Forum, [online] 5 July. Available at: <https://www.weforum.org/agenda/2021/07/renewables-cheapest-energy-source/> [Accessed: 13 April 2022].

Nadia, A. R., Isa, N. A. M. and Desa, M. K. M., 2020. Efficient single and dual axis solar tracking system controllers based on adaptive neural fuzzy inference system. Journal of King Saud University-Engineering Sciences, 32(7), pp. 459-469.

Radiuk, P. M., 2017. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. Information Technology and Management Science, 20, pp. 20-24.

Tan, M. H., Chong, K. K. and Wong, C. W., 2014. Optical characterisation of non-imaging dish concentrator for the application of dense-array concentrator photovoltaic system. Applied Optics, 53(3), pp. 475-486.

Wang, C. Y., Bochkovskiy, A. and Liao, H. Y. M., 2022. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint*, arXiv: 2207.02696v1.

Watkins, C., 1989. Learning from delayed rewards. *PhD Thesis*, University of Cambridge, Cambridge.

APPENDICES

Appendix A: Final Artificial Intelligent (AI) Sun-Tracking Algorithm

```
import argparse
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow,
non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized,
TracedModel

#Extra dependencies required for reinforcement learning
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import pickle
from matplotlib import style
import time
import serial #add Serial librart for Serial communication
style.use("ggplot")

#to connect Arduino
arduino = serial.Serial(port='COM3', baudrate=115200, timeout=.1)#Create Serial port
object called arduinoSerialData
```

```
#define parameter
XSIZE = 640
YSIZE = 480
threshold_distance = 50# define threshold distance in pixel for the motor to run
MOVE_PENALTY = 20
MOVE_REWARD= 1  #Can be 0
ON_TARGET_REWARD = 1000
epsilon = 0 #0 for retraining and run the model
EPS_DECAY = 0.9998  # Every episode will be epsilon*EPS_DECAY

start_q_table = "qtable-1661796052.pickle" # None or Filename

LEARNING_RATE = 0.1
DISCOUNT = 0.95

def write_read(x):
    arduino.write(bytes(x, 'utf-8'))
    time.sleep(0.05)
    data = arduino.readline()
    return data

#send signal to arduino
def sendsignal(desiredmove):
    byte_command = write_read(desiredmove)
    arduino.writelines(byte_command) #send a byte
    time.sleep(0.05) # wait 0.5 seconds

#functions required
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.")

# classes required
# for environment and agent
class MOVINGPOINT:
    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:
            #self.move(x=1, y=1)
            self.move(x=1, y=0)

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

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

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

    def move(self, x=False, y=False):
        if not x:
            desiredmove = str(np.random.randint(3,6))#for 5, no horizontal movement)
            sendsignal(desiredmove)
        elif x == 1:
            desiredmove = '3' #frame move left, target move right
            sendsignal(desiredmove)
        elif x == -1:
            desiredmove = '4' #frame move right, target move left
            sendsignal(desiredmove)
        if not y:
            desiredmove = str(np.random.randint(0,3))#for 0, no vertical movement)
            sendsignal(desiredmove)
        elif y == 1:
            desiredmove = '2' #frame move down, target move up
            sendsignal(desiredmove)
        elif y == -1:
            desiredmove = '1' #frame move up, target move down
            sendsignal(desiredmove)

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 = XSIZE//2
        self.y = 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

#for loading or creating of Q-table
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 motormove(moving_target,moving_point):
    test_obs = moving_point - moving_target
    test_distance = np.sqrt(test_obs[0]**2 + test_obs[1]**2)
    if test_distance >= threshold_distance: #to avoid the movement of motor if it is
within the threshold distance from the target
        obs = moving_point - moving_target
        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, 4)

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



def plot_box_point(x, img, color=None, label=None, line_thickness=None):
    # Plots one bounding box on image img
    tl = line_thickness or round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1  # line
thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    midx = int ((c1[0]+c2[0])/2)
    midy = int ((c1[1]+c2[1])/2)
    Resolution=(img.shape[1],img.shape[0])
    centre=(int(Resolution[0]/2),int(Resolution[1]/2)) #define centre coodinate of the
picture
    cv2.circle(img, (centre[0],centre[1]), radius=3, color=(255,0,255), thickness = -1)
#to plot centre point on the image
    cv2.circle(img, (midx,midy), radius=3, color=(0,0,110), thickness = -1)
    cv2.rectangle(img, c1, c2, color, thickness=tl)
    if label:
        tf = max(tl - 1, 1)  # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(img, c1, c2, color, -1)  # filled
        cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 255, 255], thickness=tf,
lineType=cv2.LINE_AA)


def detect(save_img=False):
    source, weights, view_img, save_txt, imgsz, trace = opt.source, opt.weights,
opt.view_img, opt.save_txt, opt.img_size, not opt.no_trace
    save_img = not opt.nosave and not source.endswith('.txt')  # save inference images
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    # Directories
    save_dir = Path(increment_path(Path(opt.project) / opt.name,
exist_ok=opt.exist_ok))  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  #
make dir

    # Initialize
    set_logging()
    device = select_device(opt.device)
    half = device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model
    stride = int(model.stride.max())  # model stride
    imgsz = check_img_size(imgsz, s=stride)  # check img_size

    if trace:
        model = TracedModel(model, device, opt.img_size)

    if half:
        model.half()  # to FP16
    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt',
map_location=device)['model']).to(device).eval()

    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride)

    # Get names and colors
    names = model.module.names if hasattr(model, 'module') else model.names
    colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]

    # Run inference
    if device.type != 'cpu':
        model(torch.zeros(1,                3,                imgsz,
imgsz).to(device).type_as(next(model.parameters())))  # run once
    t0 = time.time()
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)

        # Inference
        t1 = time_synchronized()
        pred = model(img, augment=opt.augment)[0]
        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres,
classes=opt.classes, agnostic=opt.agnostic_nms)
        t2 = time_synchronized()

        # Apply Classifier
        if classify:
            pred = apply_classifier(pred, modelc, img, im0s)

        # Process detections
        for i, det in enumerate(pred):  # detections per image
            if webcam:  # batch_size >= 1
                p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
            else:
                p, s, im0, frame = path, ", im0s, getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # img.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + (" if dataset.mode == 'image' else
f'_{frame}')  # img.txt
            s += '%gx%g ' % img.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-
1).tolist()  # normalized xywh
```

```
                        line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label
format
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')

                    if save_img or view_img:  # Add bbox to image
                        label = f'{names[int(cls)]} {conf:.2f}'
                        plot_box_point(xyxy, im0, label=label, color=colors[int(cls)],
line_thickness=3)
                        if names[int(cls)]=='sun':
                            c1, c2 = (int(xyxy[0]), int(xyxy[1])), (int(xyxy[2]), int(xyxy[3]))
                            Xcoordinate = int ((c1[0]+c2[0])/2)
                            Ycoordinate = int ((c1[1]+c2[1])/2)
                            centertarget = CENTERTARGET()
                            moving_point = MOVINGPOINT(Xcoordinate,Ycoordinate)
                            motormove(centertarget,moving_point)

            # Print time (inference + NMS)
            #print(f'{s}Done. ({t2 - t1:.3f}s)')

            # Stream results
            if view_img:
                cv2.imshow(str(p), im0)
                cv2.waitKey(1)  # 1 millisecond

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                    print(f" The image with the result is saved in: {save_path}")
                else:  # 'video' or 'stream'
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.release()  # release previous video writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                            save_path += '.mp4'
                        vid_writer = cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer.write(im0)

                if save_txt or save_img:
                    s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir /
'labels'}" if save_txt else ''
                    #print(f"Results saved to {save_dir}{s}")

                print(f'Done. ({time.time() - t0:.3f}s)')


if __name__ == '__main__':
                parser = argparse.ArgumentParser()
                parser.add_argument('--weights', nargs='+', type=str, default='yolov7.pt',
help='model.pt path(s)')
                parser.add_argument('--source', type=str, default='inference/images', help='source')
# file/folder, 0 for webcam
                parser.add_argument('--img-size', type=int, default=640, help='inference size
(pixels)')
                parser.add_argument('--conf-thres', type=float, default=0.25, help='object
confidence threshold')
                parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold for
NMS')
                parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or
cpu')
                parser.add_argument('--view-img', action='store_true', help='display results')
                parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
                parser.add_argument('--save-conf', action='store_true', help='save confidences in --
save-txt labels')
```

```
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--no-trace', action='store_true', help='don`t trace model')
    opt = parser.parse_args()
    print(opt)
    #check_requirements(exclude=('pycocotools', 'thop'))

    with torch.no_grad():
        if opt.update:  # update all models (to fix SourceChangeWarning)
            for opt.weights in ['yolov7.pt']:
                detect()
                strip_optimizer(opt.weights)
        else:
            detect()
```

Appendix B: Arduino Code for Auto Tracking

```
#include <Stepper.h>

int numberofsteps = 360/0.05; //0.05 steps per rev
int minimum_steps_per_move = 200; // define minimum steps per move

//the shaft of the motor shoud facing down to know the clockwise and anticlockwise
direction of the motor A and B
Stepper motorAcw(numberofsteps, 8, 9); //for motor A rotate in clockwise direction
Stepper motorAccw(numberofsteps, 10, 11);// for motor A rotate in counter-clockwise
direction
Stepper motorBcw(numberofsteps, 4,5);//for motor B rotate in clockwise direction
Stepper motorBccw(numberofsteps, 6,7);// for motor B rotate in counter-clockwise
direction

int incomingByte; //variable stores serial data

void setup()
  {
     motorAcw.setSpeed(20);
     motorAccw.setSpeed(20);
     motorBcw.setSpeed(20);
     motorBccw.setSpeed(20);
     Serial.begin(115200);
     Serial.println("Connection established...");

  }

void loop()
  {
    while (Serial.available())
      {
        incomingByte = Serial.read();
        if ((incomingByte == '0') || (incomingByte == '5'))
        {
```

```
        Serial.println("No movement is done.");
      }
   else if (incomingByte == '1') //motorAcw
   {
     motorAcw.step(minimum_steps_per_move);
     Serial.println("motor A move in clockwise success");
     Serial.println("Frame is moving down");
     Serial.println("Target is moving up");}

   else if (incomingByte == '2') // motorAccw
   {   motorAccw.step(minimum_steps_per_move);
     Serial.println("motor A move in counter-clockwise success");
     Serial.println("Frame is moving up");
     Serial.println("Target is moving down");}

   else if (incomingByte == '3') // motorBcw
   {
     motorBcw.step(minimum_steps_per_move);
     Serial.println("motor B move in clockwise success");
     Serial.println("Frame is moving left");
     Serial.println("Target is moving right");}

   else if (incomingByte == '4') // motorBccw
   {   motorBccw.step(minimum_steps_per_move);
     Serial.println("motor B move in counter-clockwise success");
     Serial.println("Frame is moving right");
     Serial.println("Target is moving left");}
 }
      }
```

Appendix C: Arduino Code for Manual Tracking

```
#include <Stepper.h>

int numberofsteps = 360/0.05; //0.05 steps per rev
int minimum_steps_per_move = 5000; // define minimum steps per move

//the shaft of the motor shoud facing down to know the clockwise and anticlockwise
direction of the motor A and B
Stepper motorAcw(numberofsteps, 8, 9); //for motor A rotate in clockwise direction
Stepper motorAccw(numberofsteps, 10, 11);// for motor A rotate in counter-clockwise
direction
Stepper motorBcw(numberofsteps, 4,5);//for motor B rotate in clockwise direction
Stepper motorBccw(numberofsteps, 6,7);// for motor B rotate in counter-clockwise
direction

int VRx = A0;
int VRy = A1;
int SW = 2;

int xPosition = 0;
int yPosition = 0;
int SW_state = 0;
int mapX = 0;
int mapY = 0;

void setup()
{
  pinMode(VRx, INPUT);
  pinMode(VRy, INPUT);
  pinMode(SW, INPUT_PULLUP);

  motorAcw.setSpeed(20);
  motorAccw.setSpeed(20);
  motorBcw.setSpeed(20);
  motorBccw.setSpeed(20);
  Serial.begin(9600);
  Serial.println("Connection established...");
}

void loop()
{
  xPosition = analogRead(VRx);
  yPosition = analogRead(VRy);
  SW_state = digitalRead(SW);

  mapX = map(xPosition, 0, 1023, -512, 512);
  mapY = map(yPosition, 0, 1023, -512, 512);

  if (mapX > 20) // motorAcw
    {
      motorAcw.step(minimum_steps_per_move);
      Serial.println("motor A move in clockwise success");}

    else if (mapX < -20) // motorAccw
    {   motorAccw.step(minimum_steps_per_move);
      Serial.println("motor A move in counter-clockwise success");}

    else if (mapY > 20) // motorBcw
    {
      motorBcw.step(minimum_steps_per_move);
      Serial.println("motor B move in clockwise success");}

    else if (mapY < -20) // motorBccw
    {   motorBccw.step(minimum_steps_per_move);
      Serial.println("motor B move in counter-clockwise success");}

  //Serial.print("X: ");
  //Serial.print(mapX);
  //Serial.print(" | Y: ");
  //Serial.print(mapY);
  //Serial.print(" | Button: ");
  //Serial.println(SW_state);

  delay(100);

}
```