UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

INSTITUTO DE ENERGÍA SOLAR

MÁSTER UNIVERSITARIO EN ENERGÍA SOLAR FOTOVOLTAICA

TRABAJO FIN DE MÁSTER

Evaluación del Recurso Solar y la Generación de Energía en los alrededores del Instituto de Energía Solar para integración en el Vehículo Eléctrico

Written by Jimmy Son

Madrid, July 2022

MÁSTER UNIVERSITARIO EN ENERGÍA SOLAR FOTOVOLTAICA

TRABAJO FIN DE MÁSTER

Título: Evaluación del Recurso Solar y la Generación de Energía en los alrededores del Instituto de Energía Solar para integración en el Vehículo Eléctrico

Autor: D. Jimmy Son

Tutor: D. Javier Macías Rodríguez

Ponente: D. Ignacio Antón Hernández

Departamento: INSTITUTO DE ENERGÍA SOLAR

MIEMBROS DEL TRIBUNAL

Presidente: D. Antonio Martí Vega

Vocal: D. Iván García Vara

Secretario: D. Pablo Palacios Clemente

Suplente: D. David Fuertes Marrón

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

Abstract

The global warming has become one of the most mentioned issues all around the world as its major causes have been the human activities, mostly from the extensive use of fossil fuels. With the rise of a global tendency to replace the fossil fuels with renewable sources, solar energy sector has been gaining a huge momentum in the past decade.

Solar energy is dependent on the weather conditions, and entails higher level of complexity when compared to the fossil fuels. Utilizing available database and computing tools, it has been made possible to analyze and draw useful information about the solar resource and estimate the performance of a PV system.

This project will study the relevance of Sky Field of View - SFV model within the urban environments in calculating the diffuse irradiance, and how it affects the solar irradiation. The basis of this work will be linked to the Vehicle Integrated PV - VIPV technology, in a sense that the goal is to measure the amount of energy a vehicle with solar panels mounted on its roof can extract from the parking spaces of the Institute of Solar Energy (IES).

In this project, a low-cost 180° panoramic camera will be used to take photos throughout the IES parking structures. With the help of a computer program and open-source libraries designed for image-processing and PV system performance, SFV as well as the solar resource and PV generation of each parking spot will be calculated.

Glossary

SFV	Sky field of view
RGM	Region growth method
POA	Plane of array
GHI	Global horizontal irradiance
DNI	Direct normal irradiance
DHI	Diffuse horizontal irradiance
TMY	Typical meteorological year
PV	Photovoltaic
VIPV	Vehicle integrated photovoltaic
AOI	Angle of incident
SAPM	Sandia PV array performance model
G	Solar constant
ISO	International Organization for Standardization
AM	Air mass
UML	Unified Modeling Language
G	Global
B	Direct
D	Diffuse
PPI	Pixels per inch
RGB	Red Green Blue
HSV	Hue Saturation Value

Table of Contents

1. Introduction

1.1. Motivation

This project idea was derived from conducting case-studies in the VIPV technology. The nature of the PV-integrated vehicles requires them to be present within the urban environments most of the time. In order for these vehicles to generate useful energy from the solar radiation, it is important to consider that the module is mounted on top of them, and understand how much solar radiation can reach the module. The urban environments imply that the visible hemispheric view from the module's perspective will vary depending on what kind, and how many obstacles are around it; these obstacles include buildings, roofs, facades, trees, lamp posts and vehicles. The obstacles will inadvertently affect the diffuse radiation reaching the module, and cause the solar resource and PV generation to diminish. This observation sparked a question of whether the conventional sky diffuse model is suitable in the world of VIPV.

To adjust and study the effects of local obstacles in VIPV's performance, a proposed SFV diffuse model is implemented for estimating the diffuse irradiance. SFV (Sky Field of View) is defined as a ratio of the visible sky to the hemispheric frame, and it will be used as a coefficient for adjusting the diffuse irradiance according to the vehicle-mounted module's hemispheric view. As the implementation of SFV reduces the diffuse component, the energy production of the VIPV system will also be reduced, rendering it more accurate. From the simulations, solar resource and VIPV production will be compared between this diffuse model and the conventional Isotropic diffuse model, which assumes the uniform composition across the sky dome.

The goal of this project is to show the extent to which the varying number of obstacles in the urban environment can affect the solar resource and the VIPV system's production – and examine the validity of the conventional diffuse model in VIPV applications.

1.2. Objectives

This project assumes the hemispheric view of distinct IES parking spot will capture a unique number of obstacles, or, a unique SFV. This means the solar resource and PV production will fluctuate between parking spots, despite their proximity.

To carry out this project, the hemispheric view of the vehicle from distinct parking spots needs to be collected for SFV calculation. This will be done by taking numerous photos in the IES parking lots with a 180° panoramic camera at a horizontal position. After the physical procedure of obtaining the images, the project will be divided into two main programming parts: 1) SFV calculation, and 2) solar resource and PV generation calculation.

Once the images are obtained, Python and an image-processing library will be used to analyze them, detecting the sky region within the hemispheric frame of each image and calculating the corresponding SFV values.

Once the SFV values are calculated, Python and a PV library will be used to calculate the solar resource and PV generation of each parking spot. A series of built-in functions of this library will be used to gather information about the solar resource; in this step, the user will be able to specify the sky diffuse model – in which the SFV values can be applied. The simulation will be run twice using both Isotropic diffuse model and SFV diffuse model, in order to highlight the relevance of SFV.

The scope of this project is limited, as the PV library implements a generic obstacle-free model – where the shadows caused by obstacles are disregarded. This would adversely affect the PV production, as shadows are not favorable to solar panels.

1.3. Report Layout

In the following chapters, this report will begin by explaining the fundamentals of solar energy such as solar radiation and its components, and generation by photovoltaic effect. This section will include distinct models for calculating the sky diffuse irradiance and emphasize on the proposed SFV model implemented in this project.

Next, an open-source PV library called PVLIB will be explained about along with the general description of a PV system. This is to explain how this tool runs the simulation and generates the main solar parameters and PV performance data for the

analysis. Then, a general description of the VIPV system along with its relationship with SFV will be covered, as this is tied to the main purpose of the project.

A substantial part of this report will focus on explaining the performed procedures, and the translation between real-world PV operation and virtual PV simulation. This includes every step involved in the IES parking lots photoshoot, digital image-processing for SFV calculation, and PVLIB modeling of solar resource and PV production via different sky diffuse models.

Finally, two sets of results will be obtained for each parking spot - solar resource and PV generation via: 1) Isotropic diffuse model, and 2) SFV diffuse model. These results will be analyzed and the conclusions will be drawn.

2. Fundamental Theory

2.1. Solar Energy

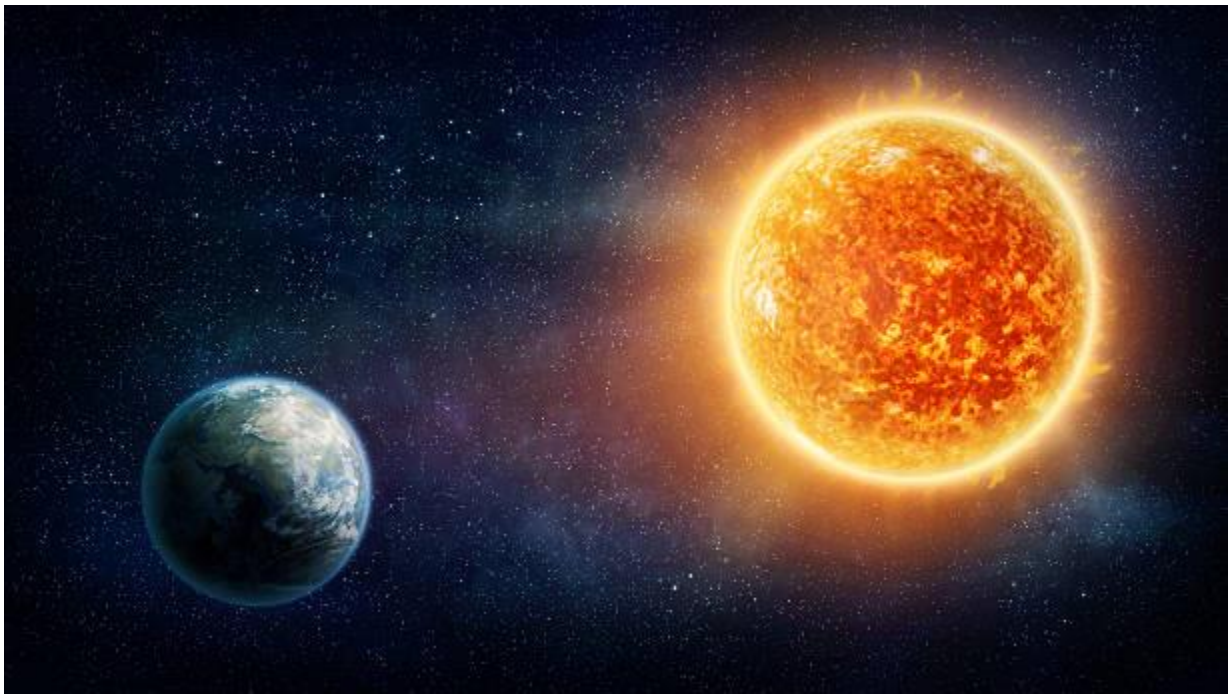


Figure1: sunlight and Earth
source: https://www.istockphoto.com/es/fotos/sunlight-and-earth

Sun is the source of the infinite and promising renewable energy, which is the basis of the PV technology.

The sunlight is a fraction of the electromagnetic radiation which the sun emits, and is a necessary body to all life on Earth. Specifically, sunlight refers to the fraction of the electromagnetic wave which includes visible, infrared and ultraviolet light. This sunlight takes about 8 minutes to reach the Earth from the surface of the sun, and arrives at the intensity, known as "solar constant," G= $1367\frac{W}{m^2}$. This radiation received at the top of Earth's atmosphere is uniformly equal amount. As the solar radiation travels down to a specific location on Earth, the irradiance varies as it is dependent on multiple factors such as weather and geography.

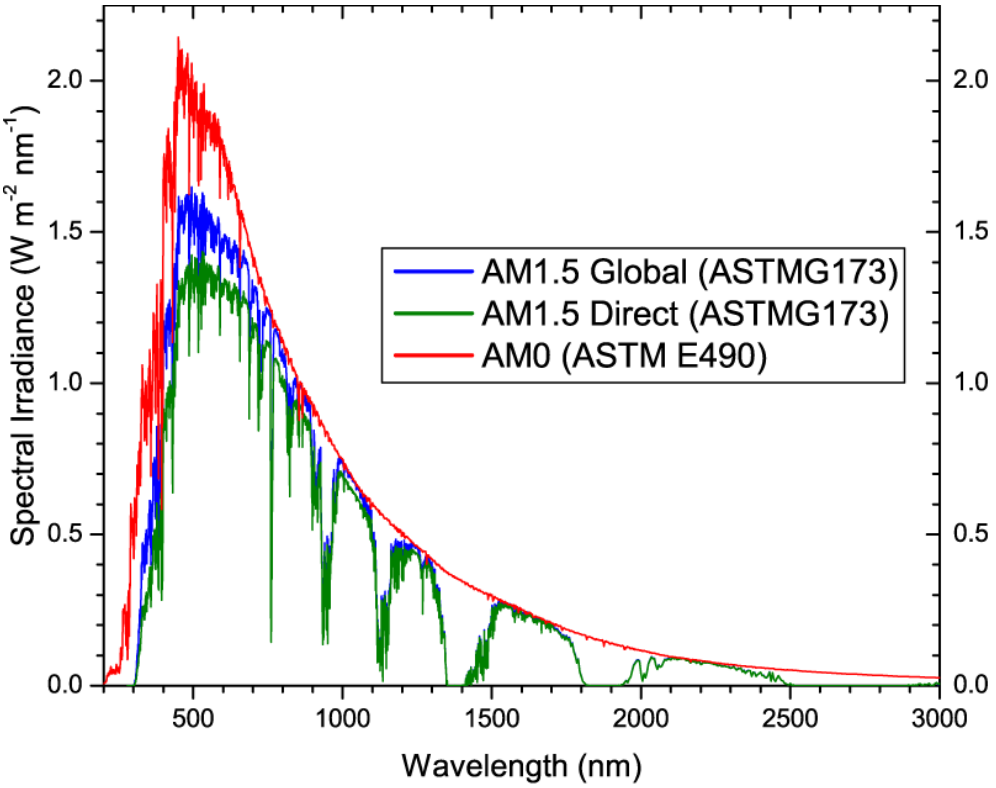


Figure2: reference solar spectra
source: https://www.pveducation.org/pvcdrom/appendices/standard-solar-spectra

Solar spectrum captures the sun's electromagnetic radiation ranging between distinct wavelengths, and is in a form of right-skewed distribution; the wavelengths of infrared, ultraviolet, and visible light absorb the most part of the spectral irradiance. Although the solar spectrum continues to change depending on the day and location, standard references have been set by the international standard ISO 9845-1.

Two standard references are accepted nowadays for terrestrial use, and those are AM1.5 G and AM1.5 D. The former has an integrated intensity of $1000\frac{W}{m^2}$ and it is used in most calculations as it is designed for flat modules, also known as the standard test condition, irradiance.

2.2. Solar Radiation

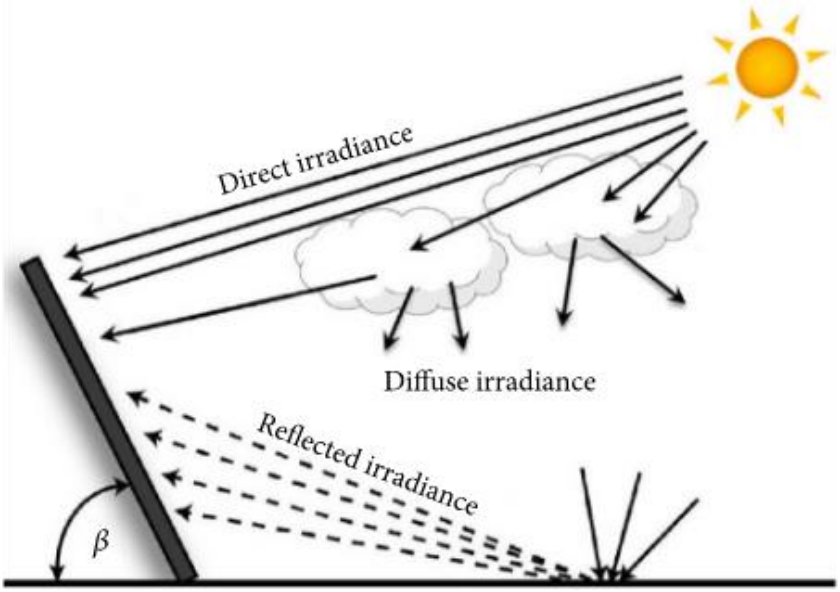


Figure3: components of global irradiance
source: https://www.pveducation.org/pvcdrom/appendices/standard-solar-spectra

The solar irradiance can be split into two main parts:

Direct(B), Diffuse(D)

- Direct irradiance is the part of the solar radiation that directly reaches the surface of Earth.
- Diffuse irradiance is the part of the solar radiation that gets absorbed, scattered or reflected by the presence of air molecules, clouds, dust and other physical matters.

The sum of direct and diffuse irradiance is called the global irradiance (β=inclination angle).

$$B(\beta)+D(\beta) = G(\beta)$$

Using these two components of the radiation, the solar pioneers came up with a number of radiation components that contain useful information and they are now used for the solar performance calculations. These components include: GHI, DNI, DHI, POA Global, POA Direct, and POA Diffuse.

The former three (GHI, DNI, DHI) capture the information about available solar resource solely based on the geography, independent of the PV components.

- GHI: Global Horizontal Irradiance,
  The total amount of radiation per unit area received by a surface horizontal surface to the ground

- DNI: Direct Normal Irradiance
  The amount of radiation received per unit area by a surface which is perpendicular to the incident sunlight

- DHI: Diffuse Horizontal Irradiance
  The amount of radiation received per unit area by a surface which does not directly come from the sunlight, but from the scattering in the atmosphere, and comes equally from all direction

POA (Plane of Array) components also capture the information about available solar resource, but in reference to the incident irradiance on the inclined PV module, instead of the horizontal surface. POA irradiance is the actual parameter used in the calculations of the performance of a PV system. In this project, the PV module is assumed to have no inclination angle as it is facing upward on the vehicle's roof ($\beta=0$).

In practice, solar radiation is measured throughout the day using pyranometer and pyrheliometer. These devices are installed in obstacle-free settings, where no part of their hemispheric view is covered by obstacles.

Pyranometer is used for measuring solar irradiance on a planar surface. Therefore, it can be placed horizontal to the ground to capture the GHI, or have a rod with sphere cover the direct beam from the sun to capture the DHI. It can measure the incident radiation on an inclined surface as well, if mounted at an angle.

Figure4: pyranometer for global irradiance measurement
source: https://es.m.wikipedia.org/wiki/Archivo:SR20_pyranometer_1.jpg

Pyrheliometer is used for measuring the direct beam solar irradiance. It is equipped with a tracking mechanism that points the device toward the sun continuously. DNI is most accurately measured by a high-quality Pyrheliometer.

Figure5: pyrheliometer for direct irradiance measurement
source:
https://es.wikipedia.org/wiki/Pirheli%C3%B3metro#/media/Archivo:DR01_pyrheliometer_1.jpg

A theoretical model used to calculate the GHI, DNI, and DHI of a clear sky is Ineichen/Perez model[6]. This model is one of the methods implemented in PVLIB as well. This model takes in input of zenith angle (between the sun's beam and the vertical direction), airmass, turbidity, altitude and extraterrestrial irradiance to calculate the clear-sky DHI by applying the zenith angle to the DNI, as shown below.

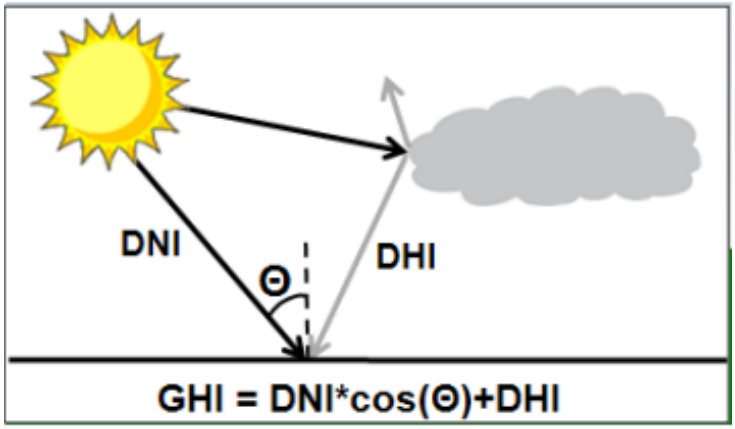


Figure6: Ineichen/Perez model for DHI calculation
source: https://www.icao.int/environmental-protection/Pages/default.aspx

POA components are calculated from the standard components (GHI, DNI, DHI) using multiple available models.

The POA Direct irradiance is calculated by adjusting the DNI by the AOI (θ), as the following[6]:

$$G_{POA,Direct} = DNI * \cos(\theta)$$

Where the angle of incident (AOI) is the angle between the sun's beam and the vector normal to the PV module.

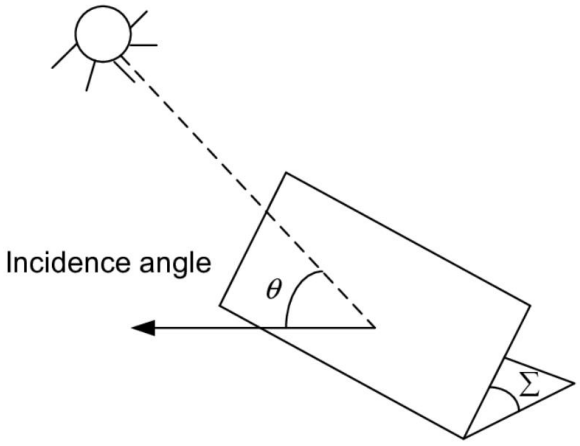


Figure7: Angle of Incidence
source:
https://www.researchgate.net/figure/The-incidence-angle-th-between-a-normal-to-the-PV-module-face-and-the-incoming-solar-beam_fig3_252018864

The POA Diffuse irradiance is calculated using multiple sky diffuse models[5] – Isotropic, Sandia, Hay and Davies, Reindl, and Perez.

This project will implement the Isotropic model, which serves as the foundation of other variant models. This model is derived by Liu-Jordan model (1961), which separates the total short-wave radiation into direct and diffuse components, and assumes that the diffuse radiation from the sky dome is uniform across the sky.

The POA sky diffuse irradiance under the Isotropic model is calculated by multiplying the DHI by a factor as the following, where β is the inclination angle[5]:

$$G_{POA,Diffuse,Isotropic} = DHI * \frac{(1 + \cos(\beta))}{2}$$

Because the diffuse component of the solar irradiance is scattered in the atmosphere, it is essential for the visible sky to be present in the hemispheric view. The direct beam would then be able to interfere with the particles and atoms to create the scattering effect. In other words, it would be inaccurate to assume full diffuse irradiance for a location within urban environment whose hemispheric view is entirely covered by obstacles. This logic applies for the reason that pyranometers are installed where their hemispheric view is completely clear of obstacles.

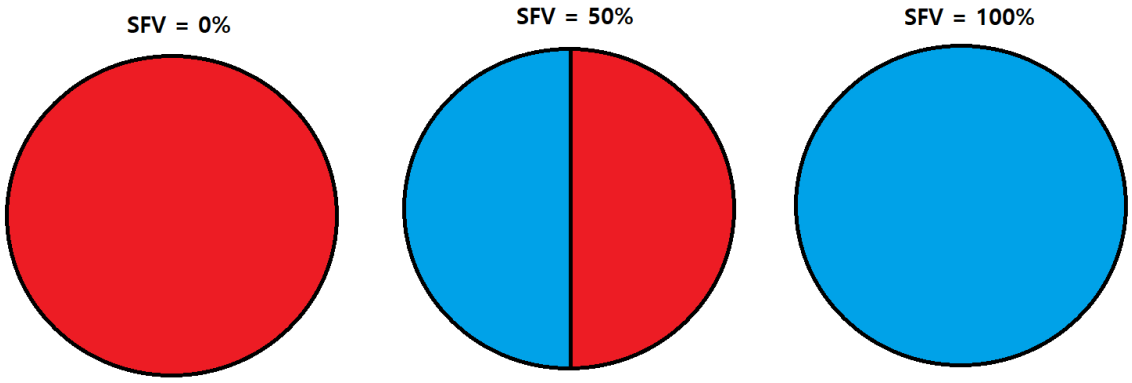


Figure8: SFV sample demonstration – 0% ,50% ,100%

In *Figure 8*, the hemispheric view has a circular frame, and the obstacles are represented by red, while the visible sky regions are represented by blue. SFV value represents the ratio of the visible sky to the hemispheric frame.

The SFV sky diffuse model is a modified version of Liu-Jordan model, where the factor $(1+\cos\beta)/2$ is replaced by the SFV value corresponding to the hemispheric view. This model assumes that the only the proportion of the hemisphere that is clearly visible from the collection plane should be considered for the calculation of diffuse irradiance – discarding the sections covered by obstacles in the whole visible hemisphere[5]. This implementation of SFV model does not take into account that sky radiance has an anisotropic character; however, the isotropic sky radiation was assumed in this project.

$$G_{POA,Diffuse,SFV} = DHI * SFV$$

This model is effective for urban environments where presence of obstacles such as trees and buildings occupy considerable amount of the visible hemisphere - IES faculty being a proper choice of our case study.

Finally, the sum of direct and diffuse components will be the POA global irradiance:

$$G_{POA,Direct} + G_{POA,Diffuse} = G_{POA,Global}$$

2.3. PV Generation

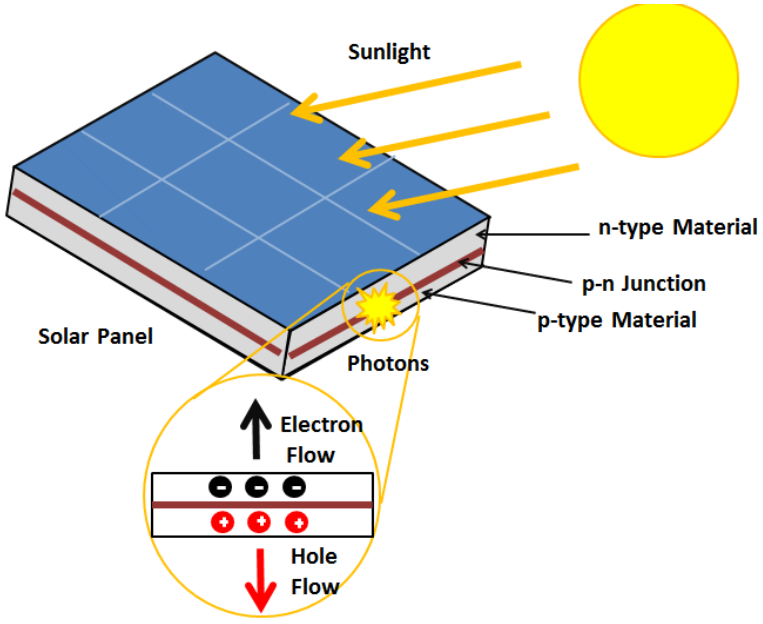


Figure9: Photovoltaic effect diagram
Source: https://energyeducation.ca/encyclopedia/Photovoltaic_effect

To determine the performance of a solar cell and a module, the electrical characteristics current(I) and voltage(V) are measured to generate an I-V curve. The curve contains detailed description of the conversion ability of the cell by providing the most important parameters including open-circuit voltage (V_{OC}), short-circuit current (I_{SC}), fill factor (FF), and maximum power point information (V_{MP}, I_{MP}, P_{MP}).

- V_{OC}: Open Circuit Voltage,
  The maximum voltage a solar cell or a module provides under open circuit condition

- I_{SC}: Short Circuit Current,
  The maximum current a solar cell or a module provides under short circuit condition

- P_{MP}: Power, Maximum Power Point
  The power at a maximum point within the I-V curve where MPP is the product of Imp and Vmp.

- FF: Fill Factor
  The relationship between the maximum power a solar cell or a module can provide under normal operating conditions and the product of Voc and Isc. This value is an overall indicator of the quality of a solar cell or a module.

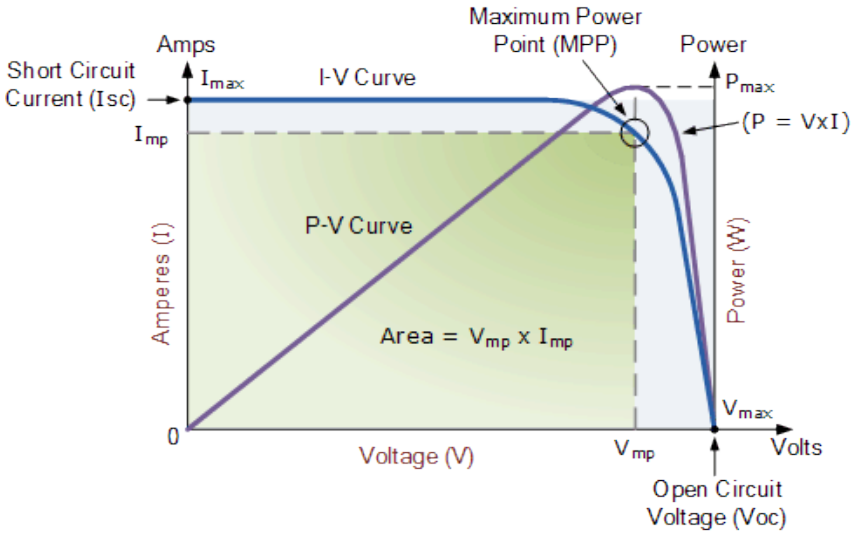


Figure10: A typical solar cell I-V graph
Source: https://www.alternative-energy-tutorials.com/photovoltaics/solar-cell-i-v-characteristic.html

A solar panel, or a module, is made of a group of solar cells that are usually in a configuration of 32, 36, 48, 60, 72, and 96 in series. There are many different types of modules nowadays, but the most commonly used are the monocrystalline silicon solar panels.

Figure11: A solar panel assembly
Source: https://www.cleanenergyreviews.info/blog/solar-panel-components-construction

PV system is defined by a combination of components whose roles facilitate the conversion of sunlight to useful energy consumed by the loads or the electrical grid. These components include: solar array, charge controller, inverter, and battery (optional).

- Solar array is a combination of modules in series and parallel, whose configuration is designed according to the compatibility of its electrical characteristics with other components in the PV system.

- Charge controller is a device that regulates the rate at which the solar array input current moves in and out of the battery and to the inverter. This is to prevent the over-charging/over-discharging of the battery.

- Inverter is a device that converts the solar array input DC current to the AC current, so that it can be sent to either the loads or the electrical grid.

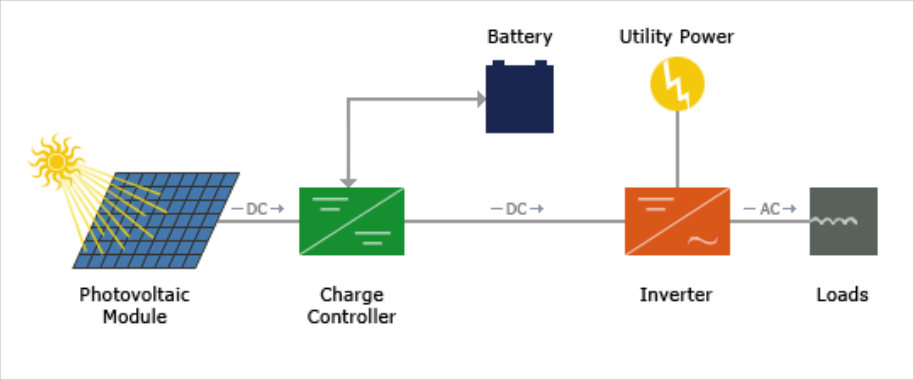


Figure12: A solar panel assembly
Source: http://www.synergyenviron.com/resources/solar-photovoltaic-systems

2.4. PVLIB Model

The section *2.2. and 2.3.* covered all necessary information - solar variables and general setup of a PV system – to convert sunlight to useful energy.

In this project, the virtual modeling paradigm used to simulate this process is created with the help of python's open-source library called PVLIB. Developed by Sandia National Laboratories, PVLIB implements various models and methods for simulating the performance of a PV system[9].

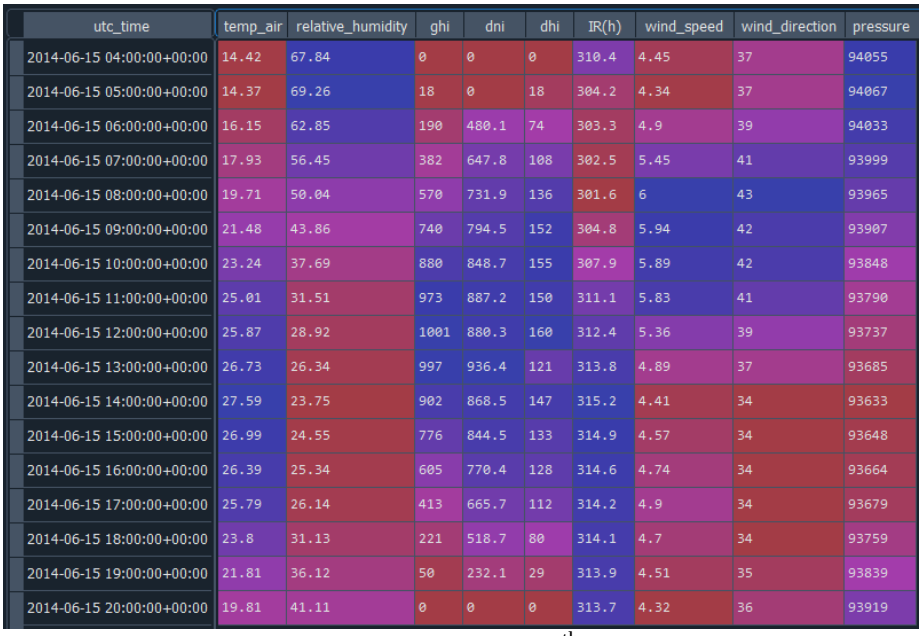
Figure13: The library logo - PVLIB
Source: https://pvlib-python.readthedocs.io/en/stable/index.html

Other popular PV simulation software include SISIFO, PVSyst, Homer Pro, Solar Pro, etc. The modeling paradigm for every program is roughly the same – quarrying weather data, calculating solar parameters using defined models, selecting PV module and inverter, and performing a simulation of the PV system.

For the global solar radiation data, a typical meteorological year (TMY) data has been recorded for every hour in a year for a given geographical location. This data is gathered for a long period of time, usually for 10 years or more, and the monthly data are collected based on which year is considered the most typical for that month. A few of the most common database include Meteonorm, NASA, PVGIS, NREL.

Conveniently, PVLIB has a built-in function for generating a typical meteorological year (TMY) data for a given location; PVLIB gathers this data from the PVGIS database. This generated TMY is a 2-dimensional data frame which includes the solar variables discussed in *Section 2.2*. - GHI, DNI, DHI. All information in PVLIB simulation is saved hourly, and the amount of data can be truncated to display only the chosen date.

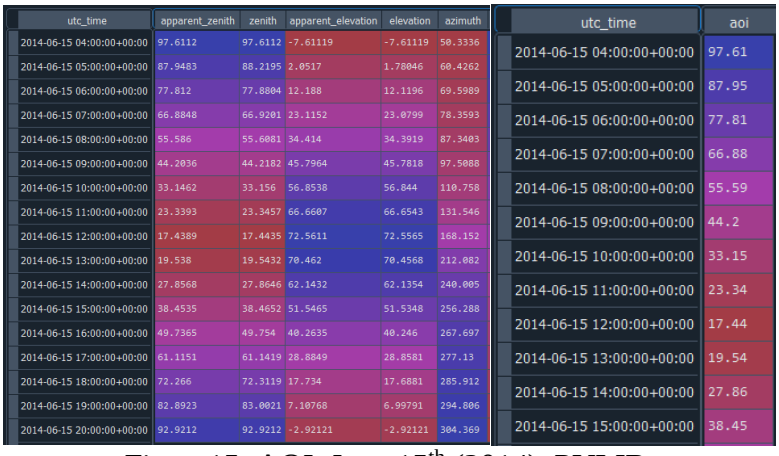
For the month of June, year 2014 has been considered to be the most typical. The following table is an example of a PVLIB generated TMY data for June 15th, 2014.



utc_time	temp_air	relative_humidity	ghi	dni	dhi	IR(h)	wind_speed	wind_direction	pressure
2014-06-15 04:00:00+00:00	14.42	67.84	0	0	0	310.4	4.45	37	94055
2014-06-15 05:00:00+00:00	14.37	69.26	18	0	18	304.2	4.34	37	94067
2014-06-15 06:00:00+00:00	16.15	62.85	190	480.1	74	303.3	4.9	39	94033
2014-06-15 07:00:00+00:00	17.93	56.45	382	647.8	108	302.5	5.45	41	93999
2014-06-15 08:00:00+00:00	19.71	50.04	570	731.9	136	301.6	6	43	93965
2014-06-15 09:00:00+00:00	21.48	43.86	740	794.5	152	304.8	5.94	42	93907
2014-06-15 10:00:00+00:00	23.24	37.69	880	848.7	155	307.9	5.89	42	93848
2014-06-15 11:00:00+00:00	25.01	31.51	973	887.2	150	311.1	5.83	41	93790
2014-06-15 12:00:00+00:00	25.87	28.92	1001	880.3	160	312.4	5.36	39	93737
2014-06-15 13:00:00+00:00	26.73	26.34	997	936.4	121	313.8	4.89	37	93685
2014-06-15 14:00:00+00:00	27.59	23.75	902	868.5	147	315.2	4.41	34	93633
2014-06-15 15:00:00+00:00	26.99	24.55	776	844.5	133	314.9	4.57	34	93648
2014-06-15 16:00:00+00:00	26.39	25.34	605	770.4	128	314.6	4.74	34	93664
2014-06-15 17:00:00+00:00	25.79	26.14	413	665.7	112	314.2	4.9	34	93679
2014-06-15 18:00:00+00:00	23.8	31.13	221	518.7	80	314.1	4.7	34	93759
2014-06-15 19:00:00+00:00	21.81	36.12	50	232.1	29	313.9	4.51	35	93839
2014-06-15 20:00:00+00:00	19.81	41.11	0	0	0	313.7	4.32	36	93919

Figure14: PVGIS-TMY, June 15th (2014), PVLIB coordinate = (40.4531, -3.7269)

As explained in the *Section 2.2.*, the position of the sun determines the AOI relative to the position of the module. While the module is fixed, the sun's position and the AOI will continuously change throughout the day – this will affect the direct beam reaching the panel, or POA Direct irradiance.
PVLIB uses coordinate information and TMY data to calculate solar position parameters such as zenith angle, azimuth angle and solar elevation – this data then is used for the calculation of AOI, as shown below.



utc_time	apparent_zenith	zenith	apparent_elevation	elevation	azimuth
2014-06-15 04:00:00+00:00	97.6112	97.6112	-7.61119	-7.61119	50.3336
2014-06-15 05:00:00+00:00	87.9483	88.2195	2.0517	1.78046	60.4262
2014-06-15 06:00:00+00:00	77.812	77.8804	12.188	12.1196	69.5989
2014-06-15 07:00:00+00:00	66.8848	66.9201	23.1152	23.0799	78.3593
2014-06-15 08:00:00+00:00	55.586	55.6081	34.414	34.3919	87.3403
2014-06-15 09:00:00+00:00	44.2036	44.2182	45.7964	45.7818	97.5088
2014-06-15 10:00:00+00:00	33.1462	33.156	56.8538	56.844	110.758
2014-06-15 11:00:00+00:00	23.3393	23.3457	66.6607	66.6543	131.546
2014-06-15 12:00:00+00:00	17.4389	17.4435	72.5611	72.5565	168.152
2014-06-15 13:00:00+00:00	19.538	19.5432	70.462	70.4568	212.082
2014-06-15 14:00:00+00:00	27.8568	27.8646	62.1432	62.1354	240.005
2014-06-15 15:00:00+00:00	38.4535	38.4652	51.5465	51.5348	256.288
2014-06-15 16:00:00+00:00	49.7365	49.754	40.2635	40.246	267.697
2014-06-15 17:00:00+00:00	61.1151	61.1419	28.8849	28.8581	277.13
2014-06-15 18:00:00+00:00	72.266	72.3119	17.734	17.6881	285.912
2014-06-15 19:00:00+00:00	82.8923	83.0021	7.10768	6.99791	294.806
2014-06-15 20:00:00+00:00	92.9212	92.9212	-2.92121	-2.92121	304.369

utc_time	aoi
2014-06-15 04:00:00+00:00	97.61
2014-06-15 05:00:00+00:00	87.95
2014-06-15 06:00:00+00:00	77.81
2014-06-15 07:00:00+00:00	66.88
2014-06-15 08:00:00+00:00	55.59
2014-06-15 09:00:00+00:00	44.2
2014-06-15 10:00:00+00:00	33.15
2014-06-15 11:00:00+00:00	23.34
2014-06-15 12:00:00+00:00	17.44
2014-06-15 13:00:00+00:00	19.54
2014-06-15 14:00:00+00:00	27.86
2014-06-15 15:00:00+00:00	38.45

Figure15: AOI, June 15th (2014), PVLIB coordinate = (40.4531, -3.7269)

The solar position as well as AOI information are saved for each hour, and can be called afterwards for the calculation of POA irradiance.

PVLIB uses its built-in function – which takes in solar position, module position, and TMY information - to calculate the POA irradiance (Global, Direct, Diffuse) with an option to choose the sky diffuse model. This step is key to the simulation of a PV system, because POA irradiance is the actual solar resource the module will be receiving to produce the energy.

This function uses the panel position, solar position, and DNI for the calculation of POA Direct irradiance, given by the formula $G_{POA,Direct} = DNI * \cos(AOI)$.

Because PVLIB is not aware of the structures of IES buildings, it also does not know which parking spots are covered in shadows throughout different times of the year. This is where the scope of this project becomes limited – as DNI is heavily affected by shadows. When a module is covered by shadows, its PV production is reduced as the direct component consists dominant share of the global irradiance. Without a proper shadowing-corrected DNI model for IES building, the result can be analyzed only for parking spots without shadows.

For the calculation of POA Diffuse irradiance, because the selected sky diffuse model is Isotropic, this function takes in DHI and scale it by a factor, which is directly proportional to the module's inclination angle, given by the formula $G_{POA,Diffuse,Isotropic} = DHI * \frac{(1+\cos(\beta))}{2}$.

To implement the SFV sky diffuse model, corresponding SFV value will be multiplied to DHI found from TMY in the previous step, given by the formula: $G_{POA,Diffuse,SFV} = DHI * SFV$.

The sum of POA Direct and Diffuse represents the POA Global irradiance.

utc_time	poa_global	poa_direct	poa_diffuse
2014-06-15 04:00:00+00:00	0	0	0
2014-06-15 05:00:00+00:00	18	0	18
2014-06-15 06:00:00+00:00	175.4	101.4	74
2014-06-15 07:00:00+00:00	362.3	254.3	108
2014-06-15 08:00:00+00:00	549.7	413.7	136
2014-06-15 09:00:00+00:00	721.6	569.6	152
2014-06-15 10:00:00+00:00	865.6	710.6	155
2014-06-15 11:00:00+00:00	964.6	814.6	150
2014-06-15 12:00:00+00:00	999.8	839.8	160
2014-06-15 13:00:00+00:00	1003	882.5	121
2014-06-15 14:00:00+00:00	914.8	767.8	147
2014-06-15 15:00:00+00:00	794.3	661.3	133
2014-06-15 16:00:00+00:00	625.9	497.9	128
2014-06-15 17:00:00+00:00	433.6	321.6	112
2014-06-15 18:00:00+00:00	238	158	80
2014-06-15 19:00:00+00:00	57.72	28.72	29
2014-06-15 20:00:00+00:00	0	0	0

Figure16: POA Irradiance, June 15th (2014), PVLIB coordinate = (40.4531, -3.7269)

The remaining variables to calculate before the simulation of a PV system are the cell temperature, and effective irradiance. PVLIB adopts Sandia PV Array Performance Model (SAPM)[9] to generate a PV module's I-V curve, which includes the parameters such as Voc, Isc, Vmp, and Imp. The SAPM model takes in the module information and calculates its performance using the adjusted values of the irradiance, and cell temperature. This is because the module information was recorded when calibrated under the standard test condition of the cell temperature at 25°C, and the solar irradiance at $1000\frac{W}{m^2}$.
In actuality, modules tend to operate at a temperature higher than 40°C, and the solar irradiance is never fixed but rather fluctuates throughout the day.

PVLIB calculates both effective irradiance and cell temperature using a number of variables found earlier - including POA components, ambient temperature, airmass, wind speed, AOI, and module information.

Finally, PVLIB implements SAPM model to generate the module's performance (dc), then calculates the energy production (ac) given the input inverter's specifications at its maximum power point, as shown below.

utc_time	i_sc	i_mp	v_oc	v_mp	p_mp	i_x	i_xx
2014-06-15 04:00:00+00:00	0	0	0	0	0	0	0
2014-06-15 05:00:00+00:00	0	0	0	0	0	0	0
2014-06-15 06:00:00+00:00	0.750974	0.678565	53.6579	44.2812	30.0476	0.731764	0.532512
2014-06-15 07:00:00+00:00	1.75088	1.57594	55.0651	45.7169	72.0471	1.70729	1.20818
2014-06-15 08:00:00+00:00	2.76838	2.48219	55.2732	45.3858	112.656	2.70136	1.85688
2014-06-15 09:00:00+00:00	3.66517	3.27469	54.9186	44.4694	145.624	3.57865	2.39642
2014-06-15 10:00:00+00:00	4.40519	3.92415	54.3975	43.467	170.571	4.30338	2.81911
2014-06-15 11:00:00+00:00	4.89968	4.35534	53.8458	42.5801	185.451	4.78806	3.0904
2014-06-15 12:00:00+00:00	5.05893	4.4929	53.4486	42.0579	188.962	4.94421	3.17605
2014-06-15 13:00:00+00:00	5.0914	4.51987	53.1236	41.6879	188.424	4.97604	3.19362
2014-06-15 14:00:00+00:00	4.66338	4.14505	52.9785	41.7863	173.206	4.55638	2.96272
2014-06-15 15:00:00+00:00	4.04975	3.60817	53.3365	42.5416	153.497	3.95517	2.61936
2014-06-15 16:00:00+00:00	3.17896	2.8416	53.6089	43.3628	123.22	3.10286	2.10815
2014-06-15 17:00:00+00:00	2.16002	1.93808	53.5133	43.8529	84.9904	2.10683	1.47399
2014-06-15 18:00:00+00:00	1.09787	0.98916	52.743	43.3934	42.9231	1.07004	0.77129
2014-06-15 19:00:00+00:00	0.224213	0.202747	48.8715	36.9871	7.499	0.218396	0.16125
2014-06-15 20:00:00+00:00	0	0	0	0	0	0	0

utc_time	0
2014-06-15 04:00:00+00:00	-0.075
2014-06-15 05:00:00+00:00	-0.075
2014-06-15 06:00:00+00:00	16.5803
2014-06-15 07:00:00+00:00	53.1906
2014-06-15 08:00:00+00:00	90.0249
2014-06-15 09:00:00+00:00	121.29
2014-06-15 10:00:00+00:00	146.274
2014-06-15 11:00:00+00:00	162.082
2014-06-15 12:00:00+00:00	164.795
2014-06-15 13:00:00+00:00	168.705
2014-06-15 14:00:00+00:00	150.82
2014-06-15 15:00:00+00:00	132.446
2014-06-15 16:00:00+00:00	102.552
2014-06-15 17:00:00+00:00	66.266
2014-06-15 18:00:00+00:00	28.7679
2014-06-15 19:00:00+00:00	1.55474
2014-06-15 20:00:00+00:00	-0.075

Figure17: module performance and energy generation, June 15th (2014), PVLIB coordinate = (40.4531, -3.7269)

To summarize the general structure of PVLIB's simulation model, a unified modeling language (UML) diagram and a function table are created. UML is a general-purpose modeling language and provides a logical understanding of the design of a system – in this case, PVLIB's modeling paradigm.

library	module	function	description
pvlib	irradiance	aoi	Calculates the angle of incidence of the solar vector on a surface
pvlib	iotools	get_pvgis_tmy	Get TMY data from PVGIS
pvlib	solarposition	get_solarposition	A convenience wrapper for the solar position calculators
pvlib	irradiance	get_total_irradiance	Determine total in-plane irradiance and its beam, sky diffuse
pvlib	temperature	sapm_cell	Calculate cell temperature per the SAPM
pvlib	pvsystem	sapm_effective_irradiance	Calculates the SAPM effective irradiance
pvlib	pvsystem	sapm	generates a PV module's I-V curve per the SAPM
pvlib	inverter	sandia	Convert DC power and voltage to AC power

Figure18: PVLIB function table

In *Figure18*, functions imported and utilized from PVLIB are identified as they appear in *Figure19* in the next page as green blocks. In the diagram, the input parameters are marked as blue and the output parameters as red for each function. This diagram visually combines all steps described in this section – from obtaining weather data (TMY) with coordinates, to estimating PV production with chains of calculated parameters. The order of execution generally moves from the left to the right beginning with the "IMAGES", as the sequence can be read by following the arrows.

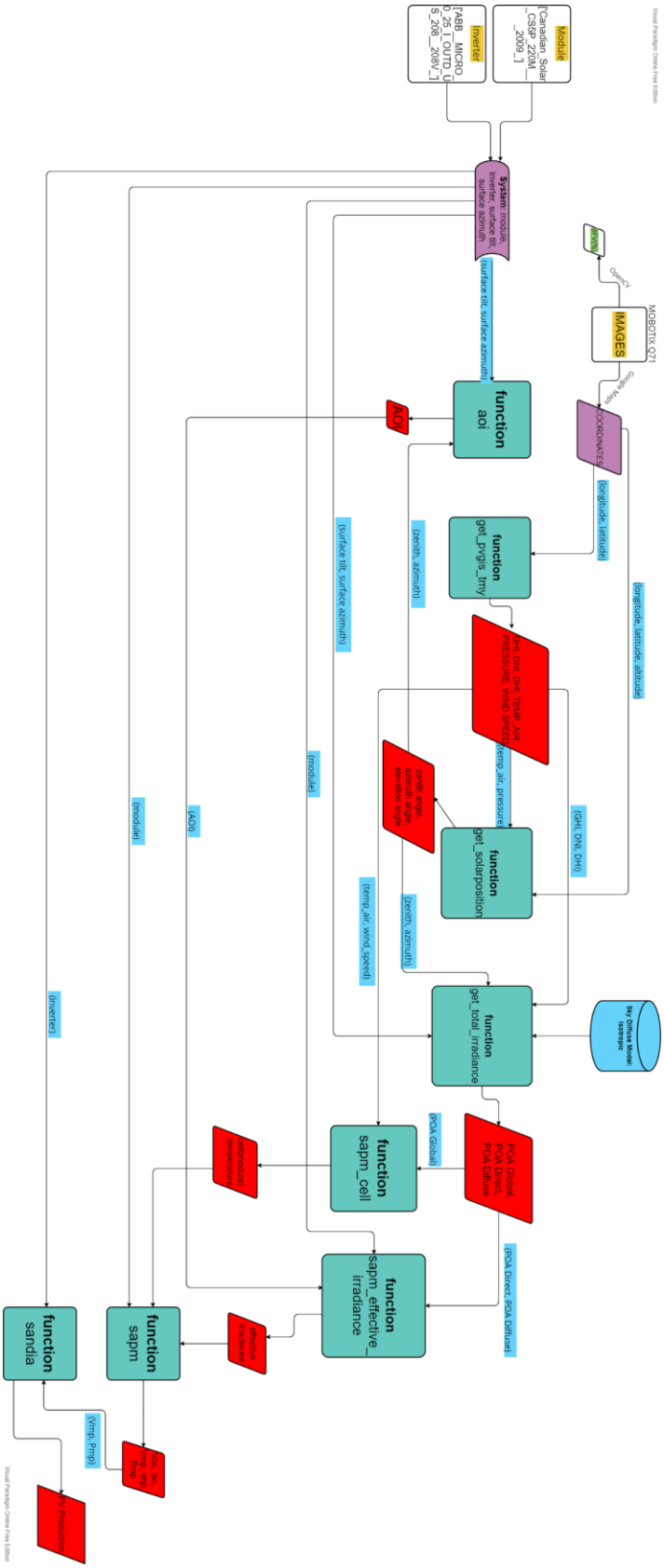


Figure19: UML diagram of PVLIB's simulation

2.5. Integration of PV in Electric Vehicle

Vehicle-Integrated Photovoltaics (VIPV) integrates PV modules into vehicles in a design-technical manner. In this vehicle, the modules blend in with the exterior and physically replace the components such as the bonnet or the roof. The electricity generated by the modules is transferred to the drive battery and electric loads, thereby increasing the mileage of the vehicle while improving its carbon-dioxide balance.

The application of PV modules in electric vehicles need to meet certain aesthetic requirements and the generated energy usually increases the mileage to several kilometers per day, and also allows the remote recharging of the vehicle battery.

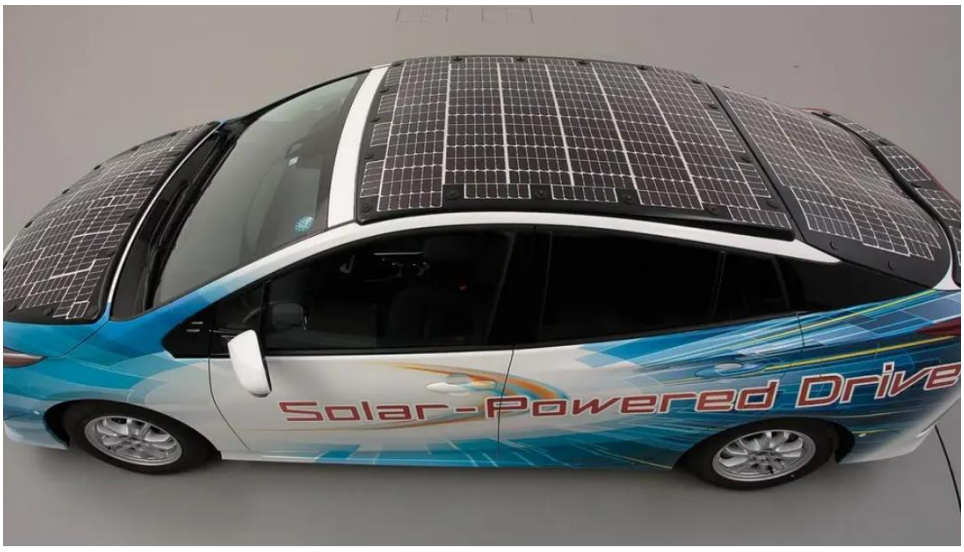


Figure20: Toyota's solar-powered electric car, VIPV
Source: https://www.businessinsider.com/toyota-solar-powered-e-car-never-needs-charging-2019-9

In section *2.2.*, SFV was defined to be an important factor in calculating the sky diffuse irradiance in urban environments. The same concept applies to the VIPVs, as they are designed to move around the cities as a transportation method. This means that the vehicles would experience continuous fluctuation in SFV on each segment of the path, as well as when parked. In other words, the visible sky of the hemispheric view from the perspective of the vehicle's roof is constantly changing.

This would affect the sky diffuse irradiance according to the SFV model implemented in this project, and eventually will reduce the energy production of the integrated modules as the vehicles moves around. This is tied to the core philosophy of this project – examination of SFV's effect on the sky diffuse irradiance and the energy production. The purpose of this is to validate the sky diffuse model for estimating the solar resource and the PV generation within the urban environments, where obstacles are often present in the sky view.

3. Procedures

3.1. All-Sky Camera – Obtaining Photos

The first step in implementing this project is to create a physical representation of the sky-view from the PV-integrated vehicle's roof throughout the urban environment. This is accomplished by going around the IES parking lots and taking photos of the sky.

However, the images need to be in the form of All-Sky before being processed, to include all visible sky regions. The All-Sky images capture a 180° hemispheric view of the sky from its position, often referred to a fish-eye view, as shown in the image below.

Figure21: All-sky image at Skywatch Obsevatory
Source: https://skywatch.colorado.edu/php/allsky.php

To achieve this 180° field of view, a camera with a specific fish-eye lens is needed. A low-cost, high-technology commercial IP camera called MOBOTIX Q71 was used to obtain the images for this project, as the compact-size camera's hemispheric all-round feature was suitable for the photoshoot.

This versatile camera is also capable of distinguishing colors at night as it equipped with combined infrared and LED white light. Its wide dynamic range feature produces a clear image at a high resolution of 12 megapixels, even under light exposure. The IP rating 66 made it to be appropriate for the outdoor use, with its operating temperature being between -40 and 65°C. Additionally, its intelligent video analysis feature and robust mobile app connectivity make it an excellent security camera.

Figure22: MOBOTIX Q71 Hemispheric
Source: https://www.mobotix.com/en/Q71

The goal of this project is to study the effect of SFV. As SFV is calculated from all-sky images, the fundamental step is to take the photos. For these images to have distinct SFV, they were taken throughout different parking spots of the IES buildings. Using MOBOTIX Q71, photos were taken in the front, rear, and side parking lots of the IES building, as marked in the figure below.

Figure23: IES parking lots – front, rear, side
Source: https://www.google.com/maps/

Each photo was taken in the center position of a parking spot, as shown in photo below. The IP camera was directly connected to a laptop and was powered by power-over-ethernet adapter with the help of extension power cords.

Figure24: parking spot photo capture

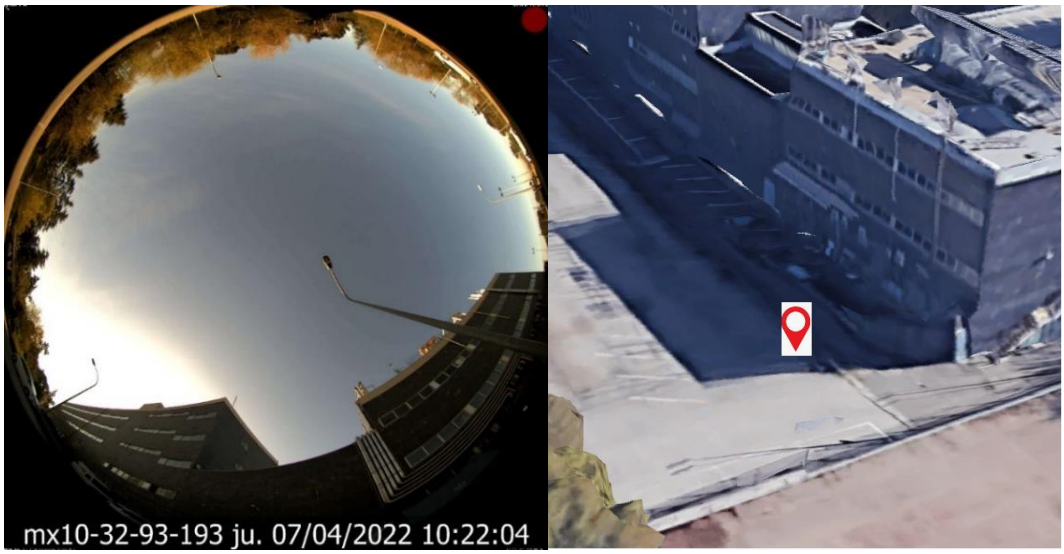


Figure25: MOBOTIX Q71 all-sky image sample

Figure25 is an example of 180° panoramic photo captured by MOBOTIX Q71. The location of this photo is marked as a red pin. A generous amount of the photo is covered by obstacles such as buildings, lamp posts and trees – confirming this project's key assumption. That is, the presence of obstacles within urban environment causes non-uniform amount of visible sky in the hemispheric view.

A few photos were captured for each spot for having the choice to select. After going through all designated parking spots, a total of 113 photos were taken. These photos were spatially identified in order as shown below.

Figure26: parking spot photo number identification map

3.2. OpenCV, Python - SFV

The obtained images now have distinct view of the sky, and the amount of visible sky can be estimated by scanning through them and giving a rough number. For instance, *Figure25(left)*, to me, seems to have 70% of its hemispheric view covered in sky (SFV = 0.70). Even though this crude method might work for a few images, human eyes cannot be on par with a computer program. This is where the image-processing comes into play, where running a detection algorithm will produce much better results.

The purpose of processing the images is to detect the sky regions more precisely, and to compute an accurate SFV value for each image. Because SFV is within a range of values between 0% and 100%, this section has a single goal of computing a coefficient between 0 and 1, for each image. In this project, the image processing was performed using Python and an open-source library called “OpenCV[8].”

Figure27: The library logo - OpenCV
Source: https://opencv.org/

To humans eyes, phots can be identified by the objects and colors in them; the resolution is determined depending on how blurry or sharp they appear to be. For the computers to identify and process images, they need to be read-in as matrices, where each matrix contains an information of a specific pixel. OpenCV loaded each image and converted it such that it could be analyzed in a digital form, represented by matrices of pixels.

Pixel, or picture element, is the smallest element of an image. Because each pixel is a sample of an image, higher number of pixels means its more accurate and clearer representation.

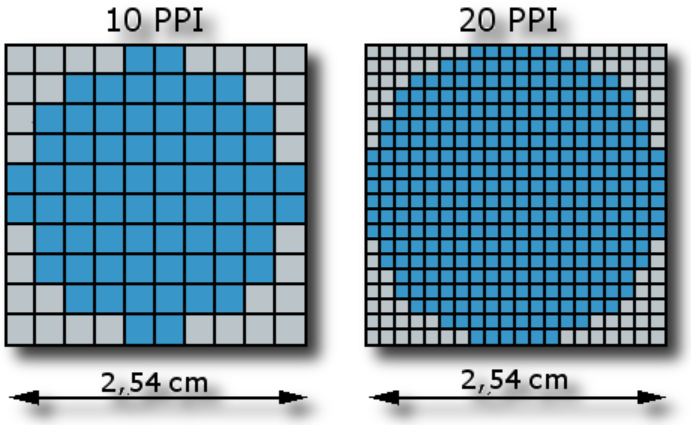


Figure28: pixel per inch comparison
Source: https://www.androidguys.com/tips-tools/what-is-ppi-and-why-is-it-important/

Cameras have a built-in sensor whose size is defined by the numbers of rows and columns of the matrix. The resolution of the image that fits within a preset matrix is defined by how many pixels can fit in a uniform space, also known as pixels per inch (PPI). For instance, the MOBOTIX Q71 took photos with a frame of 2880 pixels in both width and height, with a resolution of 96 PPI.

The digitalized matrices represent the image in x-y coordinate, where the origin (0,0) by default represents the upper-left corner pixel.

Digitalized pixels contain useful information about the color. By analyzing the pixel color, modern technology allows complex and intricate image processing such as object detection, facial recognition, and so forth. The two main color models that most image-processing libraries work with are RGB (Red, Green, Blue) and HSV (Hue, Saturation, Value).

RGB is a color model with three dimensions: red, green and blue. These three colors are mixed and produce a specific color. This color model is often depicted in a 3-D space in x-y-z axis. Because humans do not think about colors as a mixture of primary colors, this model might not be the most intuitive one in creating colors in code.

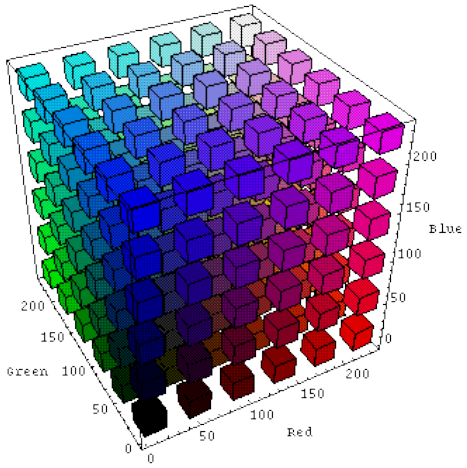


Figure29: 3-D representation of RGB color space
Source: https://engineering.purdue.edu/~abe305/HTMLS/rgbspace.htm

In computer program, images read with RGB color scheme will contain 3 channels for each color. Each color will have between 0 and 255. This is because each channel is 8 bits, thus 255 is the limit. For instance, (255, 0, 0) would represent red and (0, 255, 0) would represent green.

HSV is a color model in a cylindrical form which maps the primary colors into different dimensions, that are more intuitive for humans to imagine. For this color model, the three channels that computer program reads are Hue, Saturation and Value. For HSV, all three channels are interdependent.

- Hue: angle on primary color circle (i.e., 0 = red, 120 = green, 240 = blue)
- Saturation: amount of color used (i.e., 100% = purest color, 0% = grayscale)
- Value: brightness of the color (i.e., 100% = no black, 0% = pure black)

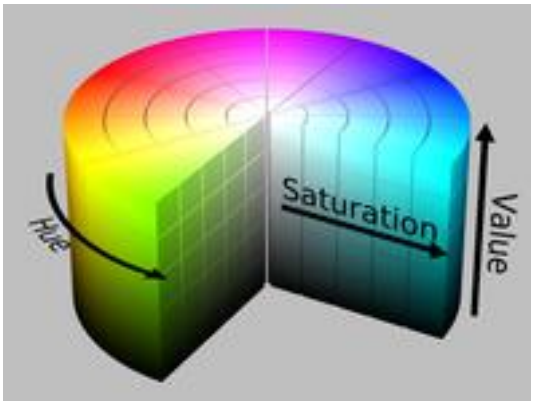


Figure30: 3-D representation of HSV color space
Source: https://en.wikipedia.org/wiki/HSL_and_HSV

Before importing the images to Python, there needs to be a modification to their physical form. MOBOTIX Q71 uses a fish-eye lens to take the 180° panoramic photos, and an optical distortion occurs during this process, namely, barrel distortion. Barrel distortion is when the straight lines appear as convex curves, and is common when wide angle lens is used. This type of distortion happens when the field of view of the lens is wider than the image sensor of the camera, causing the image to be squeezed to fit the camera's frame. With this distortion, the pixel size throughout the image would be uneven - the pixels in the center appear to be bigger than those that are close to the corners as shown below.

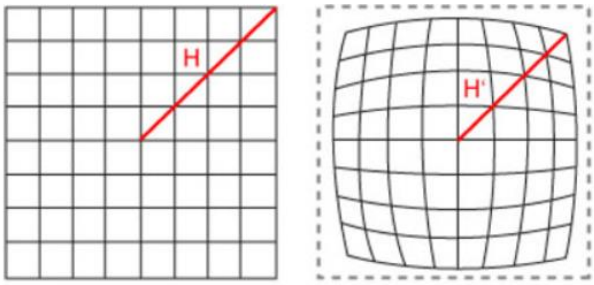


Figure31: undistorted pixels (left) vs pixels with barrel distortion (right)
Source:
http://michel.thoby.free.fr/Fisheye_history_short/International_Standards_about_Distortion.html

To undistort a distorted image, camera's optical information needs to be found. These parameters include intrinsic size of the image sensor and the radial distortion. Here, the conventional checker box method was used, where a few images of a checker box were taken with MOBOTIX Q71, and MATLAB's built-in function loaded those images and calculated the camera's parameters with input square size.

Figure32: checker board for camera parameter calculation

The obtained camera's intrinsic matrix and radial distortion were assigned as a vector called "cameraParams," and MATLAB's "undistortImage" function was used in a loop to remove the barrel distortion and flatten all images.

An example of undistorted image is shown below. In this image (right), each pixel contains information of equally sized section of the hemispheric view.

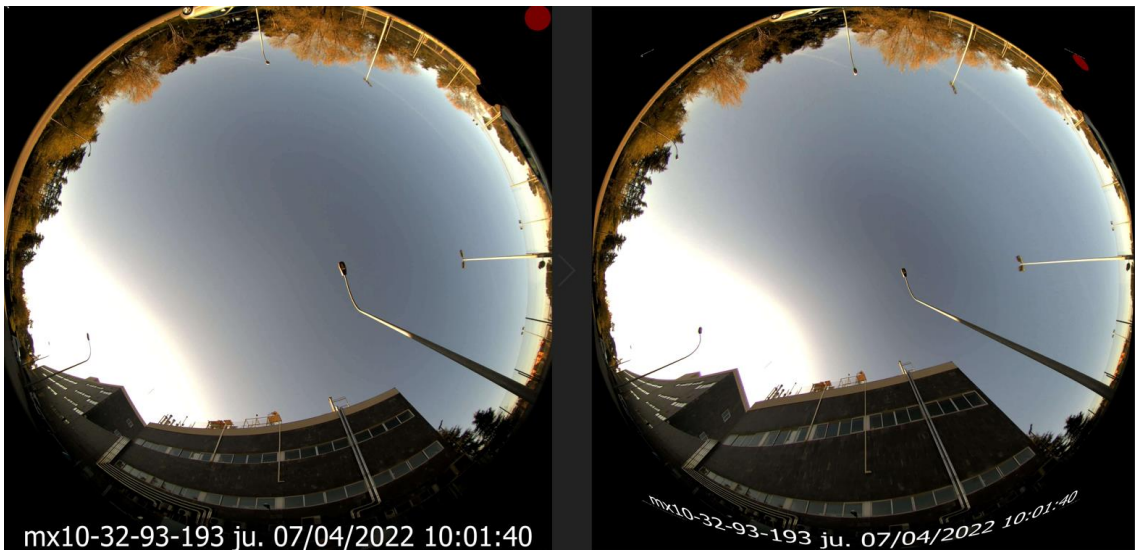


Figure33: original image (left) vs undistorted image (right)

The first step in processing the undistorted images was to rescale them. MOBOTIX Q71 has an image resolution of 2880 x 2880 pixels for hemispheric view, but undistorting the images increased the resolution up to 4647 x 4645 pixels. As this size could not fit on the laptop screen, the images were rescaled by a factor of 0.15, to a frame of 697 x 696 pixels.

Then the circular mask was created using the radius of all images. This filter was used to black out all pixels outside of the circular frame of MOBOTIX Q71.

In order to offset the cutting-off on the edges of the camera's intrinsic matrix, 10 pixels were added to the radius, half of the number of pixels in x-axis. The radius of the circular mask was defined to be 358 pixels.

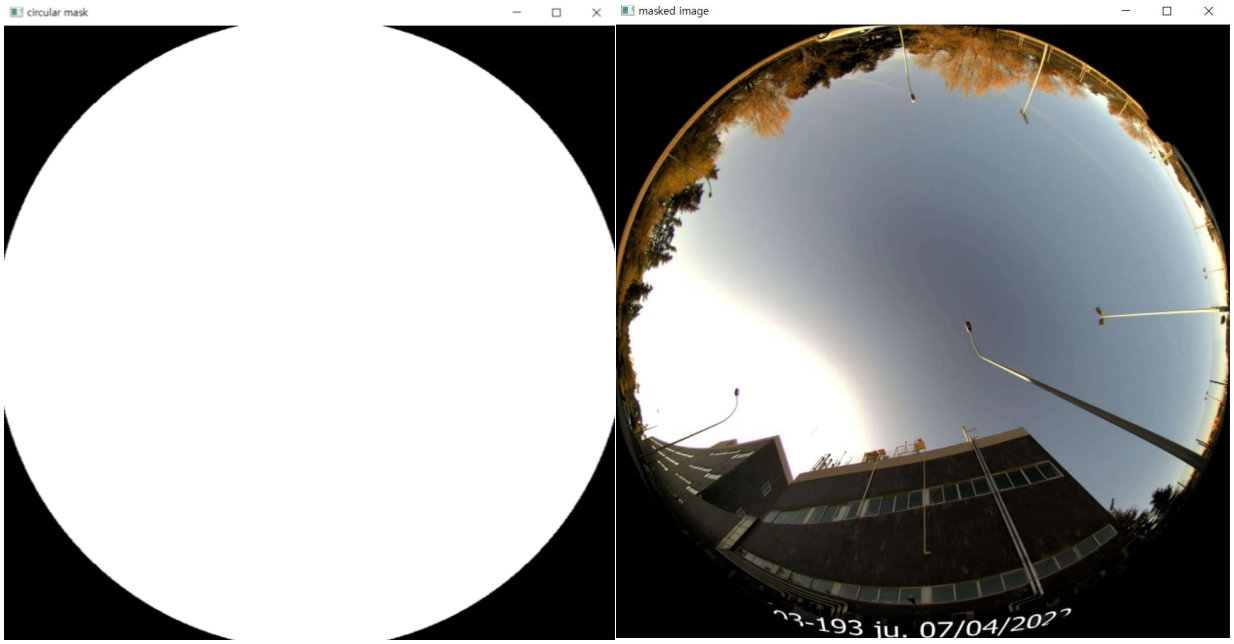


Figure34: circular mask(left), applied to undistorted image(right)

Once the circular mask was applied to all undistorted 113 photos, multiple image processing methods were used to distinguish the visible sky from the images.

Each method presented distinct problems - because the photos were taken under different weather condition, throughout different time period, and location – captured images contained varying degrees of the clouds, obstacles, and light reflections.

Figure35: examples of intense light reflection on the buildings

Figure35 demonstrates the biggest challenge faced in this project.

On the left photo, the reflected light on the buildings creates white glare that is near-identical to the light in the sky. The bottom circle glare can be fixed through contour detection with RGM as it is outside of the detectable contour. However, the top circle glare is directly touching the edges which act as a contour separating the sky from the obstacles. Therefore, a part of this glare would be included in the contour until the glare is no longer present on the building's surface.

On the right photo, an excessive amount of white glare is present the building is simply indistinguishable from the sky. Images like this were not used, as they do not contain enough information about the pixels. The image below is an example of a discarded image, as a part of building is detected as sky.

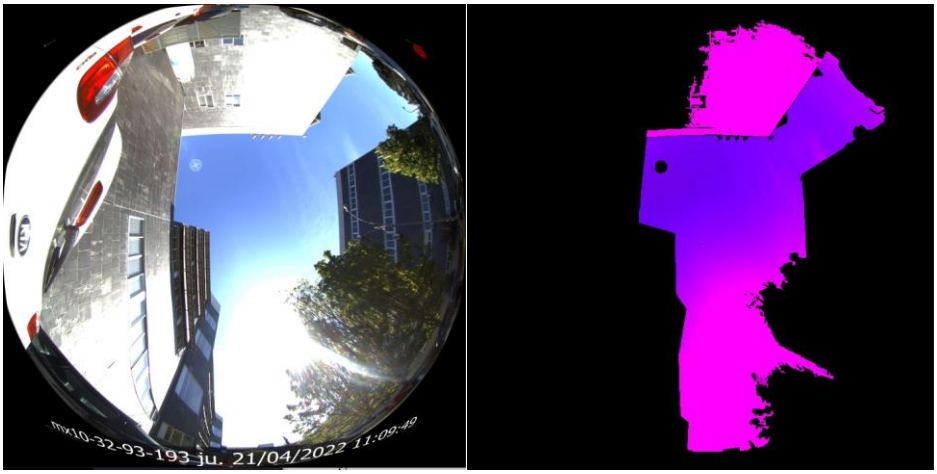


Figure36: erroneous detection of sky region due to intense light reflection

Method 1 – cv2.inRange() detection via Blue, Green, Red

The most intuitive method for detecting sky regions was to distinguish what appeared to be "blue-ish" color of the sky in the image. As explained in *Section 3.2.*, computer loads an image as matrices of pixels, where each channel represents a value of three primary color – B, G, R. This method can be implemented by telling the computer to scan through every pixel in the image, and assign pixels that show the sky color. Then, only those pixels assigned as sky color can be displayed, while the unassigned pixels can be blacked out.

OpenCV's built-in cv2.inRange() method facilitates this process, as it allows the user to set the threshold limits for each channel for color detection. For example, a variable "blue" can be assigned as [blue = cv2.inRange(image, 125, 255)]. This means that every pixel of "image" is scanned, and the pixels with a blue value between 125 and 255 are saved in a matrix named "blue." By fine-tuning these threshold values for each color channel, the first sky detection method was written.

Though this method worked fairly well, it often had erroneous detection of the sky.

Figure37: Method 1 erroneous color detection, Python

From *Figure33*, the frames of the windows on the building appear to be bluish-white. This color is quite indistinguishable from the color of the sky in this particular image, and therefore, this detection method would result erroneous, where the windows are mistakenly marked as sky as shown in *Figure37*.

The erroneously detected area was relatively miniscule to the size of the image size, and this method was used for filtering images with presence of numerous gaps, mainly caused by tree branches. This is because this method examined every pixel in the image, and did not exclude any regions.

Method 2 – Region Growth Method

Another robust method used for the detection of sky is region-growing methods (RGM). This method chooses a seed pixel, where the detection begins, and starts expanding the region, in all possible directions. To be able to compare, the image needs to be converted to a grayscale format. Given a user's input threshold value for growth criteria, the seed pixel will compare its grayscale value with its neighboring pixels, and the region will continue to expand until the difference in the grayscale values exceeds the threshold value.

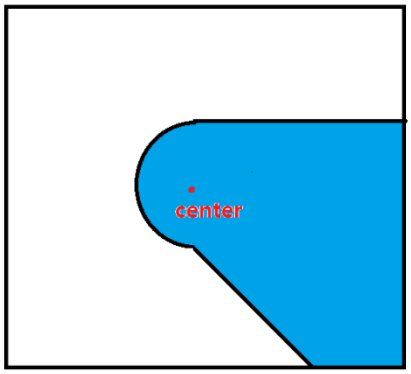


Figure38: Region Growth Method sample

In *Figure38*, performing a RGM with a center pixel as a seed pixel, the region will expand, until the pixels reach the black contour. This is because the grayscale difference between a white pixel and a black pixel (or any two pixels whose grayscale difference exceed the threshold value) will not satisfy the growth criteria. Therefore, the final "sky region" in this example, would be a region enclosed by the contour, in which the center pixel is included.

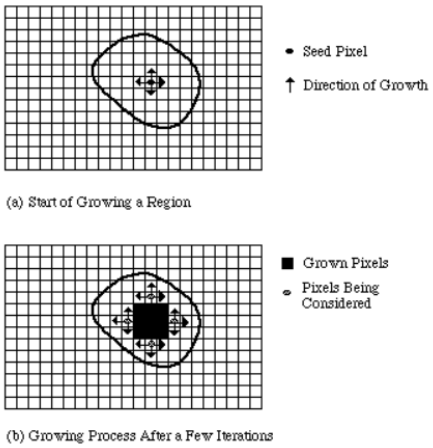


Figure39: Region Growth Method
Source: https://users.cs.cf.ac.uk/Dave.Marshall/Vision_lecture/node35.html

This method was more robust and was used for most images, as it focuses on detecting the main contour of the image. Because this method was comparing the grayscale value of pixels, rather than the color values, the sky-color detection was not as correct. However, this method created better filters when the image had clear edges separating its single, main sky region. Additionally, it excluded all other erroneously detected parts outside of the detected region.

Figure40: Method 2, Region Growth Method filter example

The downside of this method was that it could not detect multiple sky regions in the image because it would black-out everything outside of the main region. Setting the seed pixels all over the image would result in erroneous detection as all existing contours would be marked as sky, or it would simply mark the entire image as sky if the contour is not fully enclosed by defined edges. A number of the images had intense light reflection on the buildings, and disturbed the enclosure of the contour and resulted problematic. In this case, method 1 was used for the filter.

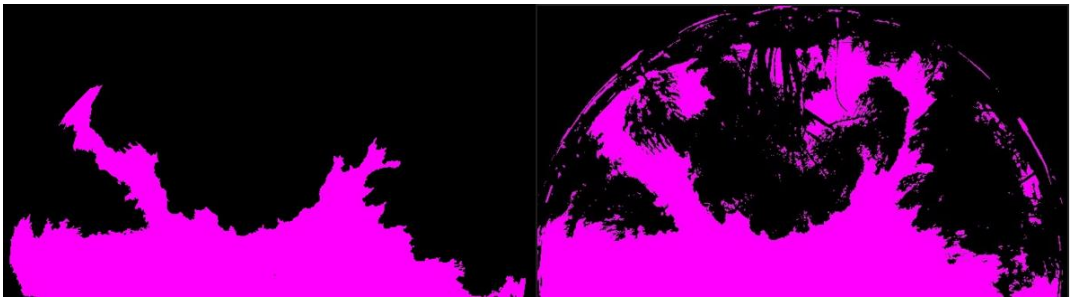


Figure41: Method 2(left) vs Method 1(right) on image with multiple regions

For the case where image did not have a main sky region enclosed by defined edges, RGM was applied twice - first to the original images, and then to the sky filter generated by Method 1. Sky filters only used black and white pixels, and contained better defined edges. This way, it could detect the main sky region as shown below.

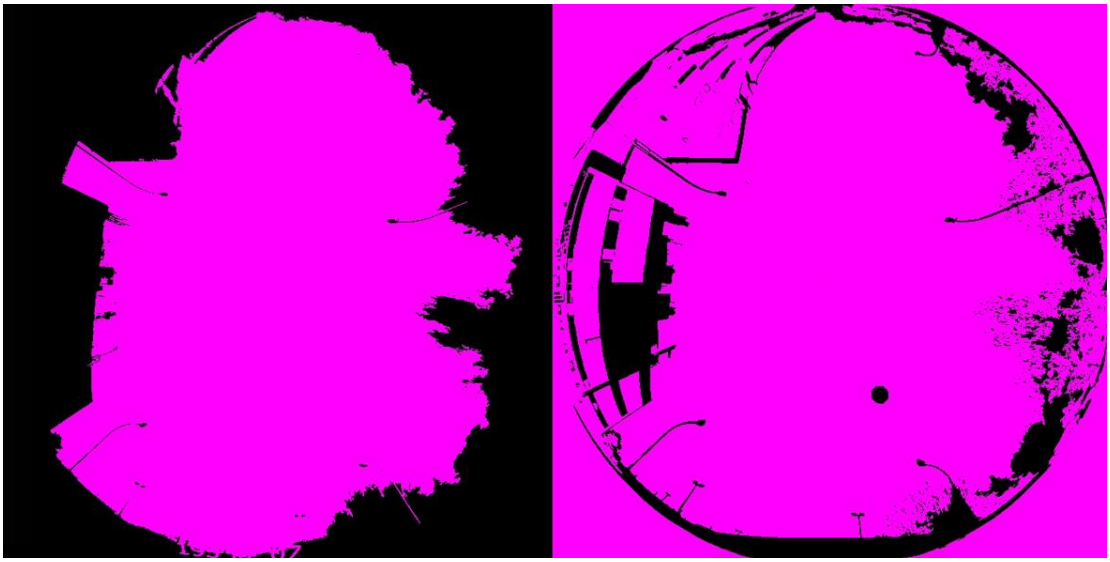


Figure42: RGM on sky mask (left) vs RGM on original image (right)

Using these methods (RGB sky filter, RGM on sky filter, RGM on original image), the final filtered images were created by masking the corresponding filter over the original image, as shown in *Figure43* below.

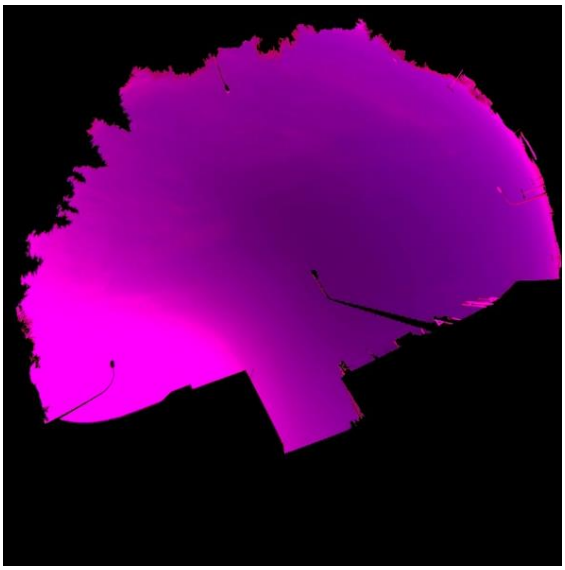


Figure43: sample of a filtered image, RGM on original image

The filters created by distinct detection methods are visual representations to show the separation between sky and obstacles. The last step in the image-processing procedure is to calculate the SFV.

Filters are represented by matrices which either call the pixels “on” or “off” by marking them either black or non-black (magenta, in examples above). Recalling the definition of SFV, it is a mere ratio of the amount of visible sky to the hemispheric frame. The amount of visible sky is represented by the number of non-black pixels, while the hemispheric frame contains a constant number of 398324 pixels.

OpenCV’s “countNonZero” function facilitates the calculation of the number of non-black pixels in a grayscale format image, and finally, dividing the number of non-zero pixels by 398324 results in SFV value. For instance, the image in *Figure38* had a SFV of 66.53%. Looping through all images, SFV values were calculated and saved - as they will be needed when modeling the SFV sky diffuse model in PVLIB simulation.

3.3. PVLIB, Python – PV generation

The final procedure of this project is to simulate the VIPV system using PVLIB, and estimating the solar resource and PV production. The simulation will be done twice: one time with the Isotropic sky diffuse model, and another with the SFV model. An open-source library PVLIB, whose workings methodically explained in *Section 2.4.*, was implemented within Python.

The simulation was run for all year, and a specific day of the year – as the latter will be used for the visual analysis of the solar resource. For a specific day of the year, a typical day in the middle of the year (June 15th) was selected as sufficient solar radiation is received around this peak generation period; additionally, it is assumed that the shadowing is not present during this day. Therefore, an image with no shadow is to be chosen for the PV simulation and result analysis.

Section 2.4. explains that the general procedure in the PVLIB simulation goes as the following:

1) gather weather data (TMY)
2) calculate solar position and AOI
3) calculate POA irradiance
4) choose a module and an inverter
5) calculate module temperature and effective irradiance
6) calculate module performance (I-V curve)
7) estimate PV production at inverter's maximum power point.

Recalling, each image is a representation of two sets of information vital for the simulation - a unique geographical coordinate, and a unique SFV value.

- A geographical coordinate of the image is linked to the gathering of TMY weather information (GHI, DNI, DHI…).
- A SFV value is used when implementing the SFV diffuse model for POA calculation.

According to the 7 steps provided above, each image goes through the following:

- An image is loaded and provides a coordinate information, which then PVLIB uses to gather hourly TMY data - exactly where the photo was taken.
- Using coordinate information and TMY data, PVLIB calculates hourly solar position and AOI (*here, inclination angle is assumed to be 0° – module is mounted on vehicle's roof)*.
- Hourly POA components (direct and diffuse) are calculated using TMY data, solar position and the chosen sky diffuse model.

  As the inclination angle is 0°, isotropic POA diffuse irradiance is equal to DHI.

$$G_{POA,Diffuse,Isotropic} = DHI * \frac{(1 + \cos(\beta))}{2} = DHI$$

$$G_{POA,Diffuse,SFV} = DHI * SFV$$

  While the sky diffuse model is set to 'isotropic,' first simulation is run. The second simulation is run after scaling the DHI by the image's SFV value.

- To estimate the PV generation, a set of commercially available module and inverter needs to be chosen. PVLIB provides a library of modules and inverters, and the following equipment was selected for the simulation:

  Module: Canadian Solar CS5P-220M (220W) Solar Panel
  Inverter: ABB MICRO-0.25-I-OUTD-US-208/240 250W MICROINVERTER

Figure44: Canadian Solar CS5P-220M (left), ABB MICRO-0.25-I-OUTD-US-208/240 (right)

- PVLIB calculates hourly module temperature, and effective irradiance via SAPM method. These two parameters are what the VIPV system is experiencing – the actual temperature of the module for being exposed in the parking spot, and the amount of solar radiation the module is receiving.
- PVLIB simulates SAPM to generate the module's hourly I-V curve using effective irradiance, module temperature and module information. The module's adjusted performance will vary its maximum output throughout the day.
- Finally, the module's performance (DC) is converted to the PV generation (AC) via chosen inverter's specification. The sum of hourly generation represents the total generation during a chosen range of time/date.

4. Results

After sorting out the images and running them through SFV calculations, a total of 86 distinct values were computed – each representing the number of obstacles within sky-view throughout the IES parking structures. These values can be found in *Appendix C*, where the index number of each image is mapped to *Figure26*.

The solar resource will be analyzed using just one parking spot; however, it is important to visualize the varying degree of SFV throughout the IES parking lots. For better representation of the SFV results, a heatmap was created using all obtained SFV values.

To create a heatmap, corresponding SFV value was assigned to each parking spot. This means, 86 SFV values were assigned to 86 distinct pixel coordinates of the Google Maps screenshot of IES parking lots. However, 86 values are not enough to cover the countless pixels of the captured entire IES parking lots.

Using only 86 data points, all other pixels in the 2-d grid with no input value were linearly interpolated. This way, the 2-d grid was completely filled with estimated values of SFV. Depending on the value of SFV, each pixel was filled with a distinct color, specified by the color bar.
Finally, the pixels representing buildings were zeroed out and marked as gray, as they are obstacles and irrelevant to this heatmap.

Figure45: IES parking lots SFV heatmap

The SFV heatmap is shown in the figure above. As the color bar indicates, the SFV value is high in red area while low in blue.

The map appears to be more-blue around the side parking lots. This is a reasonable result, as the parking spots are surrounded by the buildings and trees in this section. Also, this area would not be a desirable parking space for electric vehicles given the shadowing effect caused by the building.
The front parking lots are surrounded by buildings and trees as well, although the heatmap shows a mixture low and high SFV, and the hemispheric view seems to clear out toward the IES's entrance.

SFV values tend to be higher in the rear parking lots as many spots are exposed to the solar radiation without many obstacles surrounding them. This would be a better choice of parking for electric vehicles, as the following will show the relationship between SFV and VIPV performance with a chosen parking spot.

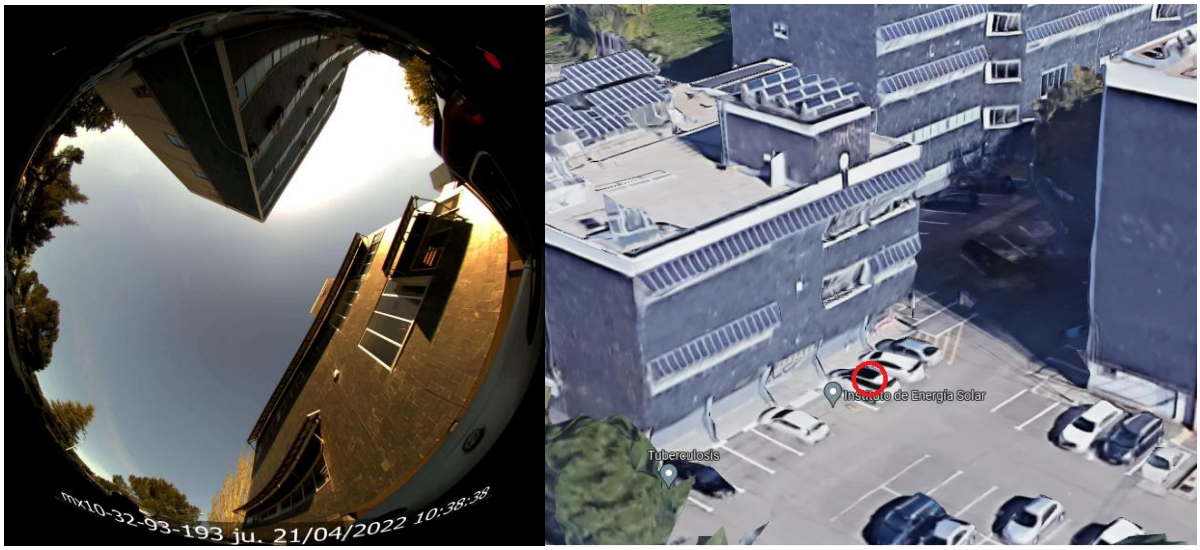


Figure46: parking_undistorted-72.jpg

In order to highlight the discrepancy in solar resource and PV generation between the two diffuse models, a parking spot with many obstacles and no shadow on June 15th was chosen for the simulation and for the analysis – "parking_undistorted-72.jpg", as shown in *Figure46* above. The undistorted image # 72 and its sky region filter are shown in the figure below.

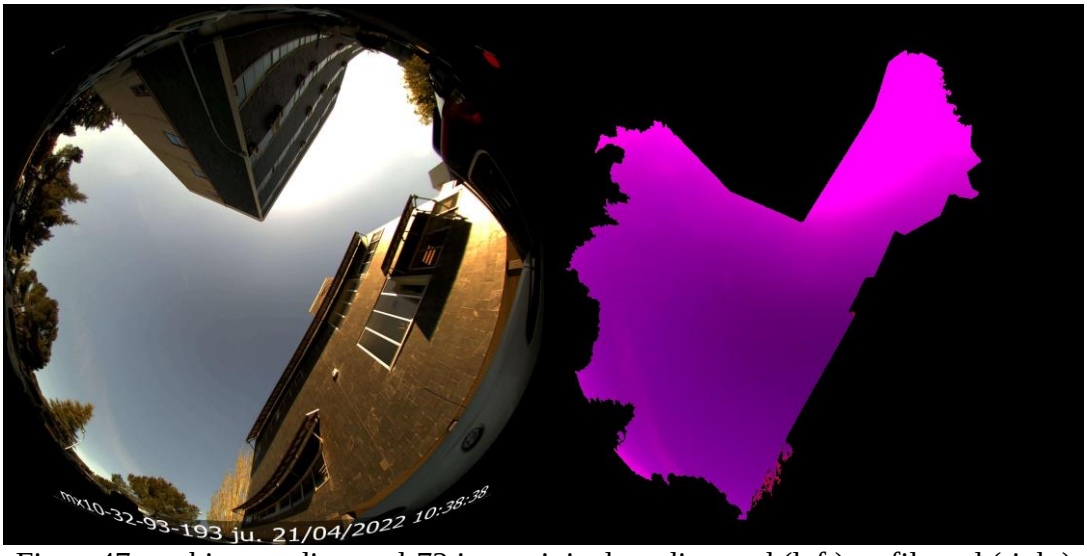


Figure47: parking_undistorted-72.jpg: original_undistorted (left) vs filtered (right)

In *Figure47*, RGM was used to detect a single sky contour and create a filter. It captures the sky region accurately, computing a SFV of 41.65%.

To observe the effect of SFV on the solar resource, POA diffuse irradiance was compared between the two models – Isotropic and SFV. The time-axis in all figures is truncated from 04:00 to 20:00 since the sunlight is not present outside this range in Madrid on June 15th.
After applying the SFV diffuse model, POA Diffuse curve is reduced by a factor of 0.58, as less than half of the image contained sky region. While the POA Direct curve maintained constant, the global curve shifted downward by the amount equal to the reduction in the diffuse component – as global is the sum of direct and diffuse.

The following figures show hourly reduction in the POA irradiance and VIPV system production.
[*coordinate = (40.4531, -3.7269), "June 15th]*

	ISOTROPIC DIFFUSE MODEL				SFV DIFFUSE MODEL				
time	poa_global	poa_direct	poa_diffuse	production (Wh)	poa_global	poa_direct	poa_diffuse	production (Wh)	%Δ poa_global
4:00	0.00	0.00	0.00	-0.08	0.00	0.00	0.00	-0.08	0.00
5:00	18.00	0.00	18.00	-0.08	7.50	0.00	7.50	-0.08	-58.35
6:00	175.36	101.36	74.00	27.40	132.18	101.36	30.82	17.85	-24.62
7:00	362.33	254.33	108.00	68.45	299.31	254.33	44.98	55.00	-17.39
8:00	549.65	413.65	136.00	108.01	470.30	413.65	56.64	92.17	-14.44
9:00	721.56	569.56	152.00	140.04	632.87	569.56	63.31	123.54	-12.29
10:00	865.61	710.61	155.00	164.21	775.16	710.61	64.56	148.43	-10.45
11:00	964.57	814.57	150.00	178.60	877.04	814.57	62.48	164.06	-9.07
12:00	999.80	839.80	160.00	181.99	906.44	839.80	66.64	166.86	-9.34
13:00	1003.48	882.48	121.00	181.47	932.88	882.48	50.40	170.23	-7.04
14:00	914.83	767.83	147.00	166.75	829.06	767.83	61.23	152.73	-9.38
15:00	794.30	661.30	133.00	147.67	716.69	661.30	55.39	134.27	-9.77
16:00	625.92	497.92	128.00	118.28	551.23	497.92	53.31	104.43	-11.93
17:00	433.57	321.57	112.00	81.08	368.22	321.57	46.65	68.03	-15.07
18:00	237.99	157.99	80.00	40.00	191.31	157.99	33.32	30.10	-19.61
19:00	57.72	28.72	29.00	5.31	40.79	28.72	12.08	1.98	-29.32
20:00	0.00	0.00	0.00	-0.08	0.00	0.00	0.00	-0.08	0.00
				1609.04				1429.48	

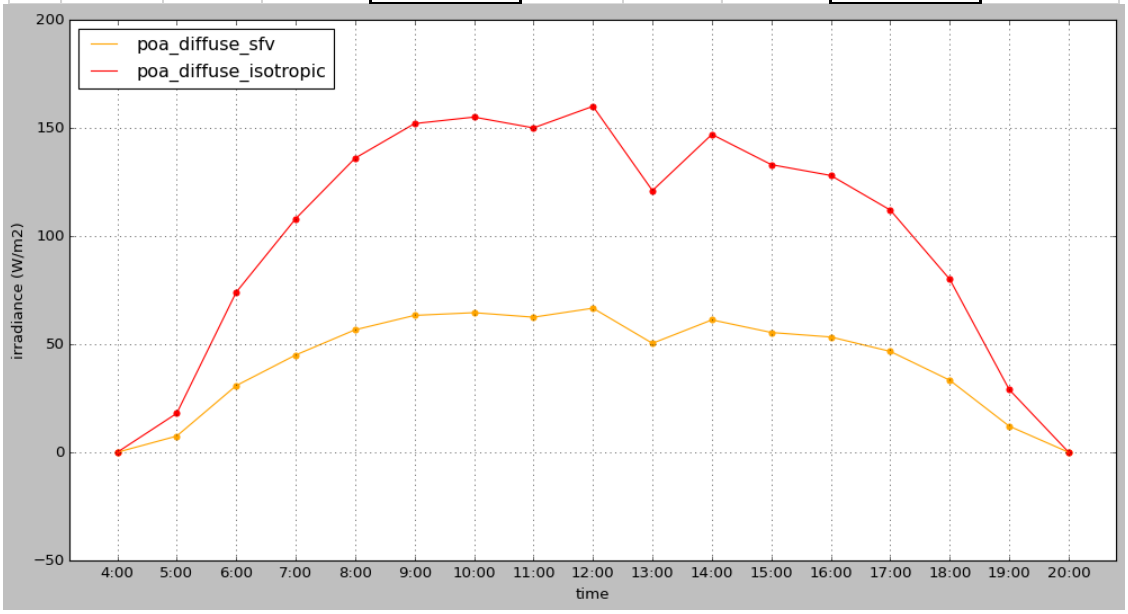


Figure 48: POA diffuse irradiance - Isotropic vs SFV diffuse model

Figure48 illustrates a drop in POA diffuse irradiance when the SFV factor is applied to the DHI. As a result, the diffuse component is reduced linearly for each hour (by 58%).
Between 8:00 and 16:00, where the curve is more stable, the global irradiance is reduced typically by 10% with the SFV model. Around noon, the POA diffuse reduced the most by about 100W/m2. In other words, covering 58% of the hemispheric sky view with obstacles can reduce the useful irradiance by about 10% during the more productive hours.

As the PVLIB calculates the module's effective irradiance using this information, the subsequent PV production will be reduced accordingly. The following plot compares the hourly production using the module "Canadian Solar CS5P-220M" and the inverter "ABB MICRO-0.25-I-OUTD-US-208/240" for both diffuse models (Isotropic and SFV).

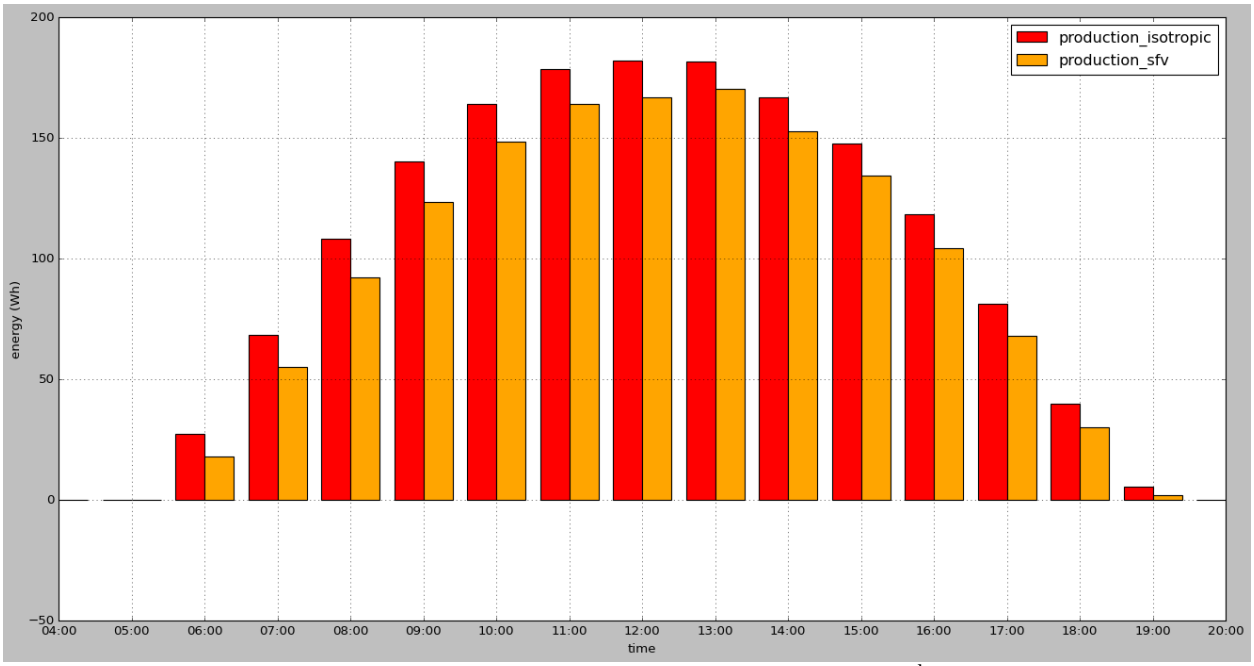


Figure 49: Hourly energy production - parking spot #72, June 15th - Isotropic vs SFV

Figure49 shows hourly production in energy, under the setup where an electric vehicle with a module on its roof is parked at coordinate [40.4531, -3.7269]. The difference between Isotropic and SFV model is around 10 to 20Wh each hour, adding up to about 200Wh, or by 11.13%. This is a considerable decrease for a single day, and would accumulate to a huge amount over the entire year.

The following table displays the production loss, both on June 15th and all year, for parking spot #72.

June 15th	**PV production - Isotropic**	**PV production - SFV(41.65%)**	**% reduction**
parking_undistorted-72.jpg	1608.51 Wh	1429.48 Wh	-11.13
All Year	**PV production - Isotropic**	**PV production - SFV(41.65%)**	**% reduction**
parking_undistorted-72.jpg	339407.28 Wh	275253.89 Wh	-18.90

Figure 50: Total PV production - parking spot #72, June 15th / All Year

From *Figure50,* the PV production was relatively reduced more for all year. This means that the effect of SFV has been more relevant during winter or any other months with diffuse-prone cloudy weather, than it has been in June. During those months, the diffuse component occupied higher fraction of the global irradiance. This is intuitive, as Madrid has clearer sky with abundant direct radiation during June, or summer.

Considering the sheer amount of PV production loss from implementing the SFV diffuse model, it can be said that the IES parking lots is a suitable representation of the proposed urban environment for this project. After all, having obstacles covering the sky view of the module, or 41.65% SFV, substantially reduces the yearly PV production by almost 20%.

Although this calculation cannot be applied to every parking spot – as the limited scope of this project requires shadowless location – majority of the front and rear parking lots are adequate for this PV simulation. And again, side parking lots needs to be avoided by VIPV system – not only for low SFV values, but also for shadowing.

6. Conclusions

This project aimed to study the SFV and its connection to the performance of a PV system - specifically for VIPV system. The purpose of this project was to show that VIPV systems experience fluctuating solar resource and PV generation, as the vehicles move around the urban environment – affecting the diffuse component of the solar radiation. The project focused on demonstrating the effect of an implementation of proposed SFV model within urban environment, where the upright hemispheric view of the sky is usually covered by obstacles. The solar resource and energy production of distinct locations were calculated and compared between the two sky diffuse models: Isotropic and SFV.

Utilizing a commercial camera MOBOTIX Q71 was such a cost-effective, convenient way of capturing all-sky images throughout the different parking spots as they were necessary for the SFV calculations. A bit of difficulty involved requiring the camera to be connected to a laptop via an Ethernet cable, while moving around the parking lots with extension power cords. Additionally, the radial distortion and the inability to remove the time marks resulted problematic.

The image processing was quite a challenge, and ended up having to discard images that were not adequate for the SFV calculation. Not only the camera did not have options to change parameters to reduce the light reflection/glare, the majority of the images were taken on a bright day when the sun was on top of the camera. For future reference, it would be a better idea to take them on a cloudy day, to remove the glares and facilitate the sky region detection. Nonetheless, sufficient number of images were properly processed and generated necessary SFVs, as well as an SFV-heatmap of the IES parking lots, for the further analysis.

The TMY and POA irradiance methods were well implemented by the PV pioneers in PVLIB library, and it was astonishing how simple, practical and robust these programs were written. The number of available solar models and PV equipment covered all the needs for this project and the procedure did not present any problems.

With the final calculations of energy production of distinct parking spots, this project successfully distinguished a noticeable difference between the two sky diffuse models.

Upon implementing the SFV model, POA diffuse irradiance is scaled by a factor of a SFV percentage, and - despite its small fraction in the total irradiance - this makes a huge difference over the year-long production. For a VIPV system in the parking structures of IES, the yearly production can be substantially reduced by about 20%, depending on where the electric vehicle is parked. Also, the results suggest that SFV model becomes more impactful during the cloudy months, around winter, when the diffuse component is more present within the global irradiance. It can be concluded that SFV diffuse model is highly relevant for VIPV applications, which exist in the urban environments.

As PV integrated electric vehicles move around the urban environments filled with obstacles, they will continue to experience such phenomenon. Finally, this suggests a potential problem in assuming all-sky hemispheric view when computing diffuse radiation for PV systems that can experience varying-SFV (i.e. VIPV.) - as this would overestimate the energy production.

With a more refined, comprehensive SFV diffuse model for VIPV, an interesting future project would be integrating an all-sky camera in the vehicle so that it displays instantaneous SFV, or perhaps a warning to park the vehicle elsewhere. Also, developing a shadowing-corrected DNI modeling would be useful as it would allow the PV simulation in all parking spots, shadowed or not.

7. References

[1] Mertens, K. (2018). Photovoltaics: Fundamentals, technology, and practice (2nd ed.). Standards Information Network.

[2] Letcher, T. M., & Fthenakis, V. M. (Eds.). (2018). A comprehensive guide to solar energy systems: With special focus on photovoltaic systems. Academic Press.

[3] Jahne, B. (2013). Digital image processing: Concepts, algorithms and scientific applications (2nd ed.). Springer.

[4] V. Badescu, 3D isotropic approximation for solar diffuse irradiance on tilted surfaces, Renew. Energy 26 (2002) 221e233, https://doi.org/10.1016/S0960-1481(01)00123-9.

[5] G. Ignacio, Diffuse irradiance on tilted planes in urban environments: Evaluation of models modified with sky and circumsolar view factors, Renewable Energy 180 (2021) 1194-1209, https://doi.org/10.1016/j.renene.2021.08.042.

[6] S. Xixi, Worldwide performance assessment of 95 direct and diffuse clear-sky irradiance models using principal component analysis, Renewable and Sustainable Energy Reviews Volume 135, January 2021, 110087, https://doi.org/10.1016/j.rser.2020.110087

[7] C. Miguel, Urban solar potential for vehicle integrated photovoltaics, Transportation Research Part D: Transport and Environment (Volume 94), May 2021, 102810, https://doi.org/10.1016/j.trd.2021.102810.

[8] Documentation, OpenCV: https://docs.opencv.org/4.x/

[9] Documentation, PVLIB: https://pvlib-python.readthedocs.io/en/stable/

8. Appendix

Appendix A. Project Code – OpenCV

```
WHITE_RGB = (255, 255, 255)
MAGENTA_RGB = (255, 0, 255)
NONZERO_CIRCLE = 398324

IMAGE_PATH = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_original_undistorted'
RESULT_PATH = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\results'
CSV_PATH = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos'

FILTER_PATH = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filter\v1'
FILTERED_PATH = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filtered\v1'

FILTER_PATH_V2 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filter\v2'
FILTERED_PATH_V2 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filtered\v2'

FILTER_PATH_V3 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filter\v3'
FILTERED_PATH_V3 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filtered\v3'

FILTER_PATH_V4 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filter\v4'
FILTERED_PATH_V4 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filtered\v4'

FILTER_PATH_V5 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filter\v5'
FILTERED_PATH_V5 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filtered\v5'

FILTER_PATH_V6 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filter\v6'
FILTERED_PATH_V6 = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_filtered\v6'

################################################################################################
"FOR FUNCTIONS USED IN SFV CALCULATION"

from cv2 import Mat
import numpy as np
import cv2
import os
from constants import WHITE_RGB, MAGENTA_RGB, NONZERO_CIRCLE

def load_images(folder_path: str) -> list:
    '''
    load images from a folder
    '''

    images = []

    for file in range(len(os.listdir(folder_path))):

        image = os.listdir(folder_path)[file]
```

```
        images.append(image)    # appending filename to the list images

    return images


def rescale_image(image: cv2.Mat, percent: int = 15) -> cv2.Mat:
    '''
    rescale the image by a factor of input percent
    '''

    width = int(image.shape[0] * percent / 100)
    height = int(image.shape[1] * percent / 100)
    dim = (width, height)

    rescaled_image = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)

    return rescaled_image


def circle_radius(image: cv2.Mat) -> int:
    '''
    calculate the radius of input image
    '''

    # IP camera cuts off square by a small margin (+10)
    radius = int((image.shape[0]/2) + 10)

    return radius


def mask_circle(image: cv2.Mat, circle_radius: int) -> cv2.Mat:
    '''
    create a white circle with the photo radius
    '''

    mask = image.copy()
    X, Y, _ = mask.shape
    center = (int(X/2), int(Y/2))
    R = circle_radius

    # region of interest, creating pixcels to draw circle on
    roi = np.zeros((X,Y), dtype=np.uint8)

    circle = cv2.circle(roi, center, R, WHITE_RGB, thickness = -1)
    circle_rgb = img2rgb(circle)

    return circle_rgb


def apply_mask(image: cv2.Mat, mask: cv2.Mat) -> cv2.Mat:
    '''
    operate bitwise_and() method on two input images
    '''

    masked_image = cv2.bitwise_and(image, mask)

    return masked_image


def mask_sky_bgr(image: cv2.Mat) -> cv2.Mat:
```

```
    '''
    create a mask for sky using RGB inRange() method
    '''

    blue = cv2.inRange(image[:,:,0], 85, 255)
    # blue = cv2.medianBlur(blue, 5)
    green = cv2.inRange(image[:,:,1], 0, 65)
    # green = cv2.medianBlur(green, 5)
    red = cv2.inRange(image[:,:,2], 0, 65)
    # red = cv2.medianBlur(red, 5)

    no_sky1 = cv2.bitwise_or(green, red)
    no_sky2 = cv2.bitwise_not(blue)
    no_sky = cv2.bitwise_or(no_sky1, no_sky2)
    sky = cv2.bitwise_not(no_sky)
    sky = img2rgb(sky)

    return sky


def max_contour(image: cv2.Mat) -> cv2.Mat:
    '''
    draw the biggest contour on input image and return it
    '''

    imgray = img2gray(image)
    #imgray = cv2.medianBlur(imgray, 5)

    contour = np.zeros(image.shape, np.uint8)

    _, thresh = cv2.threshold(imgray, 95, 255, 0)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    max_contour = max(contours, key = cv2.contourArea)

    cv2.drawContours(image, [max_contour], -1, (0,0,255), thickness = 2)
    cv2.drawContours(contour, [max_contour], -1, WHITE_RGB, thickness = cv2.FILLED)

    contour[np.all(contour == (WHITE_RGB), axis = -1)] = MAGENTA_RGB
    # The np.all() function tests if all elements in an array are True

    return image, contour


def all_contours(image: cv2.Mat) -> cv2.Mat:
    '''
    draw all contours on input image and return it
    '''
    imgray = img2gray(image)
    # imgray = cv2.medianBlur(imgray, 5)

    contour = np.zeros(image.shape, np.uint8)
    image_copy = image.copy()
    _, thresh = cv2.threshold(imgray, 95, 255, 0)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

    cv2.drawContours(image_copy, contours, -1, (0,0,255), thickness = 2)
    cv2.drawContours(contour, contours, -1, WHITE_RGB, thickness = cv2.FILLED)

    contour[np.all(contour == (WHITE_RGB), axis = -1)] = MAGENTA_RGB
    # The np.all() function tests if all elements in an array are True
```

```
166
167     return image_copy, contour
168
169
170 def choose_contours(image: cv2.Mat) -> tuple:
171     '''
172     show all contours, use interactive mode to allow user to add correct ones
173     '''
174
175     imgray = img2gray(image)
176     # imgray = cv2.medianBlur(imgray, 5)
177
178     contours_new = []
179
180     contour = np.zeros(image.shape, np.uint8)
181
182     _, thresh = cv2.threshold(imgray, 95, 255, 0)
183     contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
184
185     _, thresh_max = cv2.threshold(imgray, 95, 255, 0)
186     contours_max, _ = cv2.findContours(thresh_max, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
187
188     max_contour = max(contours_max, key = cv2.contourArea)
189
190     contour_max = contour.copy()
191     cv2.drawContours(contour_max, [max_contour], -1, MAGENTA_RGB, thickness =
cv2.FILLED)
192
193     cv2.imshow('max contour', contour_max)
194     cv2.waitKey(0)
195     cv2.destroyAllWindows()
196
197     # initially, ask to save max contour
198     save_max_contour = input('Do you want to save the max contour?\nPress \'Enter\'
to skip.\n\ty/n: ')
199     if save_max_contour == 'y':
200         contours_new.append(max_contour)
201
202     for i in range(len(contours)):
203         contour_copy = contour.copy()
204         cv2.drawContours(contour_copy, contours, i, MAGENTA_RGB, thickness =
cv2.FILLED)
205         contour_copy = img2gray(contour_copy)   # cv2.countNonZero takes in grayscale
format
206
207         contour_pixels = int(cv2.countNonZero(contour_copy))
208         if (contour_pixels/NONZERO_CIRCLE)*100 > 0.075:
209             masked_image_copy = image.copy()
210             cv2.drawContours(masked_image_copy, contours, i, MAGENTA_RGB, thickness =
cv2.FILLED)
211             cv2.imshow(f'contour #{i}', masked_image_copy)
212             cv2.waitKey(0)
213             cv2.destroyAllWindows()
214             save_contour = input('Do you want to save this contour?\nPress \'Enter\'
to skip.\n\ty/n: ')
215             if save_contour == 'y':
216                 contours_new.append(contours[i])
217
218     contours_new = tuple(contours_new)
```

```
219
220     return contours_new
221
222
223 def apply_contours(image: cv2.Mat, contours: tuple) -> cv2.Mat:
224     '''
225     apply contours to input image
226     '''
227
228     contour = np.zeros(image.shape, np.uint8)
229
230     for i in range(len(contours)):
231         contour_applied = cv2.drawContours(contour, contours, i, (MAGENTA_RGB),
thickness =cv2.FILLED)
232         image_applied = cv2.drawContours(image, contours, i, MAGENTA_RGB, thickness
=cv2.FILLED)
233
234     return contour_applied, image_applied
235
236
237 def save_images(image: cv2.Mat, name: str, path: str, type: str):
238     '''
239     save image to input path with specified name format
240     '''
241
242     cv2.imwrite(os.path.join(path, f'{type}_{name}'), image)
243
244
245 def sky_ratio(sky: cv2.Mat, circle: cv2.Mat) -> int:
246     '''
247     calculate SFV% of an input photo (sky)
248     '''
249
250     img_sky = img2gray(sky)
251     img_circle = img2gray(circle)
252
253     sky_area = cv2.countNonZero(img_sky)
254     circle_area = cv2.countNonZero(img_circle)
255
256     ratio_sky = int(sky_area/circle_area*100)
257
258     # print('sky field of view is: ' + str(ratio_sky) + '%')
259     return ratio_sky
260
261
262 def input_data(df, row, col, value):
263     '''
264     add data to specified row and column
265     '''
266
267     df.loc[row, col] = value
268
269
270 def img2gray(image: cv2.Mat) -> cv2.Mat:
271     '''
272     convert BGR formatted image to Grayscale
273     '''
274
275     image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
276
```

```
    return image_gray


def img2rgb(image: cv2.Mat) -> cv2.Mat:
    '''
    convert Grayscale formatted image to BGR
    '''

    image_bgr = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)

    return image_bgr


def img2gray_uint8(image):
    '''
    convert image to grayscale format in uint8 format
    '''

    return np.uint8(img2gray(image) * 255)


def convert_img(img, target_type_min, target_type_max, target_type):
    '''
    convert image with input target parameters
    '''

    imin = img.min()
    imax = img.max()

    a = (target_type_max - target_type_min) / (imax - imin)
    b = target_type_max - a * imax
    new_img = (a * img + b).astype(target_type)
    return new_img

################################################################################################
"FOR REGIONAL GROWTH METHOD"

import numpy as np

'''region growth method'''
class Point(object):
 def __init__(self,x,y):
  self.x = x
  self.y = y

 def getX(self):
  return self.x
 def getY(self):
  return self.y


def getGrayDiff(img,currentPoint,tmpPoint):
 return abs(int(img[currentPoint.x,currentPoint.y]) -
int(img[tmpPoint.x,tmpPoint.y]))


def selectConnects(p):
 if p != 0:
  connects = [Point(-1, -1), Point(0, -1), Point(1, -1), Point(1, 0), Point(1, 1), \
```

```
    Point(0, 1), Point(-1, 1), Point(-1, 0)]
 else:
  connects = [ Point(0, -1), Point(1, 0),Point(0, 1), Point(-1, 0)]
 return connects


def regionGrow(img,seeds,thresh,p = 1):
 height, weight = img.shape
 seedMark = np.zeros(img.shape)
 seedList = []
 for seed in seeds:
  seedList.append(seed)
 label = 1
 connects = selectConnects(p)
 while(len(seedList)>0):
  currentPoint = seedList.pop(0)

  seedMark[currentPoint.x,currentPoint.y] = label
  for i in range(8):
   tmpX = currentPoint.x + connects[i].x
   tmpY = currentPoint.y + connects[i].y
   if tmpX < 0 or tmpY < 0 or tmpX >= height or tmpY >= weight:
    continue
   grayDiff = getGrayDiff(img,currentPoint,Point(tmpX,tmpY))
   if grayDiff < thresh and seedMark[tmpX,tmpY] == 0:
    seedMark[tmpX,tmpY] = label
    seedList.append(Point(tmpX,tmpY))
 return seedMark

############################################################################################
###########
"FOR METHOD 1- RGB SKY REGION DETECTION"

import cv2
import pandas as pd
import os
from skyfieldofview_func import *
from constants import IMAGE_PATH, CSV_PATH, FILTER_PATH, FILTERED_PATH, RESULT_PATH

# change working directory
os.chdir(IMAGE_PATH)

# load images from image folder
images = load_images(IMAGE_PATH)

# csv2DataFrame for latitude, longitude, altitude, SFV(empty)
df = pd.read_csv(CSV_PATH + '\parking_lat_long_alt.csv', index_col =
'parking_undistorted-[i].jpg')

for file in images:

    image = cv2.imread(file, cv2.IMREAD_COLOR)

    # step1 - rescale the image
    rescaled_image = rescale_image(image)
    #cv2.imshow('rescaled image', rescaled_image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    # step2 - set the radius for circular mask
```

```
    radius_circle = circle_radius(rescaled_image)

    # step3 - create circular mask
    circle_mask = mask_circle(rescaled_image, radius_circle)
    #cv2.imshow('circular mask', circle_mask)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    # step4 - apply circular mask to image
    masked_image = apply_mask(rescaled_image, circle_mask)
    #cv2.imshow('masked image', masked_image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    # step5 - create sky mask
    sky_mask = mask_sky_bgr(masked_image)
    sky_mask[np.all(sky_mask == (WHITE_RGB), axis = -1)] = MAGENTA_RGB

    #cv2.imshow('sky mask', sky_mask)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()
    save_images(sky_mask, file, FILTER_PATH, 'filter')

    # step6 - apply sky mask to image
    sky_image = apply_mask(masked_image, sky_mask)
    #cv2.imshow('sky masked image', sky_image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()
    save_images(sky_image, file, FILTERED_PATH, 'filtered')

    # step7 - calculate sky field of view of the image
    sky_field_of_view = sky_ratio(sky_image, circle_mask)

    # step8 - save sky field of view(SFV) result
    input_data(df, file, 'SFV(%)', sky_field_of_view)

#print(df)
df.to_csv(RESULT_PATH + '\parking_skyfieldofview_v1.csv')

################################################################################################
"FOR METHOD 2- RGM using SKY MASK"

import cv2
import pandas as pd
import os
from skyfieldofview_func import *
from region_growth_method import *
from constants import IMAGE_PATH, CSV_PATH, FILTER_PATH_V4, FILTERED_PATH_V4,
RESULT_PATH

# change working directory
os.chdir(IMAGE_PATH)

# load images from image folder
images = load_images(IMAGE_PATH)

# csv2DataFrame for latitude, longitude, altitude, SFV(empty)
df = pd.read_csv(CSV_PATH + '\parking_lat_long_alt.csv', index_col =
'parking_undistorted-[i].jpg')
```

```
450
451 #ind = 0
452 # skyfieldofview_v4 - region growth method over sky filter
453
454 for file in images:
455
456     image = cv2.imread(file, cv2.IMREAD_COLOR)
457
458     # step1 - rescale the image
459     rescaled_image = rescale_image(image)
460     #cv2.imshow('rescaled image', rescaled_image)
461     #cv2.waitKey(0)
462     #cv2.destroyAllWindows()
463
464     # step2 - set the radius for circular mask
465     radius_circle = circle_radius(rescaled_image)
466
467     # step3 - create circular mask
468     circle_mask = mask_circle(rescaled_image, radius_circle)
469     #cv2.imshow('circular mask', circle_mask)
470     #cv2.waitKey(0)
471     #cv2.destroyAllWindows()
472
473     # step4 - apply circular mask to image
474     masked_image = apply_mask(rescaled_image, circle_mask)
475     #cv2.imshow('masked image', masked_image)
476     #cv2.waitKey(0)
477     #cv2.destroyAllWindows()
478
479     # step5 - create sky mask
480     sky_mask = mask_sky_bgr(masked_image)
481     #cv2.imshow('sky mask', sky_mask)
482     #cv2.waitKey(0)
483     #cv2.destroyAllWindows()
484
485     # step6 - apply region grow method to sky mask
486     sky_mask_gray = img2gray(sky_mask)
487     seeds = [Point(328,328),Point(328,348),Point(328,368),\
488         Point(348,328),Point(348,348),Point(348, 368),\
489         Point(368,328),Point(368,348),Point(368,328)]
490
491     sky_mask_gray = regionGrow(sky_mask_gray, seeds, 5)
492     sky_mask_rgm = convert_img(sky_mask_gray, 0, 255, np.uint8)
493     sky_mask_rgm = img2rgb(sky_mask_rgm)
494     sky_mask_rgm[np.all(sky_mask_rgm == (WHITE_RGB), axis = -1)] = MAGENTA_RGB
495
496     #cv2.imshow(f'sky mask rgm-{ind}',sky_mask_rgm)
497     #cv2.waitKey(0)
498     #cv2.destroyAllWindows()
499     save_images(sky_mask_rgm, file, FILTER_PATH_V4, 'filter')
500     #ind += 1
501
502     # step7 - apply rgm sky mask to image
503     sky_image = apply_mask(masked_image, sky_mask_rgm)
504     #cv2.imshow('sky masked image', sky_image)
505     #cv2.waitKey(0)
506     #cv2.destroyAllWindows()
507     save_images(sky_image, file, FILTERED_PATH_V4, 'filtered')
508
509     # step8 - calculate sky field of view of the image
```

```
    sky_field_of_view = sky_ratio(sky_image, circle_mask)

    # step9 - save sky field of view(SFV) result
    input_data(df, file, 'SFV(%)', sky_field_of_view)

#print(df)
df.to_csv(RESULT_PATH + '\parking_skyfieldofview_v4.csv')

################################################################################################
"FOR METHOD 2- RGM using ORIGINAL IMAGES"

import cv2
import pandas as pd
import os
from skyfieldofview_func import *
from region_growth_method import *
from constants import IMAGE_PATH, CSV_PATH, FILTER_PATH_V5, FILTERED_PATH_V5,
RESULT_PATH

# change working directory
os.chdir(IMAGE_PATH)

# load images from image folder
images = load_images(IMAGE_PATH)

# csv2DataFrame for latitude, longitude, altitude, SFV(empty)
df = pd.read_csv(CSV_PATH + '\parking_lat_long_alt.csv', index_col =
'parking_undistorted-[i].jpg')

# skyfieldofview_v5 - region growth method over original image

for file in images:

    image = cv2.imread(file, cv2.IMREAD_COLOR)

    # step1 - rescale the image
    rescaled_image = rescale_image(image)
    #cv2.imshow('rescaled image', rescaled_image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    # step2 - set the radius for circular mask
    radius_circle = circle_radius(rescaled_image)

    # step3 - create circular mask
    circle_mask = mask_circle(rescaled_image, radius_circle)
    #cv2.imshow('circular mask', circle_mask)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    # step4 - apply circular mask to image
    masked_image = apply_mask(rescaled_image, circle_mask)
    #cv2.imshow('masked image', masked_image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()

    # step5 - apply region grow method to masked image
    sky_mask_gray = img2gray(masked_image)
    seeds = [Point(328,328),Point(328,348),Point(328,368),\
```

```
        Point(348,328),Point(348,348),Point(348, 368),\
        Point(368,328),Point(368,348),Point(368,328)]


    sky_mask_gray = regionGrow(sky_mask_gray, seeds, 5)
    sky_mask_rgm = convert_img(sky_mask_gray, 0, 255, np.uint8)
    sky_mask_rgm = img2rgb(sky_mask_rgm)
    sky_mask_rgm[np.all(sky_mask_rgm == (WHITE_RGB), axis = -1)] = MAGENTA_RGB

    #cv2.imshow('sky mask rgm',sky_mask_rgm)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()
    save_images(sky_mask_rgm, file, FILTER_PATH_V5, 'filter')

    # step6 - apply rgm sky mask to image
    sky_image = apply_mask(masked_image, sky_mask_rgm)
    #cv2.imshow('sky masked image', sky_image)
    #cv2.waitKey(0)
    #cv2.destroyAllWindows()
    save_images(sky_image, file, FILTERED_PATH_V5, 'filtered')

    # step7 - calculate sky field of view of the image
    sky_field_of_view = sky_ratio(sky_image, circle_mask)

    # step8 - save sky field of view(SFV) result
    input_data(df, file, 'SFV(%)', sky_field_of_view)

#print(df)
df.to_csv(RESULT_PATH + '\parking_skyfieldofview_v5.csv')

################################################################################################
"FOR GENERATING FINAL CSV USING SELECTED IMAGES"

import cv2
import pandas as pd
import os
from skyfieldofview_func import *
from region_growth_method import *
from constants import CSV_PATH, RESULT_PATH, NONZERO_CIRCLE

original = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_selected\original'
filter = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_selected\filter'
filtered = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\photos\photo_selected\filtered'

original_list = load_images(original)
filter_list = load_images(filter)

df = pd.read_csv(CSV_PATH + '\parking_lat_long_alt.csv', index_col = 'parking_undistorted-[i].jpg')

os.chdir(original)

ind = 0

for image in original_list:

```

```
    img = cv2.imread(image)
    img = rescale_image(img)

    os.chdir(filter)
    mask = cv2.imread(filter_list[ind])

    final = apply_mask(img, mask)
    save_images(final, image, filtered, 'filtered')

    mask = img2gray(mask)
    filter_pixel = int(cv2.countNonZero(mask))

    os.chdir(original)

    sky_field_of_view = (filter_pixel/NONZERO_CIRCLE)*100
    input_data(df, image, 'SFV(%)', sky_field_of_view)

    ind += 1

df.to_csv(RESULT_PATH + '\parking_skyfieldofview_selected.csv')
```

Appendix B. Project Code – PVLIB

```
TIMEZONE = 'Etc/GMT+2'
ALTITUDE = 660
CSV_PATH = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\opencv\results'
RESULT_PATH = r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\pvlib\results'

################################################################################################################################
"FOR JUNE 15TH, PARKING_UNDISTORTED-78.JPG"

import pvlib, pandas as pd, matplotlib.pyplot as plt


# get the module and inverter specifications from SAM
sandia_modules = pvlib.pvsystem.retrieve_sam('SandiaMod')
sapm_inverters = pvlib.pvsystem.retrieve_sam('cecinverter')

module = sandia_modules['Canadian_Solar_CS5P_220M___2009_']
inverter = sapm_inverters['ABB__MICRO_0_25_I_OUTD_US_208__208V_']

temperature_model_parameters = pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']


'''
In this example we will be using PVGIS, one of the data sources available,
to retrieve a Typical Meteorological Year (TMY) which includes irradiation,
temperature and wind speed
'''

system = {'module': module, 'inverter': inverter,
          'surface_azimuth': 0}
energies = {}


# coordinate_temp = (40.45314642, -3.726735261, 'parking_undistorted-78.jpg', altitude, 'Etc/GMT+2')
altitude = 660
SFV = 33.83

tmy_temp = pd.read_csv(r'C:\Users\Jimmy Son\Desktop\UPM\Asignaturas\TFM\pvlib\tmys\tmy_june_15_parking_undistorted-78.csv',
                       index_col = 'utc_time', )
tmy_temp.index = pd.to_datetime(tmy_temp.index)

tmy_temp['dhi'] = tmy_temp['dhi'] * SFV / 100

system['surface_tilt'] = 0
solpos = pvlib.solarposition.get_solarposition(
    time=tmy_temp.index,
    latitude=40.45314642,
    longitude=-3.726735261,
    altitude=660,
    temperature=tmy_temp["temp_air"],
    pressure=pvlib.atmosphere.alt2pres(altitude),
    )
dni_extra = pvlib.irradiance.get_extra_radiation(tmy_temp.index)
airmass = pvlib.atmosphere.get_relative_airmass(solpos['apparent_zenith'])
pressure = pvlib.atmosphere.alt2pres(altitude)
am_abs = pvlib.atmosphere.get_absolute_airmass(airmass, pressure)
```

```
aoi = pvlib.irradiance.aoi(
    system['surface_tilt'],
    system['surface_azimuth'],
    solpos["apparent_zenith"],
    solpos["azimuth"],
    )
total_irradiance = pvlib.irradiance.get_total_irradiance(
    system['surface_tilt'],
    system['surface_azimuth'],
    solpos['apparent_zenith'],
    solpos['azimuth'],
    tmy_temp['dni'],
    tmy_temp['ghi'],
    tmy_temp['dhi'],
    dni_extra=dni_extra,
    model='isotropic',
    )
cell_temperature = pvlib.temperature.sapm_cell(
    total_irradiance['poa_global'],
    tmy_temp["temp_air"],
    tmy_temp["wind_speed"],
    **temperature_model_parameters,
    )
effective_irradiance = pvlib.pvsystem.sapm_effective_irradiance(
    total_irradiance['poa_direct'],
    total_irradiance['poa_diffuse'],
    am_abs,
    aoi,
    module,
    )
dc = pvlib.pvsystem.sapm(effective_irradiance, cell_temperature, module)
ac = pvlib.inverter.sandia(dc['v_mp'], dc['p_mp'], inverter)
annual_energy = ac.sum()
energies['june_15'] = annual_energy

energies = pd.Series(energies)

###############################################################################
########################################
"FOR JUNE 15TH, FOR ALL IMAGES"

import pvlib, pandas as pd, matplotlib.pyplot as plt
from constants import *


df = pd.read_csv(CSV_PATH + '\SFV_selected.csv', index_col = 'parking_undistorted-
[i].jpg', )
#print(df)


# latitude, longitude, name, altitude, timezone
altitude = ALTITUDE
timezone = TIMEZONE


# append all coordinates from csv
coordinates = []
for (idx, row) in df.iterrows():
    latitude = row.loc['latitude']
    longitude = row.loc['longitude']
```

```
    name = idx
    coordinate = (latitude, longitude, name, altitude, timezone)
    coordinates.append(coordinate)
#print(coordinates)


# append all SFV values from csv
SFV = []
for ind in range(df.shape[0]):
    sky_field_of_view = int(df['SFV(%)'][ind])
    SFV.append(sky_field_of_view)
#print(SFV)


# get the module and inverter specifications from SAM
sandia_modules = pvlib.pvsystem.retrieve_sam('SandiaMod')
sapm_inverters = pvlib.pvsystem.retrieve_sam('cecinverter')

module = sandia_modules['Canadian_Solar_CS5P_220M___2009_']
inverter = sapm_inverters['ABB__MICRO_0_25_I_OUTD_US_208__208V_']

temperature_model_parameters = pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']
['open_rack_glass_glass']


'''
In this example we will be using PVGIS, one of the data sources available,
to retrieve a Typical Meteorological Year (TMY) which includes irradiation,
temperature and wind speed
'''

system = {'module': module, 'inverter': inverter,
          'surface_azimuth': 0}
energies = {}


# need to trim date to June 15th
# apply SFV
tmys = []
for location in coordinates:
    latitude, longitude, name, altitude, timezone = location
    weather = pvlib.iotools.get_pvgis_tmy(latitude, longitude,
                                          map_variables=True)[0]
    weather.index.name = "utc_time"
    tmys.append(weather)


# apply SFV for each coordinate
#for ind in range(len(tmys)):
#    tmys[ind]['dhi'] = tmys[ind]['dhi'] * SFV[ind] / 100


# select data for June 15th (row 3961 -> row 3985)
tmys_june_15 = []
for ind in range(len(tmys)):
    tmy_june_15 = tmys[ind].iloc[3961 : 3985]
    tmys_june_15.append(tmy_june_15)


for location, weather in zip(coordinates, tmys_june_15):
```

```
    latitude, longitude, name, altitude, timezone = location
    system['surface_tilt'] = 0
    solpos = pvlib.solarposition.get_solarposition(
        time=weather.index,
        latitude=latitude,
        longitude=longitude,
        altitude=altitude,
        temperature=weather["temp_air"],
        pressure=pvlib.atmosphere.alt2pres(altitude),
    )
    dni_extra = pvlib.irradiance.get_extra_radiation(weather.index)
    airmass = pvlib.atmosphere.get_relative_airmass(solpos['apparent_zenith'])
    pressure = pvlib.atmosphere.alt2pres(altitude)
    am_abs = pvlib.atmosphere.get_absolute_airmass(airmass, pressure)
    aoi = pvlib.irradiance.aoi(
        system['surface_tilt'],
        system['surface_azimuth'],
        solpos["apparent_zenith"],
        solpos["azimuth"],
    )
    total_irradiance = pvlib.irradiance.get_total_irradiance(
        system['surface_tilt'],
        system['surface_azimuth'],
        solpos['apparent_zenith'],
        solpos['azimuth'],
        weather['dni'],
        weather['ghi'],
        weather['dhi'],
        dni_extra=dni_extra,
        model='isotropic',
    )
    cell_temperature = pvlib.temperature.sapm_cell(
        total_irradiance['poa_global'],
        weather["temp_air"],
        weather["wind_speed"],
        **temperature_model_parameters,
    )
    effective_irradiance = pvlib.pvsystem.sapm_effective_irradiance(
        total_irradiance['poa_direct'],
        total_irradiance['poa_diffuse'],
        am_abs,
        aoi,
        module,
    )
    dc = pvlib.pvsystem.sapm(effective_irradiance, cell_temperature, module)
    ac = pvlib.inverter.sandia(dc['v_mp'], dc['p_mp'], inverter)
    annual_energy = ac.sum()
    energies[name] = annual_energy


energies = pd.Series(energies)

###############################################################################
########################################
"FOR ENTIRE YEAR, FOR ALL IMAGES"

import pvlib, pandas as pd, matplotlib.pyplot as plt
from constants import *


```

```
df = pd.read_csv(CSV_PATH + '\SFV_selected.csv', index_col = 'parking_undistorted-[i].jpg', )
#print(df)


# latitude, longitude, name, altitude, timezone
altitude = ALTITUDE
timezone = TIMEZONE


# append all coordinates from csv
coordinates = []
for (idx, row) in df.iterrows():
    latitude = row.loc['latitude']
    longitude = row.loc['longitude']
    name = idx
    coordinate = (latitude, longitude, name, altitude, timezone)
    coordinates.append(coordinate)
#print(coordinates)


# append all SFV values from csv
SFV = []
for ind in range(df.shape[0]):
    sky_field_of_view = int(df['SFV(%)'][ind])
    SFV.append(sky_field_of_view)
#print(SFV)


# get the module and inverter specifications from SAM
sandia_modules = pvlib.pvsystem.retrieve_sam('SandiaMod')
sapm_inverters = pvlib.pvsystem.retrieve_sam('cecinverter')

module = sandia_modules['Canadian_Solar_CS5P_220M___2009_']
inverter = sapm_inverters['ABB__MICRO_0_25_I_OUTD_US_208__208V_']

temperature_model_parameters = pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']


'''
In this example we will be using PVGIS, one of the data sources available,
to retrieve a Typical Meteorological Year (TMY) which includes irradiation,
temperature and wind speed
'''


# tmys collect data for one year
tmys = []
for location in coordinates:
    latitude, longitude, name, altitude, timezone = location
    weather = pvlib.iotools.get_pvgis_tmy(latitude, longitude,
                                          map_variables=True)[0]
    weather.index.name = "utc_time"
    tmys.append(weather)


# apply SFV for each coordinate
#for ind in range(len(tmys)):
#    tmys[ind]['dhi'] = tmys[ind]['dhi'] * SFV[ind] / 100
```

```
287
288
289
290 system = {'module': module, 'inverter': inverter,
291           'surface_azimuth': 0}
292 energies = {}
293
294
295 for location, weather in zip(coordinates, tmys):
296     latitude, longitude, name, altitude, timezone = location
297     system['surface_tilt'] = 0
298     solpos = pvlib.solarposition.get_solarposition(
299         time=weather.index,
300         latitude=latitude,
301         longitude=longitude,
302         altitude=altitude,
303         temperature=weather["temp_air"],
304         pressure=pvlib.atmosphere.alt2pres(altitude),
305     )
306     dni_extra = pvlib.irradiance.get_extra_radiation(weather.index)
307     airmass = pvlib.atmosphere.get_relative_airmass(solpos['apparent_zenith'])
308     pressure = pvlib.atmosphere.alt2pres(altitude)
309     am_abs = pvlib.atmosphere.get_absolute_airmass(airmass, pressure)
310     aoi = pvlib.irradiance.aoi(
311         system['surface_tilt'],
312         system['surface_azimuth'],
313         solpos["apparent_zenith"],
314         solpos["azimuth"],
315     )
316     total_irradiance = pvlib.irradiance.get_total_irradiance(
317         system['surface_tilt'],
318         system['surface_azimuth'],
319         solpos['apparent_zenith'],
320         solpos['azimuth'],
321         weather['dni'],
322         weather['ghi'],
323         weather['dhi'],
324         dni_extra=dni_extra,
325         model='isotropic',
326     )
327     cell_temperature = pvlib.temperature.sapm_cell(
328         total_irradiance['poa_global'],
329         weather["temp_air"],
330         weather["wind_speed"],
331         **temperature_model_parameters,
332     )
333     effective_irradiance = pvlib.pvsystem.sapm_effective_irradiance(
334         total_irradiance['poa_direct'],
335         total_irradiance['poa_diffuse'],
336         am_abs,
337         aoi,
338         module,
339     )
340     dc = pvlib.pvsystem.sapm(effective_irradiance, cell_temperature, module)
341     ac = pvlib.inverter.sandia(dc['v_mp'], dc['p_mp'], inverter)
342     annual_energy = ac.sum()
343     energies[name] = annual_energy
344
345
346 energies = pd.Series(energies)
```

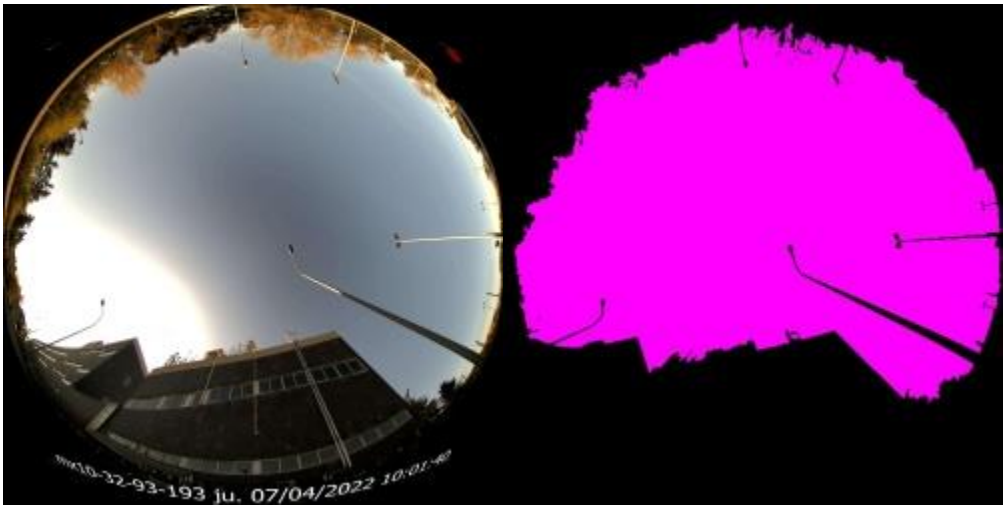
```
347
348 ##########################################################################################
    ########################################
349 "FOR GENERATING PLOTS"
350
351 import pvlib, pandas as pd, matplotlib.pyplot as plt
352 from constants import *
353
354 df = pd.read_csv(RESULT_PATH + '\parking_undistorted-78_iso_sfv.csv', index_col =
    'utc_time', )
355
356 plt.figure(figsize=(10,5))
357 plt.plot(df.index, df['poa_global_sfv'], label = 'poa_global')
358 plt.plot(df.index, df['poa_direct_sfv'], label = 'poa_direct')
359 plt.plot(df.index, df['poa_diffuse_sfv'], label = 'poa_diffuse')
360 plt.legend(loc = 'upper left')
361 plt.xlabel('time')
362 plt.ylabel('irradiance (W/m2)')
363 plt.grid()
364 plt.show()
365
366 plt.figure(figsize=(10,5))
367 plt.plot(df.index, df['ghi_sfv'], label = 'GHI')
368 plt.plot(df.index, df['dni_sfv'], label = 'DNI')
369 plt.plot(df.index, df['dhi_sfv'], label = 'DHI')
370 plt.legend(loc = 'upper left')
371 plt.xlabel('time')
372 plt.ylabel('irradiance (W/m2)')
373 plt.grid()
374 plt.show()
```

Appendix C. Tables – SFV

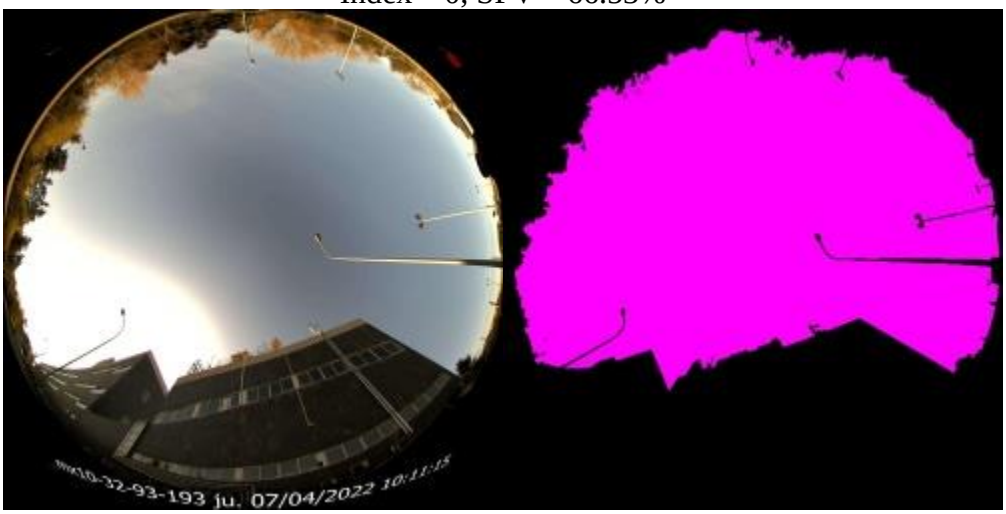
image	latitude	longitude	SFV(%)	image	latitude	longitude	SFV(%)
parking_undistorted-0.jpg	40.4536	-3.7270	66.53	parking_undistorted-41.jpg	40.4537	-3.7269	65.43
parking_undistorted-1.jpg	40.4536	-3.7270	64.91	parking_undistorted-44.jpg	40.4537	-3.7267	48.96
parking_undistorted-10.jpg	40.4536	-3.7270	64.49	parking_undistorted-45.jpg	40.4529	-3.7269	49.95
parking_undistorted-100.jpg	40.4536	-3.7270	25.16	parking_undistorted-46.jpg	40.4529	-3.7269	46.09
parking_undistorted-103.jpg	40.4536	-3.7269	25.63	parking_undistorted-48.jpg	40.4530	-3.7269	47.55
parking_undistorted-105.jpg	40.4536	-3.7268	25.32	parking_undistorted-49.jpg	40.4530	-3.7269	46.46
parking_undistorted-109.jpg	40.4533	-3.7265	24.70	parking_undistorted-5.jpg	40.4530	-3.7269	62.06
parking_undistorted-11.jpg	40.4533	-3.7266	63.55	parking_undistorted-50.jpg	40.4530	-3.7269	47.32
parking_undistorted-110.jpg	40.4533	-3.7266	25.64	parking_undistorted-51.jpg	40.4529	-3.7268	50.26
parking_undistorted-111.jpg	40.4533	-3.7266	31.19	parking_undistorted-53.jpg	40.4530	-3.7268	47.80
parking_undistorted-112.jpg	40.4533	-3.7267	36.60	parking_undistorted-54.jpg	40.4531	-3.7268	74.15
parking_undistorted-12.jpg	40.4533	-3.7267	62.67	parking_undistorted-55.jpg	40.4531	-3.7269	71.67
parking_undistorted-13.jpg	40.4533	-3.7267	59.22	parking_undistorted-6.jpg	40.4529	-3.7270	62.03
parking_undistorted-14.jpg	40.4533	-3.7268	54.46	parking_undistorted-60.jpg	40.4529	-3.7270	51.74
parking_undistorted-15.jpg	40.4533	-3.7267	60.71	parking_undistorted-61.jpg	40.4530	-3.7270	55.15
parking_undistorted-16.jpg	40.4536	-3.7267	55.42	parking_undistorted-62.jpg	40.4530	-3.7270	55.38
parking_undistorted-17.jpg	40.4533	-3.7267	68.67	parking_undistorted-63.jpg	40.4530	-3.7270	52.88
parking_undistorted-18.jpg	40.4532	-3.7267	70.12	parking_undistorted-64.jpg	40.4530	-3.7270	53.74
parking_undistorted-19.jpg	40.4532	-3.7268	70.20	parking_undistorted-67.jpg	40.4531	-3.7271	35.56
parking_undistorted-2.jpg	40.4536	-3.7267	64.79	parking_undistorted-7.jpg	40.4532	-3.7271	62.90
parking_undistorted-20.jpg	40.4536	-3.7266	69.40	parking_undistorted-72.jpg	40.4531	-3.7270	41.65
parking_undistorted-21.jpg	40.4536	-3.7266	70.08	parking_undistorted-73.jpg	40.4531	-3.7269	41.68
parking_undistorted-22.jpg	40.4537	-3.7266	70.35	parking_undistorted-74.jpg	40.4531	-3.7269	39.88
parking_undistorted-23.jpg	40.4537	-3.7266	70.15	parking_undistorted-75.jpg	40.4532	-3.7268	43.28
parking_undistorted-24.jpg	40.4537	-3.7266	69.94	parking_undistorted-76.jpg	40.4532	-3.7268	39.88
parking_undistorted-25.jpg	40.4537	-3.7267	71.21	parking_undistorted-77.jpg	40.4531	-3.7268	37.24
parking_undistorted-26.jpg	40.4537	-3.7267	72.39	parking_undistorted-78.jpg	40.4531	-3.7267	33.83
parking_undistorted-27.jpg	40.4537	-3.7267	72.16	parking_undistorted-79.jpg	40.4532	-3.7267	30.00
parking_undistorted-28.jpg	40.4537	-3.7268	71.61	parking_undistorted-8.jpg	40.4532	-3.7267	62.73
parking_undistorted-29.jpg	40.4537	-3.7268	72.24	parking_undistorted-80.jpg	40.4532	-3.7266	30.54
parking_undistorted-3.jpg	40.4537	-3.7268	64.02	parking_undistorted-81.jpg	40.4532	-3.7265	27.67
parking_undistorted-30.jpg	40.4536	-3.7269	82.56	parking_undistorted-82.jpg	40.4532	-3.7266	19.95
parking_undistorted-31.jpg	40.4536	-3.7269	83.52	parking_undistorted-83.jpg	40.4532	-3.7265	53.80
parking_undistorted-32.jpg	40.4536	-3.7269	75.76	parking_undistorted-84.jpg	40.4532	-3.7265	22.52
parking_undistorted-33.jpg	40.4536	-3.7270	74.49	parking_undistorted-85.jpg	40.4532	-3.7264	23.79
parking_undistorted-34.jpg	40.4536	-3.7270	62.70	parking_undistorted-86.jpg	40.4532	-3.7264	24.72
parking_undistorted-35.jpg	40.4536	-3.7270	73.55	parking_undistorted-87.jpg	40.4532	-3.7264	26.42
parking_undistorted-36.jpg	40.4536	-3.7272	71.46	parking_undistorted-88.jpg	40.4532	-3.7264	25.34
parking_undistorted-37.jpg	40.4536	-3.7272	68.83	parking_undistorted-89.jpg	40.4532	-3.7263	26.27
parking_undistorted-38.jpg	40.4537	-3.7271	65.26	parking_undistorted-9.jpg	40.4532	-3.7263	62.36
parking_undistorted-39.jpg	40.4537	-3.7271	62.32	parking_undistorted-90.jpg	40.4532	-3.7263	27.18
parking_undistorted-4.jpg	40.4537	-3.7270	62.78	parking_undistorted-92.jpg	40.4532	-3.7266	21.62
parking_undistorted-40.jpg	40.4537	-3.7269	68.14	parking_undistorted-99.jpg	40.4533	-3.7265	25.69

Appendix D. All-Sky Images – Original Undistorted vs Filter

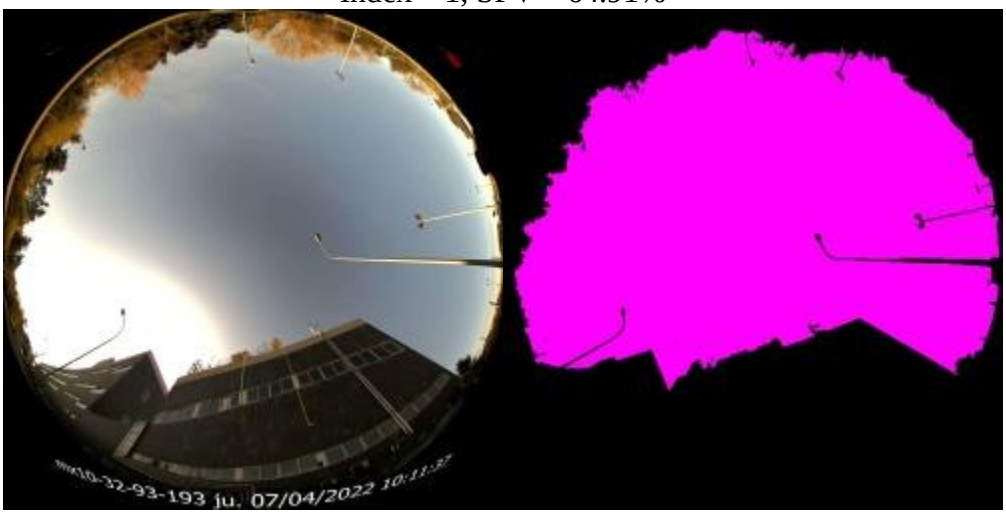
(Index → parking_undistorted-{index}.jpg)



Index – 0, SFV – 66.53%

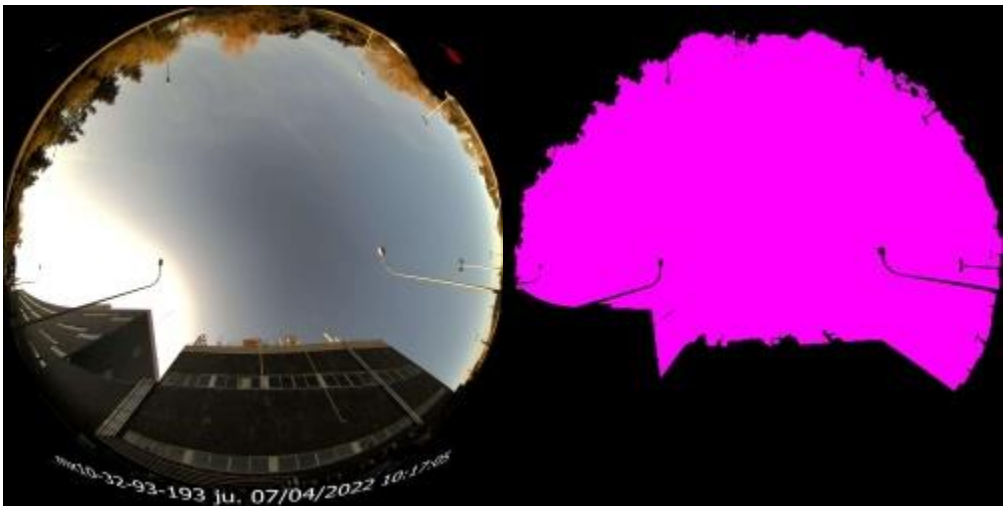


Index – 1, SFV – 64.91%



Index – 2, SFV – 64.78%

Index – 3, SFV – 64.08%



Index – 4, SFV – 62.78%

Index – 5, SFV – 62.06%

Index – 6, SFV – 62.03%

Index – 7, SFV – 62.90%

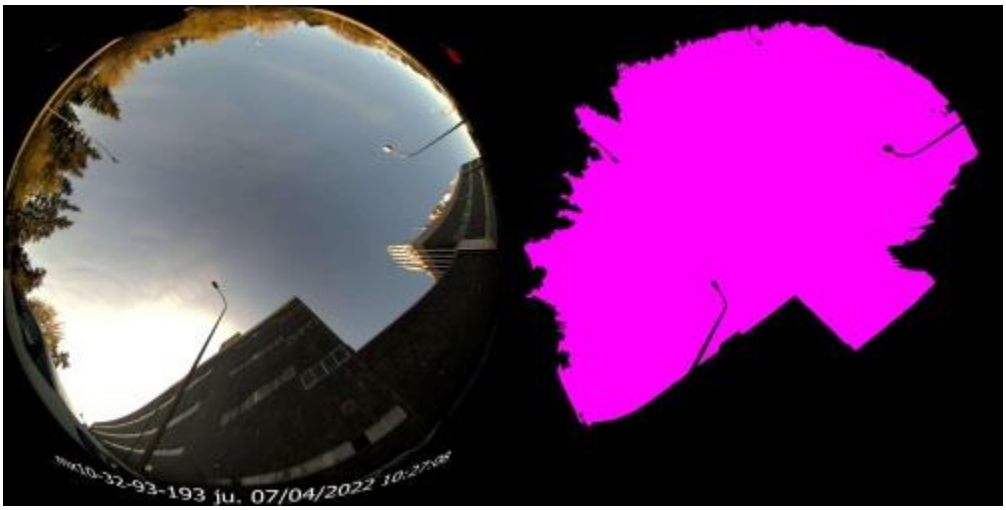
Index – 8, SFV – 62.73%

Index – 9, SFV – 62.34%

Index – 10, SFV – 64.49%

Index – 11, SFV – 63.55%

Index – 12, SFV – 62.67%



Index – 13, SFV – 59.22%

Index – 14, SFV – 54.46%

Index – 15, SFV – 60.71%

Index – 16, SFV – 55.42%

Index – 17, SFV – 68.67%

Index – 18, SFV – 70.12%

Index – 19, SFV – 70.20%

Index – 20, SFV – 69.40%

Index – 21, SFV – 70.08%

Index – 22, SFV – 70.35%

Index – 23, SFV – 70.15%

Index – 24, SFV – 69.94%

Index – 25, SFV – 71.20%

Index – 26, SFV – 72.39%

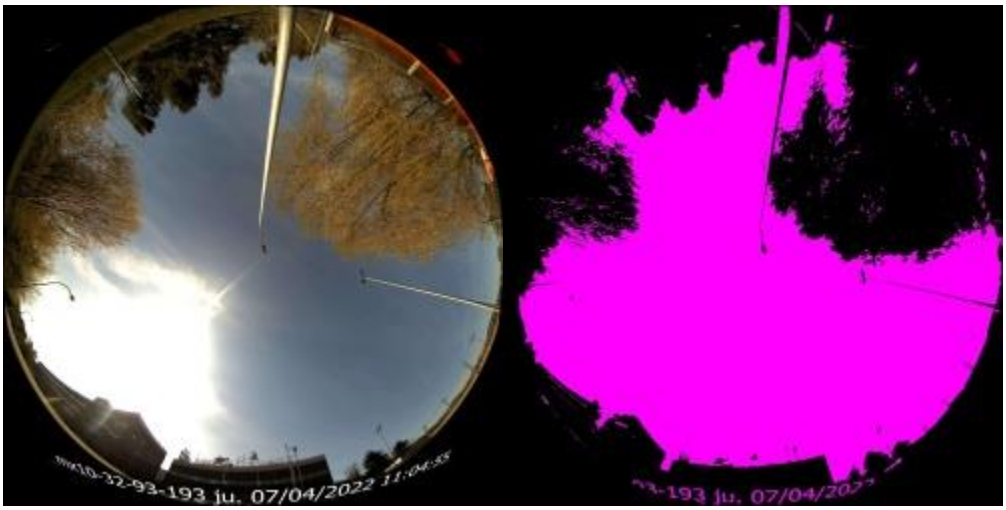
Index – 27, SFV – 72.16%

Index – 28, SFV – 71.61%

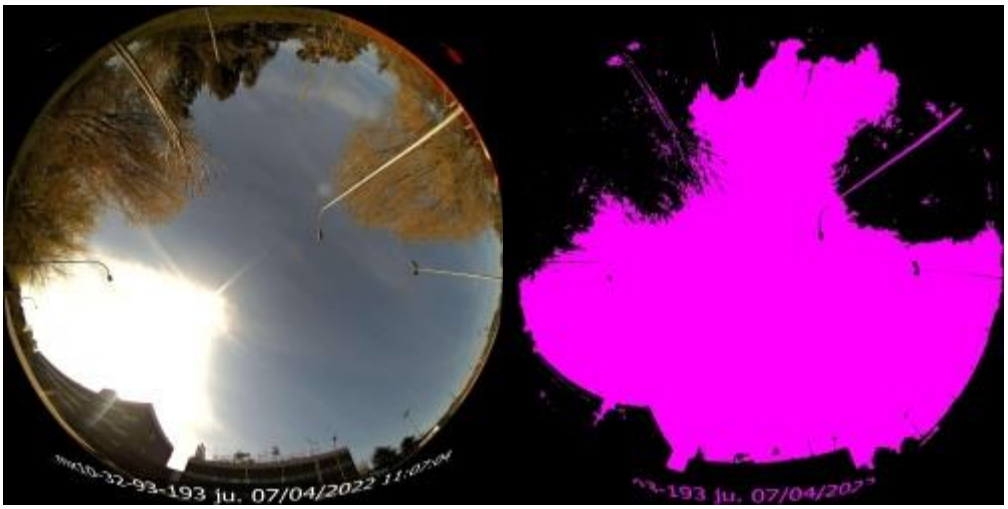
Index – 29, SFV – 72.24%

Index – 30, SFV – 82.55%

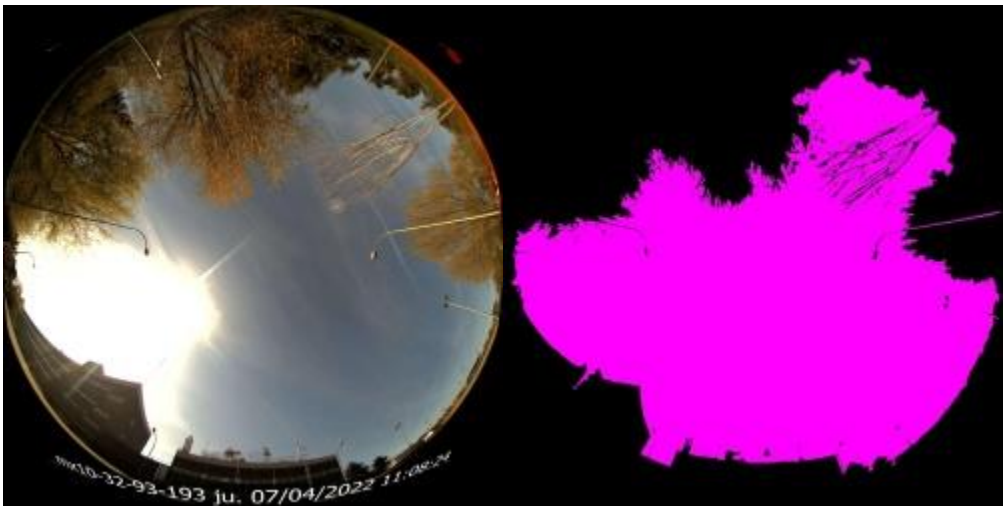
Index – 31, SFV – 83.52%



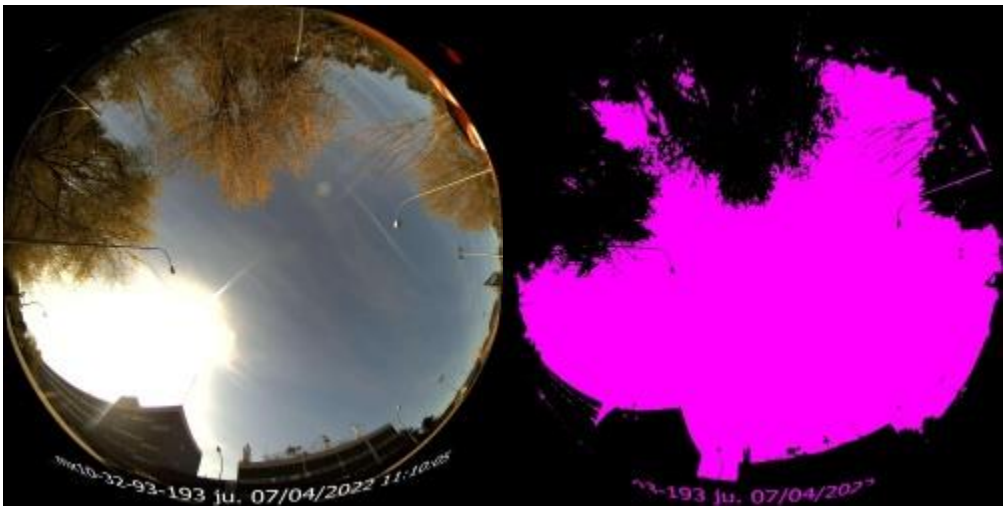
Index – 32, SFV – 75.76%



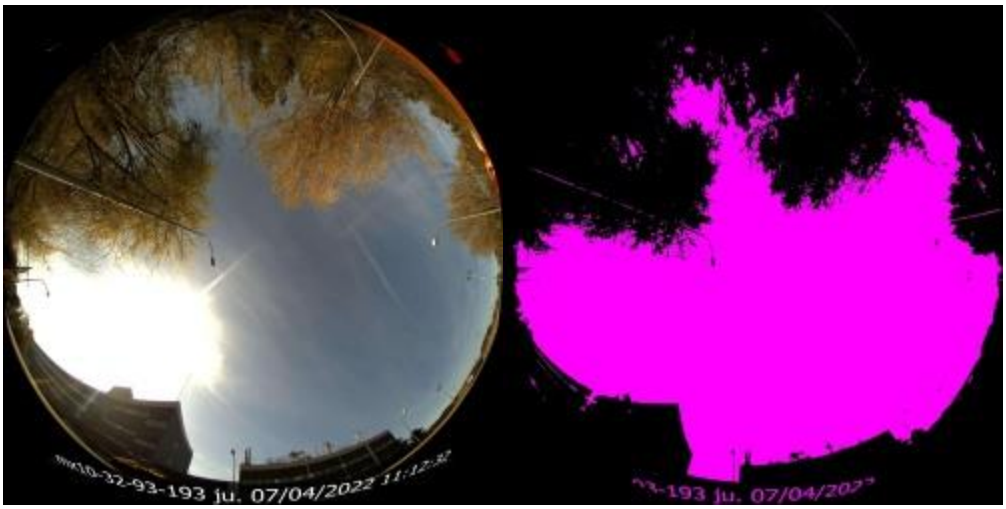
Index – 33, SFV – 74.49%



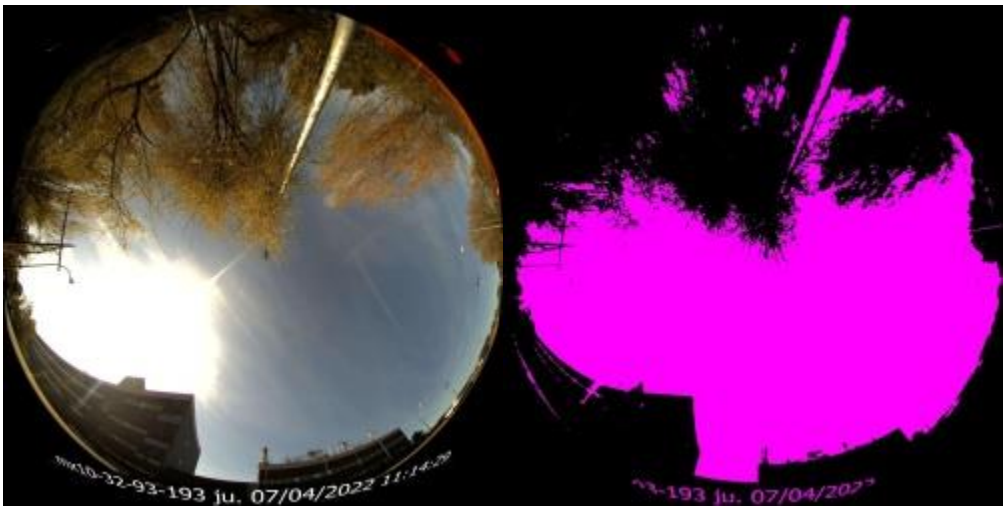
Index – 34, SFV – 62.70%



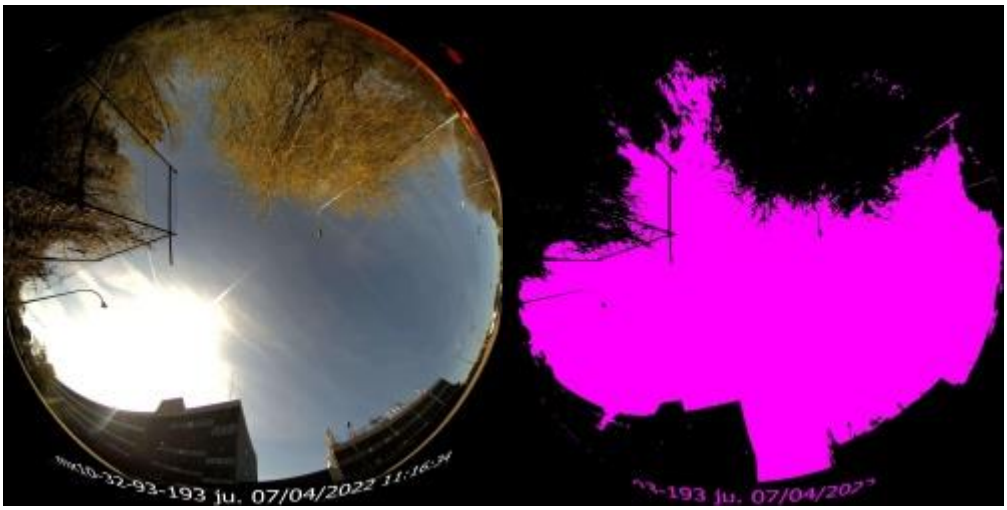
Index – 35, SFV – 73.55%



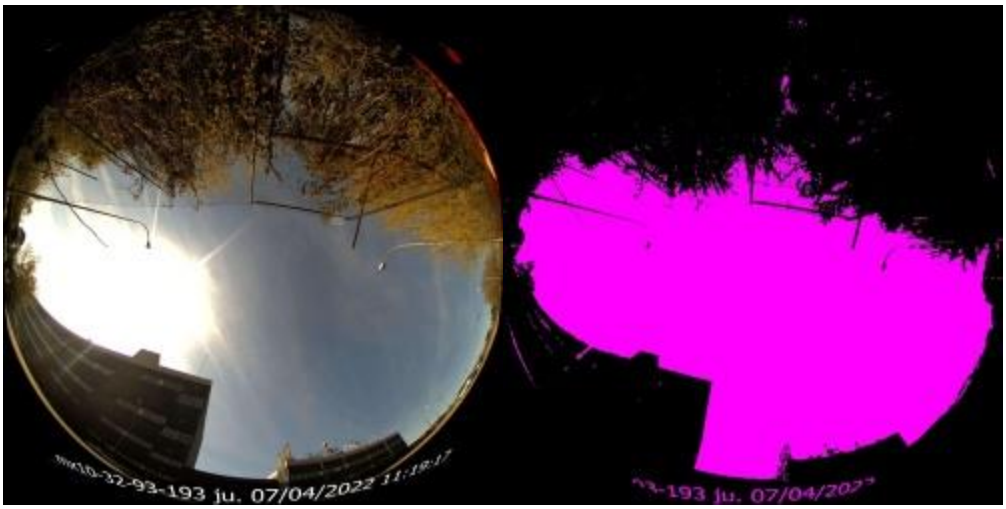
Index – 36, SFV – 71.46%



Index – 37, SFV – 68.83%



Index – 38, SFV – 65.25%



Index – 39, SFV – 62.32%

Index – 40, SFV – 68.13%

Index – 41, SFV – 65.43%

Index – 44, SFV – 48.96%

Index – 45, SFV – 49.95%

Index – 46, SFV – 46.09%

Index – 48, SFV – 47.55%

Index – 49, SFV – 46.46%

Index – 50, SFV – 47.32%

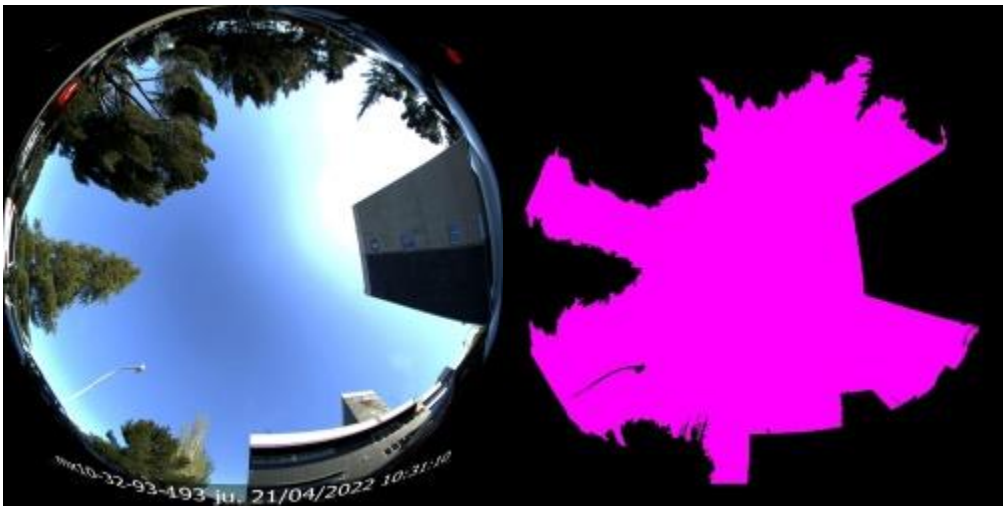
Index – 51, SFV – 50.26%

Index – 53, SFV – 47.80%

Index – 54, SFV – 74.15%

Index – 55, SFV – 71.67%

Index – 60, SFV – 51.74%



Index – 61, SFV – 55.15%

Index – 62, SFV – 55.38%

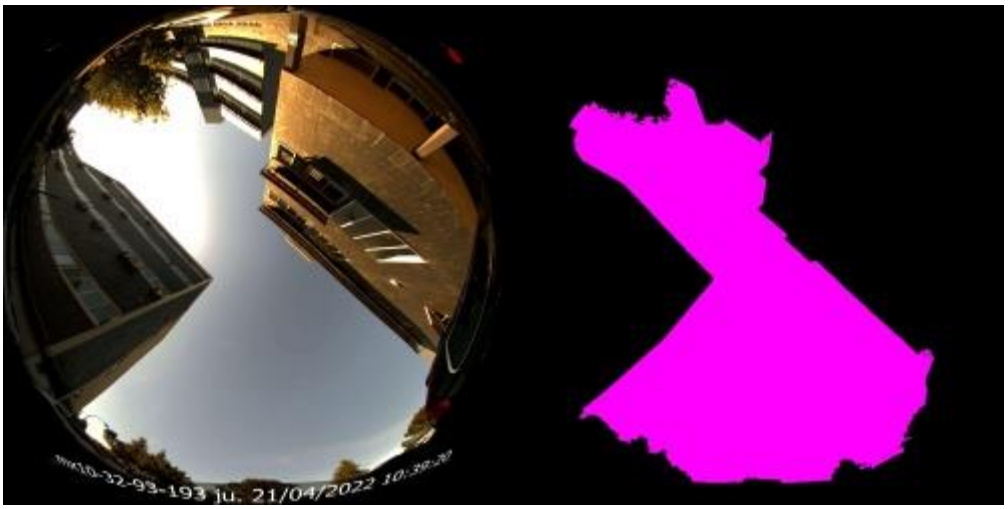
Index – 63, SFV – 52.88%

Index – 64, SFV – 53.74%

Index – 67, SFV – 35.56%

Index – 72, SFV – 41.65%

Index – 73, SFV – 41.68%



Index – 74, SFV – 39.88%

Index – 75, SFV – 43.28%

Index – 76, SFV – 39.88%

Index – 77, SFV – 37.24%

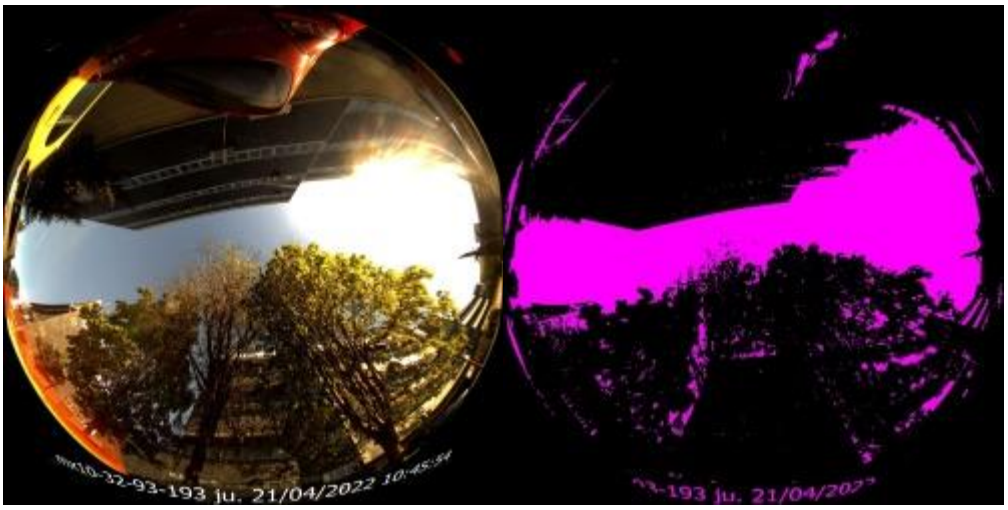
Index – 78, SFV – 33.83%

Index – 79, SFV – 30.00%

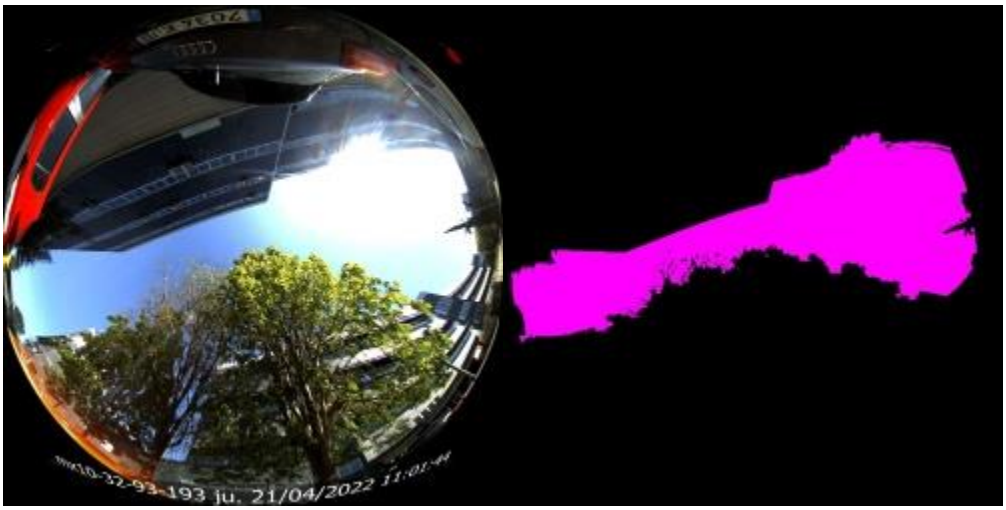
Index – 80, SFV – 30.54%

Index – 81, SFV – 27.67%

Index – 82, SFV – 19.95%



Index – 83, SFV – 53.80%

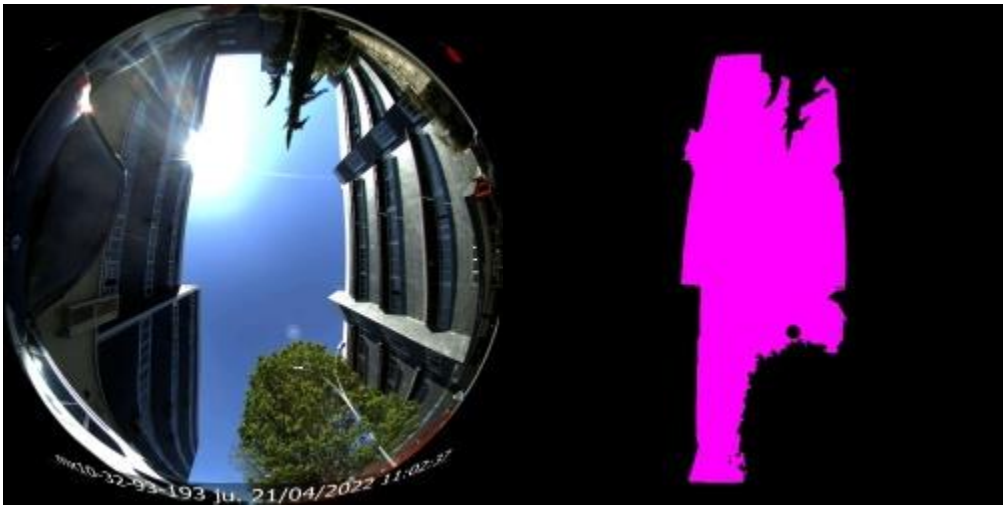


Index – 84, SFV – 22.52%

Index – 85, SFV – 23.79%

Index – 86, SFV – 24.72%

Index – 87, SFV – 26.42%



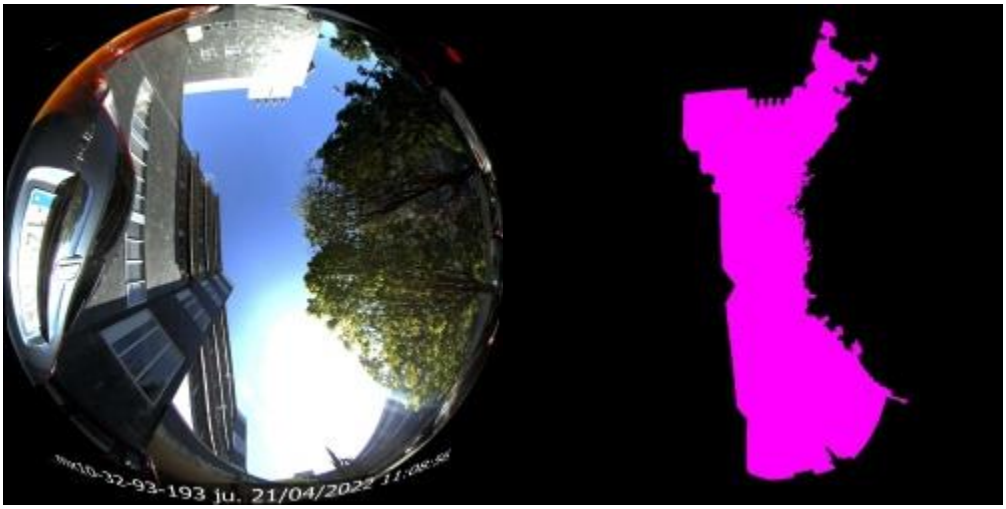
Index – 88, SFV – 25.34%

Index – 89, SFV – 26.27%

Index – 90, SFV – 27.18%

Index – 92, SFV – 21.62%

Index – 99, SFV – 25.68%



Index – 100, SFV – 25.16%

Index – 103, SFV – 25.63%

Index – 105, SFV – 25.32%

Index – 109, SFV – 24.70%

Index – 110, SFV – 25.64%

Index – 111, SFV – 31.19%

Index – 112, SFV – 36.60%