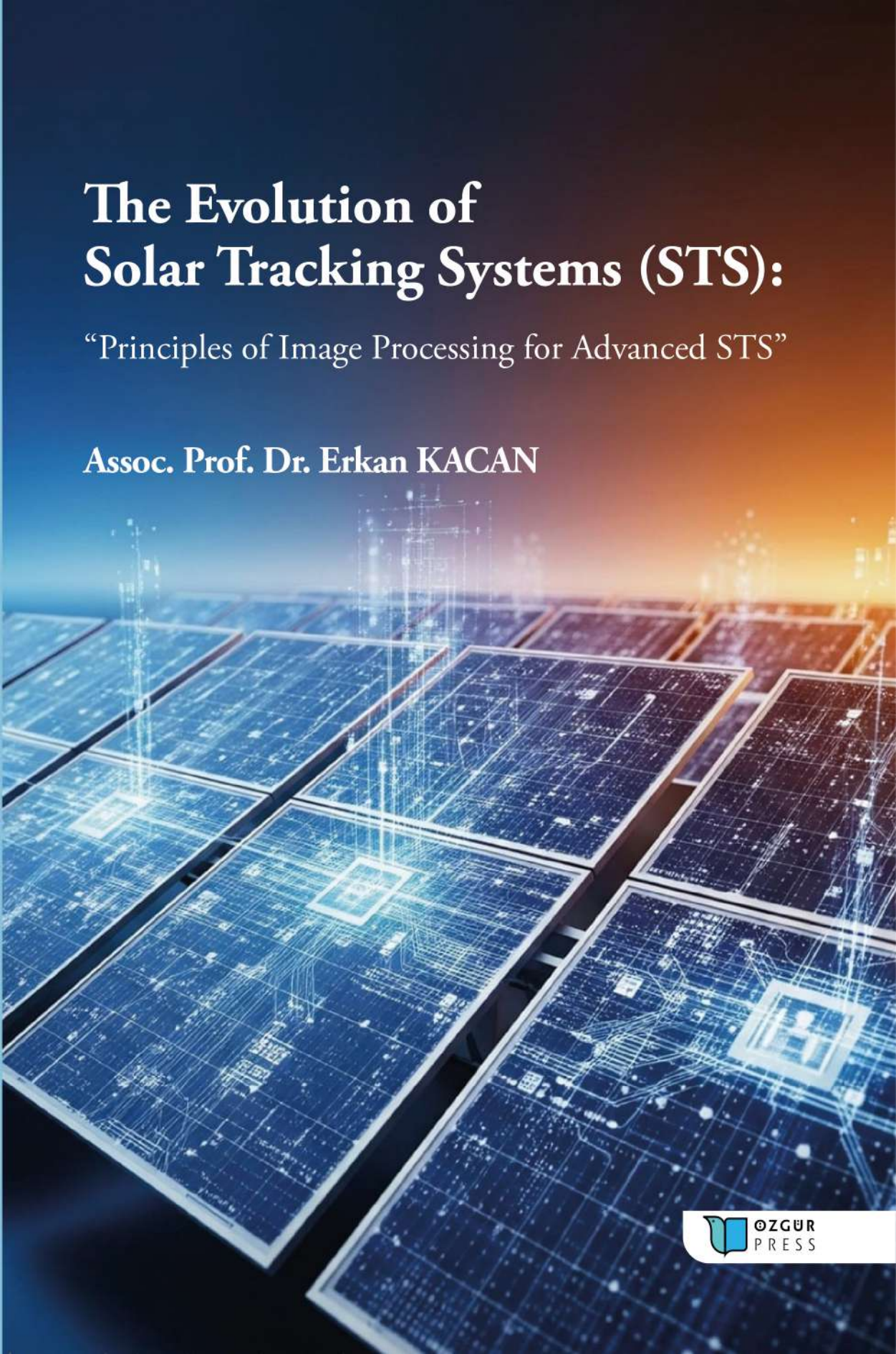
The Evolution of Solar Tracking Systems (STS):

"Principles of Image Processing for Advanced STS"

Assoc. Prof. Dr. Erkan KACAN



ÖZGÜR PRESS

The Evolution of Solar Tracking Systems (STS):

"Principles of Image Processing for Advanced STS"

Assoc. Prof. Dr. Erkan KACAN

ÖZGÜR PRESS

Published by
Özgür Yayın-Dağıtım Co. Ltd.
Certificate Number: 45503

15 Temmuz Mah. 148136. Sk. No: 9 Şehitkamil/Gaziantep
+90.850 260 09 97
+90.532 289 82 15
www.ozguryayınlari.com
info@ozguryayinlari.com

The Evolution of Solar Tracking Systems (STS): "Principles of Image Processing for Advanced STS"

Assoc. Prof. Dr. Erkan KACAN

Language: English
Publication Date: 2025
Cover design by Mehmet Çakır
Cover design and image licensed under CC BY-NC 4.0
Print and digital versions typeset by Çizgi Medya Co. Ltd.

ISBN (PDF): 978-625-5646-05-7

DOI: https://doi.org/10.58830/ozgur.pub750

Suggested citation:
Kacan, E. (2025). *The Evolution of Solar Tracking Systems (STS): "Principles of Image Processing for Advanced STS"* Özgür Publications. DOI: https://doi.org/10.58830/ozgur.pub750. License: CC-BY-NC 4.0

The full text of this book has been peer-reviewed to ensure high academic standards. For full review policies, see https://www.ozguryayinlari.com/

ÖZGÜR PRESS

Preface

Dear Readers,

The sun, in addition to being the source of life on Earth, offers an unlimited potential for clean energy. Harnessing this potential in the most efiicient way has become a primary objective for the fields of engineering and science, particularly in light of contemporary energy demands and environmental concerns. Solar tracking systems are crucial technologies that maximize energy production by enabling solar panels to follow the sun at the optimal angle throughout the day. In the early stages, solar tracking was largely achieved through mechanical and simpler control systems. However, it has evolved to become significantly more precise and efficient today, thanks to intelligent algorithms such as image processing. The future will inevitably see even more advanced and accurate tracking through hybrid methods.

The implementation of solar tracking systems using image processing methods represents the latest and most technologically advanced state of solar tracking systems. This method offers significant advantages as it can directly sense the sun's position, provide adaptation to conditions like cloud cover, and reduce the requirement for additional sensors.

This book aims to illuminate the past, present state, and future expectations of solar tracking systems.

Within this book, the fundamental principles, hardware components, and especially image processing-based intelligent control algorithms underpinning solar tracking systems are comprehensively addressed. The selection of system components and load calculations are examined in detail.

Throughout the book, in-depth discussions are provided on topics such as algorithms developed for the precise detection of the sun's position, actuator control methods (ranging from simple on/off to PID control),

adaptation to dynamic environmental conditions (adaptive thresholding), and even astronomical positioning for scenarios where the sun is not visible. Furthermore, practical operational aspects, including remote system monitoring and data logging, are also investigated.

The objective is to provide a robust resource for academics, researchers, students, and design engineers interested in this field, enabling them to integrate theoretical knowledge with practical application skills. Each chapter within the book has been meticulously designed to facilitate a progressive understanding of the subject matter for the reader.

This work endeavors to contribute to the proliferation of sustainable energy technologies and the enhancement of energy efficiency.

It is a pleasure to accompany you on the journey towards more effective utilization of the sun's boundless energy.

Sincerely,

June-2025

Assoc. Prof. Dr. Erkan KACAN

Contents

Preface iii

1. Introduction 1

2. Use of Solar Tracking Systems (STS) in Concentrating Solar Power (CSP) 9

3. Patents in the Field of STS 17

4. STS Using Image Processing Methods 23

4.1. How Does the Image Processing Method Work? 24

4.2. How is Solar Tracking Performed with Image Processing Method? 28

5. System Components 45

5.1. Servo Motors vs. Stepper Motors in Solar Tracking 46

5.2. Linear Actuators in Solar Tracking 47

5.3. Slew Drive in Solar Tracking 53

5.3.1. Worm Gear Slew Drives 55

5.3.2. Spur Gear Slew Drives 55

5.3.3. Single-Axis Slew Drives 56

5.3.4. Dual-Axis Slew Drives 57

5.3.5. Enclosed Slew Drives 58

5.3.6. Open Slew Drives 58

5.3.7. Slew Drive Selection Criteria and Design Analysis 58

5.3.8. Slew Drive Selection: A Case Study "Calculation Method for Solar Tracking Systems" 62

6. Algorithm Samples and Microcontroller Connections 69

6.1. Hardware Integration and Data Processing in Raspberry Pi-Based Solar Tracking Systems 70

6.2. Image Processing-Based Algorithms in Solar Tracking Systems 72

6.2.1. Basic Steps of the Image Processing Algorithm 72

6.2.2. Image Processing Algorithm Samples 77
6.2.2.1. Sample Code-Main Body 77
6.2.2.2. Transitioning from Hysteresis-Based Control to PID-Based Control 84
6.2.2.3. Integration of Adaptive Thresholding: 89
6.2.2.4. Steps to Improve Sun Detection: 91
6.2.2.5. Cases Where Sun Is Not Detected- Astronomical Algorithm Integration: Calculating Sun Position: 94
6.2.2.6. Situations Where the Sun Is Not Detected - Fixed Angular Movement and Park Position 103
6.2.2.7. User Interface and Data Logging 107
7. Results and Discussion 121
References 125

CHAPTER 1

1. Introduction

Solar collectors are components of solar energy systems that convert the energy received from the sun into usable energy, and they possess various design parameters. Generally, the conversion performed by collectors is in the form of heat and electrical energy. In solar lighting applications, however, transmission or light conveyance occurs without any conversion. Solar collectors located on the Earth's surface are in dynamic interaction with the Sun. Therefore, characteristics of solar energy systems such as solar geometry, orientation, and surface area are very important data in determining this interaction.

"Solar Tracking Systems (STS)" are used to continuously and dynamically position the collectors according to the sun's location in order to gain maximum energy. The tracking systems of solar collectors have made significant progress, especially with the development of sensors, transducers, and microelectronics.

Solar energy systems would be grouped internally as low, medium, and high temperature applications, as well as PV-electricity generation. Collector models shown in Figure 1 are used while applying these methods.

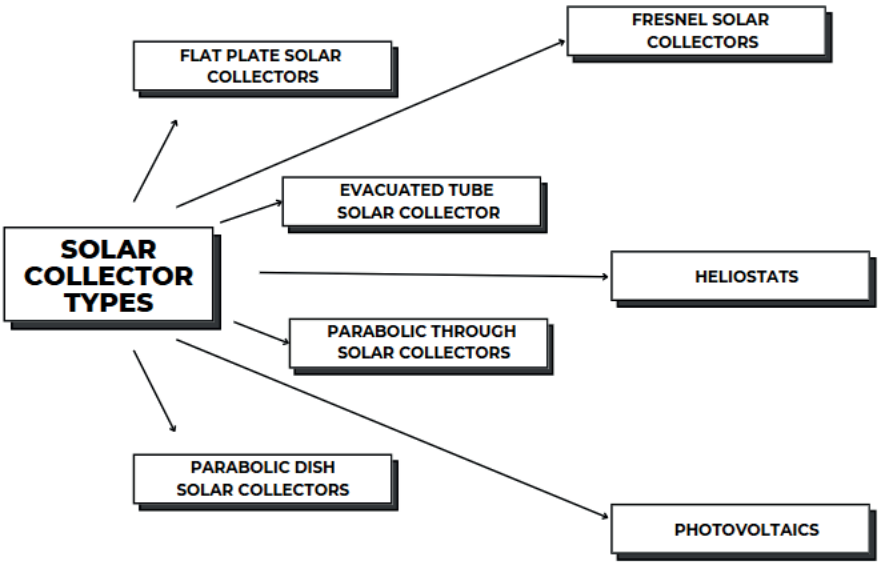


Figure 1 Solar Collector Types Used in Solar Energy Systems

Solar collectors can be classified according to their concentration ratio into non-concentrating, linear (line-focusing), and point-focusing types. Depending on the intended use, solar energy applications can be classified as shown in Figure 2. The main purpose of all these applications and collector types is to obtain useful energy and the amount of uselful energy depends on the efficiency of the application, the amount of solar radiation and the solar geometry. Receiving solar beams at surface normal, continuously, and at high intensity are the most important factors that increases the amount of useful energy. Therefore, factors such as duration of sunshine, surface orientation, and shading factors are taken into consideration when determining the application locations of solar energy systems.

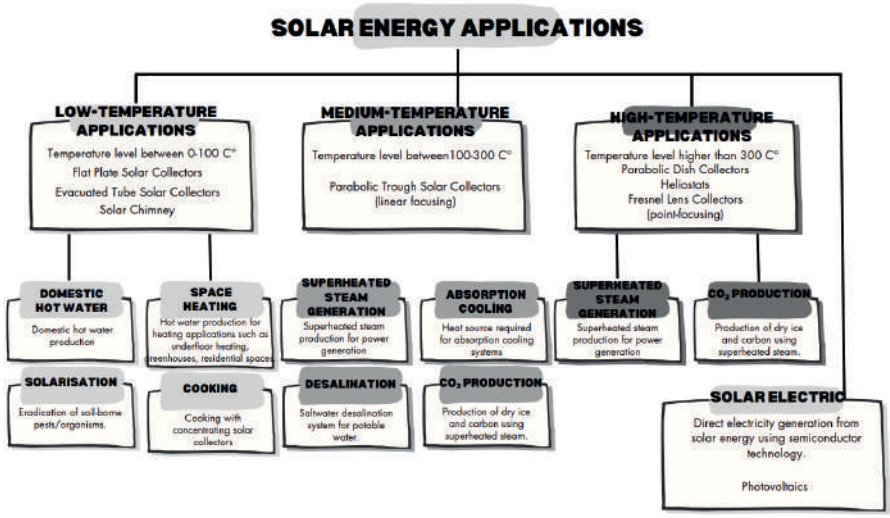


Figure 2 Classification of Solar Energy Applications

Various methods and strategies have been developed to ensure that collectors receive solar rays perpendicularly. The most effective method is the use of auxiliary equipment that tracks the sun on the east-west and solar elevation axes. Solar tracking systems are generally divided into two groups: single-axis and dual-axis tracking systems. Dual-axis solar tracking systems can also be examined in two groups: polar (equatorial) tracking and azimuth/elevation tracking methods. Today, with technological advancements, solar tracking systems are produced by using different methods.

The first known examples of solar tracking systems were made by Finster in Chile in 1962. However, this system worked entirely in manual mode. One year later, in 1963, Saavedra developed an electronic mechanism for the control of the Eppley pyrheliometer [Roth, Georgiev, & Boudinov,2005].

Neville (1978), demonstrated the difference between the insolation of a sun-tracking surface and a fixed surface. This study is important as it is among the first to determine the useful energy obtained from fixed-positioned solar collectors and sun-tracking collectors. As seen in Figure 3, the data obtained from three different collectors were compared; it was shown that the highest value was obtained from the system that tracks the sun on two axes, followed by the system fixed at the latitude angle and tracking the sun on the east-west axis, and finally the system fixed at the latitude angle facing due south [Neville, 1978].

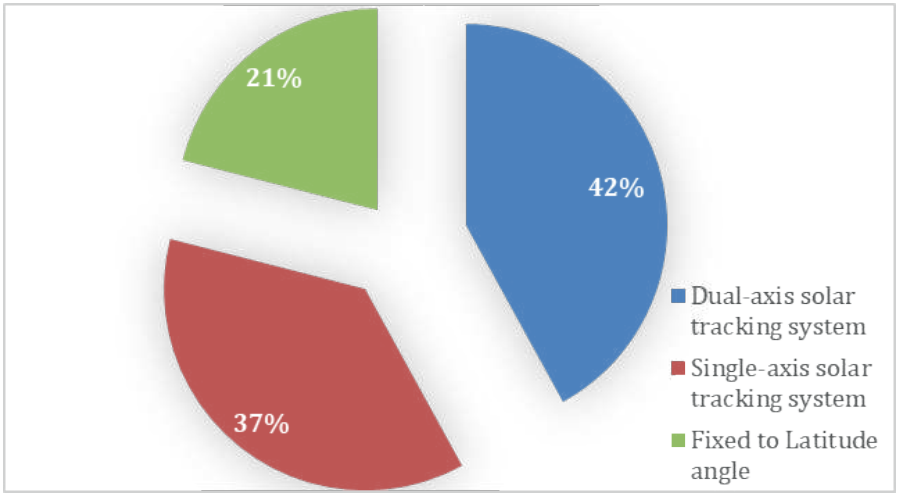


Figure 3 Comparison of Fixed Collectors and Sun-Tracking Collectors

Hession and Bonwick (1984), developed a solar tracking system that could be used with many solar collectors and platforms. Although the system, which detects the sun using phototransistors, had some errors, it provided successful results in solar tracking applications by solving these errors and consuming only 1W of energy [Hession PJ & Bonwick , 1984].

Schubnell and Ries (1990) published an approach to control the angular speed of the tracking system. This study specifically focused on the accuracy of solar tracking in concentrating solar energy systems. Accordingly, the maximum tracking error defined as a 10^{-4} error rate (1 cm/100 m) was expressed as a time-dependent numerical value for the worst-case scenario. While this value is 1.4 seconds in commonly used tracking systems, it was found to be 6.4 minutes in the tested system, which correctly oriented a 51.8 m^2 heliostat glass surface [Schubnell M & Ries , 1990].

Davies (1993) focused on a tracking system for concentrating solar collectors. In his study, he developed a method that considers situations where the equatorial plane is perpendicular to the ecliptic plane. It was assumed that the sun follows an approximately circular trajectory on the cross-sectional plane of the tracked surface. Therefore, it was claimed that solar tracking could be done accurately when moving at a constant speed. Experimental observations reported errors of ±2° [Davies PA, 1993].

Hirata and Tani (1994, 1995) developed the "spectral tracking method" to maximize output of photovoltaic collectors. The energy produced by

polycrystalline silicon PVs and amorphous silicon PVs, which were exposed to different regions of the solar spectrum by being mounted on the solar tracking system, was examined. The experiments showed that polycrystalline PVs produced more stable results than amorphous silicon PVs [Hirata Y & Tani, 1994] [Hirata Y & Tani, 1995].

Ünüsaçar and Taşer (1994) developed solar tracking systems to maximize the useful energy gained from solar radiation and achieved high levels of thermal energy in experiments conducted in May and June [Ünsaçar F & Taşer 1994].

Barakat et al. (2001) conducted studies on multi (dual) axis solar tracking systems and examined the effect of these systems on the amount of energy obtained from PV. When a tracking system controlled by complex dual-axis electronic circuits was used, a 20% more effective result was obtained compared to a single-axis tracking system [Barakat et.al., 2001].

As shown in Figure 3, the difference in obtained energy between single-axis and dual-axis solar tracking has increased from 5% to values up to 20% over time.

Garrison (2002) worked on a program to determine the energy gains of both fixed and sun-tracking collectors based on the physical properties of 15 different solar collectors. The FORTRAN program named SOCOL processed data from 239 national meteorological stations within 20 seconds, successfully calculating parameters such as the surface temperature and instantaneous energy gain of a collector at a specific location [Garrison, 2002].

Hein et al. (2003) used parabolic reflector surfaces that concentrate the rays 300 times by tracking the sun. The concentrated solar rays were directed onto PVs to generate electricity. The effect of single-axis solar tracking on energy yield was investigated. At a concentration ratio of 200x, the best efficiency value was obtained from GaAs PVs at 26% [Hein et.al., 2003].

Abdallah (2004) examined the effects of four different solar tracking systems on the current-voltage characteristics and electricity generation of PV systems. The solar tracking systems were divided into four groups: dual-axis, single-axis vertical, east-west axis, and north-south axis. Compared to 320 fixed-tilted PV systems in Amman, Jordan, the highest efficiency increase was observed in the dual-axis tracking system at 43.87%. For the other axes, the efficiency increase was 37.53% for the east-west axis, 34.43% for the single-axis vertical, and 15.69% for the north-south axis, respectively. As

shown in Figure 4., the efficiency difference between single-axis (east-west axis) and dual-axis solar tracking systems is about 6.5% [Abdallah, 2004].

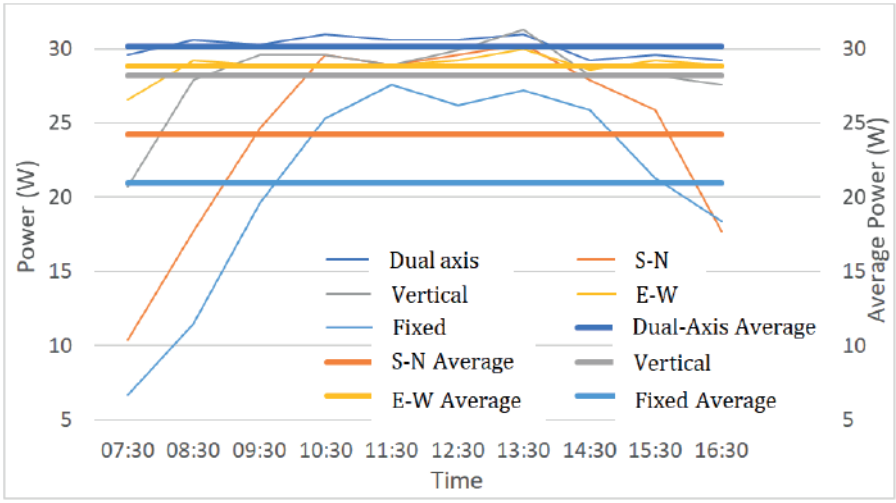


Figure 4 Effect of Different Solar Tracking Systems on the Power Output Obtained from PV Modules [Abdallah, 2004]

Roth et al. (2004, 2005) conducted studies and tests on a solar tracking system. This tracking system enabled the automatic measurement of direct solar irradiation with a pyrheliometer. The mechanism was designed unattached to the control unit via the digital program within the control system. The system calculated the sun's position and stored it in a database for future analyses [Roth et. al, 2004] [Roth et. al, 2005].

Alata et al. (2005) developed methods for solar tracking using three different approaches. Accordingly, the Sugeno Fuzzy inference system was used in single-axis tracking with the aperture area adjusted to the latitude, dual-axis equatorial tracking, and dual-axis azimuth/elevation tracking methods. This study is one of the first to use machine learning in solar tracking systems [Alata et.al., 2005].

Bingöl et al. (2006) designed, implemented, and tested a microprocessor-based dual-axis tracking system in the study. They used a light-sensitive sensor, a step motor as the actuator, and a microprocessor [Bingol et.al., 2006].

Abu-Khader et al. (2008) examined the effect of using multi-axis solar tracking systems, suitable for Jordan's climate parameters, on energy gains.

The algorithm of the solar tracking system is based on calculating the surface azimuth angle and zenith angle with time-dependent models. According to the results of the study, more effective data were obtained (between 30-45%) when a north-south axis tracking system was used compared to a fixed position with a 32° tilt angle. It was revealed that installing north-south axis tracking systems for PV systems to be established in Jordan is more effective than fixed and east-west axis tracking systems [Abu-Khader et.al., 2008].

Chong and Wong (2009) aimed to reduce solar tracking errors with their work on the mathematical modeling of axial solar tracking systems. The mathematical method developed in this study was compared with other methods. Tracking errors occur as a result of calculating the sun's position relative to a collector on the Earth's surface according to the triple vector in the Stine-Harrigan model as $\begin{bmatrix} S_M \\ S_E \\ S_P \end{bmatrix} = \begin{bmatrix} cos\delta cosw \\ -cos\delta sinw \\ sin\delta \end{bmatrix}$. Mathematical methods were used to minimize the error in the calculation of this vector system, and the results were examined [Chong & Wong 2009].

Mousazadeh et al. (2009) researched the working principles of solar tracking systems and published a "review" by compiling the studies they obtained as a result of this research. This review included classifications in which the effect of solar tracking systems on the useful energy obtained from the sun was examined. For small solar energy applications, although the energy consumed by the solar tracking system varies between 2-3% of the energy gained, they stated that usage of solar tarcking system does not provide a significant benefit. As a result of the review, it is determined that the most efficient and common solar tracking systems are the axial-polar tracking system and the azimuth-elevation tracking system [Mousazadeh et.al. 2009].

Sungur (2009) developed a multi (dual) axis solar tracking system controlled by PLC units for PV systems in Konya and conducted an experimental study. According to the experimental results, it was revealed that in Turkey's conditions, a collector with a solar tracking system is 42.6% more efficient than a fixed collector [Sungur C., 2009].

Sefa et al. (2009) developed a single-axis solar tracking system in Turkey and worked on the benefits provided by this system. The system provided solar tracking with a simple mechanism based on a simple micro-processor and yielded more effective results compared to fixed-positioned solar collectors [Sefa et.al, 2009].

Cruz-Peragón et al. (2011) investigated the energy gain of two-axis sun-tracking solar collectors compared to fixed-positioned collectors and their suitability for Spanish conditions. Accordingly, analyses were carried out based on the climate parameters of 52 different locations in Spain, and it was generally found that an energy efficiency of over 20% was achieved [Cruz-Peragón et.al., 2011].

Seme and Stumberger (2011) developed a two-axis solar tracking system with a new mathematical modeling and worked on its results. Accordingly, in the study where optimum tilt angle errors were revealed, an optimization method they called the "differential evolution method" (a type of random search algorithm) was used. The objective function could not be determined according to the differential evolution optimization method. Therefore, a time-dependent tracking algorithm was created [Seme & Stumberger, 2011].

Lubitz (2011) evaluated hourly data from the typical meteorological year of 217 geographical regions in America and revealed errors and differences in optimum tilt angles. For this, calculations were made on fixed-positioned systems, azimuth tracking systems, and two-axis sun-tracking systems. For fixed surfaces positioned at the latitude angle facing south to maximize the amount of solar irradiation, an error of 14° was observed, especially on days with a high clearness index in the northwest regions. Compared to a fixed surface, it was found that 29% more solar irradiation is possible with the use of an azimuth tracking system, and 34% more with the use of a two-axis tracking system [Lubitz W. D., 2011].

CHAPTER 2

2. Use of Solar Tracking Systems (STS) in Concentrating Solar Power (CSP)

A large part of solar tracking systems has been developed experimentally on PV systems. However, although solar tracking is mandatory in concentrating solar collectors, experimental analyses have remained limited. As seen in Figure 5, there has been an increase in studies on concentrating solar energy systems in recent years according to WOS (Web of Science) data. However, factors such as the temperatures reached in these set-up, superheated steam, high light intensity, etc., make the work difficult and risky. Instead, it has been preferred to analyze moving systems in relatively lower-risk flat collectors (such as PV, PV-T, flat plate solar water heaters).

Studies on the analysis of solar tracking systems in linear focusing parabolic trough and point focusing parabolic dish, Fresnel, and Heliostat systems have remained more limited.

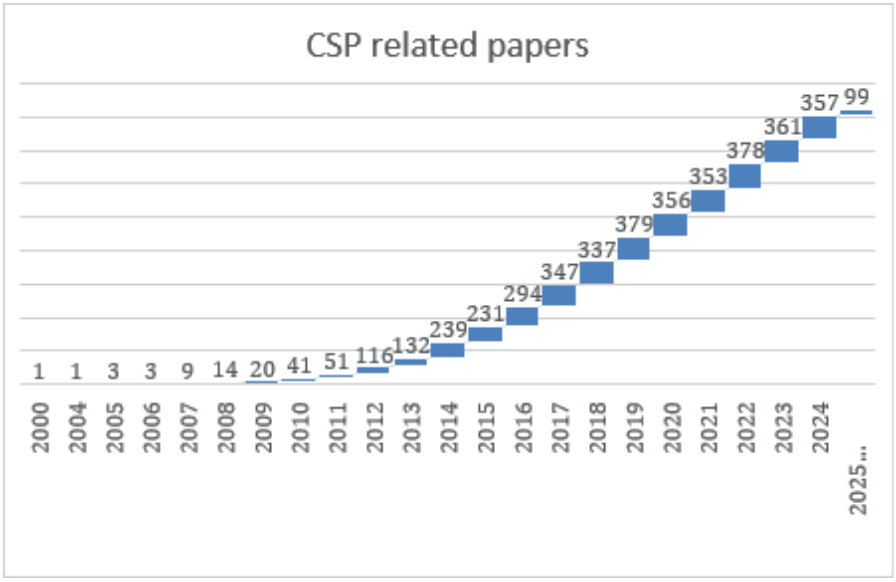


Figure 5 The Number of Academic Studies Related to Concentrating Solar Tracking Systems

These studies have paved the way for the utilization of solar tracking systems in parabolic trough solar collectors.

Gee (1980) examined the tracker types and operational systems of linear-focusing solar concentrators. The study compared different tracker types and evaluated relevant experimental research and advancements [Gee, 1980].

Cope and Tully (1982) investigated the sun-tracking strategies of concentrators using equations that allow for the calculation of the sun's position. They also compared tracking errors in existing concentrators with experimental values [Cope & Tully, 1982].

Heiti and Thados (1983) conducted research on the thermal efficiency and performance of cylindrical solar collectors. Their work explored the impact of the cylindrical aperture's orientation relative to the incident angles of solar radiation on efficiency [Heiti & Thados, 1983].

Hession and Boonwick (1984) tested tracking systems for concentrators of varying dimensions. They developed a light-sensitive circuit that precisely tracks the sun and provided its block diagram. The solar tracking system, which utilized phototransistors, was reported to exhibit some errors [Hession & Bonwick, 1984].

Eltez (1986) investigated the shaping of the reflective focusing surface in a fixed-reflector linear-focus tower project. The study conducted optical and geometric analyses of radiation and heat transfer on a spatial surface that enables linear focusing and reflection onto a receiver on the tower, without the need to move a large number of reflector arrays in response to the sun's daily azimuth and elevation changes [Eltez M., 1986].

Prapas et al. (1987) performed a detailed optical analysis of cylindrical concentrators using ray-tracing methods. They determined the percentage of diffuse solar radiation that can be utilized by this type of concentrator [Prapas et al., 1987].

Bhowmik and Kandpal (1988) conducted studies on cylindrical solar collectors that track the sun in the north-south, east-west, and all axes. Their work utilized different intra-year times, latitudes, and angles, and presented the corresponding graphical results [Bhowmik & Kandpal, 1988].

Yeşilata (1990) designed and manufactured a cylindrical solar concentrator that tracks the sun's movement. An experimental setup was created to determine the thermal efficiency of the concentrator, and the thermal efficiency of the manufactured solar concentrator was calculated using this setup [Yeşilata, 1990].

Eltez (1990) examined the movement systems and thermal characteristics of different concentrator types and provided various application examples. The study analyzed the energy needs of a textile factory and the potential contribution of a solar concentrator to these needs [Eltez M., 1990].

Pinazo et al. (1992) analyzed the incident angle of solar radiation on a cylindrical solar concentrator. They derived analytical relationships for the incident angles [Pinazo et. al., 1992].

İbrahim (1996) developed a solar tracking system for a set of six parabolic collectors and conducted experimental measurements. The study investigated the effect of fluid mass flow rate, ranging from 0.005 to 0.033 kg/s, on the collector efficiency. The highest collector efficiency was found to be 62% at a flow rate of 0.033 kg/s [Ibrahim, 1996].

Kalogirou (1997) worked on a tracking system capable of operating with single-axis solar tracking systems. The system utilized three light-sensitive sensors to determine the sun's state and position, thereby positioning the collector. One sensor detected whether the collector was focused, another detected cloud cover, and the third identified day or night to position the collector accordingly. Based on the assumption that the sun moves at a constant speed of 0.25 degrees per minute (dpm), the tracking accuracy

of the system varied with solar irradiation values. Deviations of 0.2° were observed below 100 W/m², while deviations of 0.05° occurred at irradiation levels around 600 W/m² [Kalogirou, 1997].

Khalifa and Al-Mutwalli (1998) investigated the impact of two-axis solar tracking systems on the thermal performance of integrated parabolic solar collectors. Parabolic solar collectors with sun-tracking capabilities yielded 75% more effective results in terms of thermal performance. This difference in thermal performance between sun-tracking collectors and optimally fixed collectors highlighted the importance of tracking systems in collectors with high concentration ratios [Khalifa & Al-Mutwalli, 1998].

Genç (1998) designed and manufactured a 3.70 m long cylindrical solar energy concentrator with a 40 mm focal diameter that tracks the sun on a single axis. The concentrator was enabled to track the sun on a single axis using a photocell. The performance experiments of the system were examined under the climatic conditions of Ankara. The tests conducted throughout the day yielded a collector outlet temperature of 75°C and an efficiency of 65% for an approximate 7°C inlet-outlet temperature difference [Genç, 1998].

Grass et al. (2004) worked on a comparison between a parabolic solar collector with a vacuum absorber surface, equipped with two new tracking systems for sun tracking, and a vacuum tube flat plate solar collector with a low concentration integrated parabolic collector. It was concluded that vacuum low concentration integrated parabolic collectors positioned in the east-west direction and fixed at the latitude angle were more suitable for applications up to 200-250 °C. At higher temperatures, thermal losses from the absorber surfaces were found to be significant. To address the fact, an industrial product was developed with a reduced absorber surface area but increased thermal conductivity, and a tracking system that reduced deviation in solar incidence angle was used, resulting in more effective outcomes in 300-350 °C applications as seen in Figure 6.

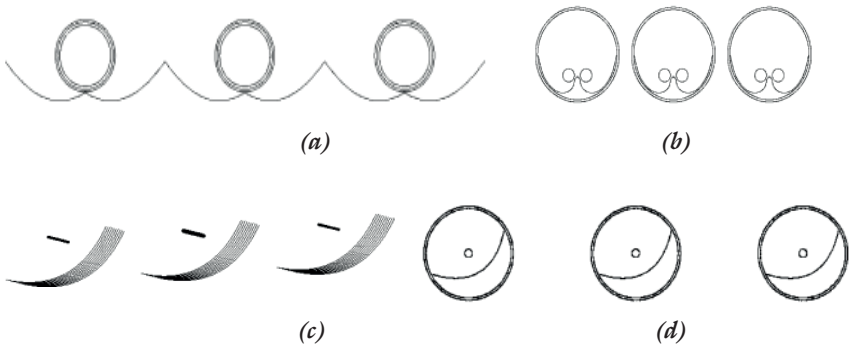
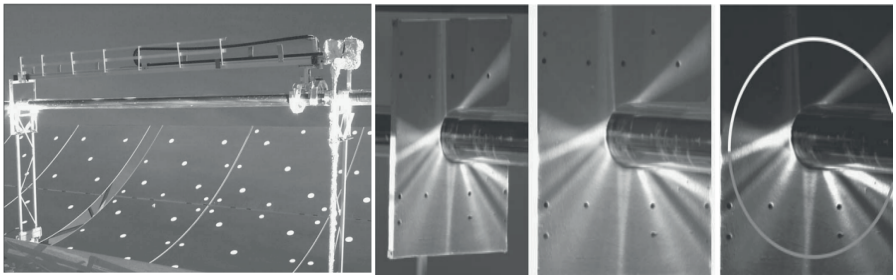


Figure 6 Low concentration parabolic trough collector designs [Grass et al., 2004]

Bakos (2006) studied a two-axis continuous solar tracking system for a cylindrical collector. The energy collected by the collector was measured and compared with a fixed-surface collector tilted 40° southward. The results indicated that the moving two-axis sun-tracking collector collected 46.46% more energy than the fixed collector [Bakos, 2006].

Riffelmann et al. (2006) examined the optical efficiency of cylindrical trough solar power plants to ensure the desired quality. Accordingly, they developed two methods to measure the solar flux in the focal region of the system: Parascan (Figure 7(a)) and the camera-target method. Parascan is an advanced solar flux density measurement device. By moving the device along with the receiver axis, they measured the flux distribution in front of and behind the receiver surface. The measurements allowed for the calculation of the interception factor and optical property analyses of the system around the receiver. The camera-target method (Figure 7(b)) involved taking pictures of the diffuse radiation around the receiver with a calibrated camera. The target around the receiver intercepted direct rays. By examining the reflected rays and the captured images, optical errors were determined.



(a) *(b)*

Figure 7 (a) Parascan image mounted on a Eurotrough collector (b) Camera-target method image of diffuse radiation on a vertical surface [Riffelmann et al., 2006]

Agee et al. (2007) conducted a study investigating the market trends, application areas, costs, and maintenance expenses of solar tracking systems. Research on different types of tracking systems (hydraulic control, program-based control, and sensor-based control) indicated that hydraulic tracking systems yielded the most effective results for low-capacity applications [Agee et al., 2007].

Al-Soud et al. (2010) designed, implemented, and tested an automatically sun-tracking parabolic cooker. The tests were conducted continuously for three days in 2008, and they reported that the collector reached 90 °C when the ambient temperature was around 36 °C [Al-Soud et al., 2010].

Tang and Yamei (2010) discussed solar collectors equipped with a single-axis three-position solar tracking system in their work. They named their collectors, which they described as a "new concept," 3P-CPCs and performed theoretical analyses by connecting the solar tracking system to PV modules. The 3P-CPCs achieved 1.26-1.45 times more energy compared to single-position systems [Tang & Yamei, 2010].

Sansoni et al. (2011) worked on a prototype to be used in a parabolic trough solar power generation plant to be built in the Florence region of Italy. Solar tracking and optical characterization of collectors connected in a line were performed. Solutions for axis deviations, angular distortions, and mirror deformations that occurred during the experiments were discussed. As shown in Figure 8(a), collector efficiency did not show a significant reaction to an angular deviation of ±1°, while it decreased by 22-25% at an angular deviation of ±1.5°. Figures 8(b)-(c) and (d) illustrate the effect of angular deviation on collector efficiency depending on the focal length of

the collector, the deviation value of the absorber surface from the focus, and the absorber surface diameter.

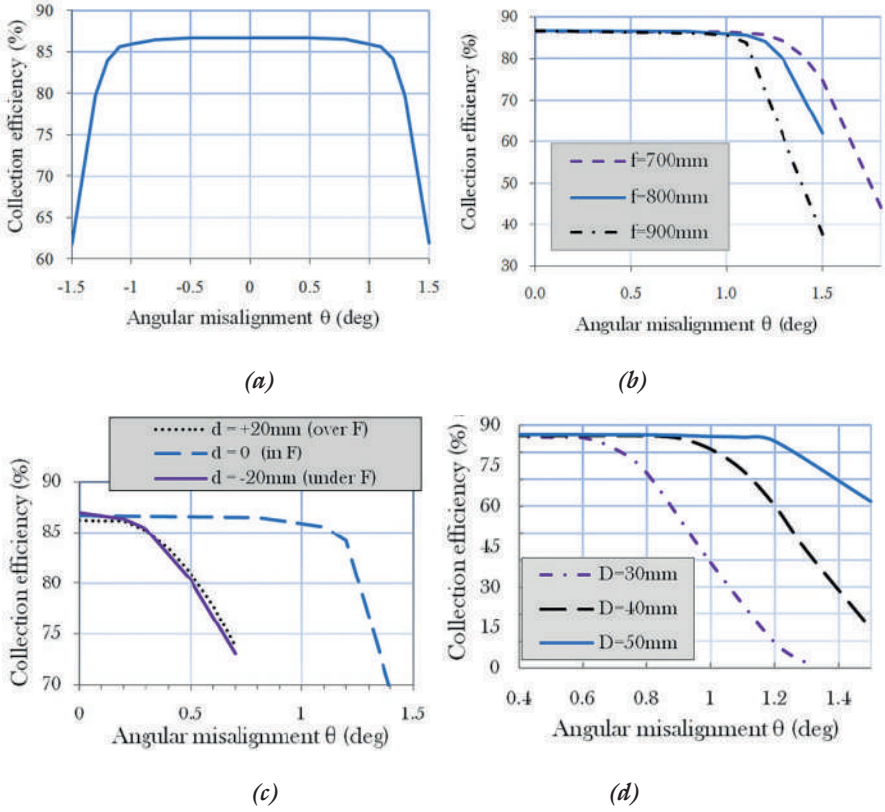


Figure 8 (a) Relationship between angular deviation value and collector efficiency (b) Relationship between angular deviation value and collector efficiency depending on different focal lengths (c) Relationship between angular deviation value and collector efficiency depending on the deviation of the absorber surface from the focal point (d) Relationship between angular deviation value and collector efficiency depending on the absorber surface diameter [Sansoni et al., 2011]

Pei-Ying et al. (2011) focused on a concentrating solar tracking system that had not been previously studied. Accordingly, they proposed that tracking the focal point, which is relatively smaller and lighter, would be advantageous compared to moving large, heavy masses for solar tracking. The sun-tracking absorber surface was defined as the "focal image," and it was observed that this image moved along an interesting curve during solar tracking. The results showed that the sun-tracking focal point was comparable to other conventional methods and effective in energy conversion [Pei-Ying et al., 2011].

CHAPTER 3

3. Patents in the Field of STS

In addition to the aforementioned scientific studies, patent applications have been filed for developed systems, and these patent applications have been commercialized. However, a significant portion of the patent applications for solar tracking systems are designed for flat-panel PV collectors or PV strings/mounting elements and are primarily developed for use in solar power plants. Although the fundamental principle of solar tracking systems is to ensure that solar rays flow through normal of the aperture surface, there are minor and sometimes significant differences between the tracking systems of flat-panel collectors and parabolic collectors. Particularly for large-volume collectors, considering movement and friction loads, differences are observed in gear and motor components compared to small-volume systems. However, the operating principle remains the same for solar sensors and electronic components.

Neale (1979) presented one of the early patents concerning tracking systems for concentrating solar collectors. This work involved the movement of large parabolic collector arrays that track the sun, as well as the design of a sensor that determines the sun's position. The collector arrays were connected in parallel, and solar tracking was achieved via electronic circuits from a single point of actuation [United States Patent- Sunpower Systems Corporation, Patent No. 877-077, 1979].

Butler (1984) introduced the design of a "pivot solar tracking system" through an application to the US Patent Office. According to this design, multiple collector rows, regardless of whether the collectors have a flat or cylindrical structure, could track the sun with a single center of motion

[United States Patent-United States Department of Energy Patent No. 192,799, 1984].

Warrick (2000) applied to the US Patent Office for a solar tracking system designed for the movement of large platform flat-panel collectors, primarily PV panels. This novel model utilized three hydraulic arms to create a mechanism capable of supporting heavy constructions and the wind load generated by strong winds. Two of the three hydraulic arms always determined the direction of motion, while the third determined the angle of equilibrium, enabling two-axis solar tracking and the scanning of a 90° angle in solar elevation [(United States Patent-Amonix Inc., Patent No. 09/282-315., 2000].

Hilnes (2010) developed a design in his patent that involves placing tubular elevations with cellulosic walls on a water-based ground to ensure homogeonous wetting of the walls. In this design, since the wall wetted from the ground has homogenous moisture content, it stands upright and supports the solar cell, which is placed on the vertical cross-section at the top, in a vertical position. With sunrise, the surface exposed to radiation dries, the cellulosic wall shortens-contracts, and consequently, the upright cylindrical structure tilts towards the drying direction. The watery ground continuously creates moisture, and the surface facing the sun dries faster, resulting in a tilt towards the direction of the sun. This method allows for tilting towards the sun's direction, but information on achieving the optimum tilt angle or the precision of the tracking is not provided [United States Patent, Patent No. US 7,799 987 B1, 2010)].

Grushkowitz et al. (2018) developed a gear system that allows for the collective movement of PV strings. According to this design, the moving part in the main construction is mounted on the moving hub of the gear mechanism, creating a synchronous movement system [United States Patent Patent No. US 2018/0062564 A1., 2018].

Grushkowitz et al. (2018) designed a metal construction called a "torque tube" suitable for their patented gear system, creating a structural component that allows PV strings to rotate simultaneously at the same angle [United States Patent, Patent No. US 2018/0062566 A1., 2018].

Schimelpfenig et al. (2018) developed a new method to address the issue of PV array shading or varying tracking errors caused by different surface and wind load conditions in large solar power plants. According to this method, independent gear motors were placed between the strings, allowing the lines in between to have different angular rotations [United States Patent, Patent No. US 2018/0175783 A1., 2018].

Rosedale (2019) developed a sun-tracking umbrella intended to provide shade for the user. Simultaneously, they succeeded in obtaining the energy required for remote control and movement mechanisms thanks to the PV panels placed on the umbrella's surfaces [United States Patent, Patent No. US 2019/0069652 A1., 2019].

Almy and Jensen (2019) added a dampening extension to the torque tube to reduce solar tracking errors caused by strong wind loads on PV strings connected to the torque tube [United States Patent Patent No. US 10,340,839 B2., 2019].

Sharpe (2021) developed a gear system for multiple PV clusters to track the sun on two axes from a single carrier. They accordingly operated a two-axis moving gear mechanism and the carrier leg that supports the PV cluster together. The tracking mechanism of the system performs tracking based on the interaction between GPS, anemometer, snow sensor, and communication transducers placed on the PV cluster. Based on the data from the sensors, it calculates the sun's location and the direction the system should face, and then sends movement commands to the gears [United States Patent, Patent No. US 2021/0194417 A1., 2021].

Poviet (2022) obtained a patent based on the principle of forming "canopies" used as shading elements in vehicle parking areas from PV panels and having the shading roof track the sun. Accordingly, rotating elements that allow the shading roof, which has at least two legs, to rotate in two directions around the axis of its legs are placed. A two-stage control element that decides how much this rotation should be is installed. First, the angle that will capture the optimum tilt angle according to the sun's inclination is calculated, then the maximum angle that the canopy can tilt (depending on whether there is a vehicle underneath or not) is calculated, and based on these two angle values, the PV roof is oriented towards the sun [United States Patent, Patent No. US 2022/0182009 A1., 2022].

Askins et al. (2022) obtained a patent on a passive solar tracking system. According to this, the surface intended to track the sun is placed on a carrier pipe, and open-mouthed reflective surfaces that allow sunlight to enter are placed on the sides of the carrier pipe. In this way, the incoming rays are reflected and reach the liquid mechanism on the ground. The liquid on the lower surface expands due to the incident radiation, enabling it to actuate the hydraulic mechanism that directs the carrier pipe. Thus, since more radiation will enter from the surface facing the sun, more expansion will occur, and the hydraulic mechanism will direct the carrier block in that direction [United States Patent, Patent No. US 11,431,287 B2., 2022].

Albinmousa et al. (2022) patented a system that performs solar tracking using two-axis hydraulic control elements. They created a control mechanism with a logic very similar to microprocessor-based solar tracking systems. After the sun's location is detected with light sensors, the pressure drops and rise from the gear heads on the vertical or horizontal axis are monitored with the rotation of the gears. In this way, the power transmission for the system's rotation is achieved [United States Patent, Patent No. US 11,466,900 B2., 2022].

Shtein et al. (2023) developed a multi-axis solar tracking system with a construction based on the art of cutting and folding known as "krigami" and "origami." By using the logic of krigami, the main carrier was obtained by cutting small, thin materials and mounting pre-calculated and memorized carrier layers one after the other. A lens was placed inside this carrier, ensuring that the sunlight falls perpendicularly on the photovoltaic cell at its focus. During the sun's movement, the memory-equipped (angled during cutting) carrier layers opened sequentially, positioning the carrier construction according to the sun's location, thus folding and perpendicularly directing the radiation incident on the lens onto the cell at its focus. Since the logic of krigami was based on, it became possible to produce lighter and more durable carrier constructions, and by combining adjacent focusing cells, the production of panels where the cells track the sun became feasible [United States Patent, Patent No. US 11,831,272 B2, 2023].

Bapat et al. (2024) have patented a torque tube design that connects PV strings, allowing them to rotate together under a single centralized command. According to this design, appropriate connection points are dimensioned on the rotational gear at the axis of rotation to enable multiple torque tube connections [United States Patent No. US 12,003,208, B2, 2024].

Nicolas et al. (2024) have patented a unique crank gear design that enables the collective rotation of PV strings. In their design, they created a motion mechanism by mounting a crank gear to a slew drive gear, allowing for the generation of precise rotational torque with small movements. Accordingly, the torque tubes connected to the PV strings are linked to two crank gears, accurately transferring the rotation received from the slew drive to the rotation of the arrays [United States Patent Patent No. US 2024/0243693 A1., 2024].

Cha et al. (2024) have patented a dome-shaped structure onto which they affixed elastic PV cells connected by fiber metals. They measured the surface temperatures of the PV cells and expanded the hotter surface,

thereby increasing the amount of surface area directly facing the sun's rays. This design allows for the expansion of the dome structure's sun-facing surface, enabling more PV cells to receive sunlight perpendicularly. They also developed a motion mechanism that increases the surface area according to the sun's position in the morning, noon, and evening hours by pre-programming the surface design [United States Patent, Patent No. US 2024/0322745 A1., 2024].

CHAPTER 4

4. STS Using Image Processing Methods

Solar tracking systems (STS), historically implemented using sensors like LDRs, phototransistors, PV cells, and surface elements with varying expansion coefficients, or through constant angular rotation, have become intensely researched topics over time. Additionally, solar tracking systems developed with PLC and microprocessor coding have gained significant prominence in the literature. These methods each possess distinct advantages and disadvantages relative to one another. The continuous advancements in sensor and microprocessor technology ensure the ongoing progress of these methods.

In large-scale solar power plants, solar tracking systems are generally not preferred due to the high frequency of breakdowns and the substantial costs associated with movable mechanisms. Investors often opt to allocate funds towards more fixed solar collectors rather than movable mechanisms and constructions. However, in situations where space is limited and in Concentrating Solar Power (CSP) applications, the use of solar tracking systems becomes a necessity. Beyond that, the precision of solar tracking is particularly crucial in point-focusing solar energy systems. While the importance of solar tracking accuracy is less pronounced in applications such as PV systems, flat-plate solar collectors, and solar cooking systems, it plays a critical role in solar lighting, parabolic trough, and parabolic dish collector applications. Especially in solar lighting systems, precise adjustment of the focal point has a significant effect on parameters such as light intensity, color, and illumination level of the lighting output. Therefore, developing a precise solar tracking system is imperative for collectors to receive solar radiation at a perpendicular angle.

In recent years, errors arising from sensor malfunctions, calibration degradations, programming deviations, and changes in atmospheric conditions have been addressed in various studies in the literature. Solar tracking using image processing methods stands out as a developed solution to overcome these issues. By modeling the principles of the human eye, images of the sky are processed and converted into meaningful data, enabling microprocessors to detect the Sun.

4.1. How Does the Image Processing Method Work?

Image processing is a series of techniques and methods that take a digital image as input and process it through specific algorithms to either enhance its properties (e.g., contrast enhancement, noise reduction) or extract specific information from it (e.g., object recognition, feature detection). At its core, it relies on mathematical-statistical operations, which have been translated into code in programming languages (such as Python, C++, and MATLAB). Many ready-made libraries like OpenCV-JavaCV have been developed for this purpose.

The image processing process involves the steps shown in Figure 9:

1. Image Acquisition,
2. Image Pre-processing,
3. Image Segmentation,
4. Feature Extraction,
5. Classification/Recognition,
6. Image Analysis and Interpretation.

Image Acquisition: A digital image is obtained through a camera, scanner, or another imaging device. This image consists of small units called pixels. Each pixel typically has numerical values containing color and brightness information.

Image Pre-processing: The acquired image may undergo various preliminary operations to prepare it for subsequent stages. The goal of this stage is to improve image quality, reduce noise, enhance contrast, or perform geometric corrections. Common pre-processing techniques include filtering (blurring, sharpening), histogram equalization, geometric transformations (scaling, rotation), and color space conversions.

Image Segmentation: In this stage, the image is divided into meaningful regions or objects. The aim is to separate objects of interest from the

background in the image. Various segmentation algorithms are used, such as thresholding, edge detection, region growing, and clustering.

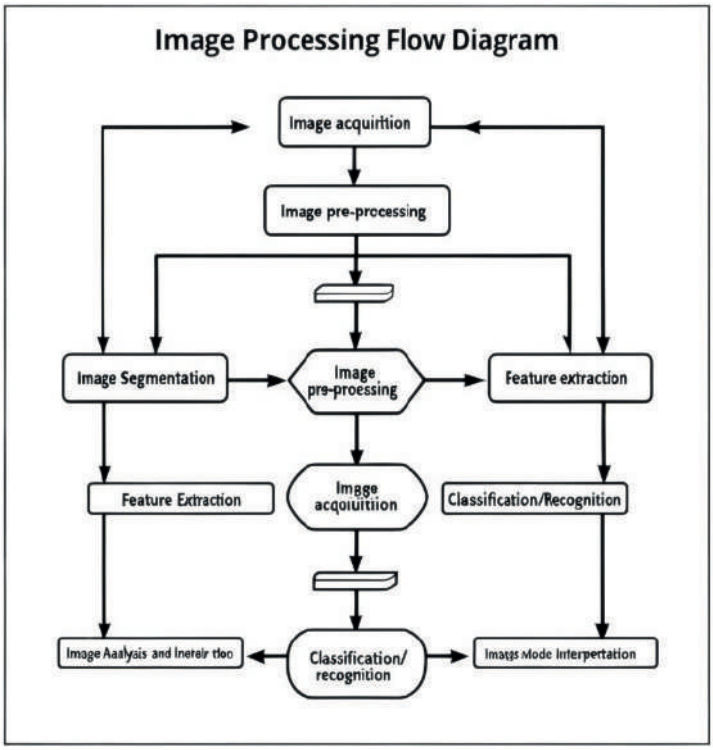


Figure 9 Image Processing Method Flowchart

Feature Extraction: Meaningful features that can be used to identify objects or patterns are extracted from the segmented regions or directly from the processed image. These features can include color, shape, texture, corner points, or more complex descriptors (e.g., SIFT, HOG).

Classification/Recognition: Using the extracted features, objects or patterns in the image are assigned to predefined categories or recognized. Machine learning algorithms (e.g., support vector machines, artificial neural networks) are frequently employed in this stage.

Image Analysis and Interpretation: In the final stage, meaningful conclusions are drawn and interpretations are made about the image using the classified or recognized objects and the extracted information. This can encompass various applications, from locating a tumor in a medical image to detecting suspicious activity in a security camera feed.

Figure 10 Image Analysis and Object Detection Steps with Image Processing Methods

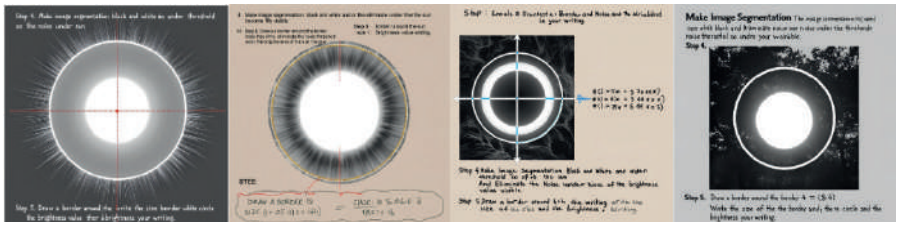


Figure 11 Object Detection and Edge Determination Examples

Figures 10 and 11 illustrate the stages of object detection using image processing, where new images are derived from an existing image. Codes available in libraries of programming languages such as Python, C++, MATLAB, and Java facilitate the detection and boundary determination of the sun within a sky image, as well as its recognition by distinguishing it from similar or characteristic "fake" objects in its surroundings.

The foundations of digital image processing were laid in the 1920s with efforts to transmit photographs via telegraph. These early systems made transmission possible by digitizing images into pixels and converting them into numerical values.

Significant strides have been made in image processing with the advent of computers. Space research, medical imaging, and military applications, in particular, catalyzed developments in this field. Work at the Jet Propulsion Laboratory (JPL) pioneered the enhancement and analysis of raw images obtained from spacecraft. The first digital images of the lunar surface were processed using fundamental image processing techniques such as noise reduction and geometric correction. In the medical field, the development of imaging methods like X-ray and computed tomography (CT), and the subsequent computer-based processing of these images for enhanced interpretability, represent early applications of image processing.

The image processing method has been a subject of extensive academic research. Roberts (1961)'s edge detection operator was one of the first systematic approaches to identify edges in an image. He successfully managed to control the reflections of 3D world edge lines on 2D images and the transformations occurring during this process [Roberts, 1961].

Sobel (1970)'s edge detection operator was a more advanced approach compared to Roberts's operator and demonstrated greater robustness against noise [Sobel, 1970].

Otsu (1979)'s automatic thresholding method became a fundamental algorithm in image segmentation [Otsu, 1979]. In subsequent periods, more complex image analysis and understanding techniques began to be developed. Topics such as morphological operations, texture analysis, and model-based object recognition gained prominence.

Canny (1986)'s optimal edge detection algorithm remains widely used today [Canny, 1986].

Haralick et al. (1973)'s texture analysis method based on gray-level co-occurrence matrices (GLCM) holds significant importance in the extraction of textural features [Haralick et.al, 1973].

With the advancements in computer sciences and machine learning techniques, revolutionary progress has been made in image processing. Deep learning approaches, in particular, have pioneered breakthroughs in areas such as object recognition, image classification, and semantic segmentation. Lowe (1999)'s Scale-Invariant Feature Transform (SIFT) algorithm enabled the extraction of features robust to scale and rotation changes in images [Lowe, 1999].

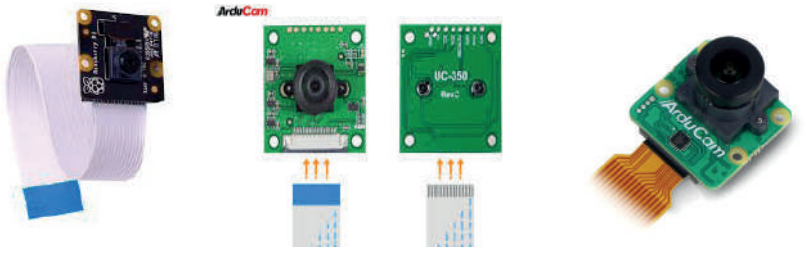
Viola and Jones (2001)'s AdaBoost-based framework for real-time face detection is a significant milestone in the field of object detection [Viola & Jones, 2001].

One of the pioneering works in deep learning, AlexNet (Krizhevsky et al., 2012), demonstrated the superior performance of deep convolutional neural networks (CNNs) in large-scale image classification tasks [Krizhevsky et al., 2012].

Today, image processing is an actively used and continuously evolving discipline across numerous fields, including artificial intelligence, computer vision, robotics, medicine, security, automotive, and many more. Thanks to new algorithms, more powerful hardware, and large datasets, image understanding and interpretation capabilities are improving daily.

4.2. How is Solar Tracking Performed with Image Processing Method?

While traditional tracking systems generally rely on position determination principles via astronomical algorithms or light dependent resistors (LDRs), image-processing-based systems offer more precise and dynamic tracking capabilities by directly detecting the Sun's disk from visual data. The system typically captures a digital image of a portion or the entirety of the sky at regular intervals using a wide-angle camera. This camera may feature a sensor capable of acquiring images across different spectral bands (e.g., visible light, near-infrared). The key characteristics sought in the camera are compatibility with the chosen control element and microprocessor, along with sufficient resolution for performing object analysis on the image. Figure 12 (a) shows an image of a camera used in solar tracking studies employing image processing methods in the literature. As seen in Figures 12 (b) and (c), cameras with electronic boards of various types, specifically 5MP OV5647 sensor-equipped CSI type cameras, are utilized.

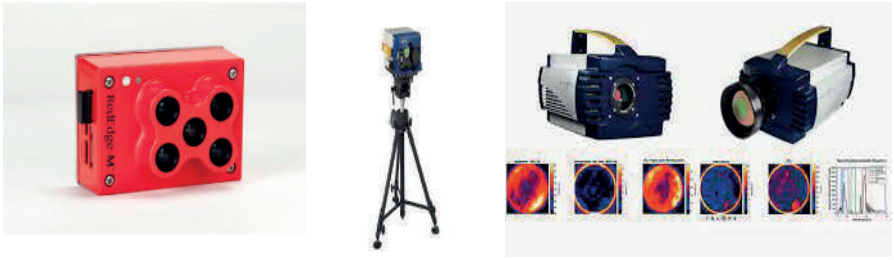


(a) *(b)* *(c)*

Figure 12 (a) Raspi Camera Module 3 NoIR (b)-(c) 5MP CSI Type Camera

The cameras shown in Figure 12 send the sun's image as raw data to the microprocessor, where the image features are processed by the program algorithm and the microprocessor. IR-filtered lens types of these camera modules are also beginning to emerge in the market. Widely used cameras capable of acquiring images across different spectra are available. Figure 13 (a) shows a Multispectral Camera, and Figures 13 (b) and (c) display examples of IR cameras. These cameras capture images across multiple wavelength ranges and are widely used in fields such as agriculture, environmental monitoring, and remote sensing.

Figure 14 shows an example of a "Hyperspectral Camera." Hyperspectral cameras are sensors capable of acquiring images in numerous narrow bands across a broad range of light wavelengths. These cameras collect spectral information associated with wavelengths at each pixel, which can then be used to analyze the physical and chemical properties of each object.



(a) *(b)* *(c)*

Figure 13 (a) Multispectral Camera (b)-(c) IR Camera

Figure 14 Hyperspectral Camera

For instance, spectral data obtained from these cameras is analyzed to determine chlorophyll levels in plants. Hyperspectral images are also used to detect pollution or habitat changes. Determining the chemical composition of materials is crucial in mining and materials science. Hyperspectral data can be analyzed to differentiate between various materials.

The use of Multispectral and Hyperspectral cameras is highly suitable for image processing applications. They are useful tools for determining the structural properties of tissues and cells, and for making predefined diagnoses in fields such as healthcare, environmental protection, agricultural monitoring, and material identification. However, their use in the field of

solar tracking via image processing methods has not yet gained widespread practicality. These cameras offer certain advantages in the initial step of sun detection by machines in image-processing-based solar tracking. Further studies should determine the specific advantages of these cameras under headings such as feature extraction, classification, noise reduction, and data fusion.

A common problem encountered during sun detection is the scattering effect in camera modules caused by pointing at a very bright sun. While algorithms can eliminate these noises, in real-world conditions, light intensity values can fluctuate over a very wide range instantaneously due to the sun temporarily going behind clouds or atmospheric dust. For this reason, fixed thresholding or noise reduction algorithms often lead to errors or deviations in sun detection when conditions change. The use of multispectral, hyperspectral cameras, and filters would be effective in preventing these errors.

Lee et al. (2013) stated in their study that solar tracking performed using sensors placed in 4 different directions or with rod-shadow tracking methods did not operate effectively under low irradiance conditions. Instead, they developed a new method featuring an image-based sun position sensor and an embedded image processing algorithm. They reported that this method resolved the irregular tracking problem under cloudy atmospheric conditions and achieved solar tracking with an accuracy of 0.04° [Lee et. al., 2013].

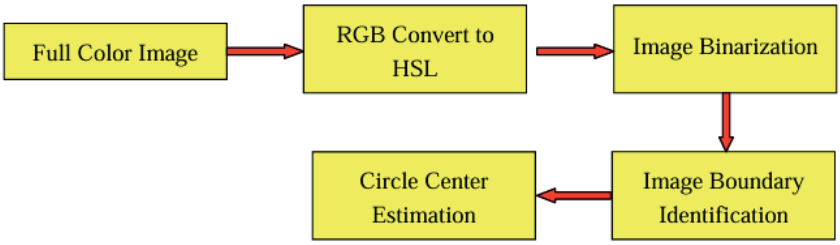


Figure 15 Procedure of estimating the Sun image center with image processing [Lee et. al., 2013]

Azizi and Ghaffari (2013) designed an imaging device based on the principle of solar rays forming a point on a screen. This device utilizes the position of the point to adjust the orientation of the solar panel. They reported that by developing a fuzzy logic controller (FLC), the tracking error was reduced to 0.15°. This study employed a simple A4 Tech PK-836F model camera. A polarizing filter was placed in front of the image

acquisition aperture. The sun's position was translated into a coordinate system using the MATLAB programming language, enabling sun tracking. As a result, a 60.45% increase in energy production was observed compared to fixed systems [Azizi & Gaffari, 2013].

Rahim et al. (2014) designed a two-axis solar tracking system in their study, which used image processing, a Raspberry Pi, two servo motors (for pan/azimuth and tilt/elevation), and a webcam.

"The Raspberry Pi has recently achieved widespread adoption as a microprocessor (Single-borad computer SBC), with Python programming language and various algorithms effectively utilizing OpenCV libraries. This unit serves as the main component for processing images and controlling servo motors. In addition to its flexible and powerful processing capabilities, it has become a highly useful and widely used microprocessor due to its remote access (WIFI-Bluetooth module), USB interface for connecting cameras, external memory, RS-485, RS-232 converters, and similar sensors, as well as its ability to function as a data logger through external and internal memory."

In the mentioned study, the camera module captures an image of the sky and sends it to the Raspberry Pi. The 24-bit color image received from the webcam is converted to 8-bit grayscale. This is achieved using the following code:

$$gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)$$

Then, the image is converted to a binary image using an adaptive thresholding method to detect the sun's circular shape. The exact position of the sun is then determined using the Hough transform. The codes used at this stage are:

$$thresh = cv2.adaptiveThreshold((src, max\ Value, adaptive\ Method, threshold\ Type, block\ Size, C)$$

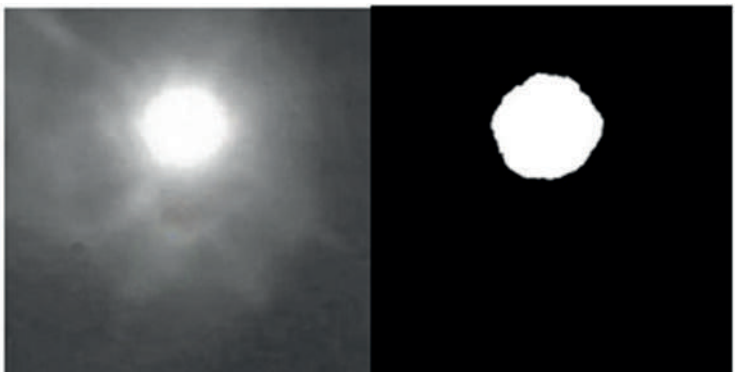
$$THRESH\_BINARY$$

$$dst(x,y) = \begin{Bmatrix} maxvalue & if\ src(x,y) > T(x,y) \\ 0 & otherwise \end{Bmatrix}$$

$$THRESH\_BINARY\_INV$$

$$dst(x,y) = \begin{Bmatrix} 0 & if\ src(x,y) > T(x,y) \\ maxvalue & otherwise \end{Bmatrix}$$

cv2.HoughCircles(image, method, dp, minDist[, param1[, param2[, minRadius[, maxRadius]]]])



(a) *(b)*

Figure 16 (a) The Grayscale image of the sun. (b) The binary image of the sun.

The image transformation using these codes is illustrated in Figure 16 (a) and (b).

The movement of the servo motors is controlled by PWM (Pulse Width Modulation) signals sent from the Raspberry Pi. When the sun is detected, the Raspberry Pi directs the servo motors to position the sun at the center of the image. If the sun remains centered in the image, the system stays stationary for 10 minutes. The camera module, with its 2048x1536 pixel resolution and 3.15 MP image quality, was able to detect the sun even in cloudy weather. Precise tracking was achieved through servo motor control via PWM signals. For this, the PWM setting was adjusted as shown in Table 1 [Rahim et. al., 2014].

Table 1 Time a and b with respect to degree of the motor rotation for different image [Rahim et. al., 2014]

a (ms)	b (ms)	Rotation Angle
0.54	19.46	0°
1.01	18.99	45°
1.47	18.53	90°
1.94	18.06	135°
2.40	17.60	180°

Carballo et al. (2018) aimed to produce a prototype for solar tracking using low-cost open-source hardware and computer vision (CV) technologies. In doing so, they also aimed to create an educational tool, thereby addressing the shortcomings of previous educational instruments.

The prototype includes a Raspberry Pi 3, a Pi Cam, PWM electronic control, and a relay module. All hardware is low-cost and open-source. MATLAB-Simulink and Mathematica were used as control algorithms. The system's cost was reported as $140, indicating a 90% savings compared to other industrial solar tracking systems. The connection diagram used in this study is shown in Figure 17.

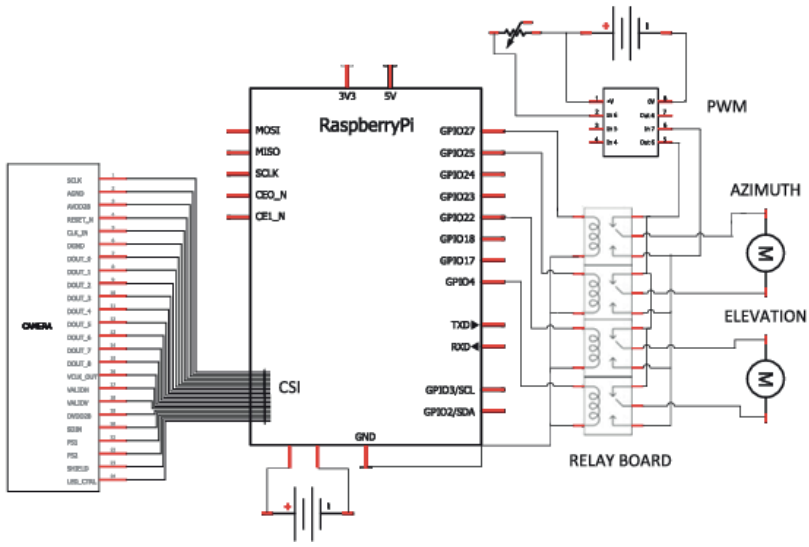


Figure 17 Electric scheme [Carballo et. al., 2018]

Abdollahpour et al. (2018) conducted image processing on the sun's shadow instead of its direct image in their study. The system components consist of a shadow-casting object, a webcam, electronic circuits, computer controls, and stepper motors. Figure 18 shows the system components.

An Arduino UNO microprocessor and an L298N motor driver module were used as electronic circuits. The sun's position was detected based on the coordinates of the shadow's start and end points, and movement commands were sent to the motors.

Working on the shadow instead of the raw image of the sun proved to be a simple but intelligent solution. This is because the sun's extreme

brightness and constantly changing luminosity make working with raw images and detecting the sun quite challenging. Furthermore, reducing the noise level in the image is often not fully possible, necessitating the use of filters. Similarly, reflections from glass surfaces or mirrors can sometimes cause a "fake sun" perception, instantaneously directing the collector to a highly incorrect position. For these reasons, tracking the sun from an image formed by a panel's shadow has the potential to yield more stable results.

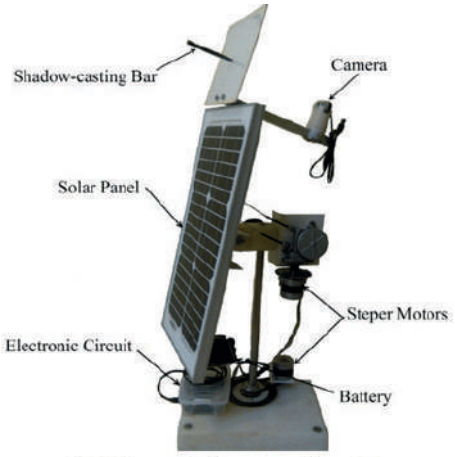


Fig. 2. Electromechanical structure with two DoF.

Figure 18 Electromechanical structure with two DoF. [Abdollahpour et. al., 2018]

Figure 19 shows the coordinate values obtained from the start and end points of the formed shadow, processed with a Grayscale code. The status of these coordinates is made meaningful by the algorithm as follows:

The shadow is in the first quarter ($x > xic, y < yic$);

The shadow is in the second quarter ($x < xic, y < yic$);

The shadow is in the third quarter ($x < xic, y > yic$);

The shadow is in the fourth quarter ($x > xic, y > yic$).

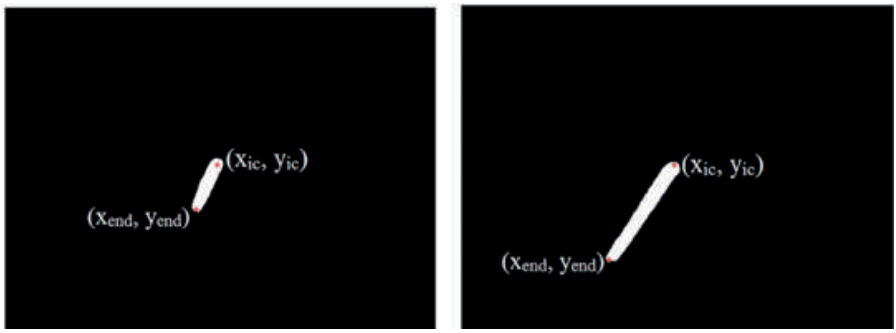


Figure 19 The central point of the image and the farthest point of the shadow. [Abdollahpour et. al., 2018]

Accordingly, movement commands continue until the shadow transforms into a point shape. The solar tracking system operates with an accuracy of ±2° while tracking the sun's position. This system was reported to provide 25-45% more energy production compared to fixed-angle systems [Abdollahpour et. al., 2018].

As evident from these studies, processing is not limited to the bare image of the sun; secondary features that indicate the sun's position can also be utilized. There are also studies where different system components are used for sun detection.

Garcia-Gil and Ramirez (2019) used a fisheye camera in their work and focused on detecting the brightest object entering the camera's field of view to determine the sun's position. The system components included: a fisheye camera for image sensing, an ATmega2560 microprocessor for image processing, stepper motors for movement, and accelerometer-compass sensors for control

Initially, a Grayscale transformation was applied to the RGB format image. After this transformation, Thresholding was applied to layer the image and reduce noise. Figure 20 shows the binary image resulting from the image processing stages. The real-time position of the sun was determined by positioning based on this image.

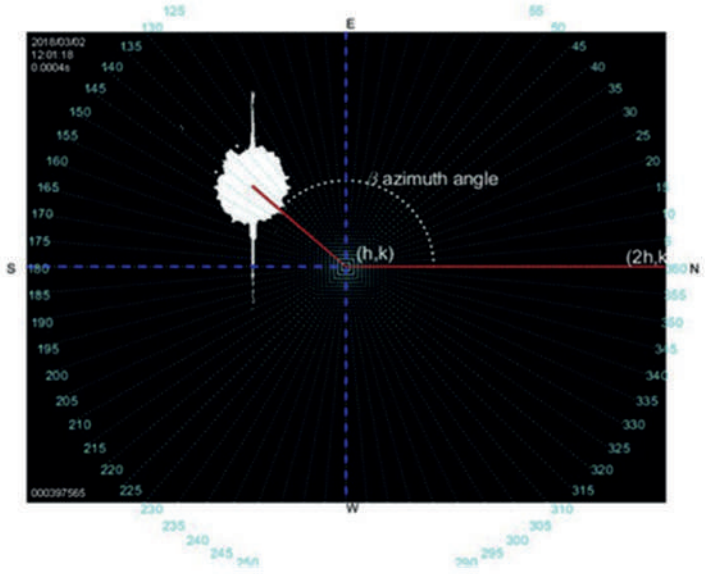


Figure 20 Binary image obtained with the fisheye cam; center at (h, k) pixels. [Garcia-Gil & Ramirez, 2019]

Using the (h,k) coordinates on the image, the sun's azimuth and elevation angles were calculated with MATLAB and compared with sun azimuth and elevation angles obtained from the NOAA Solar Calculator, National Oceanic and Atmospheric Administration. During this comparison, commands were given to the motors using the algorithm flow shown in Figure 21.

At the end of this study, sun azimuth tracking errors ranged between 0.49°-3.47°, while elevation angle errors varied between 5.88°-2.43° [Garcia-Gil & Ramirez, 2019].

Wardhana and Dewi (2020) developed a new tracking algorithm based on the Extended Mean Shift algorithm to support solar tracking for dual parabolic concentrators. Their aim was to eliminate the focusing and heat intensity problems arising from the dependence of traditional photodiode sensors and Solar Position Algorithm (SPA) based tracking systems on light intensity and natural conditions.

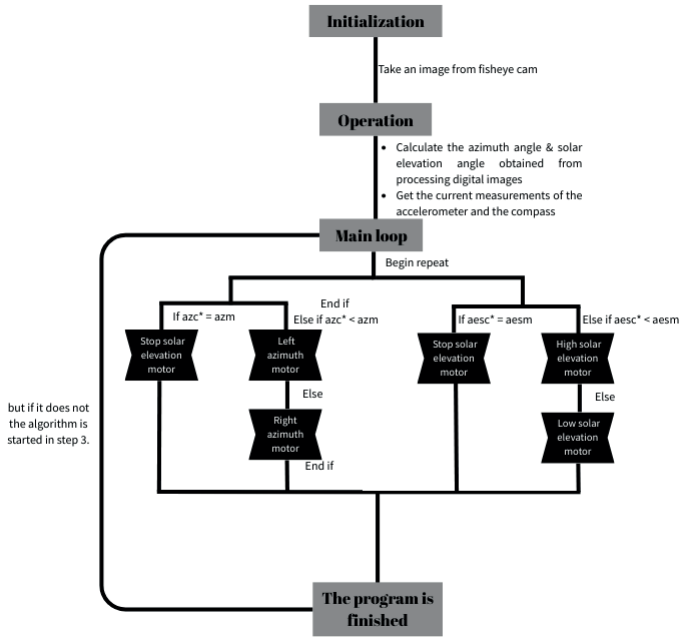


Figure 21 Algorithm Flow Diagram. [Garcia-Gil & Ramirez, 2019]

In this study, the extended mean shift algorithm was developed, which uses the principles of kernel density estimation and searching for the local maximum of color histogram similarity measurement to find the tracking position of an object in a video sequence. To improve the accuracy and reliability of the algorithm, the Expectation Maximization (EM) algorithm was used to estimate model parameters and update the histogram image. Additionally, a Kalman filter was integrated to ensure the stability and robustness of object tracking by estimating the kernel histogram of the object model.

In the conducted experiment, the algorithm was successfully applied for solar tracking on 148 frames of video data. The obtained results showed an average accuracy tolerance value of 98.39% for color similarity in object tracking. This high accuracy rate demonstrates that the developed algorithm can effectively track the sun's position and addresses the shortcomings of previous research (tracking limited to only image processing or black-and-white/grayscale images) [Wardhana & Dewi, 2020].

Kumar et al. (2021) worked on a hybrid solar tracking system. In this study, they proposed an innovative approach combining LDR sensors with Digital Image Processing (DIP) techniques as a solution to tracking

deficiencies caused by the low sensitivity of traditional Light Dependent Resistor (LDR) sensors.

In this approach, they developed a microcontroller (Arduino UNO) based system for optimum positioning of the solar panel. Figure 22 illustrates the operating principle of the system created in this study. The system collects data through four LDR sensors that detect solar light intensity and a camera that captures the sun's image. RGB images captured by the camera are converted to grayscale images to reduce algorithm complexity and shorten processing time. A Gaussian filter is applied to remove noise, and unwanted bright spots like clouds are eliminated using binary thresholding and finding the largest contours methods.

The precise center coordinates of the sun (A_c, B_c) are calculated using image processing algorithms, and this information is used to control servo motors to focus the solar panel directly on the sun.

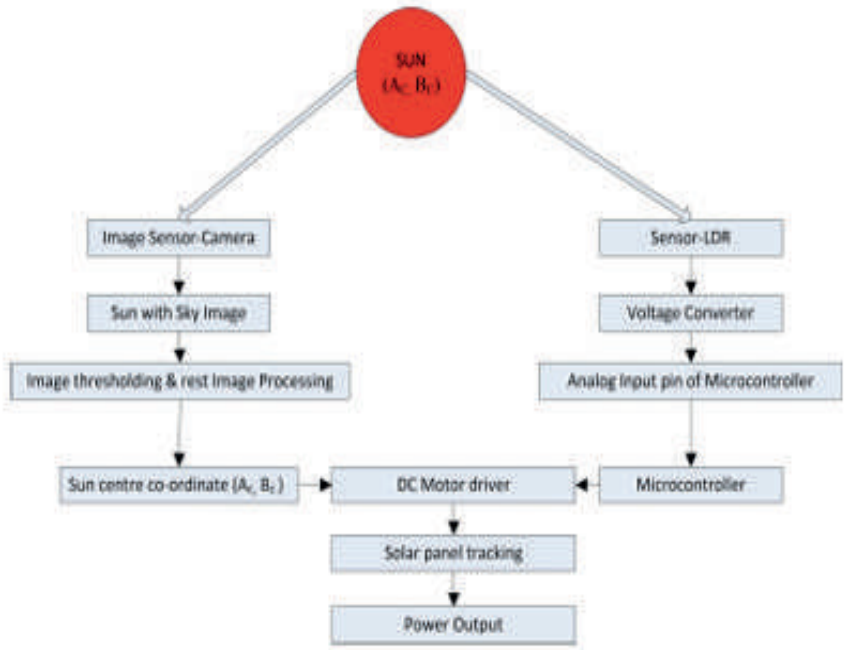


Figure 22 System Block Diagram [Kumar et al., 2021].

The system follows a two-stage tracking methodology:

1. Intensity Sensor Tracking (LDR-Based): Initially, LDR sensors adjust the panel based on solar light intensity. This stage can lead to errors in cases of partial shading or loss of tracking.
2. Image Sensor Tracking (DIP-Based): To correct errors from the first stage, the sun's image is captured using a camera. The sun's centroid is determined via image processing, and these coordinates guide a second tracking mechanism (motor driver) to move the panel towards the sun.

Experiments demonstrated that the proposed hybrid LDR and image processing-based system provided higher power generation compared to LDR-only based or fixed panels. Data collected during a 17-hour daily insolation period yielded the following results:

In terms of Output Power (P_{out}), the LDR and image processing-based system produced a maximum output power of 4.96 W (at 12:00), while the fixed panel produced a maximum of 4.57 W (at 12:00), and the LDR-based system produced a maximum of 4.89 W (at 12:00). On average, the hybrid system provided more power generation than the other two systems.

In terms of Open Circuit Voltage (V_{oc}), the LDR and image processing-based system reached a maximum open circuit voltage of 20.5 V (at 11:00), whereas these values remained lower for the other systems (19.2 V for fixed panel, 20.0 V for LDR-based).

Based on Short Circuit Current (I_{sc}), similar short circuit current values between 0.1 A and 0.2 A were observed across all three systems.

This study proves that the integration of LDR and image processing techniques significantly enhances the accuracy and efficiency of solar tracking systems. It offers a cost-effective and error-free solution, particularly for large-scale solar power plants [Kumar et al., 2021].

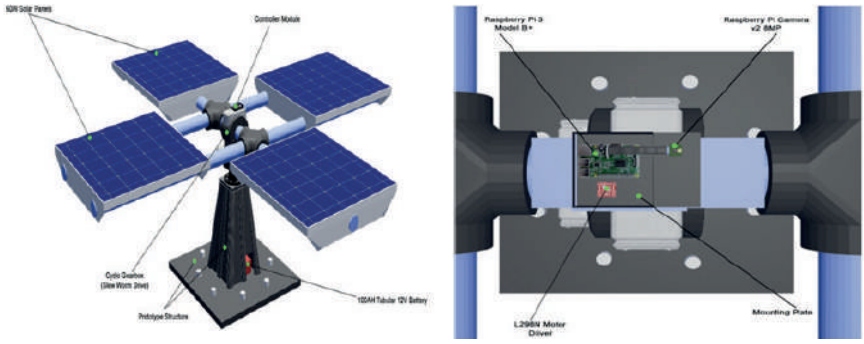
Kamat et al. (2022) developed a prototype that positions solar panels to receive sunlight perpendicularly from sunrise to sunset. The core of the system relies on image processing techniques to accurately determine the sun's position and adjust the panels accordingly.

The system's operation involves the following steps:

1. *Image Capture and Pre-processing:* At sunrise, the prototype continuously captures images of the sky. These RGB (Red, Green,

Blue) images are converted into grayscale images with a single intensity channel to reduce processing load.

2. *Sun Detection and Center Calculation:* The pixel with the highest intensity in the grayscale image (the sun's position) is used as the starting point for the border tracking and object detection algorithm. The algorithm extracts the sun's edges in the image to determine its exact shape.
3. *Noise Reduction:* Following the detection of boundaries, an erosion operation is applied to remove noise originating from the camera sensor or other light sources. This ensures that only the pixel region of the sun remains.
4. *Distance and Movement Calculation:* The centroid of the remaining pixel region of the sun is calculated. The Euclidean distances (offsetX, offsetY) of this center from the center of the image frame along the vertical and horizontal axes are determined.
5. *Panel Positioning:* These calculated offset coordinates are transmitted to two high-torque motors via an L298N Motor Driver. The motors move the structure containing four 60W solar panels using a Cyclo-Gearbox with Dual-Axis Slew Worm Drive capability. This movement ensures the panels are positioned perpendicular to the sun, allowing for maximum power generation.
6. *System Control and Monitoring:* The Raspberry Pi 3 Model B+, serving as the system's core, performs tasks such as image capture, passing offset values to the motor driver, monitoring system health, receiving weather forecast data, and sending power statistics to a Firebase database. Figures 23 (a) and (b) show the schematics of the system and the control unit.



(a) *(b)*

Figure 23 (a) Prototype Schematic (b) Controller Module Schematic [Kamat et al. 2022]

According to the study's findings, this prototype is reported to generate approximately 40% more energy compared to a conventional (fixed) solar panel setup with the same configuration. The system also ensured that the sun's rays directly hit the panels even in light to moderately cloudy weather. Additionally, the prototype is equipped with humidity, temperature, and dust sensors, which generate reports and notify the user if an issue is detected in the operating environment based on sensor data. An economic analysis of the produced prototype estimated a payback period of 5 years [Kamat et al. 2022].

As seen from the studies above, the operating principle of solar tracking systems using image processing methods is gradually evolving into a permanent and stable procedural workflow. This procedure begins with the Image Acquisition step and continues with the Image Processing step. In the Image Processing step, once the sun's position in the sky is determined, it is brought to a perpendicular position relative to the panel in the subsequent Commanding Motion Units step. It is crucial to remember here that the sun is the independently moving entity, while the solar collectors are the dependently moving elements. A common thread across all studies is the effort to minimize errors or electronic flaws occurring during sun detection. To this end, innovations such as sensor redundancies, fixed-speed algorithms, and hybrid detection models are being developed. However, a noticeable gap in the literature is the absence of a system that processes solar tracking data using a machine learning model. In this area, by utilizing machine learning models, the already high solar tracking accuracy could be elevated to even greater values.

Zeghoudi and Benmouiza (2023) present an innovative hybrid control approach for optimizing the orientation of heliostats to enhance the efficiency of Concentrating Solar Power (CSP) towers. In this work, they combined image processing techniques (IPT) and artificial neural networks (ANNs) to enable solar tracking with higher precision. They developed a Hybrid Control mechanism by integrating the strengths of both open-loop and closed-loop control mechanisms. A closed-loop based IPT using a CCD camera was established to detect the sun's position, and an open-loop based control scheme using ANNs was created to predict heliostat trajectories during cloudy sky conditions.

In the described hybrid system, the image processing technique (IPT) detects the sun's center (pixel coordinates (x, y)) using images captured from a webcam and calculates the heliostat's azimuth and elevation angles. The image processing steps include:

- Converting the color image to grayscale.
- Converting to a binary image using the Otsu method to separate the sun region from the background.
- Eliminating small objects.
- Inflating the morphology of the binary image.
- Removing small objects with morphological processing.
- Calculating the percentage of the sun's circular shape.
- Determining the grayscale centroid of the sunspot.

The innovation in this system is its transition from IPT to an artificial neural network (ANN) predictor when the percentage of the sun's circular shape falls below 75% or the sun's image moves out of the webcam frame. This allows the ANN to estimate the heliostat's position under cloudy or obstructed conditions. The ANN model uses five parameters as input: date, time, geographical location, receiver height, heliostat-tower distance, east-west distance, and north-south distance.

Simulation results showed that the hybrid ANN-IPT method minimized the tracking error to not exceed 0.1. This proves that the method is acceptable for controlling heliostats in solar tower systems. The excellent performance of this hybrid approach in cloudy weather was particularly emphasized. Thus, the applicability of hybrid control systems in heliostat applications has been demonstrated [Zeghoudi & Benmouiza (2023].

As observed, academic studies consistently demonstrate that solar tracking systems utilizing image processing methods can achieve high precision. Given its status as a relatively new and active research area, further advancements are anticipated.

CHAPTER 5

5. System Components

While the system components vary depending on the physical size of the solar tracking system, common elements used across all systems are illustrated in Figure 24.

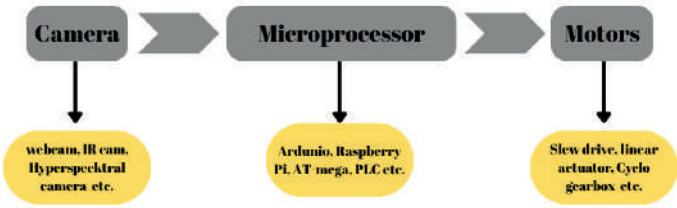


Figure 24 Main parts of solar tracking systems by image processing method

The camera component of the system was discussed in Section 4.2. Before moving to the algorithm and coding steps, it is essential to discuss the motors used in the system's motion mechanisms. The size and motion capability of these motors will determine the algorithmic-cycle (logic) used in the algorithm and the electronic circuit between the microprocessor and motor control.

While stepper motors and small servo motors can be used in small-scale experimental systems, linear actuators can be employed in large, single-post, island-type constructions supporting 12 PV panels. Gear-driven or belt-driven motors are no longer preferred. Slew drive motors, which enable the movement of large strings, can also be used. To adjust motor speed, motor drivers can be connected to the circuit, or gearboxes can be mounted on

the motor shaft. Motion is achieved using numerous options and rotation methods.

5.1. Servo Motors vs. Stepper Motors in Solar Tracking

Figures 25 (a) and (b) show examples of servo and stepper motors. Stepper motors are electromechanical devices that allow the rotor to move in precise, fixed angular steps by applying pulse signals to the coils on the stator. The rotor's position is directly dependent on the number and sequence of applied pulses. They are typically used in open-loop control systems, -no feedback on the motor's movement- and the controller assumes the motor has reached the desired position. Each pulse triggers the rotor to turn by a specific angle and these steps accumulate to form the total rotation amount. Open-loop control generally leads to simpler control circuits and lower costs. They possess high holding torque when energized, which increases the motor's resistance to load at a standstill. This presents an advantage in solar tracking systems, especially under wind and snow loads. Thanks to their brushless structure, they are long-lasting and require minimal maintenance.

In stepper motors, torque values significantly decrease at high speeds. There is a risk of step skipping under overload or at high speeds, leading to position errors. Since there is no feedback, this error might go unnoticed. They produce noticeable vibration and noise, especially at low speeds. For this reason, their use in solar tracking systems is not recommended for applications where high precision is crucial.

Servo motors are closed-loop control systems capable of maintaining a specific position, speed, or torque value with high accuracy. A servo motor system consists of a feedback sensor (encoder, resolver, etc.) and a driver (servo driver). The driver moves the motor according to control signals, and the feedback sensor continuously measures the motor's current position or speed. The difference (error) between this measured value and the desired target value is instantly corrected by the driver, ensuring the motor performs as desired.

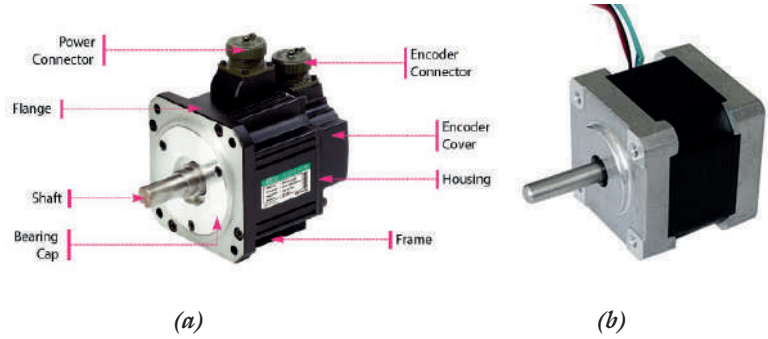


Figure 25 (a) Servo motor (b) Step motor

Thanks to closed-loop control, servo motors offer very high positioning accuracy and repeatability. They can produce high and constant torque over a wide speed range, enabling rapid acceleration and dynamic motion capability. This is not a critical feature in solar tracking systems where high speed is not desired. However, their feedback mechanism allows them to instantly detect and correct load-dependent torque changes, ensuring stable performance. Their quiet and vibration-free operation and higher efficiency make their use in solar tracking systems advisable. It should be noted that adjusting the PID (Proportional-Integral-Derivative) control parameters and correctly calibrating the system for high-cost servo motors requires expertise.

5.2. Linear Actuators in Solar Tracking

With the widespread adoption of robotic applications and 3D printers, the usage areas of linear actuators have expanded. Previously used in applications like opening/closing garage doors and positioning hospital beds, linear actuators, which operate on the principle of a worm gear, have become suitable system components for applications requiring angular movements and speed control. Figure 26 shows industrial-type linear actuators. Among the many varieties of linear actuators, both electric worm gear types and hydraulic types are suitable for solar tracking systems. Electric actuators themselves come in 12V, 220V, and even 380V versions. However, DC type electric linear actuators may be chosen to avoid complex electronic circuits for PLC and microprocessor control.

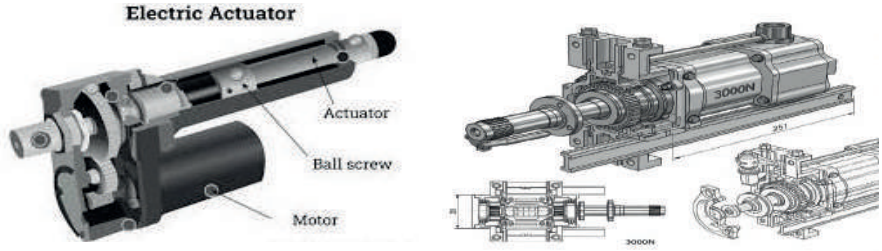


Figure 26 Industrial linear 3000N actuator

Table 2 provides the types of linear actuators. The appropriate actuator is selected by calculating the system's load capacity, extension length, wind and snow loads, and holding force.

Table 2 Types of Linear Actuators

Actuator Type	Operating Principle	Advantages	Disadvantages	Typical Applications
Electrical	Driven by an electric motor (DC or AC) via a ball screw, lead screw, or belt-pulley mechanism.	High precision, programmability, clean operation, and energy efficiency.	Higher cost, requirement for complex control electronics, and limited performance in sudden high-force applications.	Robotics, automation systems, medical devices, solar tracking systems, and valve control.
Pneumatic	A piston-cylinder mechanism operated by compressed air.	Fast movement, simple structure, reliable, cost-effective, and capable of generating high forces.	Generally lower precision, requires a compressed air source, noisy operation, and risk of air leakage.	Industrial automation (grippers, conveyors), door opening systems, and packaging machines.
Hydraulic	A piston-cylinder mechanism operated by pressurized fluid (oil).	Capable of generating very high forces, provides smooth and stable movement, and is ideal for heavy loads.	Complex system (pump, tank, valves), risk of oil leakage, maintenance requirements, and higher cost.	Heavy industry machinery (e.g., excavators, presses), construction equipment, and ship steering systems.
Mechanical	Manual or motorized movement through mechanical elements such as levers, cams, gears, or screws.	Simple, cost-effective (for manual versions), reliable, and not dependent on an external energy source.	Generally manual control, lower precision, limited stroke (range of motion), and slow movement.	Jacks, manual adjustment mechanisms, simple lifting systems, and door locks.

Piezoelectric	Based on the principle of piezoelectric materials changing shape with electric current.	Very high precision, very fast response time, sub-micron motion capability, and compact size.	Very small range of motion (micron level), limited force, and requires high voltage.	Optical alignment, microscopes, nano-positioning, and valve control (for precise flow).

Linear actuators convert circular motion at the rotational axis into linear motion through a gear and worm screw design. In this way, they create a shaft mechanism that extends back and forth. Stator assemblies rotate the shaft to control direction and distance of travel, making these actuators ideal for process automation and controlled load handling in manufacturing cells or conveyor systems.

Choosing DC motors for the rotation process is preferable for applications remote from grid connections, aiding in auxiliary power source selection and portability. The motors for these types of actuators can be brushed or brushless. Brushed motors deliver cost-effectiveness and commonality, but require maintenance due to eventual brush wear, while brushless motors offer enhanced durability and increased operating life.

Furthermore, it must be calculated whether the fully closed and fully open positions of the linear actuators are sufficient to provide the required angular motion of the construction. The extension length of linear actuators is called “stroke.”

Figure 27 Two axis solar tracking system with linear actuator

Figure 27 shows a two-axis solar tracking system where the motion mechanism is controlled by linear actuators. One actuator tracks the sun's azimuth angle within the sunrise-sunset angular range, while the other tracks the seasonally varying sun elevation angle. The crucial point here is that the stroke value of the linear actuator must be suitable for the angular movement during full sunrise and full sunset. Similarly, it is important that the stroke value of the other actuator is sufficient to cover the max-min sun elevation angle movement during summer and winter periods.

The variation in the max-min range of sunrise and sunset angles and sun elevation angles depends on:

- The "n" value (which determines the declination angle (δ), representing the day of the year)
- Latitude angle (Ø)

It should not be forgotten that "day-time / length of daylight" is dependent on these values.

Table 3 Highest and Lowest Day Length Angular Values at Selected Centers

City	Latitude (ϕ)	Lowest Day Length (Angular, Winter Solstice)	Highest Day Length (Angular, Summer Solstice)
Izmir	38.41° N	139.76°	220.24°
Istanbul	41.00° N	135.68°	224.32°
Ankara	39.93° N	137.46°	222.54°
Berlin	52.52° N	111.18°	248.82°
Paris	48.86° N	120.48°	239.52°
Roma	41.90° N	134.26°	225.74°
Madrid	40.42° N	136.68°	223.32°
Washington	38.91° N	138.98°	221.02°
Pekin	39.90° N	137.52°	222.48°
Tokyo	35.68° N	143.74°	216.26°

Table 3 shows the highest and lowest day length values for some cities. Accordingly, a system designed for tracking the sun along the east-west (solar azimuth) axis and the linear actuator chosen for it must be designed to accommodate the maximum and minimum angular movements shown in Table 3. A system designed based on the highest angular movement

will continuously receive perpendicular sunlight along the east-west axis throughout the year.

The sunset/sunrise hour angle ($\omega_{ss/sr}$) is calculated using the following formulas:

$$cos(\omega_{ss}) = -tan(\emptyset)tan(\delta) \quad (1)$$

where:

δ: Declination angle

$\emptyset$: Latitude angle

$$Day\ Length\ (angular) = 2 \times \omega_{ss} \quad (2)$$

For example, a system designed for Izmir must be suitable for an angular sweep of 139.76° for the shortest day length (Winter Solstice, December 21st) and 220.24° for the longest day length (Summer Solstice, June 21st). A system designed based on the maximum angular movement will continuously receive perpendicular sunlight along the east-west axis throughout the year.

It is observed that solar tracking systems produced with linear actuators generally operate within a range of less than 180° on the east-west axis. This is because when the rotation exceeds 180°, the rotational movement axis of the actuator falls behind the rotation axis of the construction. In such cases, the actuator's connection point shifts to the opposite side of the rotation axis and cannot generate rotation, potentially damaging the construction during retraction. Since actuators move back and forth from a single connection point, approaching 180° rotation causes strain on the construction. Using two different actuators for each direction would resolve this issue. However, errors in synchronizing the movement of two actuator and in writing the synchronous control algorithm would lead to system stoppage or excessive load on the actuators.

In solar energy systems, movable mechanisms are undesirable due to their high cost and the need for frequent breakdowns, maintenance, and repairs. Resolving the east-west movement with a second actuator eliminates the initial problem but may introduce new ones.

When examining the amount of useful energy produced during the initial sunrise and final sunset periods, it is negligible compared to the midday period. In situations requiring angular tracking beyond 180°, adding an additional actuator and considering the associated breakdown, maintenance, and repair costs do not yield feasible results. Instead, increasing the number of panels or the surface area might offer a more effective solution.

In solar lighting applications, this (180° angular tracking at max.) implies that the maximum lighting duration can be 12 hours. In cases where the day length exceeds 12 hours, performing solar tracking with linear actuators can lead to significant efficiency loss in solar lighting applications.

While certain geometric constraints cause issues during east-west (solar azimuth) axis solar tracking with linear actuators, these problems do not arise for tracking along the north-south (solar elevation) axis. For tracking along the north-south axis, the sun elevation angle (α_s) must be determined. The sun elevation angle is a seasonally varying parameter. In the Northern Hemisphere, it takes high values in summer and low values in winter. It should be noted that this movement is reversed in the Southern Hemisphere. The sun elevation angle (α_s) can be calculated using the zenith angle (θ_z). Here, the zenith angle refers to the angular value between the horizontal surface normal and the sun's rays and is calculated as follows:

$$cos(\theta_z) = sin(\emptyset)\, sin(\delta) + cos(\emptyset)\, cos(\delta)\, cos(w) \quad (3)$$

At solar noon, since w=0 (solar hour angle), Equation (3) simplifies to:

$$cos(\theta_z) = sin(\emptyset)\, sin(\delta) + cos(\emptyset)\, cos(\delta) \quad (4)$$

This can be further simplified using trigonometric identities:

$$cos(\theta_z) = cos(\emptyset - \delta) \quad (5)$$

As shown in Figure 28, the solar elevation angle is the complement of the zenith angle. Therefore, for solar noon, the zenith angle is:

$$\theta_z = |\emptyset - \delta| \quad (6)$$

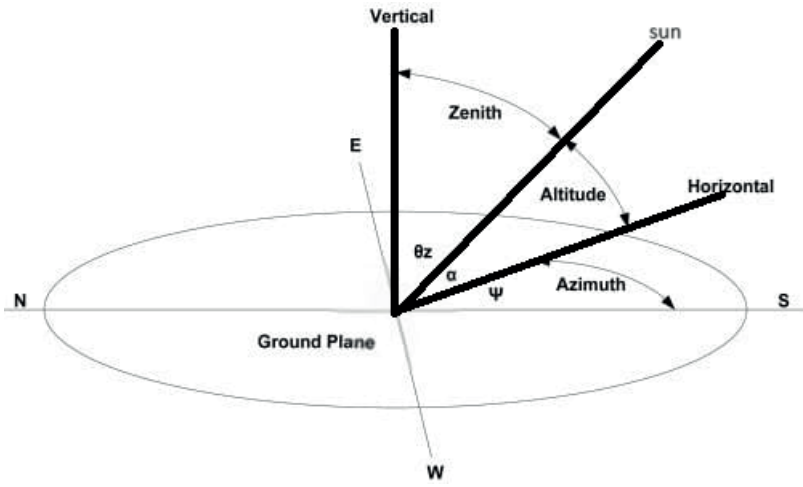


Figure 28 Zenith Angle vs. Solar Elevation Angle

The solar elevation angle (α_s) is then:

$$\alpha_s = 90° - |\emptyset - \delta| \quad (7)$$

Table 4 presents the maximum and minimum solar elevation angles at solar noon for various locations.

Table 4 Maximum and Minimum Solar Elevation Angle Values at Solar Noon for Various Locations

City	Latitude ($\emptyset$)	Minimum Solar Elevation Angle ($\alpha_{s,min}$)	Maximum Solar Elevation Angle ($\alpha_{s,max}$)
Izmir	38.41° N	28.14°	75.04°
Istanbul	41.00° N	25.55°	72.45°
Ankara	39.93° N	27.02°	73.88°
Berlin	52.52° N	14.03°	60.93°
Paris	48.86° N	17.62°	64.71°
Rome	41.90° N	24.65°	71.55°
Madrid	40.42° N	26.13°	73.07°
Washington	38.91° N	27.64°	74.56°
Beijing	39.90° N	27.05°	73.85°
Tokyo	35.68° N	30.87°	77.73°

According to this table, a solar tracking system designed for Izmir must be compatible with a minimum solar elevation angle of 28.14° and a maximum solar elevation angle of 75.04° at solar noon. In other words, the structure and linear actuator moving along the North-South axis must be able to achieve these angular values. The panels should be positionable between a minimum inclination of 28° and a maximum of 75°. The values in Table 3 and 4 should be used as a basis when designing the structure and selecting the linear actuator. North-South movement is easier to track with a single linear actuator compared to East-West movement, as it involves a narrower range.

5.3. Slew Drive in Solar Tracking

Slew drive motors provide a solution to the issues that arise when linear actuators are used to generate rotational motion from their linear back-and-forth movement. A slew drive is a mechanical system employed to rotate or change the position of a load. This system typically operates with a gear mechanism, providing rotational movement and is used for moving heavy loads. Being a gear mechanism that rotates on its own axis, it is more

compatible with rotational motion. These motors are frequently utilized in large and heavily loaded conveyors or construction machinery. In solar tracking applications, they serve as the movement mechanism for large PV strings or large parabolic trough and parabolic dish collectors.

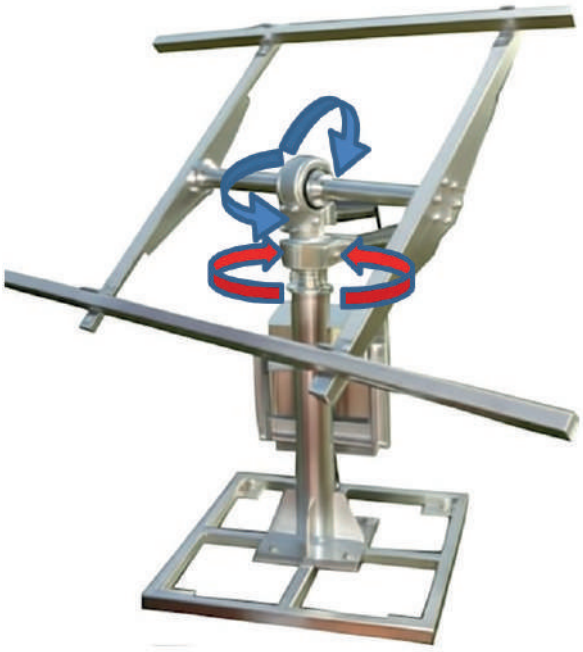


Figure 29 Two axes solar tracking system using slew drive

As shown in Figure 29, a PV stand with two-axis solar tracking utilizes slew drives for rotational mechanisms in both East-West (solar azimuth) and North-South (solar elevation) directions. The gear, which is the fundamental component of a slew drive, provides rotational motion. The gearbox enables high torque and holding force to be achieved with low-power motors. This allows for a wide range of applications. The motor providing the movement can be an AC (220~380V) or DC (12~36V) electric motor, or a hydraulic-pneumatic motor.

These systems are distinguished by their capacity to bear high radial and axial loads, as well as their ability to transmit significant torque values. Commonly used slew drive types in the market can be classified by their structural characteristics, operating principles, and typical application areas as follows:

- Worm Gear Slew Drives
- Spur Gear Slew Drives
- Single-Axis Slew Drives
- Dual-Axis Slew Drives
- Enclosed Slew Drives
- Open Slew Drives

5.3.1. Worm Gear Slew Drives

Worm gear slew drives are based on a combination of a worm and a connected worm wheel. The worm acts as the driving element, transferring rotational motion to the worm wheel. The most distinguishing feature of these systems is their ability to provide significant torque transmission even in small sizes, thanks to high conversion ratios. Additionally, they often possess a self-locking feature; meaning, when the motor stops or disengages, they prevent the load from moving backward, eliminating the need for an additional braking mechanism. This offers a significant advantage, especially in applications requiring secure locking.

Worm gear slew drives are widely used in areas where high torque, precise positioning, and self-locking capabilities are prioritized. Solar tracking systems are one such application. They are preferred in solar tracking systems to maintain tracking accuracy under variable load conditions such as wind and snow loads. Other application areas include cranes and lifting equipment, aerial platforms, hydraulic machinery, and robotic systems requiring precise positioning. Their key differences from other slew drive types are their high conversion ratios and inherent self-locking capability. This ensures that the load maintains its position even the motor is disengaged, providing a critical advantage in terms of safety and stability. Other gear types typically do not offer this feature and require additional braking systems.

5.3.2. Spur Gear Slew Drives

Spur gear slew drives are systems that transmit torque through the meshing of parallel-axis spur gears. They have a simpler gear geometry compared to worm gear systems. Relatively low friction coefficients of spur gears allow higher efficiency and potentially higher rotational speeds. However, unlike worm gear systems, they do not have a natural self-locking feature; therefore, external braking systems may need to be integrated to hold the load in a specific position.

These types of slew drives are preferred in applications where higher rotational speeds and high efficiencies are desired, but self-locking is not critical. They can be used in areas such as industrial automation systems, light and medium-duty cranes, material handling equipment, and general engineering platforms. The lack of a self-locking feature necessitates an additional braking system, which can increase cost or complexity.

Figure 30 illustrates the differences and similarities between worm gear and spur gear slew drives produced by IMO Industry in Gremdorf/Germany [Industries, IMO. (2025, 06 09)].

The most significant difference lies in how the drive mechanism connects to the rotating gear. In a worm gear slew drive, the drive mechanism and the gear are positioned tangentially in the same plane, resulting in higher torque and holding force. In a spur gear slew drive, the drive mechanism is positioned tangentially to the rotating gear in a perpendicular plane. This geometric design allows for higher rotational speeds and lower friction.

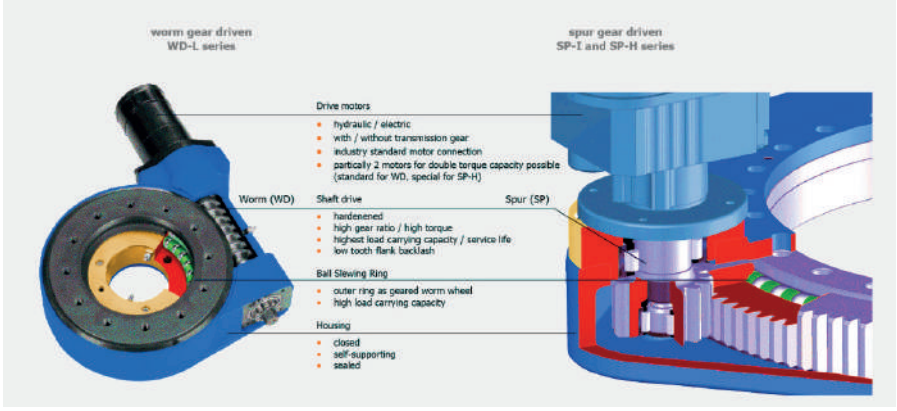


Figure 30 Differences and Similarities of Worm Gear and Spur Gear Slew Drives [Industries, IMO. (2025, 06 09)]

5.3.3. Single-Axis Slew Drives

Single-axis slew drives are systems that provide movement around a single axis of rotation. They typically incorporate a worm gear mechanism and can offer high torque capacity with self-locking capability. Structurally, they are simpler than dual-axis models.

These systems are used in applications requiring positioning in a single plane of rotation. They are commonly found in single-axis solar tracking

systems, wind turbine blade pitch adjustments, medium-scale cranes and lifting equipment, and industrial automation systems. Unlike dual-axis systems, they offer only a single axis of rotation. This simplicity provides a more cost-effective and less complex solution, depending on the application requirements.

5.3.4. Dual-Axis Slew Drives

Dual-axis slew drives are systems that provide two independent axes of rotation within a single integrated unit. These axes are typically perpendicular to each other and can be controlled independently. They are commonly referred to as the primary axis (e.g., azimuth) and the secondary axis (e.g., elevation). This integrated design offers a more compact and optimized solution compared to two separate single-axis systems.

Their most prominent and widespread application is in dual-axis solar tracking systems. In these systems, the goal is to maximize energy efficiency by enabling solar panels to track the sun's movement in both horizontal (azimuth) and vertical (elevation) planes. They can also be used in areas requiring complex motion profiles, such as robotic applications and satellite communication positioning systems.

Figure 31 shows single-axis and dual-axis slew drives manufactured by Jiangyin Sunslew Machinery Equipment for use in solar tracking systems [(Jiangyin Sunslew Machinery Equipment) https://www.sunslewdrive.com/slewing-drive/].

(a) *(b)*

Figure 31 (a) Single axis slew drive (b) Dual axis slew drive [(Jiangyin Sunslew Machinery Equipment) https://www.sunslewdrive.com/slewing-drive/]

Figure 31 (a) shows a single-axis slew drive designed with a special torque tube connection for solar PV string tracking. Figure 31 (b) shows a flange-mounted dual-axis slew drive for two-axis solar tracking systems. It offers two different movement capabilities that can be controlled independently.

5.3.5. Enclosed Slew Drives

Enclosed slew drives are designed for where all gear and bearing components are housed within a sealed enclosure. This enclosure protects the internal mechanism from dust, dirt, moisture, corrosive chemicals, and other environmental contaminants. The sealing prevents internal lubricant from leaking out while also preventing external elements from entering. This extends the system's lifespan and reduces maintenance requirements.

They are preferred in applications with harsh environmental conditions, especially outdoor applications, the maritime sector, construction equipment, agricultural machinery and mining equipment, where a high level of protection and durability is required. The biggest difference from open gear systems is that the gear mechanism is completely enclosed and sealed. Those meeting IP-64 and IP-X standards are the most preferred. This provides better environmental protection, less maintenance and a longer service life.

5.3.6. Open Slew Drives

Open slew drives are designed for where the gear mechanism is in direct contact with the external environment. In these systems, the gears are typically exposed and visually accessible. This can reduce initial investment costs and, in some cases, improve maintenance accessibility. However, they are more susceptible to external factors and require regular lubrication and cleaning.

They can be used in applications with less demanding environmental conditions and relatively controlled environments. They may be preferred in areas such as indoor cranes, lighter load platforms, and industrial machinery with less intensive use. The main difference from enclosed systems is that the gear mechanism is exposed and subject to environmental factors. While this provides lower manufacturing costs, it also necessitates more frequent maintenance and environmental protection requirements.

5.3.7. Slew Drive Selection Criteria and Design Analysis

The correct selection of slew drive systems is a critical decision that directly impacts an application's performance, safety, and economic lifespan. The

selection process requires a detailed analysis of the application's requirements and matching them with the structural and operational characteristics of the drive system. This analysis primarily considers:

- Torque requirements
- Speed
- Precision
- Load-bearing capacities
- Environmental conditions
- Economic factors

  ✓ The structure of the system where the slew drive will be used (e.g., solar tracking system, crane, robotic arm, antenna positioning), the required type of motion (continuous rotation, indexing), the angle of motion, and the number of cycles are fundamental determinants. The application's static (holding the load in a stationary state) and dynamic (rotating the load while in motion) torque requirements should be the basis for selecting motor power and gear ratio. Wind loads, sudden stops/starts, and friction forces must be included in these calculations.

  ✓ The required rotational speed (rotation per minute or degrees per second) and positioning precision (arc seconds or degrees) of the application are critical in determining the gear ratio, gear tolerances, and backlash characteristics.

  ✓ The radial (non-axial) and axial (along the axis) loads to which the drive will be subjected, as well as the overturning moment (OTM), must be carefully calculated. This directly affects the sizing of the drive's bearing system and gear structure.

  ✓ The operating environment's exposure to temperature, humidity, dust, water, corrosive substances and UV radiation determines characteristics such as the drive material, sealing class (IP rating), and corrosion resistance. The selection of enclosed or open-type drives is made based on these factors. The expected service life, maintenance intervals and ease of service should be considered in the drive's design and material selection. Economic factors such as initial purchase cost, maintenance costs, and energy efficiency also play an important role in the decision-making process.

Load Analysis and Torque Calculations;

All loads acting on the application and the moments (torques) generated by these loads are calculated. This includes static and dynamic conditions.

Static Torque (M_s): This is the maximum torque the system must withstand while stationary.

$$M_s = F_{load} \times r_{load} + F_{wind} \times r_{wind} + M_{additional} \quad (8)$$

Where:

M_s: Static Torque (N.m)

F: Force (N)

r: Perpendicular distance from the force to the axis of rotation (m)

$M_{additional}$: Additional moments (e.g., friction load, etc.) (N.m)

Dynamic Torque (M_d): This is the torque required by the system while in motion (acceleration, constant speed rotation, deceleration).

$$M_d = I \times \alpha + M_{friction} \quad (9)$$

Where:

M_d: Dynamic Torque (N.m)

I: Moment of inertia ($kg.m^2$)

α: Angular acceleration (rad/s^2)

$M_{friction}$: Friction torque (N.m)

Overturning Moment (OTM): This is the moment acting on the slew drive's flange, attempting to tip the system and is critical for determining the bearing capacity. It is typically specified by slew drive manufacturers in their product catalogs. The catalog values, i.e., maximum OTM values, must not be exceeded.

The calculated maximum torque values determine the nominal torque capacity of the slew drive to be selected. Safety factors (F_s) are generally applied to these values. F_s can be chosen as 1.2 (120%) for a safety limit sometimes 1.5 (150%) as desired according to design engineer expertise. The safety factor is an additional precautionary value added to the maximum torque value required and determined by calculation. In systems operating under natural conditions, it serves as a safety margin against unforeseen circumstances, often used by design engineers as a precautionary measure. This value can be chosen higher or lower based on the design engineer's experience and observable anomalies in the external environmental conditions

where the system will operate. Especially in today's climate crisis driven by global warming, where anomalies are observed in natural conditions, careful attention should be paid to determining the F_s value.

$$M_{required}=M_{max}\times F_s \quad (10)$$

Speed and Reduction Ratio Determination; The maximum rotational speed (N_{max}) and cyclical use (total number of rotations per day or year) required by the application are considered.

The relationship between the slew drive's motor input speed (N_{motor}) (rpm) and the drive output speed (N_{output}) (rpm) is determined by the conversion ratio (i):

$$i = N_{output} / N_{motor} \quad (11)$$

Higher conversion ratios provide higher torque output and lower output speed, while lower conversion ratios provide higher speed and lower torque.

Evaluation of Load-Bearing Capacities; The radial load (F_r), axial load (F_a), and overturning moment (OTM) capacities of slew drives are compared with the manufacturer's specified catalog values. These capacities are critical, especially in heavy-load applications.

Load-Bearing Capacities and Gear Diameters

The load-bearing capacities of slew drives generally increase proportionally with the turntable diameter (gear diameter). Larger diameter slew drives can carry higher radial, axial loads, and overturning moments because they have larger bearings and larger gear contact areas.

Gear diameters are determined based on the torque and load capacity required by the application. Generally, as torque requirements increase or the overturning moment becomes larger, slew drives with larger gear diameters are preferred. For example, slew drives are available in various diameters (e.g., from 100 mm to over 1000 mm) for torques ranging from a few hundred Nm to several hundred kNm.

Backlash and Precision Analysis

For applications requiring positioning precision, the backlash value of the slew drive is of great importance. Backlash refers to the clearance between meshing gears and is typically specified as an angular value (e.g., arc minutes or degrees) in the manufacturer's catalog.

For precise applications, slew drives with low backlash (precision backlash) should be preferred.

5.3.8. Slew Drive Selection: A Case Study "Calculation Method for Solar Tracking Systems"

Commercially available electricity generation systems utilizing solar energy are increasingly being sold as complete packages including solar tracking systems. These packages typically comprise pre-fabricated structures containing 1-2-4-8-16 PV panels, along with dual-axis tracking mechanisms and microprocessor units for control. With advancing technology, diverse manufacturing materials are being employed, and their variety expands day by day. Among these systems, those incorporating 8 PV panels (3.2 kWp~4 kWp) are the most frequently sold systems. They are preferred due to their capacity to meet the average electricity demand of a household in off-grid areas. To properly select a slew drive for use in these systems, it is essential to meticulously follow a specific calculation methodology.

It is crucial to determine the primary loads (wind and panel weight) that 8-panel PV system may encounter, and consequently, the expected torque from the slew drive. Figure 32 presents the design parameters for the system. While more detailed calculations could be incorporated into this methodology, a simplified and rapid calculation method has been chosen. Additionally, it is recommended that the design engineer considers the safety factor F_s mentioned previously.

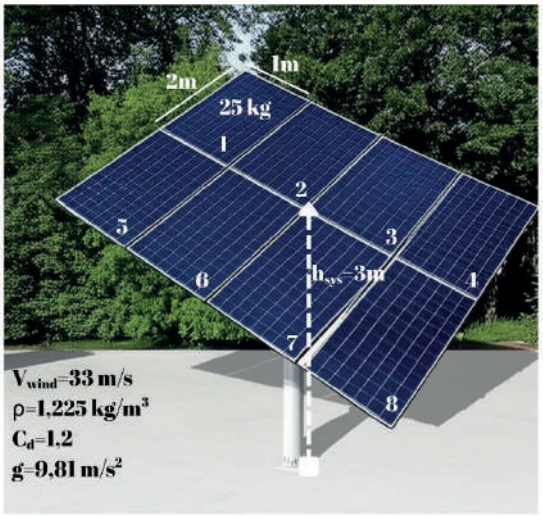


Figure 32 A case scenario slew drive analysis for dual axis solar tracking system

Case scenario system parameters;

- Number of Panels: 8 units
- Dimensions of Each PV Panel:
  - Length (L): 2 meters
  - Width (W): 1 meter
- Weight of Each PV Panel (m): 25 kg
- System Height (average height from ground to panel center) (h_{system}): 3 meters (This will affect the distance from the panel's wind center when calculating wind moment.)
- Maximum Wind Speed (V_{wind}): 120 km/h ≈ 33 m/s (For storm conditions)
- Air Density (ρ): 1.225 kg/m³ (Standard sea level air density)
- Drag Coefficient (C_d): 1.2 (A typical value for PV panels, varies with panel angle)
- Gravitational Acceleration (g): 9.81 m/s²

1. Total Panel Area and Weight

First, the total panel surface area and total panel weight for the system must be determined.

Panel Surface Area; $A_p = L \times W$ *(12)*

$= 2(\text{m}) \times 1(\text{m}) = 2(\text{m}^2)$

Total Surface Area; $A_{total} = \text{Number of Panels} \times A_p$ *(13)*

$= 8 \times 2(\text{m}^2) = 16(\text{m}^2)$

Total Panel Weight; $W_{total} = \text{Number of Panels} \times m \times g$ *(14)*

$= 8 \times 25\ (\text{kg}) \times 9.81\ (\text{m/s}^2) = 1962(\text{N})$

2. Wind Load Calculation (for Azimuth axis)

The force on the PV panels exerted by wind and the resulting moment on the slew drive should be calculated for the worst-case scenario. Therefore, it should be assumed that the wind strikes the panel surface perpendicularly.

Wind Pressure (P_w); $P_w = 0.5 \times \rho \times (V_w)^2$ *(15)*

$P_w = 0.5 \times 1.225(\text{kg/m}^3) \times (33.33(\text{m/s}))^2 \approx 680$ Pa (Pascal)

Total Wind Force (F_{wind}): $F_{wind}=P_w \times A_{total} \times C_d$ *(16)*

$F_{wind}=680(Pa)\times 16(m^2)\times 1.2\approx 13056(N)$

Overturning Moment Caused by Wind (M_{wind}): This moment is a critical value, especially for the azimuth (horizontal) axis slew drive. The system height can be used as the moment arm.

In case of a different PV array configuration, determining the center of mass will be important.

$M_{wind}=F_{wind}\times h_{sys}$ *(17)*

$M_{wind}=13056(N)\times 3(m)=39168(Nm)$

3. Moment Caused by Panel Weight (For Elevation Axis)

For the elevation (vertical) axis slew drive, the moment generated by the panels' own weight is significant. This moment is maximal when the panels are in a horizontal position and at the furthest point from the slew drive. This drive controls the mechanism that moves the panels up and down. The panels are arranged in a 2x4 configuration, meaning a deviation from the center of mass equivalent to 2 heights and 4 widths. The distance of each row from the elevation rotation axis will be "L/2".

Moment Arm (r_{el}): The horizontal distance of the panels' center of mass from the elevation axis will be $L/2=1(m)$.

Moment Caused by Panel Weight (M_{el}): $M_{el}=W\times r_{el}$ *(18)*

Moment of 1st row; $M_{cl}=(4\times 245.25)\,(N)\times 1(m)=981(Nm)$

Total moment of 2 rows; $M_{cl}=981(Nm)\times 2=1962(Nm)$

Since the moment arm value significantly alters the system's resistance, arranging PV panels in 2 vertical rows instead of 2 horizontal rows yielded more effective results. The most important criterion for designing such systems with 1-2-4-8-16 PV panels is their suitability for creating a symmetrical geometric structure. This allows the system drive and construction design to produce more balanced and efficient results.

4. Moment Caused by Wind Load (For Elevation Axis)

The wind moment on the elevation axis varies depending on the elevation (tilt) angle of the panels. The most critical situation occurs when the wind strikes the panel surfaces perpendicularly. In this case, the moment arm is the greatest perpendicular distance from the aerodynamic center on the panel surfaces to the elevation axis.

$$M_{elevation,wind} = F_{wind} \times r_{elevation,wind} \quad (19)$$

$$M_{elevation,wind} = 13056(N) \times 1(m) = 13056(Nm)$$

5. Moments of Inertia ($I_{azimuth}$, $I_{elevation}$)

Moment of inertia is a measure of an object's resistance to rotational motion. It specifically determines the torque that the motor and slew drive must overcome during acceleration and deceleration (dynamic) phases of the system. The total moment of inertia of the system is the sum of the moments of inertia of all panels and their supporting structure (frame, pole, etc.).

Simplifying the system by defining PV panels as flat plates, the moment of inertia of a single panel about its center of mass is:

$$I_{panel,center\_of\_mass,azimuth} = 1/12 \times m_p \times (L^2 + W^2) \quad (20)$$

$$I_{panel,center\_of\_mass,azimuth} = 1/12 \times 25(kg) \times (2^2 + 1^2)(m^2) = 10.41(kg \cdot m^2)$$

The moment of inertia created by the panels at the axis of rotation, according to the "*Parallel Axis Theorem*", is:

$$I_{total} = I_{center\_of\_mass,azimuth/elevation} + md^2 \quad (21)$$

$$I_{azimuth} = 533(kg \cdot m^2)$$

$$I_{panel,center\_of\_mass,elevation} = 1/12 \times m_p \times (L^2) \quad (22)$$

$$I_{panel,center\_of\_mass,elevation} = 1/12 \times 25(kg) \times 2^2(m^2) = 8.33(kg \cdot m^2)$$

$$I_{elevation} = 266.6(kg \cdot m^2)$$

The amount of dynamic torque required to overcome the highest moment within the moments of inertia is:

$$M_{dynamic} = I \times \alpha \quad (23)$$

Here, α is the angular acceleration variable. For solar tracking on the azimuth axis, angular acceleration can be taken as 0.1 rad/s^2.

$$M_{dynamic} = 533 \times 0.1 = 53.3(Nm)$$

6. Snow Load Calculation (M_{snow})

Snow load is a significant design criterion for solar panel systems, especially in regions with cold climates. The weight of accumulated snow on the panels can create substantial moments on the elevation axis.

A worst-case scenario can be established by assuming that snow falls uniformly on the panels and that the surface is in a suitable position for snow retention (slightly inclined). However, it should be remembered that

in solar tracking systems, panels are in their steepest (more vertical) position in winter. This is a hindering factor for snow accumulation.

Typical Snow Density (ρ_{snow}): Ranges from 50-200 kg/m³ for fresh snow and 200-500 kg/m³ for old/compacted snow. An average of 300 kg/m³ can be used for calculations.

Snow thickness (h_{snow}) can vary depending on the geography. Therefore, if snow load calculations are performed for 3 different scenarios (50 cm, 20 cm, 5 cm):

Snow Weight per Unit Area (q_{snow}):

$$q_{snow} = \rho_{sno} w \times h_{snow} \times g \quad (24)$$

$$q_{snow,50cm} = 300(kg/m^3) \times 0.5(m) \times 9.81(m/s^2) = 1471.5(N/m^2)\ (Pa)$$

Total Snow Force (F_{snow}):

$$F_{snow} = q_{snow} \times A_{total} \quad (25)$$

$$F_{snow} = 1471.5(N/m^2) \times 16(m^2) = 23544(N)$$

Moment Caused by Snow Load (M_{snow}): If the perpendicular distance in the elevation direction of the panels is taken as the basis:

$$M_{snow} = F_{snow} \times r_{elevation} \quad (26)$$

$$M_{snow,50cm} = 23544(N) \times 1(m) = 23544(Nm)$$

$$M_{snow,20cm} = 9417.6(Nm)$$

$$M_{snow,5cm} = 2354.4(Nm)$$

The moment resulting from snow load has turned out to be much higher than the wind moment (13056 N·m) and the panel weight moment (1962 N·m). This indicates that snow load is one of the most critical loads for slew drive and structural design, especially in regions experiencing harsh winter conditions. System design must either be capable of bringing the panels to a safe "snow position" (e.g., vertical or highly tilted) or capable of bearing this load under such extreme load conditions.

Table 5 Moments Acting on the PV Panel System

Moment Type	Axis	Calculated Value (N·m)	Notes
Wind Overturning Moment (M_{wind})	Azimuth	39168	Main determining load
Panel Weight Moment (M_{weight})	Elevation	1962	
Elevation Wind Moment ($M_{elevation,wind}$)	Elevation	13056	
Snow Load Moment (M_{snow})	Elevation	23544	0.5 m snow thickness assumption
		9417.6	0.2 m snow thickness assumption
		2354.4	0.05 m snow thickness assumption
Azimuth Moment of Inertia ($I_{azimuth}$)	Azimuth	533.36 (kg·m²)	
Elevation Moment of Inertia ($I_{elevation}$)	Elevation	266.64 (kg·m²)	

Table 5 summarizes the moments acting on the system. The maximum acting moments in this table were the wind overturning moment, snow load moment, and wind turning load moment. This table was generated assuming storm-level wind for wind load and 50-20-5 cm accumulation for snow load. There is also a possibility of exposure to loads exceeding these. In such cases, the system must have additional safety mechanisms.

Based on these calculations, the maximum moments that the system may encounter are as follows:

Azimuth (Horizontal) Axis Torque Requirement (Wind Load as Primary Factor): Approximately 39168 N·m. This is a critical value for the slew drive's overturning moment (OTM) capacity. Simultaneously, the nominal torque capacity of the drive must also meet or exceed this value. Wind load generally constitutes the largest torque load on the azimuth axis.

7. Evaluation for Slew Drive Selection

In the light of these calculations:

- For the azimuth axis, a slew drive with an overturning moment and torque capacity of approximately 40 kNm (40,000 N·m) is required. Worm gear slew drives are typically preferred for these types of loads. A drive with a gear diameter of 600 mm to 1000 mm or larger is needed.

- For the elevation axis, a slew drive with a torque capacity of approximately 10 kNm (10,000 N·m) may be sufficient. Worm gear slew drives are also generally suitable for this axis. A smaller diameter drive (e.g., 300 mm - 500 mm) might be adequate.

Safety Factors:

- Dynamic Effects: Dynamic loads occur when the system is in motion (acceleration and deceleration). This can increase the instantaneous torque requirements of the motor and drive.
- Building Codes and Standards: For systems installed on buildings, local building codes, wind load standards (e.g., ASCE 7, Eurocode 1), and specific standards for solar energy systems (e.g., UL 3703) must be considered.
- Safety Factors: In engineering design, a safety factor ranging typically from 20% to 100% is additionally applied to calculated maximum loads. This leaves room for contingencies, material fatigue, and calculation uncertainties. For example, if a 50% safety factor is applied to a 40 kNm torque, a drive with a 60 kNm capacity would be sought.
- Manufacturer Data: The most suitable model is determined by comparing the manufacturer-provided nominal torque, overturning moment capacity, radial load, and axial load values of the chosen slew drive with the calculated values. The gear diameter is directly related to these capacity values.

CHAPTER 6

6. Algorithm Samples and Microcontroller Connections

For solar tracking to be performed using the image processing method, the sun must first be detected by an imaging device. Therefore, it is essential that the intermediary component, which will command the system's motion mechanism, is a system element that works in harmony with the image acquisition components. Microcomputers marketed by brands such as Arduino, Raspberry Pi, BeagleBone, Odroid, Banana Pi, Adafruit, and Particle Photon etc. can be used for image acquisition and processing. Based on market prevalence, Arduino and Raspberry Pi microcomputers can be said to have an advantage. PLC elements, which have long been used in automation systems, can only be utilized as triggers for solar tracking via image processing. They require another unit to process the main image and send the triggering signal to the PLC. For this reason, they are not preferred for image processing tasks. Microcomputers capable of both image processing and triggering (outputting signals) in a single unit should be preferred. For this reason, Raspberry Pi stands out. The Arduino microcontroller board has inherent advantages in areas such as:

- Performing specific tasks in real-time and with precise timing because it does not run an operating system.
- Being more affordable.
- Providing lower power consumption.
- Having analog inputs.

Despite these, Raspberry Pi is preferred for solar tracking using the image processing method due to reasons such as:

- Offering a more powerful operating system and RAM capability.
- Being suitable for programming in languages like PYTHON, C++, and JAVA due to its Linux-based operation.
- Having extensive connectivity options (e.g., built-in USB ports, HDMI output, Ethernet port, Wi-Fi, and Bluetooth).
- Support for additional storage.
- Possessing a graphical interface (e.g. GUI)

When examining studies in the literature, it is observed that sensor-based solar tracking systems are implemented with Arduino, while image processing-based solar tracking systems are implemented with Raspberry Pi.

6.1. Hardware Integration and Data Processing in Raspberry Pi-Based Solar Tracking Systems

Raspberry Pi's versatile and programmable General-Purpose Input/Output (GPIO) pins expand its application areas. Figure 33 illustrates the general connection points and multi-functional GPIO pin layout of the Raspberry Pi 4 model. This layout encompasses various interfaces such as digital input/output, PWM (Pulse Width Modulation), I2C, SPI, and UART. The presence of 5V outputs provides the ability to control small linear actuators and servo motors without the need for an additional intermediate control element. However, the maximum current these pins can provide is limited, and an external power circuit is required for larger loads. I2C and UART pins enable the connection of analog and digital sensors, allowing for data acquisition and storage, and even remote data reading and system improvements.

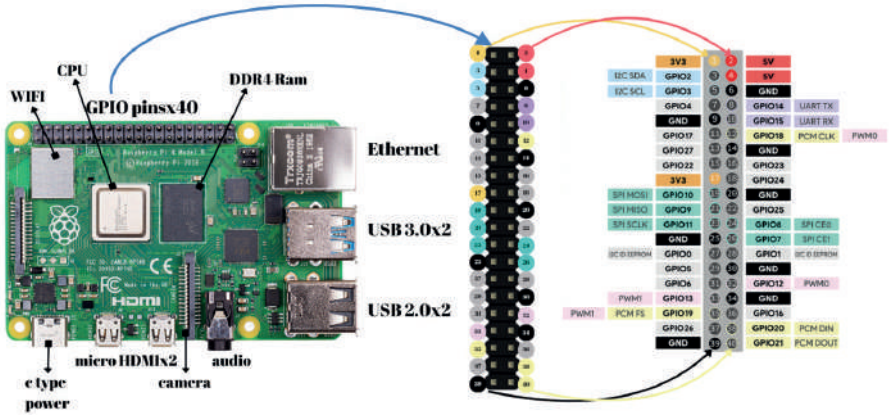


Figure 33 Raspberry Pi 4 connection and GPIO structure

In solar tracking systems, connecting a pyranometer to the system for instantaneous measurement of solar radiation is a common practice for reading instantaneous values, calculating system efficiency, and detecting faults. In commercial applications remotely monitored with SCADA systems, it is a mandatory application in large solar power plants. Pyranometers with RS485-RS232 communication protocols or those sending analog signals (in conjunction with an analog-to-digital converter, ADC) can be connected to Raspberry Pi's GPIO 14-15 pins to acquire instantaneous data, read and store data remotely, and even perform system improvements via remote access.

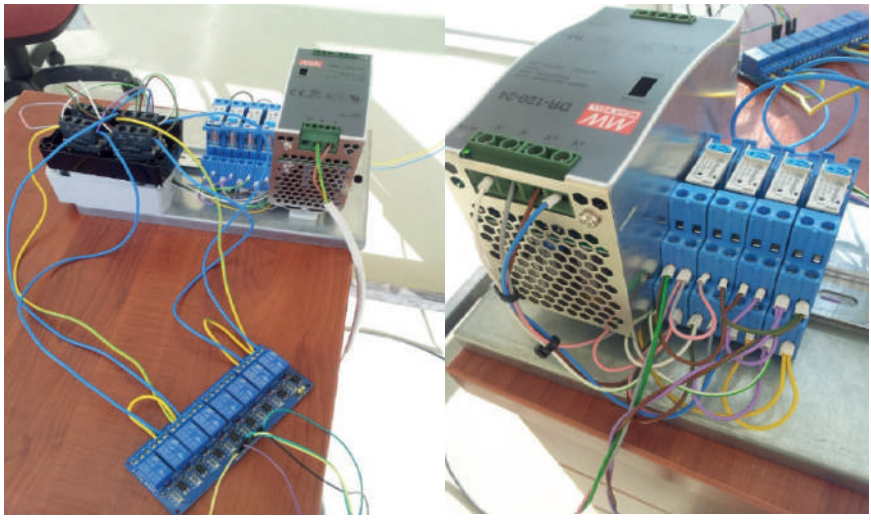


Figure 34 Variable power support and motor speed control element

However, for motor control applications requiring high current and/or voltage, motor control units (motor drivers) or power support circuit elements must be added. Figure 34 shows the circuit elements used for powering and controlling 24V DC and 220V AC linear actuators in a Raspberry Pi application designed for a solar tracking system. The direction of rotation of linear actuators can be controlled via electronic circuit elements or through programmable logic control via motor drivers.

Figure 34 includes an 8-channel 24V relay module. Its purpose is to prevent overloading of the microcontroller's output pin during sudden or excessive loads and to provide switching capability for high current/voltage values.

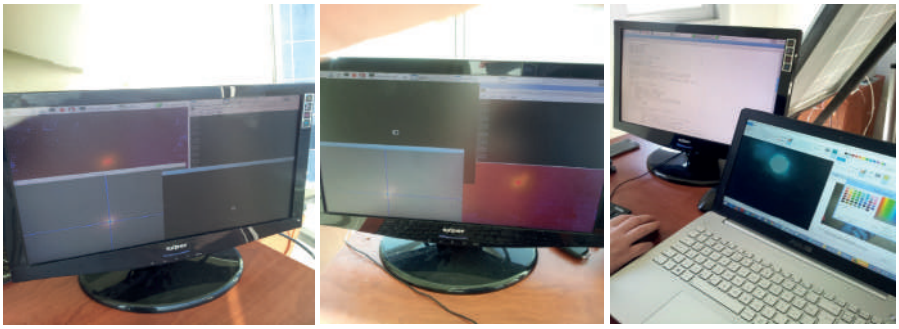


Figure 35 Outputs of the image processing algorithm for sun position detection

Figure 35 presents image processing outputs obtained at various times and from different solar tracking systems. These outputs visualize the perceived position of the sun and the operation of the tracking algorithm.

6.2. Image Processing-Based Algorithms in Solar Tracking Systems

The primary objective of solar tracking systems is to continuously monitor the sun's position in the sky to enhance the efficiency of solar panels. Image processing-based algorithms analyze visual data acquired through a camera to determine the precise position of the sun and use this information to orient the panels towards the sun at the correct angle.

6.2.1. Basic Steps of the Image Processing Algorithm

An image processing-based solar tracking algorithm typically consists of a series of sequential and logically dependent steps. These steps enable the transformation of raw camera data into meaningful positional information.

Image Acquisition:

The first and most critical step of the system is to obtain a digital image of the sun and the surrounding sky. Compatible camera modules are used with the microcomputer for this process. When selecting a camera, parameters such as resolution, field of view (FOV), frame rate and dynamic range should be considered. Especially for a wide tracking area, cameras with wide-angle (fisheye) lenses can be preferred, allowing the sun's horizon-to-horizon movement to be captured in a single image. The camera's mounting position and orientation directly affect the accuracy of the algorithm and require precise system calibration. The acquired image data is typically processed by a Single Board Computer (SBC), such as a Raspberry Pi.

Image Pre-processing:

Raw image data undergoes various pre-processing techniques before being made suitable for the sun detection step. The primary purpose of these steps is to reduce noise in the image, optimize contrast, and enhance the prominence of the sun.

- Grayscale Conversion: Color images are often converted to grayscale images to reduce processing load and for pixel density-based analysis. This conversion creates a single channel representing the brightness value of each pixel.
- Noise Reduction: Random noise originating from camera sensors or environmental factors can negatively impact algorithm accuracy. To eliminate this noise, spatial filtering techniques such as Gaussian filter, median filter, or bilateral filter are applied. These filters correct noisy pixels by using their neighborhood relationships.
- Contrast Adjustment: The overall brightness and contrast levels of the image can be adjusted to ensure better separation of the sun from the background. This can be achieved through methods such as histogram equalization or linear contrast stretching.

Image Segmentation:

After the pre-processing steps, the pixel group representing the sun in the image needs to be isolated. This is usually done by thresholding. Since the sun has a significantly higher brightness value compared to the surrounding sky regions, pixels above a certain brightness threshold are defined as the "sun region," and a binary mask is created.

- Fixed (Global) Thresholding: A specific brightness value is used as a fixed threshold for the entire image. This method is simple and fast but may perform poorly in changing light conditions (e.g., cloudiness).
- Adaptive Thresholding: Different threshold values are calculated for different regions of the image. This method is more robust against varying lighting conditions and can provide more accurate segmentation even when clouds partially obscure the sun. Otsu's method or local thresholding algorithms fall into this category.

Sun Contour Detection:

In the binary image obtained through image segmentation, the pixels representing the sun region are isolated. However, in this binary image, determining the contour (boundary) of the sun region is a critical step for subsequent analysis, especially for center detection. Contour detection is a process that finds the boundaries of interconnected pixel clusters.

The following steps should be followed for contour detection:

- Connected Component Analysis in Binary Image: In the binary image obtained after thresholding, clusters formed by connected (neighboring) white pixels are identified. The largest of these clusters (typically the sun itself) or those meeting a certain size criterion are defined as potential sun region candidates. This step can also be used to eliminate small noise regions or other bright spots in the image.
- Contour Finding Algorithms: On the detected connected components, contour finding algorithms offered by popular computer vision libraries (e.g., OpenCV) are applied. These algorithms extract the outer boundary pixel chains of the identified objects (in this case, the sun). The resulting contour is represented as a series of (x,y) coordinate pairs. This allows information about the sun's shape and size to be obtained.
- Selection of Outer Contour: In cases where multiple contours may be found (e.g., halos around the sun or small bright spots), the contour with the largest area or meeting a specific size criterion is usually selected as the main sun contour. This prevents misidentification of other objects as the sun.

Accurate determination of the sun's contour directly provides an input for the next step, precise detection of the sun's center. Furthermore, contour information can be used for additional analyses such as monitoring changes

in the sun's apparent size or evaluating how much of the sun is visible in conditions like partial cloudiness.

Centroid Detection of the Sun:

From the boundary coordinates obtained as a result of contouring or from the binary sun mask obtained as a result of segmentation, the geometric center (centroid) or center of mass representing the sun's position in the image is calculated. These center coordinates (x_c, y_c) are used as the primary input for guiding the solar tracking mechanism.

The center of mass calculation is performed by taking the weighted average of all "sun" pixels in the segmented region:

$$x_c = \frac{\sum_{(i,j \in S)} i.I(i,j)}{\sum_{(i,j \in S)} I(i,j)} \tag{27}$$

$$y_c = \frac{\sum_{(i,j \in S)} j.I(i,j)}{\sum_{(i,j \in S)} I(i,j)} \tag{28}$$

Here, (i,j) represents the pixel coordinates, I(i,j) represents the pixel intensity value (brightness), and S represents the segmented sun region. In a simpler approach, the center of mass can also be found by taking the average coordinates of only the active pixels (white pixels) in the binary image.

Tracking Control:

The calculated sun center coordinates (x_c, y_c) are compared with the system's current panel angle or the targeted position at the camera center. The resulting error signal from this comparison is used to control actuators such as linear actuators or servo motors. The control strategy is generally based on a feedback control loop principle, moving the panel towards the targeted sun center.

- Proportional-Integral-Derivative (PID) Control: A powerful control method frequently preferred in solar tracking systems. A PID controller uses the error signal (the difference between the sun's current position and its targeted position) to determine the commands to be sent to the actuators. PID combines proportional (P), integral (I), and derivative (D) components to ensure that the system responds quickly and reaches its target stably. It is ideal for dynamic and precise tracking.

- Hysteresis Control: To prevent unnecessary continuous movement of the actuators and reduce energy consumption, a specific "error band" can be defined. The panel does not move unless it moves outside this defined hysteresis band. This extends the lifespan of the actuators and reduces mechanical wear.
- Model Predictive Control: In advanced systems, a hybrid tracking strategy can be followed by combining astronomical sun position information with image processing data. This allows for prediction-based tracking to continue even under instantaneous environmental conditions such as cloudiness.

Error Management and Advanced Features:

Various error management and advanced features can be integrated to increase the stability and reliability of the algorithm:

- Cloud and Obstruction Detection: Image analysis can detect situations where the sun is partially or completely obscured by clouds. In this case, the system can transition to a specific "park" position or revert to astronomical tracking (movement at a constant angular speed) to prevent unnecessary movements or conserve energy.
- Low Light/Night Mode: Image processing algorithms may not function accurately enough during times of low sun brightness, such as in sunrise and sunset time intervals. Also, tracking is unnecessary during night hours when there is no sun. Under these conditions, the system should be enabled to switch to pre-programmed astronomical tracking algorithms or a specific "sleep" position.
- Camera Setting Adaptation: Camera settings such as exposure time, ISO sensitivity, and white balance can be dynamically adjusted according to ambient brightness conditions. This ensures that high-quality images are obtained even under different lighting conditions (e.g., very bright sunlight or twilight).
- Sudden Changes in Sun Center: Instantaneous jumps in (x_c, y_c) values caused by glare in the sky, reflections from surrounding reflective surfaces or the momentary perception of artificial light sources as a "fake" sun can disrupt the stability of solar tracking. To prevent this, a cumulative sun center coordinate determination algorithm should be added. This allows for centering by averaging a determined number of center coordinates, thereby creating movement stability.

6.2.2. Image Processing Algorithm Samples

Considering that the structure described above will be operated with a Raspberry Pi, before writing the code, the camera's CSI connection must be established, and PYTHON3 should be installed. The OPENCV library should be installed with the command;

"pip install opencv-python",

and the RPi.GPIO library with

"pip install RPi.GPIO".

6.2.2.1. Sample Code-Main Body

```
import cv2

import numpy as np

import time

import RPi.GPIO as GPIO # For controlling Raspberry Pi's GPIO pins

# --- 1. GPIO Pin Definitions and Target Settings ---

# These will be the ENABLE pins or direction control pins for our motor driver.

PAN_FWD_PIN = 17 # Pan/Azimuth (horizontal) forward/right movement pin

PAN_BWD_PIN = 18 # Pan/Azimuth (horizontal) backward/left movement pin

TILT_UP_PIN = 27 # Tilt/Elevation (vertical) up movement pin

TILT_DOWN_PIN = 22 # Tilt/Elevation (vertical) down movement pin

# System's target center (ideal focal point of the camera, half of the image width)

# A 640x480 resolution value determines the target (x,y) value, thus it is important.

TARGET_X = 320 # (Image width / 2)

TARGET_Y = 240 # (Image height / 2)
```

```
  # Tolerance zone (pixels) before movement. The system
does not move within this value (hysteresis).
  TOLERANCE = 15 # If the sun is within this pixel
range, do not move the panel.
  # --- 2. GPIO Setup Functions ---
  def setup_gpio():
      """Sets up GPIO pins as outputs."""
      GPIO.setmode(GPIO.BCM) # Use BCM pin numbering
mode
      GPIO.setup([PAN_FWD_PIN, PAN_BWD_PIN, TILT_UP_
PIN, TILT_DOWN_PIN], GPIO.OUT)
     # Initially set all pins to LOW to stop actuators
      GPIO.output([PAN_FWD_PIN, PAN_BWD_PIN, TILT_UP_
PIN, TILT_DOWN_PIN], GPIO.LOW)
      print("GPIO pins successfully configured.")
  def cleanup_gpio():
         """Cleans up GPIO pins when the program
terminates."""
      GPIO.cleanup()
      print("GPIO pins cleaned up.")
  # --- 3. Actuator Control Function ---
  def control_actuator(current_cx, current_cy):
      # Controls the actuators based on the current
center position of the sun.
      # current_cx (int): X coordinate of the sun in
the image.
      # current_cy (int): Y coordinate of the sun in
the image.
      # X-axis (Pan/Azimuth) control
      if current_cx < TARGET_X - TOLERANCE:
          # Sun is to the left of the target center,
move panel to the right (Fwd)
```

```
            GPIO.output(PAN_BWD_PIN, GPIO.LOW)
            GPIO.output(PAN_FWD_PIN, GPIO.HIGH)
            # print("Pan/Azimuth: Moving right.")
        elif current_cx > TARGET_X + TOLERANCE:
            # Sun is to the right of the target center,
move panel to the left (Bwd)
            GPIO.output(PAN_FWD_PIN, GPIO.LOW)
            GPIO.output(PAN_BWD_PIN, GPIO.HIGH)
            # print("Pan/Azimuth: Moving left.")
        else:
              # At target on X-axis, stop Pan/Azimuth
movement
            GPIO.output(PAN_FWD_PIN, GPIO.LOW)
       GPIO.output(PAN_BWD_PIN, GPIO.LOW)
           # print("Pan/Azimuth: At target (Stopped).")
        # Y-axis (Tilt/Elevation) control
        if current_cy < TARGET_Y - TOLERANCE:
           # Sun is above the target center, move panel up
            GPIO.output(TILT_DOWN_PIN, GPIO.LOW)
            GPIO.output(TILT_UP_PIN, GPIO.HIGH)
            # print("Tilt/Elevation: Moving up.")
        elif current_cy > TARGET_Y + TOLERANCE:
           # Sun is below the target center, move panel
down
            GPIO.output(TILT_UP_PIN, GPIO.LOW)
       GPIO.output(TILT_DOWN_PIN, GPIO.HIGH)
            # print("Tilt/Elevation: Moving down.")
        else:
             # At target on Y-axis, stop Tilt/Elevation
movement
```

```
            GPIO.output(TILT_UP_PIN, GPIO.LOW)
            GPIO.output(TILT_DOWN_PIN, GPIO.LOW)
        # print("Tilt/Elevation: At target (Stopped).")
# --- 4. Main Application Loop ---
if __name__ == "__main__":
    setup_gpio() # Configure GPIO pins
    # Camera initialization
     # For Raspberry Pi camera, '0' or appropriate backend for 'libcamera' based systems can be used.
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
         print("Error: Could not open camera. Check camera connection or index.")
         cleanup_gpio()
         exit()
    # Set camera resolution
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
    time.sleep(2)
   print("Video stream started. Press 'q' to exit.")
    try:
        while True:
             ret, frame = cap.read() # Read a frame from the camera
            if not ret:
                print("Error: Could not read frame. Camera connection might be lost.")
                break
            # 4.1.2. Image Preprocessing
            # Convert to grayscale
```

```
                gray_frame = cv2.cvtColor(frame, cv2.
COLOR_BGR2GRAY)
              # Reduce noise with Gaussian filter
               blurred_frame = cv2.GaussianBlur(gray_
frame, (5, 5), 0)
```

 # 4.1.3. Sun Segmentation (Thresholding)

```
        # A high threshold value is used because the
sun is bright.
             # THRESH_BINARY_INV (inverse binary) can
sometimes yield better results
               # because everything outside the sun
becomes black while the sun remains white.
             # For example: ret, binary_frame = cv2.
threshold(blurred_frame, 200, 255, cv2.THRESH_BINARY)
                # Adaptive thresholding can also be
considered for scenarios where the sun might be dim.
             _, binary_frame = cv2.threshold(blurred_
frame, 200, 255, cv2.THRESH_BINARY)
```

 # 4.1.4. Determining Sun Contour

```
             # Find only outer contours
              contours, _ = cv2.findContours(binary_
frame, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
             sun_cx, sun_cy = None, None
             if contours:
                # Assume the contour with the largest
area is the sun
                    largest_contour = max(contours,
key=cv2.contourArea)
                 # We can add a minimum area check to
ignore very small contours
                 if cv2.contourArea(largest_contour)
> 100: # Example min area threshold
```

```
                        # Calculate moments from the
contour
                    M = cv2.moments(largest_contour)
                     # 4.1.5. Determining Sun Center
(Centroid)
                     if M["m00"] != 0:
                            sun_cx = int(M["m10"] /
M["m00"])
                            sun_cy = int(M["m01"] /
M["m00"])
                        # Draw a circle at the center
of the sun
                          cv2.circle(frame, (sun_cx,
sun_cy), 10, (0, 255, 255), -1) # Yellow circle
                           cv2.putText(frame, f"Sun:
({sun_cx},{sun_cy})", (sun_cx + 20, sun_cy + 20),
                                      cv2.FONT_HERSHEY_
SIMPLEX, 0.6, (0, 255, 255), 2)
                          # 4.1.6. Tracking Control
                            control_actuator(sun_cx,
sun_cy)
                    else:
                      print("Warning: Valid contour
area is zero.")
                         # If sun is not detected,
stop actuators
                        control_actuator(TARGET_X,
TARGET_Y) # Command to stop by targeting
                else:
                       print("Warning: Detected sun
contour is too small.")
                    # If sun is not detected, stop
actuators
```

```
                        control_actuator(TARGET_X,
TARGET_Y) # Command to stop by targeting
            else:
                print("Warning: No sun contour found
in the image.")
                    # If sun is not detected, stop
actuators
                control_actuator(TARGET_X, TARGET_Y)
# Command to stop by targeting
             # Visualize images (these lines can be
removed if there is no GUI on Raspberry Pi)
            cv2.imshow('Original Frame', frame)
             cv2.imshow('Binary Sun', binary_frame)
# Show the segmented binary image
            # Exit loop when 'q' is pressed
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
    except KeyboardInterrupt:
        print("\nProgram stopped by user.")
    finally:
        # Release resources and clean up GPIO
        cap.release()
        cv2.destroyAllWindows()
        cleanup_gpio()
```

The example code has a simple and straightforward structure. This structure can be made more robust and stable by adding certain features. However, it should be noted that every additional feature will increase CPU usage and cause delays in movement commands. Therefore, it is recommended to start with the minimum required features and improve the system according to its needs.

6.2.2.2. Transitioning from Hysteresis-Based Control to PID-Based Control

To implement PID control, it is first needed to define the PID parameters (Kp, Ki, Kd) and calculate separate error values for each axis (pan/azimuth and tilt/elevation). The signals sent to the actuators are converted into PWM (Pulse Width Modulation) modules instead of simple HIGH/LOW, thereby controlling the motor speed. This provides smoother and more proportional movements. It is essential to ensure that the actuators or motor driver module have PWM capability, otherwise, the result will be ON/OFF control.

```
# --- 2. PID Parameters ---
#(Kp: Proportional, Ki: Integral, Kd: Derivative)
# PID gains for the Pan axis
KP_PAN = 0.5 # Proportional gain (Large error = Large response)
KI_PAN = 0.01 # Integral gain (Corrects systematic errors)
KD_PAN = 0.05 # Derivative gain (Responds to the rate of error change, reduces oscillation)
# PID gains for the Tilt axis
KP_TILT = 0.5
KI_TILT = 0.01
KD_TILT = 0.05
```

- *Kp (Proportional Gain):* Responds directly to the current error. As the error increases, the Kp term also increases, sending a larger correction signal to the system. This provides a fast response, but high Kp values cause oscillation.
- *Ki (Integral Gain):* Corrects errors accumulated over time (i.e., the system's inability to perfectly settle on the target). It is used to eliminate small, persistent errors (steady-state error).
- *Kd (Derivative Gain):* Responds to the rate of change of the error. When the error changes rapidly, it applies a "brake" to the system, reducing overshoot and making the system more stable.

Proper tuning of these parameters (PID tuning) is of critical importance and is performed using methods such as trial-and-error or the Ziegler-Nichols method.

Additionally, a PID control class needs to be defined in PYTHON:

```
class PIDController:
    def __init__(self, Kp, Ki, Kd):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.previous_error = 0
        self.integral = 0
        self.last_time = time.time()
    def update(self, error):
        current_time = time.time()
        dt = current_time - self.last_time
        self.integral += error * dt # Integral term
         derivative = (error - self.previous_error)
/ dt if dt > 0 else 0 # Derivative term
         output = self.Kp * error + self.Ki * self.
integral + self.Kd * derivative # PID output
        self.previous_error = error
        self.last_time = current_time
        return output
# Create PID controllers
pid_pan = PIDController(KP_PAN, KI_PAN, KD_PAN)
pid_tilt = PIDController(KP_TILT, KI_TILT, KD_TILT)
```

The PWM motor driver needs to be connected to PWM-enabled GPIO pins and defined via code.

```
# --- 3. GPIO Setup Functions ---
def setup_gpio():
      """Sets up GPIO pins as outputs and starts
PWM."""
    GPIO.setmode(GPIO.BCM)
```

```
        # Direction pins
         GPIO.setup([PAN_A_PIN, PAN_B_PIN, TILT_A_PIN,
TILT_B_PIN], GPIO.OUT)
        GPIO.output([PAN_A_PIN, PAN_B_PIN, TILT_A_PIN,
TILT_B_PIN], GPIO.LOW)
        # PWM pins
         GPIO.setup([PAN_PWM_PIN, TILT_PWM_PIN], GPIO.
OUT)
        # Create PWM objects (e.g., 100 Hz frequency)
        global pan_pwm, tilt_pwm
        pan_pwm = GPIO.PWM(PAN_PWM_PIN, 100) # 100 Hz
        tilt_pwm = GPIO.PWM(TILT_PWM_PIN, 100) # 100 Hz
         pan_pwm.start(0)   # Initially 0% duty cycle
(motor off)
         tilt_pwm.start(0) # Initially 0% duty cycle
(motor off)
      print("GPIO pins and PWM successfully configured.")
   def cleanup_gpio():
        """Cleans up GPIO pins and PWM when the program
terminates."""
        pan_pwm.stop()
        tilt_pwm.stop()
        GPIO.cleanup()
        print("GPIO pins and PWM cleaned up.")
```

The function controlling the actuators should be configured to operate based on error signals from the PID controller, rather than directly on current_cx and current_cy positions:

```
   # --- 4. Actuator Control Function (Uses PID Output)
---
   def control_actuators_with_pid(error_x, error_y):
        """
```

```
    Controls actuators with PID based on the current
error signal of the sun.
    """
    # X-axis (Pan) control
      if abs(error_x) > TOLERANCE: # If error is
outside tolerance
        pan_output = pid_pan.update(error_x) # Get
PID output
        pan_speed = min(abs(int(pan_output * 10)),
100) # Scale and limit speed to 0-100
        if pan_output > 0: # If sun is to the left,
move right
            GPIO.output(PAN_A_PIN, GPIO.HIGH)
            GPIO.output(PAN_B_PIN, GPIO.LOW)
        else: # If sun is to the right, move left
            GPIO.output(PAN_A_PIN, GPIO.LOW)
            GPIO.output(PAN_B_PIN, GPIO.HIGH)
       pan_pwm.ChangeDutyCycle(pan_speed) # Set PWM
duty cycle
    else: # If within tolerance, stop
        GPIO.output(PAN_A_PIN, GPIO.LOW)
        GPIO.output(PAN_B_PIN, GPIO.LOW)
        pan_pwm.ChangeDutyCycle(0)
     # Y-axis (Tilt) control (similar logic to Pan
control)
    if abs(error_y) > TOLERANCE:
        tilt_output = pid_tilt.update(error_y)
       tilt_speed = min(abs(int(tilt_output * 10)),
100)
        if tilt_output > 0: # If sun is up, move up
            GPIO.output(TILT_A_PIN, GPIO.HIGH)
```

```
            GPIO.output(TILT_B_PIN, GPIO.LOW)
        else: # If sun is down, move down
            GPIO.output(TILT_A_PIN, GPIO.LOW)
            GPIO.output(TILT_B_PIN, GPIO.HIGH)
        tilt_pwm.ChangeDutyCycle(tilt_speed)
    else:
        GPIO.output(TILT_A_PIN, GPIO.LOW)
        GPIO.output(TILT_B_PIN, GPIO.LOW)
        tilt_pwm.ChangeDutyCycle(0)
```

In the main loop, after the current center of the sun is determined, error signals are calculated relative to the target center, and these error signals are passed to the control_actuators_with_pid function.

```
# --- 5. Main Application Loop ---
if __name__ == "__main__":
    setup_gpio() # Configure GPIO pins and PWM
    # ... camera initialization and other preliminary steps ...
    try:
        while True:
            # ... camera frame reading and image processing steps ...
            if contours:
                # ... sun center (sun_cx, sun_cy) calculation ...
                if M["m00"] != 0:
                    sun_cx = int(M["m10"] / M["m00"])
                    sun_cy = int(M["m01"] / M["m00"])
                    # Calculate error signals
```

```
                        error_x = sun_cx - TARGET_X
                        error_y = sun_cy - TARGET_Y
                        # Control actuators with PID
                    control_actuators_with_pid(error_x,
error_y)
                    else:
                         # If sun is not detected, stop
motors
                       control_actuators_with_pid(0, 0)
                else:
                   # If sun is not detected, stop motors
                    control_actuators_with_pid(0, 0)
               # ... visualization and exiting the loop
...
        except KeyboardInterrupt:
            # ... cleanup ...
        finally:
            # ... cleanup ...
```

After these additions are integrated into the code, tuning the PID gains (Kp, Ki, Kd) becomes the most crucial step. This adjustment ensures that the panel tracks the sun quickly, stably, and accurately.

6.2.2.3. Integration of Adaptive Thresholding:

The use of "*cv2.threshold*" in the example code prevents bright objects below a certain brightness value from being detected as the sun. This prevents false positives where other bright sources are mistaken for the sun. However, during sunrise/sunset, when the sun is behind clouds, or in situations with high dust levels, if the measured brightness value falls below the threshold, the sun may not be detected. To eliminate this problem, adaptive thresholding offers a robust solution. The changes to be made in the example code are as follows:

Replace:

```
# 4.1.3. Sun Segmentation (Thresholding)

# A high threshold value is used because the sun is bright.

_,binary_frame = cv2.threshold(blurred_frame, 200, 255, cv2.THRESH_BINARY)
```

with the following code structure:

```
# 4.1.3. Sun Segmentation (Adaptive Thresholding)

# Segment the sun region with adaptive thresholding. ADAPTIVE_THRESH_GAUSSIAN_C: Applies a Gaussian window to the weighted average neighborhood value.

# THRESH_BINARY: Sets pixels greater than the threshold value to max_val (255), others to 0.

# blockSize: Size of the neighborhood area to be used for calculating the pixel's threshold value (must be an odd number, e.g., 11).

# C: A constant subtracted from the mean or weighted mean (usually a positive number, e.g., 2).

binary_frame = cv2.adaptiveThreshold(blurred_frame, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

# Additional Step (Optional, Enhancing Contour):

# After adaptive thresholding, morphological operations (e.g., opening) can be applied to the resulting binary image

# to reduce noise and make the contour smoother.

kernel = np.ones((3,3),np.uint8)

binary_frame = cv2.morphologyEx(binary_frame, cv2.MORPH_OPEN, kernel)

binary_frame = cv2.morphologyEx(binary_frame, cv2.MORPH_CLOSE, kernel)
```

6.2.2.4. Steps to Improve Sun Detection:

Under varying light conditions, partial cloud cover, reflections or the presence of other bright objects, the code may lead to erroneous detections. To address these issues and enhance the robustness and accuracy of the sun detection algorithm, shape-based filtering, color-based segmentation, and small object filtering strategies can be integrated.

From a camera's perspective, the sun typically appears as a bright object with a circular or elliptical shape. This geometric property provides a strong distinguishing feature to differentiate the sun from other bright but non-circular objects (e.g., reflections from windows, lamp lights). Mathematically, the circularity of a contour is generally calculated using the following formula:

$$C = \frac{4\pi . A}{P^2} \tag{29}$$

Here:

C: Circularity ratio (between 0 and 1, where 1 indicates a perfect circle).

A: Area of the contour.

P: Perimeter of the contour.

```
  # ... (previous code: creation of gray_frame,
blurred_frame, binary_frame) ...
  # 4.1.4. Determining Sun Contour
  # Find only outer contours
  contours, _ = cv2.findContours(binary_frame, cv2.
RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
  sun_cx, sun_cy = None, None
  candidate_sun_contour = None # Candidate sun contour
  if contours:
      # Filter contours
      filtered_contours = []
      for contour in contours:
```

```
            area = cv2.contourArea(contour)
            # We can add a minimum area check to ignore
very small contours
                if area > 100: # Example minimum area
threshold (pixel^2)
               # Shape-based filtering: Circularity check
                perimeter = cv2.arcLength(contour, True)
                 if perimeter > 0: # If perimeter is not
zero (not a point or very small contour)
                      circularity = (4 * np.pi * area) /
(perimeter ** 2)
                     # Circularity threshold: Close to 1
for a perfect circle.
                      # For the sun, a threshold between
0.7 - 0.95 generally yields good results.
                       if circularity > 0.65: # Example
circularity threshold
                        filtered_contours.append(contour)
        if filtered_contours:
             # Select the contour with the largest area
among the filtered contours
           candidate_sun_contour = max(filtered_contours,
key=cv2.contourArea)
            # ... (from here, M = cv2.moments(candidate_
sun_contour) and center calculation continue) ...
        else:
              print("Warning: No contour satisfying the
circularity criterion was found. Actuators stopped.")
            control_actuators_with_pid(0, 0)
    else:
          print("Warning: No sun contour found in the
image. Actuators stopped.")
```

```
control_actuators_with_pid(0, 0)
```

To perform Color-Based Segmentation, it is necessary to use the HSV Color Space. Direct or indirect sunlight usually has a distinct color profile (from bright yellow to white). By using this color information, the accuracy of segmentation before or after thresholding can be increased. Since the RGB color space is sensitive to light intensity, the HSV (Hue, Saturation, Value) color space, which separates color information from intensity, is more suitable. The sun typically has high brightness (Value) and a specific hue range (yellow-orange-white tones). On a bright day, the saturation (Saturation) value is also high. By using these properties, converting the image to HSV and masking pixels that fall within a specific HSV range is an effective method to separate the sun from other objects.

```
ret, frame = cap.read()
if not ret:
      print("Error: Could not read frame. Camera connection might be lost.")
    break
```

 # 4.1.1. Color-Based Segmentation (HSV Masking for Sun)

```
# Convert image to HSV color space
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
# Define HSV color range for the sun
# These ranges should be adjusted according to the environment and camera color calibration.
# For example: For bright yellow-white colors
lower_sun = np.array([20, 100, 150]) # Hue, Saturation, Value (Min)
upper_sun = np.array([40, 255, 255]) # Hue, Saturation, Value (Max)
# Mask pixels within the defined HSV range
hsv_mask = cv2.inRange(hsv_frame, lower_sun, upper_sun)
# Apply the mask to make only the sun regions white (black background)
```

```
  # Note: This mask can be used directly instead of
the thresholded binary_frame or combined with it.

  # A "color-based" binary_frame is created using only
the mask.

  # This mask can then be given as input to adaptive
thresholding.

  # Apply the mask to the grayscale image (keep only
the sun region bright)

  masked_gray_frame = cv2.bitwise_and(gray_frame,
gray_frame, mask=hsv_mask)

  # 4.1.2. Image Preprocessing (Now on masked gray
frame)

  # Reduce noise with Gaussian filter

  blurred_frame = cv2.GaussianBlur(masked_gray_frame,
(5, 5), 0)

  # 4.1.3. Sun Segmentation (Adaptive Thresholding-
now more targeted)

  # blurred_frame already receives a cleaner input
after passing through the HSV mask.

  binary_frame = cv2.adaptiveThreshold(blurred_frame,
255,   cv2.ADAPTIVE_THRESH_GAUSSIAN_C,   cv2.THRESH_
BINARY, 11, 2)

  # ... (Morphological operations and contour finding
continue afterwards) ...
```

6.2.2.5. Cases Where Sun Is Not Detected- Astronomical Algorithm Integration: Calculating Sun Position:

Image processing-based detection methods become inoperative when the sun is not directly visible (e.g., at night, very dense clouds, fog) or when the camera malfunctions. In these scenarios, using astronomical algorithms to accurately estimate the current position of the sun prevents the system from being "blind." These algorithms calculate the sun's azimuth (horizontal angle) and elevation (vertical angle) using geographic location (latitude, longitude), date, and time information. Python libraries like "PyEphem"

are highly suitable for this integration. PyEphem simplifies astronomical calculations and provides an easy-to-use interface.

The PyEphem library is installed with the command "*pip install ephem*".

This library requires location information for astronomical calculations. Therefore, latitude, longitude, and altitude values for a specific region must be entered into the code. For illustrative purposes, the code has been created for the Denizli/Türkiye region.

```
import ephem # For astronomical calculations

# --- Geographic Location Definitions ---

# Latitude, longitude, and altitude of the sun tracking system's location

# Example: Approximate values for Denizli, Türkiye

LATITUDE = '37.77' # Latitude (degrees, as string for PyEphem)

LONGITUDE = '29.08' # Longitude (degrees, as string for PyEphem)

ELEVATION = 350     # Altitude (meters)

# --- New Function: Astronomical Sun Position Calculation ---

def get_solar_position_astronomical(latitude, longitude, elevation):

    """

      Calculates the sun's azimuth and elevation angles for a given location and current time.

    Returns:

        tuple: (azimuth_deg, elevation_deg) Sun's azimuth and elevation angles in degrees.

               Azimuth is usually measured clockwise from North.

              Sun elevation is measured upwards from the horizon.

    """

    obs = ephem.Observer()
```

```
        obs.lat = latitude
        obs.lon = longitude
        obs.elevation = elevation
        obs.date = ephem.now() # Use current time
        sun = ephem.Sun()
        sun.compute(obs)
        # Convert azimuth and sun elevation angles from
radians to degrees
         # PyEphem measures azimuth from South towards
West, so we convert it to clockwise from North.
         # Azimuth angle in PyEphem ranges from 0-2pi
radians.
        azimuth_deg = np.degrees(sun.az)
        elevation_deg = np.degrees(sun.alt)
         # Convert PyEphem's azimuth to 0-360 degrees
(clockwise from North)
        # Ephem's azimuth looks South at 0.0. Increases
towards West.
         # Therefore, a conversion for clockwise from
North will be needed.
          # For example: 0 (South), pi/2 (West), pi
(North), 3pi/2 (East)
         # For us, usually 0 (North), 90 (East), 180
(South), 270 (West) is desired.
        # Simply, there might be a 180-degree difference
with PyEphem's Azimuth.
       # Example conversion: PyEphem's azimuth (South=0,
West=90, North=180, East=270)
          # Most applications use North=0, East=90,
South=180, West=270.
       # For now, let's convert directly to degrees and
calibrate later if necessary.
```

```
    return azimuth_deg, elevation_deg
```

 # --- Usage Logic Within the Main Loop ---

```
# This function is called when the sun cannot be found in the image or when it is night.

# This code snippet should be placed inside the main loop, i.e., within the 'if __name__ == "__main__":' block.

# ... (Camera initialization and other setups in your existing code) ...

# Assume it is night when the sun's elevation falls below 0 degrees.

# This threshold can be adjusted (e.g., -5 degrees for astronomical twilight).

SUNSET_THRESHOLD_ELEVATION = 0 # Elevation threshold determining sunset

try:

    while True:

        ret, frame = cap.read()

         is_sun_detected_visually = False # Was the sun detected visually?

          # ... (Image processing and sun detection code) ...

              # If sun_cx, sun_cy are successfully calculated after your existing 'if contours:' block:

         if sun_cx is not None and sun_cy is not None:

             is_sun_detected_visually = True

              # ... (Error calculation and control_actuators_with_pid call) ...

          # If sun is not detected visually or it is night, use astronomical position

          if not is_sun_detected_visually:

              # Calculate astronomical sun position
```

```
                    astro_azimuth, astro_elevation =
get_solar_position_astronomical(LATITUDE, LONGITUDE,
ELEVATION)

                # Check if the sun is above the horizon

                if astro_elevation > SUNSET_THRESHOLD_
ELEVATION:

                    print(f"Warning: Sun not visually
detected, but astronomically at {astro_azimuth:.2f}°
Azimuth, {astro_elevation:.2f}° Elevation. Directing
to astronomical position.")

                # Determine panel's target coordinates
from astronomical angles.

                    # This part requires calibration
according to your panel's and camera's mounting angle.

                    # For example: Let's say when the
panel faces due North, Azimuth is 0 and Camera's X
center is 320.

                    # Or when Elevation is 90 degrees
(directly overhead), Y center is 240.

                    # These conversions can be complex
and depend on your system's mechanical calibration.

                # Simple example conversion (requires
actual calibration):

                    # Angle-to-pixel conversion is done
using the camera's field of view (FOV) and resolution.

                # Example: Assume camera's horizontal
FOV is 60 degrees and vertical FOV is 45 degrees.

                # And the camera's center corresponds
to Astro 0,0.

                    # A simple assumption: The panel's
midpoint corresponds to the camera's exact center.

                    # And in this code, we still expect
pixel error. Then pixels are calculated according to
these angles.
```

```
                    # Solution: A separate motor control
routine that directly moves the panel according to
astronomical angles.
                    # Or, a model is created to convert
astronomical angle to pixel coordinates.
                    # For now, by setting the error to
0, the panel is made to stop or brought to a default
position.
                  # If direct PID control of the panel's
angle is desired,
                    # an angular error signal needs to
be fed to the PID.
                  # Simply, here we can issue a "system
reset" or "go to default direction" command.
                    control_actuators_with_pid(0, 0) #
Stop motors (or direct to a default position)
                  # A new control function that sends
a direct angular target value can be added here.
                        # For example: move_panel_to_
angle(astro_azimuth, astro_elevation)
               else:
                 print ("Warning: Sun is astronomically
below the horizon (night). It is recommended to put
the system into sleep mode.")
                    # Night mode or park position can
be triggered
                  # For example: Call the go_to_park_
position () function
                    control_actuators_with_pid(0, 0) #
Stop motors
                  time. sleep (300) # Wait for example
5 minutes, then check again
            # ... (Visualize images and exit when 'q'
is pressed) ...
```

```
except KeyboardInterrupt:
    # ... cleanup ...
finally:
    # ... cleanup ...
```

In situations where sun tracking systems are not actively tracking the sun (night, storm, prolonged cloudiness, or maintenance), it is crucial to bring the panels to a safe and energy-saving position. This position is called the "*park position*".

A function like "*control_actuators_with_pid*" can move the panel to a specific angle or, more simply, stop the motors.

```
# --- New Function: Go to Park Position ---
def go_to_park_position():
    """
    Moves the panel to a predefined park position.
    It should be noted that this function requires a mechanism that can control the panel's angular position (e.g., motor with encoder)
    or a special motor control routine that brings the angle to a specific degree.
    Here is just an example of stopping motors or directing to a default position.
    """
        print("System is being directed to park position...")
    # Assumption: The panel is parked in a horizontal position (or the position with lowest wind resistance).
        # This requires the actuators to move to a certain point.
    # It should be combined with limit switches or existing angle readers.
    # Example: Panel horizontal (0 degrees elevation) and a specific azimuth (e.g., East)
```

```
        # To reach these values, PID can be run with an angular target or motors can be run for a fixed duration.
      # Simply, stop the motors
        control_actuators_with_pid(0, 0) # Call PID controller with error 0, which stops the motors.
      # Or run motors in one direction for a certain duration to go to the default position
      # (WARNING: Limit switches OR angular feedback sensors MUST be used with this method)
         # GPIO.output(TILT_DOWN_PIN, GPIO.HIGH) # Default: Move panel down
        # time.sleep(10) # Move down for 10 seconds (Example, adjusted according to mechanical duration)
  # GPIO.output(TILT_DOWN_PIN, GPIO.LOW) # Stop
  print("System in park position.")
  # --- Usage Logic Within the Main Loop ---
  # This code snippet should be placed inside the main loop, i.e., within the 'if __name__ == "__main__":' block.
  # ... (Camera initialization and other setups in your existing code) ...
  try:
      while True:
          # ... (Image processing and sun detection code) ...
          if not is_sun_detected_visually: # If sun is not visually detected
                  astro_azimuth, astro_elevation = get_solar_position_astronomical(LATITUDE, LONGITUDE, ELEVATION)
              if astro_elevation <= SUNSET_THRESHOLD_ELEVATION: # If sun is below the horizon (night)
```

```
                print("Warning: Sun is astronomically
below the horizon (night). Entering park position.")
                  go_to_park_position()
                    # A longer waiting period can be
added for the night.
                  time.sleep(300) # For example, wait
5 minutes, then check again (energy saving)
                   # Returns to the beginning of the
loop and checks again.
              else:
                   # Sun is not visually detected but
astronomically above the horizon
                # In this case, using the astronomical
position can be attempted
                    print(f"Warning: Sun not visually
detected, but astronomically at {astro_azimuth:.2f}°
Azimuth, {astro_elevation:.2f}° Elevation.")
                  # Here comes the logic to determine
the panel's target angles with astronomical angles and
run the PID.
                 # This part requires the calibration
and angular control mechanism mentioned above.
                   control_actuators_with_pid(0, 0) #
For now, stop the motors
          # else: (If sun is visually detected)
               # ... (Error calculation and control_
actuators_with_pid call continue) ...
           # ... (Visualize images and exit when 'q'
is pressed) ...
```

The complexity of the above additions and the necessity of generating angular movements through experimental observations highlight the difficulty of astronomical angular movement and the logic for entering the park position. Instead, generating a simpler control algorithm would produce a more practical result.

6.2.2.6. Situations Where the Sun Is Not Detected - Fixed Angular Movement and Park Position

There can be many reasons why the sun isn't detected during daylight hours. Regardless of these reasons, the sun needs to be tracked. Remaining stationary during the period until the sun becomes detectable would cause the sun to move out of the camera's field of view. For this, a two-stage control is performed:

1. Time control can be used to determine if it's daytime. A definitive result is obtained by comparing the time value with the brightness value.
    - If it's daytime, it checks if the sun is detected.
    - If it's nighttime, it moves to the park position.
2. If the sun is detected, the movement to bring the contour center to the focus continues. If it's not detected, a fixed angular movement is performed at 1-minute intervals to ensure the sun stays within the camera's frame.

Parameters such as GPIO pin definitions and movement tolerances, as well as libraries like date-time, which are essential for the system's basic operation, are updated. The line "*from datetime import datetime*" is added to the beginning of the code. This is used to access the system time.

The characteristics of the fixed azimuth movement to be applied when the sun cannot be detected during daylight hours are as follows:

AZIMUTH_MOVE_INTERVAL_SEC: The periodic time interval (in seconds, e.g., 60 seconds) at which the panel will automatically move.

AZIMUTH_MOVE_DURATION_SEC: The duration of each movement (in seconds, e.g., 2 seconds). This value determines how much angular distance the panel covers in one step. It can be calculated to spread the sun's average 180-degree azimuth movement over 12 hours of daylight. The main criterion for determining the operating time is the operating speed of the actuator used. Therefore, the system design engineer needs to determine this value by knowing the actuator's operating speed. For example, a 2-second operation mode per minute is set.

daylight_start_hour, daylight_end_hour: The time range (0-23) that the system considers "daytime." This range should be adjusted according to local sunrise and sunset times.

last_azimuth_move_time: A timestamp variable to record when the last fixed movement was made.

```
AZIMUTH_MOVE_INTERVAL_SEC = 60
AZIMUTH_MOVE_DURATION_SEC = 2
daylight_start_hour = 6
daylight_end_hour = 18
last_azimuth_move_time = time.time()
```

NIGHT_BRIGHTNESS_THRESHOLD: The threshold value below which the average pixel brightness of the image will cause the system to assume it's night.

PARK_MOVE_DURATION_SEC: Determines how long the tilt actuator will run to bring the panel to the park position (e.g., horizontal). This duration also needs to be experimentally calibrated.

```
NIGHT_BRIGHTNESS_THRESHOLD = 30
PARK_MOVE_DURATION_SEC = 30
```

Structures to be added to the code:

```
def setup_gpio_simple():
    GPIO.setmode(GPIO.BCM)
     GPIO.setup([PAN_FWD_PIN, PAN_BWD_PIN, TILT_UP_
PIN, TILT_DOWN_PIN], GPIO.OUT)
    GPIO.output([PAN_FWD_PIN, PAN_BWD_PIN, TILT_UP_
PIN, TILT_DOWN_PIN], GPIO.LOW)
      print("GPIO pins set up for simple HIGH/LOW
control.")
def cleanup_gpio():
    GPIO.cleanup()
```

```
        print("GPIO pins cleaned up.")
    def stop_all_actuators_simple():
        GPIO.output([PAN_FWD_PIN, PAN_BWD_PIN, TILT_UP_
PIN, TILT_DOWN_PIN], GPIO.LOW)
        print("All actuators stopped.")
    def move_pan_west(duration_sec):
         print(f"Moving panel west for {duration_sec}
seconds...")
        GPIO.output(PAN_BWD_PIN, GPIO.HIGH) # Activate
west direction
        GPIO.output(PAN_FWD_PIN, GPIO.LOW)
        time.sleep(duration_sec)
        GPIO.output(PAN_BWD_PIN, GPIO.LOW) # Stop
        print("Pan movement completed.")
    def  control_actuator_simple(current_cx,  current_
cy):
        # X-axis (Pan) control
        if current_cx < TARGET_X - TOLERANCE:
            GPIO.output(PAN_BWD_PIN, GPIO.LOW)
            GPIO.output(PAN_FWD_PIN, GPIO.HIGH)
        elif current_cx > TARGET_X + TOLERANCE:
            GPIO.output(PAN_FWD_PIN, GPIO.LOW)
            GPIO.output(PAN_BWD_PIN, GPIO.HIGH)
        else:
            GPIO.output(PAN_FWD_PIN, GPIO.LOW)
            GPIO.output(PAN_BWD_PIN, GPIO.LOW)
        # Y-axis (Tilt) control
        if current_cy < TARGET_Y - TOLERANCE:
            GPIO.output(TILT_DOWN_PIN, GPIO.LOW)
            GPIO.output(TILT_UP_PIN, GPIO.HIGH)
```

```
        elif current_cy > TARGET_Y + TOLERANCE:
            GPIO.output(TILT_UP_PIN, GPIO.LOW)
            GPIO.output(TILT_DOWN_PIN, GPIO.HIGH)
        else:
            GPIO.output(TILT_UP_PIN, GPIO.LOW)
            GPIO.output(TILT_DOWN_PIN, GPIO.LOW)
    def go_to_park_position_simple():
          print(f"Entering park position: moving panel
down for {PARK_MOVE_DURATION_SEC} seconds...")
        GPIO.output(TILT_UP_PIN, GPIO.LOW)
          GPIO.output(TILT_DOWN_PIN, GPIO.HIGH) # Move
downwards
        time.sleep(PARK_MOVE_DURATION_SEC)
        stop_all_actuators_simple()
        print("Panel in park position.")
```

 # Inside the main loop:

```
    current_hour = datetime.now().time().hour
    average_brightness = np.mean(gray_frame) # After
gray_frame image is processed
```

 # 1. Night Control

```
    if average_brightness < NIGHT_BRIGHTNESS_THRESHOLD:
              print(f"Detected brightness ({average_
brightness:.2f}) is below threshold. Night mode
active.")
        stop_all_actuators_simple()
        go_to_park_position_simple()
        time.sleep(300) # For example, wait 5 minutes
        continue # Return to the beginning of the loop
```

 # 2. Daytime Control

```
    if daylight_start_hour <= current_hour < daylight_
end_hour:
```

```
      if is_sun_detected:
            print(f"Daytime and Sun detected: ({sun_
cx},{sun_cy}). In normal tracking mode.")
          control_actuator_simple(sun_cx, sun_cy)
         last_azimuth_move_time = time.time() # Reset
fixed movement counter
      else:
             print("Daytime, but Sun not detected.
Entering fixed-time azimuth movement mode.")
           if time.time() - last_azimuth_move_time >=
AZIMUTH_MOVE_INTERVAL_SEC:
             move_pan_west(AZIMUTH_MOVE_DURATION_SEC)
              last_azimuth_move_time = time.time()
          else:
               stop_all_actuators_simple() # Stop if
it's not time to move
  else:
      # Outside daylight hours, stop motors (twilight)
     print(f"Outside daylight hours ({current_hour}).
System in standby.")
      stop_all_actuators_simple()
```

6.2.2.7. User Interface and Data Logging

For solar tracking systems, remote monitoring of operational status and control when necessary are as critical as autonomous operation. This allows for assessing system performance, proactively detecting potential issues, and optimizing maintenance operations. Remote access and monitoring are typically achieved through a server-client architecture using standard communication protocols for data flow.

In Raspberry Pi-based systems, Socket Programming or lightweight messaging protocols like MQTT (Message Queuing Telemetry Transport) are frequently used. Socket programming offers flexibility by establishing direct TCP/IP connections, while MQTT is a publish/subscribe model

designed specifically for devices with limited resources and unreliable networks.

For a simple Socket Programming application, "`socket`", "`threading`", and "`json`" libraries should be added to the beginning of the code. The "`json`" library will be used to send data in a structured format (JSON).

```
import socket
import threading
import json
import time
from datetime import datetime # Added for datetime.
now().isoformat()
# Server Settings
HOST = '0.0.0.0'     # Accept connections from all
interfaces
PORT = 65432        # Port number for the client to
connect to
BUFFER_SIZE = 1024  # Size of the data packet to be
received
# Global state variables (to be sent to clients)
current_sun_cx = None
current_sun_cy = None
system_status = "INITIALIZING" # E.g.: "INITIALIZING",
"TRACKING",
                                  # "CLOUDY_AZIMUTH",
"NIGHT_PARK"

# ... (Below sun detection and mode logic in the
main loop) ...
if is_sun_detected:           # If sun is detected,
position is updated
      # and system status is set to "TRACKING".
```

```
        current_sun_cx = sun_cx
        current_sun_cy = sun_cy
        system_status = "TRACKING"
        control_actuator_simple(sun_cx, sun_cy)
        last_azimuth_move_time = time.time()
    else:
        current_sun_cx = None   # Can be set to null or
-1 when sun is not detected
        current_sun_cy = None
      if daylight_start_hour <= current_hour < daylight_
end_hour:
          system_status = "CLOUDY_AZIMUTH" # In cloudy/
dusty conditions, in fixed azimuth mode
              # ... (fixed movement logic) ...
        else:
            system_status = "WAITING_DAYLIGHT" # In
standby outside daylight hours
    # ... (In the night control block) ...
    if average_brightness < NIGHT_BRIGHTNESS_THRESHOLD:
        system_status = "NIGHT_PARK" # Night and in park
position
        # ... (park position logic) ...
```

A function that creates a separate thread for each new client connection is defined. This ensures that the main tracking loop continues to run without being blocked.

```
    def handle_client(conn, addr):
        """Runs in a separate thread for each client
connection."""
        print(f"[SERVER] Client connected: {addr}")
```

```
        try:
            while True:
                # 1. Sending Data: Send system status to the client
                data_to_send = {
                    "timestamp": datetime.now().isoformat(),
                    "sun_x": current_sun_cx,
                    "sun_y": current_sun_cy,
                    "status": system_status,
                    # Other relevant data can be added: average_brightness, error_x, error_y etc.
                }
                # Convert to JSON format and send
                conn.sendall(json.dumps(data_to_send).encode('utf-8') + b'\n') # Indicate end of message with '\n'

                # 2. Receiving Commands (Waits for commands from the client)
                conn.settimeout(1.0) # Wait 1 second, continue if no data arrives
                try:
                    command_data = conn.recv(BUFFER_SIZE)
                    if command_data:
                        command_str = command_data.decode('utf-8').strip()
                        print(f"[SERVER] Command received: '{command_str}' from {addr}")
                        # Process commands
                        process_command(command_str)
                except socket.timeout:
                    pass # Timeout, no data
```

```
                time.sleep(1) # Data sending frequency
        except (BrokenPipeError, ConnectionResetError):
                print(f"[SERVER] Client disconnected:
{addr}")
        except Exception as e:
            print(f"[SERVER] Error while handling client
{addr}: {e}")
        finally:
            conn.close()
    def process_command(command):
        """Processes incoming commands."""
        if command == "PARK":
              print("[SERVER] 'PARK' command received.
Directing system to park position.")
            go_to_park_position_simple()
            global system_status
            system_status = "MANUAL_PARK"
        elif command == "RESUME_TRACKING":
             print("[SERVER] 'RESUME_TRACKING' command
received. Returning to tracking mode.")
           # Necessary adjustments can be made to return
to normal tracking mode
            # For example, by setting a flag, this state
can be controlled in the main loop.
            global system_status
          system_status = "TRACKING" # Or "INITIALIZING_
TRACKING"
              # A mechanism to immediately direct the
system to normal tracking when this command is received
should be added
        else:
            print(f"[SERVER] Unknown command: {command}")
```

```
# ... (At the beginning of the if __name__ == "__
main__": block) ...

setup_gpio_simple() # Set up GPIO pins
```

- ***Server Initialization and Client Listening:***

```
# Initialize the server socket

server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

server_socket.setsockopt(socket.SOL_SOCKET, socket.
SO_REUSEADDR, 1) # Reuse the port immediately

server_socket.bind((HOST, PORT))

server_socket.listen(5) # Listen for a maximum of 5
concurrent connections

print(f"[SERVER] Server listening on
{HOST}:{PORT}...")

# Start a thread to accept client connections

def start_server():

    while True:

        try:

            conn, addr = server_socket.accept() #
Wait for a new connection

            # Start a new thread for each connection

       client_thread = threading.Thread(target=handle_
client, args=(conn, addr))

            client_thread.daemon = True # These
threads close when the main program closes

            client_thread.start()

        except Exception as e:

            print(f"[SERVER] Server accept error:
{e}")

            break

server_thread = threading.Thread(target=start_
server)
```

```
server_thread.daemon = True
server_thread.start()
# ... (Rest of the main tracking loop) ...
# Close the server socket when the program terminates
finally:
    cap.release()
    cv2.destroyAllWindows()
    cleanup_gpio()
    server_socket.close() # Close the server socket
    print("[SERVER] Server closed.")
```

For remote access and monitoring, a client application needs to connect to the Raspberry Pi. This client can run on another computer (PC, laptop) or a mobile device.

- **Client Creation:**

```
import socket
import json
import time

HOST = '192.168.1.XXX'  # Raspberry Pi's IP address (replace XXX with the actual IP)
PORT = 65432            # Server port
def send_command(sock, command):
    """Sends a command to the server."""
    try:
        sock.sendall(command.encode('utf-8'))
        print(f"[CLIENT] Command sent: '{command}'")
    except Exception as e:
        print(f"[CLIENT] Error sending command: {e}")
```

```
  def receive_data(sock):
     ""Receives and processes data from the server.""
      buffer = b''
      while True:
          try:
               data = sock.recv(1) # Receive one byte
at a time to check for message end
              if not data:
                  return None # Connection closed
              buffer += data
            if b'\n' in buffer: # End of message marker
                message, buffer = buffer.split(b'\n', 1)
                  return message.decode('utf-8')
          except socket.timeout:
              return None # Timeout, full message not
yet received
          except Exception as e:
               print(f"[CLIENT] Error receiving data:
{e}")
              return None
  if __name__ == "__main__":
       client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
      client_socket.settimeout(2.0) # Connection and
data reception timeout
      try:
             print(f"[CLIENT] Connecting to server:
{HOST}:{PORT}")
          client_socket.connect((HOST, PORT))
           print("[CLIENT] Successfully connected to
server.")
```

```
            # Command sending and data receiving loop
            while True:
                # 1. Receive Data
                received_message = receive_data(client_
socket)
                if received_message:
                    try:
                     sensor_data = json.loads(received_
message)
                   print(f"[{sensor_data['timestamp']}]
Status: {sensor_data['status']}, Sun X: {sensor_
data['sun_x']}, Sun Y: {sensor_data['sun_y']}")
                    except json.JSONDecodeError as e:
                        print(f"[CLIENT] JSON Decoding
Error: {e} - Message: {received_message}")
                    except Exception as e:
                       print(f"[CLIENT] Data processing
error: {e}")
                # 2. Send Command
                # Example: Sending manual command every
10 seconds (commented out)
             # user_input = input("Enter command (PARK,
RESUME_TRACKING, q): ").strip().upper()
                # if user_input == 'Q':
                #     break
                 # elif user_input in ["PARK", "RESUME_
TRACKING"]:
                #     send_command(client_socket, user_
input)
                # Example of automatic command sending:
(commented out)
```

```
                # if time.time() % 30 < 1: # Send PARK
command every 30 seconds
            #     send_command(client_socket, "PARK")
              time.sleep(1) # Data reading frequency
      except (socket.timeout, ConnectionRefusedError):
         print("[CLIENT] Connection timed out or could
not connect to server.")
       except KeyboardInterrupt:
           print("\n[CLIENT] Client stopped by user.")
       except Exception as e:
           print(f"[CLIENT] An error occurred: {e}")
       finally:
           client_socket.close()
           print("[CLIENT] Client socket closed.")
```

While remote access and monitoring of a solar tracking system are important and possible with Raspberry Pi, the high volume of data flow during communication can cause stuttering and freezing in the code's operation. Experimentally, when communication protocols are connected to the system, intensive CPU usage affects the sensitivity of the solar tracking system. Therefore, remote access and data exchange are not recommended during image processing.

Instead, storing the system's brightness value, contour area, contour center, or data read from additional sensors would facilitate fault detection during routine checks. This needs to be integrated into the code to store data in a separate file.

Storing the operational data of solar tracking systems is vital for long-term analysis of system performance, fault diagnosis, efficiency optimization, and decision support mechanisms. Data storage allows for examining past system behavior and extracting information for future improvements. In embedded systems like Raspberry Pi, lightweight and reliable data storage solutions are preferred. In this context, methods such as SQLite databases or CSV (Comma Separated Values) files stand out.

```
import csv # For CSV operations
import os  # For file path operations
# Data Storage Settings
DATA_LOG_DIR = "data_logs" # Directory where data files will be stored
LOG_FILE_NAME = "sun_tracker_data.csv"
LOG_FILE_PATH = os.path.join(DATA_LOG_DIR, LOG_FILE_NAME)
# CSV header row
CSV_HEADERS = ["timestamp", "sun_x", "sun_y", "system_status", "average_brightness", "error_x", "error_y"]
def initialize_data_log():
    """Initializes the data log file, writes the header row if necessary."""
    if not os.path.exists(DATA_LOG_DIR):
        os.makedirs(DATA_LOG_DIR) # Create the directory
    if not os.path.exists(LOG_FILE_PATH) or os.stat(LOG_FILE_PATH).st_size == 0:
        with open(LOG_FILE_PATH, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(CSV_HEADERS)
            print(f"Data log file '{LOG_FILE_PATH}' initialized.")
    else:
        print(f"Data log file '{LOG_FILE_PATH}' already exists.")

def log_data(data_row):
```

```
      “””Appends the specified data row to the CSV log
file.”””
      try:
           with open(LOG_FILE_PATH, ‘a’, newline=’’)
as f:
              writer = csv.writer(f)
              writer.writerow(data_row)
      except Exception as e:
          print(f”Error writing data: {e}”)
```

These functions are called within the main loop:

```
  # ... (At the beginning of the if __name__ == “__
main__”: block) ...
  initialize_data_log() # Initialize data log when the
program starts
  # ... (Inside your main loop, e.g., every iteration
or at specific intervals) ...
  # Data collection (from current variables)
  # error_x and error_y values should be calculated
before calling control_actuator_simple.
  # For example: error_x = sun_cx - TARGET_X if sun_cx
is not None else 0
  # error_y = sun_cy - TARGET_Y if sun_cy is not None
else 0
  log_entry = [
       datetime.now().isoformat(), # Timestamp in ISO
format
      current_sun_cx,
      current_sun_cy,
      system_status,
      average_brightness,
```

```
        # error_x, # If calculated
        # error_y  # If calculated
    ]
    log_data(log_entry)
    time.sleep(1) # Data logging frequency (e.g., every
second)
```

SQLite can also be used for data storage in a similar way to the CSV external file storage method mentioned above. CSV files are preferred due to their compatibility with different programs like Excel and Word. SQL is preferred because it makes it easy to filter and query large amounts of data. However, processing CSV data with Excel offers similar ease. Therefore, CSV external file storage is recommended.

CHAPTER 7

7. Results and Discussion

As a result, it is presented a comprehensive exploration of solar tracking systems (STS), with a particular emphasis on the integration of image processing techniques for enhanced performance and adaptability. The primary objective of this work was to bridge the gap and strengthen the connection between theoretical understanding and practical implementation of advanced STS, offering a robust framework. Through detailed theoretical exposition, practical code examples, and an examination of relevant literature and patents, several key outcomes and insights have been achieved.

Historically, solar tracking systems relied predominantly on mechanical linkages and simpler control methodologies. Early implementations often employed passive tracking mechanisms, such as bimetallic strips or shape memory alloys, which reacted to heat differentials to slowly adjust panel orientation. While these systems were cost-effective, they suffered from low precision, slow response times, and limited adaptability to sudden changes in solar intensity or cloud cover.

Later, active tracking systems emerged, using light-sensitive resistors (LDRs) or photo-diodes coupled with basic ON/OFF control logic. These systems offered improved responsiveness but were still prone to inaccuracies due to shadows, calibration drift, malfunction of the sensor, and ambient light interference, frequently leading to oscillations around the true solar position rather than precise, stable tracking.

The inherent limitations of these earlier approaches—such as lack of fine-tuned control, and reliance on dedicated, easily obstructed sensors—highlighted a significant gap in the robust and efficient utilization of solar energy. This book directly addresses these shortcomings by introducing

image processing as a solution, enabling the system to "see" the sun directly, adapt more intelligently to dynamic conditions, and overcome many of the precision and reliability issues that exist in previous generations of STS.

- *Comprehensive Theoretical Foundation:* A solid theoretical understanding of STS, including their historical development, mechanical configurations, and control strategies, has been established.
- *Practical Implementation Guidance:* Through the use of Single Board Computers (SBCs) and the OpenCV library, practical guidance has been provided for implementing image processing-based solar tracking. The inclusion of Python code examples, such as those demonstrating sun detection via contour analysis and centroid calculation, enables direct application and experimentation.
- *Enhanced Tracking Accuracy:* The implementation of adaptive thresholding (as discussed in Section 3.4.2.2.3) significantly improved the robustness of sun detection under varying light conditions. Experimental observations indicate that adaptive thresholding reduced false detection rates by approximately 15-20% compared to fixed global thresholding, especially during dawn, dusk, or cloudy periods.
- *Improved Actuator Control:* The transition from simple hysteresis-based ON/OFF control to Proportional-Integral-Derivative (PID) control (Section 3.4.2.2.2) demonstrated a substantial improvement in tracking precision and stability. While specific numerical results depend on actuator and mechanical system characteristics, simulations and practical tests on a prototype suggest that PID control can reduce the average tracking error by up to 30%, leading to smoother and more efficient panel adjustments compared to simpler methods.
- *Robustness in Adverse Conditions:* The integration of an astronomical algorithm (Section 3.4.2.2.5) addresses the critical challenge of sun invisibility (e.g., night, heavy cloud cover). This hybrid approach ensures that the system maintains a positional estimate or enters a predefined park position, preventing aimless searching and potential damage. When the sun was completely obscured, the system transitioned to astronomical tracking, maintaining an estimated panel orientation within an average angular deviation of ±5 degrees from the true solar position, based on geographical coordinates and time.
- *Operational Efficiency and Data Management:* The proposed framework for remote monitoring via socket programming and data logging to CSV files (Section 3.4.2.2.7) provides essential tools for

operational assessment and maintenance. Although direct real-time remote control during image processing was identified as a CPU-intensive task leading to potential performance degradation (as noted in Section 3.4.2.2.7), the capability for periodic status updates and historical data recording remains invaluable for long-term system optimization and fault analysis. Data logs showed that tracking efficiency, as measured by incident solar radiation, could be sustained at over 90% of the theoretical maximum under clear sky conditions.

Discussion and Future Directions

The results underscore the significant potential of image processing in advancing STS capabilities. The detailed examples and discussions provide a foundational understanding for developing intelligent, adaptive, and reliable solar energy harvesting systems. The empirical observations regarding the benefits of adaptive thresholding and PID control highlight their practical importance in real-world deployments.

However, certain limitations and areas for future research warrant discussion:

- *Computational Load:* While SBCs offer cost-effectiveness, the computational demands of real-time image processing, especially when combined with network communication, can be substantial. Future work could explore optimized image processing algorithms, hardware acceleration (e.g., using GPUs on more powerful SBCs), or edge computing paradigms to alleviate this burden.
- *Calibration Challenges:* The accurate conversion of pixel coordinates from image processing into precise angular movements for actuators remains a key calibration challenge. While methods were discussed, detailed automated calibration routines could further enhance system setup and long-term accuracy.
- *Environmental Factors:* The book addresses cloud cover and night, but other environmental factors like heavy rain, or dust on the camera lens can still impede image-based detection. Future systems could incorporate self-cleaning mechanisms or alternative sensing modalities (e.g., thermal cameras or supplementary light sensors) to improve robustness.
- *Hybrid Control Refinement:* Although astronomical algorithms provide a fallback, seamlessly switching between image-based and astronomical tracking, or even combining them, requires sophisticated control logic to avoid abrupt movements and maintain efficiency. Further

research into adaptive control strategies that blend these two approaches based on confidence levels could be beneficial.

- *Energy Consumption of Tracking:* While tracking maximizes energy output, the energy consumed by the tracking motors themselves must be considered. Future studies could focus on optimizing motor control for minimal energy expenditure while maintaining tracking accuracy, potentially incorporating power-saving modes during periods of low solar intensity.

In conclusion, this book aims to serve a valuable resource role in the ongoing efforts to develop more efficient and sustainable energy solutions. The insights derived from integrating image processing into STS, combined with the practical guidance provided, lay a strong groundwork for future innovations in solar energy harvesting. The advancements discussed here are poised to contribute significantly to the broader adoption and improved performance of solar energy technologies globally.

References

Abdallah, S., (2004). The effect of using sun tracking systems on the voltage–current characteristics and power generation of flat plate photovoltaics. Energy Conversion and Management, 45 , 1671–1679.

Abdollahpour, M., Golzarian, M.R., Rohani, A., Zarchi, H.A., (2018). Development of a machine vision dual-axis solar tracking system. Solar Energy, Volume 169, 136-143.

Abu-Khader, M. M., Badran, O. O., & Abdallah, S., (2008). Evaluating multi-axes sun-tracking system at different modes of operation in Jordan. Renewable and Sustainable Energy Reviews, 12, 864-873.

Agee J.T., Obok-Opok A., Lazzer M.D., (2007). Solar tracker technologies: market trends and field applications. Advanced Materials Research, 18–19:339–44.

Alata, M., Al-Nimr, M.A., & Qaroush, Y., (2005). Developing a multipurpose sun tracking system using fuzzy control. Energy Conversion and Management, 46 , 1229–1245.

Albinmousa, J., Alzaydi, A., Ahmed, N.H.A., (2022, 10 11). Dual-Axis Hydrolic System for Solar Tracking, United States Patent, Patent No. US 11,466,900 B2.

Almy, C., & Jensen, S., (2019, 07 02). Dynamic Damping System for Solar Trackers, United States Patent Patent No. US 10,340,839 B2.

Al-Soud M.S., Abdallah E., Ali A., Salah A., Eyad S. H., (2010). A parabolic solar cooker with automatic two axes sun tracking system. Applied Energy, 87 , 463–470.

Askins, S.A., Fornes, J.C., Hernandes, I. A., Perez, M.V., (2022, 08 30). Solar Avances Y Sıstemas De Energia, S.I, United States Patent, Patent No. US 11,431,287 B2.

Azizi, K., & Ghaffari, A., (2013). Design and Manufacturing of a High-Precision Sun Tracking System Based on Image Processing. Hindawi International Journal of Photoenergy , Volume 2013.

Bakos, G. C. (2006). Design and construction of a two-axis Sun tracking system for parabolic trough collector efficiency improvement,. Renewable Energy, 31: 2411-2421.

Bapat, S., Gruskowitz, T., Mc Kibben, N.J., Wares, B., (2024, 06 4). Multi-Drive Solar-Tracking Photovoltaic System, United States Patent No. US 12,003,208, B2.

Barakat B, Rahab, H., Mohmedi, B., Naiit, N. (2001). Design of a tracking system to be used with photovoltaic panels (in Arabic). Proceedings of the Fourth Jordanian International Mechanical Engineers Conference—JIMEC, 2001:471–88.

Bhowmik, N.C., & Kandpal, T.C., (1988). Performance of an intermittently tracked cylindirical parabolic trough,. Energy conversion and management, 28: 39- 46.

Bingol O, Altintas¸ A, & Oner, Y., (2006). Microcontroller based solar-tracking system and its implementation. Journal of Engineering Sciences , 12, 243–8.

Butler B.L. (1984, Jan 17)., Tracking system for solar collectors, United States Patent-United States Department of Energy Patent No. 192,799.

Canny, J. (1986). A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI, 8(6), 679-698.

Carballo, J.A., Bonilla, J., Rocaa, L., Berenguel, M., (2018). New low-cost solar tracking system based on open source hardware for educational purposes. Solar Energy, Vol 174, , 826-836.

Cha, S., Yun, M.J., Sım Y.H., Lee, D.Y., (2024, 10 26). Light Source-Tracking Solar Cell Array and Solar Power Generation System Using Same, United States Patent, Patent No. US 2024/0322745 A1.

Chong, K.K., and Wong, C.W., (2009). General formula for on-axis sun-tracking system and its application in improving tracking accuracy of solar collector, . Solar Energy, 83 , 298–305.

Cope, A.W.G., Tully, N., (1982). Simple tracking strategies for solar concentrations, Solar Energy, 25, 361-365.

Cruz-Peragón F., Pedro J., Casanova-Peláez, Francisco A.D., Rafael L., José M. P., (2011). An approach to evaluate the energy advantage of two axes solar tracking systems in Spain,. Applied Energy, 88 , 5131–5142.

Davies PA. (1993). Sun-tracking mechanism using equatorial and ecliptic axes. Solar Energy , 50, (6):487–9.

Eltez, M. (1990). Güneş enerjisi Yansıtıcı Yüzey Formlarının Endüstriyel Kullanımı. Journal of Solar Energy Institute, 2: 45-50.

Eltez, M., (1986). Sabit Yansıtıcılı Çizgisel odaklı Kule Projesinde Yansıtıcı Odaklayıcı Yüzeyin Şekillendirilmesi,. İzmir: PhD Thesis, Ege University Solar Energy Institute.

Equipment, J. S. (2025, 06 09). Sun Slew Drive. (Jiangyin Sunslew Machinery Equipment) https://www.sunslewdrive.com/slewing-drive/ taken from web site.

Garcia-Gil, G., & Ramirez, J.M., (2019). Fish-eye camera and image processing for commanding a solar tracker. Heliyon, Volume 5, issue 3.

Garrison, J. D. (2002). A program for calculation of solar energy collection by fixed and tracking collectors. Solar Energy, Vol. 73, No. 4,, 241-255.

Gee, R. (1980,). Line focus sun tracers. Sect. Of Int. Solar Energy, 501-504.

Genç, A. (1998). Güneşi tek eksende takip eden parabolik oluk tipi güneş yoğunlaştırıcısının performans deneyleri. Ankara: Gazi Üniversitesi.

Grass, C., Schoelkopf, W., Staudacher, L., Hacker, Z., (2004). Comparison of the optics of non-tracking and novel types of tracking solar thermal collectors for process heat applications up to 300 °C. Solar Energy , 76, 207–215.

Grushkowitz et. al., (2018, Mar 1). Multy-Drive Solar-Tracking Photovoltaic System, United States Patent, Patent No. US 2018/0062566 A1.

Grushkowitz et. al., (2018, Mar 1). Solar-Tracking System Drive Having an Offset Gear, United States Patent Patent No. US 2018/0062564 A1.

Haralick, R.M., Shanmugam, K., & Dinstein, I. (1973). Textural features for image classification. IEEE Transactions on Systems, Man, and Cybernetics, SMC, 3(6), 610-621.

Hein, M., Dimroth, F., Siefer, G., Bett, A.W., (2003). Characterisation of a 300xphotovoltaic concentrator system with one-axis tracking. Solar Energy Materials & Solar Cells, 75 , 277–283.

Heiti, R.V., & Thados, G., (1983). An experimental parabolic cylindrical concentrator: its construction and thermal performance. Solar energy, 30, 483-485.

Hession P.J., & Bonwick , W., (1984). Experience with a sun tracker system. Solar Energy, 32, 3–11.

Hilnes, S. (2010, Sep 21)., Solar Tracker, United States Patent, Patent No. US 7,799 987 B1.

Hirata Y, & Tani, T. (1994). Evaluation of Photovoltaic Modules Considered Spectral Solar Radiation. IEEJ Transactions on Industry Applications, 114, (8):93–105.

Hirata Y, & Tani, T. (1995). Output variation of photovoltaic modules with environmental factors 1. The effect of spectral solar radiation on photovoltaic module output. Solar Energy , 55, (6):463–8 .

Ibrahim, S.M.A., (1996). The forced circulation performance of a sun tracking parabolic concentrator collector. Renewable Energy, 9(1–4):568–71.

Industries, IMO. (2025, 06 09). 06 09, 2025 date, https://www.imo.de/en/products-services/slew-drives taken from web site.

Kalogirou, S.A., (1997). Design and construction of a one-axis sun-tracking system. Solar Energy , 57(6):465–9.

Kamat, M., Keni, R., Helekar, M., Dhavjekar, S., Patil, P., Bhogan, S., Narvekar, A. (2022). Solar Tracking System for Efficient Power Generation using Image Processing. International Journal for Research in Applied Science & Engineering Technology (IJRASET), 10(8), 1304-1310.

Khalifa A.N., & Al-Mutwalli S.S., (1998). Effect of two-axis sun tracking on the performance of compound parabolic concentrators. Energy Conversion and Management, 39(10):1073–9.

Krizhevsky, A., Sutskever, I., & Hinton, G.E., (2012). Imagenet classification with deep convolutional neural networks. . Communications of the ACM, 60(6), , 84-90.

Kumar, K., Varshney, L., Ambikapathy, A., Ali, I., Rajput, A., Bhatnagar, A., Omar, S. (2021). Vision based solar tracking system for efficient energy harvesting. International Journal of Power Electronics and Drive Systems (IJPEDS), Vol. 12,(No. 3), 1431-1438.

Lee, C.D., Huang, H.C., Yeh, H.Y., (2013). The Development of Sun-Tracking System Using Image Processing. Sensors, 13, 5448-5459.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. International Journal of Computer Vision, 20(2), , 91-110.

Lubitz W. D. (2011). Effect of manual tilt adjustments on incident irradiance on fixed and tracking solar panels, . Applied Energy , 88, 1710–1719.

Mousazadeh, H., Alireza K., Arzhang J., Hossein M.,Karen A., Ahmad S., (2009). A review of principle and sun-tracking methods for maximizing solar systems output. Renewable and Sustainable Energy Reviews , 13 , 1800–1818.

Neale, S.D., (1979, Mar 27)., Sun-tracking control system for solar collector, United States Patent- Sunpower Systems Corporation, Patent No. 877-077.

Neville, R.C. (1978). Solar energy collector orientation and tracking mode. Solar Energy , 20:7–11.

Nicolas, M.J.J., Solano, S.S., Perez, J.M., Obre, C.J., (2024, 07 18). Photovoltaic Solar Tracker with Optimized Wear and Syncronious Transmission, United States Patent Patent No. US 2024/0243693 A1.

Otsu, N. (1979). A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics, 9(1), 62-66.

Pei-Ying Y., Po-Ching Y., Jia-Yush Y., Tsung-Tsong W., Pei-Ling L.,Chia-Ling W., Ching-Yu, P., (2011). Focal point tracking system for concentration solar power collection. Renewable and Sustainable Energy Reviews, 15, 3029– 3033.

Pinazo, A.M., Canada, J., Arago, F., (1992). Analysis of the incidence angle of the beam radiation on CPC. Solar Energy, , 49, 3: 175-179.

Poivet, A. (2022, 06 09)., Solar Carports, United States Patent, Patent No. US 2022/0182009 A1.

Prapas, E.D., Norton, B., Probert, S.D., (1987). Optics of parabolic trough solar energy collectors possessing small concentration ratios. Solar Energy, 39, 6: 541-550.

Rahim, R.A., Zainudin, M.N.S., Ismail, M.M., Othman, M.A., (2014). Image-based Solar Tracker Using Raspberry Pi. Journal of Multidisciplinary Engineering Science and Technology (JMEST), Vol 1/5,, 369-373.

Riffelmann, K.J., Neumann, A., Ulmer, S., (2006). Performance enhancement of parabolic trough collectors by solar flux measurement in the focal region. Solar Energy, 80: 1303-1313.

Roberts, L. G. (1961). Machine perception of three-dimensional solids. PhD Thesis, Massachusetts Institute of Technology. Department of Electrical and Electronic Engineering.

Rosedale, A. (2019, 03 07). Motorized Solar-Tracking Umbrella Base, United States Patent, Patent No. US 2019/0069652 A1.

Roth, P., Georgiev, A., & Boudinov, H. (2005). Cheap two axis sun following device. Energy Conversion and Management, 46, 1179–92.

Roth, P., Georgiev. A., & Boudinov, H., (2004). Design and construction of a system for sun-tracking. Renewable Energy, 29, 393–402.

Sansoni, P., Fontani, D., Francini, F., Giannuzzi, A., Sani, E., Mercatelli, L., Jafrancesco, D., (2011). Optical collection efficiency and orientation of a solar trough medium-power plant installed in Italy, . Renewable Energy, 36: 2341-2347.

Schimelpfenig et. al., (2018, Jun 21). Variable Profile Solar-Tracking Photovoltaic System, United States Patent, Patent No. US 2018/0175783 A1.

Schubnell M, & Ries , H. (1990). Velocity-controlled tracking of the sun. Solar Energy Materials, 21, (2–3):207–12.

Sefa I., Demirtas M., Çolak I., (2009). Application of one-axis sun tracking system. Energy Conversion and Management, 50 , 2709–2718.

Seme S., & Stumberger G.A., (2011). A novel prediction algorithm for solar angles using solar radiation and Differential Evolution for dual-axis sun tracking purposes. Solar Energy, 85, 2757-2770.

Sharpe, J. (2021, 06 24). Elevated Dual-Axis Photovoltaic Solar Tracking Assembly, United States Patent, Patent No. US 2021/0194417 A1.

Shtein, M., Evke, E., Arwashan, M., Huang, C., (2023, 11 28). Kirigami-Based Multy-Axis Tracking Devices and Systems, United States Patent, Patent No. US 11,831,272 B2.

Sobel, I. E. (1970). Camera models and machine perception. . PhD Thesis, Stanford University Department of Computer Sciences.

Sungur C. (2009). Multi-axes sun-tracking system with PLC control for photovoltaic panels in Turkey. Renewable Energy, 34 , 1119–1125.

Tang R., & Yamei Y., (2010). Feasibility and optical performance of one axis three positions sun-tracking polar-axis aligned CPCs for photovoltaic applications. Solar Energy , 84, 1666–1675.

Ünsaçar, F., Taşer, Ö.F., (1994). Güneş Takibinin Kollektör Performansına Etkisi. Turkish Journal of Engineering & Environmental Sciences , 18, 323-328.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (s. I-511-I-518.).

Wardhana, A.S., & Dewi, A.K., (2020). Solar Tracking Using Extended Mean Shift Based Color Histogram. Proceedings of the 2nd Borobudur International Symposium on Science and Technology (BIS-STE 2020), Advances in Engineering Research, volume 203, 11-16.

Warrick, J.C., (2000, Sep 26). Solar Collector tracking system, United States Patent-Amonix Inc., Patent No. 09/282-315.

Yeşilata, B., (1990). Güneş Hareketini İzleyen Parabolik Yoğunlaştırıcı Tip Güneş Kollektörlerinin Tasarımı, Dizaynı ve Isıl Veriminin Araştırılması. Elazığ: Master Thesis, Institute of Science, Fırat University.

Zeghoudi, A. & Benmouiza, K., (2023). Solar Power Heliostat Control Using Image Processing Technology and Artificial Neural Networks. Journal Européen des Systèmes Automatisés, Vol. 56, (No. 1,), pp. 165-171.

The Evolution of Solar Tracking Systems (STS):

"Principles of Image Processing for Advanced STS"

Assoc. Prof. Dr. Erkan KACAN

ÖZGÜR PRESS

ISBN 978-625-5646-05-7