RESSALVA

Atendendo a solicitação do autor, George John Orbezo Alvarez, o texto completo deste documento será disponibilizado somente a partir de 08/03/2027.

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA
CÂMPUS DE BAURU

GEORGE JOHN ORBEZO ALVAREZ

ANÁLISE DE ALGORITMOS DE MOVIMENTAÇÃO DE UM HELIOSTATO

Bauru
2025

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
ampus d auru

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ME ÂNI A

GEORGE JOHN ORBEZO ALVAREZ

ANÁLISE DE ALGORITMOS DE MOVIMENTAÇÃO DE UM HELIOSTATO

Dissertação apresentada à Faculdade de Engenharia - UNESP – Campus de Bauru, para obtenção do título de Mestre em Engenharia Mecânica.
Área de Conhecimento: Ciências Térmicas.

Orientador
Prof. Dr. Elí Wilfredo Zavaleta Aguilar

Bauru
2025

Orbezo Alvarez, George John.
Análise de algoritmos de movimentação de um heliostato / George John Orbezo Alvarez. – Bauru, 2025
124 f. : il.

Dissertação (Mestrado)–Universidade Estadual Paulista (Unesp), Faculdade de Engenharia, Bauru
Orientador: Eli Wilfredo Zavaleta Aguilar

1. Precisão. 2. Processamento de imagem. 3. Movimentação hibrida. I. Universidade Estadual Paulista. Faculdade de Engenharia. II. Título.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Bauru

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE GEORGE JOHN ORBEZO ALVAREZ, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA, DA FACULDADE DE ENGENHARIA - CÂMPUS DE BAURU.

Aos 07 dias do mês de março do ano de 2025, às 9h, por meio de Videoconferência, realizou-se a defesa de DISSERTAÇÃO DE MESTRADO de GEORGE JOHN ORBEZO ALVAREZ, intitulada **ANÁLISE COMPARATIVA DE ALGORITMOS DE MOVIMENTAÇÃO DE UM HELIOSTATO**. A Comissão Examinadora foi constituída pelos seguintes membros: Prof. Dr. ELÍ WILFREDO ZAVALETA AGUILAR (Orientador(a) - Participação Virtual) do(a) Departamento de Engenharia / Unesp Câmpus de Itapeva, Prof. Dr. CELSO EDUARDO LINS DE OLIVEIRA (Participação Virtual) do(a) Departamento de Engenharia de Biossistemas / USP/Pirassununga, Prof. Dr. VICENTE LUIZ SCALON (Participação Virtual) do(a) Departamento de Engenharia Mecânica / Faculdade de Engenharia de Bauru UNESP. Após a exposição pelo mestrando e arguição pelos membros da Comissão Examinadora que participaram do ato, de forma presencial e/ou virtual, o discente recebeu o conceito final: APROVADO . Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelo(a) Presidente(a) da Comissão Examinadora.

Prof. Dr. ELÍ WILFREDO ZAVALETA AGUILAR

Documento assinado digitalmente
gov.br
ELI WILFREDO ZAVALETA AGUILAR
Data: 07/03/2025 13:37:21-0300
Verifique em https://validar.iti.gov.br

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru

PROPOSTA DE ALTERAÇÃO DO TÍTULO

A COMISSÃO EXAMINADORA PROPÕE A ALTERAÇÃO DO TÍTULO DO TRABALHO DO ALUNO: **GEORGE JOHN ORBEZO ALVAREZ**

DE: "ANÁLISE COMPARATIVA DE ALGORITMOS DE MOVIMENTAÇÃO DE UM HELIOSTATO"

PARA:

"ANÁLISE DE ALGORITMOS DE MOVIMENTAÇÃO DE UM HELIOSTATO"

Bauru, 7 de março de 2025.

gov.br
Documento assinado digitalmente
ELI WILFREDO ZAVALETA AGUILAR
Data: 07/03/2025 14:00:39-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Elí Wilfredo Zavaleta Aguilar
Orientador

FEB FACULDADE DE ENGENHARIA BAURU
Faculdade de Engenharia de Bauru – Pós-graduação
Av. Eng. Luiz Edmundo Carrijo Coube, 14-01 17033-360 Bauru - SP
tel. (14) 3103-6108 spg.feb@unesp.br www.feb.unesp.br

AGRADECIMENTOS

Agradeço aos meus pais e avô pelo apoio que deram aos meus estudos. Ao meu orientador pela confiança e pelo apoio constante no desenvolvimento deste trabalho.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Processo: 407619/2023-2, pelo apoio financeiro para o desenvolvimento deste trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) pela bolsa concedida.

RESUMO

A população mundial, seu consumo energético e contaminação estão aumentando, assim é necessário utilizar energia renovável a qual não agride o meio ambiente. A energia solar é a maior fonte de energia disponível no mundo, este fato incentiva pesquisas para seu aproveitamento. A energia solar térmica, ainda pouco utilizada na indústria, desempenharia um papel importante em diversos processos, sendo o sistema de receptor central uma tecnologia apropriada para processos térmicos de elevada temperatura. Os sistemas de receptor centrais podem ser compostos por múltiplos heliostatos para refletir a radiação direta incidente até um receptor localizado no topo da uma torre. Normalmente os heliostatos tem erros de reflexão devido a problemas estruturais e, uma calibração individual é necessária, por isso é indispensável analisar qual método de movimentação seria adequado para resolver esse problema. Assim, este trabalho apresenta uma análise de 2 algoritmos de movimentação de um heliostato, um de malha aberta e outro híbrido. Para isso, um protótipo de heliostato foi construído e programado para refletir radiação direta até um alvo. Com o sistema de movimentação de malha aberta desenvolvido obteve-se um índice RMSE de 5,9 mrad onde o erro pode ser atribuído a erros estruturais e de programação. Por outro lado, o sistema de movimentação híbrida desenvolvido obteve um índice RMSE de 1,29 mrad. Conforme os testes realizados, pode-se concluir que o sistema de movimentação híbrida obtém melhores resultados comparado a um sistema de movimentação por malha aberta, pois é independente dos erros estruturais e de instalação que um heliostato possa apresentar.

Palavras-chave: Precisão, processamento de imagem, movimentação hibrida.

ABSTRACT

The world population, its energy consumption and pollution are increasing, so it is necessary to use renewable energy which does not harm the environment. Solar energy is the largest source of energy available in the world, this fact encourages research into its use. Solar thermal energy, still little used in industry, would play an important role in several processes, with the central receiver system being an appropriate technology for high-temperature thermal processes. Central receiver systems can be composed of multiple heliostats to reflect the incident direct radiation to a receiver located at the top of a tower. Heliostats usually have reflection errors due to structural problems and individual calibration is necessary, so it is essential to analyze which movement method would be suitable to solve this problem. Therefore, this work presents an analysis of two heliostat movement algorithms, one open-loop and the other hybrid. For this, a heliostat prototype was built and programmed to reflect direct radiation to a target. With the developed open-loop motion system, an RMSE index of 5,9 mrad was obtained, where the error can be attributed to structural and programming errors. On the other hand, the developed hybrid motion system obtained an RMSE index of 1,29 mrad. According to the tests performed, it can be concluded that the hybrid motion system obtains better results compared to an open-loop motion system, since it is independent of the structural and installation errors that a heliostat may present.

Keywords: Precision, image processing, hybrid movement.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

IEA	*International Energy Agency*
INMET	Instituto Nacional de Meteorologia
INPE	Instituto Nacional de Pesquisas Espaciais
NASA	*National Aeronautics and Space Administration*
RMSE	Erro quadrático médio.
SRH	Superfície refletora do heliostato
SRC	Sistema de Receptor Central
CCD	Dispositivo de carga acoplada
CES	Concentração de Energia Solar

LISTA DE SÍMBOLOS

c_x	comprimento do alvo na horizontal (m)
c_y	comprimento do alvo na vertical (m)
cp_x	relação entre comprimento e pixels na horizontal (m/px)
cp_y	relação entre comprimento e pixles na vertical (m/px)
dif_x	diferença em pixels na horizontal dos centros do alvo e do reflexo solar (px)
dif_y	diferença em pixels na vertical dos centros do alvo e do reflexo solar (px)
E	equação do tempo (min)
GMT	horário de Greenwich (-)
H	distância vertical entre o os centros do heliostato e o alvo (m)
h_{px}	número de pixels do alvo na vertical (px)
L_{loc}	longitude local (°)
L_{st}	fuso horário local (°)
n	dia do ano (-)
$\vec{n}$	vetor unitário da superfície refletora do heliostato (-)
R	distância horizontal entre o os centros do heliostato e o alvo (m)
$\vec{s}$	vetor unitário do sol (-)
t	hora padrão (local) (min)
$\vec{t}$	vetor unitário do alvo da torre (-)
ts	hora solar (min)
w_{px}	número de pixels do alvo na horizontal (px)

Letras gregas

α_y	erro do ângulo de elevação (rad)
α_s	ângulo de elevação solar (°)
α_{SRH}	ângulo de elevação da superfície refletora do heliostato (°)
γ_n	ângulo azimutal do heliostato (°)
γ_s	ângulo azimutal solar (°)
γ_{SRH}	ângulo azimutal da superfície refletora do heliostato (°)

γ_t	ângulo azimutal do alvo (°)
γ_x	erro do ângulo de azimute (rad)
δ	declinação (°)
θ_n	ângulo zenital do heliostato (°)
θ_s	ângulo zenital do sol (°)
θ_t	ângulo zenital do alvo (°)
ϕ	Latitude (°)
ω	ângulo horário (°)

SUMÁRIO

1. INTRODUÇÃO

A população mundial aumenta a cada dia e consequentemente a demanda por energia aumenta, isso faz com que os recursos energéticos tradicionais se tornem escassos, portanto, há a necessidade de gerar energia de forma eficiente e com baixo impacto ao meio ambiente. No ano de 2023, a temperatura média da superfície da Terra foi a mais quente já registrada, conforme uma análise da NASA (Bardan, 2024). As temperaturas globais no ano passado estiveram cerca de 1,2 graus Celsius acima da média do período de referência da NASA (1951 a 1980). A crise climática que enfrentamos atualmente e a necessidade de reduzir as emissões de gases de efeito estufa impulsionaram a investigação de fontes de energia alternativas. Dentre os diversos recursos renováveis, a energia solar é a opção mais adequada, sustentável, inesgotável e infinita, a chamada energia verde (Kumar, K. et al., 2021). Este fato incentiva pesquisas sobre seu uso, embora existam algumas dificuldades, como sazonalidade, nuvens, falta de uso noturno e baixa densidade energética onde, mesmo fora da atmosfera é de 1367 W/m² (Duffie et al., 2020), a qual diminui devido aos efeitos de dispersão e absorção na atmosfera até atingir a superfície terrestre. As duas tecnologias mais utilizadas para uso do recurso solar são a fotovoltaica e a térmica. A energia solar fotovoltaica é a transformação direta da radiação solar em eletricidade, sendo que essa transformação ocorre em painéis fotovoltaicos que utilizam materiais semicondutores que apresentam o efeito fotoelétrico.

A energia solar térmica consiste na transformação da radiação solar em energia térmica para aquecer materiais (sólidos, líquidos ou gases) mediante coletores solares térmicos, onde podem ser alcançadas temperaturas de até mais de 1000 °C dependendo da tecnologia do coletor. Para temperaturas médias e altas (> 400 °C), o coletor deve possuir um concentrador, que é basicamente uma superfície que concentra a radiação solar em pequenas áreas. A tecnologia solar térmica pode ser aproveitada na indústria, sendo que no Brasil foi identificado (Solar Payback, 2017) que no setor industrial o calor é responsável por 80% (2,82 EJ) da demanda energética, sendo que 53% dessa energia provém de fontes fósseis, e 41% do calor nas indústrias utiliza temperaturas acima de 400 °C. Assim, existe a possibilidade de utilizar a energia solar para processos industriais e atingir as metas de carbono zero até 2050 as quais segundo a International Energy Agency - IEA (2021) requer de uma

transformação completa na forma como produzimos, transportamos e consumimos energia.

Para atingir temperaturas > 400 °C no receptor, o coletor deve possuir concentrador; além disso, o concentrador deve possuir sistema de rastreamento solar, pois a maioria dos concentradores, como disco parabólico, cilindro-parabólico, Scheffler ou sistema de receptor central, trabalham com radiação solar direta. Algumas aplicações comerciais são indicadas na Tabela 1.

Tabela 1. Uso de diferentes tipos de concentradores.

Concentrador	Uso
Disco parabólico	- Geração de energia elétrica, de energia térmica ou de realização de trabalho mecânico (Haag, 2007) - Cozimento de alimentos (Cerqueira et al., 2019) - Dessalinização (Kumar, K. R. et al., 2021)
Calha cilindro-parabólica	- Geração de eletricidade (Kumar et al., 2021) - Aquecimento de processos (Kumar et al., 2021)
Fresnel linear	- Movimentar turbinas e gerar eletricidade (Saettone, 2012) - Geração de vapor (Gea et al., 2009) - Dessalinização (Saettone, 2012)
Sistema de receptor central	- Geração de energia elétrica para abastecimento de processos industriais (Bezerra et al., 2018) - Produção de vapor em trocador de vapor e gerar eletricidade (Silvestre, 2016)

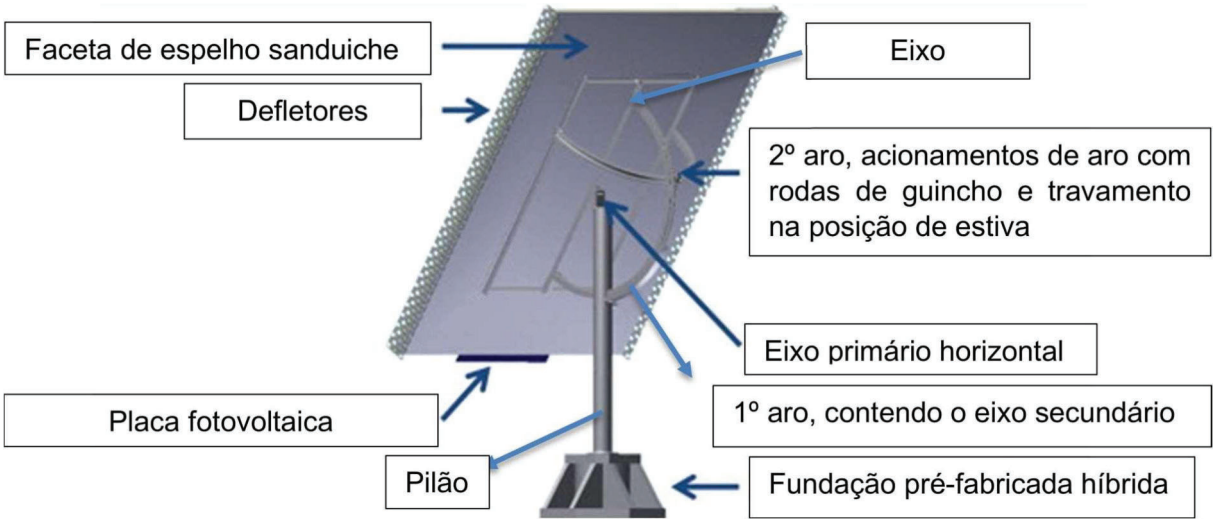
Fonte: Autoria Própria (2025).

Os sistemas receptores centrais (SRC) também chamados de sistemas de torre central, ou torre central são um tipo de coletor solar que consiste em uma torre com um receptor na sua parte superior e múltiplos heliostatos que são estruturas metálicas com dois eixos de rotação. A parte superior do heliostato possui uma superfície refletora, que em alguns casos requer uma certa curvatura para focar a radiação solar, para refletir a radiação solar direta incidente até o receptor. A superfície refletora do heliostato (SRH) pode ter geometrias diferentes, o que depende fundamentalmente

do tipo de receptor. Geralmente são feitos de vidro com uma camada reflexiva de prata ou alumínio e uma camada de tinta na parte traseira para protegê-la.

O desenho de um heliostato pode ser visto na Figura 1, encontrado no trabalho de Pfahl et al. (2013) que desenvolveram um heliostato composto por dois aros e seu sistema de movimentação.

Figura 1. Heliostato com acionamento de aros.



Fonte: Adaptado de Pfahl et al. (2013)

É provável que um dos primeiros heliostatos foi aplicado pelo cientista grego Arquimedes no ano 212 AC, que usou as propriedades reflexivas dos escudos de bronze para focar a luz solar e incendiou navios de madeira do Império Romano que sitiavam Siracusa (Office of Energy Efficiency and Renewable Energy, 2004).

Segundo Kalogirou (2009), os SRC são indicados para sistemas de maior porte (de 10 MW para cima). Estes sistemas têm sido estudados com mais intensidade desde o final da década de 1970 até ao início da década de 1980, onde surgiu a necessidade de uso de energias renováveis para lidar com a dependência das importações de petróleo (Malan, 2014). Mais tarde, em 1981, foi construída a Solar One que foi um projeto de demonstração de receptor de 10 MWe, onde foi verificada a viabilidade da tecnologia em escala de utilidade (Kolb et al., 2007). Alguns anos depois a Solar One tornou-se a Solar Two, que funcionou de 1998 a 1999 onde foram usados sais fundidos para armazenamento térmico (Jones e Stone, 1999). Em 2004, a Espanha investiu no desenvolvimento da tecnologia de concentração de energia solar (CES), cujas primeiras usinas de SRC comerciais encerraram em 2007 (PS10)

(Figura 2) e 2009 (PS20) (Lovegrove e Stein, 2012). A Gemasolar, na Espanha, seguiu em 2011 e se tornou a primeira usina de SRC do mundo com capacidade de fornecer energia térmica 24 horas/dia devido à sua alta capacidade de armazenamento térmico de 15 horas e vários painéis solares fotovoltaicos (Lata et al., 2011).

Os SRC possuem as seguintes vantagens (Kalogirou, 2009):

- Os espelhos coletam a luz solar e a concentram em um único receptor, minimizando assim o transporte de energia térmica.
- Assim como o concentrador de disco parabólico, por concentrar a radiação solar em um único receptor central e por rastrear o sol em dois eixos, possui altas taxas de concentração, de 300 a 1500, menor apenas que o disco parabólico.

Figura 2. Foto do sistema de receptor central PS10 na Espanha.

Fonte: Abengoa (2012).

Conforme o Malan (2014) tem-se dado importância à redução de custo dos SRC e ao uso de componentes eletrônicos para reduzir o erro de reflexão nos SRC. Heliostatos desalinhados ocasionam perdas de energia, como na usina Solar Two onde foi registrado perdas de 10 a 20% (Jones e Stone, 1999). Segundo Jones e Stone (1999) as especificações de projeto dos heliostatos Solar One requerem uma precisão RMSE de 2,1 mrad total para ambos os eixos.

Para reduzir esses erros de movimentação, é necessária a implementação de sistema com alinhamentos precisos dos refletores para um funcionamento eficiente, podendo ser calibrados periodicamente em função da avaliação da posição da imagem refletida, por exemplo.

Essa operação pode ser desenvolvida por sistemas de malha fechada, que utilizam dispositivos ou tecnologias de inteligência artificial para avaliar a imagem refletida, como câmeras localizadas no solo, na torre ou no próprio heliostato para obter uma medição real de reflexão, e serve como feedback ao sistema sobre o erro que deve ser corrigido. Esses métodos serão explicados na subseção 3.1.2.

É necessário analisar qual método de movimentação é capaz de reduzir erros de movimentação e melhorar a precisão dos heliostatos. Nesse sentido, este trabalho pretende realizar uma análise comparativa de algoritmos de movimentação, a fim de obter um algoritmo preciso para movimentação de um heliostato.

1.1. Objetivos

O objetivo geral do trabalho é fazer uma análise de algoritmos de movimentação de um heliostato. São objetivos específicos:

- ✓ Desenvolver um sistema de movimentação por malha aberta.
- ✓ Desenvolver um sistema de movimentação hibrida.
- ✓ Testar e comparar os erros dos sistemas de movimentação em condições reais de funcionamento com protótipo de heliostato tanto para malha aberta como hibrida.

2. CONCENTRADORES SOLARES

2.1. Energia solar

A energia solar é a maior fonte energética do planeta, é não poluente, podendo ser encontrada, em maior ou menor grau, em qualquer lugar do planeta, assim, pode ser captada e transformada em energia térmica ou elétrica no local de utilização. No final de 2022, a capacidade mundial de geração foi de 3382 GW. A energia hidrelétrica representou a maior parcela do total, com capacidade de 1393 GW. Sendo a energia solar a segunda colocada, como visto na Figura 3.

melhores resultados em comparação ao sistema de movimentação por malha aberta (erro de 5,9 mrad).

Ambos os sistemas são afetados pelas condições ambientais: vento e radiação. O vento influencia, já que embora os motores não estejam em rotação no momento da captura de imagem, o vento cria uma falsa distância entre reflexo e centro do alvo. Já, as nuvens passageiras fazem que o sistema opere em malha aberta. Nos dois casos há um aumento de erro. Além disso, ambos os sistemas são impactados pela limitação da rotação dos eixos, que possui um valor mínimo de 0,045° (0,79 mrad).

Por outro lado, o erro elevado no sistema de movimentação por malha aberta pode ser atribuído a erros de instalação ou medição, programação, bem como à influência de terreno irregular ou à falta de perpendicularidade entre os eixos. No entanto, no sistema de movimentação híbrida, esses erros mencionados anteriormente não têm grande impacto, pois o sistema está em constante calibração graças à movimentação por malha fechada, que considera apenas a imagem do reflexo solar.

6. CONCLUSÕES E SUGESTÕES

O protótipo de um heliostato foi projetado, construído e programado para refletir radiação direta para um alvo e foram comparados dois algoritmos de movimentação, um de malha aberta e outro de malha híbrida.

O sistema de movimentação por malha aberta, obteve um erro total com índice RMSE de 5,9 mrad, onde o erro é devido a erros estruturais e de programação. Embora um número maior de testes por vários dias, pode ser necessário para verificar o erro.

O sistema de movimentação híbrida, obteve um erro total com índice RMSE de 1,29 mrad, devido à redução de erros estruturais existentes no sistema de movimentação em malha aberta.

Conforme os testes realizados, pode-se concluir que o sistema de movimentação híbrida obtém melhores resultados comparado a um sistema de movimentação por malha aberta, pois é independente dos erros de instalação que um heliostato possa apresentar, além disso este sistema é corrigido a cada 10 segundos à medida que o sol se move ao longo do dia.

O sistema de movimentação por malha aberta, embora possua um processamento mais simples, depende de fatores de medição exata, como determinação do norte geográfico, comprimentos para localização do centro do alvo (horizontal e vertical desde o centro do heliostato), determinação do azimute do alvo, estrutura sem folgas e perpendicularidade dos eixos de rotação, tudo isso acarreta erros que podem ser até acumulativos ao longo do tempo.

Para trabalhos futuros, sugere-se criar um sistema híbrido utilizando outros métodos de processamento de imagens, ou implementar métodos baseados em inteligência artificial e comparar os resultados com os obtidos neste trabalho.

REFERÊNCIAS

ABENGOA. **Abengoa solar**. Sevilla: Abengoa Solar, 2012. Disponível em: https://web.archive.org/web/20120621195650/http://www.abengoasolar.com/corp/web/es/nuestras_plantas/plantas_en_operacion/espana/PS10_la_primera_torre_comercial_del_mundo.html. Acesso em: 1 mar. 2024.

AEE INSTITUTE FOR SUSTAINABLE TECHNOLOGIES. **SHIP Plants Database**. Gleisdorf: AEE INTEC, 2023. Disponível em: https://energieatlas.aee-intec.at/index.php/view/map?repository=ship&project=ship_edit. Acesso em: 23 set. 2024.

ARQUEROS, F.; JIMÉNEZ, A.; VALVERDE, A. A novel procedure for the optical characterization of solar concentrators. **Solar Energy**, Amsterdam, v. 75, n. 2, p. 135–142, ago. 2003. DOI: https://doi.org/10.1016/j.solener.2003.07.008. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X03002597. Acesso em: 30 jul. 2024.

BARDAN, R. **NASA Analysis Confirms 2023 as Warmest Year on Record**. Washington, D.C.: NASA, 2024. Disponível em: https://www.nasa.gov/news-release/nasa-analysis-confirms-2023-as-warmest-year-on-record/. Acesso em: 25 maio 2024.

BERENGUEL, M.; RUBIO, F. R.; VALVERDE, A.; LARA, P. J.; ARAHAL, M. R.; CAMACHO, E. F.; LÓPEZ, M. An artificial vision-based control system for automatic heliostat positioning offset correction in a central receiver solar power plant. **Solar Energy**, Amsterdam, v. 76, n. 5, p. 563–575, maio 2004. DOI: https://doi.org/10.1016/j.solener.2003.12.006. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X0300464X. Acesso em: 26 jul. 2024.

BERN, G.; SCHÖTTL, P.; HEIMSATH, A.; NITZ, P. Novel imaging closed loop control strategy for heliostats. **AIP Conference Proceedings**, Abu Dhabi, v. 1850, n. 1, p. 030005-1–10, jun. 2017. DOI: https://doi.org/10.1063/1.4984348. Disponível em:

https://pubs.aip.org/aip/acp/article/1850/1/030005/616701/Novel-imaging-closed-loop-control-strategy-for. Acesso em: 30 jul. 2024.

BERN, G.; SCHÖTTL, P.; VAN-ROOYEN, D. W.; HEIMSATH, A.; NITZ, P. Parallel in-situ measurement of heliostat aim points in central receiver systems by image processing methods. **Solar Energy**, Amsterdam, v. 180, p. 648–663, mar. 2019. DOI: https://doi.org/10.1016/j.solener.2019.01.051. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X19300659. Acesso em: 30 jul. 2024.

BEZERRA, P. H. S.; SERAPHIN, O. J.; OLIVEIRA, C. E. L. Previsão de performance energética por meio de simulação transiente de um sistema CSP com torre central integrado a atividades agroindustriais. **Energia na agricultura**, [*s. l.*], v. 33, n. 3, p. 264–269, nov. 2018. DOI: https://doi.org/10.17224/EnergAgric.2018v33n3p264-269. Disponível em: https://revistas.fca.unesp.br/index.php/energia/article/view/3235. Acesso em: 20 jun. 2024.

BURISCH, M.; OLANO, X.; SANCHEZ, M.; OLARRA, A.; VILLASANTE, C.; OLASOLO, D.; MONTERREAL, R.; ENRIQUE, R.; FERNÁNDEZ, J. Scalable heliostat calibration system (SHORT) - Calibrate a whole heliostat field in a single night. **AIP Conference Proceedings**, Santiago, v. 2033, n. 1, p. 040009-1–8, nov. 2018. DOI: https://doi.org/10.1063/1.5067045. Disponível em: https://pubs.aip.org/aip/acp/article/2033/1/040009/1023277/Scalable-heliostat-calibration-system-SHORT. Acesso em: 30 jul. 2024.

CARBALLO, J. A.; BONILLA, J.; BERENGUEL, M.; FERNÁNDEZ-RECHE, J.; GARCÍA, G. New approach for solar tracking systems based on computer vision, low cost hardware and deep learning. **Renewable Energy**, [*s. l.*], v. 133, p. 1158–1166, abr. 2019. DOI: https://doi.org/10.1016/j.renene.2018.08.101. Disponível em: https://www.sciencedirect.com/science/article/pii/S0960148118310516. Acesso em: 30 jul. 2024.

CERQUEIRA, T. B.; DOS SANTOS, J.; SILVA, D. L. N. Aplicação de um concentrador solar de Scheffler de 2,7 m2 para cozimento de alimentos. **Revista Brasileira de Energia solar**, [*s. l.*], v. 10, p. 88–97, mar. 2019. DOI: https://doi.org/10.59627/rbens.2019v10i2.282. Disponível em: https://rbens.org.br/rbens/issue/view/25. Acesso em: 14 fev. 2024.

CHEN, Y. T.; CHONG, K. K.; BLIGH, T. P.; CHEN, L. C.; YUNUS, J.; KANNAN, K. S.; LIM, B. H.; LIM, C. S.; ALIAS, M. A.; BIDIN, N.; ALIMAN, O.; SALEHAN, S.; REZAN, S. A.; TAM, C. M.; TAN, K. K. Non-Imaging, focusing heliostat. **Solar Energy**, Amsterdam, v. 71, n. 3, p. 155–164, 2001. DOI: https://doi.org/10.1016/S0038-092X(01)00041-X. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X0100041X. Acesso em: 7 abr. 2024.

CHIESI, M.; SCARSELLI, E. F.; GUERRIERI, R. Run-time detection and correction of heliostat tracking errors. **Renewable Energy**, [*s. l.*], v. 105, p. 702–711, 2017. DOI: https://doi.org/10.1016/j.renene.2016.12.093. Disponível em: https://www.sciencedirect.com/science/article/pii/S0960148116311624. Acesso em: 30 jul. 2024.

COLLINS, M.; POTTER, D.; BURTON, A. Design and simulation of a sensor for heliostat field closed loop control. **AIP Conference Proceedings**, Abu Dhabi, v. 1850, n. 1, p. 030009-1–7, jun. 2017. DOI: https://doi.org/10.1063/1.4984352. Disponível em: https://pubs.aip.org/aip/acp/article/1850/1/030009/616727/Design-and-simulation-of-a-sensor-for-heliostat. Acesso em: 30 jul. 2024.

COQUAND, M.; CALIOT, C.; HÉNAULT, F. Backward-gazing method for heliostats shape errors measurement and calibration. **AIP Conference Proceedings**, Abu Dhabi, v. 1850, n. 1, p. 030010-1–7, jun. 2017. DOI: https://doi.org/10.1063/1.4984353. Disponível em: https://pubs.aip.org/aip/acp/article/1850/1/030010/616766/Backward-gazing-method-for-heliostats-shape-errors. Acesso em: 30 jul. 2024.

DABROWKSI, J. **Verfahren zur Positionsbestimmung oder zur Antriebsregelung eines eine Spiegelfläche aufweisenden Heliostaten sowie System zur Positionsbestimmung oder zur Antriebsregelung des Heliostaten**. Depositante: Deutsches Zentrum für Luft- und Raumfahrt e.V. DE 102013207022 B3. Depósito: 18 abr. 2013. Concessão: 12 jun. 2014. Disponível em: https://patents.google.com/patent/DE102013207022B3/de. Acesso em: 30 jul. 2024.

DUFFIE, J. A.; BECKMAN, W. A.; BLAIR, N. **Solar engineering of thermal processes:** photovoltaics and wind. 4. ed. Hoboken, N.J: Wiley, 2013. Disponível em: https://unesp.primo.exlibrisgroup.com/discovery/fulldisplay?docid=cdi_skillsoft_books24x7_bke00052866&context=PC&vid=55UNESP_INST:UNESP&lang=pt&search_scope=MyInst_and_CI&adaptor=Primo%20Central&tab=Everything&query=any,contains,Solar%20engineering%20of%20thermal%20processes:%20photovoltaics%20and%20wind%20duffie&offset=0. Acesso em: 20 jun. 2024.

Office of Energy Efficiency and Renewable Energy. **The History of Solar**. Washington, DC: U.S. Department of Energy, 2004. Disponível em: https://www1.eere.energy.gov/solar//pdfs/solar_timeline.pdf. Acesso em: 21 jun. 2024.

FAIRMAN, P.; FARRANT, D.; CONNOR, P. Closed loop optical tracking of heliostats. **AIP Conference Proceedings**, Casablanca, v. 2126, n. 1, p. 030021-1–8, jul. 2019. DOI: https://doi.org/10.1063/1.5117533. Disponível em: https://pubs.aip.org/aip/acp/article/2126/1/030021/701652/Closed-loop-optical-tracking-of-heliostats. Acesso em: 30 jul. 2024.

FREEMAN, J.; E.U., K.; S.R., R. Study of the Errors Influencing Heliostats for Calibration and Control System Design. *In*: INTERNATIONAL CONFERENCE ON RECENT ADVANCES AND INNOVATIONS IN ENGINEERING (ICRAIE-2014), 2014, Jaipur, India. **Proceedings** [...]. Jaipur, India: IEEE, 2014. p. 1–8. Disponível em: https://ieeexplore.ieee.org/document/6909113. Acesso em: 24 jan. 2025.

FREEMAN, J.; KEERTHI, K. S.; CHANDRAN, L. R. Closed loop control system for a heliostat field. *In*: INTERNATIONAL CONFERENCE ON TECHNOLOGICAL

ADVANCEMENTS IN POWER AND ENERGY, TAP ENERGY 2015, 2015, Kollam. **Proceedings** [...]. Kollam: IEE, 2015. p. 272–277. Disponível em: https://ieeexplore.ieee.org/document/7229630. Acesso em: 19 dez. 2024.

GEA, M.; TILCA, F.; PLACCO, C.; CASO, R.; MACHACA, A.; SARAVIA, L. Acumulación térmica en hormigón de la energía solar captada por un concentrador tipo Fresnel lineal para generación de vapor. **Avances en Energías Renovables y Medio Ambiente**, [*s. l.*], v. 13, p. 113, 2009. Disponível em: https://sedici.unlp.edu.ar/handle/10915/96888. Acesso em: 11 fev. 2024.

GONZALEZ, R. C.; WOODS, R. E. **Digital image processing**. 4. ed. New York: Pearson, 2018. Disponível em: https://unesp.primo.exlibrisgroup.com/discovery/fulldisplay?docid=alma999882741506341&context=L&vid=55UNESP_INST:UNESP&lang=pt&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=any,contains,Digital%20image%20processing%20gonzalez%202018&offset=0. Acesso em: 23 out. 2024.

GROSS, W. **Closed loop tracking system using signal beam**. Depositante: Idealab. US 20170102446 A1. Depósito: 19 dez. 2016. Concessão: 13 abr. 2017. Disponível em: https://patents.google.com/patent/US20170102446A1/en. Acesso em: 2 ago. 2024.

GUANGYU, L.; ZHONGKUN, C. Heliostat attitude angle detection method based on BP neural network. **MATEC Web of Conferences**, Cape Town, v. 139, p. 00043, dez. 2017. DOI: https://doi.org/10.1051/matecconf/201713900043. Disponível em: https://www.matec-conferences.org/component/makeref/?task=show&type=html&doi=10.1051/matecconf/201713900043. Acesso em: 4 ago. 2024.

HAAG, R. **Desenvolvimento de um radiômetro espectral e metodologia para caracterização do espectro solar**. 2007. Dissertação (Mestrado em Engenharia) -

Universidade Federal do Rio Grande do Sul, Porto Alegre, 2007. Disponível em: https://lume.ufrgs.br/handle/10183/77973. Acesso em: 20 jun. 2024.

HARPER, P. J.; DREIJER, J.; MALAN, K.; LARMUTH, J.; GAUCHE, P. Use of MEMs and optical sensors for closed loop heliostat control. **AIP Conference Proceedings**, Cape Town, v. 1734, n. 1, p. 070015-1–7, maio 2016. DOI: https://doi.org/10.1063/1.4949162. Disponível em: https://pubs.aip.org/aip/acp/article/1734/1/070015/997519/Use-of-MEMs-and-optical-sensors-for-closed-loop. Acesso em: 2 ago. 2024.

HICKERSON, K. P.; REZNIK, D. **Heliostat with integrated image-based tracking controller**. Depositante: Esolar, Inc. US 008153945 B2. Depósito: 11 mar. 2011. Concessão: 10 abr. 2012. Disponível em: https://www.researchgate.net/publication/256438311. Acceso em: 30 jul. 2024.

HINES, B. **Heliostat characterization using starlight**. Depositante: Solar Reserve Technology Llc. WO 2016/205612 A1. Depósito: 22 dez. 2016. Disponível em: https://patents.google.com/patent/WO2016205612A1/en. Acceso em: 30 jul. 2024.

HOFFSCHMIDT, B. **Machbarkeitsstudie zur Entwicklung einer radargestützten Positionsregelung von Heliostatenfeldern für Solarturmkraftwerke:** HelioScan. Jülich: Solar-Institut Jülich der FH Aachen (SIJ), 2013. Disponível em: https://opus.bibliothek.fh-aachen.de/opus4/frontdoor/index/index/searchtype/all/start/2/rows/1/yearfq/2013/institutefq/Solar-Institut%2BJ%C3%BClich/docId/6549/utm_source/chatgpt.com. Acesso em: 30 jul. 2024.

INSTITUTO NACIONAL DE METEOROLOGIA. **Mapa de Estações**. Brasília, DF: INMET, 2025. Disponível em: https://mapas.inmet.gov.br/#. Acesso em: 9 jan. 2025.

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **Atlas brasileiro de energía solar**. 2. ed. São José dos Campos: INPE, 2017. v. 2. Disponível em: https://labren.ccst.inpe.br/atlas_2017.html. Acesso em: 20 jun. 2024.

INTERNATIONAL ENERGY AGENCY. **Net Zero by 2050**. Paris: IEA, 2021. Disponível em: https://www.iea.org/reports/net-zero-by-2050. Acesso em: 20 maio 2024.

INTERNATIONAL RENEWABLE ENERGY AGENCY. **Renewable energy statistics 2023 = statistiques d'énergie renouvelable 2023 = estadísticas de energía renovable 2023**. Abu Dhabi: IRENA, 2023. Disponível em: https://www.irena.org/Publications. Acesso em: 20 jun. 2024.

JONES, S.; STONE, K. Analysis of strategies to improve heliostat tracking at solar two. *In*: ASME INTERNATIONAL SOLAR ENERGY CONFERENCE, 1999, Albuquerque. **Proceedings** [...]. Albuquerque: Sandia National Laboratories, 1999. Disponível em: https://www.osti.gov/biblio/3259. Acesso em: 4 jul. 2024.

KALOGIROU, S. A. **Solar energy engineering:** processes and systems. 1. ed. Burlington, MA: Elsevier/Academic Press, 2009. Disponível em: https://unesp.primo.exlibrisgroup.com/discovery/fulldisplay?docid=alma9910099153406341&context=L&vid=55UNESP_INST:UNESP&lang=pt&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=any,contains,Solar%20energy%20engineering:%20processes%20and%20systems. Acesso em: 20 jun. 2024.

KLEIN, S. A. Calculation of Monthly Average Insolation on Tilted Surfaces. **Solar Energy**, Amsterdam, v. 19, p. 325–329, 1977. Disponível em: https://www.sciencedirect.com/science/article/pii/0038092X77900019. Acesso em: 9 abr. 2024.

KOLB, G. J.; JONES, S. A.; DONNELLY, M. W.; GORMAN, D.; THOMAS, R.; DAVENPORT, R.; LUMIA, R. **Heliostat Cost Reduction Study**. Albuquerque: Sandia National Laboratories, 2007. 167 p. (SANDIA REPORT – SAND2007-3293) Disponível em:

https://www.osti.gov/servlets/purl/909667https://www.osti.gov/servlets/purl/912923/. Acesso em: 4 ago. 2024.

KRIBUS, A.; VISHNEVETSKY, I.; YOGEV, A.; RUBINOV, T. Closed loop control of heliostats. **Energy**, [*s. l.*], v. 29, n. 5–6, p. 905–913, abr. 2004. DOI: https://doi.org/10.1016/S0360-5442(03)00195-6. Disponível em: https://www.sciencedirect.com/science/article/pii/S0360544203001956. Acesso em: 17 jun. 2024.

KUMAR, K. R.; CHAITANYA, N. V. V. K.; KUMAR, N. S. Solar thermal energy technologies and its applications for process heating and power generation - A review. **Journal of Cleaner Production**, [*s. l.*], v. 282, p. 125296, fev. 2021. DOI: https://doi.org/10.1016/j.jclepro.2020.125296. Disponível em: https://www.sciencedirect.com/science/article/pii/S0959652620353415. Acesso em: 9 jul. 2024.

KUMAR, K.; VARSHNEY, L.; AMBIKAPATHY, A.; SAKET, R. K.; MEKHILEF, S. Solar tracker transcript - A review. **International Transactions on Electrical Energy Systems**, [*s. l.*], v. 31, n. 12, dez. 2021. DOI: https://doi.org/10.1002/2050-7038.13250. Disponível em: https://onlinelibrary.wiley.com/doi/full/10.1002/2050-7038.13250. Acesso em: 20 jun. 2024.

LATA, J.; ALCALDE, S.; FERNANDEZ, D.; LEKUBE, X.; AGARRABERES, I.; MARTINEZ, E.; OLANO, J.; EGUIDAZU, I. Commissioning and operation of commercial surrounding heliostat fields for maximum but safe production. *In*: SOLARPACES, 2011, Granada. **Proceedings** [...]. Granada: SolarPACES, 2011. Disponível em: https://www.solarpaces.org/wp-content/uploads/Commissioning-and-operation-of-commercial-surrounding-heliostat-fields.pdf. Acesso em: 20 jun. 2024.

LIANG, W.; WANG, Z. Research on tracking precision of the heliostat. *In*: ISES WORLD CONGRESS, 2009, Berlin. **Proceedings** [...]. Berlin: Springer, 2009. p. 1764–1767. Disponível em: https://link.springer.com/chapter/10.1007/978-3-540-75997-3_361. Acesso em: 29 jul. 2024.

LOVEGROVE, K.; STEIN, W. **Concentrating solar power technology:** Principles, developments and applications. 2. ed. Cambridge, MA: Woodhead, 2021. Disponível em: https://unesp.primo.exlibrisgroup.com/discovery/fulldisplay?docid=alma999876334806341&context=L&vid=55UNESP_INST:UNESP&lang=pt&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=any,contains,Concentrating%20Solar%20Power%20Technology. Acesso em: 10 ago. 2024.

MALAN, K. J. **A Heliostat Field Control System**. 2014. Dissertation (Master of Engineering) - Stellenbosch University, Stellenbosch, 2014. Disponível em: https://scholar.sun.ac.za/handle/10019.1/86674. Acesso em: 29 jul. 2024.

MEMOL, M. **Heliostat Reflects Sunlight Inside**. San Francisco, CA: Instructables, 2023. Disponível em: https://www.instructables.com/Heliostat-Reflects-Sunlight-Inside/. Acesso em: 25 maio 2024.

ORDAZ-CASTILLO, J.; GARCIA-LARA, H. D.; TREJO-LUNA, N. G.; MENDEZ-DIAZ, S. An open-loop control algorithm for improved tracking in a heliostat. **Renewable energy, biomass and sustainability**, [*s. l.*], v. 6, n. 1, p. 50–56, abr. 2024. DOI: https://aldeser.org/journals/index.php/REBS/article/view/90. Disponível em: https://aldeser.org/journals/index.php/REBS/article/view/90. Acesso em: 22 jul. 2024.

PARGMANN, M.; LEIBAUER, M.; NETTELROTH, V.; QUINTO, D. M.; PITZ-PAAL, R. Enhancing heliostat calibration on low data by fusing robotic rigid body kinematics with neural networks. **Solar Energy**, Amsterdam, v. 264, nov. 2023. DOI: https://doi.org/10.1016/j.solener.2023.111962. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X23005960. Acesso em: 20 jun. 2024.

PFAHL, A.; BUCK, R.; REHSCHUH, K. **Method for controlling the alignment of a heliostat with respect to a receiver, heliostat device and solar power plant**. Depositante: Deutsches Zentrum fuer Luft- und Raumfahrt e.V. US 8651100 B2.

Depósito: 6 abr. 2009. Concessão: 18 fev. 2014. Disponível em: https://patents.google.com/patent/US8651100B2/en. Acesso em: 1 ago. 2024.

PFAHL, A.; RANDT, M.; HOLZE, C.; UNTERSCHÜTZ, S. Autonomous light-weight heliostat with rim drives. **Solar Energy**, Amsterdam, v. 92, p. 230–240, jun. 2013. DOI: https://doi.org/10.1016/j.solener.2013.03.005. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X13001023. Acesso em: 9 set. 2024.

PFAHL, A.; RHEINLÄNDER, J.; KRAUSE, A.; BUCK, R.; GIULIANO, S.; HERTEL, J.; BLUME, K.; SCHLICHTING, T.; JANOTTE, N.; RIES, A. First lay-down heliostat with monolithic mirror-panel, closed loop control, and cleaning system. **AIP Conference Proceedings**, Casablanca, v. 2126, n. 1, p. 0300421-1–8, jul. 2019. DOI: https://doi.org/10.1063/1.5117554. Disponível em: https://pubs.aip.org/aip/acp/article/2126/1/030042/701651/First-lay-down-heliostat-with-monolithic-mirror. Acesso em: 4 ago. 2024.

PRAHL, C.; GÖHRING, F.; RÖGER, M.; HILGERT, C.; ULMER, S. **Verfahren zur Vermessung von Heliostaten**. DE 102015217086 A1. Depósito: 7 set. 2015. Disponível em: https://patents.google.com/patent/DE102015217086A1/de. Acesso em: 30 jul. 2024.

PRAHL, C.; RÖGER, M.; KIEWITT, W. Advances in optical measurement techniques for solar concentrators. *In*: SOLARPACES 2009, 2009, Berlin. **Proceedings** [...]. Berlin: SolarPACES, 2009. Disponível em: https://www.researchgate.net/publication/224990464_Advances_in_Optical_Measurement_Techniques_for_Solar_Concentrators. Acesso em: 17 maio 2024.

RÖGER, M.; PRAHL, C.; ULMER, S. Heliostat shape and orientation by edge detection. **Journal of Solar Energy Engineering**, [*s. l.*], v. 132, n. 2, p. 0210021, abr. 2010. DOI: https://doi.org/10.1115/1.4001400. Disponível em: https://asmedigitalcollection.asme.org/solarenergyengineering/article/132/2/021002/455822/Heliostat-Shape-and-Orientation-by-Edge-Detection. Acesso em: 4 ago. 2024.

SAETTONE, E. Diseño, construcción y utilización de un concentrador solar tipo Fresnel lineal para desalinización. *In*: XIX SIMPOSIO PERUANO DE ENERGIA SOLAR Y DEL AMBIENTE, 2012, Puno. **Proceedings** [...]. Puno: [*s. n.*], 2012. Disponível em: https://perusolar.org/simposios/. Acesso em: 20 jun. 2024.

SÁNCHEZ-GONZÁLEZ, A.; YELLOWHAIR, J. Reflections between heliostats: Model to detect alignment errors. **Solar Energy**, Amsterdam, v. 201, p. 373–386, maio 2020. DOI: https://doi.org/10.1016/j.solener.2020.03.005. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X2030236X. Acesso em: 4 ago. 2024.

SARR, M. P.; THIAM, A.; DIENG, B. ANFIS and ANN models to predict heliostat tracking errors. **Heliyon**, Cambridge, v. 9, n. 1, jan. 2023. DOI: https://doi.org/10.1016/j.heliyon.2023.e12804. Disponível em: https://www.sciencedirect.com/science/article/pii/S2405844023000117. Acesso em: 4 jan. 2024.

SATTLER, J. C.; RÖGER, M.; SCHWARZBÖZL, P.; BUCK, R.; MACKE, A.; RAEDER, C.; GÖTTSCHE, J. Review of heliostat calibration and tracking control methods. **Solar Energy**, Amsterdam, v. 207, p. 110–132, set. 2020. DOI: https://doi.org/10.1016/j.solener.2020.06.030. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X20306447?via%3Dihub. Acesso em: 20 jun. 2024.

SCHELL, S. Design and evaluation of esolar's heliostat fields. **Solar Energy**, Amsterdam, v. 85, n. 4, p. 614–619, abr. 2011. DOI: https://doi.org/10.1016/j.solener.2010.01.008. Disponível em: https://www.sciencedirect.com/science/article/pii/S0038092X10000216. Acesso em: 30 jul. 2024.

SILVESTRE, A. D. **Desenvolvimento de um heliostato para geração heliotérmica em torres solares**. 2016. Dissertação (Mestrado em sistemas de energias

renováveis) - Universidade Federal de Paraíba, João Pessoa, 2016. Disponível em: https://repositorio.ufpb.br/jspui/handle/tede/8496. Acesso em: 20 jun. 2024.

SOLAR PAYBACK. **Solar thermal energy for industry**. Bonn: Solar Payback, 2017. Disponível em: https://www.solar-payback.com/?lang=pt-br. Acesso em: 5 jan. 2024.

SONG, Y.; HUANG, W. A vision-based fault diagnosis system for heliostats in a central receiver solar power plant. *In*: CONGRESS ON INTELLIGENT CONTROL AND AUTOMATION, 2012, Beijing. **Proceedings** [...]. Beijing: IEEE, 2012. Disponível em: https://ieeexplore.ieee.org/document/6359038. Acesso em: 17 jun. 2024.

STONE, K. W.; JONES, S. A. Analysis of solar two heliostat tracking error sources. *In*: RENEWABLE AND ADVANCED ENERGY SYSTEMS FOR THE 21ST CENTURY, 1999, Huntington Beach. **Proceedings** [...]. Huntington Beach: Boeing Co., 1999. Disponível em: https://www.osti.gov/biblio/3312. Acesso em: 8 jun. 2024.

STONE, K.; LOPEZ, C. Evaluation of the solar one track alignment methodology. *In*: ASME/JSME/JSES INTERNATIONAL SOLAR ENERGY CONFERENCE, 1995, New York. **Proceedings** [...]. New York: American Society of Mechanical Engineers, 1995. p. 521–526. Disponível em: https://www.osti.gov/biblio/113325. Acesso em: 20 jun. 2024.

TEXAS INSTRUMENTS. **DRV8825 Stepper Motor Controller IC**. Dallas: Texas Instruments, 2014. Disponível em: https://www.ti.com/lit/ds/slvsa73f/slvsa73f.pdf. Acesso em: 24 maio 2024.

YOGEV, A.; KRUPKIN, V. **Control of a heliostat field in a solar energy plant**. Depositante: Yeda Research And Development Company Ltd. US 5862799 A. Depósito: 17 jun. 1996. Concessão: 26 jan. 1999. Disponível em: https://patents.google.com/patent/US5862799A/en. Acceso em: 9 ago. 2024.

YOUSIF, C.; QUECEDO, G. O.; SANTOS, J. B. Comparison of solar radiation in Marsaxlokk, Malta and Valladolid, Spain. **Renewable Energy**, [*s. l.*], v. 49, p. 203–206,

2013. DOI: https://doi.org/10.1016/j.renene.2012.01.031. Disponível em: https://www.sciencedirect.com/science/article/pii/S0960148112000420. Acesso em: 9 dez. 2024.

ZAVODNY, M.; SLACK, M.; HUIBREGTSE, R.; SONN, A. Tower-based CSP Artificial Light Calibration System. **Energy Procedia**, [*s. l.*], v. 69, p. 1488–1497, maio 2015. DOI: https://doi.org/10.1016/j.egypro.2015.03.098. Disponível em: https://www.sciencedirect.com/science/article/pii/S187661021500404X. Acesso em: 30 jul. 2024.

ZEGHOUDI, A.; BENMOUIZA, K. Solar Power Heliostat Control Using Image Processing Technology and Artificial Neural Networks. **Journal Europeen des Systemes Automatises**, [*s. l.*], v. 56, n. 1, p. 165–171, fev. 2023. DOI: https://doi.org/10.18280/jesa.560120. Disponível em: https://www.iieta.org/journals/jesa/paper/10.18280/jesa.560120. Acesso em: 9 abr. 2024.

ANEXOS

Anexo 1. Algoritmo de movimentação por malha aberta

```
# ================================================================================
#                              IMPORTAR BIBLIOTECAS
# ================================================================================
import serial
import time
import RPi.GPIO as GPIO
import math
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
from datetime import datetime
import pandas as pd
import cv2
import numpy as np
from datetime import datetime
# ================================================================================
#                                    FUNÇÕES
# ================================================================================
# ..............................................................................
# Função principal para mover os motores para a posição inicial
# ..............................................................................
def setup_motores():
    global firstazi, firstele, bno

    time.sleep(0.25)

    deshabilita_drv8825_1()
    deshabilita_drv8825_2()
    P1_4_1()
    P1_4_2()

    time.sleep(0.5)
    print("Colocando na posição inicial...")
    mover_pos_ini_azi()
    mover_pos_ini_ele()
    print("Foi colocado na posição inicial!")
    time.sleep(5)
# ..............................................................................
# Funções para habilitar e desabilitar motores
# ..............................................................................
def habilita_drv8825_1():
    GPIO.output(ENABLE_1, GPIO.LOW)
    time.sleep(0.01)

def habilita_drv8825_2():
    GPIO.output(ENABLE_2, GPIO.LOW)
    time.sleep(0.01)

def deshabilita_drv8825_1():
    GPIO.output(ENABLE_1, GPIO.HIGH)
    time.sleep(0.01)

def deshabilita_drv8825_2():
    GPIO.output(ENABLE_2, GPIO.HIGH)
```

```
    time.sleep(0.01)
# ..........................................................................
# Funções para mover os motores no sentido horário e anti-horário
# ..........................................................................
def hor_1():
    GPIO.output(DIR_PIN_1, GPIO.HIGH)

def hor_2():
    GPIO.output(DIR_PIN_2, GPIO.HIGH)

def ahor_1():
    GPIO.output(DIR_PIN_1, GPIO.LOW)

def ahor_2():
    GPIO.output(DIR_PIN_2, GPIO.LOW)
# ..........................................................................
# Funções de micropasso para motores
# ..........................................................................
def passo_1():
    GPIO.output(STEP_PIN_1, GPIO.HIGH)
    time.sleep(TIME_FOR_STEP / 1_000_000)
    GPIO.output(STEP_PIN_1, GPIO.LOW)
    time.sleep(TIME_FOR_STEP / 1_000_000)

def passo_2():
    GPIO.output(STEP_PIN_2, GPIO.HIGH)
    time.sleep(TIME_FOR_STEP / 1_000_000)
    GPIO.output(STEP_PIN_2, GPIO.LOW)
    time.sleep(TIME_FOR_STEP / 1_000_000)

def P1_4_1():
    GPIO.output(M0_1, GPIO.LOW)  #low     #low
    GPIO.output(M1_1, GPIO.HIGH) #low     #high
    GPIO.output(M2_1, GPIO.LOW)  #high FOR 16    #low FOR 4

def P1_4_2():
    GPIO.output(M0_2, GPIO.LOW)
    GPIO.output(M1_2, GPIO.HIGH)
    GPIO.output(M2_2, GPIO.LOW)
# ..........................................................................
# Função para verificar se uma string está vazia
# ..........................................................................
def es_cadena_no_vacia(valor):
    # Verifique se o valor é uma string e não está vazio
    return isinstance(valor, str) and bool(valor.strip())
# ..........................................................................
# Função principal para mover os motores de azimute e elevação
# ..........................................................................
def lectura_mov_motores(leeCadena):
    global dato_rx_azi, dato_rx_ele, num_Pulsos_1, num_Pulsos_2, dat_azi,
dat_ele

    if len(leeCadena) > 0:
        mov_azi, mov_ele = leeCadena.split(',')
        dat_azi = float(mov_azi)
        dat_ele = float(mov_ele)

        print(f"{dat_azi}, {dat_ele} Grados")
        time.sleep(0.005)

        if dat_ele < 1:
```

```
            dat_ele = 0
        elif 1 < dat_ele <5:

            dat_ele = 1.03*dat_ele + 1#+ 1.8

        elif 5 < dat_ele:

            dat_ele = 1.03*dat_ele + 1.3#+ 1.8

        dato_rx_azi = round(dat_azi * 10 * 2.2222222)
        dato_rx_ele = round(dat_ele * 10 * 2.2222222)
        print(f"{dato_rx_azi}, {dato_rx_ele} es el dato_rx")

        habilita_drv8825_1()
        if dato_rx_azi > num_Pulsos_1:
            ahor_1()
            while dato_rx_azi > num_Pulsos_1:
                if GPIO.input(PUSHBUTTON_4) == GPIO.LOW:
                    break
                passo_1()
                num_Pulsos_1 += 1
        elif dato_rx_azi < num_Pulsos_1:
            hor_1()
            while dato_rx_azi < num_Pulsos_1:
                if GPIO.input(PUSHBUTTON_5) == GPIO.LOW:
                    break
                passo_1()
                num_Pulsos_1 -= 1

        #deshabilita_drv8825_1()

        habilita_drv8825_2()

        if dato_rx_ele > num_Pulsos_2:
            hor_2()
            while dato_rx_ele > num_Pulsos_2:
                if GPIO.input(PUSHBUTTON_1) == GPIO.LOW:
                    break
                passo_2()
                num_Pulsos_2 += 1
        elif dato_rx_ele < num_Pulsos_2:
            ahor_2()
            while dato_rx_ele < num_Pulsos_2:
                if GPIO.input(PUSHBUTTON_2) == GPIO.LOW:
                    break
                passo_2()
                num_Pulsos_2 -= 1

        #deshabilita_drv8825_2()
        leeCadena = ""
# ..........................................................................
# Função para mover para a posição inicial do azimute
# ..........................................................................
def mover_pos_ini_azi():
    # Ativa micropassos
    P1_4_1()
    # Habilita o driver
    habilita_drv8825_1()
    # Ativar direção de rotação
    hor_1()
    # O “sistema” move-se no sentido anti-horário até atingir o
```

```
    # fim do curso, movimento do motor no sentido anti-horário
    while GPIO.input(PUSHBUTTON_4) == GPIO.HIGH:
        if GPIO.input(PUSHBUTTON_5) == GPIO.LOW:
            break
        # Ativa os passos
        passo_1()

    deshabilita_drv8825_1()
# ..........................................................................
# Função para mover para a posição de elevação inicial
# ..........................................................................
def mover_pos_ini_ele():
    # Ativa micropassos
    P1_4_2()
    # Habilita o driver
    habilita_drv8825_2()
    # Ativar direção de rotação
    ahor_2()
    # O "sistema" move-se no sentido horário até atingir o
    # fim do curso, movimento do motor no sentido horário
    while GPIO.input(PUSHBUTTON_2) == GPIO.HIGH:
        if GPIO.input(PUSHBUTTON_2) == GPIO.LOW:
            break
        # Ativa os passos
        passo_2()

    deshabilita_drv8825_2()
# ..........................................................................
# Função para enviar dados para o Firebase
# ..........................................................................
def upload_data(data):
    ref.push(data)
# ..........................................................................
# Função para converter para formato de hora
# ..........................................................................
def convert(seconds):
    min, sec = divmod(seconds, 60)
    hour, min = divmod(min, 60)
    return "%02d:%02d:%02d" % (hour, min, sec)
# ..........................................................................
# Formato de hora e ômega (tempo angular) em radianos
# ..........................................................................
def Solar_Time(std_time, Lst, Lloc, E):

    nhour = datetime.strptime(std_time, "%H:%M:%S").hour
    nminute = datetime.strptime(std_time, "%H:%M:%S").minute
    nsecond = datetime.strptime(std_time, "%H:%M:%S").second

    hourtominute = nhour*60
    secondtominute = nsecond/60
    total_minute = hourtominute + nminute + secondtominute

    Solar_time_min = total_minute + 4*(Lst - Lloc) + E
    secondfin = Solar_time_min*60
    hour_standard = convert(secondfin)

    nhour_st = datetime.strptime(hour_standard, "%H:%M:%S").hour
    nminute_st = datetime.strptime(hour_standard, "%H:%M:%S").minute
    nsecond_st = datetime.strptime(hour_standard, "%H:%M:%S").second

    mintohour = nminute_st/60
```

```
    sectohour = nsecond_st/3600
    total_hour_st = nhour_st + mintohour + sectohour
    # hora angular, antes (-) ou depois (+) do meio-dia
    omega = (total_hour_st-12) *15
    omega_rad = math.radians(omega)

    return hour_standard,omega_rad
# ..........................................................................
# Função para calcular o dia do ano
# ..........................................................................
def day_of_year (dia, mes):

    if mes == 1:
        n = dia
    elif mes == 2:
        n = 31 + dia
    elif mes == 3:
        n = 59 + dia
    elif mes == 4:
        n = 90 + dia
    elif mes == 5:
        n = 120 + dia
    elif mes == 6:
        n = 151 + dia
    elif mes == 7:
        n = 181 + dia
    elif mes == 8:
        n = 212 + dia
    elif mes == 9:
        n = 243 + dia
    elif mes == 10:
        n = 273 + dia
    elif mes == 11:
        n = 304 + dia
    elif mes == 12:
        n = 334 + dia

    return n
# ..........................................................................
# Função para calcular o dia em um ano bissexto
# ..........................................................................
def day_of_year_bis (dia, mes):

    if mes == 1:
        n = dia
    elif mes == 2:
        n = 31 + dia
    elif mes == 3:
        n = 60 + dia
    elif mes == 4:
        n = 91 + dia
    elif mes == 5:
        n = 121 + dia
    elif mes == 6:
        n = 152 + dia
    elif mes == 7:
        n = 182 + dia
    elif mes == 8:
        n = 213 + dia
    elif mes == 9:
        n = 244 + dia
```

```
    elif mes == 10:
        n = 274 + dia
    elif mes == 11:
        n = 305 + dia
    elif mes == 12:
        n = 335 + dia

    return n
# ..................................................................
# Função para calcular delta
# ..................................................................
def delta_E(n):

    B = math.radians((n -1)*(360/365)) # nth day of the year
    delta_grados = 23.45*math.sin(math.radians(360*(284 + n)/365))
    delta = math.radians(delta_grados) #en radianes
    # longitudes are in degrees west, E in minutes
    E = 229.2*(0.000075 + 0.001868*math.cos(B) - 0.032077*math.sin(B) -
0.014615*math.cos(2*B) - 0.04089*math.sin(2*B))

    return delta, E
# ..................................................................
# Função para calcular elevação e azimute solar
# ..................................................................
def elev_azim(std_time, Lst, Lloc, E, delta, phi):

    # Calcular o tempo solar
    # Lst is the standard meridian for the local time zone,
    # Lloc is the longitude of the location in question
    # longitudes are in degrees west, E in minutes
    # ----------------------------------------------------------------
    Tiempo_solar, omega = Solar_Time(std_time, Lst, Lloc, E)
    # ________________________________________________________________
    #                         Ângulo de incidência
    # ----------------------------------------------------------------
    theta_s = (math.acos(math.cos(phi)*math.cos(delta)*math.cos(omega)
                         + math.sin(phi)*math.sin(delta)))
    # ________________________________________________________________
    #                     Cálculo do ângulo azimute
    # ----------------------------------------------------------------
    beta_s = (math.copysign(math.acos((math.cos(theta_s)*math.sin(phi)
                                       -
math.sin(delta))/(math.sin(theta_s)*math.cos(phi))), omega))
    # ----------------------------------------------------------------
    theta_s_grados = math.degrees(theta_s)
    beta_s_grados = math.degrees(beta_s) +180
    # ________________________________________________________________
    #           Cálculo do ângulo de elevação solar
    # ----------------------------------------------------------------
    alpha_s = 90 - theta_s_grados

    return alpha_s,beta_s_grados, theta_s_grados
# ..................................................................
# Função para calcular elevação e azimute do heliostato
# ..................................................................
def elev_azi_helio(theta_s, beta_s, theta_t, beta_t):
    # convertendo para radianos
    theta_s_rad = math.radians(theta_s)
    beta_s_rad = math.radians(beta_s)
    beta_t_rad = math.radians(beta_t)
```

```
    # Cálculo do ângulo zenital para o heliostato
    theta_h =
math.atan((math.sqrt((math.sin(theta_s_rad)**2)+(math.sin(theta_t)**2)+(2*m
ath.sin(theta_s_rad)*math.sin(theta_t))*math.cos(beta_t_rad-beta_s_rad)))/
(math.cos(theta_s_rad)+math.cos(theta_t)))

    # Cálculo do ângulo azimute para o heliostato
    beta_h = math.atan((math.sin(theta_s_rad)*math.sin(beta_s_rad) +
math.sin(theta_t)*math.sin(beta_t_rad))/
(math.sin(theta_s_rad)*math.cos(beta_s_rad) +
math.sin(theta_t)*math.cos(beta_t_rad)))

    theta_h_grados = math.degrees(theta_h)
    beta_h_grados = math.degrees(beta_h)

    # calculando a elevação do heliostato
    alpha_h = theta_h_grados
    beta_h_f = beta_h_grados +180

    return alpha_h, beta_h_f
# ..........................................................................
# Função para calcular minutos entre duas horas
# ..........................................................................
def cantidad_min(hora, hora_fin):

    nhour = datetime.strptime(hora, "%H:%M:%S").hour
    nminute = datetime.strptime(hora, "%H:%M:%S").minute
    nsecond = datetime.strptime(hora, "%H:%M:%S").second

    hourtominute = nhour*60
    secondtominute = nsecond/60
    total_minute = hourtominute + nminute + secondtominute

    nhourf = datetime.strptime(hora_fin, "%H:%M:%S").hour
    nminutef = datetime.strptime(hora_fin, "%H:%M:%S").minute
    nsecondf= datetime.strptime(hora_fin, "%H:%M:%S").second

    hourtominutef = nhourf*60
    secondtominutef = nsecondf/60
    total_minutef = hourtominutef + nminutef + secondtominutef

    minf = total_minutef - total_minute
    secf = minf*60
    return minf, secf
# ..........................................................................
# Função para converter segundos para formato de hora
# ..........................................................................
def convertsec(seconds):
    min, sec = divmod(seconds, 60)
    hour, min = divmod(min, 60)
    return "%02d:%02d:%02d" % (hour, min, sec)
# ..........................................................................
# Função para determinar se um contorno é um quadrado
# ..........................................................................
def is_square(contour, epsilon=0.04):
    # Aproximar o contorno para verificar se ele tem 4 lados
    approx = cv2.approxPolyDP(contour, epsilon * cv2.arcLength(contour,
True), True)
    if len(approx) == 4:
        return True
    return False
```

```
# ....................................................................
# Função para calcular as coordenadas do sol e do alvo
# ....................................................................
def coordenadas_imagenes(image):
    # Converter imagem em escala de cinza
    imagen_gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calcular o brilho médio
    brillo_promedio = np.mean(imagen_gris)
    # Definir um limite para classificar a imagem
    umbral_1 = 160
    umbral_2 = 75

    # Classifique a imagem
    if brillo_promedio > umbral_1:
        print("A imagem está bem iluminada.")
        # Converter imagem para espaço de cor HSV
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        # Definir a faixa de cores para vermelha no espaço HSV
        lower_red1 = np.array([0, 70, 50])
        upper_red1 = np.array([10, 255, 255])
        lower_red2 = np.array([140, 25, 0])
        upper_red2 = np.array([180, 255, 255])

        # Crie uma máscara para a cor vermelha
        mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
        mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
        mask3 = mask1 + mask2

        lower_red1n = np.array([0, 0, 255])
        upper_red1n = np.array([170, 140, 255])
        lower_red2n = np.array([140, 0, 0])
        upper_red2n = np.array([180, 255, 255])
        mask4 = cv2.inRange(hsv, lower_red1n, upper_red1n)
        mask5 = cv2.inRange(hsv, lower_red2n, upper_red2n)
        mask6 = mask4 + mask5
        #and
        mask = cv2.bitwise_and(mask3, mask6)

        # Crie um kernel quadrado
        kernel1 = np.ones((4, 4), np.uint8)
        # Crie um kernel quadrado
        kernel2 = np.ones((7, 7), np.uint8)
        # Crie um kernel quadrado
        kernel3 = np.ones((12, 12), np.uint8)

        # 1. Abertura ou erosão
        #opened_img = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel1)
        eroded_img = cv2.erode(mask, kernel1, iterations=1)
        # Aplique o filtro médio
        mask_filtered = cv2.medianBlur(eroded_img, 9)

        # 2. Dilatação
        mask_dilated = cv2.dilate(mask_filtered, kernel2, iterations=1)

        # Fechamento
        closed_img = cv2.morphologyEx(mask_dilated, cv2.MORPH_CLOSE,
kernel3)

        # Aplique um desfoque para reduzir o ruído
        blurred = cv2.GaussianBlur(closed_img, (9, 9), 0)
```

```
        # Detectando bordas usando o detector de bordas Canny
        edges = cv2.Canny(blurred, 50, 150)

        # Encontrando contornos em imagem binária
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        # inicializar variáveis para o contorno principal
        max_contour = None
        max_area = 0

        # Iterar sobre todos os contornos
        for contour in contours:
            area = cv2.contourArea(contour)
            # Se a área deste contorno for maior que a área máxima
encontrada
            if area > max_area:
                max_area = area
                max_contour = contour

        # Se um contorno com a maior área fosse encontrado
        if max_contour is not None:
            # Calcular os momentos do contorno principal
            M = cv2.moments(max_contour)

            # Calcular o centroide
            if M["m00"] != 0:
                cXr = int(M["m10"] / M["m00"])
                cYr = int(M["m01"] / M["m00"])
                # Obtém o retângulo delimitador do contorno
                x, y, width, height = cv2.boundingRect(max_contour)
            else:
                cXr, cYr = 0, 0

            # Desenhe o contorno principal e o centroide na imagem original
            cv2.drawContours(image, [max_contour], -1, (0, 255, 0), 2)
            image_copy = image.copy()  # Salvar uma cópia da imagem
original
            cv2.circle(image_copy, (cXr, cYr), 2, (0, 255, 0), -1)  #
Centroide verde
            cv2.putText(image_copy, "X:"+str(cXr)+",Y:"+str(cYr), (cXr+5,
cYr), 2, 1, (0, 255, 0), 0)

    else:
        if brillo_promedio > umbral_2:
            print('a imagem tem brilho médio')
            # Converter imagem para espaço de cor HSV
            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
            # Definir a faixa de cores para vermelha no espaço HSV
            lower_red1 = np.array([0, 255, 50])
            upper_red1 = np.array([180, 255, 255])
            lower_red2 = np.array([130, 70, 0])
            upper_red2 = np.array([180, 255, 255])

            # Crie uma máscara para vermelha
            mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
            mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
            mask3 = mask1 + mask2

            lower_red1n = np.array([0, 0, 255])
```

```
            upper_red1n = np.array([80, 255, 255])
            lower_red2n = np.array([130, 70, 50])
            upper_red2n = np.array([180, 255, 255])

            mask4 = cv2.inRange(hsv, lower_red1n, upper_red1n)
            mask5 = cv2.inRange(hsv, lower_red2n, upper_red2n)
            mask6 = mask4 + mask5

            #and
            mask = cv2.bitwise_and(mask3, mask6)
            #mask = mask3

            # Crie um kernel quadrado
            kernel1 = np.ones((4, 4), np.uint8)
            # Crie um kernel quadrado
            kernel2 = np.ones((9, 9), np.uint8)
            # 1. Abertura ou erosão
            #opened_img = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel1)
            eroded_img = cv2.erode(mask, kernel1, iterations=1)

            # Aplique o filtro médio
            mask_filtered = cv2.medianBlur(eroded_img, 5)

            # 2. Dilatação
            mask_dilated = cv2.dilate(mask_filtered, kernel2, iterations=1)

            # Aplique um desfoque para reduzir o ruído
            blurred = cv2.GaussianBlur(mask_dilated, (9, 9), 0)

            # Detectando bordas usando o detector de bordas Canny
            edges = cv2.Canny(blurred, 50, 150)

            # Encontrando contornos em imagem binária
            contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

            # inicializar variáveis para o contorno principal
            max_contour = None
            max_area = 0

            # Iterar sobre todos os contornos
            for contour in contours:
                area = cv2.contourArea(contour)
                # Se a área deste contorno for maior que a área máxima
encontrada
                if area > max_area:
                    max_area = area
                    max_contour = contour

            # Se um contorno com a maior área fosse encontrado
            if max_contour is not None:
                # Calcular os momentos do contorno principal
                M = cv2.moments(max_contour)

                # Calcular o centroide
                if M["m00"] != 0:
                    cXr = int(M["m10"] / M["m00"])
                    cYr = int(M["m01"] / M["m00"])
                    # Obtém o retângulo delimitador do contorno
                    x, y, width, height = cv2.boundingRect(max_contour)
                else:
```

```
                    cXr, cYr = 0, 0

                # Desenhe o contorno principal e o centroide na imagem
original
                cv2.drawContours(image, [max_contour], -1, (0, 255, 0), 2)
# Contorno verde
                image_copy = image.copy()  # Salvar uma cópia da imagem
original
                cv2.circle(image_copy, (cXr, cYr), 2, (0, 255, 0), -1)  #
Centroide verde
                cv2.putText(image_copy, "X:"+str(cXr)+",Y:"+str(cYr),
(cXr+5, cYr), 2, 1, (0, 255, 0), 0)

        else:

            print("A imagem está escura.")
            # Converter imagem para espaço de cor HSV
            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
            # Definir a faixa de cores para vermelha no espaço HSV
            lower_red1 = np.array([0, 255, 50])
            upper_red1 = np.array([180, 255, 255])
            lower_red2 = np.array([140, 140, 0])
            upper_red2 = np.array([180, 255, 255])

            # Crie uma máscara para vermelha
            mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
            mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
            mask3 = mask1 + mask2

            lower_red1n = np.array([0, 0, 255])
            upper_red1n = np.array([80, 255, 255])
            lower_red2n = np.array([130, 140, 50])
            upper_red2n = np.array([180, 255, 255])

            mask4 = cv2.inRange(hsv, lower_red1n, upper_red1n)
            mask5 = cv2.inRange(hsv, lower_red2n, upper_red2n)
            mask6 = mask4 + mask5

            #and
            mask = cv2.bitwise_and(mask3, mask6)

            # Crie um kernel quadrado
            kernel1 = np.ones((4, 4), np.uint8)
            # Crie um kernel quadrado
            kernel2 = np.ones((9, 9), np.uint8)

            # 1. Abertura ou erosão
            #opened_img = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel1)
            eroded_img = cv2.erode(mask, kernel1, iterations=1)

            # Aplique o filtro médio
            mask_filtered = cv2.medianBlur(eroded_img, 5)

            # 2. Dilatação
            mask_dilated = cv2.dilate(mask_filtered, kernel2, iterations=1)

            # Aplique um desfoque para reduzir o ruído
            blurred = cv2.GaussianBlur(mask_dilated, (9, 9), 0)

            # Detectando bordas usando o detector de bordas Canny
            edges = cv2.Canny(blurred, 50, 150)
```

```
            # Encontrando contornos em imagem binária
            contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

            # inicializar variáveis para o contorno principal
            max_contour = None
            max_area = 0

            # Iterar sobre todos os contornos
            for contour in contours:
                area = cv2.contourArea(contour)
                # Se a área deste contorno for maior que a área máxima
encontrada
                if area > max_area:
                    max_area = area
                    max_contour = contour

            # Se um contorno com a maior área fosse encontrado
            if max_contour is not None:
                # Calcular os momentos do contorno principal
                M = cv2.moments(max_contour)

                # Calcular o centroide
                if M["m00"] != 0:
                    cXr = int(M["m10"] / M["m00"])
                    cYr = int(M["m01"] / M["m00"])
                    # Obtém o retângulo delimitador do contorno
                    x, y, width, height = cv2.boundingRect(max_contour)
                else:
                    cXr, cYr = 0, 0

                # Desenhe o contorno principal e o centroide na imagem
original
                cv2.drawContours(image, [max_contour], -1, (0, 255, 0), 2)
# Contorno verde
                image_copy = image.copy()  # Salvar uma cópia da imagem
original
                cv2.circle(image_copy, (cXr, cYr), 2, (0, 255, 0), -1)  #
Centroide verde
                cv2.putText(image_copy, "X:"+str(cXr)+",Y:"+str(cYr),
(cXr+5, cYr), 2, 1, (0, 255, 0), 0)


    # Supondo que o maior contorno (max_contour) foi encontrado
    # Crie uma máscara com a região delimitada pelo contorno
    mask_new = np.zeros_like(image_copy, dtype=np.uint8)
    cv2.drawContours(mask_new, [max_contour], -1, (255, 255, 255),
thickness=cv2.FILLED)

    # Extraia a região de interesse (ROI) aplicando a máscara
    roi = cv2.bitwise_and(image_copy, mask_new)

    image1= cv2.cvtColor(roi, cv2.COLOR_BGR2RGB)
    # Use the cvtColor() function to grayscale the image
    gray_image = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

    # APLICANDO FILTRO GAUSSIANO PARA REMOÇÃO DE RUÍDO
    #img=cv2.imread("img.jpg")
    gaugaussian=cv2.GaussianBlur(src=gray_image,ksize=(15,15),sigmaX=0)
```

```
    # Set threshold and maxValue
    thresh = 200
    maxValue = 255
    # Basic threshold example
    th, thresh_binary = cv2.threshold(gaugaussian, thresh, maxValue,
cv2.THRESH_BINARY)

    kernel=np.ones((9,9),np.uint8)
    opening=cv2.morphologyEx(thresh_binary,cv2.MORPH_OPEN, kernel)

    contornos,jerarquia=cv2.findContours(opening,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    if len(contornos) == 0:
        print('outro filtro menor')
        # Set threshold and maxValue
        thresh = 150
        maxValue = 255
        # Basic threshold example
        th, thresh_binary = cv2.threshold(gaugaussian, thresh, maxValue,
cv2.THRESH_BINARY)
        kernel=np.ones((9,9),np.uint8)
        opening=cv2.morphologyEx(thresh_binary,cv2.MORPH_OPEN, kernel)

        contornos,jerarquia=cv2.findContours(opening,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        if len(contornos) == 0:
            print('O sistema se moverá em malha aberta')
            #image_copy = image
            #cX = cXr
            #cY = cYr
            mov_malla_abierta()

        else:
            # CÓDIGO PARA ENCONTRAR O CONTORNO PRINCIPAL
            # ------------------------------------------------------------
            l_area = []
            j=0

            for i in range(len(contornos)):
                area= cv2.contourArea(contornos[i])
                l_area.append(area)

            max = l_area[0]
            for i in range(0,len(l_area)):
                if l_area[i] > max:
                    max = l_area[i]
                    j = i

            cont=contornos[j]
            M=cv2.moments(cont)
            cX=int(M["m10"]/M["m00"])
            cY=int(M["m01"]/M["m00"])
            cv2.circle(image_copy,(cX,cY),2,(0,0,0),-1)

cv2.putText(image_copy,"X:"+str(cX)+",Y:"+str(cY),(cX+5,cY),2,0.8,(0,0,0),0
)

            # encontre o círculo envolvente mínimo
            (xc,yc),radius = cv2.minEnclosingCircle(cont)
```

```
            # converter o raio e o centro em um número inteiro
            center = (int(xc),int(yc))
            radius = int(radius)

            # Desenhe o círculo envolvente na imagem
            cv2.circle(image_copy,center,radius,(0,255,0),2)

            # a imagem é 640x480
            xii=image_copy.shape[1]
            yii=image_copy.shape[0]
            xim= 320
            yim= 240

            # Desenhando linhas
            cv2.line(image_copy,(xim,0),(xim,yii),(0,128,255),1)
            cv2.line(image_copy,(0,yim),(xii,yim),(0,128,255),1)

            # desenhando um círculo no centro e colocando coordenadas
            cv2.circle(image_copy,(xim,yim),2,(255,128,0),-1)

cv2.putText(image_copy,"X:"+str(xim)+",Y:"+str(yim),(xim+5,yim+15),2,0.8,(2
55,128,0),0)

            # Salvamos no diretório
            image2= cv2.cvtColor(image_copy, cv2.COLOR_RGB2BGR)


    else:
        # CÓDIGO PARA ENCONTRAR O CONTORNO PRINCIPAL
        # ------------------------------------------------------------
        l_area = []
        j=0

        for i in range(len(contornos)):
            area= cv2.contourArea(contornos[i])
            l_area.append(area)
        #print(l_area)

        max = l_area[0]
        for i in range(0,len(l_area)):
            if l_area[i] > max:
                max = l_area[i]
                j = i

        cont=contornos[j]
        M=cv2.moments(cont)
        cX=int(M["m10"]/M["m00"])
        cY=int(M["m01"]/M["m00"])
        cv2.circle(image_copy,(cX,cY),2,(0,0,0),-1)

cv2.putText(image_copy,"X:"+str(cX)+",Y:"+str(cY),(cX+5,cY),2,0.8,(0,0,0),0
)

        # encontre o círculo envolvente mínimo
        (xc,yc),radius = cv2.minEnclosingCircle(cont)

        # converter o raio e o centro em um número inteiro
        center = (int(xc),int(yc))
        radius = int(radius)

        # Desenhe o círculo envolvente na imagem
```

```
        cv2.circle(image_copy,center,radius,(0,255,0),2)

        # a imagem é 640x480
        xii=image_copy.shape[1]
        yii=image_copy.shape[0]
        xim= 320
        yim= 240

        # Desenhando linhas
        cv2.line(image_copy,(xim,0),(xim,yii),(255,128,0),1)
        cv2.line(image_copy,(0,yim),(xii,yim),(255,128,0),1)

        # desenhando um círculo no centro e colocando coordenadas
        cv2.circle(image_copy,(xim,yim),2,(255,128,0),-1)

cv2.putText(image_copy,"X:"+str(xim)+",Y:"+str(yim),(xim+5,yim+15),2,0.8,(2
55,128,0),0)

        # Salvamos no diretório
        image2= cv2.cvtColor(image_copy, cv2.COLOR_RGB2BGR)


    return cXr, cYr, width, height, cX, cY, image_copy
# ..........................................................................
# Função para calcular a relação entre px/mm
# ..........................................................................
def px_to_mm_m2(px_w,px_h,dif_x,dif_y,ancho_r, altura_r):

    relacion_mm_px_w = ancho_r/px_w
    relacion_mm_px_h = altura_r/px_h
    print('A relação mm/px do segundo método é:
',round(relacion_mm_px_w,2), round(relacion_mm_px_h,2))

    distancia_w = dif_x*relacion_mm_px_w
    distancia_h = dif_y*relacion_mm_px_h

    return distancia_w, distancia_h, relacion_mm_px_w, relacion_mm_px_h
# ..........................................................................
# Função para calcular o ângulo de movimento do motor
# ..........................................................................
def ang_mov(image, tol, dt, R):

    rela = dt/R

    cXr, cYr, width, height, cX, cY, image_with_detections =
coordenadas_imagenes(image)

    coord_receptor = cXr, cYr
    coord_sol = cX, cY
    print('As coordenadas do receptor são: ', coord_receptor)
    print('As coordenadas do sol são: ',coord_sol)

    # Diferenças entre o centro do receptor e o centro do sol (refletor)
    dif_x = coord_receptor[0]-coord_sol[0]
    dif_y = coord_receptor[1]-coord_sol[1]
    #print(dif_x,dif_y)

    dx2, dy2, mm_px_w, mm_px_h = px_to_mm_m2(width, height, dif_x, dif_y,
ancho_r, altura_r)

    # mrad/px
```

```
    betha_x2 = mm_px_w/dt*1000*rela
    betha_y2 = mm_px_h/dt*1000*rela
    print('A relação mrad/px para o sistema é: ',round(betha_x2,2),
round(betha_y2,2))
    print('A relação graus/px para o sistema é:
',round(math.degrees(betha_x2/1000),3),
round(math.degrees(betha_y2/1000),3))

    # mrad
    gamma_x2 = round((dif_x*betha_x2),2)
    gamma_y2 = round((dif_y*betha_y2),2)

    print(' ', gamma_x2)
    print('O erro em x em mrad para o sistema é:: ', gamma_y2)

    erro_x = gamma_x2
    erro_y = gamma_y2
    erro_f = round(math.sqrt(gamma_x2**2 + gamma_y2**2),3)
    print('O erro mrad do sistema é: ', erro_f)
    print('o erro de largura em mm: ',round(dx2,2))
    print('o erro de altura em mm: ',round(dy2,2))

    # o erro em graus que o sistema deve mover é
    mov_x = round(math.degrees(gamma_x2/1000),3)
    mov_y = round(math.degrees(gamma_y2/1000),3)
    print('angulo em graus do sistema x: ',mov_x)
    print('angulo em graus do sistema y: ',mov_y)

    if abs(mov_x) > round(tol*math.degrees(betha_x2/1000),3):
        mov_x = mov_x
    else:
        mov_x = 0

    if abs(mov_y) > round(tol*math.degrees(betha_y2/1000),3):
        mov_y = mov_y
    else:
        mov_y = 0

    print('angulo em graus que o sistema deve se mover em x: ',mov_x)
    print('angulo em graus que o sistema deve se mover em y: ',mov_y)

    return image_with_detections, mov_x, mov_y, dif_x, dif_y, erro_x,
erro_y, erro_f
# ..............................................................................
# Função para salvar a imagem não processada
# ..............................................................................
def save_img(image):
    # Extraímos as informações atuais
    info = datetime.now()
    # Extraímos data
    fecha = info.strftime('%Y_%m_%d')
    # Extraímos hora
    hora = info.strftime('%H_%M_%S')
    # formatar
    fh = f'{fecha}_{hora}'
    fechai = info.strftime('%Y/%m/%d')
    # Extraímos hora
    horai = info.strftime('%H:%M:%S')
    # formatar
    fhi = f'{fechai} {horai}'
```

```
    # Escreva a data e a hora na imagem
    texto = fhi
    # Posição do texto (canto inferior esquerdo)
    posicion = (380, 470)

    # Dimensões do fundo branco
    width = 270  # Largura do fundo
    height = 35  # Alto do fundo
    # Coordenadas do retângulo
    rect_x1, rect_y1 = posicion[0] - 5, posicion[1] - 25
    rect_x2, rect_y2 = rect_x1 + width, rect_y1 + height

    # Desenhe o retângulo branco
    cv2.rectangle(image, (rect_x1, rect_y1), (rect_x2, rect_y2), (255, 255,
255), cv2.FILLED)

    # Escala de fonte e texto
    fuente = cv2.FONT_HERSHEY_SIMPLEX
    escala = 0.7
    color_texto = (0, 0, 0)  # Cor preta para texto
    grosor = 1

    # Adicionar texto à imagem
    cv2.putText(image, texto, posicion, fuente, escala, color_texto,
grosor, cv2.LINE_AA)

    # a imagem é salva em um arquivo

cv2.imwrite('//home/George/Desktop/Imagenes/No_Procesadas/malla_cerrada/Ima
gen_'+str(fh)+ '.jpg' ,image)
# ..............................................................................
# Função para salvar a imagem processada
# ..............................................................................
def save_img_processing(image):
    # Extraímos as informações atuais
    info = datetime.now()
    # Extraímos data
    fecha = info.strftime('%Y_%m_%d')
    # Extraímos hora
    hora = info.strftime('%H_%M_%S')
    # formatar
    fh = f'{fecha}_{hora}'
    fechai = info.strftime('%Y/%m/%d')
    # Extraímos hora
    horai = info.strftime('%H:%M:%S')
    # formatar
    fhi = f'{fechai} {horai}'

    # Escreva a data e a hora na imagem
    texto = fhi
    # Posição do texto (canto inferior esquerdo)
    posicion = (380, 470)

    # Dimensões do fundo branco
    width = 270  # Largura do fundo
    height = 35  # Alto do fundo
    # Coordenadas do retângulo
    rect_x1, rect_y1 = posicion[0] - 5, posicion[1] - 25
    rect_x2, rect_y2 = rect_x1 + width, rect_y1 + height

    # Desenhe o retângulo branco
```

```
    cv2.rectangle(image, (rect_x1, rect_y1), (rect_x2, rect_y2), (255, 255,
255), cv2.FILLED)

    # Escala de fonte e texto
    fuente = cv2.FONT_HERSHEY_SIMPLEX
    escala = 0.7
    color_texto = (0, 0, 0)  # Cor preta para texto
    grosor = 1

    # Adicionar texto à imagem
    cv2.putText(image, texto, posicion, fuente, escala, color_texto,
grosor, cv2.LINE_AA)

    # a imagem é salva em um arquivo

cv2.imwrite('//home/George/Desktop/Imagenes/Procesadas/malla_cerrada/Imagen
_'+str(fh)+ '.jpg' ,image)
# ..............................................................................
# Função para descartar quadros não processados
# ..............................................................................
def clear_buffer(cap, frames=5):
    for i in range(frames):
        cap.grab()  # Descartar quadros não processados
# ==============================================================================
#                         DECLARAÇÃO DE VARIÁVEIS
# ==============================================================================
# Configuração de pinos GPIO
ENABLE_1 = 18
M0_1 = 23
M1_1 = 24
M2_1 = 25
STEP_PIN_1 = 8
DIR_PIN_1 = 7

ENABLE_2 = 12
M0_2 = 16
M1_2 = 20
M2_2 = 21
STEP_PIN_2 = 4
DIR_PIN_2 = 17

TIME_FOR_STEP = 2000

PUSHBUTTON_1 = 27
PUSHBUTTON_2 = 22
PUSHBUTTON_4 = 10
PUSHBUTTON_5 = 9

# Inicialização GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup([ENABLE_1, M0_1, M1_1, M2_1, STEP_PIN_1, DIR_PIN_1, ENABLE_2,
M0_2, M1_2, M2_2, STEP_PIN_2, DIR_PIN_2], GPIO.OUT)
GPIO.setup([PUSHBUTTON_1, PUSHBUTTON_2, PUSHBUTTON_4, PUSHBUTTON_5],
GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Variáveis globais
dato_rx_azi = 0
dato_rx_ele = 0
leeCadena = ""
valor = ""
```

```
num_Pulsos_1 = 0
num_Pulsos_2 = 0
dat_azi = 0.0
dat_ele = 0.0

isFirstValueSaved = False
funcionEjecutada = False
firstazi = 0.0
firstele = 0.0
faz = 0.0
fel = 0.0
azi = 0.0
ele = 0.0

lista =[]

# Dimensões do receptor em mm
ancho_r=214
altura_r=190
# tolerância do número de pixels
tol = 1

erro_x=0
erro_y=0
erro_f=0
dif_x = 0
dif_y = 0

now = datetime.now()
dia= now.day
mes = now.month
n = day_of_year(dia, mes)
GMT = 3 # Greenwich Mean Time, Fuso horário -3 para o Brasil
Lloc= 48.885800 # Longitude do local em graus, -49.03479832
phi = math.radians(-23.981578) # Latitude -22.35102511
Lst = GMT*15 # 15 deg/h
# Azimute do receptor/torre em graus
beta_t = 322.5
# R = RÁDIO ENTRE O HELIOSTATO E O RECEPTOR
# H = ALTURA ENTRE O HELIOSTATO E O RECEPTOR
R = 9580
H = 1283
theta_t= math.atan2(R, H)
# Calculo delta e E
delta, E = delta_E(n)
# ================================================================================
# Fim da declaração de variável
# ================================================================================
# ................................................................................
# INÍCIO DO SISTEMA
# ................................................................................
# Coloque o sistema na posição inicial
setup_motores()

# Caminho para o arquivo de configuração baixado do Firebase Console
cred = credentials.Certificate('/xxxxxxxxxxxxx.json')

# Inicializar o aplicativo Firebase (URL do projeto)
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://xxxxxxxxxxxxx.firebaseio.com'
})
```

```
# Referência ao banco de dados
ref = db.reference('/malla_cerrada/')

# Obter a hora atual
hora_actual = datetime.now().time()
# Formatar hora atual
horaa = hora_actual.strftime("%H:%M:%S")

nhour = datetime.strptime(horaa, "%H:%M:%S").hour
nminute = datetime.strptime(horaa, "%H:%M:%S").minute
nsecond = datetime.strptime(horaa, "%H:%M:%S").second

hourtominute = nhour*60
secondtominute = nsecond/60
total_minute = hourtominute + nminute + secondtominute

nminew = total_minute
nsec = nminew*60

h_fin = convertsec(nsec)
print(h_fin)

# Cálculo do azimute e elevação do sol
alpha_s, azim_s, theta_s = elev_azim(h_fin, Lst, Lloc, E, delta, phi)
# cálculo de azimute e elevação do heliostato
elev_helio, azim_helio = elev_azi_helio(theta_s,azim_s, theta_t, beta_t)

# Arredondar para 2 casas decimais
mov_azim = round(azim_helio,3)
mov_elev = round(elev_helio,3)

angulos_teoricos = str(mov_azim)+str(',')+str(mov_elev)

lectura_mov_motores(angulos_teoricos)
# tempo de espera
time.sleep(2)

now = datetime.now()
fech = now.strftime('%Y-%m-%d ')
hor = h_fin
f_h = f"{fech}{hor}"
# enviando dados para o firebase
data_to_upload =
f"{f_h},{mov_azim},{mov_elev},{dif_x},{dif_y},{erro_x},{erro_y},{erro_f}"
print(data_to_upload)
upload_data(data_to_upload)

# Inicializar a câmera
cap = cv2.VideoCapture(0)


try:
    while True:

        # Obter a hora atual
        hora_actual = datetime.now().time()
        # Formatar hora atual
        horaa = hora_actual.strftime("%H:%M:%S")

        nhour = datetime.strptime(horaa, "%H:%M:%S").hour
```

```
        nminute = datetime.strptime(horaa, "%H:%M:%S").minute
        nsecond = datetime.strptime(horaa, "%H:%M:%S").second

        hourtominute = nhour*60
        secondtominute = nsecond/60
        total_minute = hourtominute + nminute + secondtominute

        nminew = total_minute
        nsec = nminew*60

        h_fin = convertsec(nsec)
        # Cálculo do azimute e elevação do sol
        alpha_s, azim_s, theta_s = elev_azim(h_fin, Lst, Lloc, E, delta,
phi)
        # cálculo de azimute e elevação do heliostato
        elev_helio, azim_helio = elev_azi_helio(theta_s,azim_s, theta_t,
beta_t)

        # Arredondar para 2 casas decimais
        mov_azim = round(azim_helio,3)
        mov_elev = round(elev_helio,3)

        angulos_teoricos = str(mov_azim)+str(',')+str(mov_elev)
        # o sistema se move de acordo com os ângulos
        lectura_mov_motores(angulos_teoricos)

        # Captura da imagem
        image = None
        while image is None:
            # Captura da imagem
            clear_buffer(cap)
            ret, image = cap.read()  # Captura de frame
            if not ret:
                print("Error al capturar la imagen. Reintentando...")
                time.sleep(0.5)  # Aguarde um pouco antes de tentar
novamente.

        #cópia da imagem
        img_c = image.copy()

        # detecta as coordenadas de imagem e movimento do sistema
        image_with_detections, mov_x, mov_y, dif_x, dif_y, erro_x, erro_y,
erro_f = ang_mov(image, tol, dt, R)

        # Mova o sistema de acordo com os ângulos obtidos
        angulos_procesamiento, mov_azim, mov_elev = mov_sistema(mov_azim,
mov_elev, mov_x, mov_y)

        # salvar a imagem capturada
        save_img(img_c)
        # salvar a imagem procesada
        save_img_processing(image_with_detections)

        now = datetime.now()
        fech = now.strftime('%Y-%m-%d %H:%M:%S')
        # enviando dados para o firebase
        data_to_upload =
f"{fech},{mov_azim},{mov_elev},{dif_x},{dif_y},{erro_x},{erro_y},{erro_f}"
        print(data_to_upload)
        upload_data(data_to_upload)
```

```
        time.sleep(25)

except KeyboardInterrupt:
    #print("Ctrl+C foi pressionado. Saindo do loop.")
    setup_motores()
    deshabilita_drv8825_1
    deshabilita_drv8825_2
    cap.release()
```

Anexo 2. Algoritmo de movimentação híbrida

```
"""
                    ALGORITMO DE MOVIMENTAÇAO HIBRIDA
                        BY: GEORGE ORBEZO ALVAREZ

"""
# ============================================================================
#                             IMPORTAR BIBLIOTECAS
# ============================================================================
import serial
import time
import RPi.GPIO as GPIO
import math
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
from datetime import datetime
import pandas as pd
import cv2
import numpy as np
from datetime import datetime
# ============================================================================
#                                   FUNÇÕES
# ============================================================================
# ............................................................................
# Função principal para mover os motores para a posição inicial
# ............................................................................
def setup_motores():
    global firstazi, firstele, bno

    time.sleep(0.25)

    deshabilita_drv8825_1()
    deshabilita_drv8825_2()
    P1_4_1()
    P1_4_2()

    time.sleep(0.5)
    print("Colocando na posição inicial...")
    mover_pos_ini_azi()
    mover_pos_ini_ele()
    print("Foi colocado na posição inicial!")
    time.sleep(5)
# ............................................................................
# Funções para habilitar e desabilitar motores
# ............................................................................
def habilita_drv8825_1():
    GPIO.output(ENABLE_1, GPIO.LOW)
    time.sleep(0.01)

def habilita_drv8825_2():
    GPIO.output(ENABLE_2, GPIO.LOW)
    time.sleep(0.01)

def deshabilita_drv8825_1():
    GPIO.output(ENABLE_1, GPIO.HIGH)
    time.sleep(0.01)
```

```
def deshabilita_drv8825_2():
    GPIO.output(ENABLE_2, GPIO.HIGH)
    time.sleep(0.01)
# ..............................................................................
# Funções para mover os motores no sentido horário e anti-horário
# ..............................................................................
def hor_1():
    GPIO.output(DIR_PIN_1, GPIO.HIGH)

def hor_2():
    GPIO.output(DIR_PIN_2, GPIO.HIGH)

def ahor_1():
    GPIO.output(DIR_PIN_1, GPIO.LOW)

def ahor_2():
    GPIO.output(DIR_PIN_2, GPIO.LOW)
# ..............................................................................
# Funções de micropasso para motores
# ..............................................................................
def passo_1():
    GPIO.output(STEP_PIN_1, GPIO.HIGH)
    time.sleep(TIME_FOR_STEP / 1_000_000)
    GPIO.output(STEP_PIN_1, GPIO.LOW)
    time.sleep(TIME_FOR_STEP / 1_000_000)

def passo_2():
    GPIO.output(STEP_PIN_2, GPIO.HIGH)
    time.sleep(TIME_FOR_STEP / 1_000_000)
    GPIO.output(STEP_PIN_2, GPIO.LOW)
    time.sleep(TIME_FOR_STEP / 1_000_000)

def P1_4_1():
    GPIO.output(M0_1, GPIO.LOW)  #low     #low
    GPIO.output(M1_1, GPIO.HIGH) #low     #high
    GPIO.output(M2_1, GPIO.LOW)  #high FOR 16    #low FOR 4

def P1_4_2():
    GPIO.output(M0_2, GPIO.LOW)
    GPIO.output(M1_2, GPIO.HIGH)
    GPIO.output(M2_2, GPIO.LOW)
# ..............................................................................
# Função para verificar se uma string está vazia
# ..............................................................................
def es_cadena_no_vacia(valor):
    # Verifique se o valor é uma string e não está vazio
    return isinstance(valor, str) and bool(valor.strip())
# ..............................................................................
# Função principal para mover os motores de azimute e elevação
# ..............................................................................
def lectura_mov_motores(leeCadena):
    global dato_rx_azi, dato_rx_ele, num_Pulsos_1, num_Pulsos_2, dat_azi,
dat_ele

    if len(leeCadena) > 0:
        mov_azi, mov_ele = leeCadena.split(',')
        dat_azi = float(mov_azi)
        dat_ele = float(mov_ele)

        print(f"{dat_azi}, {dat_ele} Grados")
        time.sleep(0.005)
```

```
        if dat_ele < 1:
            dat_ele = 0
        elif 1 < dat_ele <5:

            dat_ele = 1.03*dat_ele + 1#+ 1.8

        elif 5 < dat_ele :

            dat_ele = 1.03*dat_ele + 1.3#+ 1.8

        dato_rx_azi = round(dat_azi * 10 * 2.2222222)
        dato_rx_ele = round(dat_ele * 10 * 2.2222222)
        print(f"{dato_rx_azi}, {dato_rx_ele} es el dato_rx")

        habilita_drv8825_1()
        if dato_rx_azi > num_Pulsos_1:
            ahor_1()
            while dato_rx_azi > num_Pulsos_1:
                if GPIO.input(PUSHBUTTON_4) == GPIO.LOW:
                    break
                passo_1()
                num_Pulsos_1 += 1
        elif dato_rx_azi < num_Pulsos_1:
            hor_1()
            while dato_rx_azi < num_Pulsos_1:
                if GPIO.input(PUSHBUTTON_5) == GPIO.LOW:
                    break
                passo_1()
                num_Pulsos_1 -= 1

        #deshabilita_drv8825_1()

        habilita_drv8825_2()

        if dato_rx_ele > num_Pulsos_2:
            hor_2()
            while dato_rx_ele > num_Pulsos_2:
                if GPIO.input(PUSHBUTTON_1) == GPIO.LOW:
                    break
                passo_2()
                num_Pulsos_2 += 1
        elif dato_rx_ele < num_Pulsos_2:
            ahor_2()
            while dato_rx_ele < num_Pulsos_2:
                if GPIO.input(PUSHBUTTON_2) == GPIO.LOW:
                    break
                passo_2()
                num_Pulsos_2 -= 1

        #deshabilita_drv8825_2()
        leeCadena = ""
# ..........................................................................
# Função para mover para a posição inicial do azimute
# ..........................................................................
def mover_pos_ini_azi():
    # Ativa micropassos
    P1_4_1()
    # Habilita o driver
    habilita_drv8825_1()
    # Ativar direção de rotação
```

```
    hor_1()
    # O "sistema" move-se no sentido anti-horário até atingir o
    # fim do curso, movimento do motor no sentido anti-horário
    while GPIO.input(PUSHBUTTON_4) == GPIO.HIGH:
        if GPIO.input(PUSHBUTTON_5) == GPIO.LOW:
            break
        # Ativa os passos
        passo_1()

    deshabilita_drv8825_1()
# ..........................................................................
# Função para mover para a posição de elevação inicial
# ..........................................................................
def mover_pos_ini_ele():
    # Ativa micropassos
    P1_4_2()
    # Habilita o driver
    habilita_drv8825_2()
    # Ativar direção de rotação
    ahor_2()
    # O "sistema" move-se no sentido horário até atingir o
    # fim do curso, movimento do motor no sentido horário
    while GPIO.input(PUSHBUTTON_2) == GPIO.HIGH:
        if GPIO.input(PUSHBUTTON_2) == GPIO.LOW:
            break
        # Ativa os passos
        passo_2()

    deshabilita_drv8825_2()
# ..........................................................................
# Função para enviar dados para o Firebase
# ..........................................................................
def upload_data(data):
    ref.push(data)
# ..........................................................................
# Função para converter para formato de hora
# ..........................................................................
def convert(seconds):
    min, sec = divmod(seconds, 60)
    hour, min = divmod(min, 60)
    return "%02d:%02d:%02d" % (hour, min, sec)
# ..........................................................................
# Formato de hora e ômega (tempo angular) em radianos
# ..........................................................................
def Solar_Time(std_time, Lst, Lloc, E):

    nhour = datetime.strptime(std_time, "%H:%M:%S").hour
    nminute = datetime.strptime(std_time, "%H:%M:%S").minute
    nsecond = datetime.strptime(std_time, "%H:%M:%S").second

    hourtominute = nhour*60
    secondtominute = nsecond/60
    total_minute = hourtominute + nminute + secondtominute

    Solar_time_min = total_minute + 4*(Lst - Lloc) + E
    secondfin = Solar_time_min*60
    hour_standard = convert(secondfin)

    nhour_st = datetime.strptime(hour_standard, "%H:%M:%S").hour
    nminute_st = datetime.strptime(hour_standard, "%H:%M:%S").minute
    nsecond_st = datetime.strptime(hour_standard, "%H:%M:%S").second
```

```
    mintohour = nminute_st/60
    sectohour = nsecond_st/3600
    total_hour_st = nhour_st + mintohour + sectohour
    # hora angular, antes (-) ou depois (+) do meio-dia
    omega = (total_hour_st-12) *15
    omega_rad = math.radians(omega)

    return hour_standard,omega_rad
# ..........................................................................
# Função para calcular o dia do ano
# ..........................................................................
def day_of_year (dia, mes):

    if mes == 1:
        n = dia
    elif mes == 2:
        n = 31 + dia
    elif mes == 3:
        n = 59 + dia
    elif mes == 4:
        n = 90 + dia
    elif mes == 5:
        n = 120 + dia
    elif mes == 6:
        n = 151 + dia
    elif mes == 7:
        n = 181 + dia
    elif mes == 8:
        n = 212 + dia
    elif mes == 9:
        n = 243 + dia
    elif mes == 10:
        n = 273 + dia
    elif mes == 11:
        n = 304 + dia
    elif mes == 12:
        n = 334 + dia

    return n
# ..........................................................................
# Função para calcular o dia em um ano bissexto
# ..........................................................................
def day_of_year_bis (dia, mes):

    if mes == 1:
        n = dia
    elif mes == 2:
        n = 31 + dia
    elif mes == 3:
        n = 60 + dia
    elif mes == 4:
        n = 91 + dia
    elif mes == 5:
        n = 121 + dia
    elif mes == 6:
        n = 152 + dia
    elif mes == 7:
        n = 182 + dia
    elif mes == 8:
        n = 213 + dia
```

```
    elif mes == 9:
        n = 244 + dia
    elif mes == 10:
        n = 274 + dia
    elif mes == 11:
        n = 305 + dia
    elif mes == 12:
        n = 335 + dia

    return n
# ..........................................................................
# Função para calcular delta
# ..........................................................................
def delta_E(n):

    B = math.radians((n -1)*(360/365)) # nth day of the year
    delta_grados = 23.45*math.sin(math.radians(360*(284 + n)/365))
    delta = math.radians(delta_grados) #en radianes
    # longitudes are in degrees west, E in minutes
    E = 229.2*(0.000075 + 0.001868*math.cos(B) - 0.032077*math.sin(B) -
0.014615*math.cos(2*B) - 0.04089*math.sin(2*B))

    return delta, E
# ..........................................................................
# Função para calcular elevação e azimute solar
# ..........................................................................
def elev_azim(std_time, Lst, Lloc, E, delta, phi):

    # Calcular o tempo solar
    # Lst is the standard meridian for the local time zone,
    # Lloc is the longitude of the location in question
    # longitudes are in degrees west, E in minutes
    # ----------------------------------------------------------------------
    Tiempo_solar, omega = Solar_Time(std_time, Lst, Lloc, E)
    # ______________________________________________________________________
    #                          Ângulo de incidência
    # ----------------------------------------------------------------------
    theta_s = (math.acos(math.cos(phi)*math.cos(delta)*math.cos(omega)
                         + math.sin(phi)*math.sin(delta)))
    # ______________________________________________________________________
    #                      Cálculo do ângulo azimute
    # ----------------------------------------------------------------------
    beta_s = (math.copysign(math.acos((math.cos(theta_s)*math.sin(phi)
                                       -
math.sin(delta))/(math.sin(theta_s)*math.cos(phi))), omega))
    # ----------------------------------------------------------------------
    theta_s_grados = math.degrees(theta_s)
    beta_s_grados = math.degrees(beta_s) +180
    # ______________________________________________________________________
    #           Cálculo do ângulo de elevação solar
    # ----------------------------------------------------------------------
    alpha_s = 90 - theta_s_grados

    return alpha_s,beta_s_grados, theta_s_grados
# ..........................................................................
# Função para calcular elevação e azimute do heliostato
# ..........................................................................
def elev_azi_helio(theta_s, beta_s, theta_t, beta_t):
    # convertendo para radianos
    theta_s_rad = math.radians(theta_s)
    beta_s_rad = math.radians(beta_s)
```

```
    beta_t_rad = math.radians(beta_t)

    # Cálculo do ângulo zenital para o heliostato
    theta_h =
math.atan((math.sqrt((math.sin(theta_s_rad)**2)+(math.sin(theta_t)**2)+(2*m
ath.sin(theta_s_rad)*math.sin(theta_t))*math.cos(beta_t_rad-beta_s_rad)))/
(math.cos(theta_s_rad)+math.cos(theta_t)))

    # Cálculo do ângulo azimute para o heliostato
    beta_h = math.atan((math.sin(theta_s_rad)*math.sin(beta_s_rad) +
math.sin(theta_t)*math.sin(beta_t_rad))/
(math.sin(theta_s_rad)*math.cos(beta_s_rad) +
math.sin(theta_t)*math.cos(beta_t_rad)))

    theta_h_grados = math.degrees(theta_h)
    beta_h_grados = math.degrees(beta_h)

    # calculando a elevação do heliostato
    alpha_h = theta_h_grados
    beta_h_f = beta_h_grados +180

    return alpha_h, beta_h_f
# ..........................................................................
# Função para calcular minutos entre duas horas
# ..........................................................................
def cantidad_min(hora, hora_fin):

    nhour = datetime.strptime(hora, "%H:%M:%S").hour
    nminute = datetime.strptime(hora, "%H:%M:%S").minute
    nsecond = datetime.strptime(hora, "%H:%M:%S").second

    hourtominute = nhour*60
    secondtominute = nsecond/60
    total_minute = hourtominute + nminute + secondtominute

    nhourf = datetime.strptime(hora_fin, "%H:%M:%S").hour
    nminutef = datetime.strptime(hora_fin, "%H:%M:%S").minute
    nsecondf= datetime.strptime(hora_fin, "%H:%M:%S").second

    hourtominutef = nhourf*60
    secondtominutef = nsecondf/60
    total_minutef = hourtominutef + nminutef + secondtominutef

    minf = total_minutef - total_minute
    secf = minf*60
    return minf, secf
# ..........................................................................
# Função para converter segundos para formato de hora
# ..........................................................................
def convertsec(seconds):
    min, sec = divmod(seconds, 60)
    hour, min = divmod(min, 60)
    return "%02d:%02d:%02d" % (hour, min, sec)
# ..........................................................................
# Função para determinar se um contorno é um quadrado
# ..........................................................................
def is_square(contour, epsilon=0.04):
    # Aproximar o contorno para verificar se ele tem 4 lados
    approx = cv2.approxPolyDP(contour, epsilon * cv2.arcLength(contour,
True), True)
    if len(approx) == 4:
```

```
        return True
    return False
# ..........................................................................
# Função para calcular as coordenadas do sol e do receptor
# ..........................................................................
def coordenadas_imagenes(image):
    # Converter imagem em escala de cinza
    imagen_gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calcular o brilho médio
    brillo_promedio = np.mean(imagen_gris)
    # Definir um limite para classificar a imagem
    umbral_1 = 160
    umbral_2 = 75

    # Classifique a imagem
    if brillo_promedio > umbral_1:
        print("A imagem está bem iluminada.")
        # Converter imagem para espaço de cor HSV
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        # Definir a faixa de cores para vermelha no espaço HSV
        lower_red1 = np.array([0, 70, 50])
        upper_red1 = np.array([10, 255, 255])
        lower_red2 = np.array([140, 25, 0])
        upper_red2 = np.array([180, 255, 255])

        # Crie uma máscara para vermelha
        mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
        mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
        mask3 = mask1 + mask2

        lower_red1n = np.array([0, 0, 255])
        upper_red1n = np.array([170, 140, 255])
        lower_red2n = np.array([140, 0, 0])
        upper_red2n = np.array([180, 255, 255])
        mask4 = cv2.inRange(hsv, lower_red1n, upper_red1n)
        mask5 = cv2.inRange(hsv, lower_red2n, upper_red2n)
        mask6 = mask4 + mask5
        #and
        mask = cv2.bitwise_and(mask3, mask6)

        # Crie um kernel quadrado
        kernel1 = np.ones((4, 4), np.uint8)
        # Crie um kernel quadrado
        kernel2 = np.ones((7, 7), np.uint8)
        # Crie um kernel quadrado
        kernel3 = np.ones((12, 12), np.uint8)

        # 1. Abertura ou erosão
        #opened_img = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel1)
        eroded_img = cv2.erode(mask, kernel1, iterations=1)
        # Aplique o filtro médio
        mask_filtered = cv2.medianBlur(eroded_img, 9)

        # 2. Dilatação
        mask_dilated = cv2.dilate(mask_filtered, kernel2, iterations=1)

        # Fechamento
        closed_img = cv2.morphologyEx(mask_dilated, cv2.MORPH_CLOSE,
kernel3)
```

```
        # Aplique um desfoque para reduzir o ruído
        blurred = cv2.GaussianBlur(closed_img, (9, 9), 0)

        # Detectando bordas usando o detector de bordas Canny
        edges = cv2.Canny(blurred, 50, 150)

        # Encontrando contornos em imagem binária
        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        # inicializar variáveis para o contorno principal
        max_contour = None
        max_area = 0

        # Iterar sobre todos os contornos
        for contour in contours:
            area = cv2.contourArea(contour)
            # Se a área deste contorno for maior que a área máxima
encontrada
            if area > max_area:
                max_area = area
                max_contour = contour

        # Se um contorno com a maior área fosse encontrado
        if max_contour is not None:
            # Calcular os momentos do contorno principal
            M = cv2.moments(max_contour)

            # Calcular o centroide
            if M["m00"] != 0:
                cXr = int(M["m10"] / M["m00"])
                cYr = int(M["m01"] / M["m00"])
                # Obtém o retângulo delimitador do contorno
                x, y, width, height = cv2.boundingRect(max_contour)
            else:
                cXr, cYr = 0, 0

            # Desenhe o contorno principal e o centroide na imagem original
            cv2.drawContours(image, [max_contour], -1, (0, 255, 0), 2)
            image_copy = image.copy()  # Salvar uma cópia da imagem
original
            cv2.circle(image_copy, (cXr, cYr), 2, (0, 255, 0), -1)  #
Centroide verde
            cv2.putText(image_copy, "X:"+str(cXr)+",Y:"+str(cYr), (cXr+5,
cYr), 2, 1, (0, 255, 0), 0)

    else:
        if brillo_promedio > umbral_2:
            print('a imagem tem brilho médio')
            # Converter imagem para espaço de cor HSV
            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
            # Definir a faixa de cores para vermelha no espaço HSV
            lower_red1 = np.array([0, 255, 50])
            upper_red1 = np.array([180, 255, 255])
            lower_red2 = np.array([130, 70, 0])
            upper_red2 = np.array([180, 255, 255])

            # Crie uma máscara para vermelha
            mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
            mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
            mask3 = mask1 + mask2
```

```
            lower_red1n = np.array([0, 0, 255])
            upper_red1n = np.array([80, 255, 255])
            lower_red2n = np.array([130, 70, 50])
            upper_red2n = np.array([180, 255, 255])

            mask4 = cv2.inRange(hsv, lower_red1n, upper_red1n)
            mask5 = cv2.inRange(hsv, lower_red2n, upper_red2n)
            mask6 = mask4 + mask5

            #and
            mask = cv2.bitwise_and(mask3, mask6)
            #mask = mask3

            # Crie um kernel quadrado
            kernel1 = np.ones((4, 4), np.uint8)
            # Crie um kernel quadrado
            kernel2 = np.ones((9, 9), np.uint8)
            # 1. Abertura ou erosão
            #opened_img = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel1)
            eroded_img = cv2.erode(mask, kernel1, iterations=1)

            # Aplique o filtro médio
            mask_filtered = cv2.medianBlur(eroded_img, 5)

            # 2. Dilatação
            mask_dilated = cv2.dilate(mask_filtered, kernel2, iterations=1)

            # Aplique um desfoque para reduzir o ruído
            blurred = cv2.GaussianBlur(mask_dilated, (9, 9), 0)

            # Detectando bordas usando o detector de bordas Canny
            edges = cv2.Canny(blurred, 50, 150)

            # Encontrando contornos em imagem binária
            contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

            # inicializar variáveis para o contorno principal
            max_contour = None
            max_area = 0

            # Iterar sobre todos os contornos
            for contour in contours:
                area = cv2.contourArea(contour)
                # Se a área deste contorno for maior que a área máxima
encontrada
                if area > max_area:
                    max_area = area
                    max_contour = contour

            # Se um contorno com a maior área fosse encontrado
            if max_contour is not None:
                # Calcular os momentos do contorno principal
                M = cv2.moments(max_contour)

                # Calcular o centroide
                if M["m00"] != 0:
                    cXr = int(M["m10"] / M["m00"])
                    cYr = int(M["m01"] / M["m00"])
                    # Obtém o retângulo delimitador do contorno
```

```
                    x, y, width, height = cv2.boundingRect(max_contour)
                else:
                    cXr, cYr = 0, 0

                # Desenhe o contorno principal e o centroide na imagem
original
                cv2.drawContours(image, [max_contour], -1, (0, 255, 0), 2)
# Contorno verde
                image_copy = image.copy()  # Salvar uma cópia da imagem
original
                cv2.circle(image_copy, (cXr, cYr), 2, (0, 255, 0), -1)  #
Centroide verde
                cv2.putText(image_copy, "X:"+str(cXr)+",Y:"+str(cYr),
(cXr+5, cYr), 2, 1, (0, 255, 0), 0)

        else:

            print("A imagem está escura.")
            # Converter imagem para espaço de cor HSV
            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
            # Definir a faixa de cores para vermelha no espaço HSV
            lower_red1 = np.array([0, 255, 50])
            upper_red1 = np.array([180, 255, 255])
            lower_red2 = np.array([140, 140, 0])
            upper_red2 = np.array([180, 255, 255])

            # Crie uma máscara para vermelha
            mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
            mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
            mask3 = mask1 + mask2

            lower_red1n = np.array([0, 0, 255])
            upper_red1n = np.array([80, 255, 255])
            lower_red2n = np.array([130, 140, 50])
            upper_red2n = np.array([180, 255, 255])

            mask4 = cv2.inRange(hsv, lower_red1n, upper_red1n)
            mask5 = cv2.inRange(hsv, lower_red2n, upper_red2n)
            mask6 = mask4 + mask5

            #and
            mask = cv2.bitwise_and(mask3, mask6)

            # Crie um kernel quadrado
            kernel1 = np.ones((4, 4), np.uint8)
            # Crie um kernel quadrado
            kernel2 = np.ones((9, 9), np.uint8)

            # 1. Abertura ou erosão
            #opened_img = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel1)
            eroded_img = cv2.erode(mask, kernel1, iterations=1)

            # Aplique o filtro médio
            mask_filtered = cv2.medianBlur(eroded_img, 5)

            # 2. Dilatação
            mask_dilated = cv2.dilate(mask_filtered, kernel2, iterations=1)

            # Aplique um desfoque para reduzir o ruído
            blurred = cv2.GaussianBlur(mask_dilated, (9, 9), 0)
```

```
            # Detectando bordas usando o detector de bordas Canny
            edges = cv2.Canny(blurred, 50, 150)

            # Encontrando contornos em imagem binária
            contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

            # inicializar variáveis para o contorno principal
            max_contour = None
            max_area = 0

            # Iterar sobre todos os contornos
            for contour in contours:
                area = cv2.contourArea(contour)
                # Se a área deste contorno for maior que a área máxima
encontrada
                if area > max_area:
                    max_area = area
                    max_contour = contour

            # Se um contorno com a maior área fosse encontrado
            if max_contour is not None:
                # Calcular os momentos do contorno principal
                M = cv2.moments(max_contour)

                # Calcular o centroide
                if M["m00"] != 0:
                    cXr = int(M["m10"] / M["m00"])
                    cYr = int(M["m01"] / M["m00"])
                    # Obtém o retângulo delimitador do contorno
                    x, y, width, height = cv2.boundingRect(max_contour)
                else:
                    cXr, cYr = 0, 0

                # Desenhe o contorno principal e o centroide na imagem
original
                cv2.drawContours(image, [max_contour], -1, (0, 255, 0), 2)
# Contorno verde
                image_copy = image.copy()  # Salvar uma cópia da imagem
original
                cv2.circle(image_copy, (cXr, cYr), 2, (0, 255, 0), -1)  #
Centroide verde
                cv2.putText(image_copy, "X:"+str(cXr)+",Y:"+str(cYr),
(cXr+5, cYr), 2, 1, (0, 255, 0), 0)


    # Supondo que o maior contorno (max_contour) foi encontrado
    # Crie uma máscara com a região delimitada pelo contorno
    mask_new = np.zeros_like(image_copy, dtype=np.uint8)
    cv2.drawContours(mask_new, [max_contour], -1, (255, 255, 255),
thickness=cv2.FILLED)

    # Extraia a região de interesse (ROI) aplicando a máscara
    roi = cv2.bitwise_and(image_copy, mask_new)

    image1= cv2.cvtColor(roi, cv2.COLOR_BGR2RGB)
    # Use the cvtColor() function to grayscale the image
    gray_image = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

    # APLICANDO FILTRO GAUSSIANO PARA REMOÇÃO DE RUÍDO
    #img=cv2.imread("img.jpg")
```

```
    gaugaussian=cv2.GaussianBlur(src=gray_image,ksize=(15,15),sigmaX=0)

    # Set threshold and maxValue
    thresh = 200
    maxValue = 255
    # Basic threshold example
    th, thresh_binary = cv2.threshold(gaugaussian, thresh, maxValue,
cv2.THRESH_BINARY)

    kernel=np.ones((9,9),np.uint8)
    opening=cv2.morphologyEx(thresh_binary,cv2.MORPH_OPEN, kernel)

    contornos,jerarquia=cv2.findContours(opening,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    if len(contornos) == 0:
        print('outro filtro menor')
        # Set threshold and maxValue
        thresh = 150
        maxValue = 255
        # Basic threshold example
        th, thresh_binary = cv2.threshold(gaugaussian, thresh, maxValue,
cv2.THRESH_BINARY)
        kernel=np.ones((9,9),np.uint8)
        opening=cv2.morphologyEx(thresh_binary,cv2.MORPH_OPEN, kernel)

        contornos,jerarquia=cv2.findContours(opening,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        if len(contornos) == 0:
            print('O sistema se moverá em malha aberta')
            #image_copy = image
            #cX = cXr
            #cY = cYr
            mov_malla_abierta()

        else:
            # CÓDIGO PARA ENCONTRAR O CONTORNO PRINCIPAL
            # ------------------------------------------------------------
            l_area = []
            j=0

            for i in range(len(contornos)):
                area= cv2.contourArea(contornos[i])
                l_area.append(area)

            max = l_area[0]
            for i in range(0,len(l_area)):
                if l_area[i] > max:
                    max = l_area[i]
                    j = i

            cont=contornos[j]
            M=cv2.moments(cont)
            cX=int(M["m10"]/M["m00"])
            cY=int(M["m01"]/M["m00"])
            cv2.circle(image_copy,(cX,cY),2,(0,0,0),-1)

cv2.putText(image_copy,"X:"+str(cX)+",Y:"+str(cY),(cX+5,cY),2,0.8,(0,0,0),0
)

            # encontre o círculo envolvente mínimo
```

```
            (xc,yc),radius = cv2.minEnclosingCircle(cont)

            # converter o raio e o centro em um número inteiro
            center = (int(xc),int(yc))
            radius = int(radius)

            # Desenhe o círculo envolvente na imagem
            cv2.circle(image_copy,center,radius,(0,255,0),2)

            # a imagem é 640x480
            xii=image_copy.shape[1]
            yii=image_copy.shape[0]
            xim= 320
            yim= 240

            # Desenhando linhas
            cv2.line(image_copy,(xim,0),(xim,yii),(0,128,255),1)
            cv2.line(image_copy,(0,yim),(xii,yim),(0,128,255),1)

            # desenhando um círculo no centro e colocando coordenadas
            cv2.circle(image_copy,(xim,yim),2,(255,128,0),-1)

cv2.putText(image_copy,"X:"+str(xim)+",Y:"+str(yim),(xim+5,yim+15),2,0.8,(2
55,128,0),0)

            # Salvamos no diretório
            image2= cv2.cvtColor(image_copy, cv2.COLOR_RGB2BGR)


    else:
        # CÓDIGO PARA ENCONTRAR O CONTORNO PRINCIPAL
        # ------------------------------------------------------------
        l_area = []
        j=0

        for i in range(len(contornos)):
            area= cv2.contourArea(contornos[i])
            l_area.append(area)
        #print(l_area)

        max = l_area[0]
        for i in range(0,len(l_area)):
            if l_area[i] > max:
                max = l_area[i]
                j = i

        cont=contornos[j]
        M=cv2.moments(cont)
        cX=int(M["m10"]/M["m00"])
        cY=int(M["m01"]/M["m00"])
        cv2.circle(image_copy,(cX,cY),2,(0,0,0),-1)

cv2.putText(image_copy,"X:"+str(cX)+",Y:"+str(cY),(cX+5,cY),2,0.8,(0,0,0),0
)

        # encontre o círculo envolvente mínim
        (xc,yc),radius = cv2.minEnclosingCircle(cont)

        # converter o raio e o centro em um número inteiro
        center = (int(xc),int(yc))
        radius = int(radius)
```

```
        # Desenhe o círculo envolvente na imagem
        cv2.circle(image_copy,center,radius,(0,255,0),2)

        # a imagem é 640x480
        xii=image_copy.shape[1]
        yii=image_copy.shape[0]
        xim= 320
        yim= 240

        # Desenhando linhas
        cv2.line(image_copy,(xim,0),(xim,yii),(255,128,0),1)
        cv2.line(image_copy,(0,yim),(xii,yim),(255,128,0),1)

        # desenhando um círculo no centro e colocando coordenadas
        cv2.circle(image_copy,(xim,yim),2,(255,128,0),-1)

cv2.putText(image_copy,"X:"+str(xim)+",Y:"+str(yim),(xim+5,yim+15),2,0.8,(2
55,128,0),0)

        # Salvamos no diretório
        image2= cv2.cvtColor(image_copy, cv2.COLOR_RGB2BGR)


    return cXr, cYr, width, height, cX, cY, image_copy
# ..................................................................
# Função para calcular a relação entre px/mm
# ..................................................................
def px_to_mm_m2(px_w,px_h,dif_x,dif_y,ancho_r, altura_r):

    relacion_mm_px_w = ancho_r/px_w
    relacion_mm_px_h = altura_r/px_h
    print('A relação mm/px do segundo método é:
',round(relacion_mm_px_w,2), round(relacion_mm_px_h,2))

    distancia_w = dif_x*relacion_mm_px_w
    distancia_h = dif_y*relacion_mm_px_h

    return distancia_w, distancia_h, relacion_mm_px_w, relacion_mm_px_h
# ..................................................................
# Função para calcular o ângulo de movimento do motor
# ..................................................................
def ang_mov(image, tol, dt, R):

    rela = dt/R

    cXr, cYr, width, height, cX, cY, image_with_detections =
coordenadas_imagenes(image)

    coord_receptor = cXr, cYr
    coord_sol = cX, cY
    print('As coordenadas do receptor são: ', coord_receptor)
    print('As coordenadas do sol são: ',coord_sol)

    # Diferenças entre o centro do receptor e o centro do sol (refletor)
    dif_x = coord_receptor[0]-coord_sol[0]
    dif_y = coord_receptor[1]-coord_sol[1]
    #print(dif_x,dif_y)

    dx2, dy2, mm_px_w, mm_px_h = px_to_mm_m2(width, height, dif_x, dif_y,
ancho_r, altura_r)
```

```
    # mrad/px
    betha_x2 = mm_px_w/dt*1000*rela
    betha_y2 = mm_px_h/dt*1000*rela
    print('A relação mrad/px para o sistema é: ',round(betha_x2,2),
round(betha_y2,2))
    print('A relação graus/px para o sistema é:
',round(math.degrees(betha_x2/1000),3),
round(math.degrees(betha_y2/1000),3))

    # mrad
    gamma_x2 = round((dif_x*betha_x2),2)
    gamma_y2 = round((dif_y*betha_y2),2)

    print(' ', gamma_x2)
    print('O erro em x em mrad para o sistema é:: ', gamma_y2)

    erro_x = gamma_x2
    erro_y = gamma_y2
    erro_f = round(math.sqrt(gamma_x2**2 + gamma_y2**2),3)
    print('O erro mrad do sistema é: ', erro_f)
    print('o erro de largura em mm: ',round(dx2,2))
    print('o erro de altura em mm: ',round(dy2,2))

    # o erro em graus que o sistema deve mover é
    mov_x = round(math.degrees(gamma_x2/1000),3)
    mov_y = round(math.degrees(gamma_y2/1000),3)
    print('angulo em graus do sistema x: ',mov_x)
    print('angulo em graus do sistema y: ',mov_y)

    if abs(mov_x) > round(tol*math.degrees(betha_x2/1000),3):
        mov_x = mov_x
    else:
        mov_x = 0

    if abs(mov_y) > round(tol*math.degrees(betha_y2/1000),3):
        mov_y = mov_y
    else:
        mov_y = 0

    print('angulo em graus que o sistema deve se mover em x: ',mov_x)
    print('angulo em graus que o sistema deve se mover em y: ',mov_y)

    return image_with_detections, mov_x, mov_y, dif_x, dif_y, erro_x,
erro_y, erro_f
# ..............................................................................
# Função para salvar a imagem não processada
# ..............................................................................
def save_img(image):
    # Extraímos as informações atuais
    info = datetime.now()
    # Extraímos data
    fecha = info.strftime('%Y_%m_%d')
    # Extraímos hora
    hora = info.strftime('%H_%M_%S')
    # formatar
    fh = f'{fecha}_{hora}'
    fechai = info.strftime('%Y/%m/%d')
    # Extraímos hora
    horai = info.strftime('%H:%M:%S')
    # formatar
```

```
    fhi = f'{fechai} {horai}'

    # Escreva a data e a hora na imagem
    texto = fhi
    # Posição do texto (canto inferior esquerdo)
    posicion = (380, 470)

    # Dimensões do fundo branco
    width = 270  # Largura do fundo
    height = 35  # Alto do fundo
    # Coordenadas do retângulo
    rect_x1, rect_y1 = posicion[0] - 5, posicion[1] - 25
    rect_x2, rect_y2 = rect_x1 + width, rect_y1 + height

    # Desenhe o retângulo branco
    cv2.rectangle(image, (rect_x1, rect_y1), (rect_x2, rect_y2), (255, 255,
255), cv2.FILLED)

    # Escala de fonte e texto
    fuente = cv2.FONT_HERSHEY_SIMPLEX
    escala = 0.7
    color_texto = (0, 0, 0)  # Cor preta para texto
    grosor = 1

    # Adicionar texto à imagem
    cv2.putText(image, texto, posicion, fuente, escala, color_texto,
grosor, cv2.LINE_AA)

    # a imagem é salva em um arquivo

cv2.imwrite('//home/George/Desktop/Imagenes/No_Procesadas/malla_cerrada/Ima
gen_'+str(fh)+ '.jpg' ,image)
# ..........................................................................
# Função para salvar a imagem processada
# ..........................................................................
def save_img_processing(image):
    # Extraímos as informações atuais
    info = datetime.now()
    # Extraímos data
    fecha = info.strftime('%Y_%m_%d')
    # Extraímos hora
    hora = info.strftime('%H_%M_%S')
    # formatar
    fh = f'{fecha}_{hora}'
    fechai = info.strftime('%Y/%m/%d')
    # Extraímos hora
    horai = info.strftime('%H:%M:%S')
    # formatar
    fhi = f'{fechai} {horai}'

    # Escreva a data e a hora na imagem
    texto = fhi
    # Posição do texto (canto inferior esquerdo)
    posicion = (380, 470)

    # Dimensões do fundo branco
    width = 270  # Largura do fundo
    height = 35  # Alto do fundo
    # Coordenadas do retângulo
    rect_x1, rect_y1 = posicion[0] - 5, posicion[1] - 25
    rect_x2, rect_y2 = rect_x1 + width, rect_y1 + height
```

```
    # Desenhe o retângulo branco
    cv2.rectangle(image, (rect_x1, rect_y1), (rect_x2, rect_y2), (255, 255,
255), cv2.FILLED)

    # Escala de fonte e texto
    fuente = cv2.FONT_HERSHEY_SIMPLEX
    escala = 0.7
    color_texto = (0, 0, 0)  # Cor preta para texto
    grosor = 1

    # Adicionar texto à imagem
    cv2.putText(image, texto, posicion, fuente, escala, color_texto,
grosor, cv2.LINE_AA)

    # a imagem é salva em um arquivo

cv2.imwrite('//home/George/Desktop/Imagenes/Procesadas/malla_cerrada/Imagen
_'+str(fh)+ '.jpg' ,image)
# ..........................................................................
# Função para movimentar o sistema com base no cálculo de processamento
# ..........................................................................
def mov_sistema(mov_azim, mov_elev, mov_x, mov_y):

    if mov_x < -0.05:

        if mov_x < -3:
            mov_azim = round(mov_azim - 1, 2)#+ mov_x
        elif -3 < mov_x < -0.3:
            mov_azim = round(mov_azim - 0.1, 2)#+ mov_x
            #mov_azim = round(mov_azim + mov_x, 2)#+ mov_x
        else:
            mov_azim = round(mov_azim - 0.05, 2)#+ mov_x
            #mov_azim = round(mov_azim + mov_x, 2)#+ mov_x
        # sistema move-se para a esquerda


    elif mov_x > 0.05:

        if mov_x > 3:
            mov_azim = round(mov_azim + 1, 2)#+ mov_x
        elif 0.3 < mov_x < 3:
            mov_azim = round(mov_azim + 0.1, 2)#+ mov_x
            #mov_azim = round(mov_azim + mov_x, 2)#+ mov_x
        else:
            mov_azim = round(mov_azim + 0.05, 2)#+ mov_x
            #mov_azim = round(mov_azim + mov_x, 2)#+ mov_x
        # sistema move-se para a dereita
    else:
        mov_azim = round(mov_azim, 2)


    if mov_y < -0.05:

        if mov_y < -3:
            mov_elev = round(mov_elev - 1, 2)#+ mov_x
        elif -3 < mov_y < -0.3:
            mov_elev = round(mov_elev - 0.1, 2)#+ mov_x
            #mov_elev = round(mov_elev + mov_y, 2)#+ mov_x
        else:
            mov_elev = round(mov_elev - 0.05, 2)#+ mov_x
```

```
            #mov_elev = round(mov_elev + mov_y, 2)#+ mov_x
        #mov_y = mov_y*-1
        # sistema sobe

    elif mov_y > 0.05:

        if mov_y > 3:
            mov_elev = round(mov_elev + 1, 2)#+ mov_x
        elif 0.3 < mov_y < 3:
            mov_elev = round(mov_elev + 0.1, 2)#+ mov_x
            #mov_elev = round(mov_elev + mov_y, 2)#+ mov_x
        else:
            mov_elev = round(mov_elev + 0.05, 2)#+ mov_x
            #mov_elev = round(mov_elev + mov_y, 2)#+ mov_x
        # sistema se move para baixo
    else:
        mov_elev = round(mov_elev, 2)

    angulos_teoricos = str(mov_azim)+str(',')+str(mov_elev)
    # o sistema se move de acordo com os ângulos
    lectura_mov_motores(angulos_teoricos)

    return angulos_teoricos, mov_azim, mov_elev
# ..........................................................................
# Função para descartar quadros não processados
# ..........................................................................
def clear_buffer(cap, frames=5):
    for i in range(frames):
        cap.grab()  # Descartar quadros não processados
# ..........................................................................
# Função para movimentar o sistema através de malha aberta
# ..........................................................................
def mov_malla_abierta():
    # Obter a hora atual
    hora_actual = datetime.now().time()
    # Formatar hora atual
    horaa = hora_actual.strftime("%H:%M:%S")

    nhour = datetime.strptime(horaa, "%H:%M:%S").hour
    nminute = datetime.strptime(horaa, "%H:%M:%S").minute
    nsecond = datetime.strptime(horaa, "%H:%M:%S").second

    hourtominute = nhour*60
    secondtominute = nsecond/60
    total_minute = hourtominute + nminute + secondtominute

    nminew = total_minute
    nsec = nminew*60

    h_fin = convertsec(nsec)
    # Cálculo do azimute e elevação do sol
    alpha_s, azim_s, theta_s = elev_azim(h_fin, Lst, Lloc, E, delta, phi)
    # cálculo de azimute e elevação do heliostato
    elev_helio, azim_helio = elev_azi_helio(theta_s,azim_s, theta_t,
beta_t)

    # Arredondar para 2 casas decimais
    mov_azim = round(azim_helio,3)
    mov_elev = round(elev_helio,3)

    angulos_teoricos = str(mov_azim)+str(',')+str(mov_elev)
```

```
    # o sistema se move de acordo com os ângulos
    lectura_mov_motores(angulos_teoricos)
# ========================================================================
#                         DECLARAÇÃO DE VARIÁVEIS
# ========================================================================
# Configuração de pinos GPIO
ENABLE_1 = 18
M0_1 = 23
M1_1 = 24
M2_1 = 25
STEP_PIN_1 = 8
DIR_PIN_1 = 7

ENABLE_2 = 12
M0_2 = 16
M1_2 = 20
M2_2 = 21
STEP_PIN_2 = 4
DIR_PIN_2 = 17

TIME_FOR_STEP = 2000

PUSHBUTTON_1 = 27
PUSHBUTTON_2 = 22
PUSHBUTTON_4 = 10
PUSHBUTTON_5 = 9

# Inicialização GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup([ENABLE_1, M0_1, M1_1, M2_1, STEP_PIN_1, DIR_PIN_1, ENABLE_2,
M0_2, M1_2, M2_2, STEP_PIN_2, DIR_PIN_2], GPIO.OUT)
GPIO.setup([PUSHBUTTON_1, PUSHBUTTON_2, PUSHBUTTON_4, PUSHBUTTON_5],
GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Variáveis globais
dato_rx_azi = 0
dato_rx_ele = 0
leeCadena = ""
valor = ""
num_Pulsos_1 = 0
num_Pulsos_2 = 0
dat_azi = 0.0
dat_ele = 0.0

isFirstValueSaved = False
funcionEjecutada = False
firstazi = 0.0
firstele = 0.0
faz = 0.0
fel = 0.0
azi = 0.0
ele = 0.0

lista =[]

# Dimensões do receptor em mm
ancho_r=214
altura_r=190
# tolerância do número de pixels
tol = 1
```

```
erro_x=0
erro_y=0
erro_f=0
dif_x = 0
dif_y = 0

now = datetime.now()
dia= now.day
mes = now.month
n = day_of_year(dia, mes)
GMT = 3 # Greenwich Mean Time, Fuso horário -3 para o Brasil
Lloc= 48.885800 # Longitude do local em graus, -49.03479832
phi = math.radians(-23.981578) # Latitude -22.35102511
Lst = GMT*15 # 15 deg/h
# Azimute do receptor/torre em graus
beta_t = 322.5
# R = RÁDIO ENTRE O HELIOSTATO E O RECEPTOR
# H = ALTURA ENTRE O HELIOSTATO E O RECEPTOR
R = 9580
H = 1283
theta_t= math.atan2(R, H)
# Calculo delta e E
delta, E = delta_E(n)
# ============================================================================
# Fim da declaração de variável
# ============================================================================
# ............................................................................
# INÍCIO DO SISTEMA
# ............................................................................
# Coloque o sistema na posição inicial
setup_motores()

# Caminho para o arquivo de configuração baixado do Firebase Console
cred = credentials.Certificate('/xxxxxxxxxxxxxxxxxxxxxxxx.json')

# Inicializar o aplicativo Firebase (URL do projeto)
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://xxxxxxxxxxxxx.firebaseio.com'
})

# Referência ao banco de dados
ref = db.reference('/malla_cerrada/')

# sistema se movimenta por malha aberta
mov_malla_abierta()

# tempo de espera
time.sleep(2)

now = datetime.now()
fech = now.strftime('%Y-%m-%d ')
hor = h_fin
f_h = f"{fech}{hor}"
# enviando dados para o firebase
data_to_upload =
f"{f_h},{mov_azim},{mov_elev},{dif_x},{dif_y},{erro_x},{erro_y},{erro_f}"
print(data_to_upload)
upload_data(data_to_upload)

# Inicializar a câmera
```

```
cap = cv2.VideoCapture(0)

# captura da imagem
image = None
while image is None:
    # Captura da imagem
    clear_buffer(cap)
    ret, image = cap.read()  # Captura de frame
    if not ret:
        print("Erro ao capturar imagem. Tentando novamente...")
        time.sleep(0.5)  # Aguarde um pouco antes de tentar novamente.



try:
    while True:

        # cópia da imagem
        img_c = image.copy()

        # detecta as coordenadas de imagem e movimento do sistema
        image_with_detections, mov_x, mov_y, dif_x, dif_y, erro_x, erro_y,
erro_f = ang_mov(image, tol, dt, R)

        # Mova o sistema de acordo com os ângulos obtidos
        angulos_procesamiento, mov_azim, mov_elev = mov_sistema(mov_azim,
mov_elev, mov_x, mov_y)

        # salvar a imagem capturada
        save_img(img_c)
        # salvar a imagem procesada
        save_img_processing(image_with_detections)

        now = datetime.now()
        fech = now.strftime('%Y-%m-%d %H:%M:%S')
        # enviando dados para o firebase
        data_to_upload =
f"{fech},{mov_azim},{mov_elev},{dif_x},{dif_y},{erro_x},{erro_y},{erro_f}"
        print(data_to_upload)
        upload_data(data_to_upload)

        # Captura da imagem
        image = None
        while image is None:

            clear_buffer(cap)
            ret, image = cap.read()  # Captura de frame
            if not ret:
                print("Erro ao capturar imagem. Tentando novamente...")
                time.sleep(0.5)  # Aguarde um pouco antes de tentar
novamente.

        time.sleep(10)


except KeyboardInterrupt:
    #print("Ctrl+C foi pressionado. Saindo do loop.")
    setup_motores()
    deshabilita_drv8825_1
    deshabilita_drv8825_2
    cap.release()
```