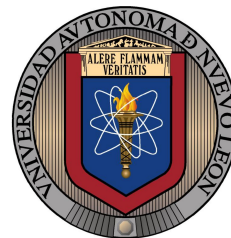




Facultad de Ingeniería Mecánica y Eléctrica

UANL

MECATRÓNICA



Sistemas de Visión

Equipo 03

Grupo:001





SEGUIDOR SOLAR PIA

FECHA DE ENTREGA: 22/11/2024

Profesor:

Ing. Mario Ángel Rico Méndez

Tabla 1: Equipo 3:

| Nombre | Matrícula | Carrera | Foto |
|-------------------------------|-----------|---------|---|
| Marcos Alberto Coronado Lopez | 1949489 | IMTC |  |
| Rene Guzman Perez | 1849342 | IMTC |  |
| Adrian Guevara Aguayo | 1942809 | IMTC |  |
| Alexander Reyes Hernández | 1853430 | IMTC |  |

Índice

| | |
|---|-----------|
| 1. Objetivo | 3 |
| 2. Introducción | 3 |
| 2.1. Adquisición de imágenes | 3 |
| 2.2. Mejoramiento de imagen transformaciones y filtrado | 5 |
| 2.3. Segmentación de umbral y binarizacion | 6 |
| 3. Descripción del diseño | 8 |
| 4. Diseño Electronico | 10 |
| 5. Desarrollo | 13 |
| 6. Conclusiones | 29 |

EQUIPO 3- 20 DE SEPTIEMBRE DE 2025-A- 23:41

1. Objetivo

El objetivo de este proyecto es que los estudiantes diseñen y desarrollen un seguidor solar de dos grados de libertad, utilizando sistemas de visión artificial para detectar y rastrear la posición del sol. El sistema deberá optimizar la captación de energía solar mediante la integración de hardware y software que permita el procesamiento de imágenes, mejorando así la precisión en el seguimiento.

2. Introducción

Empecemos definiendo que es el término “imagen monocromática” o imagen simplemente, se refiere a una función de intensidad de luz bidimensional $f(x,y)$, donde x e y indican las coordenadas espaciales y el valor de f en cualquier punto (x,y) es proporcional a la luminosidad (o nivel de gris) de la imagen en dicho punto. Una imagen digital es una imagen (función) $f(x,y)$ que ha sido discretizada tanto en coordenadas espaciales como en luminosidad. Una imagen digital puede ser considerada como una matriz cuyos índices de renglón y columna identifican un punto (un lugar en el espacio bidimensional) en la imagen y el correspondiente valor de elemento de matriz identifica el nivel de gris en aquel punto. Los elementos de estos arreglos digitales son llamados elementos de imagen o pixels.

- Adquisición de imágenes
- Mejoramiento de imagen transformaciones y filtrado
- Segmentación de umbral y binarización

2.1. Adquisición de imágenes

La adquisición de la imagen está a cargo de algún transductor o conjunto de transductores que mediante la manipulación de la luz o de alguna otra forma de radiación que es emitida o reflejada por los cuerpos, se logra formar una representación del objeto dando lugar a la imagen. Ejemplos: el ojo humano, sensores de una cámara fotográfica o de vídeo, tomógrafos. Es importante saber que, durante la etapa de adquisición, los transductores agregan ruido a la imagen. Además del ruido, los transductores poseen una resolución limitada, lo cual repercute en la apreciación de dicha imagen. El procesamiento digital de la imagen consiste en eliminar la mayor cantidad de ruido que se le agrega durante la adquisición, así como también mejorar las características de dicha imagen como: definición de contornos, color, brillo, etc., valiéndose de procedimientos y herramientas matemáticas. En esta etapa se encuentran también técnicas de codificación para el almacenamiento o bien para la transmisión.

La presentación al observador consiste en el método empleado para exponer la imagen la cual puede ser impresa o por medios electrónicos como la televisión, el monitor de una computadora, o algún

otro medio. Para la presentación de la imagen se deben considerar ciertos aspectos de percepción humana, así como las velocidades de despliegue del dispositivo utilizado. Algunos de los problemas característicos en el diseño de estos subsistemas que involucran el uso de representaciones de señales son las siguientes:

- Los dispositivos sensoriales realizan un número limitado de mediciones sobre las señales de entrada; estas mediciones deben ser adecuadas para obtener aproximaciones útiles. Decidir que mediciones realizar y como usarlas de tal manera que aproximen mejor a las señales de entrada son los problemas que deben ser resueltos.
- Para la selección del procesamiento y/o codificación que se hará sobre una señal, es necesaria una interpretación de las componentes de la señal. El modelo del sistema de visión humano puede ser utilizado en ciertas etapas de procesamiento para dicha interpretación.
- Los dispositivos de despliegue sintetizan una imagen usando un número finito de respuestas básicas de despliegue, como los puntos de fósforo utilizados en un tubo de rayos catódicos. Seleccionar el tamaño y la forma de estas respuestas de despliegue, la configuración (número y posición relativa) y como pueden ser controlados de la mejor manera óptima para obtener imágenes con la calidad/fidelidad requerida son aspectos que deben ser cubiertos.
- Realizar un breve estudio sobre el funcionamiento del sistema visual humano (Human Visual System, HVS) será de utilidad para entender mejor la forma en que percibimos las imágenes y con ello, poder explotar estas características en el tratamiento digital de imágenes. Es posible modelar el ojo humano como un sistema lineal e invariante en el tiempo (SLI). Para ello se deben tener presentes dos conceptos:
- La respuesta al impulso, que es una función que describe el comportamiento en el tiempo de un sistema, en nuestro caso el sistema es el ojo. Una vez obtenida la respuesta al impulso, se realiza la convolución de la función obtenida con cualquier otra función con el objetivo de observar y conocer la respuesta del sistema a esa nueva función



Figura 1: *Adquisición.*

2.2. Mejoramiento de imagen transformaciones y filtrado

Ajustar el contraste, filtrar morfológicamente, enfocar imágenes borrosas y procesar según regiones de interés el ajuste de imágenes es el proceso de modificar imágenes para que los resultados sean más adecuados para su visualización o su análisis posterior. Por ejemplo, puede eliminar el ruido, mejorar la nitidez o ajustar el contraste de una imagen para que sea más fácil identificar características clave.

El mejoramiento de imágenes mediante transformaciones implica modificar ciertos aspectos de una imagen para mejorar su calidad visual o destacar características específicas. Aquí tienes algunas transformaciones comunes:

Corrección de Contraste: Ajusta la diferencia entre las regiones más claras y más oscuras de una imagen, resaltando detalles.

Ajuste de Brillo: Modifica la intensidad general de la luz en una imagen, haciendo que la imagen sea más brillante o más oscura.

Filtrado: Utiliza filtros para suavizar o resaltar ciertos detalles en la imagen. Por ejemplo, un filtro de paso bajo puede suavizar, mientras que uno de paso alto puede resaltar bordes.

Equalización del Histograma: Distribuye uniformemente las intensidades de píxeles en el histograma, mejorando el contraste.

Realce de Bordes: Destaca los bordes y contornos de la imagen para mejorar la definición.

Reducción de Ruido: Elimina o reduce la presencia de ruido en una imagen, lo que puede mejorar su nitidez.

Escalamiento: Ajusta el tamaño de la imagen, ya sea para agrandar o reducir.

Rotación: Gira la imagen en un ángulo determinado.

Cambio de Color: Puede ajustar la saturación, tonalidad o equilibrio de color para mejorar la apariencia general.

Estas transformaciones se aplican con frecuencia en procesamiento de imágenes y fotografía digital para mejorar la calidad visual y resaltar características importantes. La elección de la transformación dependerá de los objetivos específicos y del tipo de imagen que estés tratando de mejorar.

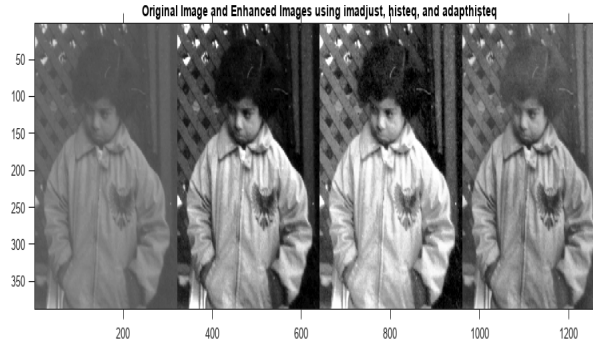


Figura 2: *Ejemplo de contraste.*

2.3. Segmentación de umbral y binarización

Binarización: es una técnica que consiste en la realización de un barrido en la matriz de la imagen digital, por medio de bucles o recursividad, con el fin de que el proceso produzca la reducción de la escala de grises a dos únicos valores. Negro ($= 0$) y blanco ($= 255$), o lo que es lo mismo, un sistema binario de ausencia y presencia de color 0-1. La comparación de cada píxel de la imagen viene determinada por el umbral de sensibilidad (valor $T = \text{Threshold}$). Sezgin y Sankur (2004), en base a las particularidades entre algoritmos categorizan los métodos de umbralización en seis grupos. Aquí añadimos uno más, los métodos globales: Histograma: métodos basados en el análisis de los picos máximos y mínimos de las curvas del histograma del suavizado de la imagen. Clustering: métodos basados en discernir como las muestras de los niveles de gris se agrupan o alternatively se modelan como una mezcla de dos gaussianas. Entropía: métodos basados en el análisis de los resultados de la aplicación de algoritmos que utilizan la entropía de las regiones frontal y de fondo, la entropía cruzada entre la imagen original y binarizada. Similitud: métodos basados en la búsqueda de una similitud entre las escalas de grises, como la tonalidad difusa, los bordes de la imagen, etc. Espaciales: métodos analíticos que usan el orden de distribución, la probabilidad y/o la correlación entre los diferentes píxeles. Globales: métodos cuyo valor del umbral es estático. Locales: métodos que adaptan el valor del umbral, de forma manual o automática, a cada píxel dependiendo de las características locales de la imagen segmentada.

Umbralizado: Una imagen en gris es binarizada consiguiendo un umbral óptimo T y con ese valor se separan los píxeles en dos regiones, una de zonas claras y otra de zonas oscuras. En la umbralización hay dos posibles situaciones: 1. Umbral único (Global thresholding). Se da cuando solamente hay dos regiones de píxeles, para separarlos se establece un único umbral T . Este tipo de umbral se obtiene fácilmente a partir de histogramas bimodales. 2. Umbral multinivel (Local thresholding). Dada una imagen con varios objetos, para separarlos hace falta más de un umbral, de forma que los píxeles que se encuentren entre cada par de umbrales T_i y T_j representarán a un objeto. Los umbrales elegidos pueden ser de varios tipos, dependiendo de las características tenidas

en cuenta para su elección. Watershed transformation (“Línea divisoria de aguas”): Se considera la imagen de grises como un relieve topográfico, donde el valor numérico de cada pixel representa la elevación en ese punto. A partir de los mínimos regionales se determina para cada uno de ellos su zona de influencia o cuenca. La separación entre las cuencas es la llamada línea divisoria de aguas. Si se efectúa esta transformación sobre una imagen gradiente, las líneas divisorias van a seguir los contornos de la imagen (gradiente elevado), llevando a cabo una segmentación de las zonas de la imagen homogéneas (bajo gradiente). Si el histograma de una imagen posee picos, podemos separar dos zonas y el umbral es aquel valor que se encuentra en el valle entre ambas.

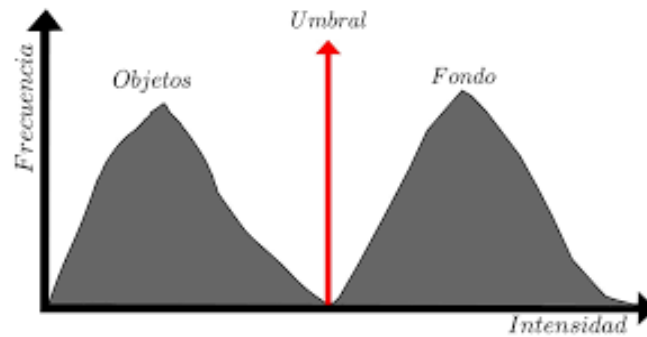


Figura 3: *Umbral*

EQUIPO 3- 20 DE SEPTIEMBRE DE

3. Descripción del diseño

En este proyecto se emplearán diversos componentes electrónicos y mecánicos para implementar un sistema de seguimiento solar, diseñado con el objetivo de maximizar la captación de energía solar. Los materiales principales incluyen una cámara para detección visual, una laptop para el procesamiento y control del sistema, un microcontrolador ESP32 para la comunicación con los actuadores, un servo motor y un motor a pasos para el movimiento del panel solar, así como un driver A4988 para el control del motor a pasos.

Cada uno de estos elementos cumple una función específica en el sistema, trabajando de manera conjunta para garantizar el funcionamiento óptimo. A continuación se presenta una descripción detallada de la función de cada componente y su integración en el proyecto.

1. Cámara (Detección del Sol)

Función: La cámara detecta la luz más brillante en el rango amarillo (simulando la posición del sol o una lámpara).

Cómo funciona en el código: La cámara captura un flujo de video en tiempo real utilizando la librería OpenCV mediante la función `cv2.VideoCapture`, que genera cuadros continuos de video. Cada cuadro es convertido al espacio de color HSV, permitiendo segmentar regiones de color amarillo con el filtro `cv2.inRange`, configurado con rangos específicos para tonos amarillos. Posteriormente, los contornos de estas áreas detectadas se extraen con `cv2.findContours`, identificando la región más brillante. Finalmente, el centroide de esta área es calculado, proporcionando la posición relativa del sol respecto al centro de la imagen.

2. Laptop

Función: La laptop procesa la imagen capturada por la cámara y controla el sistema de detección y movimiento del panel solar.

Cómo funciona en el código: El procesamiento de video ocurre en la laptop, donde los datos capturados por la cámara se analizan para determinar la posición del sol. Mediante una interfaz gráfica desarrollada con Tkinter, se ofrece al usuario un control interactivo, que incluye opciones para encender y apagar la cámara, monitorear la posición detectada del sol en tiempo real y enviar comandos al ESP32. El software interpreta la posición del punto amarillo en la imagen y define si el sol está centrado o si se requiere movimiento en algún eje. Los comandos resultantes se envían al ESP32 mediante comunicación serial usando la librería serial.

3. ESP32

Función: El ESP32 recibe los comandos enviados por la laptop y controla los actuadores del sistema.

Cómo funciona en el código: Se establece una conexión serial entre la laptop y el ESP32 utilizando la librería serial. Según los comandos recibidos, el ESP32 realiza las siguientes acciones: Si el comando es “SOL_CENTRO“, no se requiere ajuste del panel. Si se recibe “SERVO_ARRIBA“ o “SERVO_ABAJO“, el ESP32 ajusta el ángulo del servo motor para corregir la inclinación vertical. Si los comandos son “MOTOR_DERECHA“ o “MOTOR_IZQUIERDA“, se activan señales hacia el driver del motor a pasos para realizar movimientos precisos en el eje horizontal.

4. Servo Motor

Función: Ajusta la inclinación vertical del panel solar para seguir el movimiento del sol.

Cómo funciona en el código: El ESP32 interpreta los comandos recibidos, como “SERVO_ARRIBA“ o “SERVO_ABAJO“, y genera señales PWM que controlan el ángulo del servo motor. Este ajuste permite que el panel siga el desplazamiento del sol en el eje vertical, maximizando la exposición.

5. Panel Solar

Función: Es el objeto físico que sigue la posición del sol para optimizar la captación de energía solar.

Cómo funciona en el sistema: El panel solar está montado sobre un sistema que combina un servo motor para movimientos verticales y un motor a pasos para movimientos horizontales. Los ajustes realizados en ambos ejes permiten orientar el panel de forma precisa hacia la posición del sol, asegurando una captación de luz óptima.

6. A4988 (Driver del Motor a Pasos)

Función: Controla el motor a pasos encargado de ajustar la orientación horizontal del panel solar.

Cómo funciona en el sistema: El driver A4988 recibe señales del ESP32 que indican la dirección y los pasos necesarios para mover el motor. Este módulo traduce las señales de control en impulsos eléctricos, permitiendo que el motor realice desplazamientos precisos hacia la izquierda o derecha según sea necesario.

7. Motor a Pasos

Función: Realiza el movimiento horizontal del panel solar.

Cómo funciona en el código: El ESP32 genera señales para el driver A4988, especificando tanto la dirección como el número de pasos que debe ejecutar el motor. Según los comandos “MOTOR_DERECHA“ o “MOTOR_IZQUIERDA“, el motor se mueve en la dirección adecuada para alinear el panel solar con el sol. Este mecanismo garantiza precisión y control en el movimiento horizontal.

Flujo General del Sistema

Primero, la cámara detecta la luz amarilla más brillante y calcula su posición relativa al centro de la imagen. Estos datos son procesados por la laptop, que determina si el sol está centrado o requiere ajustes en la orientación del panel. Con base en esta información, se envían comandos al ESP32 a través de comunicación serial. El ESP32, a su vez, controla el servo motor y el motor a pasos para alinear el panel solar en los ejes vertical y horizontal, maximizando la captación de energía solar.

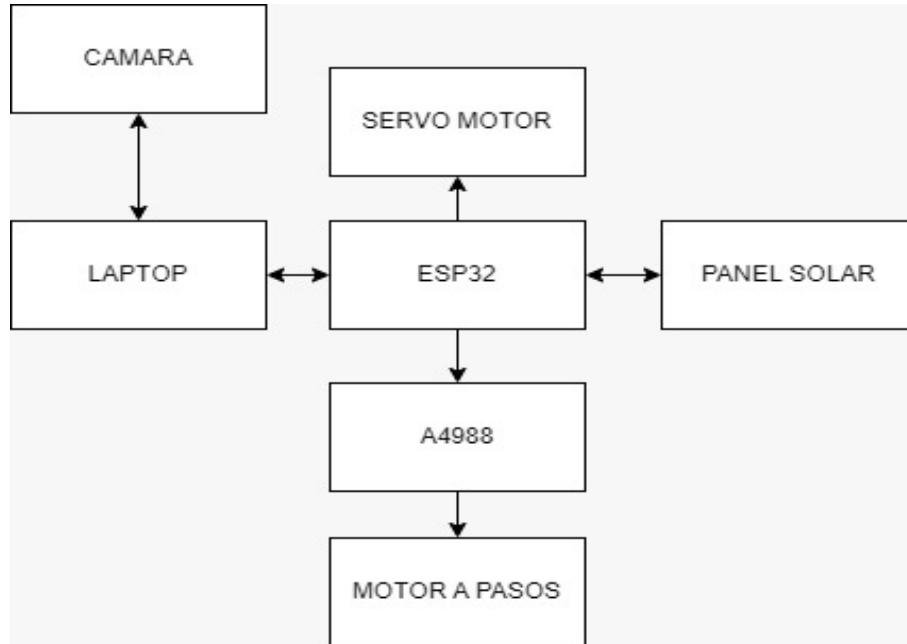


Figura 4: *Diagrama de bloques.*

4. Diseño Electronico

El sistema incluye las siguientes conexiones electrónicas clave:

■ ESP32:

- Pines GPIO configurados para el control del A4988 y el servo motor.
- Comunicación serial con la laptop.

■ Driver A4988:

- Entrada de señal de paso (STEP) y dirección (DIR) desde el ESP32.

- Conexión a los bobinados del motor a pasos y a la fuente de alimentación.
- **Servo Motor:**
 - Controlado directamente por un pin PWM del ESP32.
 - Alimentación independiente de 12V para garantizar estabilidad.
- **Fuente de Alimentación:**
 - Proporciona energía para el ESP32, el servo motor y el driver A4988.

Cálculos y Dimensionamiento

1. Dimensionamiento del Motor a Pasos:

- Torque requerido: $T = \dots$
- Selección de un motor con torque superior al requerido para garantizar precisión.

2. Control del Servo Motor:

- Ángulos de rotación calculados con base en la posición del sol en el eje vertical (elevación solar).
- PWM generado por el ESP32 para ajustar el ángulo.

3. Corriente del Driver A4988:

- Ajuste de corriente basado en la especificación del motor a pasos para evitar sobrecalentamiento.

Simulaciones

- **Procesamiento de Imagen:** Simulaciones realizadas con OpenCV para ajustar el rango de color HSV y optimizar la detección de la luz amarilla en distintas condiciones de iluminación.
- **Movimiento de Motores:** Pruebas en el simulador Arduino IDE para verificar la correcta comunicación entre la laptop, el ESP32 y los actuadores.

Con este diseño electrónico, se asegura un sistema eficiente, modular y fácilmente escalable que permite el seguimiento solar en tiempo real.

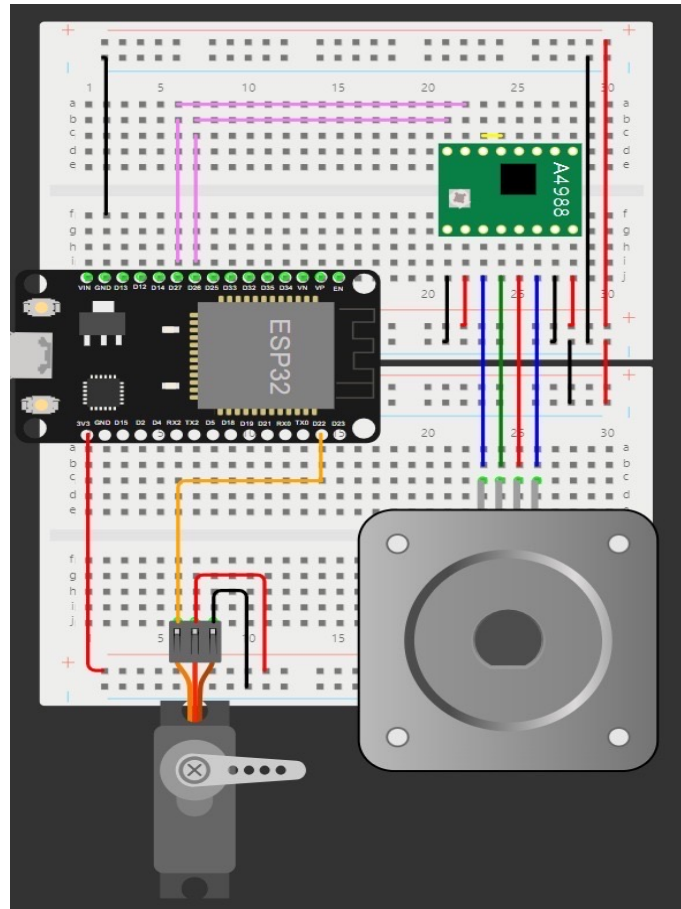


Figura 5: *Diseño del sistema electrónico*

EQUIPO 3- 20 DE FEB

5. Desarrollo

A continuación, se presentará el algoritmo diseñado para configurar el seguidor solar. Además, daremos una muy breve explicación sobre lo que realizaba cada código que fue utilizado para el correcto funcionamiento de nuestro seguidor solar, así como incluir capturas del código descrito.

```
# Configuración del puerto serial (ajusta el nombre del puerto según tu sistema)

# Función para enviar comandos al ESP32
def enviar_comando(comando):
    esp32.write((comando + '\n').encode())
    time.sleep(0.1)
```

Figura 6: Código necesario para configurar el puerto Serial

Figura 6: La codificación que se encarga de configurar el puerto y permitir el correcto funcionamiento de nuestro seguidor solar es el siguiente:

Descripción de la Función y el Diseño Electrónico

def enviar_comando(comando):

- **Parámetro comando:** Es el comando que se enviará al ESP32. Este puede ser una cadena de texto que el ESP32 interpretará para realizar una acción específica, como mover un motor o cambiar una configuración.
- `esp32.write((comando + '\n').encode())`
 - `comando + '\n'`: Agrega un carácter de nueva línea (`\n`) al final del comando. Esto es necesario porque muchos sistemas (como el ESP32) usan este carácter para identificar el final de una instrucción.
 - `.encode()`: Convierte el comando de texto en bytes, ya que la comunicación serial requiere que los datos se envíen en este formato.
 - `esp32.write()`: Envía los datos codificados al ESP32 a través del puerto serial.
- `time.sleep(0.1)`
 - Introduce un retraso de 0.1 segundos (100 milisegundos) antes de continuar con la ejecución

del programa. Esto permite que el ESP32 procese el comando antes de recibir otro.

```
# Función para actualizar el índice de la cámara seleccionado
def actualizar_indice_camara(opcion):
    global camera_index
    camera_index = opcion
```

Figura 7: Código necesario para configurar la camara

```
# Función para actualizar la información del sol y mostrarla en la interfaz
def actualizar_informacion_sol():
    info = ""
    coordenadas_sol = ""
    for nombre, ciudad in ciudades.items():
        ahora = datetime.now(pytz.timezone(ciudad.timezone))
        s = sun(ciudad.observer, date=ahora, tzinfo=ciudad.timezone)

        estado = "Día" if s['sunrise'] <= ahora <= s['sunset'] else "Noche"
        info += f"{nombre}: {estado}\nAmanecer: {s['sunrise'].strftime('%H:%M:%S')}\nAtardecer: {s['sunset'].strftime('%H:%M:%S')}\n\n"

        # Calcular azimuth y elevación
        azimuth_valor = azimuth(ciudad.observer, ahora)
        elevation_valor = elevation(ciudad.observer, ahora)
        coordenadas_sol += f"{nombre}: Azimut={azimuth_valor:.2f}, Elevación={elevation_valor:.2f}\n"

    etiqueta_info_sol.config(text=info)
    etiqueta_coordenadas_sol.config(text=coordenadas_sol)
    root.after(60000, actualizar_informacion_sol) # Actualiza cada 60 segundos
```

Figura 8: Código necesario para permitirnos configurar la información del sol y mostrarla en la interfaz

Figura 8.- La función del siguiente código es la de actualizar y mostrar la información sobre el sol (como el estado de día o noche, las horas de amanecer y atardecer, y las coordenadas solares).

1. Inicialización de variables:

info = ""

coordenadas sol = ""

Se inicializan dos cadenas vacías, info y coordenadas sol, que se llenarán con los datos del sol.

2. Obtención de la hora actual en la zona horaria de la ciudad:

```
ahora = datetime.now(pytz.timezone(ciudad.timezone))
```

Se obtiene la hora actual en la zona horaria específica de la ciudad usando pytz.

3. Cálculo de los tiempos de amanecer y atardecer:

```
s = sun(ciudad.observer, date=ahora, tzinfo=ciudad.timezone)
```

Utiliza la función sun de la librería suntime para calcular los horarios de amanecer y atardecer para la ciudad en base al observer (que es el punto geográfico de la ciudad) y la hora actual.

4. Determinación del estado del sol (día o noche):

```
estado = "Día" if s['sunrise'] <= ahora <= s['sunset'] else "Noche"
```

Si la hora actual está entre el amanecer y el atardecer, se considera que es "Día". De lo contrario, es "Noche".

5. Construcción de la información de la ciudad (estado, amanecer, atardecer):

```
info += f"nombre: {ciudad.nombre} estado: {estado} amanecer: {s['sunrise'].strftime('%H:%M:%S')} atardecer: {s['sunset'].strftime('%H:%M:%S')}
```

Se construye un texto con el estado del sol, la hora de amanecer y la hora de atardecer para cada ciudad y se va añadiendo a la variable info.

6. Cálculo de las coordenadas solares (azimut y elevación):

```
Azimuth valor = azimuth(ciudad.observer, ahora)
```

```
Elevation valor = elevation(ciudad.observer, ahora)
```

Se calculan el azimut y la elevación del sol usando las funciones azimuth y elevation. Estas funciones toman el observer (coordenadas geográficas de la ciudad) y la hora actual (ahora) para determinar en qué dirección se encuentra el sol en el cielo (azimut) y su altura (elevación).

7. Construcción de las coordenadas solares para cada ciudad:

```
Coordenadas sol += f"nombre: {ciudad.nombre} Azimut={Azimuth valor:.2f}, Elevación={Elevation valor:.2f} n"
```

Se formatea y añade a la cadena coordenadas sol las coordenadas solares de cada ciudad (azimut y elevación).

8. Actualización de las etiquetas en la interfaz gráfica:

```
Etiqueta info sol.config(text=info) Etiqueta coordenadas sol.config(text=coordenadas sol)
```

Se actualizan las etiquetas en la interfaz gráfica (etiqueta info sol y etiqueta coordenadas sol) con la información recién generada. Estas etiquetas mostrarán el estado del sol, los horarios de amanecer y atardecer, y las coordenadas solares.

9. Actualización periódica:

root.after(60000, actualizar informacion sol) Actualiza cada 60 segundos

La función root.after se utiliza para llamar nuevamente a actualizar informacion sol después de 60,000 milisegundos (60 segundos), asegurando que la información se actualice cada minuto.

```
# Función para actualizar la fecha y hora en la interfaz
def actualizar_fecha_hora():
    etiqueta_fecha_hora.config(text=f"Fecha y Hora: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    root.after(1000, actualizar_fecha_hora)
```

Figura 9: Código para configurar la fecha y hora

```
# Función para iniciar la detección del punto más brillante utilizando binarización
def iniciar_deteccion():
    global cap, camara_encendida, camera_index
    if camara_encendida:
        return # Si la cámara ya está encendida, no hacer nada

    # Usar el índice de cámara seleccionado
    cap = cv2.VideoCapture(int(camera_index))
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 800)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 400)
    camara_encendida = True

    def actualizar_video():
        if not camara_encendida:
            return # Si la cámara está apagada, no actualizar el video

        ret, frame = cap.read()
        if not ret:
            return

        # Convertir el cuadro a escala de grises
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Aplicar binarización para destacar las áreas más brillantes
        _, binarizada = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
```

Figura 10: Código para la detección del punto mas brillante


```

# Encontrar el contorno más grande (área más brillante)
contornos, _ = cv2.findContours(binariizada, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
if contornos:
    contorno_mas_grande = max(contornos, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(contorno_mas_grande)
    # Ajuste para un recuadro de detección más pequeño
    x += int(w * 0.15)
    y += int(h * 0.15)
    w = int(w * 0.5)
    h = int(h * 0.5)
    centro_x = x + w // 2
    centro_y = y + h // 2

    # Dibujar un rectángulo alrededor del área más brillante
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Obtener las dimensiones del cuadro de video y los puntos de referencia
    altura, ancho = gray.shape[:2]
    mitad_ancho = ancho // 2
    mitad_altura = altura // 2
    tolerancia = 50 # Tolerancia para considerar el centro

    # Determinar la posición del punto brillante y enviar comandos
    if abs(centro_x - mitad_ancho) <= tolerancia and abs(centro_y - mitad_altura) <= tolerancia:
        enviar_comando("SOL_CENTRO")
        print("Sol en el centro")
    elif centro_x < mitad_ancho - tolerancia:
        enviar_comando("MOTOR_IZQUIERDA")
        print("Sol a la izquierda")
    elif centro_x > mitad_ancho + tolerancia:
        enviar_comando("MOTOR_DERECHA")
        print("Sol a la derecha")
    elif centro_y < mitad_altura - tolerancia:
        enviar_comando("SERVO_ARRIBA")
        print("Sol arriba")
    elif centro_y > mitad_altura + tolerancia:
        enviar_comando("SERVO_ABAJO")
        print("Sol abajo")

    # Convertir la imagen de OpenCV al formato de Tkinter y mostrarla
    imagen = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    imagen = Image.fromarray(imagen)
    imagen_tk = ImageTk.PhotoImage(image=imagen)
    etiqueta_video.imgtk = imagen_tk
    etiqueta_video.configure(image=imagen_tk)

    etiqueta_video.after(10, actualizar_video)

actualizar_video()

```

Figura 11: Código para la detección del punto mas brillante

Figura 11.- El siguiente código es el encargado de realizar la detección del punto mas brillante que podra encontrar nuestro seguidor solar

Detalle del código

1. Inicio de la cámara:

Código Python:

if camara encendida:

return

- Verifica si la cámara ya está encendida. Si lo está, no hace nada.

Código Python:

```
cap = cv2.VideoCapture(int(camera index))
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 800)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 400)
camara encendida = True
```

- Configura la cámara utilizando un índice proporcionado (camera index).
- Ajusta la resolución del video a 800x400 píxeles.
- Marca la cámara como encendida con camara encendida = True.

2. Función interna actualizar video()

Este es un bucle que captura y procesa cada cuadro del video en tiempo real.

Código Python:

```
if not camara encendida:
    return
```

- Si la cámara está apagada, no actualiza el video.

Código Python:

```
ret, frame = cap.read()
if not ret:
    return
```

- Lee un cuadro de la cámara (frame). Si falla, abandona la actualización.

3. Conversión y binarización:

Código Python:

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
binarizada = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
```

- Convierte el cuadro a escala de grises.
- Binariza la imagen, resaltando las áreas más brillantes (valores de brillo mayores a 200).

4. Detección del contorno más brillante

Código Python:

```
contornos, _ = cv2.findContours(binarizada, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

- Encuentra los contornos en la imagen binarizada.
- Si hay contornos, selecciona el más grande (mayor área).

Código Python:

```
contorno_mas_grande = max(contornos, key=cv2.contourArea)
x, y, w, h = cv2.boundingRect(contorno_mas_grande)
```

- Calcula un rectángulo que encierra el contorno más grande.
- Ajusta las dimensiones del rectángulo para centrar y reducir la detección.

5. Cálculo del centro del rectángulo

Código Python:

```
centro_x = x + w // 2
centro_y = y + h // 2
```

- Calcula las coordenadas del centro del rectángulo.

6. Detección de posición relativa del punto brillante

Código Python:

```
altura, ancho = gray.shape[:2]
mitad_ancho = ancho // 2
mitad_altura = altura // 2
tolerancia = 50
```

- Define el centro del cuadro de video y establece una tolerancia para determinar si el punto está cerca del centro.

Código Python:

```
if abs(centro_x - mitad_ancho) >= tolerancia and abs(centro_y - mitad_altura) >= tolerancia:
    enviar_comando("SOL_CENTRO")
elif centro_x < mitad_ancho - tolerancia:
    enviar_comando("MOTOR_IZQUIERDA")
```

- Según la posición relativa del punto brillante respecto al centro, envía comandos específicos:
 - o Centro: SOL_CENTRO.
 - o Izquierda/Derecha: MOTOR_IZQUIERDA o MOTOR_DERECHA.
 - o Arriba/Abajo: SERVO_ARRIBA o SERVO_ABAJO.

7. Actualización visual

Código Python

```
imagen = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
imagen = Image.fromarray(imagen)
imagen_tk = ImageTk.PhotoImage(image=imagen)
etiqueta_video.imgtk = imagen_tk
etiqueta_video.configure(image=imagen_tk)
```

- Convierte el cuadro procesado de OpenCV a un formato compatible con Tkinter (ImageTk).
- Actualiza una etiqueta (etiqueta_video) para mostrar el cuadro en la interfaz.

8. Bucle de actualización

Código Python

```
etiqueta_video.after(10, actualizar_video)
```

- Llama a “actualizar video” cada 10 milisegundos para continuar procesando cuadros.

```
# Función para apagar la cámara y cerrar la ventana de video
def apagar_camara():
    global cap, camara_encendida
    if camara_encendida and cap is not None:
        camara_encendida = False
        cap.release()
        cv2.destroyAllWindows()
        etiqueta_video.config(image="") # Limpiar la imagen de la interfaz de Tkinter
        print("Cámara apagada")
```

Figura 12: Código para apagar la camara

Figura 12.- La codificación que nos ayuda a apagar la camara, una vez terminamos de utilizarla es la siguiente:

Descripción de la Función apagar_camara()

```
def apagar_camara():
```

- La función no recibe parámetros, ya que está diseñada para trabajar con variables globales que gestionan el estado de la cámara: `cap` y `camara_encendida`.

global `cap`, `camara_encendida`

- **global:** Permite modificar las variables globales `cap` (que representa la captura de la cámara) y `camara_encendida` (un indicador booleano que señala si la cámara está en uso).

Lógica de Apagado:

- `if camara_encendida and cap is not None:`
 - **Cámara encendida:** Sea `True`, lo que indica que la cámara está activada.
 - **Cap:** No sea `None`, lo que confirma que el objeto de captura de video (`cv2.VideoCapture`) está inicializado.
- `camara_encendida = False`
 - Cambia el estado de `camara_encendida` a `False`, indicando que la cámara ya no está activa.
- `cap.release()`
 - Libera los recursos asociados a la cámara para que puedan ser utilizados por otras aplicaciones o instancias.
- `cv2.destroyAllWindows()`
 - Cierra todas las ventanas abiertas por OpenCV que podrían estar mostrando imágenes de la cámara.
- `etiqueta_video.config(image='')`
 - Limpia el contenido de la etiqueta de video en la interfaz gráfica (Tkinter), eliminando cualquier imagen mostrada previamente.
- `print(Çámara apagada")`
 - Imprime un mensaje en la consola para confirmar que la cámara ha sido apagada correctamente.

```
# Función para actualizar el índice de la cámara seleccionado
def actualizar_indice_camara(opcion):
    global camera_index
    camera_index = opcion
```

Figura 13: *Código para actualizar el índice de la cámara*

```
# Configuración de la interfaz gráfica con Tkinter
root = tk.Tk()
root.title("Detección de la zona más luminosa y posición del sol")
root.geometry("1200x600")
# Frame para la visualización del video
```

Figura 14: *Código para controlar la interfaz*

EQUIPO 3- 20 DE SEPTIEMBRE DE 2023A-

Figura 14.- El siguiente código realiza la siguiente función

Interfaz Gráfica y Diseño de Ventanas

1. Creación de la Ventana Principal

```
root = tk.Tk()
root.title("Detección de la zona más luminosa y posición del sol")
root.geometry("1200x600")
```

- `root = tk.Tk()`: Crea la ventana principal de la aplicación, que actúa como el contenedor de todos los elementos gráficos de la interfaz.
- `root.title()`: Define el título de la ventana que aparecerá en la barra superior. En este caso, es Detección de la zona más luminosa y posición del sol.
- `root.geometry()`: Establece el tamaño inicial de la ventana, en este caso 1200 píxeles de ancho por 600 píxeles de alto.

2. Área para Mostrar el Video (Lado Izquierdo)

```
frame_izquierdo = tk.Frame(root)
frame_izquierdo.pack(side=tk.LEFT, padx=10, pady=10)
etiqueta_video = tk.Label(frame_izquierdo)
etiqueta_video.pack()
```

- `frame_izquierdo = tk.Frame(root)`: Crea un marco (contenedor) dentro de la ventana principal, ubicado en el lado izquierdo de la ventana.
 - `padx=10` y `pady=10`: Añaden un margen de 10 píxeles alrededor del marco (a la izquierda y en la parte superior).
- `etiqueta_video = tk.Label(frame_izquierdo)`: Crea una etiqueta dentro del marco, que probablemente se usará para mostrar el video en tiempo real.
- `etiqueta_video.pack()`: Empaqueta y coloca la etiqueta en el marco para que se muestre en la interfaz gráfica.

3. Panel para Mostrar Información Solar (Lado Derecho)

```
frame_derecho = tk.Frame(root)
frame_derecho.pack(side=tk.RIGHT, padx=10, pady=10)
```

- `frame_derecho = tk.Frame(root)`: Crea un marco dentro de la ventana principal para mostrar la información solar, ubicado en el lado derecho de la ventana.
- `padx=10` y `pady=10`: Añaden un margen de 10 píxeles alrededor del marco.

4. Etiquetas para Mostrar Información Sobre el Sol

```
etiqueta_info_sol = tk.Label(frame_derecho, text=, justify=tk.LEFT, font=(.Arial", 10))
etiqueta_info_sol.pack()
etiqueta_coordenadas_sol = tk.Label(frame_derecho, text=, justify=tk.LEFT, font=(.Arial",
10))
etiqueta_coordenadas_sol.pack()
etiqueta_fecha_hora = tk.Label(frame_derecho, text=, font=(.Arial", 12))
etiqueta_fecha_hora.pack()
```

- `etiqueta_info_sol`: Etiqueta que se utiliza para mostrar información relacionada con el sol, como su estado o descripción.
 - `text=`: Inicialmente no tiene texto, pero se actualizará dinámicamente.
 - `justify=tk.LEFT`: Alinea el texto hacia la izquierda dentro de la etiqueta.
 - `font=(.Arial", 10)`: Define la fuente del texto como *Arial* con un tamaño de 10 puntos.
- `etiqueta_coordenadas_sol`: Similar a la etiqueta anterior, esta probablemente mostrará las coordenadas del sol o su ubicación en algún sistema de referencia.
- `etiqueta_fecha_hora`: Etiqueta para mostrar la fecha y la hora actuales, con un tamaño de fuente de 12 puntos.

5. Espacio para los Botones de Interacción

```
frame_botones = tk.Frame(frame_derecho)
frame_botones.pack(pady=20)
```

- `frame_botones`: Crea un nuevo marco dentro del panel derecho para alojar los botones de la interfaz (aunque no se definen aquí, es probable que estén relacionados con la interacción del usuario, como iniciar o detener la detección).
- `pady=20`: Añade un margen de 20 píxeles en la parte superior e inferior del marco para separar los botones de otros elementos.


```

# Menú desplegable para seleccionar el índice de la cámara
opciones_camara = ["0", "1"] # Puedes agregar más índices si tienes más cámaras
camera_index_var = StringVar(root)
camera_index_var.set(opciones_camara[0]) # Índice de cámara predeterminado

etiqueta_camara = tk.Label(frame_botones, text="Selecciona la Cámara:")
etiqueta_camara.pack()

menu_camara = OptionMenu(frame_botones, camera_index_var, *opciones_camara, command=actualizar_indice_camara)
menu_camara.pack()

# Botón para encender la cámara
boton_encender = Button(frame_botones, text="Encender Cámara y Detección", command=iniciar_deteccion, bg="green")
boton_encender.pack(pady=5)

# Botón para apagar la cámara
boton_apagar = Button(frame_botones, text="Apagar Cámara", command=apagar_camara, bg="red")
boton_apagar.pack(pady=5)

# Botón para cerrar la aplicación
boton_cerrar = Button(frame_botones, text="Cerrar Programa", command=root.destroy)
boton_cerrar.pack(pady=5)

# Iniciar las actualizaciones de la información del sol y fecha/hora
actualizar_informacion_sol()
actualizar_fecha_hora()

# Iniciar el bucle de la interfaz gráfica
root.mainloop()

```

Figura 15: Código para controlar el menú

Figura 15.- Esta codificación nos muestra el Menú desplegable para seleccionar la cámara

Opciones de Cámara e Interacción con la Interfaz

1. Opciones de Cámara

```

opciones_camara = ['0', '1']           Podemos agregar más índices si tuviéramos más cámaras.
camera_index_var = StringVar(root)
camera_index_var.set(opciones_camara[0])      Índice de cámara predeterminado.
etiqueta_camara = tk.Label(frame_botones, text="Selecciona la Cámara:")
etiqueta_camara.pack()
menu_camara = OptionMenu(frame_botones, camera_index_var, *opciones_camara,
command=actualizar_indice_camara)

```

`menu_camara.pack()`

- `opciones_camara`: Lista de índices de cámaras disponibles. Por defecto, incluye las cámaras con índice 0 y 1, pero se pueden añadir más si hay más dispositivos conectados.
- `camera_index_var`: Variable asociada al menú desplegable para almacenar el índice seleccionado.
 - `StringVar(root)`: Crea una variable de tipo cadena asociada a la ventana principal.
 - `set(opciones_camara[0])`: Establece el índice 0 como predeterminado.
- `tk.Label(frame_botones)`: Etiqueta que muestra el texto *Selecciona la Cámara*.
- `OptionMenu`:
 - Genera un menú desplegable dentro del marco `frame_botones`.
 - `camera_index_var`: Variable que almacena la selección.
 - `opciones_camara`: Despliega las opciones de la lista `opciones_camara`.
 - `command=actualizar_indice_camara`: Ejecuta la función `actualizar_indice_camara` al seleccionar una opción, lo que permite cambiar el índice de la cámara activa.

2. Botones de Interacción

```

boton_encender = Button(frame_botones, text="Encender Cámara y Detección", command=
iniciar_deteccion, bg="green")
boton_encender.pack(pady=5)
boton_apagar = Button(frame_botones, text="Apagar Cámara", command=apagar_camara, bg="red")
boton_apagar.pack(pady=5)
boton_cerrar = Button(frame_botones, text="Cerrar Programa", command=root.destroy)
boton_cerrar.pack(pady=5)

```

- `Button(frame_botones)`: Crea botones para interactuar con la aplicación. Cada botón tiene un texto, una función asociada y un color de fondo (`bg`).
 - `boton_encender`: *Encender Cámara y Detección*.
 - `command=iniciar_deteccion`: Ejecuta la función `iniciar_deteccion` para encender la cámara y comenzar la detección.

- Fondo: Verde (`bg="green"`).
- `boton_apagar`: *Apagar Cámara*.
 - `command=apagar_camara`: Llama a la función `apagar_camara` para detener la cámara y limpiar la visualización.
 - Fondo: Rojo (`bg=red"`).
- `boton_cerrar`: *Cerrar Programa*.
 - `command=root.destroy`: Finaliza el programa y cierra la ventana principal.
- `pack(pady=5)`: Añade un margen vertical de 5 píxeles entre los botones para que no estén pegados.

3. Inicialización de Actualizaciones Automáticas

`actualizar_informacion_sol()`

`actualizar_fecha_hora()`

- `actualizar_informacion_sol`: Inicia la actualización periódica de la información solar (como estado, horarios de amanecer y atardecer, azimut y elevación).
- `actualizar_fecha_hora`: Arranca la actualización automática de la fecha y hora actual en la interfaz.

4. Bucle Principal de la Interfaz

`root.mainloop()`

- `root.mainloop()`: Inicia el bucle de eventos de Tkinter, permitiendo que la aplicación responda a las interacciones del usuario (como clics de botones o cambios en el menú desplegable). Mientras este bucle está activo, la ventana permanece abierta y funcional.

Video:



Figura 16: Video

<https://youtu.be/5Ya1zkeWAE0>

6. Conclusiones

■ Marcos Alberto Coronado Lopez 1949489 IMTC

El desarrollo de este proyecto ha sido una técnica que involucró múltiples disciplinas y desafíos. Uno de los primeros retos fue configurar la cámara para detectar la posición del sol. Utilizando OpenCV y filtros HSV pudimos lograr identificar de manera precisa la luz amarilla más brillante, pero no sin antes realizar numerosos ajustes y pruebas para asegurar la fiabilidad del sistema en diversas condiciones de iluminación.

La laptop que se usa para procesar, representó otro aspecto relevante. Diseñar una interfaz gráfica con Tkinter no solo facilitó el manejo del sistema, sino que también permitió que podamos integrar los datos visuales con los comandos enviados al ESP32. Este microcontrolador coordina los movimientos del servo motor y el motor a pasos, lo cual requirió dominar la comunicación serial y sincronizar los comandos para asegurar que el panel solar respondiera con precisión.

Implementar el driver A4988 para controlar el motor a pasos agregó un nivel de dificultad, ya que fue necesario ajustar los pasos y la dirección del movimiento para que fueran consistentes con la detección visual del sol. Surgieron algunos desafíos mecánicos al montar el sistema porque era esencial garantizar la estabilidad del panel solar mientras se ajustaban los ejes horizontal y vertical.

■ Rene Guzman Perez 1849342 IMTC

En este proyecto, se desarrolló un sistema inteligente de seguimiento solar basado en visión por computadora y control mediante un ESP32. Utilizando una cámara para detectar la luz más brillante, se integraron algoritmos de procesamiento de imágenes que permiten identificar la posición del sol o de una fuente luminosa simulada (luz amarilla). El sistema traduce esta información en movimientos de un panel solar a través de actuadores como un servo motor y un motor a pasos, optimizando la captación de energía solar.

La implementación de una interfaz gráfica en Python con Tkinter facilitó el control del sistema, permitiendo al usuario supervisar la detección en tiempo real y enviar comandos al ESP32. Por otro lado, el procesamiento de imágenes con OpenCV, junto con la comunicación serial, demostró ser una solución eficiente para la integración entre software y hardware.

Este proyecto destaca por su enfoque interdisciplinario, combinando conceptos de visión por computadora, programación, electrónica y control de motores. Además, su diseño modular asegura que pueda ser mejorado o ampliado en el futuro, como incluir nuevas funciones de predicción solar mediante modelos de machine learning o expandir su uso para aplicaciones similares en robótica o automatización industrial.

En conclusión, este sistema representa una solución práctica y eficiente para aplicaciones de seguimiento solar, contribuyendo al desarrollo de tecnologías más sostenibles y al aprendizaje integral de las disciplinas involucradas.

- Adrian Guevara Aguayo 1942809 IMTC

En conclusión un sistema de captación solar que utiliza un servomotor para orientar un panel solar en función de la luz disponible, junto con un sistema mecatrónico, optimiza la eficiencia de la captación energética. El servomotor permite ajustar la inclinación y orientación del panel en tiempo real, siguiendo el movimiento del sol y asegurando una mayor exposición a la luz solar durante todo el día. Este enfoque mecatrónico, que integra sensores, actuadores y algoritmos de control, mejora significativamente la generación de energía, aumentando la autonomía y eficiencia del sistema, lo que resulta en una solución más sostenible y rentable para la captación solar.

- Alexander Reyes Hernandez 1853430 IMTC:

Lo que aprendí en la realización de este proyecto, es que puedo decir que construir un seguidor solar con visión artificial fue todo un reto, pero a la vez una experiencia muy interesante. En la parte del seguimiento solar, se usaron herramientas como Python para procesar imágenes en tiempo real por medio de la cámara y que esta pueda ubicar la posición del sol. En nuestra documentación describimos por tal motivo lo que significaba cada código porque realmente utilizamos muchos comandos,

El diseño de la estructura fue otro punto importante. Se logró construir una base resistente y funcional, con cosas que teníamos en casa, capaz de soportar tanto el panel solar como los componentes electrónicos. Además, los motores que controlan el movimiento de los ejes se conectaron al sistema de visión artificial, y también se añadieron sensores de luz para complementar la información y mejorar la precisión.

Y ya para terminar, agregamos lo que se nos solicitó y colocamos unos paneles solares para después se pueda conectar unas baterías para almacenar la energía y así asegurar que pueda operar sin depender de una fuente externa.

Referencias

- [1] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media. Referencia clave para el desarrollo de algoritmos de visión por computadora aplicados en la detección de luz amarilla.
- [2] Cattaneo, C. A., Larcher, L. I., Ruggeri, A. I., Herrera, A. C., Biasoni, E., and Escañuelas, M. (2010). Segmentación de imágenes digitales mediante umbralizado adaptivo en imágenes de color. *Mecánica Computacional*, 29(62):6177–6193.
- [3] DataHacker (2020). 009 how to detect facial landmarks using dlib and opencv. <https://datahacker.rs/009-how-to-detect-facial-landmarks-using-dlib-and-opencv/>. Fecha de consulta: 01/11/2023.
- [4] Duffie, J. A. and Beckman, W. A. (2013). *Solar Engineering of Thermal Processes*. Wiley, 4th edition. Base teórica sobre la importancia del seguimiento solar en sistemas fotovoltaicos y su impacto en la eficiencia energética.
- [5] GeeksforGeeks (2020). Python opencv – drawkeypoints() function. <https://www.geeksforgeeks.org/python-opencv-drawkeypoints-fuction/>. Fecha de consulta: 02/11/2023.
- [6] GeeksforGeeks (2021). Python opencv – find center of contour. <https://www.geeksforgeeks.org/python-opencv-find-center-of-contour/>. Fecha de consulta: 01/11/2023.
- [7] Magro, R. (2013). Binarización de imágenes digitales y su algoritmia como herramienta aplicada a la ilustración entomológica. *Boletín de la SEA*, (53):443–464.
- [8] OpenCV (2019). Drawing functions in opencv. https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html. Fecha de consulta: 02/11/2023.
- [9] Pugh, D. (2017). *Practical Arduino Engineering: End-to-End Development with Arduino, LoRa, and IoT Cloud Platforms*. Apress. Implementación del driver A4988 y conceptos sobre el control preciso de motores a pasos.
- [10] Rosebrock, A. (2017). Detect eyes, nose, lips, and jaw with dlib, opencv, and python. <https://pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/>. Fecha de consulta: 01/11/2023.
- [11] Schwartz, R. (2019). *ESP32 Programming for the Internet of Things: ESP32 Development Step by Step*. Independently Published. Guía para configurar el ESP32 como controlador central para la comunicación serial y el control de motores.
- [12] Solano, G. (2019). Rgb en opencv – python. <https://omes-va.com/rgb/>. Fecha de consulta: 02/11/2023.

- [13] Zelle, J. (2010). *Python Programming: An Introduction to Computer Science*. Franklin, Beedle & Associates, 2nd edition. Conceptos de programación en Python aplicados en la interfaz gráfica con Tkinter y en el manejo de algoritmos de visión.

Los créditos de las fotografías pertenecen a sus respectivos autores.

EQUIPO 3- 20 DE SEPTIEMBRE DE 2025-A- 23:41