Informe Proyecto Integrador de Saberes: Cargador Solar Automatizado

Fabricio Ruiz A. – Cristhian Dávila S. – Ariana Córdova B. – Yorely Romero C. - Juan Encalada C.

Resumen

El presente informe detalla el desarrollo y la implementación del pseudocódigo y el código en lenguaje C que se utilizará para controlar un cargador solar automatizado. Este dispositivo está diseñado para proporcionar una solución eficiente y sostenible para la carga de dispositivos electrónicos utilizando energía solar.

Abstract

This report details the development and implementation of the pseudocode and C language code that will be used to control an automated solar charger. This device is designed to provide an efficient and sustainable solution for charging electronic devices using solar energy.

1. Definición del Problema:

Un factor de importancia para la eficiencia de un panel solar, se encuentra ligada a su capacidad para capturar la mayor cantidad posible de energía solar, esto puede variar debido a factores como la posición del sol, el clima y la hora del día. Por lo tanto, podemos decir que la orientación e inclinación de nuestro panel solar afecta directamente en su rendimiento, limitando así la producción de energía.

Tanto la orientación como inclinación solar que presentará nuestro panel solar se basará y dependerá mucho en el algoritmo o código planteado dentro del proyecto, su movimiento y seguimiento hacia el sol definirá la eficacia del proyecto, si el algoritmo es erróneo, no aplica las fórmulas correctas, no toma en cuenta las distintas variables que pueden existir en la orientación del sol, o no toma en cuenta la importancia de factores como la rotación de la tierra y su órbita alrededor del sol, nos presentará coordenadas de orientación, y posición erróneas, impidiendo así captar la posición del sol correcta, lo que nos lleva a no obtener las ventajas adecuadas de la radiación solar.

La necesidad de predecir y adaptarse a estos cambios en tiempo real, plantea, desafíos adicionales en la programación de sistemas de energía solar, para garantizar un suministro constante y confiable de electricidad.

El problema se complica aún más, debido a la variabilidad geográfica. Las coordenadas geográficas de la ubicación de los paneles solares determinan la altitud y la dirección del sol en ese lugar específico en un momento dado. Por lo tanto, la orientación óptima de los paneles solares variará según su ubicación geográfica, lo que requiere un enfoque personalizado para cada sitio de instalación.

En resumen, el no tener desarrollado un algoritmo adecuado para que el panel solar pueda captar el movimiento y radiación del sol, implica tener varios problemas como la limitación de la

captación de energía hasta la gestión de la eficiencia. La capacidad para adaptarse a las condiciones que varían constantemente y mejorar el rendimiento son esenciales para obtener un funcionamiento con el cual se pueda maximizar la eficiencia en la captación de la luz solar.

2. Objetivo del algoritmo:

El objetivo principal se centra en el desarrollo de un algoritmo que mediante técnicas, enfoques algorítmicos y modelos matemáticos pueda calcular la posición y orientación óptima del sol, incluyendo métodos basados en trigonometría, para maximizar la eficiencia en la captación de luz solar en cualquier momento dado, considerando factores como la ubicación geográfica (latitud y longitud), la hora del día y las condiciones climáticas.

3. Análisis e influencia de Variables:

Fecha y Hora:

- tiempo_actual: Almacena el tiempo actual.
- tiempo_local: Estructura que contiene la fecha y hora local desglosada.
- diasTranscurridos: Número de días transcurridos en el año actual.

Estas variables influyen de forma que determinan la fecha y hora actual, que son esenciales para calcular los días transcurridos del año (diasTranscurridos) y la hora en formato decimal (horaDecimal). La posición del sol varía a lo largo del año y del día, por lo que la fecha y la hora son fundamentales para determinar la posición solar.

Coordenadas Geográficas

- latitud: Es la distancia angular medida en grados desde cualquier punto de la Tierra hasta el ecuador, tanto hacia el norte como hacia el sur. Se utiliza para determinar la ubicación de un lugar en el planeta en relación con el ecuador. Influye en los cálculos debido a su efecto en la posición del sol en el cielo.

- longitud: Es la distancia angular medida en grados, minutos y segundos desde el meridiano de referencia hasta el meridiano que pasa por un punto específico en la superficie de la Tierra. Influye en los cálculos debido a su efecto en la posición del sol en el cielo.

La latitud influye directamente en el cálculo del Ángulo de Inclinación Solar y el Azimut Solar. La longitud es necesaria para ajustar el tiempo solar verdadero, ya que el mediodía solar depende de la longitud geográfica.

Constante:

PI: El valor de pi, utilizado para las conversiones entre grados y radianes.

Utilizado para convertir grados a radianes y viceversa, se considera una constante esencial y fundamental. Las funciones trigonométricas en los cálculos requieren los ángulos en radianes.

Variables para Cálculos Astronómicos (método SPA):

- termino2radian: se utiliza como un término intermedio en el cálculo de la declinación solar (declinacionSolar). Esta variable es el resultado de una

conversión específica que involucra el número de días transcurridos en el año y se usa para calcular la declinación solar, la cual describe la inclinación del sol sobre el ecuador terrestre. Por lo que influye en una serie de cálculos astronómicos.

- declinacionSolar: Es la variable que sostiene el valor de la declinación solar, la cual es la medida del ángulo que forma la posición aparente del Sol con respecto al plano del ecuador terrestre en un momento dado. Es un factor importante en el cálculo del ángulo de inclinación solar y el azimut solar, ya que determina la posición del sol en relación con el ecuador terrestre, lo que a su vez afecta la altura máxima del sol en el cielo y su posición en el horizonte.

- ecuacionTiempo: Corrección por la Ecuación del Tiempo en minutos. Es un factor importante a considerar en el cálculo del ángulo de inclinación solar y el azimut solar, ya que puede afectar la hora solar real en un lugar específico en un momento dado, lo que a su vez influye en la posición aparente del sol en el cielo y en el horizonte.

- horaDecimal: Hora del día en formato decimal (horas y fracciones de hora). Se calcula como: Hora decimal= HoraActual + Minutos/60. Influye directamente en los cálculos relacionados con la determinación de la posición del sol en el cielo a lo largo del día. Se calcula como la hora actual en formato decimal, incluyendo fracciones de hora para tener mayor precisión. Esta variable es esencial para varios pasos en el algoritmo, especialmente en la conversión de tiempo local a tiempo solar verdadero, y en la determinación del ángulo horario del sol.

- tiempoSolarVerdadero: Tiempo solar verdadero en horas. Basado en la posición real del sol en el cielo, es esencial para calcular con precisión estos ángulos, fundamentales para diversas aplicaciones como la energía solar, la agricultura y la navegación.

- AHR: Ángulo Horario en radianes. Es una medida angular que indica la posición del sol en el cielo en relación con un observador en un lugar particular y en un momento específico. Se calcula como la diferencia angular entre la línea de posición del sol y el meridiano local del observador.

- declinacionSolarRad: Declinación solar en radianes. Se obtiene al convertir la declinación solar de grados a radianes. Este ángulo describe la inclinación del sol sobre el ecuador terrestre y varía a lo largo del año. Es crucial para determinar la altitud solar y el ángulo horario del sol. La declinación solar influye en la cantidad de energía solar que llega a una superficie en un momento dado, afectando directamente la eficiencia de sistemas solares y el diseño de estructuras arquitectónicas que buscan maximizar la luz natural.

- LatitudPuntoRad: Latitud del punto en radianes. Es la conversión de la latitud geográfica a radianes. Es fundamental para cálculos astronómicos precisos. La latitud del observador influye en la altitud y el azimut del sol. Esto afecta la trayectoria solar en el cielo y es

crucial para determinar el ángulo de incidencia de la radiación solar en una superficie específica.

- AIS_grados: Ángulo de Inclinación Solar en grados. es el ángulo entre la superficie de la Tierra y los rayos solares incidentes. Este ángulo varía según la hora del día, la latitud y la época del año. Determina la intensidad de la radiación solar recibida en una superficie. Un ángulo de inclinación adecuado es crucial para optimizar la captación de energía solar en paneles fotovoltaicos y para el diseño eficiente de edificios en términos de iluminación y calefacción natural.

- AIS_rad: Ángulo de Inclinación Solar en radianes. La conversión del ángulo de inclinación solar de grados a radianes, es necesaria para realizar cálculos trigonométricos precisos. Influye en el cálculo de la altitud solar y el azimut solar. El ángulo de inclinación solar determina la eficiencia con la que la radiación solar impacta una superficie inclinada, afectando directamente el rendimiento de sistemas solares y la planificación de sombras en estructuras arquitectónicas.

- Azimut_grados: Azimut Solar en grados. El azimut solar es un término que describe la dirección en la que se encuentra el sol en el horizonte en un momento específico y en un lugar determinado. Se mide en grados en relación con el norte verdadero, y puede variar desde 0° (apuntando al norte) hasta 360°, donde 0° y 360° representan el norte, 90° el este, 180° el sur y 270° el oeste.

Las variables astronómicas utilizadas en el algoritmo son fundamentales para calcular la posición precisa del sol en el cielo en cualquier momento del día y en cualquier ubicación geográfica. El `termino2radian` se emplea para convertir grados a radianes, un paso esencial en las fórmulas trigonométricas que calculan la `declinacionSolar`, la cual describe la inclinación del sol respecto al ecuador terrestre y varía a lo largo del año, influyendo en la altura solar diaria. La `ecuacionTiempo` ajusta la discrepancia entre el tiempo solar verdadero y el tiempo medio, corrigiendo la hora decimal (`horaDecimal`) para obtener el `tiempoSolarVerdadero`, que es crucial para determinar el `AHR` (Ángulo Horario del Sol). Las conversiones a radianes de `declinacionSolarRad` y `LatitudPuntoRad` son necesarias para las funciones trigonométricas que calculan el `AIS_grados` (Ángulo de Inclinación Solar), el cual afecta la cantidad de energía solar que recibe una superficie. Finalmente, el `Azimut_grados` proporciona la dirección del sol en el plano horizontal, un dato vital para la correcta orientación de paneles solares y para maximizar la eficiencia de la captación de luz solar en aplicaciones de arquitectura y agricultura. Cada una de estas variables interrelacionadas permite al algoritmo modelar con precisión la posición solar y así optimizar diversas aplicaciones dependientes de la energía solar.

Estas variables y sus interrelaciones permiten calcular con precisión la posición del sol en el cielo en cualquier momento y lugar, lo cual es esencial para aplicaciones en astronomía,

energía solar, agricultura y arquitectura.

4. Estructuras de Datos:

struct Coordenadas

Se utiliza para almacenar y gestionar las coordenadas geográficas (latitud y longitud) de puntos.
- Contiene dos elementos de tipo doble
- Latitud: Específica del punto en grados
- Longitud: Muestra la longitud del punto en grados

Uso

- La variable de estructura llamada "coordenadas" se declara en la función main().
- El usuario solicita latitud y longitud utilizando la entrada estándar ("scanf") y las almacena en una estructura de latitud-longitud.

struct tm

La estructura tm se utiliza para gestionar y manipular fechas y horas. Está definido en la biblioteca estándar de C .
- Obtiene y gestiona la fecha y hora actual
- Accede a componentes específicos de fecha y hora (día, mes, año, hora, minuto, segundo).
- Muestra la fecha, hora y número de días del año en un formato legible para el usuario.

Para obtener la fecha y hora actuales, utilice las funciones de **time** (hora) y **localtime** (hora local).

Funciones

- **mostrarDiasTranscurridosDelAno():**
Esta función muestra el número de días transcurridos desde el inicio del año hasta la fecha actual.

- **mostrarFechaActual():**
Esta función muestra la fecha actual en formato `dd-mm-aaaa´

- **mostrarHoraActual()**
Esta función muestra la hora actual en formato `hh:mm:ss

- **calcularDeclinacionSolar(int diasTranscurridos):**
Esta función calcula y devuelve la declinación solar para un día específico del año, en grados.

- **calcularEcuacionDelTiempo(int diasTranscurridos):**
Esta función calcula y devuelve la ecuación del tiempo para un día específico del año, en minutos.

- **calcularTiempoSolarVerdadero(double horaDecimal, double longitudPunto, double ecuacionTiempo):**
Calcula y devuelve el Tiempo Solar Verdadero (TSV) en horas.

- **calcularAnguloHorario(double tiempoSolarVerdadero):**
Calcula y devuelve el ángulo horario en radianes.

- **convertirARadianes(double grados):**
Convierte un ángulo de grados a radianes.

- **convertirAGrados(double radianes):** Convierte un ángulo de radianes a grados.

- **calcularAnguloInclinacionSolar(double declinacionSolarRad, double LatitudPuntoRad, double AHR):** Calcula y devuelve el ángulo de inclinación solar en grados.

- **calcularAzimutSolar(double declinacionSolarRad, double AIS_rad, double LatitudPuntoRad, double AHR):** Calcula y devuelve el azimut solar en grados.

Cómo compila

1. Solicita al usuario ingresar la longitud y latitud del lugar.

2. Realiza varios cálculos astronómicos (declinación solar, ecuación del tiempo, tiempo solar verdadero, ángulo horario, ángulo de inclinación solar y azimut solar).

3. Muestra los resultados de estos cálculos al usuario.

Comparación

En PSeInt, la función está definida por la función y finaliza con la función final. En C, #include se usa para incluir las bibliotecas necesarias y #define se usa para definir constantes. Las funciones se definen como void, int o el tipo de retorno apropiado.

PseInt:

// Función para conteo de días //

Funcion diasPorMes ← conteoDiasMes (mes)

Fin funcion

C:

```
#include <stdio.h>

#include <time.h>

#include <math.h>

#define PI 3.14159265

// Definición de la estructura Coordenadas

Struct Coordenadas {

   Double latitud;

   Double longitud;

};
```

PSeInt utiliza funciones predefinidas como FechaActual(). En C, se utilizan las funciones de la biblioteca time.h para obtener y manipular el tiempo.

PSeInt:

Fecha = FechaActual()

C:

```
Time_t tiempo_actual = time(NULL);

Struct tm *tiempo_local = localtime(&tiempo_actual);
```

Se define una función específica en PSeInt para calcular el número de días que han transcurrido entre meses. En C, puede obtener directamente el número de días desde el comienzo del año utilizando tm_yday en la estructura tm.

PseInt:

diasTranscurridos = calculoEntreMeses(mesInicial, mesactual, diaInicial, diaactual)

C:

diasTranscurridos = tiempo_local->tm_yday + 1;

Ambas implementaciones usan la función cos, pero C debe incluir la biblioteca math.h y usar una sintaxis específica de C.

PseInt:

declinacionSolar= -23.44 * cos(termino2radian)

C:

declinacionSolar = -23.44 * cos(termino2radian);

Las funciones trigonométricas PSeInt usan sen y cos, mientras que las funciones C sen y cos de math.h.

PseInt:

EoT= 9.87*sen(2*BAH)-7.53*cos(BAH)-1.5*sen(BAH)

C:

EoT = 9.87 * sin(2 * BAH) – 7.53 * cos(BAH) – 1.5 * sin(BAH);

Ambas implementaciones utilizan fórmulas similares para calcular el tiempo solar verdadero (TSV), pero adaptadas a la sintaxis de cada lenguaje.

PseInt:

TSV= Horadecimal+(4*(LongitudPunto-LongitudEstandar)+EoT)/60

C:

TSV = horaDecimal + (coordenadas.longitud – longitudEstandar) / 15.0 + ecuacionTiempo / 60.0;

La función asin en Math.h se usa en C para calcular ángulos en radianes, similar a asen PSeInt.

PseInt:

AIS= asen(sen(declinasionSolarRad)*sen(LatitudPuntoRad)+cos(declinacionSolarRad)*cos(LatitudPuntoRad)*cos(AHR))

C:

AIS_rad = asin(sin(declinacionSolarRad) * sin(LatitudPuntoRad) + cos(declinacionSolarRad) * cos(LatitudPuntoRad) * cos(AHR));

Ambos códigos utilizan acos para calcular el azimut del sol, pero ajustan el valor según la sintaxis y las funciones matemáticas de cada idioma.

PseInt:

Azimut= acos((sen(declinacionSolarRad)-sen(AIS)*sen(LatitudPuntoRad))/(cos(AIS)*cos(LatitudPuntoRad)))

C:

Azimut_rad = acos((sin(declinacionSolarRad) * cos(LatitudPuntoRad) – cos(declinacionSolarRad) *

sin(LatitudPuntoRad) * cos(AHR)) / cos(AIS_rad));

Bibliotecas:

Las bibliotecas en C son colecciones de funciones y macros precompiladas que proporcionan una manera de reutilizar código y funcionalidades comunes sin necesidad de reescribirlas desde cero. Las bibliotecas pueden ser de dos tipos: bibliotecas estándar y bibliotecas personalizadas (o de terceros).

En este caso se ha utilizado tres bibliotecas:

- #include <stdio.h>:

Proporciona funcionalidades para realizar operaciones de entrada y salida estándar, como imprimir en la consola (printf) y leer desde la entrada estándar (scanf).

- #include <time.h>:

Proporciona funciones para manipular y obtener información sobre el tiempo y la fecha, como obtener la hora actual (time), medir el tiempo transcurrido (clock), y manipular estructuras de tiempo (struct tm).

- #include <math.h>:

Proporciona funciones matemáticas comunes, como operaciones trigonométricas (sin, cos, tan), logarítmicas (log, exp), y otras funciones matemáticas (sqrt, pow).

- #include <windows.h>:

Permite el acceso a las funcionalidades del sistema operativo Windows, incluyendo la manipulación de archivos, procesos, sincronización, y otras operaciones específicas del sistema operativo.

4. Pseudocódigo y diagramas de flujo del algoritmo desarrollado:

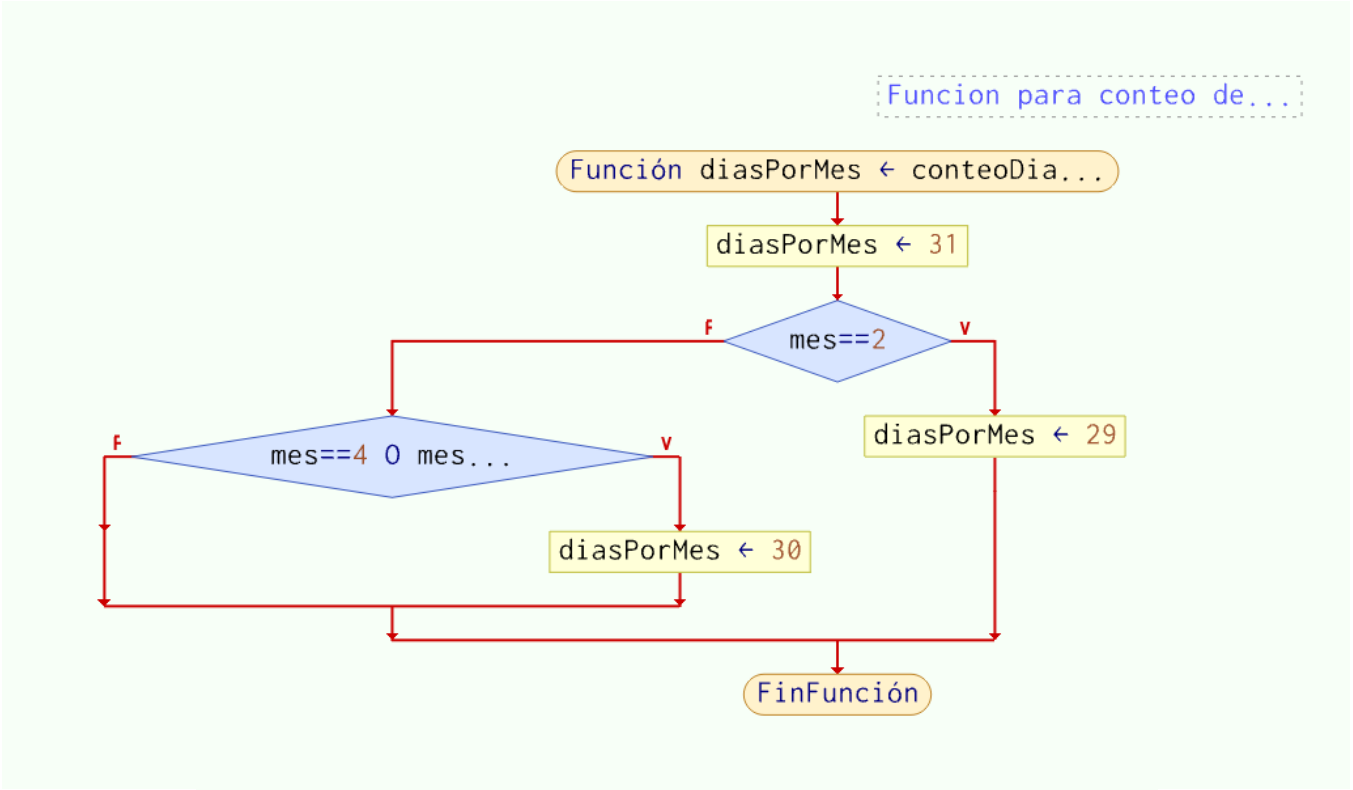
Pseudocódigo desarrollado

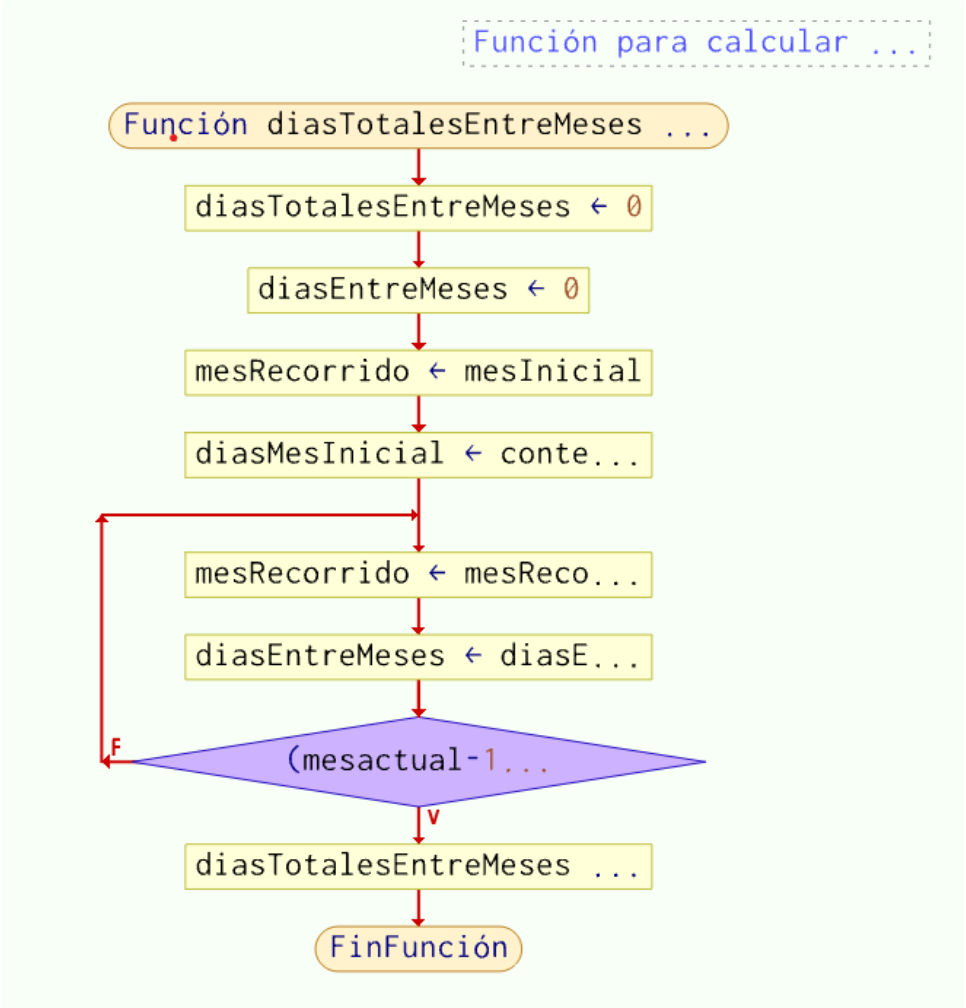
```
// Funcion para conteo de dias //
Funcion diasPorMes ← conteoDiasMes ( mes )
    diasPorMes=31
    si mes==2 Entonces
        diasPorMes=29
    sino
        si mes==4 o mes==6 o mes==9 o mes==11 Entonces
            diasPorMes=30
        fin si
    FinSi
fin funcion

// Función para calcular los días totales entre meses //
Funcion diasTotalesEntreMeses ← calculoEntreMeses ( mesInicial,
mesActual, diaInicial, diaActual)
    diasTotalesEntreMeses=0
    diasEntreMeses=0
    mesRecorrido=mesInicial
    diasMesInicial=conteoDiasMes(mesInicial);
    Repetir
        mesRecorrido=mesRecorrido+1
```

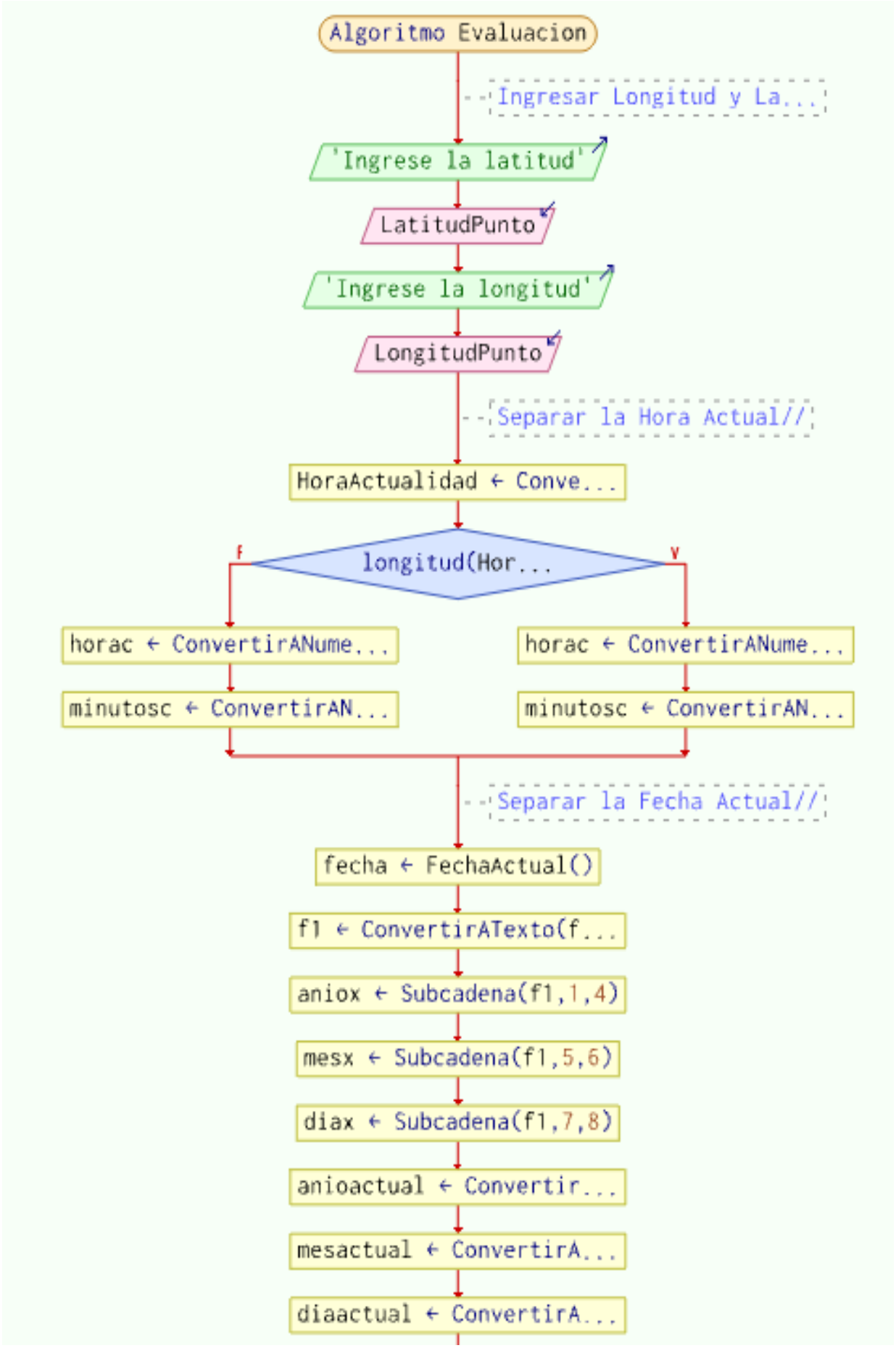
```
diasEntreMeses=diasEntreMeses+conteoDiasMes(mesRecorrido)
    Hasta Que (mesActual-1)=mesRecorrido

diasTotalesEntreMeses=diasEntreMeses+diasMesInicial+diaActual
fin funcion

Algoritmo Evaluacion
    //Ingresar Longitud y Latitud//
    Escribir "Ingrese la latitud"
    Leer latitudPunto
    Escribir "Ingrese la longitud"
    Leer longitudPunto

    //Separar la Hora Actual//
    HoraActualidad= ConvertirATexto(HoraActual())
    Si longitud(HoraActualidad)=5 Entonces
        horac= ConvertirANumero(subcadena(HoraActualidad, 1, 1))
        minutosc= ConvertirANumero(Subcadena(HoraActualidad, 2,
3))
    SiNo
        horac= ConvertirANumero(subcadena(HoraActualidad, 1, 2))
        minutosc= ConvertirANumero(Subcadena(HoraActualidad, 3,
4))
    FinSi

    //Separar la Fecha Actual//
    fecha = FechaActual()
    f1 = ConvertirATexto(fecha)
    aniox= Subcadena(f1, 1, 4)
    mesx= Subcadena(f1, 5, 6)
    diax= Subcadena(f1, 7, 8)
    anioactual= ConvertirANumero(aniox)
    mesactual=ConvertirANumero(mesx)
    diaactual=ConvertirANumero(diax)
    // Obtener la hora actual en formato decimal //
    Horadecimal= horac + (minutosc/60)

    //-------------------------------------------------------------
-------------------------------------------------------------
----------//
    // número de día del año//
    mesInicial= 01
    diaInicial= 01
    anioInicial = anioactual
    diasTranscurridos = calculoEntreMeses(mesInicial, mesactual,
diaInicial, diaactual)

    //-------------------------------------------------------------
-------------------------------------------------------------
----------//

    //Verificar si el año actual es bisiesto//
    Si anioactual % 4 == 0 y anioactual % 100 ≠ 0 o anioactual %
400 == 0 Entonces
        AñonesBisiesto = Verdadero
    Sino AñonesBisiesto = Falso
```

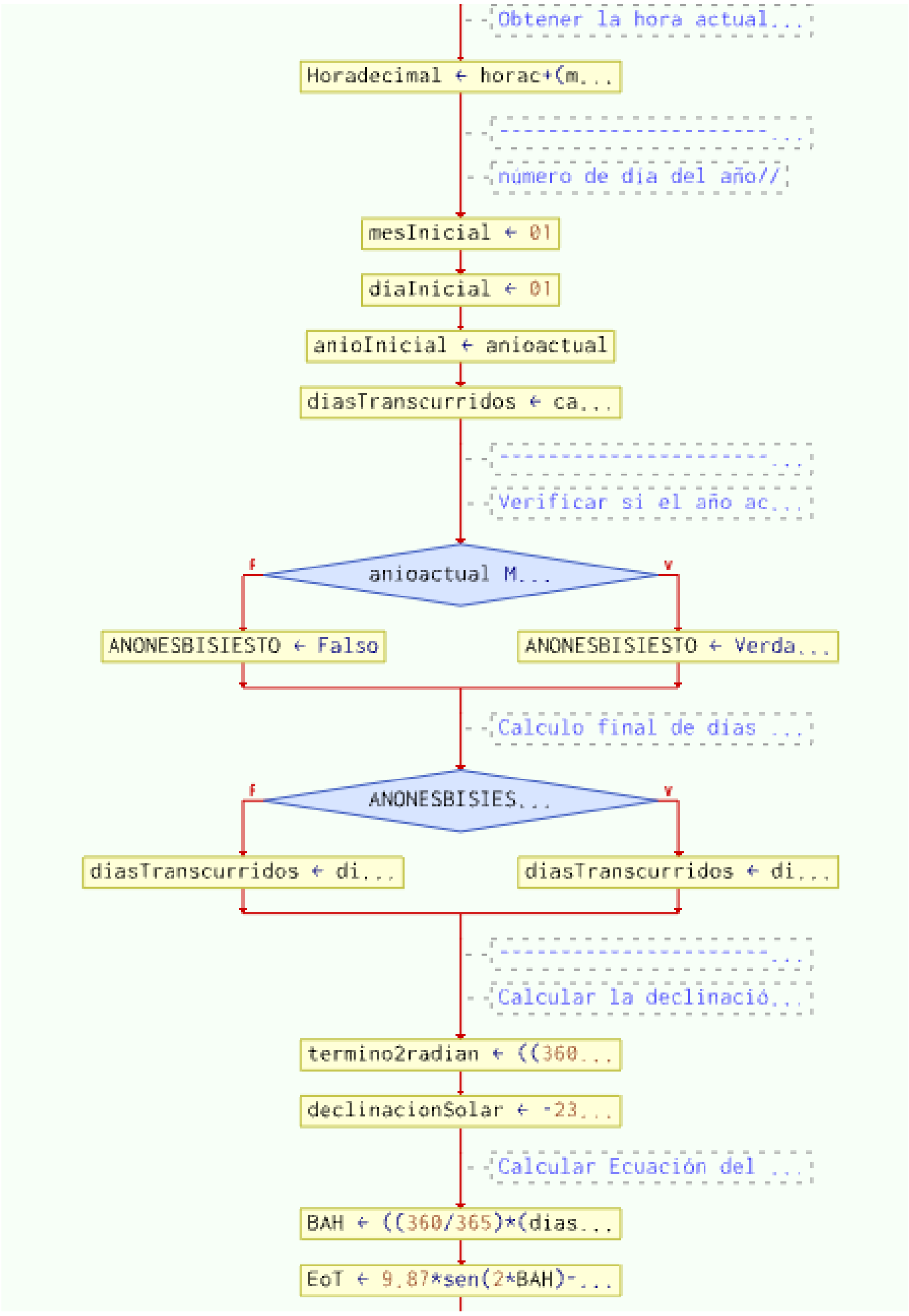
```
    Finsi

    // Calculo final de días transcurridos
    Si AñonesBisiesto= Verdadero Entonces
        diasTranscurridos ← diasTranscurridos + 2
    SiNo
        diasTranscurridos ← diasTranscurridos + 1
    FinSi

    //-------------------------------------------------------------
-------------------------------------------------------------
----------//

    //Calcular la declinación Solar//
    termino2radian= ((360/365)*(diasTranscurridos+10)) * PI/180
    declinacionSolar= -23.44 * cos(termino2radian)


    //Calcular Ecuación del Tiempo//
    BAH= ((360/365)*(diasTranscurridos-81)) * PI/180
    EoT= 9.87*sen(2*BAH)-7.53*cos(BAH)-1.5*sen(BAH)


    //Calcular TiempoSolarVerdadero//
    ZonaHo= -5
    LongitudEstandar= ZonaHo*15
    TSV= Horadecimal+(4*(LongitudPunto-LongitudEstandar)+EoT)/60

    //HoraSolar//
    HS= Trunc(TSV)
    MS= Redon(60*(TSV-HS))


    //Angulo Horario en Radianes//
    AHR= (15*(TSV-12))*PI/180

    declinacionSolarRad= declinacionSolar * PI/180

    LatitudPuntoRad= LatitudPunto * PI/180
    //Ángulo de Inclinación Solar//
    AIS=
asen(sen(declinasionSolarRad)*sen(LatitudPuntoRad)+cos(declinaci
onSolarRad)*cos(LatitudPuntoRad)*cos(AHR))

    AISGrados= AIS * 180/PI
    Escribir "El Ángulo de Inclinación Solar es: " AISGrados "°"
    //Azimut Solar//
    Azimut= acos((sen(declinacionSolarRad)-
sen(AIS)*sen(LatitudPuntoRad))/(cos(AIS)*cos(LatitudPuntoRad)))
    Si Azimut > 0 Entonces
        Azimutc = 2*PI-Azimut
    SiNo
        Azimutc = Azimut
    FinSi
    AzimutGrados= Azimutc*180/PI
    Escribir "El Azimut Solar en Grados es: " AzimutGrados "°"
FinAlgoritmo
```

Diagrama de Flujo






Algoritmo Evaluacion
Ingresar Longitud y La...
'Ingrese la latitud'
LatitudPunto
'Ingrese la longitud'
LongitudPunto
Separar la Hora Actual//
HoraActualidad ← Conve...
longitud(Hor...
F
V
horac ← ConvertirANume...
minutosc ← ConvertirAN...
horac ← ConvertirANume...
minutosc ← ConvertirAN...
Separar la Fecha Actual//
fecha ← FechaActual()
f1 ← ConvertirATexto(f...
aniox ← Subcadena(f1,1,4)
mesx ← Subcadena(f1,5,6)
diax ← Subcadena(f1,7,8)
anioactual ← Convertir...
mesactual ← ConvertirA...
diaactual ← ConvertirA...


// Obtener la hora actual...
Horadecimal ← horac+(m...
// ----------...
// número de día del año//
mesInicial ← 01
diaInicial ← 01
anioInicial ← anioactual
diasTranscurridos ← ca...
// ----------...
// Verificar si el año ac...
anioactual M...
F
V
ANONESBISIESTO ← Falso
ANONESBISIESTO ← Verda...
// Calculo final de dias ...
ANONESBISIES...
F
V
diasTranscurridos ← di...
diasTranscurridos ← di...
// ----------...
// Calcular la declinació...
termino2radian ← ((360...
declinacionSolar ← -23...
// Calcular Ecuación del ...
BAH ← ((360/365)*(dias...
EoT ← 9.87*sen(2*BAH)-...


Calcular TiempoSolarVe...
ZonaHo ← -5
LongitudEstandar ← Zon...
TSV ← Horadecimal+(4*(...
HoraSolar//
HS ← Trunc(TSV)
MS ← Redon(60*(TSV-HS))
Angulo Horario en Radi...
AHR ← (15*(TSV-12))*PI...
declinacionSolarRad ← ...
LatitudPuntoRad ← Lati...
Ángulo de Inclinación ...
AIS ← asen(sen(declina...
AISGrados ← AIS*180/PI
'El Ángulo de Inclinac...
Azimut Solar//
Azimut ← acos((sen(dec...
F
Azimut>0
V
Azimutc ← Azimut
Azimutc ← 2*PI-Azimut
AzimutGrados ← Azimutc...
'El Azimut Solar en Gr...
FinAlgoritmo

Código Desarrollado en C:

```
 #include <stdio.h>
#include <time.h>
#include <math.h>
#include <windows.h>


#define PI 3.14159265

// Definición de la estructura Coordenadas
struct Coordenadas {
    double latitud;
    double longitud;
};

// Función para mostrar los días transcurridos del año
void mostrarDiasTranscurridosDelAno() {
    time_t tiempo_actual;
    struct tm *tiempo_local;

    tiempo_actual = time(NULL); // Obtener el tiempo actual
    tiempo_local = localtime(&tiempo_actual); // Convertir el tiempo a la estructura tm

    printf("Días transcurridos del año: %d\n", tiempo_local->tm_yday + 1); // Mostrar los días transcurridos del año
}

// Función para mostrar la fecha actual
void mostrarFechaActual() {
    time_t tiempo_actual;
    struct tm *tiempo_local;

    tiempo_actual = time(NULL); // Obtener el tiempo actual
    tiempo_local = localtime(&tiempo_actual); // Convertir el tiempo a la estructura tm

    printf("La fecha actual es: %02d-%02d-%04d\n", // Mostrar la fecha actual en formato dd-mm-aaaa
           tiempo_local->tm_mday,
           tiempo_local->tm_mon + 1,
           tiempo_local->tm_year + 1900);
}

// Función para mostrar la hora actual
void mostrarHoraActual() {
    time_t tiempo_actual;
    struct tm *tiempo_local;

    tiempo_actual = time(NULL); // Obtener el tiempo actual
    tiempo_local = localtime(&tiempo_actual); // Convertir el tiempo a la estructura tm

    printf("La hora actual es: %02d:%02d:%02d\n", // Mostrar la hora actual en formato hh:mm:ss
           tiempo_local->tm_hour,
           tiempo_local->tm_min,
           tiempo_local->tm_sec);
}

// Función para calcular la declinación solar
double calcularDeclinacionSolar(int diasTranscurridos) {
    double termino2radian, declinacionSolar;
```

```
    // Calcular el término en radianes
    termino2radian = ((360.0 / 365.0) * (diasTranscurridos + 10)) * (PI / 180.0);

    // Calcular la declinación solar
    declinacionSolar = -23.44 * cos(termino2radian);

    return declinacionSolar; // Devolver la declinación solar en grados
}

// Función para calcular la ecuación del tiempo
double calcularEcuacionDelTiempo(int diasTranscurridos) {
    double BAH, EoT;

    // Calcular el término BAH en radianes
    BAH = ((360.0 / 365.0) * (diasTranscurridos - 81)) * (PI / 180.0);

    // Calcular la Ecuación del Tiempo
    EoT = 9.87 * sin(2 * BAH) - 7.53 * cos(BAH) - 1.5 * sin(BAH);

    return EoT; // Devolver la ecuación del tiempo en minutos
}

// Función para calcular el Tiempo Solar Verdadero
double calcularTiempoSolarVerdadero(double horaDecimal, double longitudPunto, double ecuacionTiempo) {
    int zonaHo = -5; // Zona horaria
    double longitudEstandar, TSV;

    // Calcular la longitud estándar en grados
    longitudEstandar = zonaHo * 15.0;

    // Calcular el Tiempo Solar Verdadero
    TSV = horaDecimal + (longitudPunto - longitudEstandar) / 15.0 + ecuacionTiempo / 60.0;

    return TSV; // Devolver el Tiempo Solar Verdadero en horas
}

// Función para calcular el ángulo horario
double calcularAnguloHorario(double tiempoSolarVerdadero) {
    double AHR = (tiempoSolarVerdadero - 12) * 15 * PI / 180;
    return AHR; // Devolver el ángulo horario en radianes
}

// Función para convertir grados a radianes
double convertirARadianes(double grados) {
    return grados * PI / 180.0; // Convertir grados a radianes
}

// Función para convertir radianes a grados
double convertirAGrados(double radianes) {
    return radianes * 180 / PI; // Convertir radianes a grados
}

// Función para calcular el Ángulo de Inclinación Solar
double calcularAnguloInclinacionSolar(double declinacionSolarRad, double LatitudPuntoRad, double AHR) {
    double AIS_rad = asin(sin(declinacionSolarRad) * sin(LatitudPuntoRad) + cos(declinacionSolarRad) *
cos(LatitudPuntoRad) * cos(AHR));
    double AIS_grados = convertirAGrados(AIS_rad);
```

```
    return AIS_grados; // Devolver el Ángulo de Inclinación Solar en grados
}

// Función para calcular el Azimut Solar
double calcularAzimutSolar(double declinacionSolarRad, double AIS_rad, double LatitudPuntoRad, double AHR) {
    double Azimut_rad;
    // Calcular el Azimut solar en radianes
    Azimut_rad = acos((sin(declinacionSolarRad) * cos(LatitudPuntoRad) - cos(declinacionSolarRad) *
sin(LatitudPuntoRad) * cos(AHR)) / cos(AIS_rad));
    // Ajustar el Azimut basado en el ángulo horario
    if (sin(AHR) > 0) {
        Azimut_rad = 2 * PI - Azimut_rad;
    }
    // Convertir a grados
    double Azimut_grados = convertirAGrados(Azimut_rad);
    // Asegurar que el azimut esté en el rango [0, 360]
    if (Azimut_grados < 0) {
        Azimut_grados += 360;
    } else if (Azimut_grados >= 360) {
        Azimut_grados -= 360;
    }
    return Azimut_grados; // Devolver el Azimut Solar en grados
}

// Función para actualizar y mostrar la orientación del panel solar
void actualizarOrientacionPanel(struct Coordenadas coordenadas) {
    int diasTranscurridos;
    double declinacion, ecuacionTiempo, tiempoSolarVerdadero, AHR, AIS_grados, Azimut_grados,
declinacionSolarRad, LatitudPuntoRad;

    // Obtener el número de días transcurridos del año
    time_t tiempo_actual = time(NULL);
    struct tm *tiempo_local = localtime(&tiempo_actual);
    diasTranscurridos = tiempo_local->tm_yday + 1;

    // Calcular la declinación solar
    declinacion = calcularDeclinacionSolar(diasTranscurridos);

    // Calcular la Ecuación del Tiempo
    ecuacionTiempo = calcularEcuacionDelTiempo(diasTranscurridos);

    // Calcular el Tiempo Solar Verdadero
    double horaDecimal = tiempo_local->tm_hour + tiempo_local->tm_min / 60.0 + tiempo_local->tm_sec / 3600.0;
    tiempoSolarVerdadero = calcularTiempoSolarVerdadero(horaDecimal, coordenadas.longitud, ecuacionTiempo);

    // Calcular el Ángulo Horario en Radianes
    AHR = calcularAnguloHorario(tiempoSolarVerdadero);

    // Convertir declinación solar y latitud del punto a radianes
    declinacionSolarRad = convertirARadianes(declinacion);
    LatitudPuntoRad = convertirARadianes(coordenadas.latitud);

    // Calcular el Ángulo de Inclinación Solar
    AIS_grados = calcularAnguloInclinacionSolar(declinacionSolarRad, LatitudPuntoRad, AHR);

    // Calcular el Azimut Solar
    double AIS_rad = convertirARadianes(AIS_grados);
    Azimut_grados = calcularAzimutSolar(declinacionSolarRad, AIS_rad, LatitudPuntoRad, AHR);
```

```
    // Mostrar la orientación del panel solar
    printf("Orientación del panel solar:\n");
    printf("  Ángulo de Inclinación Solar: %.2f grados\n", AIS_grados);
    printf("  Azimut Solar: %.2f grados\n", Azimut_grados);
}

// Función para verificar si la entrada es un número
int esNumero(char *entrada) {
    // Verificar cada caracter de la cadena
    while (*entrada) {
        if (!isdigit(*entrada) && *entrada != '.' && *entrada != '-') {
            return 0;  // No es un número válido
        }
        entrada++;
    }
    return 1;  // Es un número válido
}

int main() {
    struct Coordenadas coordenadas;

    // Variables temporales para almacenar la entrada del usuario
    char buffer[100];
    // Solicitar al usuario que ingrese la longitud del lugar
    while (1) {
        printf("Ingrese la longitud del lugar (ejemplo: -79.20422): ");
        scanf("%s", buffer);

        // Verificar si la entrada es un número válido
        if (esNumero(buffer)) {
            sscanf(buffer, "%lf", &coordenadas.longitud);
            break;  // Salir del bucle si es un número válido
        } else {
            printf("Error: Ingrese un valor numérico válido.\n");
        }
    }

    // Solicitar al usuario que ingrese la latitud del lugar
    while (1) {
        printf("Ingrese la latitud del lugar (ejemplo: -3.99313): ");
        scanf("%s", buffer);

        // Verificar si la entrada es un número válido
        if (esNumero(buffer)) {
            sscanf(buffer, "%lf", &coordenadas.latitud);
            break;  // Salir del bucle si es un número válido
        } else {
            printf("Error: Ingrese un valor numérico válido.\n");
        }
    }

    // Bucle para actualizar la orientación del panel solar continuamente
    while (1) {
        actualizarOrientacionPanel(coordenadas);
        // Esperar un minuto antes de la siguiente actualización (simulado con sleep)
        Sleep(60000);
    }
```

```
  return 0;
}
```

Código Desarrollado Arduino IDE para orientar los paneles solares siguiendo una fuente de luz:

```
#include <Servo.h>

// Definición de los pines para los servomotores y las fotorresistencias
const int servoPin1 = 9;   // Pin para el servo de orientación
const int servoPin2 = 10;  // Pin para el servo de inclinación
const int ldrPin1 = A0;    // Pin para la fotorresistencia horizontal
const int ldrPin2 = A1;    // Pin para la fotorresistencia vertical

Servo servo1;  // Servo para orientación
Servo servo2;  // Servo para inclinación

int pos1 = 90;  // Posición inicial del servo de orientación
int pos2 = 90;  // Posición inicial del servo de inclinación

void setup() {
  servo1.attach(servoPin1);  // Adjuntar el servo de orientación al pin correspondiente
  servo2.attach(servoPin2);  // Adjuntar el servo de inclinación al pin correspondiente

  servo1.write(pos1);  // Mover el servo de orientación a la posición inicial
  servo2.write(pos2);  // Mover el servo de inclinación a la posición inicial
}

void loop() {
  int ldrValue1 = analogRead(ldrPin1);  // Leer el valor de la fotorresistencia horizontal
  int ldrValue2 = analogRead(ldrPin2);  // Leer el valor de la fotorresistencia vertical

  // Ajustar la posición del servo de orientación en función del valor de la fotorresistencia horizontal
  if (ldrValue1 > 600) {
    pos1 += 1;
  } else if (ldrValue1 < 400) {
    pos1 -= 1;
  }

  // Ajustar la posición del servo de inclinación en función del valor de la fotorresistencia vertical
  if (ldrValue2 > 600) {
    pos2 += 1;
  } else if (ldrValue2 < 400) {
    pos2 -= 1;
  }

  // Limitar los rangos de los servos para evitar que se muevan fuera de sus límites físicos
  pos1 = constrain(pos1, 0, 180);
  pos2 = constrain(pos2, 0, 180);

  // Mover los servos a las nuevas posiciones
  servo1.write(pos1);
  servo2.write(pos2);

  // Esperar un pequeño intervalo antes de la siguiente lectura
  delay(100);
}
```

5. **¿Cómo nuestro diseño podría implementarse en un sistema real de paneles solares?**

Nuestro diseño se podría implementar con sensores GPS para obtener la ubicación exacta y sensores de inclinación y orientación para monitorear la posición de los paneles. Los datos recopilados se utilizarán como entrada para el algoritmo SPA, que proporciona instrucciones para ajustar automáticamente la orientación de los paneles solares mediante actuadores controlados por motores eléctricos o hidráulicos. Se podría desarrollar una interfaz de usuario intuitiva para configurar el sistema y monitorear su rendimiento, asegurando así una eficiente captación de energía solar en todo momento. La optimización del consumo de energía y recursos sería fundamental, garantizando que el sistema sea lo más eficiente posible en su conjunto.

Referencias:

[1] J. J. Rojas *Diagnóstico de fallas en un panel fotovoltaico usando un algoritmo genético*. [online]. Disponible en: http://hdl.handle.net/20.500.12622/4398.

[2] N. Clemente, Luis Antonio. "Diseño e implementación de un algoritmo para la posición óptima de un panel solar, utilizando control predictivo." Trabajo final para la obtención del título: Magister en Automatización y Control Industrial. Espol Fiec, Guayaquil, 2016. [En línea]. Disponible en: https://www.dspace.espol.edu.ec/handle/123456789/46970

[3] J. A. Gualoto Cachago y R. A. Potosí Díaz, "Desarrollo de un seguidor solar automatizado usando un sistema embebido y algoritmo de posicionamiento solar (SPA)," Tesis de Maestría, 2022. [En línea]. Disponible en:http://dspace.ups.edu.ec/handle/123456789/22899

[4] A. T. Morales, "Adaptación de algoritmo de posicionamiento solar embebido con aplicación en energía solar," 2017. [En línea]. Disponible en: https://ri-ng.uaq.mx/handle/123456789/1534

[5] Struct TM cplusplus.com. Available at: https://cplusplus.com/reference/ctime/tm/ (Accessed: 16 June 2024).

[6] M. Juan, "¿Cómo trabajar con estructuras en C++?", Stack Overflow en español. [En línea]. Disponible en: https://es.stackoverflow.com/questions/340385/c%C3%B3mo-trabajar-con-estructuras-en-c.

[7] Davidson, S.R. (no date) C++ Con clase, c.2001 Available at: https://conclase.net/c/librerias/time/tm (Accessed: 16 June 2024).

[8] "Biblioteca C -", Edu.lat. [En línea]. Disponible en: https://tutoriales.edu.lat/pub/c-standard-library/time-h/biblioteca-c-time-h. [Consultado: 16-jun-2024].

[9] "Biblioteca C -", Edu.lat. [En línea]. Disponible en:

https://tutoriales.edu.lat/pub/c-standard-library/math-h/biblioteca-c-math-h. [Consultado: 16-jun-2024].