

# ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO INDUSTRIAL ELÉCTRICO

Título del proyecto:

APLICACIÓN ROBÓTICA DE ORIENTACIÓN SOLAR

Rubén Ostiz Tellechea

Vicente Senosiain Miquelez

Pamplona, 14 de junio de 2012



## ÍNDICE

1.INTRODUCCIÓN.....	7
1.1 Objetivo.....	7
1.2 Descripción del proyecto.....	7
2. EL SISTEMA DE COORDENADAS.....	9
2.1 La esfera celeste.....	9
2.2 Datos auxiliares.....	10
3.MÉTODO DE CÁLCULO DE LA LONGITUD.....	13
3.1 Ecuación de tiempo.....	13
3.2 Método de cálculo de la hora del mediodía solar.....	15
3.3 Ecuación final.....	20
4.MÉTODO DE CÁLCULO DE LA LATITUD.....	21
4.1 Ecuación latitud.....	21
4.2 Correcciones elevación solar.....	23
4.2.1 Refracción atmosférica.....	23
4.2.2 Paralaje.....	28
4.2.3 Semidiámetro solar.....	29
4.3 Método elección de la latitud.....	30
5. OBTENCION DE LAS MEDIDAS.....	33
6. DIAGRAMAS DE FLUJO.....	35
6.1 Explicación de los símbolos utilizados.....	35
6.2 Programa principal.....	36
6.3 Subprograma corrección elevación.....	41
6.4 Subprograma cálculo acimut teórico.....	42
6.5 Subprograma cálculo declinación.....	43
6.6 Subprograma cálculo latitud.....	44
6.7 Subprograma cálculo día del año.....	46
6.8 Subprograma cálculo ecuación de tiempo.....	48
6.9 Subprograma cálculo longitud.....	49
6.10 Subprograma medida elevación solar.....	50
6.11 Subprograma medida acimut.....	51
6.12 Subprograma cero delante.....	52
6.13 Subprograma mostrar reloj.....	53
7. LIMITACIONES.....	55
8.INCERTIDUMBRES.....	57
8.1 Incertidumbre latitud.....	57
8.2 Incertidumbre longitud.....	60
8.3 Incertidumbres en kilómetros.....	64

9. COMENTARIOS FINALES.....	69
9.1 Programa robot.....	69
9.2 Conclusiones.....	69
9.3 Mejoras futuras.....	70

10.BIBLIOGRAFÍA.....	73
----------------------	----

## ANEXOS

A. LISTA DE PROGRAMAS.....	75
PROGRAMAS TEST.....	75
PROGRAMA PRINCIPAL.....	92
PROGRAMA PREPARADO PARA ROBOT.....	98
B. INSTRUCCIONES DE INSTALACIÓN ARDUINO.....	105
C. MANUAL DE PROGRAMACIÓN ARDUINO.....	109





# 1. INTRODUCCIÓN

## 1.1. OBJETIVO

El objetivo del proyecto es crear una aplicación para que un robot pueda orientarse mediante la posición del Sol. Para ello, implementaremos un programa en el que introduciendo una serie de datos sea capaz de indicarnos la latitud y la longitud donde se encuentra nuestro prototipo.

## 1.2. DESCRIPCIÓN DEL PROYECTO FIN DE CARRERA

Este será el primero de una serie de proyectos que tendrán como fin construir un robot que sea capaz de moverse libremente, de manera que pueda dirigirse a un punto cualquiera de la Tierra.

Para ello, deberemos dotar a nuestro robot de un sistema de orientación que le indique en qué posición se encuentra inicialmente. Para desarrollar este sistema utilizando el movimiento del Sol alrededor de la Tierra para que nuestro robot pueda calcular la posición de la latitud y la longitud donde se encuentra y, de esa manera, decidir hacia dónde debe moverse para llegar a su punto de destino.

Para llevar a cabo este sistema de orientación necesitaremos un sensor que nos detecte la elevación solar del lugar donde se encuentra nuestro robot y el programa que realizará los cálculos que nos indicarán la latitud y la longitud. En este proyecto solo desarrollaremos el programa que realizará los cálculos.

Para implementar nuestro programa vamos a utilizar la tecnología ARDUINO, que es una plataforma de hardware libre basada en una sencilla placa de entradas y salidas simples y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring. Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia, Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse en el mercado. El entorno de desarrollo integrado libre se puede descargar gratuitamente. Las plataformas Arduino están basadas en los microcontroladores Atmega168, Atmega328, Atmega1280, ATmega8 y otros similares, chips sencillos y de bajo coste que permiten el desarrollo de múltiples diseños. Al ser open-hardware, tanto su diseño como su distribución son libres. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia. Las distintas estrategias de programación deberán tener en cuenta que el programa debe ser lo más simple y rápido posible en ejecución, siendo la precisión un aspecto muy importante.

Todo esto hace que sea un sistema adecuado ya que de esta manera los futuros proyectos que pretendan complementar este proyecto podrán utilizar esta tecnología libre y bastante sencilla.





## 2. EL SISTEMA DE COORDENADAS

Antes de empezar con los métodos utilizados para calcular la latitud y la longitud, explicaremos que sistema de coordenadas se utiliza para posicionar el Sol en el cielo y la relación que tiene con éstas.

### 2.1. LA ESFERA CELESTE.

La Tierra se mueve alrededor del Sol cíclicamente durante el año y sobre su propio eje cada veinticuatro horas, pero como nos interesa coger al observador como referencia, supondremos que lo que se mueve es el Sol. A este movimiento imaginario lo llamaremos movimiento aparente del Sol.

Para describir este movimiento aparente necesitaremos un sistema de coordenadas en el cual utilizaremos el observador como punto de referencia. En este caso, utilizaremos la esfera celeste.

Se basa en crear una esfera imaginaria, en este caso una semiesfera, de radio arbitrario centrado en el observador sobre la cual proyectaremos el Sol. (fig1.)

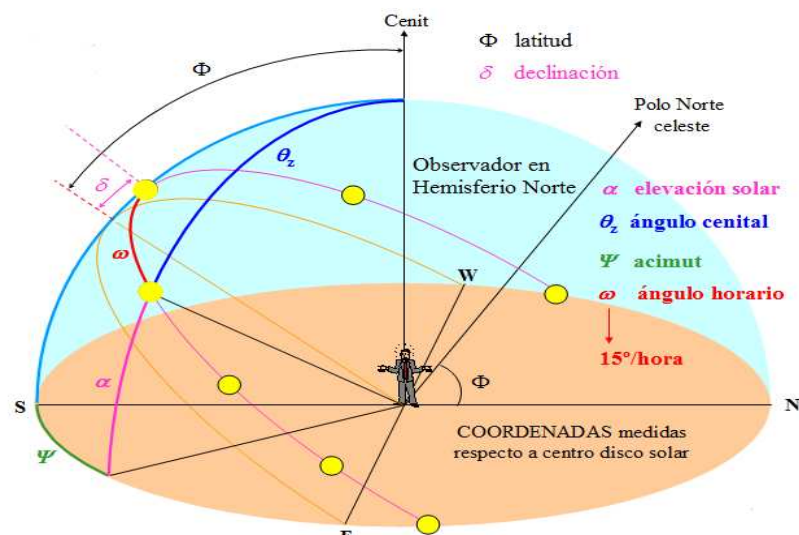


Fig.1: Semiesfera celeste

Antes de explicar las coordenadas a utilizar explicaremos algunos puntos de la esfera celeste:

1. Cénit: es el punto de intersección del hemisferio celeste con la dirección de la vertical desde el observador.
2. Horizonte celeste: es el plano perpendicular a la dirección de la vertical y divide la esfera en dos semiesferas: hemisferio visible y hemisferio invisible. Nosotros utilizaremos el hemisferio visible para realizar nuestro proyecto. (Plano coloreado de rosa de la fig. 1.)

3. Ecuador celeste: es la proyección del ecuador en la esfera celeste. (Línea naranja de la fig. 1.)

Una vez explicados los puntos más importantes de la esfera celeste, vamos a explicar las coordenadas que utilizaremos para posicionar el Sol en la esfera celeste.

1. Altura o elevación solar ( $\alpha$ ): la distancia angular del horizonte al astro. Se mide en grados desde el horizonte.
2. Ángulo cenital ( $\theta_z$ ): es el arco complementario a la elevación solar. Por tanto, se cumple que:

$$\theta_z = 90 - \alpha$$

3. Acimut ( $\Psi$ ): la distancia angular medida sobre el horizonte, desde el Sur hasta el pie del círculo máximo que pasa por el cénit y por el cuerpo celeste. Se mide en grados.

En la fig.1 podemos ver gráficamente cada uno de los puntos y coordenadas explicadas anteriormente.

## 2.2. DATOS AUXILIARES

El Sol varía su posición con el paso de las horas y de los días. Por ello, utilizaremos unos datos auxiliares que nos ayudarán a calcular esa variación.

1. Ángulo horario ( $\omega$ ): es el arco de horizonte celeste contado desde el punto de intersección del meridiano de referencia del observador hasta el círculo máximo que pasa por el cénit y por el astro. Se mide en grados o en horas. Será positivo hacia el Oeste y negativo hacia el Este. Cada hora corresponde a  $15^\circ$  ( $360^\circ/24$  horas).

$$\omega = \frac{\text{hora}_s - 12}{15}$$

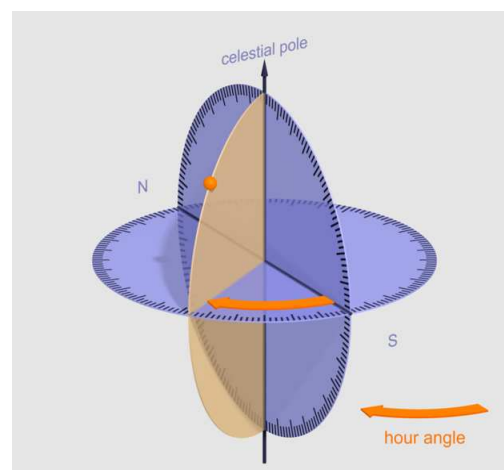


Fig.2: ángulo horario

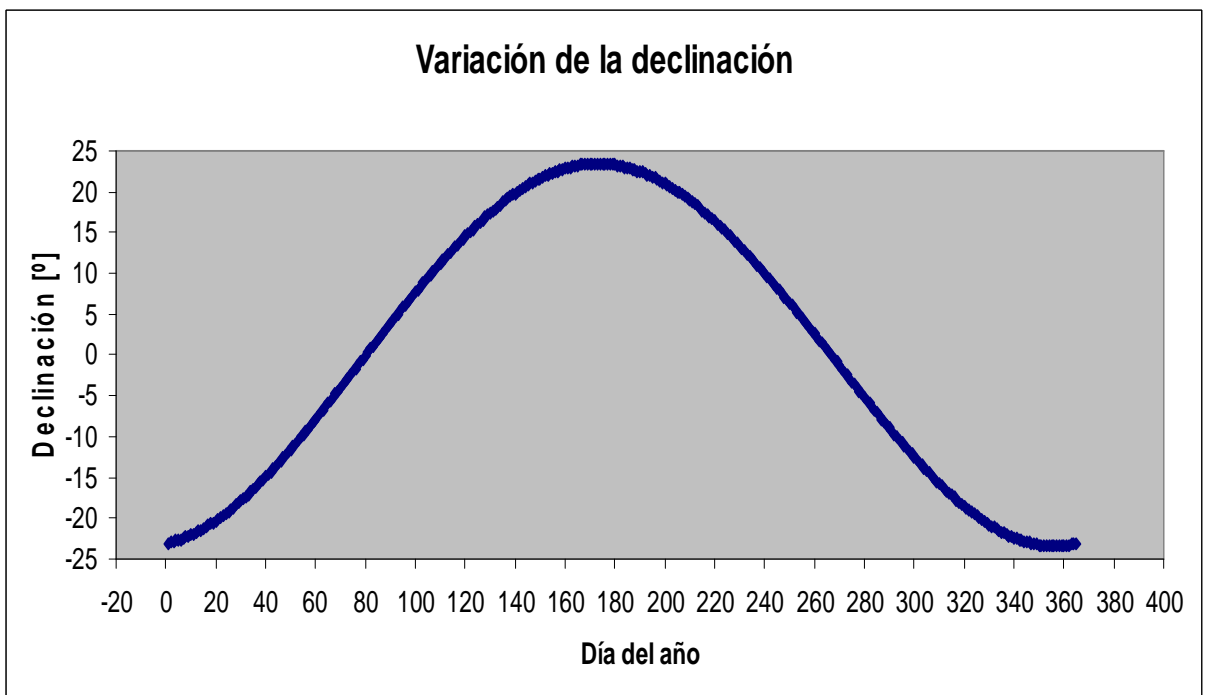
2. Declinación ( $\delta$ ): es el ángulo que forma un astro con el ecuador celeste, en nuestro caso el Sol. Se mide en grados y es positiva si está al Norte del ecuador celeste y negativa si está al Sur. La declinación varía constantemente, pero lo hace de manera insignificante durante un día, así que la calcularemos dependiendo del día del año. Para calcularlo utilizaremos la fórmula de Spencer:

$$\delta[^\circ] = (0.006918 - 0.399912 \cdot \cos \Gamma + 0.070257 \cdot \text{sen} \Gamma - 0.006758 \cdot \cos 2\Gamma + 0.000907 \cdot \text{sen} 2\Gamma - 0.002697 \cdot \cos 3\Gamma + 0.00148 \cdot \text{sen} 3\Gamma) \cdot \frac{180}{\pi}$$

Donde:

$$\Gamma = 2\pi \frac{J-1}{365} \quad \text{J es el número de día del año.}$$

La declinación varía cíclicamente durante todo un año, tal y como muestra el siguiente gráfico:





### 3. MÉTODO DE CÁLCULO DE LA LONGITUD

Durante el movimiento solar aparente, el Sol realiza un movimiento cíclico alrededor de la Tierra. Este movimiento cíclico dura alrededor de 24 horas y se denomina **día solar**. La Tierra es una esfera, por tanto, el Sol aparente se mueve alrededor de la tierra a razón de  $15^\circ/h$  ( $365/24$ ). Si queremos calcular la posición de la longitud en la que nos encontramos, deberemos calcular la hora en el meridiano de Greenwich cuando el Sol pase por nuestro meridiano.

Imaginemos que nos encontramos en Londres, donde la longitud es aproximadamente  $0^\circ$ , y nos montamos en un avión a Rio de Janeiro que se encuentra aproximadamente a  $45^\circ$  Este. Mantenemos en nuestro reloj la hora de Greenwich. Esperamos a que el reloj del aeropuerto marque las doce, momento en el que el Sol pasa por el meridiano de Rio de Janeiro, y miramos nuestro reloj. Éste marcará las nueve de la mañana. Si a la hora de Greenwich a la que pasa el Sol por el meridiano de Greenwich (las doce), le restamos la hora que marcará nuestro reloj cuando el Sol pasa por el meridiano de Rio de Janeiro y lo multiplicamos por  $15^\circ$ , nos dará  $45^\circ$ , que es la longitud a la que nos encontramos.

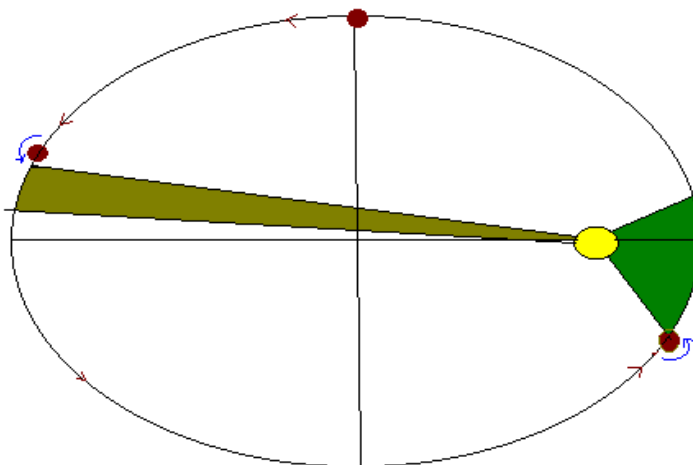
Lo expresamos matemáticamente y obtenemos la siguiente ecuación:

$$\text{Longitud} = 15 \cdot (HG - ML)$$

Siendo, HG el mediodía en el meridiano de Greenwich, donde la longitud es igual a  $0^\circ$  y ML, la hora del meridiano de Greenwich cuando se produce el mediodía en el meridiano donde nos encontramos. Las unidades de HG y ML son horas.

#### 3.1. ECUACIÓN DE TIEMPO

Debemos tener en cuenta que el día solar varía en su duración debido a que la Tierra barre áreas desiguales en el plano de la eclíptica a medida que se mueve en torno al Sol y a que el eje de la Tierra está inclinado respecto al plano de la eclíptica. En la fig. 3.1 podemos ver como se mueve la Tierra respecto del Sol.



Áreas iguales "barridas por la Tierra en tiempos iguales"

Fig. 3.1: Órbita real de la tierra respecto al Sol.

Por ello, hablaremos del día solar medio. El día solar medio es el promedio de la duración de los días solares y corresponde al movimiento de un Sol ficticio que denominaremos el Sol medio. Éste realizará un movimiento aparente que discurriría en el plano del ecuador y alrededor de la Tierra describiendo una órbita con velocidad constante. De esta manera, todos los días solares medios tendrán la misma duración. En la fig. 3.2 podemos ver cual sería la órbita del Sol con el día solar medio.

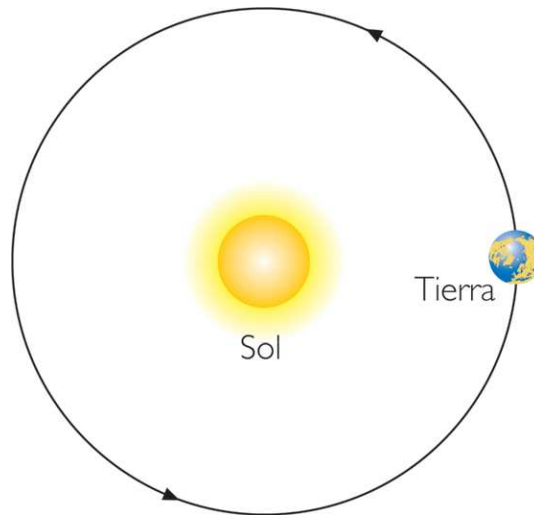


Fig. 3.2: Órbita de la tierra ficticia.

La discrepancia existente entre el movimiento aparente del Sol medio y el movimiento del Sol verdadero se llama **ecuación de tiempo**.

Ecuación de tiempo = hora solar media – hora solar verdadera

Para calcular la ecuación de tiempo utilizaremos la siguiente fórmula:

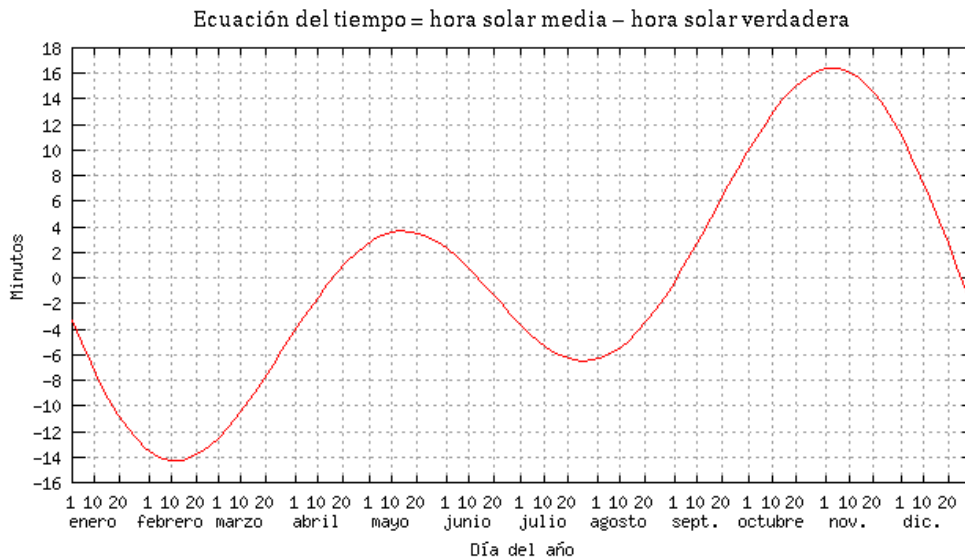
$$E_t = (0,000075 + 0,00186 \cdot \cos \Gamma - 0,032077 \cdot \text{sen} \Gamma - 0,014615 \cdot \cos 2\Gamma - 0,004089 \cdot \text{sen} 2\Gamma) \cdot (229,18)$$

Donde:

$$\Gamma = 2\pi \frac{J-1}{365}$$

J es el número de día del año. (El día 1 del año será el día 1 de Enero.)

En el siguiente gráfico podemos ver como evoluciona la ecuación de tiempo a lo largo de un año:



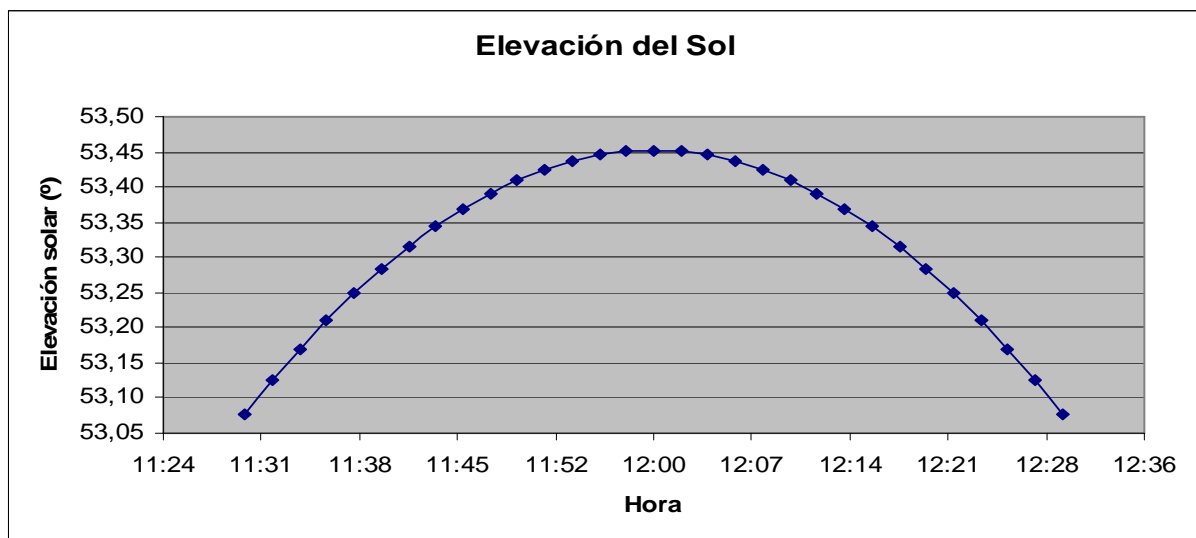
Este error deberemos corregirlo en nuestra ecuación para poder considerar el día solar medio y que todos los días del año tengan la misma duración. La fórmula nos quedará de la siguiente manera:

$$Longitud = 15 \cdot (HG - ML - E_t)$$

### 3.2 MÉTODO DE CÁLCULO DE LA HORA DEL MEDIODÍA SOLAR.

Para calcular la longitud lo primero que pensamos es medir la elevación máxima que se produce durante el día y utilizar la hora en que se produce esa elevación para calcular la longitud.

Pero desechamos este sistema debido a la incertidumbre de nuestro sensor, ya que cuando el Sol está en su elevación máxima su variación en la elevación es prácticamente nula durante un intervalo de tiempo. En el siguiente gráfico vemos como varía la elevación solar para una latitud de 60° el 22 de marzo.



En este caso, y con una incertidumbre de  $\pm 0,1^\circ$  en nuestro sensor, podemos ver que se produciría un error de aproximadamente de media hora, lo que producirá un error de aproximadamente  $8^\circ$  en la longitud. Muy grande para el propósito de nuestro proyecto.

Para solucionarlo, asumimos que el Sol describe un arco simétrico en el cielo, cogemos dos medidas de tiempo, una antes del mediodía solar ( $T_1$ ) y otra después del mediodía solar ( $T_2$ ), ambas con la misma elevación solar. La media de estas medidas ( $T_{Transit}$ ) será el momento en que el Sol pasa por su elevación máxima, y por tanto, por el mediodía solar. (Fig. 3.4)

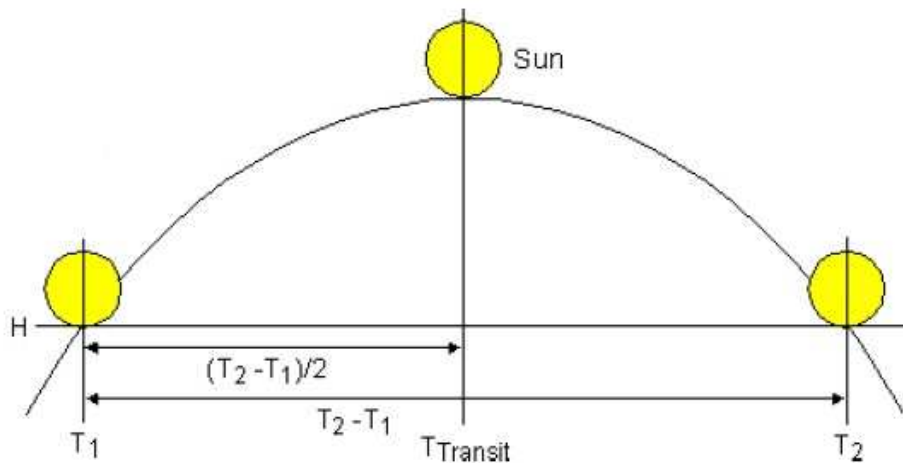


Fig. 3.4: Arco que describiría el Sol.

Como ambas mediciones se producen a la misma altitud no sería necesario realizar correcciones debido a la refracción solar, paralaje o semidiámetro solar. El intervalo que debe haber entre ambas mediciones debería ser más grande de 2 horas, ya que si no puede ser que estemos demasiado cerca de la elevación máxima.

Desafortunadamente, el arco que describiría el Sol será simétrico solo cuando la declinación sea constante durante el intervalo de la medida. Este caso solo se produce aproximadamente durante los solsticios. El resto del año  $T_{Transit}$  difiere significativamente de la media de  $T_1$  y  $T_2$  debido a lo que varía la declinación del Sol en el intervalo de la medida.

En la fig. 3.5 nos muestra como varía la elevación solar en función del tiempo y como el cambio de la declinación afecta al recorrido aparente del Sol en el cielo. El error está representado por  $\Delta T$ .



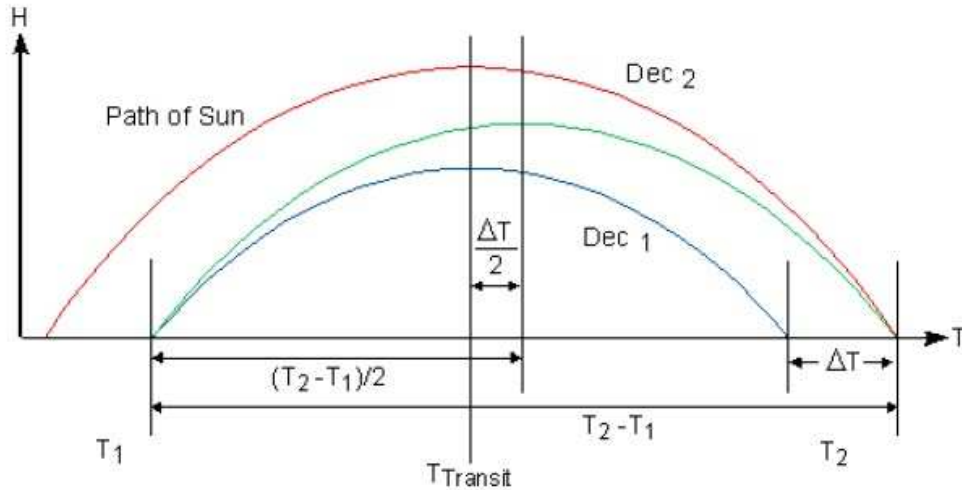


Fig. 3.5: Variación del arco.

La línea azul es la que muestra el arco que describiría el Sol si mantuviéramos la declinación que tiene el Sol cuando realizamos la primera medida constante durante todo el intervalo de la medida. La línea roja nos muestra el arco que describiría el Sol si la declinación constante fuera la que tiene el Sol cuando tomamos la segunda medida. Y la línea verde será el arco real que describiría el Sol debido a la variación de la declinación.

Además, si la declinación cambia en dirección a la paralela de la latitud donde se encuentra el robot, el paso del Sol por el meridiano local se producirá antes de  $(T_1+T_2)/2$ . Si la declinación cambia en dirección contraria a la paralela de la latitud donde se encuentra el robot el paso por el meridiano local se producirá después de  $(T_1+T_2)/2$ .

El error que se producirá en el cálculo de la longitud será despreciable cuando se encuentre en los solsticios, ya que la variación de la declinación es prácticamente nula. Y será máxima cuando se encuentre en los equinoccios ya que es cuando se produce la máxima variación de la declinación.

Podemos mejorar la longitud calculando el error de  $\Delta T$  aplicando la ecuación de altitudes iguales:

$$\Delta t \approx \left( \frac{\tan \Phi}{\sin \omega_2} - \frac{\tan \delta_2}{\tan \omega_2} \right) \cdot \Delta \delta$$

Donde:

$$\Delta \delta = \delta_2 - \delta_1$$

$\Delta t$  es el cambio del ángulo horario que corrige el cambio de la elevación solar debido al pequeño cambio que sufre la declinación.  $\Phi$  será la latitud donde se encuentra el observador.  $\delta_2$  será la declinación del Sol cuando se produce la segunda medida. Y  $\omega_2$  el ángulo horario cuando tomamos la segunda medida. Como no sabemos el valor exacto de nuestro ángulo horario, empezaremos con la siguiente aproximación:

$$\omega_2 [^\circ] \approx \frac{0.25 \cdot (T_2 [\text{min}] - T_1 [\text{min}])}{2}$$

Una vez calculado  $\Delta t$  podemos calcular el valor mejorado de  $T_2$  y sustituirlo para calcular  $T_{\text{Transit}}$ . El nuevo valor de  $T_{\text{Transit}}$  será el momento por el que pasa el Sol al mediodía si la declinación fuera constante.

$$T_2' [\text{min}] = T_2 [\text{min}] - \frac{\Delta t [^\circ]}{0.25} \qquad T_{\text{Transit}} = \frac{T_1 + T_2'}{2}$$

Todavía existirá un error residual por lo que si la precisión no fuera lo suficiente podríamos mejorarlo por iteración, realizando las operaciones anteriores con el nuevo  $T_2'$  calculado.

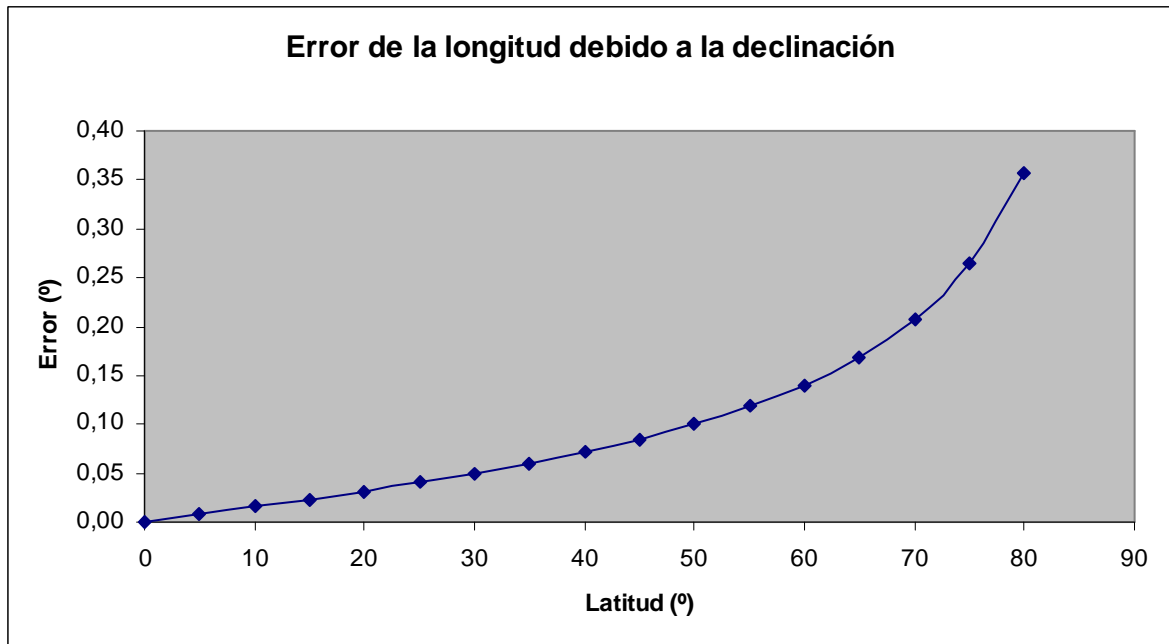
A continuación, vamos a ver que error se produciría el día 81 del año, es decir, el 21 de Marzo que es cuando se produce el equinoccio de primavera. Tomamos las medidas en el equinoccio de primavera porque es uno de los dos momentos del año en el que la variación de la declinación es máxima. Y por tanto, también lo será el error.

Tomamos las medidas para una elevación solar de  $10^\circ$  y para diferentes latitudes y obtenemos los siguientes resultados:

LATITUD	HORA 1	HORA 2	$\Delta t/2$
0	6:47	17:27	0,00
5	6:47	17:27	0,01
10	6:47	17:27	0,02
15	6:48	17:26	0,02
20	6:49	17:25	0,03
25	6:51	17:24	0,04
30	6:52	17:22	0,05
35	6:55	17:20	0,06
40	6:58	17:17	0,07
45	7:02	17:13	0,09
50	7:08	17:07	0,10
55	7:15	17:00	0,12
60	7:25	16:50	0,14
65	7:41	16:36	0,17
70	8:04	16:12	0,21
75	8:47	15:29	0,26
80	10:58	13:22	0,36

Las tres primeras columnas nos indican la latitud y las horas a las que el Sol tiene una elevación solar de  $10^\circ$ . (Datos obtenidos con la aplicación sunmotions de la Universidad de Nebraska). Y la última, el error que se produce en la longitud en el momento en el que el Sol pasa por el meridiano local.

En la siguiente gráfica representamos los datos obtenidos:



Observando la gráfica podemos sacar las siguientes conclusiones:

- En latitudes cercanas a los polos el error es mayor.
- El error empieza a ser significativo en latitudes superiores a  $60^\circ$ , donde el paralelo (perímetro de la tierra a latitud constante (fig.3.6)) de la Tierra es menor y por tanto, también el error en km que cometeremos. Por ejemplo, el error con  $60^\circ$  es de  $0,14^\circ$ , lo que traducido a km. es de aproximadamente 8 km. de error.

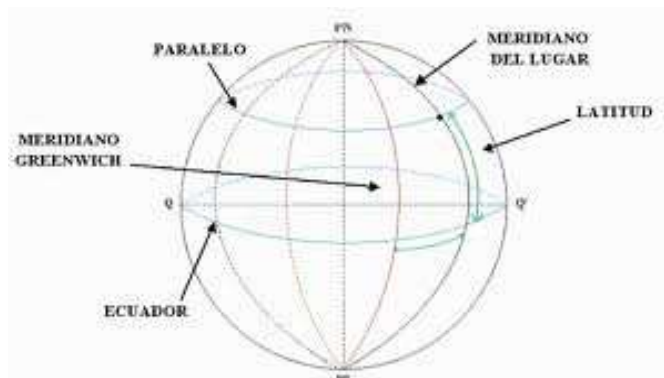


Fig.3.6.

### 3.3 ECUACIÓN FINAL

Por tanto, la ecuación que utilizaremos para calcular la longitud será la siguiente:

$$Longitud = (15 / 60) \cdot (HG[\text{min}] - ML[\text{min}] - E_t[\text{min}])$$

Donde HG será la hora de meridiano de Greenwich cuando el Sol pase por el meridiano de Greenwich, es decir, las 12 (720min.) y ML será la hora de meridiano de Greenwich cuando el Sol pase por el meridiano local. Entonces:

$$Longitud = 0.25 \cdot (720 - ML[\text{min}] - E_t[\text{min}])$$

## 4. MÉTODO DE CÁLCULO DE LA LATITUD.

### 4.1 ECUACIÓN DE LA LATITUD.

La fórmula que vamos a utilizar para calcular la latitud la obtenemos del triángulo de navegación. El triángulo de navegación (fig. 4.1.) es un triángulo esférico oblicuo formado por el polo Norte (PN), la posición en la que se encuentra el observador, en este caso nuestro robot (AP), y la posición en la que se encuentra el astro a observar, en nuestro caso el Sol (GP).

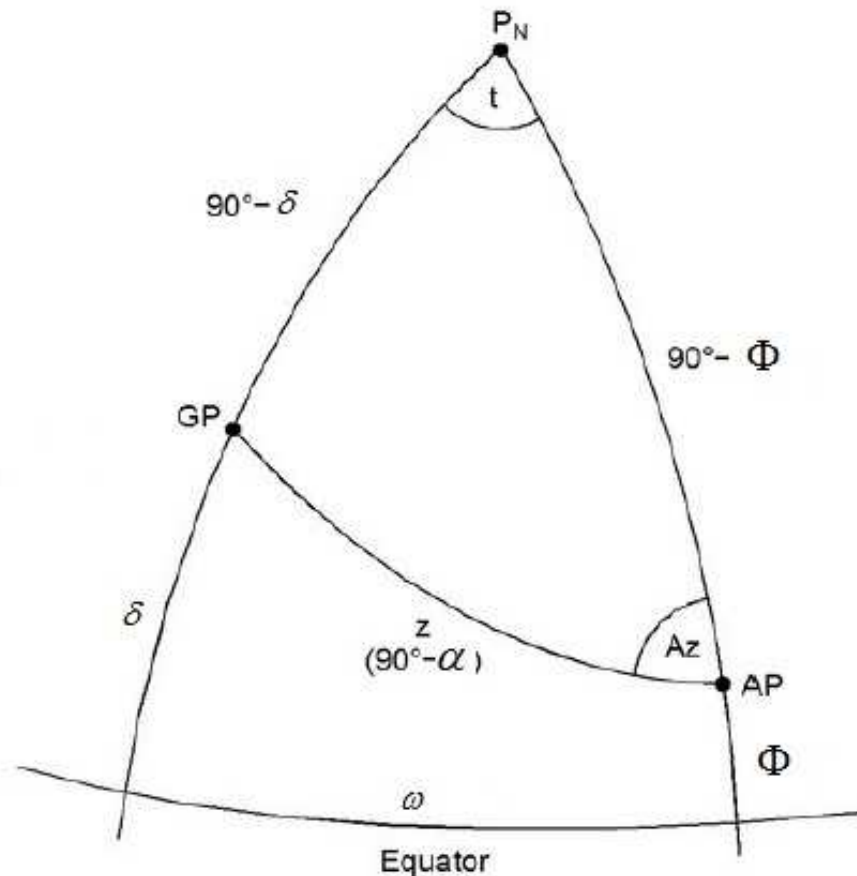


Fig. 4.1: Triángulo de navegación.

Donde  $\omega$  será el ángulo horario,  $\delta$  la declinación y  $\Phi$  la latitud donde se encuentra el observador. Con este triángulo calcularemos aplicando trigonometría el lado  $z$ . Nos quedará la siguiente fórmula:

$$\cos z = \cos(90^\circ - \Phi) \cdot \cos(90^\circ - \delta) + \operatorname{sen}(90^\circ - \Phi) \cdot \operatorname{sen}(90^\circ - \delta) \cdot \cos \omega$$

Como  $\cos(90^\circ - x) = \operatorname{sen} x$  y viceversa podemos simplificar la ecuación:

$$\cos z = \operatorname{sen} \Phi \cdot \operatorname{sen} \delta + \cos \Phi \cdot \cos \delta \cdot \cos \omega$$

Además, se da que el lado  $z$  es equivalente al arco que tenemos desde el cénit hasta el astro que queremos medir. Y la elevación solar ( $\alpha$ ) será:

$$\alpha = 90 - z$$

Entonces:

$$\text{sen}\alpha = \text{sen}\Phi \cdot \text{sen}\delta + \cos\Phi \cdot \cos\delta \cdot \cos\omega$$

Ahora despejamos la latitud, y como los demás datos los podemos medir o calcular supondremos que son una constante, e intentaremos buscar la siguiente ecuación:

$$\cos\Phi = a \cdot \text{sen}\Phi + b$$

Entonces:

$$-\cos\Phi \cdot \cos\delta \cdot \cos\omega = \text{sen}\Phi \cdot \text{sen}\delta - \text{sen}\alpha$$



$$\cos\Phi = \text{sen}\Phi \cdot \frac{\text{sen}\delta}{-\cos\delta \cdot \cos\omega} + \frac{\text{sen}\alpha}{\cos\delta \cdot \cos\omega}$$

Entonces:  $a = -\frac{\text{sen}\delta}{\cos\delta \cdot \cos\omega}$        $b = \frac{\text{sen}\alpha}{\cos\delta \cdot \cos\omega}$

Y ya tenemos la ecuación  $x = ay + b$ , lo elevamos al cuadrado y nos queda:

$$x^2 = a^2 y^2 + 2aby + b^2$$

Por otra parte tenemos:

$$\cos^2\Phi + \sin^2\Phi = 1$$

Que con la otra notación es:

$$x^2 + y^2 = 1 \quad \longrightarrow \quad x^2 = 1 - y^2$$

Sustituimos  $x^2$  en la ecuación y nos queda:

$$\begin{aligned} 1 - y^2 &= a^2 y^2 + 2aby + b^2 \\ 0 &= a^2 y^2 + 2aby + b^2 - 1 + y^2 \\ 0 &= (1 + a^2)y^2 + 2aby + b^2 - 1 \end{aligned}$$

Nos queda una ecuación de segundo grado y la resolvemos:

$$y = \frac{-2ab \pm \sqrt{4a^2b^2 - 4(1+a^2)(b^2-1)}}{2(1+a^2)}$$

Como  $y = \text{sen}\Phi$ , entonces:

$$\Phi = \text{sen}^{-1}(y)$$

Debemos tener en cuenta que al ser una ecuación de segundo grado obtendremos dos valores para la latitud. Esos valores serán la latitud en la que se cumplen esas condiciones en el hemisferio Norte y en el hemisferio Sur. Para elegir una de las dos latitudes deberemos utilizar el azimut, pero explicaremos el método en el apartado 4.3.

De los datos utilizados para calcular la latitud la elevación solar es el único que debemos medir. La declinación y el ángulo horario los podemos calcular matemáticamente con los datos introducidos o medidos.

Para realizar la medida de la elevación solar existen varios sensores que podemos utilizar, como por ejemplo, un seguidor solar de dos ejes con motores paso a paso o servomotores. Pero la elección del sensor más apropiado lo dejamos para otro proyecto.

## 4.2 CORRECCIONES DE LA ELEVACIÓN SOLAR

Una vez realizada la medición deberemos corregir los errores producidos por:

- La refracción atmosférica.
- La paralaje.
- El semidiámetro solar.

### 4.2.1. REFRACCIÓN ATMOSFÉRICA

Cuando un rayo de luz que proviene del Sol llega a la Tierra, este ha sido ligeramente desviado por la atmósfera. Este fenómeno es denominado refracción atmosférica, y ocurre siempre que el rayo entre con un ángulo menor de 90 grados.

Como no somos capaces de distinguir la curvatura del rayo de luz, suponemos que el Sol se encuentra al final de una línea recta tangente a la dirección en que medimos el rayo de luz. Por ello, cuando miramos el Sol medimos una elevación solar mayor de la real, como se muestra en la figura 4.2.

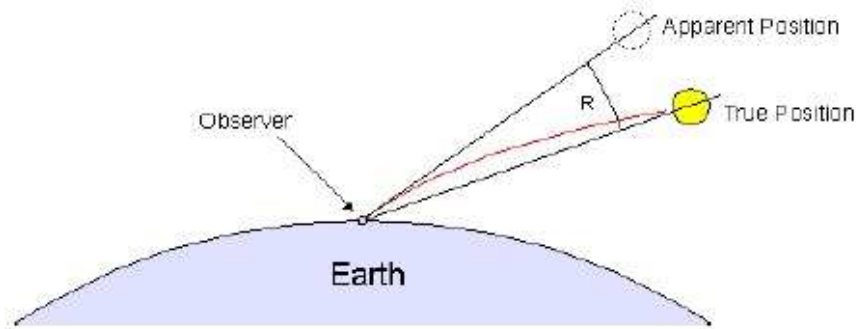


Fig. 4.2: Refracción atmosférica.

Llamamos R a la distancia angular existente entre la posición aparente y la posición real en la que se encuentra.

Existen bastantes formulas capaces de calcular esta distancia pero nosotros utilizaremos dos, la formula de Smart para elevaciones solares comprendidas entre 15° y 90° :

$$R['] = \frac{0.97127}{\tan \alpha[^\circ]} - \frac{0.00137}{\tan^3 \alpha[^\circ]}$$

Resultado en arcminutos (1/60 grados).

Y para elevaciones solares comprendidas entre 0° y 15° utilizaremos la siguiente:

$$R['] = \frac{34.133 + 4.197 \cdot \alpha + 0.00428 \cdot \alpha^2}{1 + 0.505 \cdot \alpha + 0.0845 \cdot \alpha^2}$$

Debemos tener en cuenta además que la refracción atmosférica está influenciada por la presión atmosférica y la temperatura del aire. Por ello, para obtener una corrección muy precisa deberemos multiplicarlo por un factor de corrección al que denominaremos f, y se calcula con la siguiente expresión:

$$f = \frac{P[mbar]}{1010} \cdot \frac{283}{273 + T[^\circ C]}$$

Siendo P la presión atmosférica y T la temperatura del aire. Las condiciones estándar son 1010 mbar (presión al nivel del mar aprox.) y 10° C.

En las siguientes tablas vamos a calcular como varía el error producido por la refracción atmosférica dependiendo de la elevación solar medida y la influencia que tienen la temperatura y la presión sobre él.



Error para elevaciones solares con  $f = 0,94$  ( $T=25^{\circ}$  C,  $P=1010$  mbar).

Elevación solar	Error refracción (°)	Error * f	Elevación solar	Error refracción (°)	Error * f
1	0,40	0,38	20	0,044	0,042
2	0,30	0,29	25	0,034	0,033
3	0,24	0,23	30	0,028	0,027
4	0,19	0,18	35	0,023	0,022
5	0,16	0,16	40	0,019	0,018
6	0,14	0,13	45	0,016	0,015
7	0,12	0,12	50	0,014	0,013
8	0,11	0,10	55	0,011	0,011
9	0,10	0,09	60	0,009	0,009
10	0,09	0,08	65	0,008	0,007
11	0,08	0,08	70	0,006	0,006
12	0,07	0,07	75	0,004	0,004
13	0,07	0,06	80	0,003	0,003
14	0,06	0,06	85	0,001	0,001
15	0,06	0,06	90	0,000	0,000

Error para elevaciones solares con  $f = 0,876$  ( $T=50^{\circ}$  C,  $P=1010$  mbar)

Elevación solar	Error Refraccion (°)	Error * f	Elevación solar	Error Refraccion (°)	Error * f
1	0,40	0,35	20	0,044	0,039
2	0,30	0,26	25	0,034	0,030
3	0,24	0,21	30	0,028	0,024
4	0,19	0,17	35	0,023	0,020
5	0,16	0,14	40	0,019	0,017
6	0,14	0,12	45	0,016	0,014
7	0,12	0,11	50	0,014	0,012
8	0,11	0,10	55	0,011	0,010
9	0,10	0,09	60	0,009	0,008
10	0,09	0,08	65	0,008	0,007
11	0,08	0,07	70	0,006	0,005
12	0,07	0,06	75	0,004	0,004
13	0,07	0,06	80	0,003	0,003
14	0,06	0,06	85	0,001	0,001
15	0,06	0,05	90	0,000	0,000

Error para elevaciones solares con  $f = 1,165$  ( $T = -30^{\circ} \text{C}$ ,  $P = 1010 \text{ mbar}$ ).

Elevación solar	Error Refracción (°)	Error * f	Elevación solar	Error Refracción (°)	Error * f
1	0,40	0,47	20	0,044	0,051
2	0,30	0,35	25	0,034	0,040
3	0,24	0,28	30	0,028	0,033
4	0,19	0,23	35	0,023	0,027
5	0,16	0,19	40	0,019	0,022
6	0,14	0,16	45	0,016	0,019
7	0,12	0,14	50	0,014	0,016
8	0,11	0,13	55	0,011	0,013
9	0,10	0,11	60	0,009	0,011
10	0,09	0,10	65	0,008	0,009
11	0,08	0,09	70	0,006	0,007
12	0,07	0,09	75	0,004	0,005
13	0,07	0,08	80	0,003	0,003
14	0,06	0,07	85	0,001	0,002
15	0,06	0,07	90	0,000	0,000

Error para elevaciones solares con  $f = 0,897$  ( $T = 25^{\circ} \text{C}$ ,  $P = 954 \text{ mbar}$ ).

Elevación solar	Error Refracción (°)	Error * f	Elevación solar	Error Refracción (°)	Error * f
1	0,40	0,36	20	0,044	0,039
2	0,30	0,27	25	0,034	0,031
3	0,24	0,21	30	0,028	0,025
4	0,19	0,17	35	0,023	0,021
5	0,16	0,15	40	0,019	0,017
6	0,14	0,13	45	0,016	0,015
7	0,12	0,11	50	0,014	0,012
8	0,11	0,10	55	0,011	0,010
9	0,10	0,09	60	0,009	0,008
10	0,09	0,08	65	0,008	0,007
11	0,08	0,07	70	0,006	0,005
12	0,07	0,07	75	0,004	0,004
13	0,07	0,06	80	0,003	0,003
14	0,06	0,06	85	0,001	0,001
15	0,06	0,05	90	0,000	0,000

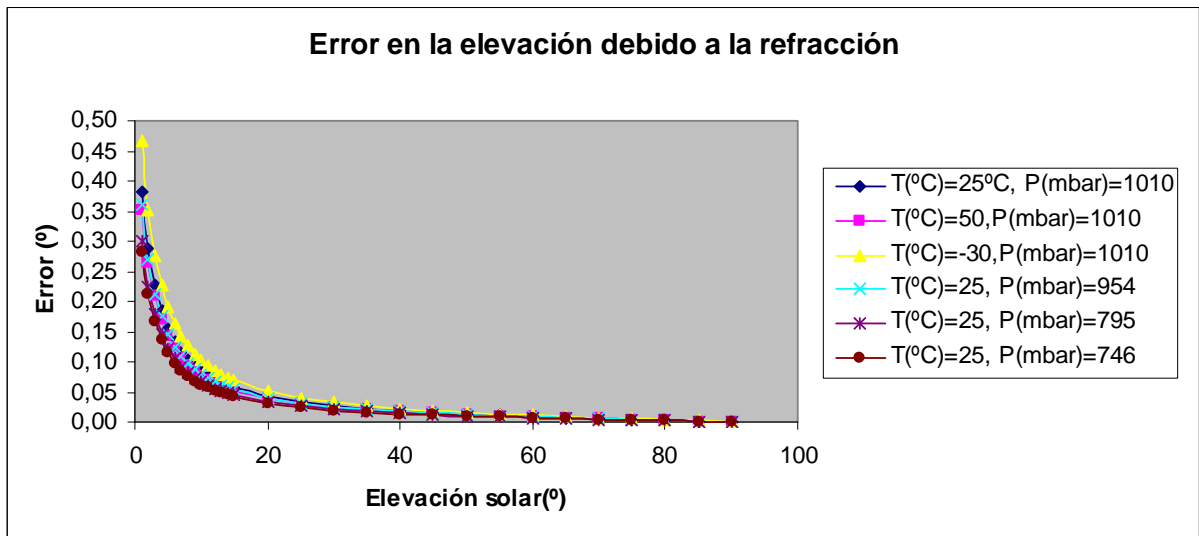
Error para elevaciones solares con  $f = 0,748$  ( $T= 25^{\circ}\text{C}$ ,  $P=795$  mbar ).

Elevación solar	Error Refraccion (°)	Error * f	Elevación solar	Error Refraccion (°)	Error * f
1	0,40	0,30	20	0,044	0,033
2	0,30	0,23	25	0,034	0,026
3	0,24	0,18	30	0,028	0,021
4	0,19	0,15	35	0,023	0,017
5	0,16	0,12	40	0,019	0,014
6	0,14	0,10	45	0,016	0,012
7	0,12	0,09	50	0,014	0,010
8	0,11	0,08	55	0,011	0,008
9	0,10	0,07	60	0,009	0,007
10	0,09	0,07	65	0,008	0,006
11	0,08	0,06	70	0,006	0,004
12	0,07	0,06	75	0,004	0,003
13	0,07	0,05	80	0,003	0,002
14	0,06	0,05	85	0,001	0,001
15	0,06	0,04	90	0,000	0,000

Error para elevaciones solares con  $f = 0,701$  ( $T= 25^{\circ}\text{C}$ ,  $P=746$  mbar ).

Elevación solar	Error Refraccion (°)	Error * f	Elevación solar	Error Refraccion (°)	Error * f
1	0,40	0,28	20	0,044	0,031
2	0,30	0,21	25	0,034	0,024
3	0,24	0,17	30	0,028	0,020
4	0,19	0,14	35	0,023	0,016
5	0,16	0,11	40	0,019	0,014
6	0,14	0,10	45	0,016	0,011
7	0,12	0,09	50	0,014	0,010
8	0,11	0,08	55	0,011	0,008
9	0,10	0,07	60	0,009	0,007
10	0,09	0,06	65	0,008	0,005
11	0,08	0,06	70	0,006	0,004
12	0,07	0,05	75	0,004	0,003
13	0,07	0,05	80	0,003	0,002
14	0,06	0,04	85	0,001	0,001
15	0,06	0,04	90	0,000	0,000

En la siguiente gráfica representamos los resultados obtenidos:



De la gráfica podemos obtener las siguientes conclusiones:

- La influencia debida a la presión y la temperatura varía el error con elevaciones solares muy cercanas a 0°. A 5° de elevación solar el error cometido debido a la presión y la temperatura no supera los 0,05°, y que es un error que podemos asumir para nuestro proyecto.
- El error será significativo con elevaciones inferiores a 10°, donde el error es 0,1° o superior. Por tanto, en nuestro programa deberemos introducir la corrección para elevaciones menores de 15°.

#### 4.2.2 PARALAJE

Los cálculos de navegación astronómica se refieren a la elevación que tiene el Sol respecto del centro de la Tierra y el horizonte astronómico. Pero cuando nosotros realizamos la medida, la realizamos desde un horizonte aparente tal y como se muestra en la figura 4.3.

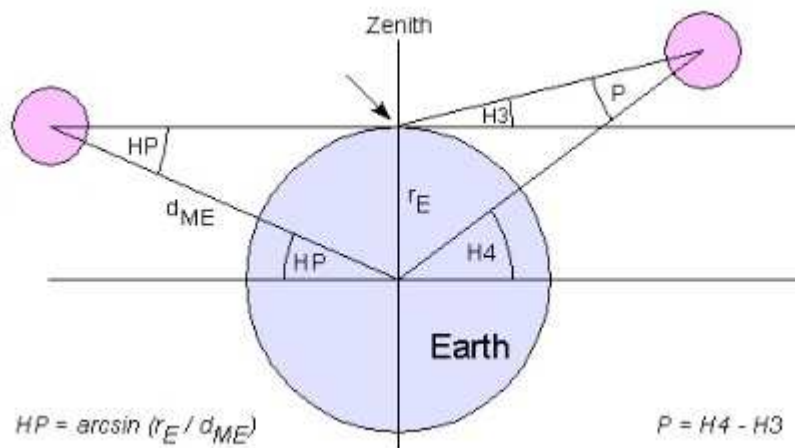


Fig. 4.2: Paralaje.

El lugar donde se encuentra la flecha es donde se encontraría el observador, la línea perpendicular al cénit que pasa por él será el horizonte aparente y la paralela a ésta, y que pasa por el centro, será el horizonte astronómico.

Existe un ángulo en que el Sol se encuentra alineado con el horizonte aparente. A este ángulo lo llamamos paralaje horizontal y es  $0,15'$  (dato obtenido del almanaque náutico (Tabla 4.1)). Para calcular la paralaje cuando no está alineado con el horizonte aparente utilizamos la siguiente fórmula:

$$P = \arcsen(\sin HP \cdot \cos \alpha) \approx HP \cdot \cos \alpha$$

Como el  $\cos \alpha$  va estar siempre comprendido entre 0 y 1 nuestro paralaje máximo será  $0,15'$  lo que es equivalente a  $2,4 \cdot 10^{-3}^\circ$ . Dado que esto es muy pequeño podremos despreciar la paralaje.

### 4.2.3 SEMIDIAMETRO SOLAR

Cuando observamos el Sol, por ejemplo, con un sextante, no somos capaces de localizar el centro del Sol con suficiente precisión. Nos pasará lo mismo con nuestro sensor.

Para solucionar este problema utilizaremos el semidiámetro solar (SD), que es la distancia angular que habrá entre el centro del sol y la circunferencia exterior del Sol, tal y como se muestra en la fig. 4.3.

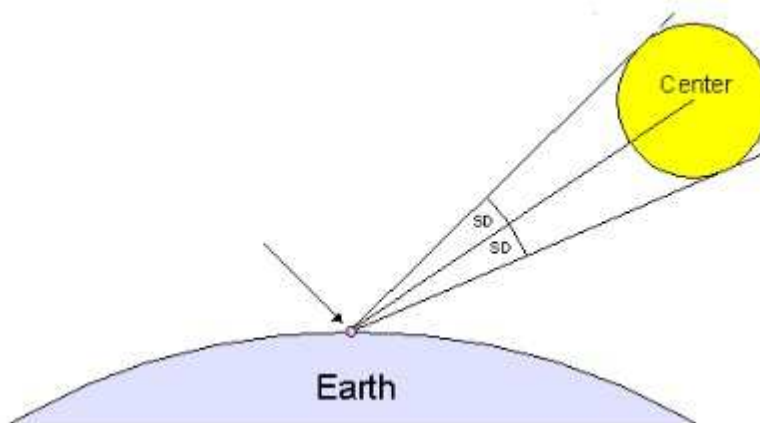


Fig. 4.3: Semidiámetro solar.

Para obtener el valor del semidiámetro solar deberemos dirigirnos al Almanaque náutico de cualquier año, ya que este va variando debido a que el Sol no es una esfera perfecta. Todos los años su variación es la misma. En la siguiente tabla podemos ver el semidiámetro solar del Sol, obtenido del almanaque náutico del 2010:

## Horizontal Parallaxes and Semi-Diameters for 2010

	Sun	Venus		Mars		Jupiter	Saturn
	SD	HP	SD	HP	SD	SD	SD
Jan 1	16'.3	0'.09	0'.08	0'.20	0'.11	0'.27	0'.13
Jan	16'.3	0'.09	0'.08	0'.22	0'.11	0'.27	0'.14
Feb	16'.2	0'.09	0'.08	0'.21	0'.11	0'.26	0'.14
Mar	16'.1	0'.09	0'.09	0'.17	0'.09	0'.26	0'.14
Apr	15'.9	0'.10	0'.09	0'.13	0'.07	0'.27	0'.14
May	15'.8	0'.11	0'.10	0'.10	0'.06	0'.29	0'.14
Jun	15'.7	0'.12	0'.12	0'.09	0'.05	0'.31	0'.13
Jul	15'.7	0'.15	0'.14	0'.08	0'.04	0'.34	0'.12
Aug	15'.8	0'.20	0'.19	0'.07	0'.04	0'.37	0'.12
Sep	15'.9	0'.31	0'.29	0'.07	0'.04	0'.39	0'.12
Oct	16'.0	0'.49	0'.47	0'.06	0'.03	0'.38	0'.12
Nov	16'.2	0'.47	0'.45	0'.06	0'.03	0'.35	0'.12
Dec	16'.2	0'.30	0'.28	0'.06	0'.03	0'.32	0'.12
Dec 31	16'.3	0'.24	0'.23	0'.06	0'.03	0'.31	0'.13

15th day of each month unless stated

Sun's HP = 0'.15

Tabla 4.1: Paralaje y semidiámetros de algunos astros. (Almanaque Náutico 2010)

Programaremos nuestro sensor para que mida la elevación solar del extremo inferior y, como podemos ver en la tabla, éste varía entre 15,7' y 16,3'. Por tanto, tomaremos un valor de 16' para realizar la corrección. Tendremos un error máximo de 0,3' en la medida que es menor de 0,01° y que podemos asumir.

Por tanto, la elevación que introduciremos en la fórmula de la latitud descrita anteriormente será:

$$\alpha_f = \alpha - R + SD$$

Siendo  $\alpha$  la elevación solar medida, R el error cometido debido a la refracción y SD el error cometido debido al semidiámetro solar.

### 4.3 MÉTODO PARA ELECCIÓN DE LA LATITUD.

Como hemos explicado anteriormente, en la ecuación de la latitud obtenemos dos valores, la latitud en el hemisferio Norte y la latitud en el hemisferio Sur. Para elegir una u otra debemos calcular el acimut teórico y el acimut real y compararlos.

Para medir el acimut real colocaremos una brújula electrónica, ya que deberemos tener una referencia para medir el ángulo acimutal. La elección del sensor y la brújula solar las dejaremos para otro proyecto.

Una vez medido el acimut real tenemos que calcular el acimut teórico para cada una de las latitudes obtenidas en el cálculo de la latitud.

La fórmula para obtener el ángulo acimutal se obtiene, al igual que la latitud, del triangulo de navegación. Aplicamos trigonometría y obtenemos la siguiente ecuación:

$$\cos(90 - \delta) = \cos(90 - \Phi) \cdot \cos z + \sin(90 - \Phi) \cdot \sin(90 - \Phi) \cdot \sin z \cdot \cos \Psi$$

Donde  $\Psi$  es el acimut. Además, sabemos que  $\cos(90^\circ - x) = \sin x$  y que  $\sin z = \cos \alpha$  por lo explicado en el apartado 4.1. Sustituimos en la ecuación y nos queda:

$$\sin \delta = \sin \Phi \cdot \sin \alpha + \cos \Phi \cdot \cos \alpha \cdot \cos \Psi$$

Resolvemos la ecuación y nos queda que:

$$\Psi = \arccos \frac{\sin \delta - \sin \Phi \cdot \sin \alpha}{\cos \Phi \cdot \cos \alpha}$$

Debemos tener en cuenta que el arc cos solo devuelve valores comprendidos entre  $0^\circ$  y  $180^\circ$ , entonces la función nos devolverá un valor negativo cuando el Sol se encuentre al Este del mediodía solar (ángulo horario ( $\omega$ ) es negativo) y un valor positivo cuando el Sol se encuentre al Oeste del mediodía solar (ángulo horario ( $\omega$ ) positivo). Por tanto, deberemos restar el valor obtenido a  $360^\circ$  para obtener el valor real de nuestro acimut.

Teniendo en cuenta que nosotros siempre realizaremos la medida después del mediodía solar debido al método utilizado para calcular la longitud, tendremos siempre que restar el resultado obtenido a  $360^\circ$ . Entonces nuestro acimut será:

$$\Psi = 360^\circ - \arccos \frac{\sin \delta - \sin \Phi \cdot \sin \alpha_f}{\cos \Phi \cdot \cos \alpha_f}$$

Una vez calculado el acimut teórico lo compararemos con el medido y elegiremos la latitud relacionada con el acimut más cercano al medido y obtendremos la latitud a la que se encuentra nuestro robot.





## 5. OBTENCIÓN DE LAS MEDIDAS

Para realizar las medidas de los ángulos horarios antes y después del mediodía solar deberemos elegir una elevación solar. Elegiremos una que nos de la opción de obtener la latitud y longitud en la mayor parte de la superficie terrestre. Nos decidimos por una elevación solar de  $5^\circ$ , ya que es lo suficientemente baja para obtener resultados a latitudes relativamente altas y lo suficientemente alta para no tener problemas para medirla con el sensor.

Para realizar la medida antes del mediodía, mediremos la elevación solar, si esta es menor de  $5^\circ$  introduciremos el programa en un bucle que realizará la medida de la elevación solar hasta que esta supere los  $5^\circ$ . En ese instante, guardaremos la hora y la elevación medidas. Si la elevación es superior a  $5^\circ$  cuando realizamos la medida, realizaremos otra medida minutos después para comprobar si el Sol se está elevando o bajando. Si se está elevando guardaremos la hora y la medida de la elevación solar, si esta bajando esperaremos a que vuelva a subir y sea mayor de  $5^\circ$  para realizar la medida.

La segunda medida se realizará cuando el Sol pase por la elevación solar guardada en la primera medida. Para ello, introduciremos el programa en un bucle de manera que cuando el sensor detecte una elevación igual o menor que la guardada en la primera medida guarde la hora de la segunda medida. Como hemos explicado anteriormente en el apartado 3, para que la medida sea fiable, el intervalo entre la primera y la segunda medida debe ser mayor de dos horas, por ello, si el intervalo es menor, nos indicará que la medida no es fiable.

Una vez obtenidos los datos llamaremos a los subprogramas correspondientes para realizar los cálculos y obtener la latitud y longitud con la metodología explicada en los apartados anteriores.

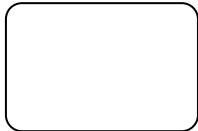


## 6. DIAGRAMAS DE FLUJO

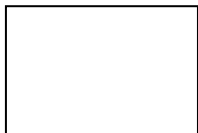
En este apartado desarrollaremos el programa implementado mediante diagramas de flujo.

### 6.1. EXPLICACIÓN DE LOS SÍMBOLOS UTILIZADOS

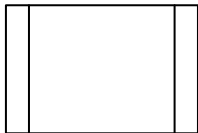
Descripción de los diferentes símbolos utilizados para realizar los diagramas de flujo:



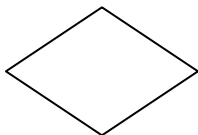
Este símbolo nos indica el inicio y el final de un diagrama de flujo.



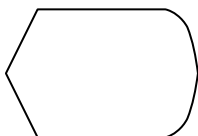
Este símbolo nos indica que se realiza una acción.



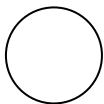
Este símbolo nos indica que llamamos a otro programa.



Este símbolo nos indica una toma de decisión y nos indica las distintas posibilidades dentro del diagrama.

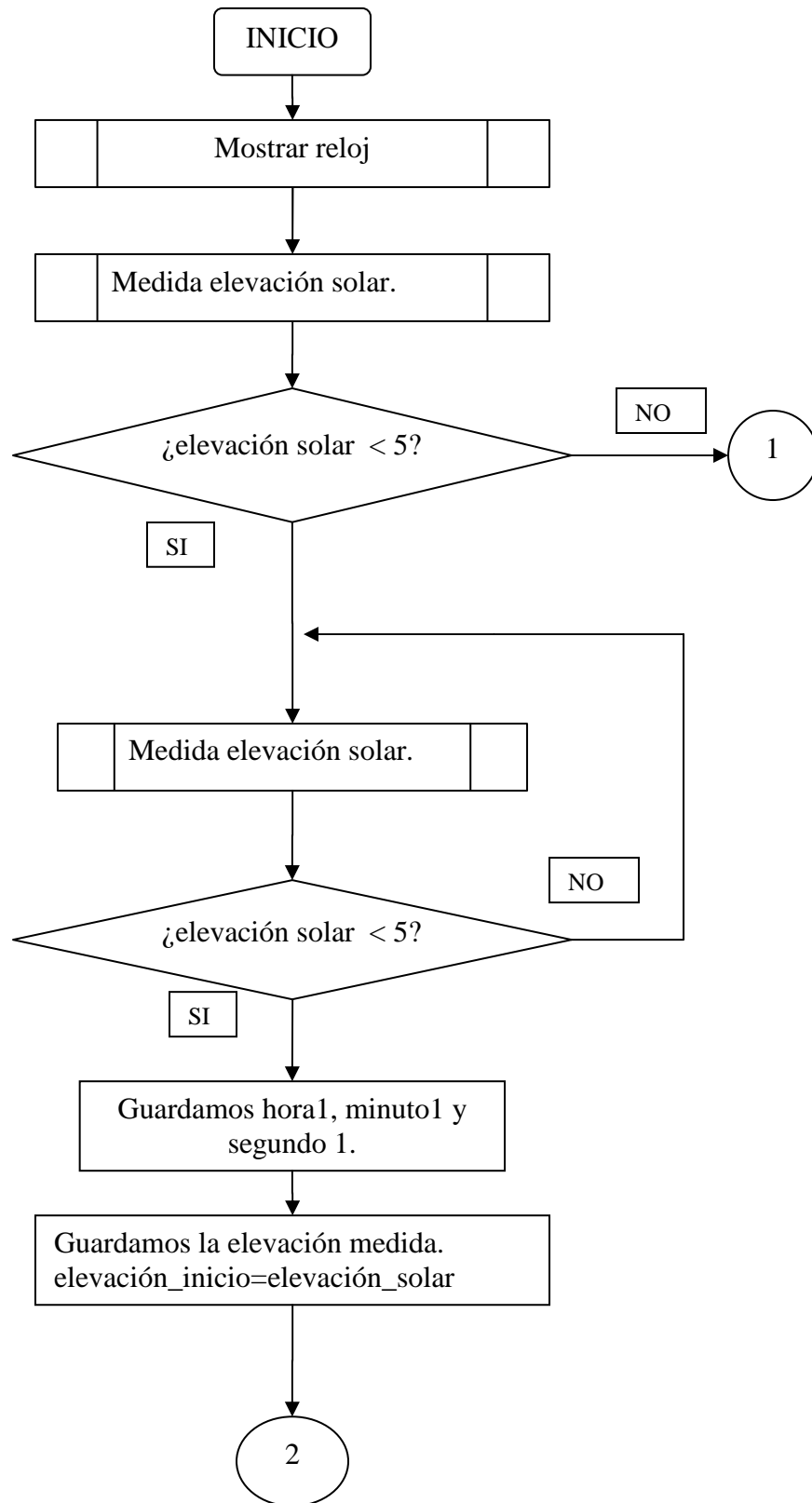


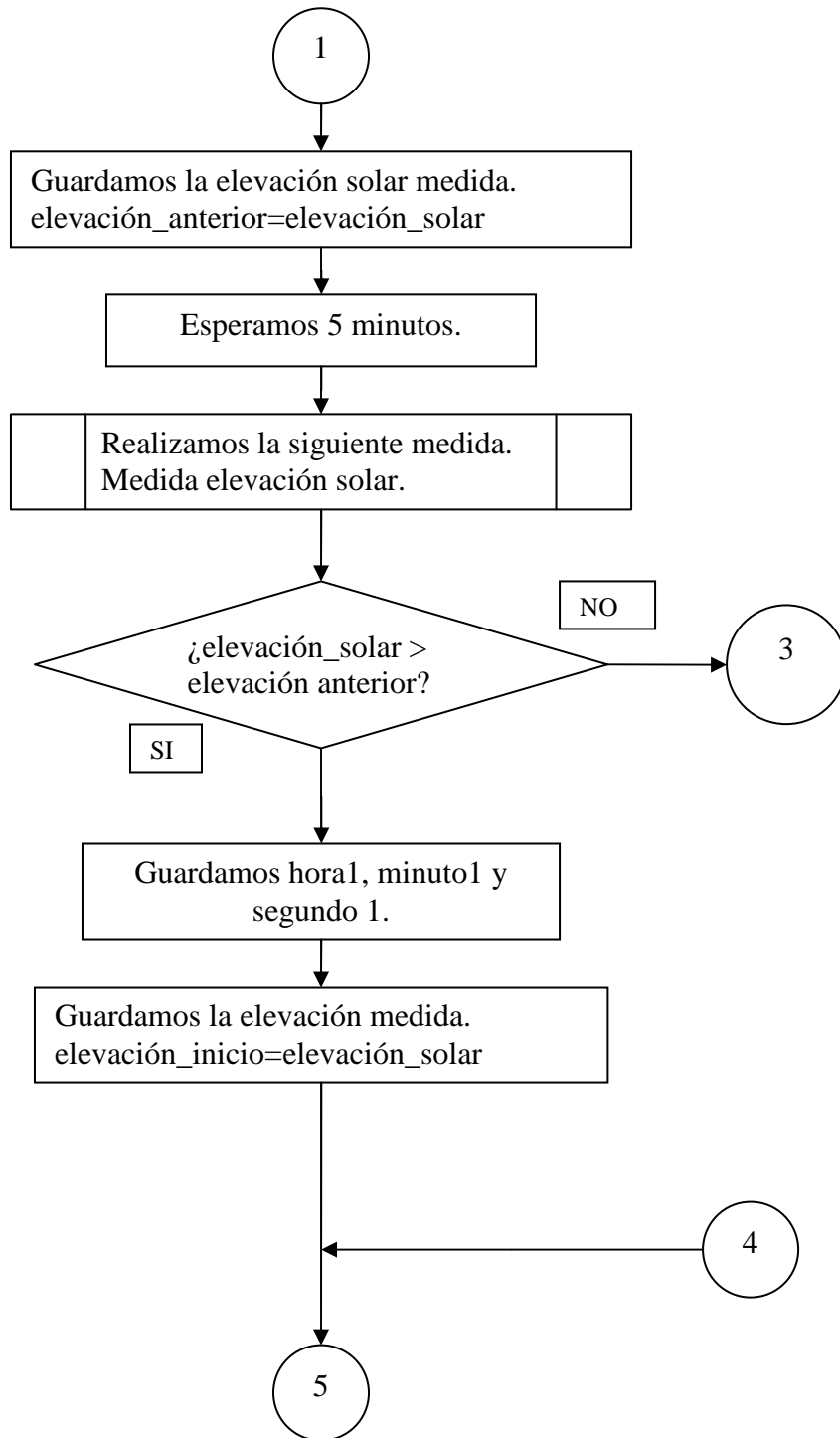
Este símbolo nos indica que se muestra en pantalla.

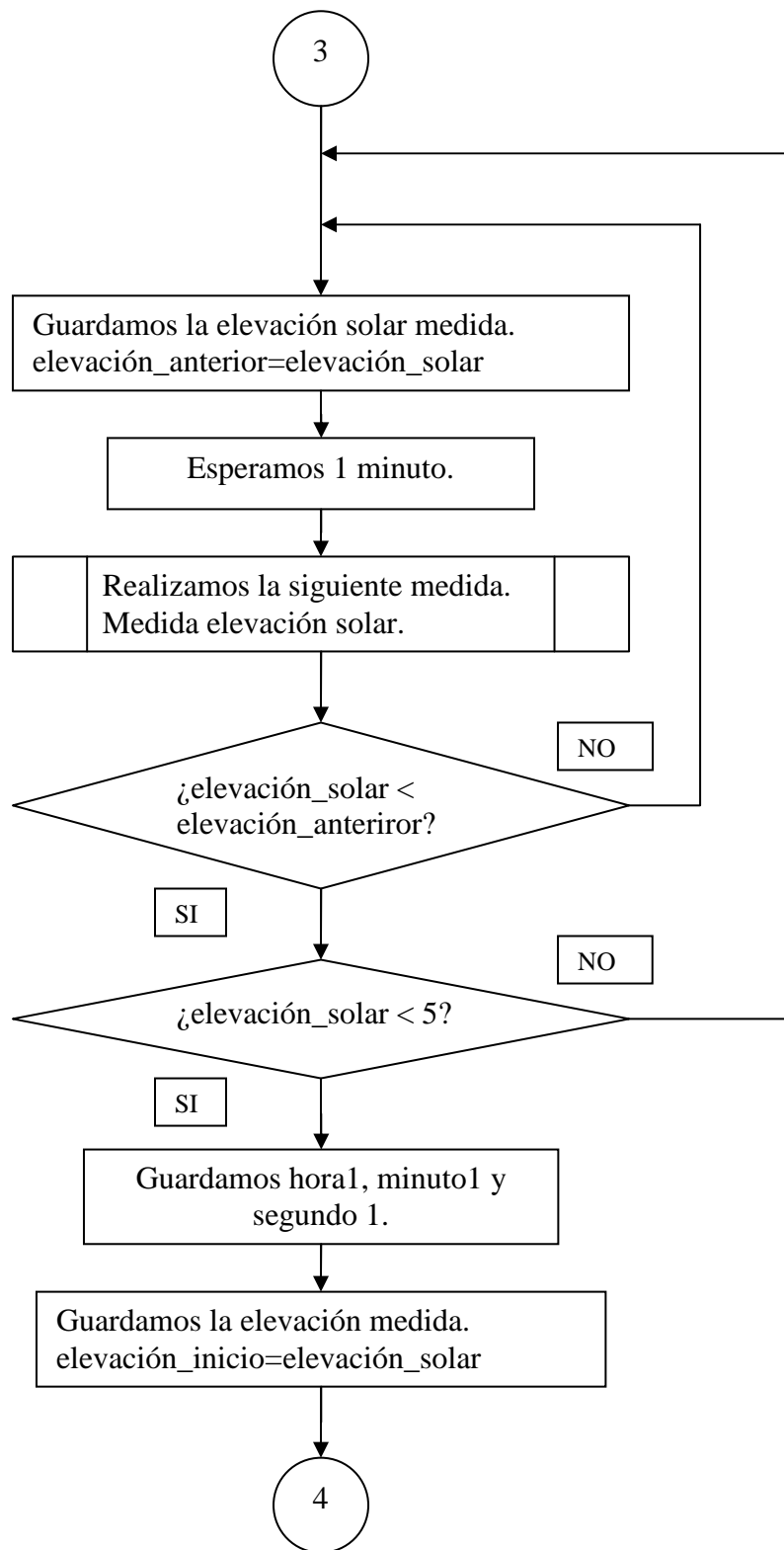


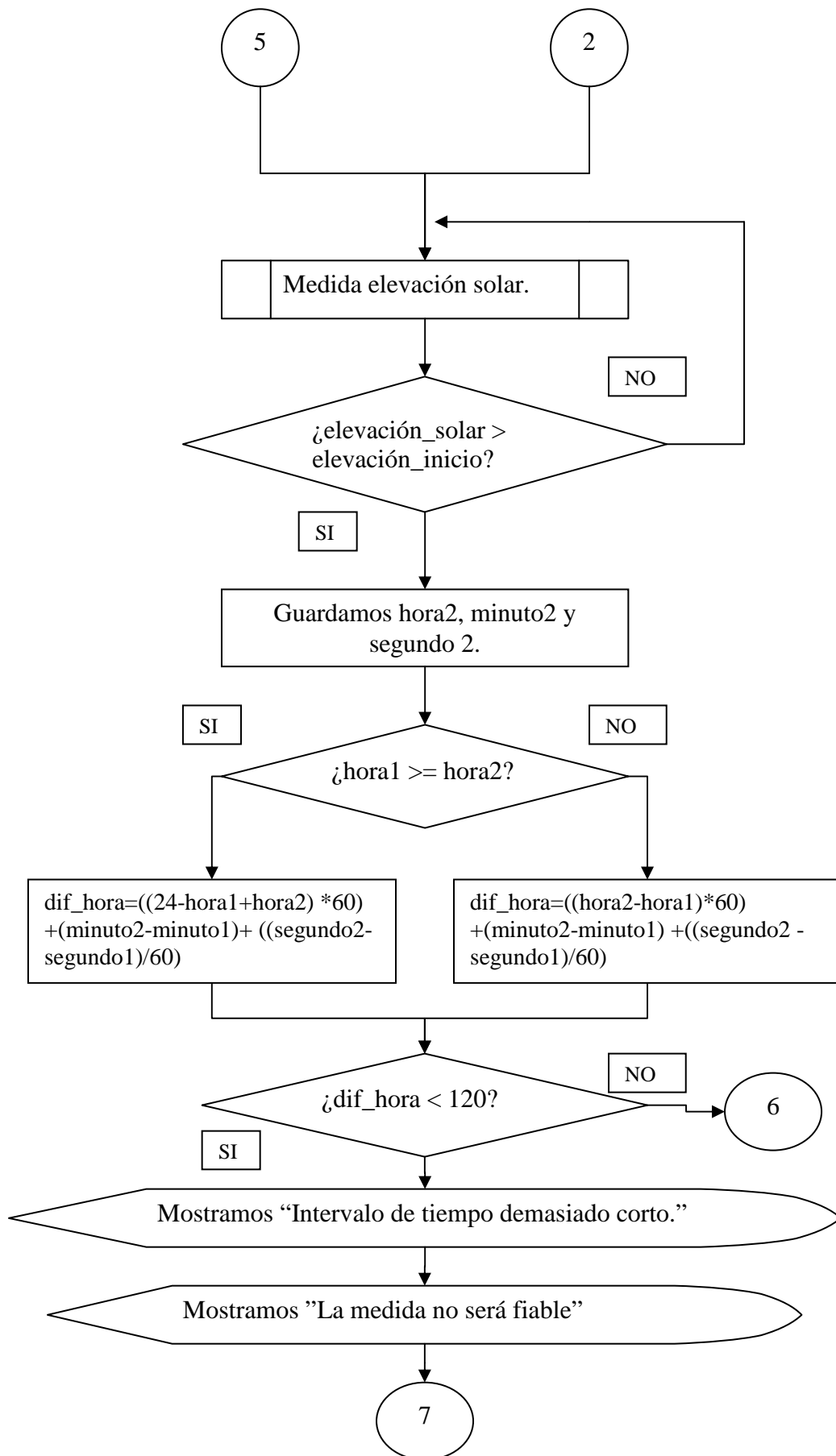
Este símbolo se utiliza como conector entre dos diagramas.

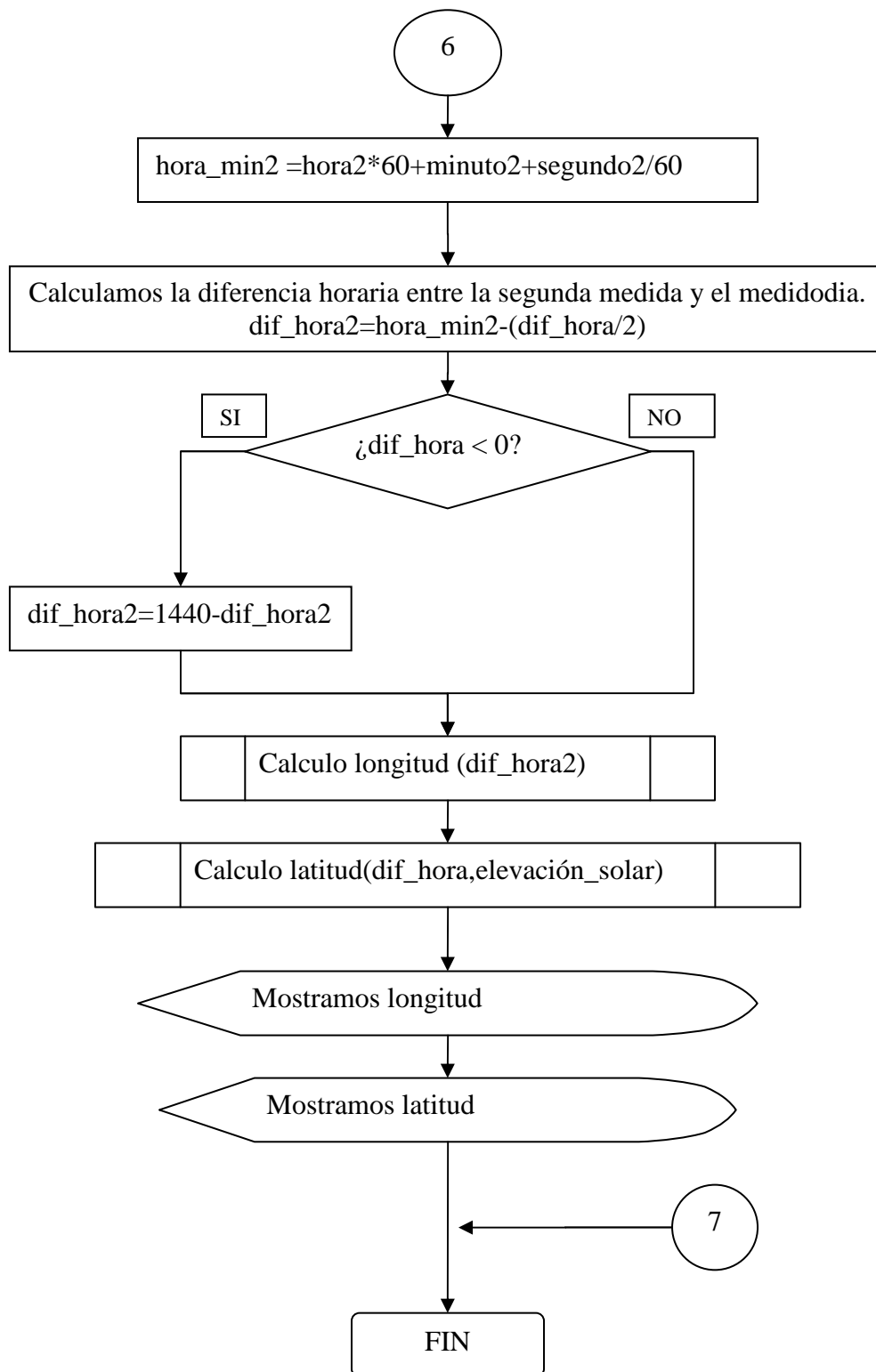
## 6.2. PROGRAMA PRINCIPAL





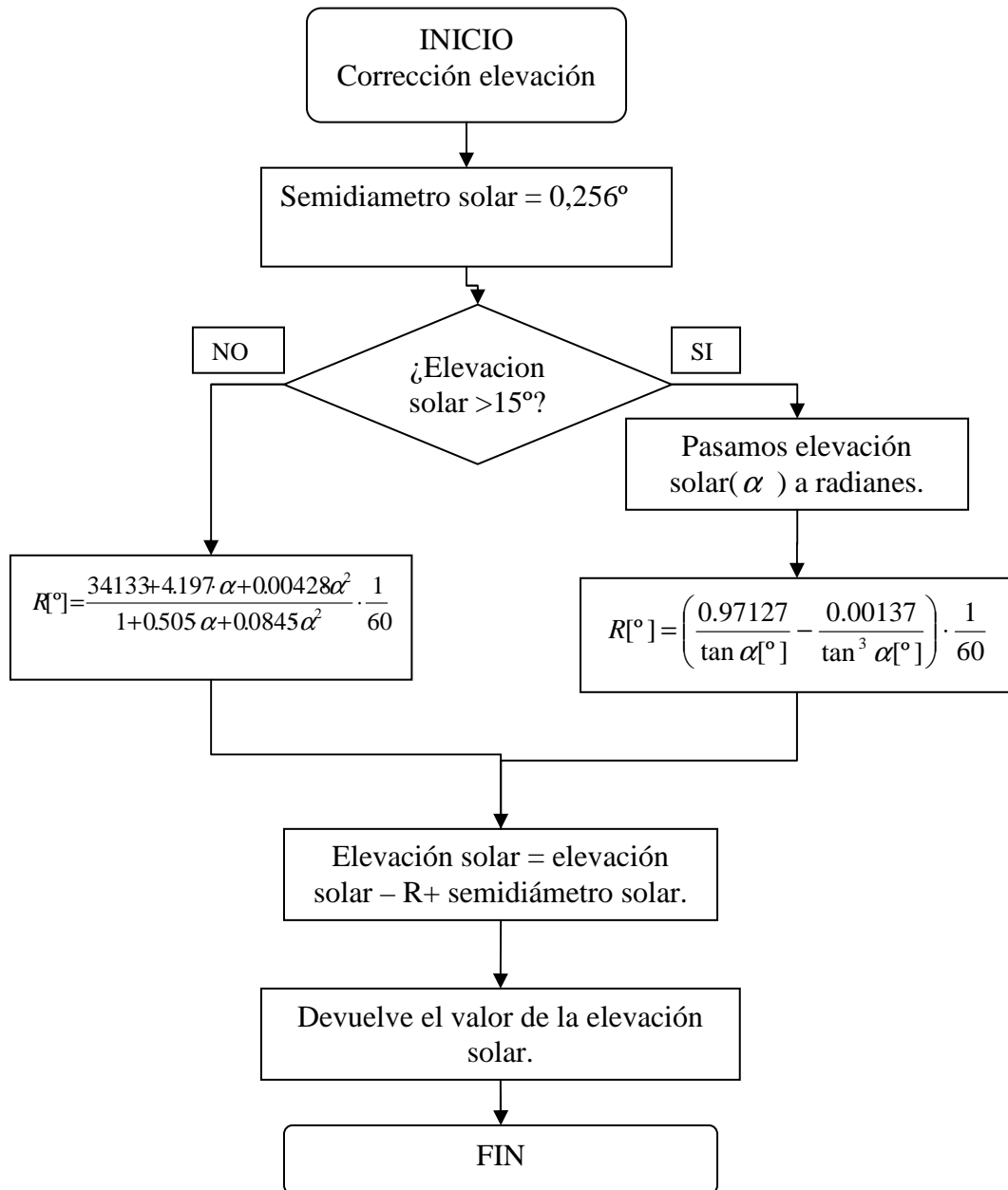




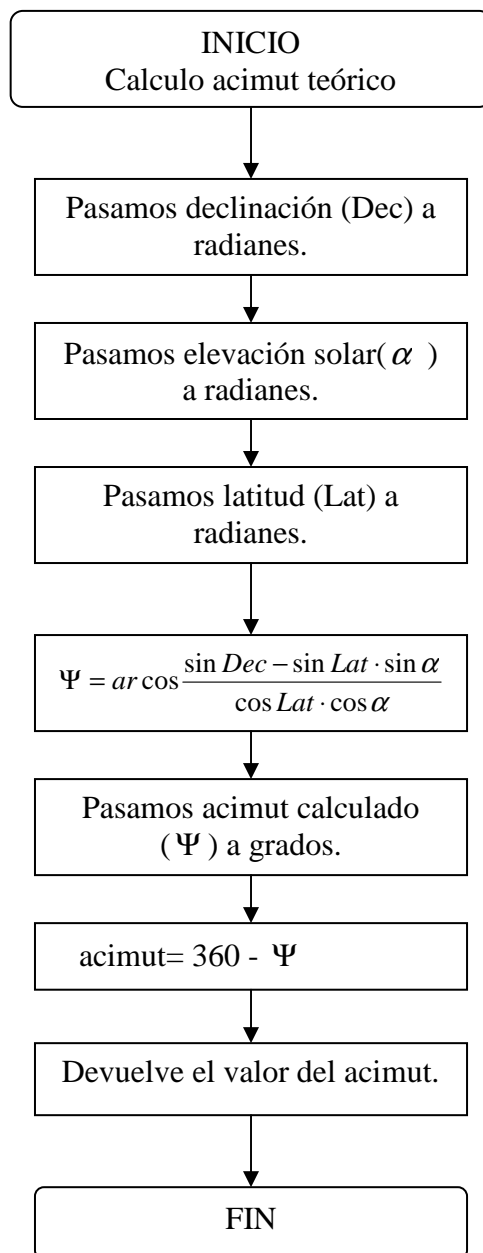




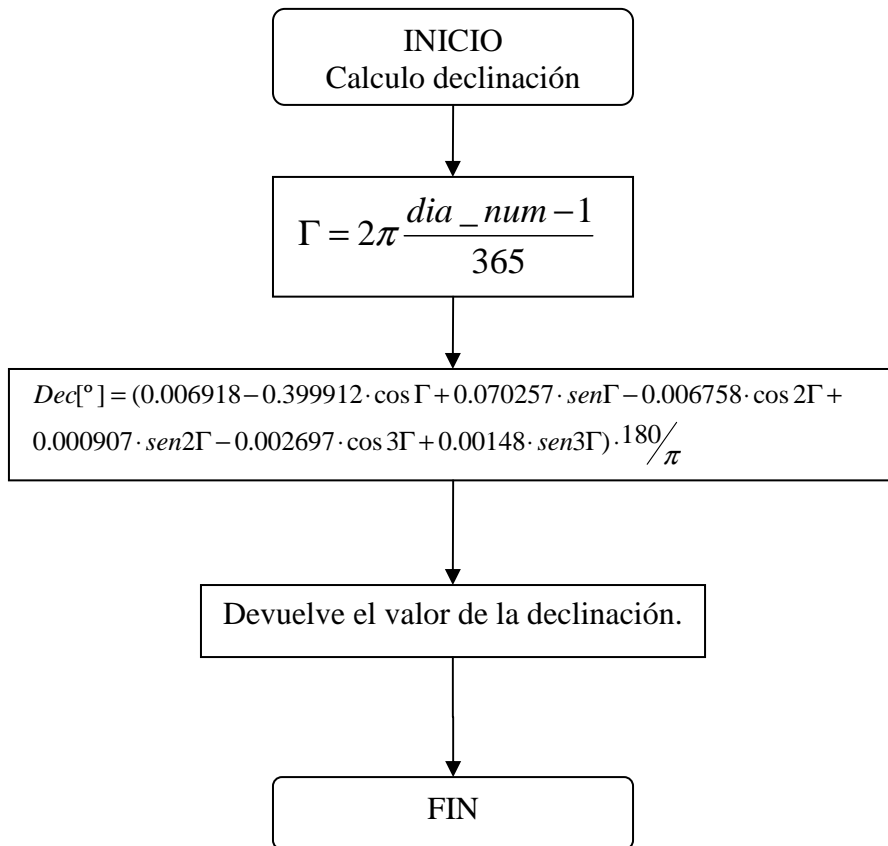
### 6.3 SUBPROGRAMA CORRECCIÓN ELEVACIÓN



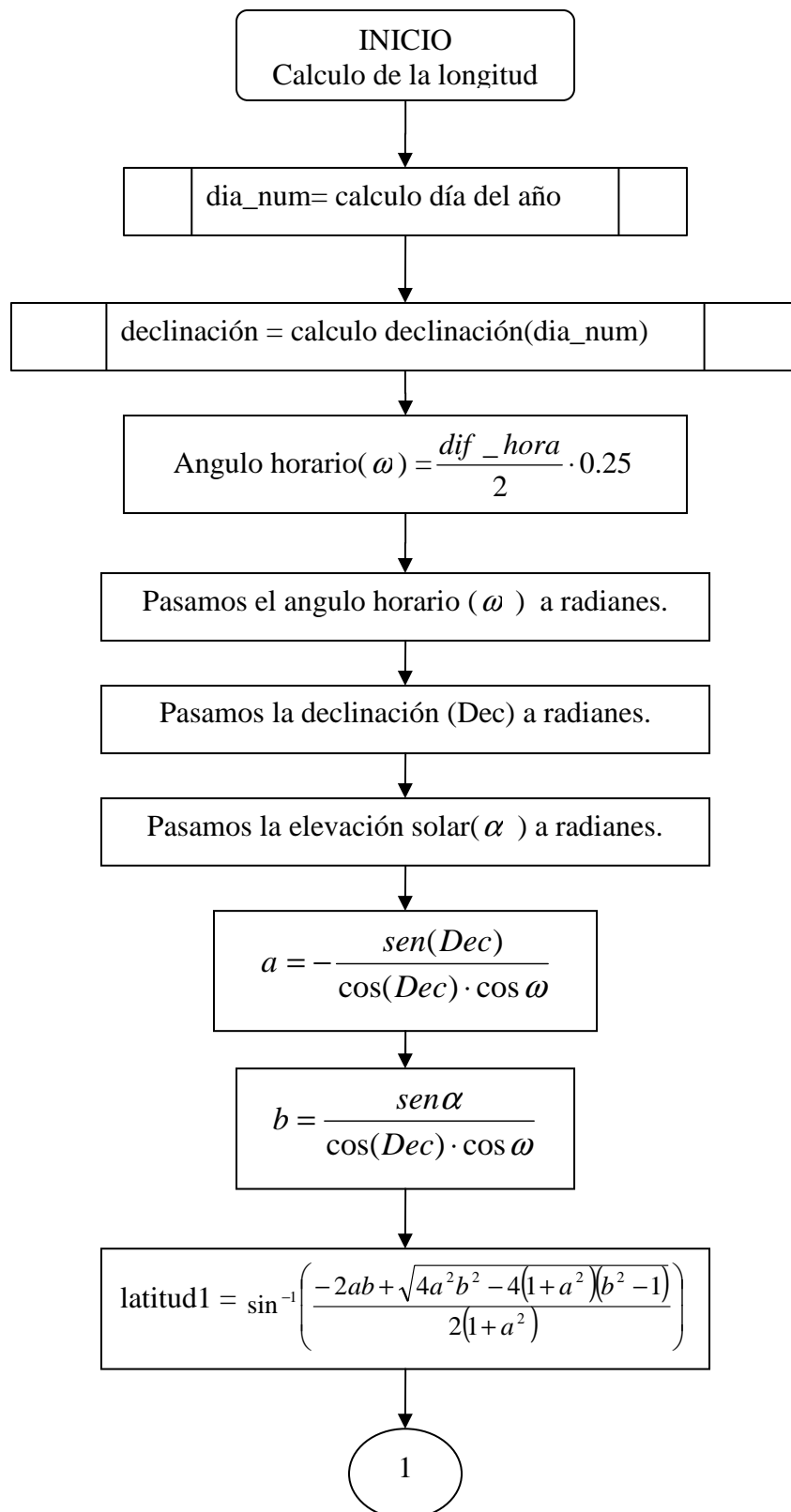
## 6.4 SUBPROGRAMA CÁLCULO ACIMUT TEÓRICO

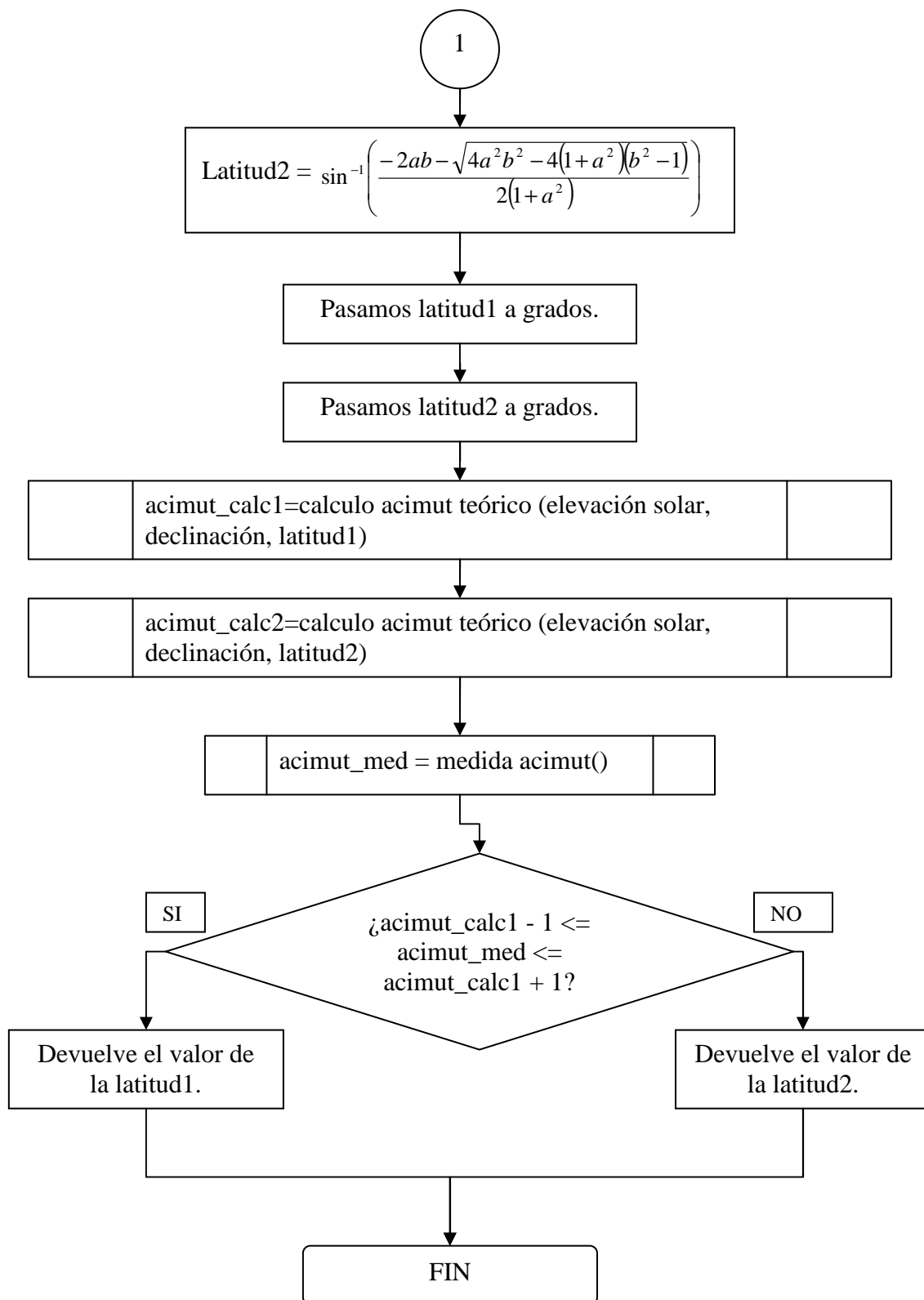


## 6.5 SUBPROGRAMA CÁLCULO DECLINACIÓN

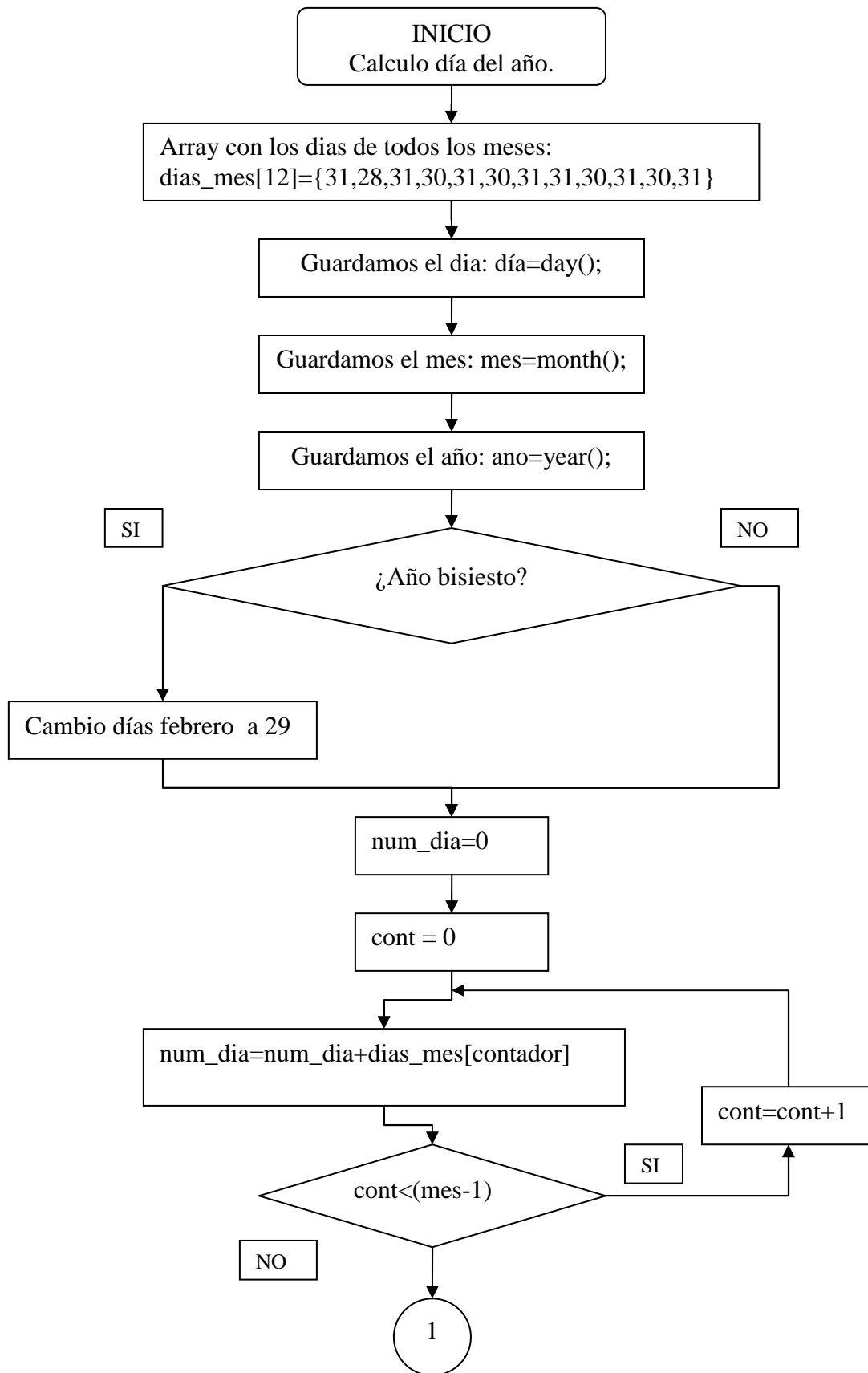


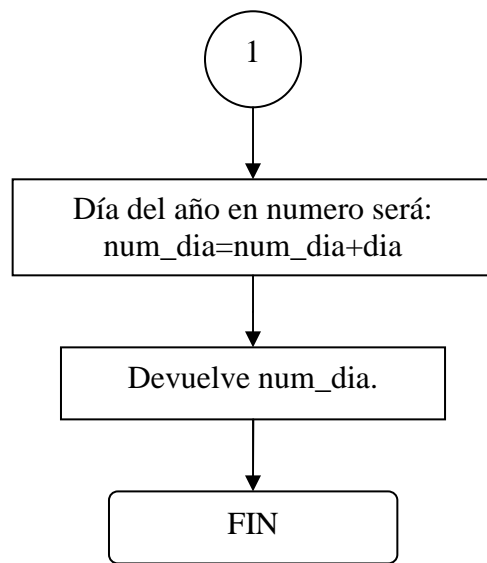
## 6.6 SUBPROGRAMA CÁLCULO LATITUD



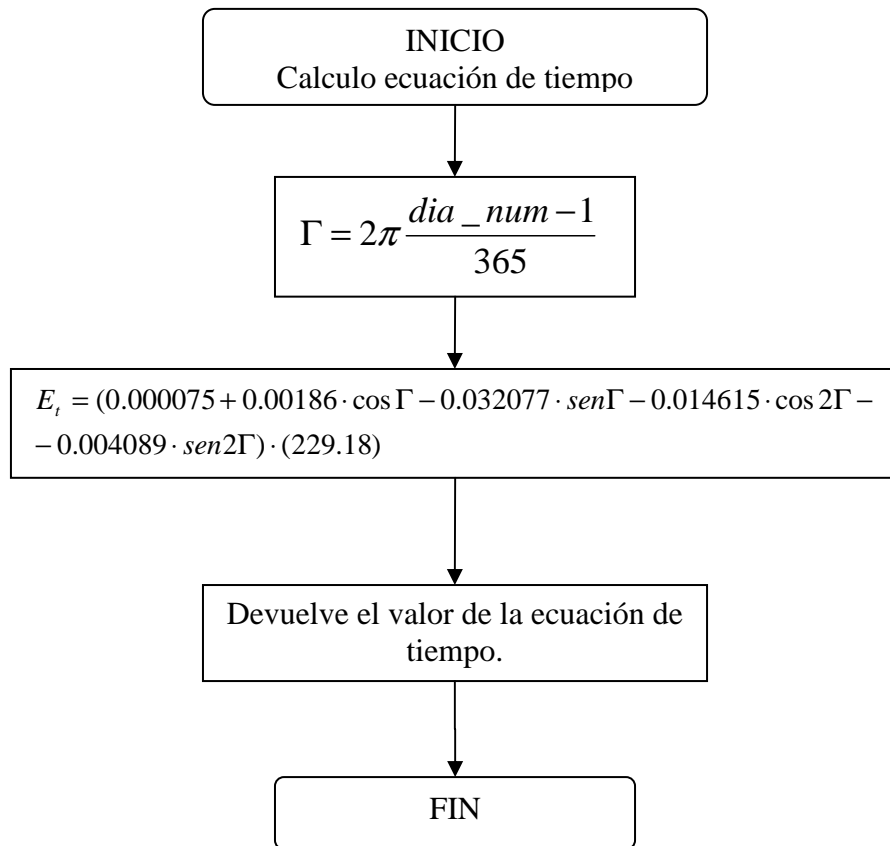


## 6.7 SUBPROGRAMA CÁLCULO DÍA DEL AÑO



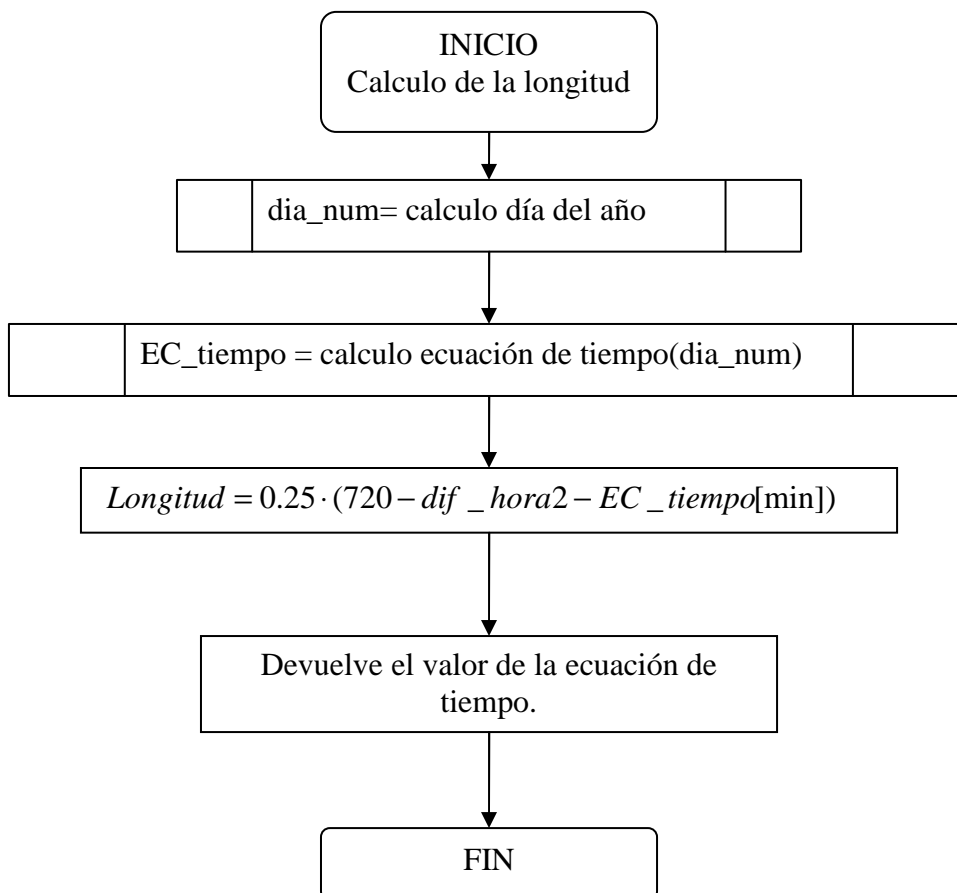


## 6.8 SUBPROGRAMA CÁLCULO ECUACIÓN DE TIEMPO

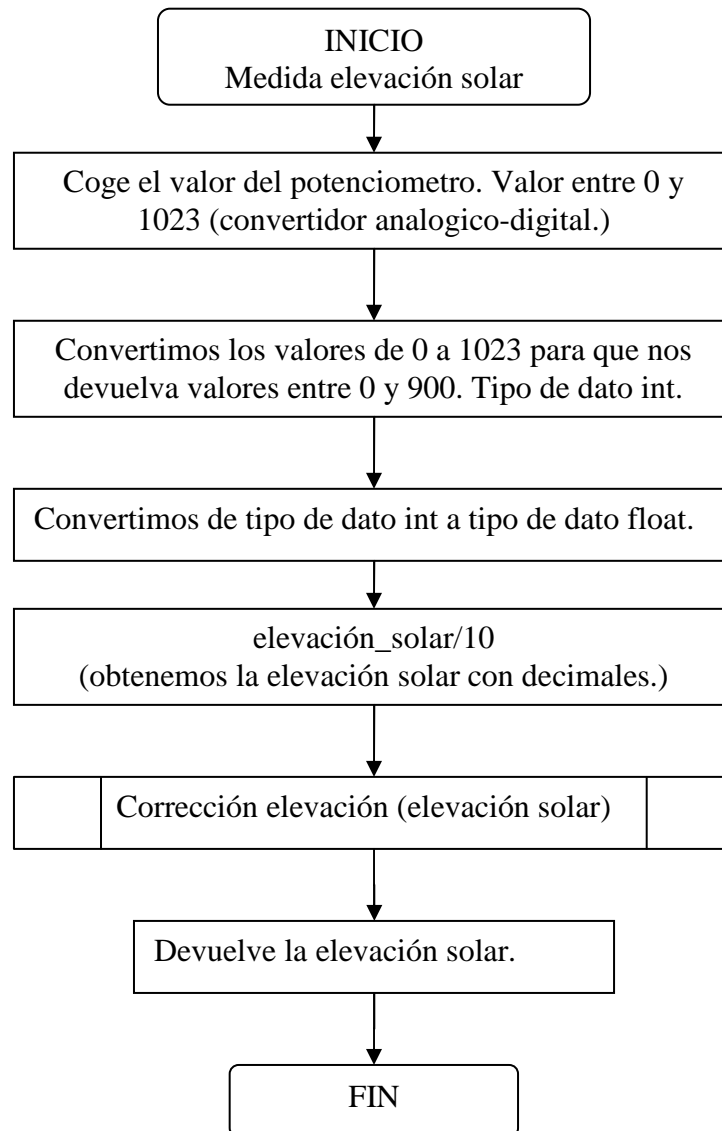




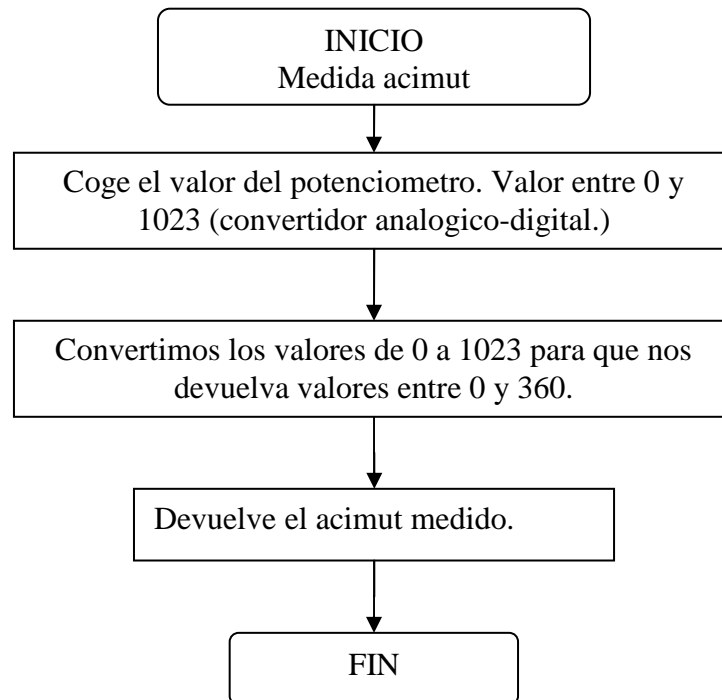
## 6.9 SUBPROGRAMA CÁLCULO LONGITUD



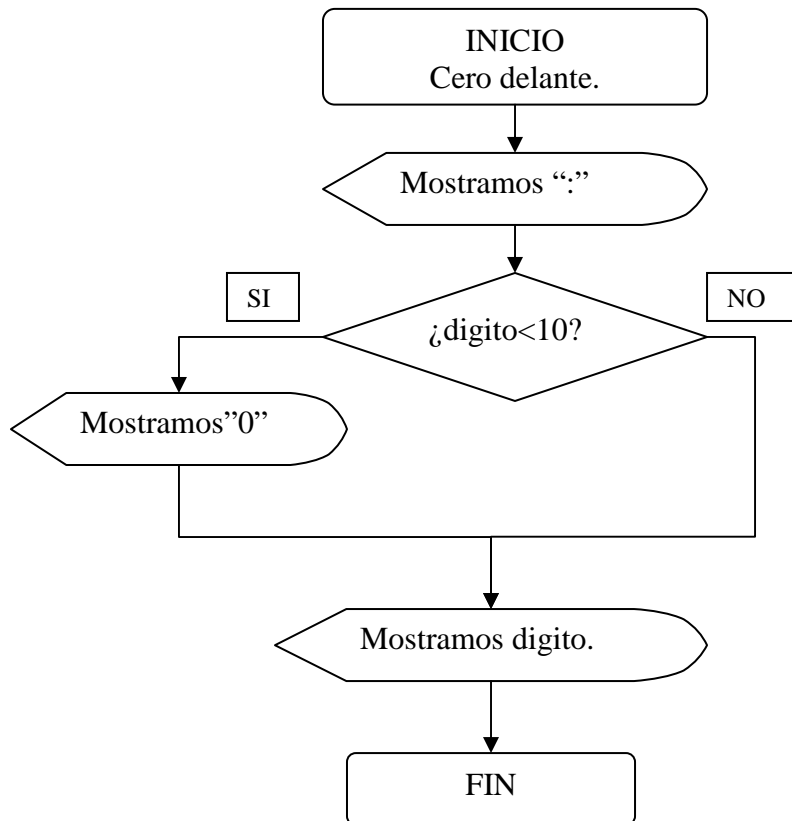
## 6.10 SUBPROGRAMA MEDIDA ELEVACIÓN SOLAR



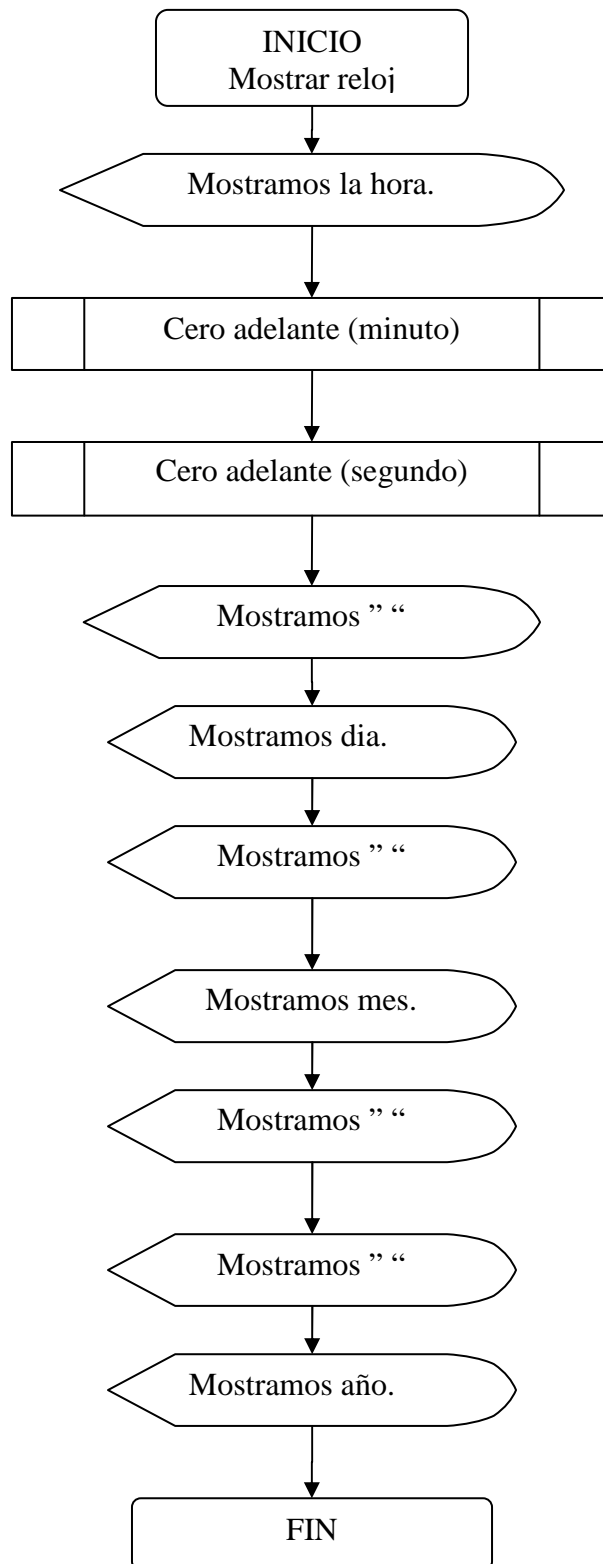
## 6.11 SUBPROGRAMA MEDIDA ACIMUT



## 6.12 SUBPROGRAMA CERO DELANTE.



### 6.13 SUBPROGRAMA MOSTRAR RELOJ.





## 7. LIMITACIONES

Deberemos tener en cuenta que el sistema tendrá algunas limitaciones porque para algunas latitudes existen algunas épocas del año en las que el Sol no sale, no supera los  $5^\circ$  de elevación solar o no desciende de los  $5^\circ$  de elevación solar.

Por ejemplo, en el solsticio de invierno, en latitudes superiores a  $62^\circ$  en el hemisferio Norte, el Sol nunca supera los  $5^\circ$  de elevación como se muestra en la figura 5.1 obtenida de la aplicación de sunmotions de la Universidad de Nebraska.

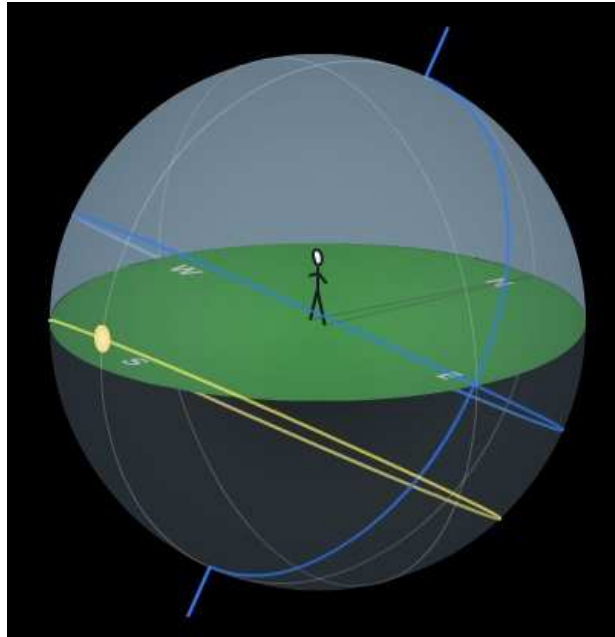


Fig. 5.1: recorrido del Sol durante el día 21 de Diciembre y  $62^\circ$  de latitud Norte.

Y en el hemisferio Sur, para latitudes superiores a  $70^\circ$ , la elevación no será menor de  $5^\circ$ , tal y como se muestra en la fig. 5.2.

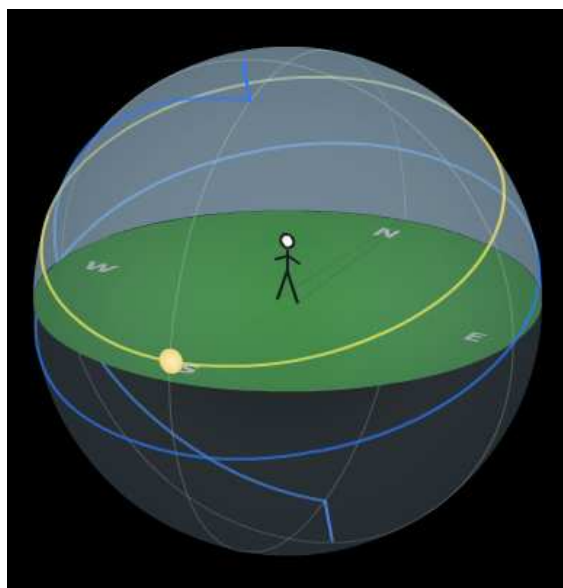


Fig.5.2: recorrido del Sol durante el día 21 de Diciembre y  $70^\circ$  de latitud Sur.

La circunferencia amarilla nos indica el recorrido que tendrá el Sol durante el día 21 de Diciembre. Como no supera o no desciende de los 5° de elevación solar, el sistema desarrollado no podrá calcular la longitud y la latitud. Se producirá el mismo caso durante el solsticio de verano pero a la inversa.

Durante los equinoccios de primavera y otoño se producirá el mismo fenómeno en latitudes muy cercanas a los polos, tal y como podemos observar en la figura 5.2, también obtenida de la aplicación sunmotions de la Universidad de Nebraska. En dicha figura se muestra el recorrido del Sol para una latitud de 85° (línea amarilla). Se observa como la elevación solar no supera los 5°, y por tanto, el sistema desarrollado no podrá calcular la latitud y la longitud.

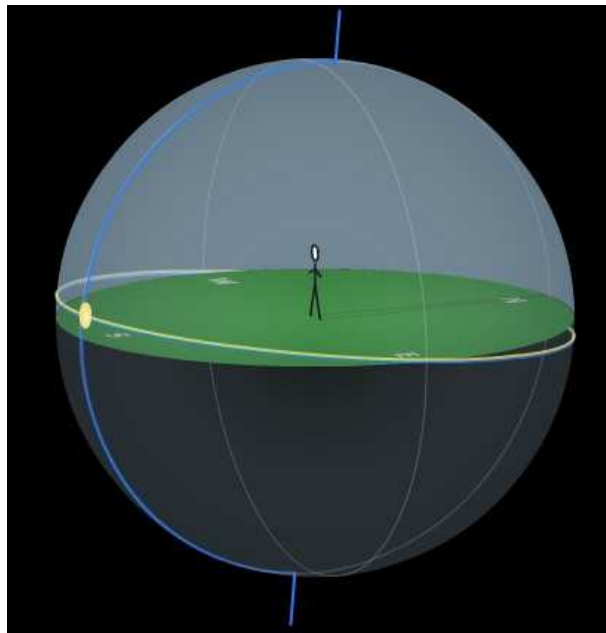


Fig. 5.1: recorrido del Sol durante el día 21 de Marzo y 85° de latitud.

En las fechas comprendidas entre equinoccios y solsticios, el recorrido del Sol variará de unas posiciones a otras. Por tanto, dependiendo de la época del año, existirán posiciones en las que el robot no podría orientarse con el sistema desarrollado.

Debido a lo explicado anteriormente, podemos concluir que el sistema desarrollado sería siempre útil para orientarse en latitudes comprendidas entre 60° Norte y 60° Sur.



## 8. INCERTIDUMBRES

### 8.1 INCERTIDUMBRE DE LA LATITUD

Para calcular la incertidumbre debida a la latitud deberemos tener en cuenta las medidas que realizamos para calcularla. Para ello, utilizaremos la fórmula que obtenemos y explicamos en el apartado 4.1:

$$y = \frac{-2ab \pm \sqrt{4a^2b^2 - 4(1+a^2)(b^2-1)}}{2(1+a^2)}$$

Donde:

$$a = -\frac{\text{sen}\delta}{\cos\delta \cdot \cos\omega} \quad b = \frac{\text{sen}\alpha}{\cos\delta \cdot \cos\omega}$$

Donde  $\delta$  es la declinación,  $\alpha$  es la elevación solar y  $\omega$  es el ángulo horario.

Para realizar este cálculo, solo realizamos una medida, la elevación solar. Hemos estimado que podríamos conseguir una incertidumbre aproximada de  $\pm 0.1^\circ$  debido al sensor, esta incertidumbre podría variar dependiendo del diseño del sensor que se realizaría en otro proyecto. Además, tendremos en cuenta otras incertidumbres debidas a que el cálculo de la declinación y el ángulo horario no son exactos.

El ángulo horario dependerá de las medidas que realicemos para calcular la longitud tal y como está explicado en apartado 3, ya que para calcular el ángulo horario cogemos la diferencia horaria entre la primera medida y la segunda y por tanto, deberemos tener en cuenta que tendremos un error debido al tiempo. El ángulo horario varia  $0,25^\circ$  por minuto y como el reloj de nuestro robot tiene una precisión de  $\pm 1\text{seg.}$ , entonces nuestro error debido al ángulo horario será de aproximadamente de  $\pm 5 \cdot 10^{-3}^\circ$ .

Deberemos tener en cuenta también que se produce un error en la medición del ángulo horario debido a la declinación, tal y como lo hemos explicado en el apartado 3.2. Como el sistema utilizado para nuestra aplicación no se puede utilizar para latitudes cercanas a los polos ( $>60^\circ$ ), suponemos una incertidumbre en la medida del ángulo horario de  $\pm 0,105^\circ$  (calculado en el apartado 3.2.). Que sumados a la incertidumbre debido al tiempo será  $\pm 0,11^\circ$ .

El error que cometemos en la declinación dependerá de cuanto varía la declinación en un día, ya que solo podemos calcular la declinación dependiendo de la fecha. Por tanto, calcularemos cuanto cambia la declinación el día que más varía durante el año, que será en los equinoccios. Calculamos la declinación para el 21 y 22 de Marzo:

$$\delta_{21\text{Marzo}} = 0.33$$

$$\delta_{22\text{Marzo}} = 0.72$$

Por tanto, el error que cometeremos será:

$$\Delta\delta = \frac{0.72 - 0.33}{2} = \pm 0.1^\circ$$

Una vez que sabemos las incertidumbres que vamos a aplicar a nuestras variables y desarrollamos las fórmulas:

$$a = -\frac{\text{sen}(\delta \pm \Delta\delta)}{\cos(\delta \pm \Delta\delta) \cdot \cos(\omega \pm \Delta\omega)} = -\tan(\delta \pm \Delta\delta) \frac{1}{\cos(\omega \pm \Delta\omega)}$$

$$b = \frac{\text{sen}(\alpha \pm \Delta\alpha)}{\cos(\delta \pm \Delta\delta) \cdot \cos(\omega \pm \Delta\omega)}$$

Y aplicamos equivalencias trigonométricas para desarrollar las fórmulas de  $L + \Delta L$ :

$$a = \frac{-\tan \delta - \tan \Delta\delta}{\cos \omega \cdot \cos \Delta\omega + \text{sen} \omega \cdot \text{sen} \Delta\omega - \tan \delta \cdot \tan \Delta\delta \cdot \cos \omega \cdot \cos \Delta\omega + \tan \delta \cdot \tan \Delta\delta \cdot \text{sen} \omega \cdot \text{sen} \Delta\omega}$$

$$b = \frac{\text{sen} \alpha \cdot \cos \Delta\alpha + \cos \alpha \cdot \text{sen} \Delta\alpha}{\cos \delta \cdot \cos \omega \cdot \cos \Delta\delta \cdot \cos \Delta\omega - \cos \delta \cdot \cos \Delta\delta \cdot \text{sen} \omega \cdot \text{sen} \Delta\omega - \text{sen} \delta \cdot \text{sen} \Delta\delta \cdot \cos \omega \cdot \cos \Delta\omega + \text{sen} \omega \cdot \text{sen} \delta \cdot \text{sen} \Delta\omega \cdot \text{sen} \Delta\delta}$$

Calculamos ahora  $a \cdot b$ ,  $a^2$  y  $b^2$ , para luego sustituirlo en la ecuación y calcular la incertidumbre de la latitud:

$$a \cdot b = \frac{-\text{sen} \alpha \cdot \cos \Delta\alpha \cdot \tan \delta - \cos \alpha \cdot \text{sen} \Delta\alpha \cdot \tan \delta - \text{sen} \alpha \cdot \cos \Delta\alpha \cdot \tan \Delta\delta - \cos \alpha \cdot \text{sen} \Delta\alpha \cdot \tan \Delta\delta}{\cos \delta \cdot (\cos \omega)^2 \cdot \cos \Delta\delta \cdot (\cos \Delta\omega)^2 - \text{sen} \delta \cdot \text{sen} \Delta\delta \cdot (\cos \omega)^2 \cdot (\cos \Delta\omega)^2 - \cos \delta \cdot (\cos \omega)^2 \cdot \cos \Delta\delta \cdot (\cos \Delta\omega)^2 \cdot \tan \delta \cdot \tan \Delta\delta + \text{sen} \delta \cdot \text{sen} \Delta\delta \cdot (\cos \omega)^2 \cdot (\cos \Delta\omega)^2 \cdot \tan \delta \cdot \tan \Delta\delta}$$

$$a^2 = \frac{(\tan \delta)^2 + 2 \cdot \tan \delta \cdot \tan \Delta\delta + (\tan \Delta\delta)^2}{(\cos \omega)^2 \cdot (\cos \Delta\omega)^2 - 2 \cdot (\cos \omega)^2 \cdot (\cos \Delta\omega)^2 \cdot \tan \delta \cdot \tan \Delta\delta + (\cos \omega)^2 \cdot (\cos \Delta\omega)^2 \cdot (\tan \delta)^2 \cdot (\tan \Delta\delta)^2}$$

$$b^2 = \frac{(\operatorname{sen} \alpha)^2 \cdot (\cos \Delta \alpha)^2 + 2 \cdot \operatorname{sen} \alpha \cdot \cos \Delta \alpha \cos \alpha \cdot \operatorname{sen} \Delta \alpha + (\cos \alpha)^2 \cdot (\operatorname{sen} \Delta \alpha)^2}{(\cos \delta)^2 \cdot (\cos \omega)^2 \cdot (\cos \Delta \delta)^2 \cdot (\cos \Delta \omega)^2 - 2 \cdot \cos \delta \cdot \operatorname{sen} \delta \cdot \cos \Delta \delta \cdot \operatorname{sen} \Delta \delta \cdot (\cos \omega)^2 \cdot (\cos \Delta \omega)^2 - (\operatorname{sen} \delta)^2 \cdot (\cos \omega)^2 \cdot (\operatorname{sen} \Delta \delta)^2 \cdot (\cos \Delta \omega)^2}$$

Una vez calculados, los sustituimos en la ecuación:

$$y = \frac{-2ab \pm \sqrt{4a^2b^2 - 4(1+a^2)(b^2-1)}}{2(1+a^2)}$$

Y obtenemos los siguientes resultados:

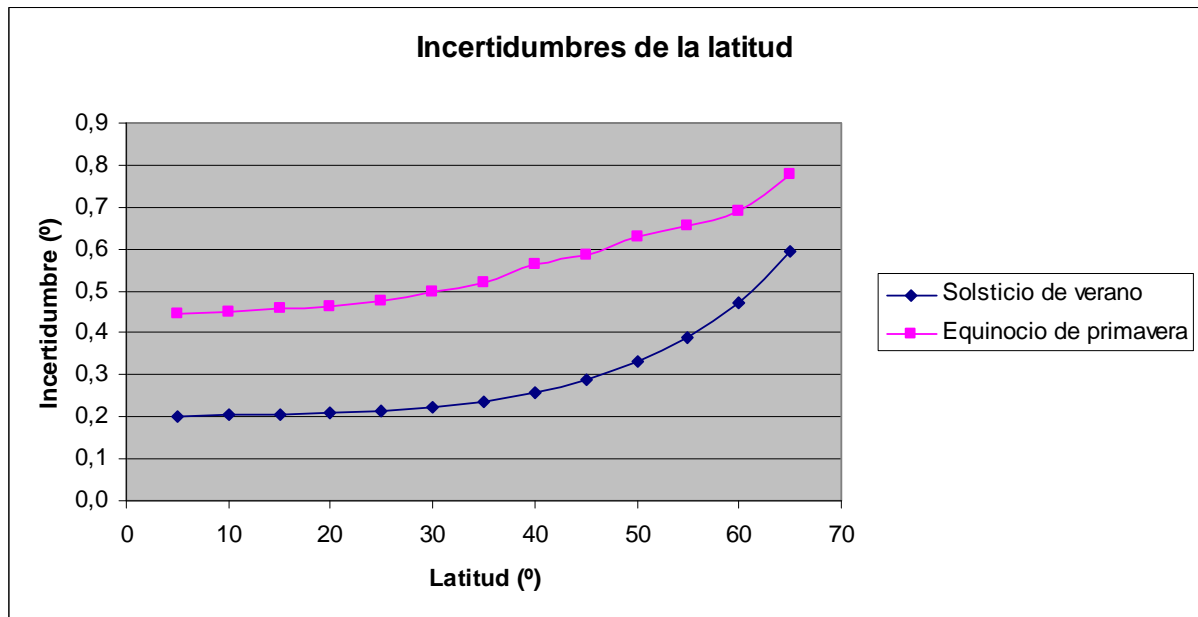
1. Para el día 21 de Marzo ( $\delta = 0,33$ ), cuando la variación de la declinación es máxima:

$\alpha$	$\omega$	Latitud+ $\Delta$ Latitud	Latitud	$\pm\Delta$ Latitud
5,0097	85	5,452	5,007	0,445
4,98109	85	10,456	10,005	0,451
4,91457	85	15,462	15,002	0,460
4,81057	85	20,468	20,003	0,465
5,12278	84,5	25,48	25,004	0,476
4,92627	84,5	30,502	30,005	0,497
5,10124	84	35,525	35,006	0,519
5,18703	83,5	40,569	40,004	0,565
5,17698	83	45,595	45,009	0,586
5,06563	82,5	50,637	50,009	0,628
5,13375	81,5	55,664	55,008	0,656
5,01945	80,5	60,698	60,007	0,691
4,92437	79	65,785	65,006	0,779

2. Para el día 21 de Junio ( $\delta = 23,46$ ), cuando la variación de la declinación es mínima:

$\alpha$	$\omega$	Latitud+ $\Delta$ Latitud	Latitud	$\pm\Delta$ Latitud
5,19115	86,5	5,199	5,000	0,199
4,86891	89	10,205	10,001	0,204
5,0221	91	15,205	15,001	0,204
4,78919	93,5	20,212	20,002	0,210
5,07756	95,5	25,215	25,002	0,213
5,0727	98	30,225	30,002	0,223
4,8692	101	35,241	35,003	0,238
4,92224	104	40,262	40,003	0,259
4,95361	107,5	45,292	45,003	0,289
5,09169	111,5	50,334	50,003	0,331
4,99769	117	55,393	55,003	0,390
5,0579	124	60,475	60,003	0,472
4,96479	135	65,595	65,003	0,592

Representamos los datos anteriores gráficamente y nos queda:



De la gráfica anterior podemos sacar las siguientes conclusiones:

- En los equinoccios la incertidumbre es mayor debido a la mayor variación de la declinación.
- Conforme nos vamos acercando a los polos la incertidumbre es mayor. Esto se produce porque cuanto más nos aproximamos a los polos la elevación máxima es menor, y por tanto, la variación de la elevación en el mismo periodo de tiempo es menor.

## 8.2 INCERTIDUMBRE DE LA LONGITUD

Para realizar el cálculo de la longitud utilizamos la fórmula explicada y obtenida en el apartado 3:

$$Longitud = 0.25 \cdot (720 - ML[\text{min}] - E_t[\text{min}])$$

Para calcular ML medimos la hora en que se producen la misma elevación solar cuando el Sol se eleva y cuando el Sol desciende. Lo que estamos midiendo realmente son dos ángulos horarios: el que se produce con una elevación solar antes del mediodía en el meridiano local y el que se produce con la misma elevación solar después del mediodía. Trás eso calculamos la hora en que se produciría el mediodía solar en el meridiano local pero en la hora de Greenwich, y por lo tanto, el ángulo horario que se produciría para esa hora en concreto. Si relacionamos el ángulo horario con el tiempo obtenemos la siguiente expresión:

$$ML[\text{min}] = \frac{\omega[^\circ]}{0.25}$$

Al realizar la medida, el ángulo horario tendría una incertidumbre debida al error de la declinación y al debido a la elevación solar.

Sabemos que:

$$\omega = \cos^{-1}(\text{sen}\alpha - (\tan \delta \cdot \tan \Phi))$$

Por tanto, si lo sustituimos en la ecuación de la longitud nos queda:

$$\text{Longitud} = 0.25 \cdot \left( 720 - \frac{\cos^{-1}(\text{sen}\alpha - (\tan \delta \cdot \tan \Phi))}{0.25} - E_t[\text{min}] \right)$$

Donde  $\alpha$  es la elevación solar,  $\delta$  es la declinación,  $\Phi$  es la latitud y  $E_t$  es la ecuación de tiempo.

Cada una de las variables tendrá una incertidumbre: la elevación solar dependerá del diseño del sensor, que estimamos que se podría conseguir una incertidumbre aproximada de  $\pm 0.1^\circ$ .

La incertidumbre de la declinación dependerá de cuanto varíe la declinación de un día para otro, y la máxima variación de la declinación se produce en los equinoccios, es decir, entre los días 21 y 22 de Marzo. Tal y como hemos calculado en el apartado anterior el error de la declinación será de  $\pm 0.1^\circ$ .

La incertidumbre de la latitud estimaremos que será la calculada en el apartado anterior:  $\pm 0.7^\circ$

La ecuación de tiempo es una variable que varía continuamente en el tiempo al igual que la declinación, por esa razón, calcularemos el día que más varía de un día para otro, ya que nosotros solo podemos calcular el valor de la ecuación de tiempo para un día determinado, y supondremos esa incertidumbre. Los días que más varía la ecuación de tiempo es entre el 20 y 21 de Septiembre:

$$E_{t20Sept} = 6,89$$

$$E_{t21Sept} = 7,26$$

Entonces, la incertidumbre de la ecuación de tiempo será:

$$\Delta E_t = \frac{7.26 - 6.89}{2} = \pm 0.19 \text{ min}$$

Además, deberemos tener en cuenta la incertidumbre del reloj a la hora de calcular la hora en la que se produce el mediodía solar que es de un segundo, es decir, una incertidumbre de  $\pm 0.017 \text{ min. } (\Delta t)$ .

Por tanto, la ecuación nos queda de la siguiente manera:

$$L \pm \Delta L = 0.25 \cdot \left( 720 - \left( \left( \frac{\cos^{-1}(\sin(\alpha \pm \Delta\alpha) - \tan(\delta \pm \Delta\delta) \cdot \tan(\Phi \pm \Delta\Phi))}{0.25} \right) \pm \Delta t \right) - (E_t \pm \Delta E_t) \right)$$

Desarrollamos y nos queda lo siguiente:

$$L + \Delta L = 0.25 \cdot \left( 720 - \left( \frac{\cos^{-1}((\sin\alpha \cdot \cos\Delta\alpha + \cos\alpha \cdot \sin\Delta\alpha) - (a))}{0.25} \right) - \Delta t - E_t - \Delta E_t \right)$$

$$a = \left( \frac{(\tan\delta \cdot \tan\Phi) + (\tan\delta \cdot \tan\Delta\Phi + (\tan\Delta\delta \cdot \tan\Phi) + (\tan\Delta\delta \cdot \tan\Delta\Phi))}{1 - (\tan\Phi \cdot \tan\Delta\Phi) - (\tan\delta \cdot \tan\Delta\delta) + (\tan\delta \cdot \tan\Delta\delta \cdot \tan\Phi \cdot \tan\Delta\Phi)} \right)$$

Y obtenemos los siguientes resultados para nuestras incertidumbres:

1. Para el día 21 de Marzo ( $\delta = 0,33$ ), cuando la variación de la declinación es máxima:

$$- \alpha = 5^\circ, E_t = -7,55^\circ$$

LATITUD	Longitud	Longitud+Error	$\Delta$ Longitud	Ang horario	Hora	Hora Greenwich
0	86,75	86,89	0,14	85,00	17,67	17:40
5	86,72	86,92	0,20	85,03	17,67	17:40
10	86,69	86,95	0,26	85,06	17,67	17:40
15	86,66	86,98	0,32	85,09	17,67	17:40
20	86,63	87,01	0,38	85,12	17,67	17:40
25	86,60	87,04	0,45	85,15	17,68	17:40
30	86,56	87,08	0,52	85,19	17,68	17:40
35	86,52	87,12	0,60	85,23	17,68	17:40
40	86,48	87,17	0,69	85,28	17,69	17:41
45	86,42	87,22	0,79	85,33	17,69	17:41
50	86,36	87,28	0,92	85,39	17,69	17:41
55	86,29	87,36	1,07	85,47	17,70	17:41
60	86,19	87,46	1,27	85,57	17,70	17:42
65	86,06	87,60	1,54	85,71	17,71	17:42

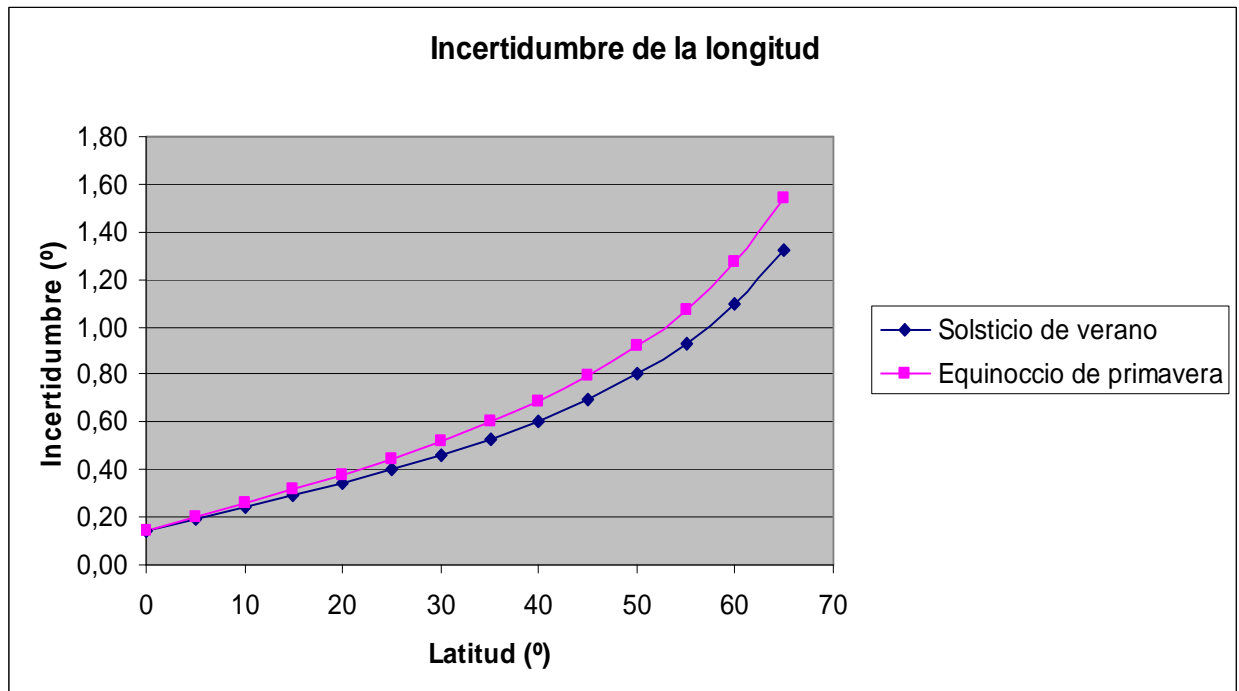
2. Para el día 21 de Junio ( $\delta = 23,46$ ), cuando la variación de la declinación es mínima:

$$-\alpha = 5^\circ, E_t = -1.54^\circ$$

LATITUD	longitud	Longitud+Error	$\Delta$ Longitud	ang horario	hora	Hora Greenwich
0	85,51	85,65	0,14	85,00	17,67	17:40
5	87,62	87,81	0,19	87,18	17,81	17:48
10	89,77	90,00	0,24	89,39	17,96	17:57
15	91,97	92,26	0,29	91,67	18,11	18:06
20	94,29	94,63	0,34	94,06	18,27	18:16
25	96,77	97,17	0,40	96,61	18,44	18:26
30	99,47	99,93	0,46	99,40	18,63	18:37
35	102,49	103,01	0,53	102,51	18,83	18:50
40	105,93	106,54	0,60	106,07	19,07	19:04
45	110,01	110,70	0,69	110,28	19,35	19:21
50	115,00	115,80	0,80	115,45	19,70	19:41
55	121,45	122,38	0,93	122,16	20,14	20:08
60	130,48	131,58	1,10	131,62	20,77	20:46
65	145,26	146,59	1,33	147,47	21,83	21:49

Para realizar los cálculos hemos introducido datos al azar de manera que la última columna nos indica la hora de Greenwich a la que se producirá el mediodía solar local, es decir, la hora de Greenwich a la que se produce el mediodía en el punto en el que se encuentra el robot.

Representando los resultados obtenidos gráficamente:



De la gráfica anterior podemos sacar las siguientes conclusiones:

- En los equinoccios el error es mayor debido a la mayor variación de la declinación.
- Conforme nos vamos acercando a los polos la incertidumbre es mayor. Esto se produce porque cuanto más nos aproximamos a los polos la elevación máxima es menor, y por tanto, la variación de la elevación en el mismo periodo de tiempo es menor.

### 8.3 INCERTIDUMBRES A KILOMETROS

Para ver más claramente el error que estamos cometiendo pasamos los datos de incertidumbres a kilómetros.

La Tierra tiene un radio medio de 6371 Km. (dato obtenido de Wikipedia), entonces calculamos el perímetro y los kilómetros por grado de latitud:

$$P = 2\pi r = 40030,2km \qquad \frac{40030,2}{360} = 111,19 \frac{km}{^\circ}$$

Multiplicamos los kilómetros por grado y la incertidumbre en cada latitud, obtenemos:

1. Para el día 21 de Marzo ( $\delta = 0,33$ ), cuando la variación de la declinación es máxima:

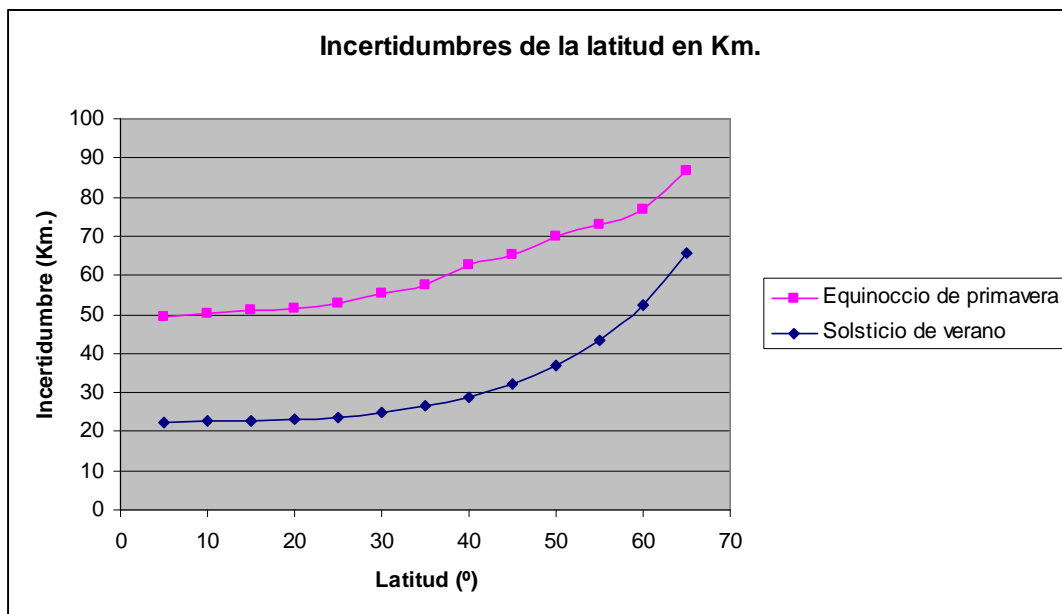
Latitud	$\Delta$ latitud[°]	$\Delta$ latitud[km]
5	0,445	49,48
10	0,451	50,15
15	0,460	51,15
20	0,465	51,70
25	0,476	52,93
30	0,497	55,26
35	0,519	57,71
40	0,565	62,82
45	0,586	65,16
50	0,628	69,83
55	0,656	72,94
60	0,691	76,83
65	0,779	86,62



2. Para el día 21 de Junio ( $\delta = 23,46$ ), cuando la variación de la declinación es mínima:

Latitud	$\Delta$ latitud[°]	$\Delta$ latitud[km]
5	0,199	22,13
10	0,204	22,68
15	0,204	22,68
20	0,210	23,35
25	0,213	23,68
30	0,223	24,80
35	0,238	26,46
40	0,259	28,80
45	0,289	32,13
50	0,331	36,80
55	0,390	43,36
60	0,472	52,48
65	0,592	65,82

Representamos los resultados obtenidos gráficamente:



Para calcular cuantos kilómetros tenemos de incertidumbre en la longitud, debemos tener en cuenta que el perímetro del paralelo la varía dependiendo de la latitud, cuanto mayor sea la latitud menor será el su perímetro.

Para calcular la longitud del paralelo para cada latitud necesitamos el radio, para ello calculamos el coseno de la latitud que corresponderá con el radio, tal y como se muestra en la figura 6.1:

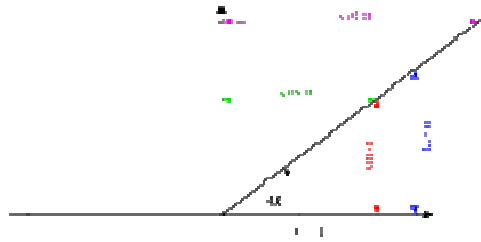


Fig. 6.1: Cálculo del radio para cada latitud.

En la fig. 6.1  $\alpha$  será la latitud y  $r$  será el radio ecuatorial de la Tierra. El radio ecuatorial de la Tierra es 6378,1km. (dato obtenido de Wikipedia). Una vez calculado el radio calcularíamos el perímetro y lo dividiríamos entre 360 para obtener cuantos kilómetros variaría por grado. Obtenemos la siguiente expresión:

$$P = \frac{2\pi r \cdot \cos \alpha}{360}$$

Si lo aplicamos a las incertidumbres calculadas anteriormente nos queda:

1. Para el día 21 de Marzo ( $\delta = 0,33$ ), cuando la variación de la declinación es máxima:

$$-\alpha = 5^\circ, E_t = -7,55^\circ$$

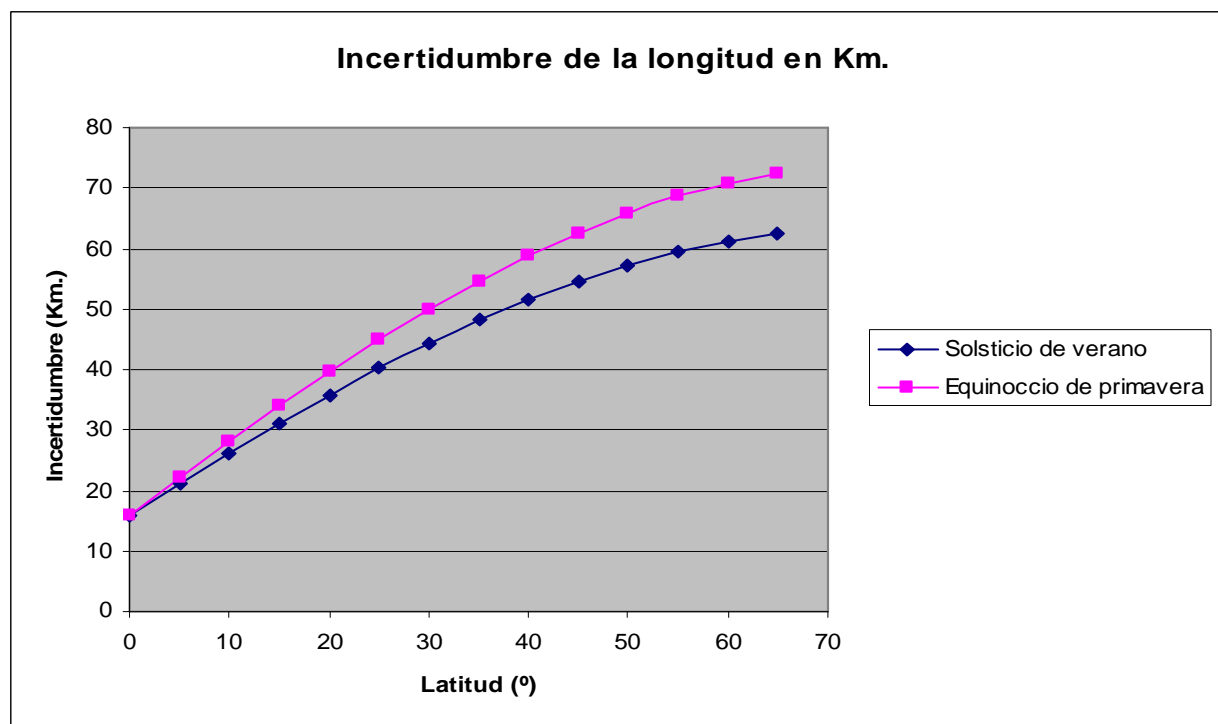
LATITUD	$\Delta$ longitud[°]	$\Delta$ longitud[km]
0	0,14	15,85
5	0,20	22,13
10	0,26	28,23
15	0,32	34,12
20	0,38	39,74
25	0,45	45,07
30	0,52	50,05
35	0,60	54,65
40	0,69	58,84
45	0,79	62,58
50	0,92	65,84
55	1,07	68,60
60	1,27	70,84
65	1,54	72,54

2. Para el día 21 de Junio ( $\delta = 23,46$ ), cuando la variación de la declinación es mínima:

$$-\alpha = 5^\circ, E_t = -1,54^\circ$$

LATITUD	$\Delta$ longitud[°]	$\Delta$ longitud[km]
0	0,14	15,77
5	0,19	21,07
10	0,24	26,21
15	0,29	31,14
20	0,34	35,84
25	0,40	40,27
30	0,46	44,39
35	0,53	48,17
40	0,60	51,59
45	0,69	54,61
50	0,80	57,22
55	0,93	59,39
60	1,10	61,11
65	1,33	62,36

Representando los resultados obtenidos gráficamente:



Con las gráficas anteriores llegamos a las mismas conclusiones que en las descritas en los dos apartados precedentes. Son los mismos datos pero en con las incertidumbres obtenidas en kilómetros.

La conclusión principal que podemos sacar de las incertidumbres en kilómetros es que para latitudes inferiores a  $60^\circ$  podemos asegurar que el robot se encontraría en un punto dentro de un cuadrado de 154km de alto por 146km de ancho, como se muestra en la figura 6.2:

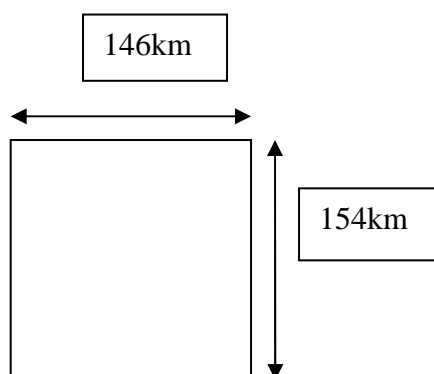


Fig. 6.2: Cuadrado de donde se encontraría el robot.

## 9. COMENTARIOS FINALES

### 9.1 PROGRAMA ROBOT

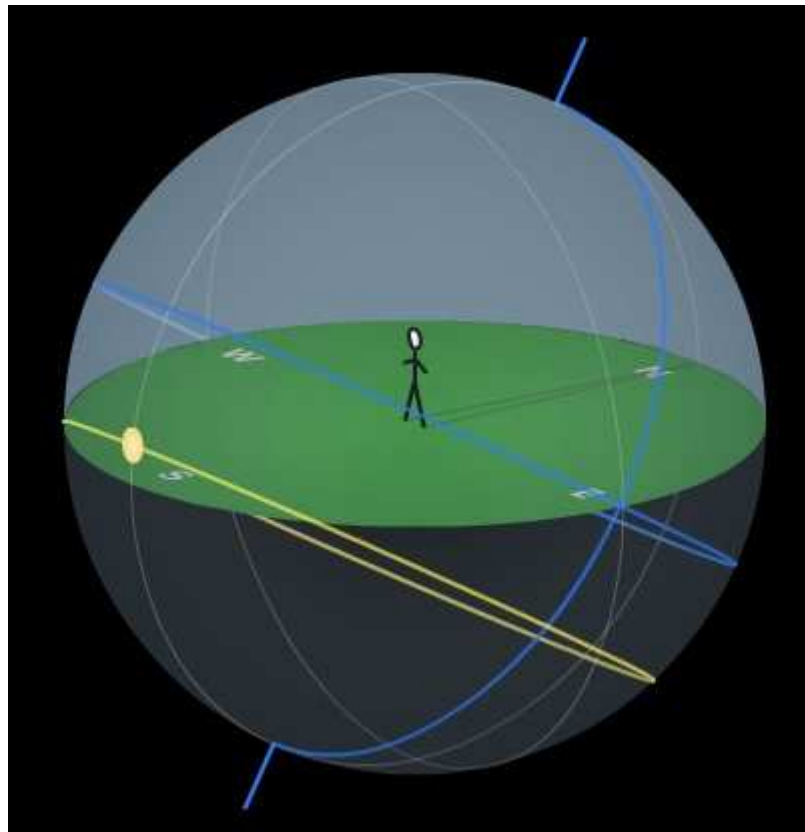
Debido a que el fin de este proyecto es que sea utilizado por otros para introducirlo en un robot, hemos adaptado el programa de manera que se pueda introducir directamente como un subprograma.

Deberemos tener en cuenta que habría que completar los subprogramas de los sensores dependiendo de cómo se diseñen éstos. Para ello, hemos indicado donde insertar los subprogramas mediante comentarios de manera que se varíe el programa lo menos posible.

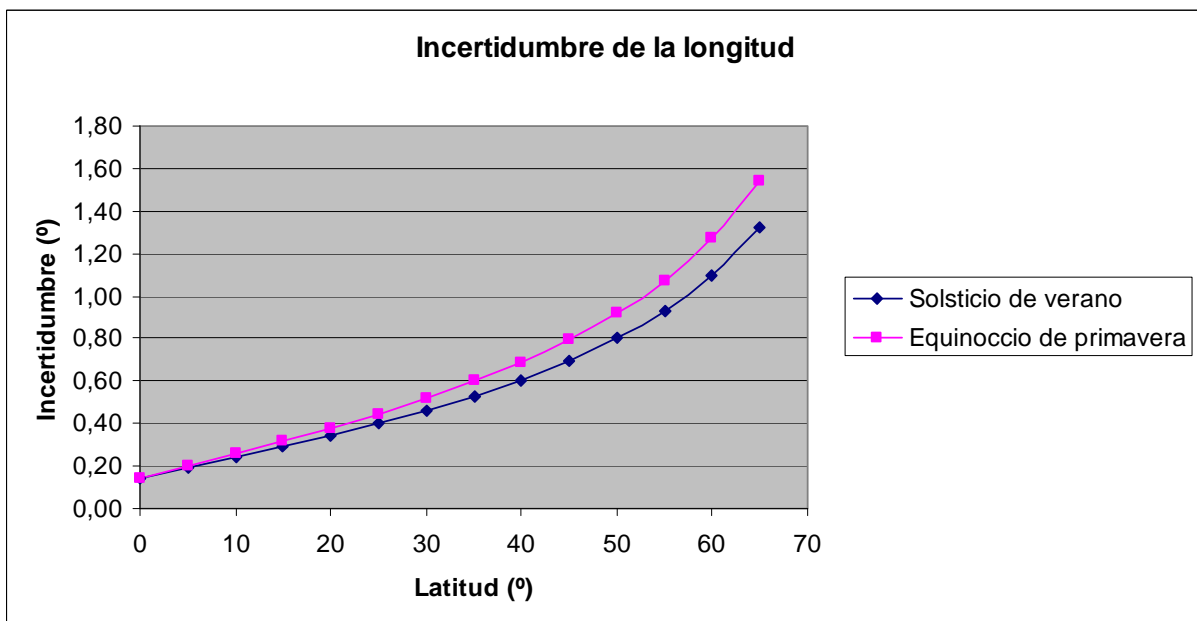
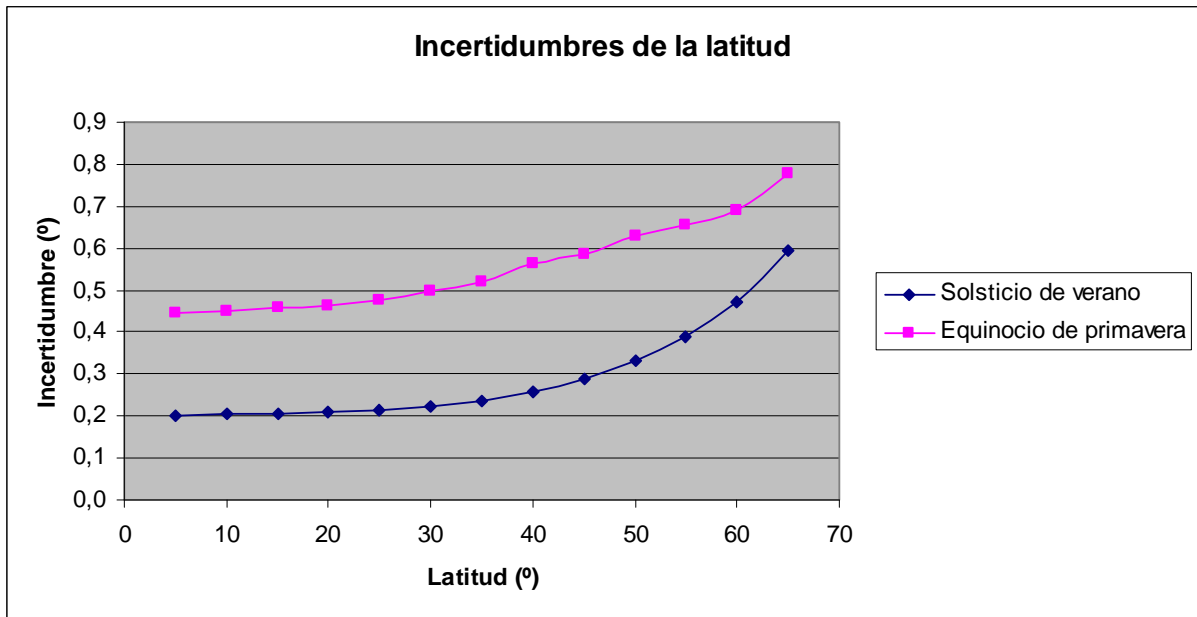
### 9.2 CONCLUSIONES

El sistema de orientación desarrollado será capaz de indicarnos la latitud y la longitud para latitudes comprendidas entre  $60^\circ$  Norte y  $60^\circ$  Sur, que es la mayor parte de la superficie terrestre.

Esto se debe a que durante el solsticio de invierno la elevación solar en el hemisferio Norte no supera los  $5^\circ$ , tal y como se muestra en la siguiente figura. Sucederá lo mismo en el hemisferio Sur durante el solsticio de verano.



Además, las incertidumbres se hacen más grandes conforme se acercan a los polos, tal y como muestran las siguientes gráficas:



Lo que hace que el sistema desarrollado será muy útil para latitudes cercanas al ecuador, donde no superan 0,5° de incertidumbre para latitudes menores de 30°. En cambio, en latitudes superiores a 50°, las incertidumbres comienzan a ser muy grandes, lo que limita mucho su aplicación.

### 9.3 MEJORAS FUTURAS

Para mejorar nuestro proyecto podríamos desarrollar un sistema de orientación basado en la posición de las estrellas. De esta manera nuestro robot podría orientarse en latitudes superiores a 60° y durante la noche.

Otra mejora podría ser incluir sensores de temperatura y de presión, con lo que mejoraríamos la precisión en la medida de la elevación solar para grandes altitudes y temperaturas muy bajas.





## 10. BIBLIOGRAFÍA

- ASTRONOMÍA ESFÉRICA Y MECÁNICA CELESTE. Juan José de Orus Navarro, María Asunción Catalá Poch, Jorge Núñez de Murga. Ed. Universidad de Barcelona.

- A SHORT GUIDE OF CELESTIAL NAVIGATION. Henning Umland. 1997.

- MOVIMIENTOS DE LA TIERRA (MOVIMIENTO APARENTE DEL SOL, DETERMINACIÓN DE LA HORA Y DE LAS COORDENADAS GEOGRÁFICAS). Alfonso Calera Belmonte y Antonio J. Barbero. Ed. UCLM

- ARDUINO COOKBOOK. Michael Margolis Ed. O'REILLY.

[1] <http://www.Arduino.cc> (Entorno Arduino)

[2] <http://astro.unl.edu/classaction/animations/coordsmotion/sunmotions.html>

[3] <http://www.wikipedia.org/> (Enciclopedia electrónica)



# ANEXO A: LISTA DE PROGRAMAS

## PROGRAMAS TEST

### 1. Programa test reloj Arduino.

```
#include <Time.h>

void setup()
{
    Serial.begin(9600);
    setTime(12,52,0,12,1,12); /* para poner la hora de inicio del
    programa.*/
}
void loop()
{
    digitalClockDisplay();//llamamos al subprograma que muestra el
    //reloj.
    delay(1000);
}
void digitalClockDisplay()
//subprograma que muestra el reloj para Arduino
{
    Serial.print(hour());//Muestra las horas en pantalla.
    printDigits_min(minute());//Muestra los minutos en pantalla.
    printDigits_seg(second());//Muestra los segundos en pantalla.
    Serial.print(" "); //Imprime un espacio en pantalla.
    Serial.print(day());//Muestra el día en pantalla.
    Serial.print(" ");
    Serial.print(month());//Muestra el mes en pantalla.
    Serial.print(" ");
    Serial.print(year());//Muestra el año en pantalla.
    Serial.println();
}
void printDigits(int digito)
//Esta función pone un 0 delante de el segundo o del minuto si es //menor
de 10.
{
    Serial.print(":");
    if(digito < 10) Serial.print('0'); //Si es menor de 10 imprime //en
pantalla un 0 antes del digito.
    Serial.print(digito); //Imprime el digito en cuestión.
}
}
```

### 2. Programa test calculo día del año.

```
int fecha_num (int dia, int mes, int ano)

void setup()
{
    Serial.begin(9600); //Inicia la conexión serie entre Arduino y //mi
PC.
}

void loop()
{
    int dia_num, dia, mes, ano;
```

```

    dia=2;
    mes=3;
    ano=2012;
    dia_num=fecha_num (dia, mes, ano);//llamamos al subprograma que
    //calcula el numero de día del año y introducimos el valor en
    //dia_num.
    Serial.print(dia_num);//muestra en pantalla el número de día del
    //año.
    delay(1000);
}

/*Subprograma que calcula el dia del año en que nos encontramos*/
int fecha_num (int dia, int mes, int ano)
{
    int dias_mes[12]={31,28,31,30,31,30,31,31,30,31,30,31};// array //con
    los dias que tiene cada mes del año.
    int num_dia,cont;
    if(ano%4==0||ano%100==0||ano%400==0) dias_mes[1]=29; //sustituye //el
    valor de los dias del mes de febrero cuando el año es //bisisesto.
    num_dia=0;
    for(cont=0;cont<(mes-1);cont++)//Bucle que va sumando los dias //de
    los meses que han pasado.
    {
        num_dia=num_dia+dias_mes[cont];
    }
    num_dia=num_dia+dia;//sumamos los dias que han pasado del mes.
    return (num_dia); //devuelve el valor al programa principal.
}

```

### 3. Programa test calculo declinación.

```

#include <math.h>//Nombre librería que utilizamos.
#define pi 3.14159 //define el nombre y el valor de una constante.

int dia_num=0; //esta variable la podremos utilizar en cualquier parte
//del programa, tanto programa principal, como subprogramas.

/*Subprogramas que utilizamos.*/
float calc_decli_spencer (int dia_num);

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    float declinacion;
    if (dia_num <365)//pequeño programa para probar todos los días //del
    año. Entra aquí si dia_num es menor de 365.
    {
        dia_num=dia_num+1; //suma al día del año un día mas.
    }
    else//Si día_num es 365 devuelve el valor de dia_num a 1.
    {
        dia_num=1;//Devuelve el contador a uno.
    }
    declinacion=calc_decli_spencer (dia_num); //Llamamos al //subprograma
    que calcula la declinación.
    Serial.println("Dia numero: ");//Muestra el texto que esta entre
    //las comillas.
}

```

```

Serial.println(dia_num); //muestra en pantalla la variable que //esta
entre paréntesis.
Serial.println("Declinacion: ");
Serial.print(declinacion);
delay(1000); //Retrasa la siguiente acción 1 segundo.
}
/*Subprograma que calcula la declinación*/
float calc_decli_spencer (int dia_num)
{
    double declinacion, tao;
    tao=((dia_num-1)*2*pi)/365;
    declinacion=((0.006918-(0.399912*cos(tao)))+(0.070257*sin(tao))-
    (0.006758*cos(2*tao))+(0.000907*sin(2*tao))-
    (0.002697*cos(3*tao))+(0.00148*sin(3*tao)))*180)/pi;
    /*Formula utilizada y explicada en la memoria para el calculo de la
    declinación.*/
    return(declinacion); // Devuelve el valor de la declinación.
}

```

#### 4. Programa test corrección elevación solar.

```

#include <math.h>
#define pi 3.14159

float correccion_elevacion(float elevacion_solar);

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    float elevacion_solar, elevacion_solar2;
    for(elevacion_solar=0;elevacion_solar < 90;
    elevacion_solar++)//Bucle que barre todos los posibles valores //de
    la elevación solar.
    {
        elevacion_solar2=correccion_elevacion(elevacion_solar);
        //Llama al subprograma de corrección.
        Serial.println(elevacion_solar2);//Muestra la elevacion_solar
        corregida.
        delay(1000);
    }
}

/*Subprograma que corrige el error de la elevación solar.*/
float correccion_elevacion(float elevacion_solar)
{
    float refraccion,elev_solar,po;
    float semidiametro_solar=0.256; //Le da el valor error al
    //semidiametro solar.
    if (elevacion_solar <= 15) //Si la elevación solar es menor o igual a
    15 realizara las siguientes acciones.
    {
        refraccion=((34.133+(4.197*elevacion_solar)+(0.00428*sq(elevacio
        n_solar)))/(1+(0.505*elevacion_solar)+(0.0845*sq(elevacion_solar
        ))))*(0.016);
    }//Calcula el error debido a la refraccion para elevaciones solares
    menores de 15 grados.
}

```

```

else
{
    elev_solar=(2*pi*elevacion_solar)/360;
    refraccion=((0.97127/tan(elev_solar))-
    (0.00137/pow(tan(elev_solar),3)))*(0.016);
} //Calcula el error debido a la refraccion para elevaciones solares
mayo
elevacion_solar=elevacion_solar-refraccion+semidiametro_solar;
//Correccion de la refracción y del semidiametro solar.
return(elevacion_solar); //Devuelve el valor de la correccion.
}

```

## 5. Programa test calculo acimut teórico.

```

#include<math.h>
#define pi 3.141593

double calc_acimut_teor (float elevacion_solar,float declinacion, double
latitud);

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    double acimut_calc1, acimut_calc2;
    float elevacion_solar, declinacion;
    double latitud1, latitud2;
    elevacion_solar=28.36;
    declinacion=-14.19;
    latitud1=41.66;
    latitud2=-74.16;
    acimut_calc1= calc_acimut_teor (elevacion_solar, declinacion,
latitud1); //Llamamos al subprograma para que calcule el acimut //con
la latitud1.
    acimut_calc2= calc_acimut_teor (elevacion_solar, declinacion,
latitud2); //Lo mismo que la accion anterior pero ahora con la
//latitud2.
    Serial.print("El acimut calculado se encuentra:");
    Serial.println(acimut_calc1); //Se muestran los acimutes
    Serial.println(acimut_calc2); //calculados.
}

double calc_acimut_teor (float elevacion_solar,float declinacion, double
latitud)
{
    float declinacion_rad, eleva_sol_rad, latitud_rad;
    double acimut;
    declinacion_rad = (2*pi*declinacion)/360; //Pasamos los datos a
//radianes ya que las operaciones trigonometricas las calcula //con
radianes.
    eleva_sol_rad =(2*pi*elevacion_solar)/360;
    latitud_rad = (2*pi*latitud)/360;
    acimut=(sin(eleva_sol_rad)*sin(latitud_rad)-
sin(declinacion_rad))/(cos(eleva_sol_rad)*cos(latitud_rad));
    acimut=acos(acimut); //calculamos el acimut
    acimut=(360*acimut)/(2*pi); //lo pasamos a grados.
}

```

```

    return(acimut); //devolvemos el valor al programa principal.
}

```

## 6. Programa test elección latitud

```

#include<math.h>
#define pi 3.141593

double calc_acimut_teor (float elevacion_solar,float declinacion, double
latitud);

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    double acimut_calc1, acimut_calc2;
    float elevacion_solar, declinacion,acimut_med;
    double latitud1, latitud2;
    elevacion_solar=28.36;
    declinacion=-14.19;
    atitud1=41.66;
    latitud2=-74.16;
    acimut_calc1= calc_acimut_teor (elevacion_solar, declinacion,
latitud1);
    acimut_calc2= calc_acimut_teor (elevacion_solar, declinacion,
latitud2);
    acimut_med = 31.45;//Valor medido por el potenciómetro del //ángulo
acimutal.
    if (acimut_calc1 - 0.5 <= acimut_med && acimut_med <
acimut_calc1+0.5)
    {
        Serial.print("La latiud es:");
        Serial.println(latitud1);
    }
    else
    {
        Serial.print("La latiud es:");
        Serial.println(latitud2);
    }
}

double calc_acimut_teor (float elevacion_solar,float declinacion, double
latitud)
{
    float declinacion_rad, eleva_sol_rad, latitud_rad;
    double acimut;
    declinacion_rad = (2*pi*declinacion)/360; //Pasamos los datos a
//radianes ya que las operaciones trigonometricas las calcula //con
radianes.
    eleva_sol_rad =(2*pi*elevacion_solar)/360;
    latitud_rad = (2*pi*latitud)/360;
    acimut=(sin(eleva_sol_rad)*sin(latitud_rad)-
sin(declinacion_rad))/(cos(eleva_sol_rad)*cos(latitud_rad));
    acimut=acos(acimut); //calculamos el acimut
    acimut=(360*acimut)/(2*pi); //lo pasamos a grados.
    return(acimut); //devolvemos el valor al programa principal.
}

```

## 7. Programa test calculo latitud.

```
#include <math.h>
#define pi 3.1415934

float calc_latitud(float dif_hora, int dia, int mes, int ano, float
elevacion_solar); //subprograma que calcula la latitud.
float calc_decli_spencer (int dia_num); //subprograma calculo //declinación
float calc_acimut_teor (float elevacion_solar, float declinacion, float
latitud); //Subprograma que calcula el acimut teórico.
int fecha_num (int dia, int mes, int ano); //subprograma que calcula //el
número de día del año.

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    float latitud;
    float elevacion_solar;
    float dif_hora; //nos indica el intervalo de tiempo entre la
//primera medida y la segunda medida. Lo utilizamos para //calcular
el ángulo horario.
    int dia, mes, ano;
    dif_hora=220;
    dia=1;
    mes=11;
    ano=2011;
    elevacion_solar=28.36;
    latitud=calc_latitud(dif_hora, dia, mes, ano, elevacion_solar);
    //Llama al subprograma y introduce el valor devuelto en la //variable
latitud.
    Serial.println("latitud devuelta");
    Serial.println(latitud);
    delay (1000);
}

/*Subprogrma utilizado para calcular la latitud*/

float calc_latitud(float dif_hora, int dia, int mes, int ano, float
elevacion_solar)
{
    int dia_num;
    float angulo_horario, decli, declinacion, elev_solar;
    float a, b, inver1, inver2, latitud1, latitud2;
    float acimut_calc1, acimut_calc2, acimut_med;
    dia_num=fecha_num (dia, mes, ano); //Calculamos el dia del año.
    declinacion=calc_decli_spencer (dia_num); //Calculamos la
//declinacion para el dia del año indicado.
    angulo_horario=(dif_hora/2)*0.25; //Calculamos el angulo horario.
    angulo_horario=(2*pi*angulo_horario)/360; //Pasamos los datos a
//radianes porque las operaciones trigonométricas las realizan //en
radianes.
    decli=(2*pi*declinacion)/360; //A radianes.
    elev_solar=(2*pi*elevacion_solar)/360; //A radianes.
    a=-((sin (decli)/(cos (decli)*cos(angulo_horario)));
    b=sin (elev_solar)/ (cos (decli)*cos(angulo_horario));
    inver1=((-2*a*b)+sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-
1)))/(2*(1+sq(a)));
```



```

    inver2=((-2*a*b)-sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-
    1)))/(2*(1+sq(a)));
    latitud1=asin(inver1);//calculamos la latitud del hemisferio //norte.
    latitud1=(360*latitud1)/(2*pi);//Volvemos a pasar los radianes a
    grados.
    Serial.println("latitudes calculadas");
    Serial.println(latitud1); //Nos muestra la latitud del //hemisferio
    Norte.
    latitud2=asin(inver2);//Hemisferio sur.
    latitud2=(360*latitud2)/(2*pi);//Volvemos a pasar los radianes a
    grados.
    Serial.println(latitud2);
    acimut_calc1= calc_acimut_teor (elevacion_solar, declinacion,
    latitud1); //Calcula el acimut teórico para la latitud del hemisferio
    Norte.
    acimut_calc2= calc_acimut_teor (elevacion_solar, declinacion,
    latitud2); //Calcula el acimut teórico para la latitud del hemisferio
    Sur.
    acimut_med = 31.45;//Valor medido por el potenciomtre del angulo
    acimutal.
    if (acimut_calc1-1 <= acimut_med && acimut_med < acimut_calc1+1)
    {
        return(latitud1); //Compara el acimut teorico con el medido
    } //y de esta manera elegiremos que latitud
    else // es en la que se encuentra nuestro
    { //reloj realmente y lo devuelven al
        return (latitud2);//programa principal.
    }
}

/*Subprograma que calcula la declinacion dependiendo el dia del año.*/
float calc_decli_spencer (int dia_num)
{
    double declinacion, tao;
    tao=((dia_num-1)*2*pi)/365;
    declinacion=((0.006918-(0.399912*cos(tao))+(0.070257*sin(tao))-
    (0.006758*cos(2*tao))+(0.000907*sin(2*tao))-
    (0.002697*cos(3*tao))+(0.00148*sin(3*tao)))*180)/pi;
    /*Formula utilizada y explicada en la memoria para el calculo de la
    declinación.*/
    return(declinacion); // Devuelve el valor de la declinación.
}

/*subprograma que calcula el acimut teórico.*/

double calc_acimut_teor (float elevacion_solar,float declinacion, double
latitud)
{
    float declinacion_rad, eleva_sol_rad, latitud_rad;
    double acimut;
    declinacion_rad = (2*pi*declinacion)/360; //Pasamos los datos a
    //radianes ya que las operaciones trigonometricas las calcula //con
    radianes.
    eleva_sol_rad = (2*pi*elevacion_solar)/360;
    latitud_rad = (2*pi*latitud)/360;
    acimut=(sin(eleva_sol_rad)*sin(latitud_rad)-
    sin(declinacion_rad))/(cos(eleva_sol_rad)*cos(latitud_rad));
    acimut=acos(acimut); //calculamos el acimut
    acimut=(360*acimut)/(2*pi); //lo pasamos a grados.
    return(acimut); //devolvemos el valor al programa principal.
}

```

```

/*Subprograma que calcula el dia del año en que nos encontramos*/

int fecha_num (int dia, int mes, int ano)
{
    int dias_mes[12]={31,28,31,30,31,30,31,31,30,31,30,31}; // array //con
    los dias que tiene cada mes del año.
    int num_dia,cont;
    if(ano%4==0||ano%100==0||ano%400==0) dias_mes[1]=29; //sustituye //el
    valor de los dias del mes de febrero cuando el año es //bisisesto.
    num_dia=0;
    for(cont=0;cont<(mes-1);cont++)//Bucle que va sumando los dias //de
    los meses que han pasado.
    {
        num_dia=num_dia+dias_mes[cont];
    }
    num_dia=num_dia+dia;//sumamos los dias que han pasado del mes.
    return (num_dia); //devuelve el valor al programa principal.
}

```

## 8.Programa test cálculo ecuación de tiempo.

```

#include <math.h>//Nombre librería que utilizamos.
#define pi 3.14159 //define el nombre y el valor de una constante.

int dia_num=0; //esta variable la podremos utilizar en cualquier parte
//del programa, tanto programa principal, como subprogramas.

/*Subprogramas que utilizamos.*/
float calc_ecu_tiempo_spencer (int dia_num);

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    float EC_tiempo;
    if (dia_num <365)//pequeño programa para probar todos los días //del
    año. Entra aquí si dia_num es menor de 365.
    {
        dia_num=dia_num+1; //suma al día del año un día mas.
    }
    else//Si día_num es 365 devuelve el valor de dia_num a 1.
    {
        dia_num=1;//Devuelve el contador a uno.
    }
    EC_tiempo=calc_ecu_tiempo_spencer(dia_num);//Llamamos al
    //subprograma que calcula la ecuación de tiempo.
    Serial.println ("La ecuación de tiempo es:");
    Serial.println(EC_tiempo);
    delay(1000); //Retrasa la siguiente acción 1 segundo.
}

/*Subprograma utilizado para calcular la ecuacio de tiempo.*/
float calc_ecu_tiempo_spencer (int dia_num)
{
    float ec_tiempo, tao;

```

```

    tao=((dia_num-1)*2*pi)/365;//Calculamos tao con el numero del día del
    año.
    ec_tiempo=(0.000075+0.001868*cos(tao)-0.032077*sin(tao)-
    0.014615*cos(2*tao)-0.04089*sin(2*tao))*(229.18);//Formula de la
    ecuación de tiempo.
    return(ec_tiempo);//devuelve el valor de la ecuación de tiempo.
}

```

## 9. Programa test cálculo longitud.

```

#include <math.h>
#define pi 3.14159

float dif_hora =520.32;
int dia =1;
int mes=11;
int ano=2012;

float calc_ecu_tiempo_spencer (int dia_num);
float calc_longitud(float dif_hora,int dia, int mes, int ano);
int fecha_num (int dia, int mes, int ano);

void setup()
{
    Serial.begin (9600);
}

void loop()
{
    float longitud;
    longitud=calc_longitud(dif_hora,dia, mes, ano);
    Serial.println ("La longitud es:");
    Serial.println(longitud);
    delay(1000);
}

float calc_longitud(float dif_hora,int dia, int mes, int ano)
{
    float EC_tiempo, longitud;
    int dia_num;
    Serial.println(dif_hora);
    dia_num=fecha_num (dia,mes,ano);//llamamos a la funcion para que
    calcule el
    Serial.println(dia_num);
    EC_tiempo=calc_ecu_tiempo_spencer(dia_num);//Llamamos al subprograma
    y calculo
    Serial.println(EC_tiempo);
    longitud =(720-dif_hora-EC_tiempo)*0.25;//Aplicamos la
    formula,longitud
    Serial.println(longitud);
    return(longitud);
}

/*Subprograma utilizado para calcular la ecuacio de tiempo.*/
float calc_ecu_tiempo_spencer (int dia_num)
{
    float ec_tiempo, tao;

```

```

    tao=((dia_num-1)*2*pi)/365;//Calculamos tao con el numero del día del
    año.
    ec_tiempo=(0.000075+0.001868*cos(tao)-0.032077*sin(tao)-
    0.014615*cos(2*tao)-0.04089*sin(2*tao))*(229.18);//Formula de la
    ecuación de tiempo.
    return(ec_tiempo);//devuelve el valor de la ecuación de tiempo.
}

/*Subprograma que calcula el dia del año en que nos encontramos*/
int fecha_num (int dia, int mes, int ano)
{
    int dias_mes[12]={31,28,31,30,31,30,31,31,30,31,30,31};// array //con
    los dias que tiene cada mes del año.
    int num_dia,cont;
    if(ano%4==0||ano%100==0||ano%400==0) dias_mes[1]=29; //sustituye //el
    valor de los dias del mes de febrero cuando el año es //bisisesto.
    num_dia=0;
    for(cont=0;cont<(mes-1);cont++)//Bucle que va sumando los dias //de
    los meses que han pasado.
    {
        num_dia=num_dia+dias_mes[cont];
    }
    num_dia=num_dia+dia;//sumamos los dias que han pasado del mes.
    return (num_dia); //devuelve el valor al programa principal.
}

```

## 10. Programa test potenciómetros.

Debido a que en nuestro proyecto no vamos a diseñar los sensores, vamos a programar un programa para simular nuestros sensores con dos potenciómetros.

```

int potPin=0;//ponemos el pin A0 como entrada del potenciómetro. Este
//potenciómetro regularemos la elevación solar "medida".
int potPin1=1; //Este potenciómetro medira el acimut "medido".

/*Llamamos a los subprogramas.*/
float elev_solar_med (void);//Subprograma que medirá lo que marcara el
//primer potenciómetro.
int ang_acimut_med (void);// Subprograma que medirá lo que marcara el
//segundo potenciómetro.

void setup()
{
    Serial.begin(9600);
}
void loop()
{
    int acimut_med;//Variable guarda el acimut medido.
    float elevacion_solar; //Variable guarda elevación solar.
    elevacion_solar=elev_solar_med();//Llamamos a la función que //mide
    el potenciómetro de la elevación solar y lo metemos en la //variable.
    Serial.println(elevacion_solar); //Muestra la elevación solar en
    //pantalla.
    delay(1000);
    acimut_med =ang_acimut_med();//Llamamos a la función que mide el
    //potenciómetro del acimut y lo metemos en la variable.
    Serial.println(acimut_med); //Muestra el acimut en pantalla.
    delay(1000);
}

```

```

/*Subprograma que recoge el dato de la elevación solar*/
float elev_solar_med (void)
{
    int elevacion_solar;
    float elev_solar;
    elevacion_solar = analogRead(potPin); //Comando para leer el //valor
    del potenciómetro.
    elevacion_solar = map(elevacion_solar, 0, 1023, 0,900); //Convertimos
    el valor que nos devuelve el convertidor analógico //digital del
    Arduino a los valores que nosotros decidamos. Los //convertimos de 0
    a 900 porque de esta manera podemos medir la //elevación solar con
    decimales.
    elev_solar = float(elevacion_solar); //Convierte un dato tipo int //en
    un dato tipo float.
    elev_solar = elev_solar/10; //Lo dividimos entre 10 y así podemos
    //obtener valor de la elevación solar con un decimal.
    return(elev_solar); //Devuelve el valor de la elevación solar.
}
int ang_acimut_med (void)
{
    int angulo_acimut, ang;
    angulo_acimut=analogRead(potPin1); //Comando para leer el valor //del
    potenciómetro.
    ang = map(angulo_acimut, 0, 1023, 0, 360); //Pasa el valor del
    //potenciómetro a valores comprendidos entre 0 y 360, que son
    //valores suficientes para nuestro propósito.
    return(ang); //Devuelve el valor del acimut.
}

```

## 11. Programa test programa principal.

```

/*librerías utilizadas*/
#include <Time.h>
#include <math.h>

/*Variables constantes*/
#define pi 3.14159

/*Variables utilizadas en todo el programa.*/
int elevacion_anterior=100; //variable utilizada para iniciar el //programa
primera vez.
float longitud; //Variable utilizada para el cálculo de la longitud.
float latitud; //Variable utilizada para el cálculo de la latitud.

/*Pines de entrada*/
int potPin=0; //ponemos el pin 0 como entrada del potenciómetro de la
//elevación solar.
int potPin1=1; //ponemos el pin 1 como entrada del potenciómetro de el
//acimut.

/*Subprogramas utilizados*/
void digitalClockDisplay(); //Subprograma utilizado para mostrar el //reloj
y la fecha.
void printDigits(int digito); //Subprograma utilizado para mostrar un //cero
delante de los minutos y segundos menores de 10.

```

```

float elev_solar_med (void); //Llamamos al primer potenciómetro para //medir
la elevación solar.
int ang_acimut_med (void); //Llamamos al segundo potenciómetro para //medir
el ángulo acimutal.
float calc_longitud(float dif_hora2); //Subprograma utilizado para
//calcular la longitud.
float calc_ecu_tiempo_spencer (int dia_num); //Subprograma utilizado para
calcular la ecuación de tiempo.
int fecha_num (void); //subprograma utilizado para calcular el número //de
día del año.
float calc_latitud(float dif_hora, float elevacion_solar); //Subprograma
para calculo de la latitud.
float calc_decli_spencer (int dia_num); //subprograma calculo //declinación.
float calc_acimut_teor (float elevacion_solar, float declinacion, float
latitud); //Subprograma calculo acimut.
float correccion_elevacion(float elev_solar); //Subprograma utilizado para
corregir la elevación solar.

void setup()
{
    Serial.begin(9600);
    setTime(12,52,0,12,1,12); /*Con este comando iniciamos el reloj y la
fecha para nuestro Arduino.(hora,minutos,segundos,dia,mes, año).*/
}
void loop()
{
    int acimut_med; //Variables utilizadas para los valores medidos //por
nuestro sensor.
    float elevacion_solar, elevacion_inicio; //Variables utilizadas
//para los valores medidos por nuestro sensor.
    int horal, hora2, minuto1, minuto2, segundo1, segundo2; //Variables
utilizadas para el calculo de dif_hora.
    float dif_hora, dif_hora2; //Variable utilizada guardar la diferencia
//entre la primera medida y la segunda. Y variable utilizada para
guardar la diferencia entre la segunda medida y la hora Greenwich
cuando se produce el mediodia solar. En minutos.
    float hora_min2; //Variable para guandar la hora 2 en minutos.
    digitalWrite(); //muestra la hora de greenwich.
    elevacion_solar=elev_solar_med(); //Metemos el valor medido por //el
potenciómetro de la elevación solar.
    if (elevacion_solar < 5) //Para realizar la primera medida vamos //a
tomar de inicio esta elevación solar ya que los errores
{// debidos al paralaje y la refracción solar son aceptables.
        do
        {
            elevacion_solar=elev_solar_med(); //Metemos el valor
//medido por el potenciómetro de la elevación solar.
            delay(60000); //una medida por minuto
        }
        while (elevacion_solar < 5); //Reliza la medida mientras la
//elevación solar sea menor de 5°. Este valor lo podemos //variar
para que haga la medida con la elevación que //queramos.
        horal=hour(); //guardan la hora, minutos y segundos de //cuando
realizamos la primera medida.
        minuto1=minute();
        segundo1=second();
        elevacion_inicio=elevacion_solar; //guardamos la elevación
//solar medida en la elevación_inicio.
    }
    else
    {

```

```

elevacion_anterior = elevacion_solar;//sustituye el valor //de
elevación por uno mas alto.
delay(300000);//cinco minutos hasta realizar la siguiente
//medida.
elevacion_solar = elev_solar_med();//Metemos el valor //medido
por el potenciómetro de la elevación solar.
if (elevacion_solar > elevacion_anterior)
{
    horal=hour();//guarda la hora.
    minuto1=minute();//guarda el minuto.
    segundo1=second();//guarda el segundo.
    elevacion_inicio=elevacion_solar;//guarda la //elevacion
solar medida en la elevación_inicio.
}
else
{
    do//doble bucle que hace que se cumpla que:1.La elevación
//solar este aumentando.
{// 2.La elevación solar es mayor de 10°.(Refracción
//solar).
        do
        {
            elevacion_anterior=elevacion_solar;
            //sustituye el valor de elevación por el
            //medido anteriormente.
            delay(60000);//una medida por minuto
            elevacion_solar=elev_solar_med();
            //Metemos el valor medido por //potenciómetro
            de la elevación solar.
        }
        while (elevacion_solar < elevacion_anterior); //Se
        produce el bucle mientras la elevación //solar este
        disminuyendo.
    }
    while (elevacion_solar < 5); //mientras la elevación
//solar sea menor de 5°
    horal=hour();//guardamos la hora, minuto y segundo de
    minuto1=minute();//la primera medida.
    segundo1=second();
    elevacion_inicio=elevacion_solar; //guarda la //elevación
solar medida en la elevación_inicio.
}
}
do
{
    elevacion_solar = elev_solar_med();//Metemos el valor //medido
por el potenciometro de la elevacion solar.
    delay (60000); // Cada minuto.
}
while (elevacion_solar > elevacion_inicio); //mientras la //elevación
solar sea mayor que la elevación medida en la //primera medida.
hora2=hour();//guardamos la hora, minutos, y segundos de la
minuto2=minute();//segunda medida.
segundo2=second();
/*calculamos la diferencia de horas entre la primera medida y la
segunda.*/
if (horal >= hora2) //Si horal es mayor o igual que hora2.
{
    dif_hora=((24-horal+hora2)*60)+(minuto2-minuto1)+((segundo2-
segundo1)/60);//calculo de dif_hora en //minutos.(precisión de
segundos)
}

```

```

}
if (hora2 > hora1) //Si hora2 es mayor que hora1.
{
    dif_hora=((hora2-hora1)*60)+(minuto2-minuto1)+((segundo2-
segundo1)/60);//calculo de dif_hora en minutos.(precisión //de
segundos)
}
if(dif_hora < 120)//si el intervalo escogido es menor de 2 //horas.
{
    Serial.println("El intervalo de tiempo es demasiado pequeño");
    Serial.println("La medida no sera fiable.");
}
else//si el intervalo escogido es mayor de 2 horas.
{
    hora_min2=hora2*60+minuto2+segundo2/60;//calcula la
dif_hora2=hora_min2-(dif_hora/2);// diferencia de tiempo entre
if (dif_hora < 0)// la segunda medida y el mediodía solar.
{
    dif_hora2=1440-dif_hora2;
}
longitud=calc_longitud(dif_hora2);//llamamos al subprograma
//para calcular la longitud.
latitud=calc_latitud(dif_hora,elevacion_solar);//llamamos //al
subprograma para que calcule la latitud.
Serial.print("La longitud será:");
Serial.print(longitud);
Serial.print("La latitud será:");
Serial.print(latitud);
}
}

/*Subprograma que muestra el reloj para Arduino*/
void digitalClockDisplay()
{
    Serial.print(hour());
    printDigits(minute());
    printDigits(second());
    Serial.print(" ");
    Serial.print(day());
    Serial.print(" ");
    Serial.print(month());
    Serial.print(" ");
    Serial.print(year());
    Serial.println();
}

/*Subprograma que pone un 0 delante de los segundos y minutos cuando estos
son menores de 10.*/
void printDigits(int digito)
{
    Serial.print(":");
    if(digito < 10) Serial.print('0');
    Serial.print(digito);
}

/*Subprograma que coge la elevación solar del sensor.*/
float elev_solar_med (void)
{
    int elevacion_solar;

```



```

float elev_solar;
elevacion_solar = analogRead(potPin);
elevacion_solar = map(elevacion_solar, 0, 1023, 0, 900);
elev_solar = float(elevacion_solar);
elev_solar = elev_solar/10;
elev_solar=correccion_elevacion(elev_solar); //Llamamos al
//subprograma que corrige la elevación solar.
return(elev_solar);
}

/*Subprograma que coge el acimut del sensor*/
int ang_acimut_med (void)
{
    int angulo_acimut, ang;
    angulo_acimut=analogRead(potPin1);
    ang = map(angulo_acimut, 0, 1023, 0, 360);.
    return(ang);
}

/*Subprograma que calcula la longitud.*/
float calc_longitud(float dif_hora2)
{
    float EC_tiempo, longitud;
    int dia_num;
    dia_num=fecha_num EC_tiempo=calc_ecu_tiempo_spencer(dia_num);
    longitud =(720-dif_hora2-EC_tiempo)*0.25;
    return(longitud);
}

/*Subprograma que calcula la ecuación de tiempo dependiendo del dia del
año*/
float calc_ecu_tiempo_spencer (int dia_num)
{
    float ec_tiempo, tao;
    tao=((dia_num-1)*2*pi)/365;
    ec_tiempo=(0.000075+0.001868*cos(tao)-0.032077*sin(tao)-
0.014615*cos(2*tao)-0.04089*sin(2*tao))*(229.18);
    return(ec_tiempo);
}

/*Subprograma que calcula el dia del año dependiendo de la fecha*/
int fecha_num (void)
{
    int dias_mes[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    int num_dia,cont,dia,mes,ano;
    dia=day();
    mes=month();
    ano=year();
    if(ano%4==0||ano%100==0||ano%400==0) dias_mes[1]=29;
    num_dia=0;
    for(cont=0;cont<(mes-1);cont++)
    {
        num_dia=num_dia+dias_mes[cont];
    }
    num_dia=num_dia+dia;
    return (num_dia);
}

/*Subprogrma utilizado para calcular la latitud*/
float calc_latitud(float dif_hora, float elevacion_solar)
{
    int dia_num;

```

```

float angulo_horario, decli, declinacion, elev_solar;
float a, b, inver1, inver2, latitud1, latitud2;
float acimut_calc1, acimut_calc2, acimut_med;
dia_num=fecha_num ();
declinacion=calc_decli_spencer (dia_num);
angulo_horario=(dif_hora/2)*0.25;
angulo_horario=(2*pi*angulo_horario)/360;
decli=(2*pi*declinacion)/360;
elev_solar=(2*pi*elevacion_solar)/360;
a=-((sin (decli)/(cos (decli)*cos(angulo_horario)));
b=sin (elev_solar)/ (cos (decli)*cos(angulo_horario));
inver1=((-2*a*b)+sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-1)))/
(2*(1+sq(a)));
inver2=((-2*a*b)-sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-1)))/
(2*(1+sq(a)));
latitud1=asin(inver1);
latitud1=(360*latitud1)/(2*pi);
latitud2=asin(inver2);
latitud2=(360*latitud2)/(2*pi);
acimut_calc1= calc_acimut_teor (elevacion_solar, declinacion,
latitud1);
acimut_calc2= calc_acimut_teor (elevacion_solar, declinacion,
latitud2);
acimut_med = ang_acimut_med ();
if (acimut_calc1-1 <= acimut_med && acimut_med < acimut_calc1+1)
{
    return(latitud1);
}
else
{
    return (latitud2);
}
}

/*Subprograma que calcula la declinacion dependiendo el dia del año.*/
float calc_decli_spencer (int dia_num)
{
    double declinacion, tao;
    tao=((dia_num-1)*2*pi)/365;
    declinacion=((0.006918-(0.399912*cos(tao)))+(0.070257*sin(tao))-
(0.006758*cos(2*tao))+(0.000907*sin(2*tao))-
(0.002697*cos(3*tao))+(0.00148*sin(3*tao)))*180)/pi;
    return(declinacion);
}

/*Subprograma que calcula el acimut teórico.*/
float calc_acimut_teor (float elevacion_solar,float declinacion, float
latitud);
{
    float declinacion_rad, eleva_sol_rad, latitud_rad;
    float acimut;
    declinacion_rad = (2*pi*declinacion)/360;
    eleva_sol_rad = (2*pi*elevacion_solar)/360;
    latitud_rad = (2*pi*latitud)/360;
    acimut=(sin(eleva_sol_rad)*sin(latitud_rad)-
sin(declinacion_rad))/(cos(eleva_sol_rad)*cos(latitud_rad));
    acimut=acos(acimut);
    acimut=(360*acimut)/(2*pi);
    acimut=360-acimut;
    return(acimut);
}

```

```

/*Subprograma que corrige el error cometido debido a la refracción
atmosferica y a el semidiametro sola.*/
float correccion_elevacion(float elev_solar)
{
    float refraccion,elevacion_solar;
    float semidiametro_solar=0.256;
    if (elevacion_solar <= 15)
    {
        refraccion=((34.133+(4.197*elev_solar)+(0.00428*sq(elev_solar))
/(1+(0.505*elev_solar)+(0.0845*sq(elev_solar))))
*(0.016);
    }
    else
    {
        elevacion_solar=(2*pi*elevacion_solar)/360;
        refraccion=((0.97127/tan(elevacion_solar))-
(0.00137/pow(tan(elevacion_solar),3)))*(0.016);
    }
    elev_solar=elev_solar-refraccion+semidiametro_solar;
    return(elev_solar);
}

```

## PROGRAMA PRINCIPAL

```
/*librerias utilizadas*/
#include <Time.h>
#include <math.h>

/*Variables constantes*/
#define pi 3.14159

/*Variables utilizadas en todo el programa.*/
int elevacion_anterior=100; //variable utilizada para iniciar el //programa
primera vez.
float longitud; //Variable utilizada para el calculo de la longitud.
float latitud; //Variable utilizada para el calculo de la latitud.

/*Pines de entrada*/
int potPin=0; //ponemos el pin 0 como entrada del potenciómetro de la
//elevación solar.
int potPin1=1; //ponemos el pin 1 como entrada del potenciómetro de el
//acimut.

/*Subprogramas utilizados*/
void digitalClockDisplay(); //Subprograma utilizado para mostrar el //reloj
y la fecha.
void printDigits(int digito); //Subprograma utilizado para mostrar un //cero
delante de los minutos y segundos menores de 10.
float elev_solar_med (void); //Llamamos al primer potenciómetro para //medir
la elevación solar.
int ang_acimut_med (void); //Llamamos al segundo potenciómetro para //medir
el ángulo acimutal.
float calc_longitud(float dif_hora2); //Subprograma utilizado para
//calcular la longitud.
float calc_ecu_tiempo_spencer (int dia_num); //Subprograma utilizado para
calcular la ecuación de tiempo.
int fecha_num (void); //subprograma utilizado para calcular el número //de
día del año.
float calc_latitud(float dif_hora, float elevacion_solar); //Subprograma
para calculo de la latitud.
float calc_decli_spencer (int dia_num); //subprograma calculo //declinación.
float calc_acimut_teor (float elevacion_solar, float declinacion, float
latitud); //Subprograma calculo acimut.
float correccion_elevacion(float elev_solar); //Subprograma utilizado para
corregir la elevación solar.

void setup()
{
    Serial.begin(9600);
    setTime(12,52,0,12,1,12); //Con este comando iniciamos el reloj y la
fecha para nuestro Arduino.(hora,minutos,segundos,dia,mes, año).*/
}
void loop()
{
    int acimut_med; //Variables utilizadas para los valores medidos //por
nuestro sensor.
    float elevacion_solar, elevacion_inicio; //Variables utilizadas
//para los valores medidos por nuestro sensor.
    int hora1, hora2, minuto1, minuto2, segundo1, segundo2; //Variables
utilizadas para el calculo de dif_hora.
```

```

float dif_hora, dif_hora2;//Variable utilizada guardar la diferencia
//entre la primera medida y la segunda.La segunda variable es para
guardar la diferencia horaria entre la sgunda medida y la hora de
Greenwich en la que se produce el mediodia local. En minutos.
float hora_min2;//Variable para guandar la hora 2 en minutos.
digitalClockDisplay();//muestra la hora de greenwich.
elevacion_solar=elev_solar_med();//Metemos el valor medido por //el
potenciómetro de la elevación solar.
if (elevacion_solar < 5)//Para realizar la primera medida vamos //a
tomar de inicio esta elevación solar ya que los errores
{// debidos al paralaje y la refracción solar son aceptables.
    do
    {
        elevacion_solar=elev_solar_med();//Metemos el valor
        //medido por el potenciometro de la elevación solar.
        delay(60000);//una medida por minuto
    }
    while (elevacion_solar < 5);//Reliza la medida mientras la
    //elevación solar sea menor de 5°.Este valor lo podemos //variar
    para que haga la medida con la elevación que //queramos.
    horal=hour();//guardan la hora, minutos y segundos de //cuando
    realizamos la primera medida.
    minutol=minute();
    segundol=second();
    elevacion_inicio=elevacion_solar;//guardamos la elevación
    //solar medida en la elevación_inicio.
}
else
{
    elevacion_anterior = elevacion_solar;//sustituye el valor //de
    elevación por uno mas alto.
    delay(300000);//cinco minutos hasta realizar la siguiente
    //medida.
    elevacion_solar = elev_solar_med();//Metemos el valor //medido
    por el potenciómetro de la elevación solar.
    if (elevacion_solar > elevacion_anterior)
    {
        horal=hour();//guarda la hora.
        minutol=minute();//guarda el minuto.
        segundol=second();//guarda el segundo.
        elevacion_inicio=elevacion_solar;//guarda la //elevacion
        solar medida en la elevación_inicio.
    }
    else
    {
        do//doble bucle que hace que se cumpla que:1.La elevación
        //solar este aumentando.
        {// 2.La elevación solar es mayor de 10°.(Refracción
        //solar).
            do
            {
                elevacion_anterior=elevacion_solar;
                //sustituye el valor de elevación por el
                //medido anteriormente.
                delay(60000);//una medida por minuto
                elevacion_solar=elev_solar_med();
                //Metemos el valor medido por //potenciómetro
                de la elevación solar.
            }
        }
    }
}
}

```

```

        while (elevacion_solar < elevacion_anterior); //Se
        produce el bucle mientras la elevación //solar este
        disminuyendo.
    }
    while (elevacion_solar < 5); //mientras la elevación
    //solar sea menor de 5°
    horal=hour();//guardamos la hora, minuto y segundo de
    minuto1=minute();//la primera medida.
    segundo1=second();
    elevacion_inicio=elevacion_solar; ;//guarda la //elevación
    solar medida en la elevación_inicio.
}
do
{
    elevacion_solar = elev_solar_med();//Metemos el valor //medido
    por el potenciómetro de la elevacion solar.
    delay (60000); // Cada minuto.
}
while (elevacion_solar > elevacion_inicio); //mientras la //elevación
solar sea mayor que la elevación medida en la //primera medida.
hora2=hour();//guardamos la hora, minutos, y segundos de la
minuto2=minute();//segunda medida.
segundo2=second();
/*calculamos la diferencia de horas entre la primera medida y la
segunda.*/
if (horal >= hora2) //Si horal es mayor o igual que hora2.
{
    dif_hora=((24-horal+hora2)*60)+(minuto2-minuto1)+((segundo2-
segundo1)/60);//calcula de dif_hora en //minutos.(precisión de
segundos)
}
if (hora2 > horal) //Si hora2 es mayor que horal.
{
    dif_hora=((hora2-horal)*60)+(minuto2-minuto1)+((segundo2-
segundo1)/60);//calcula de dif_hora en minutos.(precisión //de
segundos)
}
if(dif_hora < 120)//si el intervalo escogido es menor de 2 //horas.
{
    Serial.println("El intervalo de tiempo es demasiado pequeño");
    Serial.println("La medida no sera fiable.");
}
else//si el intervalo escogido es mayor de 2 horas.
{
    hora_min2=hora2*60+minuto2+segundo2/60;//calcula la
    dif_hora2=hora_min2-(dif_hora/2);// diferencia de tiempo entre
    if (dif_hora2 < 0)// la segunda medida y el mediodía solar.
    {
        dif_hora2=1440-dif_hora2;
    }
    longitud=calc_longitud(dif_hora2);//llamamos al subprograma
    //para calcular la longitud.
    latitud=calc_latitud(dif_hora,elevacion_solar);//llamamos //al
    subprograma para que calcule la latitud.
    Serial.print("La longitud será:");
    Serial.print(longitud);
    Serial.print("La latitud será:");
    Serial.print(latitud);
}
}
}

```

```

}

/*Subprograma que muestra el reloj para Arduino*/
void digitalClockDisplay()
{
    Serial.print(hour());
    printDigits(minute());
    printDigits(second());
    Serial.print(" ");
    Serial.print(day());
    Serial.print(" ");
    Serial.print(month());
    Serial.print(" ");
    Serial.print(year());
    Serial.println();
}

/*Subprograma que pone un 0 delante de los segundos y minutos cuando estos
son menores de 10.*/
void printDigits(int digito)
{
    Serial.print(":");
    if(digito < 10) Serial.print('0');
    Serial.print(digito);
}

/*Subprograma que coge la elevación solar del sensor.*/
float elev_solar_med (void)
{
    int elevacion_solar;
    float elev_solar;
    elevacion_solar = analogRead(potPin);
    elevacion_solar = map(elevacion_solar, 0, 1023, 0, 900);
    elev_solar = float(elevacion_solar);
    elev_solar = elev_solar/10;
    elev_solar=correccion_elevacion(elev_solar); //Llamamos al
//subprograma que corrige la elevación solar.
    return(elev_solar);
}

/*Subprograma que coge el acimut del sensor*/
int ang_acimut_med (void)
{
    int angulo_acimut, ang;
    angulo_acimut=analogRead(potPin1);
    ang = map(angulo_acimut, 0, 1023, 0, 360);
    return(ang);
}

/*Subprograma que calcula la longitud.*/
float calc_longitud(float dif_hora2)
{
    float EC_tiempo, longitud;
    int dia_num;
    dia_num=fecha_num EC_tiempo=calc_ecu_tiempo_spencer(dia_num);
    longitud =(720-dif_hora2-EC_tiempo)*0.25;
    return(longitud);
}

```

```

/*Subprograma que calcula la ecuación de tiempo dependiendo del día del
año*/
float calc_ecu_tiempo_spencer (int dia_num)
{
    float ec_tiempo, tao;
    tao=((dia_num-1)*2*pi)/365;
    ec_tiempo=(0.000075+0.001868*cos(tao)-0.032077*sin(tao)-
0.014615*cos(2*tao)-0.04089*sin(2*tao))*(229.18);
    return(ec_tiempo);
}

/*Subprograma que calcula el día del año dependiendo de la fecha*/
int fecha_num (void)
{
    int dias_mes[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    int num_dia, cont, dia, mes, ano;
    dia=day();
    mes=month();
    ano=year();
    if(ano%4==0||ano%100==0||ano%400==0) dias_mes[1]=29;
    num_dia=0;
    for(cont=0;cont<(mes-1);cont++)
    {
        num_dia=num_dia+dias_mes[cont];
    }
    num_dia=num_dia+dia;
    return (num_dia);
}

/*Subprograma utilizado para calcular la latitud*/
float calc_latitud(float dif_hora, float elevacion_solar)
{
    int dia_num;
    float angulo_horario, decli, declinacion, elev_solar;
    float a, b, inver1, inver2, latitud1, latitud2;
    float acimut_calc1, acimut_calc2, acimut_med;
    dia_num=fecha_num ();
    declinacion=calc_decli_spencer (dia_num);
    angulo_horario=(dif_hora/2)*0.25;
    angulo_horario=(2*pi*angulo_horario)/360;
    decli=(2*pi*declinacion)/360;
    elev_solar=(2*pi*elevacion_solar)/360;
    a=-((sin (decli)/(cos (decli)*cos(angulo_horario)));
    b=sin (elev_solar)/ (cos (decli)*cos(angulo_horario));
    inver1=((-2*a*b)+sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-1)))/
(2*(1+sq(a)));
    inver2=((-2*a*b)-sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-1)))/
(2*(1+sq(a)));
    latitud1=asin(inver1);
    latitud1=(360*latitud1)/(2*pi);
    latitud2=asin(inver2);
    latitud2=(360*latitud2)/(2*pi);
    acimut_calc1= calc_acimut_teor (elevacion_solar, declinacion,
latitud1);
    acimut_calc2= calc_acimut_teor (elevacion_solar, declinacion,
latitud2);
    acimut_med = ang_acimut_med ();
    if (acimut_calc1-1 <= acimut_med && acimut_med < acimut_calc1+1)
    {
        return(latitud1);
    }
    else
}

```



```

    {
        return (latitud2);
    }
}

/*Subprograma que calcula la declinacion dependiendo el dia del año.*/
float calc_decli_spencer (int dia_num)
{
    double declinacion, tao;
    tao=((dia_num-1)*2*pi)/365;
    declinacion=((0.006918-(0.399912*cos(tao)))+(0.070257*sin(tao))-
    (0.006758*cos(2*tao))+(0.000907*sin(2*tao))-
    (0.002697*cos(3*tao))+(0.00148*sin(3*tao)))*180)/pi;
    return(declinacion);
}

/*Subprograma que calcula el acimut teórico.*/
float calc_acimut_teor (float elevacion_solar,float declinacion, float
latitud);
{
    float declinacion_rad, eleva_sol_rad, latitud_rad;
    float acimut;
    declinacion_rad = (2*pi*declinacion)/360;
    eleva_sol_rad = (2*pi*elevacion_solar)/360;
    latitud_rad = (2*pi*latitud)/360;
    acimut=(sin(eleva_sol_rad)*sin(latitud_rad)-
sin(declinacion_rad))/(cos(eleva_sol_rad)*cos(latitud_rad));
    acimut=acos(acimut);
    acimut=(360*acimut)/(2*pi);
    acimut=360-acimut;
    return(acimut);
}

/*Subprograma que corrige el error cometido debido a la refracción
atmosferica y a el semidiametro sola.*/
float correccion_elevacion(float elev_solar)
{
    float refraccion,elevacion_solar;
    float semidiametro_solar=0.256;
    if (elevacion_solar <= 15)
    {
        refraccion=((34.133+(4.197*elev_solar)+(0.00428*sq(elev_solar))
        /(1+(0.505*elev_solar)+(0.0845*sq(elev_solar))))
        *(0.016);
    }
    else
    {
        elevacion_solar=(2*pi*elevacion_solar)/360;
        refraccion=((0.97127/tan(elevacion_solar))-
        (0.00137/pow(tan(elevacion_solar),3)))*(0.016);
    }
    elev_solar=elev_solar-refraccion+semidiametro_solar;
    return(elev_solar);
}

```

## PROGRAMA PREPARADO PARA ROBOT

```
/*ORIENTACION SOLAR UPNA*/

/*librerias utilizadas*/
#include <Time.h>
#include <math.h>

/*Variables constantes*/
#define pi 3.14159

/*Variables utilizadas en todo el programa.*/
int elevacion_anterior=100; //variable utilizada para iniciar el //programa
primera vez.
float longitud; //Variable utilizada para el calculo de la longitud.
float latitud; //Variable utilizada para el calculo de la latitud.

/*Subprogramas utilizados*/
void orientacion_solar(); //Seria el subprograma que calcularia la latitud y
la longitud en la que se encontraria nuestro robot.
float elev_solar_med (void); //Llamamos al primer potenci6metro para //medir
la elevaci6n solar.
int ang_acimut_med (void); //Llamamos al segundo potenci6metro para //medir
el 6ngulo acimutal.
float calc_longitud(float dif_hora2); //Subprograma utilizado para
//calcular la longitud.
float calc_ecu_tiempo_spencer (int dia_num); //Subprograma utilizado para
calcular la ecuaci6n de tiempo.
int fecha_num (void); //subprograma utilizado para calcular el n6mero //de
d6a del a6o.
float calc_latitud(float dif_hora, float elevacion_solar); //Subprograma
para calculo de la latitud.
float calc_decli_spencer (int dia_num); //subprograma calculo //declinaci6n.
float calc_acimut_teor (float elevacion_solar, float declinacion, float
latitud); //Subprograma calculo acimut.
float correccion_elevacion(float elev_solar); //Subprograma utilizado para
corregir la elevaci6n solar.

void setup()
{
    setTime(12,52,0,12,1,12); //Con este comando iniciamos el reloj y la
    fecha para nuestro Arduino. (hora, minutos, segundos, dia, mes, a6o).*/
}
void loop()
{
    orientaci6n_solar(); //llama al subprograma.
    /*Aqu6 deber6amos introducir todos las acciones y subprogramas que
    queramos para el funcionamiento de nuestro robot. Una vez ejecutado
    este subprograma podremos utilizar la longitud y latitud calculadas
    para lo que deseemos.*/
}

void orientacion_solar()
{
    int acimut_med; //Variables utilizadas para los valores medidos //por
    nuestro sensor.
    float elevacion_solar, elevacion_inicio; //Variables utilizadas
    //para los valores medidos por nuestro sensor.
```

```

int hora1, hora2, minuto1, minuto2, segundo1, segundo2; //Variables
utilizadas para el calculo de dif_hora.
float dif_hora, dif_hora2; //Variable utilizada guardar la diferencia
//entre la primera medida y la segunda. La segunda variable es para
guardar la diferencia horaria entre la segunda medida y la hora de
Greenwich en la que se produce el mediodia local. En minutos.
float hora_min2; //Variable para guardar la hora 2 en minutos.
elevacion_solar=elev_solar_med();//Metemos el valor medido por //el
potenciómetro de la elevación solar.
if (elevacion_solar < 5) //Para realizar la primera medida vamos //a
tomar de inicio esta elevación solar ya que los errores
{// debidos al paralaje y la refracción solar son aceptables.
    do
    {
        elevacion_solar=elev_solar_med();//Metemos el valor
        //medido por el potenciómetro de la elevación solar.
        delay(60000); //una medida por minuto
    }
    while (elevacion_solar < 5); //Reliza la medida mientras la
//elevación solar sea menor de 5°. Este valor lo podemos //variar
para que haga la medida con la elevación que //queramos.
    hora1=hour();//guardan la hora, minutos y segundos de //cuando
realizamos la primera medida.
    minuto1=minute();
    segundo1=second();
    elevacion_inicio=elevacion_solar; //guardamos la elevación
//solar medida en la elevación_inicio.
}
else
{
    elevacion_anterior = elevacion_solar; //sustituye el valor //de
elevación por uno mas alto.
    delay(300000); //cinco minutos hasta realizar la siguiente
//medida.
    elevacion_solar = elev_solar_med(); //Metemos el valor //medido
por el potenciómetro de la elevación solar.
    if (elevacion_solar > elevacion_anterior)
    {
        hora1=hour(); //guarda la hora.
        minuto1=minute(); //guarda el minuto.
        segundo1=second(); //guarda el segundo.
        elevacion_inicio=elevacion_solar; //guarda la //elevacion
solar medida en la elevación_inicio.
    }
    else
    {
        do //doble bucle que hace que se cumpla que: 1. La elevación
//solar este aumentando.
        { // 2. La elevación solar es mayor de 10°. (Refracción
//solar).
            do
            {
                elevacion_anterior=elevacion_solar;
                //sustituye el valor de elevación por el
                //medido anteriormente.
                delay(60000); //una medida por minuto
                elevacion_solar=elev_solar_med();
                //Metemos el valor medido por //potenciómetro
de la elevación solar.
            }
        }
    }
}
}

```

```

        while (elevacion_solar < elevacion_anterior); //Se
        produce el bucle mientras la elevación //solar este
        disminuyendo.
    }
    while (elevacion_solar < 5); //mientras la elevación
    //solar sea menor de 5°
    horal=hour();//guardamos la hora, minuto y segundo de
    minuto1=minute();//la primera medida.
    segundo1=second();
    elevacion_inicio=elevacion_solar; ;//guarda la //elevación
    solar medida en la elevación_inicio.
}
}
do
{
    elevacion_solar = elev_solar_med();//Metemos el valor //medido
    por el potenciómetro de la elevacion solar.
    delay (60000); // Cada minuto.
}
while (elevacion_solar > elevacion_inicio); //mientras la //elevación
solar sea mayor que la elevación medida en la //primera medida.
hora2=hour();//guardamos la hora, minutos, y segundos de la
minuto2=minute();//segunda medida.
segundo2=second();
/*calculamos la diferencia de horas entre la primera medida y la
segunda.*/
if (horal >= hora2) //Si horal es mayor o igual que hora2.
{
    dif_hora=((24-horal+hora2)*60)+(minuto2-minuto1)+((segundo2-
segundo1)/60);//calcula de dif_hora en //minutos.(precisión de
segundos)
}
if (hora2 > horal) //Si hora2 es mayor que horal.
{
    dif_hora=((hora2-horal)*60)+(minuto2-minuto1)+((segundo2-
segundo1)/60);//calcula de dif_hora en minutos.(precisión //de
segundos)
}
if(dif_hora < 120)//si el intervalo escogido es menor de 2 //horas.
{
    break;
}
else//si el intervalo escogido es mayor de 2 horas.
{
    hora_min2=hora2*60+minuto2+segundo2/60;//calcula la
dif_hora2=hora_min2-(dif_hora/2);// diferencia de tiempo entre
if (dif_hora < 0)// la segunda medida y el mediodía solar.
{
    dif_hora2=1440-dif_hora2;
}
longitud=calc_longitud(dif_hora2);//llamamos al subprograma
//para calcular la longitud.
latitud=calc_latitud(dif_hora,elevacion_solar);//llamamos //al
subprograma para que calcule la latitud.
}
}
}

/*Subprograma que coge la elevación solar del sensor.*/
float elev_solar_med (void)
{

```

```

    /*Aquí deberemos escribir el subprograma para controlar nuestro
    sensor de la elevación solar dependiendo de su utilización.*/
    float elev_solar;
    elev_solar=correccion_elevacion(elev_solar); //Este subprograma
    //corregira el error debido a la refraccion y el semidiametro
    //solar.
    return(elev_solar);
}

/*Subprograma que coge el acimut del sensor*/
int ang_acimut_med (void)
{
    int ang;
    /*Aquí escribiremos los comandos para controlar el sensor de control
    del acimut dependiendo como deiseñemos nuestro sensor.*/
    return(ang);
}

/*Subprograma que calcula la longitud.*/
float calc_longitud(float dif_hora2)
{
    float EC_tiempo, longitud;
    int dia_num;
    dia_num=fecha_num();
    EC_tiempo=calc_ecu_tiempo_spencer(dia_num);
    longitud =(720-dif_hora2-EC_tiempo)*0.25;
    return(longitud);
}

/*Subprograma que calcula la ecuación de tiempo dependiendo del dia del
año*/
float calc_ecu_tiempo_spencer (int dia_num)
{
    float ec_tiempo, tao;
    tao=((dia_num-1)*2*pi)/365;
    ec_tiempo=(0.000075+0.001868*cos(tao)-0.032077*sin(tao)-
    0.014615*cos(2*tao)-0.04089*sin(2*tao))*(229.18);
    return(ec_tiempo);
}

/*Subprograma que calcula el dia del año dependiendo de la fecha*/
int fecha_num (void)
{
    int dias_mes[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    int num_dia,cont,dia,mes,ano;
    dia=day();
    mes=month();
    ano=year();
    if(ano%4==0||ano%100==0||ano%400==0) dias_mes[1]=29;
    num_dia=0;
    for(cont=0;cont<(mes-1);cont++)
    {
        num_dia=num_dia+dias_mes[cont];
    }
    num_dia=num_dia+dia;
    return (num_dia);
}

/*Subprogrma utilizado para calcular la latitud*/
float calc_latitud(float dif_hora, float elevacion_solar)
{
    int dia_num;

```

```

float angulo_horario, decli, declinacion, elev_solar;
float a, b, inver1, inver2, latitud1, latitud2;
float acimut_calc1, acimut_calc2, acimut_med;
dia_num=fecha_num ();
declinacion=calc_decli_spencer (dia_num);
angulo_horario=(dif_hora/2)*0.25;
angulo_horario=(2*pi*angulo_horario)/360;
decli=(2*pi*declinacion)/360;
elev_solar=(2*pi*elevacion_solar)/360;
a=-((sin (decli)/(cos (decli)*cos(angulo_horario)));
b=sin (elev_solar)/ (cos (decli)*cos(angulo_horario));
inver1=((-2*a*b)+sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-1)))/
(2*(1+sq(a)));
inver2=((-2*a*b)-sqrt(4*sq(a)*sq(b)-4*(1+sq(a))*(sq(b)-1)))/
(2*(1+sq(a)));
latitud1=asin(inver1);
latitud1=(360*latitud1)/(2*pi);
latitud2=asin(inver2);
latitud2=(360*latitud2)/(2*pi);
acimut_calc1= calc_acimut_teor (elevacion_solar, declinacion,
latitud1);
acimut_calc2= calc_acimut_teor (elevacion_solar, declinacion,
latitud2);
acimut_med = ang_acimut_med ();
if (acimut_calc1-1 <= acimut_med && acimut_med < acimut_calc1+1)
{
    return(latitud1);
}
else
{
    return (latitud2);
}
}

/*Subprograma que calcula la declinacion dependiendo el dia del año.*/
float calc_decli_spencer (int dia_num)
{
    double declinacion, tao;
    tao=((dia_num-1)*2*pi)/365;
    declinacion=((0.006918-(0.399912*cos(tao)))+(0.070257*sin(tao))-
(0.006758*cos(2*tao))+(0.000907*sin(2*tao))-
(0.002697*cos(3*tao))+(0.00148*sin(3*tao)))*180)/pi;
    return(declinacion);
}

/*Subprograma que calcula el acimut teórico.*/
float calc_acimut_teor (float elevacion_solar,float declinacion, float
latitud);
{
    float declinacion_rad, eleva_sol_rad, latitud_rad;
    float acimut;
    declinacion_rad = (2*pi*declinacion)/360;
    eleva_sol_rad = (2*pi*elevacion_solar)/360;
    latitud_rad = (2*pi*latitud)/360;
    acimut=((sin(eleva_sol_rad)*sin(latitud_rad)-
sin(declinacion_rad))/(cos(eleva_sol_rad)*cos(latitud_rad)));
    acimut=acos(acimut);
    acimut=(360*acimut)/(2*pi);
    acimut=360-acimut;
    return(acimut);
}

```

```

/*Subprograma que corrige el error cometido debido a la refracción
atmosferica y a el semidiametro sola.*/
float correccion_elevacion(float elev_solar)
{
    float refraccion,elevacion_solar;
    float semidiametro_solar=0.256;
    if (elevacion_solar <= 15)
    {
        refraccion=((34.133+(4.197*elev_solar)+(0.00428*sq(elev_solar))
/(1+(0.505*elev_solar)+(0.0845*sq(elev_solar))))
*(0.016);
    }
    else
    {
        elevacion_solar=(2*pi*elevacion_solar)/360;
        refraccion=((0.97127/tan(elevacion_solar))-
(0.00137/pow(tan(elevacion_solar),3)))*(0.016);
    }
    elev_solar=elev_solar-refraccion+semidiametro_solar;
    return(elev_solar);
}

```





## ANEXO B: INSTRUCCIONES DE INSTALACION ARDUINO

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (p.ej. Flash, Processing, MaxMSP).

Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, así pues eres libre de adaptarlos a tus necesidades.

Arduino recibió una Mención Honorífica en la sección Digital Communities de la edición del 2006 del Ars Electronica Prix. El equipo Arduino (Arduino team) es: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, y David Mellis.

Para empezar a utilizar la placa Arduino lo primero que tienes que hacer es hacerte con una, y después descargar el software de la página principal: [www.arduino.cc](http://www.arduino.cc)

Para comenzar, sigue las instrucciones que correspondan a tu sistema operativo: Windows, Mac OS X o Linux; o las referentes a tu placa: Arduino Nano, Arduino Mini, Arduino BT, LilyPad Arduino, XBee Shield, Arduino Uno.

El texto que viene a continuación se encuentra en la página principal de arduino, con la explicación para todo tipo de sistemas operativo. Yo apporto el de Windows ya que es el más utilizado.

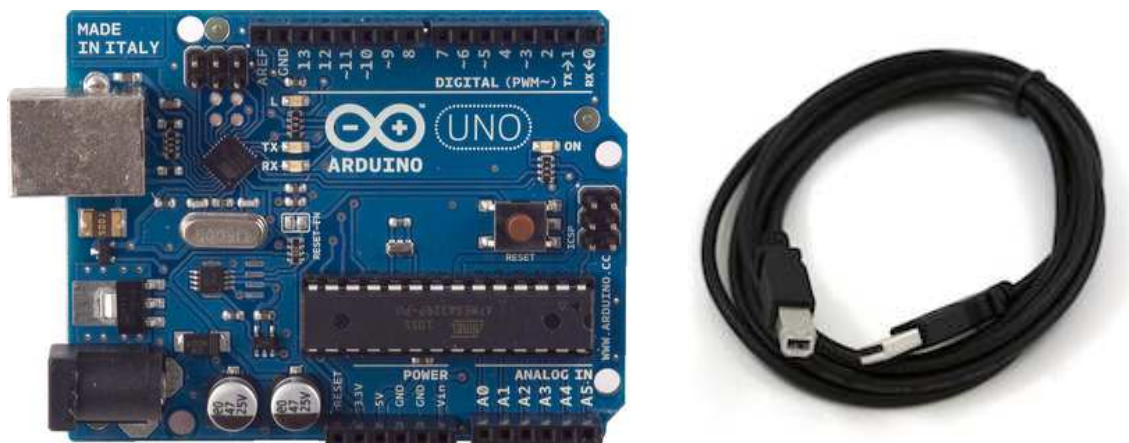
### Comenzando con Arduino en Windows

Este documento explica cómo conectar tu placa Arduino al ordenador y volcar el código de tu primer programa.

- 1 | Consigue un Arduino y un cable USB
- 2 | Descarga el IDE de Arduino
- 3 | Conecta la placa
- 4 | Instala los *drivers*
- 5 | Ejecuta la Aplicación Arduino
- 6 | Abre el ejemplo *Blink*
- 7 | Selecciona tu placa
- 8 | Selecciona tu puerto serie
- 9 | Sube el *sketch* a la placa
- 1 | Consigue un Arduino y un cable USB**

En este tutorial asumimos que estás usando una placa Arduino Duemilanove, Nano, Diecimila o UNO Si tienes cualquier otra placa necesitas leer la página correspondiente a la placa que uses en esta guía de iniciación.

También necesitarás un cable estándar USB (conexión A a conexión B), como los que se usan para conectar, por ejemplo, una impresora USB. (En el caso de la placa Arduino Nano necesitarás un cable de conexión A a conexión mini-B).



## 2 | Descarga el IDE de Arduino

Descarga la última versión de la página de descargas. Cuando la descarga finalice, descomprime el fichero. Asegúrate de mantener la estructura de directorios. Haz doble click en la carpeta *arduino-00XX* para abrirla. Deberías ver una serie de ficheros y carpetas ahí dentro.

## 3 | Conecta la placa

Conecta la placa Arduino a tu ordenador usando el cable USB. el LED verde indicador de la alimentación (nombrado como **PWR** en la placa) debería quedar encendido a partir de ese momento.

Si estás usando una placa Arduino Diecimila, necesitarás asegurarte de que la placa está configurada para alimentarse mediante la conexión USB. La fuente de alimentación se selecciona con un puente ("jumper"), una pequeña pieza de plástico que encaja en dos de los tres pins situados entre los conectores USB y de alimentación de la placa. Comprueba que el puente esté conectando los dos pins más cercanos al puerto USB de la placa.

En las placas Arduino Duemilanove y Arduino Nano la fuente de alimentación adecuada se selecciona de forma automática y no requiere de realizar ninguna comprobación en este sentido.

## 4 | Instala los *drivers*

Cuando conectas la placa, Windows debería inicializar la instalación de los *drivers* (siempre y cuando no hayas utilizado ese ordenador con una placa Arduino anteriormente). En Windows Vista y Windows 7, los drivers deberían descargarse e instalarse automáticamente.

En Windows XP, se abrirá el diálogo de instalación de Nuevo Hardware:

- Cuando te pregunten: **¿Puede Windows conectarse a Windows Update para buscar el software?** Selecciona **“No, no esta vez”**. Haz click en siguiente.
- Selecciona **“Instalar desde una lista o localización específica (Avanzado)”** haz click en siguiente.
- Asegúrate que **“Buscar los mejores drivers en estas localizaciones”** está seleccionado; deselecciona **“Buscar en medios removibles”**; selecciona **“Incluye esta localización en la búsqueda”** y navega al directorio **drivers/FTDI USB Drivers** dentro de la carpeta de Arduino que has descomprimido previamente. (La versión más reciente de los drivers se puede encontrar en la página web del fabricante del chip FTDI). Haz click en siguiente.
- El asistente de instalación buscará los drivers y te anunciará que encontró un **“USB Serial Converter”** (se traduce por Conversor USB-Serie). Haz click en finalizar.
- El asistente de instalación de hardware volverá a iniciarse. Repite los mismos pasos que antes y selecciona la misma carpeta de instalación de los drivers. Esta vez encontrará un **“USB Serial Port”** (o puerto USB-Serie).

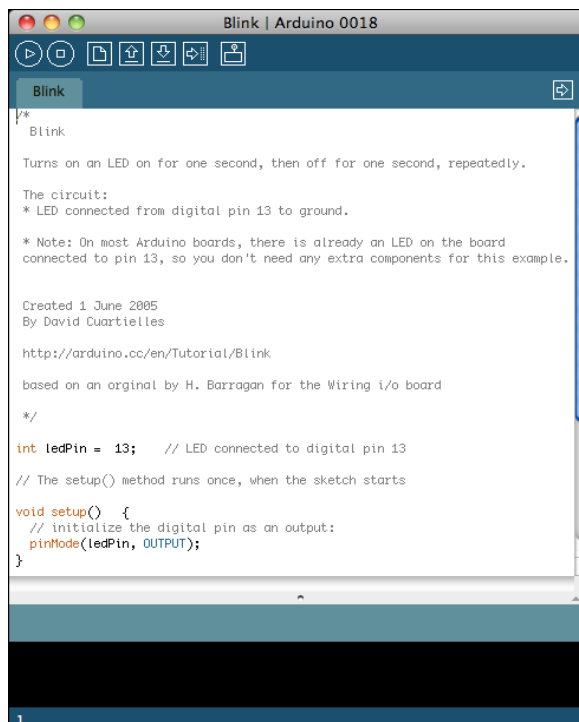
Puedes comprobar que los *drivers* se han instalado correctamente abriendo la carpeta del Administrador de Dispositivos, en el grupo *Dispositivos* del panel de control del sistema. Busca "USB Serial Port" (o *Puerto USB-Serie*) en la sección de puertos; esa es tu placa Arduino.

## 5 | Ejecuta la Aplicación Arduino

Haz doble click en la aplicación Arduino.

## 6 | Abre el ejemplo *Blink*

Abre el programa de ejemplo para hacer parpadear un LED ("LED blink"): **File > Examples > Digital > Blink**.



```
Arduino IDE - Blink | Arduino 0018
Blink
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * The circuit:
 * * LED connected from digital pin 13 to ground.
 *
 * Note: On most Arduino boards, there is already an LED on the board
 * connected to pin 13, so you don't need any extra components for this example.
 *
 * Created 1 June 2005
 * By David Cuartielles
 *
 * http://arduino.cc/en/Tutorial/Blink
 *
 * based on an original by H. Barragan for the Wiring I/O board
 */
int ledPin = 13; // LED connected to digital pin 13
// The setup() method runs once, when the sketch starts
void setup() {
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}
```

## 7 | Selecciona tu placa

Necesitarás seleccionar el tipo de placa de tu Arduino en el menú **Tools > Board**. Para las nuevas placas Arduino con el chip ATmega 328 (comprueba el texto escrito en el chip de la placa), selecciona la opción **Arduino Duemilanove, Nano w/ ATmega328 o UNO** del menú desplegable. Anteriormente las placas Arduino incluían un chip ATmega 168; para estos casos selecciona la opción **Arduino Diecimila, Duemilanove, or Nano w/ ATmega168**. (Se puede encontrar más detalles sobre los dispositivos de entrada de las placas en el menú desplegable en la página del entorno arduino.)

## 8 | Selecciona tu puerto serie

Selecciona el dispositivo serie de la placa Arduino en el menú Tools | Serial Port (*Herramientas | Puertos Serie*). Lo más probable es que sea **COM3** o mayor (**COM1** y **COM2** se reservan, por regla general para puertos serie de hardware). Para asegurarte de cual es, puedes desconectar la placa y volver a mirar el menú; el puerto de la placa habrá desaparecido de la lista. Reconecta la placa y selecciona el puerto apropiado.

## 9 | Sube el *sketch* a la placa

Ahora simplemente pulsa sobre el botón "Upload" en el Entorno Arduino. Espera unos pocos segundos - deberías ver parpadear los led RX y TX de la placa. Si el volcado del código es exitoso verás aparecer el mensaje "Done uploading" en la barra de estado. (*Aviso: Si tienes una placa Arduino Mini, NG, u otras placas, necesitarás presionar el botón de reseteo de la placa inmediatamente antes de presionar el botón "Upload" el Entorno de programación Arduino.*)



Unos pocos segundos después de finalizar el volcado del programa deberías ver cómo el led de la placa conectado al pin 13 (L) comienza a parpadear (con un color naranja). Si ocurre esto ¡enhorabuena! Ya tienes tu Arduino listo y funcionando.

Si tienes problemas, por favor, consulta nuestras sugerencias ante problemas. A partir de ahora también podrás encontrar interesante consultar:

- los ejemplos sobre el funcionamiento de distintos sensores y dispositivos.
- la sección referencia para conocer el lenguaje de programación Arduino.

Los textos de la guía "Como empezar con Arduino" están licenciados bajo Creative Commons Attribution-ShareAlike 3.0 License. El código fuente de los ejemplos en la guía está liberado como dominio público.

# ANEXO C: MANUAL DE PROGRAMACION ARDUINO

Los programas hechos con Arduino se dividen en tres partes principales: *estructura*, *valores* (variables y constantes), y *funciones*. El Lenguaje de programación Arduino se basa en C/C++.

## Estructura

- `setup()` (*inicialización*)
- `loop()` (*bucle*)

### Estructuras de control

- `if` (*comparador si-entonces*)
- `if...else` (*comparador si...sino*)
- `for` (*bucle con contador*)
- `switch case` (*comparador múltiple*)
- `while` (*bucle por comparación booleana*)
- `do... while` (*bucle por comparación booleana*)
- `break` (*salida de bloque de código*)
- `continue` (*continuación en bloque de código*)
- `return` (*devuelve valor a programa*)

### Sintaxis

- `;` (punto y coma)
- `{ }` (llaves)
- `//` (comentarios en una línea)
- `/* */` (comentarios en múltiples líneas)

### Operadores Aritméticos

- `=` (asignación)
- `+` (suma)
- `-` (resta)
- `*` (multiplicación)
- `/` (división)
- `%` (resto)

### Operadores Comparativos

- `==` (igual a)
- `!=` (distinto de)
- `<` (menor que)
- `>` (mayor que)
- `<=` (menor o igual que)
- `>=` (mayor o igual que)

## Operadores Booleanos

- && (y)
- || (o)
- ! (negación)

## Operadores de Composición

- ++ (incrementa)
- -- (decrementa)
- += (composición suma)
- -= (composición resta)
- \*= (composición multiplicación)
- /= (composición división)

## Variables

### Constantes

- HIGH | LOW
- INPUT | OUTPUT
- true | false
- Constantes Numéricas

### Tipos de Datos

- boolean (*booleano*)
- char (*carácter*)
- byte
- int (*entero*)
- unsigned int (*entero sin signo*)
- long (*entero 32b*)
- unsigned long (*entero 32b sin signo*)
- float (*en coma flotante*)
- double (*en coma flotante de 32b*)
- string (*cadena de caracteres*)
- array (*cadena*)
- void (*vacío*)

### Conversión

- char()
- byte()
- int()
- long()
- float()

# Funciones

## E/S Digitales

- pinMode()
- digitalWrite()
- digitalRead()

## E/S Analógicas

- analogRead()
- analogWrite() - *PWM (modulación por ancho de pulso)*

## E/S Avanzadas

- tone()
- noTone()
- shiftOut()
- pulseIn()

## Tiempo

- millis()
- micros()
- delay()
- delayMicroseconds()
- hour()
- minute()
- second()
- day()
- weekday()
- month()
- year()

## Matemáticas

- min() (*mínimo*)
- max() (*máximo*)
- abs() (*valor absoluto*)
- constrain() (*limita*)
- map() (*cambia valor de rango*)
- pow() (*eleva a un número*)
- sq() (*eleva al cuadrado*)
- sqrt() (*raíz cuadrada*)

## Trigonometría

- `sin()` (*seno*)
- `cos()` (*coseno*)
- `tan()` (*tangente*)

## Números Aleatorios

- `randomSeed()`
- `random()`

## Comunicación

- `Serial`

Existen una gran cantidad de funciones, y cuando queremos utilizar las funciones recogidas en estas debemos llamar a la librería correspondiente. Existen algunas librerías que vienen por defecto con el programa de Arduino cuando te los descargas pero otras debes descargarlas cargarlas en el programa y reiniciarlo para poder utilizarlas como esta descrito en el anexo C.



# 1. ESTRUCTURA

## setup()

La función setup() se establece cuando se inicia un programa -sketch. Se emplea para iniciar variables, establecer el estado de los pins, inicializar librerías, etc. Esta función se ejecutará una única vez después de que se conecte la placa Arduino a la fuente de alimentación, o cuando se pulse el botón de reinicio de la placa.

### Ejemplo

```
int buttonPin = 3;
void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop()
{
  // ...
}
```

## Loop()

Luego de crear la función setup(), la cual inicializa y prepara los valores iniciales, la función loop() hace justamente lo que su nombre sugiere, por lo tanto se ejecuta consecutivamente, permitiéndole al programa variar y responder. Úsala para controlar de forma activa la placa Arduino.

### Ejemplo

```
int buttonPin = 3;

// setup inicializa la comunicación serial y el buttonPin
void setup()
{
  beginSerial(9600);
  pinMode(buttonPin, INPUT);
}

// loop obtiene el estado del pin del botón cada vez,
// y de estar presionado, lo comunica por serial.
void loop()
{
  if (digitalRead(buttonPin) == HIGH)
    serialWrite('H');
  else
    serialWrite('L');

  delay(1000);
}
```

# ESTRUCTURAS DE CONTROL

## **if (condicional) y ==, !=, <, > (operadores de comparación)**

**if**, el cual puede ser usado en conjunto con uno o más operadores de comparación, comprueba si cierta condición se cumple, por ejemplo, si un *input* posee un valor mayor a cierto número. El formato para una comprobación *if* es el siguiente:

```
if (algunaVariable > 50)
{
    // hacer algo aquí.
}
```

Este programa comprueba si la variable *algunaVariable* es mayor a 50. Si lo es, el programa toma una acción particular. Dicho de otra forma, si la declaración escrita dentro de los paréntesis es verdadera (*true*), el código dentro de las llaves se ejecutará. Sino, el programa ignora dicho código.

Las llaves pueden ser omitidas luego de una declaración *if*. De hacer esto, la siguiente línea (definida por el punto y coma) será la única afectada por la condición.

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
    digitalWrite(LEDpin1, HIGH);
    digitalWrite(LEDpin2, HIGH);
}

// todos los ejemplos son correctos.
```

Las declaraciones a evaluar dentro de los paréntesis, requieren el uso de uno o más operadores:

### **Operadores de Comparación:**

```
x == y (x es igual a y)
x != y (x no es igual a y)
x < y (x es menor a y)
x > y (x es mayor a y)
x <= y (x es menor o igual a y)
x >= y (x es mayor o igual a y)
```

## Atención:

Ten cuidado de no usar un signo de igual solo (ej. `if (x = 10)`). Un signo de igual solo es el operador que indica la asignación de un valor, y va a asignar `10` a `x`. En su lugar usa el signo de igual doble (ej. `if (x == 10)`), el cual es el operador de comparación, y comprueba si `x` equivale a `10` o no. El último ejemplo sólo da `true` si `x` equivale a `10`, pero el ejemplo anterior (con un sólo símbolo `=`) dará siempre `TRUE`.

Esto es porque C evalúa la declaración `if (x=10)` de la siguiente manera: `10` es asignado a `x` (Recuerda que un signo `=` solo, es el operador de asignación), por lo tanto `x` ahora contiene `10`. Entonces el condicional `if` evalúa `10`, el cual siempre resulta `TRUE`, debido a que cualquier valor numérico mayor a `0` es evaluado como `TRUE`. Consecuentemente, `if (x = 10)` siempre será evaluado como `TRUE`, lo cual no es el resultado deseado cuando se usa una declaración `if`. Adicionalmente, la variable `x` será definida en `10`, lo que tampoco es una acción deseada.

`if` también puede ser parte de una estructura de control de ramificación usando la construcción `if...else`.

## if / else

`if/else` permite mayor control sobre el flujo del código que la declaración `if` básica, por permitir agrupar múltiples comprobaciones. Por ejemplo, un `input` análogo podría ser comprobado, y tomarse una acción si el valor del `input` es menor a `500`, y otra acción si es igual o mayor a `500`. El código se vería así:

```
if (pinCincoInput < 500)
{
    // acción A
}
else
{
    // acción B
}
```

`else` puede proceder a una comprobación `if`, de esta forma, se pueden realizar múltiples comprobaciones en una misma estructura de condiciones. Cada comprobación procederá a la siguiente, sólo cuando su propio resultado sea `FALSE`. Cuando el resultado sea `TRUE`, su bloque de código contenido, será ejecutado, y el programa esquivará las siguientes comprobaciones hasta el final de la estructura de comprobaciones. Si ninguna comprobación devuelve valor `TRUE`, el `else` será ejecutado, y de no haber ninguno declarado, simplemente no sucede nada.

Entonces un bloque **else if** puede ser usado con o sin **else** al final. La cantidad de declaraciones **else if**, y sus ramificaciones son ilimitadas.

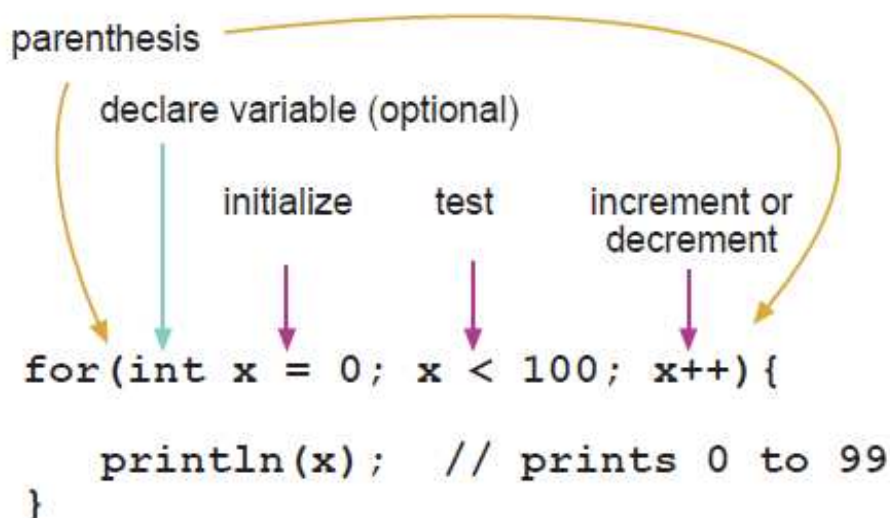
```
if (pinCincoInput < 500)
{
    // ejecutar A
}
else if (pinCincoInput >= 1000)
{
    // ejecutar B
}
else
{
    // ejecutar C
}
```

## Declaracion FOR

### Descripción

La declaración **for** es usada para repetir un bloque encerrado entre llaves. Un incremento de un contador es usado, normalmente, para aumentar y terminar con el bucle. La estructura **for** es muy útil para la mayoría de las operaciones repetitivas, y habitualmente se usa para operaciones con vectores, para operar sobre conjuntos de datos/pines. El bucle **for** tiene tres partes o argumentos en su inicialización:

```
for (initialization; condition; increment) {
//función(es);
}
```



La **initialization**, o inicialización, se produce sólo la primera vez. Cada vez que se va a repetir el bucle, se revisa la **condition**, o condición: si es cierta, el bloque de funciones (y el

incremento del contador) se ejecutan, y la condición vuelve a ser comprobada de nuevo. Si la condición es falsa, el bucle termina.

### Ejemplo

```
// Variar la intensidad de un LED usando un salida PWM
int PWMpin = 10; // En el pin 10 hay un LED en serie con una resistencia de
470 ohmios

void setup()
{
  // no es necesario nada aquí
}

void loop()
{
  for (int i=0; i <= 255; i++){
    analogWrite(PWMpin, i);
    delay(10);
  }
}
```

### Consejos de programación

El bucle **for**, en C, es mucho más flexible que otros bucles **for** en otros lenguajes, incluyendo BASIC. Cualquiera (o todos) los parámetros pueden ser omitidos, sin embargo los puntos y coma (;) son obligatorios. También las inicialización, condición e incremento pueden ser cualquier declaración en C válida, con variables independientes, y podemos usar cualquier tipo de variable, incluidos los float. Estos tipos de declaración **for** poco usuales pueden proporcionar una solución válida a algunos problemas de programación raros.

Por ejemplo, usando la multiplicación en el parámetro de incremento, podemos generar una progresión logarítmica.

```
for(int x = 2; x < 100; x = x * 1.5){
  println(x);
}
```

Este código generará: 2,3,4,6,9,13,19,28,42,63,94

Otro ejemplo es hacer apagarse/encenderse un LED poco a poco:

```
void loop()
{
  int x = 1;
  for (int i = 0; i > -1; i = i + x){
    analogWrite(PWMpin, i);
    if (i = 255) x = -1; // cambia de signo para apagarlo
    delay(10);
  }
}
```

## Sentencia switch / case

Como las sentencias **if**, **switch...case** controla el flujo de programas permitiendo a los programadores especificar diferentes códigos que deberían ser ejecutados en función de varias condiciones. En particular, una sentencia switch compara el valor de una variable con el valor especificado en las sentencias case. Cuando se encuentra una sentencia case cuyo valor coincide con dicha variable, el código de esa sentencia se ejecuta.

La palabra clave **break** sale de la sentencia switch, y es usada típicamente al final de cada case. Si una sentencia break, la sentencia switch continuaría ejecutando las siguientes expresiones ("falling-through") hasta encontrar un break, o hasta llegar al final de la sentencia switch.

### Ejemplo

```
switch (var) {
  case 1:
    //hacer algo cuando sea igual a 1
    break;
  case 2:
    //hacer algo cuando sea igual a 2
    break;
  default:
    // si nada coincide, ejecuta el "default"
    // el "default" es opcional
}
```

### Sintaxis

```
switch (var) {
  case etiqueta:
    // sentencias
    break;
  case etiqueta:
    // sentencias
    break;
  default:
    // sentencias
}
```

### Parámetros

var: la variable cuyo valor comparas con los varios "case"

etiqueta: un valor para comparar con la variable

# Bucles while

## Descripción

Los bucles **while** se ejecutan continuamente, hasta que la expresión de dentro del paréntesis, (), pasa a ser falsa. Algo debe modificar la variable comprobada, el bucle **while** nunca terminará. Lo que modifique la variable puede estar en el código, como una variable que se incrementa, o ser una condición externa, como el valor que da un sensor.

## Sintaxis

```
while(expresion){  
    // sentencia(s)  
}
```

## Parámetros

expresion - una sentencia C (booleana) que da como valor verdadero (true) o falso (false)

## Ejemplo

```
var = 0;  
while(var < 200){  
    // haz algo repetitivo 200 veces  
    var++;  
}
```

## do - while

El bucle "do" trabaja de la misma manera que el bucle "while", con la excepción de que la condición se comprueba al final del bucle, por lo que este bucle se ejecuta "siempre" al menos una vez.

```
do  
{  
    // bloque de instrucciones  
} while (condición);
```

## Ejemplo:

```
do  
{  
    delay(50);           // espera a que los sensores se estabilicen  
    x = readSensors(); // comprueba los sensores  
} while (x < 100);     //si se cumple la condición se repite el bucle
```

## break

**break** es usado para salir de los bucles **do**, **for**, o **while**, pasando por alto la condición normal del bucle. Es usado también para salir de una estructura de control **switch**.

### Ejemplo

```
for (x = 0; x < 255; x ++)  
{  
    digitalWrite(PWMPin, x);  
    sens = analogRead(sensorPin);  
    if (sens > threshold){          // bail out on sensor detect  
        x = 0;  
        break;    // sale del bucle for.  
    }  
    delay(50);  
}
```

## Continue

La sentencia **continue** omite el resto de iteraciones de un bucle (**do**, **for**, o **while**). Continúa saltando a la condición de bucle y procediendo con la siguiente iteración.

### Ejemplo

```
for (x = 0; x < 255; x ++)  
{  
    if (x > 40 && x < 120){          // crea un salto en estos valores  
        continue;  
    }  
  
    digitalWrite(PWMPin, x);  
    delay(50);  
}
```

## return

Termina una función y devuelve un valor a la función que la llama. Puede no devolver nada.

### Sintaxis

```
return;  
return valor; // ambas formas son correctas
```

### Parámetros

valor: cualquier variable o tipo constante



## Ejemplos

Una función que compara la entrada de un sensor a un umbral

```
int comprobarSensor(){
    if (analogRead(0) > 400) {
        return 1;
    }
    else{
        return 0;
    }
}
```

La palabra clave *return* es útil para depurar una sección de código sin tener que comentar una gran cantidad de líneas de código posiblemente incorrecto.

```
void loop(){
    // código magnífico a comprobar aquí

    return;

    // el resto del programa del que se desconfía
    // que nunca será ejecutado por estar detrás de return
}
```

## SINTAXIS

### **; punto y coma**

Utilizado para terminar una declaración.

#### **Ejemplo**

```
int a = 13;
```

#### **Truco**

Olvidarse de terminar una línea con el punto y coma hará que se lance un error de compilación. El texto del error puede ser obvio y referirse concretamente a la ausencia del punto y coma, pero puede no hacerlo. Si aparece un error de compilación impenetrable y aparentemente incoherente, una de las primeras cosas a hacer es comprobar que no falta ningún punto y coma en la vecindad de la línea en la que el compilador da el error.

## { Llaves

Las Llaves son un parte importante del lenguaje de programación C. Se utilizan en diferentes construcciones (ver ejemplos al final), esto a veces puede ser confuso para los principiantes.

Una llave de apertura "{" siempre debe ir seguida de una llave de cierre "}". Esta es una condición a la que se suele referir como llaves emparejadas. El IDE (Entorno Integrado de Desarrollo) Arduino incluye una característica para comprobar si las llaves están emparejadas. Sólo tienes que seleccionar una Llave o incluso hacer click en el punto de inserción que sigue inmediatamente a una llave, y su compañera lógica será seleccionada.

En la actualidad esta característica tiene un pequeño fallo, el IDE encuentra a menudo (incorrectamente), llaves en el texto que pueden estar situadas dentro de comentarios.

Los programadores principiantes y los programadores que llegan a C desde el lenguaje BASIC a menudo encuentran dificultades o grandes confusiones usando llaves. Después de todo, las llaves reemplazan el RETURN en una subrutina(función), el ENDIF en un condicional y el NEXT en un loop FOR.

Dado que el uso de las llaves es tan variado, es una buena práctica de programación escribir la llave de cierre inmediatamente después de escribir la llave de apertura, cuando se inserta una construcción que requiere llaves. Después insertar algunos saltos de línea (líneas en blanco) entre las llaves y comenzar a insertar sentencias. De esta forma tus llaves y su posición, nunca llegarán a estar desemparejadas.

Llaves desemparejadas a menudo pueden conducir a errores de compilación misteriosos y difíciles de comprender, que pueden ser complicados de rastrear en un programa largo. Debido a sus variados usos, las llaves también son increíblemente importantes para la sintaxis de un programa, el movimiento de una llave una o dos líneas, a menudo afecta de manera dramática el significado de un programa.

### Usos principales de las Llaves

#### Funciones

```
void myfunction(tipodato argumento){
    sentencia(s)
}
```

#### Loops

```
while (expresión booleana)
```

```

{
    sentencia(s)
}

do
{
    sentencia(s)
} while (expresión booleana);

for (inicialización; condición final; expresión incremento)
{
    sentencia(s)
}

```

## Sentencias Condicionales

```

if (expresión booleana)
{
    sentencia(s)
}

else if (expresión booleana)
{
    sentencia(s)
}
else
{
    sentencia(s)
}

```

## Comentarios

Los comentarios son líneas en el programa para aclararte o a otros sobre el funcionamiento del programa. Estas líneas son ignoradas por el compilador y no se exportan al procesador. No ocupan por tanto espacio en el Chip Atmega.

El único propósito de los comentarios es que entiendas o entiendan (o recuerdes) cual es el funcionamiento de tu programa. Existen dos maneras distintas de marcar una línea como comentario:

### Ejemplo

```

x = 5; // Esto es una línea simple de comentario. Todo lo que va después
de la doble barra es un comentario
    // Hasta el final de la línea

/* Esto es un comentario multilínea - úsalo para comentar bloques enteros
de código

if (gwb == 0){ // Una línea de comentario sencilla puede usarse dentro de
un comentario multilínea
x = 3; // pero no otro comentario multilínea- esto no es válido */
}
// No olvides cerrar el comentario multilínea*/

```

## Consejo

Cuando estás experimentando con código, "comentar" partes del programa es una útil de eliminar líneas que puedan contener errores. Así dejamos las líneas en el código, pero como comentario, así que serán ignoradas por el compilador. Es especialmente útil cuando estamos intentando localizar el problema, o cuando el compilador rechaza el programa.

# OPERADORES ARITMETICOS

## = operador de asignación (un solo símbolo de "igual")

Guarda el valor en la derecha del símbolo "=" dentro de la variable a la izquierda del símbolo "=".

El signo de igualdad "=" en el lenguaje de programación C se llama el operador de asignación. Tiene un significado diferente que en la clase de álgebra en el que se indica una ecuación o igualdad. El operador de asignación le dice al microcontrolador que evalúe cualquier valor o expresión en el lado derecho del signo igual, y lo almacene en la variable a la izquierda del signo igual.

## Ejemplo

```
int sensVal; // declara una variable int llamada sensVal
senVal = analogRead(0); // guarda el valor (digitalizado) del
voltaje de entrada del pin analógico 0 en SensVal
```

## Sugerencias de programación

La variable en el lado izquierdo del operador de asignación (signo "=") tiene que ser capaz de mantener el valor almacenado en ella. Si no es suficientemente grande para contenerlo, el valor almacenado en la variable será incorrecto.

No confunda el operador de asignación [ = ] (un solo signo igual) con el operador de comparación [ == ] (un signo igual doble), que evalúa si dos expresiones son iguales.

# Suma, Resta, Multiplicación y División

## Descripción

Estos operadores devuelven la suma, diferencia, producto o cociente (respectivamente) de los dos operandos. La operación se lleva a cabo utilizando el tipo de datos de los operandos, por lo que, por ejemplo, 9 / 4 resulta 2 desde 9 y 4 que son enteros int. Esto también significa que la operación puede desbordarse si el resultado es mayor que el que se puede almacenar en el

tipo de datos (por ejemplo, la suma de 1 a un int con el valor de 32.767 resulta -32.768). Si los operandos son de tipos diferentes, se utiliza el tipo del "más grande" para el cálculo.

Si uno de los números (operandos) es del tipo **float** o del tipo **double**, se usará coma flotante para el cálculo.

### Ejemplos

```
y = y + 3;  
x = x - 7;  
i = j * 6;  
r = r / 5;
```

### Sintaxis

```
result = value1 + value2;  
result = value1 - value2;  
result = value1 * value2;  
result = value1 / value2;
```

### Parámetros:

value1: cualquier variable o constante

value2: cualquier variable o constante

### Sugerencias de programación:

Debes saber que las integer constants por defecto son int, así que algunos cálculos con constantes pueden provocar desbordamiento (p.e.  $60 * 1000$  devolverá un resultado negativo)

Elige los tamaños de variable que sean suficientemente grandes como para alojar el resultado de tus calculos.

Debes saber en que punto tu variable se desbordará en su máximo, y que esto también ocurre en su mínimo. p.e.  $(0 - 1)$  o también  $(0 - - 32768)$

Para cálculos matemáticos que requieren fracciones, usa variables float, pero ten en cuenta los inconvenientes: gran tamaño, velocidades bajas de cálculo

Usa el operador de conversión (casting). Por ejemplo: `(int)myFloat` para convertir el tipo de una variable en el momento.

## % (módulo)

### Descripción

Calcula el resto de la división entre dos enteros. Es útil para mantener una variable dentro de un rango particular (por ejemplo el tamaño de un array).

### Sintaxis

resultado = dividendo % divisor

### Parametros

dividendo: el número que se va a dividir

divisor: el número por el que se va a dividir

### Devuelve

el resto de la división

### Ejemplo

```
x = 7 % 5;    // x ahora contiene 2
x = 9 % 5;    // x ahora contiene 4
x = 5 % 5;    // x ahora contiene 0
x = 4 % 5;    // x ahora contiene 4
```

### Código de ejemplo

```
/* actualiza un valor en el array cada vez que se pasa por el bucle */

int valores[10];
int i = 0;

void setup() {}

void loop()
{
  valores[i] = analogRead(0);
  i = (i + 1) % 10;    // el operador módulo prevalece sobre la variable
}
```

### Nota

El operador modulo no funciona con datos en coma flotante (float)

## **OPERADORES BOOLEANOS**

Se pueden usar dentro de operaciones condicionales o en una sentencia if.

### **&& (AND lógico)**

Verdadero sólo si ambos operadores son Verdadero, por ejemplo:

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH) { // lee dos
  pulsadores
  // ...
}
```

Es Verdadero sólo si ambas entradas estás activadas, es decir, en estado HIGH.

### **|| (OR lógico)**

Verdadero si alguno de los dos operadores es Verdadero, por ejemplo:

```
if (x > 0 || y > 0) {
  // ...
}
```

Es Verdadero si alguno de los valores x ó y es mayor que 0.

### **! (NOT)**

Verdadero si el operador es Falso, por ejemplo:

```
if (!x) {
  // ...
}
```

Es Verdadero si el valor de x es Falso (p. e. si x es igual a 0).

### **Atención.**

Asegúrate de no confundir el operador AND booleano && (doble signo &) con el operador AND para bits & (un solo signo &).

De la misma manera no hay que confundir el operador OR booleano || (doble barra vertical) con el operador OR para bits | (una sola barra vertical).

El operador NOT de bits ~ (tilde) tiene una apariencia muy diferente del operador NOT booleano ! (exclamación o "bang", como lo llaman algunos programadores), pero debes asegurarte de usar cada uno de ellos dentro del contexto adecuado.

### **Ejemplos.**

```
if (a >= 10 && a <= 20){} // Verdadero sólo si el valor de a está entre 10 y 20
```

## **OPERADORES DE COMPOSICION**

**++ (incremento) / -- (disminución)**

### **Descripción**

Incrementa o disminuye una variable

### **Sintaxis**

```
x++; // incrementa x en uno y regresa el valor anterior de x
++x; // incrementa x en uno y regresa el nuevo valor de x

x-- ; // disminuye x en uno y y regresa el valor anterior de x
--x ; // disminuye n uno y regresa el nuevo valor de x
```

### **Parámetros**

x: un entero o long (puede aceptar sin signo)

### **Devuelve**

El valor original o el resultante del incremento o disminución de la variable.

### **Ejemplos**

```
x = 2;
y = ++x; // x ahora guarda 3, y guarda 3
y = x--; // x guarda 2 de nuevo, y sigue guardando 3
```

**+= , -= , \*= , /=**

### **Descripción**

Realiza una operación matemática con una variables con respecto a otra variable o una constante. El operador += (y los demás operadores) son una forma simplificada de la sintaxis completa, tal y como se muestra más abajo.

### **Sintaxis**

```
x += y; // equivalente a la expresión x = x + y;
x -= y; // equivalente a la expresión x = x - y;
x *= y; // equivalente a la expresión x = x * y;
x /= y; // equivalente a la expresión x = x / y;
```

### **Parámetros**

x: cualquier tipo de variable



y: cualquier tipo de variable o constante

## Ejemplos

```
x = 2;  
x += 4;      // x ahora es 6  
x -= 3;      // x ahora es 3  
x *= 10;     // x ahora es 30  
x /= 2;      // x ahora es 15
```

## 2.VARIABLES

### CONSTANTES

Las constantes variables que vienen predefinidas en el lenguaje de Arduino. Se usan para facilitar la lectura de los programas. Clasificamos las constantes en grupos.

#### **Las que definen niveles lógicos, verdadero (true) y falso (false) (Constantes Booleanas)**

Existen dos constantes para representar si algo es cierto o falso en Arduino: **true**, y **false**.

##### **false**

false es el más sencillo de definir. false se define como 0 (cero).

##### **true**

true se define la mayoría de las veces como 1, lo cual es cierto, pero tiene una definición más amplia. Cualquier entero que es *no-cero* es TRUE, en un sentido Booleano. Así, en un sentido Booleano, -1, 2 y -200 son todos true.

Ten en cuenta que las constantes *true* y *false* se escriben en minúsculas, al contrario que HIGH, LOW, INPUT, y OUTPUT.

#### **Las que definen el nivel de los pines, nivel alto (HIGH) y nivel bajo (LOW)**

Cuando leemos o escribimos en un pin digital, existen sólo dos valores que podemos obtener o asignar : **HIGH** y **LOW**.

##### **HIGH**

El significado de HIGH (en referencia a un pin) depende de si el pin está configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin se configura como entrada (INPUT) usando pinMode, y se lee con digitalRead, el microcontrolador nos retornará HIGH si en el pin hay 3 voltios o más.

Un pin puede ser configurado como entrada (INPUT) usando pinMode, y después establecerlo a HIGH con digitalWrite, esto conectará el pin a 5 Voltios a través de una resistencia interna de 20K, resistencia pull-up , la cual establecerá el pin al estado de lectura HIGH a menos que la conectemos a una señal LOW a través de un circuito externo.

Cuando un pin se configura como salida (OUTPUT) con `pinMode`, y se establece a HIGH con `digitalWrite`, el pin tiene 5V. En este estado puede usarse como *fuentes* de corriente, e.j. Luz y LED que se conectan a través de resistencias en serie a masa (tierra), o a otro pin configurado como salida y establecido a LOW.

## LOW

El significado de LOW difiere también según esté configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin está configurado como entrada (INPUT) con `pinMode`, y se lee con `digitalRead`, el microcontrolador retornará LOW si el voltaje presente en el pin es de 2V o menor.

Cuando un pin es configurado como salida (OUTPUT) con `pinMode`, y establecido a LOW con `digitalWrite`, el pin tiene 0 voltios. En este estado puede *meter* corriente, por ejemplo, Luz y LED que se conectan a través de resistencias en serie a +5 voltios, o a otro pin configurado como salida, y establecido a HIGH.

## Las que definen los pines digitales, INPUT y OUTPUT

Los pines digitales pueden ser usados como **entrada (INPUT)** o como **salida (OUTPUT)**. Cambiando un pin de INPUT a OUTPUT con `pinMode()` el comportamiento eléctrico del pin cambia drásticamente.

### Pins configurados como entradas

Los pins de Arduino (Atmega) configurados como **INPUT** con `pinMode()` se dice que se encuentran en un estado de alta impedancia. Una forma de explicar esto es que un pin configurado como entrada se le aplica una muy baja demanda, es decir una resistencia en serie de 100 Megohms. Esto lo hace muy útil para leer un sensor, pero no para alimentar un LED.

### Pins configurados como salidas

Los pins configurados como **salida (OUTPUT)** con `pinMode()` se dice que están en estado de baja impedancia. Esto implica que pueden proporcionar una sustancial cantidad de corriente a otros circuitos. Los pins de Atmega pueden alimentar (proveer de corriente positiva) o meter (proveer de masa) hasta 40 mA (miliamperios) de corriente a otros dispositivos/circuitos. Esto lo hace muy útil para alimentar LED's pero inservible para leer sensores. Los pins configurados como salida pueden deteriorarse o romperse si ocurre un cortocircuito hacia los 5V o 0V. La cantidad de corriente que puede proveer un pin del Atmega no es suficiente para la mayoría de los relés o motores, y es necesario añadir circuitería extra.

## Constantes numéricas o Integer

*Constantes Integer* son números utilizados directamente en un *sketch*, como 123. Por defecto, éstos números son tratados como `int`, pero puedes cambiarlo con las letras *U* y *L* (ver abajo). Normalmente, las constantes *integer* son tratadas como enteros base 10 (decimales), pero se puede utilizar notación especial (formateadores) para ingresar números en otras bases.

Base	Ejemplo	Formateador	Comentario
10 (decimal)	123	Ninguno.	
2 (binario)	B1111011	Antecede "B"	Sólo funciona con valores de 8 bits (0 to 255). Caracteres 0-1 válidos.
8 (octal)	0173	Antecede "0"	Caracteres 0-7 válidos.
16 (hexadecimal)	0x7B	Antecede "0x"	Caracteres 0-9, A-F, a-f válidos.

**Decimal** es base 10. Esta es la matemática de sentido común con que se conocen. Para constantes sin otros prefijos, se asume el formato decimal.

Ejemplo:

```
101 // igual a 101 decimal ((1 * 10^2) + (0 * 10^1) + 1)
```

**Binary** es base dos. Sólo caracteres 0 y 1 son válidos.

Ejemplo:

```
B101 // igual a 5 decimal ((1 * 2^2) + (0 * 2^1) + 1)
```

El formateador binario sólo funciona en *bytes* (8 bits) entre 0 (B0) y 255 (B11111111). Si resulta conveniente ingresar un entero (*int*, 16 bits) de forma binaria, puedes hacer un procedimiento de dos pasos, como a continuación:

```
valorInt = (B11001100 * 256) + B10101010; // B11001100 es el 'byte' alto.
```

**Octal** es base ocho. Sólo caracteres de 0 hasta 7 son válidos. Los valores Octales son indicados por el prefijo "0"

Ejemplo:

```
0101 // igual a 65 decimal ((1 * 8^2) + (0 * 8^1) + 1)
```

## Atención

Es posible generar un *bug* o error difícil de encontrar al incluir (de forma involuntaria) un cero antecediendo una constante, logrando así que el compilador interprete tu constante como octal.

**Hexadecimal (ó hex)** es base dieciséis. Los caracteres válidos son del 0 al 9, y las letras desde la *A* hasta la *F*; *A* tiene el valor de 10, *B* es 11, *C* es 12, *D* es 13, *E* es 14, y la *F*, como ya habrás adivinado, es 15. Los valores hexadecimales se indican con el prefijo "0x". Nota que los valores de la *A* a la *F*, pueden ser escritos en mayúscula o minúscula.

## Ejemplo:

```
0x101 // igual a 257 decimal ((1 * 16^2) + (0 * 16^1) + 1)
```

## Formateadores *U & L*

Por defecto, una constante en enteros es tratada como `int` con las limitaciones concomitantes en los valores. Para especificar una constante en enteros con otro tipo de datos, continúa con:

- Una 'u' ó 'U' para forzar la constante a un formato de datos *unsigned*. Ejemplo: 33u
- Una 'l' ó 'L' para forzar la constante a un formato de datos *long*. Ejemplo: 100000L
- Un 'ul' ó 'UL' para forzar la constante a un formato de datos *unsigned long*. Ejemplo: 32767ul

# TIPOS DE DATOS

## booleanos

Un **booleano** sólo puede tomar dos valores, Verdadero o Falso. Cada booleano ocupa un único byte en la memoria.

## Ejemplo:

```
int LEDpin = 5; // LED en el pin 5
int switchPin = 13; // pulsador en el pin 13, su otra patilla conectada
en GND

boolean running = false; // crea la variable booleana running y le asisga
el valor Falso (false)

void setup()
{
    pinMode(LEDpin, OUTPUT);
```

```

    pinMode(switchPin, INPUT);
    digitalWrite(switchPin, HIGH);    // conecta la resistencia pull-up
    interna del pin 13
}

void loop()
{
    if (digitalRead(switchPin) == LOW)
    { // si el pulsador es accionado la pull-up mantiene el pin en estado
HIGH
        delay(100);                // retardo para impedir un rebote en
el pulsador
        running = !running;        // invierte el valor de la variable
running
        digitalWrite(LEDpin, running) // enciende el LED
    }
}

```

## char

### Descripción

Es un tipo de dato que ocupa un byte de memoria y almacena un valor de carácter. Los caracteres literales se escriben con comillas simples: 'A' (para varios caracteres -strings- utiliza dobles comillas "ABC").

De todas maneras los caracteres son almacenados como números. Puedes ver su codificado en la [tabla ASCII](#). Con esto podemos entender que es posible realizar cálculos aritméticos con los caracteres, en este caso se utiliza el valor ASCII del carácter (por ejemplo 'A' + 1 tiene el valor de 66, ya que el valor ASCII de la letra mayúscula A es 65). Mira [Serial.println](#) para tener mas información de como son traducidos a números los caracteres.

El tipo de datos char tiene signo. esto significa que codifica números desde -128 hasta 127. Para un dato de un byte (8bits), utiliza el tipo de dato "byte".

### Ejemplo

```

char miChar = 'A';
char miChar = 65;    // los dos son equivalentes

```

## byte

### Descripción

Un byte almacena un número sin signo de 8-bit, desde 0 hasta 255.

### Ejemplo

```

byte b = B10010; // "B" es el formateador binario (B10010 = 18
decimal)

```

# int

## Descripción

Integers (Números enteros) son el principal tipo de datos para almacenar números, y guardan valores de **2 bytes**. Esto produce un rango entre  $-32,768$  hasta  $32,767$  (valor mínimo de  $-2^{15}$  y un valor máximo de  $(2^{15}) - 1$ ).

Variables tipo **Int**, almacenan números negativos con una técnica llamada Complemento a dos. El *bit* más alto, a veces llamado como "*sign*" *bit*, indica que el número es negativo. Se invierte el valor de cada uno de los *bits*, es decir, se realiza el complemento a uno, y se suma 1 al número obtenido.

La placa Arduino, se encarga de tratar con números negativos por ti, para que las operaciones aritméticas trabajen de manera transparente y en la forma esperada.

## Ejemplo

```
int ledPin = 13;
```

## Sintaxis

```
int var = val;
```

- var - nombre de la variable *int*.

- val - valor asignado a dicha variable.

## Consejo

Cuando las variables son hechas para exceder su límite, éstas vuelven a su capacidad mínima, ésto sucede en ambas direcciones:

```
int x
x = -32,768;
x = x - 1; // x ahora contiene 32,767 - vuelve a empezar en
dirección contraria.
```

```
x = 32,767;
x = x + 1; // x ahora contiene -32,768 - vuelve a empezar.
```

# unsigned int

## Descripción

Los enteros sin firma (unsigned int) son los mismos enteros de modo que almacenan un valor de dos bytes. En lugar de almacenar números negativos, sólo almacenan valores positivos, generando un rango útil desde 0 a  $65,535$  ( $2^{16} - 1$ ).

La diferencia entre los enteros sin firma y los enteros (firmados), reside en que el bit más alto, a veces referenciado como el bit "firmado", es interpretado. En el tipo de dato `int` de Arduino (que es firmado), si el bit más alto es "1", el número es interpretado como un número negativo, y los otros 15 bits son interpretados con complemento a 2.

### Ejemplo

```
unsigned int ledPin = 13;
```

### Sintaxis

```
unsigned int var = val;
```

- var - el nombre de tu variable `unsigned int`

- val - el valor que asignas a esa variable

### Consejo de codificación

Cuando las variables sobrepasan su capacidad máxima dan la vuelta a su mínima capacidad. Ten en cuenta que esto sucede en ambas direcciones.

```
unsigned int x
x = 0;
x = x - 1;          // x ahora contiene 65535 - da la vuelta en dirección
negativa
x = x + 1;          // x ahora contiene 0 - da la vuelta
```

## long

### Descripción

Las variables de tipo Long son variables de tamaño extendido para almacenamiento de números, y 32 bits (4 bytes), desde -2,147,483,648 hasta 2,147,483,647.

### Ejemplo

```
long speedOfLight = 186000L;    // ver Constantes Integer para la
explicación de la 'L'
```

### Sintaxis

```
long var = val;
```

- var - nombre de la variable tipo Long
- val - valor asignado a la variable



# unsigned long

## Descripción

Las variable long sin firmar (unsigned long) son variables extendidas para almacenar números, y almacenar 32 bits (4 bytes). Por el contrario que las variables long estándar, las unsigned long no almacenan números negativos, haciendo que su rango sea de 0 a 4,294,967,295 ( $2^{32} - 1$ ).

## Ejemplo

```
unsigned long tiempo;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print("Tiempo: ");
  tiempo = millis();
  //imprime el tiempo desde el inicio del programa
  Serial.println(tiempo);
  // espera un segundo para no enviar cantidade masivas de datos
  delay(1000);
}
```

## Sintáxis

```
unsigned long var = val;
```

- var - el nombre de tu variable long
- val - el valor que asignas a tu variable long

# float

## Descripción

El tipo variable para los números en coma flotante (número decimal). Estos números son usados, habitualmente, para aproximar valores analógicos y continuos, debido a que ofrecen una mayor resolución que los enteros. Las variables tipo **float** tienen el valor máximo 3.4028235E+38, y como mínimo pueden alcanzar el -3.4028235E+38. Ocupan 4bytes (32bits).

Los **floats** tienen una precisión de 6 o 7 dígitos decimales. Esto significa el número total de dígitos, no el número a la derecha de la coma decimal. Al contrario que en otras plataformas, donde tu podrías obtener mayor precisión usando una variable tipo double (por ejemplo, por encima de 15 dígitos), en Arduino los double tienen el mismo tamaño que los float.

Los números en coma flotante no son exactos, y muchos proporcionan falsos resultados cuando son comparados. Por ejemplo,  $6.0 / 3.0$  puede no ser igual a  $2.0$ . Debes comprobar que el valor absoluto de la diferencia entre los números pertenezca a un rango pequeño.

La matemática en coma flotante es mucho más lenta que la matemática de enteros para realizar operaciones, por lo que deberías evitarla si, por ejemplo, un bucle tiene que ejecutarse a la máxima velocidad para funciones con temporizaciones precisas. Los programadores normalmente suelen asignar unas longitudes para convertir las operaciones de coma flotante en cálculos con enteros, para aumentar la velocidad.

## Ejemplos

```
float myfloat;  
float sensorCalbrate = 1.117;
```

## Sintaxis

```
float var = val;
```

- **var** - el nombre de la variable tipo flota
- **val** - el valor que le asignas a esa variable

## Código de ejemplo

```
int x;  
int y;  
float z;  
  
x = 1;  
y = x / 2; // y ahora contiene 0, la parte entera de la operación  
z = (float)x / 2.0; // z ahora contiene 0.5 (se debe usar 2.0, en lugar  
de 2)
```

# double

## Descripción

Número en coma flotante de doble precisión. Ocupa 4 bytes.

La implementación "double" en Arduino es exactamente lo mismo que la FLOAT, sin ganar nada de precisión.

## Consejo

Los usuarios que porten código de otras fuentes y que incluyan variable tipo double deberían examinar el código para ver si la precisión necesaria es diferente a la que se puede lograr en Arduino.

# string

## Descripción

Los strings se representan como arrays de caracteres (tipo char) que terminan con el caracter NULL.

## Ejemplos

Todas las siguientes son declaraciones válidas de strings.

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

## Posibilidades de declaración de strings

- Declarar un array de caracteres sin inicializarlo como en Str1
- Declarar un array de caracteres (con un caracter extra) y el compilador añadirá el caracter NULL requerido, como en STR2

Explicitar el caracter NULL, Str3

- Inicializar con un string constante entre comillas dobles; el compilador medirá el tamaño del array para ajustar el string constante y caracter NULL para finalizar, Str4
- Inicializar el array con un tamaño explícito y un string constante, Str5
- Inicializar el array, dejando un espacio extra para un string más largo, Str6

## Terminación NULL

Generalmente, los strings se finalizan con un caracter NULL (código ASCII 0). Esto permite a funciones (como Serial.print()) establecer dónde está el final del string. De otra forma, seguiría leyendo los siguientes bytes de la memoria que no forman parte del string.

Esto significa que tu string necesita tener espacio para un caracter más del texto que quieres que contenga. Esto es por lo que Str2 y Str5 necesitan 8 caracteres, incluso aunque "arduino" tenga sólo 7 - la última posición es automáticamente completada con un caracter NULL. Str4 será automáticamente dimensionada a 8 caracteres, uno para el NULL extra. En Str3, hemos incluido nosotros mismos el caracter NULL (escrito como '\0').

Ten en cuenta que es posible tener un string sin un caracter NULL al final (por ejemplo, si tú has especificado la longitud de Str2 como 7 en lugar de 8). Esto romperá la mayoría de funciones que usen strings, por lo que no deberías hacerlo intencionadamente. Por lo tanto, si

detectas algún comportamiento extraño (operando con los caracteres, no con el string), este podría ser el problema.

## Comillas simples o dobles

Los string siempre se definen entre comillas dobles ("Abc") y los caracteres siempre se definen dentro de comillas simples ('A').

## Envolviendo string largos

Puedes envolver strings largos de ésta manera:

```
char miString[] = "Esta es la primera linea"  
" esta es la segunda linea"  
" etcetera";
```

## Arrays de Strings

A menudo es conveniente, al trabajar con grandes cantidades de texto, como proyectos con displays LCD, configurar un array de strings. Como los strings son en sí mismo arrays, esto es un ejemplo real de un array bidimensional.

En el código de abajo, el asterisco después del tipo de dato char "char\*" indica que es un array de "punteros". Todos los nombres de arrays son punteros, por lo que esto es necesario para crear un array de arrays. Los punteros son una de las partes más exóticas de C como para que los principiantes puedan llegar a entenderlas, pero no es necesario entender punteros en detalle para usarlos con efectividad.

## Ejemplo

```
char* miStrings[]={"Este es el string 1", "Este es el string 2", "Este es  
el string 3",  
"Este es el string 4", "Este es el string 5", "Este es el string 6"};  
  
void setup(){  
  Serial.begin(9600);  
}  
  
void loop(){  
  for (int i = 0; i < 6; i++){  
    Serial.println(miStrings[i]);  
    delay(500);  
  }  
}
```

# Arrays

Una matriz o "array" es una colección de variables que son accedidas mediante un número de índice. Los "arrays" en el lenguaje de programación C, en el cual está basado Arduino, pueden ser complicadas, pero usar "arrays" simples es relativamente sencillo.

## Creando (Declarando) una matriz o Array

Todos los métodos de abajo son formas válidas de crear (declarar) una matriz.

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hola";
```

Puedes declarar una matriz sin inicializarla como en myInts.

En myPins se declara una matriz sin establecer un tamaño explícito. El compilador cuenta el número de elementos y crea la matriz con el tamaño apropiado.

Finalmente, puedes tanto declarar con un tamaño la matriz e inicializarla al mismo tiempo, como en mySensVals

## Accediendo a una matriz o Array

Los Arrays son **zero indexed**, esto significa que, al referirse a una matriz, el primer elemento de la matriz está en el índice 0. Por lo tanto:

```
mySensVals[0] == 2, mySensVals[1] == 4, y sucesivos.
```

Esto también significa que en una matriz con 10 elementos. el índice 9 es el último elemento.

Por lo tanto:

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
// myArray[9] contiene 11
// myArray[10] es invalido y contiene información aleatoria (de otra
dirección de memoria)
```

Por esta razón hay que tener cuidado en el acceso a las matrices. El acceso más allá del final de una matriz (usando un número de índice mayor que el tamaño declarado - 1) resultará la lectura de la memoria que está en uso para otros fines. La lectura de estos lugares probablemente no va a hacer mucho mal, excepto la lectura de datos no válidos. Escribir en las localidades de memoria aleatoria es definitivamente una mala idea y, a menudo puede conducir a resultados inesperados como fallos o mal funcionamiento del programa. Esto también puede ser un error difícil encontrar.

A diferencia de BASIC o JAVA, el compilador de C no realiza ninguna comprobación para ver si el acceso a una matriz está dentro de los límites del tamaño de la matriz que ha declarado.

### **Para asignar un valor a una matriz:**

```
mySensVals[0] = 10;
```

### **To retrieve a value from an array:**

```
x = mySensVals[4];
```

## **Matrices y los bucles FOR**

Las matrices se utilizan muchas veces en el interior de bucles **for**, donde el contador de bucle se utiliza como el índice de cada elemento de la matriz. Por ejemplo, para imprimir los elementos de una matriz a través del puerto serie, se podría hacer algo como esto:

```
int i;
for (i = 0; i < 5; i = i + 1) {
    Serial.println(myPins[i]);
}
```

## **void**

La palabra reservada **void** se usa sólo en la declaración de funciones. Indica que se espera que no devuelva información a la función donde fué llamada.

### **Ejemplo:**

```
// varias acciones se llevan a cabo en las funciones "setup" y "loop"
// pero no se reporta ninguna información al programa principal.

void setup()
{
    // ...
}

void loop()
{
    // ...
}
```

# **CONVERSIONES**

## **char()**

### **Descripción**

Convierte un valor a un tipo de dato char.

### **Sintaxis**

char(x)

### **Parámetros**

x: un valor de cualquier tipo

### **Devuelve**

char

## **byte()**

### **Descripción**

Convierte una variable a un tipo de datos byte.

### **Sintaxis**

byte(x)

### **Parámetros**

x: una variable de cualquier tipo.

### **Devuelve**

byte

## **int()**

### **Descripción**

Convierte un valor al tipo de datos int.

### **Sintaxis**

int(x)

## **Parámetros**

x: un valor de cualquier tipo.

## **Retorna**

int

## **long()**

### **Descripción**

Convierte un valor al tipo de datos long.

### **Sintaxis**

long(x)

### **Parámetros**

x: un valor para cualquier tipo

### **Devuelve**

long

## **float()**

### **Descripción**

Convierte una variable a tipo float.

### **Sintaxis**

float(x)

### **Parámetros**

x: una variable de cualquier tipo

### **Devuelve**

número de coma flotante (float)



# 3.FUNCIONES

## E/S DIGITALES

### pinMode()

#### Descripción

Configura el pin especificado para comportarse como una entrada o una salida. Mira la descripción de [pines digitales](#) para más información.

#### Sintaxis

pinMode(pin, modo)

#### Parametros

pin: el numero del pin que se desea configurar

modo: INPUT (Entrada) o OUTPUT (Salida)

#### Devuelve

Nada

#### Ejemplo

```
int ledPin = 13;           // LED conectado al pin digital 13

void setup()
{
  pinMode(ledPin, OUTPUT); //configura el pin del LED como salida
}

void loop()
{
  digitalWrite(ledPin, HIGH); // Enciende el LED
  delay(1000);                // espera un segundo
  digitalWrite(ledPin, LOW);  // Apaga el LED
  delay(1000);                // Espera un segundo
}
```

#### Nota

Una entrada analógica puede ser usada como un pin digital, refiriéndose a ellos desde el número 14 (entrada analógica 0) a 19 (entrada analógica 5).

# digitalWrite()

## Descripción

Escribe un valor HIGH o LOW hacia un pin digital.

Si el pin ha sido configurado como OUTPUT con `pinMode()`, su voltaje será establecido al correspondiente valor: 5V ( o 3.3V en tarjetas de 3.3V) para HIGH, 0V (tierra) para LOW.

Si el pin es configurado como INPUT, escribir un valor de HIGH con `digitalWrite()` habilitará una resistencia interna de 20K conectada en pullup (ver el [tutorial de pines digitales](#)). Escribir LOW invalidará la resistencia. La resistencia es suficiente para hacer brillar un LED de forma opaca, si los LEDs aparentan funcionar, pero no muy iluminados, esta puede ser la causa. La solución es establecer el pin como salida con la función `pinMode()`.

**NOTA:** El pin digital número 13 es más difícil de usar que los otros pines digitales por que tiene un LED y una resistencia adjuntos, los cuales se encuentran soldados a la tarjeta, y la mayoría de las tarjetas se encuentran así. Si habilitas la resistencia interna en pullup, proporcionará 1.7V en vez de los 5V esperados, por que el LED soldado en la tarjeta y resistencias bajan el nivel de voltaje, significando que siempre regresará LOW. Si debes usar el pin número 13 como entrada digital, usa una resistencia externa conectada a pulldown.

## Sintaxis

`digitalWrite(pin, valor)`

## Parameters

pin: el número de pin

valor: HIGH o LOW

## Devuelve

nada

## Ejemplo

```
int ledPin = 13;           // LED conectado al pin digital 13
void setup()
{
  pinMode(ledPin, OUTPUT); // establece el pin digital como salida
}

void loop()
{
  digitalWrite(ledPin, HIGH); // enciende el LED
  delay(1000);                // espera por un segundo
  digitalWrite(ledPin, LOW);  // apaga el LED
  delay(1000);                // espera por un segundo
}
```

Establece el pin número 13 a HIGH, hace un retraso con la duración de un segundo, y regresa el pin a LOW.

### Nota

Los pines analógicos pueden ser usados como pines digitales con los números 14 (entrada analógica número 0) hasta 19 (entrada analógica número 5).

## digitalRead()

### Descripción

Lee el valor de un pin digital especificado, HIGH o LOW.

### Sintaxis

digitalRead(pin)

### Parámetros

pin: el número de pin digital que quieres leer (*int*)

### Devuelve

HIGH o LOW

### Ejemplo

```
int ledPin = 13; // LED conecado al pin digital número 13
int inPin = 7;   // botón (pushbutton) conectado al pin digital número 7
int val = 0;     // variable donde se almacena el valor leído

void setup()
{
  pinMode(ledPin, OUTPUT);      // establece el pin digital número 13 como
  salida
  pinMode(inPin, INPUT);       // establece el pin digital número 7 como
  entrada
}

void loop()
{
  val = digitalRead(inPin);     // leer el pin de entrada
  digitalWrite(ledPin, val);    // establece el LED al valor del botón
}
```

Establece el pin número 13 al mismo valor que el pin número 7, que es una entrada.

### Nota

Si el pin no esta conectado a algo, digitalRead() puede regresar HIGH o LOW (y esto puede cambiar aleatoriamente).

Los pines analógicos pueden ser usados como pines digitales con los números 14 (entrada analógica número 0) hasta 19 (entrada analógica número 5).

## E/S ANALÓGICAS

### **analogRead()**

#### **Descripción**

Lee el valor de tensión en el pin analógico especificado. La placa Arduino posee 6 canales (8 canales en el Mini y Nano y 16 en el Mega) conectados a un conversor analógico digital de 10 bits. Esto significa que convertirá tensiones entre 0 y 5 voltios a un número entero entre 0 y 1023. Esto proporciona una resolución en la lectura de: 5 voltios / 1024 unidades, es decir, 0.0049 voltios (4.9 mV) por unidad. El rango de entrada puede ser cambiado usando la función analogReference().

El conversor tarda aproximadamente 100 microsegundos (0.0001 segundos) en leer una entrada analógica por lo que se puede llevar una tasa de lectura máxima aproximada de 10.000 lecturas por segundo.

#### **Sintaxis**

analogRead(pin)

#### **Parámetros**

pin: Indica el número del pin de la entrada analógica que deseamos leer (0 a 5 en la mayoría de las placas, de 0 a 7 en las Mini y Nano y de 0 a 15 en las Mega)

#### **Devuelve**

int (0 a 1023)

#### **Nota**

Si la entrada analógica que vamos a leer no está conectada a nada, el valor que devolverá la función analogRead() fluctuará dependiendo de muchos valores (los valores de otras entradas analógicas, que tan cerca está tu mano de la entrada en cuestión, etc.)

#### **Ejemplo**

```
int analogPin = 3; // el pin analógico 3 conectado al dial de un
potenciómetro (terminal central del potenciómetro)
// los terminales exteriores del potenciómetro
conectados a +5V y masa respectivamente
```

```

int val = 0;           // declaración de la variable en la que se
almacenará el valor leído por el conversor.

void setup()
{
  Serial.begin(9600);      // Inicialización del modulo Serial.
}

void loop()
{
  val = analogRead(analogPin); // lee el valor de tensión del pin
  Serial.println(val);        // envía el valor leído vía serial.
}

```

## analogWrite()

### Descripción

Escribe un valor analógico (**PWM**) en un pin. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor. Después de llamar a la función **analogWrite()**, el pin generará una onda cuadrada estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función **analogWrite()** (o una llamada a las funciones **digitalRead()** o **digitalWrite()** en el mismo pin). La frecuencia de la señal PWM sera de aproximadamente 490 Hz.

En la mayoría de las placas Arduino (aquellas con el ATmega168 o ATmega328), se podrá generar señales PWM en los pines 3, 5, 6, 9, 10, y 11. En la placa Arduino Mega, se puede llevar a cabo con los pines desde el 2 hasta el pin 13. Las placas Arduino más antiguas que posean el chip ATmega8 solo podrán usar la función `analogWrite()` con los pines 9, 10 y 11. No hace falta configurar el pin como salida para poder usar la función `analogWrite()`.

La función *analogWrite* no tienen ninguna relación con los pines de entrada analógicos ni con la función *analogRead*.

### Sintaxis

```
analogWrite(pin, valor)
```

### Parámetros

pin: Es el pin en el cual se quiere generar la señal PWM.

valor: El ciclo de trabajo deseado comprendido entre 0 (siempre apagado) y 255 (siempre encendido).

### Devuelve

Nada

## Notas y problemas conocidos.

Las señales PWM generadas en los pines 5 y 6 poseerán ciclos de trabajo superiores a lo esperado. Esto es así por que para esos dos pines se utiliza el mismo temporizador que se utiliza en las funciones `millis()` y `delay()`. Este efecto se notará mucho más en ciclos de trabajo bajos (por ejemplo de 0 a 10) y puede ser que aunque configuremos esos pines con una señal de ciclo de trabajo cero no llegue a ser verdaderamente 0.

## Ejemplo

Produce una señal donde conectamos el LED, cuyo ciclo de trabajo es proporcional a la tensión leída en el potenciómetro.

```
int ledPin = 9;           // LED conectado al pin digital 9
int analogPin = 3;       // potenciómetro conectado al pin 3
int val = 0;             // variable en el que se almacena el dato leído

void setup()
{
  pinMode(ledPin, OUTPUT); // sets the pin as output
}

void loop()
{
  val = analogRead(analogPin); // lee la tensión en el pin
  analogWrite(ledPin, val / 4); // los valores de analogRead van desde 0 a
  1023 y los valores de analogWrite values van desde 0 a 255, por eso
  ajustamos el ciclo de trabajo a el valor leído dividido por 4.
}
```

## E/S AVANZADAS

### tone()

#### Descripción

Genera una onda cuadrada de la frecuencia especificada (y un 50% de ciclo de trabajo) en un pin. La duración puede ser especificada, en caso contrario la onda continua hasta que haya una llamada a `noTone()`. El pin puede conectarse a un zumbador piezoeléctrico u otro altavoz que haga sonar los tonos.

Solo puede generarse un tono cada vez. Si un tono está sonando en un pin diferente, la llamada a `tone()` no tendrá efecto. Si el tono está sonando en el mismo pin, la llamada establecerá la nueva frecuencia.

El uso de la función `tone()` interferirá con la salida PWM en los pins 3 y 11 (en otras placas distintas de la Mega).

**NOTA:** si quieres hacer sonar diferentes tonos en múltiples pins necesitas llamar a noTone() en un pin antes de llamar a tone() en el siguiente pin.

### **Sintaxis**

tone(pin, frecuencia)

tone(pin, frecuencia, duración)

### **Parámetros**

pin: el pin en el que generar el tono

frecuencia: la frecuencia del tono en hercios.

duración: la duración del tono en milisegundos (opcional)

**NOTA:** las frecuencias audibles por el oído humano van de los 20Hz a los 20KHz por lo que el parámetro "frecuencia" debería estar comprendido entre estos dos valores.

## **noTone()**

### **Descripción**

Detiene la generación de la señal cuadrada que se activa al hacer uso de la función tone(). No tiene efecto si no se está generando ningún tono.

**NOTA:** si quieres generar tonos diferentes en múltiples pines , necesitas usar la función noTone() en el pin antes de llamar a la función tone() en el siguiente pin.

### **Sintaxis**

noTone(pin)

### **Parámetros**

pin: el pin en el cual se va a parar de generar el tono.

### **Devuelve**

Nada

# shiftOut()

## Descripción

Desplaza un byte de datos bit a bit. Empieza desde el bit más significativo (el de más a la izquierda) o el menos significativo (el más a la derecha). Cada bit se escribe siguiendo su turno en un pin de datos, después de conmutar un pin de reloj (señal de reloj) para indicar que el bit está disponible.

Esto es conocido como protocolo serie síncrono y es la forma común que utilizan los microcontroladores para comunicarse con sensores y con otros microcontroladores. Ambos dispositivos permanecen sincronizados, y se comunican a velocidades cercanas a las máximas, hasta que ambos compartan la misma línea de reloj. En la documentación del hardware interno de los chips se hace referencia a menudo a menudo a esta característica como Serial Peripheral Interface (SPI).

## Sintaxis

```
shiftOut(pinDatos, pinReloj, ordenBits, valor)
```

## Parametros

pinDatos: el pin en el cual extraer cada bit (*int*)

pinReloj: el pin que hay que conmutar cada vez que a un **pinDatos** le ha sido enviado el valor correcto (**int**)

ordenBits: en qué orden desplazar los bits; si hacia el **MSBFIRST** (bit más significativo primero) o hacia el **LSBFIRST** (bit menos significativo primero).

valor: los datos que rotar. (*byte*)

## Retorno

Ninguno

## Nota

El **pinDatos** y **pinReloj** deben estar ya configurados como salida con una llamada previa a pinMode().

**shiftOut** se encuentra actualmente configurada para extraer un byte (8 bits) por lo que necesita realizar una operación de dos pasos para extraer valores más grandes de 255.

```
// Haz esto para una comunicación serie MSBFIRST (primero el bit más
significativo)
int datos = 500;
// rota el byte más alto
shiftOut(pinDatos, pinReloj, MSBFIRST, (datos >> 8));
// rota el byte más bajo
```



```

shiftOut(datos, pinRelej, MSBFIRST, datos);

// O haz esto para una comunicación serie LSBFIRST (primero el bit menos
significativo)
datos = 500;
// rota el byte más bajo
shiftOut(pinDatos, pinRelej, LSBFIRST, datos);
// rota el bit más alto
shiftOut(pinDatos, pinRelej, LSBFIRST, (datos >> 8));

```

## Ejemplo

```

//*****//
// Name      : shiftOutCode, Hello World           //
// Author    : Carlyn Maw, Tom Igoe               //
// Date      : 25 Oct, 2006                       //
// Version   : 1.0                                //
// Notes     : Código para utilizar un registro de desplazamiento//
//             : 74HC595 para contar de 0 a 255    //
//*****//

//Pin conectado al pin ST_CP del 74HC595
int latchPin = 8;
//Pin conectado al pin SH_CP del 74HC595
int clockPin = 12;
////Pin conectado al pin DS del 74HC595
int dataPin = 11;

void setup() {
  // Configura como salida los pines que se direccionan en el bucle
  principal (loop)
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  //rutina de conteo
  for (int j = 0; j < 256; j++) {
    //pone a nivel bajo el latchPin y lo mantienen a nivel bajo todo el
    tiempo que estés transmitiendo
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, j);
    //vuelve a poner el latchPin a nivel alto para señalar que
    //no sigue siendo necesario escuchar más información
    digitalWrite(latchPin, HIGH);
    delay(1000);
  }
}

```

# pulseIn()

## Descripción

Lee un pulso (ya sea HIGH —alto— o LOW —bajo—) en un pin. Por ejemplo, si **value** es **HIGH**, **pulseIn()** espera a que el pin sea **HIGH**, empieza a cronometrar, espera a que el pin sea **LOW** y entonces detiene la medida de tiempo. Devuelve la anchura del pulso en microsegundos. Interrumpe la medida y devuelve 0 si el pulso no ha comenzado en un tiempo especificado.

La medida del tiempo en esta función ha sido determinada de forma empírica y está sujeta a errores en pulsos largos. Funciona correctamente en pulsos con una anchura de 10 microsegundos a tres minutos.

## Sintaxis

```
pulseIn(pin, value)
pulseIn(pin, value, timeout)
```

## Parámetros

pin: el número del pin en el que se realiza la medida del pulso. (*int*)  
value: tipo de pulso. Puede ser HIGH o LOW. (*int*)  
timeout (opcional): el tiempo en microsegundos máximo a esperar antes de que se inicie el pulso. (*unsigned long*)

## Devuelve

el ancho del pulso (en microsegundos) ó 0 si el pulso no ha empezado antes del timeout (*unsigned long*)

## Ejemplo

```
int pin = 7;
unsigned long duracion;

void setup()
{
  pinMode(pin, INPUT);
}

void loop()
{
  duracion = pulseIn(pin, HIGH);
}
```

# TIEMPO

## millis()

### Descripción

Devuelve el tiempo en milisegundos transcurridos desde que se arranco la placa Arduino con el programa actual. Este número de desbordará (volverá a cero), después de aproximadamente 50 días.

### Parámetros

Ninguno

### Devuelve

Tiempo en milisegundos desde que el programa se inició (*long sin signo (unsigned long)*)

### Ejemplo

```
unsigned long time;

void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.print("Tiempo: ");
  time = millis();
  //Imprime el tiempo desde que se inició el programa
  Serial.println(time);
  // espera un segundo para no enviar demasiados datos
  delay(1000);
}
```

### Consejo:

Ten en cuenta que el parámetro que devuelve millis() es un long sin signo por lo que pueden producirse errores si el programador intenta llevar a cabo operaciones matemáticas con otros tipos de dato como enteros.

# micros()

## Descripción

Devuelve el número de microsegundos desde que la placa Arduino comenzó a correr el programa. Este número eventualmente volverá a 0 (*overflow*), luego de aproximadamente 70 minutos. A 16 MHz, en las placas Arduino (por ejemplo, *Duemilanove* ó *Nano*), éstas función tiene una resolución de cuatro milisegundos (por ejemplo, el valor devuelto es siempre múltiplo de 4). A 8 MHz, en placas Arduino (como *LilyPad*), tiene una resolución de 8 microsegundos.

*Nota:* En un milisegundo hay 1.000 microsegundos y 1,000,000 microsegundos en un segundo.

## Parámetros

Ninguno.

## Devuelve

El número de microsegundos desde el inicio del programa (*unsigned long*)

## Ejemplo

```
unsigned long time;

void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Tiempo: ");
  time = micros();
  //Imprime el tiempo desde el comienzo del programa
  Serial.println(time);
  // espera un segundo para no enviar datos una cantidad exagerada de
  datos.
  delay(1000);
}
```

# delay()

## Descripción

Pausa el programa por un tiempo determinado (en milisegundos) especificado por un parámetro. Hay 1000 milisegundos en un segundo.

## Sintaxis

delay(ms)

## Parámetros

ms: el número de milisegundos que se desea pausar el programa (*unsigned long*)

## Devuelve

nada

## Ejemplo

```
int ledPin = 13; // LED conectado al pin digital 13.

void setup()
{
  pinMode(ledPin, OUTPUT); // declara que el pin digital se va a usar como
  salida
}

void loop()
{
  digitalWrite(ledPin, HIGH); // enciende el LED
  delay(1000); // espera durante un segundo
  digitalWrite(ledPin, LOW); // apaga el LED
  delay(1000); // espera durante un segundo
}
```

## Advertencia

Aunque es fácil crear un LED parpadeante con la función `delay()` y muchos *sketches* usan pausas cortas para estas tareas, el uso de `delay()` en un *sketch* tiene problemas importantes. Mientras se pausa el programa no se leen sensores, ni se hacen cálculos, ni puede haber manipulación de los pines. En definitiva, hace que (casi) toda la actividad se pare. Una alternativa para controlar el tiempo es el uso de la función `millis()` y el *sketch* mostrado abajo. Normalmente se evita el uso de `delay()` para tiempos mayores de decenas de milisegundos excepto en programas muy simples.

Algunas cosas siguen funcionando mientras se ejecuta la función `delay()` porque no se deshabilitan las interrupciones. La comunicación serie de los pines RX sigue funcionando, los valores de PWM (`analogWrite`) y los estados de los pines se mantienen y las interrupciones siguen funcionando correctamente.

## delayMicroseconds()

### Descripción

Detiene brevemente el programa por la cantidad en tiempo (en microsegundos) especificada como parámetro. Existen mil microsegundos en un milisegundo, y un millón de microsegundos en un segundo.

Actualmente, el valor más grande producirá un retraso exacto es 16383. Esto puede cambiar en una futura versión de Arduino. Para retrasos más largos que algunos miles de microsegundos, deberías usar `delay()` .

## Sintaxis

`delayMicroseconds(us)`

## Parametros

us: el número de microsegundos a detener (*unsigned int*)

## Devuelve

Nada

## Ejemplo

```
int outPin = 8; // selecciona el pin 8

void setup()
{
  pinMode(outPin, OUTPUT); // establece el pin digital como salida
}

void loop()
{
  digitalWrite(outPin, HIGH); // establece en encendido el pin
  delayMicroseconds(50); // espera por 50 microsegundos
  digitalWrite(outPin, LOW); // establece en apagado el pin
  delayMicroseconds(50); // espera por 50 microsegundos
}
```

configura el pin número 8 para trabajar como pin de salida. Envía una serie de pulsos con 100 microsegundos de período.

## Advertencias y problemas conocidos

Esta función trabaja de manera exacta en el rango de 3 microsegundos y valores superiores. No podemos asegurar que la función `dealyMicroseconds` funcionará de forma precisa para valores menores de retraso.

As of Arduino 0018, `delayMicroseconds()` no longer disables interrupts. A partir de Arduino 0018, `delayMicroseconds()` ya no invalida las interrupciones

## **hour()**

### **Descripción**

Devuelve la hora en formato numérico (0-23).

### **Sintaxis**

```
hour();
```

### **Devuelve**

El número de horas transcurridas desde que se inicio el reloj.

## **minute()**

### **Descripción**

Devuelve los minutos en formato numérico (0-60).

### **Sintaxis**

```
minute();
```

### **Devuelve**

El número de minutos que nos da el reloj.

## **second()**

### **Descripción**

Devuelve los segundos en formato numérico (0-60).

### **Sintaxis**

```
second();
```

### **Devuelve**

El número de segundos que nos da el reloj.

## **day()**

### **Descripción**

Devuelve el día del mes en formato numerico.(0-31)

### **Sintaxis**

day ();

### **Devuelve**

Dia del mes.

## **month()**

### **Descripción**

Devuelve el mes del año en formato numerico.(0-12)

### **Sintaxis**

month ();

### **Devuelve**

Mes del año.

## **year()**

### **Descripción**

Devuelve el año en el que nos encontramos en formato numerico.

### **Sintaxis**

year ();

### **Devuelve**

El año en el que nos encontramos.

### **Ejemplo**

En el ejemplo siguiente podemos ver como se utilizan las funciones de tiempo para mostrar la hora a la que nos encontramos. setTime(); es la función que utilizamos para iniciar el reloj.



```

/*
 * Time sketch
 *
 */

#include <Time.h>
void setup()
{
  Serial.begin(9600);
  setTime(12,0,0,1,1,11); // set time to noon Jan 1 2011
}
void loop()
{
  digitalClockDisplay();
  delay(1000);
}
void digitalClockDisplay(){
  // digital clock display of the time
  Serial.print(hour());
  printDigits(minute());
  printDigits(second());
  Serial.print(" ");
  Serial.print(day());
  Serial.print(" ");
  Serial.print(month());
  Serial.print(" ");
  Serial.print(year());
  Serial.println();
}
void printDigits(int digits){
  // utility function for clock display: prints preceding colon and leading 0
  Serial.print(":");
  if(digits < 10)
  Serial.print('0');
  Serial.print(digits);
}

```

## **MATEMATICAS**

**min(x, y)**

### **Descripción**

Calcula el mínimo de dos números.

### **Parámetros**

x: el primer número, cualquier tipo de dato  
y: el segundo número, cualquier tipo de dato

### **Devuelve**

El más pequeño entre los dos números.

## Ejemplos

```
sensVal = min(sensVal, 100); // asigna sensVal al menor entre sensVal y 100
                               // asegurando que nunca pasará de 100.
```

## Nota

La función `max()` es usualmente usada para limitar el límite inferior del rango de una variable, mientras que la función `min()` es usada para limitar el límite superior del rango de una variable.

## Advertencia

Debido a la forma en la que se ha implementado la función `min()`, evite realizar operaciones en el argumentos de la función ya que podría producir errores.

```
min(a++, 100); // evite esto - puede producir errores

a++;
min(a, 100); // use este método en cambio - mantenga cualquier operación
              fuera de los paréntesis
```

## max(x, y)

### Descripción

Calcula el máximo de dos números.

### Parámetros

- x: El primer número, cualquier tipo de dato.
- y: El segundo número, cualquier tipo de dato.

### Devuelve

El mayor de ambos parámetros.

### Ejemplo

```
sensVal = max(sensVal, 20); // asigna a sensVal su propio valor o, de ser
                             superior, 20.
                               // (una forma efectiva de asegurarse que el
valor mínimo de sensVal sea 20)
```

## Nota

`max()` suele ser usado para restringir el valor más bajo del rango de una variable, mientras que `min()` es utilizado para restringir el valor máximo del rango.

## Atención

Debido a la forma en que la función `max()` es implementada, debes evitar usar otras funciones al definir los parámetros, puede derivar en resultados incorrectos.

```
max(a--, 0); // evitar esto - puede dar resultados incorrectos.

a--; // en su lugar, hacerlo así -
max(a, 0); // mantener cualquier operación fuera de los paréntesis.
```

## abs(x)

### Descripción

Calcula el valor absoluto de un número.

### Parámetros

x: El numero cuyo valor absoluto deseamos calcular

### Devuelve

x: si x is mayor o igual que 0.

-x: si x es menor que 0.

### Precaución

Debido a la forma en la que se ha implementado la función `abs()`, evite usar otras funciones como parámetro debido a que puede ocasionar que se devuelva un resultado incorrecto.

```
abs(a++); // evite esto - produce errores en el resultado

a++; // hazlo de esta manera -
abs(a); // mantenga cualquier otra operación fuera de la función
```

## constrain(x, a, b)

### Descripción

Restringe un número a un rango definido.

### Parámetros

x: el número a restringir, cualquier tipo de datos.

a: El número menor del rango, cualquier tipo de datos.

b: El número mayor del rango, cualquier tipo de datos.

## Retorna

**x**: si **x** está entre **a** y **b**

**a**: si **x** es menor que **a**

**b**: si **x** es mayor que **b**

## Ejemplo

```
sensVal = constrain(sensVal, 10, 150);  
// limita el valor del sensor entre 10 y 150
```

## map(value, fromLow, fromHigh, toLow, toHigh)

### Descripción

Re-mapea un número desde un rango hacia otro. Esto significa que, un valor (*value*) con respecto al rango *fromLow*-*fromHigh* será mapeado al rango *toLow*-*toHigh*.

No se limitan los valores dentro del rango, ya que los valores *fuera de rango* son a veces objetivos y útiles. La función **constrain()** puede ser usada tanto antes como después de esta función, si los límites de los rangos son deseados.

Ten en cuenta que los límites "inferiores" de algún rango pueden ser mayores o menores que el límite "superior" por lo que *map()* puede utilizarse para revertir una serie de números, por ejemplo:

```
y = map(x, 1, 50, 50, 1);
```

La función maneja correctamente también los números negativos, por ejemplo:

```
y = map(x, 1, 50, 50, -100);
```

también es válido y funciona correctamente.

La función *map()* usa matemática de enteros por lo que no generará fracciones, aunque fuere el resultado correcto. Los resultados en fracciones se truncan, y no son redondeados o promediados.

### Parámetros

**value**: el número (valor) a mapear.

**fromLow**: el límite inferior del rango actual del valor.

**fromHigh**: el límite superior del rango actual del valor.

**toLow**: límite inferior del rango deseado.

**toHigh**: límite superior del rango deseado.

## Devuelve

El valor mapeado.

## Ejemplo

```
/* Mapea un valor análogo a 8 bits (0 a 255) */
void setup() {}

void loop()
{
  int val = analogRead(0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

## Apéndice

Para los interesados en el funcionamiento de la función, aquí está su código:

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

## pow(base, exponente)

### Descripción

Calcula el valor de un número elevado a otro número. Pow() puede ser usado para elevar un número a una fracción. Esta función es útil para generar datos exponenciales o curvas.

### Parámetros

base: base que queremos elevar (*float*) (Coma flotante)

exponente: la potencia a la que se quiere elevar la base (*float*) (Coma flotante)

### Devuelve

El resultado de la exponenciación (*double*)

## sq(x)

### Descripción

Calcula el cuadrado de un numero: el numero multiplicado por el mismo.

### Parámetros

x: el numero, cualquier tipo de dato

**Devuelve**

El cuadrado del numero

**sqrt(x)****Descripción**

Calcula la raíz cuadrada de un numero.

**Parámetros**

x: el numero, cualquier tipo de dato

**Devuelve**

doble, la raíz cuadrada del numero.

**TRIGONOMETRIA****sin(rad)****Descripción**

Calcula el seno de un ángulo (en radianes). El resultado será entre -1 y 1.

**Parametros**

rad: el ángulo en radianes (*float*)

**Retorno**

El seno del ángulo (*double*)

**cos(rad)****Descripción**

Calcula el coseno de un ángulo (en radianes). El resultado estará entre -1 y 1.

**Parámetros**

rad: el ángulo en radianes (*float*)

**Retorna**

El coseno del ángulo ("double")

## **tan(rad)**

### **Descripción**

Calcula la tangente de un ángulo (en radianes). El resultado estará entre el menos infinito y el infinito.

### **Parámetros**

rad: el ángulo en radianes (*float*)

### **Retorno**

La tangente del ángulo (*double*)

## **NUMEROS ALEATORIOS**

## **randomSeed(seed)**

### **Descripción**

randomSeed() inicializa el generador de números pseudoaleatorios, haciéndole empezar en un punto arbitrario de su secuencia aleatoria. Esta secuencia, aunque muy larga y aleatoria, es siempre la misma.

Si es importante que la secuencia de valores generada por random() difiera en ejecuciones sucesivas de un programa, es recomendable utilizar randomSeed() (*seed* en inglés, semilla) para inicializar el generador de números aleatorios con una entrada mínimamente aleatoria como analogRead() en un pin desconectado.

No obstante, puede ser útil usar secuencias pseudoaleatorias que se repitan exactamente. Esto se consigue llamando a randomSeed() con un número fijo antes de empezar la generación de la secuencia.

### **Parámetros**

long, int - recibe un número para generar la semilla.

### **Devuelve**

no devuelve nada

## Ejemplo

```
long numAleatorio;

void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop(){
  randomNumber = random(300);
  Serial.println(numAleatorio);

  delay(50);
}
```

## random()

### Descripción

La función *random* genera números pseudoaleatorios.

### Sintaxis

```
random(max)
random(min, max)
```

### Parámetros

min - límite inferior del valor aleatorio, inclusive (*opcional*)

max - límite superior del valor aleatorio, exclusive (se devuelve hasta el anterior)

### Devuelve

un número aleatorio entre min y max (*long*)

### Nota:

Si es importante que la secuencia de valores generada por `random()` difiera en ejecuciones sucesivas de un programa, es recomendable utilizar `randomSeed()` (*seed* en inglés, semilla) para inicializar el generador de números aleatorios con una entrada mínimamente aleatoria como `analogRead()` en un pin desconectado.

No obstante, puede ser útil usar secuencias pseudoaleatorias que se repitan exactamente. Esto se consigue llamando a `randomSeed()` con un número fijo antes de empezar la generación de la secuencia.



## Ejemplo

```
long numAleatorio;

void setup(){
  Serial.begin(9600);
  // si la entrada analógica 0 no está conectada,
  // la llamada a randomSeed() recibirá ruido estático
  // (analógico) y se generarán diferentes semillas
  // cada vez que se ejecute el sketch.
  randomSeed(analogRead(0));
}

void loop() {
  // escribe un número aleatorio de 0 a 299
  numAleatorio = random(300);
  Serial.println(numAleatorio);

  // escribe un número aleatorio de 10 a 19
  numAleatorio = random(10, 20);
  Serial.println(numAleatorio);

  delay(50);
}
```

## Serial

Se utiliza para la comunicación entre la placa Arduino y un ordenador u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie (también conocido como UART o USART): **Serial**. Se comunica a través de los pines digitales 0 (RX) y 1 (TX), así como con el ordenador mediante USB. Por lo tanto, si utilizas estas funciones, no puedes usar los pines 0 y 1 como entrada o salida digital.

Puedes utilizar el monitor del puerto serie incorporado en el entorno Arduino para comunicarte con la placa Arduino. Haz clic en el botón del monitor de puerto serie en la barra de herramientas y selecciona la misma velocidad en baudios utilizada en la llamada a begin().

La placa Arduino Mega tiene tres puertos adicionales de serie: **Serial1** en los pines 19 (RX) y 18 (TX), **Serial2** en los pines 17 (RX) y 16 (TX), **Serial3** en los pines 15 (RX) y 14 (TX). Para utilizar estos pines para comunicarse con el ordenador personal, necesitarás un adaptador USB adicional a serie, ya que no están conectados al adaptador USB-Serie de la placa Arduino Mega. Para usarlos para comunicarse con un dispositivo serie externo TTL, conecta el pin TX al pin RX del dispositivo, el RX al pin TX del dispositivo, y el GND de tu Arduino Mega a masa del dispositivo. (No conectes estos pines directamente a un puerto serie RS232, que operan a +/- 12V y esto puede dañar la placa Arduino.)

# APLICACIÓN ROBOTICA DE ORIENTACION SOLAR

Rubén Ostiz Tellechea



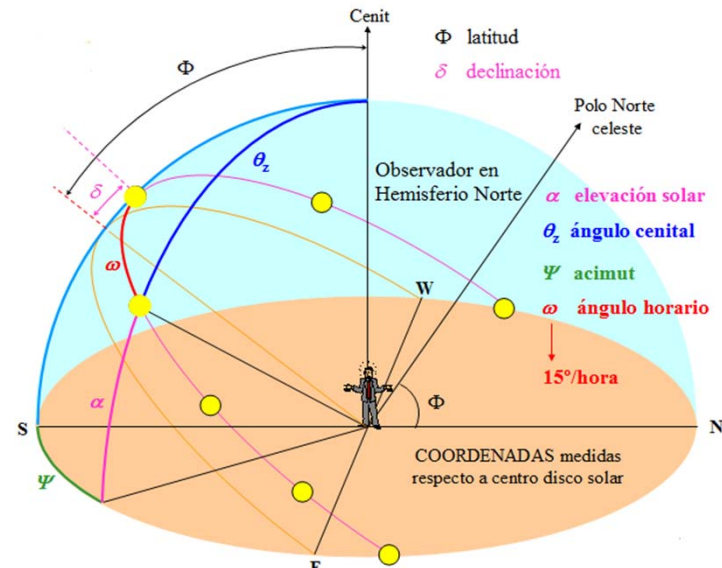
# 1. OBJETIVO Y DESCRIPCION DEL PFC

- Implementar un programa capaz de calcular la latitud y la longitud a través de la posición del Sol.
- Datos de entrada: hora y fecha en Greenwich, elevación y acimut del Sol.
- Para el programa implementado utilizaremos un microprocesador Arduino debido a su sencillo lenguaje de programación. Además de su bajo coste ya que no son necesarias ningún tipo de licencia para su utilización.

APLICACION ROBOTICA DE ORIENTACION SOLAR

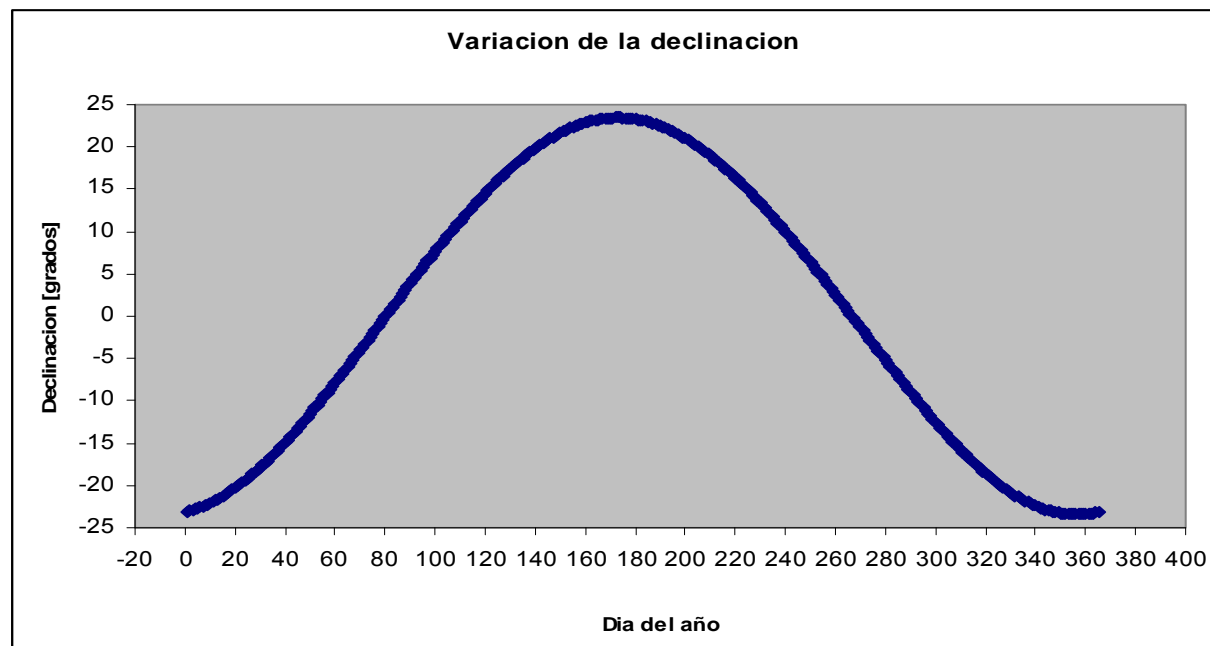
## 2. EL SISTEMA DE COORDENADAS

- 2.1 ESFERA CELESTE
  - COORDENADAS UTILIZADAS
    - Elevación solar ( $\alpha$ )
    - Acimut( $\Psi$ )
  - DATOS AUXILIARES
    - Ángulo horario( $\omega$ )
    - Declinación( $\delta$ )



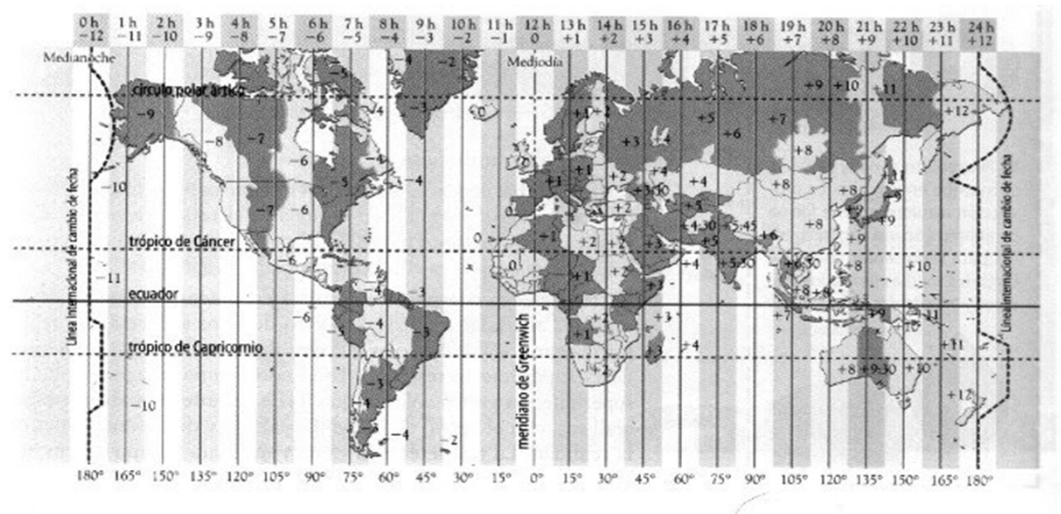
APLICACION ROBOTICA DE ORIENTACION SOLAR

## 2. EL SISTEMA DE COORDENADAS



APLICACION ROBOTICA DE ORIENTACION SOLAR

### 3. METODO CALCULO LONGITUD



$$\text{Longitud} = 15 \cdot (\text{HG} - \text{ML})$$

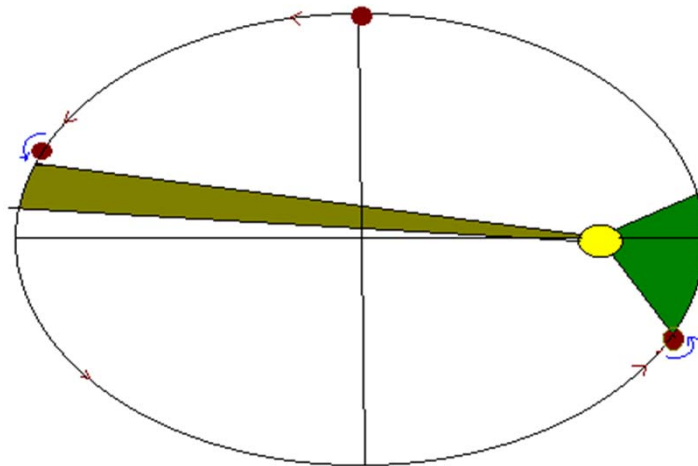
HG → Hora en el meridiano de Greenwich.

ML → Hora de Greenwich en el meridiano local.

APLICACION ROBOTICA DE ORIENTACION SOLAR

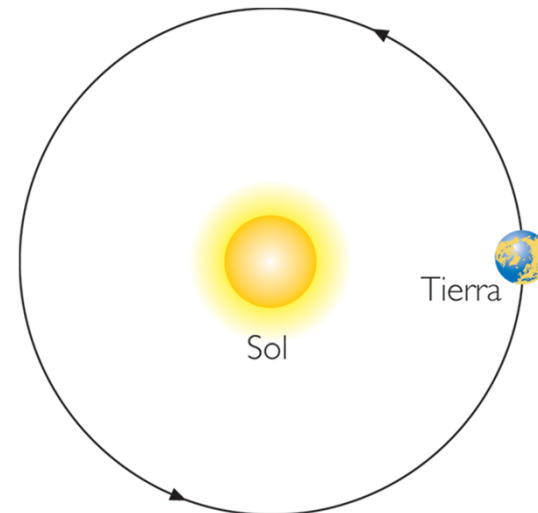
# 3.1. ECUACION DE TIEMPO

Día Solar



Áreas iguales "barridas por la Tierra en tiempos iguales

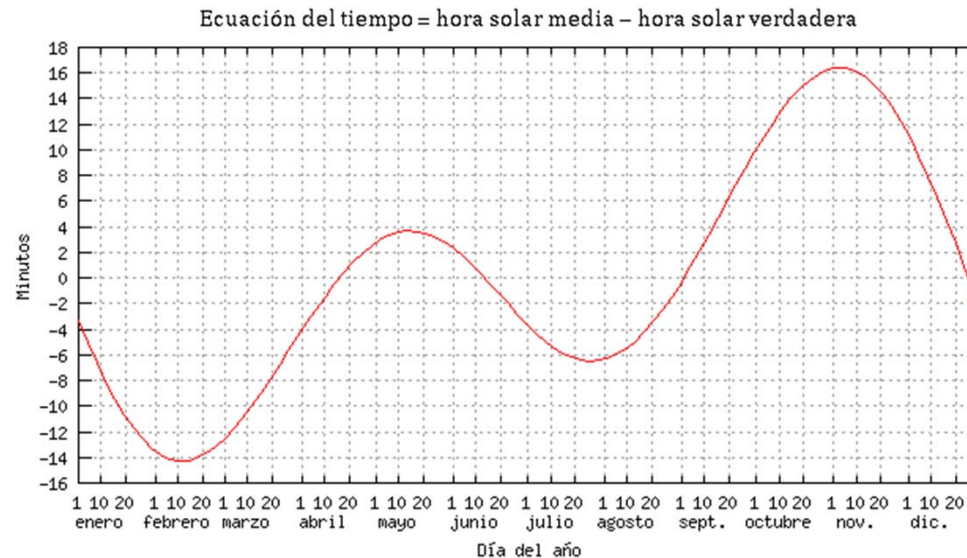
Día solar medio



APLICACION ROBOTICA DE ORIENTACION SOLAR

# 3.1. ECUACION DE TIEMPO

- La discrepancia existente entre el movimiento aparente del Sol medio y el movimiento del Sol verdadero se llama **ecuación de tiempo**.

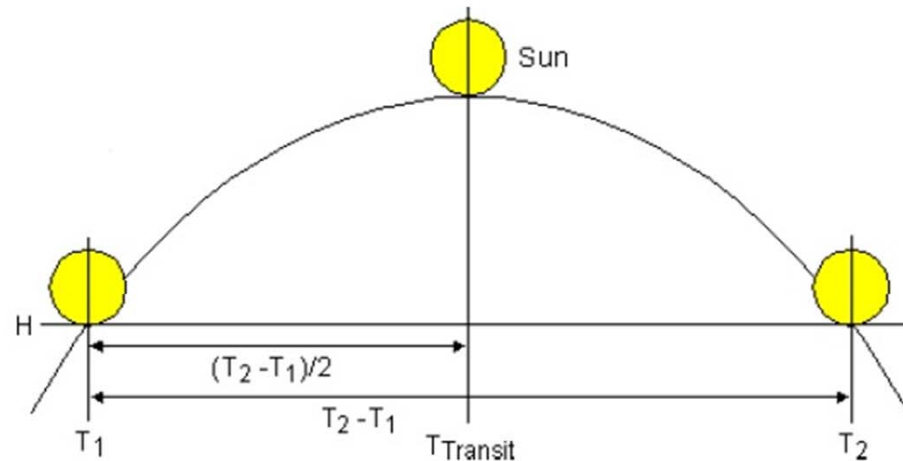


APLICACION ROBOTICA DE ORIENTACION SOLAR



## 3.2. METODO DE CALCULO DE LA HORA DEL MEDIODIA SOLAR

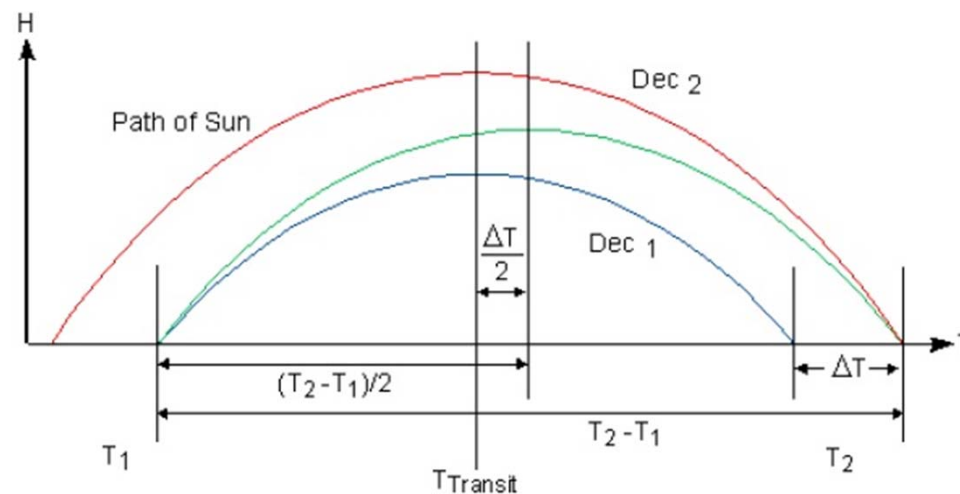
- Si asumimos que el Sol describe un arco simétrico en el cielo, cogemos dos medidas de tiempo, una antes del mediodía solar ( $T_1$ ) y otra después del mediodía solar ( $T_2$ ), ambas con la misma elevación solar. La media de estas medidas ( $T_{\text{Transit}}$ ) será el momento en que el sol pasa por su elevación máxima, y por tanto, por el mediodía solar.



APLICACION ROBOTICA DE ORIENTACION SOLAR

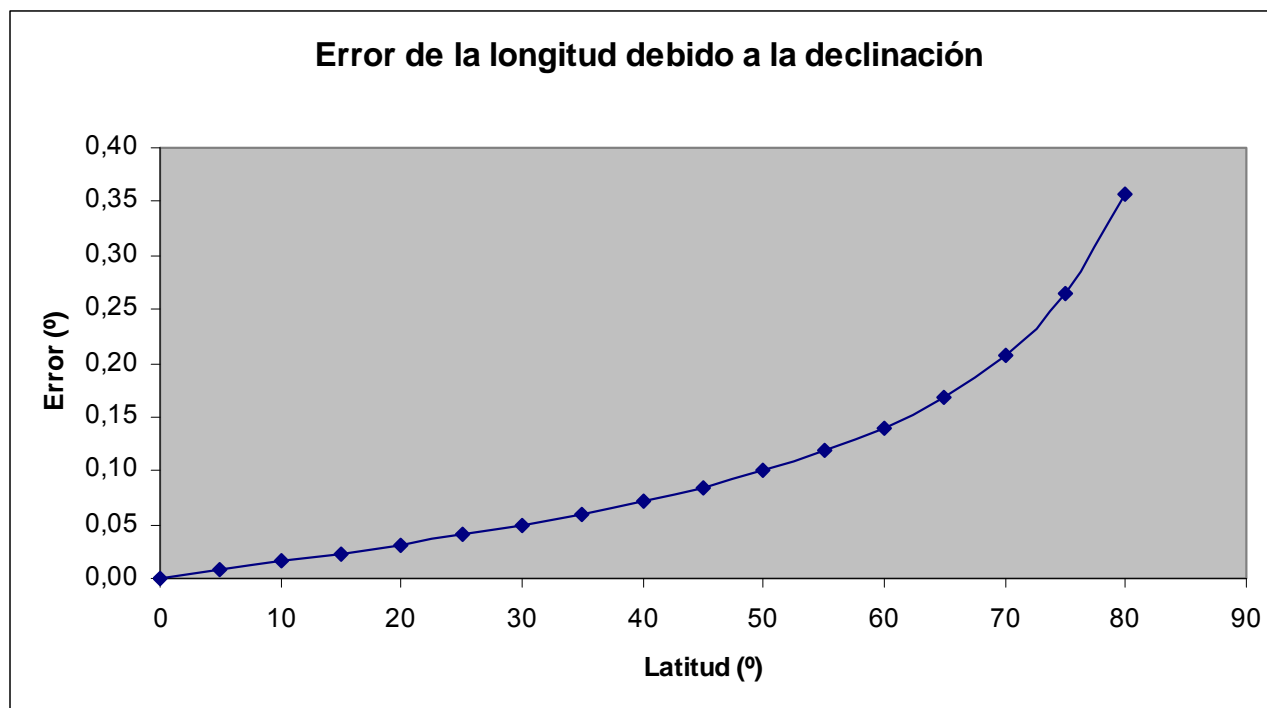
## 3.2. METODO DE CALCULO DE LA HORA DEL MEDIODIA SOLAR

- Desgraciadamente, el arco que describiría el Sol será simétrico solo cuando la declinación sea constante, caso que solo se produce aproximadamente durante los solsticios.



APLICACION ROBOTICA DE ORIENTACION SOLAR

## 3.2. METODO DE CALCULO DE LA HORA DEL MEDIODIA SOLAR



APLICACION ROBOTICA DE ORIENTACION SOLAR

## 3.3. ECUACION FINAL

- Aplicando la ecuación de tiempo y el cálculo del mediodía solar nos queda:

$$\textit{Longitud} = 0.25 \cdot (HG[\text{min}] - ML[\text{min}] - E_t[\text{min}])$$

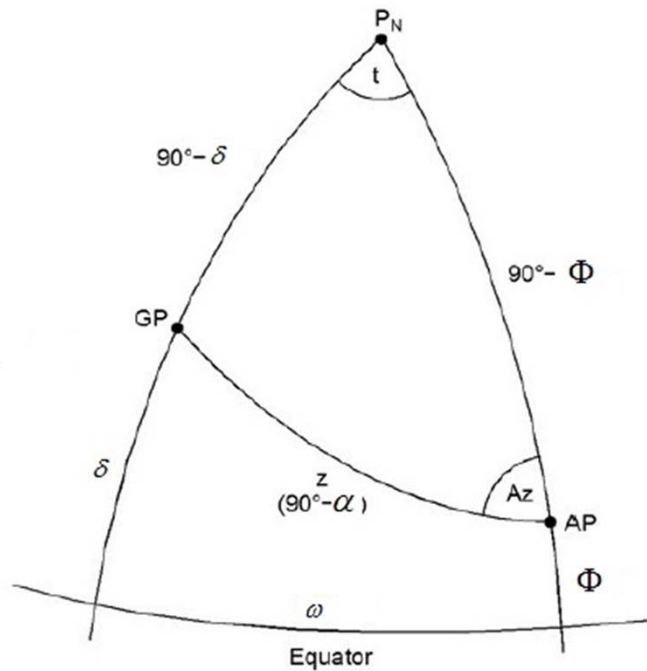
Donde HG será la hora de meridiano de Greenwich cuando el Sol pase por el meridiano de Greenwich, es decir, las 12 (720min.) y ML será la hora de meridiano de Greenwich cuando el sol pase por el meridiano local. Nos queda:

$$\textit{Longitud} = 0.25 \cdot (720 - ML[\text{min}] - E_t[\text{min}])$$

APLICACION ROBOTICA DE ORIENTACION SOLAR

# 4.1. ECUACION DE LA LATITUD

- Triangulo de navegaci3n



APLICACION ROBOTICA DE ORIENTACION SOLAR

## 4.1. ECUACION DE LA LATITUD

- Obtenemos la siguiente formula:

$$\text{sen } \alpha = \text{sen } \Phi \cdot \text{sen } \delta + \cos \Phi \cdot \cos \delta \cdot \cos \omega$$

Buscamos:

$$\cos \Phi = a \cdot \text{sen } \Phi + b$$

$$a = -\frac{\text{sen } \delta}{\cos \delta \cdot \cos \omega}$$

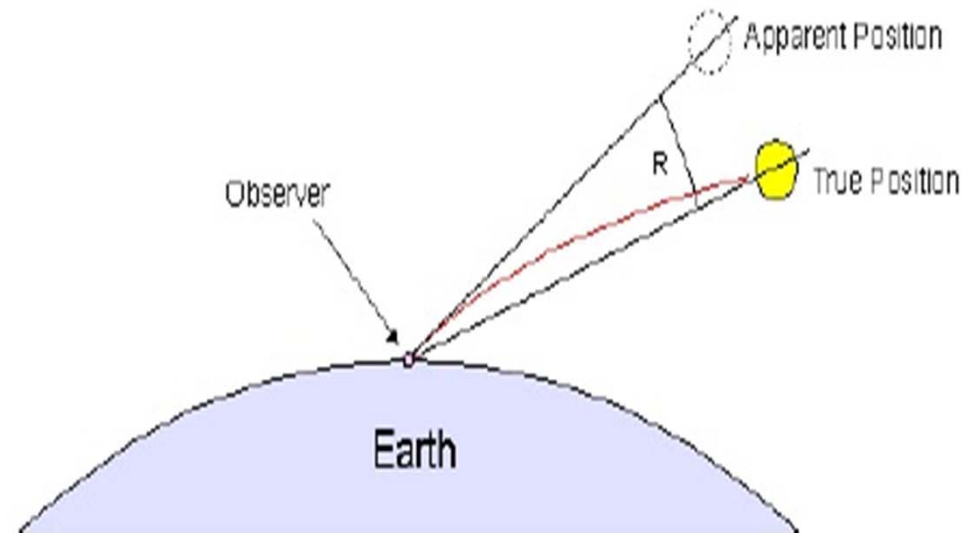
$$b = \frac{\text{sen } \alpha}{\cos \delta \cdot \cos \omega}$$

Al final:

$$\Phi = \text{sen}^{-1} \left( \frac{-2ab \pm \sqrt{4a^2b^2 - 4(1+a^2)(b^2-1)}}{2(1+a^2)} \right)$$

APLICACION ROBOTICA DE ORIENTACION SOLAR

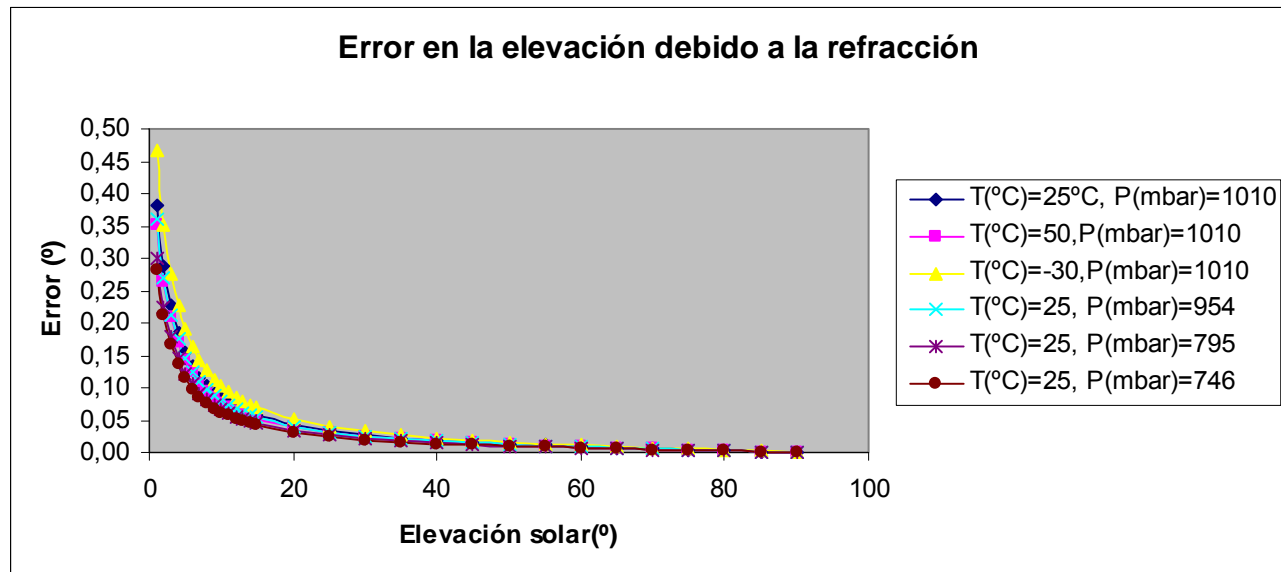
## 4.2.1 CORRECCION REFRACCION ATMOSFERICA



APLICACION ROBOTICA DE ORIENTACION SOLAR

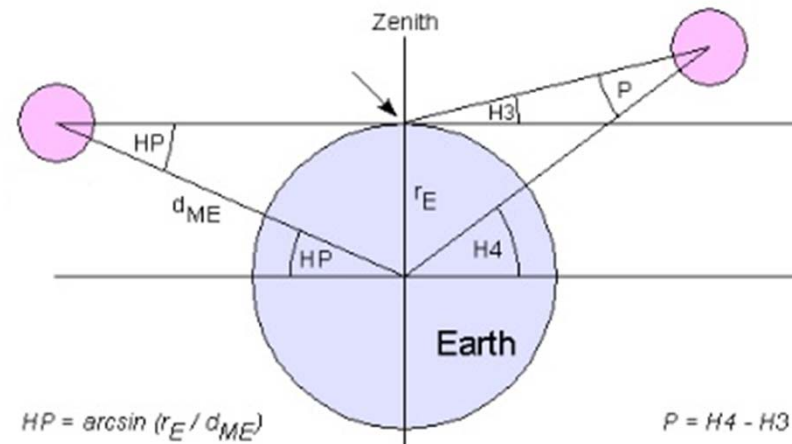
## 4.2.1. CORRECCION REFRACION ATMOSFERICA

- Error debido a la refracción atmosférica con diferentes factores de corrección:





## 4.2.2. CORRECCION PARALAJE

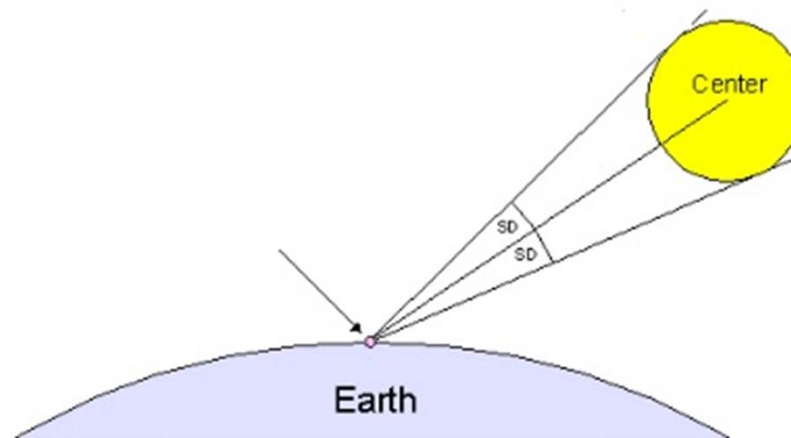


Calculamos el paralaje con:

$$P = \arcsen(\sin HP \cdot \cos \alpha) \approx HP \cdot \cos \alpha$$

Error máximo con HP (0.15' para el Sol) y que equivale a 0,0024°.

## 4.2.3. CORRECCION SEMIDIAMETRO SOLAR



Varia entre 15,7' y 16,3' , así que cogeremos la media que es 16' y equivale a  $0,256^\circ$ .

## 4.3. METODO ELECCION LATITUD

- En el cálculo de la latitud, obtenemos dos valores, latitud Norte y latitud Sur, para elegir entre ambas en cual esta nuestro robot, debemos medir el acimut real y compararlo con el acimut teórico.
- Para calcular el acimut teórico obtenemos la formula del triángulo de navegación:

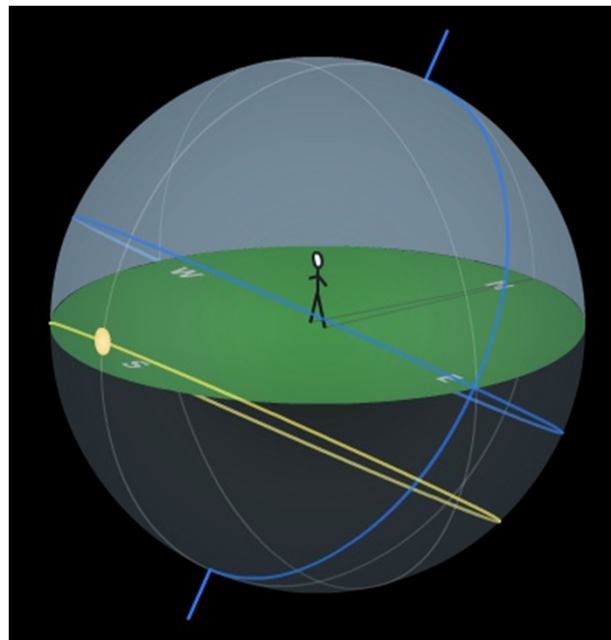
$$\sin \delta = \sin \Phi \cdot \sin \alpha + \cos \Phi \cdot \cos \alpha \cdot \cos \Psi$$



$$\Psi = \arccos \frac{\sin \delta - \sin \Phi \cdot \sin \alpha}{\cos \Phi \cdot \cos \alpha}$$

# 7. LIMITACIONES

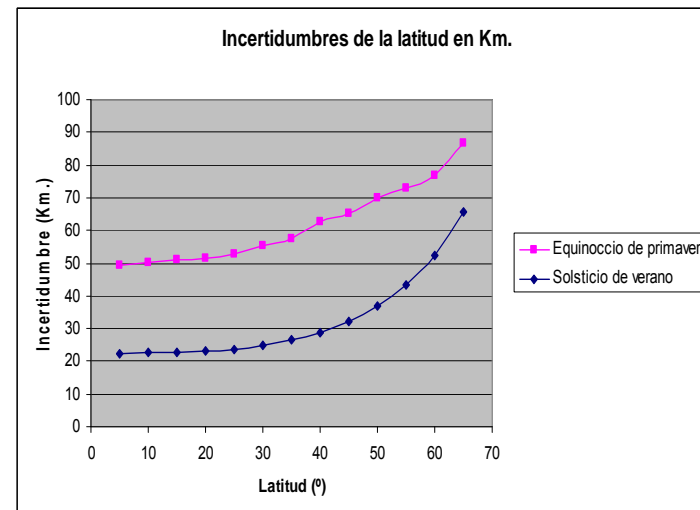
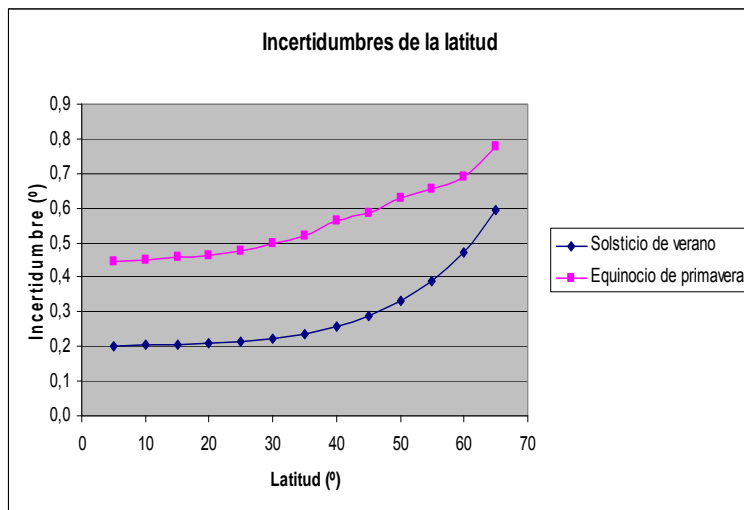
- Recorrido del Sol durante el solsticio de invierno:  
Latitud 60° Norte



APLICACION ROBOTICA DE ORIENTACION SOLAR

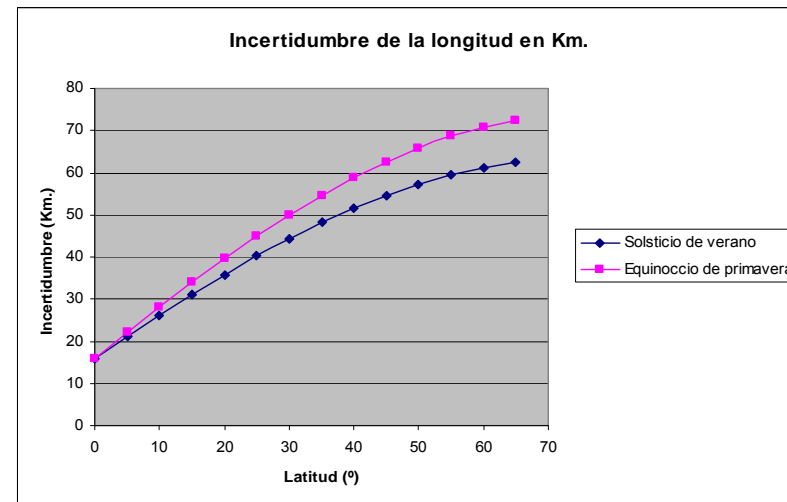
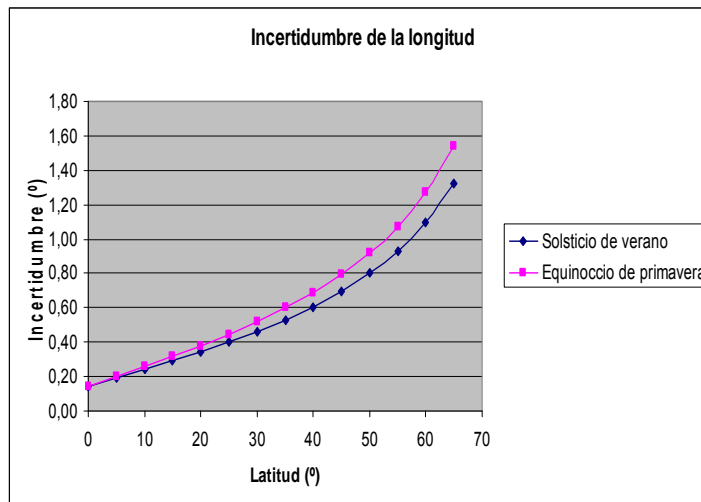
# 8. INCERTIDUMBRES

- En la siguientes gráficas vemos como varía la incertidumbre de la latitud dependiendo de la latitud a la que se encuentra el observador.



# 8. INCERTIDUMBRES

- En la siguientes gráficas vemos como varía la incertidumbre de la longitud dependiendo de la latitud a la que se encuentra el observador.



APLICACION ROBOTICA DE ORIENTACION SOLAR

# CONCLUSIONES Y FUTURAS MEJORAS

- CONCLUSIONES
  - El sistema de orientación capaz de indicarnos la latitud y la longitud para latitudes comprendidas entre 60° Norte y 60° Sur.
  - La incertidumbre aumenta conforme nos vamos alejando del ecuador.
- FUTURAS MEJORAS
  - Complementar con sistema de orientación basado en la posición de las estrellas.

APLICACION ROBOTICA DE ORIENTACION SOLAR