Trabajo de Fin de Grado

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Instalación de Pan and Tilt en Rosbot 2.0 Pro

Autor: Fernando Castro Dorado

Tutores:

José Ramón Domínguez Frejo

Javier García Martín

Dpto. de Ingeniería de Sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo de Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Instalación de Pan and Tilt en Rosbot 2.0 Pro

Autor:
Fernando Castro Dorado

Tutor:
José Ramón Domínguez Frejo
Javier García Martín

Dpto. de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo de Fin de Grado: Instalación de Pan and Tilt en Rosbot 2.0 Pro

Autor: Fernando Castro Dorado

Tutor: José Ramón Dominguez Frejo
Javier García Martín

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Cuando estaba escribiendo los últimos párrafos de este TFG tuve una sensación extraña, pues estaba un paso más cerca de cerrar una de las etapas más importantes por las que he pasado en mi vida: la carrera universitaria. Llegados a este punto, lo único que me sale de dentro es mirar hacia atrás, concretamente hace 4 años; cuando un "adolescente", que no sabía ni conectar dos LEDs en una *protoboard*, preparaba con entusiasmo y, en parte con cierto respeto, las maletas para salir por primera vez de su pueblo para empezar un nuevo capítulo en su vida. Es increíble pensar como cuatro años se han hecho tan cortos como un suspiro y, al mismo tiempo tan intensos, llenos de recuerdos y lecciones de vida.

Sin embargo, quienes hacen realmente especial una etapa de la vida es la gente que te acompaña en el camino y, en mi caso en particular solo me cabe estar agradecido porque no podría haber estado acompañado de mejores personas. Por este motivo quiero aprovechar este espacio para darles las gracias.

En primer lugar, estoy muy agradecido a mi familia, que ha estado ahí apoyando y no solo con el esfuerzo económico que ha permitido que esté escribiendo estas líneas ahora. A mi madre, por inculcarme lo importante que es tener amor propio, ser exigente consigo mismo y por ayudarme cada día buscar una mejor versión de mí mismo; a mi padre, por enseñarme a no rendirme nunca incluso aunque las condiciones externas no estén a mi favor; a mi hermana Sofía, por estar siempre intentando consolarme cuando las cosas no salen como las he previsto y por ser una de las mayores alegrías de mi vida y; como no, a mi abuela Pepa por ser mi mayor fan y estar cuidándome y apoyándome en todos los momentos importantes por los que he pasado desde que era un niño.

Por otro lado, quiero dar gracias a todos los amigos que he conocido, pero especialmente a aquellos que llevan 4 años haciendo el papel de familia. A Jesús, por ser esa persona que siempre está ahí para ayudar sea lo que sea y aportar su punto de vista que le hace tan especial; a Manu, por hacerme reír siempre y alegrar los días más tristes; a Diego, no solo por ser el mejor compañero de clase que pude tener, sino por ser como ese hermano con el que te peleas todos los días pero que sabes que es imprescindible. También a Ale y a Carlos.

En general, doy gracias por toda la gente que haya pasado por mi vida durante estos años, bien sea de paso o no, porque todas y cada una de ellas han aportado su granito de arena para ayudarme a convertirme en la persona que soy hoy.

También quisiera dar las gracias a *Solar Mems Technologies* por todo lo que he aprendido allí durante los últimos meses y, en especial a mi tutor Antonio.

Por último, quiero agradecer a la Universidad de Sevilla haber sido mi punto de partida y a la ETSI por haberme ayudado a espabilar y a aprender a lidiar con los problemas. En especial quiero darles las gracias a José Ramón y a Javier por haber sido mis tutores durante estos meses y haberme ayudado con cualquier duda que me surgiese.

Fernando Castro Dorado

Alumno de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla

Sevilla, 2022

Resumen

El objetivo de este trabajo fin de grado será emplear conocimientos de programación, electrónica, control y robótica para dotar al UGV *Rosbot 2.0 Pro* de un sistema de medición de radiación solar avanzado. La idea será implementar el sistema en su conjunto para aplicaciones de eficiencia solar.

El sistema de medición se basará en acoplar un sensor solar de la compañía *Solar Mems Technologies* a una estructura mecánica denominada *Pan & Tilt*. Dicha estructura utiliza el movimiento de dos servomotores que permitirá ajustar la posición del sensor solar lo más perpendicular posible a los rayos de Sol. De esta forma conseguiremos obtener una medida más precisa, pudiendo aprovechar así la precisión que puede llegar a proporcionar el sensor solar.

El microcontrolador encargado de realizar el control de nuestro sistema de medida será el *Arduino Nano 33 IoT*. A su vez, este mismo estará comunicado con el sistema operativo del robot móvil (ROS) mediante comunicación serie con el fin de intercambiar datos cada vez que sea requerido. Por su parte, el UGV, mediante el entorno ROS, será el encargado de realizar los cálculos necesarios para el control.

Queremos obtener conclusiones sobre si, utilizando este sistema medida, somos capaces de estimar la radiación solar de forma más precisa

Abstract

The aim of this Final Degree Project will be to use knowledge of programming, electronics, control and robotics to provide the UGV *Rosbot 2.0 Pro* UGV with an advanced solar radiation measurement system. The goal will be to implement the system for Solar efficiency applications.

The measurement system will be based on coupling a solar sensor from *Solar Mems Technologies* to a mechanical structure called *Pan & Tilt*. This structure uses the movement of two servomotors to adjust the position of the solar sensor as perpendicular as possible to the Sun's rays. Thereby, we will be able to obtain an accurate measurement, thus being able to improve the accuracy that the solar sensor can provide.

The microcontroller in charge of controlling our measurement system will be an *Arduino Nano 33 IoT*. It will be also connected to the operating system of the mobile robot (ROS) through serial communication to exchange data whenever necessary. The UGV, through the ROS environment, will be in charge of performing the required calculations for control.

We want to obtain conclusions about if it is worth to use the measurement system to estimate an accurate value for solar radiation.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A^*	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
e.o.c.	En cualquier otro caso
e	número e
IRe	Parte real
IIm	Parte imaginaria
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
arcsen	Función arco seno
arcos	Función arco coseno
sen	Función seno
$\sin^x y$	Función seno de x elevado a y
$\cos^x y$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
∂y ∂x	Derivada parcial de y respecto
$x_\circ$	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
<	Menor o igual
>	Mayor o igual
\	Backslash
$\Leftrightarrow$	Si y sólo si

1 INTRODUCCIÓN

Tenemos un práctico reactor de fusión en el cielo
llamado Sol, no tienes que hacer nada, solo funciona.
Aparece todos los días.

- Elon Musk -

En la actualidad vivimos en un mundo donde uno de los mayores peligros a los que nos enfrentamos es la contaminación, una contaminación que altera el curso de la vida y de la naturaleza que es innegablemente debida a la acción del ser humano en el ambiente. Esta trae consigo consecuencias nefastas: se origina un cambio climático debido al efecto invernadero que produce alteraciones en el medio acabando con cantidad de especies vegetales; la fauna local se ve obligada a irse y, en ocasiones quedan condenadas a la extinción.

Las centrales de energías no renovables tradicionales (centrales de ciclo combinado, carbón petroquímicas, etc) no es más que otro medio de producir contaminación ambiental; además, la explotación de los yacimientos para obtener combustible produce la contaminación de aguas y suelos rompiendo así el ciclo del carbono [1]. Sin embargo, este no es su único factor contaminante, pues el transporte del combustible conlleva emisiones de efecto invernadero con también graves impactos en la naturaleza (gaseoductos, oleoductos, etc); por no hablar de todos las emisiones de gases nocivos resultantes del proceso de combustión de las centrales (CO_2, NOx, SOx, COVs entre otros tantos) y que son expulsados al aire [2] que respiramos. Esto desemboca en resultados graves de salud aumentando el riesgo de infecciones respiratorias, enfermedades cardíacas, derrames cerebrales y cáncer de pulmón [2].

Además, está más que demostrado que no podremos vivir de estos tipos de energía eternamente pues, como su propio nombre indica, son limitadas y deberán ser sustituidas en cualquier momento, dándonos problemas de abastecimiento de energía. En el gráfico de la figura 1-2 [3] vemos el tiempo estimado que le quedan a estos recursos.

Por estos motivos se debe buscar una alternativa que necesariamente pasa por el uso de energía renovables, a pesar de que, actualmente, aún no sean más eficientes que las tradicionales. No obstante, implantar este tipo de energía de forma íntegra es un cambio lento que se ha estado llevando a cabo a lo largo de los años ya que, no solo es un problema para la ciencia conseguir aumentar su eficiencia, sino que también hay cambiar la mentalidad de una sociedad aún apegada a las energía no renovables.

Son numerosas las ventajas que presentan las renovables [4]:

- Combaten el cambio climático disminuyendo la contaminación.
- Ayudan a ahorrar ciertos recursos naturales.
- Son seguras tanto para las personas como el medio ambiente y generan menos gases de efecto invernadero y menos residuos tóxicos.
- Son inagotables, lo que solucionará el problema de la falta de abastecimiento a corto plazo.

- Económicamente su coste es previsible y se puede planificar, ya que no permite la especulación de los mercados al ser inagotables

Figura 1-1: Anuario Statistical Review of World Energy 2006

No obstante, no todos son ventajas en las energías renovables, pues actualmente presentan algunos inconvenientes que se plantean como desafíos [5]. Algunos de estos retos son:

- Depende directamente de fenómenos atmosféricos. En las centrales tradicionales la cantidad de potencia generada puede ser, bajo ciertos límites, controlada por la acción del ser humano en cualquier momento. Sin embargo, el clima escapa del alcance del hombre y debemos basarnos en predicciones. Por otra parte, los ecosistemas particulares de cada zona hacen imposible que todos los países puedan obtener las mismas cantidades de potencia.
- Hoy en día no se alcanza la misma eficiencia que en las centrales de energía no renovables.
- Presenta problemas de almacenamiento, por lo que es necesario consumir la energía prácticamente en el momento.
- Hay algunas centrales que requieren mucho espacio (varias hectáreas) como los campos eólicos o centrales solares

La ciencia y la ingeniería llevan años lidiando con estos problemas planteando y llevando a cabo estudios y proyectos (como el que veremos a continuación) en este campo. Afortunadamente, a pesar de que no es un camino fácil, se están realizando grandes avances en las últimas décadas.

Desde un punto de vista económico, España podría ser un país que, gracias a las renovables puede prosperar. La gran variedad de climas que presenta en las distintas partes del país le permite poder aprovechas las condiciones ambientales a su favor para captar grandes cantidades de potencia con el fin de poder abastecerse y, no solo ahorrar comprarles energía a otros países, sino que también podría vender la energía restante no utilizada.

Es importante mirar este cambio desde el punto de vista económico ya que, lamentablemente, en el mundo en el que vivimos todo gira en torno a intereses políticos y económicos. Sin embargo, personalmente considero que este cambio debe hacerse desde el corazón y con el convencimiento de que es nuestra responsabilidad como seres vivos racionales que habitamos en el planeta y que lo necesitamos más que este a nosotros.

La solución está en todos, aportando con cada pequeño detalle y, en mi caso en particular, como futuro ingeniero, enfocando mi carrera a trabajar por lograr un mundo mejor y que opere con una tecnología limpia y eficiente.

Antes de comenzar, para poder entender la idea de este trabajo, primero debe ser contextualizado dentro de la aplicación donde va a ser destinado. Para ello, es inevitable mencionar el “proyecto de investigación” en el que está enmarcado.

Dicho proyecto, llamado ***OCONTSOLAR*** **(**Optical Control of Thermal Solar Energy Systems) es un Proyecto Europeo liderado por el catedrático Eduardo Fernández Camacho que cuenta con varios investigadores, entre ellos los tutores de este trabajo de fin de grado: José Ramón Domínguez Frejo y Javier García Martín.

El proyecto tiene como objetivo desarrollar nuevos métodos de control para usar sensores móviles montados en drones (o UAV) y vehículos terrestres no tripulados (UGV) como parte fundamental del control de sistemas.

Las plantas solares serán usadas para casos de estudio, con el objetivo de optimizar su operación usando predicciones y estimaciones de la irradiancia espacial. No obstante, los resultados obtenidos podrán ser aplicados a otro tipo de sistemas como es del control de tráfico en autovías y grandes ciudades, administración de energía en construcciones, etc.

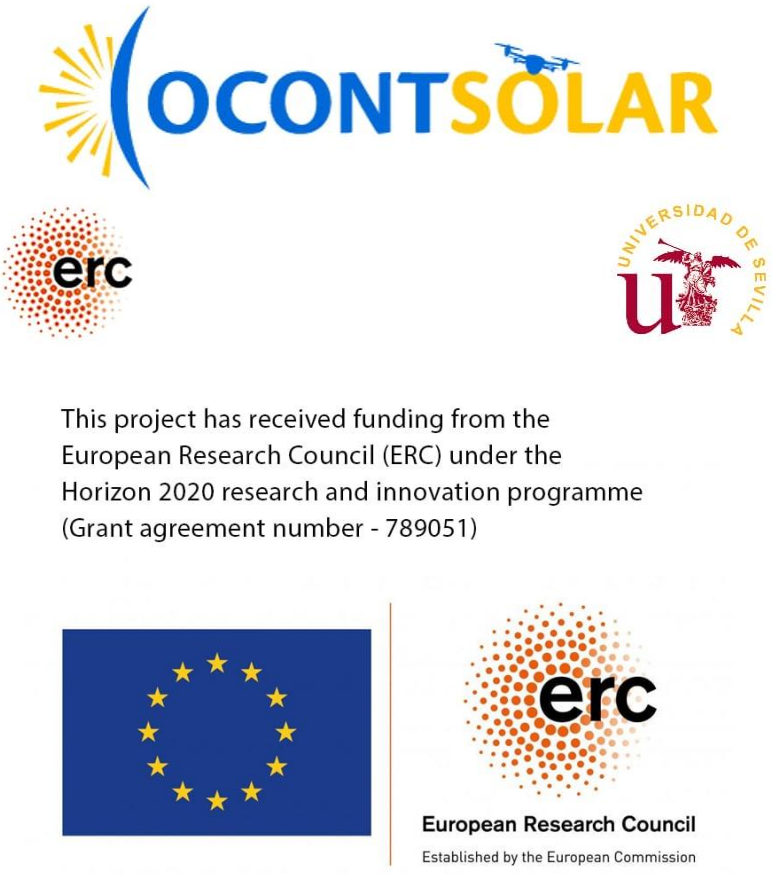

Figura 1-2: Introducción al proyecto

Los principales objetivos del proyecto son los siguientes:

- Desarrollo de métodos de control de una flota de sensores móviles como parte esencial del control de sistemas.
- Desarrollo de métodos de estimación de irradiancia solar espacial con una flota de sensores montados en UAV's y UGV's.
- Nuevos algoritmos de control predictivo en los que se usen sensores solares móviles que ayuden a poder conseguir una mayor eficiencia en las centrales termosolares.

Es en este último punto donde se va a encajar este TFG, pues el fin será crear un sensor de irradiancia solar móvil formado por un UGV y un sistema de medición con el sensor acoplado al mismo.

En un futuro, podría ser incorporado en otros robots con el fin de conseguir crear una flota que creen una red de

sensores móviles inalámbricos que puedan trabajar de forma autónoma en una plata solar para poder realizar las labores de estimar la radiación solar en ellas.

2 ESTADO DEL ARTE

Este capítulo va a tratar, resumidamente, de poner de manifiesto algunas de las investigaciones realizadas acerca de los temas que vamos a tratar en este proyecto y las aplicaciones donde irá destinado.

Por un lado, vamos a explicar uno de los conceptos más importantes para el trabajo: las ecuaciones nos van a ayudar a estimar la posición del Sol con respecto a un punto de la Tierra, aunque, para simplificar, en este proyecto nos referiremos a ellas como "ecuaciones solares". Estos cálculos van a ser primordiales para el desarrollo del proyecto debido a que van a ser directamente implementados en los algoritmos del robot.

Por otro lado, este trabajo va a ser dedicado a la elaboración de un sensor móvil que, en este caso se tratará de un sensor solar incorporado a un UGV. Todo esto tiene la finalidad de poder crear más sensores móviles que formen una red inalámbrica de sensores móviles formadas por vehículos no tripulados que se comuniquen entre sí.

Es por esta razón que le vamos a dedicar una sección de este capítulo a las redes de sensores móviles; así conseguiremos ubicar nuestro trabajo dentro de este contexto.

2.1 Ecuaciones Solares

2.1.1 Introducción

Estas ecuaciones sirven para calcular la altitud y la azimut del Sol en un punto determinado de la superficie terrestre. Estos dos ángulos que tenemos como incógnita nos definirán la posición del Sol desde un cierto sistema de referencia. De esta forma, tan solo debemos que girar el servo *Pan* los ángulos que indique la azimut y el servo *Tilt* el ángulo de altitud hallado.

No obstante, antes de proseguir con las ecuaciones, vamos a definir que son estos dos parámetros que vamos a tratar de obtener.

- **Azimut**: Es el ángulo que forma el Norte y el cuerpo celeste (el Sol en nuestro caso), medido en el sentido de rotación de las agujas de un reloj alrededor del horizonte de la persona. Este ángulo nos permite determinar la dirección de un cuerpo celeste, es decir, si este está en el Norte su acimut será de 0º, al Este 90º, al Sur 180º y al Oeste 270º. La figura 2-1 muestra que es la azimut visualmente.

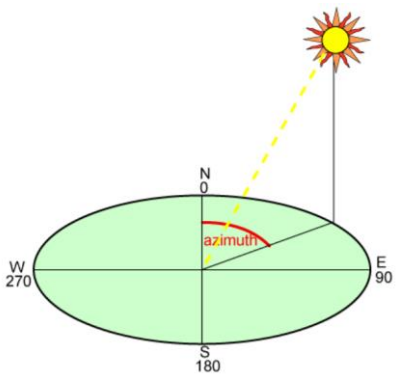

Figura 2-1: Azimut.

- **Altitud**: Se trata del ángulo que forman el cuerpo celeste (siendo el Sol el que nos interesa) y el horizonte del sistema de referencia del observador. La figura 2-2 muestra un dibujo en el que se muestran los ángulos de altitud (β) y azimut (α) para poder diferenciarlos bien.

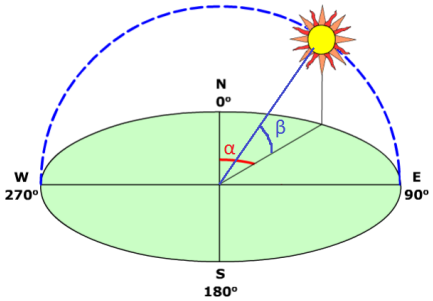

Figura 2-2: Ángulos de azimut (α) y altitud (β).

2.1.2 Desarrollo de las ecuaciones

Después de haber explicado que son los ángulos de altitud y azimuth vamos a pasar a explicar cuales son los cálculos necesarios que hay que seguir para poder obtenerlos[7]. Para ello, primero vamos a explicar cual va a ser el sistema de referencia en el que vamos a trabajar. En la figura 2-3 vemos un dibujo que nos ilustra la escena de un rayo de Sol (simulado con el vector S) llegando a la Tierra. Dicha figura nos va a ayudar a comprender que es lo que vamos a hacer y que es lo que estamos calculando en cada momento.

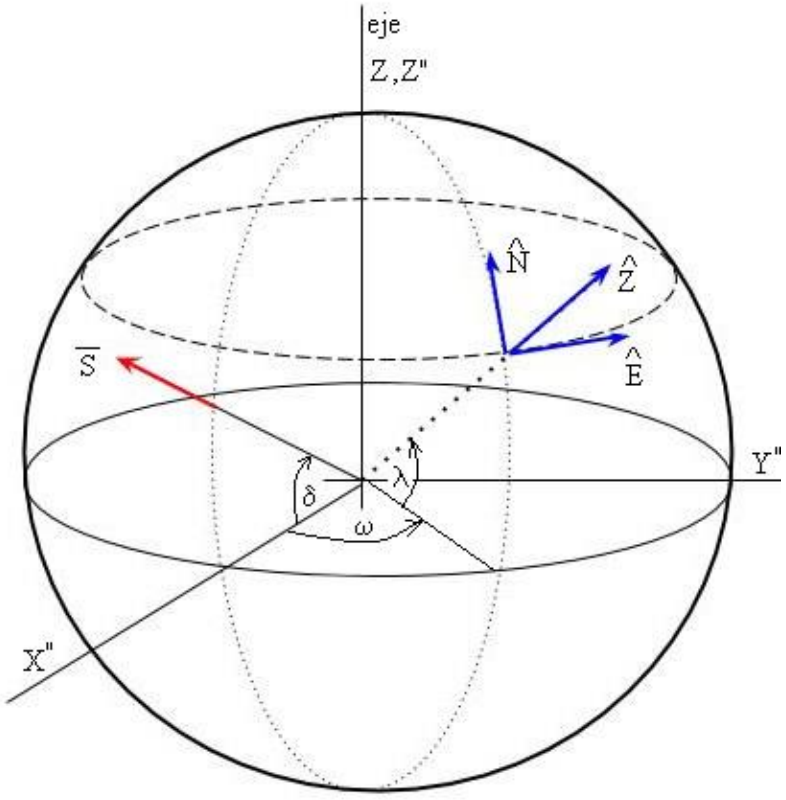

Figura 2-3: Vector Solar proyectado sobre el centro de la Tierra.

Antes de proseguir vamos a detallar que es cada uno de los parámetros que se pueden observar en la figura:

- ***δ***: Se le denomina *declinación* y hace referencia al ángulo que forma el vector S (siempre contenido en el plano X''Z'') con el eje X''.
- ***ω***: Se trata del *ángulo horario* y es el ángulo que se formado en el plano X''Y'' con respecto al eje Y' que indica donde se encuentra un punto determinado de la superficie terrestre que, en nuestro caso será el punto donde se encuentra el observador.
- ***λ***: Es la *latitud* que, como sabemos, indica la inclinación vertical de un punto de la superficie terrestre

con respecto al plano del Ecuador.

- ***ONEZ*:** Es el sistema referencia de un punto de la superficie terrestre para cual obtenemos la dirección de los puntos cardinales.

En la figura vemos como un rayo de Sol llega a la Tierra y el proyectado sobre los ejes del centro de la Tierra, que son los que forman el sistema de referencia móvil OX''Y''Z'' donde los vectores serán i'', j'' y z''. Si proyectamos el vector referido al rayo solar S, vemos que tendría la siguiente forma:

$$S = \cos(\delta) * i'' + \text{sen}(\delta) * j''$$

La idea de las Ecuaciones Solares será poder coger este vector, proyectado sobre el sistema de referencia móvil del centro de la Tierra, y proyectarlo sobre el sistema de referencia de un punto cualquiera de la superficie de la Tierra, el cual será donde se encuentre nuestro observador. Es decir, queremos ver el vector S desde el sistema de referencias *ONEZ*.

A continuación, vamos a proseguir con el cálculo de nuestras ecuaciones. Lo primero que vamos a calcular va a ser calcular la declinación solar. Hay varias formas de poder calcularlas, algunas de ellas se muestran a continuación [24]:

- *Perrin de Brichanbaut (1975):*

$$\delta = ArcSin(0{,}4 * \text{sen}\ (2 * \pi * \frac{dn - 82}{365}))$$

- *Cooper (1969):*

$$\delta = 23{,}45^o * \left(\frac{\pi}{180^o}\right) * \text{sn}\ (2 * \pi * \frac{284 + dn}{365})$$

- *García (1994):*

$$\delta = -23{,}45^o * \left(\frac{\pi}{180^o}\right) * \cos\ (2 * \pi * \frac{9 + dn}{365})$$

- *Spencer (1971):*

$$\delta = 0.006918 - 0.399912 * \cos(\tau) + 0.070257 * \text{sen}(\tau) - 0.0067 * \cos(2\tau) + 0.0009907 * \text{sen}(2\tau) - 0.002697 * \cos(3\tau) + 0.00148 * \text{sen}(3\tau)$$

En estas fórmulas propuestas, el parámetro *dn* se refiere al día juliano del año, que puede variar entre 1 y 365. A su vez el parámetro τ hacer referencia a:

$$\tau = \text{dn} * \frac{2 * \pi}{365}$$

De todas las fórmulas que acabamos de mostrar, vamos a quedarnos con la propuesta por Spencer, ya que es más eficiente computacionalmente [ref] y, como queremos implementarlo en un robot, este hecho nos interesará notablemente. Además, a la hora de estimar la declinación solar mediante los modelos de García o de Cooper, tendremos menos precisión y menos continuidad [ref] con respecto al establecido por Spencer. En la figura 2-4 vemos que el valor de la declinación estimada por los métodos de García y Cooper presenta una diferencia promedio de un 20% con respecto a la de Spencer.

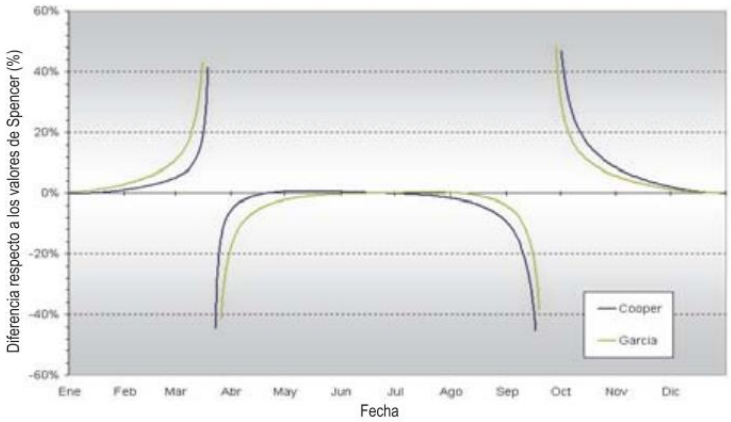

Figura 2-4: Gráfico de comparación de fórmulas de García y Cooper con respecto a Spencer.

En la figura 2-5, vamos a ver como varía la declinación (obtenida mediante la ecuación de Spencer y de Cooper) a lo largo del año sabiendo que esta dependerá directamente del día juliano en el que nos encontremos.

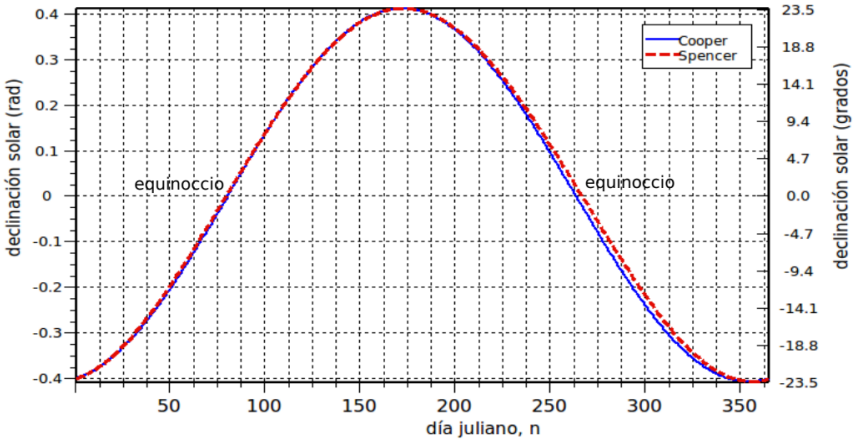

Figura 2-5: Variación de la declinación solar a lo largo del año

Resumiendo, hasta ahora hemos visto como obtener el ángulo referido a la declinación (δ) que será el ángulo con el que llega el rayo solar con respecto al plano ecuatorial de la Tierra. A continuación, nos tocará tratar el tema del punto situado sobre la superficie terrestre. Como ya hemos visto previamente, dicho punto queda definido mediante dos ángulos: la latitud (ω) y el ángulo horario (λ).

Comenzamos calculando ω. Para hallar este parámetro necesitamos introducir la ecuación del tiempo (E_t), fórmula que necesitaremos para nuestros cálculos.

Para entender que es la ecuación del tiempo, debemos conocer el concepto de tiempo solar verdadero; este se basa en la rotación de la Tierra sobre su eje polar y el movimiento de traslación alrededor del Sol. Por otra parte, un día solar se refiere al tiempo en el que el Sol completa un ciclo alrededor de un observador estacionario que se encuentre en la Tierra. Sin embargo, debido al movimiento en una órbita elíptica y a la inclinación de su eje de rotación con respecto al plano de la eclíptica la Tierra no rota a velocidad constante y la duración del día solar difiere en unos minutos (mas o menos) de las 24 horas nominales.

Para poder modelar este error de manera matemática, se crea el término “tiempo solar medio”. Este es el intervalo de tiempo resultante de una esfera terrestre ficticia de la que suponemos que tendrá un movimiento de rotación uniforme o constante alrededor de nuestra estrella, el Sol. De esta forma, Spencer desarrolló otra fórmula que calculaba el error o diferencia que se crea entre el tiempo solar medio y el tiempo solar verdadero. Esta fórmula, mostrada a continuación (con resultado expresado en radianes, aunque al multiplicar por 229,18 lo pasamos a minutos), varía en función del día en el que nos encontremos:

$$E_T = (0{,}000075 + 0.001868 * \cos(\tau) - 0.032077 * \text{sen}(\tau) - 0.014615 * \cos(2\tau) - 0.04089 * \sin(2\tau)) * 229{,}18$$

En la figura 2-6 vemos una gráfica que muestra la variación anual de la ecuación de tiempo. Habiendo calculado la ecuación del tiempo, podemos obtener el valor del tiempo solar aparente mediante la siguiente expresión:

$$Ts = Hora\ oficial - diferencia\ por\ mes - corrección\ longitud + E_T$$

Donde el término "diferencia por mes" es la diferencia producida en el horario en función de la estación en la que nos encontremos, sabiendo que el horario de invierno y de verano no son el mismo. Por otro lado, "corrección de longitud" tiene que ver con la longitud del punto de la superficie terrestre donde nos encontremos y tiene un valor igual a *abs(4*longitud)*.

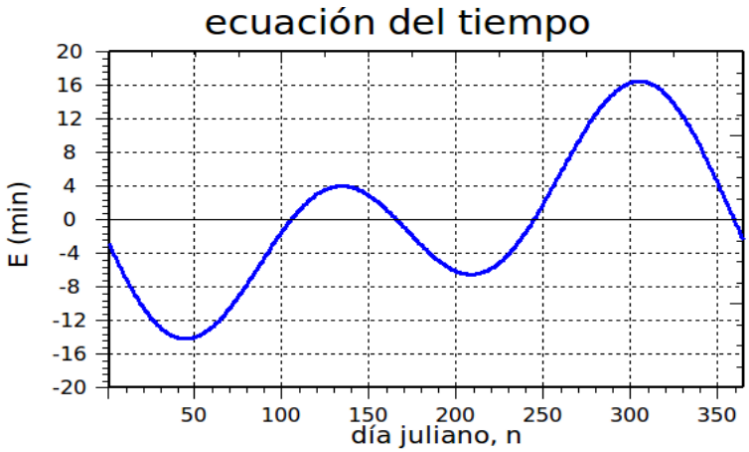

Figura 2-6: Variación de la ecuación del tiempo a lo largo del año.

Después de haber calculado la hora solar verdadera, recordando nuestro objetivo principal, procedemos a continuar hallando el ángulo horario ω. Este ángulo está ligado a la hora solar aparente y se calcula como:

$$\omega = \big((Ts - 12) * 15\big) * \frac{\pi}{180}$$

Finalmente, el último parámetro que nos queda por definir sería la latitud λ; sin embargo, este será un dato que podemos obtener sin necesidad de recurrir a ecuaciones y que dependerá del punto terrestre en el que nos encontremos.

2.1.3 Proyección del vector solar S

Tras haber obtenido todas las incógnitas necesarias para definir nuestro punto de observación en la superficie de la Tierra, tan solo nos quedaría crear el sistema de referencia referido al mismo y proyectar sobre él el vector solar.

Recordamos que nuestro sistema de referencia móvil está formado por los vectores unitarios *E*, *N* y *Z* que son

perpendiculares entre sí y hacen referencia a los puntos cardinales. Lo primero que haremos será intentar proyectar estos ejes sobre el sistema de referencia local *OX''Y''Z''*, con vectores unitarios *i''*, *j''* y *k''*.

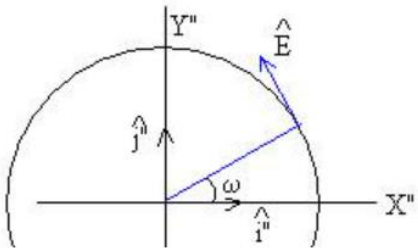

Figura 2-7: Vector unitario E en el plano OX''Y''.

Si nos fijamos en la figura 2-7, por trigonometría podemos obtener el vector unitario E gracias al ángulo horario ya calculado mediante la siguiente expresión:

$$\hat{E} = -\operatorname{sen}(\omega) * i'' + \cos(\omega) * j''$$

Para obtener el vector Z, vamos a hacer uso de las coordenadas esféricas [8]. Recordamos que las coordenadas esféricas son un sistema de coordenadas tridimensional. Este sistema tiene la forma (ρ, θ, φ), en donde, ρ es la distancia desde el origen hasta el punto, θ es el ángulo formado con respecto al eje x y φ es el ángulo formado con respecto al eje z.

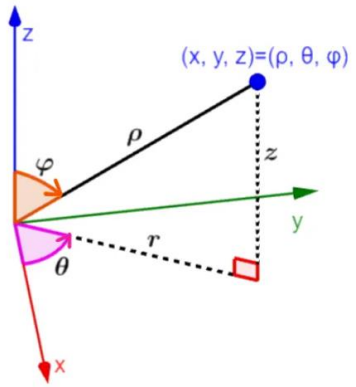

Figura 2-8: Sistema de coordenadas esféricas.

En este sistema de coordenadas, los valores correspondientes a lo que sería el sistema de referencia cartesiano son:

$$x = \rho * \operatorname{sen}(\varphi) * \cos(\theta)$$
$$y = \rho * \operatorname{sen}(\varphi) * \operatorname{sen}(\theta)$$
$$x = \rho * \cos(\varphi)$$

Con estos datos, podemos caracterizar nuestro problema; sabiendo que el eje Z es el que va desde el origen de coordenadas en el centro de la Tierra hasta el punto y que su es de módulo unitario (ρ = 1, φ= 90 - λ y θ=ω), nos queda la expresión mostrada a continuación:

$$\hat{Z} = \cos(\lambda) * \cos(\omega) * i'' + \cos(\lambda) * \operatorname{sen}(\omega) * j'' + \operatorname{sen}(\lambda) * k''$$

Finalmente, con E y con Z, podemos hallar el último eje que nos queda, N, utilizando la herramienta del producto vectorial ya que sabemos que este último debe ser perpendicular a los primeros mencionados.

$$\hat{N} = \hat{Z}\ x\ \hat{E} = \begin{vmatrix} i'' & j'' & k'' \\ \cos(\lambda) * \cos(\omega) & \cos(\lambda) * \sin(\omega) & \sin(\lambda) \\ -\sin(\omega) & \cos(\omega) & 0 \end{vmatrix}$$

Finalmente obtenemos que el valor de N en función de las coordenadas cartesianas es:

$$\hat{N} = -\text{sen}(\lambda) * \cos(\omega) * i'' - \cos(\lambda) * \text{sen}(\omega) * j'' + \cos(\lambda) * k''$$

Una vez que hemos obtenido el sistema de coordenadas de nuestro punto, ya podremos finalmente proyectar el vector S sobre el mismo haciendo uso del producto escalar

$$S_N = \vec{S} * \hat{N} = -cos(\omega) * sen(\lambda) * cos(\delta) + cos(\lambda) * sen(\delta)$$
$$S_E = \vec{S} * \hat{E} = -sen(\omega) * cos(\delta)$$
$$S_Z = \vec{S} * \hat{Z} = cos(\omega) * cos(\lambda) * cos(\delta) + sen(\lambda) * sen(\delta)$$

Obtenemos así el vector S proyectado sobre el sistema de referencia ONEZ, que queda como:

$$\vec{S} = (-cos(\omega) * sen(\lambda) * cos(\delta) + cos(\lambda) * sen(\delta)) * \hat{N} - sen(\omega) * cos(\delta) * \hat{E} + (cos(\omega) * cos(\lambda) * cos(\delta) + sen(\lambda) * sen(\delta) * \hat{Z}$$

Finalmente, recordando el significado de azimut y elevación, nos damos cuenta de que el vector solar puede expresarse en función de estos dos ángulos, hecho que observamos en la figura 2-9.

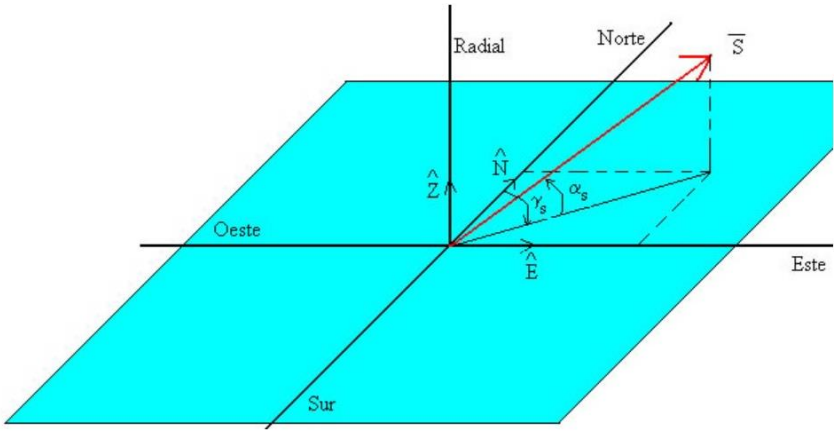

Figura 2-9: Ángulos solar en función de azimut y altitud.

En este caso, el vector solar tomaría una expresión diferente:

$$\vec{S} = cos(\alpha s) * cos(\gamma s) * N + sen(\gamma s) * cos(\alpha s) * E + \text{sen}\ (\alpha s)Z$$

Si queremos averiguar los valores de altitud (αs) y azimut (γs), tan solo debemos resolver el sistema de ecuaciones resultante de igualar término a término este vector con el obtenido anteriormente en función de la declinación, la latitud y el ángulo horario. Como conclusión obtenemos estos valores para la azimut y la altitud:

$$\alpha s = \text{asen}\ (\cos(\lambda) * \cos(\delta) * \cos(\omega) + \text{sen}(\lambda) * \text{sen}\ (\delta))$$

$$\gamma s = \text{acos}\ (\frac{\cos(\lambda) * \text{sen}(\delta) - \cos(\omega) * \text{sen}(\lambda) * \cos(\delta)}{\cos\ (\alpha s)})$$

2.2 Redes de sensores en robot móviles

Como ya se ha comentado previamente, vamos a introducir el campo de las redes inalámbricas de sensores para poder contextualizar uno de los puntos tratados del Proyecto: Crear una flota de UGV's y UAVs que puedan estar trabajando en conjunto, de forma autónoma y en coordinación para estimar la radiación solar.

2.2.1 Tecnología inalámbrica

Hasta ahora hemos estado hablando de redes de sensores, pero todavía no hemos explicado que es. [12]Se trata de una red de ordenadores pequeños, a los que denominaremos "nodos" equipados con sensores y que colaboran entre ellos para la realización de una tarea o más tareas comunes. Estos sensores presentan capacidades sensitivas y de actuación inalámbrica de modo que nos permite formar una red de comunicación sin infraestructura física preestablecida ni administración central.

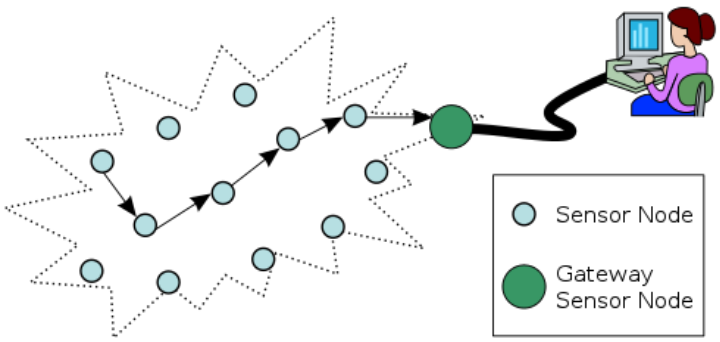

Figura 2-10: Estructura de una red de nodos.

Estas redes de sensores y actuadores inalámbricos se utilizan en muchas aplicaciones industriales, como la monitorización y control de procesos industriales, monitorización de la salud mediante máquinas, sistemas ciberfísicos, domótica, detección ambiental, sistemas de transportes etc.

Actualmente las flotas de redes de sensores móviles son de gran utilidad para numerosas aplicaciones; por ejemplo, en el campo de la agricultura. En este sector se pueden utilizar robots para poder medir variables atmosféricas como la distribución de las precipitaciones sobre el terreno, así como características del mismo

que nos den datos acerca de la fertilidad o estado, como el pH de la tierra etc. Gracias a la obtención de estos

datos se podrán tomar decisiones acerca de que zonas del terreno deben regarse o en cuales merece la pena plantar unas semillas u otras.

En cualquier caso, el pilar básico fundamental para que estas redes funcionen en las tecnología inalámbrica que hay detrás de ellas. El ejemplo de aplicación más común es el caso de los sistemas de transporte inteligentes (ITS). Utilizaremos esta aplicación para explicar el funcionamiento de las redes de sensores móviles [12]. En función del ambiente donde vayan a ser aplicados, su comunicación inalámbrica se divide en dos grupos:

- Aplicaciones que requieran alta seguridad y fiabilidad.
- Aplicaciones que no requieran alta seguridad.

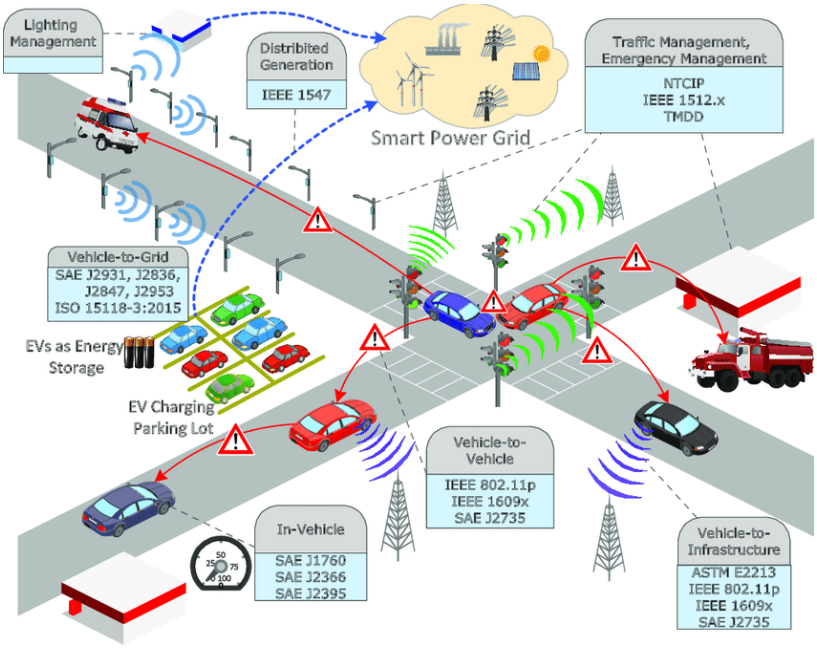

Figura 2-11: Ejemplo ITS con normativa IEEE 802.11.

La diferencia existente entre ambas radica en que la primera nombrada se dedica a enviar mensajes entre los vehículos para que puedan comunicarse entre ellos dándose datos acerca de posibles zonas de accidentes o atascos, estados de la carretera, etc. Por su parte, el segundo grupo está más dedicado al ocio.

Debido a ello, actualmente, para poder utilizar redes de sensores inalámbricas de forma segura, se elaboraron una serie de protocolos de comunicación basados en el estándar 802.11 [13]. Creado por el IEEE, en él encontramos una serie de normas por las que se rigen las comunicaciones inalámbricas. En este conjunto de normas podemos encontrar la arquitectura necesaria para Wireless Access In Vehicular Environments (WAVE) [12] que está enfocado para establecer sistemas de comunicaciones vehiculares y operan entre las bandas de frecuencia de 5,8 MHz y 5,9 MHz con un ancho de banda de 75 MHz.

Dentro de este modo de comunicación distinguimos entre dos tipos:

- Vehicular-To-Vehicular (V2V): La comunicación se realiza entre vehículos.
- Vehicular-To-Infraestructure (V2I): Los nodos pueden comunicarse con infraestructuras con localización fija.

Los sistemas de comunicaciones V2V están libres de usar una infraestructura y tan solo se comunican con los nodos que están dentro de su alcance, para que esto se produzca los vehículos han de estar dotados de sistemas On Board Unit (OBUs) [14], esto quiere decir que los robots tengan montados sobre ellos dispositivos que faciliten las comunicaciones. Con esto podemos crear así redes ad-hoc [15], permitiendo una alta movilidad

dinámica en la red.

Por otro lado, los sistemas V2I llevan incorporados a su vez, no solo sistemas OBUs, sino que también Road-Side Unit (RSUs). Estos últimos hacen referencia a un subsistema a pie de carretera que facilita el intercambio de información entre los vehículos y la infraestructura. Con esta fusión conseguimos que los OBUs puedan intercambiar información con los RSUs y, a su vez, estos puedan conectarse a internet mediante putos de accesos a través de IGWs, adquiriendo así un direccionamiento IP propio.

Esto permite que la gestión de movilidad esté garantizada y los paquetes de nodos sean alcanzables. Podemos ver un ejemplo de esta estructura de nodos en la figura 2-11.

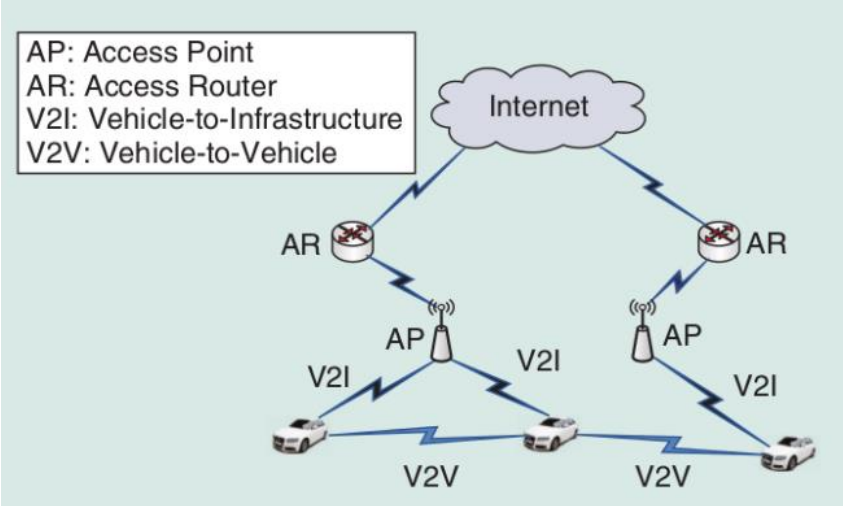

Figura 2-12: Red de estructura WAVE

Aplicando lo que acabamos de ver al caso que nos concierne [28], llegamos a la conclusión de que en el proyecto tendremos una flota de UAVs y UGVs (siendo estos los nodos) entre los cuales exista un protocolo de comunicación V2I de forma que puedan intercambiar datos acerca de las condiciones atmosféricas, posición de otros vehículos, zonas donde la radiación es máxima etc. Por último, cada uno de los robots presentará un protocolo de comunicación V2I para poder enviar esta información al exterior con el fin de monitorizar tareas, medidas y procesos con el objetivo de que se pueda supervisar el correcto funcionamiento de la red de sensores.

2.3 Estimación de la radiación en plantas solares

En esta última sección vamos a enfocarnos en el objetivo principal de la flota de vehículos no tripulados: estimar la radiación solar en una planta solar.

2.3.1 Energía termosolar y fotovoltaica

Hasta ahora hemos estado hablando de plantas de energía fotovoltaicas y plantas termosolares, sin embargo, aunque a priori las primeras de ellas son más conocidas, las segundas también son muy utilizadas. En primer lugar vamos a explicar brevemente la diferencia entre ambos tipos de energía y vamos a ver cuales son las aplicaciones hacia donde van dirigidas.

Como bien se puede intuir, ambos tipos de energía usan al Sol como fuente principal de suministro. El clima que tenemos en España favorece el aprovechamiento de las energías fotovoltaicas y termosolares y, teniendo en cuenta la gran cantidad de horas de luz diarias de las que gozamos, podemos realizar un disfrute máximo de las mismas. [16] Cada una de las energías presenta ciertas diferencias con respecto a la otra y ambas son utilizadas para distintas aplicaciones.

La energía termosolar se obtiene a partir de un proceso térmico, pues utiliza el calor del Sol captado por la placa para poder calentar un líquido. En a energía fotovoltaica, la placa capta la radiación emitida por el Sol y la convierte en electricidad para ser utilizada.

La energía termosolar es usada para normalmente en los siguientes casos [16]:

- Calentar agua o ACS (agua caliente sanitaria).
- Apoyo a los sistemas de calefacción.
- Climatización de piscinas.
- A nivel industrial también puede usarse para generar electricidad, que es caso tratado en el Proyecto.

Por otro lado, la energía fotovoltaica, al igual que la cualquier otro medio que genere energía eléctrica, dependiendo de la potencia generada, será empleada en iluminación, abastecimiento eléctrico de electrodomésticos etc.

Figura 2-13: Placa termosolar (1) y placa fotovoltaica (2).

En la figura 2-12 podemos visualizar un ejemplo de cada una de los tipos de placas que son empleados. El la primera imagen podemos distinguir la placa termosolar ya que vemos un depósito que guardará en su interior en fluido que vaya a ser calentado.

En cuanto al precio y sostenibilidad de ambos tipos de energía, cabe destacar que la energía fotovoltaica será más cara al incluir baterías y circuitería que permitan almacenar la energía y hacerla más eficiente, sin embargo es más sencilla de mantener. Por otro lado, la planta termosolar será más barata pero más difícil de mantener.

2.3.2 Algoritmos de estimación de la radiación solar

Independientemente del tipo de si queremos obtener energía fotovoltaica o termosolar, el punto clave será poder obtener la mayor radiación del Sol posible. Es por ello que, para poder enfocar las placas en una dirección captando la mayor eficiencia, es de vital importancia conocer el valor de la irradiancia y en que zonas de la planta solar alcanza su mayor valor.

Tradicionalmente se han utilizado los pirheliómetros, [17] sensores fijos que miden la componente directa de la radiación solar (llamada DNI por sus siglas en inglés). Esto hace referencia a la irradiancia resultante del flujo radiante solar desde un ángulo sólido definido por el cual el eje está perpendicular al plano de la superficie del receptor

Figura 2-14: Pirheliómetro

No obstante, en la práctica, las medidas obtenidas por este instrumento van a verse afectadas por una serie de factores como es el caso de las sombras que forman las nubes o el viento, que puede atraer nubes de otra dirección. Esto hace que estimar la radiación en un una planta solar de la forma más precisa posible no sea algo directo o trivial. Así es como surge la necesidad de elaborar métodos que nos permitan realizar esta función correctamente; uno de estos métodos pasa por utilizar vehículos no tripulados como sensores móviles y cámaras que, junto a los fijos, puedan facilitar la tarea.

En este caso, para poder explicar como son aplicados este tipo de algoritmos, vamos a basar esta sección en las investigaciones expuestas en un artículo [18].

El planteamiento general de este problema será partir de un mapa de la planta solar en el que vamos a definir varios puntos de medidas (figura 2-14 [18]) donde, tanto robots como sensores fijos (anemómetros y pirheliómetros), se van a encargar de tomar medidas de la radiación, calcular el área que ocupan las nubes mediante las cámaras y estimar cual será la posición de las mismas en el futuros sabiendo la dirección del viento obtenida de los anemómetros.

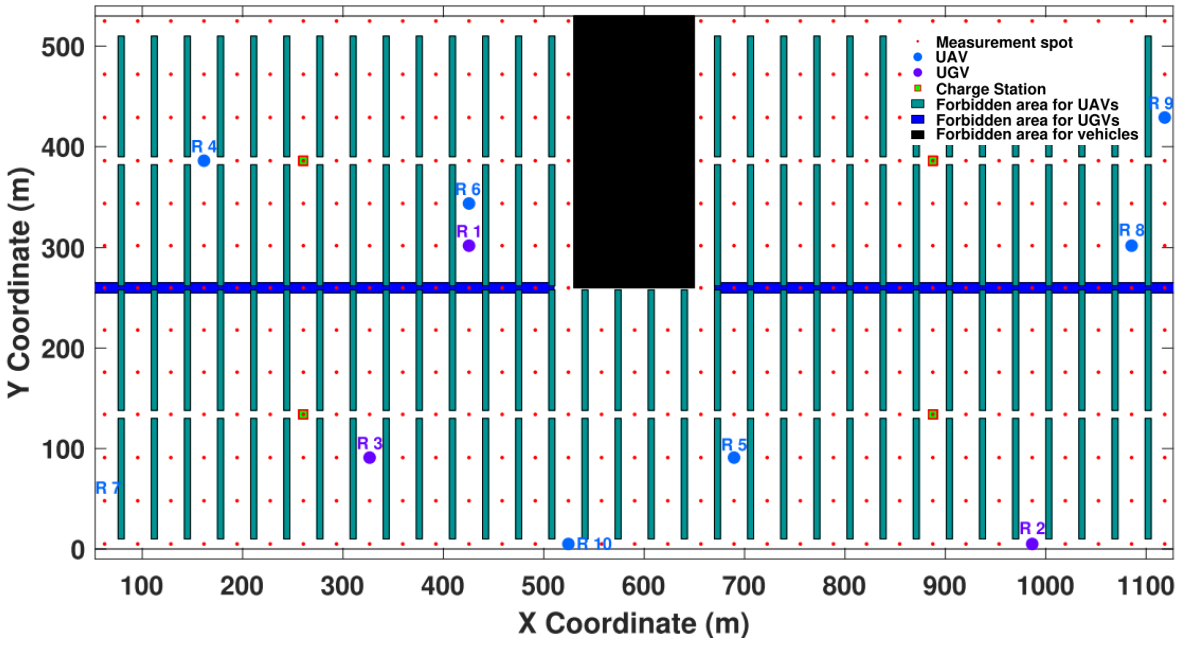

Figura 2-15: Planta solar modelada para el problema.

La idea será utilizar todas estas medidas para poder obtener un mapa de calor de la planta solar donde podamos visualizar cuales son las zonas donde mayor radiación podamos captar y, por tanto, más energía podamos conseguir. Para llegar a este mapa, se pueden aplicar técnicas de control predictivo, como es el caso propuesto en el artículo [18].

Una vez hemos obtenido el mapa de radiación, podemos usar el control predictivo para mejorar la eficiencia en las centrales [25, 26, 29].

3 HARDWARE

En este capítulo vamos a centrarnos en la parte física (o hardware) del sistema, es decir, vamos a describir que componentes forman nuestro sensor móvil. Dentro del sistema conjunto, vamos a distinguir dos secciones: el sistema de medida formado por el *Pan&Tilt* y el robot móvil utilizado, el *Rosbot 2.0 Pro*.

3.1 ROSbot 2.0 Pro

Como ya se ha comentado previamente, el robot empleado para la realización de este sistema se trata del *ROSbot 2.0 Pro*, el cual se muestra en la figura 3-1.

Figura 3-1: ROSbot 2.0 Pro

La idea de haber escogido este robot como pilar de nuestro sensor móvil surge de que el *Dpto de Sistemas y Automática* de la Escuela cuenta con varios ejemplares que utilizan para labores de investigación en el proyecto *OCONTSOLAR*.

3.1.1 Descripción del hardware

El *ROSbot 2.0 Pro* [6] es un robot móvil terrestre de arquitectura diferencial que con cuenta con 4 ruedas; sensores infrarrojos que ayudan a detectar obstáculos; un LIDAR que le permite mapear o localizarse dentro de un mapa, una cámara RGBD, que no solo le otorga visión del entorno, sino que también le facilita la distancia de los objetos que son visualizados; así como sensores de distancia, velocidad y orientación basados en la propia odometría que proporcionan los encoders de las ruedas.

Podemos apreciar las dimensiones del robot en la figura 3-2. De la misma forma, observaremos más características principales como el peso, la autonomía (entre otras) en la tabla. 3-1

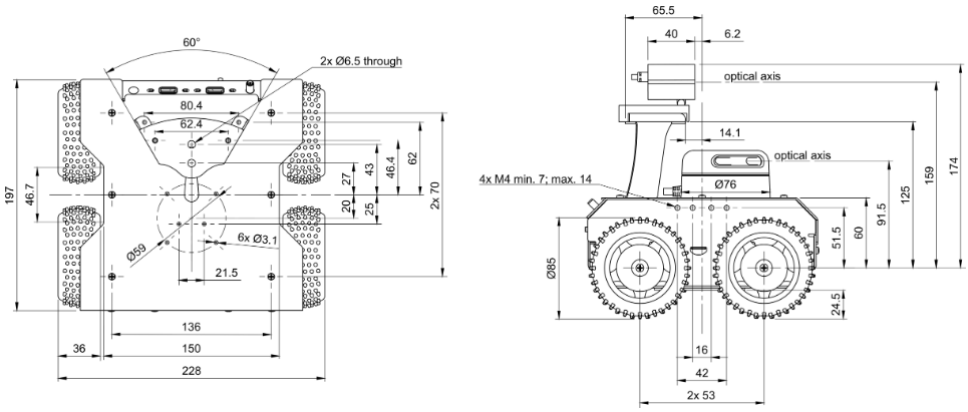

Figura 3-2: Dimensiones del robot.

Parámetro	Descripción
Dimensión con cámara y LIDAR	200x235x220mm [LxWxH]
Dimensión sin cámara	200x235x146mm [LxWxH]
Dimensión sin cámara y LIDAR	200x235x106mm [LxWxH]
Peso	2,84kg (con camara y LIDAR), 2,45kg (sin cámara ni LIDAR)
Diámetro de la rueda / holgura / distancia entre ejes	97mm / 30,6mm / 105mm
Material del chassis	Placa de aluminio con recubrimiento de polvo de 1,5 mm de espesor
Velocidad de traslación máxima	1 m/s
Velocidad angular máxima	420º/s (7,33 rad/s)
Máxima capacidad de carga	
Tiempo de autonomía	1,5h – 5h

Tabla 1-1: Dimensiones y características del Robot

En el diagrama de bloques de la figura 3-3 se puede encontrar una estructura general para entender el funcionamiento del robot móvil, dentro del cual, podemos ver que la parte de procesamiento y control se engloban en dos bloques principales: la placa CORE2 y el SBC.

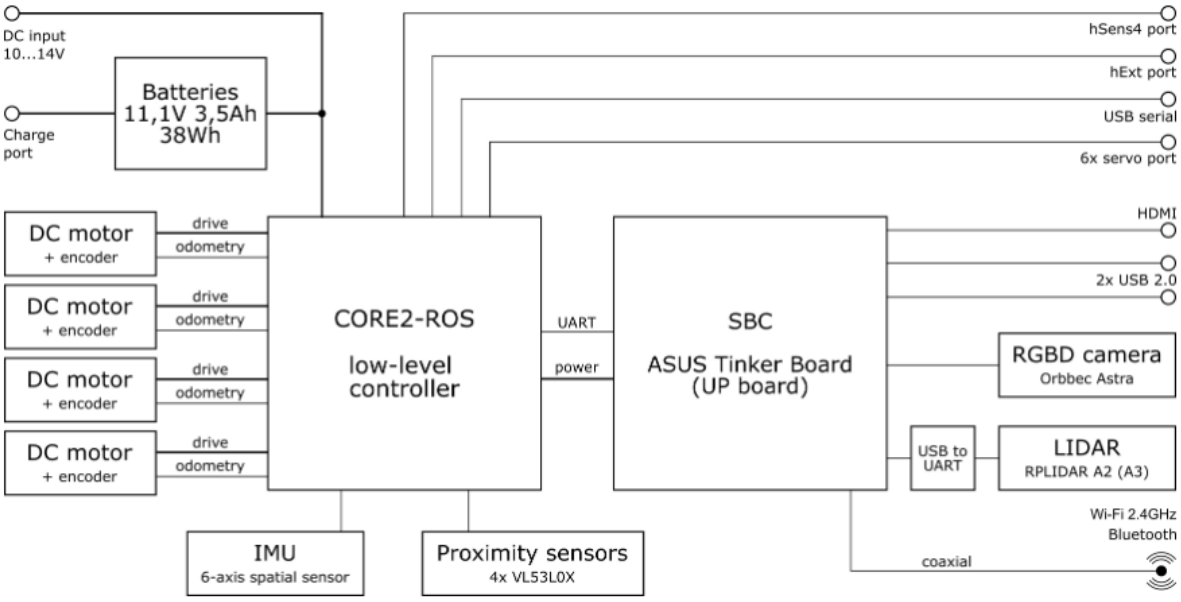

Figura 3-3: Diagrama del Robot

la parte más accesible por el usuario y será la encargada de recibir órdenes de control del procesador, con el que estará comunicado asíncronamente. Está directamente conectado a los actuadores del robot y se encarga de realizar las tareas de control de estos; de la misma manera, recibe información de otro tipo de periféricos. En la figura 3-4 podemos ver una ilustración de esta placa y en la tabla 3-2, se encuentran descritas sus partes.

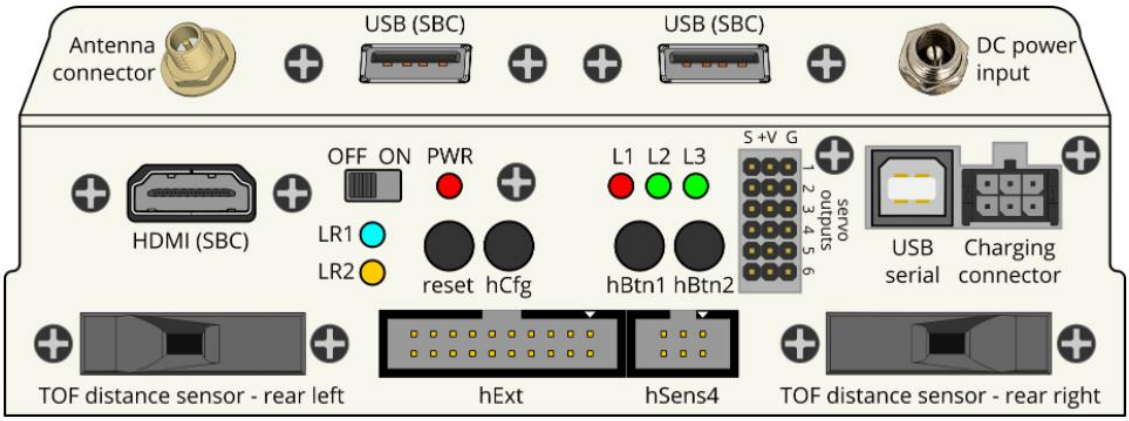

Figura 3-4: Placa CORE2.

Componente	Cantidad	Descripción
Conector de antena	1	Antena Wi-Fi de socket RP-SMA
USB	2	Puerto USB 2.0 desde SBC
HDMI	1	Salida HDMI desde SBC
Interruptor de potencia	1	Enciende y apaga Rosbot
LEDs	6	LR1 (R), LR2 (Y), L1 (R), L2 (G), L3 (G), PWR (R)
Botón de reset	1	Reinicia CORE2
hBtn	2	Pines de botones programables
Salidas para Servo	6	Salida PWM para Servos
USB puerto serie	1	Puerto serie simulado en el firmware en CORE2
Conector de carga	1	Conector de 6 pines para cargar las baterías
Entrada de potencia DC	1	Entrada para 12V y hasta 5 A
Sensor de distancia Time-of-Flight	2	Sensor de distancia de hasta más de 200 cm de rango
hExt	1	12xGPIO, 7x ADC, SPI, I2C, UART
hSens	1	4xGPIO, ADC, UART
hCfg	1	No tiene funcionaliad

Tabla 3-2: Componentes de la placa CORE2

nuestro sistema robótico. Como se acaba de comentar, este procesador va a desarrollar las instrucciones y/u órdenes que transmitirá a la placa CORE2. Recibe información directa de la cámara y el LIDAR y los procesa, ya que son los sensores más complicados. A parte de esto, será el elemento que tenga instalado el sistema operativo del robot que nos permita su programabilidad. Este sistema, basado en Linux es ROS.

Los componentes incorporados en el robot están indicados y descritos en la tabla 3-3.

Componente	Cantidad	Descripción
SBC	1	
LIDAR	1	RpLidar A3, 360º y rango mayor de 25 m
Sensor infrarrojo	4	VL53L0X Time-of-Flight. Sensor de distancia
CORE2	1	Controlador de tiempo real basado en el microcontrolador STM32F407
Motor DC	4	Xinhe Motor XH-25D, Motor: RF-370, 6VDC nominal, 5000rpm, velocidad sin carga a la salida: 165 rpm, torque: 2.9 kg*cm, current: 2.2A, ratio de marcha: ~34 (exactamente es 30613/900), encoder: magnetico, 48ppr, 12 poles
IMU	1	Reinicia CORE2
Cámara RGBD	1	Orbbec Astra de 640x480 de tamaño y profundidad
Baterías	3	Li-lon 18650, baterías recargables, capacidad 3500mAh, voltaje nominal de 3,7V
Antena	1	Conectada al módulo de Wi-Fi ASUS Tinker Board

Tabla 3-3: Componentes del *Rosbot 2.0 Pro*

3.2 Sistema de medición.

Si bien en la sección anterior se detalló el modelo robótico que conformará el Proyecto, en la presente vamos a ver de qué comoponentes consta nuestro sistema de medida.

Como ya se comentó, el sistema tratará de obtener medidas de la irradiación solar que proporciona un sensor. Los valores de las magnitudes medidas de dicho sensor serán lo más precisas posibles mientras más perpendicular se encuentre a la incidencia de los rayos de Sol. Para encontrar este ajuste de posición utilizamos un *pan&tilt*.

3.2.1 ISS-A60 (Sensor solar)

3.2.1.1 Origen del sensor solar

El elemento protagonista del sistema de medición que vamos a diseñar se trata del sensor *ISS-A60* (figura 3-5), el cual pertenece a la serie de sensores *ISS-AX* [22]. No obstante antes de pasar a explicar su funcionalidad, previamente vamos a proceder a contextualizarlo para entender mejor el funcionamiento que se esconde tras esta tecnología.

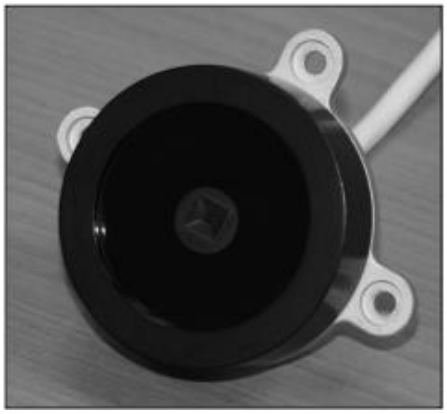

Figura 3-5: Sensor ISS-AX

El dispositivo fue desarrollado por *Solar Mems Technologies* (figura 3-6); esta empresa originariamente surgió como spin-off de la Universidad de Sevilla por parte del Dpto de Electrónica de la Escuela. Se dedica al desarrollo de sensores solares, siendo líderes mundiales en este tipo de dispositivos. La compañía tiene dos ramas principales: la rama espacial y la rama energética.

- La rama especial está especialmente dedicada al diseño de este tipo de dispositivos para ser implementados en satélites. En este campo, cuenta con más de 1500 unidades en órbita.
- El sector de la energía es en el que más se encuadra el sensor escogido. Este sector está dedicado a aplicaciones fotovoltaicas y relacionadas con la energía

Figura 3-6: Logotipo de *Solar Mems Technologies*

3.2.1.2 Características del ISS-A60

Los sensores de la serie *ISS-AX* son sistemas microelectrónicos que miden el ángulo de incidencia del Sol [22]. Está basado en fotodetectores repartidos en cuatro cuadrantes. La luz es guiada al detector a través de una ventana sobre el sensor. Dependiendo del ángulo de incidencia del haz de luz, se inducen corrientes en los fotodectores de los cuatro cuadrantes, lo cual, nos permite medir la señal que los fotodiodos generan.

En las figuras 3-7 y 3-8 se muestran bocetos que nos ayudan a visualizar el funcionamiento que acabamos de explicar.

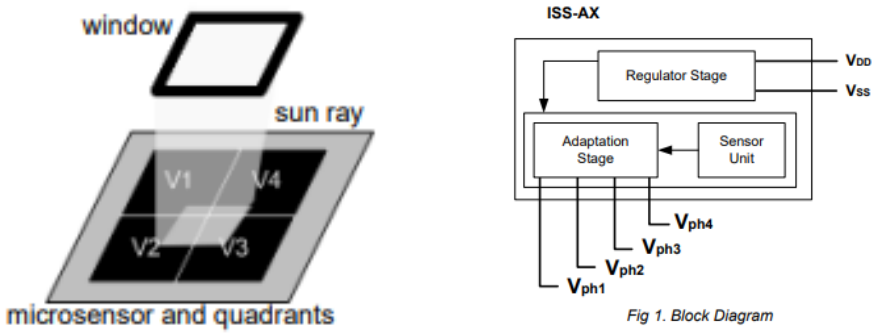

Figura 3-7: Microsensor basado en cuadrantes y Ventana

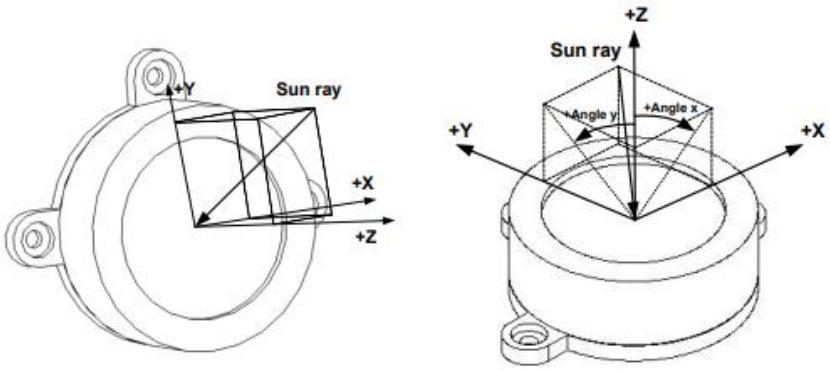

Figura 3-8: Ejemplo de incidencia de rayo solar

De las características que presenta el ISS-A60, vamos a destacar las siguientes que, vamos a tener en cuenta para el diseño [22]:

- Tensión de alimentación de 5 -12 V, siendo más recomendable la tensión de 5 V. Podemos conseguir estas tensiones con la propia alimentación del robot (Vease table 3-3).
- Las medidas se obtienen con un ángulo de incidencia de 0º. Por esta razón, la idea será conseguir este relación de perpendicularidad del sensor con el Sol.
- Campo de visión del sensor de 120º. Esta característica es importante ya que si nos encontramos fuera de este rango, el dispositivo no tomará medidas correctamente. Es por ello que, en este Proyecto, la idea principal es realizar un control que no se base en las medidas del ángulo de incidencia, sino en las *Ecuaciones de Spencer*. De esta manera conseguimos colocar el sensor en dirección al Sol con un error pequeño, asegurando que nos encontramos en el rango de visión. Posteriormente, en futuros Proyectos, se intentará afinar la precisión del control utilizando las medidas de la posición del Sol que nos da el dispositivo de medición para así poder conseguir una mayor precisión en la obtención de la magnitud a medir, la radiación del Sol.
- Temperatura de operación entre -40º y 85º. Nos permite operar con el dispositivo en un amplio intervalo de temperaturas haciendo que esta no resulte un problema a la hora de medir.

3.2.1.3 Ecuaciones de las medidas

Como ya se ha visto, las salidas que nos da el sensor hacen referencia a los fotodiodos. Pero esta medida no indica de ninguna manera la posición en la que se encuentra el Sol. Para poder conseguir estas medidas, necesitamos realizar una serie de cálculos [22].

Para calcular en ángulo X (que podemos ver en la figura 3-8):

$$X_1 = V_{PH3} + V_{PH4}$$
$$X_2 = V_{PH1} + V_{PH2}$$
$$F_x = \frac{X_2 - X_1}{X_2 + X_1}$$

$$ÁnguloX = arctg(C * F_x)$$

Para el ángulo en el eje Y (también en figura 3-8):

$$Y_1 = V_{PH1} + V_{PH4}$$
$$Y_2 = V_{PH2} + V_{PH3}$$
$$F_Y = \frac{Y_2 - Y_1}{Y_2 + Y_1}$$

$$ÁnguloY = arctg(C * F_y)$$

El parámetro *"C"* varia en función del modelo *ISS-AX*; para el caso que nos corresponde, el *ISS-A60*, nuestro valor sería siguiente:

$$C = 1.871$$

3.2.2 Pan&tilt

A continuación, vamos a pasar a comentar el elemento que va a trabajar como la parte móvil del sistema de medida: el *pan&tilt* (ejemplo en figura 3-9). Se trata de una estructura formada por dos servomotores que se encuentran dispuestos de manera que hacen que el efector de la misma se oriente hacia una cierta posición.

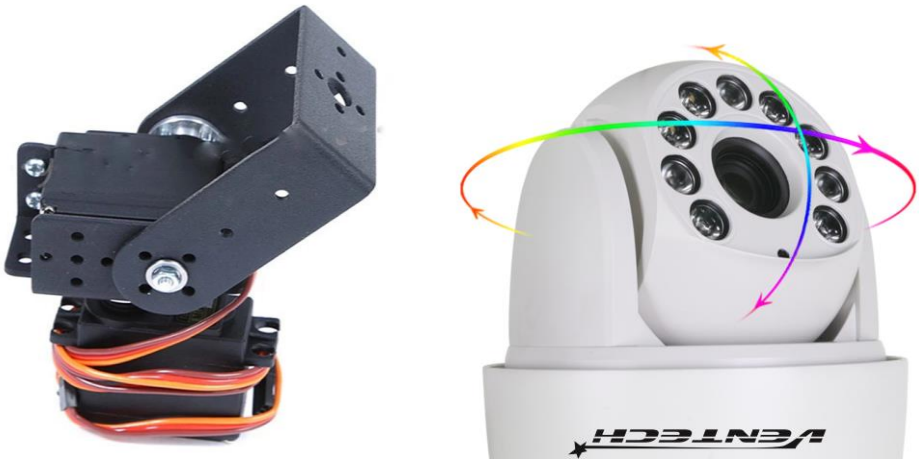

Figura 3-9: Ejemplos de *pan&tilt*

Para poder ajustar la posición de la estructura, los servomotores ofrecen dos movimientos:

- El movimiento *pan*, realizado por el servo que actúa como base de la estructura y ofrece un movimiento horizontal.
- El movimiento *tilt*, efectuado por el segundo servo, acoplado al anterior y que realiza un movimiento vertical.

Existen numerosos modelos de este tipo de componente, así como también hay varios tipos; es por esta razón vamos a necesitar una guía para escoger entre todas las alternativas. En nuestro caso, los elementos que nos ayudarán a descartar entre las posibles opciones serán los servomotores y el material de la estructura. Con respecto a los tipos, nos quedaremos con uno parecido al modelo de la izquierda de la figura 3-9 debido a que nos va a posibilitar incorporar cualquier otro dispositivo, siendo el sensor solar ISS-A60 el aplicable para el caso que nos concierne.

Por otro lado, como ya se ha mencionado, el modelo de servomotores y el material de la estructura van a ser el resto de los factores limitantes. Resumiendo, las características que necesitamos para el *pan&tilt* son:

- Un material rígido para la estructura, que pueda soportar el peso del sensor sin romperse.
- Servomotores que puedan suministrar el suficiente par como para que el peso del *ISS-A60* no suponga un entorpecimiento para el movimiento y la precisión.
- Por la misma razón que se acaba de detallar, es necesario que el engranaje de los servos sea metálico para dotar de rigidez al sistema.
- Es importante que las condiciones ambientales no deterioren estos componentes. Hay que tener en cuenta que el sistema está pensado para trabajar en exteriores.

Teniendo en cuenta todas estas condiciones, llegamos a la selección de un modelo que combina las características mencionadas. Lo vemos en la figura 3-10.

Figura 3-10: *Pan&tilt* escogido para el Proyecto

El *pan&tilt* utiliza una estructura metálica, de manera que conseguimos la rigidez que buscábamos. Por otra parte, el sevo utilizado se corresponde con el modelo de referencia *DS-3218mg* (figura 3-11). Las características principales de este servomotor son las siguientes:

- Voltaje de funcionamiento con un rango entre 4.8-6V, siendo 5 V el nivel de tensión preferido.
- A 5 V, obtenemos un par de 19 Kg/cm, pudiendo llegar a soportar hasta un peso de 20 kg. Con estos valores tendremos una fuerza más que suficiente como para no tener problemas con los 100g del sensor solar, consiguiendo que no entorpezca al funcionamiento de los servos.
- Es impermeable, lo que facilitará que el dispositivo no se rompa o deteriore si por algún casual se encuentra en un ambiente húmedo o día lluvioso. Debido al tipo de aplicaciones hacia dónde va

dedicado, no es muy probable que se den estas condiciones ambientales; a pesar de eso, había que tener este factor en cuenta.

Figura 3-11: Servomotor *DS-3218 mg*

3.2.3 Arduino Nano 33 IoT (ABX00027)

El último componente que falta por definir para nuestro sistema de medida va a ser el cerebro encargado de dirigir movimientos y pasarle información al robot. Este elemento no es otro que el microcontrolador.

A la hora de escoger este componente se consideraron varias opciones. Una de ellas era utilizar la propia placa del micro que ya contiene el propio *Rosbot 2.0 Pro* y programarlo todo desde dentro del robot. El problema de esta selección radica en los escasos pines GPIO que presente teniendo en cuenta que, sin contar con los referentes a comunicación, que tienen función alternativa de GPIO, tan solo nos quedarían 5 pines que puedan configurarse como puertos de entrada y salida. De estos pines restantes, ya de por sí, 4 de ellos son utilizados para la lectura de los fotodectores del *ISS-A60*, quedando así tan solo un pin de GPIO (más los de UART, SPI e I2C, los cuales no podríamos utilizar para comunicación si se utilizan como GPIO).

La segunda opción que se barajó fue diseñar el sistema de medida como un complemento por separado del robot, de tal manera que se pueda conectar y desconectar cada vez que se necesite. De esta forma, no estaríamos utilizando el micro del sistema robótico, sino que necesitaríamos otro.

Con esto conseguiríamos preservar los pines de comunicación del robot por si acaso en algún futuro fuera necesario utilizarlos para posibles ampliaciones de periféricos. Lo mismo ocurre con el pin GPIO restante. Por otra parte, en el micro del sistema de medición solar tendremos más pines que pueden configurarse como entradas o salidas, por si fuera necesario incorporar nuevos sensores o actuadores. En caso de que el robot lo precise, puede tener control de dichos pines o simplemente leer su valor analógico gracias a la comunicación UART con el micro que los posee.

El microcontrolador que se va a implementar se trata del *Arduino 33 Nano IoT* (con nº de referencia *ABX00027*), el cual podemos observar en la figura 3-12. Las principales características o criterios en los que se ha basado la elección de este micro son:

- No ocupa mucho espacio (45x18 mm) y puede incorporarse en una PCB ya que existen librerías para su diseño en silicio.
- Contiene una IMU (LSM6DSL) que incluye un giroscopio y un acelerómetro; este componente es fundamental para la aplicación nos concierne debido a que, como se verá más adelante, para calcular la posición del Sol, necesitamos una brújula para saber la inclinación en la que nos encontramos con respecto al Norte y así posicionar bien el sistema de medida.
- Opera 3.3 V. A pesar de que los servomotores escogidos no operen a este voltaje, al ser un valor común, nos serviría para leer varios tipos de sensores, así como controlar otro tipo de actuadores. Para nuestro caso, al usar servos que operan a 5 V, tan solo tendremos que amplificar algunos pines. Así podremos tener una mezcla de pines que operen a los dos niveles de tensión en función de si necesitamos unos u otros.
- También podremos leer entradas analógicas ya que el dispositivo contiene un CAD de 10 bits, aunque las entradas no pueden tener un valor superior a 3,3 V.

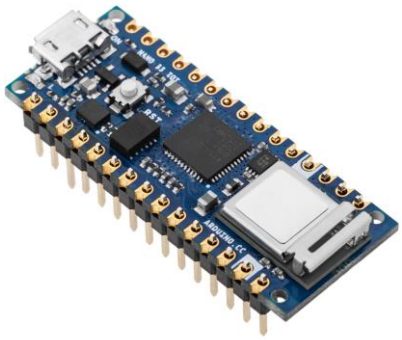

Figura 3-12: Arduino Nano 33 IoT.

3.2.4 Brújula HMC5883L

Se trata de un sensor que nos permite estimar el valor del campo magnético todos los ejes de referencia (X, Y, Z). Con esto podemos obtener una estimación de la orientación de la brújula digital con respecto al campo magnético de la Tierra. Nosotros utilizaremos este sensor en el proyecto para poder conocer la orientación del UGV con respecto al Norte. Esta información es necesaria ya que, en función de este dato tendremos que modificar el ángulo de giro de la azimuth calculada.

La comunicación entre la brújula y el *Arduino* se realiza mediante comunicación I2C y la tensión de voltaje que recibe nuestro microcontrolador irá desde los 1,8 V a los 3,3 V.

Figura 3-13: Brújula digitañ

4 SOFTWARE

Análogamente a como se hizo en la anterior división explicando la parte física (hardware) con la que se va a trabajar en el proyecto, en este capítulo veremos cual el software con el que se va a desarrollar el trabajo. Para nuestro caso en particular, estos son tres: *ROS* (Robot Operating System), como sistema operativo que nos permite trabajar con el robot; *Arduino* como software de programación de nuestro microcontrolador y; por último, *Altium* como programa de diseño de PCBs

4.1 Robot Operating System (ROS)

En esta primera sección, vamos a exponer y explicar que es ROS [23], para qué sirve, como funciona y por qué vamos a utilizar nosotros.

ROS es un framework para el desarrollo de software dedicado a robots, este como su propio nombre indica ofrece la función de sistema operativo para los dispositivos robóticos. Fue creado originalmente en el 2007 como un soporte para un proyecto basado en Inteligencia Artificial en el *Laboratorio de Inteligencia Artificial de Stanford*. Desde entonces ha seguido en continua evolución hasta llegar a la actualidad, donde realiza el papel de sistema operativo provee servicios como el control de dispositivos de bajo nivel, la implementación de funcionalidades comunes el paso de mensajes entre procesos y el mantenimiento de paquetes.

Su composición se basa en una arquitectura de grafos, donde el procesamiento se realiza en los nodos que pueden recibir y mandar mensajes de sensores, control o estados. Actualmente funciona en sistemas UNIX (Ubuntu) y también puede instalarse en sistemas Windows, aunque se trabaja para adaptarlo al resto de sistemas operativos. No obstante, normalmente lo más común es trabajar en sistemas operativos basados en UNIX y que respeten las normas POSIX, en nuestro caso utilizaremos Ubuntu, la versión 18.04.

Figura 4-1: ROS

Los nodos que estábamos comentando en la introducción se tratan de archivos ejecutables (que suelen programarse en C++ y en Python). Estos nodos pueden ejecutarse por separado o al mismo tiempo creando otro tipo de archivos programados en XML: los *launch*. Los nodos pueden comunicarse entre sí mediante a unos canales denominados *topics*, a través de los cuales son capaces de publicar información en ellos para que esta sea enviada y distribuida o bien, suscribirse a un tópico para captar la información que emite. En el primero de los casos, estaríamos hablando de un nodo *Publisher* y en el segundo de un nodo *Suscriber*. No obstante, un solo nodo puede desempeñar ambos papeles.

Estos nodos representan todos los procesos que tienen lugar en ROS y estos son capaces de comunicarse entre sí. El proceso principal se denomina ROS Master y es, como su propio nombre indica, el maestro, ya que se encarga de la creación e inicialización de nodos además de la conexión entre los mismos.

La arquitectura básica se considera descentralizada ya que los mensajes o llamadas no pasan necesariamente

por el ROS Master, sino que este solo se encarga de crear la conexión para que puedan funcionar de manera independiente entre si. Esto permite la posterior conexión con PCs en el exterior. En la figura 4.2 vemos un esquema del función

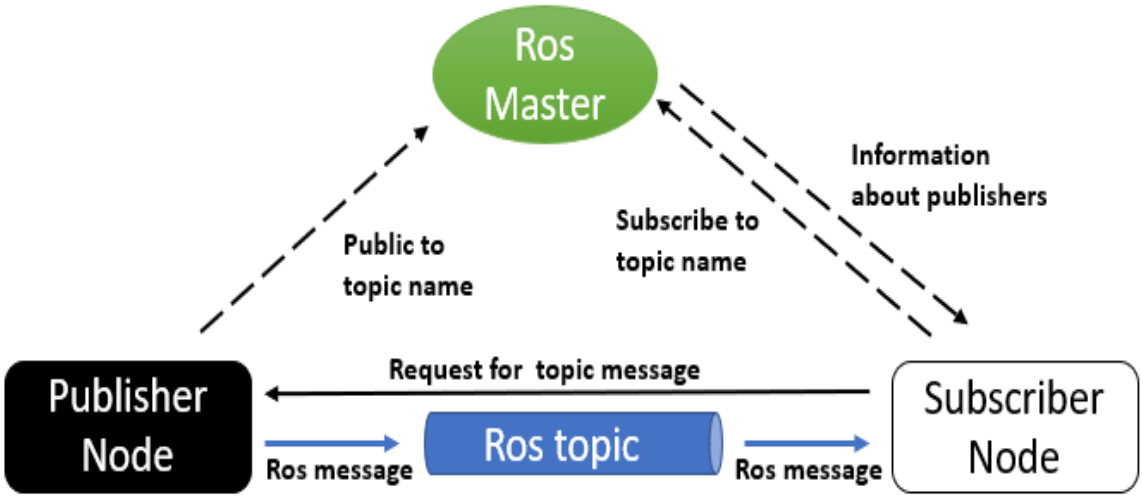

Figura 4-2: Estructura de funcionamiento de ROS.

Se cuenta también con un parameter server encargado de la gestión de la memoria y los archivos de manera ordenada. Es tan necesario para trabajar como el ROS Master y por tanto ambos se suelen iniciar de manera conjunta mediante la orden "ROSCORE", esta debe ejecutarse siempre antes de empezar a trabajar con el programa para que todo sea inicializado.

Nuestro sistema tratará de un sensor móvil conformado por un sistema de medida acoplado al *Rosbot 2.0 Pro*, siendo este ultimo el "cerebro" principal del todo el equipo. Recordamos que la idea general para el futuro sería que este robot pueda comunicarse y compartir información con otros vehículos no tripulados que se encuentren realizando tareas de medición, creando así una red sensores móviles.

También acabamos de ver que, actualmente, la programación de robots móviles se lleva a cabo mediante ROS y, es por ello que todas las tareas importantes (comunicación, información, etc) que desempeña el robot van a ser tratadas por medio de este sistema operativo. Es por ello que sería lógico pensar que será el propio *Rosbot 2.0 Pro* el encargado de gestionar el intercambio de información con el resto de robots y realizar los cálculos necesarios para poder realizar el control de todos sus periféricos, de forma que, el sistema de medición, al ser un sistema a parte del robot realizaremos la siguiente rutina: el robot, desde ROS, realizará el cálculo de las ecuaciones solares y se lo pasará al *Arduino* como referencias y ya será este último el encargado de implementar esas referencias en el control de los servos.

Por otro lado, las medidas del sensor solar, tras ser captadas pasan directamente al robot por medio de unos paquetes de ROS ya programados previamente en otro TFG.

4.2 Arduino

Arduino (logo en figura 4-3) es una compañía de desarrollo de software y hardware libres, así como una comunidad internacional que diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos que puedan detectar y controlar objetos del mundo real. Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios.

Figura 4-3: Logo de *Arduino*.

Los productos que vende la compañía son distribuidos como Hardware y Software Libre, bajo la Licencia Pública General de GNU (GPL) y la Licencia Pública General Reducida de GNU (LGPL), permitiendo la manufactura de las placas Arduino y distribución del software por cualquier individuo. Las placas Arduino están disponibles comercialmente en forma de placas ensambladas o también en forma de kits.

Los diseños de las placas Arduino usan diversos microcontroladores y microprocesadores. Generalmente el hardware consiste de un microcontrolador Atmel AVR, conectado bajo la configuración de "sistema mínimo" sobre una placa de circuito impreso a la que se le pueden conectar placas de expansión a través de la disposición de los puertos de entrada y salida presentes en la placa seleccionada. Las shields complementan la funcionalidad del modelo de placa empleada, agregando circuitería, sensores y módulos de comunicación externos a la placa original. La mayoría de las placas Arduino pueden ser alimentadas por un puerto USB y las placas Arduino pueden ser programadas a través del puerto serie que incorporan haciendo uso del Bootloader que traen programado por defecto. El *software* de Arduino consiste de dos elementos: un entorno de desarrollo (IDE) (basado en el entorno de *processing* y en la estructura del lenguaje de programación Wiring), y en el cargador de arranque (*bootloader*, por su traducción al inglés) que es ejecutado de forma automática dentro del microcontrolador en cuanto este se enciende. Las placas Arduino se programan mediante un computador, usando comunicación serie.

En este TFG vamos a hacer uso de esta herramienta de programación para programar un modelo de estos microcontroladore con el fin de encargarnos del control de los servomotores del sistema de medida.

Antes hemos explicado que será el robot el encargado de realizar los cálculos necesarios para el control del sistema de medición; posteriormente, este estará conectado con el *Arduino* mediante comunicación por puerto serie y le enviará el resultado de dichos cálculos. Cuando nuestro dispositivo hardware reciba estos cálculos, se va a encargar de mover los servos del *Pan&Tilt* en la dirección que sea precisa para que el sensor pueda posicionarse lo más perpendicular posible a los rayos solares.

4.3 Altium Designer

El ultimo software que vamos a ver se trata de *Altium Designer*, es un programa utilizado para diseño de PCBs al igual que *Eagle*, ambos son muy empleados en la industria. *Altium* le permite al diseñador poder tener una visión de todos los aspectos del proceso de diseño gracias a que ofrece un entorno de diseño muy unificado.

Aunque antes de continuar, debemos saber que es una PCB [9]. Esta se trata de una tarjeta de circuito impreso en la que los componentes y conductores se encuentran contenidos dentro de una estructura mecánica. Se trata, por tanto, de una forma más compacta y elegante de implementar un circuito.

Figura 4-4: Logo *Altium Designer*.

Lo primero que se va a comentar para entender el funcionamiento de este software es el concepto de librería. Para entenderlo bien, vamos a relacionarlo con lo que ocurre con cualquier lenguaje de programación, cuando queremos recurrir a ciertas funciones o valores de parámetros, tenemos que incluir en nuestro código librerías en las cuales dichas funciones se encuentren definidas y programadas. De esta manera podremos utilizarlas en nuestro programa siempre que queremos.

Análogamente, en *Altium* también tendremos librerías, solo que es en esta ocasión, estas librerías harán referencia a componentes electrónicos; de manera que, una vez que las instalemos, podremos utilizar dichos componentes en nuestros diseños cuando los necesitemos.

Para entenderlo mejor, vamos a poner un ejemplo: si nosotros queremos utilizar un modelo de condensador en específico para el incluirlo en la placa que no se encuentre entre las librerías por defecto del programa. Lo primero que debemos hacer es buscar en internet un archivo que contenga sus librerías. Una vez que las encontremos, las debemos instalar en nuestro espacio de trabajo; tras hacerlo ya podremos incluir el ese modelo de condensador en nuestra PCB.

Como se puede ver, este factor siempre nos limita a la hora de realizar muchos tipos de diseño ya que la mayoría de los componentes que normalmente queremos utilizar no se encuentran entre las librerías que vienen ya instaladas en *Altium*, por lo que tendremos que recurrir a internet a páginas de pago para poder descargarlas. No obstante, al tratarse de un software tan utilizado, al igual que ocurre con *Eagle* (otro software de diseño conocido), muchos fabricantes ya ofrecen un modelo de librerías para cada uno de los dispositivos y existen páginas donde se pueden descargar gratuitamente. Aunque de la misma manera hay muchos dispositivos que no vienen con una librería asociada.

Cada vez que queremos realizar el diseño de una placa o PCB en *Altium Designer*, hay que seguir dos pasos fundamentales:

- Diseño del esquemático.
- Diseño físico y rutado de la PCB.

4.3.1 Diseño esquemático

La idea de esta etapa será crear el circuito que queremos diseñar, es decir, obtener todos los bloques de las librerías e interconectarlos entre sí. El objetivo es diseñar los planos del circuito de la PCB, donde se pueden ver cuales son las señales de entrada a la PCB de

Esta fase es la más importante y crítica debido a que es la que va a determinar el funcionamiento de la PCB, es por ello que debemos de leer bien los documentos *Datasheet* de los chips que vamos a utilizar en la placa para saber que es lo que estamos conectando a sus distintos pines. De la misma manera hay que tener clara la funcionalidad de nuestro circuito.

En la figura 4-5 se muestra un ejemplo de un modelo de diseño esquemático.

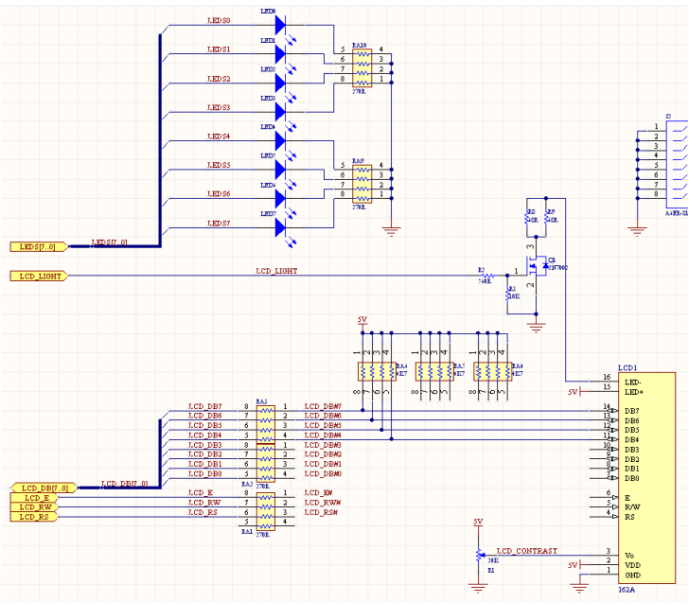

Figura 4-5: Ejemplo de diseño esquemático de un circuito.

4.3.2 Rutado y diseño físico de la placa

Una vez que ya hemos desarrollado nuestro circuito y hemos creado el modelo esquemático del mismo, vamos a pasar al siguiente paso. Hasta ahora lo que hemos hecho ha sido de manera simbólica (simplemente hemos creado bloques y los hemos interconectados entre sí). Sin embargo, todo ha sido sobre el papel y los planos. En esta fase pasaremos de un diseño esquemático a uno más físico que se acerque de forma muy precisa a la PCB de la realidad.

En esta parte, inicialmente nos vamos a encontrar las huellas de todos los componentes desordenadas en el espacio. La huella de un componente es la forma física que presenta en la placa y que nos muestra donde debemos soldarlo. Nuestra tarea será colocar las huellas de la forma más "ordenada" posible, intentando que las conexiones entre todos los dispositivos se puedan realizar de forma sencilla tratando de no interferir en las conexiones de otros componentes.

Posteriormente, vamos a rutar estas conexiones. Es decir, vamos a transformar esas conexiones ficticias en caminos que entre distintos pines o nodos de voltaje. Estos caminos lo que harán será aislar el silicio que se encuentra dentro del mismo con el del resto de la placa.

En las figuras 4-6 y 4-7 vemos como es el proceso de este rutado.

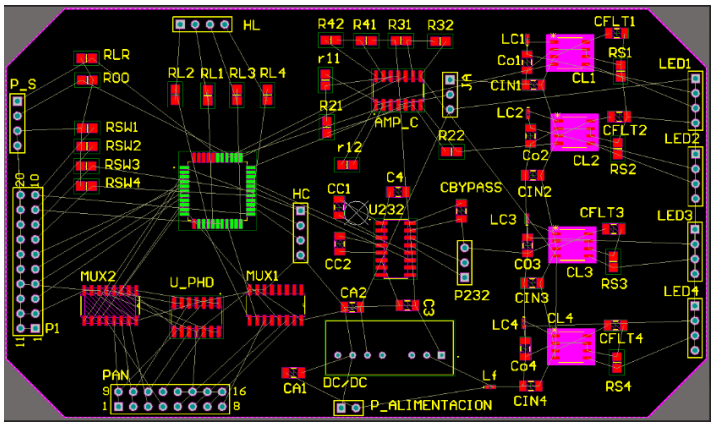

Figura 4-6: PCB con elementos aun sin rutar.

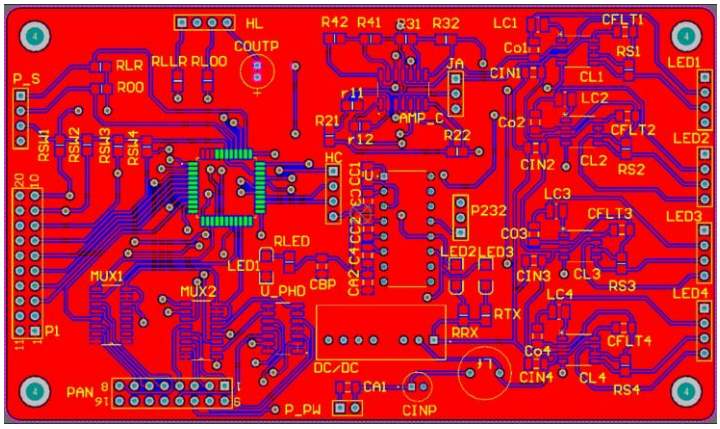

Figura 4-7: PCB con rutado acabado.

Tras acabar con esta etapa, estaremos listo para mandar nuestro diseño de la PCB a alguna empresa para que se encargue de su fabricación.

5 PROGRAMACIÓN EN ROS

5.1 Introducción

En este capítulo vamos a tratar de explicar todo lo referido a la parte de programación en el entorno ROS para la realización del proyecto. Esta parte se va a subdividir en dos secciones principales: el desarrollo de una comunicación con *Arduino* y la creación de un nodo de cálculo de las ecuaciones de solares ya vistas.

Cabe mencionar que esta parte va a ser desarrollada en el sistema operativo *Ubuntu 18.4* y que la versión de ROS empleada para la codificación es *ROS Melodic*. Este último dato debe tenerse en cuenta ya que, en función de la versión utilizada, los comandos utilizados pueden verse ligeramente modificados.

5.2 Comunicación con Arduino

El primer problema al que nos enfrentamos en el desarrollo del trabajo es mandar información desde nuestro robot hasta *Arduino* y viceversa. Nuestro objetivo principal es poder realizar, desde el propio robot, una serie de cálculos referidos a las ecuaciones solares que posteriormente vamos a implementar utilizando el *Arduino*. Para esto, necesitamos poder crear un tipo de conexión con el fin de poder realizar el intercambio de datos.

Arduino IDE dispone de numerosas librerías que nos simplifican mucho el trabajo, desde librerías para controlar el ángulo al que gira un actuador hasta algunas que nos permiten comunicarnos con varios dispositivos y poder enviarles información. También existe una librería que crea una interfaz entre ROS y el microcontrolador. Cuando usamos esta librería, el *Arduino* puede enviar y recibir mensajes al PC que contenga el framework de manera que, vistos desde este lado, estos mensajes son convertidos en *topics*. De esta forma, el microcontrolador podrá publicar datos en topic, así como suscribirse a otros para recibir información del mismo modo que lo haría un nodo en ROS. De hecho, se podría decir que estaría actuando como una especie de nodo hardware de ROS. Es por ello por lo que vamos a ver como instalar esta librería en nuestro sistema operativo [23].

El primer paso que debemos seguir es instalar un paquete de ROS que se encargue de instalar estas librerías. Para ello vamos a abrir un terminal y escribir el siguiente comando:

sudo apt-get install ros-melodic-rosserial-arduino

Con esto instalamos los paquetes necesarios para poder crear la interfaz que buscamos. Posteriormente, en un nuevo terminal, nos vamos a dirigimos al directorio donde se encuentren los archivos relacionados con *Arduino*. Una vez en él, nos dirigimos a una carpeta denominada "*libraries*" (como se muestra en la figura 5-1) y ejecutamos el siguiente comando:

rosrun rosserial_arduino make_libraries.py

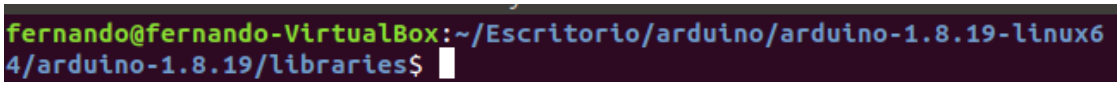

Figura 5-1: Directorio donde debemos ejecutar el comando

Tras ejecutar el comando, vamos a cerciorarnos de que en el directorio de la figura 5-1 (al menos en mi caso en

particular) se ha creado una carpeta denominada *ros_lib*, como se ilustra en la figura 5-2.

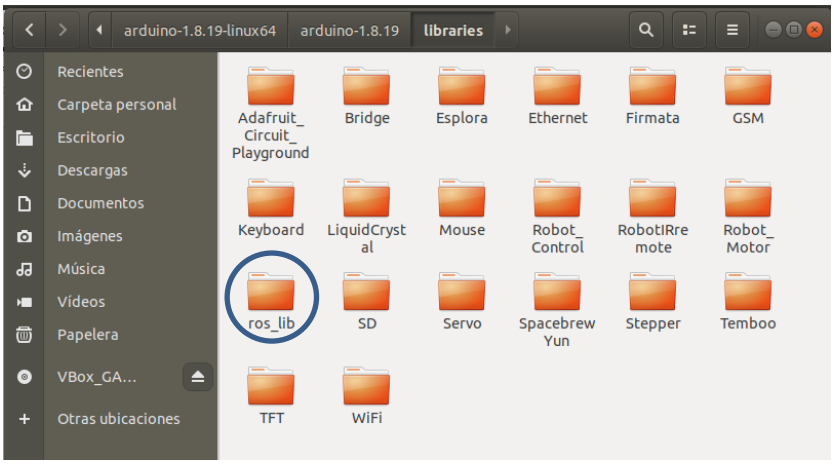

Figura 5-2: Instalación *ros_lib*.

Después de haber instalado el paquete, vamos a corroborar que funciona. Con el fin de realizar esta comprobación, el paquete instalado dispone de un código de ejemplo denominado *Blink*. Este código es un tipo de "*Hola mundo*" para la interfaz entre *Arduino* y ROS ya que, simplemente trata de encender y apagar un LED. Cuando publicamos un *topic*, el LED se enciende y se mantiene en este estado hasta el momento en el que volvamos a publicar otro *topic*, que en dicho caso se apagará. El código de este programa se muestra en la figura 5-3.

```
#include <ros.h>
#include <std_msgs/Empty.h>

ros::NodeHandle  nh;

void messageCb( const std_msgs::Empty& toggle_msg){
  digitalWrite(LED_BUILTIN, HIGH-digitalRead(LED_BUILTIN));   // blink the led
}

ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb );

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  nh.initNode();
  nh.subscribe(sub);
}

void loop()
{
  nh.spinOnce();
  delay(1);
}
```

Figura 5-3: Código de ejemplo.

El funcionamiento del código es sencillo; al inicio se incluyen las librerías de ROS y la del tipo de mensaje que vamos a utilizar. Posteriormente se crea un nodo del tipo *suscriber* que se encuentre suscrito al *topic "toggle_led"*. Cada vez que le llegue un mensaje a través de ese *topic*, se ejecutará la función de retorno *messageCb()*. Dicha función cambiará el estado del LED, de encendido a apagado o de apagado a encendido en función de como se encuentre.

El próximo paso será subir el código a la placa. No obstante, al encontrarnos en una máquina virtual, este paso no será directo ya que habrá que buscar el puerto serie por el que pasa el *Arduino*. Si queremos averiguarlo, tan solo tendremos que ejecutar el comando *dmesg* y buscamos el puerto que nos importa. En la figura 5-4 nos damos cuenta de que, para nuestro caso, el puerto serie deseado es "/dev/ttyACM0" [23].

```
[  190.946799] usb 2-2: Manufacturer: Arduino (www.arduino.cc)
[  190.946800] usb 2-2: SerialNumber: 85734323331351714181
[  190.958979] cdc_acm 2-2:1.0: ttyACM0: USB ACM device
[  190.963700] usbcore: registered new interface driver cdc_acm
[  190.963700] cdc_acm: USB Abstract Control Model driver for USB modems and ISD
N adapters
```

Figura 5-4: Ejecución comando dmesg

Por último, antes de poder probar el código de ejemplo, debemos de modificar los permisos de nuestro dispositivo de puerto serie. Para ello hacemos uso del siguiente comando:

sudo chmod 777 /dev/ttyACM0

Con esta línea de comando otorgamos permisos de lectura, escritura y ejecución de archivos a la placa por parte del usuario y del superusuario.

Tras concluir este paso, podremos compilar el código de ejemplo en nuestra placa. Cabe mencionar que, al estar creando una conexión serie del *Arduino* con ROS, no podremos utilizar la herramienta *Serial Port* de nuestro micro; si en algún momento la necesitásemos, tendríamos que crear una conexión con otros pines de la placa, en caso de que disponga de ellos.

Abrimos un nuevo terminal y ejecutamos la orden *roscore* para que llamar al "nodo maestro" de ROS y, por ende, encender nuestro servidor. Después, desde otro terminal, vamos a llamar a la siguiente orden:

rosrun rosserial_python serial_node.py port:=/dev/ttyACM0 baud:=57600

El comando *rosrun* nos permite ejecutar un nodo de nuestro espacio de trabajo (*workspace*). En este caso, el nodo que vamos a llamar, como bien se indica, es "*serial_node*" perteneciente al paquete "*rosserial_python*". Este nodo es el encargado de realizar la conversión de los *topics* que vayan dirigidos hacia *Arduino*.

Al nodo se le pasan como referencia dos parámetros:

- Port: puerto serie utilizado.
- Baud: Velocidad de transmisión de información en baudios.

Desde *Arduino IDE*, compilamos nuestro programa y lo subimos a la placa. En el momento en el que se encuentre subido, si previamente hemos ejecutado los comandos ya vistos, la comunicación entre ROS y la placa es creada.

Ahora, para poder probar el código tan solo tendremos que publicar *topics* y observar si el LED correspondiente al pin 13 (pin) cambia su valor cuando encendemos o apagamos. Abrimos un terminal nuevo y usamos la orden de ROS *"rostopic pub"*. Esta publicará mensajes sobre el *topic* que inidiquemos. El comando que debemos ejecutar es:

rostopic pub toggle_led std_msgs/Empty –once

En la figura 5-5 podemos observar que el código funciona perfectamente y, por tanto, la comunicación se ha realizado con éxito. A partir de ahora podremos dedicarnos al desarrollo del código.

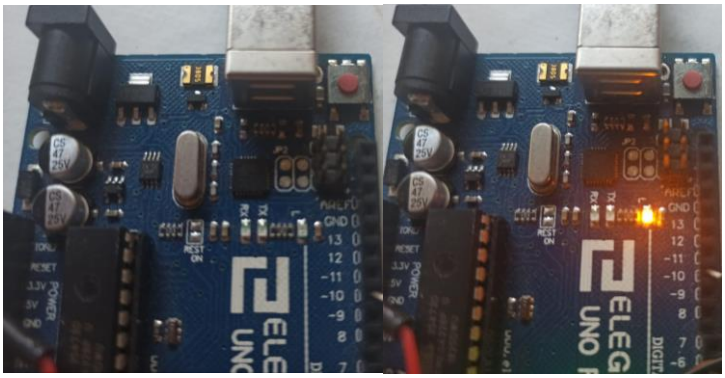

Figura 5-5: Comprobación de la comunicación

5.3 Programación en C++

Tras haber solucionado el problema de la interfaz de comunicación entre *Arduino* y ROS, podemos proceder a la codificación de las ecuaciones solares que ya se vieron previamente en este documento. La idea general será hacer uso de ROS para calcular la azimut y altitud del Sol desde el UGV con el objetivo de enviarle dichos ángulos a la placa de control. La idea será enviar estos datos a la placa por medio de topics. Por otro lado, necesitaremos hacer uso del paquete desarrollado en un TFG [19] a este que contiene un driver para que el robot pueda recibir las medidas tomadas por el sensor.

Se va a utilizar C++ como lenguaje de programación ya que, al ser un lenguaje es más eficiente para el procesador que tener que recurrir a un lenguaje interpretado como por ejemplo sería el caso de *Python*.

5.3.1 Programación de las ecuaciones solares

En esta subsección explicaremos las nociones más importantes para comprender el funcionamiento del nodo que ejecuta los cálculos necesarios para obtener la altitud y azimut solares. El primer paso que vamos a seguir será crear un paquete de ROS que contenga todos los archivos necesarios para el desarrollo y ejecución del código. Esto lo hacemos abriendo un terminal en la carpeta */src* de nuestro directorio de ROS (*/ros_workspace* en mi caso particular) y escribiendo el comando:

catkin_create_pkg tfg_pkg roscpp

Con este comando, veremos que hemos conseguido crear un paquete de ROS que dependerá del paquete *roscpp*, el cual, a su vez es un paquete que contiene una librería de ROS básica que nos permitirá realizar la programación del paquete en un C++.

Una vez creado, el siguiente paso será crear el nodo que contenga la programación necesaria para poder ejecutar el código.

Nos dirigiremos, dentro de nuestro paquete, al directorio */src* y creamos un archivo de texto con extensión *".cpp"* (importante ya que esto indica que nuestro programa está codificado en C++ y debe compilarse); en mi caso he denominado a este paquete como *"tfg_pkg_node.cpp"*. Hecho esto último ya podremos acceder a este archivo y modificarlo con el fin de escribir un código sobre él.

Con respecto al código, a pesar de que se encuentra adjunto y perfectamente comentado al final de esta memoria, vamos a explicar los conceptos básicos que lo rigen para hacer más fácil su comprensión.

Recordando el objetivo principal del nodo, que consistía en poder enviar los ángulos de altitud y azimut solares mediante mensajes; la primera tarea consistió en realizar un estudio de que tipo de mensajes utilizar para poder

enviar estos datos. Finalmente, se llegó a la conclusión de que una buena opción sería utilizar *geometry_msgs::Pose2D*. Esta clase está definida para definir una posición geométrica y presenta los siguientes campos:

float64 x

float64 y

float64 theta

A pesar de que sirvan para definir una posición en el planos *x* e *y* para la azimut y la altitud respectivamente puesto que los ángulos obtenidos serán números decimales. De este modo, vamos a guardar nuestros ángulos en la variable *angs* de tipo *geometry_msgs::Pose2D*. El último elemento de la clase no va a ser utilizado.

Por otro lado, vamos a necesitar publicar estos mensajes a través de un *topic*. Este elemento de ROS lo crearemos nosotros y lo denominaremos *"/angulos"*.

En resumen, va a existir una variable *"angulos"* que va a publicar en el *topic "/angulos"* un mensaje de tipo *geometry_msgs:Pose2D* que contenga la información sobre los ángulos calculados con las ecuaciones solares.

Por otra parte, se va a crear un estructura de tiempo de tipo *tm* que contenga la fecha actual, la cual, inicialmente debemos introducir nosotros a mano. Esta estructura se va a seguir actualizando en tiempo real para modificar los parámetros calculados mediante un hilo que se encargue de contar el tiempo. El hilo ejecuta una función de arranque denominada *time_routine*.

Finalmente, en el un bucle infinito vamos a calcular las ecuaciones solares explicadas en el estado del arte por cada iteración y, seguidamente publicar los resultados en *"/angulos"*.

Por último, para poder ejecutar este nodo junto al resto, vamos a crear un archivo *.launch* para facilitar la ejecución concurrente del sistema mediante un solo comando.

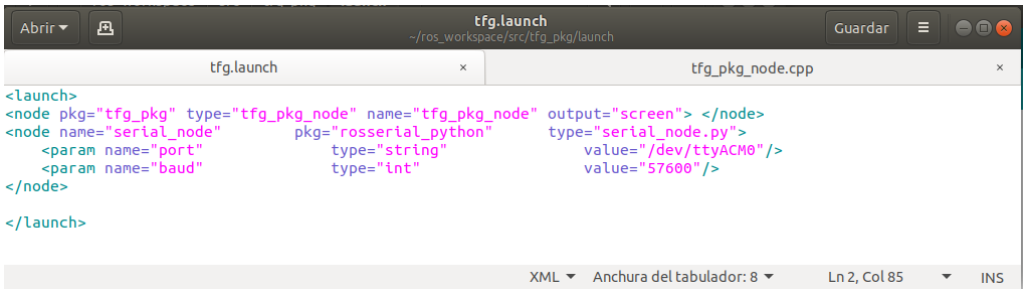

Figura 5-6: Archivo *.launch*.

6 PROGRAMACIÓN EN ARDUINO

6.1 Introducción

De la misma manera que en el capítulo anterior vimos lo relativo al desarrollo del proyecto en el entorno ROS, en esta sección nos dedicaremos a detallar los puntos más importantes de la programación del sistema de medición a través de *Arduino ISE*.

La programación de nuestro microcontrolador está enfocada en realizar el movimiento de los servomotores en función de las medidas tomadas. No obstante, a la hora de realizar los experimentos no se contaba con el *Rosbot* que contiene el nodo desarrollado encargado de tomar las medidas. Por este motivo, a pesar de que la placa está diseñada para que sea el propio robot quien reciba directamente las medidas del sensor, ha sido codificado también un programa alternativo donde el microcontrolador también gestione la entrada de datos de la radiación y calcule las ecuaciones de Spencer. De esta manera el funcionamiento del sistema puede ser probado independientemente de que contemos con el robot o no.

6.2 Librerías empleadas

Antes de explicar el funcionamiento del código, cabe mencionar previamente las librerías que han sido utilizadas para su realización. En el capítulo anterior se comentó el conjunto de librerías incluidas en *ros_lib*. Vimos que estas le permiten a nuestro microcontrolador comunicarse con ROS y poder programar la placa como si se tratase de un propio nodo de este sistema operativo pudiendo así suscribirse a *topics* y publicar sobre los mismos.

Sin embargo, estas no son las únicas bibliotecas que vamos a utilizar, pues debido a los actuadores y sensores que van a componer el sistema, vamos a recurrir a otra serie de librerías que nos van a facilitar el trabajo mediante funciones.

La primera de ellas es *Servo.h*. Esta incluye todas las funciones necesarias para poder trabajar con los servos de forma sencilla como si fueran objetos (de programación). Esta librería no necesitaremos instalarla porque es tan utilizada que ya suele estar incluida en el software. Las funciones que más nos van a interesar para controlar los servomotores son:

- ***Servo "nombre del objeto"***: Nos sirve para declarar nuestro periférico como una estructura de tipo servomotor. Para explicar el resto de funciones vamos a suponer que el nombre de nuestro objeto será "*s_azimut*".
- ***s_azimut.attach(int pin)***: Esta función, que debe incluirse en el *void_setup()* sirve para indicar que pin (con PWM) va a controlar el movimiento del servo.
- ***s_azimut.write(int ang)***: Con esta llamada podremos realizar un movimiento sobre el eje del servo para relice un giro hasta el ángulo *ang* que se le pase como referencia. Para la brújula vamos a usar el modelo HMC5883L; este modelo va a requerir la instalación de dos librerías.

Para la brújula vamos a usar el modelo HMC5883L; este modelo va a requerir la instalación de dos librerías. Este modelo utiliza un protocolo de comunicación I2C [11] para poder comunicarse con la placa y enviarle la información de las medidas que toma. La libería *Wire.h* [21] sirve como interfaz que facilita este protocolo para recibir datos.

Normalmente esta librería se encuentra ya instalada en el entorno de desarrollo; sin embargo, en caso de no tenerla, puede ser instalada fácilmente. Para ello, tan solo tenemos que dirigirnos a la barra de herramientas y vamos a *Programa>>Incluir Librería>>Administrar Bibliotecas*. Visualizaremos una ventana como la que aparece en la figura 6-1. En ella buscamos la librería que queramos y la instalamos, en este caso será *Wire.h* como ha sido mencionado. La única función que nos interesa utilizar es *Wire.begin()*, que sirve para habilitar la comunicación I2C.

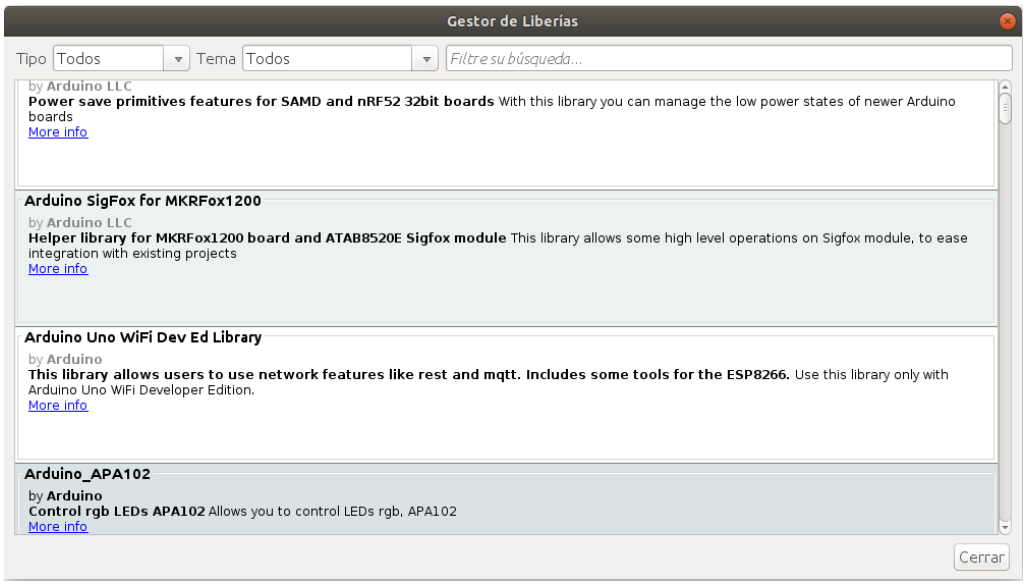

Figura 6-1: Instalación librerías.

Finalmente, la última librería que vamos a instalar será la propia que tiene la brújula, *MechaQMC5883.h*, que incluirá las llamadas necesarias para calcular el valor de la orientación directamente. Normalmente no se encuentra disponible para instalar desde el *Gestor de librerías*, por tanto, hay que descargarla previamente de Internet e instalarla como .zip. El procedimiento será parecido a la instalación anterior: en la barra de herramientas *Programa>>Incluir Librería>>Añadir Biblioteca .zip*.

De forma similar a como ocurría con los servomotores, estas funciones nos permitirán tratar el sensor como un objeto a un alto nivel de programación. Presenta más llamadas, pero las que nos interesan son:

- ***MechaQMC5883* "nombre del objeto"**: Nos permite declarar una estructura de este tipo. En nuestro caso usaremos qmc.
- ***qmc.init()***: Inicializamos el sensor.
- ***qmc.read(&x,&y,&z,&a)***: Lee los valores analógicos sobre los 3 ejes inerciales ('x', 'y' ,'z') y también lee el valor de la azimuth ('a').

Por último, antes de proceder con la programación lo único que nos queda por instalar es el driver para poder trabajar sobre nuestro modelo de *Arduino*, que recordamos que era *Arduino Nano 33 IoT*. Esta instalación se realiza de forma similar a las anteriores. En la barra de herramientas nos redirigimos a *Herramientas>>Placa>>Gestor de Tarjetas.* Se abrirá una ventana donde debemos buscar nuestro microcontrolador y lo instamos.

Con esto último concluimos con los preparativo que hay llevar a cabo para poder desarrollar el algoritmo.

6.3 Código para usar con Rosbot

Este primer código estará desarrollado para ser implementado en el *Arduino* soldado a la PCB y, por ello, al *Rosbot*. La idea reside en intentar conseguir que se comporte como un nodo y, por este motivo debe ser programado con las funciones de las bibliotecas de ROS, entre las cuales nos encontramos *rosserial,* que nos va a permitir comunicarnos con el robot.

Si recordamos el planteamiento inicial de nuestro problema, lo único que debería hacer este nodo sería leer los datos obtenidos referentes a la altitud y azimuth calculadas por ROS. Estos datos sumados a los de la brújula, nos van a permitir calcular unos nuevos de valores de altitud y azimuth corregidas. Una vez calculados estos valores tan solo nos quedará pasárselos como referencia a la función encargada de mover los *servos* [27]. Esto lo estará realizando en el bucle principal indefinidamente.

Los valores de la brújula los vamos a leer de forma análoga a como hemos visto en la anterior sección, utilizando las funciones de las librerías que hemos explicado. Por otra parte, para obtener los valores de la azimuth y altitud de ROS vamos a recurrir al *topic* que creamos en el capítulo previo al presente (*"angulos"*). A través de este enviábamos los mensajes del tipo *Pose2D* con la azimuth y altitud calculadas, así que no debemos olvidar incluir la cabecera de para poder trabajar con este tipo de mensajes.

Seguidamente creamos una variable a la que vamos a denominar *sub* que nos servirá para suscribirnos al *topic*. El siguiente paso es crear la función de arranque que va a interrumpir el programa cada vez que se reciba algún mensaje; esta tendrá por nombre *"void recibe_ang()"*. Esta función tiene como finalidad guardar dentro de las variables *altitude_ros* y *azimuth_ros* el valor de los mensajes recibidos por el *topic*, tal y como se muestra en la figura 6-2.

```
void recibe_ang(const geometry_msgs::Pose2D &msg){
  azimut_ros = msg.x;
  altitud_ros = msg.y;

}
```

Figura 6-2: Rutina de arranque.

6.4 Código para usar sin Rosbot

Inicialmente la intención era realizar el código visto en la sección anterior. No obstante, debido a una serie de problemas logísticos no se pudo llegar a probar el funcionamiento real de la placa, entonces se ha elaborado este programa para poder probar el funcionamiento de los algoritmos desarrollados mediante pruebas con el fin de poder graficar resultados.

En este caso no necesitará comunicación con ROS, por tanto los recursos que obteníamos desde el robot debemos de calcularlos desde el propio *Arduino*. Es decir, tanto los cálculos de las ecuaciones que estiman la posición solar como las propias medidas tomadas por el sensor. Vamos a mantener el código del movimiento de los servomotores y vamos a incluir unas funciones para tomar medidas del sensor y calcular la azimuth y altitud.

Para poder calcular los ángulos aplicaremos los mismos cálculos realizados en el capítulo 5; debemos implementar las mismas ecuaciones. La única modificación que se ha realizado sobre este código es la adición de la biblioteca *Timelib.h* para poder trabajar con el tiempo en *Arduino* creando estructuras en tiempo real.

Por otra parte tan solo nos quedaría comentar como vamos a leer las medidas del ISS-A60; para ello vamos a necesitar fijarnos en las ecuaciones vistas en la sección 3.1.2.3 de esta memoria. En ellas vemos como a partir de las medidas de los fotosensores somos capaces de calcular cuales son los ángulos de incidencia del rayo solar sobre el ISS-A60. Es así como nuestro problema se reduce a leer 4 entradas analógicas, que se puede hacer sencillamente con las funciones básicas que incluye *Arduino* y realizar sobres ellas los cálculos necesarios para encontrar los ángulos. Para esta parte del nos vamos a basar en otro TFM [20].

7 DISEÑO DE LA PCB

7.1 Introducción

En este capítulo trataremos de ver como ha sido el desarrollo del circuito impreso que va a realizar la conexión entre nuestro robot y nuestro sistema de medición.

Se ofrecerá una visión de como es el funcionamiento de la placa, así como de su *pinout* para poder ver las prestaciones que nos puede dar y los posibles periféricos a los que pudiera ser conectada.

7.2 Diseño esquemático

Este diseño de alto nivel trata, como fue comentado en la sección 4.3.2 del capítulo 4, consiste en crear el circuito que queremos implementar sin tener en cuenta parámetros físicos de la placa como es el caso del grosor de las pistas de conexión o tamaño de los componentes. Es una etapa de puro diseño electrónico.

En nuestro caso, al querer conseguir un sistema electrónico que presentará varios elementos y secciones, nos interesará realizar previamente a un diagrama de bloques a mayor alto nivel para poder simplificar el desarrollo de nuestro circuito. En la figura 7-1 vemos dicho diagrama de bloques, el cual pasaremos a explicar a continuación.

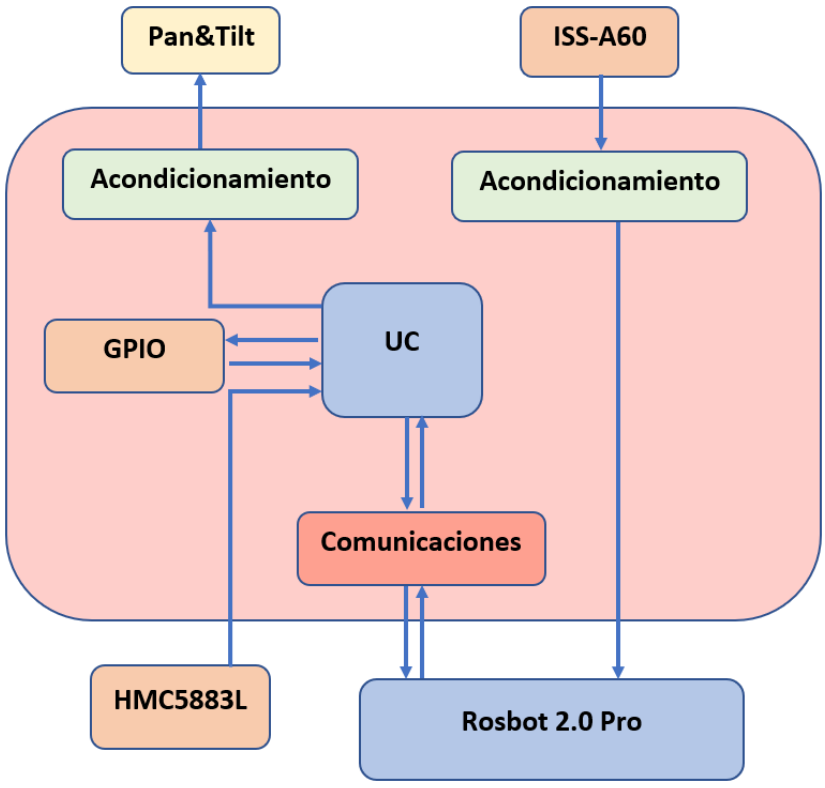

Figura 7-1: Diagrama de bloques de la PCB

A pesar de que se pasará posteriormente a detallar el desarrollo de cada uno de los bloques por separado, vamos a explicar de forma breve el funcionamiento de la PCB.

El microcontrolador será el encargado de leer la señal analógica que proporciona la brújula HMC5883L, a su vez estará conectado también con el robot para recibir y enviar información. En función de los datos captados, calculará la señal de control necesaria para enviar a los *servos* y deberá mandarla a los mismos. Hay que tener en cuenta que dicha señal debe ser acondicionada de tal forma que los motores puedan operar con ella y realizar el movimiento.

Por otro lado, el robot deberá recibir las medidas del sensor solar. Para poder utilizar los paquetes desarrollados e instalados en el *Rosbot* ya existentes, las señales no pasarán por el micro e irán directamente conectadas al robot. De la misma manera que se va a realizar con las señales de los *servos*, las de los fotodiodos también van a ser acondicionadas a las condiciones que exija el robot.

Finalmente, con los pines restantes que, como veremos, serán bastantes, se realizará un puerto GPIO para el microcontrolador con el fin de poder incorporar otros sensores o actuadores por si acaso en futuro fuera necesario.

7.2.1 Unidad de Control (UC)

La unidad de control se trata del *Arduino Nano 33 IoT*. Este dispositivo será el encargado de llevar a cabo la mayoría de las tareas de nuestro sistema. Para realizar un diseño electrónico debemos comprobar el *pinout* de este componente, el cual se encuentra ilustrado en la figura 7-2.

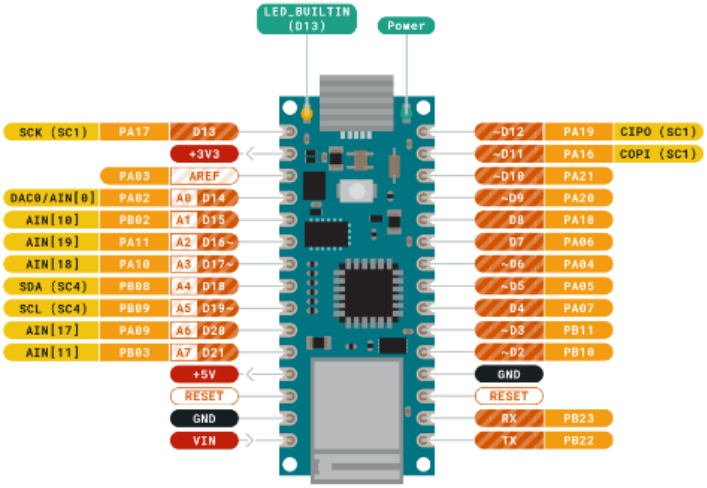

Figura 7-2: Pinout de Arduino 33 IoT

Se puede comprobar que el microcontrolador cuenta con varios pines, muchos de ellos son digitales que pueden ser utilizados como entradas o salidas. Por otro lado, vemos que también cuenta con pines analógicos de entradas para poder leer datos o medidas de sensores.

Si nos fijamos bien, podemos ver que no presenta salidas analógicas, pues este *Arduino* no cuenta con un convertidor digital-analógico; por esta razón, si queremos simular una salida de forma analógica tenemos que recurrir al PWM, que se podrá utilizar en ciertos pines que lo admitan (aquellos que presentan un "~"). Cabe mencionar que estos pines no tienen por qué ser usados para aplicaciones que requieran PWM, simplemente pueden ser entradas o salidas digitales

Sin embargo, como bien sabemos, el PWM no proporciona una salida analógica ya que se encarga de jugar con el valor medio del ancho de pulso de valores digitales. Si quisiéramos obtener una señal puramente analógica tan solo tendríamos que implementar un filtro RC (con una frecuencia de corte mucho menor que la frecuencia del propio PWM) a la salida de cada uno de los pines donde queramos obtener una salida de este tipo de naturaleza. No obstante, en nuestro caso no será necesario recurrir a esta solución ya que los componentes que, en nuestro caso serán los servomotores, funcionan correctamente con señales PWM.

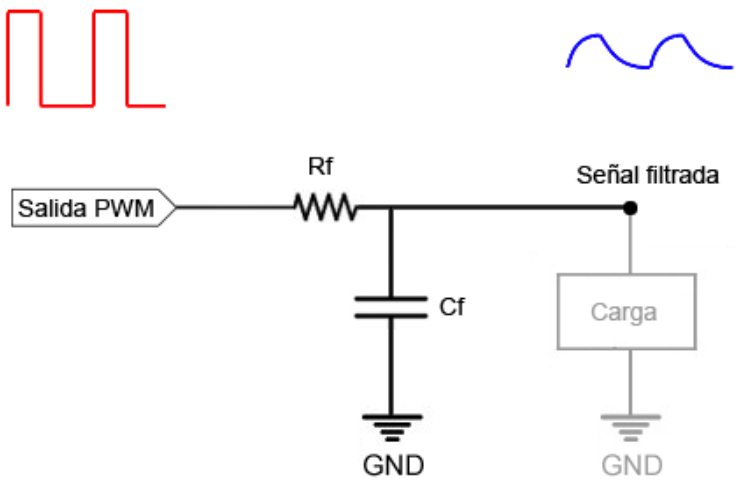

Figura 7-3: Filtro RC para señal PWM.

Tras repasar los pines del microcontrolador, ya podemos comenzar a trabajar con ellos para realizar asignaciones.

Los primeros pines que vamos a asociar serán los pines digitales. Tal y como se puede ver en la figura 7-12, la asignación será de la siguiente forma:

- Los pines D11 y D12 hacen referencia a las señales MOSI y MISO respectivamente, que sirven para poder establecer un protocolo de comunicaciones por si fuera necesario transmitir datos con otro circuito integrado. Estas señales digitales irán directamente a un conector
- Los pines RX y TX también son digitales y, al igual que los anteriores, se utilizan para comunicar el microcontrolador con otros periféricos o módulos de comunicación que sirvan de puente entre ambos dispositivo para establecer un protocolo de comunicación (como es el caso del RS232 [10], UART, etc). Para facilitar el conexionado con posibles módulos de comunicación irán en un conector junto a un pin de GND.
- Los pines D5, D6, D9 y D10, los cuales pueden ser utilizados como señales PWM pasarán por la etapa de acondicionamiento. Esta etapa, como se verá en la próxima subsección, será encargará de amplificar la señal que proporciona el micro de 3,3 V a 5 V. Dos de estas señales D5 y D6 irán dirigidas a los movimientos de los motores del *pan&tilt*. Las otros dos restantes se dejan para libre uso o para posibles actuadores que funcionen a 5 V.
- El resto de los pines digitales (D2-D4, D7, D8 y D11) se conectarán a un conector para poder ser utilizados como señales digitales de entrada y salida para poder incorporar otro tipo de dispositivos de lectura o escritura digital que pudiesen ser necesarios para la PCB, como si fuera un GPIO.

Por otro lado, los pines de entradas analógicas serán distribuidos de la siguiente manera:

- Los pines A4 y A5 son referidos a las señales SDA y SCL respectivamente. Estas entradas definen un protocolo de comunicación serial I2C. La brújula que se utiliza para el proyecto (HMC5883L) funciona mediante este protocolo. Por esta razón, estos pines serán dirigido hacia un conector junto a una señal de 3,3 V y GND para poder facilitar un conexionado directo con la brújula. De este modo no haría falta soldar la brújula a la placa y tan solo debemos introducirla en el conector.
- Como a priori no necesitamos utilizar las entradas analógicas de los fotodiodos del sensor solar, vamos a utilizar el resto de las entradas analógicas como pines de libre uso para poder leer otro tipo de sensores que se puedan incorporar a la placa por si acaso llegasen a ser útiles en un futuro (sensores de

temperatura, humedad, etc). Nuestro microcontrolador, como ya hemos visto tiene un voltaje de operación de 3,3 V, por ello tan solo podrá leer entradas de este nivel de tensión. No obstante, se ha preparado adicionalmente un circuito de acondicionamiento para señales de sensores que funcionen a 5 V con el fin de convertirlas a 3,3 V. De este modo podremos recibir medidas de sensores de ambos niveles de tensión. Ambas tendrán sus propios conectores.

  - Los pines que permiten leer entradas analógicas de un rango de 0 V a 5 V son A6 y A7.
  - Los pines que permiten leer entradas analógicas de un rango de 0 V a 3,3 V son A1, A2 y A3.

- Existen algunos sensores cuyo rango de tensión no está referenciado sobre la misma tensión de referencia (GND) que nuestra PCB, sino que tienen la suya propia que ha podido ser alimentada por otro circuito a parte u otra fuente. Para este tipo de sensores, el *Arduino* propone una solución: la tensión del pin A0 está referida sobre la tensión de referencia del pin AREF. De este modo, utilizando estos pines podemos incorporar a nuestro microcontrolador una entrada analógica diferencial para solventar este tipo de casos.

7.2.2 Acondicionamiento de señales

Esta parte de la PCB tratará de realizar una modificación sobre ciertas señales para poder acondicionarlas al dispositivo hacia donde vayan a ser utilizadas.

Como observamos en el datasheet del microcontrolador, vemos que el *Arduino* trabaja a 3,3V, esto quiere decir que tanto las señales de entradas como las de salida de este trabajarán a esta señal. Análogamente ocurre lo mismo con el *Rosbot*, solo puede leer señales de 3,3V.

No obstante, los dispositivos con los que vamos a trabajar, es decir, tanto los actuadores del *pan&tilt* como los fotodiodos del sensor solar funcionan tienen un rango de operación de 0V a 5V, de tal manera que necesitaremos modificarla. Por un lado, debemos amplificar la señal PWM del micro que controlará los servos para transformar para que esta pase a ser de 0V a 5V; por otro lado, debemos atenuar las medidas tomadas por el sensor para que su rango vaya de 0V a 3,3V.

Para escalar el rango de tensión de los fotodiodos y atenuar así su señal vamos a recurrir a un circuito clásico, el divisor resistivo. En un principio se planteó la idea de utilizar un OPAM de forma que, jugando con su configuración y el valor de sus resistencias pudiéramos conseguir la ganancia que queremos. Sin embargo, al tratarse de un circuito digital le dotará de ruido a las medidas disminuyendo la precisión de los sensores utilizados.

Así llegamos a la conclusión de que la solución más eficiente sería utilizar el divisor resistivo; este circuito funciona muy bien para ajustar rangos de tensiones y, mientras no sea empleado para alguna aplicación de alimentación (que no es el caso), va a dar buenos resultados. Además, tan solo proporcionará el ruido que tengan las dos resistencias que, siempre será menor que el de un OPAMP.

La fórmula de este circuito, ilustrado en la figura 7-4, será:

$$\frac{Vout}{Vin} = G = \frac{R2}{R1 + R2}$$

Nosotros queremos una ganancia de 0.67 para que, aplicando una tensión de entrada (Vin) de 5V obtengamos una salida de 3,3V. De esta forma, si fijamos un valor de 20kΩ para R1, podemos obtener el valor de la segunda resistencia. Quedamos así definido nuestro circuito con los siguientes valores:

$$R1 = 20k\Omega$$

$$R2 = 10k\Omega$$

Para el caso de las señales de los fotodiodos, tras pasar por este acondicionamiento, van a concluir en un conector de donde serán llevadas al robot. En el caso de las señales analógicas A6 y A7, estas irán directamente conectadas al microcontrolador.

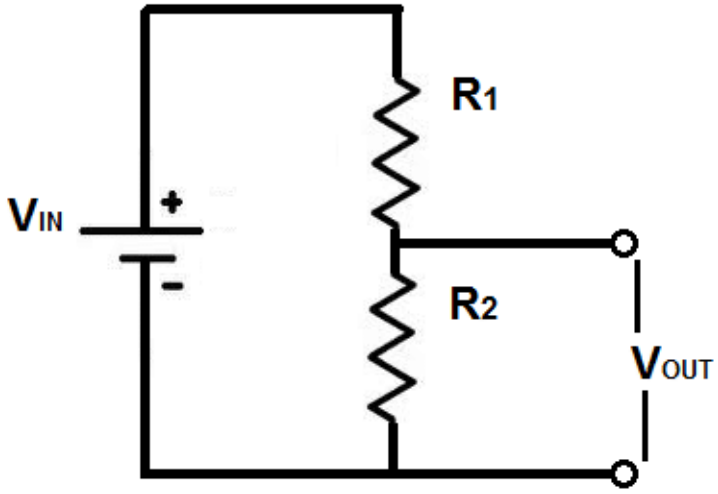

Figura 7-4: Divisor resistivo.

En el caso de las señales de los actuadores querremos realizar el proceso contrario, queremos obtener una señal de 5V a la salida cuando a la entrada le aplicamos 3,3V. En este caso sí utilizaremos un amplificador operacional ya que, al tratarse de una señal PWM, el ruido que pueda llegar a provocar el OPAM no va a afectar de la misma forma que podría perjudicar a una señal analógica (rizado). Se va a emplear la configuración no inversora.

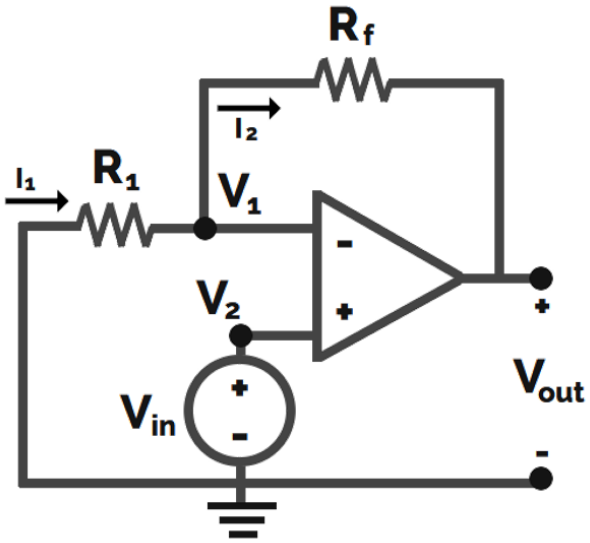

Figura 7-5: Amplificador operacional en configuración no inversora.

La fórmula de esta arquitectura viene dada por:

$$\frac{Vout}{Vin} = G = 1 + \frac{R2}{R1}$$

En este caso, la ganancia que queremos es aproximadamente de un valor de 1,51. Tras realizar varios cálculos con distintas resistencias normalizadas, se encontró que algunos de los posibles valores óptimos para las resistencias eran:

$$R1 = 8{,}2k\Omega$$

$$R2 = 4{,}4k\Omega$$

La ganancia obtenida con estos valores resistivos es de 1,53, por ello podemos decir que no nos hemos alejado mucho y que la solución será bastante aceptable y eficiente. Usaremos resistencias del valor de de *kΩ* para forzar una corriente del orden de *mA*.

7.2.3 Comunicaciones

La mayor parte de lo referido a este subapartado ya ha sido comentada en el bloque del micro, pues se explicaban los pines de RX y TX, MOSI y MISO y SDA y SCL. Todos estos pines, sin contar los dos últimos no van a ser utilizados en un principio, solo si en algún momento son requeridos.

Sin embargo, queda por mencionar la parte fundamental del sistema de comunicaciones: el flujo de datos con el robot. Esta va a ser realizada por medio de un cable USB. Al constar nuestro *Arduino* de una ranura para este tipo de conexiones, no será necesario incluirlo en la PCB.

Finalmente, tras haber comentado todas las secciones que conforman la PCB, realizamos su diseño esquemático, que es mostrado en la figura 7-6.

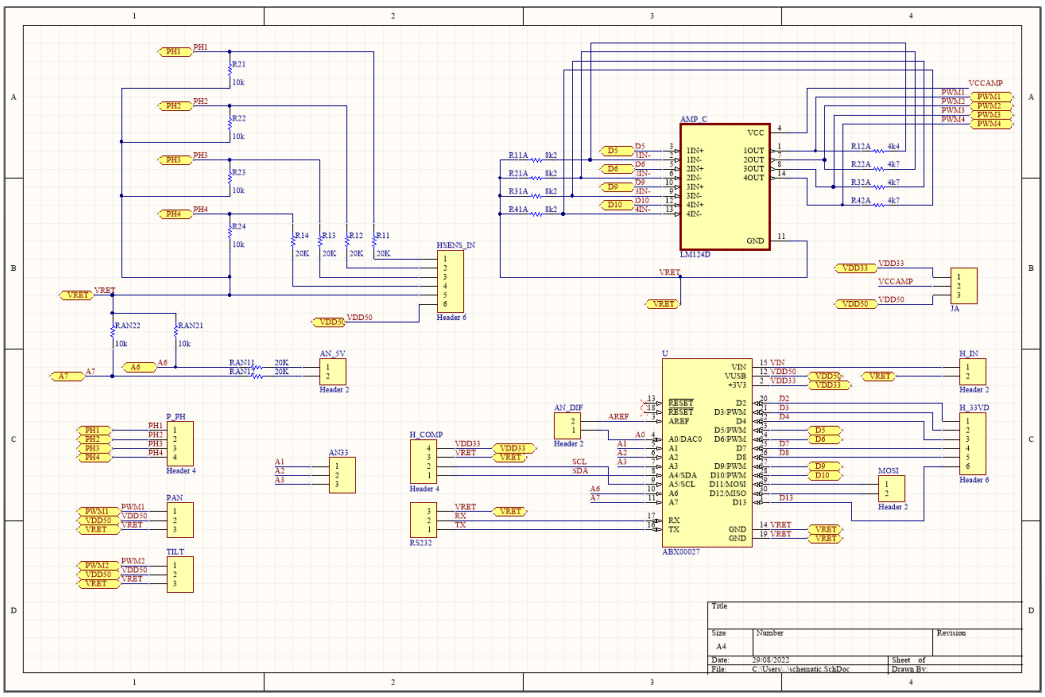

Figura 7-6: Diseño esquemático de la PCB.

7.3 Diseño físico de la PCB

Después de haber diseñado el circuito electrónico de nuestra placa, tocará pasar a la etapa de rutado. En esta etapa ya vamos a pasar a tener en cuenta ciertos parámetros reales, como es el caso del tamaño de la PCB y los componentes que la conforman.

Para ello, lo primero que necesitamos saber antes de proseguir son los dispositivos que vamos a utilizar. El microcontrolador ya fue elegido y explicado en el capítulo 3; los elementos restantes serán las resistencias, los conectores y el amplificador.

En el caso de las resistencias se ha decidido optar por unas de tipo SMD (de montaje en placa) con métrica y encapsulado normalizado 0805. Estas resistencias tendrán unas dimensiones de 0,8'' x 0,5'', es decir serán muy pequeñas. Con esto conseguimos una PCB más pequeña, lo que nos permitirá hacer que sea más sencillas de incorporar en el robot. Además, el coste de estas resistencias no llega ni siquiera al céntimo en la mayoría de los casos, lo cual abarata mucho el precio de la placa.

Para el amplificador diferencial, vamos a usar el LM124D. La idea de usar este dispositivo surge de que ya se ha trabajado con él para otros proyectos y conozco como funciona. Además, no es necesario que sea alimentado con una tensión enegativa, de modo que nos ahorraremos incluir inversores en la PCB.

Finalmente, para los conectores, al no trabajar con altas intensidades que puedan poner en riesgo al usuario que vaya a tratar manualmente con la placa, podemos permitirnos poner *headers* macho 2,54 mm de paso, los cuales son muy baratos. Para conectar la brújula vamos utilizar *headers* de tipo hembra.

Una vez han sido escogidos todos los componentes por fin podemos pasar a la etapa de rutado. En la figura 7-7 podemos observar que en un principio nos encontraremos todos los componentes desordenados. El tamaño de las pistas, al trabajar con corrientes del orden de pocos miliamperios, será de un grosor de 0,254 mm. Finalmente vemos como en la figura 7-8 la placa ya se encuentra diseñada por completo.

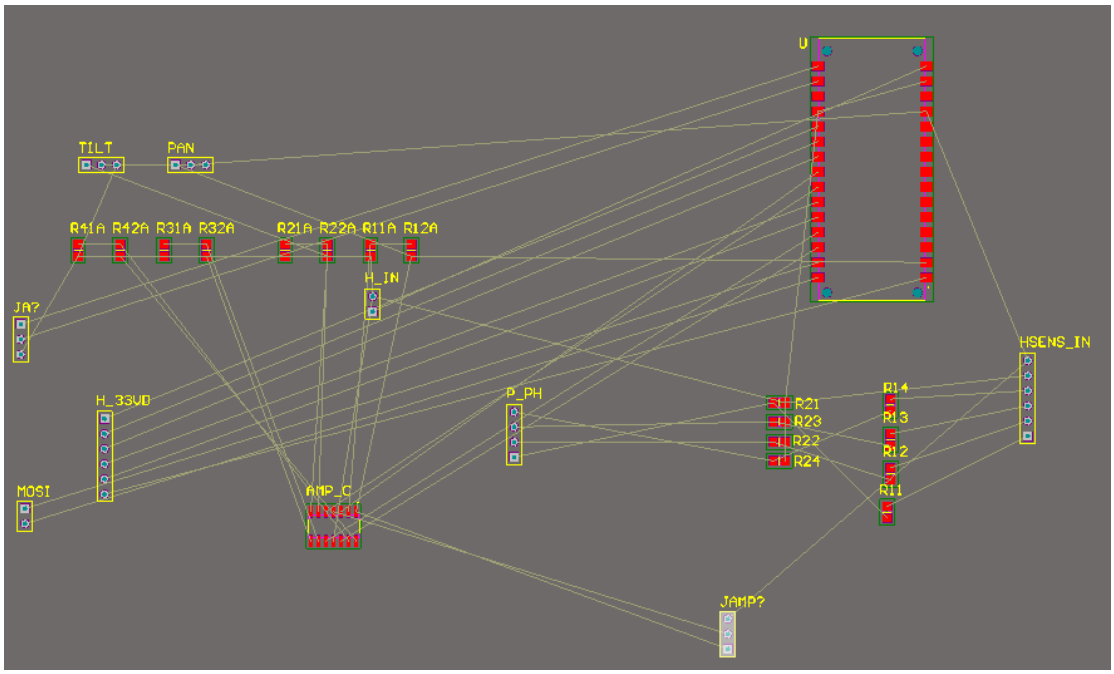

Figura 7-7: PCB en fase previa al rutado.

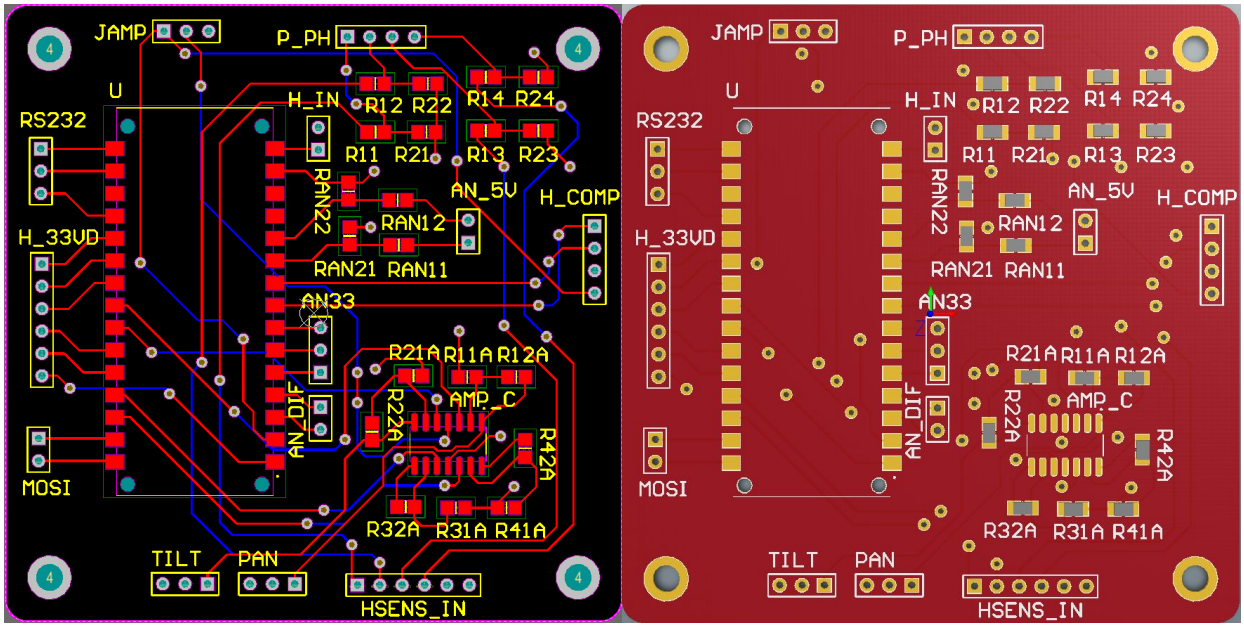

Figura 7-8: diseño final de la PCB.

7.4 Pinout de conexiones

Por ultimo, vamos a realizar una tabla que represente el *pinout* de la PCB diseñada. Para ello, vamos a fijarnos en los conectores y recordamos que el pin número 1 de cada conector viene marcado por la forma de un cuadrado (mientras que los otros tienen forma de círculo) y el resto se ordinan de forma consecutiva este (2, 3, 4…).

Nº pin	Valor	Descripción
1	PAN/TILT	Señal PWM 0-5V
2	VCC	5V
3	GND	Tensión de referencia

Tabla 7-1: Conectores PAN y TILT

Nº pin	Valor	Descripción
1	VCC	3,3 V
2	VAMP	Alimentación del OPAM
3	VCC	5 V

Tabla 7-2: Jumper JAMP

Nº pin	Valor	Descripción
1	A1	Entrada analógica 0-3,3V
2	A2	Entrada analógica 0-3,3V
3	A3	Entrada analógica 0-3,3V

Tabla 7-3: Conector AN33

Nº pin	Valor	Descripción
1	AREF	Entrada analógica 0-3,3V
2	A0	Entrada analógica AREF-3,3V

Tabla 7-4: Conector AN_DIF

Nº pin	Valor	Descripción
1	A6	Entrada analógica 0-5V
2	A7	Entrada analógica 0-5V

Tabla 7-5: Conector AN_5V

Nº pin	Valor	Descripción
1	D11	Señal PWM 0-3,3V o SC1
2	D12	Señal PWM 0-3,3V o SC1

Tabla 7-6: Conector MOSI

Nº pin	Valor	Descripción
1	VIN	Entrada de potencia alternativa
2	GND	Tensión de referencia

Tabla 7-7: Conector H_IN

Nº pin	Valor	Descripción
1	TX	Salida digital para com.
2	RX	Entrada digital para com.
3	GND	Tensión de referencia

Tabla 7-8: Conector RS232

Nº pin	Valor	Descripción
1	D2	GPIO 0-3,3V
2	D3	GPIO 0-3,3V
3	D4	GPIO 0-3,3V
4	D7	GPIO 0-3,3V
5	D8	GPIO 0-3,3V
6	D13	GPIO 0-3,3V

Tabla 7-9: Conector H_33

Nº pin	Valor	Descripción
1	PH1_in	Entrada analógica 0-5V
2	PH2_in	Entrada analógica 0-5V
3	PH3_in	Entrada analógica 0-5V
4	PH4_in	Entrada analógica 0-5V
5	GN	Tensión de referencia
6	VCC	5 V

Tabla 7-10: Conector HSENS_IN

Nº pin	Valor	Descripción
1	PH1_out	Entrada analógica 0-3,3V
2	PH2_out	Entrada analógica 0-3,3V
3	PH3_out	Entrada analógica 0-3,3V
4	PH4_out	Entrada analógica 0-3,3V

Tabla 7-11: Conector P_PH

Finalmente, soldamos nuestra PCB y observamos nuestro sistema de medición en la figura 7-9.

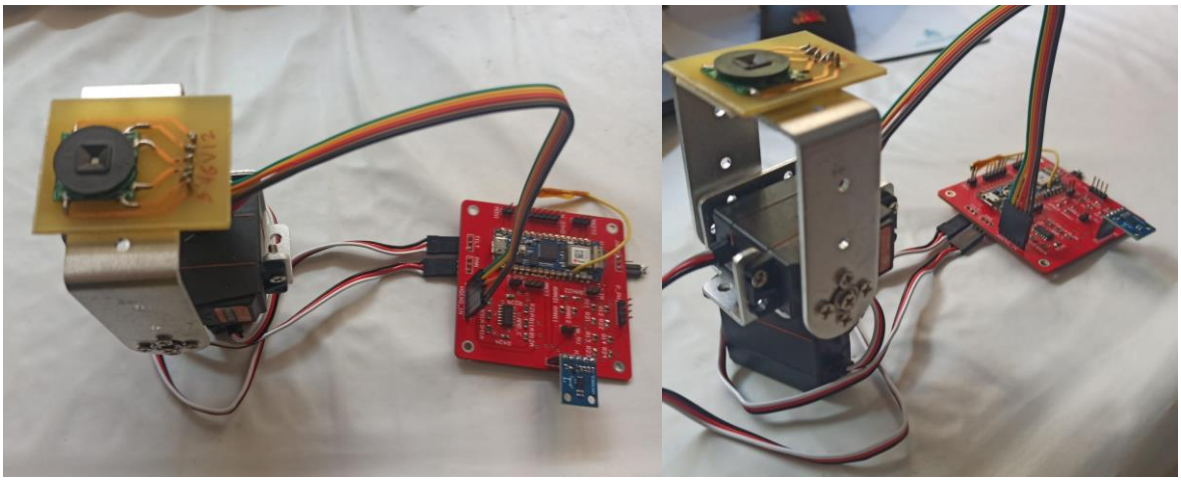

Figura 7-9: Sistema de medición final

8 CONCLUSIONES Y FUTURAS MEJORAS

8.1 Conclusiones

Recordando el objetivo de este Proyecto, nuestra intención era conseguir crear un sistema de medida de radiación solar para ser incorporado sobre un UGV con el fin de poder crear un sensor móvil lo más preciso posible. La idea sería haber realizado pruebas para ver la viabilidad de utilizar un *pan&tilt* de este tipo combinado con el *Solar Mems* y comprobar si somos capces de estimar la radiación solar de forma precisa.

Sin embargo, debido a una serie de factores no se han podido realizar pruebas del sistema funcionando como una sola unidad. Por un lado, el *Arduino Nano 33 IoT* soldado a la PCB comenzó a dar problemas a la hora de cargar programas y, por tanto, impidió que se pudieran realizar pruebas utilizando la PCB, de forma que se tuvo que recurrir a realizar ciertos experimentos con el *Aduino UNO*.

Por otro lado, una parte del TFG fue desarrollada a lo largo de la época de verano y no se disponía del *Rosbot* para comprobar el funcionamiento nuestro sistema con el robot.

Por estos motivos, las conclusiones que podemos obtener de este Proyecto se basan en las pruebas que se han realizado de nuestro sistema de medida por separado, es decir, de cada uno de los componentes que lo conforman.

En primer lugar, hay que mencionar que se ha conseguido desarrollar un nodo en ROS capaz de poder calcular la azimuth y altitud del Sol en tiempo real correctamente, librando al *Arduino* de la carga computacional que podría llegar a darle las ecuaciones no lineales que se utilizan. De este modo es el robot, que tiene un procesador más potente, el encargado de realizar los cálculos difíciles.

Por otra parte, se ha logrado exitosamente crear una conexión entre *Rosbot* y *Arduino* que les permita intercambiar información a través de *topics*. En nuestro caso, somos capaces de poder enviar los datos calculados referentes a la posición del Sol al *Arduino* para que este tome decisiones acerca de cómo girar los servomotores. Por tanto, podemos decir que hemos coseguido orientar nuestro *pan&tilt* en la dirección de los rayos solares gracias a la coordinación entre ambos nodos.

Por último, en lo que a la PCB respecta, antes de que el *Arduino Nano* comenzase a fallar se realizaron pruebas para comprobar que los distintos módulos de la PCB funcionaban. En primer lugar, se comenzó probando el funcionamiento de las etapas de acondicionamiento. En la parte de amplificación de la señal conseguimos obtener una señal PWM de rango 0-5V de modo que llegamos a comprobar que los servomotores podían moverse correctamente.

Por otra parte, también se comprobó que, mediante el divisor de tensiones conseguíamos escalar perfectamente la señal de los fotodiodos que tenían un rango de 0-5V a uno de 0-3,3V. Por ello, podemos suponer que salidas están preparadas para ser introducidas dentro del *Rosbot* y poder ejecutar el nodo de medición previamente desarrollado en un TFM [19].

8.2 Líneas de trabajo futuro

Para concluir esta memoria, en esta sección se van a comentar las futuras tareas que se deberían implementar sobre el trabajo desarrollado.

- Para empezar, se debe estudiar a fondo cual es el problema que ocurre con el *Arduino* soldado a la PCB ya que, cada vez que es conectado a cualquier ordenador, salta una notificación avisando de que no se puede reconocer el dispositivo. Personalmente, tras haber pasado varias horas consultando en

foros intentando solucionar este error, considero que es probable que el *Arduino* haya llegado a un cortocircuito debido a algún tipo de sobrecarga de tensión o intensidad.

- Una vez se haya solucionado este problema de hardware, el siguiente paso sería instalar en el *Rosbot* el paquete desarrollado y modificar el fichero *.launch* para poder ejecutar en paralelo el nodo encargado de tomar las medidas del *ISS-A60* (elaborado en [19]).
- Tras la última tarea, se podrían comenzar a realizar experimentos con el fin de obtener gráficas que nos muestren la fiabilidad de las medidas tomadas con nuestro sistema ya incorporado en el *Rosbot* y compararlas con la radiación real.
- Finalmente, la última tarea que se puede proponer sería afinar las medidas tomadas. Esto podríamos conseguirlo utilizando, en primer lugar, las ecuaciones de Spencer para orientar el *pan&tilt* de manera aproximada hacia el Sol. Posteriormente se usarían las medidas que proporciona el *Solar Mems* acerca de los ángulos de incidencia de los rayos solares sobre el sensor para poder implementar un control en lazo cerrado (un PID por ejemplo) tratando de hacer el error resultante lo más pequeño posible para obtener una medida de radiación más precisa.

REFERENCIAS

[1] Wikipedia. *Ciclo del carbono*. Wikipedia. Online:
https://es.wikipedia.org/wiki/Ciclo_del_carbono

[2] Organización Panamericana de la Salud. *Calidad del aire*. Paho. Online:
https://www.paho.org/es/temas/calidad-aire#:~:text=La%20exposición%20a%20altos%20niveles,vulnerable%2C%20niños%2C%20adultos%20mayores%20y

[3] Linea Verde Ceuta. *Guía de buenas prácticas sobre medio ambiente*. TRACE. Online:
http://www.lineaverdeceutatrace.com/lv/guias-buenas-practicas-ambientales/energia/cuales-son-las-consecuencias-del-malgasto-de-energia.asp#:~:text=Las%20consecuencias%20derivadas%20del%20uso,dependencia%20energética%20y%20contaminación%20ambiental.

[4] Luminia. *Razones para usar energías renovables*. Luminia. Online:
https://luminaenergia.es/razones-para-usar-energia-renovable/#:~:text=Permiten%20ahorrar%20recursos%20naturales%2C%20generan,petróleo%20que%20no%20se%20reponen.

[5] A. Ivette. *Ventajas y desventajas de las energías renovables*. Economipedia. Online:
https://economipedia.com/definiciones/ventajas-y-desventajas-de-las-energias-renovables.html

[6] Husarion. *ROSBOT (2 / 2 PRO/ 2R)*. Husarion. Online:
https://husarion.com/manuals/rosbot/

[7] G. Abal. *Fundamentos del Recurso Solar*. LES EDU. Online
http://les.edu.uy/FRS/clases/U1P2_MAS.pdf

[8] Neurochispas. *Coordenadas esféricas- Fórmulas y Ejercicios*. Neurochispas. Online:
https://www.neurochispas.com/wiki/coordenadas-esfericas/#2-¿que-son-las-coordenadas-esfericas

[9] Z. Peterson. *What is a PCB?*. Altium Resources. Online
https://resources.altium.com/es/p/what-is-a-pcb

[10] O. Weis. *Diferencia entre RS232 y RS485*. Virtual Serie Port. Online:

https://www.virtual-serial-port.org/es/article/what-is-serial-port/rs232-vs-rs485.html#:~:text=La%20interfaz%20RS232%20o%20TIA,comunicaciones%20de%20datos%20(DCE).

[11] Wkipedia. *I2C*. Wikipedia. Online:

https://es.wikipedia.org/wiki/I²C

[12] Navarro Alabarta, J. (2015). *Evaluación mediante simulación de redes de sensores aplicadas a robots móviles y vehículos ligeros* [TFM, Universidad Politécnica de Valencia]. Online:

https://riunet.upv.es/bitstream/handle/10251/60707/Navarro%20-%20Evaluación%20mediante%20simulación%20de%20redes%20de%20sensores%20aplicadas%20a%20robots%20móviles.pdf?sequence=2&isAllowed=y

[13] Wikipedia. *Norma 802.11 IEEE*. Wikipedia. Online:

https://es.wikipedia.org/wiki/IEEE_802.11

[14] Cornell. *On-Board unit*. Cornell. Online:

https://www.law.cornell.edu/definitions/index.php?width=840&height=800&iframe=true&def_id=b314e44117eee975b86a1afbf70641fd&term_occur=999&term_src=Title:47:Chapter:I:Subchapter:D:Part:90:Subpart:M:90.371

[15] Wikipedia. *Red ad-hoc inalámbrica. Wikipedia*. Online:

https://es.wikipedia.org/wiki/Red_ad_hoc_inalámbrica

[16] A. Rodríguez. *Energías solar térmica y fotovoltaica: diferencias y ventajas*. Proinco. Online:

https://blog.proinco.es/diferencias-energia-solar-termica-y-fotovoltaica/#:~:text=Proceso%3A,radiación%20solar%20en%20energía%20eléctrica.

[17] J. L de Benito. *¿Qué son y para qué sirven los piranómetros y los pirheliómetros?*. Energy News. Online:

https://www.energynews.es/que-son-y-para-que-sirven-los-piranometros/

[18] J.G Martin, J.M Maestre, E.F Camacho. (2021). Spatial irradiance estimatition in a thermosolar power plant by a mobile robot sensor network. *Solar Energy.* Sevilla: Elsevier, Solar Energy 220 (2021), 735-744.

[19] García Díaz, J. (2021). *Instalación de Sensor Solar Mems en Rosbot 2.0 Pro con la Plataforma ROS* [TFM, Universidad Sevilla].

[20] Luque Martínez, I. (2022). *Direccionamiento y control de gimbal para seuimiento solar en una planta solar térmica* [TFM, Universidad Sevilla]

[21] Arduino. *Wire.h*. Arduino. Online:

https://www.arduino.cc/reference/en/language/functions/communication/wire/

[22] Solar Mems Technologies S.L. *Sun Sensor ISS-AX analog sensor*. Arazim. Online:
https://arazim.co.il/wp-content/uploads/SolarMems/ISSAX_Technical_Specifications.pdf

[23] Joseph. L. (2018). *Robot Operating System (ROS) for Absolute Begginers, Robotics Programming Made Easy.* Apress

[24] C. Cabrera, A. Posadas, L. Matos. (2007). Estimación horaria de la irradiancia solar terrestre total. *Revista de Instituto de Investigaciones FIGMMG*: Vol. 10, Nº 19 (2007) 72-77.

[25] E.F. Camacho, M. Berenguel. Control of Solar Energy Systems. *IFAC Proceedings*: Elsevier, IFAC Volumes, Volume 45, Issue 15, 2012, Pages 848-85.

[26] E.F. Camacho, JRD Frejo. Centralized and distributed Model Predictive Control for the maximization of the thermal power of solar parabolic-trough plants. *Solar Energy*: Volume 204, 1 July 2020, Pages 190-199.

[27] Muñoz Padilla, A. (2021). *Diseño de un sistema para medir la radiación solar directa* [TFM, Universidad Sevilla]

[28] Martinez Zapata, F. (2020). *Gestión de una flota de robots móviles terrestres* [TFG, Universidad Sevilla]

[29] Masero, E., D. Frejo, J. R., Maestre, J.M., F. Camacho, Eduardo (2021). A light clustering model predictive control approach to maximize thermal power in solar parabolic-trough plants. Solar Energy, 214, 531-541

ANEXOS

2.3. Nodo de ROS

```
#include <ros/ros.h>
#include <nav_msgs/Odometry.h>
#include <string.h>
#include <geometry_msgs/Pose.h>
#include <geometry_msgs/Pose2D.h>
#include <std_msgs/Bool.h>
#include <math.h>
#include <time.h>
//#include <pthread.h>
#include <boost/thread/thread.hpp>
#define PI 3.14159265358979323846264338327950288419 71


//Web para comprobar los ángulos
//https://suelosolar.com/hora-solar

ros::Publisher angulos; //Variable para publicar en un topic


pthread_t time_thread;

double t0=0, t=0, tiempo=1;

//Cambiar el tiempo en función del día (mes 0-11)
int t_h=17, t_min=55, t_s=0, t_d=18, t_mes=4;
geometry_msgs::Pose2D angs;

struct tm fecha_act, fecha_ini={0,0,0,0,1,122};
time_t diferencia;

int dif_hora, suma, dias;

float horas_dia, minutos, horas, segundos;

//Vector que recoge los días que tiene cada mes
int dia_mes[]={31,29,31,30,31,30,31,31,30,31,30,31};


//Parámetros necesarios
float latitud=37.013809*(PI/180);
float longitud=-5.97317;
float correccion=abs(longitud*4);
float declinacion=-1;
float ang_diario, dif_tot;
float Ts, et, w;

//Ángulos que queremos calcular
double acimut, altitud;
//float seno;
```

```
//Rutina que va a ejecutar el hilo encargado de ir contando el tiempo
void *time_routine(void *arg);


//Código principal
int main(int argc, char **argv){
        ros::init(argc, argv, "trajectory_node");
        ros::NodeHandle n;
        angulos = n.advertise<geometry_msgs::Pose2D>("/angulos", 1);
        ros::Rate loop_rate(50);



        pthread_t time_thread;
        t0=0;
        do{
        t0 = ros::Time::now().toSec();
        } while(t0==0);



        if(0!=pthread_create(&time_thread,NULL,time_routine, &t0))
                return -1;

                do{
                        //Vemos cuanto tiempo ha pasado desde que comenzo el año
                        diferencia=mktime(&fecha_act)-mktime(&fecha_ini);
                        horas_dia=horas+minutos/60+segundos/3600;


                        //Tenemos en cuenta el hoario de verano
                        if(fecha_act.tm_mon<3 || fecha_act.tm_mon>9) dif_hora=60;
                        else dif_hora=120;

                        //Comprobamos si el año es bisiesto
                        if((fecha_act.tm_year+1900)%4==0) dia_mes[1]=29;
                        else dia_mes[1]=28;

                        //Contamos los días que han pasado desde que empezó el año
                        for(int i=0; i<=fecha_act.tm_mon-1;i++) suma+=dia_mes[i];
                        dias=suma+fecha_act.tm_mday;
                        suma=0;


                        //Sabiendo el números de días calculamos tau y declinación
                        ang_diario=(2*PI/365)*(dias-1);
                        //printf("\nAngulo diario: %.2f",ang_diario);
                        declinacion=0.006918-0.399912*cos(ang_diario)
                        +0.070257*sin(ang_diario)-0.006758*cos(2*ang_diario)
                        +0.0009907*sin(2*ang_diario)-0.002697*cos(3*ang_diario)
                        +0.00148*sin(3*ang_diario);
```

```
                        //Ecuacion del tiempo (en minutos)
                        et=(0.000075+0.001868*cos(ang_diario)-0.032077*sin(ang_diario)
                        -0.014615*cos(2*ang_diario)-0.04089*sin(2*ang_diario))*229.18;


                        //Cálculo del tiempo solar verdadero (en horas)
                        dif_tot=dif_hora+correccion-et;
                        Ts=horas_dia-dif_tot/60;
                        //printf("\nLocal:%.2f\tSolar:%.2f",horas_dia, Ts);


                        //Obtenemos altitud y acimut

                        w=((Ts-12)*15)*(PI/180);
                        //printf("\n%.2f",w*180/PI);

                        altitud=asin(cos(latitud)*cos(declinacion)*cos(w)
                        +sin(declinacion)*sin(latitud));

                        acimut=acos((cos(latitud)*sin(declinacion)-cos(w)*sin(latitud)*cos(declinacion))/cos(altitud));

                        //seno=sin(acimut);

                        if(w>0) acimut=2*PI-acimut;
                        //if(seno>0) acimut=2*PI-acimut;

                        printf("\nacimut:%.2f \taltitud: %.2f",acimut*180/PI, altitud*180/PI);

                        angs.x = acimut;
                        angs.y = altitud;



                        ros::spinOnce();
                        loop_rate.sleep();


                }
                while (ros::ok());

}


void *time_routine(void *arg){
```

```
while (ros::ok()){
        t = ros::Time::now().toSec() - t0;

        //Calculamos la fecha actual

        if(t>=tiempo){
                angulos.publish(angs);
                t_s++;
                tiempo++;
                if(t_s>=60){
                        t_min++;
                        t_s=0;
                        if(t_min>=60){
                                t_min=0;
                                t_h++;
                                if(t_h>=24){
                                        t_h=0;
                                        t_d++;
                                        if(t_d>=dia_mes[t_mes]){
                                                t_d=0;
                                                t_mes++;

                                        }

                                }
                        }
                }

                //printf("\n%d/%d/2022\t%d:%d:%d",t_d,t_mes,t_h,t_min,t_s);
                horas=t_h; minutos=t_min; segundos=t_s;
                fecha_act={t_s,t_min,t_h,t_d,t_mes,2022-1900};
        }
}
```

2.4. Nodo en *Arduino*

```
#include <Wire.h>
#include <MechaQMC5883.h>
#include <ros.h>
#include <std_msgs/Empty.h>
#include <geometry_msgs/Pose2D.h>
#include <Servo.h>

ros::NodeHandle nh;

Servo servo_azimut;
Servo servo_altitud;

MechaQMC5883 qmc;

//Función de control de los servos
void servos(int );

//Parámetros calculados de la brujula
int ang_brujula=0, x, y, z;


float altitud_ros=90, azimut_ros=90;
int movimiento, alt;
int altitud, acimut;

//Variables del tiempo
unsigned long TiempoMuestreo=60000, ahora=0, pasado=0;
int CambioTiempo;

//Función de retorno
void recibe_ang(const geometry_msgs::Pose2D &msg){
  azimut_ros = msg.x;
  altitud_ros = msg.y;

}

//Nos suscribimos al topic para recibir los angulos
ros::Subscriber<geometry_msgs::Pose2D> sub("/angulos", &recibe_ang);

void setup() {

  pinMode(13,OUTPUT);
  digitalWrite(13,HIGH);

  //Configuramos el puerto serie
  Serial.begin(57600);

  //Configuramos servos
  servo_azimut.attach(5); //pin con pwm
  servo_azimut.write(90);
  servo_altitud.attach(90); //pin con pwm
  servo_altitud.write(6);

  //Conexión con ROS
  nh.initNode();
  nh.subscribe(sub);

  //Inicializamos la brujula
  Wire.begin();
  qmc.init();
}
```

```
void loop() {

  ahora=millis();
  CambioTiempo=ahora-pasado;

  //Actualizamos la posición cada minuto
    if(CambioTiempo>=TiempoMuestreo){
      //altitud_int=altitud_ros;
      //azimut_int=azimut_ros;

      //Leemos la medida de la brujula
      qmc.read(&x,&y,&z,&ang_brujula);
      ang_brujula=ang_brujula * RAD_TO_DEG;
      ang_brujula=ang_brujula-acimut;
      if(ang_brujula<0) ang_brujula = ang_brujula +360;

      //Calculamos los movimientos de los servos
       Servos(ang_brujula);

      nh.spinOnce();
      pasado=ahora;
    }

}



void Servos(int angulo){
  if(acimut<= 180){
    if(angulo>acimut){
      if(angulo-acimut<180){
        movimiento = angulo- acimut;
        if(movimiento>165) movimiento=165;
        else if(movimiento<15) movimiento=15;
        servo_azimut.write(movimiento);
        alt=altitud;
        servo_altitud.write(alt); }
     else{
       movimiento = angulo - (acimut+180);
       if(movimiento>165) movimiento=165;
       else if(movimiento<15) movimiento=15;
       servo_azimut.write(movimiento);
       alt=180- altitud;
       servo_altitud.write(alt); }
    }
     else{
         movimiento = angulo - acimut+180;
       if(movimiento>165) movimiento=165;
       else if(movimiento<15) movimiento=15;
       servo_azimut.write(movimiento);
       alt=180- altitud;
       servo_altitud.write(alt); }
    }
    else{
      if(angulo>acimut){
        movimiento = angulo- acimut;
       if(movimiento>165) movimiento=165;
        else if(movimiento<15) movimiento=15;
        servo_azimut.write(movimiento);
        alt=altitud;
        servo_altitud.write(alt);}
```

```
      else{
        if(acimut-angulo>180){
        movimiento = angulo- acimut+360;
        if(movimiento>165) movimiento=165;
        else if(movimiento<15) movimiento=15;
        servo_azimut.write(movimiento);
        alt=altitud;
        servo_altitud.write(alt);
        }
       else{
         movimiento = angulo- (acimut-180);
         if(movimiento>165) movimiento=165;
         else if(movimiento<15) movimiento=15;
         servo_azimut.write(movimiento);
         alt=180-altitud;
         servo_altitud.write(alt);
       }
    }
  }
}
```