Proyecto Fin de Máster

Máster en Ingeniería Industrial

Instalación de Sensor Solar Mems en Rosbot 2.0 Pro con la Plataforma ROS

Autor: Jaime García Díaz

Tutores:

Jose Ramón D. Frejo

Javier García Martín

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

Sevilla, 2021

Proyecto Fin de Máster
Máster en Ingeniería Industrial

Instalación de Sensor Solar Mems en Rosbot 2.0 Pro con la Plataforma ROS

Autor:
Jaime García Díaz

Tutor:
José Ramón D. Frejo
Javier García Martín

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

Sevilla, 2021

Proyecto Fin de Carrera: Instalación de Sensor Solar Mems en Rosbot 2.0 Pro con la Plataforma ROS

Autor: Jaime García Díaz

Tutores: Jose Ramón Jose Ramón D. 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, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Como no podía ser de otra manera, agradecer en primer lugar a las personas que han estado apoyándome en mi día a día, a mis padres, sin los cuales no podría haber llegado hasta aquí. Gracias por todo el apoyo que me han dado a lo largo de estos seis años de carrera, así como por haberme ayudado en los momentos de mayor agobio y tensión, por ponérmelo todo más fácil para centrarme en mis estudios, y querer lo mejor siempre para mi futuro.

Agradecimientos a mis cuatro hermanos, los cuales han estado para animarme y darme apoyo a lo largo de esta larga etapa.

Al resto de mi familia, por estar en momentos tan importantes como mi graduación, y por desearme siempre triunfar en todo lo que me proponga.

A mis amigos y compañeros, junto a los cuales he pasado grandes experiencias tanto académicas como socialmente, gracias a los cuales, hasta los días de examen se hacían más amenos.

También debo agradecer tanto a la Universidad donde he realizado el grado en Badajoz, como el Máster en Sevilla, a todos aquellos profesores, que me han transmitido conocimientos y ganas de estudiar esta carrera.

Y por supuesto a mis tutores José Ramón y Javier, por acogerme en este proyecto y darme la oportunidad de participar en él.

Jaime García Díaz

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

Badajoz, 2021.

Resumen

Este proyecto, se realiza con Rosbot 2.0 Pro, un robot móvil desarrollado por la empresa Husarion que funciona a través de la programación mediante ROS (Robot Operating System).

Este robot se escoge debido a que el departamento cuenta con varios ejemplares recientemente adquiridos.

Los elementos principales, desde donde se controla el robot, son una placa Core 2 donde se encuentra el firmware que controla todos los sensores y actuadores que utiliza en su funcionamiento, además de una SBC (Single Board Computer) la cual lleva integrado el sistema operativo Linux.

Cuenta además con Lidar, cámara, sensores de proximidad, motores de DC, baterías con una carga de hasta 5 horas, conexiones externas incorporadas de la SBC como HDMI o USB, así como entradas y salidas para incorporar sensores y actuadores en la placa CORE 2, antena Wifi, etc.

El objetivo del proyecto es la incorporación del sensor Solar Mems al robot, para incorporarlo al proyecto de investigación OCONTSOLAR, consistente en la utilización de una flota robótica como red de sensores móviles para la estimación de la radiación solar a lo largo y ancho de las plantas solares. Este proyecto se está llevando en la escuela de ingenierías industriales, desarrollado por el grupo de investigación de Automática y Robótica Industrial, con Eduardo Fernández Camacho como Investigador Principal.

Más concretamente la idea es que el robot reciba datos de la incidencia y radiación solar, con la idea de que, en el futuro, envíe señales al resto de actuadores que posé para su movimiento, así como alguno incorporado, para que, de forma autónoma, busque las posiciones con mayor radiación.

Finalmente, se realizan pruebas experimentales recibiendo la incidencia de la luz solar y artificial, variando la incidencia y ángulo de inclinación y analizando los resultados gráficamente.

Abstract

This project has been realized with Rosbot 2.0 Pro, a mobile robot developed by the Husarion company that works through ROS programming (Robot Operating System).

This robot was chosen because the department bought some of these ones recently.

The robot is controlled manly whit CORE2 board where is programed the firmware that controls all sensors and actuators that are used in its operation. As well as a SBC (Single Board Computer) which has the operating system integrated Linux.

The robot has LiDAR, camera, proximity sensors, DC motors, batteries with charge of up to 3 hours, built-in external SBC connections such as HDMI or USB, inputs and outputs to incorporate sensors and actuators on the CORE2 board, Wifi antenna…

The objective of this project is the incorporation of the Solar Mems sensor to Rosbot 2.0 Pro, to incorporate it into the OCONTSOLAR research project, consisting of the use of a robotic fleet as a mobile sensor network for estimating solar radiation throughout the solar plants. This project is being carried out at the school of Industrial Engineering, developed by the Automatics and Industrial Robotics research group, with Eduardo Fernández Camacho as Principal Investigator.

In particular, the idea is that the robot receives data on the incidence and solar radiation, with the idea that in the future, it will send signals to the rest of the actuators, so that, autonomously, look for the positions with the highest radiation.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Objetivos del proyecto y pasos a seguir.

El objetivo del presente proyecto es la incorporación de un sensor de radiación solar, que mida tanto el ángulo de incidencia del sol, como el nivel de radiación, ya que puede tener un gran beneficio y distintas aplicaciones utilizarlo en ámbitos donde se aproveche la energía solar, como pueda ser en plantas solares térmicas [1]. En dichas centrales, también pueden tener su utilidad para calcular un MPC como se observa en [2]. Ambas citas son artículos relacionados con el uso de robots en el proyecto OCONTSOLAR ya mencionado.

El robot móvil que se utiliza en este proyecto se programa mediante la plataforma de ROS para realizar una integración de las distintas partes y control de este, además de una programación de firmware de más bajo nivel en una placa CORE 2.

Se dividirá el proyecto, de forma que cualquier usuario pueda realizar la instalación de cualquier tipo de sensor o actuador en el robot de forma sencilla.

En primer lugar, se mencionará el estado del arte en el ámbito que nos ocupa, como son las centrales solares, los seguidores solares pirheliómetro y piranómetro, y el sensor utilizado en el proyecto de la empresa Solar Mems.

A continuación, se hará conocer el robot que se va a utilizar, así como su plataforma de programación.

Finalmente, se explicará el proceso llevado a cabo para la instalación del sensor y se darán resultados experimentales para la comprobación del funcionamiento.

1.2 Introducción a la robótica

1.2.1 Antecedentes de la robótica

Desde épocas muy remotas, el hombre ha deseado construir máquinas que tengan forma de seres humanos y le ayuden a realizar las operaciones que le resultan aburridas o peligrosas. A diferencia de un empleado humano, una máquina nunca se cansaría ni enfermaría, y siempre estaría dispuesta a trabajar [3]. Los elementos que pueden funcionar automáticamente se utilizan desde épocas tan remotas como la antigua Grecia, sin embargo, es hasta mediados del siglo veinte cuando se lograron materializar los primeros robots industriales. Estos robots industriales distaban mucho de los sueños de poder contar con una máquina con forma de ser humano. Casi cincuenta años después de la aparición de los primeros robots se sigue trabajando en el diseño y fabricación de estas máquinas similares al ser humano.

En 1961 se instala el primer robot industrial “unimate” en la fábrica de automóviles “General Motors” (figura 1-1), en Tenton (Nueva Jersey USA). Realizaba operaciones de carga y descarga de piezas en una máquina de fundición por inyección de metal.

Figura 1-1. Primer robot Industrial "Unimate"

1.2.2 Bases de la robótica actual

Los robots hoy en día no necesitan asemejarse al ser humano, sino cubrir las necesidades de este, de la forma más eficiente posible. El empleo de robots se ha hecho tan popular dado que son unos excelentes auxiliares en tareas con alto grado de riesgo, o en trabajos que suelen ser desagradables para el ser humano. Pueden encargarse de hacer tareas monótonas y repetitivas durante 24 horas sin bajar su rendimiento. Dentro de sus líneas principales, están las de ensamblado, transporte de piezas, así como los procesos de soldadura y pintura. En la actualidad su campo de aplicación incrementa rápidamente, llegando a puntos inimaginables.

Un robot conjunta diferentes disciplinas como la electrónica, la mecánica, los sistemas computacionales, etc. Exigiendo una plena y bien definida interacción entre cada una de ellas. Cada disciplina aporta elementos fundamentales que son los que determinan la efectividad en el desempeño del robot.

A la hora de realizar un proyecto en robótica, se deben tener en cuenta 5 aspectos imprescindibles que todo robot puede alcanzar en función de la tarea que ocupa:

- **Localización**: consiste en determinar la posición del robot con relación a un mapa dado del entorno.
- **Percepción**: proceso mediante el cual, permite a los robots comunicarse con el entorno por medio de sensores y actuadores.
- **Mecánica**: se trata de la constitución física de los robots. Los componentes que hacen posible el movimiento de este.
- **Control**: indican a cada parte del robot como y cuando funcionar. El componente principal es el procesador o unidad central de procesamiento (CPU).
- **Electrónica**: almacena y envía energía eléctrica a todos los componentes que la requieran. Está formado por baterías y cableado eléctrico.

La programación de un robot es muy diversa, pudiendo utilizar diferentes softwares de programación, los cuales pueden ser específicos de la asociación que lo comercializa o de usos más genéricos, mediante distintas placas de control o con hardware propio.

En el caso que nos ocupa (como se comentará en los siguientes apartados), el Sofware se desarrolla en Robot Operative System (ROS), un Framework especialmente diseñado para el desarrollo de software para robots basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros, el logotipo se puede observar en la figura 1-2.

Figura 1-2. Sistema operativo ROS.

1.2.3 Robots

Los robots, como se explica en [4], son maquinas en las que se integran componentes mecánicos, eléctricos, electrónicos y de comunicaciones, dotadas además de un sistema informático para ser controlados en tiempo real, para la percepción del entorno y la programación. Están compuestos de sensores que reciben datos de entrada, que se conectan con la computadora y disponen de microprocesadores que ordenan al robot la ejecución de las acciones requeridas.

Ofrecen dos tipos de flexibilidad:

- **Flexibilidad mecánica**: es proporcionada por el sistema mecánico articulado que la forma, pudiendo variar su extremo libre en el espacio.
- **Flexibilidad de programación**: proporcionada por el computador. Al poder cambiar a los diferentes programas desarrollados por el programador, puede cambiar la función del robot.

La movilidad de un robot dependerá de los elementos mecánicos que tenga, así como los grados de libertad que lo forman. El grado de libertad de un robot es el número de movimientos independientes que puede realizar.

El esquema general del sistema de un robot [4], se puede observar en la figura 1-3, donde se identifican las partes especificadas en el apartado 1.2.2, como el sistema mecánico, sensores para tener una percepción y localización en el entorno, y un sistema de control para medir y estimar la siguiente actuación del robot, que se transmitirá a través de los actuadores, por medio del sistema electrónico.

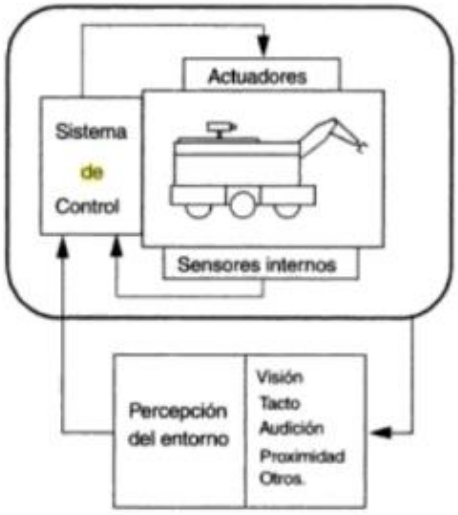

Figura 1-3. Esquema básico de un robot.

El uso de robots trae ventajas e inconvenientes [5].

- Ventajas:
    - Se obtiene una mayor precisión en las tareas, sobre todo cuando se trata de tareas monótonas.
    - Su uso en tareas consideradas peligrosas para el ser humano evita los incidentes y accidentes que puedan ocurrir.

- Mayor velocidad en el proceso.
- Reducción de costos a largo plazo, debido al aumento de productividad.
- Alcanza lugares donde el ser humano no puede llegar.
- Aumento en la calidad.
- Pueden trabajar 24 h al día sin descanso, con la única necesidad del correcto mantenimiento preventivo y correctivo.

- Desventajas:
  - Elevado coste, que a corto plazo no resulta rentable, dada la necesidad de grandes inversiones iniciales.
  - Pueden ser hackeados por programadores especializados, produciendo un cambio en las tareas específicas del robot.
  - Necesidad de grandes gastos de energía para su funcionamiento.

Una de las características del uso de robots es la sustitución de la mano de obra humana. Esto conlleva la reducción de puestos de trabajo, principalmente en fábricas, donde se busca un aumento de la productividad, reduciendo en gran medida los costes de la mano de obra, siendo necesario un aumento de personal más cualificado.

1.2.4 Aplicaciones de robots

Como se indica en [6], la robótica es una de las expresiones de la tecnología cuya aplicación se ha extendido a diversos contextos de la vida del hombre. Además de sus diversas aplicaciones en la industria, hace presencia facilitando y mejorando actividades como:

- Vuelos no tripulados en el estudio del mundo submarino (figura 1-4 [7]).

Figura 1-4. Vehículo submarino no tripulado (AUV).

- La limpieza de piscinas (figura 1-5).

Figura 1-5. Robot de piscina Dolphin S200.

- Aplicaciones en la construcción (figura 1-6).

Figura 1-6. Robot de construcción de doble brazo con función de control remoto.

- La exploración del espacio exterior con robots como Opportunity, el Spirit, el Rocky IV, la Misión Robótica Juno y el Curiosity (figura 1-7).

Figura 1-7. Robot para uso en el espacio (Curiosity).

- En el ámbito del entretenimiento: la creación de robots, como Aibo de Sony (figura 1-8) que simulan características de una mascota, robots que pueden jugar al fútbol, robots móviles, humanoides y muchos otros en los cuales se aplican los últimos adelantos tecnológicos en sonido, reconocimiento y síntesis de voz e inteligencia artificial.

Figura 1-8. Perro Robot Aibo de Sony [8].

- Una de las aplicaciones muy demandadas hoy en día en robots, es en el área de la medicina, donde además de emplearse como prótesis en clientes con discapacidad, también se usan para otras aplicaciones como telecirugía, en la que los robots pueden ser controlados de manera remota por los cirujanos, permitiendo las operaciones a distancia (es el caso del Sistema quirúrgico Da Vinci). En la figura 1-9 se observa el robot Da Vinci.

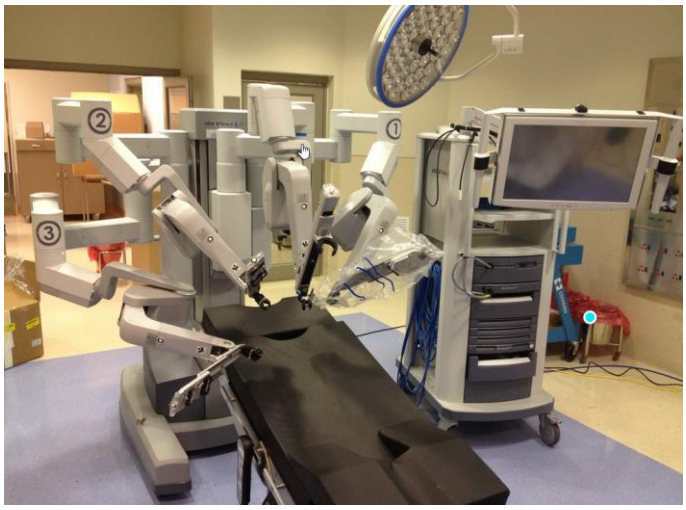

Figura 1-9. Robot de uso medicinal (Da Vinci).

En cuanto a las aplicaciones de **robots de uso industrial** (figura 1-10), se pueden clasificar en las siguientes categorías:

- Manipulación de materiales: se encargan de la carga y descarga para otras máquinas como pueda ser la colocación de herramientas o transferencia de material (pick and place).
- Operaciones de procesado o transformación: soldadura, pintado, revestimientos con spray, etc.
- Montaje e inspección: se encarga del correcto montaje y la verificación de la calidad.

Figura 1-10. Robots de uso industrial en fábrica de coches.

2 ESTADO DEL ARTE

Dado el objetivo de este proyecto, conviene hacer una mención al estado del arte, referente a las centrales solares y los medidores de irradiancia utilizados en el mercado.

2.1 Centrales solares

Como se indica en [9], hacia 1970 las fuentes renovables de energía empezaron a considerarse una alternativa a las energías tradicionales, tanto por su disponibilidad presente y futuro garantizado, como por su menor impacto ambiental, y por esta razón fueron llamadas energías alternativas. Actualmente muchas de estas energías son una realidad, no una alternativa, por lo que el nombre de alternativas ha quedado en desuso para ser utilizado el término renovables.

Las centrales solares son instalaciones destinadas a utilizar radiación procedente de sol para generar energía eléctrica. Existen dos tipos de instalaciones que producen electricidad a partir de la energía solar:

- **Sistemas fotovoltaicos:** la obtención de energía eléctrica se produce a través de paneles fotovoltaicos que captan la energía luminosa del sol, para transformarla en energía eléctrica (figura 2-1). Para conseguir la transformación se emplean celdas fotovoltaicas formadas por metales sensibles a la luz que desprenden electrones cuando los rayos de luz inciden sobre ellos, generando energía eléctrica [10]. Están formados por celdas hechas a base de silicio puro con adición de impurezas de ciertos elementos químicos, siendo capaces de generar cada una de 2 a 4 Amperios, a un voltaje de 0.46 a 0.48 Voltios.

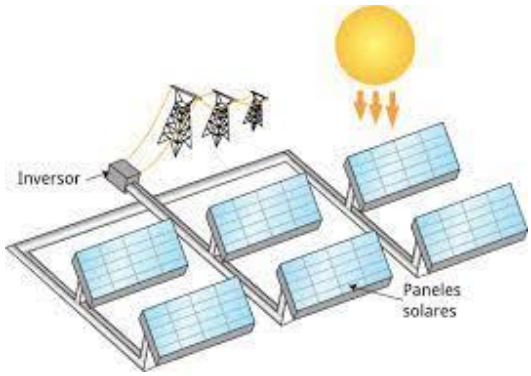

Figura 2-1. Central solar fotovoltaica.

- Sistemas térmico-solares: consiste en el empleo de la radiación solar incidente sobre la superficie terrestre para el calentamiento de un fluido que se hace pasar posteriormente por una etapa de turbina. Tras la etapa compuesta por los equipos solares, como son el concentrador óptico y receptor solar, el proceso se asemeja con las tecnologías termoeléctricas convencionales basadas en la conversión mecánica del calor, calentando un fluido que posteriormente mueve una turbina, y esta, genera energía eléctrica a partir del movimiento de un alternador. Este proceso se puede observar en la figura 2-2.

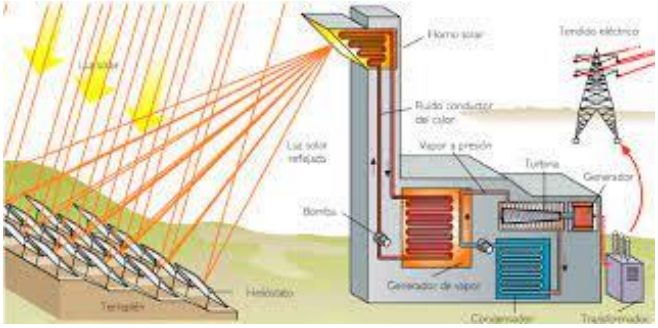

Figura 2-2. Central térmica-solar.

Cuando las placas solares están expuestas a la luz solar, el ángulo con el que los rayos llegan a la superficie de la placa (ángulo de incidencia), es directamente proporcional a la cantidad de energía generada. Cuanto más perpendicular sea el ángulo, más energía produce el panel fotovoltaico, por ello se hace uso de los seguidores solares, los cuales orientan los paneles de manera que su superficie forme 90˚ con los rayos solares.

Hay dos tipos de **seguidores solares según su movimiento**:

- **Seguidos de un eje:** mueve el panel solar en un eje, normalmente alineado norte-sur. Permite al panel moverse de este a oeste. Tienen un menor coste, mayor simplicidad, y la posibilidad de su adaptación a cubiertas, pero realizan un seguimiento menos preciso.
- **Seguidor de dos ejes:** mueve el panel solar en dos direcciones, uno alineado norte-sur y otro este-oeste. Este tipo de sistemas están diseñados para maximizar la producción de energía durante todo el año. Puede variar la orientación según la estación, además de seguir al sol durante el día. Por lo que poseen un seguimiento solar más preciso, con mayores rendimientos, con un mayor coste.

Según su algoritmo de seguimiento, podemos distinguir dos tipos de seguidores solares:

- **Seguidores por punto luminoso:** poseen un sensor que les indica cual es punto del cielo más luminoso y al que debe apuntar. El algoritmo de este tipo de seguidor basa su funcionamiento en la señal integrada por uno o varios sensores. Dependiendo de dicha señal se envía un comando de control a uno o varios motores para que se posicionen en el punto más adecuado de luminosidad.
- **Seguidores con programación astronómica**: estos, mediante un programa, conocen en qué punto debería estar el Sol a cada hora y apuntan a dicha posición. Presenta una total independencia de las condiciones climáticas, ya que su algoritmo no requiere de sensores que indiquen cual es el punto más luminoso. El seguimiento en este caso depende únicamente de una serie de ecuaciones que predicen la ubicación del Sol en cualquier momento

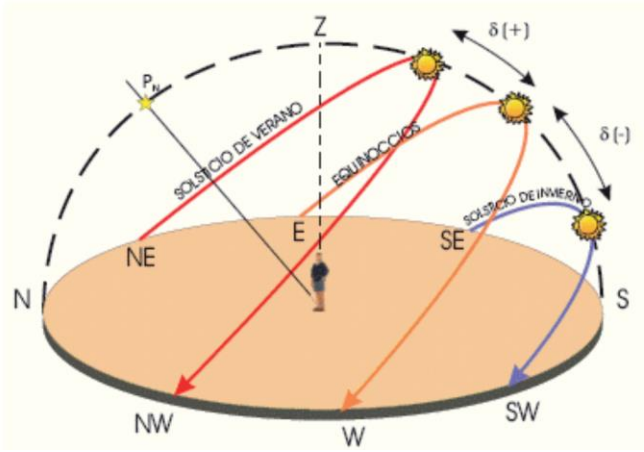

Figura 2-3. Movimiento relativo del sol sobra la superficie terrestre.

Las coordenadas celestes observadas en la figura 2-4 son:

- Ángulo de Azimut solar: ángulo que se mide en planta o vista superior (se da en grados respecto al Norte en sentido horario).
- Ángulo de Altura solar: ángulo que se mide en corte o elevación (se da en grados respecto a la horizontal).

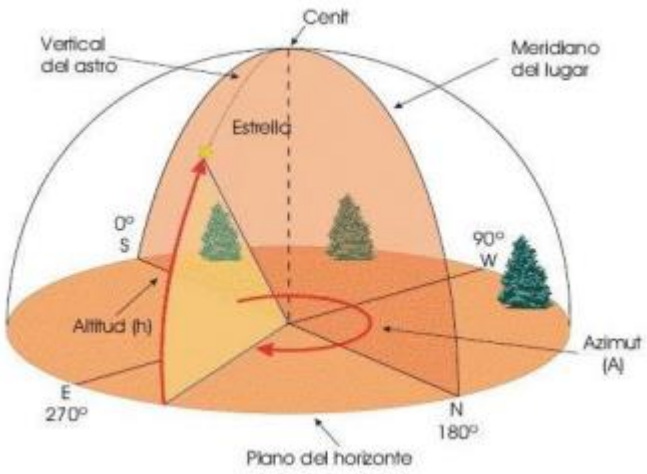

Figura 2-4. Bóveda celeste-coordenadas celestes.

2.2 Sensores de Radiación Solar

2.2.1 Pirheliómetro

Un pirheliómetro [11] es un instrumento para la medición de la irradiancia de un haz de luz solar. La luz del sol entra en el instrumento a través de una ventana y es dirigida sobre una termopila, que convierte el calor en una señal eléctrica, cuyo voltaje se transforma a vatios por metro cuadrado. Se utiliza junto con un sistema de seguimiento solar para mantener el instrumento orientado al sol. El pirheliómetro se utiliza a menudo en la misma configuración con un piranómetro, el cual será descrito en el siguiente apartado.

El Pirheliómetro se desarrolló con la finalidad de establecer medidas cuantitativas del calor emitido por los rayos solares, consideradas como las primeras mediciones cuantitativas que se realizaban en la historia.

Los estudiosos de la materia manifiestan que Pouillet, físico creador del Pirheliómetro, estableció estimaciones de 1228 vatios por metro cuadrado, que se encontraba bastante cerca de las mediciones actuales de 1367 vatios por metro cuadrado.

Se trata de un instrumento de medida muy utilizado como se pude ver en [12].

Figura 2-5. Pirheliómetro.

2.2.2 Piranómetro

Un piranómetro [13] (también llamado solarímetro y actinómetro) es un instrumento meteorológico utilizado para medir de manera muy precisa la radiación solar incidente sobre la superficie de la Tierra. Se trata de un sensor diseñado para medir la densidad del flujo de radiación solar (kilovatios por metro cuadrado) en un campo de 180 grados [14].

Existen dos tipos de piranómetro:

- **Piranómetro Térmico:** se constituye por una pila termoeléctrica contenida en un alojamiento con dos semiesferas de cristal. La pila termoeléctrica está constituida por una serie de termopares, colocados horizontalmente, cuyos extremos están soldados con unas barras de cobre verticales solidarias a una placa de latón maciza. El conjunto está pintado con un barniz negro, para absorber la radiación. El flujo de calor originado por la radiación se transmite a la termopila, generándose una tensión eléctrica proporcional a la diferencia de temperatura entre los metales de los termopares.
- **Piranómetro Fotovoltaico:** el principio de funcionamiento tiene como fundamento el efecto fotoeléctrico. La radiación incide sobre un fotodiodo, que es capaz de diferenciar el espectro solar por la frecuencia de la onda electromagnética, y de ese modo, mediante la lectura de voltaje, conocer los datos de radiación. Son más sensibles a pequeñas irregularidades y cambios, debido a que no tienen inercia térmica.

Figura 2-6. Piranómetro.

2.2.3 Sensor Solar MEMS

Solar MEMS es una empresa que fabrica sensores solares para naves espaciales pequeñas y medianas. Dispone de distintos modelos de sensor de seguimiento solar.

En este proyecto, se utiliza el modelo ISS-AX (figura 2-7), el cual mide el ángulo de incidencia solar en dos ejes ortogonales con respecto a la perpendicular. Dicha información se obtiene de 4 salidas analógicas que posee el sensor.

Se logra obtener una gran sensibilidad a un bajo coste, dada las dimensiones geométricas obtenidas mediante fabricación con la tecnología MEMS (microelectromechanical systems). Posee cuatro cuadrantes fotoreceptores que reciben la incidencia solar a través de una ventana, cuyo valor analógico dependerá del ángulo de incidencia solar.

Este sensor no mide el nivel de radiación directamente, pero se puede obtener de forma proporcional, dado los valores analógicos obtenidos.

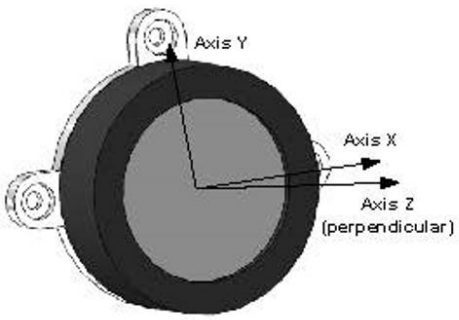

Figura 2-7. Sensor Solar MEMS ISS-AX y sistema de referencia.

3 ROSBOT PRO 2.0

En este Proyecto, se hace uso del robot Rosbot 2.0 Pro, el cual se puede observar en la figura 3-1.

Figura 3-1. Rosbot 2.0 Pro.

3.1 Descripción del Hardware

Rosbot 2.0 Pro es un robot móvil 4x4 con conducción autónoma, cuenta con 4 ruedas independientes con un motor en cada una de ellas, un sensor LIDAR, una cámara Astra RGBD y varios sensores de distancia, velocidad y orientación. Las medidas se pueden observar en la figura 3-2 y en la tabla 3-1.

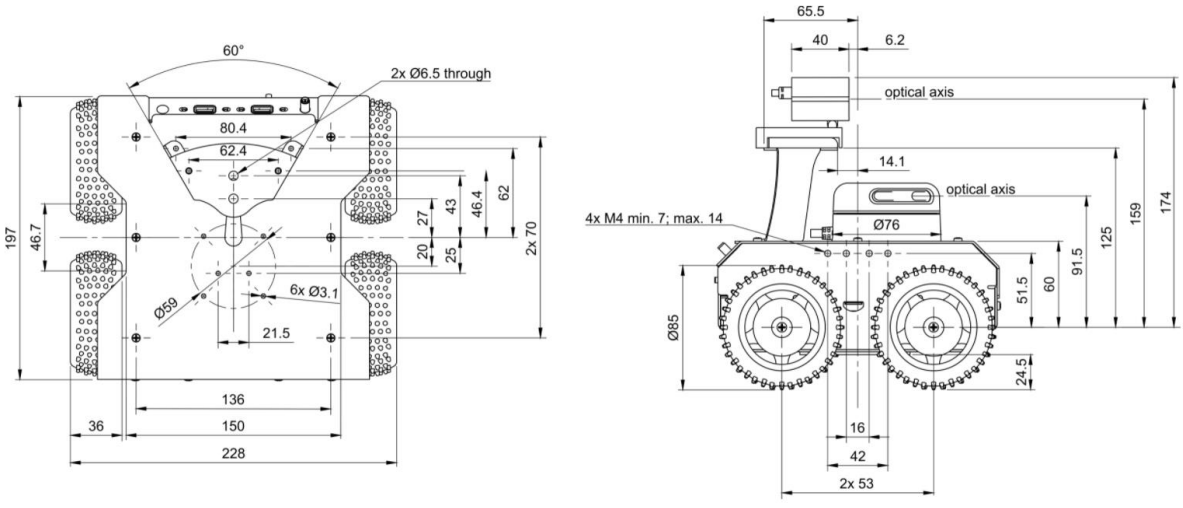

Figura 3-2. Dimensiones Rosbot 2.0 Pro.

Tabla 3-1. Atributos de Rosbot 2.0 Pro.

Atributo	**Frecuencia central de transmisión**
Dimensión con cámara y LiDAR	200 x 235 x 220mm / 7.87 x 9.25 x 8.66in [L x W x H]
Dimensión sin cámara	200 x 235 x 146mm / 7.87 x 9.25 x 5.74in [L x W x H]
Dimensión sin cámara ni LiDAR	200 x 235 x 106mm / 7.87 x 9.25 x 4.17in [L x W x H]
Peso	2,84kg / 100oz (con camara y LiDAR), 2.45kg / 86oz (sin camara ni LiDAR)
Diámetro/Holgura/Distancia de ruedas	85mm / 22mm / 105mm
Material del bastidor	Placa de aluminio con recubrimiento en polvo de 1.5 mm de grosor
Velocidad de traslación máxima	1.0 m/s
Velocidad de rotación máxima	420 grados/s (7.33 rad/s)
Capacidad de carga máxima	10 kg/ 352oz sin estar en continuo uso
Duración de la batería	1.5h-5h

Lleva integrado una SBC (UpBoard UP [15]) con el sistema operativo Linux en el cual se encuentra instalado ROS. Además, tiene una conexión directa con una placa CORE2 que utiliza un microcontrolador STM32F407, el cual se encarga de manejar el firmware del robot [16].

Incorpora también una serie de sensores de proximidad y servo motores DC, enconder, una cámara RGBD, un LIDAR, y diferentes conexiones. La estructura se puede observar en el diagrama de bloques de la figura 3-3.

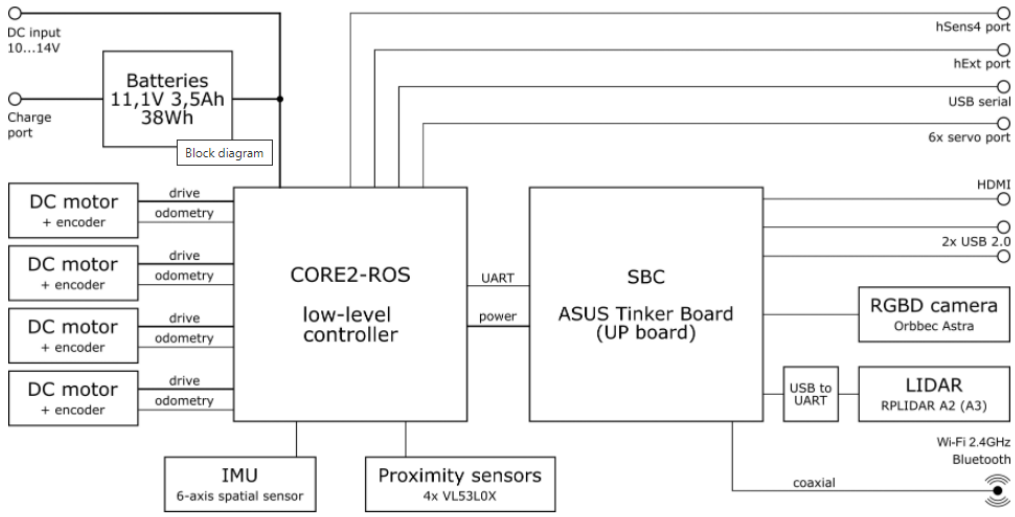

Figura 3-3. Diagrama de bloques de ROSbot 2.0 PRO.

La relación detallada de los diferentes componentes del robot, se pueden ver en la tabla 3-2.

Tabla 3-2. Descripción de los componentes.

Atributo	**Cantidad**	**Descripción**
Sensor de distancia por infrarrojos	4	Sensor de distancia de tiempo de vuelo VL53L0X con alcance de gasta 200 cm [17].
CORE2	1	Controlador en tiempo real basado en microcontrolador STM32F407.
motor de corriente continua	4	Xinhe Motor XH-25D, Motor utilizado: RF-370, 6VDC nominal, 5000 rpm, velocidad sin carga en el eje de salida: 165 rpm, par de parada: 2.9 kg·cm, corriente de parada: 2.2 A, relación de transmisión: ~34 (30613/900), codificador: magnético, 48ppr, 12 polos
Sensor de IMU	1	Potente sensor de aceleración/giroscopio/magnetómetro de 9 ejes con MPU-9250 [18] o sensor inteligente de orientación absoluta de 9 ejes BNO055 [19].
Cámara RGBD	1	Orbbec Astra con un tamaño de imagen RGB de 640x480 y un tamaño de imagen de profundidad de 640x480.
Pilas	3	Li-Ion 18650 protegidas, baterías recargables, 3500 mAh de capacidad, 3,7 V de voltaje nominal. Nota: El dispositivo puede enviarse indistintamente con baterías similares.
Antena	1	Conectado directamente al módulo Wi-Fi ASUS Tinker Board. Utiliza un cable RP-SMA (m) <-> I-PEX MHF4 para conectar la antena con SBC.
SBC	1	Placa superior con 4 GB de RAM, Intel Atom Z8350 de 1,92 GHz de cuatro núcleos como CPU, gráficos Intel® HD 400 como GPU y eMMC de 32 GB. El SBC se ejecuta en un sistema operativo basado en Ubuntu, personalizado para usar ROS.
LiDar	1	RpLidar A3, 360 grados y hasta 25 m de alcance [20].

La placa CORE2 cuenta con una serie de pines para realizar la conexión de sensores externos, los cuales serán de aplicación para la instalación del sensor Solar Mems, como se mencionará en el apartado 5.1. La hoja de características para el nombramiento del conexionado se observa en la figura 3-4 y los pines del robot en la figura 3-5.

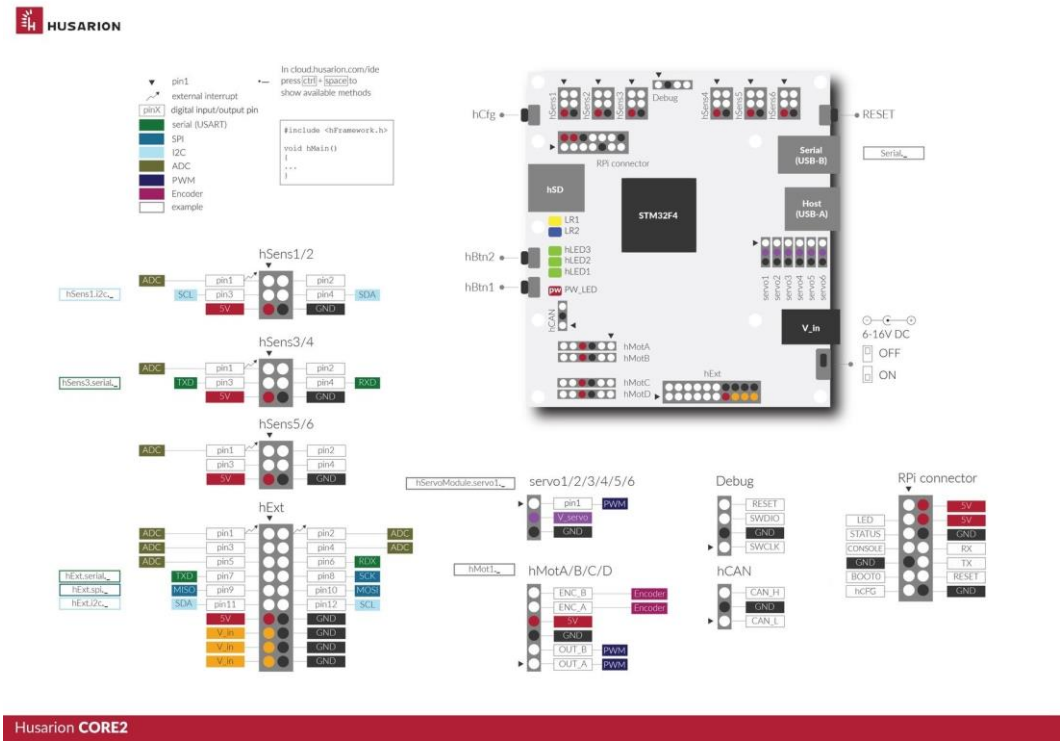

Figura 3-4. Distribución de conexionado de placa CORE2.

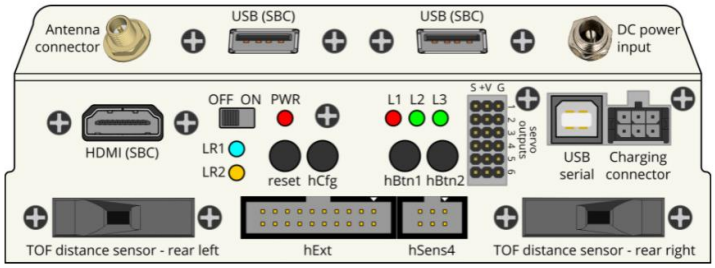

Figura 3-5. Conexiones Rosbot 2.0 Pro

3.2 Descripción del Software

En cuanto al software, podemos diferenciar:

- Sistema de bajo nivel en placa CORE 2 desarrollado mediante el codificador Visual Studio.
- El entorno operativo Ubuntu 18.04, el cual se encuentra instalado en la SBC (Up Board) y contiene todos los componentes necesarios para trabajar con ROS. Dentro de la SBC se incluye la propia MicroSD de 32 Gb, donde está instalado el sistema operativo y sirve como memoria para el dispositivo.
- El Framework de desarrollo de ROS, el cual se explicará con detalle en el apartado 4.

4 ENTORNO ROS

4.1 ROS

ROS (Robot Operating System) es un framework para el desarrollo de software para robots. Fue desarrollado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford.

El sistema operativo de robots (ROS), como se especifica en [21], es un conjunto de bibliotecas de software y herramientas que le ayudan a crear aplicaciones de robot. Desde controladores hasta algoritmos de última generación y con potentes herramientas de desarrollo, todo en código abierto (código de un programa que se distribuye libremente y de manera gratuita que puede ser usado y modificado por los usuarios sin ninguna restricción). Provee de servicios que se esperarían de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comunes, pasaje de mensajes entre procesos y manejo de paquetes.

Se trata de un sistema operativo multidominio y multiplataforma, ya que es compatible con Linux, Windows y macOS, lo que permite el desarrollo y la implementación sin problemas de la autonomía en el robot, la administración de back-end (parte del desarrollo web que se encarga de que toda la lógica de una página web funcione) y las interfaces de uso.

ROS se publica como distribuciones (ROS Noetic, ROS Foxy Fitzroy, ROS Galactic Gechelone, etc).

El objetivo primario de ROS es reusar código en la investigación y desarrollarlo dentro de la robótica. Para ello, utiliza una estructura distribuida en procesos llamados Nodos, que permite el diseño individualizado, pero fácilmente acoplable a los demás procesos. Estos procesos se pueden agrupar en paquetes y pilas, que fácilmente pueden ser intercambiados, compartidos y distribuidos. Los nodos se encargan de manejar dispositivos o algoritmos informáticos, cada nodo para una tarea separada, de forma que se consiga obtener un nivel de granularidad lo más fina posible (figuras 4-1 y 4-2 [22]). Los nodos pueden comunicarse entre sí mediante topic (información unidireccional) o servicios (información bidireccional). El proceso se puede observar en la figura 4-3.

Figura 4-1. Ejemplo de granularidad gruesa

Figura 4-2. Ejemplo de granularidad fina

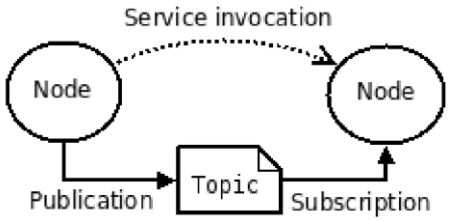

Figura 4-3. Esquema de funcionamiento de ROS.

4.2 ROS en Rosbot 2.0 Pro

En la figura 4-4 se observa la arquitectura de nodos de ROS en Rosbot 2.0 Pro, la cual se ha obtenido del propio robot una vez se han ejecutado todos los nodos de este proyecto, incluyendo el tópico añadido para el sensor Solar Mems. Los tópicos se muestran en un rectángulo, y los nodos en un óvalo.

En primer lugar, el nodo principal "/serial_node" se subscribe a los tópicos de orden de movimiento del robot (/cmd_vel y /cmd_ser), donde se indica la velocidad y dirección del mismo. De los tópicos enviados por /serial_node, cabe mencionar los sensores de proximidad delanteros (range/fl, /range/fr) y traseros (range/rl, /range/rr) del robot, el tópico /adc el cual se programa como se indica en el apartado 6.1, junto con el nodo que se subscribe a este (/sensor_subscriber), programado como se indica en 6.2, además de los tópicos de velocidad (/velocity) y posición (/pose) en los que se encuentra el robot en el momento que publica el mensaje. El nodo "rqt_gui_py_node_2045" es el ejecutado con el comando $rqt_graph en la terminal de Linux.

Todos estos tópicos se encuentran más detallados en la tabla 4-1.

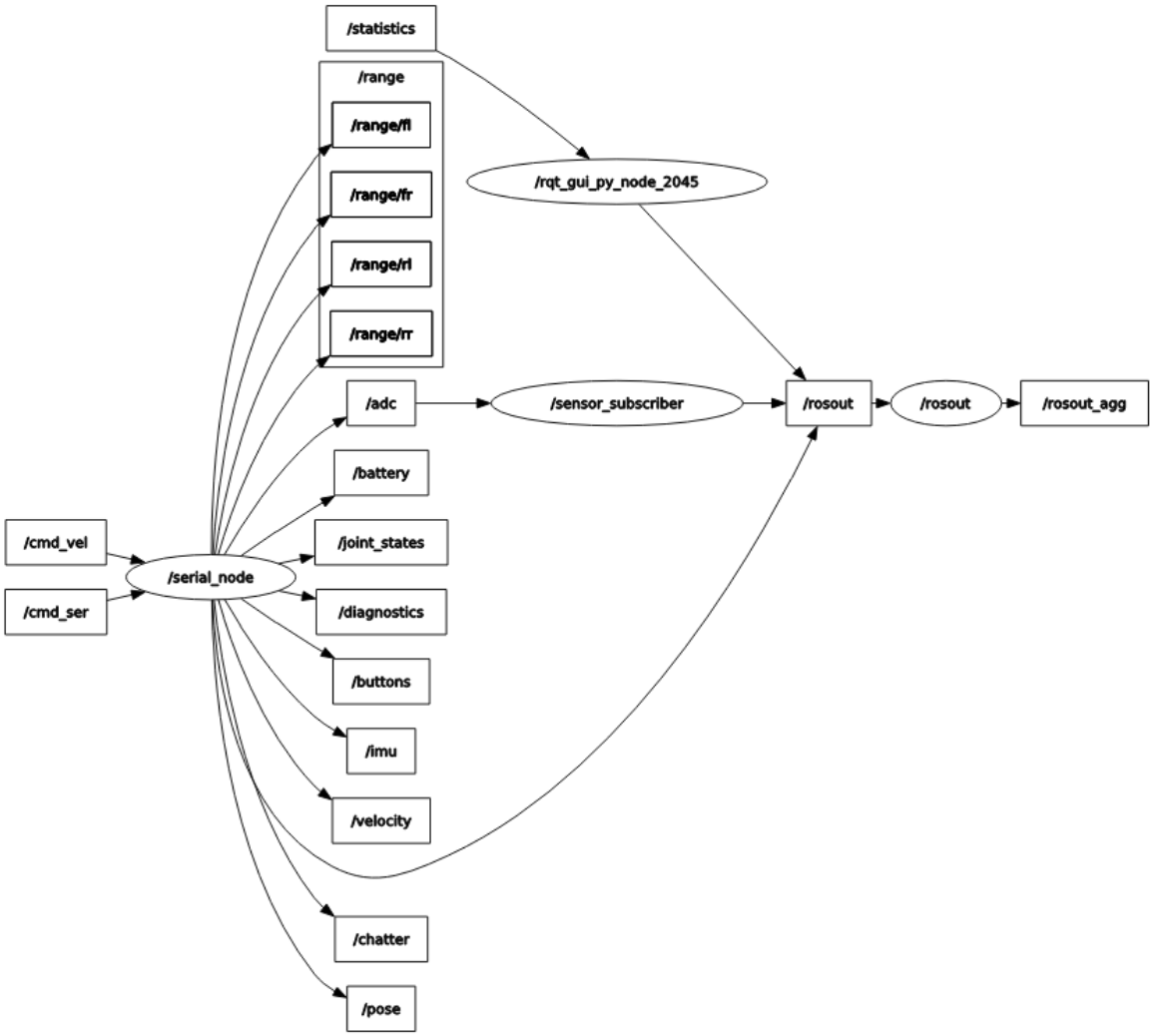

Figura 4-4. Arquitectura de nodos de Rosbot 2.0 Pro.

Tabla 4-1. Arquitectura de ROS en Rosbot 2.0 Pro.

Topic	**Tipo de mensaje**	**Dirección**	**Nodo**	**Descripción**
/mpu9250	Rosbot_ekf/Imu	Publicador	/serial_node	Datos de IMU sin procesar en tipo de mensaje personalizado
/range/fl	Sensor_msgs/Range	Publicador	/serial_node	Datos sin procesar del sensor de rango delantero izquierdo
/range/fr	Sensor_msgs/Range	Publicador	/serial_node	Datos sin procesar del sensor de rango frontal derecho
/range/rl	Sensor_msgs/Range	Publicador	/serial_node	Datos sin procesar del sensor de rango trasero izquierdo
/range/rr	Sensor_msgs/Range	Publicador	/serial_node	Datos sin procesar del sensor de rango trasero derecho
/joint_states	Sensor_msgs/JointState	Publicador	/serial_node	Ángulo de rotación de las ruedas
/battery	Sensor_msgs/BatteryState	Publicador	/serial_node	Voltaje de la batería
/buttons	std_msgs/UInt8	publicador	/serial_node	Esado de los botones de usuario
/pose	Std_msgs/PoseStamped	publicador	/serial_node	Posición basada en codificadores
/odom/Wheel	Nav_msgs/Odometry	publicador	/msg_conversion	Odometría basada en codificadores de las ruedas

/velocity	Geometry_msgs/Twist	publicador	/serial_node	Odometría basada en codificadores
/imu	Sensor_msgs/Imu	Publicador	/msgs_conversion	Datos del IMU empaquetados en el tipo de mensaje ROS estándar
/odom	Nav_msgs/Odometry	Publicador	/rosbot_ekf	Odometría basada en la fusión de sensores
/tf	Tf2_msgs/TFMessage	Publicador	/rosbot_ekf	Posición de ROSbot basada en la fusión del sensor
/set_pose	Geometry_msgs/ PoseWithCovarianceStamped	Suscriptor	/rosbot_ekf	Permitir establecer el estado personalizado de ekf
/cmd_vel	Geometry_msgs/Twist	Suscriptor	/serial_node	Comandos de velocidad
/config	Rosbot_ekf/Configuration	Servidor	/serial_node	Permite controlar el comportamiento de la placa CORE2
/adc	Rosserial_mbed/adc	Suscriptor	/solar_sensor	Recibe datos de los pines del 1 al 4 y transforma los datos (preparado para sensor Solar Mems)

5 INCORPORACIÓN DEL SENSOR SOLAR MEMS

A continuación, se desarrolla el proceso de la instalación del sensor en el robot.

5.1 Configuración de Hardware del sensor con Rosbot 2.0 Pro

El sensor instalado para medir la incidencia de los rayos del sol es el ISS-AX [23]. Este sensor está formado por una ventana que recibe la luz del sol, distribuyendo la misma en cuatro fotodetectores que generan una salida analógica en función de la incidencia solar recibida. En la figura 5-1 se puede observar la ventana por la que traspasa la luz solar e incide en los cuatro cuadrantes.

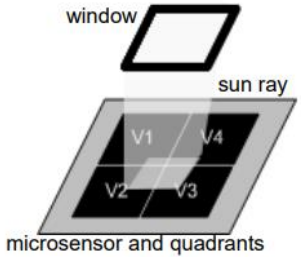

Figura 5-1. Cuadrantes del microsensor instalado.

Tiene 6 pines de conexionado, dos de entrada (alimentación) y cuatro de salida (uno para cada valor analógico) (figura 5-2).

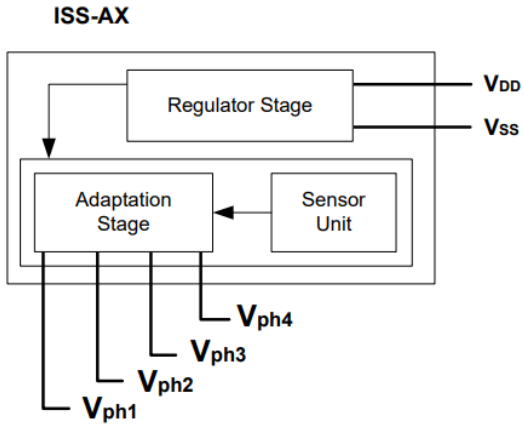

Figura 5-2. Salidas y entradas sensor ISS-AX.

Entre las características del sensor, se pueden destacar las siguientes:

- Campo de visión de 120 grados (modelo ISS-A60).
- Tensión de alimentación de 5-12 V, con una tensión recomendada de 5 V.
- Temperatura de operación entre -40º y 85º.
- Mayor exactitud en la medida, con un ángulo de incidencia de cero grados (perpendicular a la fuente).

Mediante las ecuaciones que rigen el comportamiento del sensor (obtenidas del datasheet insertado en el Anexo B), podemos calcular el ángulo de incidencia solar. Las ecuaciones se detallan a continuación:

- Para el cálculo del ángulo X:

$$X_1 = V_{PH3} + V_{PH4}$$

$$X_2 = V_{PH1} + V_{PH2}$$

$$F_X = \frac{X_2 - X_1}{X_2 + X_1}$$

$$AnguloX = arctg(C \cdot F_X) \quad (5\text{–}1)$$

- Para el cálculo del ángulo Y:

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

$$AnguloY = arctg(C \cdot F_X) \quad (5\text{–}2)$$

El valor de la constante C depende del modelo de sensor. Para un sensor ISS-A60 tiene el valor C=1.871.

V_{PH1},V_{PH2} , V_{PH3} y V_{PH4}, son los 4 voltajes obtenidos de los fotodiodos de la figura 5-3.

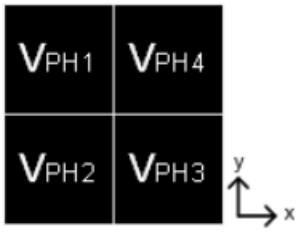

Figura 5-3. Fotodiodos del sensor y ejes.

Con las ecuaciones indicadas, se calcula los ángulos de incidencia X e Y, observados en la figura 5-4.

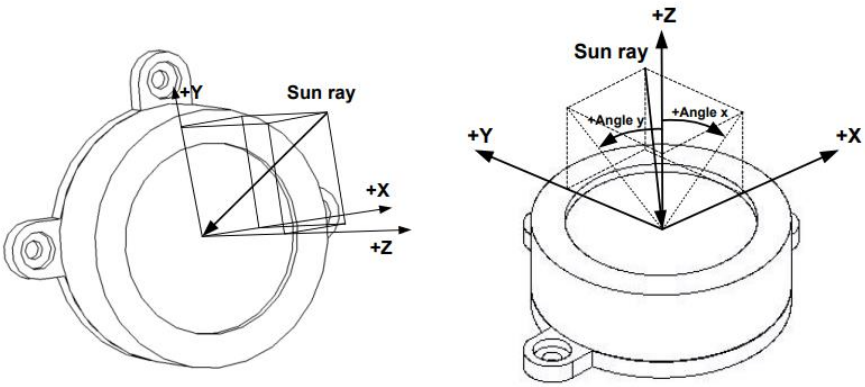

Figura 5-4. Ángulo de incidencia del sol en el sensor.

Para realizar el conexionado del sensor, dado que las salidas de este son analógicas, se hará uso de los pines que posee el robot en la placa Core2.

Debido a que la placa CORE2 solo recibe señales de 0 a 3.3 V, pudiendo dañarse en caso de tensiones de mayor magnitud, es necesario el diseño de una placa con un divisor de tensión, que sirva de adaptación para cada salida analógica.

Para realizar el diseño del divisor de tensión, se calculan las resistencias para una tensión de entrada variable entre 0 y 5 V, y una tensión de salida entre 0 y 3.3 V.

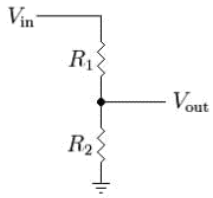

Figura 5-5. Divisor de tensión.

$$V_{OUT} = \frac{R_2}{R_1 + R_2} \cdot V_{in} \tag{5–3}$$

Para un voltaje de entrada máximo de 5 V, el de salida debe ser de 3.3 V

$$3.3 = \frac{R_2}{R_1 + R_2} \cdot 5 \tag{5–4}$$

$$R_1 \approx 2 \cdot R_2 \tag{5–5}$$

Se obtienen los voltajes deseados, con las resistencias:

$$R_1 = 20k\Omega$$

$$R_2 = 10k\Omega$$

Se fabrica la placa manualmente, mediante una PCB de soldadura manual, en la que se conectan las cuatro salidas analógicas del sensor en la posición Vin del divisor de tensión, y llevando las salidas Vout a los pines de entrada del robot que se observan en la figura 5-6.

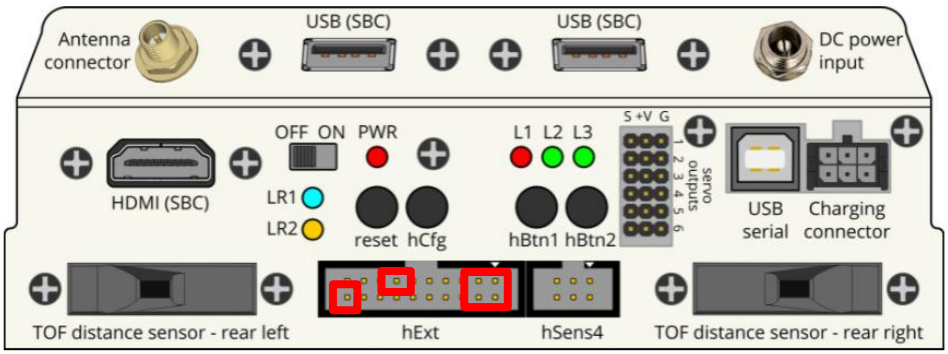

Figura 5-6. Pines de conexión del sensor y la fuente de alimentación de 5 V (rojo).

Dichas entradas corresponden con la posición 1, 2, 3 y 4 de la figura 5-7 y la 13 y 20 para la alimentación de 5 V.

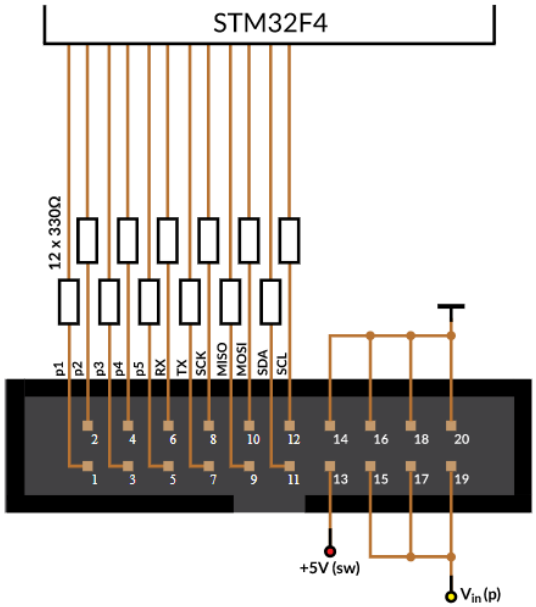

Figura 5-7. Enumeración de pines de placa CORE 2 del robot.

La programación del conexionado en la placa CORE 2 se desarrollará en el apartado 6.1.

La configuración final, una vez instalado el sensor junto a la PCB, se observa en la figura 5-8.

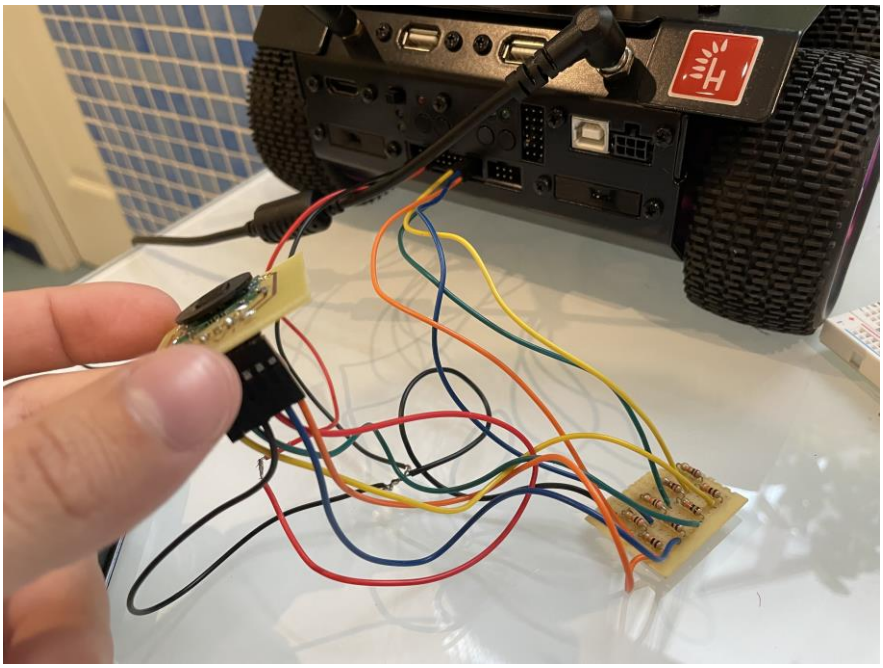

Figura 5-8. Sensor y placa PCB conectados a Rosbot 2.0 Pro

5.2 Configuración de firmware del robot mediante rosbot-stm32-firmware.

Para enviar los datos del sensor a la SBC, hay que configurar el firmware de la placa CORE2. Para ello, se ha hecho uso del paquete de ROS más actualizado (rosbot-stm32-firmware [24]).

5.2.1 Instalación de programas y dependencias

Para el uso del paquete, es necesario la instalación de diversos programas:

- **Mbed Os** [25].
- **Visual Studio Code IDE** [26]. Plataforma de programación en diferentes lenguajes, incluido los utilizados en este proyecto (C++ y Python).
    - **Microsoft C/C++ extensión**: extensión para Visual Studio, es una instalación interna desde la plataforma Visual Studio Code IDE.
    - **PlatformIO IDE**: descarga interna de Visual Studio. Se utiliza para el desarrollo de dispositivos embebidos, realiza la compilación y ejecución de programas en placas CORE 2, Arduino, etc.
- **Stm32loader:** paquete utilizado para flashear el firmware en la placa CORE 2 [27].

Se deben cargar las dependencias dentro del paquete de ROS "rosbot-stm32-firmware", mediante el comando en la terminal de Linux:

$ git submodule update --init –recursive

En la figura 5-9 se observa la carpeta "rosbot-stm32-firmware" una vez abierta en Visual Studio junto a las descargas ya realizadas.

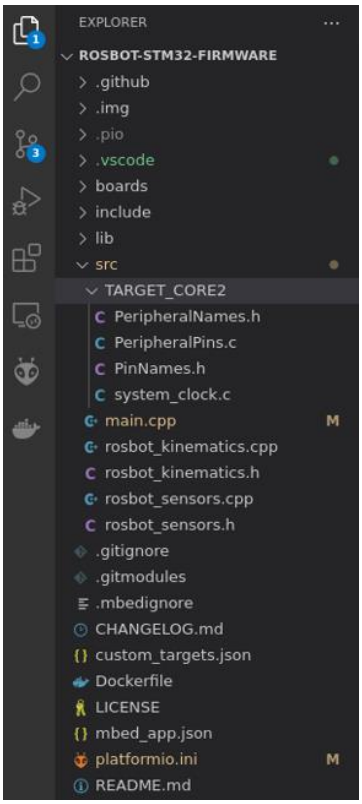

Figura 5-9. Paquete Rosbot-Stm32-Firmware abierto en Visual Studio Code.

5.2.2 Compilación

Para compilar, en primer lugar, se tendrá en cuenta la versión de ROS que se está utilizando. Por defecto, el paquete está preparado para ROS Noetic. En caso de compilar para distribuciones anteriores, habrá que eliminar "-D ROS_NOETIC_MSGS=1" de "platformio.ini" (figura 5-10).

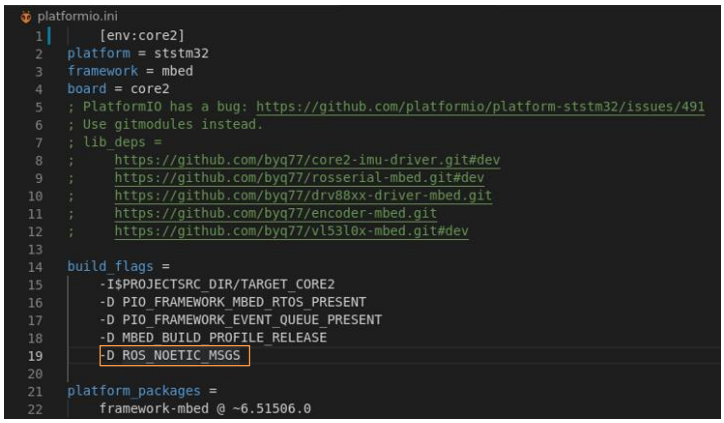

Figura 5-10. Directorio platformio.ini para compilación.

Para compilar el firmware creado, se usará “PlatformIO: Build”.

Para incrementar la velocidad de compilación, se crea un file llamado “.mbedignore”, donde se ha incluido el siguiente contenido:

cellular/*

cryptocell/*

deprecated_warnings/*

lorawan/*

lwipstack/*

nanostack/*

netsocket/*

nfc/*

unsupported/*

5.2.3 Flasheado

El flasheado del firmware modificado, se realiza con el paquete stm32loader, dado que dicho paquete se encuentra instalado de fabrica en el robot.

Una vez modificado el firmware, habrá que dirigirse a la carpeta “./pio/core2”, la cual se encuentra dentro del paquete rosbot-stm32-firmware. En esta carpeta se encuentra el nuevo firmware desarrollado con el nombre “firmware.bin”, que habrá que instalar en la placa CORE 2, mediante los siguientes comandos:

$ sudo stm32loader -c <your_sbc> -u -W

$ sudo stm32loader -c <your_sbc> -e -v -w firmware.bin

En el caso de Rosbot 2.0 Pro:

$ sudo stm32loader -c upboard -u -W

$ sudo stm32loader -c upboard -e -v -w firmware.bin

5.3 Configuración de software del robot utilizando ROS Noetic en Linux.

Una vez se ha compilado y flasheado el firmware del Robot en la placa CORE 2, cuya programación se incluye en el apartado 6.1 de esta memoria, se realiza la unión con el resto de los nodos del robot, a través de la SBC.

Para ello se creará un Workspace de ROS [28] para realizar la unión del sensor instalado, junto con el resto de los paquetes que se encargan del funcionamiento del robot. Para este proceso se ha realizado los pasos del tutorial

de husarion de Rosbot 2.0 Pro [29], junto con los tutoriales generales de Ros [30].

El workspace creado se ha llamado “ros_workspace” y se ha incluido las carpetas:

- **Rosbot_ekf:** carpeta donde se programa el software de movimiento del robot, conectándose a la placa CORE 2 a través del paquete “rosserial” [31].
- **Rosserial:** paquete que actúa de puente entre la SBC y la placa CORE 2 [32].
- **Solar_sensor:** carpeta que se ha creado para incorporar el sensor en el entorno de trabajo de ROS del robot.

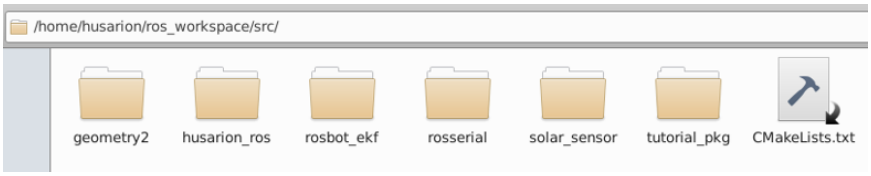

Figura 5-11. Carpetas de ROS en Rosbot 2.0 Pro.

Dentro de la carpeta de “solar_sensor”, se crean 3 subcarpetas:

- **Src:** para incorporar el nodo “configuración_sensor.cpp”. El desarrollo de la programación se incluye en el apartado 6.2.
- **Msg:** se añade un tipo de mensaje necesario para suscribirse a los datos que envía el sensor instalado “adc.msg”.
- **Launch:** se crea un file (rosserial.launch) desde el que se realiza la ejecución de todos los nodos necesarios para el control del robot.

6 PROGRAMACIÓN DEL ROBOT 2.0 PRO PARA LA INSTALACIÓN DEL SENSOR SOLAR MEMS

6.1 Programación de firmware de CORE 2 con microcontrolador stm32F407

La programación del sensor se incluye en el "main.cpp" del paquete "firmware-stm32-new". Es necesario activar los pines del robot indicados en el apartado 2.1, en los que se conecta el sensor.

Las lecturas son enviadas a la SBC mediante el tópico creado "/adc", junto con el resto de los tópicos del robot. Para crear estos tópicos, se ha utilizado el sistema de ROS dentro de la placa CORE 2, el cual tiene una sintaxis similar a la desarrollada desde la SBC con Linux.

En primer lugar, añadimos las bibliotecas necesarias para poder utilizar los pines del robot ("PinNames.h" y "PeripheralNames.h") como se observa en la figura 6-1.

```
23  #include <sensor_msgs/Range.h>
24  #include "tf/tf.h"
25  #include "tf/transform_broadcaster.h"
26  #include <std_msgs/UInt8.h>
27  #include <rosbot_ekf/Configuration.h>
28  #include <map>
29  #include <string>
30
31  #include "mbed.h"
32  #include "PeripheralNames.h"
33  #include "PinNames.h"
34
35  #include <ros.h>
36  #include <std_msgs/String.h>
37  #include <rosserial_mbed/Adc.h>
```

Figura 6-1. Librerías añadidas en código main.cpp de CORE2.

Como ya se mencionó en el apartado 5.1, se van a utilizar los pines 1, 2, 3 y 4 de los hext, cuyos nombres se muestran en la figura 6-2.

```
C PinNames.h
C system_clock.c
main.cpp M
rosbot_kinematics.cpp
C rosbot_kinematics.h
rosbot_sensors.cpp
C rosbot_sensors.h
.gitignore
.gitmodules
.mbedignore
CHANGELOG.md
custom_targets.json
Dockerfile
LICENSE
mbed_app.json
platformio.ini M
README.md
```

```
402   SENS_POWER_ON = PG_4, // SENSOR 5V TOGGLE
403
404   /**** EXT PORT PINS ****/
405   EXT_PIN1 = PF_3,
406   EXT_PIN2 = PF_10,
407   EXT_PIN3 = PF_4,
408   EXT_PIN4 = PF_5,
409   EXT_PIN5 = PC_0,
410   EXT_PIN5_ALT0 = PC_0_ALT0,
411   EXT_PIN5_ALT1 = PC_0_ALT1,
412   EXT_PIN6 = PD_6, // RX
413   EXT_PIN7 = PD_5, // TX
414   EXT_PIN8 = PB_13, // SCK SPI
415   EXT_PIN9 = PC_2, // MISO SPI
416   EXT_PIN9_ALT0 = PC_2_ALT0,
417   EXT_PIN9_ALT1 = PC_2_ALT1,
418   EXT_PIN10 = PC_3, // MOSI SPI
419   EXT_PIN10_ALT0 = PC_3_ALT0,
420   EXT_PIN10_ATL1 = PC_3_ALT1,
421   EXT_PIN11 = PF_0, // SDA I2C
422   EXT_PIN12 = PF_1, // SCL I2C
```

Figura 6-2. Nombre de pines hext en placa CORE2.

Se incluye el tipo de mensaje del paquete "rosserial_mbed", mencionado en el apartado 2.3. La librería a añadir es "rosserial_mbed/Adc.h" como se observa en la figura 6-2. El mensaje recoge los datos de tipo uint16, los cuales son números enteros de 2^{16} bits (65536 bits), que equivale a la medida tomada por los pines del robot mencionados en el apartado 5.1 de 0 – 3.3 V (0 – 65536 bits).

Se crea un objeto con el tipo de mensaje mencionado, como se observa en la figura 6-3.

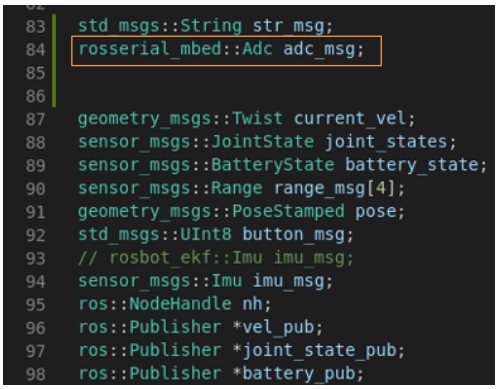

Figura 6-3. Objeto de tipo rosserial_mbed.

Se añade el publicador con el nombre del tópico "/adc" como se indica en la figura 6-4. De esta manera, podremos enviar datos a través del publicador creado.

Figura 6-4. Publicador con tópico /adc y tipo de mensaje enviado adc_msg.

Se incorpora la función "averageAnalog(PinName pin)", cuya entrada es el pin que se está analizando, y salida la media de 4 valores analógicos obtenida mediante un bucle for, el cual lee la entrada analógica, transformándola a bits mediante la función "AnalogIn(pin).read_u16( )", como se puede ver en la figura 6-5.

Los valores obtenidos, se almacenan en cuatro variables de tipo "PinName" (adc0, adc1, adc2, adc3).

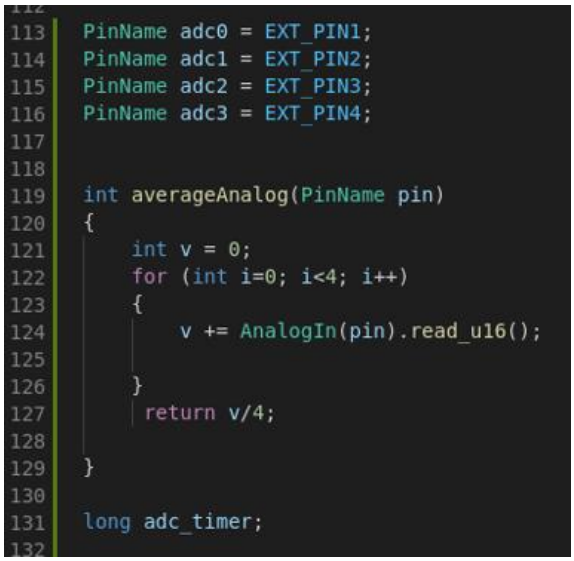

Figura 6-5. Variables y función averageAnalog ( ).

Dentro del int main( ), se inicializa el publicador "p" con el tópico "/adc", como se observa en la figura 6-6 y 6-7.

```
925  int main()
926  {
927      int spin_result;
928      int err_msg = 0;
929      uint32_t spin_count = 1;
930      float curr_odom_calc_time, last_odom_calc_time = 0.0f;
931
932      ThisThread::sleep_for(100);
933      sens_power = 1; // sensors power on
934      ThisThread::sleep_for(100);
935      odom_watchdog_timer.start();
936
937      rk = rosbot_kinematics::RosbotKinematics::kinematicsType(0);
938      rk->setOdomParams();
939      RosbotDrive &drive = RosbotDrive::getInstance();
940      MultiDistanceSensor &distance_sensors = MultiDistanceSensor::getInstance();
941
942      drive.setupMotorSequence(MOTOR_FR, MOTOR_FL, MOTOR_RR, MOTOR_RL);
943      drive.init(rosbot_kinematics::custom_wheel_params, RosbotDrive::DEFAULT_REGULATOR_PARAMS);
944      drive.enable(true);
945      drive.enablePidReg(true);
946
947      button1.mode(PullUp);
948      button2.mode(PullUp);
949      button1.fall(button1Callback);
950      button2.fall(button2Callback);
951
952      nh.initNode();
```

Figura 6-6. Int main( ) donde debe incluirse el nuevo publicador creado.

```
983      nh.advertiseService(config_srv);
984      nh.subscribe(cmd_vel_sub);
985      nh.subscribe(cmd_ser_sub);
986
987      initBatteryPublisher();
988      initPosePublisher();
989      initVelocityPublisher();
990      initRangePublisher();
991      initJointStatePublisher();
992      initImuPublisher();
993      initButtonPublisher();
994
995
996      nh.advertise(chatter);
997      nh.advertise(p);
998      char hello[13] = "hello world!";
999
1000     DigitalOut led = LED1;
1001
```

Figura 6-7. Añadimos el publicador con nh.advertise(p).

El firmware del robot tiene un bucle while(1), el cual se repite mientras se ejecuta el "rosserial.launch" indicado en el apartado 5.3, enviando los tópicos creados por los desarrolladores. Se incluye por tanto en este bucle el tópico como se muestra en la figura 6-8, donde se ha utilizado la función creada "averageAnalog", para obtener la media de cada entrada analógica del sensor e incluirla en el objeto creado "adc_msg" de tipo "rosserial_mbed::adc", para publicar el mensaje en el tópico "/adc".

Finalmente se publica el mensaje con "p.publish(&adc_msg)".

Figura 6-8. Se calcula la media de los valores obtenidos y se publica el mensaje.

El tópico "/adc" envía por tanto los datos obtenidos del sensor, convertidos a bits.

6.2 Programación de software del robot utilizando ROS Noetic en Linux

Para la programación del software de control del robot, mediante ROS Noetic en Linux, se ha utilizado el código que se observa en las figuras 6-9 y 6-10, para suscribirse al tópico que se ha creado desde la placa CORE 2 como se indica en el apartado 6.1.

El código se ha incluido en el nodo "configuración_sensor.cpp", el cual se encuentra indicado en el apartado 5.3. El nodo se suscribe al topico "/adc", recibiendo los valores en el objeto "msg".

Se crea una estructura tipo "información_sensor" para almacenar la información de los ángulos de incidencia y de la radiación.

Se añade la función "delay()" para recibir los datos al iniciar el subscriptor con una retención de tiempo que se estime en la variable "secs". El motivo de crear esta función, y no utilizar la propia función que incorpora ROS (ros::Rate [33]), es que cuando se programa con dicha función, no se ha conseguido detener el tiempo de envío de mensajes en el bucle del subscriptor.

Se han escogido únicamente dos valores analógicos de los fotoreceptores para medir la radiación (pines 2 y 3), debido al propio funcionamiento del sensor, ya que, al realizar las pruebas experimentales, se comprueba, que dichos valores varían en mayor proporción que los otros dos, debido a que los fotoreceptores de estos pines, son los que reciben la mayor parte de la incidencia del sol, como se puede ver en la figura 5-1, y en la figura 7-2.

El motivo de representar la radiación en bits, se debe a que no se conoce los valores de radiación en la medida, ya el datasheet del sensor solo menciona el ángulo de incidencia y no considera mediciones máximas y mínimas de radiación solar, por lo que no se pueden realizar estimaciones de la conversión a W/m^2, ni tampoco se conocen valores reales de la radiación tomada el día 04/11/2021 en Badajoz a las 17:15 para hacer un cálculo proporcional.

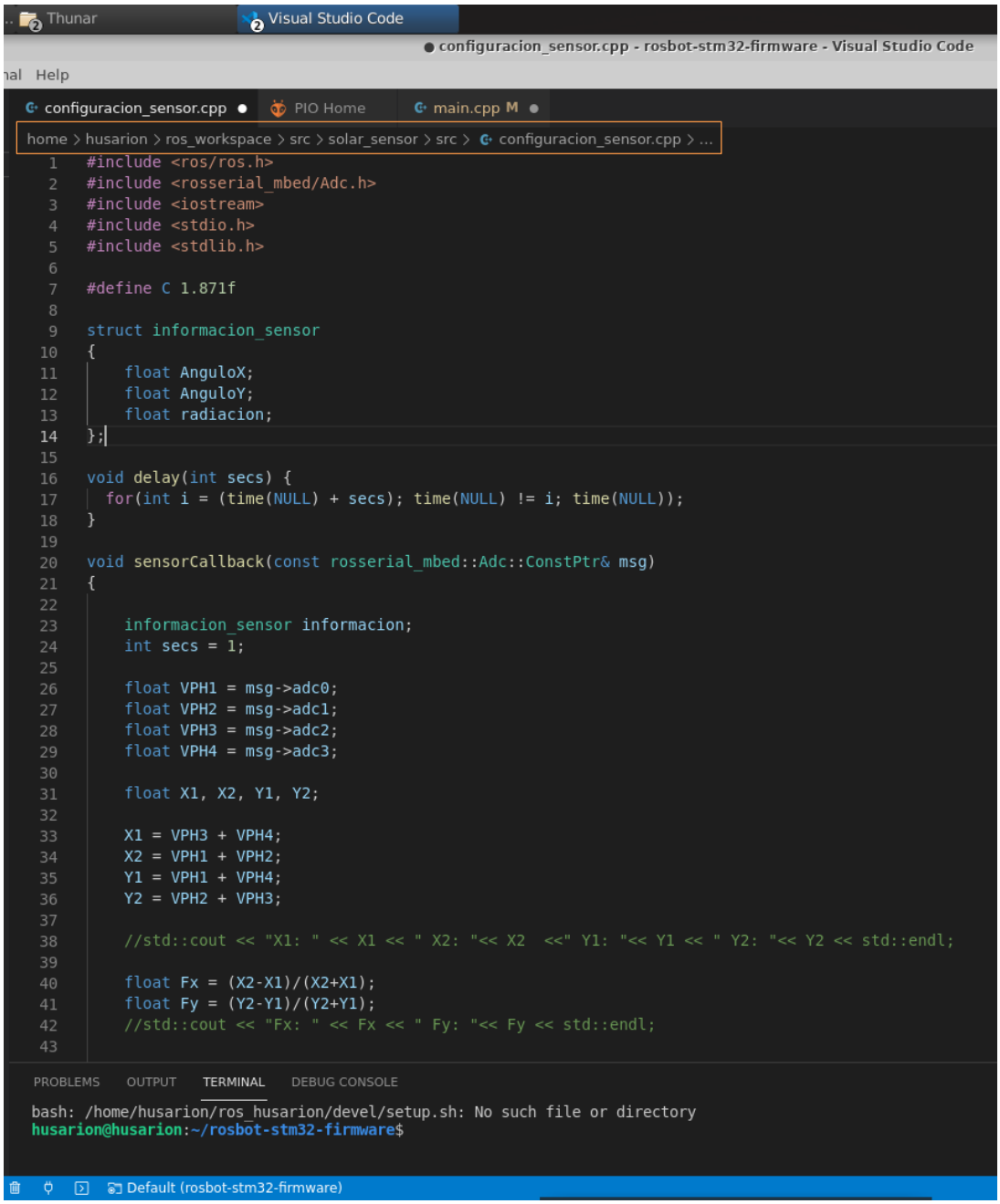

Figura 6-9. Programa configuración_sensor.cpp

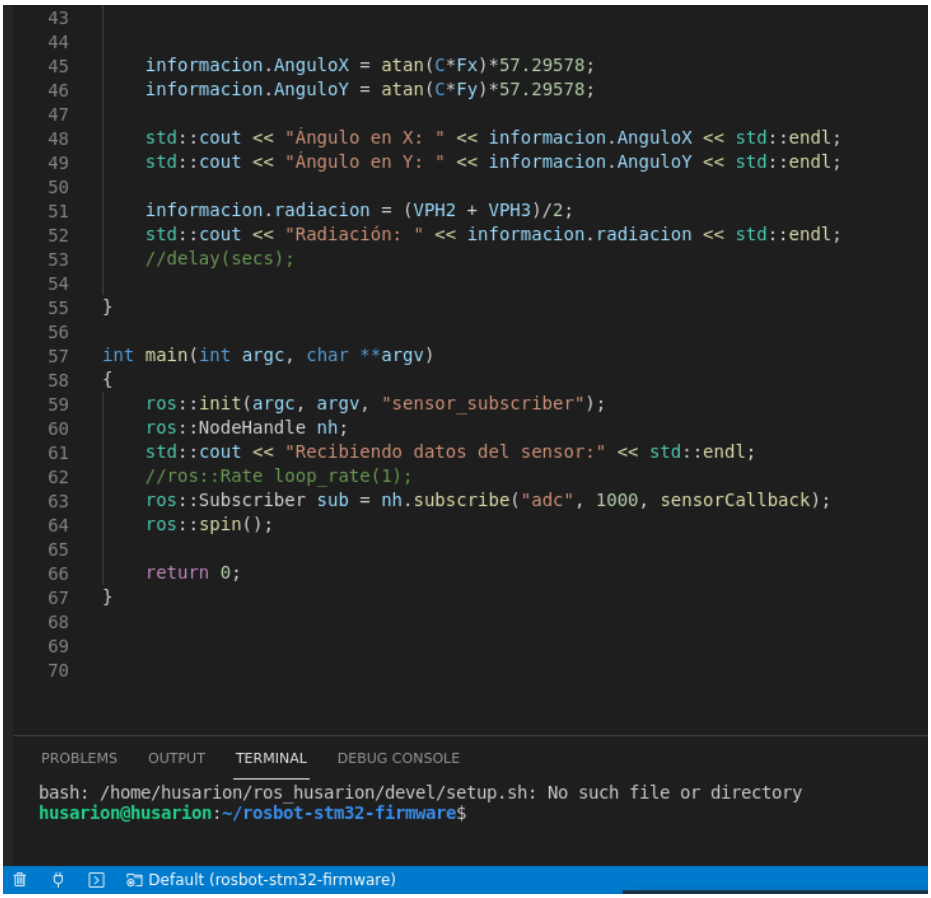

Figura 6-10. Programa configuración_sensor.cpp 2.

6.3 Flasheado

Antes de iniciar cualquier nodo, al abrir un terminal en Linux, es necesario cargar el setup.bash. En este caso, se utilizar el comando:

$ source ~/ros_workspace/devel/setup.bash

Cuando se inicia el fichero launch, no se precisa ejecutar el máster, ya que se inicia de forma automática en el robot.

Se crea un fichero .launch como se menciona en el apartado 5.3, el cual se observa en la figura 6-11.

```
<launch>
  <arg name="serial_port" default="/dev/ttyS4"/>
  <arg name="serial_baudrate" default="525000"/>
  <node pkg="rosserial_python" type="serial_node.py" name="serial_node" output="screen">
    <param name="port" value="$(arg serial_port)"/>
    <param name="baud" value="$(arg serial_baudrate)"/>
  </node>
</launch>
```

Figura 6-11. Launch para iniciación del robot y sensor.

Mediante este launch se ha introducido el puerto de conexión de la placa CORE 2 con la SBC y el serial_badruate del Rosbot 2.0 Pro [34].

De esta forma se inician todos los nodos del robot, mediante el puente, a través del paquete de “rosserial_python” que se encuentra dentro del paquete “rosserial” instalado en el “ros_workspace”. Se puede ver en la figura 6-12 el resultado de la ejecución del launch.

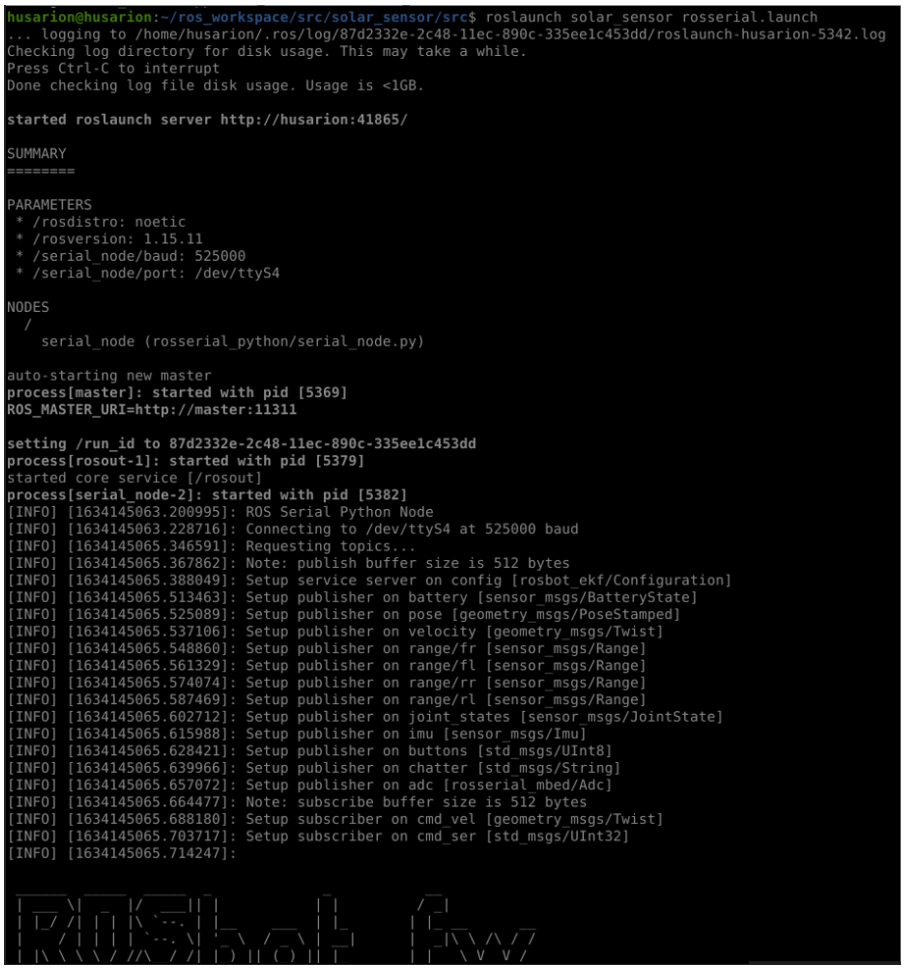

Figura 6-12. Ejecucuón de launch de inicialización del robot.

En la figura 6-13 se utiliza el comando “rostopic list -v” para mostrar la lista de publicadores y subscriptores.

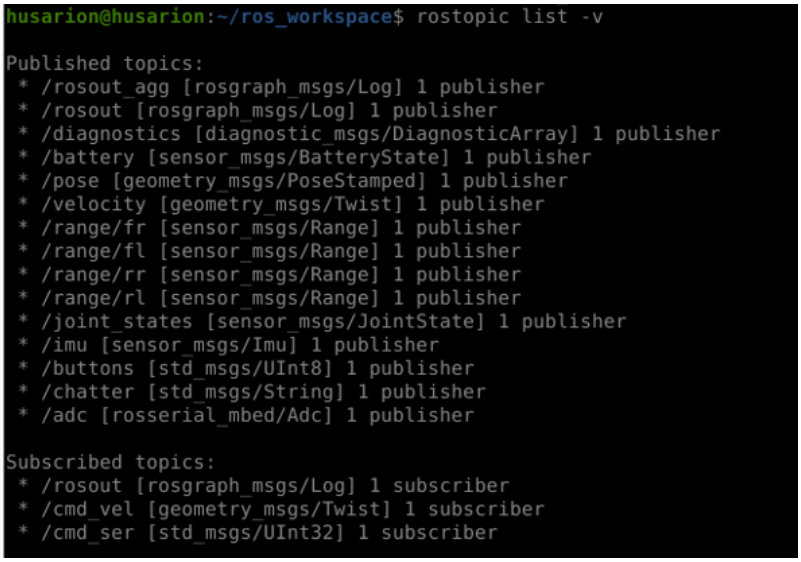

Figura 6-13. Publicadores y subscriptores. No inicializado subscriptor “configuración_sensor.cpp”.

7 RESULTADOS EXPERIMENTALES

Mediante el comando “rostopic echo /adc” observado en la figura 7-1, podemos ejecutar el tópico “/adc” en la terminal, para comprobar los datos de salida, e identificar los valores obtenidos en la placa CORE 2, los cuales como ya se mencionó en el apartado 5, se encuentran en bits.

```
husarion@husarion:~/ros_workspace$ source devel/setup.bash
husarion@husarion:~/ros_workspace$ rostopic echo /adc
```

Figura 7-1. Carga de setup.bash y utilización del comando rostopic echo.

Se puede ver en la figura 7-2, algunos de los resultados obtenidos con el sensor conectado, y variando la radiación con la luz artificial del teléfono móvil.

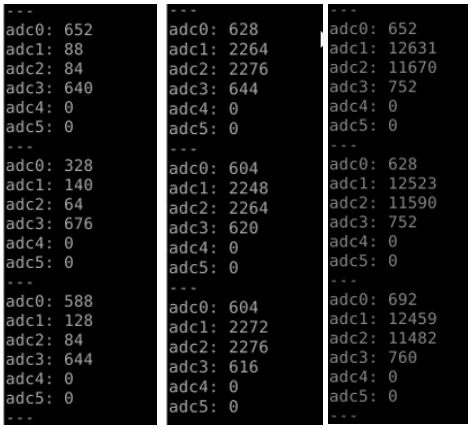

Figura 7-2. Valores en bits tomados en la placa CORE 2.

Se observa que la incidencia solar en los pines adc1 y adc2, es más sensible que en los pines adc 0 y adc3. En principio se entiende que, al estar fabricado el sensor para medir el ángulo de incidencia solar, los valores no son desorbitados.

Se inicializa el subscriptor, que realiza la transformación a grados, como se observa en el apartado 6.2, y se obtienen valores del sensor, al variar el ángulo de incidencia de la linterna, obteniendo los datos de la figura 7-3.

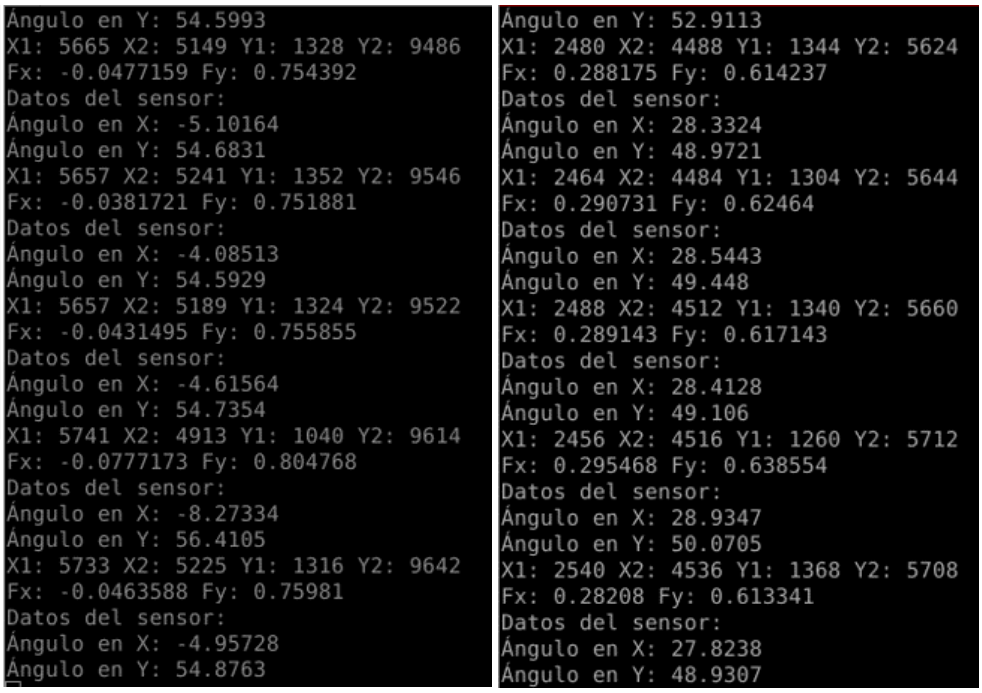

Figura 7-3. Resultados experimentales de ejecutar el nodo subscrito a los datos del sensor.

7.1 Resultados experimentales con luz artificial [35]

7.1.1 Resultados

En la figura 7-4, se puede observar los resultados de los ángulos de incidencia al realizar las pruebas con la linterna del teléfono móvil. Para acceder al video de la prueba realizada pulse **aquí** [35].

Se comienza con luz ambiente de la habitación donde se realizan las pruebas, posteriormente se aproxima la linterna y se ha variado el ángulo de incidencia en círculos alrededor de la ventana del sensor.

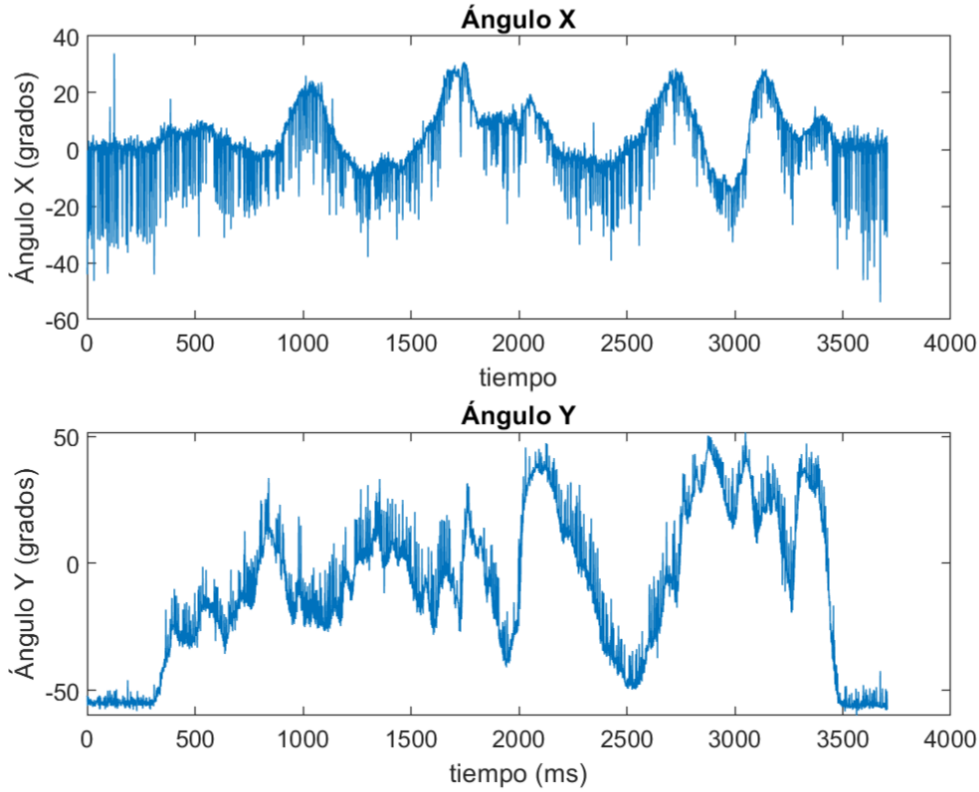

Figura 7-4. Ángulo de incidencia X e Y en función de la variación del teléfono móvil.

Se observa en la figura 7-5 la radiación obtenida en bits.

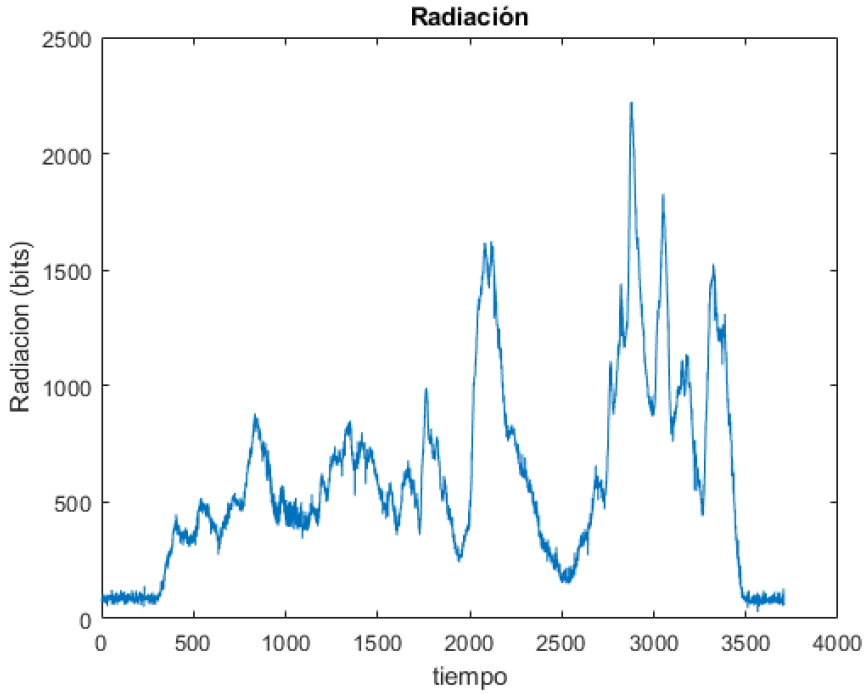

Figura 7-5. Variación de radiación en función de la luz proyectada.

7.1.2 Análisis de resultados obtenidos

Se analiza a continuación las distintas zonas de la gráfica de los resultados obtenidos.

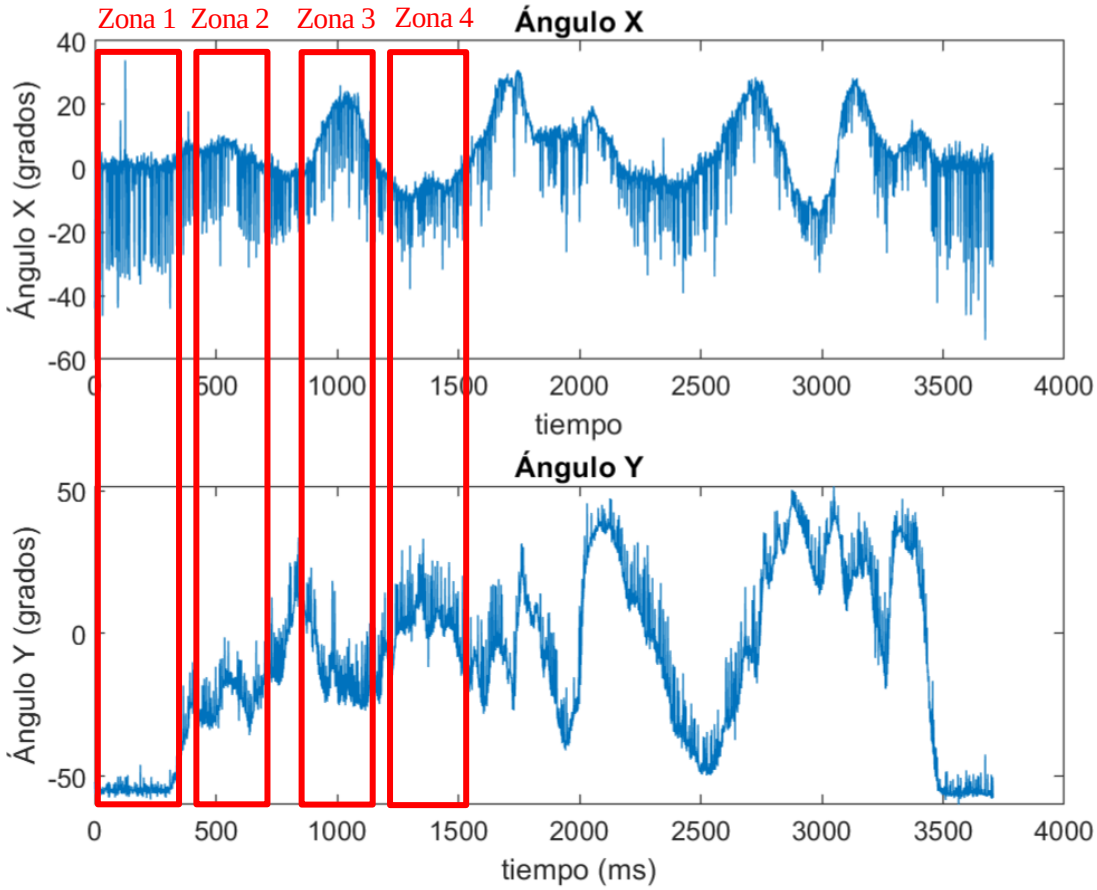

Figura 7-6. Ángulos de incidencia del sol.

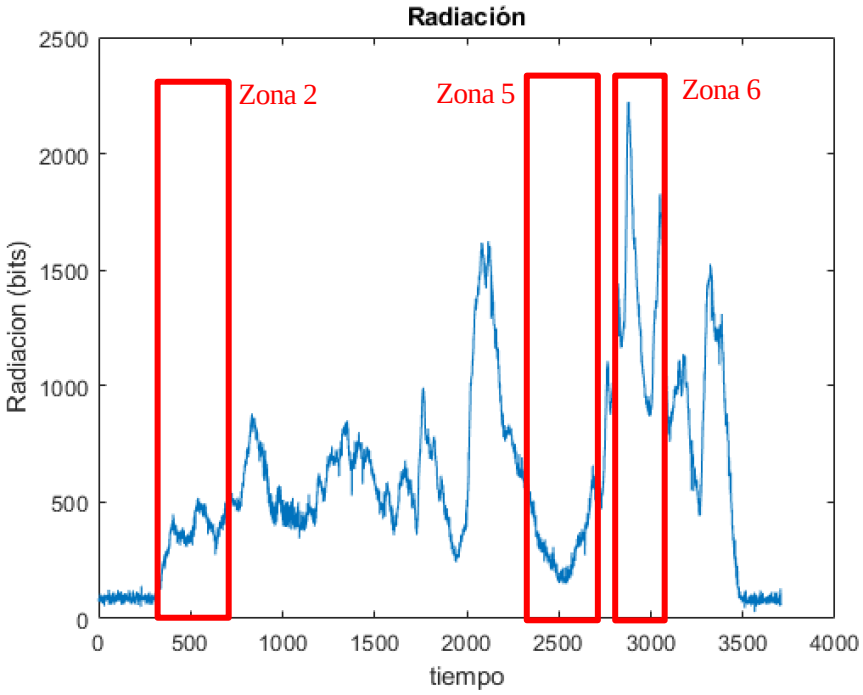

Figura 7-7. Radiación del sol.

En la figura 7-6 se observa la zona 1 y 2, en la cual se puede ver un incremento en la radiación al acercarse la luz de la linterna, y una variación del ángulo Y elevada en comparación con el ángulo X. Esto es debido a la incidencia de la luz en el extremo superior del sensor, y por tanto debe de estar recibiendo mayor inclinación en uno de los ángulos.

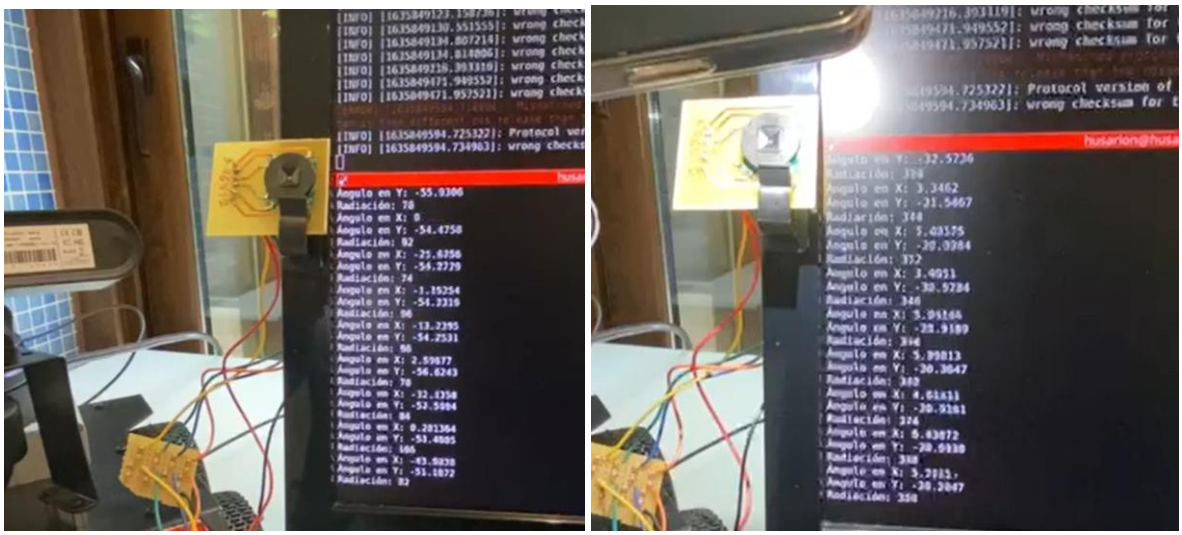

Figura 7-8. Zona 1 y zona 2

Sin embargo en la figura 7-9, se observa como la luz pasa de la zona 3 a 4, donde la incidencia de la luz está en el lado opuesto del caso anterior (zona 2), obteniendo resultados opuestos en los ángulos, pero un aumento de radiación.

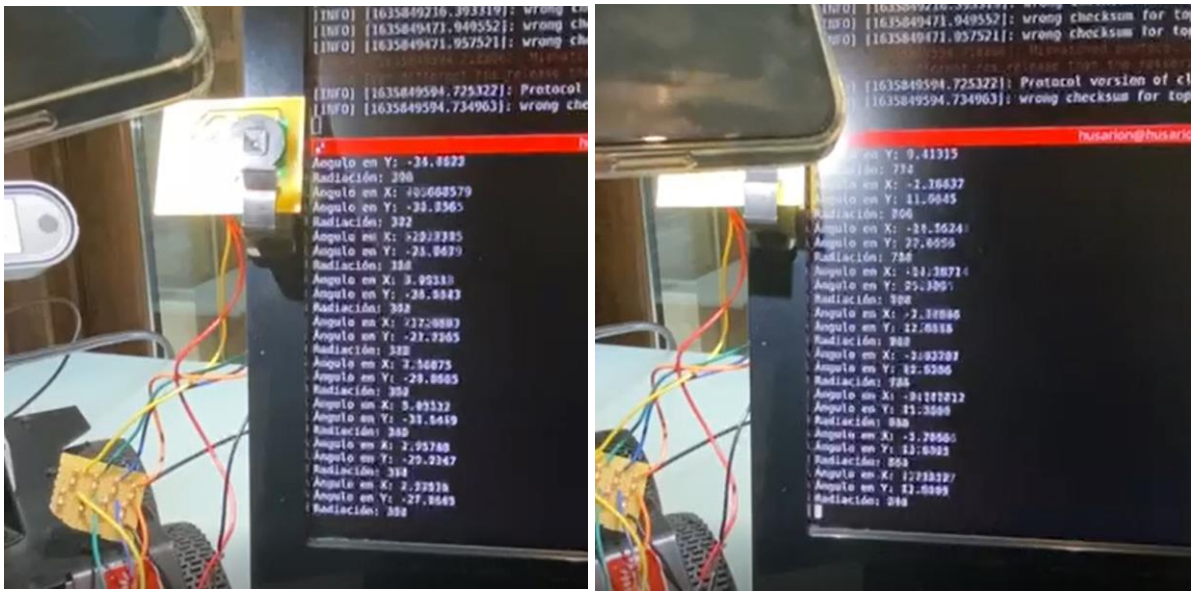

Figura 7-9. Zona 3 y 4

Finalmente, en la imagen 7-10 se aleja y aproxima la linterna, disminuyendo en gran medida la radiación y aumentando al pasar de la zona 5 a la 6.

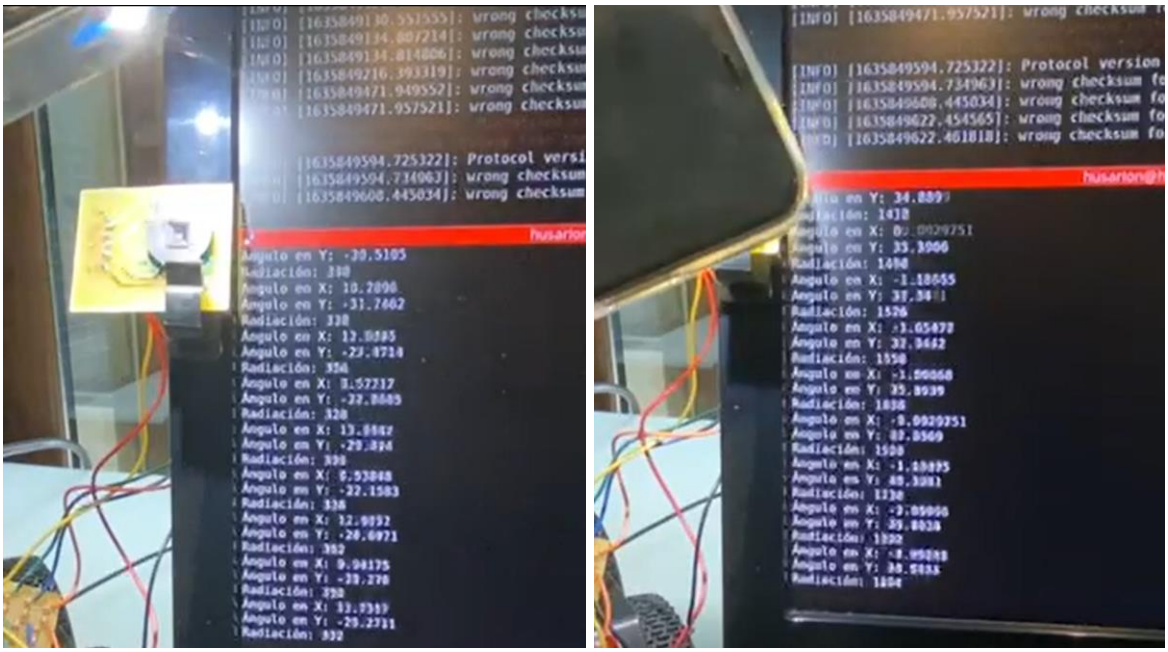

Figura 7-10. Zona 5 y 6

7.2 Resultados experimentales con luz solar

7.2.1 Resultados

En segundo lugar, se realizan pruebas experimentales de la actuación del sensor ante la luz solar y la sombra. Para acceder al video de la prueba experimental pulse **aquí** [36].

Se realizan distintas variaciones de la radiación incidiente, así como del ángulo de inclinación.

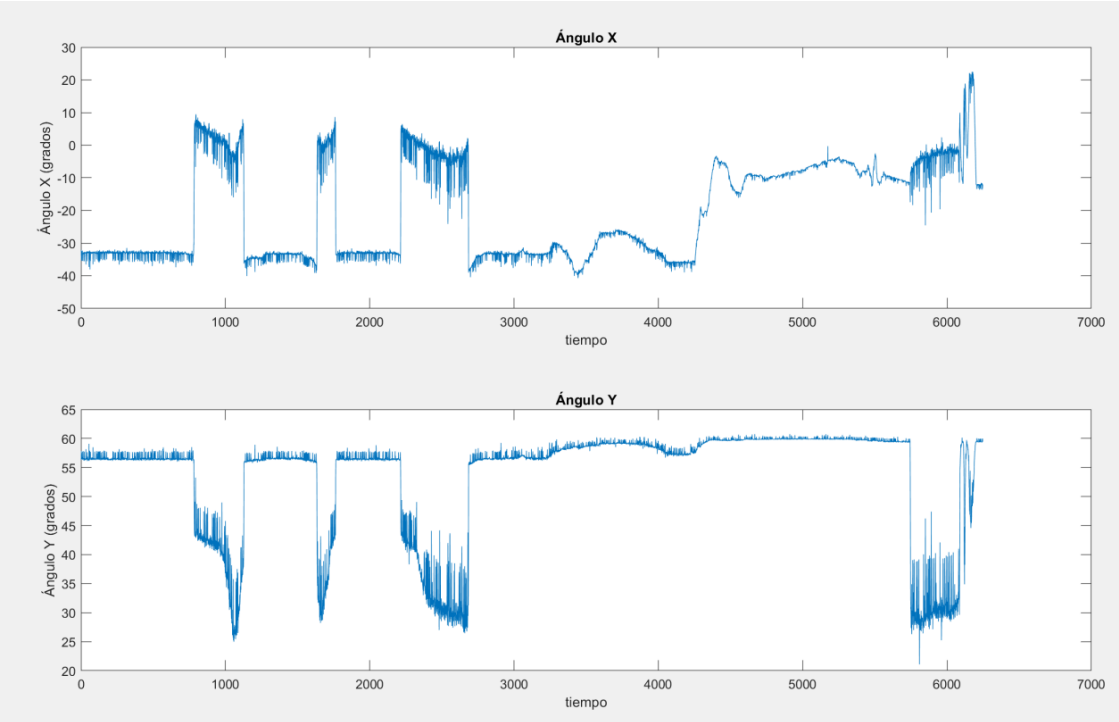

Figura 7-11. Ángulo de incidencia X e Y.

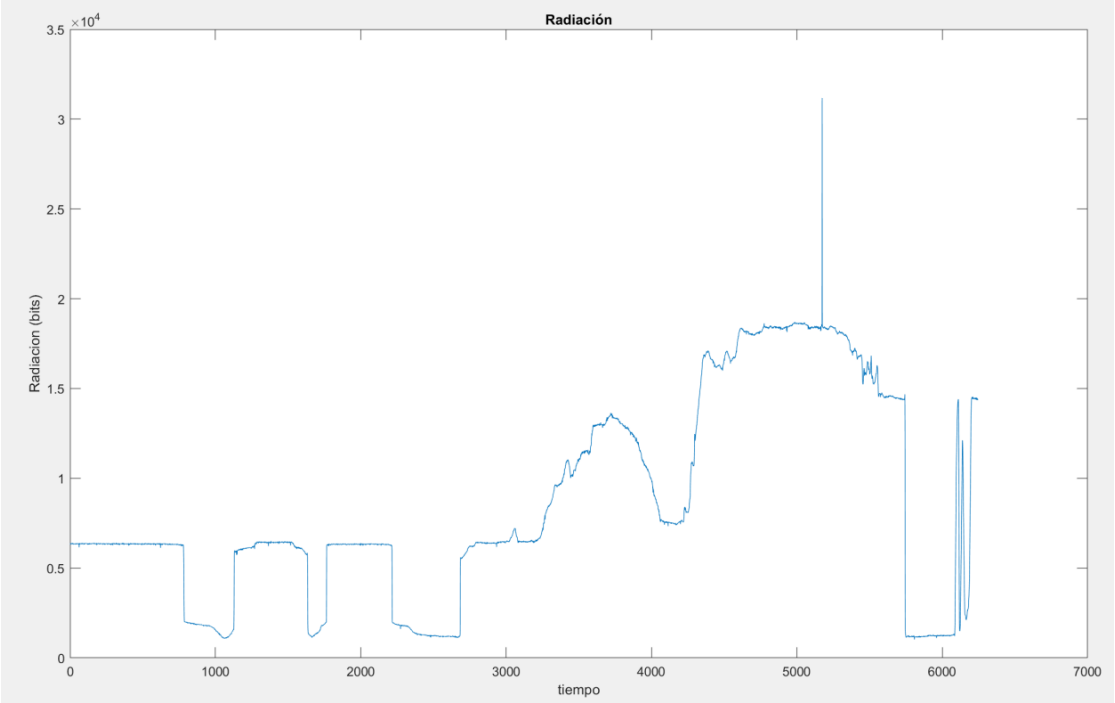

Figura 7-12. Radiación.

7.2.2 Análisis de resultados obtenidos

A continuación, se observa la gráfica dividida en distintas zonas en función de la incidencia solar.

- Zona 1 y 3: Radiación al sol con ángulo de inclinación de -34º en X y 56º en Y.
- Zona 2 y 4: Radiación a la sombra con ángulo de inclinación cuyos valores oscilan alrededor de 0 en en X y en Y decaen hasta 26 º.
- Zona 5: Disminución del ángulo de inclinación en X aumentando la radiación.
- Zona 6: Disminución del ángulo de inclinación en X hasta la dirección perpendicular con el sol, obteniendose la mayor radiación de la prueba experimental.

Dado la posición en la que se encuentra el sensor en la prueba realizada, se observa una mayor variación del ángulo X. Esto implica que la orientación de los ejes del sensor es la mostrada en la figura 7-19, en la que el ángulo de incidencia del sol se aproxima a un ángulo X de cero grados.

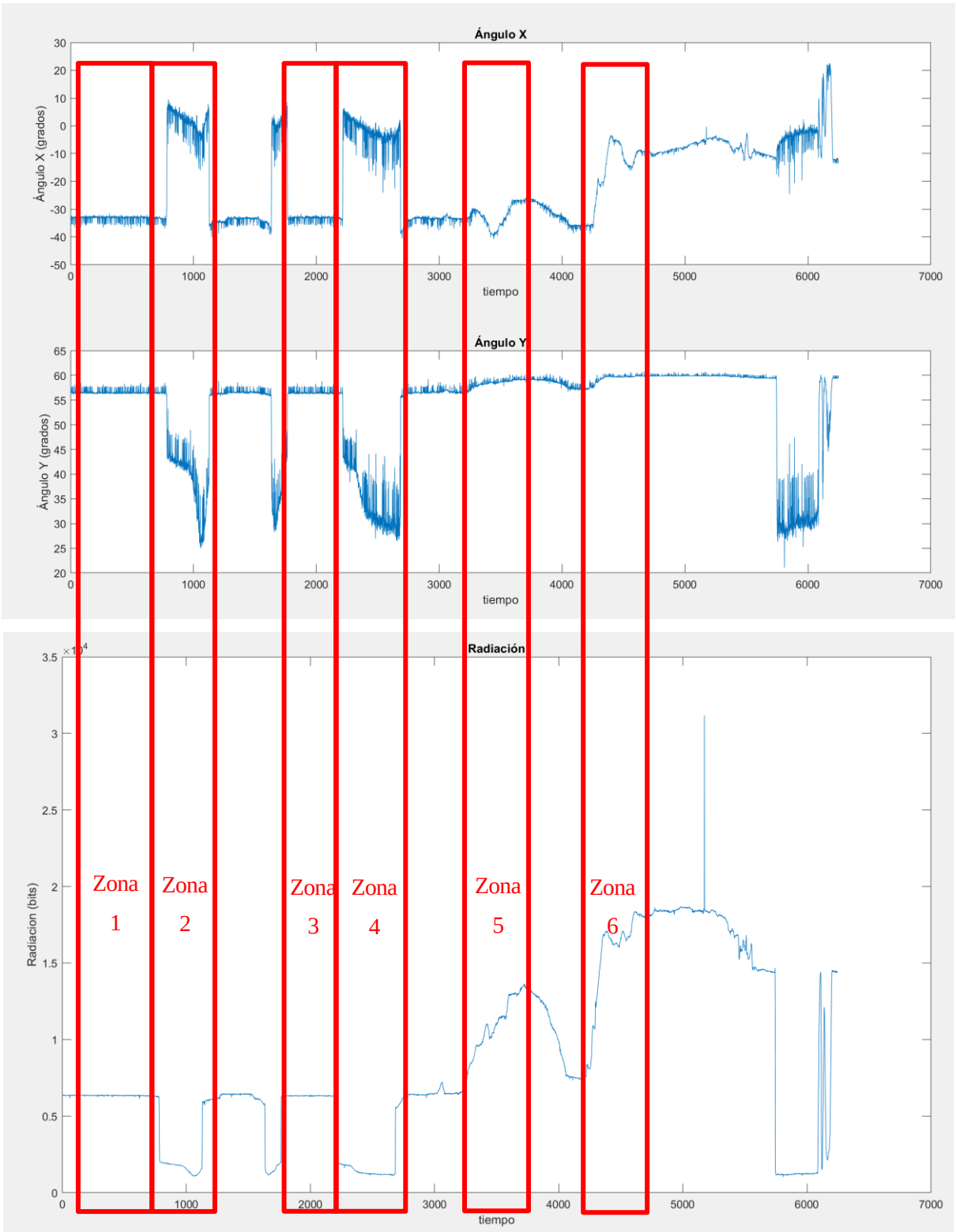

Figura 7-13. Ángulo de incidencia X e Y en función de la radiación solar.

Figura 7-14. Zona 1.

Figura 7-15. Zona 2.

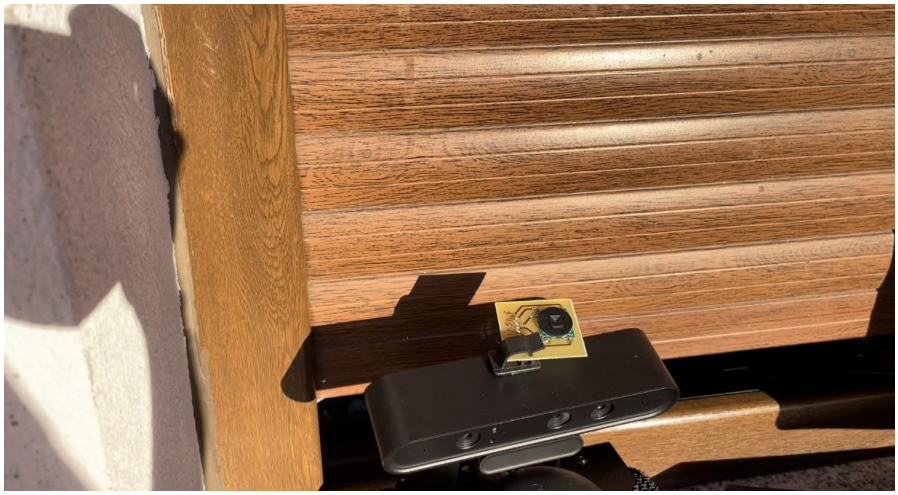

Figura 7-16. Zona 3.

Figura 7-17. Zona 4.

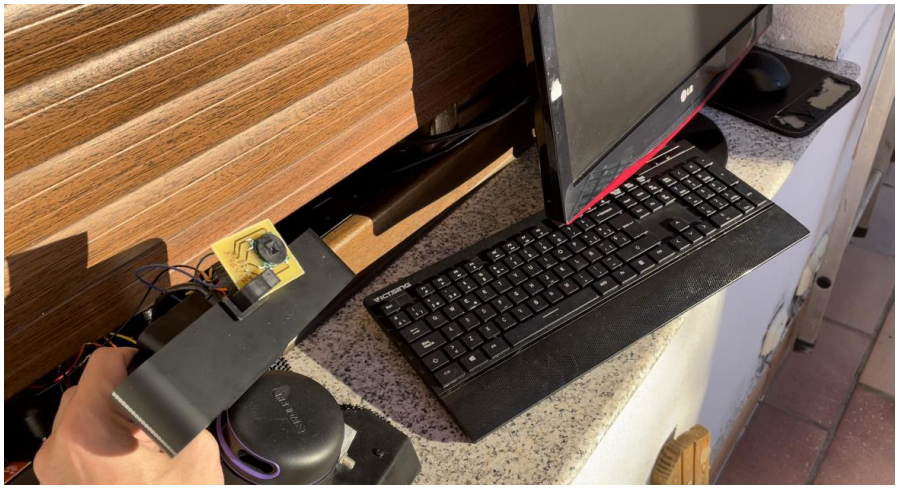

Figura 7-18. Zona 5.

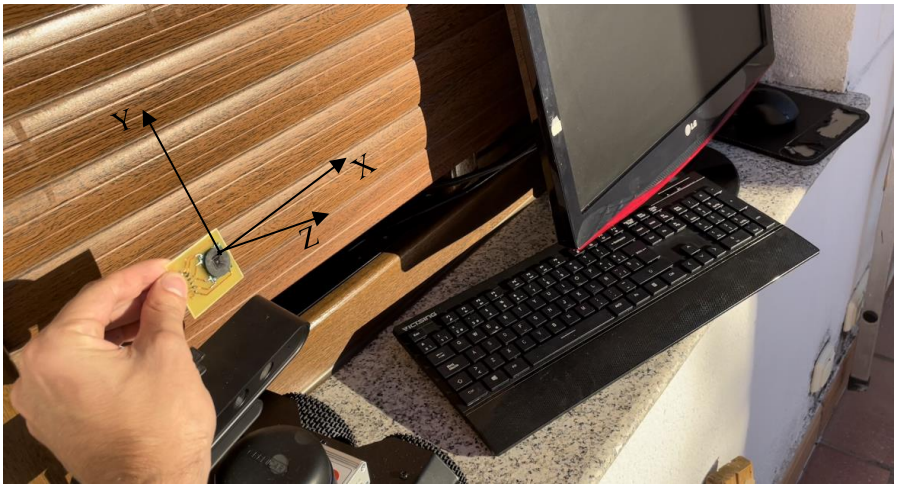

Figura 7-19. Zona 6.

8 CONCLUSIÓN

En este proyecto, se ha logrado instalar el sensor Solar MEMS ISS-AX al robot Rosbot 2.0 Pro para su uso en el proyecto de investigación OCONTSOLAR llevado a cabo en la escuela de Ingenierías Industriales de Sevilla, para tomar datos del ángulo de incidencia y de la radiación solar. Para su instalación y programación, se ha hecho uso de la plataforma ROS.

Se realizan pruebas experimentales, variando la incidencia y ángulo de inclinación y analizando los resultados gráficamente. Se comprueba que las mediciones tomadas coinciden con un incremento o decremento en la radiación además de una correcta medición del ángulo de incidencia. Por otro lado, se puede observar la inestabilidad del sensor, tomando valores que no se mantienen constantes.

Queda por verificar la medida del sensor de radiación, haciendo una conversión de las unidades a W/m^2 para obtener una medida de radiación real.

Como trabajo futuro queda la incorporación de un actuador Pan and Tilt. Este consiste en un sistema de sujeción con diferentes ángulos de giro a través de servos. La idea es incorporar el pan and tilt a Rosbot 2.0 Pro a una altura que supere la altura de la cámara, anclando el sensor a este dispositivo. Se incorporará al robot mediante los pines de salida, y se instalará y programará con el mismo proceso desarrollado en este proyecto.

Para la incorporación del Pan and Tilt, se hará uso de las ecuaciones de Spencer. Se ha encontrado un paquete donde se realiza la programación de un pan and tilt, el cual podría ser de utilidad para el trabajo mencionado [37].

El objetivo de esto es que el pan and tilt se mueva en función del ángulo de incidencia solar, intentando colocarse lo más perpendicular al sol posible, ya que es el punto donde la medida es más exacta.

Una ver realizado la incorporación del actuador mencionado, se juntará todo el proceso con el movimiento del robot, para poder desplazarlo de manera autónoma a los puntos necesarios.

REFERENCIAS

[1] J.G. Martín, J.R.D Frejo, R.A. García, E.F. Camacho, «Multi-robot task allocation problem with multiple nonlinear criteria using branch and bound and genetic algorithms,» *Intelligent Service Robotics*, pp. 1-21, 2021.

[2] J.R.D Frejo and E.F. Camacho, «Centralized and distributed Model Predictive Control for the maximization of the thermal power of solar paraolic-trough plants,» *Solar Energy 204*, pp. 190-199, 2020.

[3] J. M. D. González, «Robótica y Prótesis Inteligentes.,» *Revista digital Universitaria*, vol. 6, nº 1, 2004.

[4] A. O. Baturone, "Robótica Manipuladores y robots móviles", marcombo Boixareu Editores, 2001.

[5] «WIKILIBROS,» 2020. [En línea]. Available: https://es.wikibooks.org/wiki/Rob%C3%B3tica/Ventajas_y_desventajas_de_los_Robots.

[6] P. A. L. Ramírez, «Aprendizaje de y con robótica, algunas experiencias,» *Revista Educación*, nº 37(1), 43-63,ISSN:2215-2644, p. 21, 2013.

[7] MUNDO, «Avión malasio: lo buscarán submarinos no tripulados,» 2014.

[8] L. d. Barco, «Sony Aibo, el carismático perro robot de la compañía, llega a Europa,» *hipertextual*, 30 agosto 2018.

[9] Instituto Superior Politécnico José Antonio Echeverría, «Seguidor Solar, optimizando el aprovechamiento de la energía solar,» *Ingeniería Energética ISSN 1815-5901*, vol. XXXVI, nº 2, 2015.

[10] G. Arencibia-Carballo, «La importancia del uso de paneles solares en la generación de energía eléctrica,» *Redvet ISSN 1695-7504*, vol. 17, nº 9, 2016.

[11] «Wikipedia,» 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Pirheli%C3%B3metro.

[12] N. L. Forero, W. Meza, M.A. Martinez, L.M. Caicedo, G. Gordillo, «Estimación del Valor Medio Mensual del Índice de Claridad Atmosférico,» *Revista Colombiana de Física*, vol. Vol. 40 , nº Nº 1, 2008.

[13] E. P. Tutistar, «Modelo para calibrar piranómetro con la referencia mundial de radiación solar, utilizando un piranometro con banda de sombra,» *El Hombre y la Máquina* , Vols. %1 de %2ISSN 0121-0777, nº 45, 2014.

[14] «Wikipedia,» 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Piran%C3%B3metro.

[15] «UP Board Series,» [En línea]. Available: https://up-board.org/up/specifications/.

[16] Husarion, «Husarion DOCS,» [En línea]. Available: https://husarion.com/tutorials/ros-tutorials/1-ros-introduction.

[17] life.augmented, «pololu "VL53L0X",» [En línea]. Available: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/dillanes_l_a/capitulo1.pdf#:~:text=%E2%80%A2%2

0Robot%20es%20una%20m%C3%A1quina%20controlada%20por%20computadora,barata%20y%20precisa%20que%20los%20seres%20humanos%20%5B1%5D..

[18] InvenSense Inc, «invensense.tdk.com "MPU-9250",» [En línea]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf.

[19] BOSCH , «dl.btc.pl "BNO055",» [En línea]. Available: https://dl.btc.pl/kamami_wa/bst_bno055_ds000_12.pdf.

[20] SLAMTEC, «RPLIDAR A2,» [En línea]. Available: https://www.slamtec.com/en/Lidar/A2.

[21] Robotics, Open, «ROS,» 2021. [En línea]. Available: https://www.ros.org/.

[22] «Openwebinar,» [En línea]. Available: https://openwebinars.net/accounts/login/?next=/academia/aprende/programacion-ros/2657/.

[23] Solar Mems Technologies S.L., *"Sun Sensor ISS-AX",* Sevilla.

[24] GitHub, Inc, «husarion/rosbot-stm32-firmware,» 20 Octubre 2021. [En línea]. Available: https://github.com/husarion/rosbot-stm32-firmware.

[25] arm Limited, «arm MBED "mbedOs",» [En línea]. Available: https://os.mbed.com/mbed-os/.

[26] Microsoft, «VisualStudio,» [En línea]. Available: https://code.visualstudio.com/.

[27] Github, Inc, «github/stm32loader,» [En línea]. Available: https://github.com/husarion/stm32loader.

[28] Open Robotics, «Ros.org,» [En línea]. Available: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment.

[29] husarion , «Husarion DOCS,» [En línea]. Available: https://husarion.com/tutorials/.

[30] Open Robotics, «Ros.org,» [En línea]. Available: http://wiki.ros.org/ROS/Tutorials.

[31] GitHub, Inc, «husarion/rosbot_ekf,» [En línea]. Available: https://github.com/husarion/rosbot_ekf.

[32] Open Robotics, «Ros.org,» 2015. [En línea]. Available: http://wiki.ros.org/rosserial_python.

[33] J. Faust, «Docs.ros.org,» 2013. [En línea]. Available: http://docs.ros.org/en/diamondback/api/rostime/html/classros_1_1Rate.html .

[34] GitHub, Inc, «husarion/rosbot-stm32-firmware/rosserial-interface,» [En línea]. Available: https://github.com/husarion/rosbot-stm32-firmware#rosserial-interface.

[35] J. G. Díaz, «YouTube,» "Pruebas experimentales con luz artificial de sensor solar mems en rosbot 2.0 Pro", 2021. [En línea]. Available: https://www.youtube.com/watch?v=2FYffCTpQI0.

[36] J. G. Díaz, «YouTube,» "Video experimental Rosbot 2.0 Pro con sensor Solar Mems", 2021. [En línea]. Available: https://youtu.be/2NjqYCxBOSs.

[37] isaac879, «GitHub/Pan-Tilt-Mount,» [En línea]. Available: https://github.com/isaac879/Pan-Tilt-Mount.

ANEXO A ARCHIVOS PROGRAMADOS

A.1 Codigo main.cpp para el firmware del robot (partes editadas)

```
/** @file main.cpp
 * ROSbot firmware.
 *
 * @copyright GNU GPL-3.0
 */
#include <rosbot_kinematics.h>
#include <rosbot_sensors.h>
#include <ImuDriver.h>
#include <ros.h>
#include <sensor_msgs/JointState.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/PoseStamped.h>
#include <std_msgs/UInt32.h>
#include <sensor_msgs/Imu.h>
// #include <rosbot_ekf/Imu.h>

#if defined(ROS_NOETIC_MSGS) //TODO: add information to README about compiling for noetic/melodic (flags)
    #include <sensor_msgs_noetic/BatteryState.h>
#else
    #include <sensor_msgs/BatteryState.h>
#endif

#include <sensor_msgs/Range.h>
#include "tf/tf.h"
#include "tf/transform_broadcaster.h"
#include <std_msgs/UInt8.h>
#include <rosbot_ekf/Configuration.h>
#include <map>
#include <string>

#include "mbed.h"
#include "PeripheralNames.h"
#include "PinNames.h"

#include <ros.h>
#include <std_msgs/String.h>
#include <rosserial_mbed/Adc.h>

static const char EMPTY_STRING[] = "";

static const char WELLCOME_STR[] = "\n\n"
#if defined(WELLCOME_BANNER)
                                   " ______ _____  _____ _                 _              __\n"
                                   " | ___ \\|  _  |/  ___| |               | |            / _|\n"
                                   " | |_/ /| | | |\\ `--. | |__    ___   | |_       | |_ __      __\n"
                                   " |    / | | | | `--. \\| '_ \\  / _ \\ | __|      |  _|\\ \\ /\\ / /\n"
                                   " | |\\ \\ \\ \\_/ //\\__/ /| |_) || (_) || |_       | |   \\ V  V / \n"
                                   " \\_| \\_| \\___/ \\____/ |_.__/  \\___/  \\__|      |_|    \\_/\\_/ \n\n"
#endif
                                   " Firmware version: " ROSBOT_FW_VERSION
                                   "\n\n";
```

```
#if USE_WS2812B_ANIMATION_MANAGER
#include <AnimationManager.h>
AnimationManager *anim_manager;
static int parseColorStr(const char *color_str, Color_t *color_ptr)
{
    uint32_t num;
    if (sscanf(color_str, "%*c%X", &num) != 1)
        return 0;
    color_ptr->b = 0xff & num;
    color_ptr->g = 0xff & (num >> 8);
    color_ptr->r = 0xff & (num >> 16);
    return 1;
}
#endif

#define MAIN_LOOP_INTERVAL_MS 10
#define IMU_I2C_FREQUENCY 100000L
#define IMU_I2C_SCL SENS2_PIN3
#define IMU_I2C_SDA SENS2_PIN4

extern Mail<ImuDriver::ImuMesurement, 10> imu_sensor_mail_box;
const char *imu_sensor_type_string[] = {
    "BNO055_ADDR_A",
    "BNO055_ADDR_B",
    "MPU9250",
    "MPU9255",
    "UNKNOWN"};
char imu_description_string[64] = "";
ImuDriver *imu_driver_ptr;

std_msgs::String str_msg;
rosserial_mbed::Adc adc_msg;
```

```
geometry_msgs::Twist current_vel;
sensor_msgs::JointState joint_states;
sensor_msgs::BatteryState battery_state;
sensor_msgs::Range range_msg[4];
geometry_msgs::PoseStamped pose;
std_msgs::UInt8 button_msg;
// rosbot_ekf::Imu imu_msg;
sensor_msgs::Imu imu_msg;
ros::NodeHandle nh;
ros::Publisher *vel_pub;
ros::Publisher *joint_state_pub;
ros::Publisher *battery_pub;
ros::Publisher *range_pub[4];
ros::Publisher *pose_pub;
ros::Publisher *button_pub;
ros::Publisher *imu_pub;


ros::Publisher chatter("chatter", &str_msg);
ros::Publisher p("adc", &adc_msg);

geometry_msgs::TransformStamped robot_tf;
tf::TransformBroadcaster broadcaster;

rosbot_kinematics::RosbotOdometry odometry;

PinName adc0 = EXT_PIN1;
PinName adc1 = EXT_PIN2;
PinName adc2 = EXT_PIN3;
PinName adc3 = EXT_PIN4;
```

```
int averageAnalog(PinName pin)
{
    int v = 0;
    for (int i=0; i<4; i++)
    {
        v += AnalogIn(pin).read_u16();

    }
     return v/4;
}
long adc_timer;

volatile bool distance_sensors_enabled = false;
volatile bool joint_states_enabled = false;
volatile bool tf_msgs_enabled = false;

DigitalOut sens_power(SENS_POWER_ON, 0);

DigitalOut led2(LED2, 0);
DigitalOut led3(LED3, 0);
InterruptIn button1(BUTTON1);
InterruptIn button2(BUTTON2);
volatile bool button1_publish_flag = false;
volatile bool button2_publish_flag = false;

volatile bool is_speed_watchdog_enabled = true;
volatile bool is_speed_watchdog_active = false;
int speed_watchdog_interval = 1000; //ms

Timer odom_watchdog_timer;
volatile uint32_t last_speed_command_time = 0;

rosbot_sensors::ServoManger servo_manager;

rosbot_kinematics::RosbotKinematics *rk = nullptr;


.
.
.
.
.

int main()
{
    int spin_result;
    int err_msg = 0;
    uint32_t spin_count = 1;
    float curr_odom_calc_time, last_odom_calc_time = 0.0f;

    ThisThread::sleep_for(100);
    sens_power = 1; // sensors power on
    ThisThread::sleep_for(100);
    odom_watchdog_timer.start();

    rk = rosbot_kinematics::RosbotKinematics::kinematicsType(0);
    rk->setOdomParams();
    RosbotDrive &drive = RosbotDrive::getInstance();
    MultiDistanceSensor &distance_sensors = MultiDistanceSensor::getInstance();

    drive.setupMotorSequence(MOTOR_FR, MOTOR_FL, MOTOR_RR, MOTOR_RL);
    drive.init(rosbot_kinematics::custom_wheel_params, RosbotDrive::DEFAULT_REGULATOR_PARAMS);
    drive.enable(true);
    drive.enablePidReg(true);

    button1.mode(PullUp);
    button2.mode(PullUp);
    button1.fall(button1Callback);
    button2.fall(button2Callback);

    nh.initNode();
```

```
nh.initNode();

bool distance_sensors_init_flag = false;
bool imu_init_flag = false;
bool welcome_flag = true;

//TODO: add /diagnostic messages
int num_sens_init;
if ((num_sens_init = distance_sensors.init()) > 0)
{
    distance_sensors_init_flag = true;
}

I2C *i2c_ptr = new I2C(IMU_I2C_SDA, IMU_I2C_SCL);
i2c_ptr->frequency(IMU_I2C_FREQUENCY);

ImuDriver::Type type = ImuDriver::getType(i2c_ptr, 2);
sprintf(imu_description_string, "Detected sensor: %s\r\n", imu_sensor_type_string[type]);

if (type != ImuDriver::UNKNOWN)
{
    imu_driver_ptr = new ImuDriver(i2c_ptr, type);
    imu_driver_ptr->init();
    imu_driver_ptr->start();
    imu_init_flag = true;
}

ros::Subscriber<geometry_msgs::Twist> cmd_vel_sub("cmd_vel", &velocityCallback);
ros::Subscriber<std_msgs::UInt32> cmd_ser_sub("cmd_ser", &servoCallback);
ros::ServiceServer<rosbot_ekf::Configuration::Request, rosbot_ekf::Configuration::Response> config_srv("config", responseCallback);


nh.advertiseService(config_srv);
nh.subscribe(cmd_vel_sub);
nh.subscribe(cmd_ser_sub);

initBatteryPublisher();
initPosePublisher();
initVelocityPublisher();
initRangePublisher();
initJointStatePublisher();
initImuPublisher();
initButtonPublisher();


nh.advertise(chatter);
nh.advertise(p);
char hello[13] = "hello world!";

DigitalOut led = LED1;
```

```
#if USE_WS2812B_ANIMATION_MANAGER
    anim_manager = AnimationManager::getInstance();
    anim_manager->init();
#endif

#if defined(MEMORY_DEBUG_INFO)
    print_debug_info();
#endif /* MEMORY_DEBUG_INFO */

    if (imu_init_flag)
        imu_driver_ptr->start();

    if (distance_sensors_init_flag)
    {
        uint8_t *data = distance_sensor_commands.alloc();
        *data = 1;
        distance_sensor_commands.put(data);
        distance_sensors_enabled = true;
    }

    while (1)
    {
        //mensaje chatter
        led = !led;
        str_msg.data = hello;
        chatter.publish( &str_msg );


        adc_msg.adc0 = averageAnalog(adc0);
        adc_msg.adc1 = averageAnalog(adc1);
        adc_msg.adc2 = averageAnalog(adc2);
        adc_msg.adc3 = averageAnalog(adc3);

        p.publish(&adc_msg);

        nh.spinOnce();


       if (is_speed_watchdog_enabled)
       {
           if (!is_speed_watchdog_active && (odom_watchdog_timer.read_ms() - last_speed_command_time) > speed_watchdog_interval)
           {
               RosbotSpeed speed = {0.0, 0.0, 0.0};
               RosbotDrive &drive = RosbotDrive::getInstance();
               rk->setRosbotSpeed(drive, speed);
               is_speed_watchdog_active = true;
           }
       }

#if USE_WS2812B_ANIMATION_MANAGER
       if (!nh.connected())
           anim_manager->enableInterface(false);
#endif

       if (spin_count % 2 == 0)
       {
           curr_odom_calc_time = odom_watchdog_timer.read();
           RosbotDrive &drive = RosbotDrive::getInstance();
           rk->updateRosbotOdometry(drive, odometry, curr_odom_calc_time - last_odom_calc_time);
           last_odom_calc_time = curr_odom_calc_time;
       }

       if (button1_publish_flag)
       {
           button1_publish_flag = false;
           if (!button1)
           {
               button_msg.data = 1;
               if (nh.connected())
                   button_pub->publish(&button_msg);
           }
       }
```

```
if (button2_publish_flag)
{
    button2_publish_flag = false;
    if (!button2)
    {
        button_msg.data = 2;
        if (nh.connected())
            button_pub->publish(&button_msg);
    }
}

if (spin_count % 5 == 0) /// cmd_vel, odometry, joint_states, tf messages
{
    current_vel.linear.x = sqrt(odometry.robot_x_vel * odometry.robot_x_vel + odometry.robot_y_vel * odometry.robot_y_vel);
    current_vel.angular.z = odometry.robot_angular_vel;
    current_vel = rk->getTwist(odometry);
    pose.pose.position.x = odometry.robot_x_pos;
    pose.pose.position.y = odometry.robot_y_pos;
    pose.pose.orientation = tf::createQuaternionFromYaw(odometry.robot_angular_pos);

     pose.header.stamp = nh.now();
     if (nh.connected())
     {
         pose_pub->publish(&pose);
         vel_pub->publish(&current_vel);
     }

     if (joint_states_enabled)
     {
         pos[0] = odometry.wheel_FL_ang_pos;
         pos[1] = odometry.wheel_FR_ang_pos;
         pos[2] = odometry.wheel_RL_ang_pos;
         pos[3] = odometry.wheel_RR_ang_pos;
         joint_states.position = pos;
         joint_states.header.stamp = pose.header.stamp;
         if (nh.connected())
             joint_state_pub->publish(&joint_states);
     }

     if (tf_msgs_enabled)
     {
         robot_tf.header.stamp = pose.header.stamp;
         robot_tf.transform.translation.x = pose.pose.position.x;
         robot_tf.transform.translation.y = pose.pose.position.y;
         robot_tf.transform.rotation.x = pose.pose.orientation.x;
         robot_tf.transform.rotation.y = pose.pose.orientation.y;
         robot_tf.transform.rotation.z = pose.pose.orientation.z;
         robot_tf.transform.rotation.w = pose.pose.orientation.w;
         if (nh.connected())
             broadcaster.sendTransform(robot_tf);
     }
 }
```

```
if (spin_count % 40 == 0)
{
    battery_state.voltage = rosbot_sensors::updateBatteryWatchdog();
    if (nh.connected())
        battery_pub->publish(&battery_state);
}

osEvent evt1 = distance_sensor_mail_box.get(0);
if (evt1.status == osEventMail)
{
    SensorsMeasurement *message = (SensorsMeasurement *)evt1.value.p;
    if (message->status == MultiDistanceSensor::ERR_I2C_FAILURE)
    {
        err_msg++;
        if (distance_sensor_commands.empty() && err_msg == 3)
        {
            if (nh.connected())
                nh.logerror("I2C error. Restarting VL53L0X sensors...");
            uint8_t *data = distance_sensor_commands.alloc();
            *data = 2;
            distance_sensor_commands.put(data);
            data = distance_sensor_commands.alloc();
            *data = 1;
            distance_sensor_commands.put(data);
            err_msg = 0;
        }
    }
    else
    {
        err_msg = 0;
        for (int i = 0; i < 4; i++)
        {
            range_msg[i].header.stamp = nh.now(message->timestamp);
            range_msg[i].range = message->range[i];
            if (nh.connected())
                range_pub[i]->publish(&range_msg[i]);
        }
    }
    distance_sensor_mail_box.free(message);
}
```

```
        }
        // }
        // if(spin_count % 20 == 0 && distance_sensors_enabled) // ~ 5 HZ
        // {
        //     uint16_t range;
        //     ros::Time t = nh.now();
        //     for(int i=0;i<4;i++)
        //     {
        //         range = distance_sensors->getSensor(i)->readRangeContinuousMillimeters(false);
        //         range_msg[i].header.stamp = t;
        //         range_msg[i].range = (range != 65535) ? (float)range/1000.0f : -1.0f;
        //         if(nh.connected()) range_pub[i]->publish(&range_msg[i]);
        //     }
        // }

        osEvent evt2 = imu_sensor_mail_box.get(0);

        if (evt2.status == osEventMail)
        {
            ImuDriver::ImuMesurement *message = (ImuDriver::ImuMesurement *)evt2.value.p;

            if(nh.connected())
            {
                int i = 0;
                imu_msg.header.stamp = nh.now(message->timestamp);
                imu_msg.orientation.x = message->orientation[i];
                imu_msg.angular_velocity.x = message->angular_velocity[i];
                imu_msg.linear_acceleration.x = message->angular_velocity[i++];
                imu_msg.orientation.y = message->orientation[i];
                imu_msg.angular_velocity.y = message->angular_velocity[i];
                imu_msg.linear_acceleration.y = message->angular_velocity[i++];
                imu_msg.orientation.z = message->orientation[i];
                imu_msg.angular_velocity.z = message->angular_velocity[i];
                imu_msg.linear_acceleration.z = message->angular_velocity[i++];
                imu_msg.orientation.w = message->orientation[i];
                imu_pub->publish(&imu_msg);
            }

            imu_sensor_mail_box.free(message);
        }


        // LOGS
        if (nh.connected())
        {
            if (welcome_flag)
            {
                welcome_flag = false;
                nh.loginfo(WELLCOME_STR);
                if (!distance_sensors_init_flag)
                    nh.logerror("VL53L0X sensors initialisation failure!");
                if (!imu_init_flag)
                    nh.logerror("No IMU sensor detected!");
                else
                    nh.loginfo(imu_description_string);
            }
        }
        else
        {
            welcome_flag = true;
        }

        nh.spinOnce();

        // if((spin_result=nh.spinOnce()) != ros::SPIN_OK)
        // {
        //     // nh.logwarn(spin_result == -1 ? "SPIN_ERR" : "SPIN_TIMEOUT");
        //     do {}while(0); // do nothing at the moment
        // }
        spin_count++;
        ThisThread::sleep_for(MAIN_LOOP_INTERVAL_MS);
    }
}
```

A.2 Codigo Solar_Sensor_Node.cpp

```
#include <ros/ros.h>
#include <rosserial_mbed/Adc.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#define C 1.871f

struct informacion_sensor
{
   float AnguloX;
   float AnguloY;
   float radiacion;
};

void delay(int secs) {
  for(int i = (time(NULL) + secs); time(NULL) != i; time(NULL));
}

void sensorCallback(const rosserial_mbed::Adc::ConstPtr& msg)
{

   informacion_sensor informacion;
   int secs = 1;

   float VPH1 = msg->adc0;
   float VPH2 = msg->adc1;
   float VPH3 = msg->adc2;
   float VPH4 = msg->adc3;

   float X1, X2, Y1, Y2;

   X1 = VPH3 + VPH4;
   X2 = VPH1 + VPH2;
   Y1 = VPH1 + VPH4;
   Y2 = VPH2 + VPH3;
```

```
    //std::cout << "X1: " << X1 << " X2: "<< X2  <<" Y1: "<< Y1 << " Y2: "<< Y2 << std::endl;

    float Fx = (X2-X1)/(X2+X1);
    float Fy = (Y2-Y1)/(Y2+Y1);
    //std::cout << "Fx: " << Fx << " Fy: "<< Fy << std::endl;


    informacion.AnguloX = atan(C*Fx)*57.29578;
    informacion.AnguloY = atan(C*Fy)*57.29578;

    std::cout << "Ángulo en X: " << informacion.AnguloX << std::endl;
    std::cout << "Ángulo en Y: " << informacion.AnguloY << std::endl;

    informacion.radiacion = (VPH2 + VPH3)/2;
    std::cout << "Radiación: " << informacion.radiacion << std::endl;
    //delay(secs);

}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "sensor_subscriber");
    ros::NodeHandle nh;
    std::cout << "Recibiendo datos del sensor:" << std::endl;
    //ros::Rate loop_rate(1);
    ros::Subscriber sub = nh.subscribe("adc", 1000, sensorCallback);
    ros::spin();

    return 0;
}
```

A.3 File de mensaje Adc.msg

```
uint16 adc0
uint16 adc1
uint16 adc2
uint16 adc3
uint16 adc4
uint16 adc5
```

A.4 File de rosserial.launch creado para rosbot 2.0 Pro en paquete solar_sensor

```
<launch>
  <arg name="serial_port" default="/dev/ttyS4"/>
  <arg name="serial_baudrate" default="525000"/>
  <node pkg="rosserial_python" type="serial_node.py" name="serial_node" output="screen">
    <param name="port" value="$(arg serial_port)"/>
    <param name="baud" value="$(arg serial_baudrate)"/>
  </node>
</launch>
```

A.5 File CMakeLists.txt creado en paquete solar_sensor

```
cmake_minimum_required(VERSION 3.0.2)
project(solar_sensor)
add_compile_options(-std=c++11)
find_package(catkin REQUIRED COMPONENTS
  roscpp
  message_generation
  rosserial_mbed
)
################################################
## Declare ROS messages, services and actions ##
################################################

## Generate added messages and services with any dependencies listed here
 generate_messages(
   DEPENDENCIES
   std_msgs  # Or other packages containing msgs
   rosserial_mbed
 )
```

```
#####################################
## catkin specific configuration ##
#####################################
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES solar_sensor
  CATKIN_DEPENDS roscpp
#  DEPENDS system_lib
)
###########
## Build ##
###########
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)
 add_executable(${PROJECT_NAME}_node src/configuracion_sensor.cpp)
## Specify libraries to link a library or executable target against
 target_link_libraries(${PROJECT_NAME}_node
   ${catkin_LIBRARIES}
 )
```

A.6 File package.xml de paquete solar_sensor

```
<?xml version="1.0"?>
<package format="2">
  <name>solar_sensor</name>
  <version>0.0.0</version>
  <description>The solar_sensor package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="husarion@todo.todo">husarion</maintainer>



  <!-- One license tag required, multiple allowed, one license per tag -->
```

```
<!-- Commonly used license strings: -->
<!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
<license>TODO</license>


<!-- Url tags are optional, but multiple are allowed, one per tag -->
<!-- Optional attribute type can be: website, bugtracker, or repository -->
<!-- Example: -->
<!-- <url type="website">http://wiki.ros.org/solar_sensor</url> -->


<!-- Author tags are optional, multiple are allowed, one per tag -->
<!-- Authors do not have to be maintainers, but could be -->
<!-- Example: -->
<!-- <author email="jane.doe@example.com">Jane Doe</author> -->


<!-- The *depend tags are used to specify dependencies -->
<!-- Dependencies can be catkin packages or system dependencies -->
<!-- Examples: -->
<!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
<!--   <depend>roscpp</depend> -->
<!--   Note that this is equivalent to the following: -->
<!--   <build_depend>roscpp</build_depend> -->
<!--   <exec_depend>roscpp</exec_depend> -->
<!-- Use build_depend for packages you need at compile time: -->
<!--   <build_depend>message_generation</build_depend> -->
<!-- Use build_export_depend for packages you need in order to build against this package: -->
<!--   <build_export_depend>message_generation</build_export_depend> -->
<!-- Use buildtool_depend for build tool packages: -->
<!--   <buildtool_depend>catkin</buildtool_depend> -->
<!-- Use exec_depend for packages you need at runtime: -->
<!--   <exec_depend>message_runtime</exec_depend> -->
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for building documentation: -->
<!--   <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
```

```
  <build_depend>rosserial_mbed</build_depend>

  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rosserial_mbed</build_export_depend>

  <exec_depend>roscpp</exec_depend>

  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

ANEXO B DATASHEET SENSOR ISS-AX

Solar MEMS Technologies S.L.

Sun Sensor ISS-AX

Analog sensor

Technical Specifications

Features

Two orthogonal axes sun sensor
Wide or narrow field of view
High accuracy
4 analog outputs
Low power consumption
Wide operating voltage range: 5÷12 V
Industrial temperature range: - 40° to 85°
Reduced size
Low weight
IP65 protection
Reverse polarity protection

Applications

Sun tracking/pointing systems
Solar Trackers
Heliostats
Photovoltaic
CSP, CPV and HCPV
Stirling

ISS-AX sun sensor measures the incident angle of a sun ray in both orthogonal axes. The high sensitivity reached is based on the geometrical dimensions of the design.

Its characteristics make it a suitable tool for high accurate sun-tracking and positioning systems, with low power consumption and high reliability.

ISS-AX sun sensor has been designed with a unique and novel own technology based on MEMS fabrication processes to achieve high integrated sensing structures at low cost.

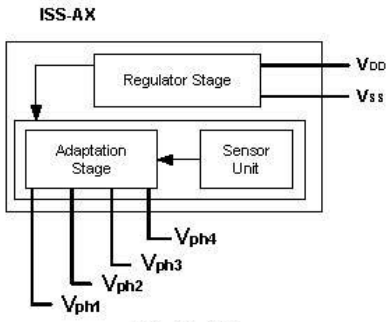

Fig 1. Block Diagram

Contents

Figures

Tables

1. General Specifications

Parameter	ISS-A60	ISS-A25	ISS-A15	ISS-A5	Unit	Comments
Sensor type	2 axes	2 axes	2 axes	2 axes	-	Orthogonal
Field of view (FOV)	120	50	30	10	º	Aperture of the cone of view
Accuracy (*)	< 10	< 10	< 10	< 10	%	3σ
Precision (*)	< 0,06	< 0,04	< 0,02	< 0.01	º	Sensitivity
Average consumption	9	9	9	9	mA	
Dimensions						
Diameter	80	80	80	80	mm	
Height	27	27	27	27	mm	
Weight	100	100	100	100	g	
Level of protection	IP65	IP65	IP65	IP65		CEI 60529 Standard
Expected life time of 10 years + Pressure test at 0,05 mbar and 25ºC						

(*) Accuracy test in Laboratory: cable of 2 meters, collimated light source, radiation of 900 W/m², CAD resolution of 10 bits, and filter stage with sampling frequency of 50 Hz and bandwidth of 0,4 Hz.

Table 1. General Specifications

Different models of the ISS-AX are offered, differing in the field of view (FOV) of the sensor. The accuracy of the sensor is inversely proportional to the field of view. All these models have been tested on solar trackers with Solar MEMS Helios Controller.

2. Electrical characteristics

Symbol	Parameter	Min	Typical	Max	Unit
V_{DD}	Supply voltage	5	5	12	V
I_{DD}	Feed current	-	9	-	mA
Vph	Photodiode voltages (analog outputs)	0	-	4,5	V
Recommended					
V_{DD}	Supply voltage	5	-	12	V
V_r	Supply voltage ripple	0	-	100	mVpp
T_{OP}	Operating temperature	-40	-	85	ºC
Absolute maximum					
V_{DD}	Supply voltage	0	-	16	V
T_{OP}	Operating temperature	-40	-	85	ºC

Table 2. Electrical characteristics

Reverse polarity protection.

3. Sun Sensor ISS-AX

ISS-AX sensor measures the incidence angles of a solar radiation respect to its perpendicular. This information is provided through 4 analog outputs.

3.1. Description

ISS-AX measures the incidence angle of a sun ray in both axes, based on a quadrant photodetector device. The sunlight is guided to the detector through a window above the sensor. Dependent of the angle of incidence, the sunlight induces photocurrents in the four quadrants of the detector.

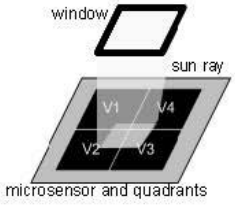

Fig 2. Microsensor of ISS-AX

3.2. Reference Axes

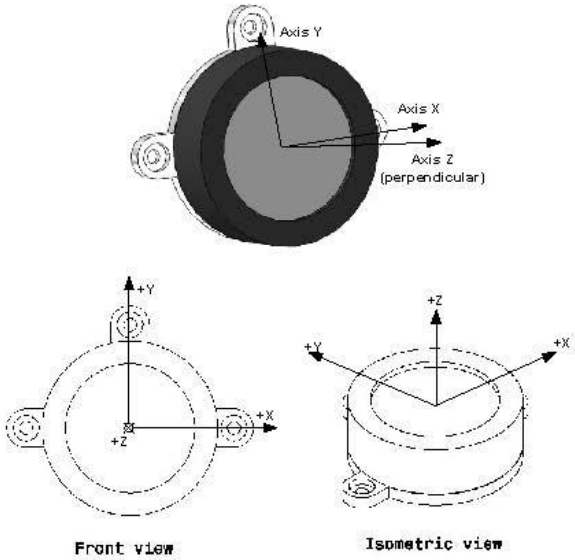

Fig 3. ISS-AX reference system

Z axis is perpendicular to the sensor base plane.

3.3. Measurements

The *Angle X* and *Angle Y* specify the angular position of the incident sun ray inside the field of view of the ISS-AX sensor.

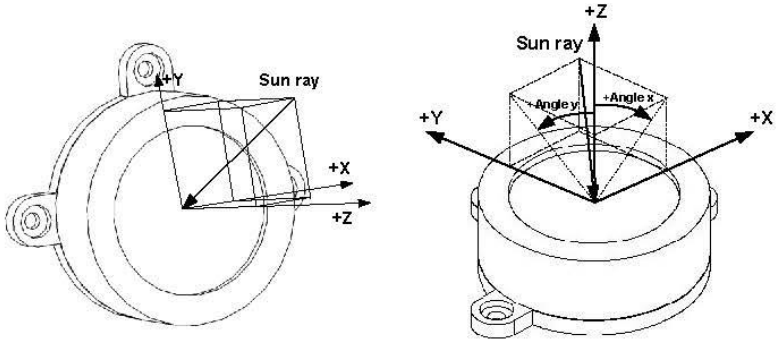

Fig 4. References for measured angles

Angle X and *Angle Y* of the incident ray can be obtained with a simple **set of equations** involving the four photodiode voltages generated by the sensor (V_{PH1}, V_{PH2}, V_{PH3}, *and* V_{PH4}):

$$X_1 = V_{PH3} + V_{PH4} \qquad Y_1 = V_{PH1} + V_{PH4}$$

$$X_2 = V_{PH1} + V_{PH2} \qquad Y_2 = V_{PH2} + V_{PH3}$$

$$F_X = \frac{X_2 - X_1}{X_2 + X_1} \qquad F_Y = \frac{Y_2 - Y_1}{Y_2 + Y_1}$$

$$Angle\ X = arctg\left(C \cdot F_X\right) \qquad Angle\ Y = arctg\left(C \cdot F_Y\right)$$

Type	Value
ISS-A60	1,871
ISS-A25	0,477
ISS-A15	0,324
ISS-A5	0,130

Table 3. Values of the parameter C according to the type of sensor ISS-AX (Geometric Correction)

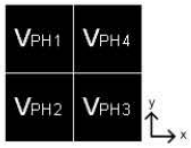

Fig 5. References for the photodiodes

The accuracy of the sensor increases when receiving a radiation perpendicular to the sensor, close to zero degrees in X and Y. This is an outstanding feature that makes it suitable for tracking applications. The **accuracy** can be **increased** in more than one order of magnitude by compensating the offset error after the installation of the sensor by means of **Calibration or Alignment**.

The use of a **filtering stage is recommended** (for example: 50 Hz sampling frequency and 0,4 Hz bandwidth).

4. Calibration or Alignment

Calibration process increases the accuracy of the ISS-AX sun sensor. The zero degrees position is the pair of angles X/Y that it is measured on the installation according to, for example, the maximum power generated of a CPV panel.

Example of calibration process: Solar Tracker:

1. Install the Sun Sensor ISS-AX on the tracker.
   The sensor must be installed according to the solar panels, in the same plane, as well as possible. This will improve the use of the sun sensor field of view.

2. Control the solar tracking to get the maximum power generated.
   Get the Angle X and Angle Y of the sun sensor ISS-AX. This pair will be your zero degrees position for the maximum power generation.

3. Control the solar tracker in closed-loop using your new zero degrees position of the ISS-AX sun sensor as reference: rectify the angles measured by the sensor according to this zero degrees position.

5. Recommendations

Depending on the application of the Sun Sensor ISS-AX, we recommend the use of the following models:

- Solar Tracker with Photovoltaic:
  The accuracy requirements are not demanding, so it's recommended to **use the ISS-A60** model, **to get a wide field of view**.
- Solar Tracker with CPV or similar:
  The accuracy requirements are very demanding, so it's recommended to **use the ISS-A5** model, **to get high accuracy and narrow field of view**, because a wide field of view increases the effects of the **environmental conditions** on the accuracy of the sun sensor: clouds effect and seeing of the ground.
- Other applications:
  It depends on the demanding of the field of view and the accuracy.

6. Electrical interface

Colour	Terminal	Type	Comments
Red	V_{DD}	Power	Power Supply
Blue	V_{SS}	Power	Ground
Yellow	Vph1	Analog output	Photodiode 1: upper-left
Green	Vph2	Analog output	Photodiode 2: lower-left
Brown	Vph3	Analog output	Photodiode 3: lower-right
White	Vph4	Analog output	Photodiode 4: upper-right
Grey	Rtn	Analog reference	Signal return
Pink	-	-	Not connected
Shield	-	-	Connect to the blue wire

Table 4. Electrical interface

The housing of the sun sensor ISS-AX is isolated electrically.

7. Mechanical data

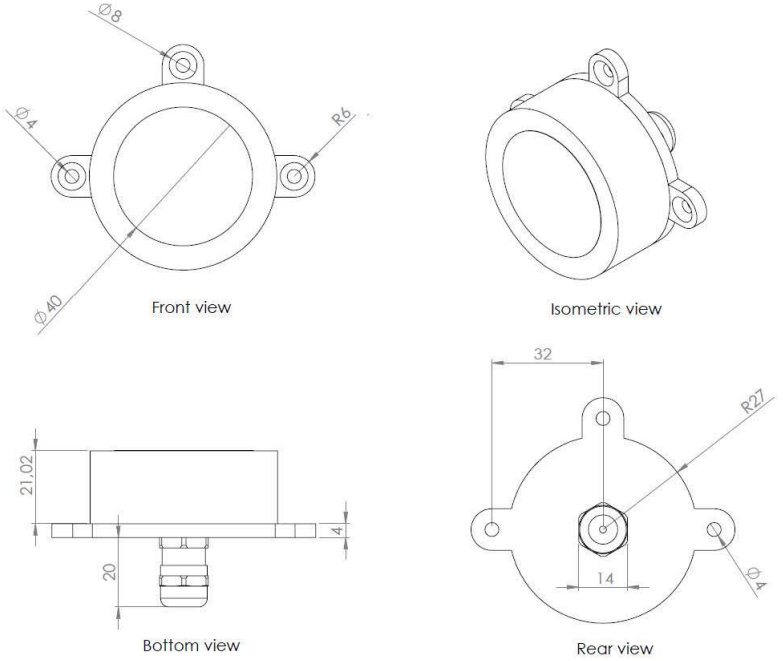

Fig 6. ISS-AX dimensions

The box of the ISS-AX sensor is composed of a top and bottom housing, both made of Aluminum 6082: it has good corrosion resistance. The top housing has a protective coating of anodizing and it is black lacquered, and the bottom housing has a protective coating of matt anodizing.

8. Warranty

Solar MEMS Technologies S.L. warrants the ISS-AX sun sensor to the original consumer purchaser any product that is determined to be defective for the following terms will be repaired, or replaced.

The warranty is one year from date of purchase.

The product in question must be sent to Solar MEMS Technologies S.L. (address is shown below) within the warranty period and the original consumer purchaser must comply with the following conditions, to be eligible for repair or replacement under this warranty:

- The product must not have been modified or altered in any way by an unauthorized source.
- The product must have been installed in accordance with the installation instructions and the technical specifications.

This limited warranty does not cover:

- Damage due to improper installation;
- Accidental or intentional damages;
- Misuse, abuse, corrosion, or neglect;
- Product impaired by severe conditions, such as excessive wind, ice, storms, lightning strikes or other natural occurrences;
- Damage due to improper packaging on return shipment.

Any and all labor charges for troubleshooting, removal or replacement of the product are not covered by this warranty and will not be honored by Solar MEMS Technologies S.L.

Return shipping to Solar MEMS Technologies S.L. must be pre-paid by the original consumer purchaser. Solar MEMS Technologies S.L. will pay the normal return shipping charges to original consumer purchaser within the European Union countries only.

Address of Solar MEMS Technologies S.L.
Solar MEMS Technologies S.L.
Parque Científico Tecnológico Cartuja 93.
Tecnoincubadora Marie Curie.
C/ Leonardo da Vinci 18, Planta 1, Módulo 2.
C.P. 41092, Seville, Spain.
E-mail: smt@solar-mems.com
Phone: (+34) 954 460 113

Solar MEMS has a system of quality and environment according to the ISO 9001 and ISO 14001 standards, provided by the certification company Applus CTC.