ESCUELA DE INGENIERÍAS INDUSTRIALES

Ingeniería de Sistemas y Automática

TRABAJO FIN DE GRADO

CONTROL CINEMÁTICO AVANZADO DEL ROBOT MÓVIL KROBOT CON SENSORES DE ENTORNO

Grado en

Ingeniería Electrónica, Robótica y Mecatrónica

Autor: JESSICA BEDOYA RINCÓN

Tutor: ANTONIO J. MUÑOZ RAMÍREZ

MÁLAGA, SEPTIEMBRE de 2.018

RESUMEN

En este Trabajo de Fin de Grado se parte del robot móvil Krobot con movimiento de tipo *skid-steer* perteneciente a la plataforma Piero con el de seguir avanzando con el sistema de control que ya poseía. En concreto, se trabajarán con sensores de entorno (láseres, IMUs, cámaras) y también se realizará una actualización del ordenador de abordo por una Raspberry Pi, un ordenador de pequeño tamaño y grandes recursos para realizar la comunicación y el procesamiento de la información necesaria.

El principal objetivo es integrar la información recogida por el sensor de unidad inercial (IMU) correspondiente a giros, inclinaciones y aceleraciones con las medidas tomada dentro de un radio de exploración por un sensor láser para conseguir orientar al robot de una forma más fiable. Para ello se utilizarán técnicas de comunicación con ROS que aseguren una completa sincronización entre todas las partes a la vez estando controlando todo el sistema mediante un algoritmo de programación desarrollado en la plataforma Simulink.

PALABRAS CLAVE

ROBÓTICA, SENSOR, LÁSER, IMU, RASPBERRY PI, SENSE HAT, MATLAB, SIMULINK, COMUNICACIÓN, ROS, CONTROL, LINUX

ABSTRACT

This Final Degree Project parts of the mobile robot Krobot with skid-stter movement belonging to the Piero platform with the aim of moving forward with the control system that it already had. In particular, we will work with environment sensors like lasers, IMUs, cameras, etc. We also Will update the on-board computer changing it for a Raspberry Pi, a small computer with large resources to carry out the communication and processing of the necessary information. .

The main objective is to integrate the information collected by the inertial unit sensor (IMU) corresponding to turns, inclinations and actions with the measurements taken within a scanning radius by a laser sensor to get orientation of the robot in a more reliable way. For this, communication techniques with ROS will be used to ensure complete synchronization between all the elements while controlling the entire system through a programming algorithm developed on the Simulink platform.

KEY WORDS

ROBOTICS, SENSOR, LÁSER, IMU, RASPBERRY PI, SENSE HAT, MATLAB, SIMULINK, COMUNICATION, ROS, CONTROL, LINUX

AGRADECIMIENTOS

A mi madre, por darme más de lo que puedo pedir.

A José María, por ser mi amigo y compañero a lo largo de toda la carrera.

Al profesor Antonio, por redescubrirme la robótica

Gracias a todos por hacer posible completar mis estudios.

CONTENIDO

Resumen 3
Palabras clave 3
Abstract 4
Key Words 4
Agradecimientos 5
Contenido 7
Índice de figuras 9
Capítulo 1: Introducción 12
1.1 Antecedentes 12
1.2 Objetivos 13
1.3 Metodología De Trabajo 13
1.4 Estructura De La Memoria 14
Capítulo 2: Entorno De Trabajo 15
2.1 Hardware 15
2.1.1 Raspberry PI 3 Model B 15
2.1.2 Sense Hat 17
2.1.3 Láser 2D Hokuyo UBG-04LX-F01 19
2.1.4 Router 3 COM Wireless 11n 21
2.1.5 Baterías 21
2.1.6 Conversor DC/DC 22
2.2 Software 23
2.2.1 Matlab y Simulink 23
2.2.2 Bitvise SSH Client 24
2.2.3 Sistema operativo Raspbian/Ubuntu 16.4 25
2.2.4 ROS 26
Capítulo 3: Comunicación De ROS En Simulink y Matlab 27
3.1 Principales características de ROS 27
3.2 Principales comandos 28
3.3 ROS y simulink 28
3.3.1 Programa básico en MATLAB 30
3.3.2 Programa básico en Simulink 31

3.3.3 Mensajes personalizados en Simulink 33
Capítulo 4: Sense Hat 36
4.1 Características de la IMU del Sense Hat 36
4.2.1 Acelerómetro 36
4.2.2 Giroscopio 36
4.2.3 Magnetómetro 37
4.2 Configuración de la Raspberry con la imagen de SIMULINK 37
4.2 Primeros experimentos con el módulo sense hat 40
4.3 Programación en Simulink 41
4.3.1 Diseño del Primer Filtro 42
4.3.2 Comparación y combinación de señales 45
4.3.3 Diseño del filtro final 46
4.3.4 Resultados 48
4.4 Implementación con ROS 53
Capítulo 5: Implementación Sensor Láser 2D 56
5.1 Programa oficial 56
5.2 Programa en Matlab que permite mostrar por pantalla los puntos del entorno 58
5.3 Comunicación con ros 59
Capítulo 7: Conclusiones 66
Capítulo 8: Líneas de Trabajo Futuras 67
Referencias bibliográficas 68

ÍNDICE DE FIGURAS

Figuras Capítulo 2

Figura 2. 1 Raspberry Pi 3 B 15
Figura 2. 2 Ordenador PICO-ITX P820 16
Figura 2. 3 Sense Hat 17
Figura 2. 4 Instalación del Sense Hat en la Raspberry 18
Figura 2. 5 Componentes Sense Hat 19
Figura 2. 6 Raspberry Pi encendida 19
Figura 2. 7 Hokuyo UBG-04LX-F01 20
Figura 2. 8 Rango de medida del láser 20
Figura 2. 9 3COM Wireless 11n Router 21
Figura 2. 10 Baterias 22
Figura 2. 11 Conversor DC/DC 22
Figura 2. 12 Power Bank alimentando a la Raspberry Pi 3 23
Figura 2. 13 Logotipo Matlab y Simulink 23
Figura 2. 14 Entorno de trabajo Simulink y librería Sense HAT 24
Figura 2. 15 CMD 25
Figura 2. 16 Explorador de archivos 25
Figura 2. 17 Logotipo Raspbian 26
Figura 2. 18 Distribución de ROS 26
Figura 2. 19 Logotipo ificial ROS 26

Figuras Capítulo 3

Figura 3. 1 ROS toolbox 28
Figura 3. 2 Inicialización del ROS Master 29
Figura 3. 3 Ejemplo de mensaje 30
Figura 3. 4 Acceso al contenido del topic 30
Figura 3. 5 Mensaje Matlab y Raspberry 31
Figura 3. 6 Comprobación de que se ha publicado el mensaje 31
Figura 3. 7 Librería ROS 32
Figura 3. 8 Configuración parámetros 32
Figura 3. 9 Modelo en Simulink 33
Figura 3. 10 Información del topic 33
Figura 3. 11 Mensaje tipo string 34
Figura 3. 12 Matlab Fuction 34
Figura 3. 13 Ubicación función Edit Data 34
Figura 3. 14 Editor de Datos 35

Figuras Capítulo 4

Figura 4. 1 Sense Hat 36
Figura 4. 2 Ejes de coordenadas giroscopio 37
Figura 4. 3 Instalación imagen 1 38
Figura 4. 4 Instalación imagen 2 38
Figura 4. 5 Instalación imagen 3 38
Figura 4. 6 Instaalación imagen 4 39
Figura 4. 7 Búsqueda Dirección IP Raspberry 39
Figura 4. 8 Conexión SSH 39
Figura 4. 9 Instalación de ROS mediante comandos 40
Figura 4. 10 Ejemplos Sense Hat 40
Figura 4. 11 Leds Space Invaders 41
Figura 4. 12 Leds Rainbow 41
Figura 4. 13 Configuración de parámetros 42
Figura 4. 14 Señal sin filtrar 42
Figura 4. 15 Giroscopio con Filtro de Mediana 43
Figura 4. 16 Filtro de la mediana 43
Figura 4. 17 Obtención del Offset 44
Figura 4. 18 44
Figura 4. 19 Comparación de señales 45
Figura 4. 20 Comparativa del angulo sacado del giroscopio y del acelerometro 45
Figura 4. 21 Señales comparativas con filtro offset y mediana 46
Figura 4. 22 Esquemático superior filtro final 47
Figura 4. 23 Esquemático de los filtros 48
Figura 4. 24 Experimento Roll 49
Figura 4. 25 Experimento Roll pi/2 49
Figura 4. 26 Experimento Roll Reposo 50
Figura 4. 27 Experimento Pitch pi/2 50
Figura 4. 28 Experimento Pitch reposo 51
Figura 4. 29 Experimento Yaw giro 52
Figura 4. 30 Experimento Yaw reposo 52
Figura 4. 31 Esquemático de communicacion ROS 53
Figura 4. 32 Peticionador 54
Figura 4. 33 Topic /lectura 54
Figura 4. 34 Información publicada en el topic /lectura 55
Figura 4. 35 Bloque giroscopio 55
Figura 4. 36 Matlab Function 56

Figuras Capítulo 5

Figura 5. 1 Ventana URG Benri Plus 57
Figura 5. 2 Ejemplo funcionamiento del programa 58
Figura 5. 3 Imagen generada con el programa de Matlab 59

Figura 5. 4 Conexión Bitvise 60
Figura 5. 5 Conexión wifi 60
Figura 5. 6 Página de ROS 61
Figura 5. 7 Repositorio de Github 61
Figura 5. 8 Explorador SFTP 61
Figura 5. 9 Catkin_make 62
Figura 5. 10 Diseño mapeo 65

ÍNDICE DE TABLAS

Tabla 1. 1 Comparativa de ordenadores de abordo 16
Tabla 1. 2 Características Sense Hat 18
Tabla 1. 3 Características Hokuyo 21

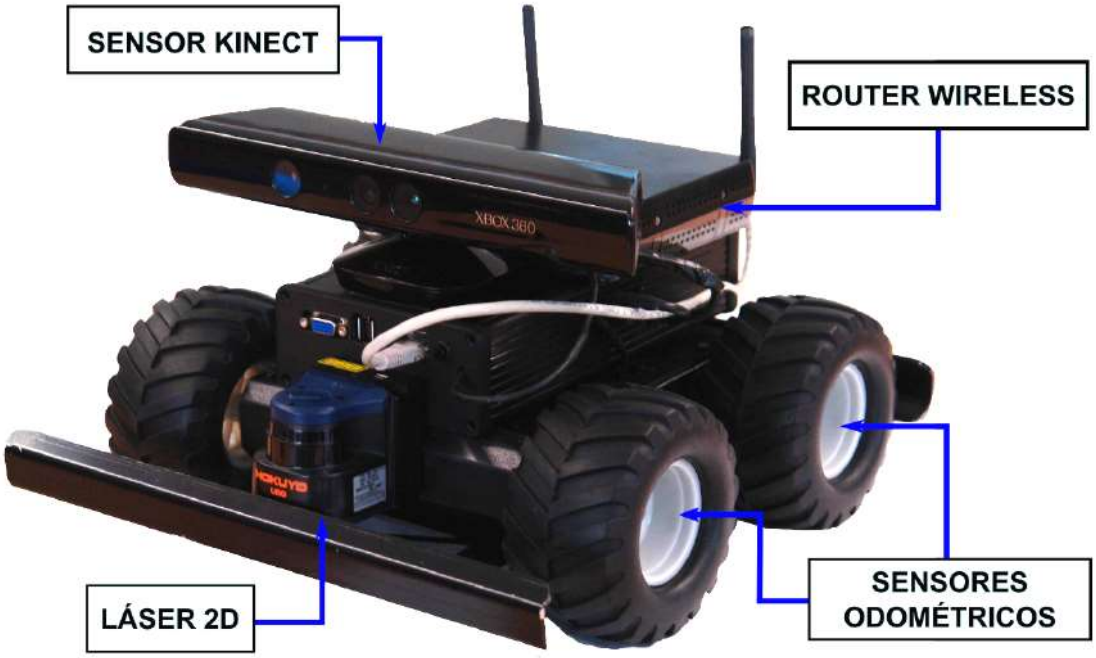
CAPÍTULO 1: INTRODUCCIÓN

1.1 ANTECEDENTES

Este Trabajo de Fin de Grado parte del modelo del robot Krobot hecho por Raúl Pimentel [1] para el Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga. Posteriormente, fue modificado por Carlos Miguel Jiménez [2], dónde se implementó un control de velocidad y posición simple para el robot Krobot. Además, se realizó la actualización del hardware y firmware del sistema junto con la incorporación de sensores que no pudieron llegar a utilizarse por la excesiva extensión del proyecto.

El robot Krobot posee 4 ruedas y se desplaza mediante tracción por deslizamiento o *skid steer,* donde las ruedas de cada lado deben tener la misma velocidad. Este tipo de desplazamiento aporta gran estabilidad al diseño y la ventaja de poder girar sobre su centro. Se utiliza comúnmente en vehículos oruga como tanques y bulldozer. Además, el desplazamiento *skeed steer* está estrechamente relacionado con la transmisión diferencial, reemplazando la rueda "loca" por ruedas motrices adicionales a cada lado.

En este TFG se implementarán nuevos elementos tales como un nuevo ordenador de a bordo, la Raspberry Pi 3 [3]que destaca, no por su alta potencia de procesamiento, sino por su alta versatilidad y la gran comunidad que la respalda teniendo, incluso, soporte para el software Simulink[4] de Matlab empleado habitualmente en el área de ingeniería. Además, se añadirá al diseño el sensor láser 2D de la marca Hokuyo[5] con el fin de realizar mapas del entorno del robot para mejorar el fiabilidad de su posición.



1.2 OBJETIVOS

El objetivo principal del trabajo de alto nivel consistirá en el desarrollo de un algoritmo de control cinemático para robots con tracción tipo *skid-steering* implementado sobre el robot móvil Krobot, donde los sensores del entorno permitan cerrar el bucle de control con una mayor precisión. Los objetivos desglosados sería los siguientes:

- Aprender a trabajar con ROS en Simulink
- Entender y desarrollar un programa con la IMU funcional para el robot
- Capturar datos del entorno con el sensor láser
- Combinar la información proporcionada por los dos sensores
- Implementación de los algoritmos elaborados con el sistema de bajo nivel

1.3 METODOLOGÍA DE TRABAJO

1. Familiarización con *midleware* ROS y su funcionamiento tanto en Raspberry pi (Ubuntu) como su utilización en Simulink
2. Estudio del Sense Hat, especialmente el sensor IMU incorporado y diseño de un filtro que, con la combinación de sus componentes, nos permita obtener de la forma más precisa posible su posición y/u orientación.

3. Estudio del funcionamiento del sensor láser 2D de Hokuyo y desarrollo de una aplicación que nos permita ubicar el robot en el espacio.
4. Diseño de un programa en Simulink que permita unir la información de ambos sensores para que trabajen simultáneamente y envíen la información utilizando ROS.
5. Integración de la cinemática de sensores de bajo nivel en la fase de pruebas para observar los resultados del diseño, corrección de errores, obtención de resultados definitivos.
6. Elaboración de la documentación del TFG, demostración final y realización de la presentación del trabajo en su conjunto.

1.4 ESTRUCTURA DE LA MEMORIA

En los siguientes puntos se describirá brevemente el contenido de cada capítulo de la memoria:

- En el capítulo 2 se realiza una descripción de los componentes del robot móvil Krobot relacionados para la realización del trabajo de fin de grado
- En el capítulo 3 se realiza un acercamiento a ROS con Simulink.
- En el capítulo 4 se introduce el funcionamiento del Sense Hat y de los distintos algoritmos para trabajar con él.
- En el capítulo 5 se presenta el sensor láser y las dinámicas utilizadas para trabajar con él.
- En el capítulo 6 se resumen los principales problemas a la hora de trabajar.
- En el capítulo 7 se exponen las conclusiones y las líneas de trabajo futuras.
- En el último apartado se incluyen las referencias bibliográficas.

CAPÍTULO 2: ENTORNO DE TRABAJO

En los siguientes apartados de este capítulo se van a describir los elementos hardware, compuestos por el ordenador de abordo, los sensores y la electrónica de potencia, que conforman el sistema de alto nivel del robot Krobot. Además, también se hablará del entorno de programación de los mismos y del software en el que se realizarán las aplicaciones de comunicación y control y procesamiento de la información.

2.1 HARDWARE

2.1.1 Raspberry PI 3 Model B

La Raspberry Pi es un ordenador de pequeñas dimensiones con bajo consumo y un coste reducido, lo que la convierte en una opción ideal para los proyectos de robótica. En especial, el modelo 3B de la Raspberry Pi se trata de uno de los más potentes de la familia con gran compatibilidad de dispositivos (cámaras, láseres, pantallas, sensores inerciales, etc.) y soporte oficial y de código abierto para las distintas distribuciones de Linux que soporta.

Además, esta versión incorpora conexión Wifi 802.11n y Bluetooth 4.1 de bajo consumo, lo que permite ahorrar espacio dando la posibilidad de eliminar otros dispositivos externos que cumplen esta función.

Figura 2. 1 Raspberry Pi 3 B

En el diseño inicial del Krobot, el ordenador de abordo lo conformaba un ordenador modelo PICO-ITX P820 (Figura 2.2) de la marca EPIA. El factor determinante de la sustitución del mismo por una Raspberry Pi fue su menor tamaño, menor consumo, las prestaciones extra que posee (Wifi) y la ventaja de suprimir el disco duro de estado sólido externo por algo tan pequeño como una tarjeta Micro SD.

Figura 2. 2 Ordenador PICO-ITX P820

A continuación, se muestra una tabla donde se comparan las principales características de ambos ordenadores.

	PICO-ITX P820	**Raspberry PI 3 Model B**
Capacidad de procesamiento	1.2 GHz VIA nano E	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
Tamaño memoria RAM	2 GB	1GB LPDDR2 SDRAM
Almacenamiento	Disco duro de estado sólido de 80 GB	Tarjeta microSD 32GB
Tensión de alimentación	12 V	5 V
Dimensiones	10cm x 7.2cm	8.5cm x 5.6
Sistema operativo	Windows XP	Raspbian/Ubuntu

Tabla 1. 1 Comparativa de ordenadores de abordo

2.1.2 Sense Hat

Sense HAT (Hardware Attached on Top) es un módulo oficial de Raspberry Pi diseñado para la misión Astro Pi en la cual estudiantes británicos podían desarrollar aplicaciones que el astronauta Tim Peake en la ISS las pudiera probar en la Raspberry Pi.

El módulo cuenta con diversos sensores como lo son el giroscopio, acelerómetro, magnetómetro, sensor de temperatura, sensor de presión barométrica y de humedad relativa. Además, cuenta con una matriz 8x8 de LEDs RGB programable y un joystick de cinco botones. Para desarrollar el proyecto del control del robot, haremos uso de los sensores que conforman la IMU (unidad de medición inercial) donde el dispositivo permite capturar hasta 952 muestras por segundo.

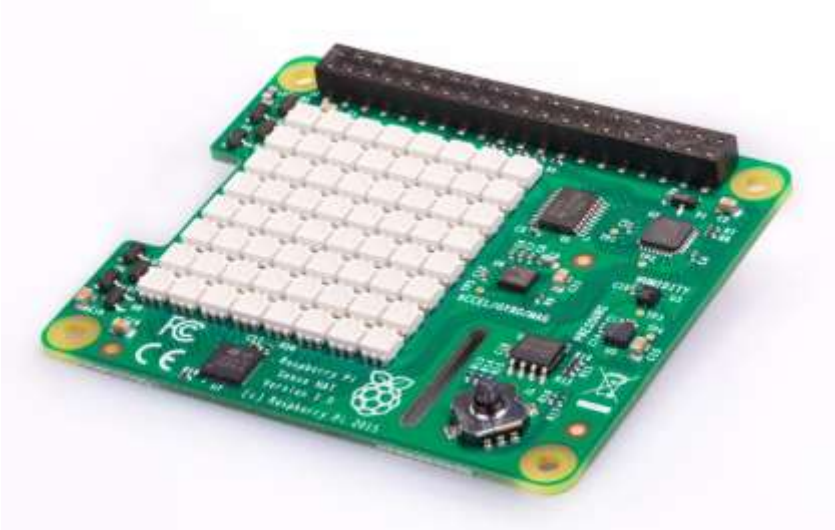


Figura 2. 3 Sense Hat

El módulo Sense HAT se conecta a la Raspberry Pi en los 40 pines GPIO disponibles, tal y como se muestra en la Figura 4. Todos sus sensores se conectan a través de I2C y la programación de la placa se hace mediante la interfaz SPI de la Raspberry, lo cual permite reprogramarla o agregar firmware.

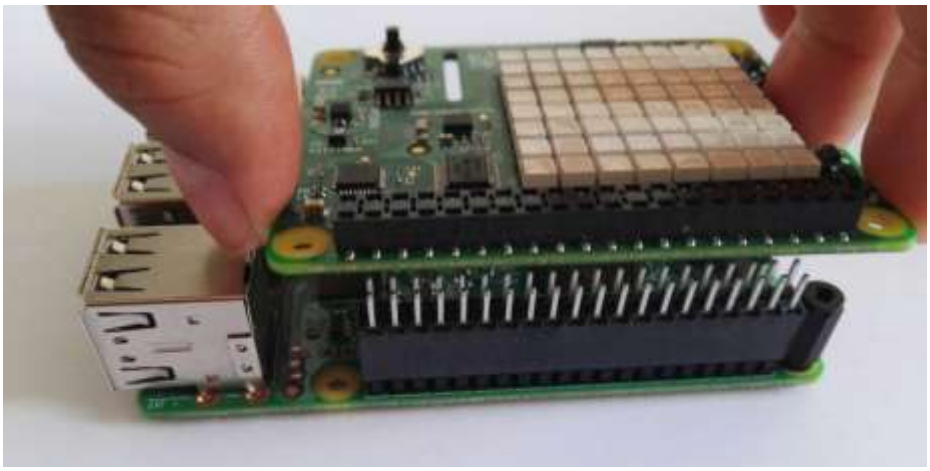


Figura 2. 4 Instalación del Sense Hat en la Raspberry

Módulo	Especificaciones
Giroscopio	Sensor de ratio angular: ±245/500/2000dps
Acelerómetro	Sensor de aceleración lineal: ±2/4/8/16 g
Magnetómetro	Sensor magnético: ±4/8/12/16 gauss
Barómetro	260 – 1260 hPa (la precisión depende de la temperatura y la presión, ±0.1 hPa en condiciones normales)
Sensor de Temperatura	Precisión en la temperatura de ± 2 °C en el rango de 0-65 °C
Sensor de Humedad Relativa	Precisión de ±4.5% en el rango de 20-80%rH , precisión de ± 0.5 °C en el rango de 15- 40 °C)
Matriz de LEDs	8x8 elementos con 15 bits de resolución
Joystick	5 botones, escaneado de 80 veces por segundo

Tabla 1. 2 Características Sense Hat

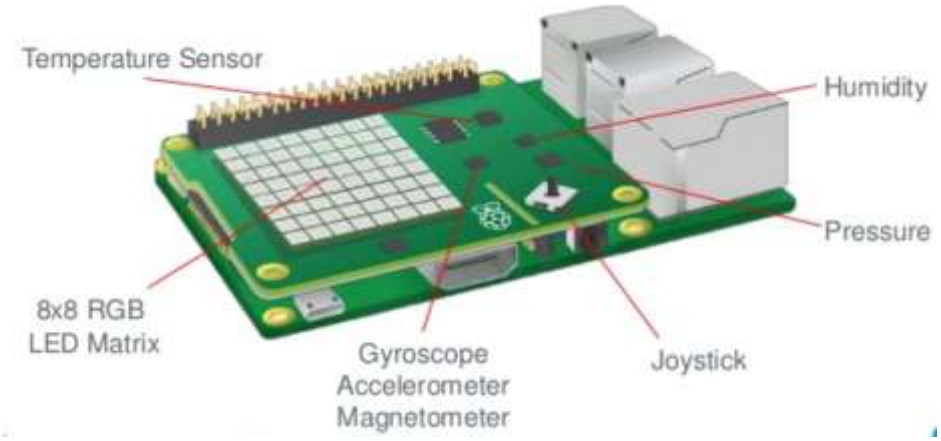


Figura 2. 5 Componentes Sense Hat

Figura 2. 6 Raspberry Pi encendida

2.1.3 Láser 2D Hokuyo UBG-04LX-F01

El escáner láser Hokuyo UBG-04LX-F01 es un dispositivo pequeño, preciso y de alta velocidad perfecto para aplicaciones robóticas. El láser se utilizará para detectar los obstáculos que se encuentren en la trayectoria del robot y su entorno en general.

Este modelo está provisto de comunicación RS232 y USB siendo reconocido por el ordenador como un puerto serie virtual (COMx). Puede obtener datos del entorno en un rango de 4 metros con resolución milimétrica a una velocidad de muestreo de 28 metros por segundo, tres veces más rápido que los modelos

convencionales. Además, debido a su bajo consumo de energía, este dispositivo puede usarse en plataformas operadas con baterías.

Figura 2. 7 Hokuyo UBG-04LX-F01

El funcionamiento del sensor láser Hokuyo es sencillo: el dispositivo emite un rayo láser infrarrojo de 785nm de longitud de onda para medir la distancia a los objetos en un plano de medición mediante un espejo rotatorio que desvía los haces de luz en un área de barrido semicircular de 240º de rango. El principio de medición de la distancia se basa en el cálculo de la diferencia de fase, debido a la cual es posible obtener una medida estable con mínima influencia del color del objeto y reflectancia.

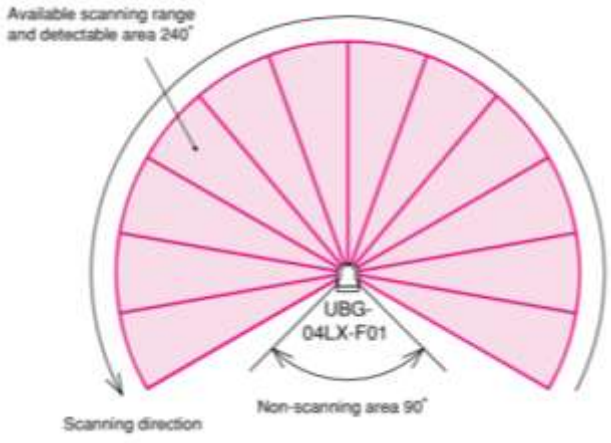


Figura 2. 8 Rango de medida del láser

En la siguiente tabla se recogen las especificaciones más relevantes del sensor láser.

Alimentación	12 V DC
Rango de detección	20 – 5600 mm
Resolución angular	240º

Interfaz	USB 2.0, RS-232C
Tiempo de escaneo	28m/segundo
Peso	260g

Tabla 1. 3 Características Hokuyo

2.1.4 Router 3 COM Wireless 11n

Este dispositivo será el puente de enlace entre la comunicación del sistema de alto nivel, es decir, el ordenador de abordo, con el control de bajo nivel mediante una placa ESP8266 encargada de enviar información al Arduino.

Sin embargo, es posible prescindir de este mecanismo si se realiza una adecuada conexión directa entre el propio WiFi de la raspberry Pi 3 con el módulo WiFi ESP8266.

Figura 2. 9 3COM Wireless 11n Router

2.1.5 Baterías

Para alimentar los sensores y el ordenador de abordo se precisan dos baterías en serie con las mismas características con el fin de obtener la tensión necesaria para el correcto funcionamiento de todo el sistema.

El elemento más exigente en cuanto a tensión es el sensor láser el cual funciona correctamente a partir de los 12 V, mientras que la Raspberry Pi 3 sólo requiere de 5 V. De este modo, la tensión máxima que aportan las baterías es de 14.4V de tensión nominal cumpliéndose así los requisitos, y el máximo valor que puede almacenar son 16.8V.

Figura 2. 10 Baterías

2.1.6 Conversor DC/DC

Para alimentar los dispositivos que vamos a utilizar se requiere tener una tensión constante. Para ello se utilizó en el diseño inicial el convertidor RP30-2412SFW de la casa Recom1. Este conversor proporciona una tensión de salida de 12 V a partir de una tensión de entrada comprendida en el rango de 9V a 16 V con una corriente de salida máxima de 2.5 A.

Figura 2. 11 Conversor DC/DC

Sin embargo, con la electrónica de potencia actualmente instalada no es posible alimentar la Raspberry Pi ya que ésta funciona a 5V y la tensión que proporciona el conversor, como hemos señalado anteriormente, es de 12 V. Por este motivo, se hará uso de una batería externa (Power Bank) con una capacidad de 10000mAh y 2.1 A de salida, suficiente para que la Raspberry funcione correctamente con periféricos.

Figura 2. 12 Power Bank alimentando a la Raspberry Pi 3

2.2 SOFTWARE

2.2.1 Matlab y Simulink

Matlab (Matrix Laboratory) es un lenguaje de programación utilizado para realizar cálculos técnicos científicos de propósito general que permite visualizar y graficar los resultados y cuya variable básica de trabajo es la matriz.

Simulink es una toolbox especial de Matlab que sirve para modelar, simular y analizar el comportamiento de los sistemas dinámicos mediante bloques en un entorno gráfico (GUI). El comportamiento de los sistemas que se van a modelar se puede definir mediante funciones de transferencia, operaciones matemáticas simples con bloques propios o complejas dentro de funciones de Matlab, entre otros.

Figura 2. 13 Logotipo Matlab y Simulink

Además, una de las características más destacables de Simulink es el gran abanico de librerías que posee con multitud de bloques diseñados para ejecutar una determinada función. Por ejemplo, existe una librería propia de Raspberry Pi que dispone de bloques específicos para el módulo Sense HAT, que permite obtener directamente el valor de cualquiera de los sensores o incluso modificar la matriz de LEDs.

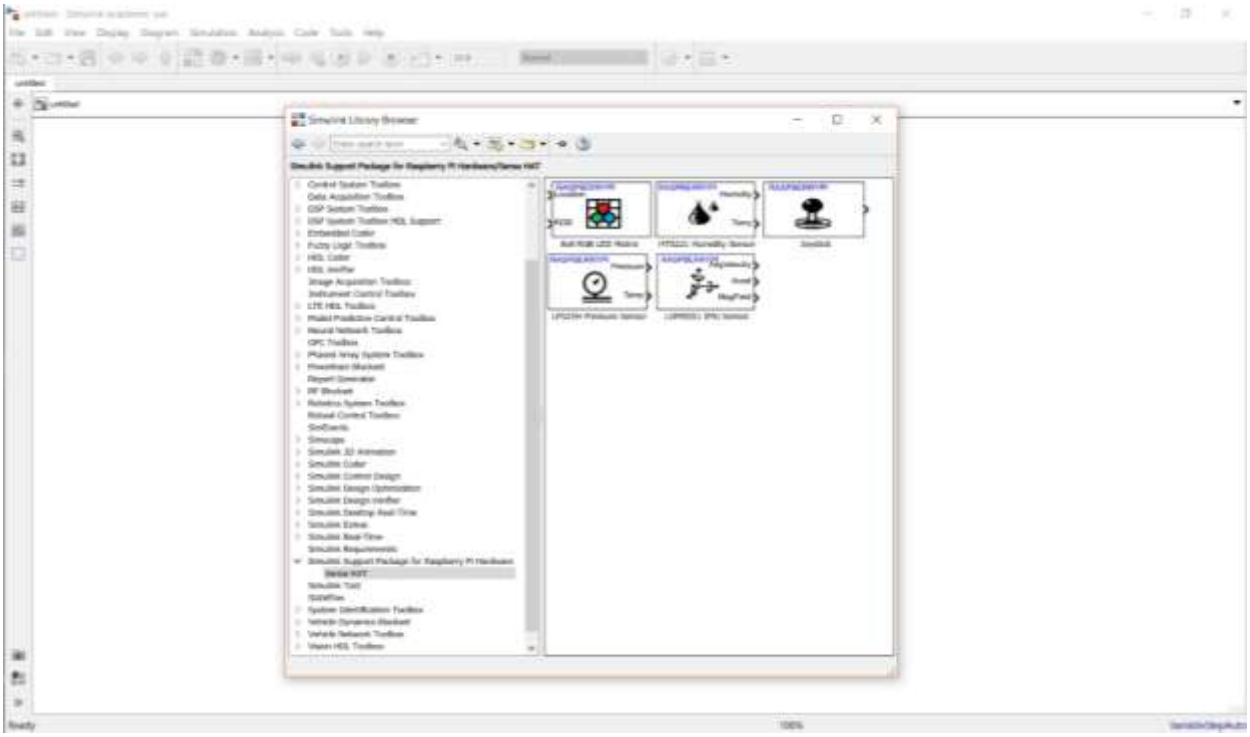


Figura 2. 14 Entorno de trabajo Simulink y librería Sense HAT

2.2.2 Bitvise SSH Client

Bitvise SSH Client[6] para Windows es un cliente de conexión bastante potente y gratuito, que nos permitirá establecer comunicación mediante SSH, SFTP, RDP y tiene muchas opciones de configuración.

De los protocolos anteriores, haremos uso principalmente del SSH. El SSH (o Secure SHell) es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. A diferencia de otros protocolos de comunicación, SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas. Por defecto se utiliza el puerto 22.

Por tanto, gracias a este programa podremos acceder a la Raspberry Pi, tanto para ejecutar comandos y programas mediante la consola de comandos (CMD) de

forma remota (Figura 2.15) como añadir y modificar elementos desde un navegador de archivos (Figura 2.16).

Figura 2. 15 CMD

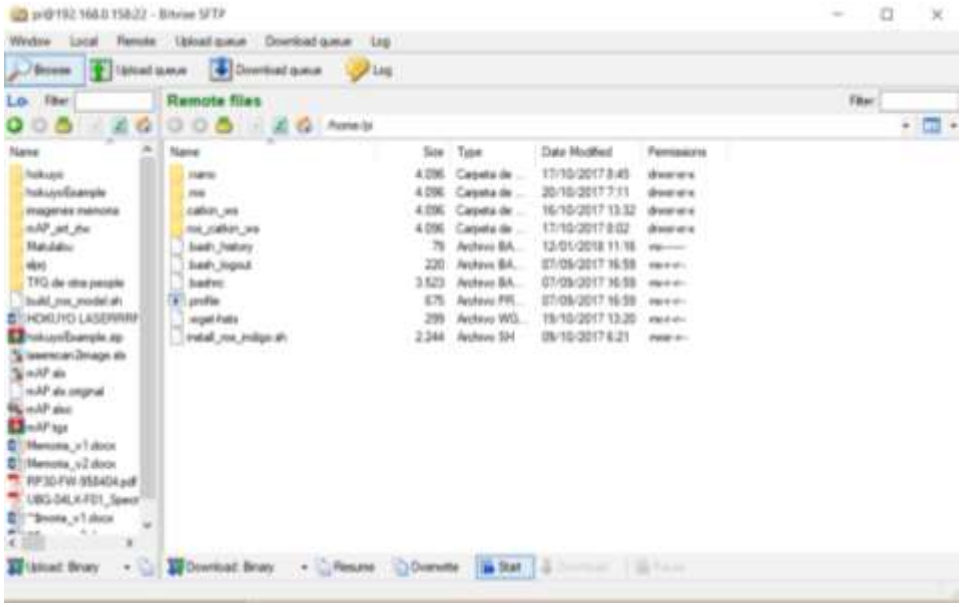


Figura 2. 16 Explorador de archivos

2.2.3 Sistema operativo Raspbian/Ubuntu 16.4

Raspbian[7] es un sistema operativo de código libre basado en Debian que está optimizado específicamente para el funcionamiento en placas Raspberry Pi. Al ser una distribución GNU/Linux, contiene multitud de repositorios y una comunidad gigantesca donde es posible encontrar los programas necesarios para desempeñar las mismas funcionalidades que un ordenador personal.

Por otro lado, Ubuntu, al igual que Raspbian, es una distribución de Linux basado en la arquitectura de Debian.

En el inicio del proyecto se utilizó la última versión de Raspbian (Stretch) proporcionada en el paquete de Simulink para Raspberry Pi, la cual traía la configuración necesaria para su total compatibilidad con Simulink. En ella se realizaron todas las pruebas y modelos iniciales para la familiarización con el módulo Sense HAT y ROS.

Sin embargo, debido a incompatibilidades con el nodo ROS del sensor láser Hokuyo, el sistema operativo Raspbian fue sustituido por Ubuntu 16.04. Ubuntu[8], al igual que Raspbian, es una distribución de Linux basado en la arquitectura de Debian. Y la imagen utilizada se obtuvo de Ubiquiy Robotics[9] la cual venía preconfigurada con ROS Kinetic correctamente instalado y que permtió el acceso al láser 2D.

Figura 2. 17 Logotipo Raspbian

2.2.4 ROS

ROS (Robot Operating System)[7] es un entorno de trabajo flexible, de código libre para el desarrollo de software para robots. Engloba librerías y herramientas para facilitar el desarrollo de comportamientos complejos y robustos en robots. Proporciona abstracción de hardware, control de dispositivos de bajo nivel,

Figura 2. 19 Logotipo ificial ROS

Figura 2. 18 Distribución de ROS

implementación de las funciones de uso común, intercambio de mensajes entre procesos y gestión de paquetes.

CAPÍTULO 3: COMUNICACIÓN DE ROS EN SIMULINK Y MATLAB

En este capítulo se expondrán los elementos que constituyen el middleware ROS, así como una breve introducción sobre su funcionamiento en Simulink mediante modelos básicos de comunicación.

3.1 PRINCIPALES CARACTERÍSTICAS DE ROS

- **Maestro**: Es imprescindible para que los nodos puedan comunicarse entre sí. El Maestro proporciona el registro de nombres y localiza el resto de los elementos que se encuentran en ejecución.
- **Nodos**: Los nodos son procesos que realizan la computación, proveen y consumen información mediante la suscripción a topics. Los nodos también proporcionan servicios, es decir, una solicitud a un nodo particular y se pueden configurar de acuerdo con ciertos parámetros.
- **Topics**: Los topics se denominan buses sobre los cuales los nodos intercambian mensajes. Para el intercambio de información, los nodos pueden suscribirse o publicar en cualquier topic de forma "anónima".
- **Servicios**: Los servicios permiten que un nodo (el cliente del servicio) solicite una determinada operación a otro nodo (el servidor del servicio) y responde con una respuesta estructurada.
- **Parámetros**: El servidor de parámetros contiene un diccionario compartido de los parámetros de configuración de todos los nodos en ejecución. Los parámetros pueden ser consultados y modificado en cualquier momento.
- **Mensajes**: Los nodos se comunican entre sí a través del intercambio de mensajes. Un mensaje es simplemente una estructura de datos, que contiene campos clasificados. Estos campos pueden contener datos de tipo estándar (int, float, bool) y arrays de este tipo de datos.

3.2 PRINCIPALES COMANDOS

- **roscd:** cambia a un directorio específico de un paquete.
- **roscore**: ejecuta todos los procesos necesarios para la ejecución del sistema completo de ROS.
- **roscreate-pkg**: crea e inicializa paquetes.
- **rosnode**: comandos para nodos
  - rosnode info nodo: muestra información del nodo en cuestión
  - rosnode kill nodo: finaliza el nodo
  - rosnode list: muestra los nodos que se están ejecutando
  - rosnode ping nodo: comprueba que se tiene conexión con el nodo
- **rosrun**: permite ejecutar cualquier nodo
- **rostopic**: comandos para Topics:
  - rostopic echo: muestra los mensajes que hay en el topic (como si se tratase de un suscriptor)
  - rostopic find: encuentra un topic
  - rostopic info: muestra la información que hay en el topic como los suscriptores, publicadores, tipo de mensaje, etc.
  - rostopic list: muestra una lista con los topics activos
  - rostopic pub: publica un mensaje en un topic activo
- **roswtf**: permite chequear si algo va mal. Ejecutamos roscd y después roswtf.

3.3 ROS Y SIMULINK

Para utilizar ROS en Matlab Y Simulink es necesario tener instalado el paquete (Figura 3.1) que nos permitirá utilizar las funciones de ROS, suscribirnos a topics, publicar mensajes, etc.

Figura 3. 1 ROS toolbox

Ahora, para comprobar si funciona correctamente, creamos un master de ROS en Matlab y publicamos unos mensajes. Para lanzar el máster, introducimos en la ventana de comandos "rosinit" y esto configurará el master como el localhost, a menos que indiquemos otra opción.

Figura 3. 2 Inicialización del ROS Master

3.3.1 Programa básico en MATLAB

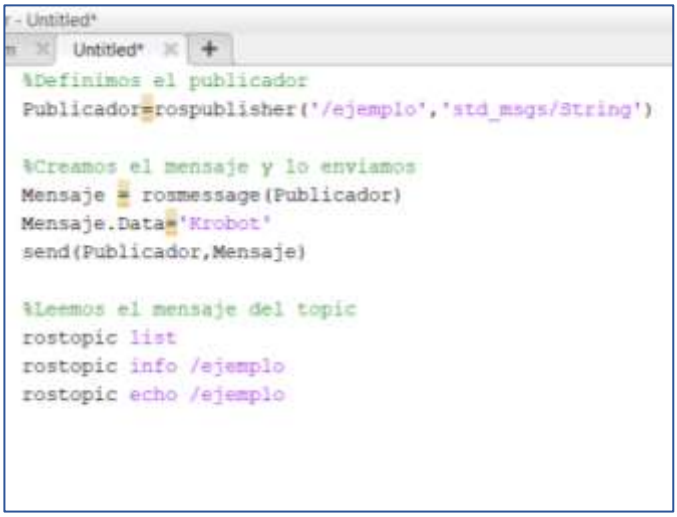


Figura 3. 3 Ejemplo de mensaje

Figura 3. 4 Acceso al contenido del topic

A continuación, para publicar el mensaje tenemos que crear un publicador indicando el topic donde se quiere escribir y qué clase de mensaje se va a utilizar. Para crear el mensaje se debe utilizar el publicador que ya hemos definido y ahora basta con rellenar el campo "Data" del mensaje para darle el contendido y, finalmente, poder enviarlo.

Ahora realizamos el mismo proceso pero iniciando el máster en la Raspberry Pi y comprobamos que el mensaje se ha publicado satisfactoriamente en el topic. En este caso, el mensaje es Hello Raspberry Pi y publicamos directamente en el topic /rosout.

Nota: El comando openShell(r) realiza conexión remota SSH con la Rapsberry Pi, igual que lo hace el programa Bitvise.

```
>> rosinit('NodeHost','10.10.10.15')
Initializing ROS master on http://3506L2W7PC05:11311/.
Initializing global node /matlab_global_node_61578 with 1
>> rostopic list
/rosout
>> r = raspberrypi
openShell(r)

r =

  raspberrypi with properties:

    DeviceAddress: '10.10.10.10'
             Port: 22

>> rostopic list
/rosout
>> b = rospublisher('/rosout');
msg = rosmessage(b);
msg.Msg = 'Hello Raspberry Pi!'
send(b,msg);
```

Figura 3. 5 Mensaje Matlab y Raspberry

```
pi@Rpi3Piero:~ $ sudo ping 10.10.10.15
PING 10.10.10.15 (10.10.10.15) 56(84) bytes of data.
64 bytes from 10.10.10.15: icmp_seq=1 ttl=128 time=3.88 ms
64 bytes from 10.10.10.15: icmp_seq=2 ttl=128 time=4.54 ms
64 bytes from 10.10.10.15: icmp_seq=3 ttl=128 time=4.64 ms
64 bytes from 10.10.10.15: icmp_seq=4 ttl=128 time=4.58 ms
64 bytes from 10.10.10.15: icmp_seq=5 ttl=128 time=11.1 ms
^C
--- 10.10.10.15 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 3.886/5.769/11.191/2.726 ms
pi@Rpi3Piero:~ $ rostopic echo /rosout
header:
  seq: 0
  stamp:
    secs: 0
    nsecs:         0
  frame_id: ''
level: 0
name: ''
msg: Hello Raspberry Pi!
file: ''
function: ''
```

Figura 3. 6 Comprobación de que se ha publicado el mensaje

3.3.2 Programa básico en Simulink

La librería ROS de Simulink posee todos los elementos necesarios para establecer una correcta y fluida comunicación mediante ROS. Entre ellos se encuentran *Publisher, subcriber, Blank Messages, etc.* (Figura 3.7). Son muy sencillos de utilizar y el tipo de mensaje se encuentra en una lista desplegable desde la que se puede seleccionar.

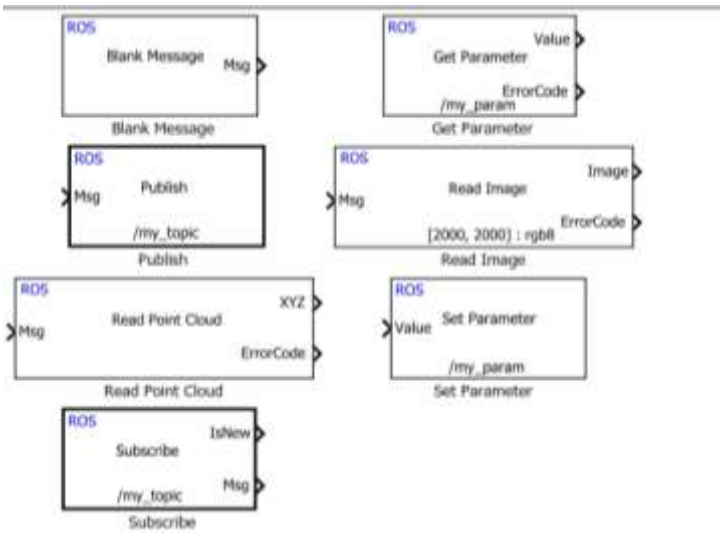


Figura 3. 7 Librería ROS

Para crear el primer programa, primero, debemos seleccionar en los parámetros de configuración el hardware que vamos a utilizar, es decir, ROS (Figura 3.6). En la lista desplegable también aparece la opción *Raspberry Pi-Robot Operating System(ROS),* sin embargo, no permite ejecutar el programa en modo externo en la Raspberry Pi para la realización de pruebas, por lo que se desechó esta opción.

Figura 3. 8 Configuración parámetros

A continuación, creamos un modelo sencillo donde rellenamos un mensaje en blanco con los valores de las señales seno y coseno y lo publicamos en el topic /prueba. Es importante seleccionar el tipo de mensaje, en este caso es geometry_msgs tipo Point

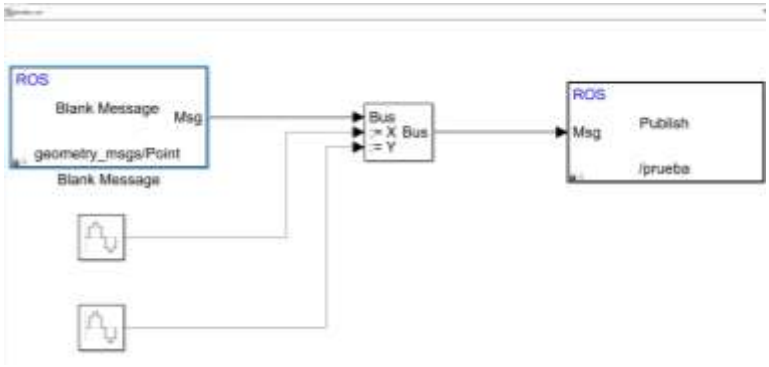


Figura 3. 9 Modelo en Simulink

Por último, desde la consola de comandos de MATLAB, realizamos un *echo* al *topic* para ver la información que se está publicando.

Figura 3. 10 Información del topic

3.3.3 Mensajes personalizados en Simulink

En la página oficial de MATLAB, se puede descargar un modelo que posee muchos tipos de mensaje en bloques a modo de ejemplo,

Se ha adaptado el bloque que rellena el mensaje en blanco con el mensaje tipo string deseado. El bloque es un script de MATLAB donde tiene como entradas el mensaje en blanco y el string y su salida es el mensaje en el formato adecuado para publicarlo. De este modo, se guarda la longitud del mensaje tipo string y se convierte el mensaje a uint32 en el campo destinado para ello (Figura 3.13). Para su correcto funcionamiento, se debe modificar el nombre del bus para que enlace con el modelo de

Simulik y el tipo de mensaje, en este caso el nombre es otra_prueba2 y el mensaje es tipo string (Figura 3.14).

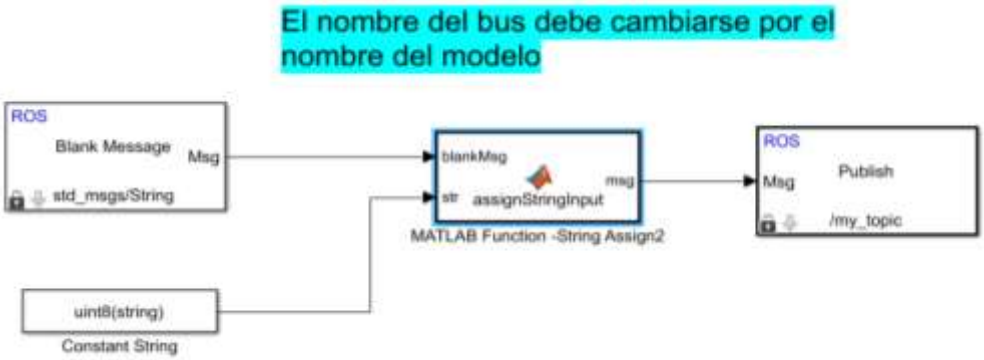


Figura 3. 11 Mensaje tipo string

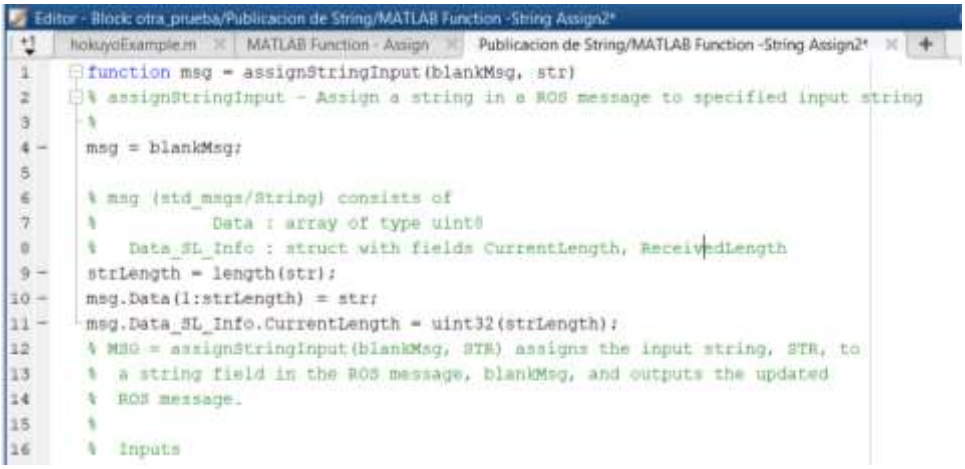


Figura 3. 12 Matlab Fuction

Figura 3. 13 Ubicación función Edit Data

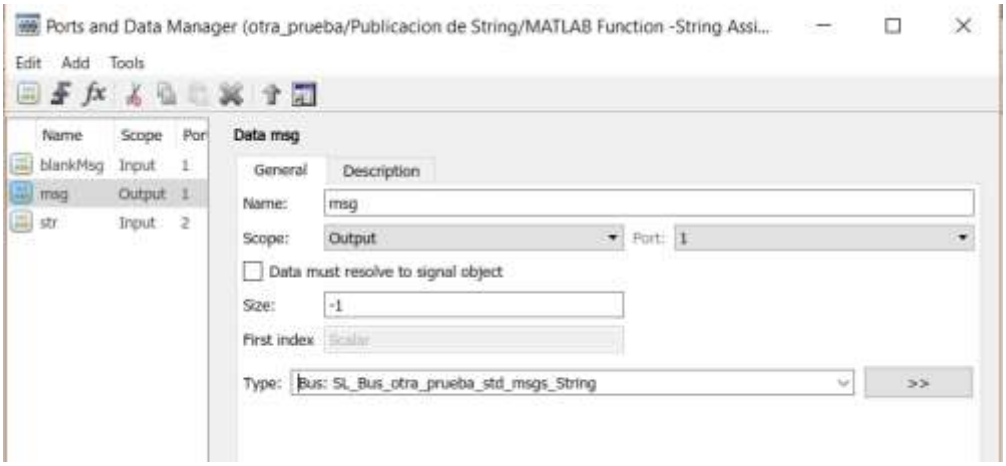


Figura 3. 14 Editor de Datos

CAPÍTULO 4: SENSE HAT

En este capítulo se explicarán las componentes de la IMU (Unidad de Medida Inercial), para conocer las diferencias entre ellas y poder aplicar adecuadamente el acondicionamiento de la señal mediante Simulink que pueda, posteriormente, implementarse en el diseño del robot móvil Krobot y así ayudar a calcular su orientación.

4.1 CARACTERÍSTICAS DE LA IMU DEL SENSE HAT

Figura 4. 1 Sense Hat

4.2.1 Acelerómetro

El acelerómetro es un dispositivo que permite medir la aceleración en los ejes X, Y y Z independientemente, por lo que podemos conocer a la vez la magnitud y dirección de la aceleración que sufre el sistema. En nuestro caso, el acelerómetro nos será muy útil ya que nos permitirá conocer la orientación en cada eje respecto a la posición inicial considerando las aceleraciones que se están midiendo

Sin embargo, su comportamiento no es adecuado para determinar la velocidad de un sistema ni su posición ya que acumula mucho ruido y errores en la medición a lo largo del tiempo, lo que conlleva efectos de deriva. Además, es muy sensible a vibraciones las cuales pueden causar ruidos de alta frecuencia en la señal de salida.

4.2.2 Giroscopio

El giroscopio mide la orientación de un objeto. Estos sensores son puramente diferenciales, es decir, adquieren ángulos relativos respecto a una referencia arbitraria. Posee tres grados de libertad: Pitch (arriba y abajo), Yaw (izquierda y derecha) y Roll (giro sobre el eje X), tal y como se aprecia en la Figura 4.2.

Son sensores que proporcionan una medida muy precisa en tiempos cortos, aunque a medio y largo plazo presentan una señal que se va desviando del valor real.

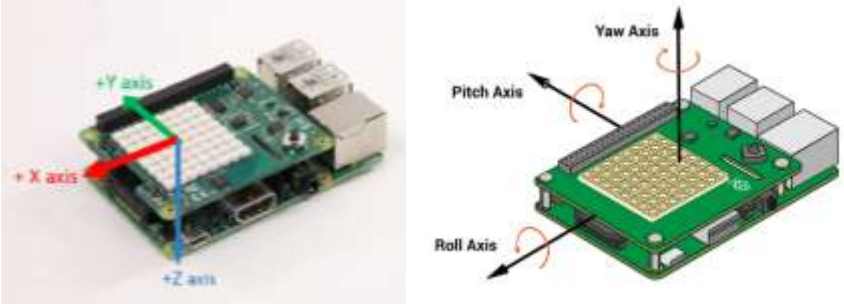


Figura 4. 2 Ejes de coordenadas giroscopio

4.2.3 Magnetómetro

El magnetómetro se utiliza para medir la fuerza y la dirección del campo magnético de la Tierra y que el sistema reconozca el Norte magnético. La medida tomada por el sensor tiene como resultado tres señales correspondientes a la fuerza y la dirección del campo magnético en los ejes X, Y y Z.

4.2 CONFIGURACIÓN DE LA RASPBERRY CON LA IMAGEN DE SIMULINK

En primer lugar, nos vamos a la ventana de *Add-ons* de Matlab y en el paquete de Simulink support for Raspberry Pi seleccionamos la opción *setup*. Para instalar la imagen basta con seguir el asistente que posee el propio paquete. La versión de Raspbian es Stretch y viene con la distribución de ROS INDIGO.

Figura 4. 3 Instalación imagen 1

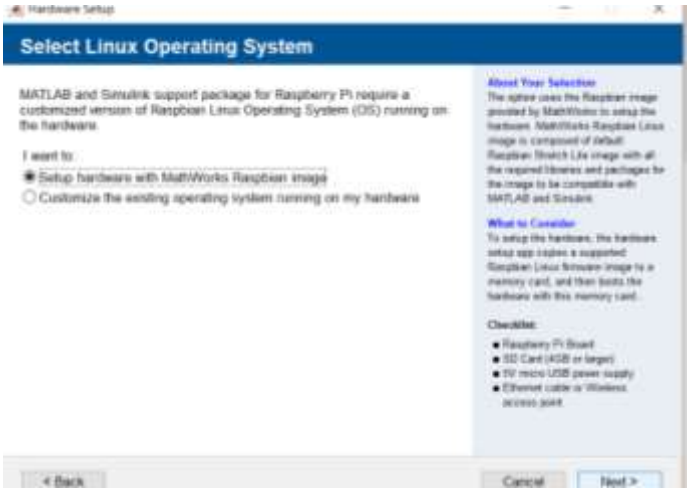


Figura 4. 4 Instalación imagen 2

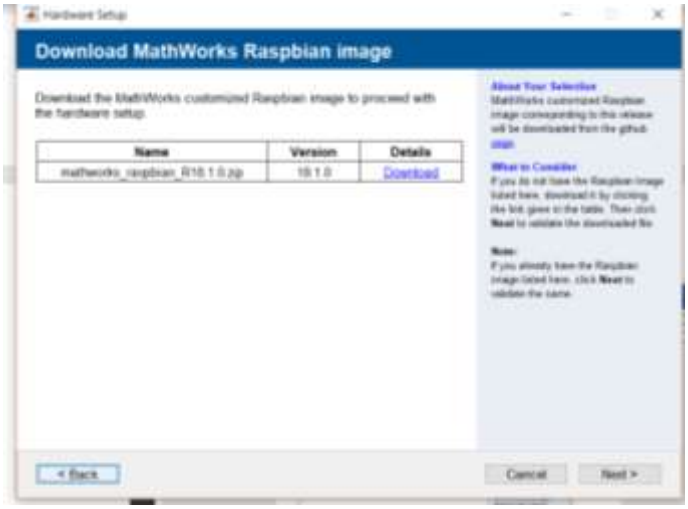


Figura 4. 5 Instalación imagen 3

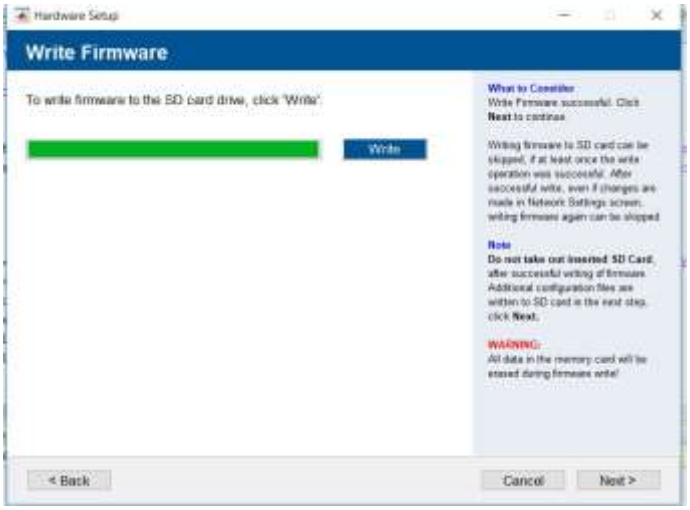


Figura 4. 6 Instaalación imagen 4

Una vez completada la instalación de la imagen, es recomendable instalar el sistema de ROS cuyo instalador se encuentra en la carpeta principal de la imagen. Primero establecemos conexión remota SSH con la Raspberry Pi y después ejecutamos el comando sh install_ros_indigo.sh en la ventana de comandos (Figura 4.9).

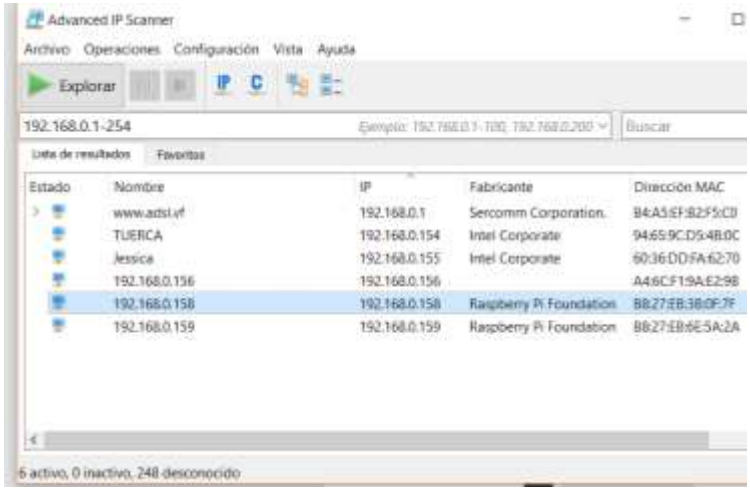


Figura 4. 7 Búsqueda Dirección IP Raspberry

Figura 4. 8 Conexión SSH

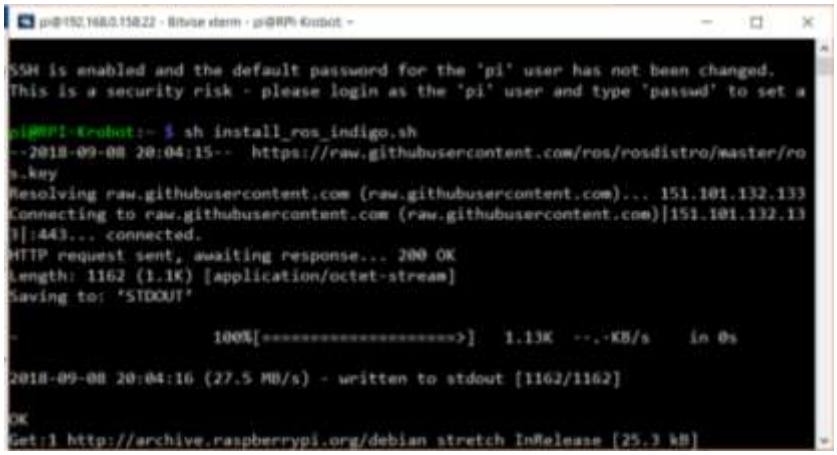


Figura 4. 9 Instalación de ROS mediante comandos

4.2 PRIMEROS EXPERIMENTOS CON EL MÓDULO SENSE HAT

La imagen proporcionada por Simulink posee instalado el paquete correspondiente del Sense Hat. Para comprobar su correcto funcionamiento, ejecutaremos algunos archivos de ejemplo que se encuentran alojados dentro del mismo paquete. El comando necesario para este propósito es *sudo Python "archivo".py*.

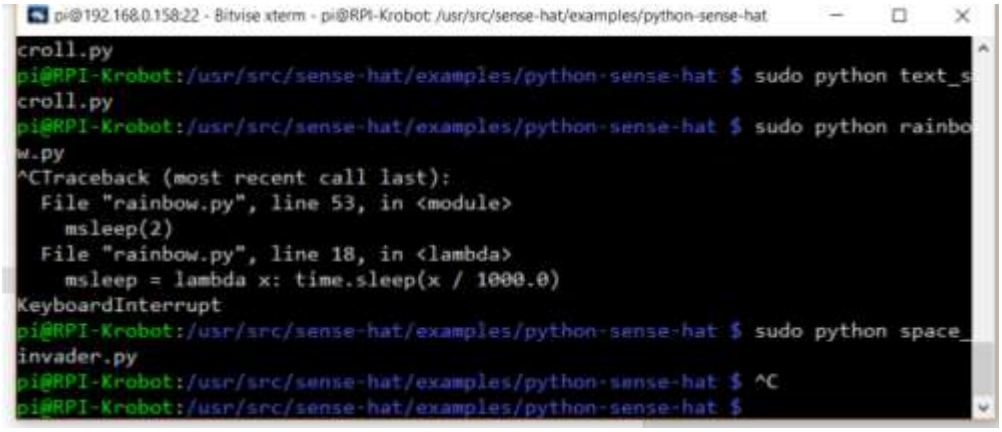


Figura 4. 10 Ejemplos Sense Hat

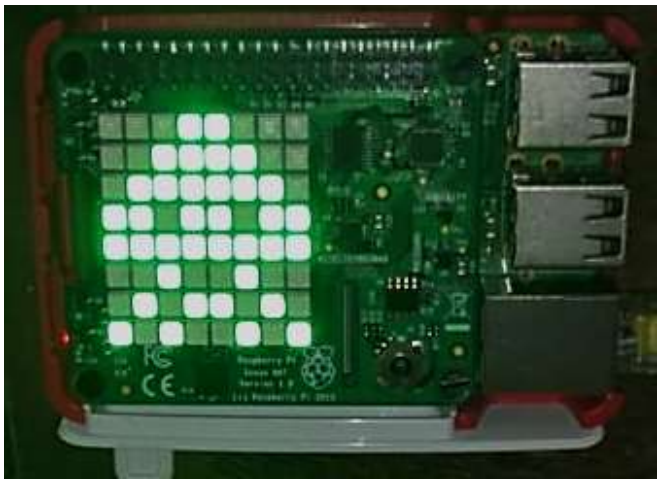


Figura 4. 11 Leds Space Invaders

Figura 4. 12 Leds Rainbow

4.3 PROGRAMACIÓN EN SIMULINK

En primer lugar, para poder establecer conexión con la Raspberry Pi y poder programarla, se debe indicar su dirección IP en la configuración de parámetros (Figura 4.13).

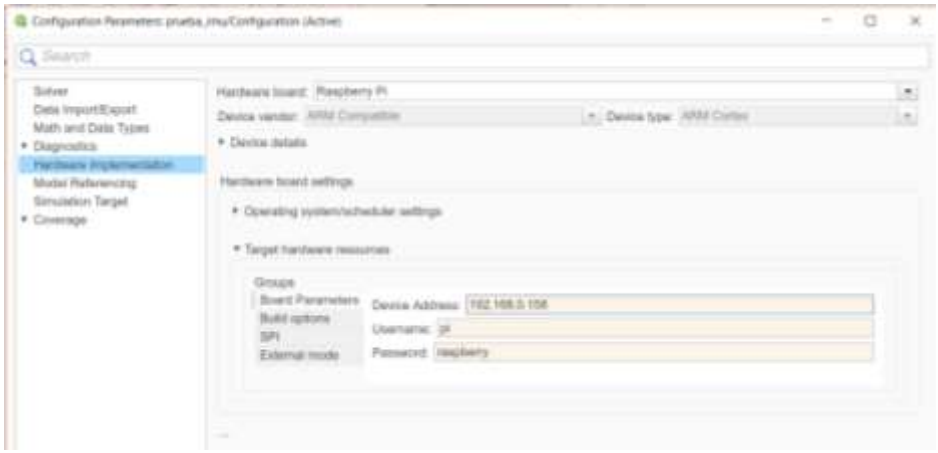


Figura 4. 13 Configuración de parámetros

4.3.1 Diseño del Primer Filtro

Se procede a realizar una lectura de las señales sin ninguna modificación para comprobar cómo se comporta el sistema ante ausencia de perturbaciones, y se observa que las señales de salida son muy ruidosas y poseen picos puntuales nada deseables para el control del sistema (Figura 4.15). Por este motivo, se procede a construir un filtro de mediana, que en esencia se comporta como un filtro de paso alta donde se desprecian los picos puntuales. En la figura 4.16 se puede observar claramente cómo hemos eliminado esa componente de la señal.

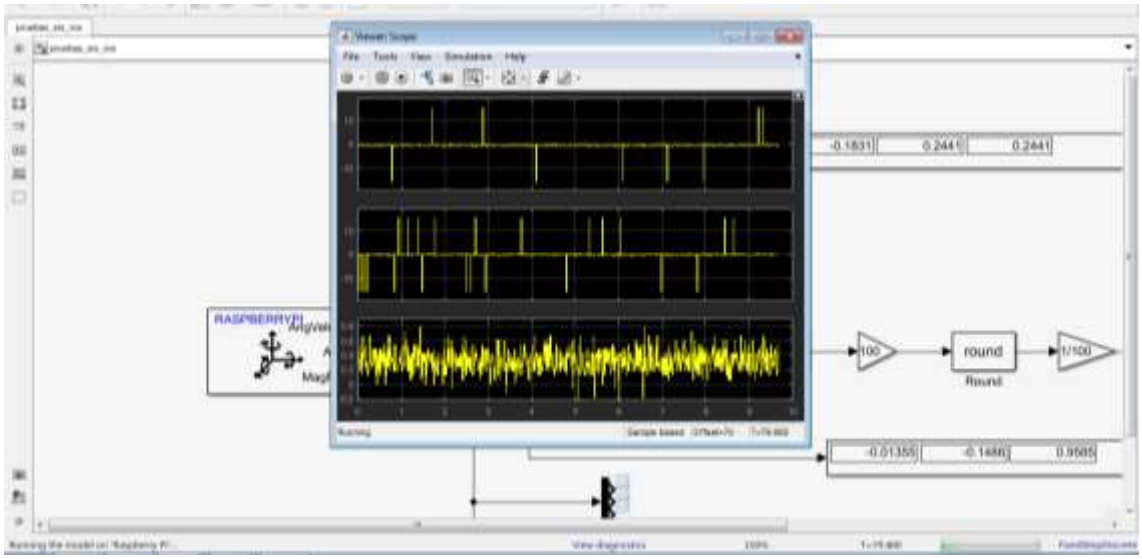


Figura 4. 14 Señal sin filtrar

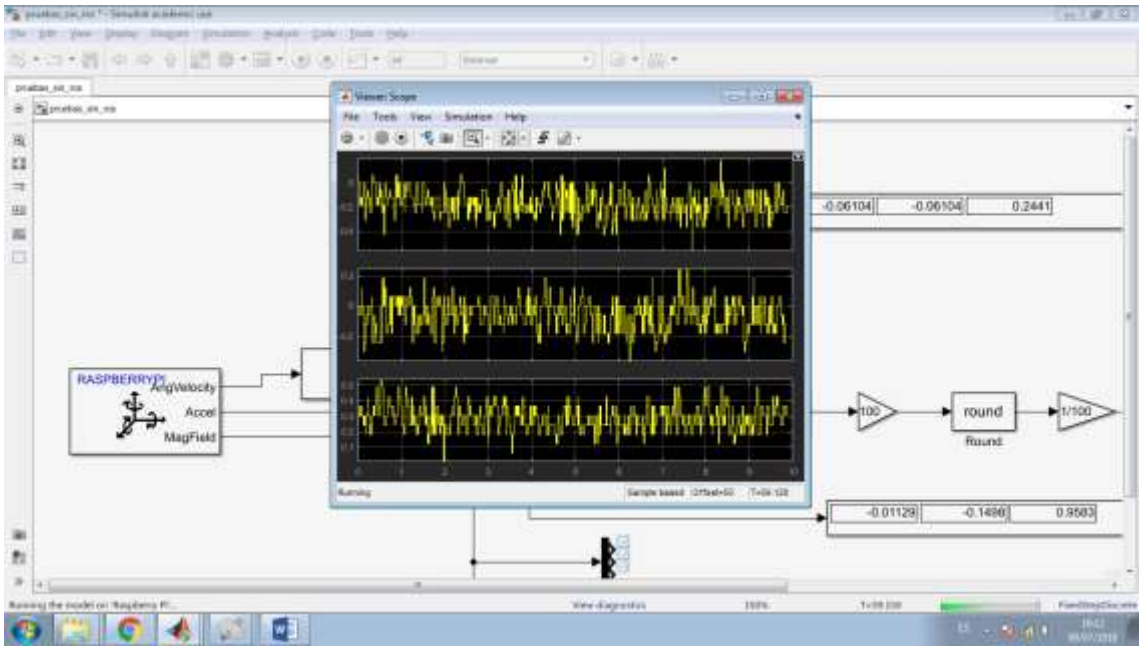


Figura 4. 15 Giroscopio con Filtro de Mediana

Ahora que se han eliminado los picos y vemos la señal más de cerca, se aprecia que también tiene bastante ruido, ya que, aunque el sensor se encuentra en estado de reposo, presenta variaciones muy grandes. Por ejemplo, en el eje x, la señal se mueve entre -0.4 y 0.1. Para intentar eliminar este ruido, o al menos atenuar su efecto, se va a utilizar un filtro de Offset (un filtro de la media) que actúa como un filtro de baja frecuencia, ya que calcula durante unos segundos la media de la señal durante el reposo para aumentar la precisión. En la figura 4.17 podemos ver como se ha obtenido el offset durante la calibración usando un bloque integral para acumular el valor de la media.

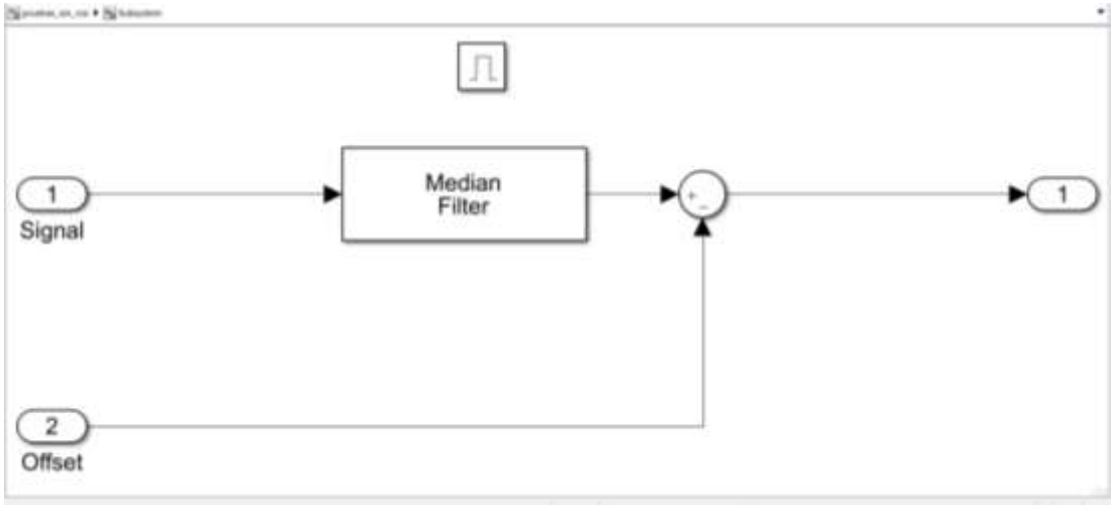


Figura 4. 16 Filtro de la mediana

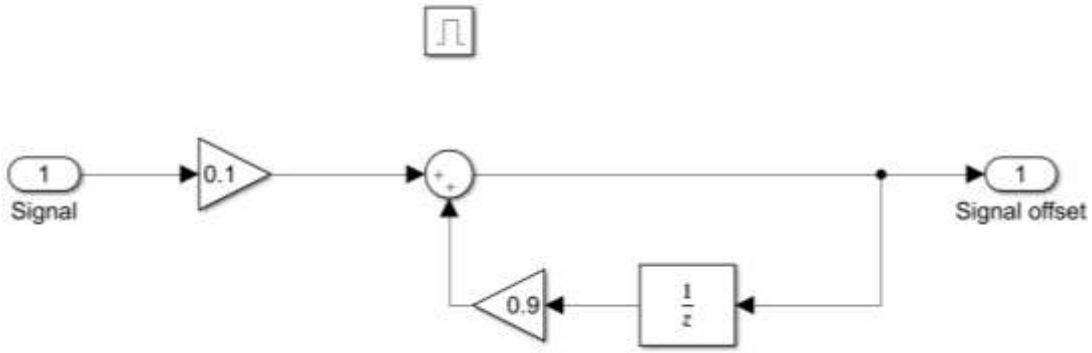


Figura 4. 17 Obtención del Offset

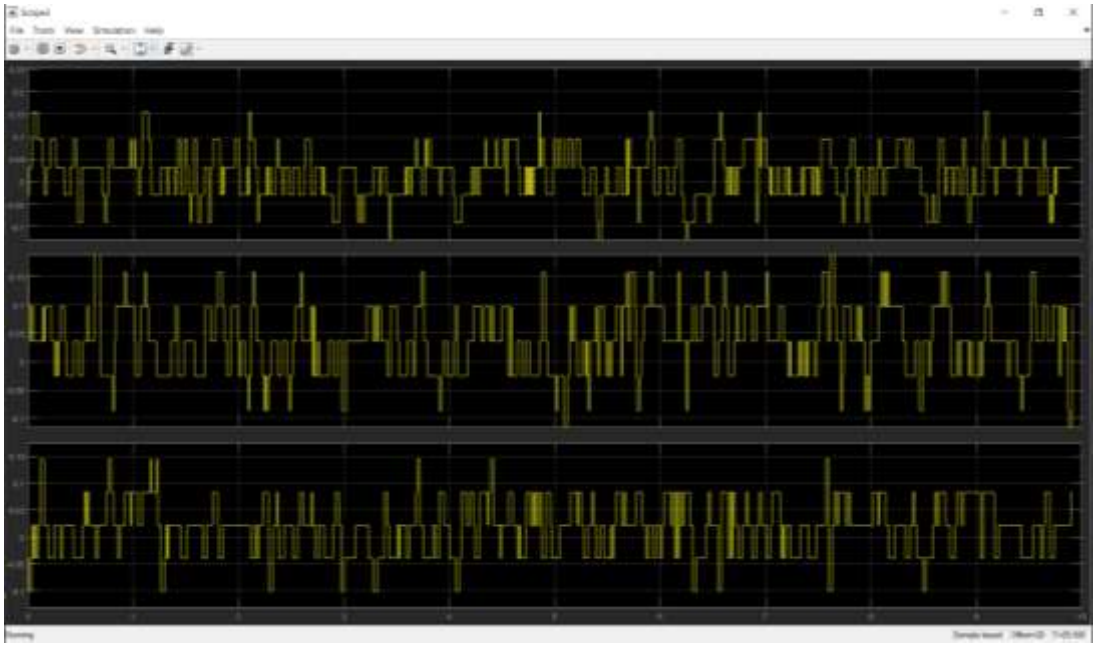


Figura 4. 18

Como se puede observar en la figura 4.17, el ruido ha disminuido considerablemente al aplicar el filtro de la media. Sin embargo, se ha conseguido eliminar tanto como se deseaba y, si se intenta muestrear la señal a largo plazo, el ruido latente causa un error integral que hace que la señal no pueda ser utilizable por sí misma para la obtención de la orientación del robot.

4.3.2 Comparación y combinación de señales

Para continuar el estudio del sensor inercial, vamos a comparar las señales angulares que podemos llegar a obtener por separado usando el giroscopio y la combinación de una señal mediante la arcotangente de dos señales del acelerómetro.

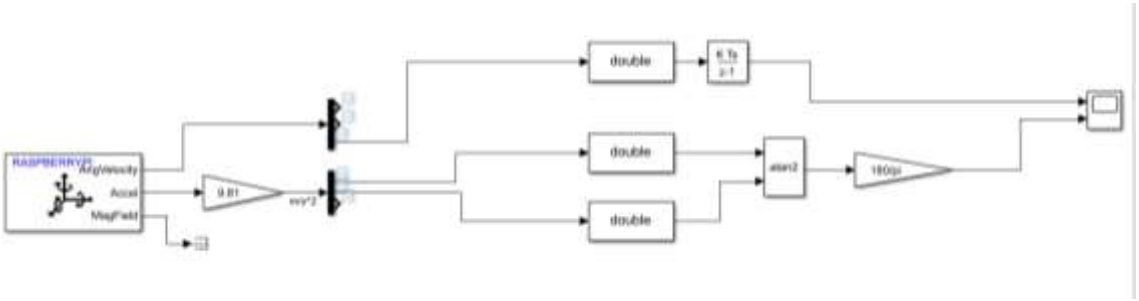


Figura 4. 19 Comparación de señales

Tal y como se esperaba, las señales son demasiado ruidosas para poder ser interpretadas, aunque ya podemos pronosticar que la segunda señal de la Figura 4.19, perteneciente al giroscopio se ve menos afectada por el ruido que la del acelerómetro (primera señal). Una vez estudiado este caso, se procederá a repetir el experimento usando los filtros de mediana y offset explicados anteriormente.

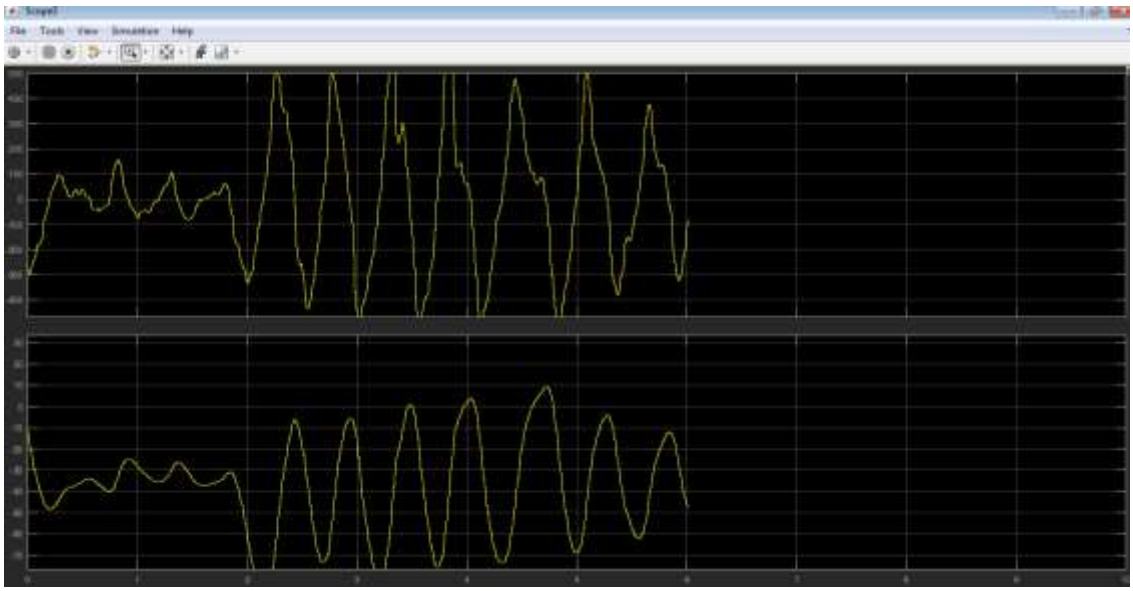


Figura 4. 20 Comparativa del angulo sacado del giroscopio y del acelerometro

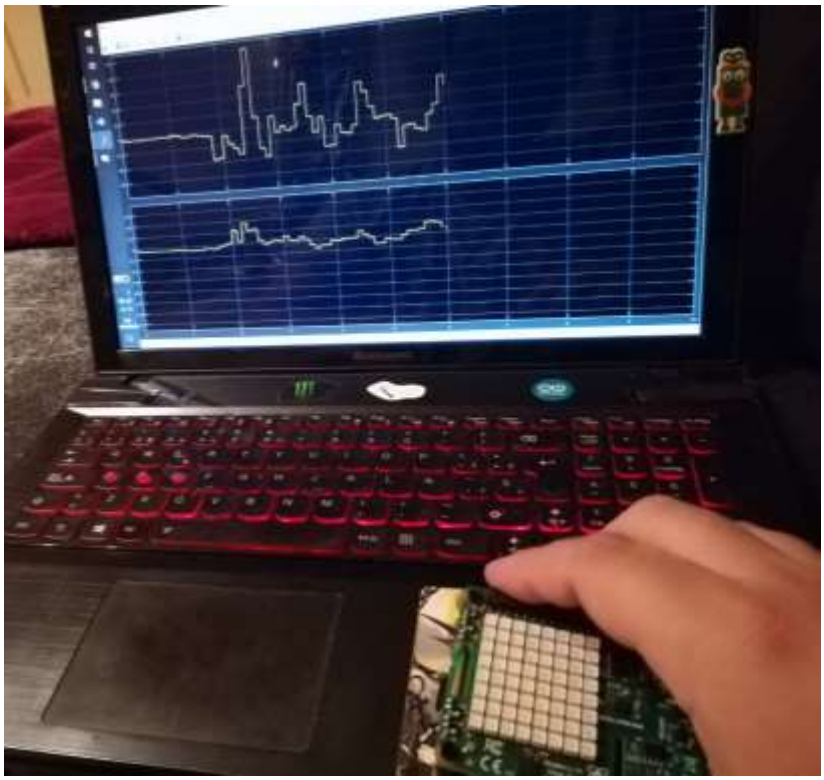


Figura 4. 21 Señales comparativas con filtro offset y mediana

Ahora las señales se encuentran en un rango mucho más admisible. Se puede apreciar cómo la señal del acelerómetro sigue presentando bastante ruido cuando es sometida a reiteradas variaciones de ángulo, mientras que el giroscopio, tal y como se había explicado en el punto anterior, posee un ruido muy pequeño pero que se acumula poco a poco con la integral, por lo que la fiabilidad es más discutible a medida que transcurre el tiempo.

4.3.3 Diseño del filtro final

La solución más funcional en este suceso consistirá en filtrar ambas señales de forma complementaria para que podamos obtener las ventajas de cada una eliminando sus problemas, es decir, obtendremos la fiabilidad a corto plazo del giroscopio despreciando el error que presenta el acelerómetro y confiaremos en los valores de largo plazo del acelerómetro, consiguiendo corregir el error integral que nos da el giroscopio.

En las Figuras 4.20 y 4.21 se presenta el modelo del filtro final para nuestro sistema. Este diseño consiste hacer pasar la señal por un filtro de offset y media para eliminar las perturbaciones iniciales que nos molestan y, posteriormente, combinar las señales del acelerómetro y el giroscopio en un filtro complementario para obtener una señal más precisa a partir de la combinación de ambas.

Para obtener cada una de las componentes Pitch, Roll y Yaw, se utiliza la combinación de su correspondiente señal procedente del giroscopio más la combinación de las dos componentes de la aceleración complementarias a ella. Por ejemplo, la componen Roll está formada por la señale en el eje X del giroscopio más las componentes Y y Z de la aceleración donde el resultado de hacer la arcotangente de ambas nos proporciona una componente X que puede compararse con la del giroscopio.

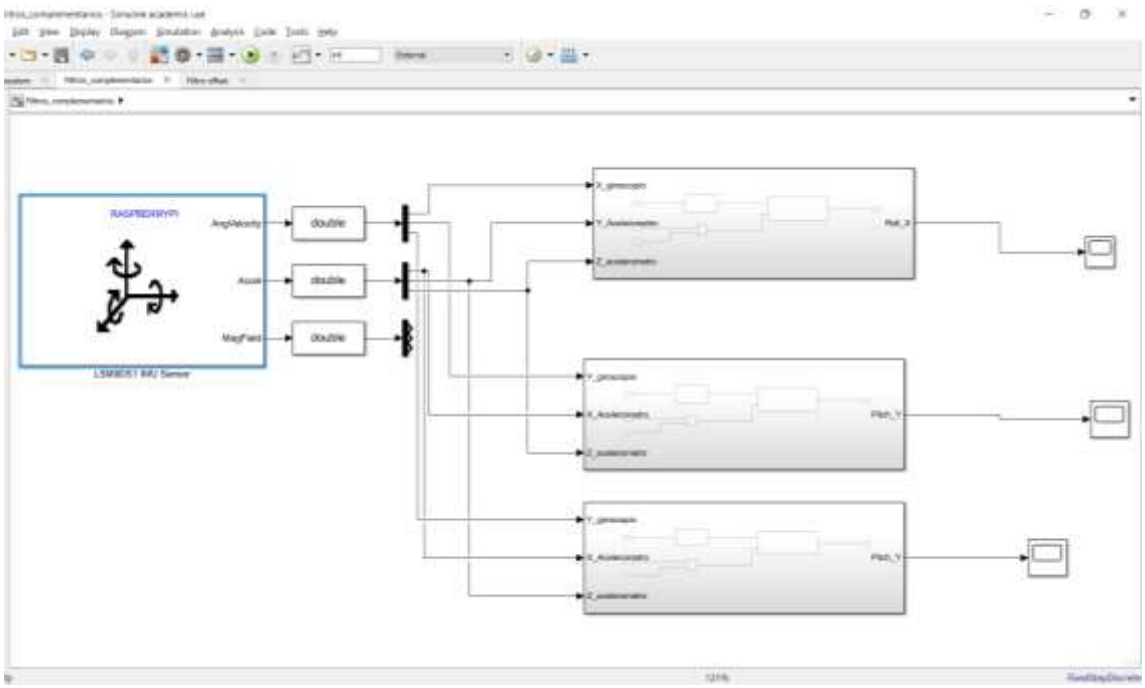


Figura 4. 22 Esquemático superior filtro final

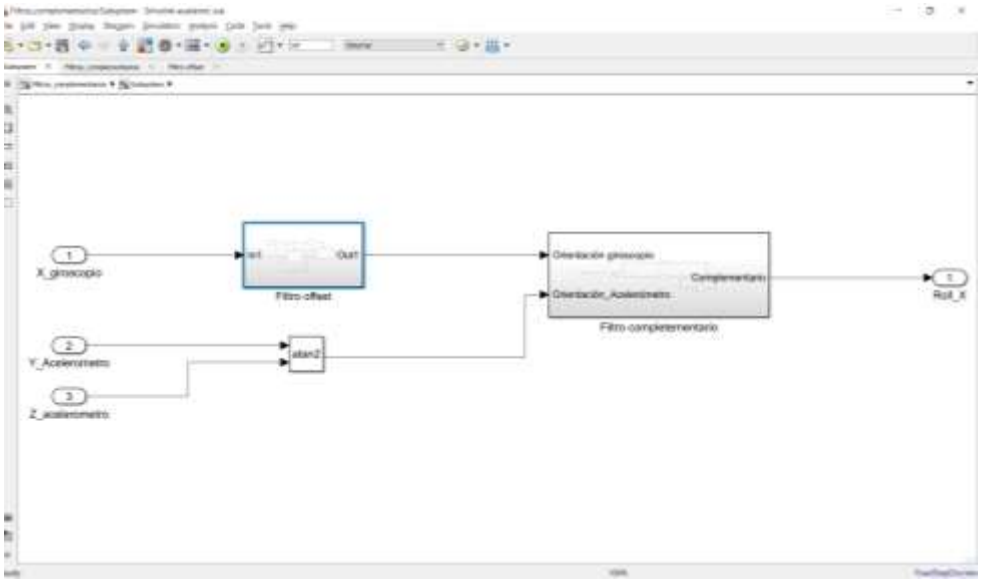


Figura 4. 23 Esquemático de los filtros

4.3.4 Resultados

Señales procedentes del Roll: Para el experimento primero se dejó el dispositivo en situación de reposo para ver cómo la señal estaba calibrada desde un principio en el cero. A continuación, se rotó sobre el eje X hasta que la posición de la Raspberry era de 90º respecto a la posición inicial y, por último, se volvió a dejar la placa en el estado inicial. Tal y como se puede apreciar en las siguientes figuras, el experimento dio resultados favorables pasando de 0 a -1.5 (aproximadamente pi/2) y retornando a su valor de equilibrio.

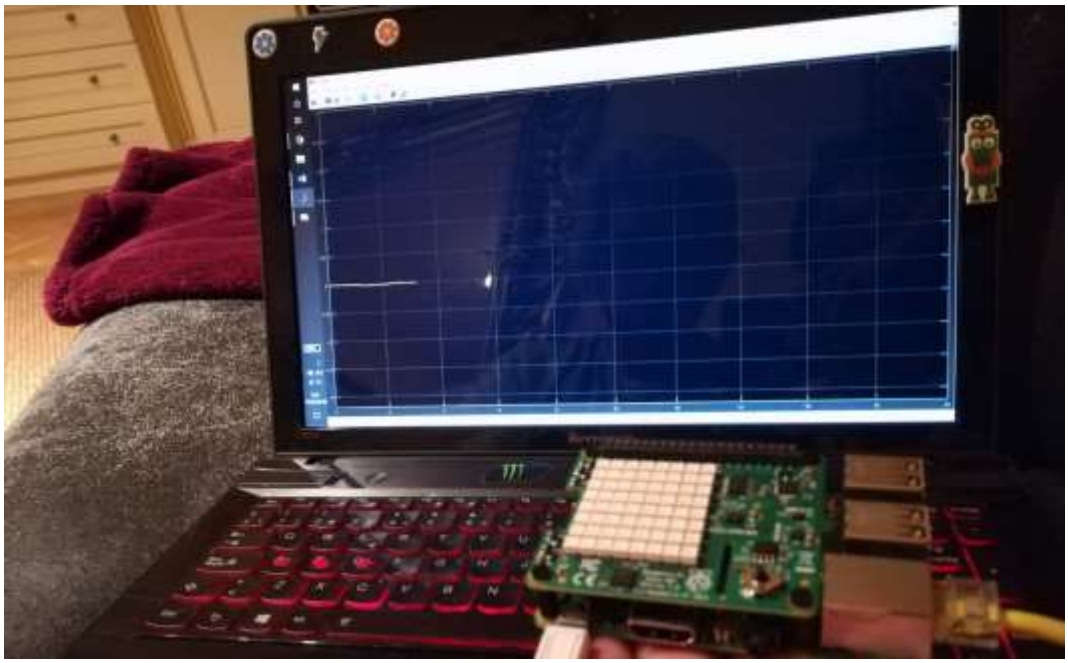


Figura 4. 24 Experimento Roll

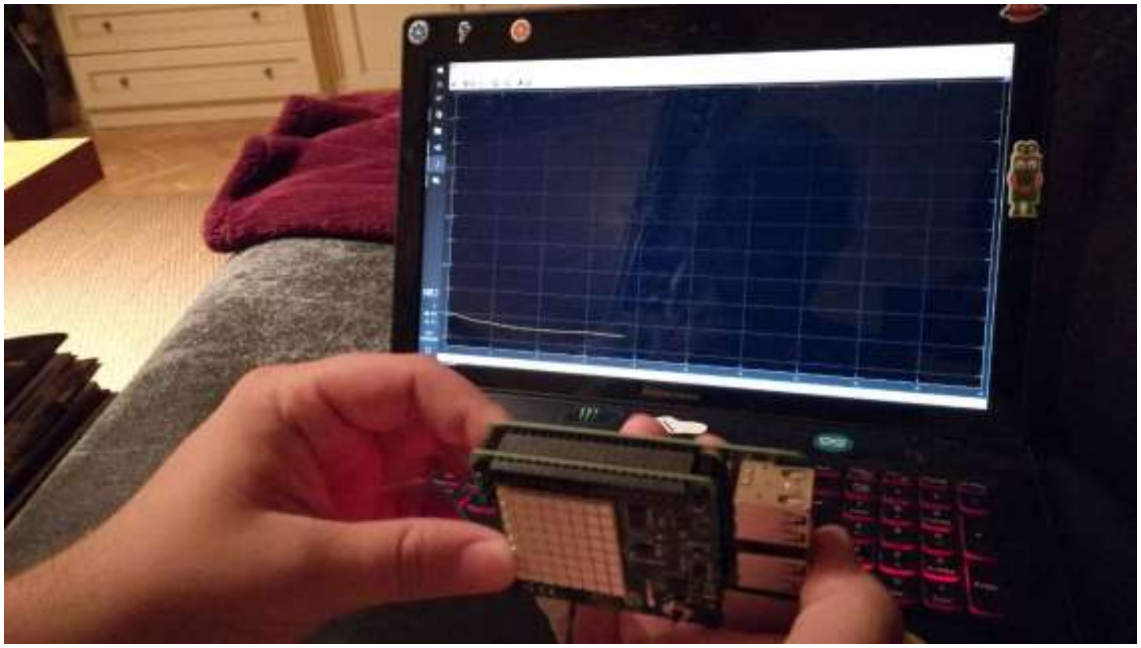


Figura 4. 25 Experimento Roll pi/2

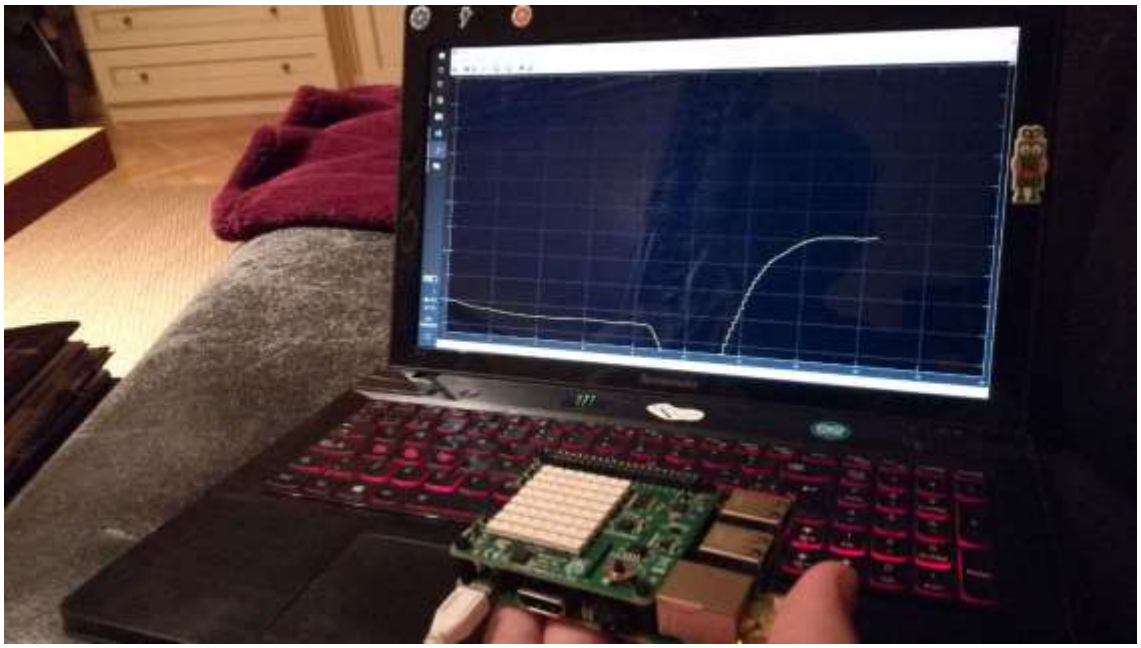


Figura 4. 26 Experimento Roll Reposo

Señales procedentes del Pitch: Se ha realizado el mismo experimento que para la componente Roll y también se han obtenido resultados satisfactorios.

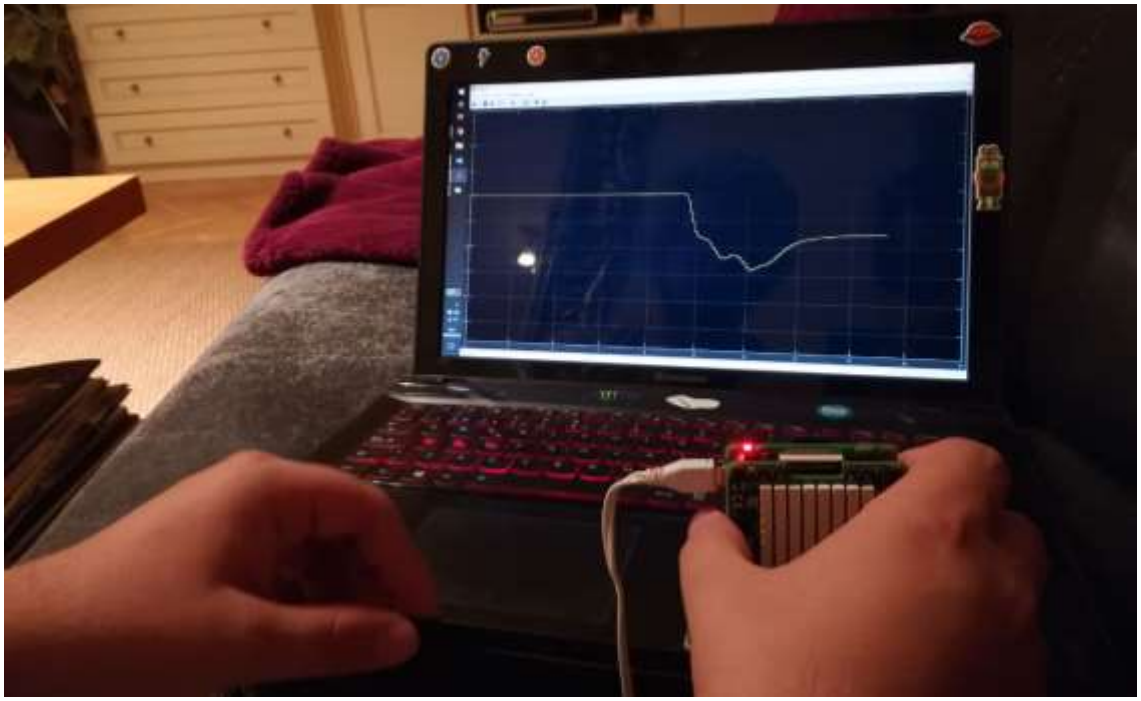


Figura 4. 27 Experimento Pitch pi/2

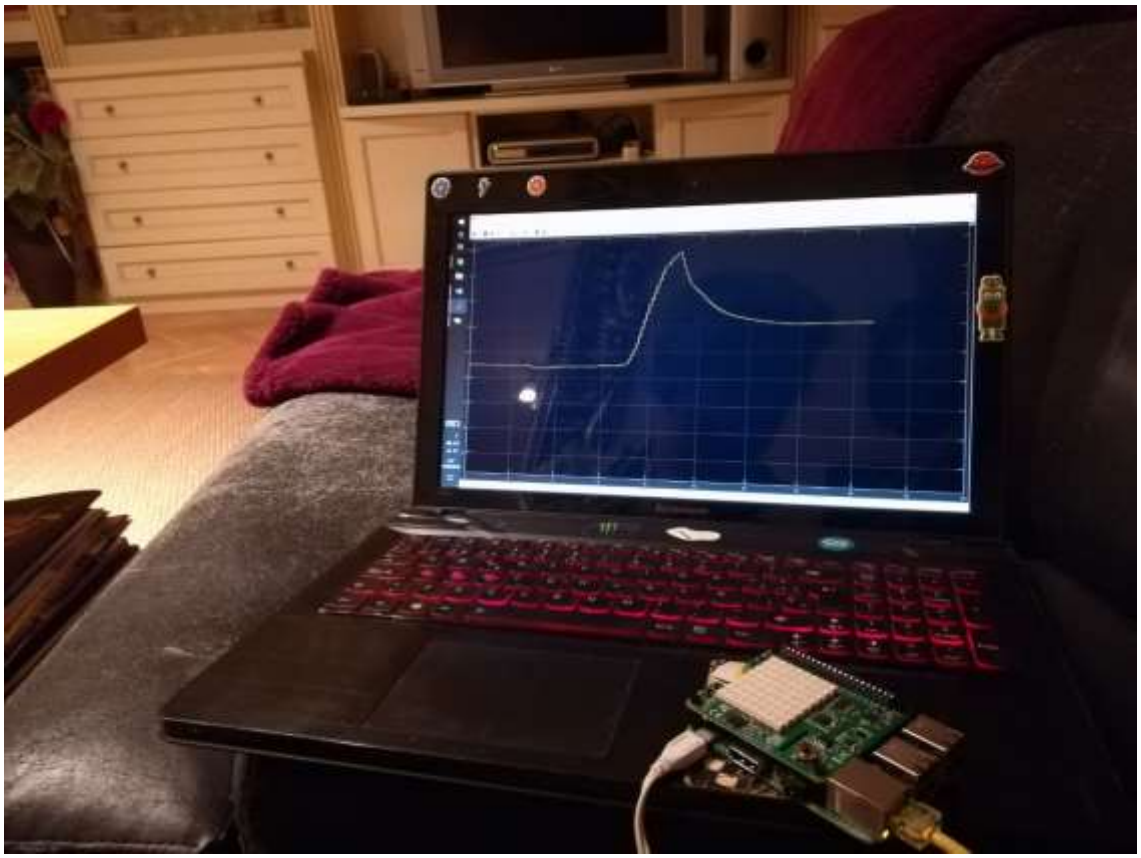


Figura 4. 28 Experimento Pitch reposo

Señales procedentes del Yaw: Repetimos la experiencia y el sistema se comporta como se esperaba. El filtro complementario ha sido un éxito.

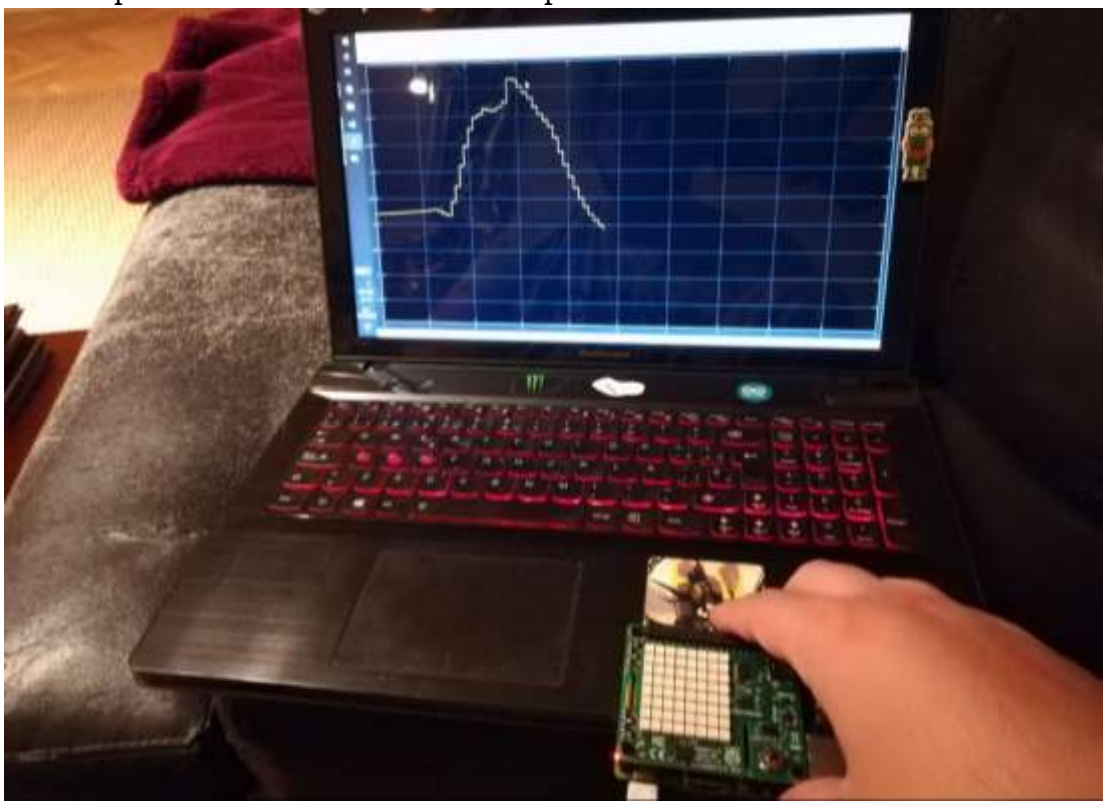


Figura 4. 29 Experimento Yaw giro

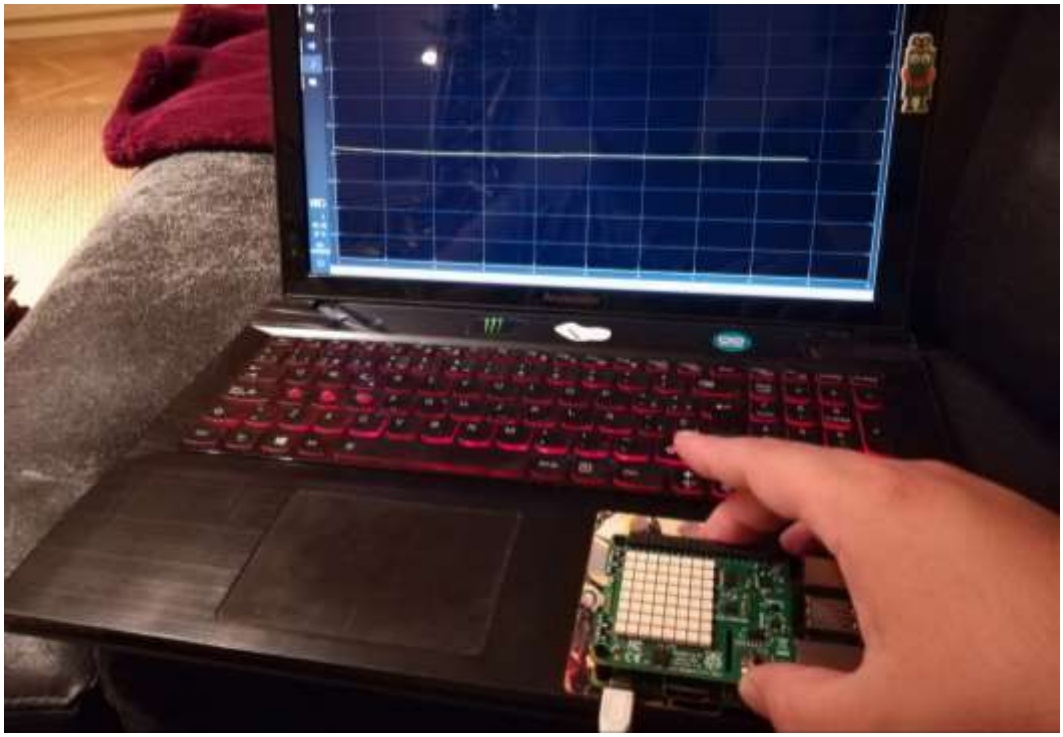


Figura 4. 30 Experimento Yaw reposo

4.4 IMPLEMENTACIÓN CON ROS

En el Capítulo 3 se realizó un modelo sencillo de comunicación con ROS en Simulink. Ahora se empleará ese conocimiento para aplicarlo al diseño que nos interesa para poder intercambiar información mediante la publicación y suscripción a topics correspondientes a los sensores de los que disponemos.

El esquemático del diseño es el mostrado en la Figura 4.31 donde se explicarán los bloques que lo componen.

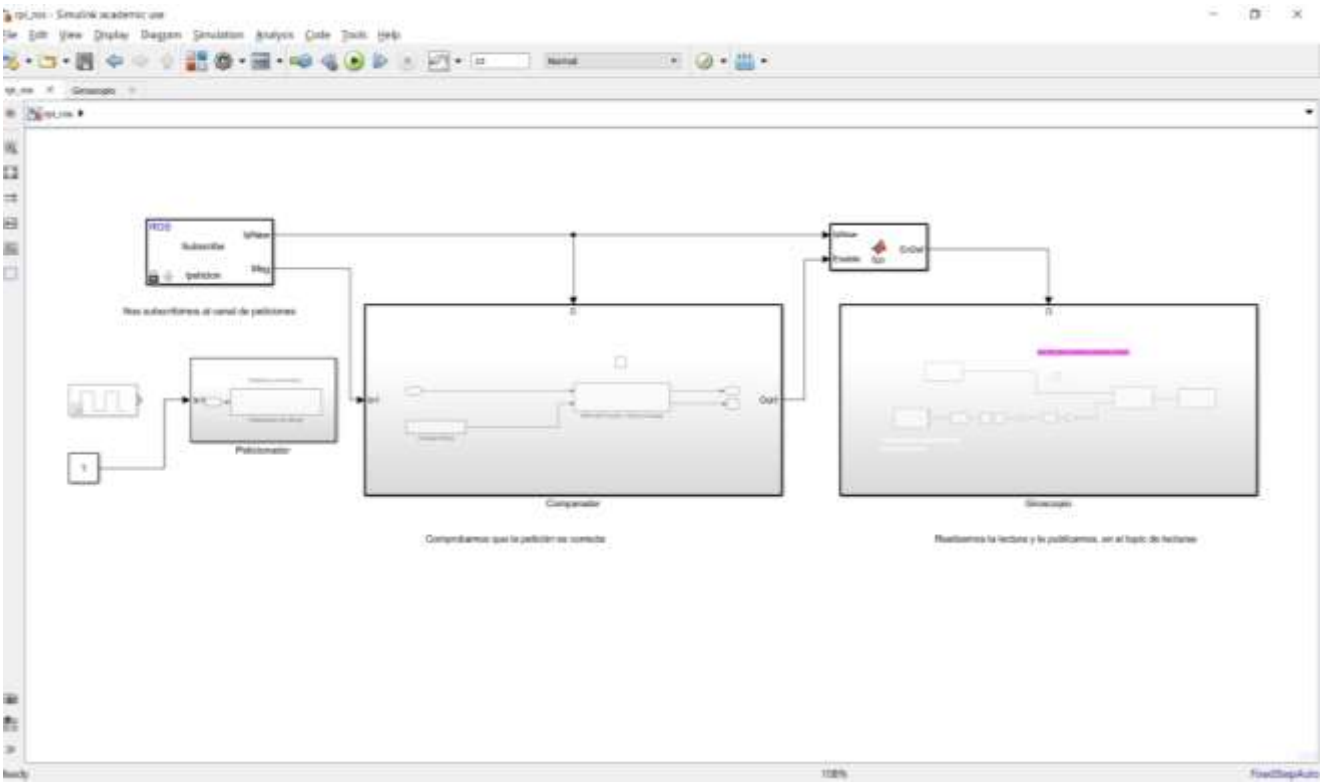


Figura 4. 31 Esquemático de communicacion ROS

En primer lugar, tenemos el "Peticionador" (4.32), este bloque que se encarga de generar señales de petición de lectura cuando se desee a través de un protocolo de comunicación basado en la recepción del string "lectura". El mismo sistema está suscrito al topic /petición, el cual, al recibir el mensaje exacto de "lectura" habilita la publicación de los valores de la IMU en el topic de /lectura (Figura 4.33).

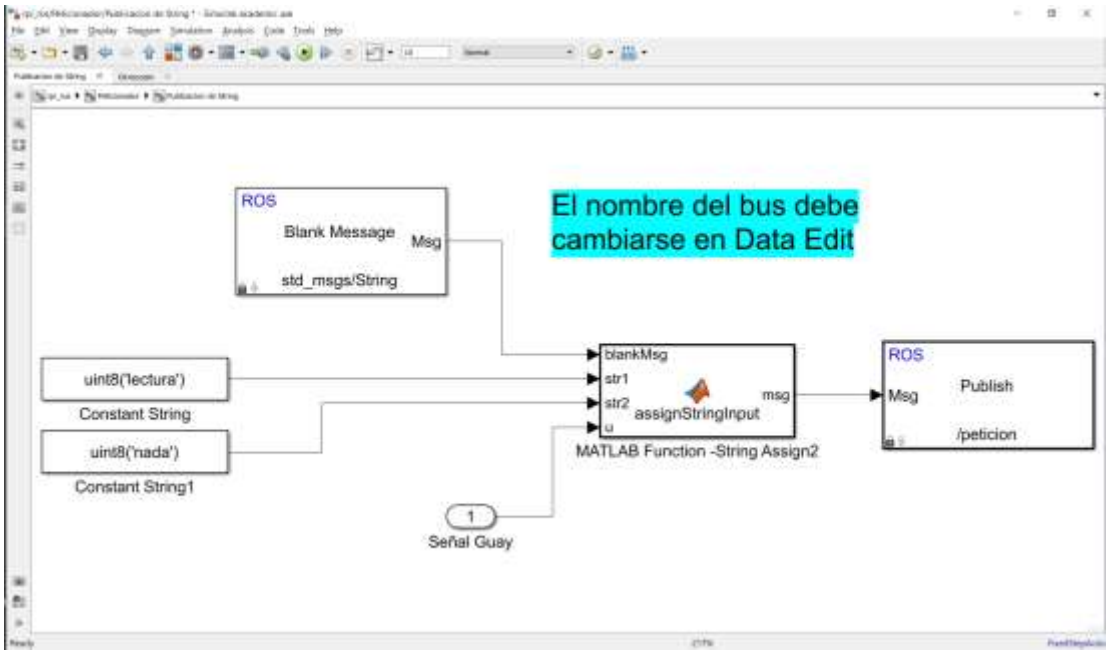


Figura 4. 32 Peticionador

Figura 4. 33 Topic /lectura

Figura 4. 34 Información publicada en el topic /lectura

En el bloque giroscopio se rellena un mensaje en blanco con los datos capturados por la IMU para ser publicados en el topic /lectura cuando el bloque está habilitado. La habilitación de este bloque la lleva a cabo el sistema comparador el cual se encarga de comparar el mensaje del topic /petición con el string /lectura y, si coinciden, se habilita el intercambio de información. Por otro lado, el bloque Matlab Fuction nos sirve para convertir la información de la IMU en un vector de tres coordenadas tipo Point capaz de ser reconocido por la comunicación de ROS.

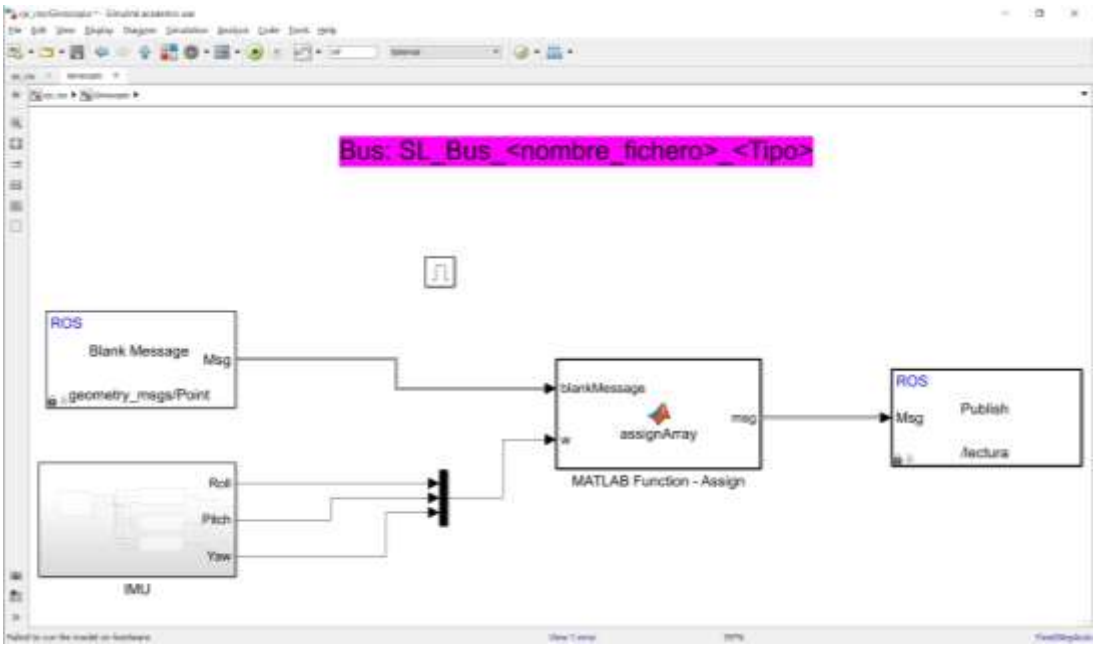


Figura 4. 35 Bloque giroscopio

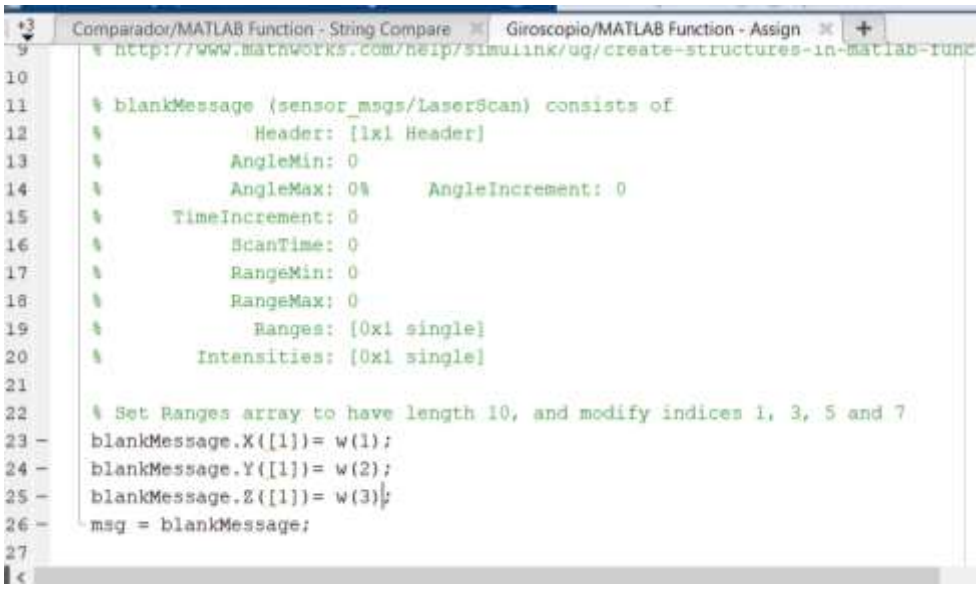


Figura 4. 36 Matlab Function

CAPÍTULO 5: IMPLEMENTACIÓN SENSOR LÁSER 2D

En este capítulo introduciremos el sensor láser de la marca Hokuyo para capturar la información del entorno empezando desde el programa más básico hasta el modelo que se comunica en Simulink con ROS.

5.1 PROGRAMA OFICIAL

Para la primera toma de contacto con el sensor láser 2D, descargaremos y ejecutaremos el programa URG Benri Plus (Figura 5.1) proporcionado en la página oficial de Hokuyo[10].

El programa es muy sencillo y basta con conectar el sensor vía USB al ordenador y seleccionar en el programa el puerto COM correspondiente al láser para establecer la comunicación. Por último, hacemos *click* en el símbolo de conectar y todos los datos del sensor se actualizarán automáticamente en el programa permitiendo visualizar el área de barrido del láser en tiempo real.

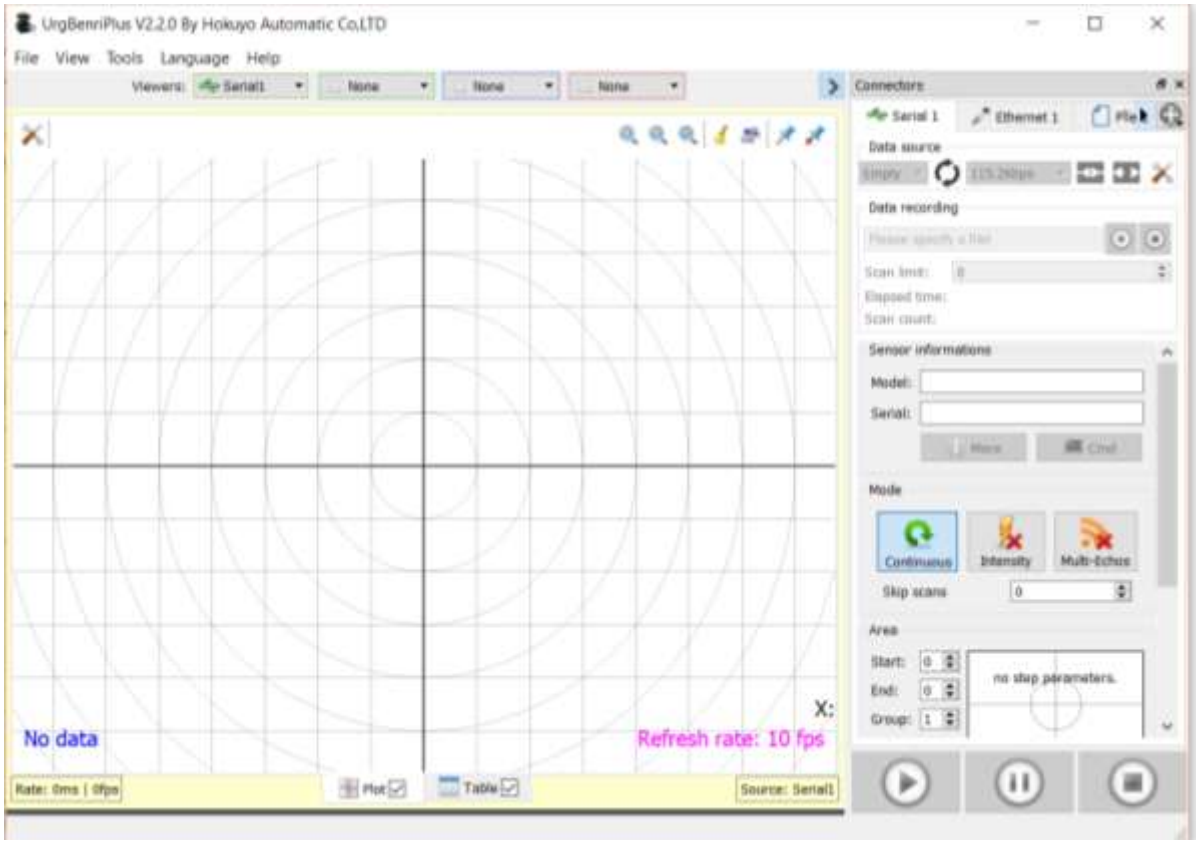


Figura 5. 1 Ventana URG Benri Plus

A continuación, se puede apreciar la zona de exploración del sensor láser, donde las zonas más alargadas se corresponden con los obstáculos que están más alejados de él.

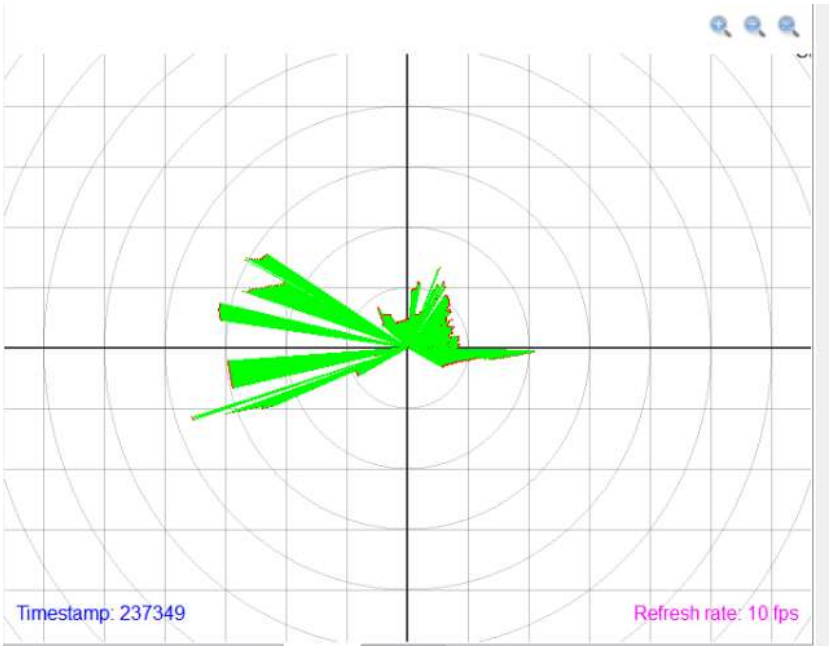


Figura 5. 2 Ejemplo funcionamiento del programa

5.2 PROGRAMA EN MATLAB QUE PERMITE MOSTRAR POR PANTALLA LOS PUNTOS DEL ENTORNO

Para probar el funcionamiento del sensor con MATLAB se utilizará un programa que se encuentra en la propia librería de addons de MATLAB[11] que funciona directamente con el sensor Hokuyo y, como configuración previa para su funcionamiento, sólo requiere que conectemos el láser por USB al ordenador y declarar qué puerto COM se está utilizando. El programa en cuestión se encarga de leer todos los valores que captura el sensor y decodificarlo para mostrar mediante un gráfico de puntos (Figura 3.15) la distancia a los obstáculos que se encuentran en la zona de exploración del sensor.

El programa funcionó correctamente con el Hokuyo UBG 04 LX, sin embargo, en el momento de la prueba no se realizaron capturas del experimento y , debido a que el sensor láser se averió, se expone una imagen de ejemplo (Figura 3.15) similar a los resultados obtenidos durante la ejecución.

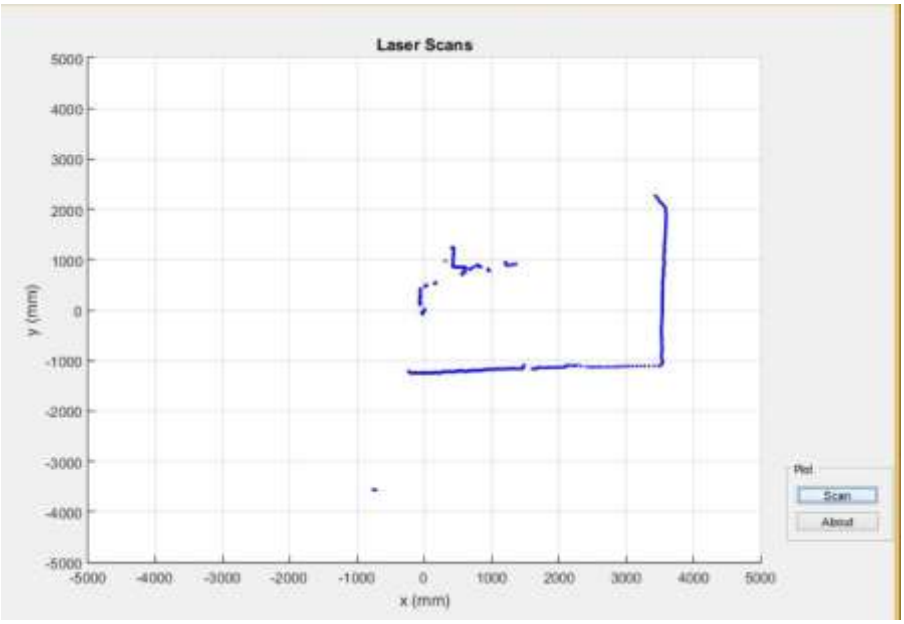


Figura 5. 3 Imagen generada con el programa de Matlab

5.3 COMUNICACIÓN CON ROS

En primer lugar, nos conectaremos al WiFi de la Raspberry Pi para controlarla de forma remota mediante SSH gracias al programa Bitvise. El nombre de la red es #ubiquityrobot y la contraseña es robotseverywhere (esta información se encuentra alojada en la página oficial de ubiquity robot[9].

Figura 5. 5 Conexión wifi

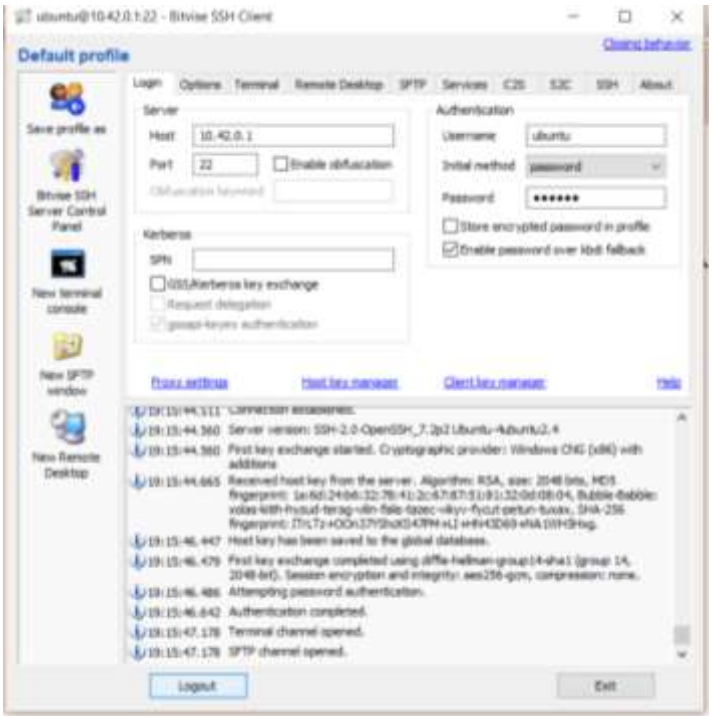


Figura 5. 4 Conexión Bitvise

Una vez establecida la conexión, debemos descargar los paquetes correspondientes al nodo del láser. En la wiki oficial de ROS podemos encontrar la descripción del nodo y un link que nos redirecciona al repositorio de GitHub donde se encuentra alojado el paquete urg_node correspondiente al nodo compatible con los láseres de la marca Hokuyo. Este nodo está configurado de manera que realiza lecturas de forma automática y las publica en el topic /scan utilizando el tipo de mensaje LaserScan.

Además de instalar el urg_node en nuestra Raspberry Pi, es preciso tener instalado, principalmente, el paquete urg_c, un paquete que provee de las librerías necesarias para que el urg_node funcione correctamente, así como la siguiente lista de nodos básicos de ROS (Figura 5.6). Los nodos que necesita el sistema para hacer funcionar el urg_node los podemos conocer o bien mirando en la página de la wiki de ROS o, en su defecto, al utilizar el comando catkin_make, cada vez que la compilación llegue a su límite, nos dirá cuál es el paquete requerido para poder continuar.

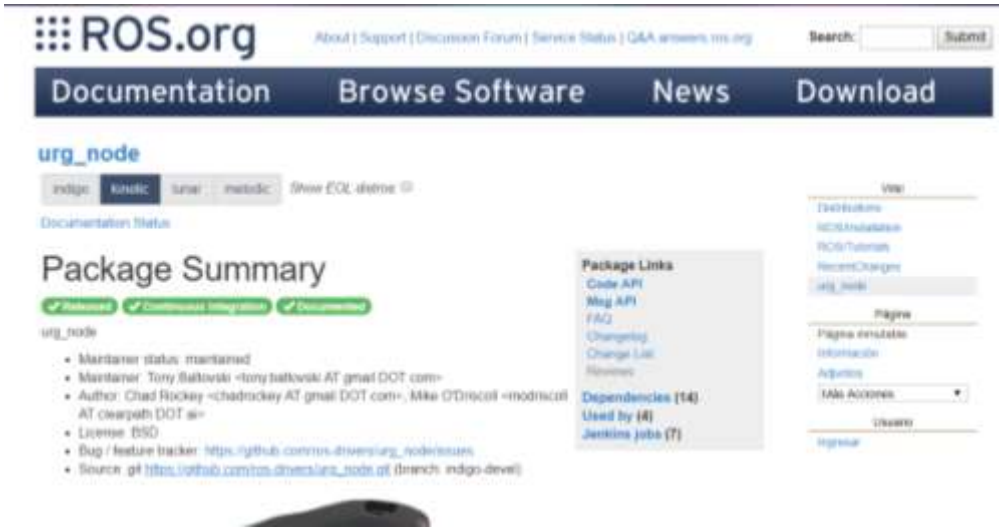


Figura 5. 6 Página de ROS

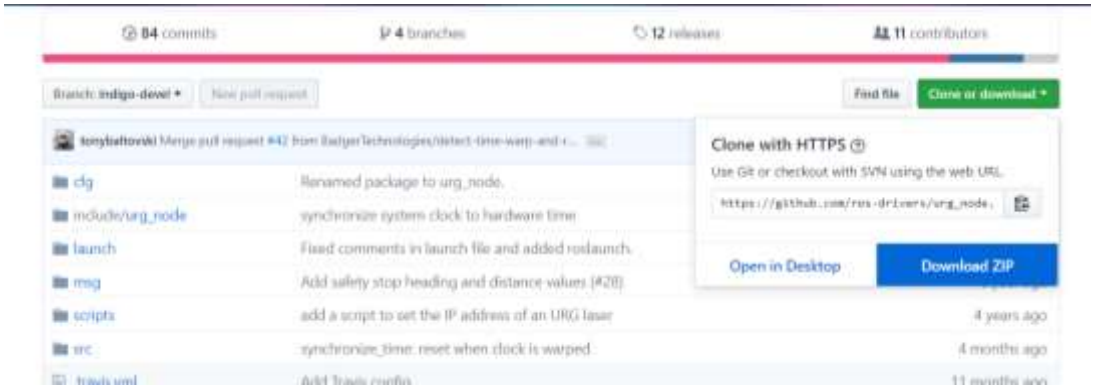


Figura 5. 7 Repositorio de Github

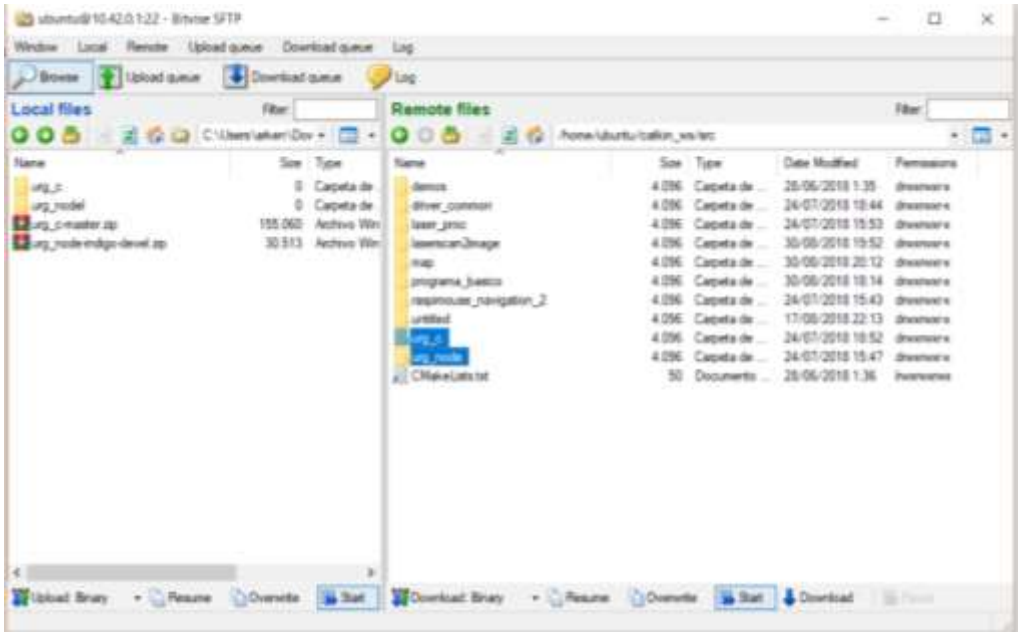


Figura 5. 8 Explorador SFTP

Descargamos los paquetes, los descomprimimos y los copiamos en la ruta catkin_ws/src, carpeta donde deberán ir todos los paquetes correspondientes a los nodos de ROS. Una vez hecho esto, nos vamos a la ubicación catkin_ws desde la ventana de comandos y compilamos los paquetes mediante un catkin_make.

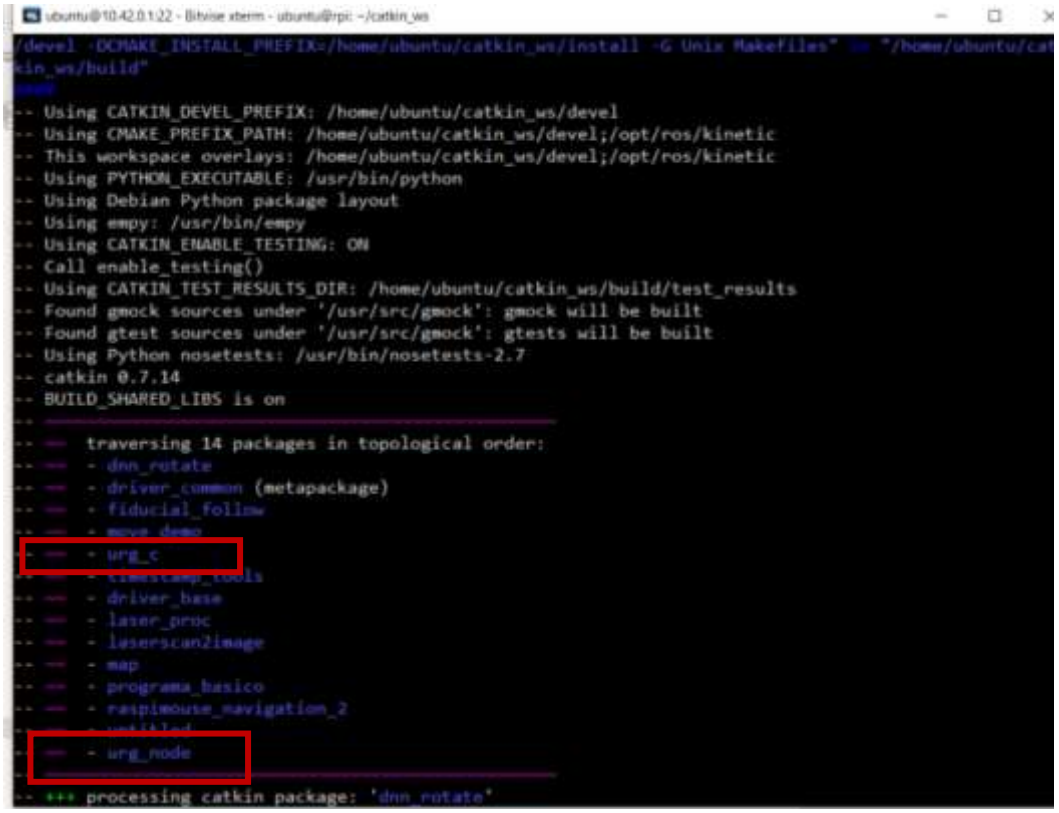


Figura 5. 9 Catkin_make

Una vez terminada correctamente la compilación, debemos indicar el puerto por el que el láser enviará la información al nodo. Para ello ejecutamos las siguientes líneas en la ventana de comandos que establecerá el puerto y la identificación del dispositivo (el sensor debe estar conectado a la Raspberry Pi y encendido para que haya conexión).

```
$ rosrun urg_node getID /dev/ttyACM0

Device at /dev/ttyACM0 has ID H0807228
```

Por último, sólo falta empezar el envío de datos a través del nodo, por lo que lanzamos el urg_node con el siguiente comando.

```
Rosrun urg_node urg_node
```

Ahora que el nodo está enviando información, procedemos a establecer comunicación con Simulink para trabajar con los datos procedentes del sensor. La Figura 5.4 es un modelo obtenido de [12] donde su funcionamiento es muy lógico y sencillo.

La primera función recoge toda la información importante del tipo LaserScan para procesar la ubicación de los objetos detectados. En primer lugar, se coge el vector de rangos que almacenan todas las distancias de medida obtenidas por el láser en el rango disponible. De este vector se despreciarán los puntos que tengan valores iguales a las medidas más pequeñas y grandes ya que indican objetos que están fuera del rango de medida (muy cerca o muy lejos del sensor. Lo siguiente será crear un vector de ángulos asociado a cada medida, para ello necesitamos conocer la cantidad total de medidas que lo obtenemos del current Lenght. El valor de ángulo mínimo, el valor de ángulo máximo y el incremento angular, con todo esto obtener el vector de ángulos será tan fácil como crear un vector con el mismo tamaño que el de medida y rellenar cada posición con la siguiente fórmula:

angulo(i) =angulo_minimo+(i-1)*incremento_angulo

```
function [angles,dists] = fcn(AngleMin, AngleMax,
AngleIncrement, ...
                TimeIncrement, RangeMin, RangeMax, numRanges,
Ranges )
%#codegen
% this function transforms the laser scan into cartesian
coordinates, so
% that you can use other tools to group them together, make an
image, etc.
if numRanges > 0
angleMin = double(AngleMin);
angleMax = double(AngleMax);
angleIncrement = double(AngleIncrement);
% assert(angleMax > angleMin);
% assert((angleMax-angleMin)/angleIncrement > 1);

angleMin = -pi/2;
angleMax = pi/2;
angleIncrement = pi/179;
timeIncrement = TimeIncrement;
rangeMin = RangeMin;
rangeMax = RangeMax;
ranges = Ranges;
ranges(ranges>RangeMax) = NaN;
ranges(ranges<RangeMin) = NaN;
%rangesLength = scan.Ranges_SL_Info.ReceivedLength;
%intensities = scan.Intensities;
%coder.varsize('angles',[180 1]);
assert(numRanges < 360);
```

```
angles=double(zeros(numRanges,1));
for ii=1:numRanges
    angles(ii) = double(angleMin) + double(ii-
1)*(angleIncrement);
end
% angles = [angleMin:angleIncrement:angleMax]';
dists = ranges;
assert(length(dists) == numRanges);
assert(length(dists) == length(angles));
else
    angles=double(0);
    dists=single(0);
end
```

Con esto ya tenemos las dos salidas del bloque: El vector de rangos filtrado y los ángulos asociados a los puntos medidos. Finalmente, para conocer la posición exacta de los puntos, basta con resolver x(i)=ranges(i)*cos(angulo(i)) para obtener la componente en X y y(i)=ranges(i)*sin(angulo(i)) para la componente en Y. Gracias a esta conversión, ya tenemos todos los puntos referenciados en coordenadas cartesianas y se puede trabajar cómodamente con ellos.

```
function points = fcn(angles,dists)
%#codegen
% x values align with "forward" points
xs = dists.*cos(angles);
% y values align with left (positive) points
ys = dists.*sin(angles);
points = [xs,ys];
```

Este modelo se estuvo probando con el sensor Láser Hokuyo ya que era precisamente lo que se buscaba implementar, sin embargo, no se puede confirmar del todo su compatibilidad ya que durante el proceso de experimentación el sensor se averió.

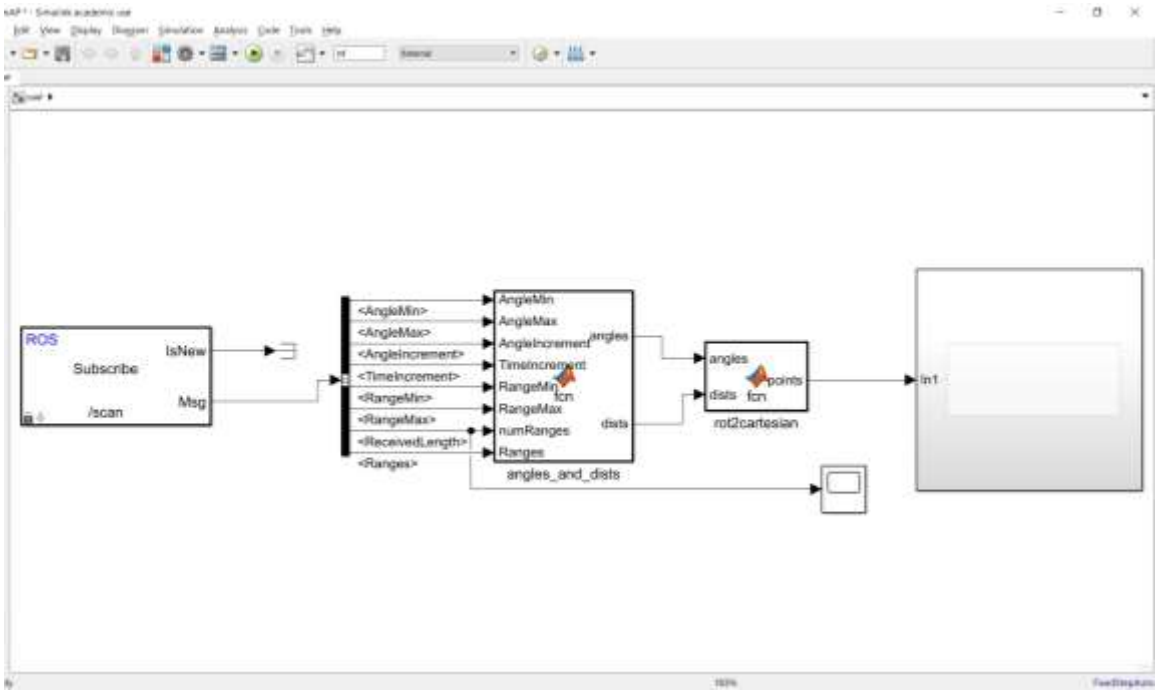


Figura 5. 10 Diseño mapeo

CAPÍTULO 7: CONCLUSIONES

El trabajo con la IMU tuvo cierto grado de dificultad, sin embargo, se obtuvieron resultados satisfactorios donde el sistema respondía correctamente ante los cambios y, además, se consiguió una correcta comunicación con ROS.

Por otro lado, el sensor láser resultó ser incompatible con la versión de Raspbian utilizada para el Sense Hat, por lo que se perdió mucho tiempo intentando encontrar un sistema operativo con ROS capaz de entablar la comunicación. Una vez encontrado una imagen preparada, se trabajó sobre ella, sin embargo, se descubrió que el sistema operativo utilizado, Ubunu 16.04, no reconocía el módulo Sense Hat. Al final se optó por trabajar con cada sensor en el sistema operativo que lo soportaba impidiendo de este modo poder integrar ambas partes en un único algoritmo.

En cuanto a los objetivos se completaron casi todos, sin embargo, los problemas técnicos, como la avería del sensor, y de software impidieron que se cumpliesen todos los al cien por cien, por lo que se proponen como líneas de trabajo futuras.

CAPÍTULO 8: LÍNEAS DE TRABAJO FUTURAS

Las líneas de trabajo futuras para seguir desarrollando el robot móvil Krobot relacionadas con el alto nivel son:

- Conseguir un sistema operativo capaz de funcionar con ROS, Simulink y todos los sensores en conjunto.
- Realizar un algoritmo de navegación local a partir de los datos del láser.
- Implementar un algoritmo para trabajar con la cámara Kinect.
- Realizar un control del sistema cerrado capaz de reaccionar automáticamente ante obstáculos.

REFERENCIAS BIBLIOGRÁFICAS

[1] R. Pimentel-Castillo, "Diseño y construcción de un robot móvil de dimensiones reducidas dotado de sensores y capacidad de teleoperación con interfaz táctil," Universidad de Málaga, 2012.

[2] C. M. Jiménez, "CONTROL DE ROBOT MÓVIL KROBOT," Universidad de Málaga, 2017.

[3] "Raspberry Pi 3 Model B - Raspberry Pi." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 26-Aug-2018].

[4] "Simulink." [Online]. Available: https://es.mathworks.com/products/simulink.html. [Accessed: 17-Aug-2017].

[5] "Scanning Rangefinder Distance Data Output/UBG-04LX-F01 Product Details | HOKUYO AUTOMATIC CO., LTD." [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=164. [Accessed: 25-Apr-2018].

[6] Bitvise Limited, "Bitvise SSH Client | Bitvise." [Online]. Available: https://www.bitvise.com/ssh-client. [Accessed: 01-Sep-2018].

[7] Debian Project, "FrontPage - Raspbian." [Online]. Available: https://www.raspbian.org/FrontPage. [Accessed: 01-Sep-2018].

[8] Canonical Ltd., "The leading operating system for PCs, IoT devices, servers and the cloud | Ubuntu." [Online]. Available: https://www.ubuntu.com/. [Accessed: 01-Sep-2018].

[9] Ubiquity Robotics, "Ubiquity Robotics," 2017. [Online]. Available: https://ubiquityrobotics.com/index.html. [Accessed: 01-Sep-2018].

[10] "HOKUYO AUTOMATIC CO., LTD." [Online]. Available: https://www.hokuyo-aut.jp/. [Accessed: 05-Sep-2018].

[11] S. Shrestha, "Hokuyo URG-04LX LIDAR Driver for MATLAB - File Exchange - MATLAB Central," 2012. [Online]. Available: https://es.mathworks.com/matlabcentral/fileexchange/36700-hokuyo-urg-04lx-lidar-driver-for-matlab?s_tid=srchtitle. [Accessed: 08-Sep-2018].

[12] "From LaserScan to a Simulink image | CPS-VO." [Online]. Available: https://cps-vo.org/node/34057. [Accessed: 10-Sep-2018].